J. S. Mobach.

Siemens System 7·000
Reference Manual and Instruction List

1st Edition January 1976
1st Revision February 1977
2nd Edition February 1978

Details on the planned support by the system software and individual Siemens System 7.000 hardware models for the architectural features discussed in this manual may be requested from your nearest Siemens office.

SIEMENS AKTIENGESELLSCHAFT

Contents

1. General

With its high computing-power, wide variety of main memory and peripheral configurations, extremely high system and channel throughput rates and its user-oriented system functions, the Siemens System 7.000 is one of the most versatile and cost-effective DP systems offered on the market today.

The most distinguishing feature of the Siemens System 7.000 is its systematic use of state-of-the-art system architecture, employing highly integrated circuitry and semiconductor technology. A series of new functions provide for full exploitation of the system capabilities: dynamic address translation (virtual addressing), page chaining, extended floating-point facility, timers, block multiplexor channel and multiprocessor operation, combined with a high level of system reliability, availability and maintainability.

2. Siemens System 7.000

The Siemens System 7.000 caters to the specific demands of the user. Its instruction set surpasses that of previous systems in performance and functional scope and is suitable for both commercial and for technical and scientific applications. The systems's area of application is enhanced by the wide range of storage facilities in real and virtual operation. Stack instructions and channel commands offer new programming and application options.

The dynamic address translation facility and appropriate operating system support provide the user with a storage space of over 16 million bytes in the case of virtual addressing, irrespective of the real memory configuration.

The Siemens System 7.000 also allows flexible connection of a wide variety of peripheral devices. Punched card equipment, printers, communication controllers and non-buffered I/O devices are operated via the byte multiplexor channel. The block multiplexor channels, on the other hand, are particularly suitable for interfacing buffered and high-performance peripherals, such as magnetic tape and disk devices.

An ideal system configuration can be obtained for each user simply by selecting the components best suited to his needs with regard to performance, functional characteristics and I/O capabilities.

2.1. Compatibility

The central units in the Siemens System 7.000 are all compatible, both upwards and downwards, although the individual models vary in performance, structure and functional implementation.
This compatibility ensures that programs will have the same results on each model in the 7.000 system, providing that:

- The same system functions are used in each case
- The program does not exploit any capabilities peculiar to one specific model
- Relationship between instruction times, transfer rates, channel program times and timer functions are not evaluated
- Only freely available storage areas are occupied
- The program run is not affected by intentional interrupts such as illegal format or illegal operation code

The physical data media - card, tape and disk pack - are fully compatible within the 7.000 System and with other systems, as are the data recording formats they use.

The Assembler, ALGOL, ANS COBOL, FORTRAN and PL/1 programming languages permit a simple and trouble-free transfer of programs from one 7.000 System model to another.

2.2. Availability

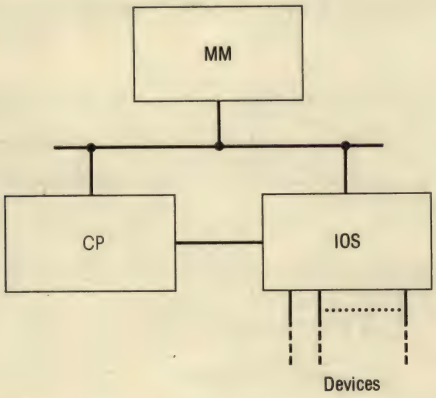
Availability is the ability of a system to accept and successfully process a user-specified task. The capabilities of the Siemens System 7.000 computers enhance the availability and thus the processing power of the system, whilst restricting the effects of errors on the program run and providing a precise error diagnosis. This high level of availability is achieved by means of:

- Memory protection in real and virtual operation which, in conjunction with the dynamic address translation facility, protects the memory areas against unauthorized access or destruction of their contents.
- The dynamic address translation facility, which effectively isolates the individual programs from one another, whilst still allowing them to share generally available system resources.
- Error detection and recovery (ECC) in main memory, the parity checks in the registers and data paths of the central unit, and automatic instruction retry, which allows intermittent errors to be bypassed.
- Automatically run cyclic diagnostic microprograms and I/O error routines, which provide a continuous assessment of the current system status and which, due to the accuracy of their diagnostic messages, permit rapid error processing and recovery.
- Indicators on the console in conjunction with system messages, which reduce the possibility of operating errors.

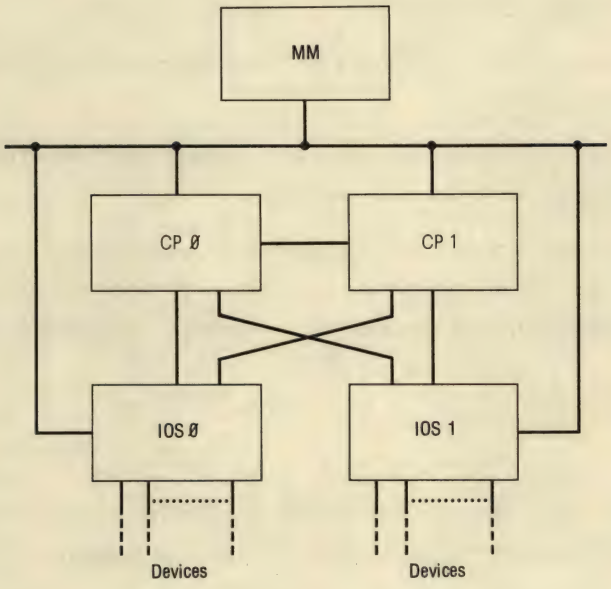
3. System Structure

The central units of the Siemens System 7.000 comprise the following functional units: the main memory system (MM), the central processor (CP), and the input/output system (IOS). The peripheral devices are linked to the IOS via device controllers and the standard I/O interface. Fig. 3-1 shows the functional structure of a monoprocessor, and that of a multiprocessor with two CPs and two IOSs.

Fig. 3-1



Functional Structure of a Monoprocessor



Functional Structure of a Multiprocessor with two CPs and two IOSs

3.1. Main Memory System

The main memory is responsible for directly supplying the central unit with information. Before programs and data can be processed they must first be loaded into main memory from the input devices.

The physical main memory system consists of a fast, large-capacity semiconductor memory; in the case of the more powerful central units, one or more ultra-fast buffer storages (caches) are preconnected to the semiconductor memory. The physical structure of the main memory system is not visible to the program and only influences the performance level.

Data held in main memory is lost when the installation is powered down; after powering up, the main memory must be resupplied with information.

3.1.1. Data Formats

The data in the main memory, processors and channels is organized in units of 8-bit bytes. All addresses and lengths are specified as byte multiples (except: the Load Bit Field (LBF) and Store Bit Field (STBF) instructions).

The bits in a byte are numbered from left to right, Ø through 7. The bytes in a fixed or variable-length data field are similarly numbered from left to right beginning with Ø. The bit or byte with number Ø is also designated as the high-order bit or byte, and that with the highest number, the low-order bit or byte.

Bytes can be combined to form groups:

Halfword:	2 consecutive bytes
Word:	4 consecutive bytes
Doubleword:	8 consecutive bytes
Byte field:	Any number of consecutive bytes

The address of a group always indicates the high-order byte in the group.

A group of consecutive bits aligned with any bit-oriented address is also called a bit field. The address indicates the high-order bit.

When the processor writes a byte or bit field to main memory, only the field addressed is affected, even when the physical data path is wider than the field.

For error detection and correction, one or more check bits are transmitted in the system together with the bytes or byte groups. The check bits are generated and processed by hardware without program access. In this Reference Manual, the check bits are omitted from all data format and storage capacity specifications.

3.1.2. Addressing

The main memory system is addressed byte-serially, by a 24-bit binary number starting from the left with Ø. The address space for a program is thus 2^{24} bytes = 16 Megabytes.

In general, only part of the address space is available as real memory. The real memory can physically vary in the way its memory modules are organized, but is always continuously addressable and starts with the address Ø.

Within the address space, instructions and data are addressed by effective addresses. The effective address can be virtual or real.

When the address translation facility is switched off, the effective address directly indicates a storage location in real memory: effective address and real address are identical. When the address translation facility is on, the effective address is virtual and is dynamically translated into a real address during the access.

The address space forms a closed loop where address $2^{24}-1$ wraps around to address Ø. A byte field crossing this boundary is processed in precisely the same way as an equivalent byte field which does not cross it. When an attempt is made to access a real memory byte which lies outside the installed memory, an error is flagged.

3.1.3. Alignment

Part of the data in main memory must be aligned. A byte group is aligned when the address is a multiple of the group length. A word address, for instance, must be a multiple of four.

Instructions must always be located on halfword boundaries, and channel commands on doubleword boundaries. Alignment is sometimes also necessary for the fixed-length memory operands of privileged and non-privileged instructions.

3.2. Central Processor

The central processor, which constitutes the central control unit of the computer, controls instruction execution, handles program interrupts, executes initial program loading, provides timers etc. Though the physical structure of the control unit varies between the different central processors within the 7.000 System, the logical function remains the same: execution of a valid instruction provides the same result on every installation.

The set of instructions which a central processor can execute breaks down into three subsets: non-privileged, privileged, and I/O instructions. Privileged and I/O instructions can only be executed when the central processor is in the privileged system state. The non-privileged instructions are responsible for the transfer and processing of the data, program branches, data format conversions, calling of subroutines and management of the stacks, calling of the system programs and a number of control functions.

The data processed can be fixed-length binary and floating-point numbers, variable-length decimals or fixed and variable-length logical information. Processing in the individual models is either serial or parallel with varying data path widths; the logical result is however always the same.

3.2.1. General Registers

The program addresses up to 16 4-byte general registers. These general registers can be used as base and index registers in address computations, for transferring addresses or for holding operands in binary arithmetic and logical operations.

The registers are numbered Ø through 15 and are addressed by the R fields in the instructions. For some instructions, two adjacent general registers are combined to form an 8-byte field. In these cases, the program must address a register with an even number R, containing the high-order bytes. Register R+1 will then contain the low-order bytes. Other instructions can process up to 16 general registers at one time.

3.2.2. Floating-point Registers

Four 8-byte floating-point registers numbered Ø, 2, 4 and 6 are provided for floating-point calculations. They can hold either a short, 4-byte or a long, 8-byte floating-point number. The short floating-point number is contained in the four high-order bytes of the register; during operations with short floating-point numbers, the four low-order bytes are ignored and remain unchanged.

In order to accomodate extended floating-point numbers, the registers Ø, 2 and 4, 6 can be combined to form a 16-byte field each.

3.2.3. Control Registers

The processor control information contained in the Program Counter Register (PCR), Interrupt Status Register (ISR), Interrupt Mask Register (IMR) etc. can only be altered by privileged instructions in the system state (see 9. Privileged Instructions).

3.2.4. Processor States

The central processor has four processing states. The state in which the processor is operating at any instant in time is termed the current processor state. Each processor state possesses a set of general and control registers and can thus control a program in the central processor independently of all the other states. The processor states are designated P1, P2, P3 and P4.

In P1 and P2, the complete set of general and control registers is available. In P3 and P4, the number of general registers is limited and several program-related control registers are not provided (see 5.3).

A set of floating-point registers is shared by all processor states.

3.3. Input/Output System

Before a program can be processed in the central processor, it must first be loaded with its relevant data from a peripheral device into main memory. The results of the program must then be output via a device. The connection between the central processor and the device is provided by I/O channels in the I/O system via the standard I/O interface to the device controller and thence to the device.

To relieve the central processor of peripheral device communication, the I/O system operates to a large extent independently of and parallel to program processing in the processor. It is activated by the central processor at the start of an I/O operation and, in the simplest case, only reports back after completion of its operation,issuing a completion message. The completion message leads to a program

interrupt, and the central processor starts a special operating system program which coordinates the terminated I/O operation with program processing.

3.3.1. Channels

The I/O channels can function either independently of the central processor, using their own control facilities, or be integrated in the central processor. Integrated channels which operate in the time-division multiplex mode use parts of the central processor for performing I/O operations. The logical function of the channels is independent of their implementation; only the maximum permissible data transfer rate for the I/O system is model-dependent.

There are three types of channel: byte multiplexor, block multiplexor and selector channels. A block multiplexor channel can function in the block multiplex or the selector mode. The byte and block multiplexor channels permit simultaneous processing of several I/O operations. The block multiplexor and selector channels in an I/O system permit considerably higher data transfer rates than the corresponding byte multiplexor channels.

Functionally, each multiplexor channel consists of several subchannels, and each of these subchannels may be assigned a peripheral device. A subchannel cannot execute several I/O operations simultaneously.

3.3.2. Standard I/O Interface

The channels communicate with the device controllers via the Siemens System 7.000 standard I/O interface. The data transfer format and the type and sequence of the control procedures via this interface are largely independent of the type of channel and the type of device controller.

4. Program Implementation

The processor processes the instructions of the program sequentially in ascending order of addresses. This procedure is primarily controlled by the Program Counter Register (PCR) and the Interrupt Status Register (ISR).
A change in sequential processing can be caused by branch instructions, interrupts or the instructions PC, EX and EXST.

4.1 Instructions

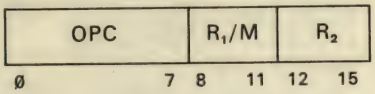
An instruction consists of two parts:

1. Operation code, which defines the operation to be performed.
2. Identification of the operands to be processed.

4.1.1. Instruction Types

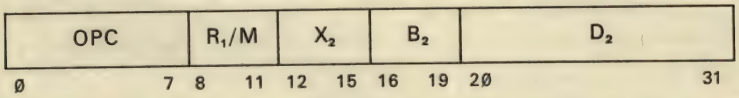
Instructions must be halfword aligned in main memory and are one, two or three halfwords long. There are five types of instruction, which differ in their methods of operand addressing.

1. RR Type:



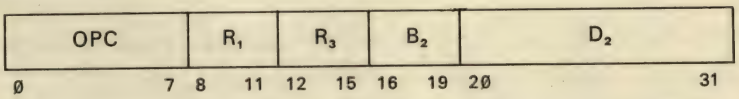
Both operands are located in registers.

2. RX Type:



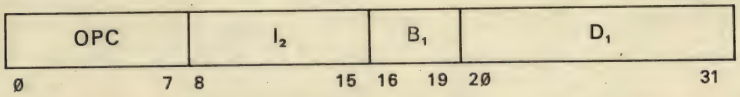
The first operand is in the register and the second in a memory location which is addressable via an index register.

3. RS Type:



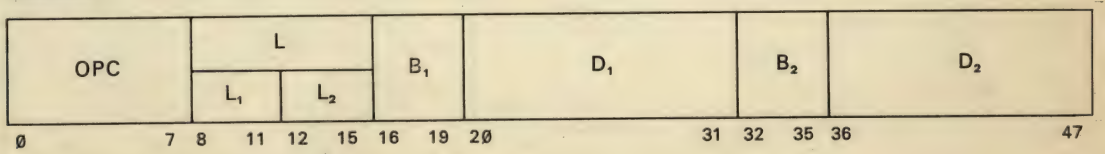
The first operand is in the register and the second in a memory location.

4. SI Type:



The first operand is in a memory location and the second is specified directly in the instruction.

5. SS Type:



Both operands are in memory locations.

The instruction formats are divided into fields whose lengths and meanings are defined as follows:

Field	Length	Meaning
OPC	8 bits	Operation code
R,M	4 bits	Identification of a general or floating-point register containing the operand, or a 4-bit mask
B	4 bits	Identification of a general register containing the base address
X	4 bits	Identification of a general register containing the index address
D	12 bits	Displacement address
I	8 bits	Immediate operand or 8-bit mask
L	4 bits/8 bits	Operand length specification

In this Reference Manual, four bits are always combined and represented as a hexadecimal digit.

The numbers below each format description indicate the number of the bit position.

Indices designate the operand to which the field is assigned.

The first byte of each instruction contains the operation code.

The two high-order bits indicate the format and thus the length of the instruction:

Bits Ø, 1	Format	Length
ØØ	RR	1 halfword
Ø1	RX	2 halfwords
1Ø	RS/SI	2 halfwords
11	SS	3 halfwords

In the case of the SI-type instructions Function Call (FCAL) and Execute Stack (EXST), the second byte in the instruction (I field) is also interpreted as part of the operation code.

Similarly, the data contained in the B2 and D2 fields in some RS and SS-type instructions is not designed to address a storage operand, but to provide a more accurate description of the operation.

The remaining instruction fields contain information for addressing the operands or, in the case of I2, are interpreted as immediate operands.

4.1.2. Operands

The operands required for instruction execution are located either in the main memory, the registers or directly in the instruction.

In SI-type instructions, the contents of the I field are used as the immediate operand. In some RR and RS-type instructions, the contents of an R field form the immediate operand.

Register operands are located in general, floating-point or control registers. General and floating-point registers are addressed by the R field of the instructions. The operand length is implied in the instruction and is usually 4 bytes in the case of general registers, and 8 bytes in the case of floating-point registers. Operands in floating-point registers are only addressed in the R field of floating-point instructions. With some instructions, the register operands in general registers can be masked byte by byte using a 4-bit mask.

In the following instructions, operands are processed in implicitly declared general registers: Load BitField (LBF), Store BitField (STBF), Translate and Test (TRT), and Edit and Mark (EDMK).

Operands in control registers are normally 4 bytes long and are addressed either implicitly or by the R field in privileged instructions.

Operands in main memory are either implicitly assigned a fixed length, or their length is coded in bytes in the 4 or 8-bit L field of the instruction, or it is specified in bytes as a 24-bit binary number in a register. The maximum length of a storage operand is thus 16 Megabytes. The LBF and STBF instructions can process 1 to 32-bit memory operands aligned on any bit boundaries. In the case of the Test under Mask instruction (TM), one byte in memory can be masked bit by bit using an 8-bit mask.

4.1.3. Main Memory Addresses

The information needed to compute the effective main memory addresses is contained in fields B, X and D of the instruction. D contains the 12-bit displacement, and B and X address general registers which contain the 24-bit base or index address in bit positions 8-31. To compute the memory address, the displacement is extended to the left with 12 leading binary zeros; the base address, displacement and, with RX-type instructions, the index address, are then added together as 24-bit binary numbers. Overflow, if any, is ignored.

If a field B = Ø or X = Ø, the address computation is performed with the base or index address = Ø, irrespective of the actual contents of general register Ø.

An instruction can use the same register for memory addressing and as an operand register; the operation is only executed after completion of the address computation. The effective address thus computed normally addresses the operand directly, but in the case of certain instructions, may also be the first of a chain with up to 5 levels of indirect addressing.

With the Move Long (MVCL) and Compare Logical Long (CLCL) instructions, the effective memory addresses are taken directly from the general registers specified by fields R1 and R2. In the case of branch instructions, binary and decimal shift instructions and in one special function of EXST, the effective address does not indicate a memory operand but is itself interpreted as an immediate operand.

4.2. Instruction Execution

The instructions of the program are normally processed in the order in which they are arranged in main memory, i.e. in ascending order of addresses. Instruction execution takes place in three stages:

1. The instruction is read that is indicated by the address in the current Program Counter Register (PCR).
2. The length of the instruction read is added to the instruction address, so that the PCR addresses the next instruction to be executed.
3. The instruction read is executed.

This normal form of sequential execution can be interrupted by branch instructions, subroutine calls and program interrrupts.

4.2.1. Program Branches

Branch instructions are provided to perform program branches. A branch can be performed:

1. For logical decisions on the basis of the result of the preceding instruction.
2. When executing a program loop as a result of the contents of general registers.
3. For subroutine calls, the return address being stored in a general register.

Logical decisions are made by the relevant branch instructions on the basis of the condition code set by the preceding instruction. The condition code is a 2-bit indicator which is set by the majority of instructions to summarize the result of the instruction execution. It permits 4 different indications, e.g. "result zero", "operand 1 high", "selected bits all ones", "channel busy", etc.

4.2.2. Subroutines

Subroutines are called by the Branch and Link (BAL, BALR) and Execute Stack (EXST) instructions.

EXST is particularly effective and convenient for supporting subroutine calls: subroutines can be called in several levels both under an address and, cataloged and supplied with additional control information, under a number. The information required for the return to the superordinate program level is entered in control and data stacks; the return itself requires simply one instruction. In the case of shareable, reenterable programs (common codes), work areas can be created in the data stack for the called program. Control and data stacks are available for every processor state.

4.2.3. Program Interrupts

Certain events during execution of a program automatically lead to a program interrupt. The events may either have been synchronously caused by the interrupted program (e.g. program error or paging queue), or they may occur independently of and asynchronously to the interrupted program (e.g. channel interrupt, console interrupt, timer interrupt).

The interrupt event automatically activates processor state P3 (or P4) and in this state initiates the interrupt analysis program. The interrupted program in the deactivated processor state remains operable, provided the interrupt was not caused by a program or machine error. The continuation data need not be saved. After the interrupt has been serviced, the interrupted program can be restarted by activating the interrupted processor state with the Program Control (PC) instruction.

A program interrupt only takes effect after termination or completion of an instruction and before commencement of a new (or the same) instruction. The Move Long (MVCL) and Compare Logical Long (CLCL) instructions form an exception to this rule and can be interrupted after completion of certain instruction phases. An interrupt cannot be effected after the Program Control (PC) instruction in the test mode.

If several interrupt events occur at the same time, the interrupts are processed according to a fixed order of priorities.

Program interrupts can be permitted or inhibited using bit masks assigned in the Interrupt Mask Registers (IMRs) of the processor states. However, a masked interrupt remains present and takes effect as soon as the mask allows. An additional 4-bit program mask (PM) in the Program Counter Register (PCR) permits suppression of the program errors "fixed-point overflow", "decimal overflow", "exponent underflow" and "significance error". When suppressed, the associated interrupts do not remain present.

4.3. Memory Access

With respect to processing of the program instructions by the central processor, a distinction must be made between the scheduled sequence and the actual sequence of processing. According to the scheduled sequence, the instructions are executed individually in the order in which they are stored in main memory. Each instruction is completed before processing of the logically following instruction commences. According to the scheduled sequence, a synchronous program interrupt (which has not been masked) also takes effect immediately after the instruction which caused it.

The actual sequence in which the instructions are processed can however be different: instructions can be prefetched, instruction execution can overlap with memory accesses of other instructions, or instructions can be processed in overlapped fashion. The result, however, always looks as if the instructions had been executed in the scheduled sequence. If necessary, this is ensured by means of an interference logic in the processor.

When, however, not only the program in one processor, but also the channel programs and programs in other processors within a multiprocessor system have to be considered,interference can occur in main memory between the programs and channel programs, since processors and channels share main memory. In the case of interference, it should be noted that one instruction can contain several memory requests, each request can consist of several accesses and that the actual sequence can vary from the scheduled one. During execution of an instruction, main memory is not reserved exclusively for the accesses of this instruction. (Exceptions are the Test and Set (TS), Test and Set Real (TSR), Compare and Swap (CS) and Compare Double and Swap (CDS) instructions).

5. Control

Central processor operation is controlled by

- The current central processor state
- The current program state
- The program instructions to be processed

The program state is determined by the program related data stored in the control and general registers.

Additional functions, such as memory protection and program monitoring, are provided to enhance the programmability and the reliability of the system. Six timers not only enable the program to read from an absolute time-scale with a high degree of accuracy, but also to measure the elapsing of prescribed time intervals. Externally initiated functions load the system, and issue diagnostic information.

5.1. Central Processor State

The operating state of the central processor can be divided into three independent components with the following values:

1. Machine state : Stop/Busy/Initial Program Load (IPL)
2. Processor state P1/P2/P3/P4
3. User or system status (non-privileged or privileged).

These components vary in the way they influence the processor and in the way they are changed.

5.1.1. Machine State

When in the Busy state, the processor processes instructions and takes interrupts. The Busy state is adopted after a successful IPL (or succesfully completed Ready function) or when the Start function is initiated. After IPL (or Ready) instruction processing commences with the instruction at real memory address Ø, and after Start, with the instruction to which the current PCR is pointing.

The Stop state is adopted after initiation of the Halt function in the Busy state and after every processor reset (see 5.7).

The machine state can change from Busy to Stop only at points of interruptibility, that is to say at the end of an instruction or an instruction phase.

If the Load function is initiated while the processor is in the Stop state, the IPL state is adopted (after processor hardware initialization) i.e. the specific functions for IPL (input operation, first instruction, cf. 5.7) are executed. After IPL, the processor is in P1 in the system status.

In multiprocessor systems, a Ready state similar to the IPL state (cf. 5.7) can be adopted.

The time-of-day clock runs independently of the machine state, but the other timers are confined to the Busy state (cf. 5.6).

5.1.2. Processor State

The four processor states are designated P1, P2, P3 and P4. In P1 and P2, user programs and program interrupts are processed and in P3 and P4, program interrupts are analyzed. At any point in time the processor will be in one of these four processor states, which will be called the current processor state.

Each processor state has its own set of general and control registers and can thus control a program in the central processor independently of all the other processor states. This permits efficient handling of program interrupts since no information has to be saved in memory when the interrupt handling routine is called or when control is returned to the interrupted program.

A number of collective registers shared by all the processor states are also available; these include the floating-point registers (cf. 5.3).

The processor is controlled by the registers of the current processor state. The individual registers of the inactive states have no effect on the processor.

When there is a program interrupt in P1 or P2, the processor automatically switches to P3 or P4 if the interrupt is due to a machine check, power failure or I/O error. Any interrupt in P3 is treated as a machine check and leads to P4. The return to the interrupted processor state is effected via the Program Control (PC) instruction.

All the timers run in P1 and P2; the interval timer and the program timers are deactivated for P3 and P4.

5.1.3. User and System Status

The processor's user status and system status differ in the number of valid instructions they will allow: in the system status (privileged), all instructions are valid, including the privileged and I/O instructions that can modify control registers, storage keys and timers, handle traffic between the central processors or start I/O activity; in the user status, only the non-privileged instructions are permissible: calling a privileged or I/O instruction results in a program interrupt.

The central processor is in the user status when bit 15 of control register ISR is set, and in the system status when this bit is reset.

A change from user status to system status can be made when a new processor state is activated by a PC instruction or by an interrupt.

The timers run both in the user status and in the system status.

5.2. Mode of Control

The central processor can operate either in the non-extended or extended I/O mode. The two modes vary in their provision for I/O system and channel configurations on the one hand, and in the interface for the exchange of information between central processor and I/O system on the other.

In the non-extended mode, only one I/O processor with a maximum of one byte multiplexor and six selector channels can be connected to the central processor. The address of the first channel command is in main memory. In the extended mode, up to four I/O processors with a maximum of 32 channels can be connected. The address of the first channel command is in a general register.

In a multiprocessor configuration all the central processors must operate in the extended mode.

5.3. Program State

The state of a program being processed by the central processor is determined by the contents of the general, control and, in certain cases, floating-point registers of the processor state assigned to that particular program. The contents of all the individual and collective registers of one processor state constitute the "program context", the contents of the individual registers, the "program status", and the contents of the individual control registers, the "reduced program status" (Fig. 5-1).

The term "register" gives no indication of the method of implementation; parts of the program context can be in main memory as well as being components of a faster semiconductor storage.

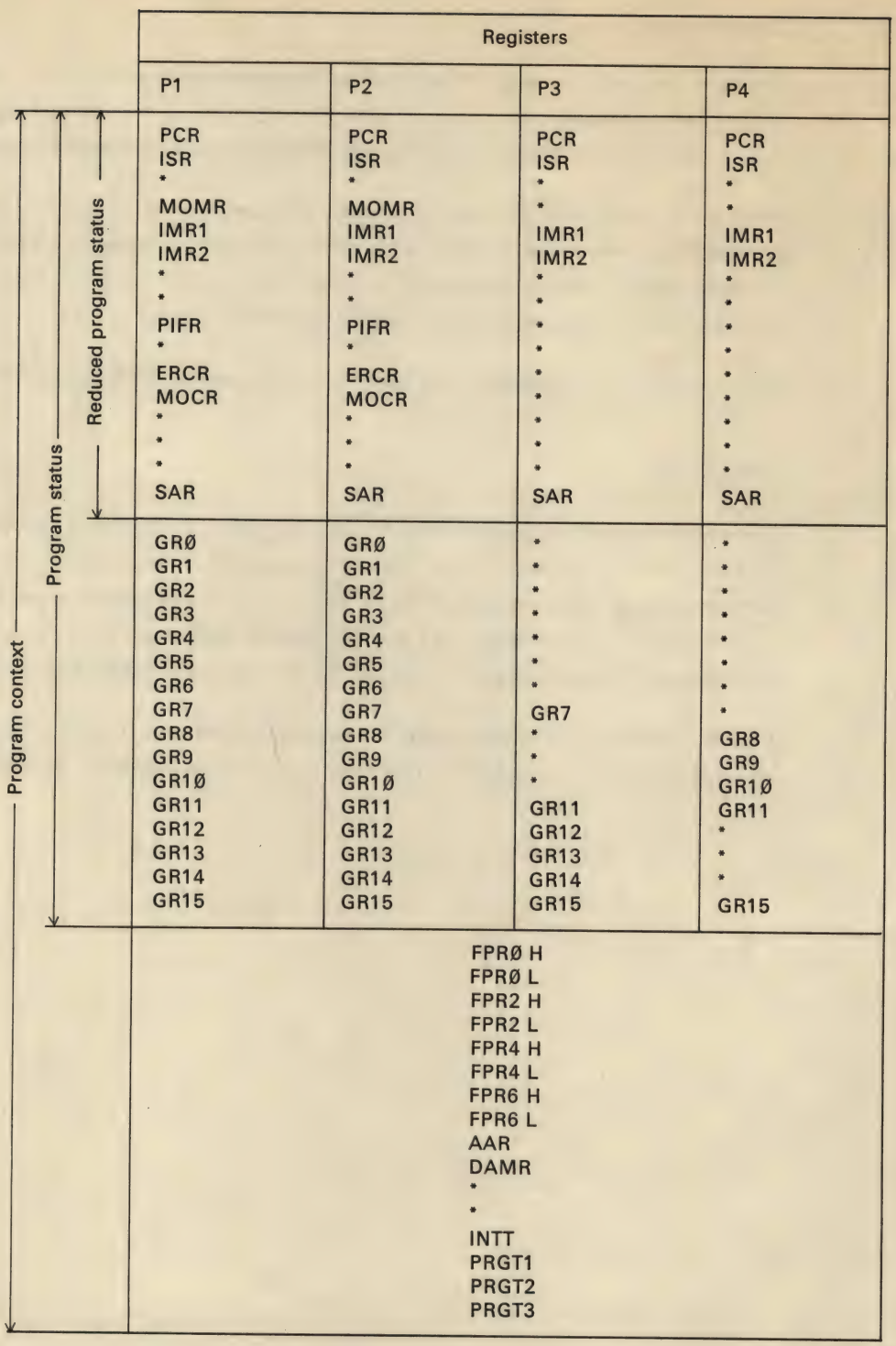


Fig. 5-1 Program Context

GR = General register

FPRH = Floating-point register, high-order word

FPRL = Floating-point register, low-order word

5.3.1. Program Status

The program status is determined by the contents of the individual general and control registers of a processor state (cf. Fig. 5.1). It is loaded and stored by means of the privileged Load Status of Program (LSP) and Store Status of Program (SSP) instructions. The program-related control registers PIFR, ERCR, MOCR and MOMR, and a number of general registers are available only in processor states P1 and P2. All the registers forming the program status are one word long (except the Interrupt Mask Register IMR).

The non-listed bits are not used.

Program Counter Register PCR

Bit no.	Name	Function
Ø-1	ILC	Instruction Length Code: ILC specifies in binary code the number of halfwords by which NIA (see below) must be decremented so that after an interrupt the last instruction executed can be addressed. Exception: ILC=Ø is not defined.
2-3	CC	Condition Code: The condition code provides summarized information on the result of the last instruction to change the condition code (cf. 4.2.1). The condition code can also be loaded with the non-privileged Set Program Mask (SPM) instruction.
4-7	PM	Program Mask: Bits 4-7 mask the fixed-point overflow, decimal overflow, exponent underflow and significance error interrupts (in that order). If a bit in the program mask is not set, setting of the bit in the PIFR corresponding to the interrupt is inhibited. The program mask is loaded with the non-privileged SPM instruction.
8-31	NIA	Next Instruction Address: Bits 8-31 contain the effective main memory address of the next instruction to be executed.

Interrupt Status Register ISR

Bit no.	Name	Function
Ø-2	ISI	Interrupted State Identifier: When there is a change in processor state due to an interrupt, ISI codes the interrupted state according to the following table: ISI | Interrupted state ØØØ | P4 ØØ1 | P3 Ø1Ø | P2 Ø11 | P1
4	AUDT	Audit Mode: When AUDT=1, the audit mode is on: in the event of a branch, the effective address of the instruction which caused the branch is stored in the 64-word audit table in main memory (cf.5.5.2).
6	PAWM	Permit Audit Wrap Interrupt Mask: When PAWM is set, an interrupt is requested as soon as the 64-word audit table is full. When PAWM=Ø, the audit table is overwritten starting at the lowest address (cf. 5.5.2.).
8-11	ISRKEY	Protection Key: Contains the 4-bit key for real memory protection (cf.5.4.1).
12	ISO	ISO Code: When ISO is set, decimal operands are coded in ISO code, otherwise in EBCDIC.
13	T	Translate Mode: When T=1, the address translation facility is on and effective addresses are treated as virtual addresses; when T=Ø, the address translation facility is off.
15	N	Non-privileged: When N=1, the processor is in the user status and only non-privileged instructions are executed. When N=Ø (system status), the entire instruction set is legal.
16	DAMC	Data Address Match Check: If DAMC is set, every data access in main memory causes the effective address to be compared with the address in the DAMR and, on a match, an interrupt to be requested (cf. 5.5.3).

Bit no.	Name	Function
17		Reserved for special use
18-19	RSI	Ring State Indicator: RSI contains the virtual memory protection 2-bit ring state (cf. 5.4.2).
20		Reserved for special use
21-23	EMULC	Emulator Control: If EMULC ≠ Ø, these bits identify the instruction set to be emulated.
24-31	CALL	Call Field: After execution of the Supervisor Call (SVC) instruction, this byte of the current ISR holds the contents of the R1/R2 field of SVC.

Program Interrupt Flag Register PIFR

When a bit in the PIFR is set, the associated interrupt is requested. The interrupt is taken if permitted by the corresponding mask bit in the IMR.

Bit no.	Interrupt
Ø	Test mode
1	Fixed-point overflow
2	Decimal overflow
3	Exponent underflow
4	Significance error
5	Divide error
6	Exponent overflow
7	Data error
8	Address error
9	Operation code trap
10	Privileged operation
11	Supervisor call
12	Paging queue
13	Paging error
23	Interval timer
24	Program timer 1
25	" " 2
26	" " 3
29	Data address match check

Additional non-program-related interrupts are flagged in the Interrupt Flag Register (IFR). This is a doubleword register not forming part of the program context. Bit positions occupied in the PIFR are not used in the IFR; bit positions occupied in the first word of the IFR are not used in the PIFR.

Interrupt Mask Register IMR

The Interrupt Mask Register is a doubleword register. Each mask bit masks an associated bit in the PIFR or IFR. If the mask bit is set, the interrupt can be taken; if the mask bit is not set, the interrupt request is held pending.

Error Cause Register ERCR

This register contains detailed information on error causes associated with certain program-related interrupts.

Bit no.	Meaning
Ø-7	Error class
8-15	Error subclass
19-31	Segment and page number

Monitor Mask Register MOMR

Bit positions 16-31 of the Monitor Mask Register contain the masks for 16 possible monitor classes which can be specified in the I2 field of the Monitor Call (MC) instruction.

Monitor Call Register (MOCR)

The Monitor Call Register contains the following information after successful completion of a Monitor Call instruction:

Bit no.	Information
Ø-7	Monitor Class
8-31	Monitor code (effective main memory address from the MC instruction)

Stack Address Register SAR

The Stack Address Register contains in bit positions 8-31 the 8-word-oriented effective address of the control stack header.

5.3.2. Program Context

The program context embraces the program status and also the four floating-point registers, the Audit Address Register, the Data Address Match Register and four timers. The registers are available to all processor states. The program context can be loaded and stored by means of the privileged Load Status of Program (LSP) and Store Status of Program (SSP) instructions.

The floating-point registers are doubleword registers consisting of a high-order part FPRH, and a low-order part FPRL. The format of the timers is given in 5.6.

Audit Address Register AAR

The Audit Address Register is a fullword register containing in bit positions 8-31 a real, word-oriented main memory address which points to a word in the audit table. After each instruction address entry, AAR is incremented by four (modulo 256).

Data Address Match Register DAMR

The Data Address Match Register is a fullword register containing an effective main memory address in bit positions 8-31. When DAMC is on, the 21 high-order bits of this address are compared with the corresponding bits of the addresses used for data accessing.

5

5.4. Memory Protection

Programs belonging to several users and system programs can be memory-resident at the same time. In order to exclude as far as possible erroneous or illegal interference between programs certain areas of memory, both on the real and the virtual level, can be protected. The only program with access to one of these areas is the one with the appropriate key in the ISR.

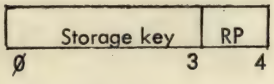
For the purposes of protection, main memory is arranged into:

1. 2-KB*) blocks positioned at 2-KB boundaries on the real level
2. Areas (up to 64 KB each) specified by page tables on the virtual level

5.4.1. Real Memory Protection

Each 2-KB block in real memory is assigned a 4-bit storage key. The storage key is set and tested by means of the privileged Set Storage Key (SSK) and Insert Storage Key (ISK) instructions.

The format is as follows:



*) 1 KB = 1024 bytes

Bit 4, the read-protect bit (RP), controls whether protection should be against just write accesses (bit 4=Ø) or against both read and write access (bit 4=1).

Bits Ø-3 constitute the actual 4-bit storage key. When the central processor accesses the memory, this key is compared with the 4-bit protection key in the ISR of the current processor state; in the case of I/O system access, it is compared with the 4-bit protection key in the Channel Address Word (CAW).

Access is permitted if

either protection key = ØØØØ

or storage key = ØØØØ

or storage key = protection key

or access is read and only write protection is in force (bit 4=Ø).

If none of the conditions is satisfied, a program interrupt is requested.

5.4.2. Virtual Memory Protection

Each area of protection in virtual memory covered by a page table is assigned a 2-bit ring number for operand read accesses and a 2-bit ring number for operand write accesses. These ring numbers are held in the segment table entry.

In the case of a central processor read or write access in the translate mode, the corresponding ring number is compared with the 2-bit-Ring State Indicator (RSI) in the current ISR. Ring protection is not effective when instructions are being read.

Access is permitted if

either RSI $\leq$ ring number

or an instruction is being read.

In all other cases, a program interrupt is requested.

5.5. Program Monitoring

The central processors of the Siemens System 7.000 aid the programmer both in monitoring the program run and in fault location.
Facilities are available for:

- Tracing the program run on the basis of markers in the program
- Recording the branches performed to facilitate program retracing
- Monitoring access to certain data
- Executing the program in the absence of fixed-length memory operand alignment
- Measuring run times and frequencies

Program monitoring can however considerably reduce processor efficiency.

5.5.1. Monitor Call

When the monitor class in the I2 field of the instruction matches one of the classes specified in the Monitor Mask Register (MOMR), the Monitor Call (MC) instruction causes the monitored program to be interrupted and control to be passed to the monitoring program. The monitor class and the 24-bit monitor code are made available to the monitoring program in the Monitor Call Register (MOCR). If the bit corresponding to the monitor class is not set in the MOMR, the instruction has no effect.

It is thus possible to trace a program run or to measure run times, frequencies or similar by inserting Monitor Call at appropriate points in the program to be monitored.

5.5.2. Audit Mode

To implement the audit mode a continuous 64-word area of main memory known as the audit table can be assigned, in which the instruction addresses of all instructions involving branches are entered. The audit table is addressed via the Audit Address Register (AAR); after each entry the address is automatically incremented by four.

When bit 4 of the current ISR is set, the audit mode is on and the addresses are entered in the audit table.

If the audit table is full, either the address in the AAR is automatically set to the start of the table (when bit 6=Ø) or an interrupt is requested (when bit 6=1).

An entry is made in the audit table when branches are performed with the following instructions:
BC, BCR, BAL, BALR, BCT, BCTR, BXH, BXLE, EXST.

5.5.3. Data Address Match Check

When bit 16 in the current ISR is set, the 21 high-order bits of each effective data address are compared with the corresponding bits in the Data Address Match Register (DAMR). If there is a match, a program interrupt is requested at the end of the instruction.

The comparison handles all data addresses on a doubleword basis. Read accesses to instructions are not monitored.

5.5.4. Test Mode

In the test mode, a program interrupt is forced after execution of each instruction in the monitored program. For this purpose, the program state is changed to the monitored program by the Program Control (PC) instruction in the test mode. A program interrupt is not possible after the PC in the test mode; after the subsequent instruction, however, the test mode interrupt is requested.

5.5.5 Byte Boundary

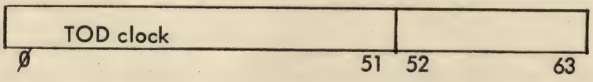
The fixed-length memory operands of the privileged and non-privileged instructions must be aligned, for example a word must be located on a word boundary.

5.6 Timers

5.6.1 Time-of-Day Clock

The TOD clock provides the program with an absolute time-scale with a resolution of 1 μs and a range of over 140 years.

It is in the form of a binary counter with the following format:



The TOD clock can be in the not-set, set or not-operational state. It is set by the privileged special function Set Clock (SCK) and can be read by means of the non-privileged Store Clock (STCK) instruction. Unintentional setting of the TOD clock can be prevented by a manual security switch which "blocks" the clock. Prior to the first successful SCK, the clock is in the not-set state.

When the TOD clock is operational, the contents of bit position 51 are incremented by one every μs, irrespective of the central processor operating state. Bit position 31 is then incremented approximately every 1.05 seconds; the high-order word of the TOD clock corresponds roughly to seconds.

A carry propagated out of bit position Ø is ignored; the clock starts counting again from zero.

The clock will drift no more than 10 seconds a day.

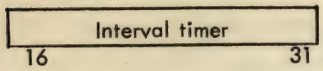
It does not generate a program interrupt.

In a multiprocessor configuration, only one TOD clock is active.

5.6.2. Interval Timer

The operating system can load the interval timer INTT with a value corresponding to a time from 0.0001 to 6.5536 s. The elapse of this time is marked by a program interrupt.

The interval timer is a binary counter with the following format:



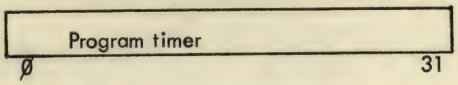
The counter is decremented by one every 100 µs providing that the processor is in P1 or P2 and the value of the interval timer is not equal to zero. If the interval timer assumes a value of zero, a program interrupt is requested.

The interval timer can be loaded and stored by means of the privileged LSP and SSP instructions.

5.6.3. Program Timers

Each central processor is equipped with three program timers. The operating system can load them with a value corresponding to a time from 0.0001 s to approximately 120 h. The elapse of this time is marked by a program interrupt.

A program timer is a binary counter with the following format:



The counter is decremented by one every 100 µs providing that the processor is in P1 or P2 and the value of the program timer is not equal to zero.
If a program timer assumes a value of zero, a program interrupt is requested.
The program timers can be loaded and stored by means of the privileged LSP and SSP instructions.

5.6.4. Elapsed Time Clock

The Elapsed Time Clock (ETC) provides the central processor with an interrupt request at fixed intervals of one second. The ETC is always running when the processor is Busy. The ETC is not accessible to the program.
In a multiprocessor system, an ETC is implemented in each processor and the ETCs run independently of one another.

5.7. External Initiated Functions

This section contains descriptions of several functions which are directly initiated by the operator. They include central unit reset and load, and saving diagnostic information after operating system failure.

5.7.1. Reset

Machine-dependent functions, such as General Reset and System Reset, are provided for resetting the system.
After central unit power-on, General Reset is required to initialize the central processor, I/O system and main memory system so that instructions and channel commands can be executed. This includes, for example, resetting error flags, supplying the processors and main memory with valid check bits, re-

setting the hardware control below the level visible in the program context, resetting the control-register and main-memory contents, resetting the TOD clock, etc.
The System Reset function is used, for example, in the stop state after a machine check; it is not intended to affect the information contained in the program context. Only the error flags, the check bits and the hardware control are reset.

5.7.2 Initial Program Loading (IPL)

The IPL function places the central processor in the IPL state. In this state, an input operation from an external device, e.g. a disk storage, is executed first, followed by the instruction located in main memory at address zero. After successful completion of these activities, the central processor changes to the Busy state.

The IPL function is activated after the relevant machine-dependent initialization has been performed (powering on, resetting), and internally prepares the central unit for the IPL state:

1. Processor state P1 is adopted.
2. In P1, the PCR, ISR, PIFR and IMR are loaded with zero. Other control registers are loaded with machine-dependent values or, if necessary, are provided with valid check bits.
3. The control mode is set according to the channels permitted for IPL on the machine in question.
4. The error recovery mode is set (see 11. Machine Error Reporting and Recovery).

5.7.3 Diagnostic Initial Program Loading

Diagnostic IPL differs from normal IPL in that the first 256 bytes in main memory and the entire program context together with additional, to some extent machine-dependent information from the central processor and the I/O Processor, are all saved in a specific main memory area before this information is modified or reset. The address of the save area is made available to the program by means of a general register.

In a multiprocessor system, first the information from the central processor responsible for the diagnostic IPL is entered in the save area, then the information from the assigned I/O processor, and finally, in the same format, the information from the other processors.

5.7.4 Ready Function

In a multiprocessor system, the Ready function places the processor which did not perform the IPL procedure in a state similar to the IPL state. In contrast to the IPL state however, the second processor remains passive until the Alert CPU (ACPU) instruction causes it to execute the instruction located at real address zero in main memory.

6. Dynamic Address Translation

6.1. Virtual Memory

Dynamic address translation facilitates implementation and effective utilization of virtual memory. The virtual memory concept makes available to the user a far greater addressable working storage capacity than could be provided by the physical main memory. The maximum address space of the virtual memory is 16 Megabytes (16,777,216 bytes) but smaller memory sizes may also be specified. As far as the program is concerned the virtual memory is accessed with the appropriate addresses in exactly the same way as the real memory so that it appears to the user as if he has a normal main memory. Of course the program and the associated data, or at least those parts currently required, must be present in the physical main memory at execution time, i.e. the contents of the virtual memory have to be loaded into main memory.

The principle of virtual memory only requires that those program portions currently required for execution be present in main memory; the remainder of the program can reside on an external storage, only being brought into main memory little by little, keeping pace with program execution, or at the very latest when required by the program flow itself. This method whereby the contents of main memory are continually changing in accordance with the state of the program flow makes for more efficient use of the available free memory space since new program portions constantly replace those already processed or not immediately required. However, the user is not faced with any special programming restrictions or considerations since any program portions modified by program execution are copied back into the external storage before being overwritten by new program sections. If a particular part of a program should be needed again later, it can be recalled into main memory without any difficulty.

Virtual memory and physical main memory are divided into equal sized sections, referred to as pages. The program is loaded from virtual memory into main memory page by page. From a technical viewpoint the assignment of virtual pages to real pages is completely free, i.e. any virtual page may be loaded into any real page. Tables set up in main memory specify for each virtual page which real page it is allocated to and whether the page is already present in main memory. The tables also contain information concerning any program access restrictions that may exist relating to certain pages, and indicate whether the program has accessed a page or modified the contents of a page.

The actual relationship established between virtual and real pages is of no consequence to the program since the addresses relating to virtual memory are retained within the program. The program addresses the main memory through virtual addresses. Then during memory accessing each virtual address is translated into the associated real address pinpointing the actual main memory location addressed through the virtual address. Address translation is performed automatically during program execution with the aid of the previously mentioned tables and without program support. A detailed description of the address translation flow and the tables used is contained in the following.

6.2. Real and Virtual Memory Addressing

There are two methods by which the program can address main memory : real memory addressing and virtual memory addressing. With real addressing the addresses used by the program relate directly to main memory. With virtual addressing these addresses relate to the virtual memory, which resides on an external storage medium and is mapped in main memory at the time of access, and must be translated into the associated real addresses prior to access.

All memory addresses have a standard length of 24 bits. They contain no indication as to whether they are to be regarded as real or virtual memory addresses. Hence all memory addresses are termed effective addresses. A single bit, known as the T bit, in the Interrupt Status Register ISR specifies how an effective address is to be interpreted. T=Ø signifies real memory addressing with each effective address representing a real address.
T=1 signifies virtual memory addressing where each effective address represents a virtual address which has to be translated.

6.3. Address Translation Mechanism

For mapping the virtual memory in main memory, both memories are divided into pages. Each page contains 2 KB (2,Ø48 bytes) and will always begin at a 2-KB address boundary. Mapping takes place on a page-for-page basis, i.e. the byte addresses are identical within a virtual page and the associated real page. Hence during address translation only the virtual page number is replaced by the associated real page number. This is performed with the aid of tables which are set up in main memory.

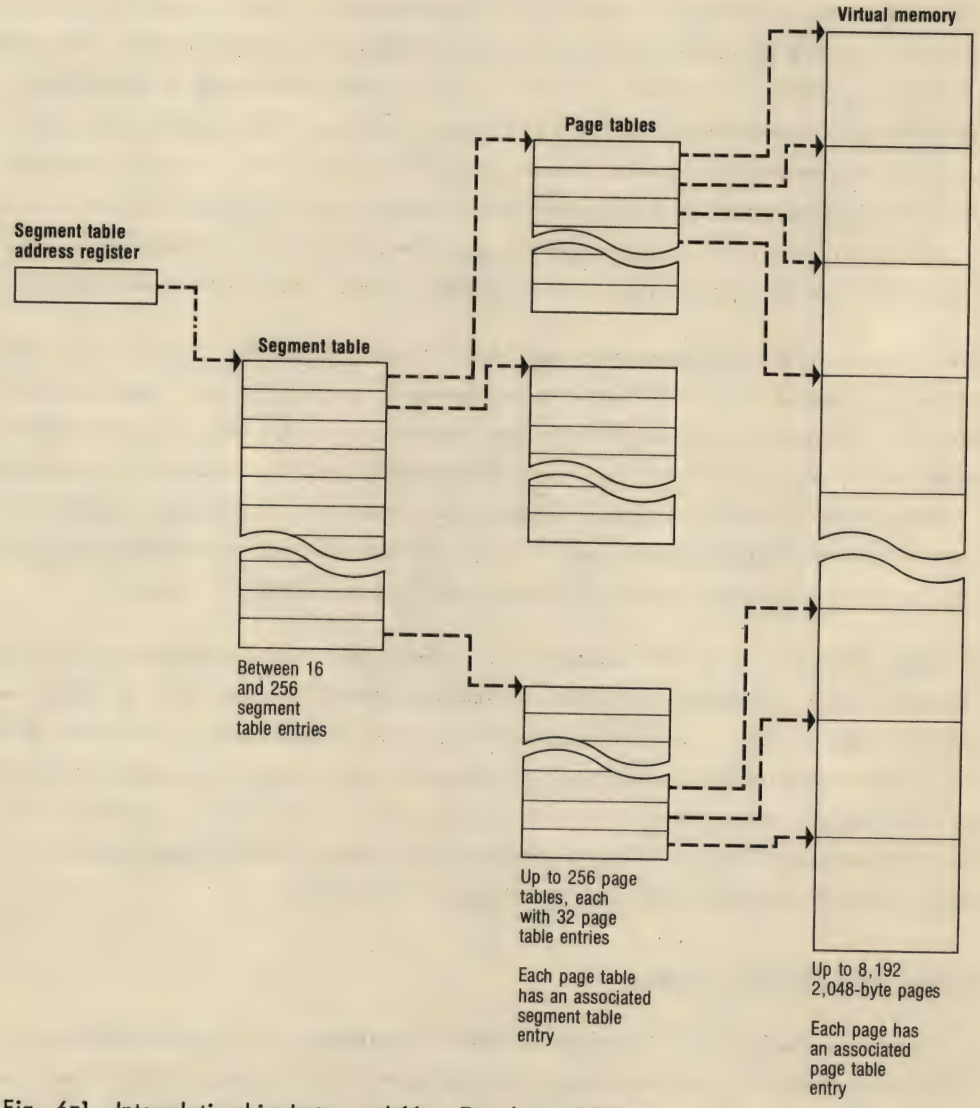


Fig. 6-1 Interrelationships between Address Translation Tables

For each group of 32 consecutively addressed pages in virtual memory, representing an address space of 64 KB (65,536 bytes) and termed a segment, a table known as a page table is set up. For each of the 32 virtual pages the table contains the associated real page number, if one has been allocated, and information as to whether the page is present in main memory and whether the page has been accessed or its contents modified. The number of page tables set up is variable, as is their position in main memory. This information is defined in a further, superordinate table, the segment table.

The segment table is responsible for at least 16 consecutively addressed segments in virtual memory, representing an address space of 1 MB (1,Ø48,576 bytes). The address space controlled by the segment table can be increased in 1-MB stages to the maximum of 16 MB (16,777,216 bytes).
For each segment defined the table contains the start address of the associated page table along with information concerning possible access restrictions relating to the particular segment. The position of the segment table in main memory is variable. The Segment Table Address Register STAR contains the start address of the segment table. The interrelationships between the tables are illustrated in Fig. 6-1. During address translation, access must be made to the segment table and to the associated page table.

For purposes of address translation the virtual address is subdivided into segment number (VSEG), page number (VPAG) and byte location number (LOC) :

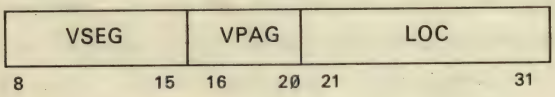
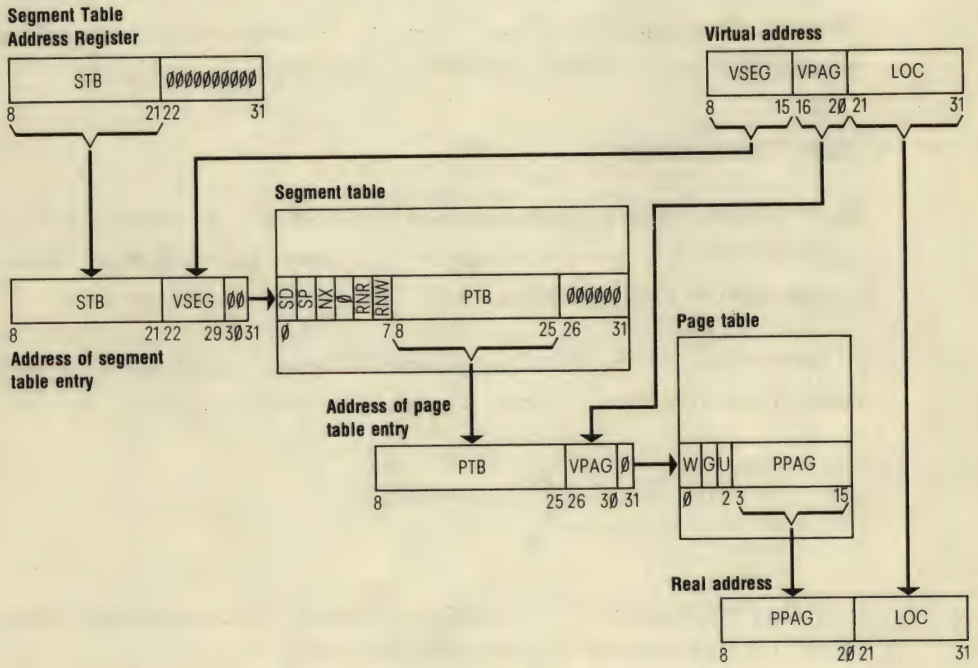


Fig. 6-2 Address Translation Schematic



The address of the associated segment table entry is obtained by inserting the segment number VSEG in the segment table start address contained in the Segment Table Address Register. This segment table entry contains the start address of the associated page table. The address of the page table entry is produced by substituting the page number VPAG in this page table start address. This page table entry contains the associated real page number. Combination of the real page number and the byte location number LOC gives the complete real address. The address translation concept is illustrated in Fig. 6-2. A detailed description of the entire process follows in 6.6.

6.4. Virtual Memory Protection

For memory protection purposes the entire virtual address space can be subsetted at the segment level into four areas of protection called rings. Each segment is identified in the corresponding segment table entry by a 2-bit ring number for read accesses (RNR) and a 2-bit ring number for write accesses (RNW), which identify the subset of the address space (protection ring) to which the segment belongs. The two ring numbers selected may be different whereby the segment can belong to two different protection rings depending on whether a read or write access is performed.

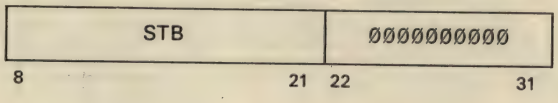
Subsetting of the virtual address space into areas protected or unprotected against access provides four levels of privilege, determined by the value of the Ring State Indicator RSI. The lowest RSI value (highest level of privilege) permits access to the entire address space, i.e. in all four protection rings, while the highest RSI value (lowest level of privilege) permits access only to the smallest subset, a single protection ring. The RSI occupies 2 bits in the Interrupt Status Register ISR. It is loaded by the program.

During each read or write access, the RSI is compared with the appropriate ring number of the segment concerned. Access is permitted if the RSI is less than or equal to the ring number of the referenced segment; otherwise, an interrupt condition due to a protection error is flagged.

6.5. Address Translation Tables

The segment table and the page tables defined therein are set up in main memory. The start address of the segment table is contained in the Segment Table Address Register STAR and the start addresses of the page tables are held in the segment table.

The Segment Table Address Register can be loaded and stored by the program. The segment table address STA contained in it, a 24-bit real address, has the following format:

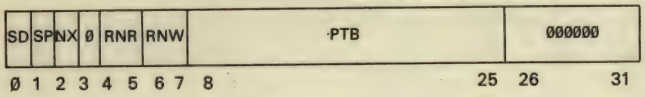


Bits 8-21, the high-order bits of the segment table address, are called segment table base STB.
Bits 22-31 must be zero, i.e. the start of the segment table is always aligned on a 256-word boundary.

If the program attempts to load the Segment Table Address Register with an address which does not have the correct alignment or is outside available main memory, an interrupt condition due to a system paging error is generated.

A segment table entry is 32 bits long. A segment table comprises between 16 and 256 such entries and is thus between 64 and 1,Ø24 bytes long. The length of the segment table is indicated by the Segment Table Length Register STLR. This register can be loaded and stored by privileged instructions.

The segment table entry contains the start address of the associated page table and also control bits. It represents a segment in the virtual memory and has the following format:



Bit Ø

Bit Ø indicates whether the segment (the associated page table) is defined (SD=1) or undefined (SD=Ø).

Bit 1

Bit 1 indicates whether the segment (the associated page table) is present in main memory (SP=1) or not (SP=Ø).

Bit 2

Bit 2 indicates whether execute accesses are permitted in this segment (NX=Ø) or not (NX=1).

Bit 3

Bit 3 must be zero.

Bits 4-5

These bits contain the ring number for read accesses and specify in conjunction with the Ring State Indicator RSI whether read accesses to data in this segment are permitted (RSI $\leq$ RNR) or not (RSI $>$ RNR).

Bits 6-7

These bits contain the ring number for write accesses and specify in conjunction with the Ring State Indicator RSI whether write accesses are permitted in this segment (RSI $\leq$ RNW) or not (RSI $>$ RNW).

Bits 8-25

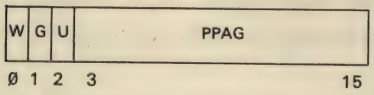
These are the high-order bits of the page table address and are called page table base PTB.

Bits 26-31

These bits must be zero.

The start address of the page table, contained in bits 8-31, is a real address. The page table always starts at a 16-word boundary and is 64 bytes long. It comprises 32 page table entries, each 16 bits long. The page table entry contains the real page number and control bits.

It has the following format:



Bits Ø and 2

These bits indicate as follows whether the page is defined, whether it is present in main memory and whether it has been written into :

W	U	
Ø	Ø	Page not present in main memory
1	Ø	Page undefined
Ø	1	Page not written into
1	1	Page written into

Bit 1

Bit 1 indicates whether the program has accessed the page (G=1) or not (G=Ø).

Bits 3-15

These bits contain the real page number PPAG.

6.6. Address Translation Flow

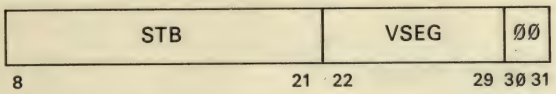
Dynamic address translation starts with the complete effective address and ends up by producing the associated real address for the physical access to main memory.

Halfword, word or doubleword alignment can be defined for effective addresses and this can be specifically checked when the addresses are used. If a checked address is found not to have the required alignment, address translation is not performed; instead, an interrupt condition due to a user addressing error is flagged (bit 8 is set in the Program Interrupt Flag Register PIFR and error class ØC is entered in the Error Cause Register ERCR). An interrupt condition due to incorrect alignment has priority over all other interrupt conditions which might occur during translation of this address.

The translation flow is as follows: If the T bit in the Interrupt Status Register ISR is not set, then the effective address is already a real address and no further action is required. If the T bit is set, then the effective address will be interpreted as a virtual address, unless it is specifically defined as a real address, and translated into the associated real address using the segment and page tables.

Before the segment table is accessed, a check is made as to whether the virtual address belongs to the address space defined by the segment table. The contents of bits 8-11 of the virtual address may not be greater than the contents of the Segment Table Length Register STLR, otherwise the virtual address is undefined and an interrupt condition due to a user existence error is generated.

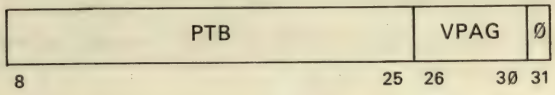
The address for access to the segment table entry is formed as follows by appending the segment number VSEG (bits 8-15 of the virtual address) to the segment table base STB (bits 8-21 from the Segment Table Address Register STAR) :



The following checks are made after the segment table entry has been accessed :

- Is the segment defined (SD=1) ? If not, an interrupt condition due to a user existence error is generated.
- Is access to the segment permitted ? That is, the NX bit must be zero for an execute access, and the ring number for read accesses RNR or write accesses RNW must be not less than the Ring State Indicator RSI for a read access to data or a write access respectiveley. If not, an interrupt condition due to a protection error is generated.
- Is the segment (the page table) present in main memory (SP=1) ? If not, an interrupt condition due to paging queue is generated.
- Are bits 3 and 26-31 zero ? If not, an interrupt condition due to a system paging error is generated.

The address for access to the page table entry is formed as follows by appending the page number VPAG (bits 16-2Ø of the virtual address) to the page table base PTB (bits 8-25 from the segment table entry) :

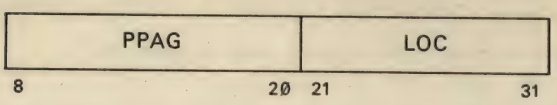


If this address is outside available main memory, an interrupt condition due to a system paging error is generated.

After the page table entry has been accessed, a check is made as to whether the page is defined and is present in main memory. If not, an interrupt condition due to a user existence error or paging queue is generated.

When a page is accessed for the first time, the G bit is set and when a page is written into for the first time, the W bit is also set, i.e. when U=1 and G=Ø, the G bit is set; when U=1 and W=Ø and a write access is made, the W bit is set.

The complete real address is formed by appending the byte location number LOC (bits 21-31 of the virtual address) to the real page number PPAG (bits 3-15 from the page table entry):



Provision of the real address for access to the physical main memory completes the address translation process.

6.7. Program Interrupts due to Address Translation Errors

Interrupt conditions can occur during address translation due to errors in the address translation process or due to paging. Unless otherwise specified, a paging queue interrupt will always lead to suppression of the current instruction. Each of these interrupt conditions is flagged by a bit set in the Program Interrupt Flag Register PIFR and classified more precisely by an entry in the Error Cause Register ERCR. A number of interrupt conditions may exist simultaneously during an address translation process but only the highest priority interrupt is flagged and serviced. As soon as an interrupt condition can be determined, address translation is terminated and subsequent accesses to segment or page table are not then carried out. Neither are the G and W bits set. In the following these interrupt conditions are listed and described in the order of their priority. The bit set in the Program Interrupt Flag Register PIFR and the entry made in the Error Cause Register ERCR are also given. The segment and page number (bits 8-20 of the virtual address concerned) is entered in the error register for all the following interrupt conditions.

1. User Existence Error

   PIFR bit 13; ERCR class 08, subclass 04
   The contents of bits 8-11 of the virtual address are greater than the contents of the Segment Table Length Register STLR, i.e. the virtual address is outside the virtual memory defined by the segment table.

2. User Existence Error

   PIFR bit 13; ERCR class 08, subclass 08
   The SD bit in the referenced segment table entry is zero, i.e. no page table exists for the virtual address.

3. Protection Error

   PIFR bit 13; ERCR class 04, subclass 00
   The NX bit is set in the referenced segment table entry and an execute access is to be performed; or, the ring number for read accesses RNR or the ring number for write accesses RNW is less than the Ring State Indicator RSI and a read access to data or a write access, respectively, is to be performed.

4. Paging Queue

   PIFR bit 12; ERCR class 3Ø, subclass ØØ
   The SP bit in the referenced segment table entry is zero, i.e. the segment (the page table) is not present in main memory.

5. System Paging Error

   PIFR bit 8; ERCR class 1Ø, subclass ØØ
   In the referenced segment table entry, the must-be-zero bits 3 and 26-31 are nonzero.

6. System Paging Error

   PIFR bit 8; ERCR class 1Ø, subclass ØØ
   The page table address lies outside available main memory.

7. User Existence Error

   PIFR bit 13; ERCR class Ø8, subclass ØC
   In the referenced page table entry, the U bit = Ø and the W bit = 1, i.e. no real page exists for the virtual address.

8. Paging Queue

   PIFR bit 12; ERCR class 3Ø, subclass ØØ
   In the referenced page table entry, the U bit = Ø and the W bit = Ø, i.e. the associated page is not present in main memory.

6.8. Handling of Tables and Buffers for Table Entries

Setting up, management, update and deactivation of address translation tables are performed exclusively by the operating system. The only influence exerted by the address translation facility on the contents of tables consists in the setting of the G and W bits.

Depending on the particular model, a central unit may be provided with buffers for translation table entries which serve to reduce continuous accesses to tables in the case of frequently used pages. A table update with a normal storage instruction may result in the table and the buffer containing different information. Hence special privileged functions are available to the program managing the address translation tables which make it possible in multiprocessor configurations to coordinate the other central processors with respect to the impending table update, to manipulate bytes in the address translation tables, and to clear any buffers for translation table entries in all central processors.

7. Program Interrupts

7.1. Execution of Program Interrupts

When a program interrupt occurs, the Program Counter Register (PCR) and the other registers assigned to the current program must be saved, the information on the cause of the interrupt loaded, and new registers including the PCR activated. Program processing continues with the instruction addressed by the new PCR.

The registers are saved principally by switching processor state. Each central processor is equipped with 4 processor states, each of which has a PCR and a set of other registers. There are also a number of registers which are shared by all of the processor states (see 3.2). The processor states are independent of the central unit's operating state, i.e. only one processor state can be effective at any one time regardless of whether the central unit is in the processing state or in the stop state. For example, if the central processor is in processor state P1 and a program is being run, the P1 registers will automatically be addressed.

There are two possible ways of altering the processor state:

1. By means of the privileged Program Control (PC) instruction.
2. Automatically, by means of a program interrupt.

The PC instruction can be used to switch to any one of the four processor states P1, P2, P3 and P4. A program interrupt, however, will only switch to P4 (in the case of machine errors) or P3 (for all other types of interrupt conditions). The tasks performed in the four processor states are as follows :

- P1 - normally execution of user programs
- P2 - execution of system program segments
- P3 - analysis of program-related interrupts
- P4 - analysis of machine checks

Program processing,including program interrupts, is thus based on the following principle: a user program runs in processor state P1; if an interrupt condition occurs, the user program is automatically interrupted, i.e. P3 (P4) is activated, resulting in deactivation of the P1 registers including the PCR; the next instruction to be executed is addressed by the P3 (P4) PCR and forms part of the interrupt analysis. This analysis results in the calling of the requisite system programs in P2 to handle the interrupt condition, if, after termination of these programs, no further interrupt requests are pending, P1 is reactivated and the user program continues.

7.1.1. Interrupt Conditions

The causes of the interrupt conditions are listed in Table 7-1. There are two classes of interrupt :

Machine errors, which cause a change to P4, and all other errors which switch to P3. Each interrupt condition is identified by a bit in a register.

A distinction is also made between program-related interrupts, which are stored in the Program Interrupt Flag Registers (PIFR) for P1 and P2, and system-related interrupts, which are stored in the Interrupt Flag Register (IFR).

Table 7-1 Interrupt Conditions

Name	Cause	PIFR bit	IFR bit	Switch to	Priority	Weight	Pre-mask	IMR mask
Test mode	PC instruction	Ø		P3	64	7C		Ø
Fixed-point overflow	Instruction flow	1		P3	63	78	PCR(4)	1
Decimal overflow		2		P3	62	74	PCR(5)	2
Exponent underflow		3		P3	61	7Ø	PCR(6)	3
Significance error		4		P3	6Ø	6C	PCR(7)	4
Divide error		5		P3	59	68		5
Exponent overflow		6		P3	58	64		6
Data error		7		P3	57	6Ø		7
Address error		8		P3	56	5C	ISR(6)[1]	8
Operation code trap		9		P3	55	58		9
Privileged operation		1Ø		P3	54	54		1Ø
Supervisor call		11		P3	53	5Ø		11
Paging queue		12		P3	52	4C		12
Paging error		13		P3	51	48		13
Console interrupt	Console		14	P3	5Ø	44		14
Elapsed time clock	Timer		15	P3	49	4Ø		15
Channel requests	I/O		16	P3	48	3C		16
			17	P3	47	38		17
			18	P3	46	34		18
			19	P3	45	3Ø		19
			2Ø	P3	44	2C		2Ø
			21	P3	43	28		21
			22	P3	42	24		22
Interval timer	Timer	23		P3	41	2Ø		23
Program timer 1	Timer	24		P3	4Ø	1C		24
Program timer 2		25		P3	39	18		25
Program timer 3		26		P3	38	14		26
I/O processor check	I/O		27	P4	5	1Ø		27
Data address match check	Instruction flow	29		P3	3	Ø8	ISR(16)	29
Machine check	Hardware		3Ø	P4	2	Ø4		3Ø
Power failure	Power supply		31	P4	1	ØØ		31

1) Only applies to audit wrap

Table 7-1 Interrupt Conditions (contd)

Name	Cause	PIFR bit	IFR bit	Switch to	Priority	Weight	Pre-mask	IMR mask
Channel requests	I/O		32	P3	37	FC		32
			.	.	.	.		.
			.	.	.	.		.
			.	.	.	.		.
			.	.	.	.		.
			.	.	.	.		.
			56	P3	13	9C		56
Not specified			57					
Not specified			58					
CPU Alert	Central processor		59	P3	1Ø	9Ø		59
Not specified			6Ø					
I/O processor check	I/O processors		61	P4	8	88		61
			62	P4	7	84		62
			63	P4	6	8Ø		63

When a program interrupt is performed, the corresponding interrupt flag is reset and general register 15 of the new processor state is loaded with a corresponding weight as to indicate each interrupt condition. In the case of program errors, additional information is stored in the Error Cause Register (ERCR).

7

7.1.2. Interrupt Masking

Interrupt conditions can be masked in the central processor, i.e. they do not necessary lead to a program interrupt. Each interrupt condition is assigned a bit in the Interrupt Mask Register (IMR), an interrupt only being permitted when the corresponding mask bit = 1. If the mask bit = Ø, a program interrupt is not performed. This does not mean that the interrupt condition is lost; it remains set in the Program Interrupt Flag Register and can result in an interrupt after the mask has been lifted.

This method of masking applies to all interrupt conditions. An additional form of masking is available for certain program errors (pre-masking), where the interrupt condition is lost before it can be set in the Program Interrupt Flag Register if it is not reported by the condition code. The following errors can be suppressed by the program mask in the PCR : fixed-point overflow, decimal overflow, exponent underflow and significance error. Data address match checks, on the other hand, are suppressed by one bit in the Interrupt Status Register (ISR). A special type of address error, audit wrap, can also be masked by a bit in the ISR.

7.1.3. Interrupt Timing

A program interrupt is permissible only between two instructions. Interrupt conditions normally occur during execution of an instruction but instead of causing an immediate program interrupt they are held pending until the current instruction is concluded.

This does not apply to interruptible instructions, which can be interrupted after partial execution. An interruptible instruction cannot however be interrupted at any point : only at the end of a phase in instruction execution. Each phase thus forms a non-interruptible instruction.

7.1.4. Instruction Execution

7.1.4.1. Types of Instruction Conclusion

The execution of an instruction is always concluded in one of the following four ways :

- Completion
- Termination
- Suppression
- Nullification

If instruction execution is completed, the instruction will have been executed through to its conclusion with correct results. The address in the PCR indicates the next instruction to be executed.

When an instruction is terminated, execution ends at a random location, usually with unusable results. The address in the PCR indicates the next instruction to be executed.

If an instruction is suppressed, no operation is performed other than that the PCR is incremented by the length of the instruction and the instruction length code is set. That is to say, the contents of any result fields, including the condition code, are not changed and the instruction can be retried with the same operands after the PCR has been reset.

Instruction nullification is the same as suppression except that the PCR is automatically decremented by the length of the instruction. Instructions can only be nullified upon completion of a phase of an interruptible instruction.

Note :

Updating an interruptible instruction is not classed as changing result fields since instruction execution can subsequently be performed as though no operands had been changed.

7.1.4.2. Execution of Interruptible Instructions

Execution of an interruptible instruction is completed when all the (non-interruptible) phases of the instruction have been concluded.

If a phase of an interruptible instruction can be executed through to its conclusion because no program error has occurred to result in suppression or termination, the instruction is nullified, i.e. the address in the PCR indicates the interruptible instruction.

If a phase of an interruptible instruction is suppressed or terminated, the PCR addresses the next instruction to be executed.

7.2. Machine Check Interrupt

Hardware errors are reported by a machine check interrupt.

All interrupt conditions due to machine checks result in the processor state switching to P4. These are machine check (IFR bit 3Ø; hardware error in central processor), I/O processor check (IFR bits 27, 61, 62, 63; hardware error in an I/O processor), and power failure (IFR bit 31; fault in the central power supply).

When a hardware error is detected, the central processor or the I/O system stores detailed information in main memory, which is then evaluated during the error analysis in P4.

7.3. Program-Related Interrupts

7.3.1. Program Interrupt Conditions

Program-related interrupt conditions include, besides program errors, test supports such as data address match checks or test mode, and interrupts caused by interval or program timers. All these conditions can be set in a Program Interrupt Flag Register (PIFR), and additional information can be specified in the Error Cause Register (ERCR).

The program-related interrupts result in the processor state switching to P3.

7.3.1.1. Test Mode

The test mode interrupt request is used as a programming aid in program testing. The test mode (PIFR bit Ø) can be set by the privileged Program Control (PC) instruction. This necessitates the execution of an instruction (in the new processor state) after the PC instruction, followed by a program interrupt if the test mode mask bit is set. If, after the execution of one instruction, other interrupt conditions besides test mode are pending, the condition with the highest priority is processed.

7.3.1.2. Fixed-Point Overflow

Fixed-point overflow is recognized when a carry occurs out of the high-order position in fixed-point arithmetic instructions or when significant bits are lost in arithmetic left-shift instructions.

The interrupt flag (PIFR bit 1) is not set if the corresponding program mask = Ø.

The instruction is completed by ignoring the lost data and setting condition code 3.

7.3.1.3. Decimal Overflow

Decimal overflow is recognized when one or more significant (high-order) digits are lost due to too small a result field in a decimal operation.

The interrupt flag (PIFR bit 2) is not set if the corresponding program mask = Ø. In this case, the instruction is completed by ignoring lost digits and setting condition code 3.

7.3.1.4. Exponent Underflow

Exponent underflow is recognized when the result characteristic of a floating-point addition, subtraction, multiplication, halving or division is less than Ø and the result mantissa is not Ø. The instruction is completed.

The interrupt flag (PIFR bit 3) is not set if the program mask for exponent underflow = Ø.

The program mask also determines the further handling of the result. If the mask bit is Ø (in which case the condition is not reported), the sign, characteristic and mantissa are set to Ø, making the result true zero. If the mask bit is 1, the mantissa remains correct and is normalized, the sign remains correct and the characteristic is 128 greater than the correct value.

7.3.1.5. Significance Error

A significance error is recognized when the mantissa of the intermediate result (i.e. the result before normalization) in normalized floating-point addition or subtraction instructions, or the mantissa of the result in non-normalized floating-point addition or subtraction instructions = Ø. The instruction is completed.

The interrupt flag (PIFR bit 4) is not set, if the corresponding program mask = Ø.

The program mask also determines the further handling of the result. If the mask bit is Ø, the result is made true zero, otherwise the characteristic and the sign of the result are not altered.

7.3.1.6. Divide Error

A divide error is recognized when one of the following conditions is fulfilled :

1. Decimal divide error.
   The quotient in decimal division exceeds the specified data field size.
2. Fixed-point divide error.
   The quotient in fixed-point division or the result of a Convert to Binary (CVB) instruction is outside the range of values -2,147,483,648 to +2,147,483,647.
3. Floating-point divide error.
   The mantissa of the divisor in floating-point division = Ø.

When a divide error occurs, the interrupt flag is set (PIFR bit 5) and instruction execution is suppressed.

7.3.1.7. Exponent Overflow

Recognition of exponent overflow depends upon whether the result exponent of floating-point addition, subtraction, multiplication or division is greater than 127 and whether the result mantissa is Ø.

The instruction is completed and the interrupt flag (PIFR bit 6) is set. The mantissa is normalized and correct, the sign remains correct and the characteristic is 128 less than the correct value.

7.3.1.8. Data Error

A data error is recognized when one of the following conditions is fulfilled :

1. Invalid digit or sign codes in an operand in decimal instructions or the Convert to Binary (CVB), Edit (ED) or Edit and Mark (EDMK) instructions.

   Note :

   If the decimal operands overlap incorrectly (i.e. they overlap each other but the two low-order bytes do not have the same address), a decimal digit error or a decimal sign error is automatically recognized.

2. The multiplicand in the Multiply Decimal (MP) instruction has insufficient leading zeros.

When a data error occurs, the interrupt flag (PIFR bit 7) is set. Instruction execution is suppressed in the case of an illegal sign or if an invalid digit occurs in the Compare Decimal (CP) or Convert to Binary (CVB) instructions. In all other cases, an invalid digit or multiplicand error results in termination of instruction execution.

7.3.1.9. Address Error

The following list contains all the address errors which are not due to address translation. The address translation errors (read access, write access) are described in 6. Dynamic Address Translation.

1. Physical access is recognized when, upon memory access, the real address lies outside available main memory, or if a protection error occurs.
2. An invalid I2 code is recognized when there is an incorrect value in the I2 field in the Store Clock (STCK) or Monitor Call (MC) instruction.
3. An execute error is recognized when an Execute instruction specifies that an Execute or Load/Store Word Indirect (LWI/STWI) instruction is to be performed.
4. Specifying an odd number for a doubleword register causes a doubleword register error.
5. An invalid floating-point register number occurs when a register number other than Ø, 2, 4 or 6 is specified in a floating-point instruction, or a register number other than Ø or 4 is specified in a floating-point instruction with extended operands.
6. An alignment error is recognized when an instruction is not on a halfword boundary, or an operand is not on the specified halfword, word or doubleword boundary.

7. An indirect nesting error is recognized when more than 5 levels of indirect addressing occur during an LWI or STWI instruction.

8. A multiplier/divisor size error is recognized when the multiplier or divisor in a decimal operation consists of more than 15 digits and a sign, or when the length of the first operand is not greater than that of the second in decimal multiplication or division.

9. An address error occurs in the Call by Number (CALN) instruction when the value of the branch index (CALNF) is greater than the vector table length (CALNVL).

10. A segment table address register error is recognized when the Segment Table Address Register is to be loaded with an address which is outside available main memory or not on a 256-word boundary.

11. Stack link errors occur when the extensible stack extents are not linked upon overflow/underflow during a Pop or Push instruction.

12. Audit wrap (when audit mode is on).
When the 64-word audit table is full, an address error is reported (audit table modulo 64 is addressed, therefore entry Ø would be overwritten the next time) if ISR bit 6 (permit audit wrap mask) = 1. If the permit audit wrap mask is not set, the overflow is not reported.

13. If the selected monitor mask bit = 1 in the MC instruction (Monitor Call), a special-class address error is reported in the Error Cause Register. This is not however a genuine program error.

After an address error has occurred, PIFR bit 8 is set and error class ØC and subclass ØØ are loaded into the Error Cause Register (ERCR). The following errors do not comply with this procedure :

- When the real address is outside main memory, i.e. constituting a physical access error, classes 1Ø, ØØ or ØC, Ø8 (T=Ø) are set depending on the T bit (ISR bit 13).
- When protection errors occur (physical access), classes ØC, ØC are set.
- When an audit wrap occurs classes 2Ø, ØØ are set.
- In the MC instruction classes 1C, ØØ are set.

In general, the instructions are suppressed when an address error occurs. Only in the case of physical access can the instruction, under certain circumstances, be terminated. With audit wrap and monitor call, the instruction is completed.

7.3.1.10. Operation Code Trap

Operation code trap is reported in response to one of the following events :

1. An operation code, which is not included in the instruction set of the central processor is to be decoded or the I2 field of the priviliged Function Call (FCAL), Control CPU (CCPU) or Control IOC (CIOC) instruction contains an incorrect value. The flags PIFR bit 9, is set and classes 28, ØØ are set in the ERCR.

2. Stack error.

   A control stack address (CSA) or a data stack address (DSA) in an Execute Stack (EXST) instruction lies outside the specified boundaries, or one of the addresses used (SAR, CSA, DSA, CALNVA) is not aligned as specified. The flag, PIFR bit 9, is set and classes 2C,ØØ are set in the ERCR.

When an operation code trap is recognized, instruction execution is suppressed.

7.3.1.11. Privileged Operation

This error occurs when a privileged instruction is to be executed and the N bit (ISR bit 15) is not set to zero. The flag, PIFR bit 1Ø, is set and instruction execution suppressed.

7.3.1.12. Supervisor Call

This interrupt request is caused by execution of the Supervisor Call (SVC) instruction. Bits 8-15 of the instruction are loaded into the low-order byte of the Interrupt Status Register (ISR) current during execution of the SVC, and, in this way, made available to the system program. The interrupt flag (bit 11) is set in the PIFR.

7.3.1.13. Paging Queue

A paging queue (PIFR bit 12) can only occur where a virtual address is to be converted to a real address. Instruction execution is suppressed when a paging queue is recognized.

7.3.1.14. Paging Error

Like the paging queue, a paging error may be recognized when a virtual address is to be converted to a real address. Instruction execution is suppressed.

7.3.1.15. Interval Timer

The program can load the interval timer with any value (from Ø.ØØØ1 to 6.5536 seconds). After the specified time has elapsed, the corresponding interrupt flag (PIFR bit 23) is set asynchronously for the current program. The interval timer runs in processor states P1 and P2.

7.3.1.16. Program Timer

Three program timers are provided, whose functions correspond to those of the interval timer except that the loadable value can be from Ø.ØØØ1 to 429,496.7296 seconds. After the specified time has elapsed, the flag PIFR bit 24 (program timer 1), PIFR bit 25 (program timer 2) or PIFR bit 26 (program timer 3) is set asynchronously for the current program. The program timers run in processor states P1 and P2.

7.3.1.17. Data Address Match Check

When the DAMC test support is on (ISR bit 16 = 1), bits 8-28 of the operand address are compared with bits 8-28 of the Data Address Match Register (DAMR) in every operand access. The DAMC interrupt flag (PIFR bit 29) is set when the 21 compared bits of the operand address match the comparison address. Since the comparison ignores the 3 low-order bits, testing accuracy is a doubleword.

The DAMR is loaded with the Load Status of Program (LSP) instruction. If the T bit in the ISR is set, the comparison if performed on the virtual address. DAMC cannot be used for instruction accesses or any of the exceptions listed in 6. Dynamic Address Translation.

The instruction is completed.

7.3.2. Data Access Error Recognition

Address translation errors and errors in physical access may still occur during instruction and data accesses even if only part of an instruction or operand connot be accessed.
Accessing of the full operand length is not necessary in the case of all instructions since some of them may already have been properly completed. If it is not possible to access the whole operand, the following points should be considered :

1. An error is not reported if part of an operand cannot be accessed in the Compare Logical (CLC) or Compare Logical Long (CLCL) instruction; the operation is completed properly with the operand parts available (due to unequal operands).

2. In the Translate (TR), Translate and Test (TRT), Edit (ED) and Edit and Mark (EDMK) instructions no access errors are reported for unused operand parts.

3. If, in a Move with Offset (MVO), Pack (PACK) or Unpack (UNPK) instruction, a part of the second operand cannot be accessed but the available operand part is nevertheless sufficient for proper completion of the instruction, no access errors are reported.

4. In the Move Long (MVCL) instruction no access errors are reported for the unused parts of the second operand.

7.3.3. Program Error Priority

The program-related interrupts, i.e. program errors, test mode, interval and program timers, byte boundary alignment and DAMC are reported in the Program Interrupt Flag Register (PIFR) of the current processor state (P1 or P2). Additional information for a more detailed analysis of the program errors is set in the current ERCR.

Table 7-2 summarizes the priority of program error recognition. The program errors are divided into four groups, group 1 having the highest priority and group 4 the lowest. The arrangement of the errors within the groups does not imply priority. When a program error from a particular group is reported, this means that errors from a group of lower priority will no longer be recognized and that no errors from higher-priority groups have occurred.

In Table 7-2, each program error is followed by details of the corresponding bit in the PIFR, the information stored in the ERCR and the type of instruction conclusion involved, where :

S = Suppression

T = Termination

C = Completion

Table 7-2 Priority of Program Error Recognition

Group	Instruction read and instruction execution program errors	PIFR bit	ERCR			Action
			Error class	Error subclass	Segment and page number	
1.	- Halfword alignment (instruction)	8	ØC	ØØ	zero	S
	- Read access (instruction)					
	User existence error	13	Ø8	Ø4,Ø8,ØC	valid	S
	Paging queue	12	3Ø	ØØ	valid	S
	System paging error	8	1Ø	ØØ	valid	S
	Protection error	13	Ø4	ØØ	valid	S
	- Physical access (instruction)					
2.	- Operation code trap	9	28	ØØ	zero	S
	- Privileged operation	1Ø	-	-	-	S
	- Operation code trap due to invalid I2 code in FCAL	9	28	ØØ	zero	S
	- Invalid I2 code	8	ØC	ØØ	zero	S
3.	- Execute error	8	ØC	ØØ	zero	S
	- Doubleword register error	8	ØC	ØØ	zero	S
	- Floating-point register error	8	ØC	ØØ	zero	S
	- Alignment error	8	ØC	ØØ	zero	S
	- Indirect nesting error	8	ØC	ØØ	zero	S
	- Multiplier/divisor size error	8	ØC	ØØ	zero	S
	- CALN address error	8	ØC	ØØ	zero	S
	- Read access					
	User existence error	13	Ø8	Ø4,Ø8,ØC	valid	S
	Paging queue	12	3Ø	ØØ	valid	S
	System paging error	8	1Ø	ØØ	valid	S
	Protection error	13	Ø4	ØØ	valid	S
	- Write access					
	User existence error	13	Ø8	Ø4,Ø8,ØC	valid	S
	Paging queue	12	3Ø	ØØ	valid	S
	System paging error	8	1Ø	ØØ	valid	S
	Protection error	13	Ø4	ØØ	valid	S
	- Physical access	8	ØC	Ø8,ØC	zero	S, T
	- Segment table address register error	8	1Ø	ØØ	valid [1]	S
	- Decimal format invalid sign	7	-	-	-	S
	- Decimal format invalid digit	7	-	-	-	T [2]
	- Stack link error	8	ØC	ØØ	zero	S
	- Stack error	9	2C	ØØ	zero	S

1) The segment and page numbers of B3/D3 are loaded.
2) S for the CP and CVB instructions.

Table 7-2 Priority of Program Error Recognition (contd)

Group	Instruction read and instruction execution program errors	PIFR bit	ERCR			Action
			Error class	Error subclass	Segment and page number	
4.	- Decimal divide error	5	-	-	-	S
	- Decimal multiplicand error	7	-	-	-	T
	- Fixed-point divide error	5	-	-	-	S
	- Floating-point divide error	5	-	-	-	S
	- Decimal overflow	2	-	-	-	C
	- Fixed-point overflow	1	-	-	-	C
	- Exponent overflow	6	-	-	-	C
	- Exponent underflow	3	-	-	-	C
	- Significance error	4	-	-	-	C
	- Audit wrap	8	2Ø	ØØ	zero	C
	- Monitor call	8	1C	ØØ	zero	C

7.4. Asynchronous Interrupts

Asynchronous interrupt conditions are flagged, like machine checks, in the Interrupt Flag Register (IFR) and result in a switch to P3.

The occurrence of an asynchronous interrupt condition does not influence the execution of the current instruction. The program interrupt is not initiated until instruction execution has been concluded.

7.4.1. Console Interrupt

This interrupt request (IFR bit 14) is set when the COIN button is pressed on the console.

7.4.2. Elapsed Time Clock

The ETC interrupt condition (IFR bit 15) occurs when the ETC issues an interrupt request. It does this at fixed intervals of 1 second.

7.4.3. Channel Requests

Each I/O channel is assigned a bit in the IFR, which is set when the channel in question issues an interrupt request. Bits 16-22 and 32-56 of the IFR are reserved for channel requests.

The channel issues an interrupt request to the central processor when one of the following events is detected: termination interrupt, program controlled interrupt, attention interrupt or channel free interrupt.

When a program interrupt is performed in response to a channel request, status information is made available to the system in the I/O channel registers. The format of these registers together with definitions of the channel interrupt events are to be found in 12. Input/Output Operations.

7.4.4. Alert CPU

This interrupt (IFR bit 59) is set upon execution of the privileged Alert CPU (ACPU) instruction, a special function of FCAL.
The instruction is designed for use in multiprocessor systems so that one central processor can call another or itself.

7.5. Program Interrupt Priority

Several interrupt conditions can occur during the execution of one instruction. For example, a program error may have been detected, a timer may have expired, a channel request may be pending, a machine check may have occurred or an old interrupt request may still be stored.

If the Interrupt Mask Register allows several of these interrupts, the one with the highest priority is taken first. The other interrupt conditions remain set in the Interrupt Flag Register or in the Program Interrupt Flag Register to be handled later in order of priority.

The priorities of the program interrupts are given in Table 7-1.

8. Multiprocessor Operation

In order to still further enhance system availability and processing power while at the same time improving resource utilization, the structure of the Siemens System 7.000 can be used in multiprocessor systems. In this type of environment the central processors must operate in the extended I/O mode.

8.1. Configuration

A multiprocessor system consists of two central processors and up to four I/O systems (I/O processors), with a common main memory. The two central processors are distinguishable by a processor number, Ø or 1, which is established when the system is installed. This number can be requested in the system status using the special function Store CPU Number (STNU). Each of the four possible I/O systems also has a number, from Ø to 3, set at installation time.

In a multiprocessor system, either central processor can initiate I/O operations in any of the I/O systems. I/O system interrupt requests are issued to either both central processors or only one, depending on the model.

Model-dependent factors determine whether both central processors in a system have their own time-of-day-clock or whether only one is implemented. Even when both central processors are equipped with a time-of-day clock, only one is used in multiprocessor operation. As a result, programs always refer to the same time-of-day clock, regardless of which central processor they are running on.

Unlike the time-of-day clock, the elapsed time clock must not be used for timing purposes in a multiprocessor system by programs which are not always run on the same central processor since, depending on the model, the ETC may be installed in any central processor, where it can generate interrupt requests independently of the ETC in the second central processor.

8.2. Memory Addressing

Access to the common main memory is identical for all processors, i.e. a specific memory location has the same real address for both central processors and the four possible I/O systems, and each processor can access any address in the memory available. By the same token, all the memory areas for the processors are protected with the same storage key.

8.3. Processor Communication

There are two different possible types of processor communication in a multiprocessor environment, depending on the types of processor involved. Communication between the two central processors is implemented by means of the special function Alert CPU (ACPU), with which one central processor can transfer one byte of information to the other and generate an interrupt request. Prior to any communication it must be checked whether all call processor instructions have been executed. This serialization may be obtained by using branch instruction BCR 15,0.

Communication between a central processor and an I/O system concerns solely the execution of I/O instructions and I/O interrupt handling.

Contention problems arising during actual communication between two processors due to simultaneous requests are resolved automatically by the processors concerned and are not apparent to the user.

9. Privileged Instructions

9.1. Privileged Protection

A privileged instruction can only be executed if the privileged mode is set in the current processor state. The privileged operation interrupt condition occurs if execution of a privileged instruction is attempted in the non-privileged mode. The operation is suppressed.

9.2. Interrupt Analysis

Interrupts may be generated due to a large variety of circumstances encountered during instruction execution.

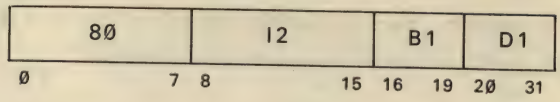
Interrupts which are independent of privileged operation, such as operand misalignment and paging, are enumerated in the individual privileged instruction descriptions. (For a complete description of these interrupts, see 7. Program Interrupts).

9.3. Description of Privileged Instructions

9.3.1. Idle (IDL)

Type: SI

Format:



Description:

Under control of the I2 field, this operation generates an idle mode within the central processor or activates a signal which, for example, will ring an audible alarm.

This idle mode can only be terminated by an interrupt condition. During the idle mode, the IDLE indicators of the console and maintenance panel are on.

Condition Code:

Unchanged

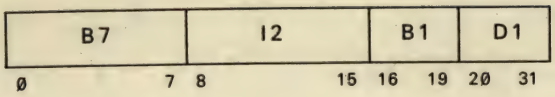
Interrupt Priorities:

Error	Action
Privileged operation	S

9.3.2. Function Call (FCAL)

Type: SI

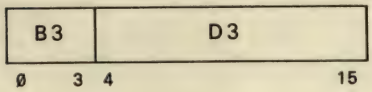
Format:



Description:

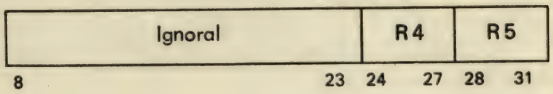
This instruction provides for several special functions, which are specified individually in the I2 field. There are three address formats:

Indirect, where B1/D1 addresses a halfword in main memory with the following contents:



Direct, where B1/D1 is the address of the operand.

Pseudo RR, where bits 24-31 of B1/D1 contain the R4 and R5 fields which specify two general registers.



9

9.3.2.1. Load Segment Table Address and Length

Function Call (I2 Code) : 2Ø

Format: Indirect

Description:

This special function loads the Segment Table Address Register (STAR) and the Segment Table Length Register (STLR) with the contents of the main memory word which is indirectly specified via the address field and which contains the segment table address and segment table length.

In addition, this special function clears in the central processor any copies of table entries and other prefetched information which are no longer valid.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Operation code trap (when address translation hardware is not installed, or invalid I2 code)	S
Alignment error	S
Read access	S
Physical access	S
Segment table address register error	S

9.3.2.2. Store Segment Table Address and Length (SSAL)

Function Call 12 Code : 21

Format : Indirect

Description :

This special function stores the contents of the Segment Table Address Register (STAR) and the Segment Table Length Register (STLR) into the main memory word specified indirectly via the address field.

Condition Code :

Unchanged

Interrupt Priorities :

Error	Action
Operation code trap (when address translation hardware is not installed, or invalid 12 code)	S
Alignment error	S
Write access	S
Physical access	S

9.3.2.3. Store Interrupt Flag Register (STIF)

Function Call 12 Code : 62

Format : Direct

Description:

This special function stores the contents of the Interrupt Flag Register (IFR) into a main memory double-word which is directly addressed by B1/D1. The contents of the IFR remain unchanged.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Alignment error	S
Write access	S
Physical access	S

9.3.2.4. Store I/O Status (STIO)

Function Call 12 Code : 63

Format: Direct

Description:

This special function stores the I/O status information into a 4-word main memory area addressed directly by B1/D1.

The I/O status information comprises the contents of the channel registers CAR, CCR1, CCR2 and DSR of the channel from which the last I/O interrupt was taken.

The contents of the four registers remain unchanged.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Alignment error	S
Write access	S
Physical access	T

9.3.2.5. Test and Set Real (TSR)

Function Call 12 Code : 66

Format : Direct

Description:

This special function tests and sets a byte in main memory, which is addressed directly by B1/D1.

The byte addressed by B1/D1 is read, used to set the condition code, and then set to all ones.

During execution of this special function, no other main memory access is possible.

Condition Code:

Ø Specific bit of (B1/D1) was zero
1 Specific bit of (B1/D1) was one
2 -
3 -

Interrupt Priorities:

Error	Action
Read and write access	S
Physical access	S

9.3.2.6. Set Clock (SCK)

Function Call 12 Code : 67

Format : Direct

Description:

This special function replaces the current value of the time-of-day clock with the contents of the main memory doubleword addressed by B1/D1.

Condition Code:

Ø Clock value set
1 Clock value secure
2 -
3 Clock not operational

Interrupt Priorities:

Error	Action
Alignment error	S
Read access	S
Physical access	S

9.3.2.7. Store CPU Identification (STID)

Function Call 12 Code : 68

Format : Direct

Description:

This special function stores the central processor identification into a main memory doubleword addressed by B1/D1.

Central processor identification is a doubleword hardware facility, in which are hardwired the central processor serial number and model number, and also the length of the logout area.

This identification is set at installation and cannot be altered.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Alignment error	S
Write access	S
Physical access	S

9.3.2.8. Store CPU Number (STNU)

Function Call 12 Code : 69

Format : Direct

Description:

This special function stores the contents of the Central Processor Number Register in a main memory byte addressed by B1/D1.

The central processor number is 1-byte binary encoded number which is set at installation on a multi-processor configuration to uniquely identify each central processor in the configuration.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Operation code trap (if feature is not installed)	S
Write access	S
Physical access	S

9.3.2.9. Load Word Real (LDWR)

Function Call 12 Code : 8Ø

Format : Pseudo RR

Description:

This special function loads a general register with the contents of a main memory word which is designated by the real address contained in the other general register.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Alignment error	S
Physical access	S

9.3.2.10. Load Halfword Real (LDHR)

Function Call 12 Code : 81

Format : Pseudo RR

Description:

This special function loads a halfword from main memory into a general register. The other halfword of the general register is set to zero.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Alignment error	S
Physical access	S

9.3.2.11. Store Word Real (STWR)

Function Call 12 Code : 82

Format : Pseudo RR

Description:

This special function stores a word from a general register into a word in main memory designated by the real address contained in the other general register.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Alignment error	S
Physical access	S

9.3.2.12. Store Halfword Real (STHR)

Function Call 12 Code : 83

Format : Pseudo RR

Description:

This special function stores a halfword from a general register into a halfword in main memory designated by the real address contained in the other general register.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Alignment error	S
Physical access	S

9.3.2.13. Alert CPU (ACPU)

Function Call 12 Code : 84

Format : Pseudo RR

Description:

This special function provides the primary means of communication between the central processors of a multiprocessor configuration. It causes the CPU alert interrupt flag to be set in the Interrupt Flag Register of the addressed central processors. In addition, a one-byte message from main memory can be sent to the addressed central processors.

Execution of this special function is terminated when all the addressed and operable central processors acknowledge receipt of ACPU, or when the maximum response time has elapsed.

Condition Code:

Ø ACPU acknowledged by all addressed central processors.

1 Previous ACPU from same central processor, where a message was sent, has not yet been processed.

2 No central processor is addressed.

3 ACPU not acknowledged by all addressed central processors.

Interrupt Priorities:

Error	Action
Read access	S
Physical access	S

Notes:

Before addressing a special function it must be checked whether all previous instructions have been executed. Branch instruction BCR 15,0 may be used for this purpose.

9.3.2.14. Coordinate CPU (COOP)

Function Call 12 Code : 86

Format : Pseudo RR

Description:

This special function serves to store a main memory byte addressed by one general register into another general register and subsequently modify the byte in main memory. During execution of this special function, no other central processor of a multiprocessor configuration can access main memory. Additionally, the address translation table entries in all central processors are cleared.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	S

Notes:

Before addressing a special function it must be checked whether all previous instructions have been executed. Branch instruction BCR 15,0 may be used for this purpose.

9.3.2.15. Trace Virtual Address (TVA)

Function Call 12 Code : 85

Format : Pseudo RR

Description:

This special function permits access to the real address of the segment table entry and of the control bits from the segment table entry, and also to the translated real address associated with a given virtual address.

The virtual address is contained in the general register specified by R4. It is replaced by the translated real address. The real address and the control bits of the associated segment table entry are placed in the general register specified by R5.

Condition Code:

Ø Addressed page is defined and is present in main memory. Translation table entry is loaded.

1 Addressed page not defined or not in main memory.

2 Page table not in main memory, segment not defined or real address of page table entry lies outside available memory. Contents of general registers specified by R4 and R5+1 are unaltered.

3 Contents of bits 8-11 of virtual address are greater than segment table length. Contents of general registers specified by R4, R5 and R5+1 are unaltered.

Interrupt Priorities:

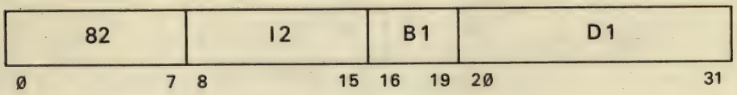
Error	Action
Operation code trap (when address translation hardware is not installed)	S
Doubleword register error	S

/

9.3.3. Program Control (PC)

Type : SI

Format:



Description:

This instruction specifies the termination of program execution in the current processor state, and the initiation of another state under control of the immediate byte in the I2 field.

The current processor state is deactivated and B1/D1 is stored in the PCR of the state being terminated.

Initiation of a processor state means its control registers are activated.

Condition Code:

The condition code indicators of the state being terminated are preserved in the state's PCR. The condition code in the PCR of the initiated state is then used to set the condition code indicators.

Interrupt Priorities:

Error	Action
Privileged operation	S

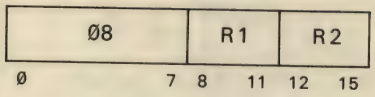
If the program test bit is set in the I2 field, no program interrupts are allowed between Program Control (PC) and the next instruction.

If processor state P3 or P4 is initiated, the interval timer and the three program timers are stopped; they are started or restarted when processor state P1 or P2 is initiated.

9.3.4. Set Storage Key (SSK)

Type : RR

Format :



Description:

The storage key addressed by the contents of the general register specified by R2 is set according to the value contained in the register specified by R1.

Condtion Code:

Unchanged

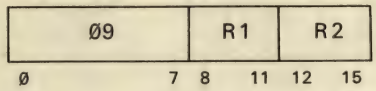
Interrupt Priorities:

Error	Action
Privileged operation	S
Address error	S

9.3.5. Insert Storage Key (ISK)

Type : RR

Format :



Description :

The storage key of the real memory page addressed by the contents of the general register specified by R2 is inserted in the general register specified by R1.

Condition Code :

Unchanged

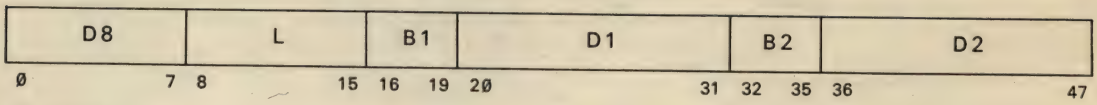
Interrupt Priorities :

Error	Action
Privileged operation	S
Address error	S

9.3.6. Load Status of Program (LSP)

Type : SS

Format:



Description:

This instruction loads operands from main memory starting at the location specified by the second address B2/D2 into central processor registers, according to the subfunction specified in the first address B1/D1.

Subfunctions:

1. Load Scratch Pad

   This subfunction loads operands from main memory starting at the location specified by the second address B2/D2 into the scratch-pad registers starting at the location specified by the first address B1/D1. The length field L contains an 8-bit count which specifies the number of scratch-pad words to be loaded. A count of zero specifies one word to be loaded.

2. Load Program Context

   This subfunction loads the program context, i.e. the complete set of registers for a particular processor state, including those registers common to all states. The subfunction and the processor state are specified in the first address B1/D1. The second address B2/D2 points to the first word to be loaded.

   The length field is not used.

3. Load Program Status

   This subfunction loads the program status, i.e. the complete set of registers for a particular processor state, but not including those registers common to all states. The subfunction and the processor state are specified in the first address B1/D1. The second address B2/D2 points to the first word to be loaded.

   The length field is not used.

4. Load Reduced Program Status

   This subfunction loads a register set in the same way as the Load Program Status subfunction, but the set does not include the general registers.

   The length field is not used.

5. Load Register Set

This subfunction serves to load single registers or a register set of the program context. The subfunction, the register address and the processor state are specified in the first address B1/D1. The second address B2/D2 points to the first word to be loaded.

The length field is not used.

Notes:

1. The second address B2/D2 must be word-oriented.
2. The central processor uses scratch-pad utility registers for these operations. If these registers are addressed by the Load Scratch Pad subfunction, results are unpredictable.

Condition Code:

Unchanged

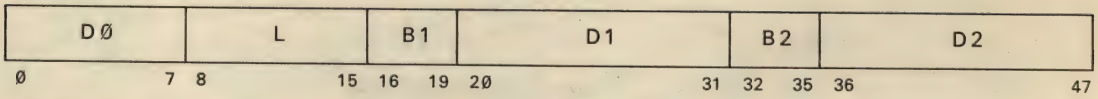
Interrupt Priorities:

Error	Action
Privileged operation	S
Address Error	S
Alignment error	S
Physical access	T
Read access	S

9.3.7. Store Status of Program (SSP)

Type: SS

Format :



Description:

This instruction stores the contents of central processor registers specified by the first address B1/D1 into main memory starting at the location specified by the second address B2/D2, according to the subfunction specified in B1/D1.

Subfunctions :

1. Store Scratch Pad

   This subfunction stores operands from the scratch-pad registers starting at the location specified by the first address B1/D1 into main memory starting at the location specified by the second address B2/D2. The length field L contains an 8-bit count which specifies the number of scratch-pad words to be stored.
   A count of zero specifies one word to be stored.

2. Store Program Context

   This subfunction stores the program context, i.e. the complete set of registers for a particular processor state, including those registers common to all states. The subfunction and the processor state are specified in the first address B1/D1. The second address B2/D2 specifies the memory location at which the first scratch-pad word is to be stored.

   The length field is not used.

3. Store Program Status

   This subfunction stores the program status, i.e. the complete set of registers for a particular processor state, but not including those registers common to all states. The subfunction and the processor state are specified in the first address B1/D1. The second address B2/D2 specifies the memory location at which the first scratch-pad word is to be stored.

   The length field is not used.

4. Store Reduced Program Status

   This subfunction stores a register set in the same way as the Store Program Status subfunction, but the set does not include the general registers.

   The length field is not used.

5. Store Register Set

This subfunction serves to store single registers or a register set of the program context. The subfunction, the register address and the processor state are specified in the first address B1/D1. The second address B2/D2 specifies the memory location at which the first scratch-pad word is to be stored.

The length field is not used.

Notes:

1. The second address B2/D2 must be word-oriented.
2. The central processor uses scratch-pad utility registers for these operations. If these registers are addressed by the Store Scratch Pad subfunction, results are unpredictable.

Condition Code:

Unchanged

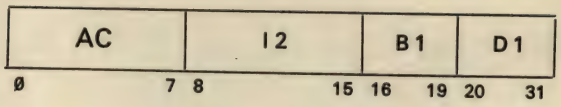
Interrupt Priorities:

Error	Action
Privileged operation	S
Address error	S
Alignment error	S
Physical access	T
Write access	S

9.3.8. Control CPU (CCPU)

Type: SI

Format:



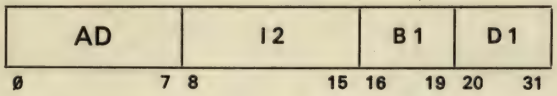
Description:

This instruction provides for model-dependent control and error handling functions in Siemens System 7.000 central processors.

9.3.9. Control I/O Control (CIOC)

Type: SI

Format:



Description:

This instruction provides for model-dependent control and error handling functions in Siemens System 7.000 I/O systems.

10. Non-Privileged Instructions

10.1. General

The non-privileged instructions are available to all users.
Execution of each instruction, both under normal conditions and in the presence of program errors, is described below.

10.2. Instruction Types

There are five different types of instructions varying in the type of operand addressing and in the instruction length:

RR The operands are in registers.

RX The first operand is in a register, the second operand at an indexed main memory location.

RS The first operand is in a register, the second operand is at a main memory location, the third operand is in a register.

SI The first operand is at a main memory location, the second operand is contained immediately in the instruction.

SS The operands are at main memory locations.

The five types of instructions are shown in Fig. 10-1.

RR Type

The general register specified by R1 contains the first operand. The general register specified by R2 contains the second operand. In floating-point operations, R1 and R2 designate the two floating-point registers containing the operands. The two operands can be identical and are then designated by the same addresses (R1 = R2).

RX Type

The general register specified by R1 contains the first operand. To obtain the address of the second operand (X2/B2/D2), the contents of the general registers specified by X2 and B2 are added to the displacement contained in the D2 field. In floating-point operations, R1 designates the floating-point register containing the first operand.

RS Type

This type is used for shift instructions, branch instructions and the Load Multiple and Store Multiple instructions.

Shift Instructions:

The general register specified by R1 contains the first operand. The contents of the general register specified by B2 are added to the contents of the D2 field. The sum (B2/D2) specifies the number of bits by which the operand is shifted. The R3 field is ignored.

Branch Instructions:

The general register specified by R1 contains the first operand. To obtain the branch adress (B2/D2), the contents of the general register specified by B2 are added to the displacement contained in the D2 field. The general register specified by R3 contains the third operand.

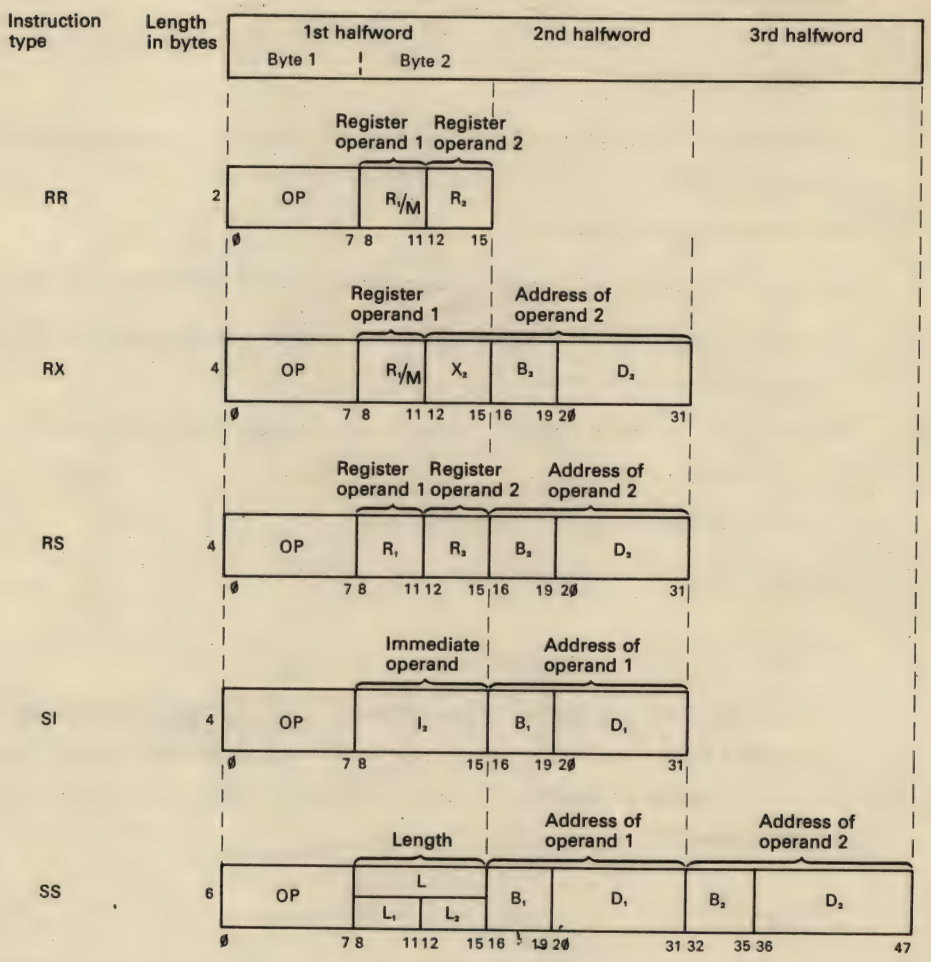


Fig. 10-1 Instruction Formats

Field definitions:

Op	Operation code
R1, R2	4-bit designators of general registers (floating-point registers in the case of floating-point instructions) used for operands
R3	4-bit designator of a general register used for operands
X2	4-bit designator of a general register used for indexing
B1, B2	4-bit designators of general registers used for base addressing
D1, D2	12-bit displacements
I2	8-bit immediate operand
L1, L2	4-bit operand length designators
L	8-bit operand length designator
M	4-bit mask

Load/Store Multiple Instructions:

The R1 and R3 fields specify the first and the last general register to be loaded or stored. To obtain the start address of the second operand (B2/D2), the contents of the general register specified by B2 are added to the displacement contained in the D2 field.

SI Type

To obtain the start address of the first operand (B1/D1), the contents of the general register specified by B1 are added to the displacement contained in the D1 field. The second operand is the I2 field of the instruction.

SS Type

To obtain the address of the leftmost byte of the first operand (B1/D1), the contents of the general register specified by B1 are added to the displacement contained in the D1 field. The L1 field specifies the number of bytes less 1 in the first operand. To obtain the address of the second operand (B2/D2), the contents of the general register specified by B2 are added to the displacement contained in the D2 field. The L2 field specifies the number of bytes less 1 in the second operand. If the operands have the same length, the L field specifies the number of bytes less 1 in both operands.

Notes:

1. If a zero appears in the X2, B1 or B2 field, the corresponding address or shift-amount component is ignored. An instruction can specify the same general register both for address modification and for operand location.

2. The operand adresses are formed prior to instruction execution. These are referred to as effective addresses, i.e. real addresses of operands in real main memory for the real mode, and virtual addresses in virtual memory for the virtual mode.

3. The result of the instruction execution replaces the first operand except for the Store Character and the Convert to Binary instructions where the result replaces the second operand.

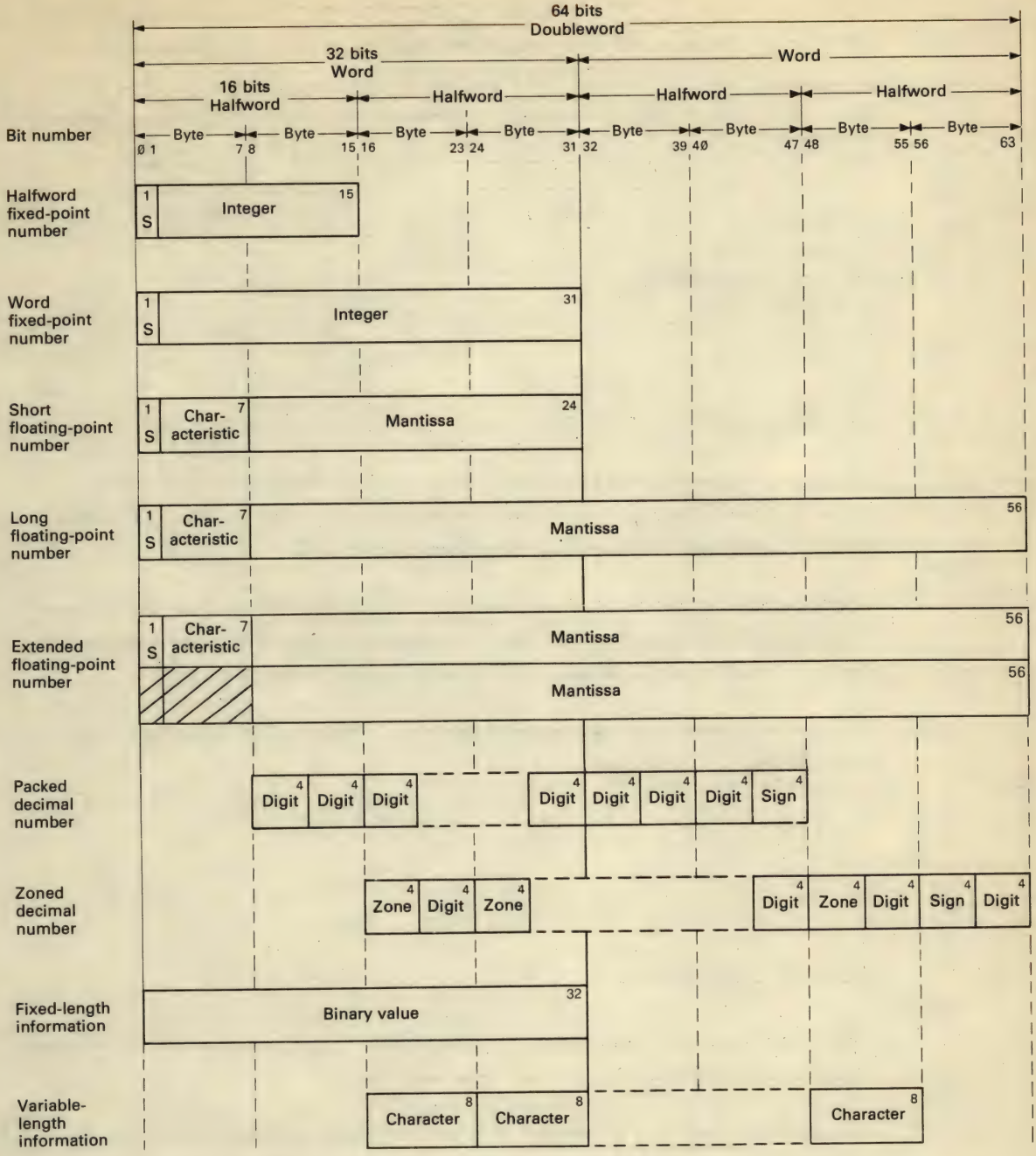
4. A variable-length result is never stored outside the field specified by the address and length.

5. The contents of all registers and memory locations not specified by an instruction remain unchanged except for the Edit and Mark and the Translate and Test instructions.

10.3. Data Formats

The data formats used by the Siemens System 7.000 are shown in Fig. 10-2.

Bit numbering and bit significance in a byte is illustrated in Fig. 10-3.



NOTE: Numbers in upper right corners of blocks indicate number of bits used
S = Sign

Fig. 10-2 Data Formats

	Left hexadecimal digit				Right hexadecimal digit				Left hexadecimal digit				Right hexadecimal digit			
Bit signifiance	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit position	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
	Byte 0								Byte 1							

Fig. 10-3 Bit Numbering

10.4 Byte Boundary Alignment

Byte boundary alignment is not suported in the Siemens System 7.000. It would permit memory accesses by almost all non - privileged instructions at any byte boundary, i.e. fixed-point, floating - point, and logical operands might appear at any byte boundary.

As byte boundary alignment was originally scheduled to be a standard feature of the Siemens System 7.000, the respective instruction executions are completed (C) in some CPU's despite byte boundary alignment, instead of being suppressed (S) as indicated in tables 10-1 to 10-10.

C/S stands for the completion or suppression of instructions. Addressing error is specified for the cause of the interrupt in each case.

10.5. Synchronous Program Interrupts

Program errors may occur during the execution of instructions. In most cases these errors do not allow orderly execution of the instruction and must therefore be reported.

Each program error may cause an interrupt. Since such interrupts occur during the execution of an instruction and can therefore be directly related to the instruction causing the interrupt, they are called "synchronous interrupts".

Program errors are recorded in the PIFR of the current processor state. In addition, for many program errors supplementary information is recorded in the ERCR of the current processor state.

All program errors and the associated synchronous interrupts result in one of the following conclusions of the instruction:
completion, termination, or suppression.

- Completion (C) implies that a generally correct result was generated, but some event has occurred which requires careful result interpretation.
- Termination (T) implies that instruction execution was left off at some undefined point, generally with unusable results.
- Suppression (S) implies that the instruction may be executed again with identical operands by decrementing the next instruction address (NIA) in the PCR by twice the value indicated by the instruction length code (ILC) and then reinitiating execution. The condition code is unpredictable for those instructions which modify the condition code.

  At instruction conclusion, whether by completion, termination, or suppression, NIA is set to NIA + 2 x ILC (NIA = next instruction address at the beginning of instruction execution).

A summary of synchronous interrupts is provided in Section 7.

Tables 10-1 to 10-10 list the possible program errors for each instruction and also indicate in each case whether the instruction goes to completion (C), is terminated (T), or suppressed (S).

The following program errors which may occur during the execution of any instruction are not listed in the tables:

Halfword alignment
Read access
Physical access } relating to the instruction address
Operation code trap
Test mode

Table 10-1 Data Transfer Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction Type	Read access	Write access	Physical access	Doubleword register error	Indirect nesting error	Byte boundary alignment *)	Data address match check	Word alignment
IC	Insert Character	RX	S		S				C	
ICM	Insert Characters under Mask	RS	S		T				C	
L	Load Word	RX	S		S			S	C	
LBF	Load Bit Field	SI	S		S				C	
LH	Load Halfword	RX	S		S			S	C	
LM	Load Multiple	RS	S		T			S	C	
LR	Load Word	RR								
LWI	Load Word Indirect	RX	S		S		S		C	S
MVC	Move	SS	S	S	T				C	
MVCL	Move Long	RR	S	S	T	S			C	
MVI	Move	SI		S	S				C	
MVN	Move Numerics	SS	S	S	T				C	
MVO	Move with Offset	SS	S	S	T				C	
MVZ	Move Zones	SS	S	S	T				C	
ST	Store Word	RX		S	S			S	C	
STBF	Store Bit Field	SI		S	S				C	
STC	Store Character	RX		S	S				C	
STCM	Store Characters under Mask	RS		S	T			S	C	
STH	Store Halfword	RX		S	S				C	
STM	Store Multiple	RS		S	T			S	C	
STWI	Store Word Indirect	RX		S	S		S		C	S

*) Leads to addressing error.

Table 10-2 Branch Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction Type	Physical access	Audit wrap		
BAL	Branch and Link	RX	C	C		
BALR	Branch and Link	RR	C	C		
BC	Branch on Condition	RX	C	C		
BCR	Branch on Condition	RR	C	C		
BCT	Branch on Count	RX	C	C		
BCTR	Branch on Count	RR	C	C		
BXH	Branch on Index High	RS	C	C		
BXLE	Branch on Index Low or Equal	RS	C	C		

Table 10-3 Logicql Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction Type	Read access	Write access	Physical access	Byte boundary alignment *)	Data address match check
N	AND	RX	S		S	S	C
NC	AND	SS	S	S	T		C
NI	AND	SI		S	S		C
NR	AND	RR					
O	OR	RX	S		S	S	C
OC	OR	SS	S	S	T		C
OI	OR	SI		S	S		C
OR	OR	RR					
TM	Test under Mask	SI	S		S		C
X	Exclusive OR	RX	S		S		C
XC	Exclusive OR	SS	S	S	T		C
XI	Exclusive OR	SI		S	S		C
XR	Exclusive OR	RR					

*) Leads to addressing error.

Table 10-4 Binary Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Read access	Physical access	Doubleword register error	Byte boundary alignment *)	Data address match check
AL	Add Logical	RX	S	S		S	C
ALR	Add Logical	RR					
CL	Compare	RX	S	S		S	C
CLC	Compare Logical	SS	S	S			C
CLCL	Compare Logical Long	RR	S	S	S		C
CLI	Compare Logical	SI	S	S			C
CLM	Compare Logical Characters under Mask	RS	S	S			C
CLR	Compare Logical	RR					
SL	Subtract Logical	RX	S	S		S	C
SLDL	Shift Left Double Logical	RS			S		
SLL	Shift Left Single Logical	RS					
SLR	Subtract Logical	RR					
SRDL	Shift Right Double Logical	RS			S		
SRL	Shift Right Single Logical	RS					

*) Leads to addressing error.

Table 10-5 Fixed-Point Instructions for Signed Binary Numbers - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Read access	Physical access	Doubleword register error	Fixed-point divide error	Fixed-point overflow	Byte boundary alignment *)	Data address match check
A	Add Word	RX	S	S			C	S	C
AH	Add Halfword	RX	S	S			C	S	C
AR	Add Word	RR					C		
C	Compare Word	RX	S	S				S	C
CH	Compare Halfword	RX	S	S				S	C
CR	Compare Word	RR							
D	Divide	RX	S	S	S	S		S	C
DR	Divide	RR			S	S			
LCR	Load Complement	RR					C		
LNR	Load Negative	RR							
LPR	Loand Positive	RR					C		
LTR	Load and Test	RR							
M	Multiply Word	RX	S	S	S			S	C
MH	Multiply Halfword	RX	S	S				S	C
MR	Multiply Word	RR			S				
S	Subtract Word	RX	S	S			C	S	C
SH	Subtract Halfword	RX	S	S			C	S	C
SR	Subtract Word	RR					C		
SLA	Shift Left Single	RS					C		
SLDA	Shift Left Double	RS			S		C		
SRA	Shift Right Single	RS							
SRDA	Shift Right Double	RS			S				

*) Leads to addressing error.

Table 10-6 Decimal Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Multiplier/divisor/size error	Read access	Decimal format invalid sign	Decimal divide error	Fixed-point divide error	Write access	Decimal format invalid digit	Physical access	Decimal multiplicand error	Decimal overflow	Data address match check	Byte boundary alignment
AP	Add Decimal	SS		S	S			S	T	T		C	C	
CP	Compare Decimal	SS		S	S				S	S			C	
CVB	Convert to Binary	RX		S	S		C		S	S			C	C
CVD	Convert to Decimal	RX						S		T			C	C
DP	Divide Decimal	SS	S	S	S	S		S	T	T			C	
MP	Multiply Decimal	SS	S	S	S			S	T	T	T		C	
PACK	Pack	SS		S				S		T			C	
SP	Subtract Decimal	SS		S	S			S	T	T		C	C	
SRP	Shift and Round Decimal	SS		S	S			S	T	T		C	C	
UNPK	Unpack	SS		S				S		T			C	
ZAP	Zero and Add	SS		S	S			S	T	T		C	C	

Table 10-7 Floating-Point Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Floating-point register error	Register error extended	Read access	Write access	Physical access	Floating-point divide error	Significance error	Exponent overflow	Exponent underflow	Byte boundary alignment *)	Data address match check
AD	Add Normalized (long)	RX	S		S		S		C	C	C	S	C
ADR	Add Normalized (long)	RR	S						C	C	C		
AE	Add Normalized (short)	RX	S		S		S		C	C	C	S	C
AER	Add Normalized (short)	RR	S						C	C	C		
AU	Add Unnormalized (short)	RX	S		S		S		C	C		S	C
AUR	Add Unnormalized (short)	RR	S						C	C			
AW	Add Unnormalized (long)	RX	S		S		S		C	C		S	C
AWR	Add Unnormalized (long)	RR	S						C	C			
AXR	Add Normalized (extended)	RR	S	S					C	C	C		
CD	Compare (long)	RX	S		S		S					S	C
CDR	Compare (long)	RR	S										
CE	Compare (short)	RX	S		S		S					S	C
CER	Compare (short)	RR	S										
DD	Divide (long)	RX	S		S		S	S		C	C	S	C
DDR	Divide (long)	RR	S					S		C	C		
DE	Divide (short)	RX	S		S		S	S		C	C	S	C
DER	Divide (short)	RR	S					S		C	C		
HDR	Halve (long)	RR	S								C		
HER	Halve (short)	RR	S								C		
LCDR	Load Complement (long)	RR	S										
LCER	Load Complement (short)	RR	S										
LD	Load (long)	RX	S		S		S					S	C
LDR	Load (long)	RR	S										
LE	Load (short)	RX	S		S		S					S	C
LER	Load (short)	RR	S										

*) Leads to addressing error.

Table 10-7 (contd.) Floating-Point Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Floating-point register error	Register error extended	Read access	Write access	Physical access	Floating-point divide error	Significance error	Exponent overflow	Exponent underflow	Byte boundary alignment *)	Data address match check
LNDR	Load Negative (long)	RR	S										
LNER	Load Negative (short)	RR	S										
LPDR	Load Positive (short)	RR	S										
LPER	Load Positive (short)	RR	S										
LRDR	Load Rounded (extended to long)	RR	S	S						C			
LRER	Load Rounded (long to short)	RR	S							C			
LTDR	Load and Test (long)	RR	S										
LTER	Load and Test (short)	RR	S										
MD	Multiply (long)	RX	S		S		S			C	C	S	C
MDR	Multiply (long)	RR	S							C	C		
ME	Multiply (short to long)	RX	S		S		S			C	C	S	C
MER	Multiply (short to long)	RR	S							C	C		
MXD	Multiply (long to extended)	RX	S	S	S		S			C	C	S	C
MXDR	Multiply (long to extended)	RR	S	S						C	C		
MXR	Multiply (extended)	RR	S	S						C	C		
SD	Subtract Normalized (long)	RX	S		S		S		C	C	C	S	C
SDR	Subtract Normalized (long)	RR	S						C	C	C		
SE	Subtract Normalized (short)	RX	S		S		S		C	C	C	S	C
SER	Subtract Normalized (short)	RR	S						C	C	C		
STD	Store (long)	RX	S			S	S					S	C
STE	Store (short)	RX	S			S	S					S	C

*) Leads to addressing error.

Table 10-7 (contd.) Floating-Point Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Floating-point register error	Register error extended	Read access	Write access	Physical access	Floating-point divide error	Significance error	Exponent overflow	Exponent underflow	Byte boundary alignment *)	Data address match check
SU	Subtract Un-normalized (short)	RX	S		S		S		C	C		S	C
SUR	Subtract Un-normalized (short)	RR	S						C	C			
SW	Subtract Un-normalized (long)	RX	S		S		S		C	C		S	C
SWR	Subtract Un-normalized (long)	RR	S						C	C			
SXR	Subtract Normalized (extended)	RR	S	S					C	C	C		

*) Leads to addressing error.

Table 10-8 Stack Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Stack error	Read access	Stack link error	Write access	Physical access	Word alignment	Doubleword alignment	Data address match check	CALN address error	Audit wrap
EXST	Execute Stack	RS										
CALC	Call by Location	RS	S	S		S	S			C		C
CALN	Call by Number	RS	S	S		S	S			C	S	C
MSAR	Move Stack Address	RS	S	S		S	S					
RET	Return	RS	S	S		S	S			C		C
STMS	Store Multiple in Stack	RS	S	S		S	T			C		
POP	Pop	RS		S	S	S	S	C	C	C		
PUSH	Push	RS		S	S	S	S	C	C	C		

Table 10-9 Edit Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Read access	Write access	Physical access	Decimal format invalid digit	Data address match check
ED	Edit	SS	S	S	T	T	C
EDMK	Edit and Mark	SS	S	S	T	T	C
TR	Translate	SS	S	S	T		C
TRT	Translate and Test	SS	S		T		C

Table 10-10 Miscellaneous Instructions - Interrupt Summary

Mnemonic	Instruction name	Instruction type	Execute error	Invalid 12 code	Write access	Physical access	Data address match check	Halfword alignment	Read access	Monitor call	Word alignment	Doubleword register error	Doubleword alignment
CDS	Compare double and Swap	RS			S	S	C		S			S	S
CS	Compare and Swap	RS			S	S	C		S		S		
EX	Execute	RX	S			S		S	S				
LA	Load Address	RX											
MC	Monitor Call	SI		S						C			
SPM	Set Program Mask	RR											
STCK	Store Clock	SI		S	S	T	C						
SVC	Supervisor Call	RR											
TS	Test and Set	SI			S	S	C		S				

10.6. Data Transfer Instructions *

These instructions move data without examining the bits in the data. The source location and the destination location can be either in memory or in a general register. Data in the source location is not altered unless the source and destination locations overlap in memory-to-memory operations.
The unit of data transferred can have one of the following field sizes:

1-16 words, 1 word, 1 halfword, 1-256 bytes, 1-16 M byte, 1 byte, 1-16 halfbytes, or a 1-32 bit field.

Data transfer can be accomplished form :

- Memory to register
- Register to memory
- Register to register
- Memory to memory

The instructions and the field sizes are displayed in Table 10-11.

Table 10-11 Summary of Data Transfer Instructions

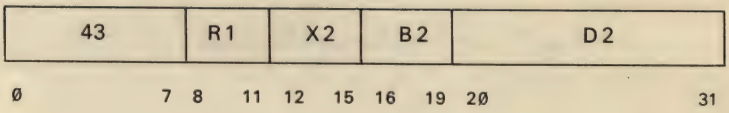
Source and destination of transfer	Units of data	Instructions
Memory to register	Several words	LM
	Word	L, LWI
	Halfword	LH
	Byte	IC
	Bit field	LBF
	Bytes	ICM
Register to memory	Several words	STM
	Word	ST, STWI
	Halfword	STH
	Byte	STC
	Bit field	STBF
	Bytes	STCM
Register to register	Word	LR
Memory to memory	Bytes	MVC, MVCL
	Halfbyte (zones)	MVZ
	Halfbyte (numeric)	MVN
	Halfbytes	MVO
	Immediate byte	MVI

* Data transfer in floating-point registers is discussed separately in 10.12, Floating-Point Instructions. Condition code: Only two data transfer instructions change the condition code, Insert Characters under Mask (ICM) and Move Long (MVCL).

10.6.1. Insert Character (IC)

Type : RX

Format : IC



Description :

Operand 2 (1 Byte, X2/B2/D2) is loaded into byte 3 of the general register specified by R1. Bytes Ø, 1, and 2 of the general register are unaltered.

Condition Code:

Unchanged

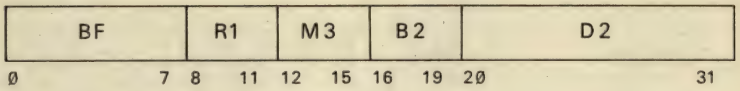
Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Data address match check	C

10.6.2. Insert Character under Mask (ICM)

Type : RS

Format : ICM



Description :

Bytes from operand 2 (B2/D2) are inserted into the general register specified by R1 under control of a mask. The 4 bits of M3 are used as the mask. They correspond one for one, in ascending order, with the 4 bytes in R1. The bytes corresponding to ones in the mask are replaced in R1 by consecutive bytes from operand 2 starting with B2/D2. Operand 2 is equal in length to the number of ones in the mask. The bytes in R1 corresponding to zeros in the mask remain unchanged.

If the mask is zero or if all inserted bytes are zero, the condition code is set to zero. Otherwise, the condition code is set according to the high-order bit of the first inserted byte.
If this bit is one, the condition code is set to 1 indicating a negative value; if the bit is zero, the condition code is set to 2 and indicates the positive value of the inserted characters.

Condition Code :

Ø All inserted bytes are zero, or mask is zero
1 First bit of inserted field is one
2 First bit of inserted field is zero
3 -

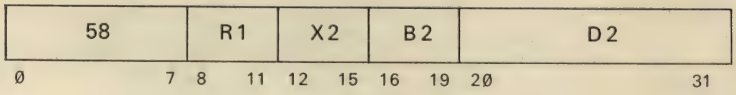
Interrupt Priorities :

Error	Action
Read access	S
Physical access	T
Data address match check	C

10.6.3. Load Word (L)

Type : RX

Format : L



Description :

Operand 2 (X2/B2/D2) is loaded into the general register specified by R1.

Condition Code:

Unchanged

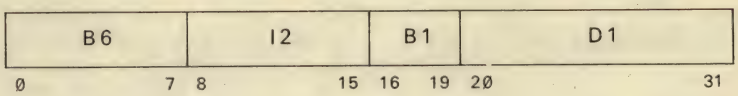
Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C
Data address match check	C

10.6.4. Load Bit Field (LBF)

Type: SI

Format: LBF



Description:

This instruction loads a bit field into the low-order bits of a general register and sets the remaining bits to zero.

The general register is determined by the processor state.
The following assignment applies:

Processor state	General register
P1	2
P2	2
P3	14
P4	10

The 5 high-order bits of the I2 field specify the bit field length (BFL) and the 3 low-order bits the bit field displacement (BFD). The number of the bits loaded will be one more than the value of BFL, i.e. 1 to 32 bits can be loaded. The start address of operand 1 (B1/D1) identifies the first byte from which bits will be loaded. Bits in the byte are numbered from Ø (high-order bit) to 7 (low-order bit). BFD specifies which bit is the leftmost to be loaded. Successive higher-numbered bits are loaded, continuing into the next higher-numbered bytes as necessary.

Condition Code:

Unchanged

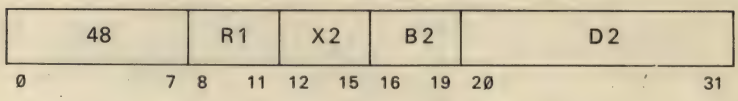
Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Data address match check	C

10.6.5. Load Halfword (LH)

Type: RX

Format: LH



Description:

Operand 2 (X2/B2/D2) is loaded into the general register specified by R1 and expanded from a halfword to a fullword by propagating the sign bit of operand 2 through the 16 high-order bits of R1.

Condition Code:

Unchanged

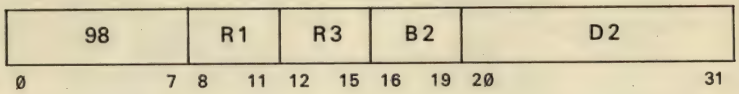
Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a halfword boundary)	S/C
Data address match check	C

10.6.6. Load Multiple (LM)

Type : RS

Format : LM



Description :

A set of consecutive general registers (with 15 to Ø wraparound) beginning with the register specified by R1 and ending with the register specified by R3 is loaded from operand 2. Operand 2 starts at word location B2/D2.
If R1 and R3 specify the same register, only one word is loaded.

Condition Code:

Unchanged

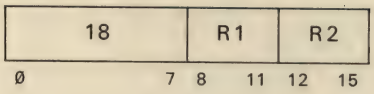
Interrupt Priorities:

Error	Action
Read access	S
Physical access	T
Byte boundary alignment (B2/D2 not a word boundary)	S/C
Data address match check	C

10.6.7. Load Word (LR)

Type: RR

Format: LR



Description:

The contents of the general register specified by R2 are loaded into the general register specified by R1.

Condition Code:

Unchanged

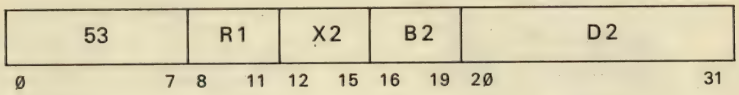
Interrupt Priorities:

None

10.6.8. Load Word Indirect (LWI)

Type: RX

Format: LWI

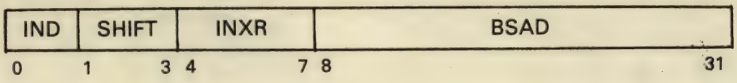


Description:

This instruction loads a word from memory into the general register specified by R1. The address of the word to be loaded is determined by a chain of indirect addresses starting with operand 2 (X2/B2/D2). LWI provides the facility for indirectly accessing a word in memory through a maximum of five indirect addressing levels.

R1 specifies the general register to be loaded. Operand 2 is the first indirect access word (IAF word), and consists of four fields.

IAF Format:



A word address is created from the IAF word by a two-step process. First, if INXR=Ø, the contents of the general register specified by INXR are accesses and shifted left by the number of bits specified in the SHIFT field. The result of this operation, called INDEX, is set to zero if INXR is zero. INDEX and the contents of the BSAD field are then added to obtain a word address which addresses either the word to be loaded (IND=Ø) or another IAF (IND=1).

If the IND bit of the fifth IAF word is one, a program interrupt will occur due to an indirect nesting error.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Indirect nesting error	S
Word alignment	S
Data address match check	C

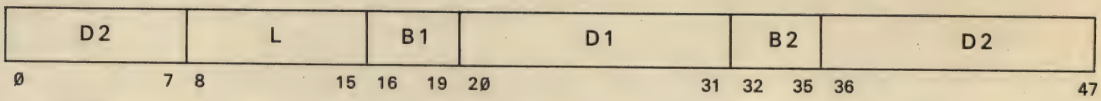
Notes:

1. All memory addresses must be word addresses (bits 3Ø, 31 zero).
2. The instruction must not be accessed by the Execute (EX) instruction.

10.6.9. Move (MVC)

Type: SS

Format: MVC



Description:

Operand 2 (B2/D2) is moved one byte at a time from left to right into operand 1 (B1/D1).

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Data address match check	C

Notes:

The operand fields may overlap. If the operand 1 address is one location to the right of the operand 2 address, the first byte of operand 2 is propagated through operand 1

10.6.10. Move Long (MVCL)

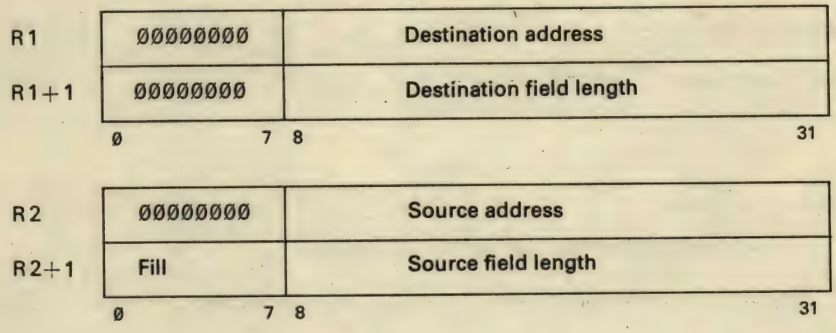
Type: RR

Format: MVCL

Description:

This instruction moves one memory field (operand 2) into another memory field (operand 1).

R1 and R2 each designate the even register of the even-odd pairs of general registers R1, R1+1 and R2, R2+1 which contain the required parameters.



Bits 8-31 of R1 and R2 contain the start addresses of operand 1 (destination address) and operand 2 (source address) respectively.

Bits 8-31 of R1+1 and R2+1 contain the counts for operand 1 (destination field length) and operand 2 (source field length) respectively.

Bits Ø-7 of R2+1 contain the padding character.

Bits Ø-7 of R1, R1+1 and R2 are ignored.

The source field is moved into the destination field one byte at a time from left to right. The operation is ended when the number of bytes specified in the operand 1 count has been moved from the source field into the destination field. If operand 2 is shorter than operand 1, the remaining bytes of operand 1 are filled with the padding character. The bytes moved are not changed or inspected.

The two count fields are compared prior to instruction execution to determine the condition code, and a check is made for destructive overlap of the operands. This is the case if a portion of operand 1 is used as a source after data has already been moved from operand 2 into this portion. When the operands overlap destructively, instruction execution is suppressed, the condition code is set to 3, and bits Ø-7 of R1 and R2 are set to zero.

To enable external interrupts to be handled during instruction execution, operation is halted and an interrupt permitted whenever the destination address or the source address reaches a page boundary. The contents of the general registers change during instruction execution. These registers always contain current addresses and lengths, thus permitting instruction execution to be resumed correctly after each interruption.

Condition Code:

Ø Operand 1 and operand 2 counts

1 Operand 1 count is low

2 Operand 1 count is high

3 No movement performed due to destructive overlap

Interrupt Priorities:

Error	Action
Doubleword register error	S
Read access *)	S
Write access *)	S
Physical access	T
Data address match check **)	C

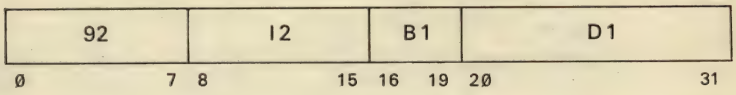
*) For each execution phase, i.e. within page boundaries.

**) May occur at the end of an execution phase prior to instruction conclusion.

10.6.11. Move (MVI)

Type: SI

Format: MVI



Description:

The immediate byte I1 is placed in the operand 1 location (B1/D1).

Condition Code:

Unchanged

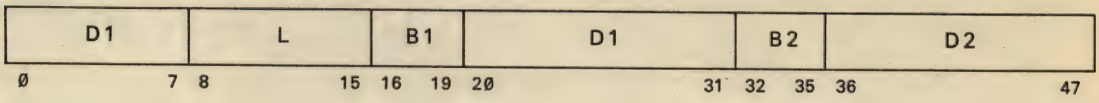
Interrupt Priorities:

Error	Action
Write access	S
Physical access	S
Data address match check	C

10.6.12. Move Numerics (MVN)

Type: SS

Format: MVN



Description:

The 4 low-order bits (numeric field) of each byte of operand 2 (B2/D2) are moved from left to right into the corresponding bits of the bytes of operand 1 (B1/D1).

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Data address match check	C

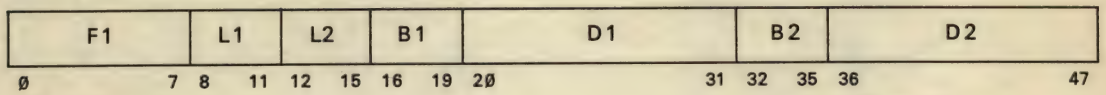
Notes:

1. If the operand 1 address is one location to the right of the operand 2 address, the numeric field of the first byte of operand 2 is propagated through the numeric fields of all bytes of operand 1.
2. The 4 high-order bits (zone field) of the bytes of operand 1 are unchanged.

10.6.13. Move with Offset (MVO)

Type: SS

Format: MVO



Description:

Operand 2 (B2/D2) is moved from right to left into operand 1 (B1/D1) to the left of, and adjacent to, the 4 low-order bits of operand 1. The operand fields may overlap. If operand 1 is not large enough, to receive all halfbytes of operand 2, the remaining halfbytes are ignored. If operand 2 is shorter than operand 1, operand 2 is extended with high-order zeros.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Data address match check	C

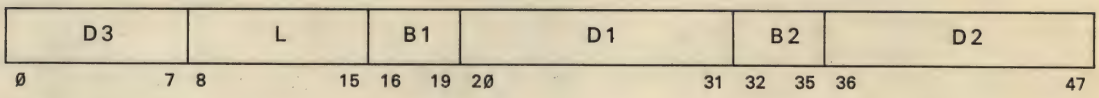
Notes:

1. At the beginning of execution the addresses of operand 1 and operand 2 are checked. An interrupt condition may occur on checking the address of operand 2 that would not occur during execution. This interrupt will occur if operand 1 is shorter than operand 2 and the addresses of the unused bytes of operand 2 cause the interrupt condition. It is a user programming responsibility to avoid this condition.
2. The 4 low-order bits of operand 1 are attached to the right of operand 2. Operand 2 is then offset 4 bits to the left and moved into the field of operand 1. If necessary, operand 2 is extended with high-order zeros.
3. Operands 1 and 2 are not checked for valid codes.

10.6.14. Move Zones (MVZ)

Type: SS

Format: MVZ



Description:

The 4 high-order bits (zone field) of each byte of operand 2 (B2/D2) are moved from left to right into the corresponding bits of the bytes of operand 1 (B1/D1).

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Data address match check	C

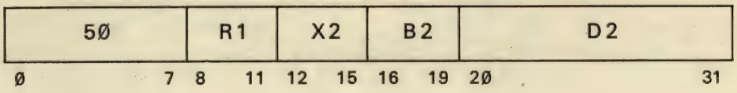
Notes:

1. If the operand 1 address is one location to the right of the operand 2 address, the zone field of the first byte of operand 2 is propagated through the zone fields of all bytes of operand 1.
2. The 4 low-order bits (numeric field) of the bytes of operand 1 are unchanged.

10.6.15. Store Word (ST)

Type: RX

Format: ST



Description:

The contents of the general register specified by R1 are stored into operand 2 (X2/B2/D2).

Condition Code:

Unchanged

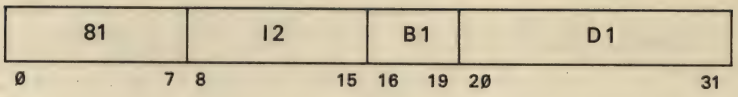
Interrupt Priorities:

Error	Action
Write access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C
Data address match check	C

10.6.16. Store Bit Field (STBF)

Type: SI

Format: STBF



Description:

This instruction stores 1 to 32 low-order bits of a general register into operand 1.

The general register is determined by the processor state. The following assignment applies:

Processor state	General register
P1	2
P2	2
P3	14
P4	10

The 5 high-order bits of the I2 field specify the bit field length (BFL) and the 3 low-order bits the bit field displacement (BFD). The number of bits stored will be one more than the value of BFL. BFD specifies the number of the bit in the leftmost byte of operand 1 (B1/D1), starting at which the bit field is to be stored. Successive higher-numbered bits are stored, continuing into the next higher-numbered bytes as necessary. Bits in a byte are numbered from Ø (higher-order bit) to 7 (low-order-bit).

Condition Code:

Unchanged

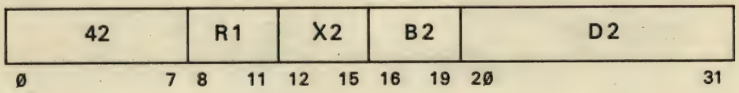
Interrupt Priorities:

Error	Action
Write access	S
Physical access	S
Data address match check	C

10.6.17. Store Character (STC)

Type: RX

Format: STC



Description:

The rightmost byte (byte 3) of the general register specified by R1 is stored into operand 2 (X2/B2/D2).

Condition Code:

Unchanged

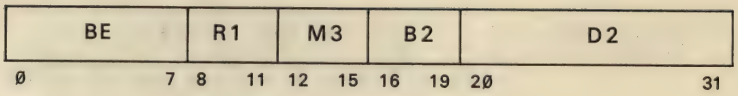
Interrupt Priorities:

Error	Action
Write access	S
Physical access	S
Data address match check	C

10.6.18. Store Characters under Mask (STCM)

Type: RS

Format: STCM



Description:

Bytes selected from the general register specified by R1 under control of a mask are stored consecutively into operand 2 (B2/D2). Operation is from left to right. The 4 bits of the mask M3 correspond one for one with the 4 bytes of R1. The bytes of R1 corresponding to ones in the mask are stored in consecutive locations beginning with (B2/D2). The number of bytes to be stored is equal to the number of one-bits in the mask.

Condition code:

Unchanged

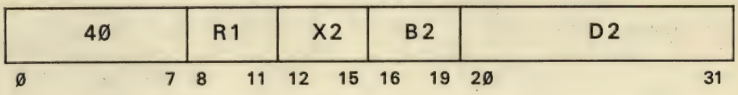
Interrupt Priorities:

Error	Action
Write access	S
Physical access	T
Data address match check	C

10.6.19. Store Halfword (STH)

Type: RX

Format: STH



Description:

The two rightmost bytes (bytes 2 and 3) of the general register specified by R1 are stored into operand 2 (X2/B2/D2).

Condition Code:

Unchanged

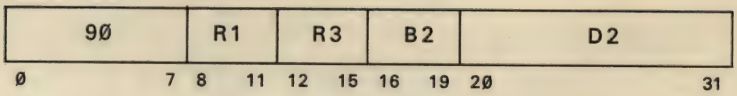
Interrupt Priorities:

Error	Action
Write access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a halfword boundary)	S/C
Data address match check	C

10.6.20. Store Multiple (STM)

Type: RS

Format: STM



Description:

A set of consecutive general registers (with 15 to Ø wraparound) beginning with the register specified by R1 and ending with the register specified by R3 is stored into operand 2. Operand 2 starts at word location B2/D2. If R1 and R3 specify the same register, only one word is stored.

Condition Code:

Unchanged

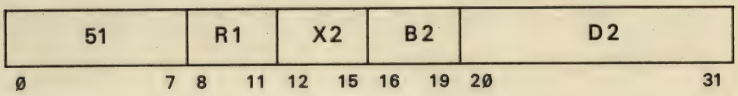
Interrupt Priorities:

Error	Action
Write access	S
Physical access	T
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C
Data address match check	C

10.6.21. Store Word Indirect (STWI)

Type: RX

Format: STWI

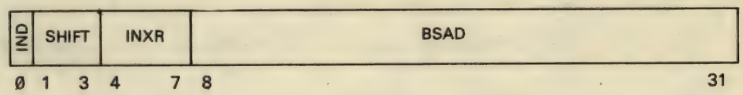


Description:

This instruction stores the contents of the general register specified by R1 into memory. The memory address is determined by a chain of indirect addresses starting with operand 2 (X2/B2/D2). STWI provides the facility for indirectly accessing a word in memory through a maximum of five indirect addressing levels.

Operand 2 is the first indirect access word (IAF word), and consists of four fields.

IAF Format:



A word address is created from the IAF word by a two-step process. First, if INXR=Ø, the contents of the general register specified by INXR are accessed and shifted left by the number of bits specified in the SHIFT field. The result of this operation, called INDEX, is set to zero if INXR is zero. INDEX and the contents of the BSAD field are then added to obtain a word address which addresses either another IAF (IND=1) or the memory word (IND=Ø) into which the contents of R1 are to be stored.

If the IND bit of the fifth IAF word is one, a programm interrupt will occur due to an indirect nesting error.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Write access	S
Physical access	S
Indirect nesting error	S
Word alignment	S
Data address match check	C

Notes:

1. All memory addresses must be word addresses
2. The instruction must not be accessed by the Execute (EX) instruction.

10.7. Branch Instructions

Under normal circumstances the central unit processes the instructions in the order in which they are stored in memory. Branch instructions can cause a break in this normal sequential execution. They provide for referencing another subroutine or, in the case of program loops, repeating a segment of coding or continuing to the next instruction in sequence. When a branch is to be made, the next instruction address (NIA) in the Program Counter Register (PCR) is substituted by the address specified in the branch instruction.

There are unconditional and conditional branch instructions. Unconditional branch instructions always cause the instruction address in the PCR to be substituted by the branch address.
Conditional branch instructions cause either the next instruction in sequence or the instruction specified by the branch address to be performed depending on the condition code or the contents of a general register.

When the BALR and BAL instructions are executed, the instruction address from the PCR is saved before the branch address is placed in the PCR. This saved address serves as a return address.

The PCR normally contains the address of the next instruction to be executed. This address is obtained in the instruction staticizing phase by adding the length (in bytes) of the current instruction to the NIA. After this instruction has been executed, the next instruction is read with the new NIA.

Branch instructions are RR, RX, or RS type. The branch address is contained in the general register specified by R2 or in word location (X2/B2/D2) or (B2/D2).

The condition code is not changed by any of the branch instructions.

10.7.1 Audit Mode

The audit mode is an effective means for backward tracing on branch instructions. If a program error occurs when the audit mode is on, the addresses of the last 64 branch instructions that caused a branch are available in the audit table, permitting the program flow to be traced back to the error source.

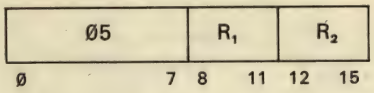
The audit mode is on when the AUDT bit in the Interrupt Status Register (ISR) is a one. If a branch is made and AUDT=1, the address of this branch instruction is stored in the audit table. The audit table is a contiguous main memory area of 64 words and is addressed by the Audit Address Register (AAR). The AAR contains a 24-bit real address which points to the next free word in the audit table and is incremented by 4 modulo 256 after each entry in the audit table.

When the audit table is full (i.e. when the low-order byte of the AAR is zero after incrementing) and the PAWM bit is set in the ISR, an audit wrap error is generated. Bit 8 is set in the Program Interrupt Flag Register (PIFR) and an audit wrap (error class 2Ø) is indicated in the Error Cause Register (ERCR).

10.7.2. Branch and Link (BALR, BAL)

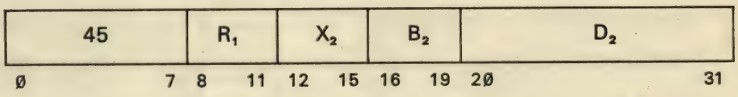
Type: RR

Format: BALR



Type: RX

Format: BAL



Description:

The branch address (R2 or X2/B2/D2) is saved; then the contents of the 32-bit Program Counter Register (PCR) are loaded into the general register specified by R1. The saved branch address is then loaded into the PCR. The condition code and the program mask remain unchanged. The instruction length code is set to zero. When the audit mode is on, the address of any BAL instruction and of a BALR instruction with R2 nonzero is entered in the audit table.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Physical access	C
Audit wrap	C

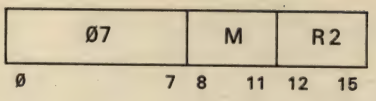
Notes:

1. If R2 of the BALR instruction is zero, the contents of the PCR are saved and no branch is made.
2. The contents of R2 of the BALR instruction are only changed if R1 = R2.

10.7.3 Branch on Condition (BCR, BC)

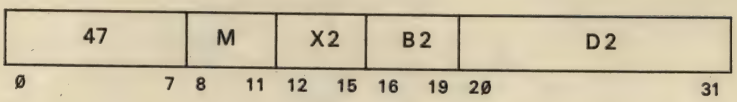
Type: RR

Format: BCR



Type: RX

Format: BC



Description:

If the condition code corresponds to one of the values specified by the mask field M, the Program Counter Register (PCR) is loaded with the branch address (R2 or X2/B2/D2). The condition code and the program mask remain unchanged. The instruction length code is set to zero. If the audit mode is on, the address of the branch instruction is entered in the audit table.

If the condition code does not correspond to any of the values specified by the mask field M, the PCR remains unchanged and the next instruction in sequence in memory is executed.

The positions in the mask field correspond left to right to the condition codes as follows:

Instruction bit	Condition code
8	Ø
9	1
1Ø	2
11	3

If all bits in the mask field M are one (instruction: BCR), or if the RR field is zero, serialization is performed, i.e. all processor instructions are executed and no successor instruction is prefetched until execution of the current instruction has been completed.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Physical access	C
Audit wrap	C

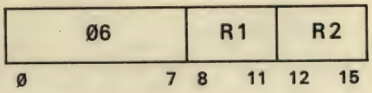
Notes:

1. If all the bits in the mask field are one, an unconditional branch is performed.
2. If all the bits in the mask field are zero, or if R2 of the BCR instruction is zero, no branch is performed. Then serialization in the processor working mode follows. All previous instructions are executed, no further instruction information is prefetched.
3. The contents of R2 are not changed.

10.7.4. Branch on Count (BCTR, BCT)

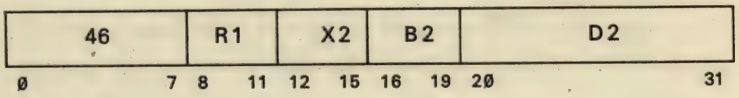
Type: RR

Format: BCTR



Type: RX

Format: BCT



Description:

The contents of the general register specified by R1 are decremented by one algebraically. Overflow occurring on transition from the maximum negative number to the maximum positive number is ignored.

If the result is not zero, the saved branch address (R2 or X2/B2/D2) is loaded into the PCR. The condition code and the program mask are not changed. The instruction length code is set to zero, and the address of the branch instruction is entered in the audit table if the audit mode is on.

If the result is zero, the PCR remains unchanged and the next instruction in sequence in memory is executed.

Condition code:

Unchanged

Interrupt Priorities:

Error	Action
Physical access	C
Audit wrap	C

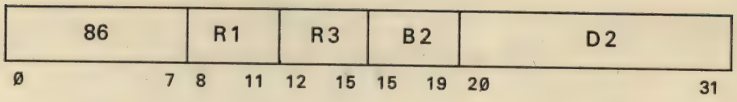
Notes:

1. As in fixed-point arithmetic, the subtraction affects all 32 bits.
2. The contents of R2 remain unchanged.
3. If R2 of the BCTR instruction is zero, the contents of R1 are reduced by one but no branch is made.

10.7.5. Branch on Index High (BXH)

Type: RS

Format: BXH



Description:

The contents of the general register specified by R3 are added to the contents of the general register specified by R1. A binary fixed-point addition is performed and any overflow occurring is ignored. The sum in R1 is compared with a comparand. If the register address of R3 is even, the contents of R3+1 serve as comparand; if the address is odd, the contents of R3 serve as comparand.

If the sum in R1 is high, the saved branch address (B2/D2) is loaded into the PCR. The condition code and the program mask are not changed. The instruction length code is set to zero. If the audit mode is on, the address of the branch instruction is entered in the audit table.

If the sum in R1 is low or equal, the PCR remains unchanged and the next instruction in sequence in memory is executed.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Physical access	C
Audit wrap	C

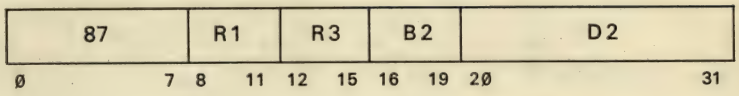
Notes:

1. The sum is stored in R1 after the comparison, regardless of the result.
2. The contents of R3 or R3+1 are not changed.

10.7.6. Branch on Index Low or Equal (BXLE)

Type: RS

Format: BXLE



Description:

The contents of the general register specified by R3 are added to the contents of the general register specified by R1. A binary fixed-point addition is performed and any overflow occurring is ignored. The sum in R1 is compared with a comparand. If the register address of R3 is even, the contents of R3+1 serve as comparand; if the address is odd, the contents of R3 serve as comparand.

If the sum in R1 is low or equal, the saved branch address (B2/D2) is loaded into the PCR. The condition code and the program mask are not changed. The instruction length code is set to zero. If the audit mode is on, the address of the branch instruction is entered in the audit table.

If the sum in R1 is high, the PCR remains unchanged and the next instruction in sequence in memory is executed.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Physical access	C
Audit wrap	C

Notes:

1. The sum is stored in R1 after the comparison, regardless of the result.
2. The contents of R3 or R3+1 are not changed.

10.8. Logical Instructions

These instructions are used for the logical processing of data. The logical operations AND, OR, and Exclusive OR are provided. The operands may be byte or word-oriented and are treated as strings of bits. The specified operation is performed bit by bit on the two operands with the result replacing the first operand except for the Test under Mask instruction which leaves the two operands unchanged.

10.8.1. Arithmetic

The arithmetic for the logical instructions AND, OR and Exclusive OR is shown in the following truth table:

Corresponding bits in		Result bit		
Operand 1	Operand 2	AND	OR	Exclusive OR
Ø	Ø	Ø	Ø	Ø
Ø	1	Ø	1	1
1	Ø	Ø	1	1
1	1	1	1	Ø

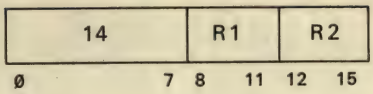
Condition code utilization for logical instructions:

Instruction	Condition code setting			
	Ø	1	2	3
AND	Zero	Not zero	-	-
OR	Zero	Not zero	-	-
Exclusive OR	Zero	Not zero	-	-
Test under Mask	Zero	Mixed	-	One

10.8.2. AND (NR, N, NI, NC)

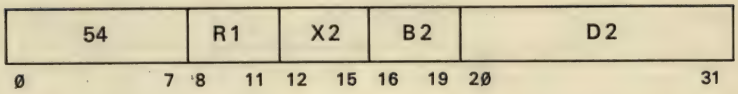
Type: RR

Format: NR



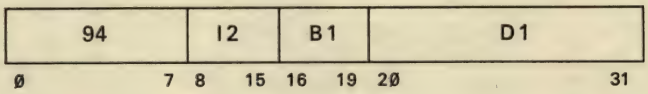
Type: RX

Format: N



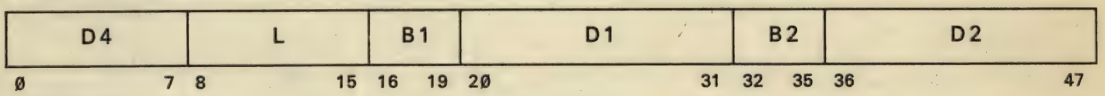
Type: SI

Format: NI



Type: SS

Format: NC



Description:

An AND operation is performed on the corresponding bits of the two operands. The result replaces operand 1 in the general register specified by R1, or location B1/D1, depending on the type of instruction. Operand 2 is either the contents of the general register specified by R2, location X2/B2/D2, the immediate byte in I2 or location B2/D2, depending on the type of instruction.

Condition Code:

Ø	Result is zero
1	Result not zero
2	-
3	-

Interrupt Priorities:

Error \ Instruction	Action			
	NR	N	NI	NC
Write access	-	-	S	S
Read access	-	S	-	S
Physical access	-	S	S	T
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	S/C	-	-
Data address match check	-	C	C	C

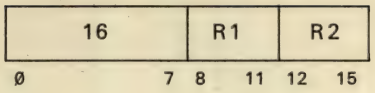
Notes:

1. Operand 2 is only changed if, in the case of NC, the operands overlap.
2. Operation is from left to right.
3. All operands and results are valid.

10.8.3. OR (OR, O, OI, OC)

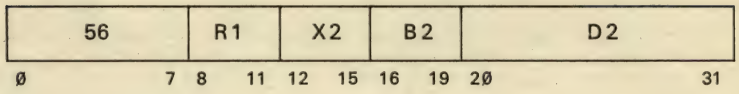
Type: RR

Format: OR



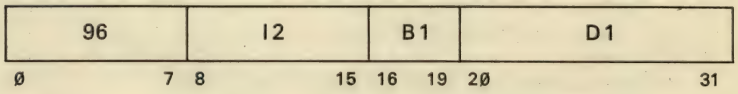
Type: RX

Format: O



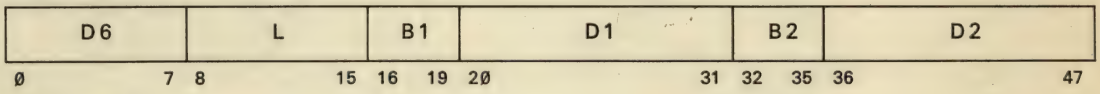
Type: SI

Format: OI



Type: SS

Format: OC



Description:

An OR operation is performed on the corresponding bits of the two operands. The result replaces operand 1 in the general register specified by R1, or location B1/D1, depending on the type of instruction. Operand 2 is either the contents of the general register specified by R2, location X2/B2/D2, the immediate byte in I2 or location B2/D2, depending on the type of instruction.

Condition Code:

Ø Result is zero

1 Result not zero

2 -

3 -

Interrupt Priorities:

Error \ Instruction	Action			
	OR	O	OI	OC
Write access	-	-	S	S
Read access	-	S	-	S
Physical access	-	S	S	T
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	S/C	-	-
Data address match check	-	C	C	C

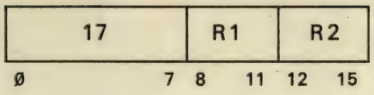
Notes:

1. Operand 2 is only changed, if, in the case of OC, the operands overlap.
2. Operation is from left to right.
3. All operands and results are valid.

10.8.4. Exclusive OR (XR, X, XI, XC)

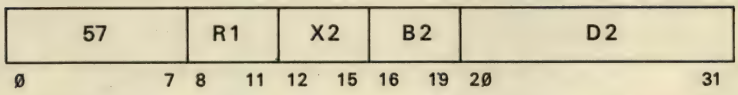
Type: RR

Format: XR



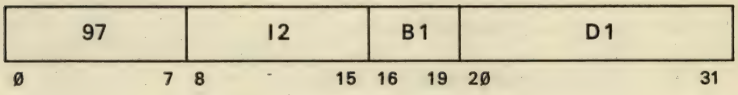
Type: RX

Format: X



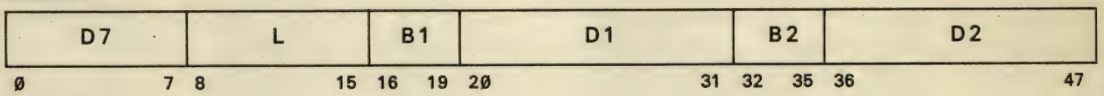
Type: SI

Format: XI



Type: SS

Format: XC



Description:

An Exclusive OR operation is performed on the corresponding bits of the two operands. The result replaces operand 1 in the general register specified by R1, or location B1/D1, depending on the type of instruction. Operand 2 is either the contents of the general register specified by R2, location X2/B2/D2, the immediate byte in I2 or location B2/D2, depending on the type of instruction.

Condition Code:

Ø Result is zero

1 Result not zero

2 -

3 -

Interrupt Priorities:

Error \ Instruction	Action			
	XR	X	XI	XC
Write access	-	-	S	S
Read access	-	S	-	S
Physical access	-	S	S	T
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	S/C	-	-
Data address match check	-	C	C	C

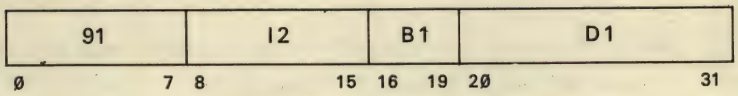
Notes:

1. Operand 2 is only changed if, in the case of XC, the operands overlap.
2. Operation is from left to right.
3. All operands and results are valid.
4. These instructions can be used to complement a number (ones complement).

10.8.5. Test under Mask (TM)

Type: SI

Format: TM



Description:

The condition code is determined by the state of the bits in operand 1 (B1/D1) selected by the mask (I2 field).

The immediate operand in the I2 field of the instruction is an 8-bit mask. The 8 bits in the mask are made to correspond one for one with the 8 bits of operand 1 (B1/D1).

If a mask bit is 1, the corresponding bit in operand 1 is tested. If a mask bit is Ø, the corresponding bit in operand 1 is ignored.

If all operand 1 bits tested are zero, the condition code is set to Ø. The condition code is also set to Ø if all mask bits are zero. If all bits tested are one, the condition code is set to 3.

The operands are not changed.

Condition Code:

Ø Selected bits all zeros, or mask is all zeros
1 Selected bits mixed zeros and ones
2 -
3 Selected bits all ones

Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Data address match check	C

10.9. Binary Instructions

These instructions are used for performing binary operations on unsigned binary numbers. The RR, RX, SI, and SS types are used.

10.9.1. Representation of Numbers

Binary instructions operate on fullword and doubleword operands which represent unsigned 32- or 64-bit integers and contain no specially designated bits.

Subtract and Compare instructions operate in twos complement notation of operands.

10.9.2. Arithmetic

The execution of arithmetic binary instructions is based on the binary arithmetic described under 10.10.2.

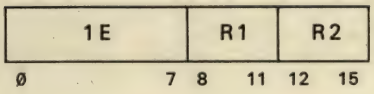
The binary operands however have no sign position and there is no facility for overflow recognition. As compared to binary arithmetic, the following differences must be observed:

- There are no condition codes for greater/less than zero.
- There is no condition code for overflow.
- When shifts to the right are performed, zeros are always shifted in from the left.

10.9.3. Add Logical (ALR, AL)

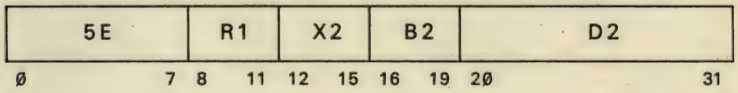
Type: RR

Format: ALR



Type: RX

Format: AL



Description:

Operand 1 and operand 2 are added logically (32 bits unsigned) and the sum is loaded into R1. Operand 1 is the contents of the general register specified by R1; operand 2 the contents of the general register specified by R2, or location X2/B2/D2, depending on the type of instruction.

Condition Code:

Ø Sum is zero, no carry
1 Sum is not zero, no carry
2 Sum is zero, carry occurred
3 Sum is not zero, carry occurred

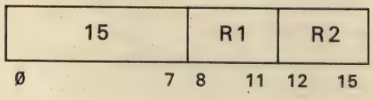
Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C
Data address match check	C

10.9.4. Compare Logical (CLR, CL, CLI, CLC)

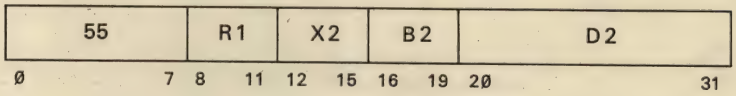
Type: RR

Format: CLR



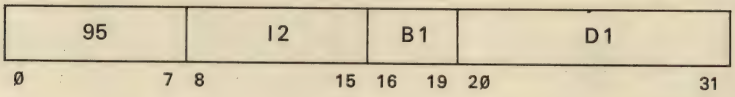
Type: RX

Format: CL



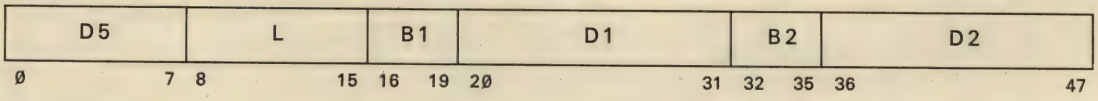
Type: SI

Format: CLI



Type: SS

Format: CLC



Description:

Operand 1 is compared with operand 2 bit by bit from left to right. The operation terminates when an inequality is found or when the operand bytes have been exhausted. On inequality, the operand whose last bit position compared contains a zero is the smaller operand.

Operand 1 is the contents of the general register specified by R1, or location B1/D1, depending on the type of instruction.
Operand 2 is the contents of the general register specified by R2, location X2/B2/D2, the immediate byte in I2 or location B2/D2, depending on the type of instruction.

Condition Code:

Ø Operands are equal
1 Operand 1 is low
2 Operand 1 is high
3 -

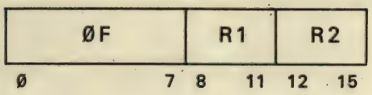
Interrupt Priorities:

Error \ Instruction	Action			
	CLR	CL	CLI	CLC
Read access	-	S	S	S
Physical access	-	S	S	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	S/C	-	-
Data address match check	-	C	C	C

10.9.5. Compare Logical Long (CLCL)

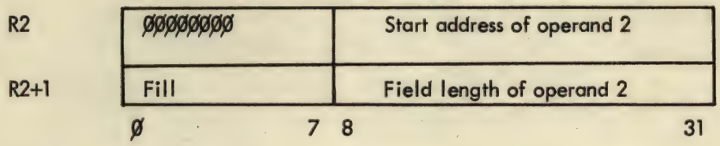
Type: RR

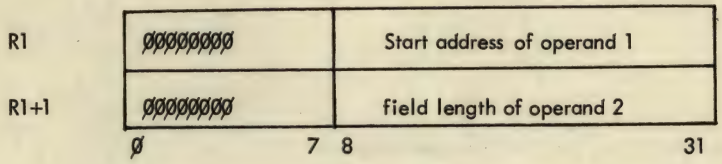
Format: CLCL



Description:

Operand 1 is compared logically with operand 2 and the result is indicated by the condition code. R1 and R2 each designate the even register of the even-odd pairs of general registers R1, R1+1 and R2, R2+1 which contain the required parameters.





Bits 8-31 of R1 and R2 contain the start addresses of operand 1 and operand 2 respectively.

Bits 8-31 of R1+1 and R2+1 contain the field lengths (number of bytes) of operand 1 and operand 2 respectively.

Bits Ø-7 of R2+1 contain the padding character.

Bits Ø-7 of R1, R1+1 and R2 are ignored.

The comparison is carried out from left to right, the operands being treated as unsigned binary numbers. The operation terminates as soon as an inequality is detected or the end of the longer operand is reached. If the operands are of unequal length, the shorter operand is considered extended with the padding character to the length of the longer operand. The operands remain unchanged.

If both operand lengths are zero, the operands are considered equal.
To permit the processing of external interrupts occurring during instruction execution, operation is halted and an interrupt permitted whenever the address of operand 1 (or the address of operand 2 if operand 1 is the shorter operand and has already been exhausted) reaches a page boundary. The contents of the general registers are used during instruction execution. These registers always contain current addresses and lengths, thus permitting instruction execution to be resumed correctly after each interruption.
If the operation ends because of mismatch, the addresses in R1 and R2 identify the nonmatching pair of bytes. The field lengths in R1+1 and R2+1 are decremented by the number of matching bytes unless the mismatch occurred with the padding character. In this case the field length of the shorter operand is zero.

If the two operands inclusive of padding characters, if any, are equal, both field lengths are zero at instruction end. The addresses are incremented by the respective field lengths without padding characters.

Condition Code:

Ø Operands are equal, or both field lengths are zero

1 Operand 1 is low

2 Operand 1 is high

3 -

Interrupt Priorities:

Error	Action
Doubleword register error	S
Read access *)	S
Physical access *)	S
Data address match check **)	C

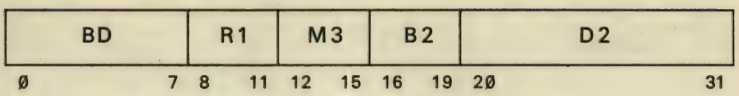
*) For each execution phase, i.e. within page boundaries.

**) May occur at the end of an execution phase prior to instruction conclusion.

10.9.6. Compare Logical Characters under Mask (CLM)

Type: RS

Format: CLM



Description:

Operand 2 (B2/D2) is compared with the contents of the general register specified by R1 under control of a mask. The result determines the condition code.

The four bits in the mask (M3 field) correspond in ascending order with the four bytes of R1. The bytes in R1 corresponding to ones in the mask are considered as a contiguous field and are compared with operand 2. Operand 2 is equal in length to the number of one-bits in the mask. The bytes in R1 corresponding to zeros in the mask are ignored.

The comparison is made from left to right, the operands being treated as unsigned binary integers.

Condition Code:

Ø Selected bytes are equal, or mask is zero
1 Selected bytes in R1 are low
2 Selected bytes in R1 are high
3 -

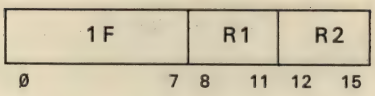
Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Data address match check	C

10.9.7. Subtract Logical (SLR, SL)

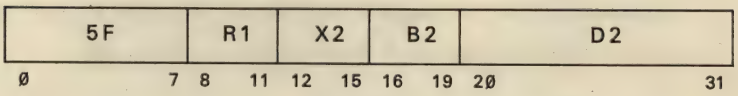
Type: RR

Format: SLR



Type: RX

Format: SL



Description:

Operand 2 is subtracted logically (32 bits unsigned) from operand 1 by logically adding the twos complement of operand 2 to operand 1. The result is stored in R1. A carry (out of bit Ø) may occur either on complementing operand 2 (if operand 2 was zero) or on adding.

Operand 1 is the contents of the general register specified by R1; operand 2 the contents of the general register specified by R2, or location X2/B2/D2, depending on the type of instruction.

Condition Code:

Ø -
1 Result not zero, no carry
2 Result is zero, carry occurred
3 Result not zero, carry occurred

Interrupt Priorities:

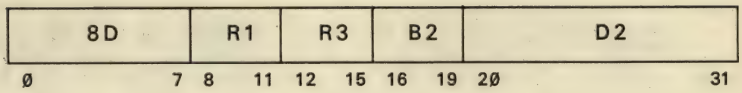
(For SL only)

Error	Action
Read access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C
Data address match check	C

10.9.8. Shift Left Double Logical (SLDL)

Type: RS

Format: SLDL



Description:

The contents of the pair of general registers specified by R1 and R1+1 (R1 even-numbered) are considered as a doubleword and shifted left by the number of bit positions specified by the low-order six bits of the address B2/D2. The remaining bits of B2/D2 and the R3 field are ignored. The high-order bits shifted out of the doubleword are lost and zeros are shifted in from the right.

Condition Code:

Unchanged

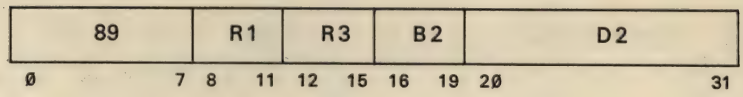
Interrupt Priorities:

Error	Action
Doubleword register error	S

10.9.9. Shift Left Single Logical (SLL)

Type: RS

Format: SLL



Description:

The contents of the general register specified by R1 are shifted left by the number of bit positions specified by the low-order six bits of the address B2/D2. The remaining bits of B2/D2 and the R3 field are ignored. The high-order bits shifted out are lost and zeros are shifted in from the right.

Condition Code:

Unchanged

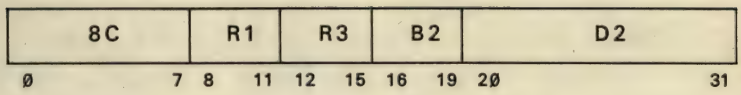
Interrupt Priorities:

None

10.9.10. Shift Right Double Logical (SRDL)

Type: RS

Format: SRDL



Description:

The contents of the pair of general registers specified by R1 and R1+1 (R1 even-numbered) are considered as a doubleword and shifted right by the number of bit positions specified by the low-order six bits of the address B2/D2. The remaining bits of B2/D2 and the R3 field are ignored. The low-order bits shifted out of the odd-numbered register are lost and zeros are shifted in from the left.

Condition Code:

Unchanged

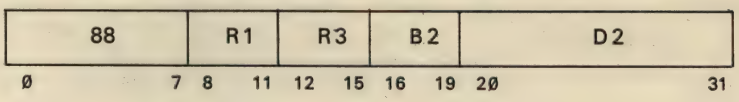
Interrupt Priorities:

Error	Action
Doubleword register error	S

10.9.11. Shift Right Single Logical (SRL)

Type: RS

Format: SRL



Description:

The contents of the general register specified by R1 are shifted right by the number of bit positions specified by the low-order six bits of the address B2/D2. The remaining bits of B2/D2 and the R3 field are ignored. The low-order bits shifted out are lost and zeros are shifted in from the left.

Condition Code:

Unchanged

Interrupt Priorities:

None

10.10. Fixed-Point Instructions for Signed Binary Numbers

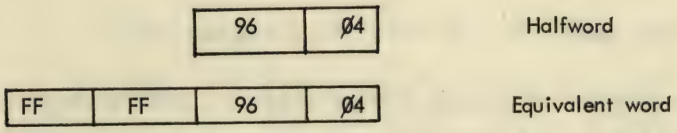
This group of instructions provides the operations necessary to add, subtract, multiply, divide, shift, compare and do sign manipulation on fixed-point, binary, signed operands. The RR, RX, and RS types are used.

10.10.1. Representation of Numbers

The fixed-point instructions operate on halfword, word or doubleword operands which represent signed binary numbers in twos complement integer notation.

A halfword operand is converted to a word operand by propagating the high-order bit (sign bit) of the halfword through the high-order 16 bits of the word and loading the halfword into the low-order 16 bits of the word.

Example:



10.10.2. Arithmetic

The binary arithmetic which is required for the execution of fixed-point instructions is defined as follows:

- A bit add is the operation using as input one bit from the first operand (operand 1), one bit from the second operand (operand 2), and one bit called PI (propagate in). The output is a single-bit result and a single-bit PO (propagate out).
  The definition of this function is:

Operand 1	Ø	1	Ø	1	Ø	1	Ø	1
Operand 2	Ø	Ø	1	1	Ø	Ø	1	1
PI	Ø	Ø	Ø	Ø	1	1	1	1
Result	Ø	1	1	Ø	1	Ø	Ø	1
PO	Ø	Ø	Ø	1	Ø	1	1	1

- A field add is the bit-by-bit operation right to left on operand 1 and operand 2, producing a result. For each bit add, PI is the PO value of the preceding bit add. If the fields are of different lengths, the shorter is extended to the left propagating the high-order bit.

- Initial propagate (IP) is the PI for the first bit add (on the low-order bit) of a field add.
- Carry is name of the PO value of the high-order bit add of a field add.
- A binary add on operand 1 and operand 2 is defined as a field add with IP equal to zero.
- The ones complement of operand 2 is the result of changing every one bit to a zero and every zero bit to a one in operand 2.
- A binary subtract on operand 1 and operand 2 (operand 1 minus operand 2) is defined as a field add on operand 1 and the ones complement of operand 2 with IP equal to one.
- An overflow occurs on a field add if PI and PO for the high-order bit add are different.
- The sign of result is positive if the high-order bit is zero, and negative if the high-order bit is one. A positive result may be equal to or greater than zero. A negative result is always less than zero.
- The magnitude of a result is zero if all bits are zero.
- Signed binary numbers are represented in twos complement notation.
- The twos complement of operand 1 is formed by adding ...ØØØ1 to the ones complement of operand 1. If operand 1 is composed of a one followed by all zeros (maximum negative number), its twos complement is undefined. (Overflow).
- The magnitude of a positive number is its binary value. The magnitude of a negative number is the magnitude of the twos complement of that number. A sample, using twos complement, of three-bit binary numbers is given in the following table with their decimal equivalents.
- Arithmetic right-shifting causes the sign bit to the propagated through the vacated high-order bit positions. Low-order bits shifted out are lost.
- Arithmetic left-shifting causes low-order zeros to be shifted in. The sign bit in the leftmost position remains unchanged while the remaining bits are shifted. If a bit shifted out disagrees with the sign bit, a fixed-point overflow occurs.

Twos complement number	Decimal equivalent
Ø11	+3
Ø1Ø	+2
ØØ1	+1
ØØØ	Ø
111	-1
11Ø	-2
1Ø1	-3
1ØØ	-4

10.10.3. Condition Code

The condition code is not changed on Multiply or Divide instructions. On all other instructions it is set in accordance with the following general scheme:

Ø Result is zero

1 Result is less than zero

2 Result is greater than zero

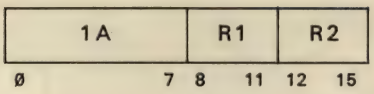
3 Overflow

Overflow does not occur on CR, C, CH, MR, M, MH, DR, D, LTR, LNR, SRA and SRDA instructions.

10.10.4. Add Word (AR, A)

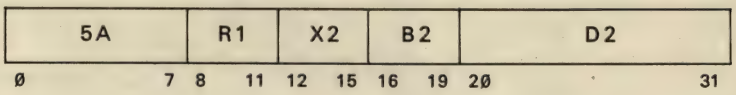
Type: RR

Format: AR



Type: RX

Format: A



Description:

Operand 1 (contents of the general register specified by R1) and operand 2 (memory word X2/B2/D2 or contents of the general register specified by R2) are added, and the result is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Interrupt Priorities:

Instruction	Action	
Error	A	AR
Read access	S	-
Physical access	S	-
Fixed-point overflow	C	C
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C	-
Data address match check	C	-

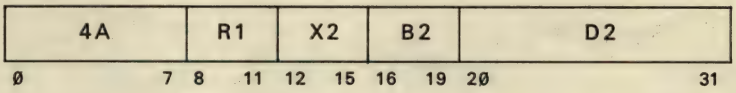
Notes:

All 32 bits of both operands participate in the addition. If the carries into and out of the sign position disagree, a fixed-point overflow condition exists. The result has a sign that is the opposite of the true algebraic result. If the fixed-point overflow mask bit (bit 4 in the PCR) is set to one, a fixed-point overflow interrupt occurs.

10.10.5. Add Halfword (AH)

Type: RX

Format: AH



Description:

Operand 2 (contents of location X2/B2/D2) is expanded to a fullword and added to the contents of the general register specified by R1.
The result is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Fixed-point overflow	C
Byte boundary alignment (X2/B2/D2 not a halfword boundary)	S/C
Data address match check	C

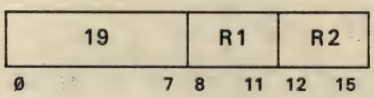
Notes:

All 32 bits of both operands participate in the addition. If the carries into and out of the sign position disagree, a fixed-point overflow condition exists. The result has a sign that is the opposite of the true algebraic result. If the fixed-point overflow mask bit (bit 4 in the PCR) is set to one, a fixed-point overflow interrupt occurs.

10.10.6. Compare Word (CR, C)

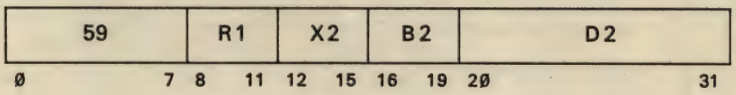
Type: RR

Format: CR



Type: RX

Format: C



Description:

Operand 1 (contents of the general register specified by R1) and operand 2 (contents of location X2/B2/D2 or the general register specified by R2) are compared algebraically.

Condition Code:

Ø Operands are equal
1 Operand 1 is low
2 Operand 1 is high
3 -

Interrupt Priorities:

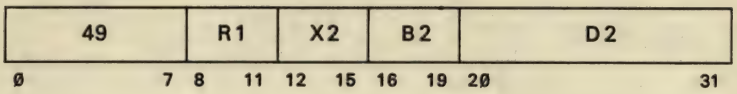
(For C only)

Error	Action
Read access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C
Data address match check	C

10.10.7. Compare Halfword (CH)

Type: RX

Format: CH



Description:

Operand 2 (contents of location X2/B2/D2) is expanded to a fullword (see 10.10.1) and compared algebraically with operand 1 (contents of the general register specified by R1).

Condition Code:

Ø Operands are equal
1 Operand 1 is low
2 Operand 1 is high
3 -

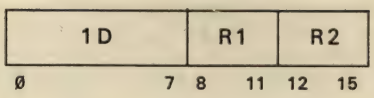
Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a halfword boundary)	S/C
Data address match check	C

10.10.8. Divide (DR, D)

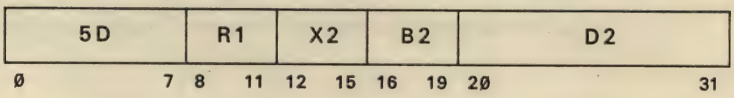
Type: RR

Format: DR



Type: RX

Format: D



Description:

The pair of general registers specified by R1 and R1+1 (R1 even-numbered) contains the 64-bit dividend, which is divided by the divisor (memory word X2/B2/D2 or the contents of the general register specified by R2). The quotient (32 bits) is placed into R1+1 and the remainder (32 bits) into R1. The sign of the quotient is determined by the rules of algebra. The remainder has the same sign as the quotient except that a zero quotient or a zero remainder is always positive.

If the quotient cannot be expressed as a 32-bit twos complement number, a divide error interrupt occurs. The dividend in R1 and R1+1 is not changed in this case.

Condition Code:

Unchanged

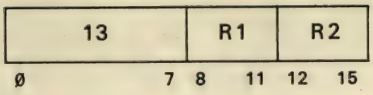
Interrupt Priorities:

Instruction / Error	Action	
	D	DR
Doubleword register error	S	S
Read access	S	-
Physical access	S	-
Fixed-point divide error	S	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C	-
Data address match check	C	-

10.10.9. Load Complement (LCR)

Type: RR

Format: LCR



Description:

The twos complement of operand 2 (contents of the general register specified by R2) is loaded into the general register specified by R1. If operand 2 is the maximum negative number, a fixed-point overflow interrupt occurs. Operand 2 is then loaded unchanged into R1 since the twos complement of the maximum negative number is again the maximum negative number.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

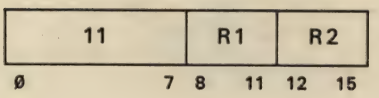
Interrupt Priorities:

Error	Action
Fixed-point overflow	C

10.10.10. Load Negative (LNR)

Type: RR

Format: LNR



Description:

If operand 2 (contents of the general register specified by R2) is negative, it is loaded unchanged into the general register specified by R1; if operand 2 is positive, its twos complement is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 -
3 -

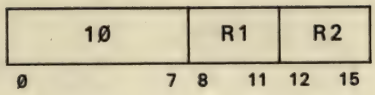
Interrupt Priorities:

None

10.10.11. Load Positive (LPR)

Type: RR

Format: LPR



Description:

If operand 2 (contents of the general register specified by R2) is positive, it is loaded unchanged into the general register specified by R1; if operand 2 is negative, its twos complement is loaded into R1.

If operand 2 is the maximum negative number, a fixed-point overflow interrupt occurs. Operand 2 is then loaded unchanged into R1 since the twos complement of the maximum negative number is again the maximum negative number.

Condition Code:

Ø Result is zero
1 -
2 Result is greater than zero
3 Overflow

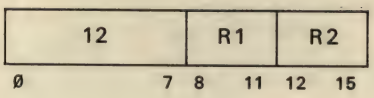
Interrupt Priorities:

Error	Action
Fixed-point overflow	C

10.10.12. Load and Test (LTR)

Type: RR

Format: LTR



Description:

Operand 2 (contents of the general register specified by R2) is loaded into the general register specified by R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

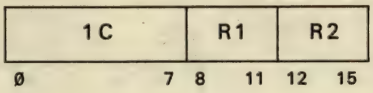
Interrupt Priorities:

None

10.10.13. Multiply Word (MR, M)

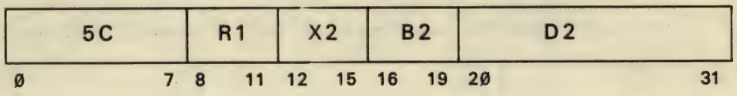
Type: RR

Format: MR



Type: RX

Format: M



Description:

Operand 1 (contents of the general register specified by R1+1 - odd-numbered) is multiplied by operand 2 (memory word X2/B2/D2 or contents of the general register specified by R2). The product is always a 64-bit signed binary number which is loaded into the register pair R1, R1+1. R1 must be an even-numbered register. The sign of the product is determined algebraically. A zero product is always positive.

Condition Code:

Unchanged

Interrupt Priorities:

Error \ Instruction	Action	
	M	MR
Doubleword register error	S	S
Read access	S	-
Physical access	S	-
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C	-
Data address match check	C	-

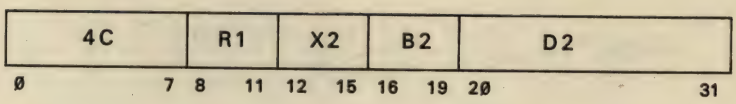
Notes:

Only when two maximum negative numbers are multiplied does the product exceed 62 significant bits This product has 63 significant bits (sign included). Insignificant bit positions are filled with the sign bit.

10.10.14. Multiply Halfword (MH)

Type: RX

Format: MH



Description:

Operand 2 (contents of location X2/B2/D2) is expanded to a fullword and multiplied by the contents of the general register specified by R1. The low-order 32 bits of the product are loaded into R1.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Byte boundary alignment (X2/B2/D2 not a halfword boundary)	S/C
Data address match check	C

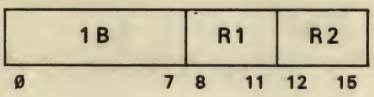
Notes:

If the product contains more than 32 (up to 47) significant bits, the bits to the left of the low-order 32 bits are ignored. The overflow is not indicated.

10.10.15. Subtract Word (SR, S)

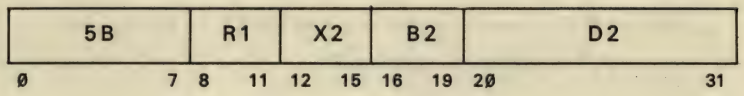
Type: RR

Format: SR



Type: RX

Format: S



Description:

Operand 2 (memory word X2/B2/D2 or contents of the general register specified by R2) is subtracted from operand 1 (contents of the general register specified by R1) by adding operand 1 and the twos complement of operand 2. The result is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Interrupt Priorities:

Error \ Instruction	Action	
	S	SR
Read access	S	-
Physical access	S	-
Fixed-point overflow	C	C
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C	-
Data address match check	C	-

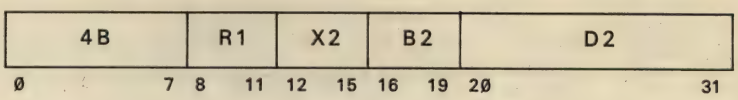
Notes:

If the carries into and out of the sign position disagree, a fixed-point overflow condition exists. The difference has a sign that is the opposite of the true algebraic result. If the fixed-point overflow mask bit (bit 4 in the PCR) is set to one, a fixed-point overflow interrupt occurs.

10.10.16. Subtract Halfword (SH)

Type: RX

Format: SH



Description:

Operand 2 (contents of location X2/B2/D2) is expanded to a fullword and subtracted from operand 1 (contents of the general register specified by R1) by adding operand 1 and the twos complement of operand 2. The result is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Fixed-point overflow	C
Byte boundary alignment (X2/B2/D2 not a halfword boundary)	S/C
Data address match check	C

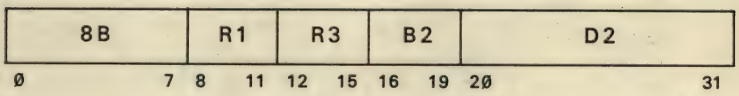
Notes:

If the carries into and out of the sign position disagree, a fixed-point overflow condition exists. The result has a sign that is the opposite of the true algebraic result. If the fixed-point overflow mask bit (bit 4 in PCR) is set to one, a fixed-point overflow interrupt occurs.

10.10.17. Shift Left Single (SLA)

Type: RS

Format: SLA



Description:

The integer part of the operand contained in the general register specified by R1 is shifted left the number of bits specified by the low-order six bits of the address B2/D2. The remaining bits of B2/D2 and the R3 field are ignored. The sign is not changed; bits shifted out to the left are lost and zeros are shifted in from the right to fill the vacated low-order bit positions.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Interrupt Priorities:

Error	Action
Fixed-point overflow	C

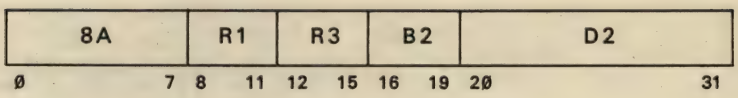
Notes:

If any bit shifted out is different from the sign bit, a fixed point overflow condition exists. If the fixed-point overflow mask bit (bit 4 in the PCR) is set to one, a fixed-point overflow interrupt occurs.

10.10.18. Shift Right Single (SRA)

Type: RS

Format: SRA



Description:

The integer part of the operand contained in the general register specified by R1 is shifted right the number of bits specified by the low-order six bits of the address B2/D2. The remaining bits of B2/D2 and the R3 field are ignored. Bits shifted out to the right are lost; the sign remains unchanged and the sign bit is propagated through the vacated high-order bit positions.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

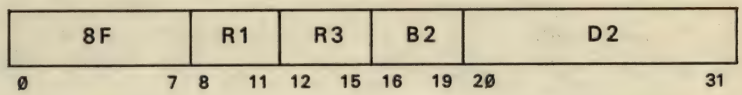
Interrupt Priorities:

None

10.10.19. Shift Left Double (SLDA)

Type: RS

Format: SLDA



Description:

The integer part of the doubleword operand contained in the general registers specified by R1 and R1+1 (R1 even-numbered) is shifted left the number of bits specified by the low-order six bits of the address B2/D2. The remaining bits of B2/D2 and the R3 field are ignored. The sign is not changed; bits shifted out to the left are lost and zeros are shifted in from the right to fill the vacated low-order bit positions.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Interrupt Priorities:

Error	Action
Doubleword register error	S
Fixed-point overflow	C

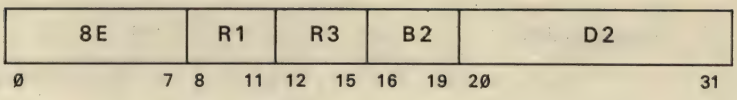
Notes:

If any bit shifted out is different from the sign bit, a fixed-point overflow condition exists. If the fixed-point overflow mask bit (bit 4 in the PCR) is set to one, a fixed-point overflow interrupt occurs.

10.10.20. Shift Right Double (SRDA)

Type: RS

Format: SRDA



Description:

The integer part of the doubleword operand contained in the general registers specified by R1 and R1+1 (R1 even-numbered) is shifted right the number of bits specified by the low-order six bits of the address B2/D2. The remaining bits of B2/D2 and the R3 field are ignored. Bits shifted out to the right are lost; the sign remains unchanged and the sign bit is propagated through the vacated high-order bit positions.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Interrupt Priorities:

Error	Action
Doubleword register error	S

10.11. Decimal Instructions

Decimal arithmetic is performed on data in packed format. In this format two decimal digits are placed in one byte (4 bits each). The operands may be variable in length and must contain a sign in the rightmost halfbyte (4 bits).

All decimal instructions are two-address, SS-type instructions. The instruction set includes addition, subtraction, multiplication, division, and comparison. Since data sent to, and from, external devices is usually in zoned (unpacked) format, i.e. one digit in one byte, there are also instructions for converting to, and from, packed and zoned format.

The binary/decimal conversion instructions, unlike the other instructions in this group, are RX type.

10.11.1. Representation of Numbers

The format for decimal data are:

Packed Format

Byte		Byte		Byte		Byte	
Digit	Digit	Digit	Digit	Digit	Digit	Digit	Sign

In packed format, one byte represents two decimal digits. The rightmost halfbyte of the field represents the sign of the decimal number.

Zoned Format

Byte		Byte		Byte		Byte	
Zone	Digit	Zone	Digit	Zone	Digit	Sign	Digit

In zoned format, the low-order four bits of each byte contain the decimal digit and the high-order four bits contain the zone. The high-order four bits of the rightmost byte of a field contain the sign of the decimal number.

In the case of packed decimal operands, each decimal digit is coded by 4 bits. Each byte thus contains two digits. The operands may be variable in length and contain the sign in the rightmost halfbyte. Digits and signs are coded as follows:

Digit	Code	Sign	Code
Ø	ØØØØ	+	1Ø1Ø
1	ØØØ1	-	1Ø11
2	ØØ1Ø	+	11ØØ
3	ØØ11	-	11Ø1
4	Ø1ØØ	+	111Ø
5	Ø1Ø1	+	1111
6	Ø11Ø		
7	Ø111		
8	1ØØØ		
9	1ØØ1		

The coding of zone portions in the results of decimal operations depends on the ISO bit in the Interrupt Status Register (ISR).
The signs are the same. In the EBCDIC mode (ISO=Ø) the following codes are used:

Sign		Zone
Plus	Minus	
11ØØ	11Ø1	1111

In the ISO mode (ISO=1) the following codes are used:

Sign		Zone
Plus	Minus	
11ØØ	11Ø1	ØØ11

10.11.2. Instruction Execution

Decimal arithmetic instructions - two-address SS-type instructions - operate from right to left. The memory addresses B1/D1, B2/D2 specify the leftmost byte of the respective operands and the length specifies the number of bytes to the right of the addressed byte (L = total number of bytes less one.)

The operand fields can begin at any byte in memory and can vary in length from 1 to 16 bytes. Results are always placed in the operand 1 field and can thus never exceed the length of that field. If a decimal arithmetic operation results in a carry of the operand 1 field, a decimal overflow interrupt occurs. If operand 1 is longer than operand 2, operand 2 is extended with leading zeros up to the length of operand 1 during instruction execution (in addition and subtraction only). This extension never changes the contents of memory.

In the instructions Pack and Unpack, overlapping is permitted without any restrictions since no checking is made for valid coding of digits, zones, and signs.

In all other decimal instructions, checking is made for valid digits and signs. The overlapping of fields is recognized as an invalid digit or an invalid sign unless the rightmost bytes of both fields coincide in memory.

10.11.3. Condition Code

The condition code is not changed on Multiply Decimal, Divide Decimal, Pack, Unpack, Convert to Binary, and Convert to Decimal instructions.

In the other instructions it is set as follows:

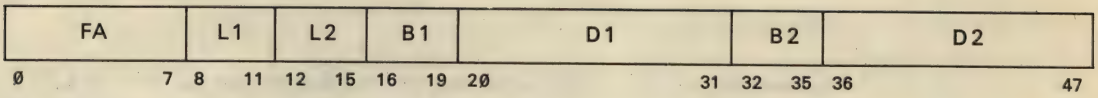
Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow (except Compare Decimal: not used)

10.11.4. Add Decimal (AP)

Type: SS

Format: AP



Description:

The operand specified by the second address B2/D2 is added algebraically to the operand specified by the first address B1/D1, taking into acount the signs.
The result is stored in the field specified by the first address. The condition code is set according to the magnitude and the sign of the result.

The operands must be in packed format and may vary in length from 1 to 16 bytes. If operands overlap, their rightmost byte location must be identical. Therefore, a number can be added to itself.

Decimal overflow may occur when two operands are added.
Either of the following conditions will cause an overflow:

1. A carry out of the high-order digit position of the result.
2. Operand 2 is longer than operand 1 and significant result digits are lost.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Interrupt Priorities:

Error	Action
Read access	S
Decimal format invalid sign	S
Write access	S
Decimal format invalid digit	T
Physical access	T
Decimal overflow	C
Data address match check	C

Notes:

1. Operand 2 is only altered if the operand fields overlap.

2. Processing is from right to left.

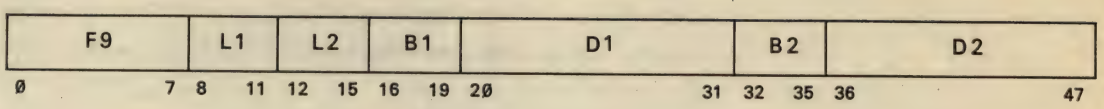
3. A zero result is always positive except where high-order digits are lost because of overflow. If overflow occurs, a zero result has the sign of the correct result.

4. All signs and digits are checked for valid codes.

10.11.5. Compare Decimal (CP)

Type: SS

Format: CP



Description:

The operand specified by the first address B1/D1 is compared algebraically with the operand specified by the second address B2/D2.
The condition code is set according to the result of the comparison.

The operands must be in packed format and may vary in length from 1 to 16 bytes. If the operands are unequal in length, the shorter operand is extended with high-order zeros. If the operands overlap, their rightmost byte location must be identical.

Overflow cannot occur during this instruction.

Condition Code:

Ø Operands are equal
1 Operand 1 is low
2 Operand 1 is high
3 -

Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Decimal format invalid digit	S
Decimal format invalid sign	S
Data address match check	C

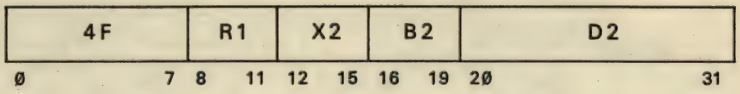
Notes:

1. Both operands are unaltered.
2. Comparison is from right to left.
3. A positive zero compares equally to a negative zero.
4. All signs and digits are checked for valid codes.

10.11.6. Convert to Binary (CVB)

Type: RX

Format: CVB



Description:

The doubleword memory operand specified by the second address X2/B2/D2 is converted from packed decimal to binary format and placed in the general register specified by R1. The number is treated as a right-justified signed integer both before and after conversion.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Physical access	S
Decimal format invalid sign	S
Decimal format invalid digit	S
Byte boundary alignement (X2/B2/D2 not a doubleword boundary)	S/C
Fixed-point divide error	C
Data address match check	C

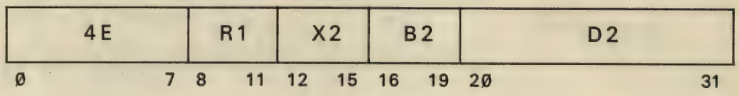
Notes:

1. The doubleword memory operand (15 digits with sign in the low-order 4 bits) is checked for valid sign and digit codes.
2. The doubleword memory operand is not changed.
3. Negative decimal numbers are converted to binary numbers in twos complement notation.
4. A negative decimal zero is converted to a positive binary zero.
5. If the decimal number to be converted is greater than +2, 147, 483, 647 or less than -2, 147, 483, 648, the low-order 32 bits of the result are placed in the R1 and a divide error interrupt occurs.

10.11.7. Convert to Decimal (CVD)

Type: RX

Format: CVD



Description:

The contents of the general register specified by R1 are converted from binary to decimal format and stored at the doubleword location in memory specified by the second address X2/B2/D2. The number is treated as a right-justified signed integer both before and after conversion.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Write access	S
Physical access	T
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	S/C
Data address match check	C

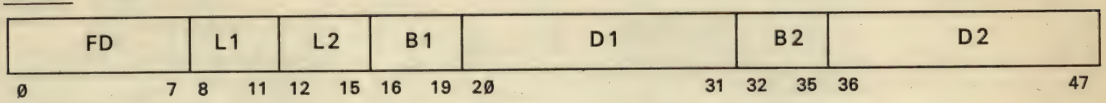
Notes:

1. The result is placed in the doubleword memory location as a packed decimal number consisting of 15 digits and the sign in the low-order 4 bits.
2. The operand specified by the first address is not altered.
3. The decimal equivalents of the maximum binary numbers which can be converted are +2, 147, 483, 647 and -2, 147, 483, 648. Overflow cannot occur.

10.11.8 Divide Decimal (DP)

Type: SS

Format: DP



Description:

The operand (dividend) specified by the first address B1/D1 is divided by the operand (divisor) specified by the second address B2/D2.

The result (quotient and remainder - both signed integers) is placed in the operand 1 field: the quotient on the left, and the remainder with the same length as the divisor, on the right. Quotient and remainder are right-aligned in their field portions.

The operands must be in packed format and may be variable in length. The operands may overlap if their rightmost byte locations coincide. Operand 2 (divisor) must be shorter than operand 1 (dividend) and may not be longer than 8 bytes (15 digits and sign).
No overflow is possible. A quotient that is longer than the permissible number of decimals hints at a decimal divide error. Instruction execution is suppressed.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Decimal format invalid digit	T
Decimal format invalid sign	S
Multiplier/divisor size error	S
Decimal divide error	S
Data address match check	C

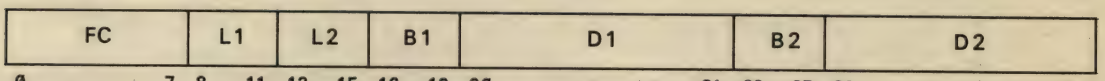
Notes:

1. The leftmost halfbyte of the dividend must be zero.
2. Alldigits and signs are checked for valid codes.
3. Operand 2 is only altered if the operand fields overlap.
4. The sign of the quotient is determined by the rules of algebra. The sign of the remainder is the same as the sign of the dividend.
5. The first address plus (L1-L2) specifies the address of the remainder, L2+1 the length of the remainder.
6. Decimal divide errors can be located by means of a rough subtraction. The left-aligned decimal number of the divisor field is compared to the decimal number of the dividend field decremented by 1. If, with this alignment, the divisor is smaller than or equal to the dividend, a decimal divide error will occur.

10.11.9. Multiply Decimal (MP)

Type: SS

Format: MP



Description:

The operand (multiplicand) specified by the first address B1/D1 is multiplied by the operand (multiplier) specified by the second address B2/D2. The product is placed right-justified in the field specified by the first address. The operands must be in packed format and may be variable in length. The operands may overlap if their rightmost byte locations coincide.

Operand 2 (multiplier) must be shorter than operand 1 (multiplicand) and may not be longer than 8 bytes (15 digits and sign). The multiplicand must have at least as many bytes of high-order zeros as number of bytes in the multiplier. The maximum product size is 31 digits.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Decimal format invalid digit	T
Decimal format invalid sign	S
Decimal multiplicand error	T
Multiplier/divisor size error	S
Data address match check	C

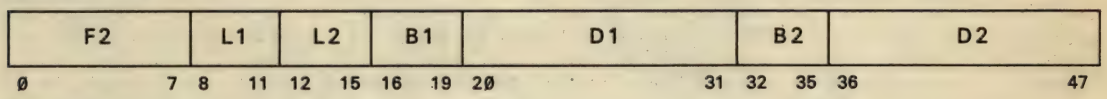
Notes:

1. Operand 2 is only altered if the operand fields overlap.
2. Overflow cannot occur.
3. The sign of the product is determined by the rules of algebra even if one or both operands are zero. This means that a minus zero result is possible.
4. All digits and signs are checked for valid codes.

10.11.10. Pack (PACK)

Type: SS

Format: PACK



Description:

The operand specified by the second address B2/D2 is converted from zoned to packed format. The result is placed in the field specified by the first address B1/D1.

The operand specified by the second address must be in zoned format. The sign is obtained from the zone portion of the rightmost byte of the operand 2 field and is placed in the rightmost halfbyte of the operand 1 field (result). All other zones are ignored. The 4-bit numeric portions (stripping the 4-bit zone) of each byte are then placed adjacent to the sign and to each other to produce the result. Processing is from right to left, one byte at a time.

The result is extended with high-order zeros if the operand 1 field is not filled by the digits transferred from operand 2. If the operand 1 field is not long enough to contain all significant digits form operand 2, the remaining high-order digits are ignored. In this case the operation is executed as if each result byte were stored after the two relevant bytes of operand 2 had been read. The operands may overlap.
Two operand 2 bytes are needed for each result byte except for the rightmost byte of the result field which requires only the rightmost operand 2 byte.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Data address match check	C

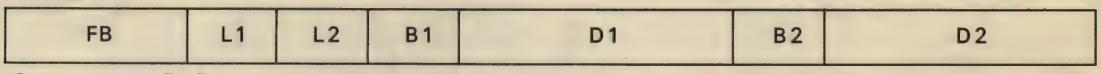
Notes:

1. Signs and digits are not checked for valid codes.
2. Operand 2 is only altered if the operand fields overlap.
3. At the beginning of the operation, the first and second addresses are checked. A paging queue or read access error condition may occur on checking a part of operand 2 that is not actually used during the operation (the result field (B1/D1) is too short to contain all the digits). It is a user programming responsibility to handle this condition.

10.11.11. Subtract Decimal (SP)

Type: SS

Format: SP



Description:

The operand specified by the second address B2/D2 is subtracted algebraically from the operand specified by the first address B1/D1, taking into account the signs. The result is stored in the field specified by the first address. The condition code is set according to the magnitude and the sign of the result.

The operands must be in packed format and may vary in length from 1 to 16 bytes.
If operands overlap, their rightmost byte location must coincide. It is therefore possible to subtract a number from itself.

When two operands are subtracted, decimal overflow may occur. Either of the following conditions will cause an overflow:

1. A carry out of the high-order digit position of the result.
2. Operand 2 is longer than operand 1 and significant result digits are lost.

Condition Code:

Ø Result is zero.
1 Result is less than zero.
2 Result is greater than zero
3 Overflow.

Interrupt Priorities:

Error	Action
Read access	S
Decimal format invalid sign	S
Write access	S
Decimal format invalid digit	T
Physical access	T
Decimal overflow	C
Data address match check	C

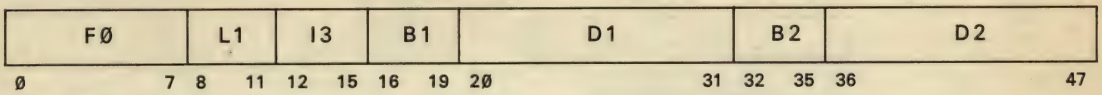
Notes:

1. Operand 1 is only altered if the operand fields overlap.
2. Processing is from right to left.
3. A zero result is always positive except when high-order digits are lost because of overflow. When overflow occurs, a zero result has the sign of the correct difference.
4. All signs and digits are checked for valid codes.

10.11.12. Shift and Round Decimal (SRP)

Type: SS

Format: SRP



Description:

The operand specified by the first address B1/D1 is shifted in the direction and the number of digit positions specified by the second address B2/D2, and, in right shifting, is rounded by the rounding factor given in the I3 field.

Bits 26-31 of the address B2/D2 are used to indicate the direction and the number of digit positions to be shifted. Bits Ø-25 of the address B2/D2 are ignored.

If bit 26 is zero, bits 27-31 represent a binary number whose value determines the number of digit positions to be shifted to the left.

If bit 26 is one, bits 27-31 represent the twos complement of a binary number whose value determines the number of digit positions to be shifted to the right.

Operand 1 is in packed decimal format. The sign and the decimal digits are checked for valid codes. Only the digit portion is shifted; the sign remains in its original position. Zeros are supplied to the vacated digit positions. The condition code is set according to the result of the operation.

The sign of a zero result is made positive.

If a significant digit is shifted out of the high-order digit position during left shift, a decimal overflow occurs.

For right shift, the 4 bits of the immediate operand I3 are used as a rounding factor which is added to the last digit shifted out of the field on the right. Any carry resulting from the decimal addition is propagated to the left. The rounding factor and operand 1 are assumed positive for the purpose of this addition.

Except for validity checking and the participation in rounding digits shifted out of the low-order digit position are ignored and are lost.

The validity of the rounding factor is checked regardless of the direction and the number of positions shifted.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

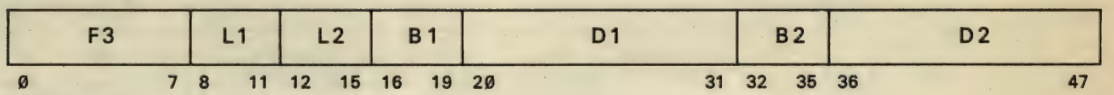
Interrupt Priorities:

Error	Action
Read access	S
Decimal format invalid sign	S
Write access	S
Decimal format invalid digit	T
Physical access	T
Decimal overflow	C
Data address match check	C

10.11.13. Unpack (UNPK)

Type: SS

Format: UNPK



Description:

The operand specified by the second address B2/D2 is converted from packed to zoned format. The result is placed in the field specified by the first address B1/D1.

Each byte (8 bits) of the packed operand 2 field represents two 4-bit digits. Each of these 4-bit digits is placed in the low-order 4 bit positions of a byte in the operand 1 field. If the decimal code is EBCDIC, a zone code of $1111_{(2)}$ is inserted in the high-order 4 bit positions of each byte. If the decimal code is ISO, a zone code of $0011_{(2)}$ is inserted. These zones are inserted in all but the zone portion of the rightmost byte, which receives the sign of the packed operand.

If the operand 1 field is not long enough to contain all significant digits from operand 2, the remaining high-order digits are ignored. If the operand 2 field is too short when unpacked to fill operand 1, operand 2 is extended with high-order zero digits before unpacking.

Operands may overlap. In this case each byte is handled separately as if each result byte were stored after the corresponding operand byte had been read.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Data address match check	C

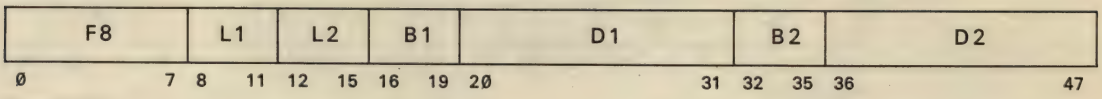
Notes:

1. Signs and digits are not checked for valid coding.
2. Operand 2 is only altered if the operand fields overlap.
3. Processing is from right to left.
4. At the beginning of the operation, the first and second addresses are checked. A paging queue or read access error condition may occur on checking a part of operand 2 that is not actually used during the operation (the result field (B1/D1) is too short to contain all the digits). It is a user programming possibility to handle this condition.

10.11.14. Zero and Add (ZAP)

Type: SS

Format: ZAP



Description:

The operand specified by the second address B2/D2 is loaded into the location specified by the first address B1/D1. The operation is equivalent to an addition to zero. The condition code is set according to the magnitude and sign of the result.

Operand 2 must be in packed format and may vary in length from 1 to 16 bytes. High-order zeros are supplied as required. Operands may overlap if their rightmost byte locations coincide, or if the rightmost byte of operand 1 is to the right of the rightmost byte of operand 2.
Decimal overflow occurs during execution if operand 2 is longer than operand 1 and significant result digits are lost.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Interrupt Priorities:

Error	Action
Read access	S
Decimal format invalid sign	S
Write access	S
Decimal format invalid digit	T
Physical access	T
Decimal overflow	C
Data address match check	C

Notes:

1. Operand 2 is only altered if the operand fields overlap.
2. Only operand 2 is checked for valid sign and digit codes.
3. Processing is from right to left.
4. A zero result is always positive except when high-order digits are lost because of overflow. When overflow occurs, a zero result has the sign of operand 2.

10.12. Floating-Point Instructions

Floating-point instructions provide the capability to process operands of large magnitude with precise results.

A floating-point number consists of three parts : a sign, a characteristic and a mantissa. The sign portion applies to the mantissa. The mantissa is a hexadecimal number with an assumed radix point to the left of the high-order digit. The caracteristic defines the exponent (exponent = characteristic minus 64). The number 16 is the associated base. The quantity that the floating-point number represents is obtained by multiplying the mantissa by the base raised to the power represented by the exponent.

Four floating-point registers are provided for the execution of floating-point operations. These registers are each 64 bits long and are numbered Ø, 2, 4 and 6.

The floating-point instructions are RX and RR type. The R1 and R2 fields contain the floating-point register addresses. Included in the floating-point instruction set are instructions for loading, adding, subtracting, comparing, multiplying, dividing, storing, and controlling signs of all floating-point operands.

Multiplication and division always produce normalized results. Instructions are available for addition and subtraction which can produce normalized and unnormalized results. Operands can be normalized, or unnormalized, in any floating-point operation.

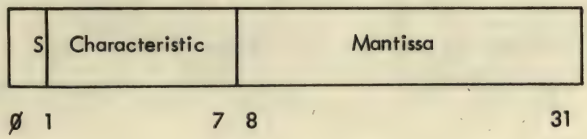
10.12.1. Data Formats

Floating-point numbers are fixed in length and are either fullword (short), doubleword (long), or two doublewords (extended) in format.

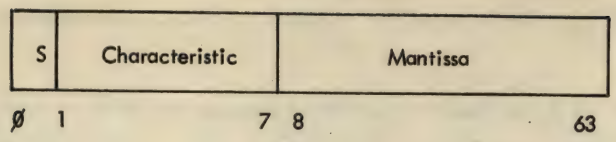
The first bit in all formats is the sign of the mantissa. A 1 bit represents a minus sign and a Ø bit represents a plus sign. The next seven bits represent the characteristic. The mantissa contains 6 hexadecimal digits (short floating-point number) or 14 hexadecimal digits (long floating-point number). The extended form contains 28 hexadecimal digits in two 14-digit mantissa fields. A guard digit (4 bits) is included in the intermediate results to exclude rounding errors for the instructions Add Normalized, Subtract Normalized, Add Unnormalized, Subtract Unnormalized, Compare, Halve and Multiply.

Floating-point instructions with short operands allow faster processing and use less storage. Because floating-point registers are 64 bits long, the low-order 32 bits are ignored when dealing with short operands. When the short format is specified, all operands and the result (except product) are 32 bits long. When using the long format, which provides greater precision, all operands are 63 bits long and require the full register. Extended operands require a pair of registers, either (Ø,2) or (4,6).

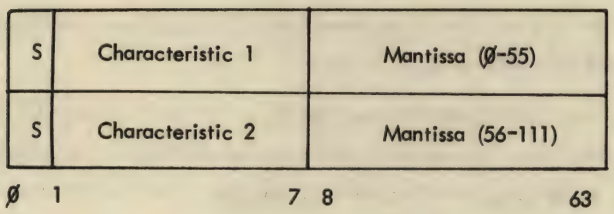
Short:



Long:



Extended:



The extended operands appear as a pair of adjacent floating-point registers (Ø,2) or (4,6) and the following rules apply:

1. The second sign bit is ignored in all operands; the two signs are set alike in all results.
2. The second characteristic is ignored in all operands; the second characteristic is set to 14 less than the first in all results. Underflow in the second characteristic is compensated by adding 128, but otherwise ignored).
3. The mantissas are considered a single, 28-digit mantissa in all extended operations and results.

The alignment of short, long, and extended floating-point numbers in memory and the floating-point registers is as follows:

	Memory	Floating-point register
Short	Word boundary	Ø, 2, 4, 6
Long	Doubleword boundary	Ø, 2, 4, 6
Extended	Not defined in memory	Ø, 4

10.12.2. Representation of Numbers

The mantissa is always represented in hexadecimal and is to be regarded as an absolute value.
An assumed radix points is alwasy immediately to the left of the high-order digit of the mantissa.

The characteristic, bis 1-7, indicates the power to which the base 16 must be raised. The range of the exponent is from -64 to +63, corresponding to the binary range of the characteristic from Ø to 127.

The exponent is equal to the characteristic minus 64.

Characteristic	Decimal equivalent	Exponent
$(1111111)_2$	127	+63
$(1000111)_2$	71	+ 7
$(0000000)_2$	0	-64

The sign of a result from addition, subtraction, multiplication, or division with a zero mantissa is positive. A zero sign, zero characteristic, and zero mantissa in a floating-point number is called true zero.

10.12.3. Normalization

A floating-point number with a mantissa containing a non-zero, high-order, hexadecimal digit is called a normalized number. An unnormalized number has one or more high-order hexadecimal zero digits in the mantissa. To change an unnormalized number into a normalized number, the mantissa is shifted to the left until the high-order digit is non-zero. Then, the characteristic is decremented by the number of digits shifted.

Generally, normalization occurs when the intermediate arithmetic result is changed to the final result. However, in multiplication and division operations, normalization occurs before the arithmetic process.

Floating-point operations are performed with, or without, normalization. Most operations are performed in only one way; however, addition and subtraction may be performed either way as specified.

When normalization is not performed, high-order zeros in the result mantissa are not eliminated. Depending on the original operands, the result may, or may not, be normalized.

Initial operands in both normalized and unnormalized operations need not be in normalized form. Because normalization takes place on hexadecimal digits (4 bits), the three high-order bits of a normalized mantissa can be zero.

10.12.4. Mnemonic Operation Code

To distinguish between

- RR and RX type instructions
- short, long and extended operands
- normalized and unnormalized results

the mnemonic operation code in Assembler observes the following conventions:

- If the last letter of the mnemonic is "R", the instruction is RR type. Otherwise the instruction is RX type.

Further differentiation is provided by the next to last letter for RR instructions, and by the last letter for RX instructions.

o This letter is interpreted as follows:

E - Short operands, normalized result on add or subtract instructions

D - Long operands, normalized result on add or subtract instructions

U - Short operands, unnormalized result

W - Long operands, unnormalized result

X - Extended operands, normalized result

10.12.5. Condition Code

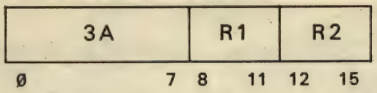
Instructions that cause the condition code to be changed are shown in the following table. All other floating-point instructions leave the condition code unchanged.

Instruction	Operands	Condition Code			
		Ø	1	2	3
Add Normalized	Short/long extended	Zero Zero	< Zero < Zero	> Zero > Zero	- -
Add Unnormalized	Short/long	Zero	< Zero	> Zero	-
Compare	Short/long	Equal	Low	High	-
Load and Test	Short/long	Zero	< Zero	> Zero	-
Load Complement	Short/long	Zero	< Zero	> Zero	-
Load Negative	Short/long	Zero	< Zero		-
Load Positive	Short/long	Zero		> Zero	-
Subtract Normalized	Short/long extended	Zero Zero	< Zero < Zero	> Zero > Zero	- -
Subtract Unnormalized	Short/long	Zero	< Zero	> Zero	-

10.12.6. Add Normalized (AER, ADR, AE, AD)

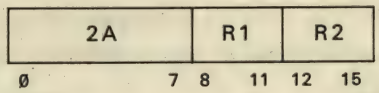
Type: RR (short operands)

Format: AER



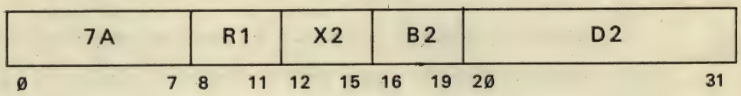
Type: RR (long operands)

Format: ADR



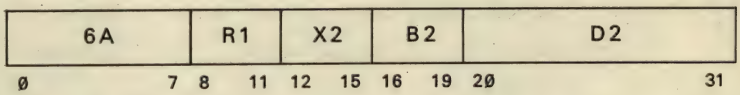
Type: RX (short operands)

Format: AE



Type: RX (long operands)

Format: AD



Description:

Operand 1 (contents of the floating-point register specified by R1) and operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2) are added, and the normalized result is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Interrupt Priorities:

Error \ Instruction	Action			
	AER	ADR	AE	AD
Floating-point register error	S	S	S	S
Read access	-	-	S	S
Physical access	-	-	S	S
Significance error	C	C	C	C
Exponent overflow	C	C	C	C
Exponent underflow	C	C	C	C
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	-	S/C	-
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	-	-	-	S/C
Data address match check	-	-	C	C

Notes:

1. The addition of two floating-point numbers consists in characteristic comparison (if necessary, the characteristics are scaled) and mantissa addition. If the characteristics of the two operands are different, the mantissa with the smaller characteristic is shifted right as many hexadecimal digits as the difference. The hexadecimal digit shifted out last is retained as a guard digit in the addition. The guard digit of a mantissa that is not shifted is zero. The larger characteristic - either if both are equal - becomes the characteristic of the intermediate sum. The intermediate sum is formed by adding the mantissas. If an overflow carry occurs, the intermediate sum is shifted right one hexadecimal digit and its characteristic incremented by 1 (modulo 128). If the characteristic becomes larger than 127 and the intermediate sum is not zero, an exponent overflow interrupt occurs.

2. When short operands are added, the intermediate sum consists of 7 hexadecimal digits and a possible carry; for long operands, it consists of 15 hexadecimal digits and a possible carry. The low-order digit is the guard digit produced during scaling of the characteristics. If the characteristics were equal, the low-order digit of the intermediate sum is zero.

3. If the intermediate sum is zero and the significance error mask bit (bit 7 in the PCR) is set to one, a significance error interrupt occurs. If both the intermediate sum and the significance error mask bit are zero, the result is set to true zero (sign, characteristic, mantissa), the condition code is set to zero, and no significance error interrupt occurs.

4. To form a normalized mantissa, the intermediate sum is shifted left (the vacated low-order digits are made zero) as many hexadecimal digits as it contains high-order hexadecimal zeros, and the characteristic is decremented by an equal number (modulo 128). If the exponent underflows and the exponent underflow mask bit (bit 6 in the PCR) is set to one, an exponent underflow interrupt occurs. In this case, the mantissa is correct and normalized, the sign is correct and the characteristic is 128 larger than the correct value.

   If the exponent underflow mask bit is zero, no exponent underflow interrupt occurs. The result is set to true zero and the condition code is set to zero.

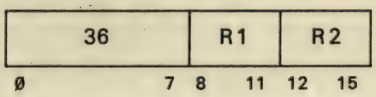
5. The result mantissa is obtained from the normalized intermediate sum by dropping the guard digit.

6. A zero result mantissa is always positive.

10.12.7. Add Normalized (AXR)

Type: RR (extended operands)

Format: AXR



Description:

Operand 1 (contents of the floating-point register pair specified by R1) and operand 2 (contents of the floating-point register pair specified by R2) are added, and the normalized result is loaded into the register pair specified by R1. R1 and R2 must designate the low-numbered register of a pair, i.e. register Ø or 4.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Interrupt Priorities:

Erorr	Action
Floating point register error	S
Register error extended	S
Significance error	C
Exponent overflow	C
Exponent underflow	C

Notes:

1. The addition of two floating-point numbers consists in characteristic comparison (if necessary, the characteristics are scaled) and mantissa addition. If the characteristics of the two operands are different, the mantissa with the smaller characteristic is shifted right as many hexadecimal digits as the difference. The hexadecimal digit shifted out last is retained as a guard digit in the addition. The guard digit of a mantissa that is not shifted is zero. The larger characteristic - either if both are equal - becomes the characteristic of the intermediate sum. The intermediate sum is formed by adding the mantissas. If an overflow carry occurs, the intermediate sum is shifted right one hexadecimal digit and its characteristic incremented by 1 (modulo 128). If the characteristic becomes larger than 127 and the intermediate sum is not zero, an exponent overflow interrupt occurs.

2. The intermediate sum consists of 29 hexadecimal digits and a possible carry. The low-order digit is the guard digit produced during scaling of the characteristics. If the characteristics were equal, the low-order digit of the intermediate sum is zero.

3. If the intermediate sum is zero and the significance error mask bit (bit 7 in the PCR) is set to one, a significance error interrupt occurs. If both the intermediate sum and the significance error mask bit are zero, the result is set to true zero (both signs, both characteristics, both mantissas), the condition code is set to zero, and no significance error interrupt occurs.

4. To form a normalized mantissa, the intermediate sum is shifted left (the vacated low-order digits are made zero) as many hexadecimal digits as it contains high-order hexadecimal zeros, and the characteristic is decremented by an equal number (modulo 128). If the exponent underflows and the exponent underflow mask bit (bit 6 in the PCR) is set to one, an exponent underflow interrupt occurs. In this case, the mantissa is correct and normalized, the sign is correct and the characteristic is 128 larger than the correct value.
   If the exponent underflow mask bit is zero, no exponent underflow interrupt occurs. The result is set to true zero and the condition code is set to zero.

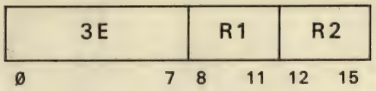
5. The result mantissa (28 hexadecimal digits) is obtained from the normalized intermediate sum by dropping the guard digit.

6. The sign of the result, the characteristic of the normalized intermediate sum and the high-order 14 hexadecimal digits or the result mantissa form the high-order part (doubleword) of the result. The sign of the low-order part of the result is made equal to the sign of the high-order part. The characteristic of the low-order part is 14 less (modulo 128) than that of the high-order part, the low-order 14 hexadecimal digits of the result mantissa are the mantissa portion of the low-order part of the result.

10.12.8. Add Unnormalized (AUR, AWR, AU, AW)

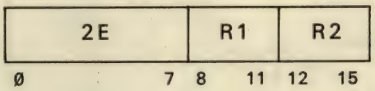
Type: RR (short operands)

Format: AUR



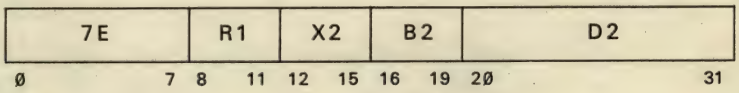
Type: RR (long operands)

Format: AWR



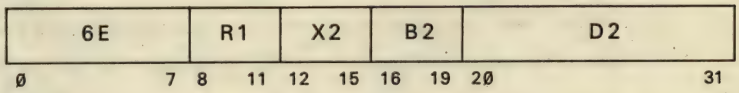
Type: RX (short operands)

Format: AU



Type: RX (long operands)

Format: AW



Description:

Operand 1 (contents of the floating-point register specified by R1) and operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2) are added, and the unnormalized result is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Interrupt Priorities:

Error \ Instruction	Action			
	AUR	AWR	AU	AW
Floating-point register error	S	S	S	S
Read access	-	-	S	S
Physical access	-	-	S	S
Significance error	C	C	C	C
Exponent overflow	C	C	C	C
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	-	S/C	-
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	-	-	-	S/C
Data address match check	-	-	C	C

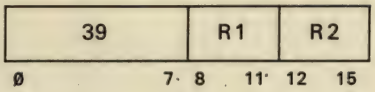
Notes:

1. The addition of two floating-point numbers consists in characteristic comparison (if necessary, the characteristics are scaled) and mantissa addition. If the characteristics of the two operands are different, the mantissa with the smaller characteristic is shifted right as many hexadecimal digits as the difference. The hexadecimal digit shifted out last is retained as a guard digit in the addition. The guard digit of a mantissa that is not shifted is zero. The larger characteristic - either if both are equal - becomes the characteristic of the intermediate sum. The intermediate sum is formed by adding the mantissas. If an overflow carry occurs, the intermediate sum is shifted right one hexadecimal digit and its characteristic incremented by 1 (modulo 128). If the characteristic becomes larger than 127 and the result mantissa is not zero, an exponent overflow interrupt occurs.
2. The result mantissa is obtained from the intermediate sum by dropping the guard digit.
3. If the result mantissa is zero and the significance error mask bit (bit 7 in the PCR) set to is one, a significance error interrupt occurs. If both the result mantissa and the significance error mask bit are zero, the result is set to true zero (sign, characteristic, mantissa) the condition code is set to zero, and no significance error interrupt occurs.
4. A zero result mantissa is always positive.

10.12.9. Compare (CER, CDR, CE, CD)

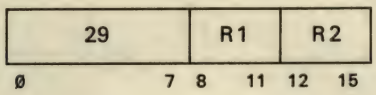
Type: RR (short operands)

Format: CER



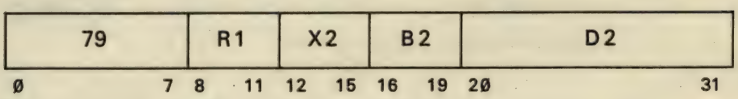
Type: RR (long operands)

Format: CDR



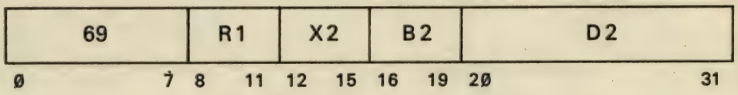
Type: RX (short operands)

Format: CE



Type: RX (long operands)

Format: CD



Description:

Operand 1 (contents of the floating-point register specified by R1) is compared with operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2) by subtracting operand 2 from operand 1 as in the Subtract Normalized instruction. The normalized result, which is not stored, determines the condition code.

Condition Code:

Ø Operands are equal
1 Operand 1 is low
2 Operand 1 is high
3 -

Interrupt Priorities:

Error \ Instruction	Action			
	CER	CDR	CE	CD
Floating-point register error	S	S	S	S
Read access	-	-	S	S
Physical access	-	-	S	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	-	S/C	-
Byte boudary alignment (X2/B2/D2 not a doubleword boundary)	-	-	-	S/C
Data address match check	-	-	C	C

Notes:

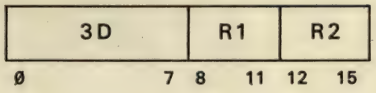
1. Scaling of the characteristics and normalizing of the intermediate result are performed as in the Subtract Normalized instruction.

2. Exponent overflow and underflow are ignored. If the result mantissa is zero, the condition code is set to zero without interrupt.

10.12.10. Divide (DER, DDR, DE, DD)

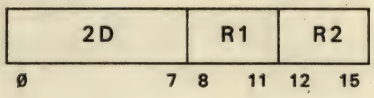
Type: RR (short operands)

Format: DER



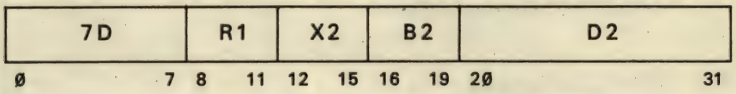
Type: RR (long operands)

Format: DDR



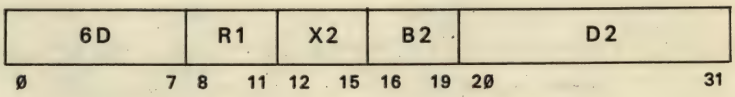
Type: RX (short operands)

Format: DE



Type: RX (long operands)

Format: DD



Description:

Operand 1 (contents of the floating-point register specified by R1 = dividend) is divided by operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2 = divisor), and the quotient is loaded into R1.

Condition Code:

Unchanged

Interrupt Priorities:

Error \ Instruction	Action			
	DER	DDR	DE	DD
Floating-point register error	S	S	S	S
Read access	-	-	S	S
Physical access	-	-	S	S
Floating-point divide error	S	S	S	S
Exponent overflow	C	C	C	C
Exponent underflow	C	C	C	C
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	-	S/C	-
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	-	-	-	S/C
Data address match check	-	-	C	C

Notes:

1. Floating-point division consists in characteristic subtraction and mantissa division. The difference between the dividend and divisor characteristics plus 64 is used as the characteristic of the intermediate quotient. The sign of the quotient is determined by the rules of algebra.

   The operand mantissas are normalized, the intermediate quotient characteristic being reduced by the number of high-order hexadecimal zeros of the dividend and increased by the number of high-order hexadecimal zeros of the divisor. The normalized mantissas are then divided to form the mantissa of the quotient. If the mantissa of the dividend was greater than or equal to that of the divisor, the quotient is shifted right one hexadecimal digit and the intermediate quotient characteristic incremented by one to form the final quotient characteristic.

2. If the final quotient characteristic exceeds 127, an exponent overflow interrupt occurs. If the exponent underflows and the exponent underflow mask bit (bit 6 in the PCR) is set to one, an exponent underflow interrupt occurs. In this case, the mantissa is correct and normalized, the sign is correct and the characteristic is 128 larger than the correct value.

   If the exponent underflow mask bit is zero, no exponent underflow interrupt occurs. The final quotient is set to true zero and the condition code is set to zero.

3. If the divisor mantissa is zero, a divide error interrupt occurs.

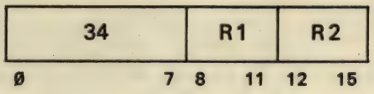
4. If the dividend mantissa is zero, the final quotient is set to true zero (sign, characteristic, mantissa).

5. Any remainder is ignored.

0.12.11. Halve (HER, HDR)

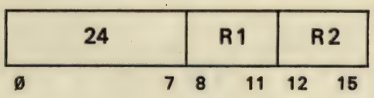
Type: RR (short operands)

Format: HER



Type: RR (long operands)

Format: HDR



Description:

Operand 2 (contents of the floating-point register specified by R2) is divided in half, i.e. its mantissa is shifted right one bit, placing the contents of the low-order bit position into the high-order bit position of the guard digit and introducing a zero into the high-order bit position of the mantissa. The normalized result is loaded into the floating-point register specified by R1.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Floating-point register error	S
Exponent underflow	C

Notes:

1. If the exponent underflows and the exponent underflow mask bit (bit 6 in the PCR) is set to one, an exponent underflow interrupt occurs. In this case, the mantissa is correct and normalized, the sign is correct and the characteristic is 128 larger than the correct value.

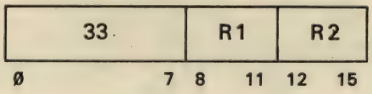
   If the exponent underflow mask bit is zero, no exponent underflow interrupt occurs. The result is set to true zero.

2. If the operand 2 mantissa is zero, the result is set to true zero (sign, characteristic, mantissa). No significance error is recognized.

10.12.12. Load Complement (LCER, LCDR)

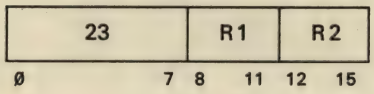
Type: RR (short operands)

Format: LCER



Type: RR (long operands)

Format: LCDR



Description:

Operand 2 (contents of the floating-point register specified by R2) is loaded into the floating-point register specified by R1 and the sign is changed to the opposite value. A negative zero result is possible.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

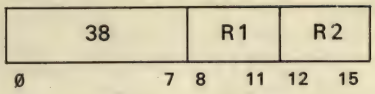
Interrupt Priorities:

Error	Action
Floating-point register error	S

10.12.13. Load (LER, LDR, LE, LD)

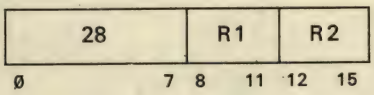
Type: RR (short operands)

Format: LER



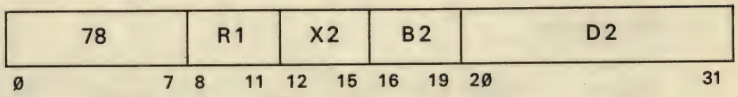
Type: RR (long operands)

Format: LDR



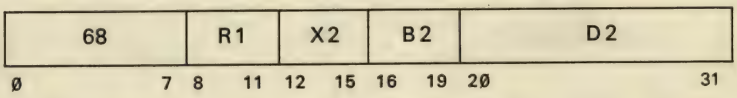
Type: RX (short operands)

Format: LE



Type: RX (long operands)

Format: LD



Description:

Operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2) is loaded into the floating-point register specified by R1.

Condition Code:

Unchanged

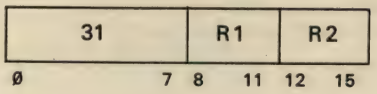
Interrupt Priorities:

Error \ Instruction	Action			
	LER	LDR	LE	LD
Floating-point register error	S	S	S	S
Read access	-	-	S	S
Physical access	-	-	S	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	-	S/C	-
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	-	-	-	S/C
Data address match check	-	-	C	C

10.12.14. Load Negative (LNER, LNDR)

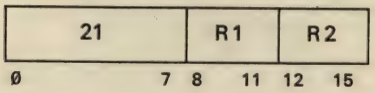
Type: RR (short operands)

Format: LNER



Type: RR (long operands)

Format: LNDR



Description:

Operand 2 (contents of the floating-point register specified by R2) is loaded into the floating-point register specified by R1 and the sign bit is set to one (minus). A negative zero result is possible.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 -
3 -

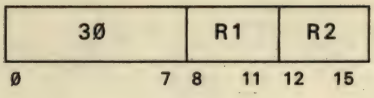
Interrupt Priorities:

Error	Action
Floating-point register error	S

10.12.15. Load Positive (LPER, LPDR)

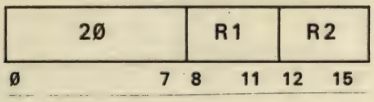
Type: RR (short operands)

Format: LPER



Type: RR (long operands)

Format: LPDR



Description:

Operand 2 (contents of the floating-point register specified by R2) is loaded into the floating-point register specified by R1 and the sign bit is set to zero (plus).

Condition Code:

Ø Result is zero
1 -
2 Result is greater than zero
3 -

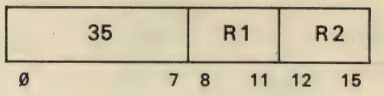
Interrupt Priorities:

Error	Action
Floating-point register error	S

10.12.16. Load Rounded (LRER, LRDR)

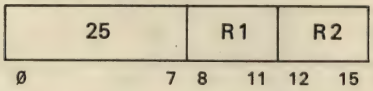
Type: RR (long operand 2, short operand 1)

Format: LRER



Type: RR (extended operand 2, long operand 1)

Format: LRDR



Description:

Operand 2 (contents of the floating-point register specified by R2) is rounded to the next smaller format, and the result is loaded into the floating-point register specified by R1.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Floating-point register error	S
Register error extended (LRDR)	S
Exponent overflow	C

Notes:

1. Rounding to the short format consists in adding a one to the contents of bit position 32 of operand 2 and propagating the carry, if any, to the left. For rounding to the long format, the contents of bit position 8 of the low-order part of operand 2 are added to the contents of bit position 63 of the high-order part of operand 2.

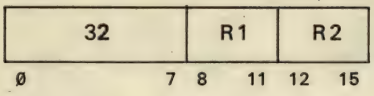
2. If rounding causes a carry out of the high-order digit position of the mantissa, the mantissa is shifted right one digit position and the characteristic is incremented by one.

3. The result is not normalized.

10.12.17. Load and Test (LTER, LTDR)

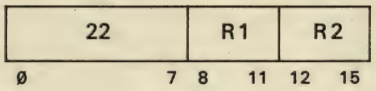
Type: RR (short operands)

Format: LTER



Type: RR (long operands)

Format: LTDER



Description:

Operand 2 (contents of the floating-point register specified by R2) is loaded into the floating-point register specified by R1. The sign and magnitude of the loaded operand determine the condition code.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

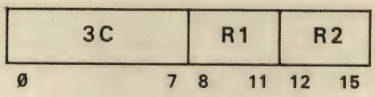
Interrupt Priorities:

Error	Action
Floating-point register error	S

10.12.18. Multiply (MER, MDR, ME, MD)

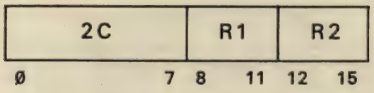
Type: RR (short multiplier and multiplicand, long product)

Format: MER



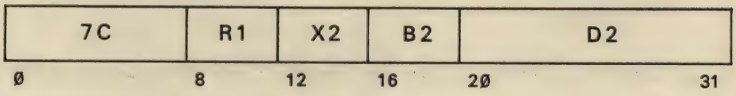
Type: RR (long operands)

Format: MDR



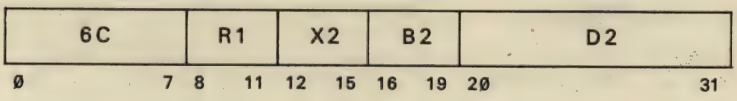
Type: RX (short multiplier and multiplicand, long product)

Format: ME



Type: RX (long operands)

Format: MD



Description:

Operand 1 (contents of the floating-point register specified by R1 = multiplicand) is multiplied by operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2 = multiplier), and the normalized product is loaded into R1.

Condition Code:

Unchanged

Interrupt Priorities:

Error \ Instruction	Action			
	MER	MDR	ME	MD
Floating-point register error	S	S	S	S
Read access	-	-	S	S
Physical access	-	-	S	S
Exponent underflow	C	C	C	C
Exponent overflow	C	C	C	C
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	-	S/C	-
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	-	-	-	S/C
Data address match check	-	-	C	C

Notes:

1. Floating-point multiplication consists in characteristic addition and mantissa multiplication. The sum of the characteristics less 64 is used as the characteristic of the intermediate product. The sign of the product is determined by the rules of algebra. The operand mantissas are normalized, the characteristic being reduced by the number of high-order hexadecimal zeros of the two mantissas. The normalized mantissas are then multiplied to form the mantissa of the intermediate product. The intermediate product (15 hexadecimal digits for long operands) is then normalized.

2. For both short and long operands, the product mantissa consists of 14 hexadecimal digits.

3. If the final product characteristic exceeds 127, an exponent overflow interrupt occurs. If the exponent underflows and the exponent underflow mask bit (bit 6 in the PCR) is set to one, an exponent underflow interrupt occurs. In this case, the mantissa is correct and normalized, the sign is correct and the characteristic is 128 larger than the correct value.

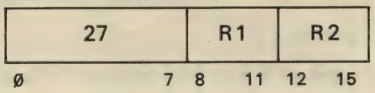
   If the exponent underflow mask bit is zero, no exponent underflow interrupt occurs. The final product is set to true zero.

4. If the product mantissa is zero, the final product is set to true zero (sign, characteristic, mantissa). Any exponent underflow or overflow occurring is ignored.

10.12.19. Multiply (MXDR, MXD)

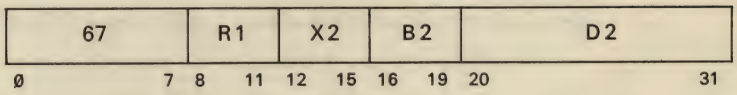
Type: RR (long multiplier and multiplicand, extended product)

Format: MXDR



Type: RX (long multiplier and multiplicand, extended product)

Format: MXD



Description:

Operand 1 (contents of the floating-point register specified by R1 = long multiplicand) is multiplied by operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2 = long multiplier), and the normalized extended product is loaded into the register pair specified by R1. R1 must designate the low-numbered register of the pair, i.e. register Ø or 4.

Condition Code:

Unchanged

Interrupt Priorities:

Error \ Instruction	Action	
	MXDR	MXD
Floating-point register error	S	S
Register error extended	S	S
Read access	-	S
Physical access	-	S
Exponent overflow	C	C
Exponent underflow	C	C
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	-	C
Data address match check	-	C

Notes:

1. Floating-point multiplication consists in characteristic addition and mantissa multiplication. The sum of the characteristics less 64 is used as the characteristic of the intermediate product. The intermediate product obtained by multiplying the two operand mantissas consists of 28 hexadecimal digits and is normalized to form the final product.

2. The sign of the product is determined by the rules of algebra.

3. If all the digits of the product mantissa are zero, the final product is set to true zero (sign, characteristic, mantissa).

4. If the final product characteristic exceeds 127, an exponent overflow interrupt occurs. If the exponent underflows and the exponent underflow mask bit (bit 6 in the PCR) is set to one, an exponent underflow interrupt occurs. In this case, the mantissa is correct and normalized, the sign is correct and the characteristic is 128 larger than the correct value.

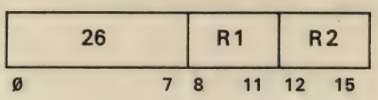
   If the exponent underflow mask bit is zero, no exponent underflow interrupt occurs. The final product is set to true zero.

5. The sign and characteristic of the product and the high-order 14 hexadecimal digits of the product mantissa form the high-order part (doubleword) of the final product. The sign of the low-order part of the product is the same as the sign of the high-order part. The characteristic of the low-order part is 14 less (modulo 128) than that of the high-order part. The low-order 14 hexadecimal digits of the product mantissa are the mantissa portion of the low-order part of the product.

10.12.20. Multiply (MXR)

Type: RR (extended operands)

Format: MXR



Description:

Operand 1 (contents of the floating-point register pair specified by R1 = extended multiplicand) is multiplied by operand 2 (contents of the floating-point register pair specified by R2 = extended multiplier), and the extended normalized product is loaded into R1. R1 and R2 must designate the low-numbered register of a pair, i.e. register Ø or 4.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Floating-point register error	S
Register error extended	S
Exponent overflow	C
Exponent underflow	C

Notes:

1. Floating-point multiplication consists in characteristic addition and mantissa multiplication. The sum of the characteristic less 64 is used as the characteristic of the intermediate product. To form the intermediate product, the operands are normalized and their mantissas multiplied. The intermediate product mantissa consists of 29 hexadecimal digits. If the high-order digit position of the intermediate product mantissa is not zero, its high-order 28 hexadecimal digits form the final product mantissa. If the high-order digit position of the intermediate product mantissa is zero, the mantissa is shifted left one digit position, and the intermediate product characteristic is reduced by one. The high-order 28 hexadecimal digits of the normalized intermediate product form the final product mantissa.
2. The sign of the product is determined by the rules of algebra.
3. If all the digits of the product mantissa are zero, the final product is set to true zero (sign, characteristic, mantissa).
4. If the final product characteristic exceeds 127, an exponent overflow interrupt occurs. If the exponent underflows and the exponent underflow mask bit (bit 6 in the PCR) is set to one, an exponent underflow interrupt occurs. In this case, the mantissa is correct and normalized, the sign is correct and the characteristic is 128 larger than the correct value.

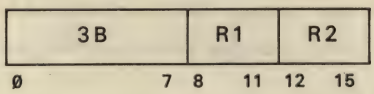
If the exponent underflow mask bit is zero, no exponent underflow interrupt occurs. The final product is set to true zero.

5. The sign ana characteristic of the product and the high-order 14 hexadecimal digits of the product mantissa form the high-order part (doubleword) of the final product. The sign of the low-order part of the product is the same as the sign of the high-order part. The characteristic of the low-order part is 14 less (modulo 128) than that of the high-order part. The low-order 14 hexadecimal digits of the product mantissa are the mantissa portion of the low-order part (doubleword) of the product.

10.12.21. Subtract Normalized (SER, SDR, SE, SD)

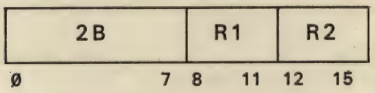
Type: RR (short operands)

Format: SER



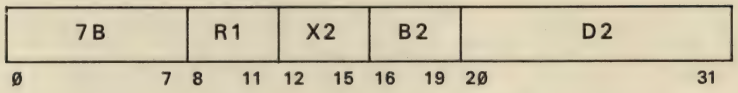
Type: RR (long operands)

Format: SDR



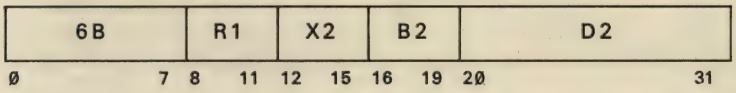
Type: RX (short operands)

Format: SE



Type: RX (long operands)

Format: SD



Description:

Operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2) is subtracted from operand 1 (contents of the floating-point register specified by R1), and the normalized result is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Interrupt Priorities:

Error \ Instruction	Action			
	SER	SDR	SE	SD
Floating-point register error	S	S	S	S
Read access	-	-	S	S
Physical access	-	-	S	S
Significance error	C	C	C	C
Exponent overflow	C	C	C	C
Exponent underflow	C	C	C	C
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	-	S/C	-
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	-	-	-	S/C
Data address match check	-	-	C	C

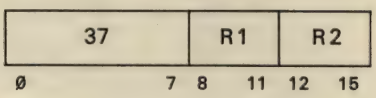
Notes:

Operand 2 is subtracted from operand 1 by inverting the sign of operand 2 and adding the operands as in Add Normalized.

10.12.22. Subtract Normalized (SXR)

Type: RR (extended operands)

Format: SXR



Description:

Operand 2 (contents of the floating-point register pair specified by R2) is subtracted from operand 1 (contents of the floating-point register pair specified by R1), and the normalized result is loaded into the register pair specified by R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Interrupt Priorities:

Error	Action
Floating-point register error	S
Register error extended	S
Significance error	C
Exponent overflow	C
Exponent underflow	C

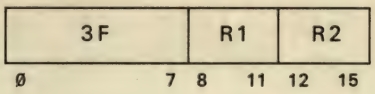
Notes:

Operand 2 is subtracted from operand 1 by inverting the sign of operand 2 and adding the operands as in Add Normalized.

10.12.23. Subtract Unnormalized (SUR, SWR, SU, SW)

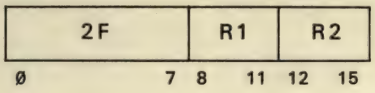
Type: RR (short operands)

Format: SUR



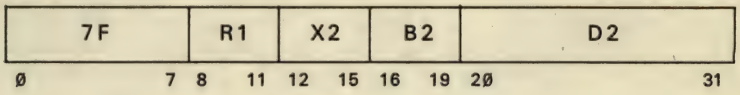
Type: RR (long operands)

Format: SWR



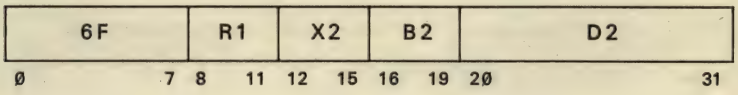
Type: RX (short operands)

Format: SU



Type: RX (long operands)

Format: SW



Description:

Operand 2 (contents of the floating-point register specified by R2 or location X2/B2/D2) is subtracted from operand 1 (contents of the floating-point register specified by R1) and the unnormalized result is loaded into R1.

Condition Code:

Ø Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Interrupt Priorities:

Instruction / Error	Action			
	SUR	SWR	SU	SW
Floating-point register error	S	S	S	S
Read access	-	-	S	S
Physical access	-	-	S	S
Significance error	C	C	C	C
Exponent overflow	C	C	C	C
Byte boundary alignment (X2/B2/D2 not a word boundary)	-	-	S/C	-
Byte bondary alignment (X2/B2/D2 not a doubleword boundary)	-	-	-	S/C
Data address match check	-	-	C	C

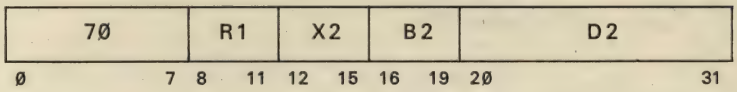
Notes:

Operand 2 is subtracted form operand 1 by inverting the sign of operand 2 and adding the operands as in Add Normalized.

10.12.24 Store (STE, STD)

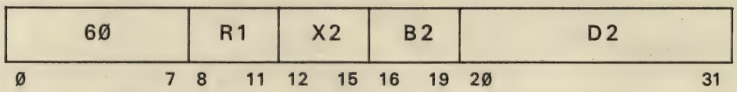
Type: RX (short operands)

Format: STE



Type: RX (long operands)

Format: STD



Description:

Operand 1 (contents of the floating-point register specified by R1) is stored in the operand 2 memory area specified by X2/B2/D2.

Condition Code:

Unchanged

Interrupt Priorities:

Error \ Instruction	Action	
	STE	STD
Floating-point register error	S	S
Write access	S	S
Physical access	S	S
Byte boundary alignment (X2/B2/D2 not a word boundary)	S/C	-
Byte boundary alignment (X2/B2/D2 not a doubleword boundary)	-	S/C
Data address match check	C	C

10.13. Stack Instructions

The stack is a memory area where each time a new entry is made, the previously entered information is pushed down a level and each time a removal is made, it is popped up a level. Thus, the information entered last is the first which can be accessed again.

The stack instructions are used to handle two different kinds of stacks:

- Control stack and data stack
- Extensible stacks

The system comprising the control stack and the data stack is handled by the Execute Stack (EXST) instruction, which provides the following special functions:

- Call by Location (CALC)
- Call by Number (CALN)
- Store Multiple in Stack (STMS)
- Move Stack Address (MSAR)
- Return (RET)

The extensible stack system is handled by the following instructions:

- Pop (POP)
- Push (PUSH)

10.13.1. Control and Data Stacks

The system comprising the control and data stacks occupies four main memory areas per processor state:

Control stack

Data stack

Control stack header

CALN vector table

The control stack and the control stack header occupy adjacent areas in main memory (see Fig. 10-4).

The Stack Address Register (SAR), which is available for each processor state separately, contains an effective address pointing to the control stack header. The control stack header contains the effective addresses of entries in the control stack, data stack and CALN vector table.

The control and data stack system supports handling of subroutines and reentrant programs.

The control stack is provided to save the Program Counter Register (PCR) and the Interrupt Status Register (ISR) of the program calling the subroutine. The data stack serves to save the general registers used by the calling program, and serves as a work area for the called program.

The CALN vector table permits indirect addressing of the called subroutines.

10.13.1.1. Data Stack

The data stack is a continous area in main memory Information is entered from higher to lower addresses and removed in the reverse direction. The data stack bounds are specified by the data stack lower bound address (DSLBA) and the data stack upper bound address (DSUBA). The data stack address (DSA) points to the last stack entry.

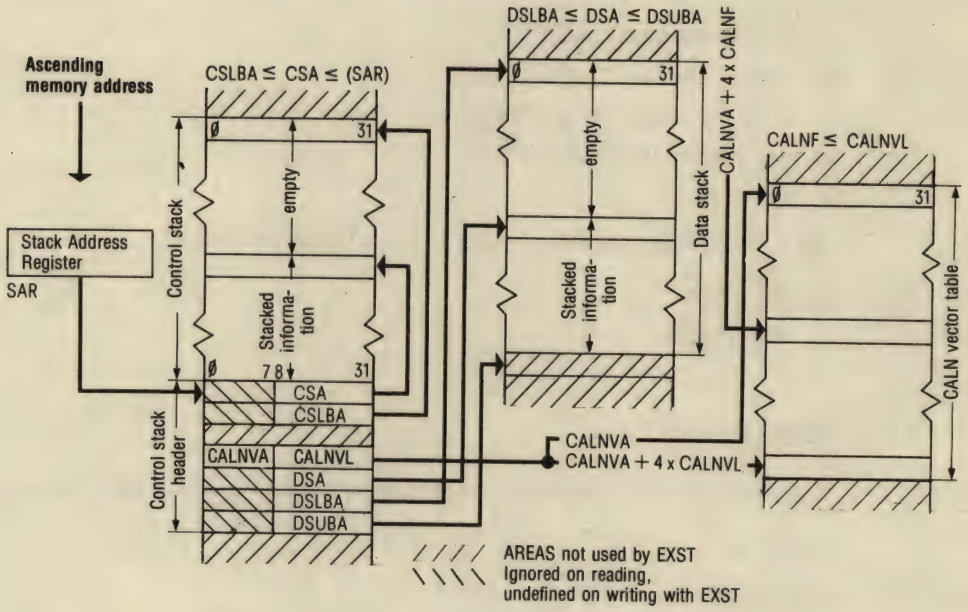


Fig. 10-4 Control and Data Stack System

CSA	Current control stack address
CSLBA	Control stack lower bound address
CALNVL	CALN vector table length
CALNVA	CALN vector table address
DSA	Current data stack address
DSLBA	Data stack lower bound address
DSUBA	Data stack upper bound address
CALNF	Branch index

DSLBA, DSUBA and DSA are maintained in the control stack header. DSA must be word-oriented and must lie between the data stack bounds (DSLBA $\leq$ DSA $\leq$ DSUBA).
These conditions are checked by STMS, RET and MSAR. DSLBA and DSUBA may be any byte addresses. (In Fig. 10-4, DSLBA and DSUBA are assumed to be word-oriented).

The Store Multiple in Stack (STMS) special function stores the contents of a set of general registers in the data stack beginning at DSA minus four. The register addresses are processed in ascending order and the stack addresses in descending order. The control stack is loaded with the register address of the last general register stored, the number of general registers stored, and the address of the last entry in the data stack.

The inverse operation of STMS is performed by the Return (RET) special function. RET removes the relevant general registers from the data stack. The register addresses are processed in descending order and the stack addresses in ascending order.

Any data may be entered in the data stack, e.g. by means of the Move (MVC) instruction. The Move Stack Address (MSAR) special function is provided to perform the necessary modification of DSA.

10.13.1.2. Control Stack

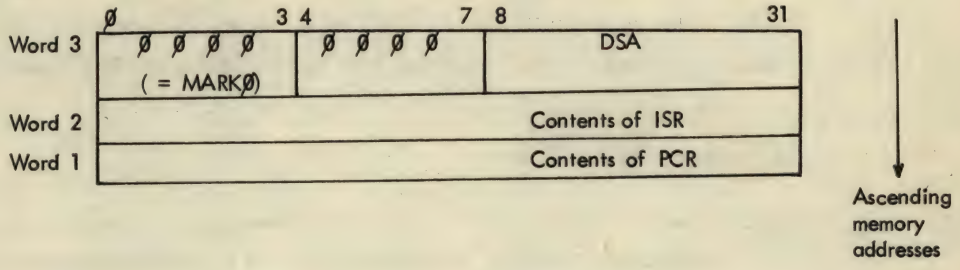
Like the data stack, the control stack is a continuous area in main memory. Information is entered from higher to lower addresses and removed in the reverse direction. The lower control stack bound is given by the control stack lower bound address (CSLBA). The upper bound is determined by the contents of the Stack Address Register (SAR). The control stack address (CSA) always points to the last stack entry.

CSLBA and CSA are maintained in the control stack header. CSA must be word-oriented and must lie between the control stack bounds. (CSLBA $\leq$ CSA $\leq$ contents of SAR).
These conditions are checked by CALC, CALN, MSAR, STMS and RET.
CSLBA may be any byte address.

The control stack permits two types of entries:

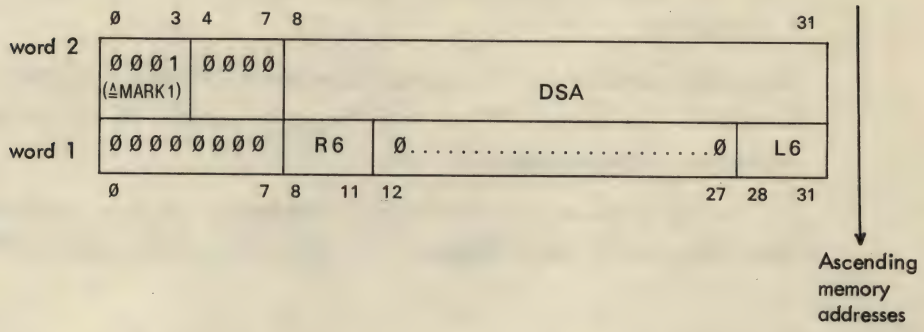
- CALC entries (by the CALC and CALN special functions)
- STMS entries (by the STMS special functions).

A CALC entry comprises 3 words and has the following format:



The Call by Location (CALC) and Call by Number (CALN) special functions generate the CALC entries. Beginning at CSA minus four, these special functions store the current PCR in word 1, the current ISR in word 2 and the current DSA in word 3. Bits Ø-3 of word 3 are designated MARKØ and identify the entry as a CALC entry. After the CALC entry has been made, CSA is decremented and points to word 3 of the entry. Control is then transferred to the subroutine called.

An STMS entry comprises 2 words and has the following format :



The Store Multiple in Stack (STMS) special function stores the contents of a set of general registers in the data stack and generates the STMS entry. Beginning at CSA minus four, STMS stores in word 1 the address of the last general register stored (R6) and the number of general registers stored minus one (L6). In word 2, STMS stores DSA after execution of the special function; this is the address of the data stack entry containing the last general register stored (R6). Bits Ø-3 of word 2 are designated MARK1 and identify the entry as an STMS entry. After the STMS entry has been made, CSA is decremented and points to word 2 of the entry.

The Return (RET) special function accesses the control stack via the current CSA.

If RET finds an STMS entry in the control stack, the general registers specified in word 1 of the entry are loaded from the data stack in accordance with DSA in word 2. CSA and DSA are then incremented and point to the next entries. The next entry in the control stack is processed. If this is an STMS entry, it is processed as just described.

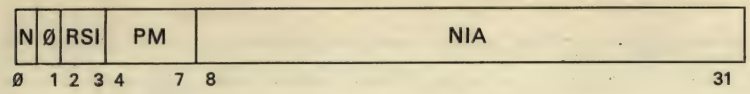
If RET finds a CALC entry in the control stack, the ISR and PCR are loaded from the control stack, DSA is transferred to the control stack header, CSA is incremented to point to the next entry, and RET terminates.

The special functions CALC, CALN , STMS and RET access the control stack with a protection key of Ø and a ring number of Ø, irrespective of the current ISR key and Ring State Indicator (RSI) value.

10.13.1.3. CALN Vector Table

The CALN vector table is a continuous area in main memory beginning at the word-oriented CALN vector table address (CALNVA). The length of the table is specified by the 8-bit CALN vector table length CALNVL). CALNVL specifies the number of words minus one, CALNVL = Ø thus defines a CALN vector table containing one word. CALNVA and CALNVL are maintained in the control stack header. The one-word entries in the CALN vector table are used by the Call by Number (CALN) special function to define a subset of the ISR and the start address of the called subroutine.

The entries have the following format:

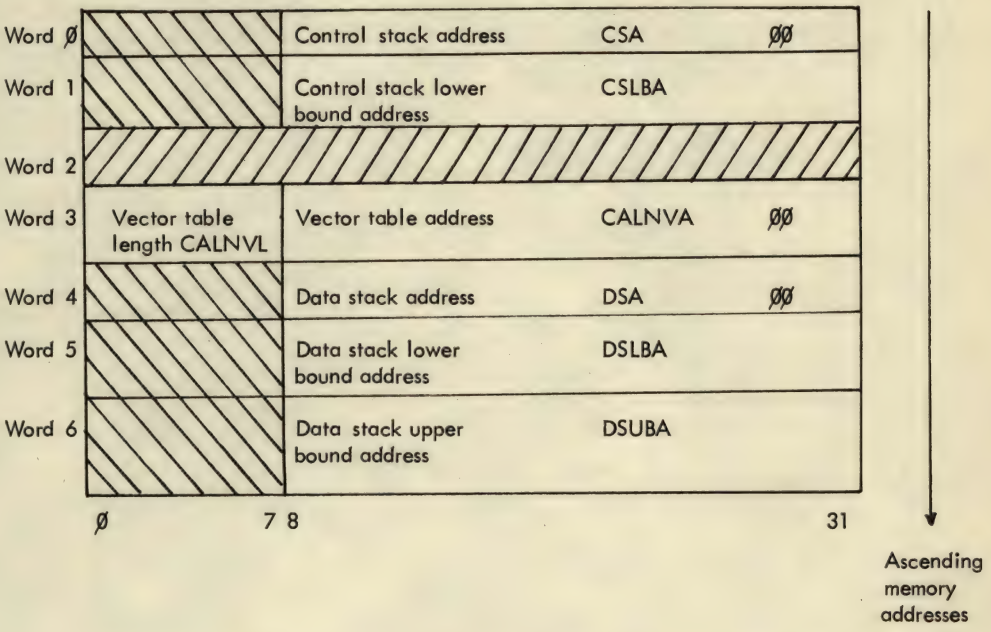


Bit	loaded into
Ø	N bit of ISR
1	ignored by hardware, but must be zero
2-3	RSI bits of ISR
4-7	PM of PCR
8-31	NIA of PCR

The CALN special function accesses the CALN vector table with a protection key of Ø and a ring number of Ø, irrespective of the current ISR key and RSI value.

10.13.1.4. Control Stack Header

The control stack header comprises a 7-word main memory area adjacent to the upper bound of the control stack. It has the following format:



DSA, DSLBA, DSUBA, CSA and CSLBA are described in 10.13.1.1 and 10.13.1.2. The high-order byte of these words is ignored on reading, and undefined on storing. CALNVL and CALNVA are described in 10.13.1.3. Word 2 is not used.

All special functions handling the data and control stack system access the control stack header with a protection key of Ø and a ring number of Ø, irrespective of the current ISR key and RSI value.

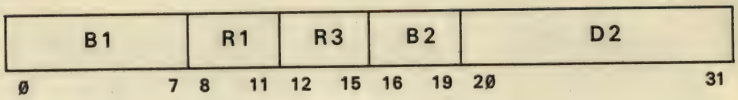
10.13.1.5. Stack address Register SAR

The Stack Address Register is a CPU register, available separately for each processor state. Bits Ø-7 are not used, bits 8-31 contain the effective 8-word-oriented stack address which points to word Ø of the associated control stack header. In addition, this stack address determines the upper bound of the control stack.

10.13.2. Execute Stack (EXST)

Type: RS

Format: EXST



Description:

This instruction handles the data and control stack system. The R3 field of the instruction defines the special function to be performed as follows:

R3 field (Hex)	Special function	Reference
Ø	Call by Location (CALC)	10.13.2.1
1	Call by Number (CALN)	10.13.2.2
2	Store multiple in Stack (STMS)	10.13.2.4
3	Return (RET)	10.13.2.5
4-15	Move Stack Address (MSAR)	10.13.2.3

Use of the R1, B2 and D2 fields depends on the special function defined by the R3 field.

Condition Code:

This also depends on the special function defined by the R3 field.

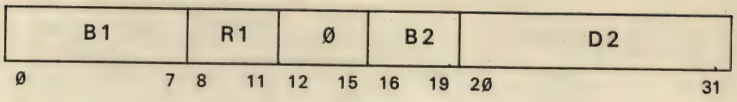
Interrupt Priorities:

See description of each special function.

10.13.2.1. Call by Location (CALC)

Type: RS

Format: CALC



Description:

This special function generates a CALC entry in the control stack, updates CSA in the control stack header, and performs a direct branch.

Detailed Operation:

The SAR contains the address of the control stack header, word Ø. Words Ø (CSA), 1 (CSLBA) and 4 (DSA) are read from the control stack header.

CSA points to word 3 of the last CALC entry (word 2 of the last STMS entry). CSA is decremented by 4 and the new CALC entry (words 1, 2 and 3) is generated in the control stack. Upon instruction end, CSA in the control stack header points to word 3 of the CALC entry just made.

If the R1 field is not zero, DSA is also read from the control stack header and loaded into the general register specified by R1.

Finally the PCR is loaded with the branch address B2/D2. The instruction length code (ILC) is set to zero. If the audit mode is on, the address of the EXST instruction is entered in the audit table.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Stack error	S
Read access	S
Write access	S
Physical access	S
Data address match check	C
Physical access (to audit table)	C
Audit wrap	C

Notes:

1. The control stack and the control stack header are accessed as if the protection key and ring number in the ISR were zero.

2. CALC checks the following:

   a) 8-word alignment of SAR

   b) Word alignment of CSA

   c) CSA within stack bounds

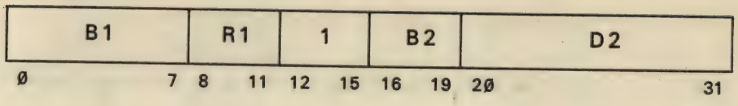
   Violation of a), b) or c) generates a stack error.

3. The data address match check is made on accessing the control stack.

10.13.2.2. Call by Number (CALN)

Type: RS

Format: CALN



Description:

This special function generates a CALC entry, updates CSA in the control stack header and performs an indirect branch via the CALN vector table.

Detailed Operation:

The SAR contains the address of control stack header, word Ø.
Words Ø (CSA), 1 (CSLBA), 3 (CALNVL and CALNVA) and 4 (DSA) are read from the control stack header.

CSA points to word 3 of the last CALC entry (word 2 of the last STMS entry). CSA is decremented by 4 and the new CALC entry (words 1, 2 and 3) is generated in the control stack. Upon instruction end, CSA in the control stack header points to word 3 of the CALC entry just made.

If the R1 field is not zero, DSA is also read from the control stack header and loaded into the general register specified by R1.

The branch address for this instruction is contained in an entry in the CALN vector table. This entry is addressed by the vector table address CALNVA (word 3 in the control stack header) and the branch index CALNF (bits 24-31 of the address B2/D2). The address of the entry is calculated as CALNVA+4 xCALNF. CALNF must not be greater than the vector table length CALNVL. The entry is loaded into N, RSI, PM and NIA of the ISR and also into the PCR. The instruction length code (ILC) is set to zero. If the audit mode is on, the address of the EXST instruction is entered in the audit table.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Stack error	S
CALN address error	S
Read access	S
Write access	S
Physical access	S
Data address match check	C
Physical access (to audit table)	C
Audit wrap	C

Notes:

1. All memory accesses are executed as if the protection key and ring number in the ISR were zero.

2. CALN checks the following:

   a) 8-word alignment of SAR
   b) Word alignment of CSA
   c) Word alignment of CALNVA
   d) CSA within stack bounds
   e) CALNF not greater than CALNVL

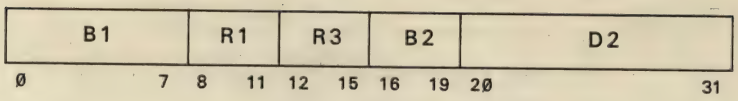
   Violation of a) through d) generates a stack error; violation of e) generates a CALN address error.

3. The data address match check is made on accessing the control stack and the CALN vector table.

10.13.2.3. Move Stack Address (MSAR)

Type: RS

Format: MSAR



Description:

This special function serves to load a word from the control stack into the general register specified by R1, or to modify the DSA within the data stack bounds and load it modified or unmodified into R1. The R3 field determines which individual functions are performed. It must contain a hexadecimal number greater than, or equal to, 4. The following codes apply:

R3 field (bits 12-15)		Operation
Ø1XX		The control stack word addressed by CSA + M[1] is loaded into bits 8-31 of R1. CSA and the control stack header remain unchanged.
> 1ØØØ		These codes define two groups of functions, A and B. One function from group A must always be combined with one function from group B.
1ØXX	Group A	M[1] is added to DSA in the control stack header.
11XX	Group A	M[1] is subtracted from DSA in the control stack header.
1XØX	Group B	R1 remains unchanged.
1X1Ø	Group B	DSA is loaded unmodified into bits 8-31 of R1.
1X11	Group B	DSA, modified by a group A function, is loaded into bits 8-31 of R1.

1) M is the smallest number divisible by 4 and greater than or equal to, the address specified by B2/D2. Bits marked "X" are don't-care bits.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Stack error	S
Read access	S
Write access	S
Physical access	S

Notes:

1. The control stack header is accessed as if the protection key and ring number in the ISR were zero.

2. MSAR checks the following:

   a) 8-word alignment of SAR

   b) If bit 12 in R3 field = 1 : Word alignment of DSA

   c) If bit 12 in R3 field = 1 : DSA ± M within stack bounds (DS ∠ BA ≦ DSA ± M ≦ DSUBA)

   d) If bit 12 in R3 field = Ø : Word alignment of CSA

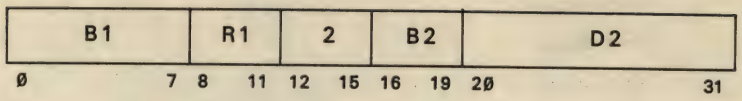
   e) If bit 12 in R3 field = Ø : CSA + M within stack bounds (CS ∠ BA ≦ CSA + M ∠ SAR)

   Violation of a) through e) generates a stack error.

10.13.2.4. Store Multiple in Stack (STMS)

Type: RS

Format: STMS



Description:

This special function stores a set of general registers in the data stack and generates an STMS entry in the control stack.
Return (RET) is the complementary special function.

Detailed Operation:

The R1 field specifies the first general register to be stored in the data stack. Bits 28-31 of the address defined by B2/D2 specify the number of general registers to be stored minus one (L6). A zero value of L6 signifies that one general register is to be stored. Bits 8-27 of B2/D2 are ignored by the hardware, but must be zero.

The first general register is stored in data stack location DSA minus 4. The remaining registers up to the total of L6+1, in order of ascending register addresses (with 15 to Ø wraparound), are stored consecutively in adjacent data stack locations in descending order of main memory addresses.
Upon instruction end, DSA in the control stack header points to the entry in the data stack which contains the last register stored. An STMS entry is generated in the control stack. Finally, CSA in the control stack header is decremented by 8, so as to point to word 2 of the STMS entry.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Stack error	S
Read access	S
Write access	S
Physical access	T
Data address match check	C

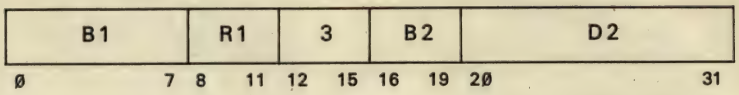
Notes:

1. The control stack and the control stack header are accessed as if the protection key and ring number in the ISR were zero.
2. STMS checks the following:
   a) 8-word alignment of SAR
   b) Word alignment of CSA
   c) Word alignment of DSA
   d) CSA and DSA within stack bounds.
   Violation of a) through d) generates a stack error.
3. The data address match check is made on accessing the data stack and the control stack.

10.13.2.5. Return (RET)

Type: RS

Format: RET



Description:

This special function loads a set of general registers, the ISR and the PCR from the control and data stacks.

Detailed Operation:

The SAR contains the address of word Ø in the control stack header. The word is read; it contains CSA which points to word 3 of the last CALC entry (word 2 of the last STMS entry). The entry is fetched. If this is an STMS entry, identified by MARK1, RET loads the general registers defined by fields R6 and L6 of the control stack entry from the data stack. Reading from the data stack commences at the word addressed by DSA of the STMS entry and proceeds in ascending order of data stack addresses. R6 in STMS word 1 defines the first general register to be loaded. Loading proceeds in order of descending register addresses (with Ø to 15 wraparound). A total of L6+1 registers are loaded. A zero value of L6 signifies that one register is to be loaded. When L6+1 registers have been loaded, CSA is switched to word 2 or word 3 of the preceding entry. If this is an STMS entry with MARK1, processing is as described above. If it is a CALC entry, identified by MARKØ, DSA of the entry is placed in word 4 of the control stack header. Special function execution is then controlled by the R1 field as follows:

R1 field (bits 8-11)	Operation
ØØCC	ISR is loaded from word 2, PCR from word 1, of the CALC entry. The condition code is set as specified by bits 1Ø and 11 of the R1 field.
Ø1XX	ISR and PCR (including the condition code) are loaded from words 2 and 1 of the CALC entry.
1ØXX	ISR and PCR are loaded from words 2 and 1 of the CALC entry. The condition code is unchanged.
11XX	ISR and PCR (including the condition code) remain unchanged.

In all cases, CSA is incremented by 12 and points to word 2 or word 3 of the preceding entry. The address D2/B2 is ignored. After processing of a CALC entry, RET terminates.

If a branch occurs and the audit mode is on, the address of the EXST instruction is entered in the audit table.

Condition Code:

The R1 field determines setting of the condition code as specified by the above table.

Interrupt Priorities:

Error	Action
Stack error	S
Read access	S
Write access	S
Physical access	S
Data address match check	C
Physical access (to audit table)	C
Audit wrap	C

Notes:

1. The control stack, control stack header and CALN vector table are accessed as if the protection key and ring number in the ISR were zero.
2. RET checks the following:
   a) 8-word alignment of SAR
   b) Word alignment of CSA
   c) CSA within stack bounds
   d) Word alignment of DSA in STMS entry
   e) DSA in STMS entry within stack bounds
3. The data address match check is made on accessing the control stack and the data stack.

10.13.3. Extensible Stacks

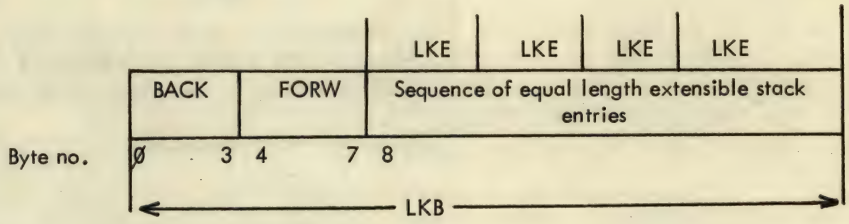
The extensible stacks permit efficient dynamic memory allocation and deallocation within a program. The memory space concerned need not be a continuous area. An extensible stack comprises an optional number of extents of equal length, and the extensible stack word.

An extent consists of an 8-byte extent header and a sequence of equal length extensible stack entries. The extensible stack word contains the length specification for the extensible stack entries (W), the start address of the currently active extensible stack entry (POINT), the length specification for the extensible stack extents (ELIM), and the length specification for the used area of an extent (USED), i.e. the total length of the entries made during a PUSH operation, and of the entries of an extent remaining to be processed during a POP operation.

The extents are linked forward and backward via the extent header and overflow/underflow from one to the next is automatically.

Two instructions, PUSH and POP, are provided for handling the extensible stack. They handle address management, putting the address of the next entry (POINT) into the general register specified by R1. PUSH increments POINT, and POP decrements POINT. The instructions do not perform any data transfers into or out of the stack. PUSH is used when an item is to be added to the stack, and POP when an item is to be deleted. Note that with PUSH, the updated length specification for the extent is always placed in the header of the next extent.

Extents have the following format:

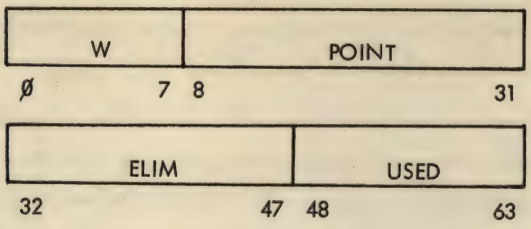


Byte No.	Field	Description
Ø-3	BACK	Bytes 1-3 contain the start address of the preceding extent. Byte Ø must normally be zero. In the first extent of an extensible stack, the entire BACK field (bytes Ø-3) is filled with ones.
4-7	FORW	Bytes 5-7 contain the start address of the subsequent extent. Byte 4 must normally be zero. In the last extent, the entire FORW field (bytes 4-7) is filled with ones.
8 up		Sequence of equal length extensible stack entries (LKE).

An extent always begins at a word boundary and so BACK and FORW must be word-oriented.

The extensible stack word is a doubleword in main memory which must be doubleword-oriented.

It has the following format:

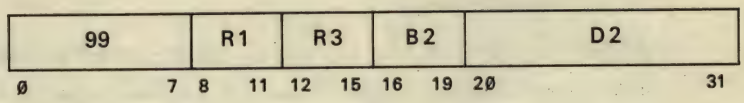


Bit no.	Field	Description
ø-7	W	Length minus one of each entry in the stack. Each entry has the length LKE = (W+1) bytes.
8-31	POINT	Start address of the currently active entry (length W+1).
32-47	ELIM	Length of extents. ELIM = LKB-LKE-8. ELIM must be an integer multiple of (W+1), otherwise ELIM is adapted to the next lower integer multiple of (W+1) by means of USED when, during the execution of PUSH, the next extent is entered.
48-63	USED	Length of the used area of an extent. If only the first entry is active, USED is zero. If all entries are active, then USED = ELIM. USED must always be an integer multiple of (W+1). This is not checked by hardware.

10.13.3.1. Push (PUSH)

Type: RS

Format: PUSH



Description:

The main memory address B2/D2 of the instruction must be doubleword-oriented; it is used to address the extensible stack word. The extensible stack word contains the length W of the extensible stack entries (bits Ø-7), the pointer address POINT up to which the extensible stack is open (bits 8-31), the

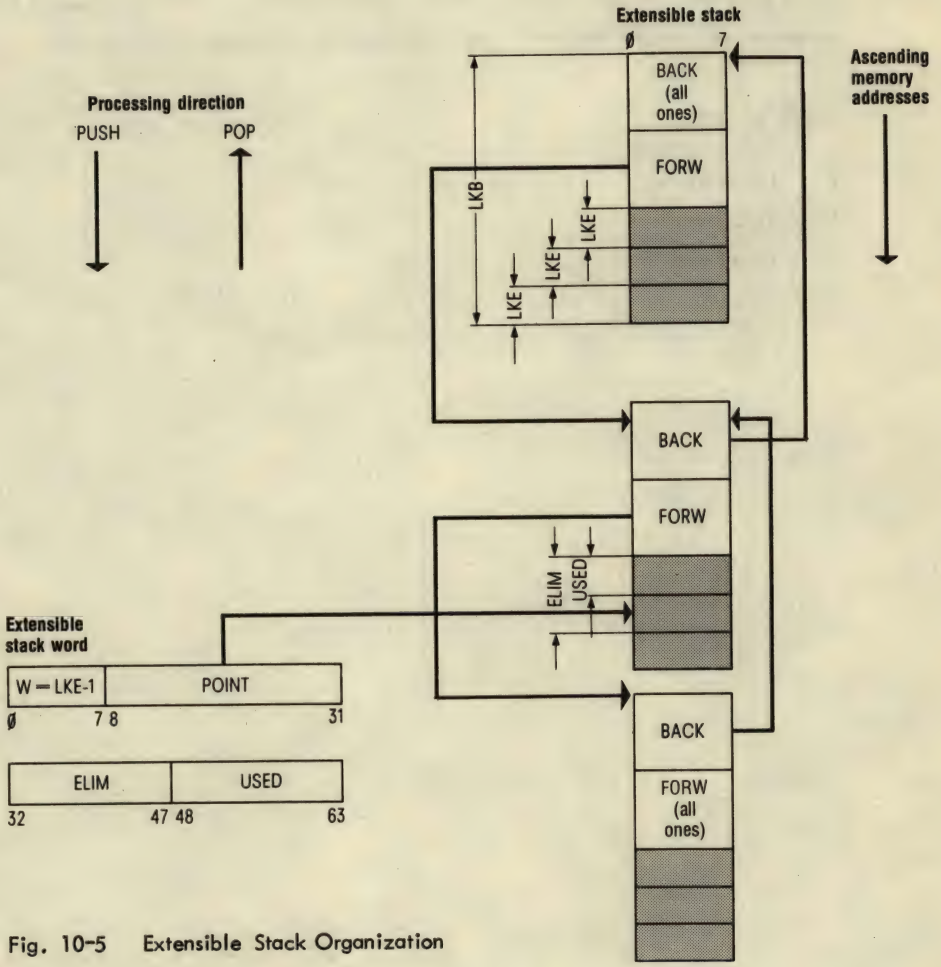


Fig. 10-5 Extensible Stack Organization

BACK	Start address of preceding extent
FORW	Start address of subsequent extent
LKB	Extensible stack extent
LKE	Extensible stack entry
W	Length of entries
POINT	Start address of currently active entry
ELIM	Length of extents
USED	Length of used area of extent

length ELIM of the extents (bits 32-47) and the length USED of the used area of the extent (bits 48-63).

If USED+W+1 $\leq$ ELIM, USED and POINT are incremented by (W+1) in the extensible stack word. The new value of POINT is loaded into the general register specified by R1 (bits Ø-7 are not used) and the condition code is set to Ø.

If USED +W+1 $>$ ELIM, the word address POINT-USED-4 is formed which points to the entry FORW in the extent header. If FORW contains all ones, the address of FORW, i.e. POINT-USED-4, is loaded into R1 and the condition code is set to 1.

If FORW does not contain all ones, the word address FORW is used to read the entry BACK in the extent header of the subsequent extent. This must contain the address of BACK in the extent header of the processed extent, i.e. POINT-USED-8. In this case, in the extensible stack word POINT is set to the new value FORW+8, ELIM is made equal to USED, and USED is set to zero. R1 is loaded with the new value of POINT and the condition code is set to 2.

If the entry BACK in the header of the new extent does not contain the address of BACK in the old extent header, a stack link error is generated and the instruction is not executed.

Condition Code:

Ø	No overflow
1	Stack overflow
2	Extent overflow
3	-

Interrupt Priorities:

Error	Action
Read access	S
Stack link error	S
Write access	S
Physical access	S
Word alignment (POINT not word-oriented on entering another extent)	C
Doubleword alignment (B2/D2 not doubleword-oriented)	C
Data address match check	C

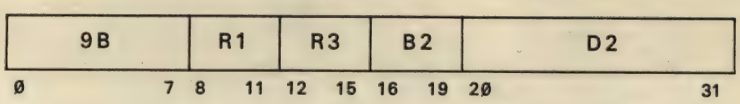
Note:

The high-order byte of FORW/BACK is only examined during the all-ones check. The R3 field of the instruction is not used.

10.13.3.2. Pop (POP)

Type: RS

Format: POP



Description:

The main memory address B2/D2 of the instruction must be doubleword-oriented; it is used to address the extensible stack word. The extensible stack word contains the length W of the extensible stack entries (bits Ø-7), the pointer address POINT up to which the extensible stack is open (bits 8-31), the length ELIM of the extents (bits 32-47) and the length USED of the used area of the extent (bits 48-63).

If USED-W-1 ≥ Ø, USED and POINT are decremented by (W+1) in the extensible stack word. The new value of POINT is loaded into the general register specified by R1 (bits Ø-7 are not used) and the condition code is set to Ø.

If USED-W-1 < Ø, the word address POINT-USED-8 is formed which points to the entry BACK in the extent header. If BACK contains all ones, POINT is loaded into R1 and the condition code is set to 1.

If BACK does not contain all ones, the word address BACK+4 is used to read the entry FORW in the extent header of the preceding extent. This must contain the address of BACK in the extent header of the processed extent, i.e. POINT-USED-8. In this case, in the extensible stack word POINT is set to the new value BACK+8+ELIM and USED is made equal to ELIM. R1 is loaded with the new value of POINT (bits Ø-7 are not used) and the condition code is set to 2.

If the entry FORW in the header of the new extent does not contain the address of BACK in the old extent header, a stack link error is generated and the instruction is not executed.

Condition Code:

Ø	No underflow
1	Stack underflow
2	Extent underflow
3	-

Interrupt Priorities:

Error	Action
Read access	S
Stack link error	S
Write access	S
Physical access	S
Word alignment (POINT not word-oriented on entering another extent)	C
Doubleword alignment (B2/D2 not doubleword-oriented)	C
Data address match check	C

Note:

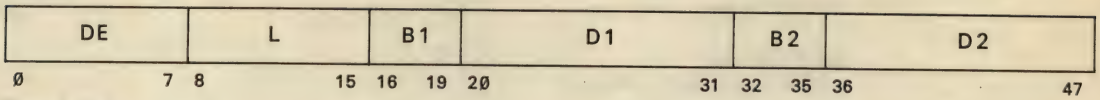
The high-order byte of FORW/BACK is only examined during the all-ones check. The R3 field of the instruction is not used.

10.14. Edit Instructions

10.14.1. Edit (ED)

Type: SS

Format: ED



Description:

The Edit instruction edits data fields into a form ready for printing. The variable-length source field specified by the second address B2/D2, containing packed decimal data, is changed to zoned format with editing under the control of a mask pattern. The edited result replaces the mask pattern specified by the first address B1/D1.

Editing includes sign control, punctuation control, suppression and protection of leading zeros, and also facilitates programmed blanking for all-zero fields. Several fields may be edited in one operation and text can be combined with numeric information.

The mask bytes (B1/D1), referred to as pattern characters, control insertion of the zoned decimal digits into the pattern field. The L field defines the length of the pattern.

The source field (B2/D2) has the packed format and contains source bytes, each halfbyte of which is termed a source digit. The high-order 4 bits (leftmost halfbyte) of a source byte must specify a decimal digit code, i.e. ØØØØ-1ØØ1. The sign codes 1Ø1Ø-1111 are illegal in this context and cause a program interrupt. The low-order 4 bits (rightmost halfbyte) may specify either a sign or a decimal digit.

The operands are processed from left to right: the pattern field, byte by byte; the source field, with a few exceptions, halfbyte by halfbyte. If the halfbytes contain decimal digits, they are changed to zoned format and, in accordance with the editing rules, placed in the pattern field or replaced by fill characters. If the leftmost halfbyte contains a sign code, a program interrupt occurs and the instruction terminates. If the rightmost halfbyte contains a sign code, it is not placed in the pattern field. The next character is fetched from the pattern to control processing of the next halfbyte from the source field.

Overlapping of the pattern and source fields yields unpredictable results.

The following codes apply for decimal digits, source digits and signs in the pattern and source fields:

Codes	Definition
ØØØØ ⟶ 1ØØ1	Digits
1Ø1Ø, 11ØØ, 111Ø, 1111	Plus sign
1Ø11, 11Ø1	Minus sign
1111	Zone/EBCDIC
ØØ11	Zone/ISO

During the editing process, each character of the pattern is affected in one of three ways:

1. It is left unchanged (i.e. it is a message character),
2. It is replaced by a source digit expanded to zoned format,
3. It is replaced by the first character in the pattern, called the fill character.

Which of the three actions takes place is determined by one or more of the following: the type of the pattern character, the state of the significance indicator, and whether the source digit examined, if valid, is zero or nonzero.

The pattern characters allow all codes. Four types are distinguished according to control function as follows:

Pattern character name	Hexadecimal code	
	EBCDIC	ISO
Digit selector	2Ø	8Ø
Significance starter	21	81
Field separator	22	82
Message character	any other	any other

The first character of the pattern is used as the fill character, irrespective of its type. As the fill character, it can have any code.

The detection of either a digit selector or a significance starter in the pattern causes an examination to be made of the significance indicator and of a source digit. As a result, either the expanded source digit or the fill character, as appropriate, is selected to replace the pattern character. Additionally, the significance starter is used to set the significance indicator to the on state.

The field separator identifies indvidual fields in a multiple-field editing operation. It is always replaced by the fill character and sets the significance indicator to the off state.

Message characters in the pattern either remain unchanged or are replaced by the fill character, depending on the state of the significance indicator.

The significance indicator controls zero suppression (when in the on state, zero suppression does not take place). The significance indicator is set to the off state at the start of the editing operation and can be set to the on or off state during processing of the pattern. It is set to the on state when the source digit is a nonzero decimal digit, or when the significance starter is encountered in the pattern, and if in both instances no reset condition exists.

The following table summarizes the significance indicator function:

Pattern character	Conditions		Result
	Source digit	Low-order source digit is a plus sign	State of significance indicator after digit examination
Digit selector	1-9	No	On
	Ø-9	Yes	Off
Significance starter	Ø-9	No	On
	Ø-9	Yes	Off
Field separator	*	*	Off

* Not applicable

In all other cases the significance indicator remains unchanged.

Message Character

These characters cannot be replaced by expanded source field digits. They are inserted between the source digits. In the case of zero suppression, they can be replaced by fill characters. The following table applies:

	Condition	Result character
Pattern Character	Significance indicator	
Message character	On	Message character

Source Digits

The source field digits are converted to zoned format depending on their value, the state of the significance indicator and the pattern character, and replace the digit selector and significance starter characters in the pattern. When the significance indicator is off, leading zero source digits are replaced by the fill character. Source digits with a value between 1 and 9 are placed in the pattern in the form of zoned decimal digits.

The following table applies:

Pattern character	Conditions		Result character
	Significance indicator	Source digit	
Digit selector	Off	1-9	Source digit
	On	Ø-9	Source digit
Significance starter	Off	1-9	Source digit
	On	Ø-9	Source digit

The first character of the pattern is used as the fill character. It applies in the case of leading zeros and, when the significance indicator is off, replaces leading zeros and any interspersed message characters in the result.
The following table applies:

Pattern character	Conditions		Result character
	Significance indicator	Source digit	
Digit selector	Off	Ø	Fill character
Significance starter	Off	Ø	Fill character
Field separator	*	*	Fill character
Message character	Off	*	Fill character

* Not applicable.

The following table summarizes the control functions of all the pattern characters:

Pattern character	Conditions			Results	
	Previous state of significance indicator	Source digit	Low-order source digit is a plus sign	Result character	State of significance indicator after digit examination
Digit selector	Off	ø	x	Fill character	Off
		1-9	No	Source digit	On
		1-9	Yes	Source digit	Off
	On	ø-9	No	Source digit	On
		ø-9	Yes	Source digit	Off
Significance starter	Off	ø	No	Fill character	On
		ø	Yes	Fill character	Off
		1-9	No	Source digit	On
		1-9	Yes	Source digit	Off
	On	ø-9	No	Source digit	On
		ø-9	Yes	Source digit	Off
Field separator	x	xx	xx	Fill character	Off
Message character	Off	xx	xx	Fill character	Off
	On	xx	xx	Message character	On

x No effect on result character and new state of significance indicator.

xx Not applicable because source digit not examined.

Condition Code:

ø The source field has a zero value, or the mask pattern does not contain a digit selector or significance starter; the significance indicator may be off or on.

1 The result field has a nonzero value and the significance indicator is on (i.e. a negative sign or no sign at all was detected in the source field).

2 The result field has a nonzero value and the significance indicator is off (i.e. a positive sign was detected in the source field).

3 -

Note:

The condition code setting reflects only the field following the last (rightmost) field separator of the pattern for multiple-field editing operations.

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Decimal format invalid digit	T
Data address match check	C

Notes:

1. The fill character is obtained from the pattern as part of the editing operation. The first character of the pattern is used as a fill character and is left unchanged in the result, except:

   - o when it is a digit selector, or
   - o when it is a significance starter.

   In these cases, a source digit is examined and, when nonzero, inserted in the result field.

2. If the fill character is a blank, if no significance starter appears in the pattern, and if the source field is all zeros, the editing operation blanks the result field.

3. To facilitate blanking out all-zero result fields, or triggering negative field special processing, the condition code is used to indicate the sign and value of the last field edited. This indication can be evaluated at the termination of the editing operation.

4. The source field is generally shorter than the pattern because the source field digits are expanded to zoned format in the result field.

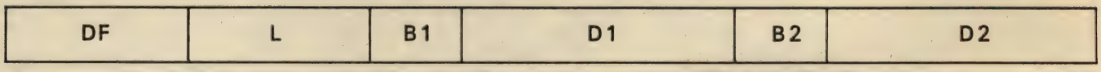
5. The total number of digit selectors and significance starters in the pattern must equal the number of source digits to be edited.

6. When the CPU checks for the system paging error and the paging queue interrupt conditions, it is assumed that the length of the source field is equal to that of the pattern. In cases where this source field extends over a page boundary and a condition exists for the second page, which generates a paging error and paging queue interrupt request, this interrupt is superfluous if the actual length of the source field is such that it is accommodated within the first page.

10.14.2. Edit and Mark (EDMK)

Type: SS

Format: EDMK



Description:

The variable-length source field specified by the second address B2/D2, containing packed decimal data, is changed to zoned format with editing under the control of a mask pattern.
The edited result replaces the mask pattern specified by the first address B1/D1, and determines the condition code. Up to this point, the Edit and Mark instruction is identical to the Edit instruction. Additionally, with the Edit and Mark instruction, the address of the first significant result digit is recorded in general register 1 (general register 13 in P3; general register 9 in P4), providing that the significance indicator was previously off.
That is to say, bit positions 8-31 of general register 1 (13 or 9) are loaded with the address of the first significance digit, if the significance indicator is set to the on state at this time.

Bit positions Ø-7 are not changed. The address is not recorded if the significance indicator is set to the on state by a significance starter in the pattern.

The Edit and Mark instruction permits the insertion of floating currency symbols, signs, relational operators and other editing symbols ($, +, -, <, >, etc.). The address loaded into the register is one byte to the right of the address where such a symbol would be inserted. (The Branch on Count instruction, with zero in the R2 field, can be used to reduce the loaded address by one). Since the address is not loaded when the significance indicator is set to the on state by the significance starter, the address of the byte immediately to the right of the significance starter in the pattern should be loaded into general register 1 (13 or 9) before an Edit and Mark instruction is executed.

Condition Code:

Ø The source field has a zero value, and the significance indicator may be off or on.

1 The result field has a nonzero value, and the significance indicator is on (i.e. a negative sign was detected in the source field).

2 The result field has a nonzero value, and the significance indicator is off (i.e. a positive sign was detected in the source field).

3 -

Note:

The condition code setting reflects only the field following the last (rightmost) field separator of the pattern for multiple-field editing operations.

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Decimal format invalid digit	T
Data address match check	C

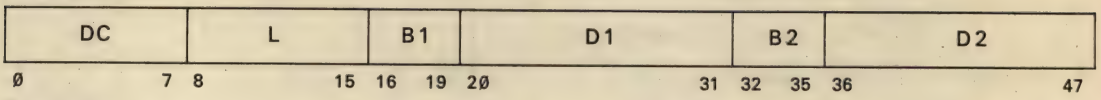
Notes:

1. All notes of the Edit instruction are applicable to the Edit and Mark instruction.
2. When a single instruction is used to edit multiple fields, the address of the first significant digit of each field (with significance indicator off) is loaded into the register. Thus, only the address of the first significant digit of the last field processed will be available upon completion of the instruction.

10.14.3. Translate (TR)

Type: SS

Format: TR



Description:

The variable-length operand 1 specified by the first address B1/D1 is translated byte by byte, according to the translation table (operand 2) specified by the second address B2/D2. The result replaces the bytes in the operand 1 field.

The bytes of the operand 1 field are termed the argument bytes. They are processed from left to right, one byte at a time. Address B2/D2 defines the start of the translation table. Each argument byte is added (binary) to the address B2/D2. This sum, in turn, addresses a byte location within the table, which is termed a function byte. The function byte at this location replaces the original argument byte in operand 1. The operation terminates when the operand 1 bytes have been exhausted.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	T
Data address match check	C

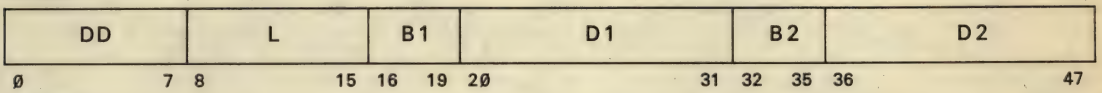
Notes:

1. The translate table is unaltered unless overlap occurs.
2. The field to be translated and the translation table are addressed by their leftmost byte.
3. The length of a table, in general, must be 256 bytes, unless the domain of argument bytes is limited to a specific subset by the program and data.
4. The L field specifies the length of operand 1 minus one (binary ØØØØØØØ1 = 2 bytes).

10.14.4. Translate and Test (TRT)

Type: SS

Format: TRT



Description:

The variable-length operand 1 specified by the first address B1/D1 is translated byte by byte, according to the translation table (operand 2) specified by the second address B2/D2. The operand specified by the first address is not changed.

The bytes of the operand 1 field are termed the argument bytes. They are processed from left to right, one byte at a time. Address B2/D2 defines the start of the translation table. Each argument byte is added (binary) to the address B2/D2. This sum, in turn, addresses a byte location within the table, which is termed a function byte.

Then, the function byte retrieved from the table is inspected for all zeros. If the function byte is all zeros, the operation proceeds to the next argument byte and continues processing.

If the function byte is not all zeros, the instruction inserts the address of the argument byte in bit positions 8-31 of general register 1 (general register 13 in P3; general register 9 in P4) and inserts the retrieved nonzero function byte in bit positions 24-31 of general register 2 (general register 14 in P3; general register 10 in P4). Bit positions Ø-7 of general register 1 (13 or 9) and bit positions Ø-23 of general register 2 (14 or 10) are unaltered.

The operation terminates when a nonzero function byte is accessed or when the operand 1 bytes have been exhausted.

Condition Code:

Ø All accessed function bytes zero
1 Non-terminal function byte nonzero
2 Last accessed function byte nonzero
3 -

Interrupt Priorities:

Error	Action
Read access	S
Physical access	T
Data address match check	C

Notes:

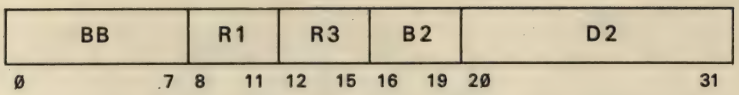
1. If nonzero function bytes do not occur, general registers 1 (13 or 9) and 2 (14 or 10) are not altered.
2. Operand 1 and the translation table are addressed by their leftmost byte.
3. The L field specifies the length of operand 1 minus one.

10.15. Miscellaneous Instructions

10.15.1. Compare Double and Swap (CDS)

Type: RS

Format: CDS



Description:

Operand 1 (contents of the general register pair specified by R1) is compared with operand 2 (B2/D2). If they are equal, operand 3 (contents of the general register pair specified by R3) is stored in location B2/D2. If they are unequal, operand 2 (B2/D2) is loaded into R1.

R1 and R3 must each specify the even-numbered register of a pair (R1, R1+1; R3, R3+1). Operand 2, a doubleword in main memory, must be doubleword-oriented. The result of the comparison, equality or inequality, determines the condition code.
Access to operand 2 is preceded by serialization, i.e. all previous processor instructions must have been executed before operand 2 is handled.

The CDS instruction can be used for program coordination in a multiprocessing environment because it is impossible for:

- the result stored at location B2/D2 to be falsified by memory cycles of the other central processors.
- the process of storing operand 2 to affect the instruction flow in the other central processors or to falsify the results.

Condition Code:

Ø Operands 1 and 2 are equal, operand 2 replaced by operand 3
1 Operands 1 and 2 are not equal,
operand 1 replaced by operand 3
2 -
3 -

Interrupt Priorities:

Error	Action
Doubleword register error	S
Read access	S
Write access	S
Physical access	S
Doubleword alignment	S
Data address match check	C

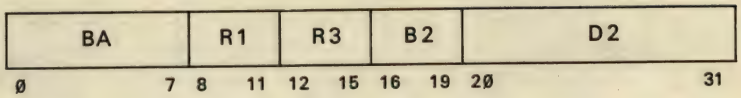
Note:

I/O cycles may be executed concurrently with the CDS instruction. Thus, the instruction cannot be used for coordination with channel programs.

10.15.2. Compare and Swap (CS)

Type: RS

Format: CS



Description:

Operand 1 (contents of the general register specified by R1) is compared with operand 2 (B2/D2). If they are equal, operand 3 (contents of the general register specified by R3) is stored in location B2/D2. If they are unequal, operand 2 (B2/D2) is loaded into R1.

Operand 2, word in main memory, must be word-oriented. The result of the comparison, equality or inequality, determines the condition Code.

Access to operand 2 is preceded by serialization, i.e. all previous processor instructions must have been executed before operand 2 is handled.

The CS instruction can be used for program coordination in a multiprocessing environment because it is impossible for:

- the result stored at location B2/D2 to be falsified by memory cycles of the other central processors.
- the process of storing operand 2 to affect the instruction flow in the other central processors or to falsify the results.

Condition Code:

Ø Operands 1 and 2 are equal, operand 2 replaced by operand 3.
1 Operands 1 and 2 are not equal, operand 1 replaced by operand 2.
2 -
3 -

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	S
Word alignment	S
Data address match check	C

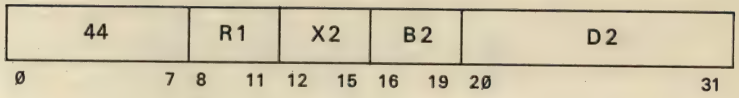
Note:

I/O cycles may be executed concurrently with the CS instruction. Thus, the instruction cannot be used for coordination with channel programs.

10.15.3. Execute (EX)

Type: RX

Format: EX



Description:

The subject instruction in the location specified by the second address X2/B2/D2 is modified by the contents of the register specified by the first address (R1). X2/B2/D2 must be halfword-oriented. The modified instruction is then executed and control is returned to the instruction following the Execute instruction.

Condition Code:

May be set by the subject instruction; otherwise, unchanged.

Interrupt Priorities:

Error	Action
Execute error	S
All interrupts of the subject instruction	S, T, C

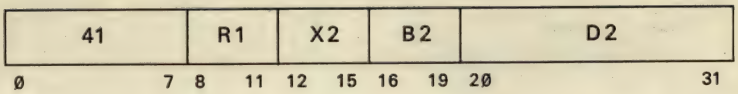
Notes:

1. The instructions Execute, Load Word Indirect, and Store Word Indirect may not be the subject of an Execute instruction.
2. Bits 8-15 of the subject instruction are ORed with bits 24-31 or the register specified by the first address (R1).
3. If R1 is zero, no modification takes place.
4. The contents of R1 and the subject instruction in main memory are unaltered.
5. The ILC is set to two (length of the Execute instruction) and the PCR is set to the address of the instruction following the Execute instruction.
6. When the subject instruction is a successful branch instruction, the PCR is updated by the branch address.
7. Interrupts are inhibited until the subject instruction has been completed.

10.15.4. Load Address (LA)

Type: RX

Format: LA



Description:

The contents of the registers specified by the X2 and B2 fields are added to the displacement contained in the D2 field to obtain an address. Any carry beyond the low-order 24 bits is ignored. This is the address that is loaded into the register specified by the first address (R1).

The eight high-order bits of the register are set to zeros.

Condition Code:

Unchanged

Interrupt Priorities:

None

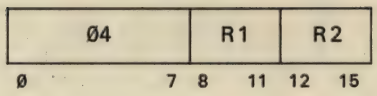
Notes:

1. R1, X2 and B2 may specify the same register; however R1 only may specify register Ø.
2. This instruction can be used to increment the low-order 24 bits of a general register (other than register Ø) by the contents of the D2 field.
   The register to be incremented is specified by R1, and either X2 (with B2 set to zero) or B2 (with X2 set to zero). Since R1 and X2 or B2 must specify the same register, register Ø cannot be incremented (a zero in the B2 or X2 field indicates that the corresponding address component is absent).
3. Main memory is not accessed by this instruction.

10.15.5. Set Program Mask (SPM)

Type: RR

Format: SPM



Description:

Bits 2-7 of the general register specified by the first address (R1) determine the new program mask and the new condition code setting for the current processor state.

Condition Code:

The condition code is set according to bits 2 and 3 of the general register specified by R1 as follows:

2	3	Result
Ø	Ø	Set condition code Ø
Ø	1	Set condition code 1
1	Ø	Set condition code 2
1	1	Set condition code 3

Interrupt Priorities:

None

Program Mask:

The program mask is set according to bits 4-7 of the general register specified by R1 as follows:

Bit	Result
4	Fixed-point overflow
5	Decimal overflow
6	Exponent underflow
7	Significance error

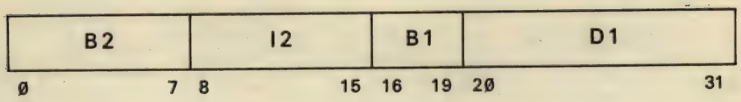
Notes:

1. The contents of the PCR and the general register specified by R1 are unaltered.
2. The R2 field of the instruction is ignored.

10.15.6. Store Clock (STCK)

Type: SI

Format: STCK



Description:

The current value of the time-of-day clock is stored at the eight-byte field designated by the operand address B1/D1. The value of the clock is expressed as a fixed-point number consisting of a sign and a 63-bit integer field. Zeros are provided for the low-order bit positions of the integer field that are not updated by the clock (bit 52-63 of the eight-byte field)

The I2 field must contain the value $(05)_{16}$

When the clock is in the error state or is not operational (the clock is disabled or its power is down), the value stored by the instruction is zero.

The quality of the clock value stored by the instruction is indicated by the resultant condition code.

Condition Code:

Ø Clock in set state
1 Clock in not-set state
2 Clock in error state
3 Clock not operational

Interrupt Priorities:

Error	Action
Invalid I2 code	S
Write access	S
Physical access	T
Data address match check	C

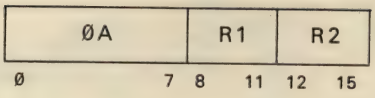
Notes:

Condition code Ø indicates that the clock's value provides a valid measure of elapsed time since the last time it was set. This code normally indicates that the clock's value is a valid time-of-day and calendar indication. Condition code 1 indicates that the clock's value is the elapsed time since the power for the clock was turned on. In this case, the value may be used for elapsed time measurements but is not a valid time-of-day indication. Condition code 2 and 3 indicate that the value provided by Store Clock cannot be used for time measurement or indication.

10.15.7. Supervisor Call (SVC)

Type: RR

Format: SVC



Description:

The R1 and R2 fields provide an interruption code which supplies the link to the program which is to be called by the instruction. This code is placed in the low-order byte of the Interrupt Status Register (ISR) of the processor state in which this instruction is issued. The supervisor call interrupt flag bit (priority 53) is set in the Program Interrupt Flag Register (PIFR) and an interrupt may occur depending on the corresponding mask bit in the Interrupt Mask Register (IMR) of the current state.

Condition Code:

Unchanged

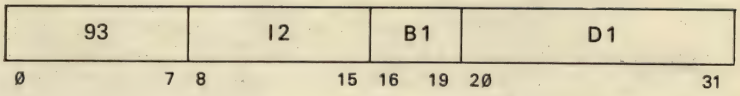
Interrupt Priorities:

No interrupts other than that inherent in the execution of SVC can occur during its execution.

10.15.8. Test and Set (TS)

Type: SI

Format: TS



Description:

This instruction is used to test and set a byte in memory in a manner that allows multiprocessor coordination. The I2 field of the instruction (bits 8-15) is not used.

The byte at the location specified by address B1/D1 is fetched, tested and the high-order bit (bit Ø) is used to set the condition code. Then the addressed byte is set to all ones.

It is impossible for:

- the byte stored at location B1/D1 to be falsified by memory cycles of the other central processors or by channel programs between testing and setting.
- the process of storing the byte into B1/D1 to affect the instruction flow in the other central processors or to falsify the results.

Serialization precedes execution of the instruction, i.e. all previous processor instructions must have been executed.

Condition Code:

Ø Bit Ø was zero
1 Bit Ø was one
2 -
3 -

Interrupt Priorities:

Error	Action
Read access	S
Write access	S
Physical access	S
Data address match check	C

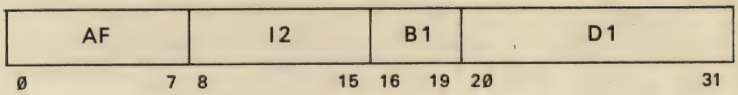
Notes:

The byte used in TS may only be reset by means of an instruction requiring only one physical access (e.g. MVI). Otherwise multiprocessor coordination cannot be guaranteed.

10.15.9. Monitor Call (MC)

Type: SI

Format: MC



Description:

This instruction tests whether a specific mask bit in the Monitor Mask Register (MOMR) is set. If this is the case, bit 8 of the Program Interrupt Flag Register (PIFR) is set and an address error interrupt (monitor call interrupt) occurs.

The MOMR contains 16 monitor mask bits in bit positions 16-31. Each of these mask bits designates one of 16 monitor classes in ascending order. The low-order 4 bits of the I2 field contain a binary number specifying one of 16 monitor classes.

When the MOMR bit corresponding to the class specified by the I2 field is one, a monitor call interrupt occurs. Bit 8 of the appropriate PIFR (address error) is set to one, and $(1C)_{16}$ is entered in bit positions Ø-7 of the Error Cause Register (ERCR). The contents of the I2 field are loaded into bit positions Ø-7 of the Monitor Call Register (MOCR), and the address computed from B1/D1 is loaded into bit positions 8-31 of the MOCR.

When the MOMR bit corresponding to the class specified by the I2 field is zero, no interrupt occurs.

The high-order 4 bits of the I2 field must be zeros.

Condition Code:

Unchanged

Interrupt Priorities:

Error	Action
Invalid I2 code (bits 8-11 not zeros)	S
Monitor call (not an error, but a desired address error interrupt with error class 1C in the ERCR)	C

11. Machine Error Reporting and Recovery

11.1. Error Detection

The 7.000 System central units detect machine errors and react automatically. Error information generated by the hardware (error logout) is used in selecting which of the following actions is performed to recover the error :

- o Retry of the affected instruction, either automatically or by the operating system
- o Retry of the affected I/O operation
- o Retry of the affected user program (restart at the check point or new start)
- o Retry of the system (restart at the system checkpoint or new start of the operating system)

Method and extent of error detection are machine-dependent.
The error information thus consists of a machine-independent portion valid for all central units and a machine-dependent portion.

11.2. Error Classification

The errors detected by the central unit are subdivided into the following 4 classes:

Central Processor Errors

Central processor errors are generally detected by parity checks on the data paths and registers.

I/O Errors

Errors in the I/O system are detected during the execution of I/O operations.

Main Memory Errors

Errors in the main memory system are detected during the execution of main memory operations.

Environmental Errors

The detection of environmental erros is machine-dependent.

Power failure and overheat in the central processor, I/O system or main memory are not reported as environmental errors but cause a power failure interrupt.

11.3. Central Processor Error Handling

The central processor executes the following routines automatically:

- o Error logout
- o Instruction retry
- o Machine check
- o Central processor stop

The error mode set determines which of these routines must be executed and in what sequence.

11.3.1. Error Control Modes

The following modes can be set in the Error Control Register :

Quiet Mode

The central processor automatically attempts an instruction retry.
If instruction retry is not possible or if an error again occurs during the retry, error logout is performed and a machine check interrupt occurs. If instruction retry is successful, no error logout takes place and no machine check interrupt occurs.

Logout Mode

The central processor performs error logout which is then followed by an automatic instruction retry.
If instruction retry is successful, not successful or not possible, a machine check interrupt occurs.

Machine Check Interrupt Processing Mode

The machine check interrupt processing mode is set automatically during a machine check interrupt.
The central processsor automatically attempts an instruction retry. If instruction retry is not possible or if an error again occurs during a retry, a central processor stop occurs. If instruction retry is successful, error logout is not performed and no program interrupt occurs.

Machine Error Stop Mode

A central processor stop occurs in the event of an error in this mode. Instruction retry is not attempted and error logout is not performed.

Alarm Inhibit Mode

In this mode, a machine-dependent amount of the error reporting logic is disabled.

11.3.2. Error Logout

During error handling, the central processor generates error information which it stores in main memory (error logout). This error information consists of the standard error information and the detailed error information.

The machine-independent standard error information consists of the central processor error word which has the following format:

Bit Ø

Main memory error
Bit Ø=1 when an error was detected in main memory.

Bit 1

Central processor error
Bit 1=1 when an error was detected in the central processor.

Bit 2

Error affecting an I/O operation

Bit 2=1 when a central processor error or main memory error occurred during the execution of an I/O operation.

Bit 3

Environmental error

Bit 3=1 when an environmental error was detected.

Bit 4

System error

Bit 4=1 when a central processor error or main memory error occurred affecting processor state change (program interrupt or Program Control instruction), interface communication, or data exchange with another central processor.

Bit 4 is also set if a program error occurred during operation in processor state P3 or, if possible, P4.

Bit 5

Instruction retry successful

Bit 5=1 if automatic instruction retry was successful.

Bit 6

Timer error

Bit 6=1 if, after an error, the contents of the program timers or time-of-day clock are no longer correct.

Bit 7

Main memory inoperable

Bit 7=1 when the central processor has detected that main memory is inoperable.

Bits 8-11

Central processor model identifier

Bits 8-11 contain a 4-bit binary number which identifies the central processor model that performed error logout.

Bit 12

Instruction retriable

Bit 12=1 when the instruction is retriable by software.

Bits 13-16

Bits 13-16 contain a 4-bit binary count which specifies the number of instruction retries performed by hardware.

Bits 17-31

Not used.

The format and number of words of detailed error information are machine-dependent.

If an uncorrected error occurs during error logout, a central processor stop occurs.

11.3.3. Instruction Retry

After the detection of an error, the central processor automatically retries the affected instruction without software support. This attempt is apparent on the machine language level only through bit 5 in the central processsor error word which is set depending on the result of the instruction retry, and through the number of instruction retries specified in bits 13-16.

If an uncorrected error occurs during instruction retry, the retry is regarded as unsuccessful.

11.3.4. Machine Check Interrupt

The central processor enters the machine check interrupt processing mode.

Bit 3Ø (machine check interrupt) in the Interrupt Flag Register (IFR) is set to one.

If bit 3Ø=1 in the Interrupt Mask Register (IMR), a machine check interrupt is taken and the central processor enters the P4 state.

If an unrecoverable error occurs during the operations mentioned above, a central processor stop occurs.

11.4. Main Memory Errors

Main memory errors are not handled as independent error conditions, but indirectly as central processor errors or I/O errors, depending on whether the central processor or the I/O system initiated the affected main memory operation.

11.5. Environmental Errors

Instruction retry is not attempted for environmental errors. Error logout is performed and a machine check interrupt occurs immediately.

12. Input/Output Operations

12.1. Introduction

An input/output (I/O) operation consists of the transfer of a number of data bytes between the main memory system and the data medium in the peripheral devices under the control of a program running in the central unit. Control bytes are transferred in the same way to or from the device controller. The method of control is the same for all 7.000 System machines.

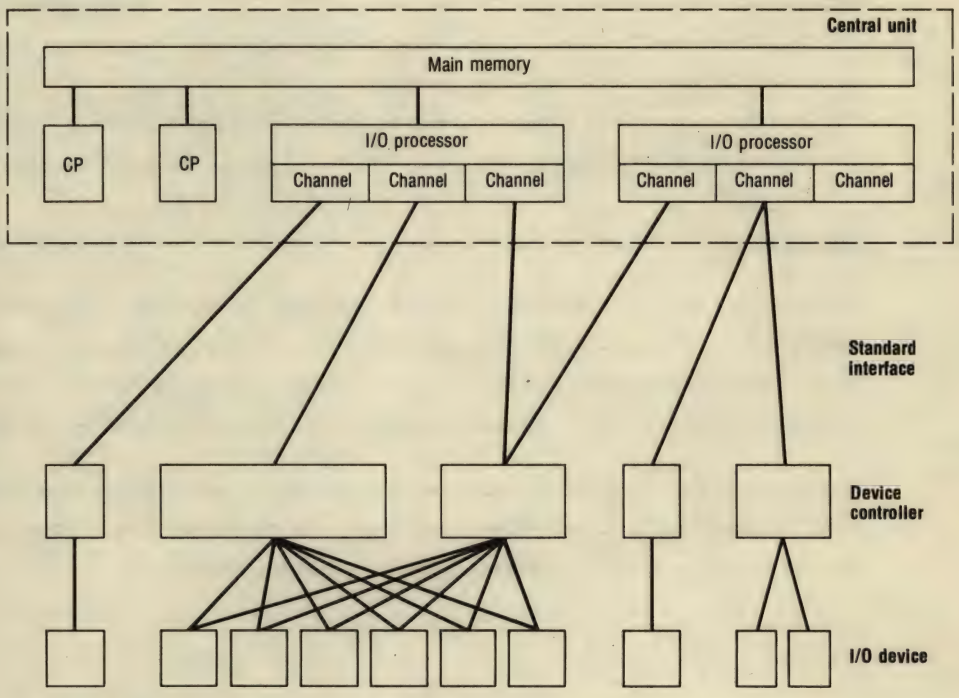
The data bytes are transferred sequentially, one at a time, the order being identical at the source and the destination.

The main memory system and the I/O devices are linked functionally by I/O processors, I/O channels and device controllers.

There are normally several I/O channels assigned to one I/O processor. In some cases, extensive sections of the I/O system may be integrated in the central processor.

One device controller may support one or more I/O devices. Together they are regarded as a peripheral unit.

The functions peculiar to the various device types are implemented in the individual devices and device controllers. It is thus possible to implement a standardized interface between device controller and channel, the Siemens System 7.000 standard I/O interface.



CP = Central processor

Fig. 12-1 I/O System Structure

12.1.1. I/O Operational Sequence

All I/O operations are performed independently by the I/O channels. The program must provide control parameters before selecting and activating the I/O processor, channel and device with an I/O instruction.

The transfer of control information for the initiation and execution of an I/O operation is effected on three different levels :

- The I/O machine instruction, a component of the executing program, is decoded by the central unit. It specifies the I/O processor, the channel and the device for the I/O operation to be performed.
  The program provides the initial value of the Channel Address Word (CAW) prior to instruction execution.
- The channel program is decoded and executed by the channel. It consists of one or more Channel Command Words (CCW), which specify the type of operation to be carried out, determine the start addresses of the I/O areas in main memory and the number of bytes to be transferred, and control channel program execution.
- The command code of the CCW is sent to the device controller as an interface command.

The program is notified of the termination of an I/O operation by a channel interrupt request.

12.1.1.1. I/O Initiation

The I/O operation is initiated by the Start Device (SDV) instruction providing that the selected system components (I/O processor, channel, device controller and device) are in a position to execute the I/O operation. The contents of the Channel Address Word (CAW) and the first Channel Command Word (CCW) are transferred to the control registers of the selected channel. The channel command specified in the first CCW is issued to the device controller.
The Start Device instruction is completed and the central unit is free to continue program processing. The result of the I/O initiation is communicated to the program by the condition code setting.

12.1.1.2. Data Transfer

After initiation of an I/O operation, a set of data or control bytes can be transferred between main memory and the device or device controller. Transfer is controlled by the device controller, which issues a service request for each byte. If the channel permits the service request, a data transfer occurs. In addition, the information in the control registers of the channel is updated for the next byte transfer.

A data transfer involves either one main memory area, which is defined by the data address in a single CCW, or several non-contiguous main memory areas. Input/output of a data block to/from more than one main memory area occurs during data chaining and page chaining.

In the case of data chaining, the data transfer operation does not end after processing of a main memory area with the transfer of the specified number of bytes. The control registers of the channel are loaded according to the next CCW and the data transfer operation continues.

Page chaining supports the execution of I/O operations when virtual memory addressing is employed. Each time a page boundary is crossed, the new real address is computed and the control registers of the channel are loaded.

Since there is no change in the type of operation during data and page chaining, functioning of the device controller and device is not affected.

12.1.1.3. End Servicing

After the number of bytes specified in the channel command has been transferred, end of data transfer is communicated to the device controller. This, as also in the case of an end condition in the device controller or peripheral device, causes the device controller to issue another data service request with an end condition.

During end servicing, the channel determines whether the I/O operation is to be terminated, or the channel program continued with the next channel command.

Command chaining is performed when specified by the CCW processed last and no condition has occurred in the channel or the device controller to prevent it. The next CCW is then read from main memory and stored in the control register of the channel, and the new channel command is sent to the device controller.

In the absence of command chaining, the device controller is asked to request an interrupt.

12.1.1.4. Interrupt Servicing

The interrupt request from the device controller causes the interrupt flag assigned to that channel to be set in the Interrupt Flag Register (IFR). The interrupt is taken providing that the corresponding bit is set in the Interrupt Mask Register (IMR) of current processor state.

The interrupt informs the program of the termination and the result of the I/O operation and also of the status of the channel, device controller and device.

12.2. I/O Device Connection

12.2.1. Devices

I/O devices provide the link with the external storage media, handle data interchange between remote systems and facilitate communication between system and environment.

An I/O device is normally connected via one device controller to one channel. Devices can also be connected to several device controllers and channels.

12.2.2. Device controllers

The device controllers monitor and control the functions of the I/O devices and adapt the individual characteristics of the different device types so that these may be serviced in a uniform manner by the channel and standard I/O interface. Controllers supporting only a single device are normally integrated in the device.

Multi-device controllers on the other hand communicate with the individual devices (drives) via internal interfaces, and incorporate all those functions which are common to all the devices. These are primarily control of sequences on the standard I/O interface and status reporting to the channel. Since, with most device types which connect to mulit-device controllers (e.g. disk storages, magnetic tape devices etc.), only one device can communicate with the controller at any one time, a large part of the control logic for the devices is accommodated centrally in the device controller.

12.2.3. I/O Channels

The I/O channels control data transfer between the device controllers and main memory. The channel receives the control information upon I/O initiation in the form of a channel program. The remainder of the operation is controlled and monitored independently by the channel.
Operations in one channel can take place simultaneously to operations in other channels and processors.

After normal or abnormal termination of the channel program, the channel provides status information.

The central units of the Siemens System 7.000 are provided with three types of I/O channels:

- o Byte multiplexor channel
- o Block multiplexor channel
- o Selector channel

The execution of an I/O operation as described under 12.1.1. and the format of the Channel Command Words are identical for all types of I/O channel.

The different types of channel are distinguished by the extent to which they permit simultaneous execution of several I/O operations and thus several channel programs.

The facilities which control one of the simultaneous I/O operations and store the relevant control and status information form a subchannel. One of the factors which determine an I/O channel's ability to handle several operations at a time is the number of subchannels available. The number of subchannels determines the maximum possible number of simultaneous I/O operations. There are a maximum of 256 device addresses per I/O channel, and thus also a maximum of 256 subchannels.

12.2.3.1. Byte Multiplexor Channel (BYMUX)

One BYMUX channel can perform a number of simultaneous I/O operations. Byte multiplex operation is effected in such a way that after transfer of each individual byte, the channel can process a service request from another device controller for a further I/O operation. In the case of multi-device controllers, simultaneous execution of I/O operations for the devices connected to a controller depends on whether the device controller can operate in the byte multiplex mode.
Normally, it is the relatively slow devices which are connected to a BYMUX channel.

12.2.3.2. Block Multiplexor Channel (BLMUX)

A BLMUX channel operates in either the block multiplex or selector mode under program control.

Block multiplex operation is effected in such a way that after transfer of a physical block, the channel can disconnect from the device and accept an I/O operation for another device.

Block multiplexing is only possible with device controllers which can operate in this mode.

Several fast devices are normally connected to the BLMUX channel via one or more device controllers.

A BLMUX channel operating in the selector mode is analogous to a selector channel.

12.2.3.3. Selector Channel

The selector channel can only perform one I/O operation at a time. A further operation can be initiated only after processing of the termination interrupt. Therefore only a relatively small number of fast devices are connected to one selector channel.

12.2.3.4. Configuration

The maximum configuration depends upon the operating mode:

The non-extended I/O mode provides for one BYMUX channel and up to six BYMUX channels.

Each BLMUX channel can be replaced by a selector channel.

On the extended I/O mode, up to four I/O processors can be implemented - each with a maximum of 16 channels, although the total must not exceed 32. There are no restrictions on the number of channels of a particular type.

The maximum number of devices per channel is limited in both modes to 256.

12.3. I/O Control

The central processor controls input/output with the four privileged machine instructions:

- Start Device (SDV)
- Halt Device (HDV)
- Test Device (TDV)
- Check Channel (CKC)

These instructions are termed the I/O instructions.

Upon execution of an I/O instruction, the addressed channel receives all the control information necessary to independently control the I/O exchange between the central unit and the connected I/O devices. Execution of the I/O instruction ends with the condition code being set in accordance with the result.

In the case of a Start Device (SDV) instruction and a correctly initiated I/O operation (condition code Ø), the program is informed of the termination of the I/O operation by a termination interrupt. It can also be informed of the channel program flow by a program controlled interrupt.

12.3.1. Addressing

The I/O instruction addresses a specific channel and (except for the Check Channel (CKC) instruction) a specific device connected to that channel. In the extended I/O mode, the number of the I/O processor to which the addressed channel is assigned must also be specified. Address field B1/D1 of the I/O instructions is used to specify the I/O processor number, channel number and device address.
In the non-extended I/O mode, a maximum of one byte multiplexor channel and six block multiplexor channels can be addressed; since only one I/O processor is available, I/O processor addressing does not apply. In the extended I/O mode, up to four I/O processors may be implemented, each with a maximum of 16 addressable channels, but not more than a total of 32.
Up to 256 I/O devices can be addressed per channel, irrespective of the I/O mode. The number of installable channels may be further restricted within the given addressing scope, depending on the particular model of central unit.

12.3.2. Status Reporting

During the execution of I/O instructions, a check is made on the status of I/O processor, channel and device with respect to instruction initiation. If unsatisfactory or if there are formal errors in the channel program, the instruction terminates with the condition code set to 1 or 3, and the program is provided with the relevant status information. The program also receives status information whenever it services a channel interrupt request.

This status information comprises:

- Condition Code - only after the execution of I/O instructions
- I/O Processor Not Available (INA) bit
- I/O Error (IOE) bit
- Channel Status Byte (CSB)
- Standard Device Byte 1 (SDB1)
- Standard Device Byte 2 (SDB2) - only when an attention interrupt occurs.

The program receives this information automatically. Under certain circumstances, additional status information is prepared in the device in the form of sense bytes. The program must request this device-dependent information from the device controller with a Sense command.

12.3.2.1. Condition Code

The condition code is set when an I/O instruction is executed. The 2-bit condition code indicates the result of the I/O instruction, but not of any I/O operation which may have been initiated. Since the condition code is set according to status information collected during instruction execution, the status of the I/O processor can to a certain extent be deduced from the condition code. The meaning of the condition code differs from instruction to instruction and is therefore detailed in the description of the individual I/O instructions (see 12.3.3).

12.3.2.2. I/O Processor Not Available (INA)

The INA bit is set during I/O instruction executions,or when a channel interrupt occurs, and the addressed I/O processor either is not installed or is inoperable and has not reported an I/O error.

12.3.2.3. I/O Error (IOE)

The IOE bit is set when the channel or I/O processor becomes inoperable owing to a hardware error. The IOE bit is only applicable when the INA bit is not set.

12.3.2.4. Channel Status Byte (CSB)

The channel status byte contains information concerning the status of the channel when a channel interrupt occurs, or on termination of one of the instructions Start Device (SDV), Halt Device (HDV) or Test Device (TDV) with condition code 1.

Three of the eight bits in the channel status byte identify in coded form the type of interrupt (termination interrupt, program controlled interrupt, channel free interrupt, attention interrupt) and, in the case of a block multiplexor channel, the current operating mode of the channel (selector or block multiplex mode); thus they are only significant when channel interrupts are taken.

The five remaining bits indicate the occurrence of abnormal conditions in the channel, with each abnormal event being assigned one bit. The following abnormal conditions are flagged in the channel status byte:

Incorrect Length (INCL)

When set, the INCL bit indicates that the number of bytes actually transferred over the standard I/O interface was less than the number specified by the byte count in the Channel Command Word (the final count is greater than zero). The incorrect length indication relates only to the last CCW executed before termination of the I/O operation. INCL is not set if the suppress length indicator (SLI) flag bit is set in the executed channel command.

Channel Program Check (CPC)

When set, the CPC bit indicates that a programming error has been detected in the channel program. The following conditions cause the CPC bit to be set :

- The address of the CCW is not doubleword-aligned.
- The address of the CCW is outside the available main memory.
- The data address specified for a data transfer is outside the available main memory.
- The command code in the CCW is illegal.

Memory Protect Check (MPC)

When set, the MPC bit indicates that a memory access to read the next CCW, or to read or write data has been rejected due to a protection error.

Channel Data Check (CDC)

When set, the CDC bit indicates that the channel has received a data byte with bad parity from the standard I/O interface during an input operation. The erroneous byte is replaced by the systems error byte $(FF)_{16}$ and the input operation is continued. However, command chaining is suppressed and the operation is terminated with a termination interrupt. If the channel detects an output byte with bad parity, this error can be reported either by the CDC or CCC bit in the channel status byte or by the IOE bit. The way in which an output data error is reported depends on the particular model of central unit.

Channel Control Check (CCC)

When set, the CCC bit indicates that a hardware error has been detected in the channel, or an illegal bit combination has been found in the standard device byte during end servicing.

Note:

The occurrence of a CPC, MPC or CCC condition results in the immediate termination of the current I/O operation, and a termination interrupt is generated. If any of these errors occurs during execution of the Start Device (SDV) instruction, the instruction is terminated with condition code 1 and the I/O operation is not initiated.

12.3.2.5. Standard Device Byte 1 (SDB1)

The standard device byte 1 indicates the status of an I/O device during I/O interrupt servicing (attention interrupt or termination interrupt) or on completion of one of the instructions Start Device (SDV), Halt Device (HDV) or Test Device (TDV) with condition code 1.
The eight bits of SDB1 are defined as follows:

Attention Interrupt Request

When set, this bit indicates that a device requires servicing, or a device wants to indicate a status change.

Termination Interrupt Pending

When set, this bit indicates that a termination interrupt condition exists in an I/O device.

Device Busy

When set, this bit indicates that the specified device is busy and cannot accept another channel command.

Control Busy

When set, this bit indicates that the specified device controller is busy and cannot accept another channel command.

Device End

When set, this bit indicates that the device has terminated.
Another channel command can be accepted by the device if the device busy bit is not set.

Secondary Indicator

When set, this bit indicates that the specified device has additional status indicators (sense bytes) to be tested. These indicators can be stored in memory by the Sense 1 command.

Device Inoperable

When set, this bit indicates that the specified device is either inoperable or not installed.

Status Modifier

This bit is only used with command chaining. When set, it indicates that the next Channel Command Word is to be skipped.

12.3.2.6. Standard Device Byte 2 (SDB2)

The standard device byte 2 is a qualifier to define the reason for an attention interrupt request. The information indicated by SDB2 is device dependent. SDB2 is only stored when an attention interrupt occurs, and only with certain device types.

12.3.2.7. Sense Bytes

Errors and special conditions in a device are stored in one or more sense bytes in the device controller. This is indicated by the secondary indicator bit in SDB1. Sense bytes are brought into main memory by the Sense 1 command.

12.3.3. I/O Instructions

Four I/O instructions (SI type) are provided :

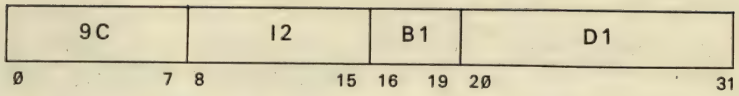
- Start Device (SDV)
- Halt Device (HDV)
- Test Device (TDV)
- Check Channel (CKC)

In contrast to the normal definition of this instruction type, the address field B1/D1 of I/O instructions is not used to address a location in memory but to identify an I/O channel, device and (when applicable) I/O processor. Under certain circumstances, status information is stored during instruction execution.

12.3.3.1. Start Device (SDV)

Type : SI

Format : SDV



Description:

Address field B1/D1 specifies the channel, the device and, in the extended I/O mode, also the I/O processor, to which the instruction applies. The I2 field is not used and must be zero.

On instruction execution, the channel status byte and the standard device byte are used to check whether the I/O processor, channel and device are capable of performing an I/O operation. If a block multiplexor channel is addressed and the operation is to be initiated in the block multiplex mode, a check is made as to whether the addressable device is capable of performing the I/O operation in the block multiplex mode.

The Channel Address Word (CAW) is read from its location in main memory in the non-extended I/O mode, or from general register 11 of the current processor state in the extended I/O mode, where it was stored prior to instruction execution. The first Channel Command Word (CCW) to be executed is read using the address contained in the CAW. The CAW and CCW are checked for formal errors.

In the absence of any conditions inhibiting the execution of an I/O operation, the operation is initiated. The CAW and CCW are transferred to the channel control registers and the condition code is set to Ø. The channel control registers enable the channel to control the I/O operation independently. If any condition inhibiting initiation of the I/O operation is detected during execution, the condition code is set to 1, 2 or 3.

Condition Code:

Ø I/O operation initiated and channel proceeding with execution.

1 I/O operation not initiated; CSB and SDB have been stored.

2 I/O operation not initiated; channel, device controller or device is busy, or a selector mode attention interrupt is pending.

3 I/O operation not initiated; I/O processor, channel or device is not available or an I/O error has occurred. Status information (INA and IOE bits) has been stored.

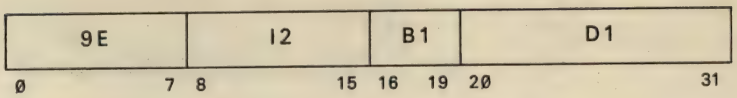
Interrupt Priorities:

Error	Action
Privileged operation	S

12.3.3.2. Halt Device (HDV)

Type : SI

Format : HDV



Description:

If the device specified by address field B1/D1 is busy and has not yet terminated the current operation, the channel and device are caused to terminate this operation immediately and to issue a premature termination interrupt request.

If the device is not busy, or is still busy but has already terminated the current operation, the status of channel and device remains unchanged and the instruction is completed with the condition code indicating the result of the instruction.

Condition Code:

Ø Specified device not busy, or is busy, but has already terminated the current operation.

1 CSB and SDB have been stored.

2 Specified device is busy and has not yet terminated current operation. Channel and device were made to effect immediate termination of I/O operation.

3 Status information (INA and IOE bits) has been stored.

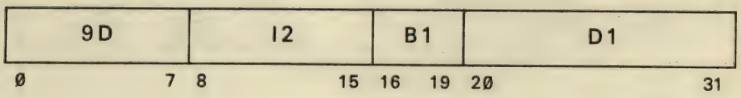
Interrupt Priorities:

Error	Action
Privileged operation	S

12.3.3.3. Test Device (TDV)

Type : SI

Format : TDV



Description:

A check is made as to whether the channel, device and I/O processor specified by address field B1/D1 are available for execution of an I/O operation. The status of channel and device is not changed.

Condition Code:

Ø I/O processor, channel and device are available for execution of an I/O operation.

1 CSB and SDB have been stored.

2 Device or channel is busy, or a selector mode attention interrupt is pending.

3 Status information (INA and IOE bits) has been stored.

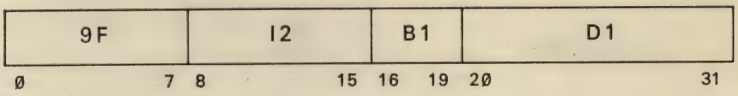
Interrupt Priorities:

Error	Action
Privileged operation	S

12.3.3.4. Check Channel (CKC)

Type : SI

Format : CKC



Description:

The status of the I/O processor and channel specified by address field B1/D1 is checked. The result of the status inquiry is reported by setting of the appropriate condition code, and status information is stored when applicable. The status of the channel is not changed by the instruction.

Condition Code:

Ø
- Specified channel is a BYMUX channel which is available and not in the error state.
- Specified channel is a BLMUX channel which is available, not in error state, not busy, and has no selector mode attention interrupt pending.

1 Specified BLMUX channel is available, not in error state, and has a selector mode attention interrupt pending.

2 Specified BLMUX channel is available and busy.

3 Status information (INA and IOE bits) has been stored.

Interrupt Priorities:

Error	Action
Privileged operation	S

12.4. Execution of I/O Operations

Before initiation of the first I/O operation by means of the Start Device instruction, the channel program must be generated in main memory. It consists of either a single Channel Command Word (CCW) or a number of CCWs which are processed sequentially.
Individual CCWs are doubleword-aligned; succesive CCWs are stored consecutively in ascending order, or are linked by the Transfer in Channel command. The first CCW in the channel program is read using the memory address specified in the Channel Address Word (CAW). The address of each subsequent CCW is obtained from the CCW address plus eight of the CCW processed last, or from the address specified in the Transfer in Channel command.

The CCW (except Transfer in Channel) contains all the information needed to control transfer of a data block between main memory and I/O device. The command code specifies the channel command for the next I/O operation to be performed. This channel command initiates the appropriate I/O operation which involves all those activities in the channel and the I/O device which take place after reading of the CCW and transfer of the command code to the device up to final end servicing.

The data address, i.e. the real main memory address of the first data byte to be transferred, and the byte count, which indicates the maximum number of data bytes which can be transferred with this CCW, determine the position and size of the I/O buffer area in main memory. Data is transferred between this I/O buffer and the addressed device. Data organization is identical in the buffer and on the data medium in the device. Data is normally written to, or read from, memory locations consecutively in ascending order of addresses commencing with the leftmost byte of the I/O buffer. An exception to this rule is the Read Reverse command where transfer commences with the rightmost byte of the input area and continues in descending order of addresses.

In order to facilitate I/O operation in virtual main memory systems, the I/O buffer specified by the CCW can be subdivided by page chaining into real 2K pages, which need not be contiguous areas in main memory.

When the transfer specified in the CCW has been completed, the subsequent channel program flow depends upon the result of this transfer and the state of the flag bits (see Table 12-1).

If a further CCW from the channel program is to be executed, this CCW is automatically read from main memory and loaded into the appropriate channel control registers. This process is known as chaining. In the case of data chaining, the current I/O operation is assigned an additional I/O buffer in main memory. Thus, several areas in main memory can be allocated for the data to be transferred within one I/O operation without affecting the I/O device. In the case of command chaining, a new I/O operation is initiated within the channel program. Thus, several I/O operations for one device are chained together, without the need to execute a Start Device instruction each time.

The channel program terminates normally when all the CCWs in the channel program have been processed correctly and the last CCW in the chain has been processed. A channel interrupt is generated and the machine program is informed of the termination of the channel program.

Flag bits of the CCW processed last			Current state of byte count in channel control register	Abnormal conditions arising from transfer just performed	Next operation in channel program flow
DC	CC	SLI			
1	X	X	Ø	Ø	Data chaining
Ø	1	Ø	Ø	Ø	Command chaining
Ø	1	1	X	Ø	Command chaining
Ø	Ø	Ø	Ø	Ø	* Channel program termination
Ø	Ø	1	X	Ø	* Channel program termination
Ø	Ø	Ø	≠ Ø	Ø	** Channel program abort
1	X	1	X	Ø	"
1	X	Ø	≠ Ø	Ø	"
Ø	1	Ø	≠ Ø	Ø	"
X	X	X	X	1	"

*≙ Normal termination of channel program

**≙ Premature termination of channel program in the event of impermissible conditions

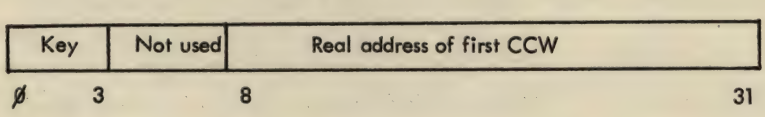
Table 12-1 Channel Operation after Processing of a CCW

The channel program aborts (terminates prematurely) when there are still unprocessed CCWs in the channel program but abnormal conditions in the channel program, channel or device prevent correct chaining, or the I/O operation has been terminated by the Halt Device instruction.
A channel interrupt is generated and the machine program is informed of the channel program abort.

Warning: Do not change the individual CCWs dynamically. The efficiency of such measures depends on channel implementation and can therefore not be guaranteed.

12.4.1 Channel Address Word (CAW)

The Channel Address Word (CAW) is used by the Start Device instruction. It contains the protection key and the address of the first Channel Command Word (CCW) which is required for control of the I/O device. The CAW has the following format :



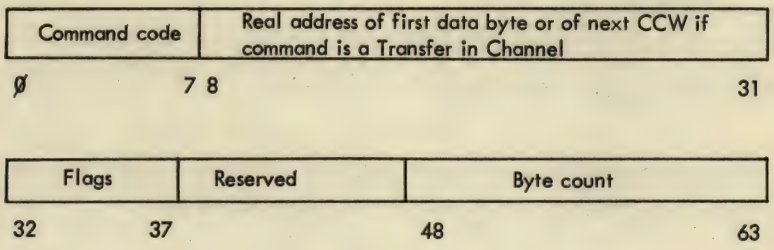
Bits Ø-3 contain the protection key for memory access on reading CCWs and transferring data.

Bits 8-31 contain the real address of the leftmost byte of the initial CCW to be used in the I/O operation. The address is doubleword-aligned.

Before an I/O operation is initiated by the Start Device instruction, the CAW must be generated by the program and stored at a defined location. In the non-extended I/O mode, this is main memory locations $(72)_{10}$ through $(75)_{10}$; in the extended I/O mode, the CAW is stored in general register 11 of the current processor state.

12.4.2. Channel Command Word (CCW)

The Channel Command Word (CCW) supplies the information for controlling the operation of an I/O device. It is 64 bits long and must be aligned on a doubleword boundary. The CCW has the following format :



Bits Ø-7 contain the command code, which specifies the operation to be performed by the I/O device.

Bits 8-31 contain the real address of the first byte in main memory, or if the command is a Transfer in Channel, the address of the next CCW to be executed.

Bits 32-37 contain the flag bits for controlling channel activities during an I/O operation. These comprise the following :

CD - Chain Data flag
CC - Chain Command flag
SLI - Suppress Length Indication flag
SKIP - Skip flag
PCI - Program Controlled Interrupt flag
PC - Page Chaining flag

Bits 48-63 contain the count of the number of bytes to be transferred to or from main memory during an I/O operation (from 1 to 65,536 bytes). An initial count of zero specifies the maximum number of bytes to be transferred.

Notes:

CCWs should not be changed "dynamically". Due to implementation the consequences of such measures are not foreseeable.

12.4.3 Channel Commands

The complete set of channel commands and their command codes (bits 0-7 in the CCW) are listed in the following table.

Command code	Command	
MMMMM1Ø1	Read 1	Input commands
MMMM111Ø	Read 2	
MMMMMØ1Ø	Read Reverse	
MMMMØØØ1	Sense	
MMMMMØ11	Write	Output commands
MMMMM1ØØ	Write Erase	
MMMMM111	Write Control	
XXXX1ØØ1	Transfer in Channel	

Table 12-2 Channel Commands

Bits marked M are modifier bits which designate variations of the channel command specific to particular I/O devices. They have no influence on channel functions.

Bits marked X are don't-care bits.

The program may use only the codes specified in Table 12-2.
The channel checks the command code in the CCW during execution of the Start Device instruction or during command chaining. If the command code is found to be illegal, then a channel program check (CPC) condition exists and the I/O operation is terminated.

If the CCW contains a legal command code but the addressed device is not capable of performing the operation specified (e.g. a read command issued to a printer), the device rejects the command and causes the I/O operation to terminate.

12.4.3.1. Read 1/Read 2

The Read 1 and Read 2 commands transfer information from the specified I/O device to main memory in ascending order.

12.4.3.2. Read Reverse

The Read Reverse command transfers information from the specified I/O device to main memory in descending order. The real address specified by the CCW is the rightmost main memory location for parity. input area.

12.4.3.3. Sense

The Sense command transfers information from the specified device controller to main memory. The information transferred indicates abnormal conditions that have occurred as a result of the last operation performed by the I/O device. The channel executes this command in the same way as a Read 1/Read 2 command except that when a device is connected via the Siemens System 4004 standard I/O interface, a channel data check (CDC) condition cannot occur because the sense bytes are not checked for parity.

12.4.3.4. Write/Write Erase

The Write/Write Erase commands transfer information from main memory to the specified I/O device. If the Write Erase command (applicable only for magnetic tape) is programmed, data is transferred to the device but not written to tape. The tape is erased in accordance with the byte count.

12.4.3.5. Write Control

The Write Control command transfers information from main memory to the specified I/O device. The device controller interprets this data as control information.

12.4.3.6. Transfer in Channel

The Transfer in Channel command provides chaining of CCWs that are not located in adjacent doublewords in main memory. An unconditional branch to the address of the next CCW is made. The branch address (bits 8-31 of the CCW) must specify a doubleword-oriented real main memory address. This command cannot be the first command in a chain. A Transfer in Channel command can address another Transfer in Channel Command. The contents of bits 32-63 are ignored. The flag bits of the preceding command remain effective.

12.4.4. Chain Data (CD)

If the chain data (CD) flag in the CCW is set, this indicates that the following CCW is to be used by the current I/O operation. Data chaining enables an I/O device to receive data from, or transfer data to, non-contiguous areas of main memory, using a single Start Device instruction.

When the CCW has a lapsed byte count and the CD flag is set, the next CCW in sequence is automatically fetched from main memory. The command code of the new CCW is ignored unless it specifies a Transfer in Channel, in which case branching is performed until a CCW is fetched which does not specify a Transfer in Channel. The real memory address (bits 8-31), the new flag bits and the new byte count become valid and replace the former contents of the channel control registers. The current operation is continued using the new values, and the I/O device is unaware that data chaining is taking place. If any of the following channel status byte conditions occur, data chaining is suppressed, the channel program terminates and an interrupt request is made :

- Channel Program Check (CPC)
- Memory Protect Check (MPC)
- Channel Control Check (CCC)
- Channel Data Check (CDC) - if the operation is a Write
- Incorrect Length (INCL)

12.4.5. Chain Command (CC)

If the chain command (CC) flag in the CCW is set, this indicates that on termination of the current I/O operation, a further operation is to be automatically chained by means of a new CCW, and a channel interrupt to inform the program of the termination is not required. Command chaining enables a specific I/O device to perform a series of different operations, using a single Start Device instruction.

When the operation specified by one CCW is completed, and the CC flag is set, the next CCW or the next CCW but one is automatically fetched. If the new command code specifies a Transfer in Channel, branching is performed until a CCW is fetched which does not specify a Transfer in Channel. The new CCW becomes valid and replaces the former contents of the channel control registers. The new operation is performed.

During command chaining, the contents of the standard device byte determine whether the next CCW or the next CCW but one is to be executed. In this way it is possible, in conjunction with Transfer in Channel, to perform conditional branches in the channel program. The branch condition is specified by the device controller.

If any of the following conditions occur, command chaining is suppressed, the channel program terminates and an interrupt request is made :

- Channel Program Check (CPC)
- Memory Protect Check (MPC)
- Channel Data Check (CDC)
- Channel Control Check (CCC)
- Incorrect Length (INCL), i.e. the byte count has not lapsed, and the suppress length indication flag is zero.

Apart from the above conditions, the contents of the standard device byte also influence command chaining. If the CD flag is set, the CC flag is ignored and data chaining is performed.

12.4.6. Suppress Length Indication (SLI)

The suppress length indication (SLI) flag affects execution of command chaining and controls setting of the incorrect length (INCL) flag in the channel status byte.

If the SLI bit is set in the CCW, command chaining can be performed irrespective of the byte count. If the SLI bit is not set, command chaining takes place only if the byte count has lapsed.

The INCL flag in the channel status byte, which is sent to the program during the channel interrupt, is always reset when the SLI bit is set. If the SLI bit is not set, the INCL flag is set if the byte count of the last CCW has not lapsed. In this case, the INCL flag indicates whether as many bytes were transferred as specified in the byte count of the last CCW.

12.4.7. Skip (SKIP)

The skip (SKIP) flag, which can be used only with Read 1, Read 2, Read Reverse, or Sense commands, can be re-specified for each CCW. If the SKIP bit is set, the data transfer to main memory specified by this command is suppressed. Portions of an information block can thus be suppressed during an input operation. The SKIP flag is of no consequence to the I/O device during an I/O operation, but affects only channel operation and suppresses data transfer to main memory.
The real memory address in bits 8-31 is ignored and thus cannot cause a channel program check (CPC) or memory protect check (MPC) condition. Channel data check (CDC) conditions are not reported because data parity is not checked.

12.4.8. Program Controlled Interrupt (PCI)

The program controlled interrupt (PCI) flag offers a way of notifying the program of the progress of an I/O operation by means of a channel interrupt request.

If the PCI bit is set, a request for a program controlled interrupt is generated after the CCW has been fetched from main memory and the first data byte has been transferred over the I/O interface. No interrupt request is generated if the operation is terminated before transfer of the first byte. The PCI bit and the program interrupt it causes do not affect execution of the I/O operation.

12.4.9. Page Chaining (PC)

The page chaining (PC) flag enables the single I/O area in main memory specified by a single CCW to be subdivided into several real 2K pages which need not be contiguous in main memory. The PC bit is only effective when the page chaining switch is set and, in the case of data input, the SKIP flag is reset.

If the PC bit is set and effective, page chaining is performed in the following way: The data transfer specified in the CCW is executed in the normal way until the last data byte of a real 2K page is transferred. This is usually the upper page boundary (bits 21-31 of the memory address are ones); it is the lower boundary for a Read Reverse command (bits 21-31 of the memory address are zeros).

Page chaining is effected before the next byte is transferred to or from memory. Using channel key zero, the channel control unit reads the page chaining table entry assigned to the real page just used. The contents, supplied with additional zeros or ones, form the real address of the next data byte, which is placed in the channel control register.

Page chaining only replaces the actual data address in the control register; the contents of other registers are not affected. Additionally, page chaining applies only to the I/O areas in main memory, but not to the channel programs themselves.

Data chaining and command chaining are not affected by the page chaining mechanism. If the channel program crosses a page boundary, the pages must be adjacent in main memory. Otherwise, a Transfer in Channel command has to be used.

12.5. Channel Interrupts

12.5.1. Channel Interrupt Processing

A channel interrupt is an interruption of the currently executing machine program as the result of an interrupt request from an I/O channel. A channel interrupt request is generated if any of the following four interrupt events is detected in a channel :

- o Termination interrupt
- o Program controlled interrupt
- o Attention interrupt
- o Channel free interrupt

An interrupt request from a channel causes setting of the associated bit in the Interrupt Flag Register (IFR). The central processor services the program interrupt if the corresponding bit is set in the Interrupt Mask Register (IMR) of the current processor state.

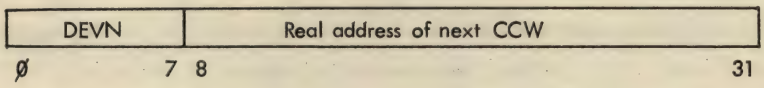
Status information is obtained during handling of an interrupt request in as much as this is necessary for, and relevant to, the current interrupt processing. Depending on the cause of the interrupt, this information relates to the device, channel and/or channel program; it is made available to the program together with the I/O processor number (IOCN), channel number (CHN) and, except for the channel free interrupt, device number (DEVN) in the common set of I/O channel registers (four 32-bit words). In the non-extended I/O mode, the contens of the I/O channel registers are also duplicated in a channel-specific set of registers in the scratch pad. Which of the saved status information is valid in each particular case is determined by the interrupt event. The I/O channel registers are transferred to main memory by the privileged instruction Store Status of Program (SSP) or Store I/O Status (STIO).

12.5.2. I/O Channel Registers

A set of four I/O channel registers is provided, in which status information is stored in the event of channel interrupt :

- o Channel Address Register (CAR)
- o Channel Command Register 1 (CCR1)
- o Channel Command Register 2 (CCR2)
- o Device Status Register (DSR)

12.5.2.1. Channel Address Register (CAR)



Bits Ø-7

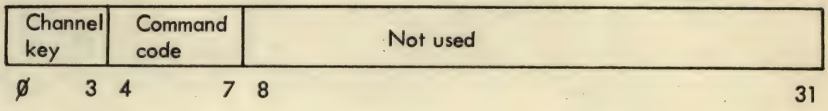
Contain the device number of the interrupting device.

Bits 8-31

Contain the real address of the next CCW to be executed (address of the last (previous) CCW incremented by eight).

12.5.2.2. Channel Command Register 1 (CCR1)

Format :



Bits Ø-3

Contain the channel key.

Bits 4-7

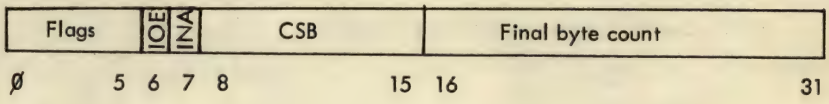
Contain the four low-order bits of the command code.

Bits 8-31

Not used.

12.5.2.3. Channel Command Register 2 (CCR2)

Format :



Bits Ø-5

Contain the flag bits obtained from the last CCW.

Bit 6

Contains the I/O error (IOE) bit.

Bit 7

Contains the I/O processor not available (INA) bit.

Bits 8-15

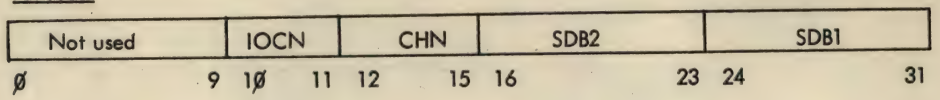
Contain the current channel status byte (CSB).

Bits 16-31

Contain the final byte count of the last CCW.
The difference between the initial byte count and the final byte count is the number of bytes transferred during execution of the last CCW.

12.5.2.4. Device Status Register (DSR)

Format :



Bits Ø-9

Not used.

Bits 1Ø-11

In the extended I/O mode, these bits contain the number of the I/O processor (IOCN); otherwise, they are ignored.

Bits 12-15

Contain the channel number (CHN). In the extended I/O mode, bit 12 contains the most significant bit of the channel number; otherwise, bit 12 is ignored.

Bits 16-23

Contain standard device byte 2 (SDB2).

Bits 24-31

Contain standard device byte 1 (SDB1).

12.5.3. Termination Interrupt

After termination of the operations initiated by the Start Device (SDV) instruction, a termination interrupt request is generated. Termination is possible as follows :

- Normal termination (the channel program goes to completion)
- Occurrence of an abnormal event (conditions arise in the device controller or channel which prevent continuation of the channel program)
- Execution of the Halt Device (HDV) instruction during an I/O operation.

Details regarding the circumstances at termination of an I/O operation are reported in the channel status byte and the standard device byte 1.

12.5.4. Program Controlled Interrupt

A program controlled interrupt request is generated when a CCW is fetched, in which the program controlled interrupt (PCI) flag bit is set. A program controlled interrupt does not affect the current I/O operation. A program controlled interrupt request can only be generated for a busy device.

12.5.5. Attention Interrupt

An attention interrupt request is generated by specific device-dependent conditions within a device controller. An attention interrupt request may not be generated for a busy device.

12.5.6. Channel Free Interrupt

A channel free interrupt request is generated by a block multiplexor channel as the result of a Start Device instruction being executed when the addressed channel is busy in the block multiplex mode. In this case, the Start Device instruction terminates with the condition code set to 2 because of the busy state of the channel. The interrupt requests is generated as soon as the channel becomes not busy.

12.6. I/O Error Handling

12.6.1. Error Detection

I/O errors, i.e. errors which are detected by the error detection facilities in the I/O units, can be classified into two groups :

- Errors which can be assigned to one I/O operation and have no effect on other I/O operations or on operations in the central processors: When these errors occur, the I/O operation is either not initiated or is terminated; the status indicators (see 12.3.2) are set. This procedure depends on the error mode set; further action is machine-dependent.
- Errors which cannot be assigned to one I/O operation :
  Action depends on the error mode set and is machine-dependent.

12.6.2. Error Control Modes

The same error control modes are used as for central processor error handling (see 11.3.1). Mode setting for an I/O processor, however, is not controlled by the Error Control Register alone; depending on the model of central unit, it can also be influenced by control parameters from the I/O processor.

12.6.3. Error Handling

In the quiet mode, machine check interrupt processing mode and logout mode, I/O errors which affect only one I/O operation result in non-initiation or termination of the operation.

In the machine error stop mode, any error results in an I/O error stop.
In the alarm inhibit mode, error reporting is suppressed but the status indicators are set.

Further details regarding error handling are machine-dependent.

12.6.4. I/O Error Interrupt

Retry of incorrectly executed I/O operations is only possible by the operating system. The error is reported to the operating system by an I/O error interrupt request. A bit is set in the Interrupt Flag Register (IFR) and, unless prevented by the corresponding Interrupt Mask Register (IMR) bit, the P4 processor state is entered.

In the logout mode and the quiet mode, error logout is performed for each I/O error interrupt request.

12.6.5. I/O Error Logout

The error logout consists of the standard I/O error word and additional detailed I/O error words, and is stored in main memory. The standard I/O error word has the following format :

Bit Ø

Main memory error

Bit 1

Machine-dependent

Bit 2

I/O error

Bit 3

I/O processor affected
If bit 3=1, the I/O processor is affected; if bit 3=Ø, a trunk or a channel is affected.

Bit 4

Channel affected
If bit 4=1, the channel specified by bits 2Ø-23 is affected, if bit 4=Ø, the I/O processor or a single trunk is affected.

Bit 5

Trunk affected

If bit 5=1, a trunk is affected. Bits 12-15 and 2Ø-23 specify the trunk and channel numbers respectively.

If bit 5=Ø, the I/O processor or the channel is affected.

Bit 6

Device number available

If bit 6=1, bits 24-31 contain the number of the device affected.

Bit 7

Machine-dependent

Bits 8-11

I/O processor identifier

Bits 12-15

Trunk number

The trunk number is valid when bit 5=1.

Bit 16

Machine-dependent

Bits 17-19

Not used

Bits 2Ø-23

Channel number

The channel number is valid when bit 4=1 or bit 5=1.

Bits 24-31

Device number

The device number is valid when bit 6=1.

Appendix 1 Instructions Arranged by Name

Name	Mnemonic	Code	Type	Reference
Add Decimal	AP	FA	SS	10.11. 4
Add Halfword	AH	4A	RX	10.10. 5
Add Logical	ALR	1E	RR	10. 9. 3
Add Logical	AL	5E	RX	10. 9. 3
Add Normalized (extended)	AXR	36	RR	10.12. 7
Add Normalized (long)	ADR	2A	RR	10.12. 6
Add Normalized (long)	AD	6A	RX	10.12. 6
Add Normalized (short)	AER	3A	RR	10.12. 6
Add Normalized (short)	AE	7A	RX	10.12. 6
Add Unnormalized (long)	AWR	2E	RR	10.12. 8
Add Unnormalized (long)	AW	6E	RX	10.12. 8
Add Unnormalized (short)	AUR	3E	RR	10.12. 8
Add Unnormalized (short)	AU	7E	RX	10.12. 8
Add Word	AR	1A	RR	10.10. 4
Add Word	A	5A	RX	10.10. 4
AND	NC	D4	SS	10. 8. 2
AND	NR	14	RR	10. 8. 2
AND	N	54	RX	10. 8. 2
AND	NI	94	SI	10. 8. 2
Branch and Link	BALR	05	RR	10. 7. 2
Branch and Link	BAL	45	RX	10. 7. 2
Branch on Condition	BCR	07	RR	10. 7. 3
Branch on Condition	BC	47	RX	10. 7. 3
Branch on Count	BCTR	06	RR	10. 7. 4
Branch on Count	BCT	46	RX	10. 7. 4
Branch on Index High	BXH	86	RS	10. 7. 5
Branch on Index Low or Equal	BXLE	87	RS	10. 7. 6
Check Channel	CKC	9F	SI	12. 3. 3. 4
Compare (long)	CDR	29	RR	10.12. 9
Compare (long)	CD	69	RX	10.12. 9
Compare (short)	CER	39	RR	10.12. 9
Compare (short)	CE	79	RX	10.12. 9
Compare and Swap	CS	BA	RS	10.15. 2
Compare Decimal	CP	F9	SS	10.11. 5
Compare Double and Swap	CDS	BB	RS	10.15. 1
Compare Halfword	CH	49	RX	10.10. 7
Compare Logical	CLC	D5	SS	10. 9. 4
Compare Logical	CLR	15	RR	10. 9. 4
Compare Logical	CL	55	RX	10. 9. 4
Compare Logical	CLI	95	SI	10. 9. 4
Compare Logical Characters under Mask	CLM	BD	RS	10. 9. 6
Compare Logical Long	CLCL	0F	RR	10. 9. 5
Compare Word	CR	19	RR	10.10. 6
Compare Word	C	59	RX	10.10. 6
Control CPU	CCPU	AC	SI	
Control IOC	CIOC	AD	SI	
Convert to Binary	CVB	4F	RX	10.11. 6
Convert to Decimal	CVD	4E	RX	10.11. 7
Divide	DR	1D	RR	10.10. 8
Divide	D	5D	RX	10.10. 8
Divide (long)	DDR	2D	RR	10.12.10
Divide (long)	DD	6D	RX	10.12.10
Divide (short)	DER	3D	RR	10.12.10
Divide (short)	DE	7D	RX	10.12.10
Divide Decimal	DP	FD	SS	10.11. 8

Name	Mnemonic	Code	Type	Reference
Edit	ED	DE	SS	10.14. 1
Edit and Mark	EDMK	DF	SS	10.14. 2
Exclusive OR	XC	D7	SS	10. 8. 2
Exclusive OR	XR	17	RR	10. 8. 4
Exclusive OR	X	57	RX	10. 8. 4
Exclusive OR	XI	97	SI	10. 8. 4
Execute	EX	44	RX	10.15. 3
Execute Stack	EXST	B1	RS	10.13. 2
Call by Location	CALC	B1 0		10.13. 2. 1
Call by Number	CALN	B1 1		10.13. 2. 2
Move Stack Address	MSAR	B1 X		10.13. 2. 3
Return	RET	B1 3		10.13. 2. 5
Store Multiple in Stack	STMS	B1 2		10.13. 2. 4
Function Call	FCAL	B7	SI	9. 3. 2
Alert CPU	ACPU	B784		9. 3. 2.13
Coordinate CPU	COOP	B786		9. 3. 2.14
Load Halfword Real	LDHR	B781		9. 3. 2.10
Load Segment Table Address and Length	LSAL	B720		9. 3. 2. 1
Load Word Real	LDWR	B780		9. 3. 2. 9
Set Clock	SCK	B767		9. 3. 2. 6
Store CPU Identification	STID	B768		9. 3. 2. 7
Store CPU Number	STNU	B769		9. 3. 2. 8
Store Halfword Real	STHR	B783		9. 3. 2.12
Store Interrupt Flag Register	STIF	B762		9. 3. 2. 3
Store I/O Status	STIO	B763		9. 3. 2. 4
Store Segment Table Address and Length	SSAL	B721		9. 3. 2. 2
Store Word Real	STWR	B782		9. 3. 2.11
Test and Set Real	TSR	B766		9. 3. 2. 5
Trace Virtual Address	TVA	B785		9. 3. 2.15
Halt Device	HDV	9E	SI	12. 3. 3. 2
Halve (long)	HDR	24	RR	10.12.11
Halve (short)	HER	34	RR	10.12.11
Idle	IDL	80	SI	9. 3. 1
Insert Character	IC	43	RX	10. 6. 1
Insert Characters under Mask	ICM	BF	RS	10. 6. 2
Insert Storage Key	ISK	09	RR	9. 3. 5
Load (long)	LDR	28	RR	10.12.13
Load (long)	LD	68	RX	10.12.13
Load (short)	LER	38	RR	10.12.13
Load (short)	LE	78	RX	10.12.13
Load Address	LA	41	RX	10.15. 4
Load and Test	LRT	12	RR	10.10.12
Load and Test (long)	LTDR	22	RR	10.12.17
Load and Test (short)	LTER	32	RR	10.12.17
Load Bit Field	LBF	B6	SI	10. 6. 4
Load Complement	LCR	13	RR	10.10. 9
Load Complement (long)	LCDR	23	RR	10.12.12
Load Complement (short)	LCER	33	RR	10.12.12
Load Halfword	LH	48	RX	10. 6. 5
Load Multiple	LM	98	RS	10. 6. 6
Load Negative	LNR	11	RR	10.10.10
Load Negative (long)	LNDR	21	RR	10.12.14
Load Negative (short)	LNER	31	RR	10.12.14
Load Positive	LPR	10	RR	10.10.11
Load Positive (long)	LPDR	20	RR	10.12.15
Load Positive (short)	LPER	30	RR	10.12.15
Load Rounded (extended to long)	LRDR	25	RR	10.12.16
Load Rounded (long to short)	LRER	35	RR	10.12.16
Load Status of Program	LSP	D8	SS	9. 3. 6
Load Word	LR	18	RR	10. 6. 7
Load Word	L	58	RX	10. 6. 3
Load Word Indirect	LWI	53	RX	10. 6. 8

Name	Mnemonic	Code	Type	Reference
Monitor Call	MC	AF	SI	10.15. 9
Move	MVC	D2	SS	10. 6. 9
Move	MVI	92	SI	10. 6.11
Move Long	MVCL	0E	RR	10. 6.10
Move Numerics	MVN	D1	SS	10. 6.12
Move with Offset	MVO	F1	SS	10. 6.13
Move Zones	MVZ	D3	SS	10. 6.14
Multiply (extended)	MXR	26	RR	10.12.20
Multiply (long)	MDR	2C	RR	10.12.18
Multiply (long)	MD	6C	RX	10.12.18
Multiply (long to extended)	MXDR	27	RR	10.12.19
Multiply (long to extended)	MXD	67	RX	10.12.19
Multiply (short to long)	MER	3C	RR	10.12.18
Multiply (short to long)	ME	7C	RX	10.12.18
Multiply Decimal	MP	FC	SS	10.11. 9
Multiply Halfword	MH	4C	RX	10.10.14
Multiply Word	MR	1C	RR	10.10.13
Multiply Word	M	5C	RX	10.10.13
OR	OC	D6	SS	10. 8. 3
OR	OR	16	RR	10. 8. 3
OR	O	56	RX	10. 8. 3
OR	OI	96	SI	10. 8. 3
Pack	PACK	F2	SS	10.11.10
Pop	POP	98	RS	10.13. 3. 2
Program Control	PC	82	SI	9. 3. 3
Push	PUSH	99	RS	10.13. 3. 1
Set Program Mask	SPM	04	RR	10.15. 5
Set Storage Key	SSK	08	RR	9. 3. 4
Shift and Round Decimal	SRP	F0	SS	10.11.12
Shift Left Double	S LDA	8F	RS	10.11.12
Shift Left Double Logical	SLDL	8D	RS	10. 9. 8
Shift Left Single	SLA	8B	RS	10.10.17
Shift Left Single Logical	SLL	89	RS	10. 9. 9
Shift Right Double	SRDA	8E	RS	10.10.20
Shift Right Double Logical	SRDL	8C	RS	10. 9.10
Shift Right Single	SRA	8A	RS	10.10.18
Shift Right Single Logical	SRL	88	RS	10. 9.11
Start Device	SDV	9C	SI	12. 3. 3. 1
Store (long)	STD	60	RX	10.12.24
Store (short)	STE	70	RX	10.12.24
Store Bit Field	STBF	81	SI	10. 6.16
Store Character	STC	42	RX	10. 6.17
Store Characters under Mask	STCM	BE	RS	10. 6.18
Store Clock	STCK	B2	SI	10.15. 6
Store Halfword	STH	40	RX	10. 6.19
Store Multiple	STM	90	RS	10. 6.20
Store Status of Program	SSP	D0	SS	9. 3. 7
Store Word	ST	50	RX	10. 6.15
Store Word Indirect	STWI	51	RX	10. 6.21
Subtract Decimal	SP	FB	SS	10.11.11
Subtract Halfword	SH	48	RX	10.10.16
Subtract Logical	SLR	1F	RR	10. 9. 7
Subtract Logical	SL	5F	RX	10. 9. 7
Subtract Normalized (extended)	SXR	37	RR	10.12.22
Subtract Normalized (long)	SDR	2B	RR	10.12.21
Subtract Normalized (long)	SD	6B	RX	10.12.21
Subtract Normalized (short)	SER	3B	RR	10.12.21
Subtract Normalized (short)	SE	7B	RX	10.12.21

Name	Mnemonic	Code	Type	Reference
Subtract Unnormalized (long)	SWR	2F	RR	10.12.23
Subtract Unnormalized (long)	SW	6F	RX	10.12.23
Subtract Unnormalized (short)	SUR	3F	RR	10.12.23
Subtract Unnormalized (short)	SU	7F	RX	10.12.23
Subtract Word	SR	1B	RR	10.10.15
Subtract Word	S	5B	RX	10.10.15
Supervisor Call	SVC	0A	RR	10.15. 7
Test and Set	TS	93	SI	10.15. 8
Test Device	TDV	9D	SI	12. 3. 3. 3
Test under Mask	TM	91	SI	10. 8. 5
Translate	TR	DC	SS	10.14. 3
Translate and Test	TRT	DD	SS	10.14. 4
Unpack	UNPK	F3	SS	10.11.13
Zero and Add	ZAP	F8	SS	10.11.14

Appendix 2 Instructions Arranged by Mnemonic

Mnemonic	Code	Type	Name	Reference
A	5A	RX	Add Word	10.10. 4
AD	6A	RX	Add Normalized (long)	10.12. 6
ADR	2A	RR	Add Normalized (long)	10.12. 6
AE	7A	RX	Add Normalized (short)	10.12. 6
AER	3A	RR	Add Normalized (short)	10.12. 6
AH	4A	RX	Add Halfword	10.10. 5
AL	5E	RX	Add Logical	10. 9. 3
ALR	1E	RR	Add Logical	10. 9. 3
AP	FA	SS	Add Decimal	10.11. 4
AR	1A	RR	Add Word	10.10. 4
AU	7E	RX	Add Unnormalized (short)	10.12. 8
AUR	3E	RR	Add Unnormalized (short)	10.12. 8
AW	6E	RX	Add Unnormalized (long)	10.12. 8
AWR	2E	RR	Add Unnormalized (long)	10.12. 8
AXR	36	RR	Add Normalized (extended)	10.12. 7
BAL	45	RX	Branch and Link	10. 7. 2
BALR	05	RR	Branch and Link	10. 7. 2
BC	47	RX	Branch on Condition	10. 7. 3
BCR	07	RR	Branch on Condition	10. 7. 3
BCT	46	RX	Branch on Count	10. 7. 4
BCTR	06	RR	Branch on Count	10. 7. 4
BXH	86	RS	Branch on Index High	10. 7. 5
BXLE	87	RS	Branch on Index Low or Equal	10. 7. 6
C	59	RX	Compare Word	10.10. 6
CCPU	AC	SI	Control CPU	
CD	69	RX	Compare (long)	10.12. 9
CDR	29	RR	Compare (long)	10.12. 9
CDS	BB	RS	Compare Double and Swap	10.15. 1
CE	79	RX	Compare (short)	10.12. 9
CER	39	RR	Compare (short)	10.12. 9
CH	49	RX	Compare Halfword	10.10. 7
CIOC	AD	SI	Control IOC	
CKC	9F	SI	Check Channel	12. 3. 3. 4
CL	55	RX	Compare Logical	10. 9. 4
CLC	D5	SS	Compare Logical	10. 9. 4
CLCL	0F	RR	Compare Logical Long	10. 9. 5
CLI	95	SI	Compare Logical	10. 9. 4
CLM	BD	RS	Compare Logical Characters under Mask	10. 9. 6
CLR	15	RR	Compare Logical	10. 9. 4
CP	F9	SS	Compare Decimal	10.11. 5
CR	19	RR	Compare Word	10.10. 6
CS	BA	RS	Compare and Swap	10.15. 2
CVB	4F	RX	Convert to Binary	10.11. 6
CVD	4E	RX	Convert to Decimal	10.11. 7
D	5D	RX	Divide	10.10. 8
DD	6D	RX	Divide (long)	10.12.10
DDR	2D	RR	Divide (long)	10.12.10
DE	7D	RX	Divide (short)	10.12.10
DER	3D	RR	Divide (short)	10.12.10
DP	FD	SS	Divide Decimal	10.11. 8
DR	1D	RR	Divide	10.10. 8
ED	DE	SS	Edit	10.14. 1
EDMK	DF	SS	Edit and Mark	10.14. 2
EX	44	RX	Execute	10.15. 3
EXST	B1	RS	Execute Stack	10.13. 2
CALC	B1 0		Call by Location	10.13. 2. 1
CALN	B1 1		Call by Number	10.13. 2. 2
MSAR	B1 X		Move Stack Address	10.13. 2. 3
RET	B1 3		Return	10.13. 2. 5
STMS	B1 2		Store Multiple in Stack	10.13. 2. 4

Mnemonic	Code	Type	Name	Reference
FCAL	B7	SI	Function Call	9. 3. 2
ACPU	B784		Alert CPU	9. 3. 2.13
COOP	B786		Coordinate CPU	9. 3. 2.14
LDHR	B781		Load Halfword Real	9. 3. 2.10
LDWR	B780		Load Word Real	9. 3. 2. 9
LSAL	B720		Load Segment Table Address and Length	9. 3. 2. 1
SCK	B767		Set Clock	9. 3. 2. 6
SSAL	B721		Store Segment Table Address and Lenth	9. 3. 2. 2
STHR	B783		Store Halfword Real	9. 3. 2.12
STID	B768		Store CPU Identification	9. 3. 2. 7
STIF	B762		Store Interrupt Flag Register	9. 3. 2. 3
STIO	B763		Store I/O Status	9. 3. 2. 4
STNU	B769		Store CPU number	9. 3. 2. 8
STWR	B782		Store Word Real	9. 3. 2.11
TSR	B766		Test and Set Real	9. 3. 2. 5
TVA	B785		Trace Virtual Address	9. 3. 2.15
HDR	24	RR	Halve (long)	10.12.11
HDV	9E	SI	Halt Device	12. 3. 3. 2
HER	34	RR	Halve (short)	10.12.11
IC	43	RX	Insert Character	10. 6. 1
ICM	BF	RS	Insert Characters under Mask	10. 6. 2
IDL	80	SI	Idle	9. 3. 1
ISK	09	RR	Insert Storage Key	9. 3. 5
L	58	RX	Load Word	10. 6. 3
LA	41	RX	Load Address	10.15. 4
LBF	B6	SI	Load Bit Field	10. 6. 4
LCDR	23	RR	Load Complement (long)	10.12.12
LCER	33	RR	Load Complement (short)	10.12.12
LCR	13	RR	Load Complement	10.10. 9
LD	68	RX	Load (long)	10.12.13
LDR	28	RR	Load (long)	10.12.13
LE	78	RX	Load (short)	10.12.13
LER	38	RR	Load (short)	10.12.13
LH	48	RX	Load Halfword	10. 6. 5
LM	98	RS	Load Multiple	10. 6. 6
LNDR	21	RR	Load Negative (long)	10.12.14
LNER	31	RR	Load Negative (short)	10.12.14
LNR	11	RR	Load Negative	10.10.10
LPDR	20	RR	Load Positive (long)	10.12.15
LPER	30	RR	Load Positive (short)	10.12.15
LPR	10	RR	Load Positive	10.10.11
LR	18	RR	Load Word	10. 6. 7
LRDR	25	RR	Load Rounded (extended to long)	10.12.16
LRER	35	RR	Load Rounded (long to short)	10.12.16
LSP	D8	SS	Load Status of Program	9. 3. 6
LTDR	22	RR	Load and Test (long)	10.12.17
LTER	32	RR	Load and Test (short)	10.12.17
LTR	12	RR	Load and Test	10.10.12
LWI	53	RX	Load Word Indirect	10. 6. 8
M	5C	RX	Multiply Word	10.10.13
MC	AF	SI	Monitor Call	10.15. 9
MD	6C	RX	Multiply (long)	10.12.18
MDR	2C	RR	Multiply (long)	10.12.18
ME	7C	RX	Multiply (short to long)	10.12.18
MER	3C	RR	Multiply (short to long)	10.12.18
MH	4C	RX	Multiply Halfword	10.10.14
MP	FC	SS	Multiply Decimal	10.11. 9
MR	1C	RR	Multiply Word	10.10.13
MVC	D2	SS	Move	10. 6. 9
MVCL	0E	RR	Move Long	10. 6.10
MVI	92	SI	Move	10. 6.11
MVN	D1	SS	Move Numerics	10. 6.12

Mnemonic	Code	Type	Name	Reference
MVO	F1	SS	Move with Offset	10. 6.13
MVZ	D3	SS	Move Zones	10. 6.14
MXD	67	RX	Multiply (long to extended)	10.12.19
MXDR	27	RR	Multiply (long to extended)	10.12.19
MXR	26	RR	Multiply (extended)	10.12.20
N	54	RX	AND	10. 8. 2
NC	D4	SS	AND	10. 8. 2
NI	94	SI	AND	10. 8. 2
NR	14	RR	AND	10. 8. 2
O	56	RX	OR	10. 8. 3
OC	D6	SS	OR	10. 8. 3
OI	96	SI	OR	10. 8. 3
OR	16	RR	OR	10. 8. 3
PACK	F2	SS	Pack	10.11.10
PC	82	SI	Program Control	9. 3. 3
POP	9B	RS	Pop	10.13. 3. 2
PUSH	99	RS	Push	10.13. 3. 1
S	5B	RX	Subtract Word	10.10.15
SD	6B	RX	Subtract Normalized (long)	10.12.21
SDR	2B	RR	Subtract Normalized (long)	10.12.21
SDV	9C	SI	Start Device	12. 3. 3. 1
SE	7B	RX	Subtract Normalized (short)	10.12.21
SER	3B	RR	Subtract Normalized (short)	10.12.21
SH	4B	RX	Subtract Halfword	10.10.16
SL	5F	RX	Subtract Logical	10. 9. 7
SLA	8B	RS	Shift Left Single	10.10.17
SLDA	8F	RS	Shift Left Double	10.10.19
SLDL	8D	RS	Shift Left Double Logical	10. 9. 8
SLL	89	RS	Shift Left Single Logical	10. 9. 9
SLR	1F	RR	Subtract Logical	10. 9. 7
SP	FB	SS	Subtract Decimal	10.11.11
SPM	04	RR	Set Program Mask	10.15. 5
SR	1B	RR	Subtract Word	10.10.15
SRA	8A	RS	Shift Right Single	10.10.18
SRDA	8E	RS	Shift Right Double	10.10.20
SRDL	8C	RS	Shift Right Double Logical	10. 9.10
SRL	88	RS	Shift Right Single Logical	10. 9.11
SRP	F0	SS	Shift and Round Decimal	10.11.12
SSK	08	RR	Set Storage Key	9. 3. 4
SSP	D0	SS	Store Status of Program	9. 3. 7
ST	50	RX	Store Word	10. 6.15
STBF	81	SI	Store Bit Field	10. 6.16
STC	42	RX	Store Character	10. 6.17
STCK	B2	SI	Store Clock	10.15. 6
STCM	BE	RS	Store Characters under Mask	10. 6.18
STD	60	RX	Store (long)	10.12.24
STE	70	RX	Store (short)	10.12.24
STH	40	RX	Store Halfword	10. 6.19
STM	90	RS	Store Multiple	10. 6.20
STWI	51	RX	Store Word Indirect	10. 6.21
SU	7F	RX	Subtract Unnormalized (short)	10.12.23
SUR	3F	RR	Subtract Unnormalized (short)	10.12.23
SVC	0A	RR	Supervisor Call	10.15. 7
SW	6F	RX	Subtract Unnormalized (long)	10.12.23
SWR	2F	RR	Subtract Unnormalized (long)	10.12.23
SXR	37	RR	Subtract Normalized (extended)	10.12.22
TDV	9D	SI	Test Device	12. 3. 3. 3
TM	91	SI	Test under Mask	10. 8. 5
TR	DC	SS	Translate	10.14. 3
TRT	DD	SS	Translate and Test	10.14. 4
TS	93	SI	Test and Set	10.15. 8
UNPK	F3	SS	Unpack	10.11.13

Mnemonic	Code	Type	Name	Reference
X	57	RX	Exclusive OR	10. 8. 4
XC	D7	SS	Exclusive OR	10. 8. 4
XI	97	SI	Exclusive OR	10. 8. 4
XR	17	RR	Exclusive OR	10. 8. 4
ZAP	F8	SS	Zero and Add	10.11.14

Appendix 3 Instructions Arranged by Operation Code

Code	Mnemonic	Type	Name	Reference
04	SPM	RR	Set Program Mask	10.15. 5
05	BALR	RR	Branch and Link	10. 7. 2
06	BCTR	RR	Branch on Count	10. 7. 4
07	BCR	RR	Branch on Condition	10. 7. 3
08	SSK	RR	Set Storage Key	9. 3. 4
09	ISK	RR	Insert Storage Key	9. 3. 5
0A	SVC	RR	Supervisor Call	10.15. 7
0E	MVCL	RR	Move Long	10. 6.10
0F	CLCL	RR	Compare Logical Long	10. 9. 5
10	LPR	RR	Load Positive	10.10.11
11	LNR	RR	Load Negative	10.10.10
12	LTR	RR	Load and Test	10.10.12
13	LCR	RR	Load Complement	10.10. 9
14	NR	RR	AND	10. 8. 9
15	CLR	RR	Compare Logical	10. 9. 4
16	OR	RR	OR	10. 8. 3
17	XR	RR	Exclusive OR	10. 8. 4
18	LR	RR	Load Word	10. 6. 7
19	CR	RR	Compare Word	10.10. 6
1A	AR	RR	Add Word	10.10. 4
1B	SR	RR	Subtract Word	10.10.15
1C	MR	RR	Multiply Word	10.10.13
1D	DR	RR	Divide	10.10. 8
1E	ALR	RR	Add Logical	10. 9. 3
1F	SLR	RR	Subtract Logical	10. 9. 7
20	LPDR	RR	Load Positive (long)	10.12.15
21	LNDR	RR	Load Negative (long)	10.12.14
22	LTDR	RR	Load and Test (long)	10.12.17
23	LCDR	RR	Load Complement (long)	10.12.12
24	HDR	RR	Halve (long)	10.12.11
25	LRDR	RR	Load Rounded (extended to long)	10.12.16
26	MXR	RR	Multiply (extended)	10.12.20
27	MXDR	RR	Multiply (long to extended)	10.12.19
28	LDR	RR	Load (long)	10.12.13
29	CDR	RR	Compare (long)	10.12. 9
2A	ADR	RR	Add Normalized (long)	10.12. 6
2B	SDR	RR	Subtract Normalized (long)	10.12.21
2C	MDR	RR	Multiply (long)	10.12.18
2D	DDR	RR	Divide (long)	10.12.10
2E	AWR	RR	Add Unnormalized (long)	10.12. 8
2F	SWR	RR	Subtract Unnormalized (long)	10.12.23
30	LPER	RR	Load Positive (short)	10.12.15
31	LNER	RR	Load Negative (short)	10.12.14
32	LTER	RR	Load and Test (short)	10.12.17
33	LCER	RR	Load Complement (short)	10.12.12
34	HER	RR	Halve (short)	10.12.11
35	LRER	RR	Load Rounded (long to short)	10.12.16
36	AXR	RR	Add Normalized (extended)	10.12. 7
37	SXR	RR	Subtract Normalized (extended)	10.12.22
38	LER	RR	Load (short)	10.12.13
39	CER	RR	Compare (short)	10.12. 9
3A	AER	RR	Add Normalized (short)	10.12. 6
3B	SER	RR	Subtract Normalized (short)	10.12.21
3C	MER	RR	Multiply (short to long)	10.12.18
3D	DER	RR	Divide (short)	10.12.10
3E	AUR	RR	Add Unnormalized (short)	10.12. 8
3F	SUR	RR	Subtract Unnormalized (short)	10.12.23

Code	Mnemonic	Type	Name	Reference
40	STH	RX	Store Halfword	10. 6.19
41	LA	RX	Load Address	10.15. 4
42	STC	RX	Store Character	10. 6.17
43	IC	RX	Insert Character	10. 6. 1
44	EX	RX	Execute	10.15. 3
45	BAL	RX	Branch and Link	10. 7. 2
46	BCT	RX	Branch on Count	10. 7. 4
47	BC	RX	Branch on Condition	10. 7. 3
48	LH	RX	Load Halfword	10. 6. 5
49	CH	RX	Compare Halfword	10.10. 7
4A	AH	RX	Add Halfword	10.10. 5
4B	SH	RX	Subtract Halfword	10.10.16
4C	MH	RX	Multiply Halfword	10.10.14
4E	CVD	RX	Convert to Decimal	10.11. 7
4F	CVB	RX	Convert to Binary	10.11. 6
50	ST	RX	Store Word	10. 6.15
51	STWI	RX	Store Word Indirect	10. 6.21
53	LWI	RX	Load Word Indirect	10. 6. 8
54	N	RX	AND	10. 8. 2
55	CL	RX	Compare Logical	10. 9. 4
56	O	RX	OR	10. 8. 3
57	X	RX	Exclusive OR	10. 8. 4
58	L	RX	Load Word	10. 6. 3
59	C	RX	Compare Word	10.10. 6
5A	A	RX	Add Word	10.10. 4
5B	S	RX	Subtract Word	10.10.15
5C	M	RX	Mutliply Word	10.10.13
5D	D	RX	Divide	10.10. 8
5E	AL	RX	Add Logical	10. 9. 3
5F	SL	RX	Subtract Logical	10. 9. 7
60	STD	RX	Store (long)	10.12.24
67	MXD	RX	Multiply (long to extended)	10.12.19
68	LD	RX	Load (long)	10.12.13
69	CD	RX	Compare (long)	10.12. 9
6A	AD	RX	Add Normalized (long)	10.12. 6
6B	SD	RX	Subtract Normalized (long)	10.12.21
6C	MD	RX	Multiply (long)	10.12.18
6D	DD	RX	Divide (long)	10.12.10
6E	AW	RX	Add Unnormalized (long)	10.12. 8
6F	SW	RX	Subtract Unnormalized (long)	10.12.23
70	STE	RX	Store (short)	10.12.24
78	LE	RX	Load (short)	10.12.13
79	CE	RX	Compare (short)	10.12. 9
7A	AE	RX	Add Normalized (short)	10.12. 6
7B	SE	RX	Subtract Normalized (short)	10.12.21
7C	ME	RX	Multiply (short to long)	10.12.18
7D	DE	RX	Divide (short)	10.12.10
7E	AU	RX	Add Unnormalized (short)	10.12. 8
7F	SU	RX	Subtract Unnormalized (short)	10.12.23
80	IDL	SI	Idle	9. 3. 1
81	STBF	SI	Store Bit Field	10. 6.16
82	PC	SI	Program Control	9. 3. 3
86	BXH	RS	Branch on Index High	10. 7. 5
87	BXLE	RS	Branch on Index Low or Equal	10. 7. 6
88	SRL	RS	Shift Right Single Logical	10. 9.11
89	SLL	RS	Shift Left Single Logical	10. 9. 9
8A	SRA	RS	Shift Right Single	10.10.18
8B	SLA	RS	Shift Left Single	10.10.17
8C	SRDL	RS	Shift Right Double Logical	10. 9.10

Code	Mnemonic	Type	Name	Reference
8D	SLDL	RS	Shift Left Double Logical	10. 9. 8
8E	SRDA	RS	Shift Right Double	10.10.20
8F	SLDA	RS	Shift Left Double	10.10.19
90	STM	RS	Store Multiple	10. 6.20
91	TM	SI	Test under Mask	10. 8. 5
92	MVI	SI	Move	10. 6.11
93	TS	SI	Test and Set	10.15. 8
94	NI	SI	AND	10. 8. 2
95	CLI	SI	Compare Logical	10. 9. 4
96	OI	SI	OR	10. 8. 3
97	XI	SI	Exclusive OR	10. 8. 4
98	LM	RS	Load Multiple	10. 6. 6
99	PUSH	RS	Push	10.13. 3. 1
9B	POP	RS	Pop	10.13. 3. 2
9C	SDV	SI	Start Device	12. 3. 3. 1
9D	TDV	SI	Test Device	12. 3. 3. 3
9E	HDV	SI	Halt Device	12. 3. 3. 2
9F	CKC	SI	Check Channel	12. 3. 3. 4
AC	CCPU	SI	Control CPU	
AD	CIOC	SI	Control IOC	
AF	MC	SI	Monitor Call	10.15. 9.
B1	EXST	RS	Execute Stack	10.13. 2
B1 X	MSAR		Move Stack Address	10.13. 2. 3
B1 0	CALC		Call by Location	10.13. 2. 1
B1 1	CALN		Call by Number	10.13. 2. 2
B1 2	STMS		Store Multiple in Stack	10.13. 2. 4
B1 3	RET		Return	10.13. 2. 5
B2	STCK	SI	Store Clock	10.15. 6
B6	LBF	SI	Load Bit Field	10. 6. 4
B7	FCAL	SI	Function Call	9. 3. 2
B720	LSAL		Load Segment Table Address and Length	9. 3. 2. 1
B721	SSAL		Store Segment Table Address and Length	9. 3. 2. 2
B762	STIF		Store Interrupt Flag Register	9. 3. 2. 3
B763	STIO		Store I/O Status	9. 3. 2. 4
B766	TSR		Test and Set Real	9. 3. 2. 5
B767	SCK		Set Clock	9. 3. 2. 6
B768	STID		Store CPU Identification	9. 3. 2. 7
B769	STNU		Store CPU Number	9. 3. 2. 8
B780	LDWR		Load Word Real	9. 3. 2. 9
B781	LDHR		Load Halfword Real	9. 3. 2.10
B782	STWR		Store Word Real	9. 3. 2.11
B783	STHR		Store Halfword Real	9. 3. 2.12
B784	ACPU		Alert CPU	9. 3. 2.13
B785	TVA		Trace Virtual Address	9. 3. 2.15
B786	COOP		Coordinate CPU	9. 3. 2.14
BA	CS	RS	Compare and Swap	10.15. 2
BB	CDS	RS	Compare Double and Swap	10.15. 1
BD	CLM	RS	Compare Logical Characters under Mask	10. 9. 6
BE	STCM	RS	Store Characters under Mask	10. 6.18
BF	ICM	RS	Insert Characters under Mask	10. 6. 2
D0	SSP	SS	Store Status of Program	9. 3. 7
D1	MVN	SS	Move Numerics	10. 6.12
D2	MVC	SS	Move	10. 6. 9
D3	MVZ	SS	Move Zones	10. 6.14
D4	NC	SS	AND	10. 8. 2
D5	CLC	SS	Compare Logical	10. 9. 4
D6	OC	SS	OR	10. 8. 3
D7	XC	SS	Exclusive OR	10. 8. 4
D8	LSP	SS	Load Status of Program	9. 3. 6

Code	Mnemonic	Type	Name	Reference
DC	TR	SS	Translate	10.14. 3
DD	TRT	SS	Translate and Test	10.14. 4
DE	ED	SS	Edit	10.14. 1
DF	EDMK	SS	Edit and Mark	10.14. 2
F0	SRP	SS	Shift and Round Decimal	10.11.12
F1	MVO	SS	Move with Offset	10. 6.13
F2	PACK	SS	Pack	10.11.10
F3	UNPK	SS	Unpack	10.11.13
F8	ZAP	SS	Zero and Add	10.11.14
F9	CP	SS	Compare Decimal	10.11. 5
FA	AP	SS	Add Decimal	10.11. 4
FB	SP	SS	Subtract Decimal	10.11.11
FC	MP	SS	Multiply Decimal	10.11. 9
FD	DP	SS	Divide Decimal	10.11. 8

Appendix 4 Table of Instruction-Dependent Condition Codes

Instruction	Condition Code			
	0	1	2	3
Add Decimal	Zero	< Zero	> Zero	Overflow
Add Halfword	Zero	< Zero	> Zero	Overflow
Add Logical	Zero, no carry	Not zero, no carry	Zero, carry	Not zero, carry
Add Normalized	Zero	< Zero	> Zero	-
Add Unnormalized	Zero	< Zero	> Zero	-
Add Word	Zero	< Zero	> Zero	Overflow
AND	Zero	Not zero	-	-
Check Channel	Available	Selector mode attention interrupt	BLMUX busy	Inoperable
Compare (short/long)	Equal	Operand 1 low	Operand 1 high	-
Compare and Swap	Equal	Not equal	-	-
Compare Decimal	Equal	Operand 1 low	Operand 1 high	-
Compare Double and Swap	Equal	Not equal	-	-
Compare Halfword	Equal	Operand 1 low	Operand 1 high	-
Compare Logical	Equal	Operand 1 low	Operand 1 high	-
Compare Logical Long	Equal or field lengths = zero	Operand 1 low	Operand 1 high	-
Compare Logical Characters under Mask	Equal or mask = zero	Operand 1 low	Operand 1 high	-
Compare Word	Equal	Operand 1 low	Operand 1 high	-
Edit	Source field = zero	Source field < zero	Source field > zero	-
Edit and Mark	Source field = zero	Source field < zero	Source field > zero	-
Exclusive OR	Zero	Not zero	-	-
Execute Stack Return	See 10.13.2.5			
Function Call				
Alert CPU	ACPU acknowledged by all processors	ACPU with message not yet processed	No processor addressed	ACPU not acknowledged by all processors
Set Clock	Set	Secure	-	Not operational
Test and Set Real	Zero	One	-	-
Trace Virtual Address	Page defined and in memory	Page not defined or not in memory	Page table cannot be addressed	Virtual address too large

Instruction	Condition Code			
	0	1	2	3
Halt Device	Not busy	CSB and SDB stored	Busy, termination accepted	Inoperable
Insert Characters under Mask	Zero	1st bit = 1	1st bit = Ø	-
Load and Test	Zero	< Zero	> Zero	-
Load and Test (short/long)	Zero	< Zero	> Zero	-
Load Complement	Zero	< Zero	> Zero	Overflow
Load Complement (short/long)	Zero	< Zero	> Zero	-
Load Negative	Zero	< Zero	-	-
Load Negative (short/long)	Zero	< Zero	-	-
Load Positive	Zero	-	> Zero	Overflow
Load Positive (short/long)	Zero	-	> Zero	-
Move Long	Operands 1 and 2 counts equal	Operand 1 count low	Operand 1 count high	Destructive overlap
OR	Zero	Not zero	-	-
Pop	No underflow	Stack underflow	Extent underflow	-
Push	No overflow	Stack overflow	Extent overflow	-
Set Program Mask	According to bits 2 and 3 of the general register specified by R1			
Shift and Round Decimal	Zero	< Zero	> Zero	Overflow
Shift Left Single	Zero	< Zero	> Zero	Overflow
Shift Left Double	Zero	< Zero	> Zero	Overflow
Shift Right Single	Zero	< Zero	> Zero	-
Shift Right Double	Zero	< Zero	> Zero	-
Start Device	I/O initiated	CSB and SDB stored	Busy, or selector mode attention interrupt	Inoperable
Store Clock	Set State	Not-set state	Error state	Not operational
Subtract Decimal	Zero	< Zero	> Zero	Overflow
Subtract Halfword	Zero	< Zero	> Zero	Overflow
Subtract Logical	-	Not zero, no carry	Zero, carry	Not zero, carry
Subtract Normalized (extended)	Zero	< Zero	> Zero	-
Subtract Normalized (short/long)	Zero	< Zero	> Zero	-
Subtract Unnormalized (short/long)	Zero	< Zero	> Zero	-
Subtract Word	Zero	< Zero	> Zero	Overflow
Test and Set	Zero	One	-	-
Test Device	Available	CSB and SDB stored	Busy, or selector mode attention interrupt	Inoperable

Instruction	Condition Code			
	0	1	2	3
Test under Mask	All zero	Mixed	-	All one
Translate and Test	All bytes zero	Non-terminal byte nonzero	Last byte nonzero	-
Zero and Add	Zero	< Zero	> Zero	Overflow

Appendix 5

Powers of 2 Table

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.01562 5
128	7	0.00781 25
256	8	0.00390 625
512	9	0.00195 3125
1,024	10	0.00097 65625
2,048	11	0.00048 82812 5
4,096	12	0.00024 41406 25
8,192	13	0.00012 20703 125
16,384	14	0.00006 10351 5625
32,768	15	0.00003 05175 78125
65,536	16	0.00001 52587 89062 5
131,072	17	0.00000 76293 94531 25
262,144	18	0.00000 38146 97265 625
524,288	19	0.00000 19073 48632 8125
1,048,576	20	0.00000 09536 74316 40625
2,097,152	21	0.00000 04768 37158 20312 5
4,194,304	22	0.00000 02384 18579 10156 25
8,388,608	23	0,00000 01192 09289 55078 125
16,777,216	24	0.00000 00596 04644 77539 0625
33,554,432	25	0.00000 00298 02322 38769 53125
67,108,864	26	0.00000 00149 01161 19384 76562 5
134,217,728	27	0.00000 00074 50580 59692 38281 25
268,435,456	28	0.00000 00037 25290 29846 19140 625
536,870,912	29	0.00000 00018 62645 14923 09570 3125
1,073,741,824	30	0.00000 00009 31322 57461 54785 15625
2,147,483,648	31	0.00000 00004 65661 28730 77392 57812 5
4,294,967,296	32	0.00000 00002 32830 64365 38696 28906 25
8,589,934,592	33	0.00000 00001 16415 32182 69348 14453 125
17,179,869,184	34	0.00000 00000 58207 66091 34674 07226 5625
34,359,738,368	35	0.00000 00000 29103 83045 67337 03613 28125
68,719,476,736	36	0.00000 00000 14551 91522 83668 51806 64062 5
137,438,953,472	37	0.00000 00000 07275 95761 41834 25903 32031 25
274,877,906,944	38	0.00000 00000 03637 97880 70917 12951 66015 625
549,755,813,888	39	0.00000 00000 01818 98940 35458 56475 83007 8125
1,099,511,627,776	40	0.00000 00000 00909 49470 17729 28237 91503 90625
2,199,023,255,552	41	0.00000 00000 00454 74735 08864 64118 95751 95312 5
4,398,046,511,104	42	0.00000 00000 00227 37367 54432 32059 47875 97656 25
8,796,093,022,208	43	0.00000 00000 00113 68683 77216 16029 73937 98828 125
17,592,186,044,416	44	0.00000 00000 00056 84341 88608 08014 86968 99414 0625
35,184,372,088,832	45	0.00000 00000 00028 42170 94304 04007 43484 49707 03125
70,368,744,177,664	46	0.00000 00000 00014 21085 47152 02003 71742 24853 51562 5
140,737,488,355,328	47	0.00000 00000 00007 10542 73576 01001 85871 12426 75781 25
281,474,976,710,656	48	0.00000 00000 00003 55271 36788 00500 92935 56213 37890 625
562,949,953,421,312	49	0.00000 00000 00001 77635 68394 00250 46467 78106 68945 3125
1,125,899,906,842,624	50	0.00000 00000 00000 88817 84197 00125 23233 89053 34472 65625
2,251,799,813,685,248	51	0.00000 00000 00000 44408 92098 50062 61616 94526 67236 32812 5
4,503,599,627,370,496	52	0.00000 00000 00000 22204 46049 25031 30808 47263 33618 16406 25
9,007,199,254,740,992	53	0.00000 00000 00000 11102 23024 62515 65404 23631 66809 08203 125
18,014,398,509,481,984	54	0.00000 00000 00000 05551 11512 31257 82702 11815 83404 54101 5625
36,028,797,018,963,968	55	0.00000 00000 00000 02775 55756 15628 91351 05907 91702 27050 78125
72,057,594,037,927,936	56	0.00000 00000 00000 01387 77878 07814 45675 52953 95851 13525 39062 5
144,115,188,075,855,872	57	0.00000 00000 00000 00693 88939 03907 22837 76476 97925 56762 69531 25
288,230,376,151,711,744	58	0.00000 00000 00000 00346 94469 51953 61418 88238 48962 78381 34765 625
576,460,752,303,423,488	59	0.00000 00000 00000 00173 47234 75976 80709 44119 24481 39190 67382 8125
1,152,921,504,606,846,976	60	0.00000 00000 00000 00086 73617 37988 40354 72059 62240 69595 33691 40625
2,305,843,009,213,693,952	61	0.00000 00000 00000 00043 36808 68994 20177 36029 81120 34797 66845 70312 5
4,611,686,018,427,387,904	62	0.00000 00000 00000 00021 68404 34497 10088 68014 90560 17398 83422 85156 25
9,223,372,036,854,775,808	63	0.00000 00000 00000 00010 84202 17248 55044 34007 45280 08699 41711 42578 125
18,446,744,073,709,551,616	64	0.00000 00000 00000 00005 42101 08624 27522 17003 72640 04349 70855 71289 0625

2^n	n
18,446,744,073,709,551,616	64
36,893,488,147,419,103,232	65
73,786,976,294,838,206,464	66
147,573,952,589,676,412,928	67
295,147,905,179,352,825,856	68
590,295,810,358,705,651,712	69
1,180,591,620,717,411,303,424	70
2,361,183,241,434,822,606,848	71
4,722,366,482,869,645,213,696	72
9,444,732,965,739,290,427,392	73
18,889,465,931,478,580,854,784	74
37,778,931,862,957,161,709,568	75
75,557,863,725,914,323,419,136	76
151,115,727,451,828,646,838,272	77
302,231,454,903,657,293,676,544	78
604,462,909,807,314,587,353,088	79
1,208,925,819,614,629,174,706,176	80
2,417,851,639,229,258,349,412,352	81
4,835,703,278,458,516,698,824,704	82
9,671,406,556,917,033,397,649,408	83
19,342,813,113,834,066,795,298,816	84
38,685,626,227,668,133,590,597,632	85
77,371,252,455,336,267,181,195,264	86
154,742,504,910,672,534,362,390,528	87
309,485,009,821,345,068,724,781,056	88
618,970,019,642,690,137,449,562,112	89
1,237,940,039,285,380,274,899,124,224	90
2,475,880,078,570,760,549,798,248,448	91
4,951,760,157,141,521,099,596,496,896	92
9,903,520,314,283,042,199,192,993,792	93
19,807,040,628,566,084,398,385,987,584	94
39,614,081,257,132,168,796,771,975,168	95
79,228,162,514,264,337,593,543,950,336	96
158,456,325,028,528,675,187,087,900,672	97
316,912,650,057,057,350,374,175,801,344	98
633,825,300,114,114,700,748,351,602,688	99
1,267,650,600,228,229,401,496,703,205,376	100
2,535,301,200,456,458,802,993,406,410,752	101
5,070,602,400,912,917,605,986,812,821,504	102
10,141,204,801,825,835,211,973,625,643,008	103
20,282,409,603,651,670,423,947,251,286,016	104
40,564,819,207,303,340,847,894,502,572,032	105
81,129,638,414,606,681,695,789,005,144,064	106
162,259,276,829,213,363,391,578,010,288,128	107
324,518,553,658,426,726,783,156,020,576,256	108
649,037,107,316,853,453,566,312,041,152,512	109
1,298,074,214,633,706,907,132,624,082,305,024	110
2,596,148,429,267,413,814,265,248,164,610,048	111
5,192,296,858,534,827,628,530,496,329,220,096	112
10,384,593,717,069,655,257,060,992,658,440,192	113
20,769,187,434,139,310,514,121,985,316,880,384	114
41,538,374,868,278,621,028,243,970,633,760,768	115
83,076,749,736,557,242,056,487,941,267,521,536	116
166,153,499,473,114,484,112,975,882,535,043,072	117
332,306,998,946,228,968,225,951,765,070,086,144	118
664,613,997,892,457,936,451,903,530,140,172,288	119
1,329,227,995,784,915,872,903,807,060,280,344,576	120
2,658,455,991,569,831,745,807,614,120,560,689,152	121
5,316,911,983,139,663,491,615,228,241,121,378,304	122
10,633,823,966,279,326,983,230,456,482,242,756,608	123
21,267,647,932,558,653,966,460,912,964,485,513,216	124
42,535,295,865,117,307,932,921,825,928,971,026,432	125
85,070,591,730,234,615,865,843,651,857,942,052,864	126
170,141,183,460,469,231,731,687,303,715,884,105,728	127
340,282,366,920,938,463,463,374,607,431,768,211,456	128

Appendix 6

Hexadecimal and Decimal Conversion Tables

The first table of this appendix provides for direct conversion of hexadecimal and decimal numbers in these ranges :

Hexadecimal	Decimal
ØØØØØ to Ø1FFF	ØØØØØØ to ØØ8191

To convert numbers outside these ranges, and to convert fractions, refer to the hexadecimal and decimal conversion tables and techniques following the direct conversion table.

Direct Conversion Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
0001	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
0002	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
0003	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
0004	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
0005	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
0006	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
0007	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
0008	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
0009	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
000A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
000B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
000C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
000D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
000E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
000F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
0010	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
0011	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
0012	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
0013	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
0014	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
0015	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
0016	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
0017	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
0018	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
0019	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
001A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
001B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
001C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
001D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
001E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
001F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
0020	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
0021	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
0022	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
0023	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
0024	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
0025	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
0026	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
0027	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
0028	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
0029	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
002A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
002B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
002C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
002D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
002E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
002F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
0030	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
0031	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
0032	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
0033	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
0034	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
0035	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
0036	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
0037	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0038	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
0039	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
003A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
003B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
003C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
003D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
003E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
003F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
0040	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
0041	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
0042	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
0043	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
0044	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
0045	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
0046	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
0047	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
0048	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
0049	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
004A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
004B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
004C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
004D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
004E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
004F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
0050	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
0051	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
0052	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
0053	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
0054	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
0055	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
0056	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
0057	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
0058	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
0059	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
005A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
005B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
005C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
005D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
005E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
005F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
0060	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
0061	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
0062	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
0063	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
0064	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
0065	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
0066	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
0067	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
0068	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
0069	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
006A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
006B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
006C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
006D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
006E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
006F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0070	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
0071	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
0072	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
0073	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
0074	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
0075	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
0076	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
0077	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
0078	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
0079	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
007A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
007B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
007C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
007D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
007E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
007F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
0080	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
0081	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
0082	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
0083	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
0084	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
0085	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
0086	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
0087	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
0088	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
0089	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
008A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
008B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
008C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
008D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
008E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
008F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
0090	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
0091	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
0092	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
0093	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
0094	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
0095	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
0096	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
0097	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
0098	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
0099	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
009A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
009B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
009C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
009D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
009E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
009F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
00A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
00A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
00A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
00A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
00A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
00A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
00A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
00A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
00A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
00AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
00AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
00AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
00AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
00AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
00AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
00B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
00B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
00B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
00B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
00B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
00B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
00B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
00B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
00B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
00B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
00BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
00BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
00BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
00BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
00BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
00BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
00C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
00C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
00C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
00C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
00C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	2150	3151
00C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
00C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
00C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
00C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
00C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
00CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
00CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
00CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
00CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
00CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
00CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
00D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
00D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
00D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
00D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
00D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
00D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
00D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
00D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
00D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
00D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
00DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
00DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
00DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
00DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
00DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
00DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
00E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
00E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
00E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
00E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
00E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
00E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
00E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
00E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
00E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
00EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
00EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
00EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
00ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
00EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
00EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
00F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
00F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
00F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3387
00F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
00F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
00F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
00F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
00F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
00F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
00F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
00FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
00FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
00FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
00FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
00FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
00FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0100	004096	004097	004098	004099	004100	004101	004102	004103	004104	004105	004106	004107	004108	004109	004110	004111
0102	004112	004113	004114	004115	004116	004117	004118	004119	004120	004121	004122	004123	004124	004125	004126	004127
0102	004128	004129	004130	004131	004132	004133	004134	004135	004136	004137	004138	004139	004140	004141	004142	004143
0103	004144	004145	004146	004147	004148	004149	004145	004151	004152	004153	004154	004155	004156	004157	004158	004159
0104	004160	004161	004162	004163	004164	004165	004166	004167	004168	004169	004170	004171	004172	004173	004174	004175
0105	004176	004177	004178	004179	004180	004181	004182	004183	004184	004185	004186	004187	004188	004189	004190	004191
0106	004192	004193	004194	004195	004196	004197	004198	004199	0041200	004201	004202	004203	004204	004205	004206	004207
0107	004208	004209	004210	004211	004212	004213	004214	004215	004216	004217	004218	004219	004220	004221	004222	004223
0108	004224	004225	004226	004227	004228	004229	004230	004231	004232	004233	004234	004235	004236	004237	004238	004239
0109	004240	004241	004142	004243	004244	004245	004246	004247	004248	004249	004250	004251	004252	004253	004254	004255
010A	004256	004257	004258	004259	004260	004261	004262	004263	004264	004265	004266	004267	004268	004269	004270	004271
010B	004272	004273	004274	004275	004276	004277	004278	004279	004280	004281	004282	004283	004284	004285	004286	004287
010C	004288	004289	004290	004291	004292	004293	004294	004295	004296	004297	004298	004299	004300	004301	004302	004303
010D	004304	004305	004306	004307	004308	004309	004310	004311	004312	004313	004314	004315	004316	004317	004318	004319
010E	004320	004321	004322	004323	004324	004325	004326	004327	004328	004329	004330	004331	004332	004333	004334	004335
010F	004336	004337	004338	004339	004340	004341	004342	004343	004344	004345	004346	004347	004348	004349	004350	004351
0110	004352	004353	004354	004355	004356	004357	004358	004359	004360	004361	004362	004363	004364	004365	004366	004367
0111	004368	004369	004370	004371	004372	004373	004374	004375	004376	004377	004378	004379	004380	004381	004382	004383
0112	004384	004385	004386	004387	004388	004389	004390	004391	004392	004393	004394	004395	004396	004397	004398	004399
0113	004400	004401	004402	004403	004404	004405	004406	004407	004408	004409	004410	004411	004412	004413	004414	004415
0114	004416	004417	004418	004419	004420	004421	004422	004423	004424	004425	004426	004427	004428	004429	004430	004431
0115	004432	004433	004434	004435	004436	004437	004438	00439	004440	004441	004442	004443	004444	004445	004446	004447
0116	004448	004449	004450	004451	004452	004453	004454	004455	004456	004457	004458	004459	004460	004461	004462	004463
0117	004464	004465	004466	004467	004468	004469	004470	004471	004472	004473	004474	004475	004476	004477	004478	004479
0118	004480	004481	004482	004483	004484	004485	004486	004487	004488	004489	004490	004491	004492	004493	004494	004495
0119	004496	004497	004498	004499	004500	004501	004502	004503	004504	004505	004506	004507	004508	004509	004510	004511
011A	004512	004513	004514	004515	004516	004517	004518	004519	004520	004521	004522	004523	004524	004525	004526	004527
011B	004528	004529	004530	004531	004532	004533	004534	004535	004536	004537	004538	004539	004540	004541	004542	004543
011C	004544	004545	004546	004547	004548	004549	004550	004551	004552	004553	004554	004555	004556	004557	004558	004559
011D	004560	004561	004562	004563	004564	004665	004566	004567	004568	004569	004570	004571	004572	004573	004574	004575
011E	004576	004577	004578	004579	004580	004581	004582	004583	004584	004585	004586	004587	004588	004589	004590	004591
011F	004592	004593	004594	004595	004596	004597	004598	004599	004600	004601	004602	004603	004604	004605	004606	004607
0120	004608	004609	004610	004611	004612	004613	004614	004615	004616	004617	004618	004619	004620	004621	004622	004623
0121	004624	004625	004626	004627	004628	004629	004630	004631	004632	004633	004634	004635	004636	004637	004638	004639
0122	004640	004641	004642	004643	004644	004645	004646	004647	004648	004649	004650	004651	004652	004653	004654	004655
0123	004656	004657	004658	004659	004660	004661	004662	004663	004664	004665	004666	004667	004668	004669	004670	004671
0124	004672	004673	004674	004675	004676	004677	004678	004679	004680	004681	004682	004683	004684	004685	004686	004687
0125	004688	004689	004690	004691	004692	004693	004694	004695	004696	004697	004698	004699	004700	004701	004702	004703
0126	004704	004705	004706	004707	004708	004709	004710	004711	004712	004713	004714	004715	004716	004717	004718	004719
0127	004720	004721	004722	004723	004724	004725	004726	004727	004728	004729	004730	004731	004732	004733	004731	004735
0128	004736	004737	004738	004739	004740	004741	004742	004743	004744	004745	004746	004747	004748	004749	004750	004751
0129	004752	004753	004754	004755	004756	004757	004758	004759	004760	004761	004762	004763	004764	004765	004766	004767
012A	004768	004769	004770	004771	004772	004773	004774	004775	004776	004777	004778	004779	004780	004781	004783	004783
012B	004784	004785	004786	004787	004788	004789	004790	004791	004792	004793	004794	004795	004796	004797	004798	004799
012C	004800	004801	004802	004803	004804	004805	004806	004807	004808	004809	004810	004811	004812	004813	004814	004815
012D	004816	004817	004818	004819	004820	004821	004822	004823	004824	004825	004826	004827	004828	004829	004830	004831
012E	004832	004833	004834	004835	004836	004837	004838	004839	004840	004841	004842	004843	004844	004845	004846	004847
012F	004848	004849	004850	004851	004852	004853	004854	004855	004856	004857	004858	004859	004860	004861	004862	004863
0130	004864	004865	004866	004867	004868	004869	004870	004871	004872	004873	004874	004875	004876	004877	004878	004879
0131	004880	004881	004882	004883	004884	004885	004886	004887	004888	004889	004890	004891	004892	004893	004894	004895
0132	004896	004897	004898	004899	004900	004901	004902	004903	004904	004905	004906	004907	004908	004909	004910	004911
0133	004912	004913	004914	004915	004916	004917	004918	004919	004920	004921	004922	004923	004924	004925	004926	004927
0134	004928	004929	004930	004931	004932	004933	004934	004935	004936	004937	004938	004939	004940	004941	004942	004943
0135	004944	004945	004946	004947	004948	004949	004950	004951	004952	004953	004954	004955	004956	004957	004958	004959
0136	004960	004961	004962	004963	004964	004965	004966	004967	004968	004969	004970	004971	004972	004973	004974	004975
0137	004976	004977	004978	004979	004980	004981	004982	004983	004984	004985	004986	004987	004988	004989	004990	004991

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0138	004992	004993	004994	004995	004996	004997	004998	004999	005000	005001	005002	005003	005004	005005	005006	005007
0139	005008	005009	005010	005011	005012	005013	005014	005015	005016	005017	005018	005019	005020	005021	005022	005023
013A	005024	005025	005026	005027	005028	005029	005030	005031	005032	005033	005034	005035	005036	005037	005038	005039
013B	005040	005041	005042	005043	005044	005045	005046	005047	005048	005049	005050	005051	005052	005053	005054	005055
013C	005056	005057	005058	005059	005060	005061	005062	005063	005064	005065	005066	005067	005068	005069	005070	005071
013D	005072	005073	005074	005075	005076	005077	005078	005079	005080	005081	005082	005083	005084	005085	005086	005087
013E	005088	005089	005090	005091	005092	005093	005094	005095	005096	005097	005098	005099	005100	005101	005102	005103
013F	005104	005105	005106	005107	005108	005109	005110	005111	005112	005113	005114	005115	005116	005117	005118	005119
0140	005120	005121	005122	005123	005124	005125	005126	005127	005128	005129	005130	005131	005132	005133	005134	005135
0141	005136	005137	005138	005139	005140	005141	005142	005143	005144	005145	005146	005147	005148	005149	005150	005151
0142	005152	005153	005154	005155	005156	005157	005158	005159	005160	005161	005162	005163	005164	005165	005166	005167
0143	005168	005169	005170	005171	005172	005173	005174	005175	005176	005177	005178	005179	005180	005181	005182	005183
0144	005184	005185	005186	005187	005188	005189	005190	005191	005192	005193	005194	005195	005196	005197	005198	005199
0145	005200	005201	005202	005203	005204	005205	005206	005207	005208	005209	005210	005211	005212	005213	005214	005215
0146	005216	005217	005218	005219	005220	005221	005222	005223	005224	005225	005226	005227	005228	005229	005230	005231
0147	005232	005233	005234	005235	005236	005237	005238	005239	005240	005241	005242	005243	005244	005245	005246	005247
0148	005248	005249	005250	005251	005252	005253	005254	005255	005256	005257	005258	005259	005260	005261	005262	005263
0149	005264	005265	005266	005267	005268	005269	005270	005271	005272	005273	005274	005275	005276	005277	005278	005279
014A	005280	005281	005282	005283	005284	005285	005286	005287	005288	005289	005290	005291	005292	005293	005294	005295
014B	005296	005297	005298	005299	005300	005301	005302	005303	005304	005305	005306	005307	005308	005309	005310	005311
014C	005312	005313	005314	005315	005316	005317	005318	005319	005320	005321	005322	005323	005324	005325	005326	005327
014D	005328	005329	005330	005331	005332	005333	005334	005335	005336	005337	005338	005339	005340	005341	005342	005343
014E	005344	005345	005346	005347	005348	005349	005350	005351	005352	005353	005354	005355	005356	005357	005358	005359
014F	005360	005361	005362	005363	005364	005365	005366	005367	005368	005369	005370	005371	005372	005373	005374	005375
0150	005376	005377	005378	005379	005380	005381	005382	005383	005384	005385	005386	005387	005388	005389	005390	005391
0151	005392	005393	005394	005395	005396	005397	005398	005399	005400	005401	005402	005403	005404	005405	005406	005407
0152	005408	005409	005410	005411	005412	005413	005414	005415	005416	005417	005418	005419	005420	005421	005422	005423
0153	005424	005425	005426	005427	005428	005429	005430	005431	005432	005433	005434	005435	005436	005437	005428	005439
0154	005440	005441	005442	005443	005444	005445	005446	005447	005448	005449	005450	005451	005452	005453	005454	005455
0155	005456	005457	005458	005459	005460	005461	005462	005463	005464	005465	005466	005467	005468	005469	005470	005471
0156	005472	005473	005474	005475	005476	005477	005478	005479	005480	005481	005482	005483	005484	005485	005486	005487
0157	005488	005489	005490	005491	005492	005493	005494	005495	005496	005497	005498	005499	005500	005501	005502	005503
0158	005504	005505	005506	005507	005508	005509	005510	005511	005512	005513	005514	005515	005516	005517	005518	005519
0159	005520	005521	005522	005523	005524	005525	005526	005527	005528	005529	005530	005531	005532	005533	005534	005535
015A	005536	005537	005538	005539	005540	005541	005542	005543	005544	005545	005546	005547	005548	005549	005550	005551
015B	005552	005553	005554	005555	005556	005557	005558	005559	005560	005561	005562	005563	005564	005565	005566	005567
015C	005568	005569	005570	005571	005572	005573	005574	005575	005576	005577	005578	005579	005580	005581	005582	005583
015D	005584	005585	005586	005587	005588	005589	005590	005591	005592	005593	005594	005595	005596	005597	005598	005599
015E	005600	005601	005602	005603	005604	005605	005606	005607	005608	005609	005610	005611	005612	005613	005614	005615
015F	005616	005617	005618	005619	005620	005621	005622	005623	005624	005625	005626	005627	005628	005629	005630	005631
0160	005632	005633	005634	005635	005636	005637	005638	005639	005640	005641	005642	005643	005644	005645	005646	005647
0161	005648	005649	005650	005651	005652	005653	005654	005655	005656	005657	005658	005659	005660	005661	005662	005663
0162	005664	005665	005666	005667	005668	005669	005670	005671	005672	005673	005674	005675	005676	005677	005678	005679
0163	005680	005681	005682	005683	005684	005685	005686	005687	005688	005689	005690	005691	005692	005693	005694	005695
0164	005696	005697	005698	005699	005700	005701	005702	005703	005704	005705	005706	005707	005708	005709	005710	005711
0165	005712	005713	005714	005715	005716	005717	005718	005719	005720	005721	005722	005723	005724	005725	005726	005727
0166	005728	005729	005730	005731	005732	005733	005734	005735	005736	005737	005738	005739	005740	005741	005742	005743
0167	005744	005745	005746	005747	005748	005749	005750	005751	005752	005753	005754	005755	005756	005757	005758	005759
0168	005760	005761	005762	005763	005764	005765	005766	005767	005768	005769	005770	005771	005772	005773	005774	005775
0169	005776	005777	005778	005779	005780	005781	005782	005783	005784	005785	005786	005787	005788	005789	005790	005791
016A	005792	005793	005794	005795	005796	005797	005798	005799	005800	005801	005802	005803	005804	005805	005806	005807
016B	005808	005809	005810	005811	005812	005813	005814	005815	005816	005817	005818	005819	005820	005821	005822	005823
016C	005824	005825	005826	005827	005828	005829	005830	005831	005832	005833	005834	005835	005836	005837	005838	005839
016D	005840	005841	005842	005843	005844	005845	005846	005847	005848	005849	005850	005851	005852	005853	005854	005855
016E	005856	005857	005858	005859	005860	005861	005862	005863	005864	005865	005866	005867	005868	005869	005870	005871
016F	005872	005873	005874	005875	005876	005877	005878	005879	005880	005881	005882	005883	005884	005885	005886	005887

	0	1	2	3	4	5	6	7	8	9	A	B	C	d	E	F
0170	005888	005889	005890	005891	005892	005893	005894	005895	005896	005897	005898	005899	005900	005901	005902	005903
0171	005904	005905	005906	005907	005908	005909	005910	005911	005912	005913	005914	005915	005916	005917	005918	005919
0172	005920	005921	005922	005923	005924	005925	005926	005927	005928	005929	005930	005931	005932	005933	005934	005935
0173	005936	005937	005938	005939	005940	005941	005942	005943	005944	005945	005946	005947	005948	005949	005550	005551
0174	005952	005953	005954	005955	005956	005957	005958	005959	005960	005961	005962	005963	005964	005965	005966	005967
0175	005968	005969	005970	005971	005972	005973	005974	005975	005976	005977	005978	005979	005980	005981	005982	005983
0176	005984	005985	005986	005987	005988	005989	005990	005991	005992	005993	005994	005995	005996	005997	005998	005999
0177	006000	006001	006002	006003	006004	006005	006006	006007	006008	006009	006010	006011	006012	006013	006014	006015
0178	006016	006017	006018	006019	006020	006021	006022	006023	006024	006025	006026	006027	006028	006029	006030	006031
0179	006032	006033	006034	006035	006036	006037	006038	006039	006040	006041	006042	006043	006044	006045	006046	006047
017A	006048	006049	006050	006051	006052	006053	006054	006055	006056	006057	006058	006059	006060	006061	006062	006063
017B	006064	006065	006066	006067	006068	006069	006070	006071	006072	006073	006074	006075	006076	006077	006078	006079
017C	006080	006081	006082	006083	006084	006085	006086	006087	006088	006089	006090	006091	006092	006903	006094	006095
017D	006096	006097	006098	006099	006100	006101	006102	006103	006104	006105	006106	006107	006108	006109	006110	006111
017E	006112	006113	006114	006115	006116	006117	006118	006119	006120	006121	006122	006123	006124	006125	006126	006127
017F	006128	006129	006130	006131	006132	006133	006134	006135	006136	006137	006138	006139	006140	006141	006142	006143
0180	006144	006145	006146	006147	006148	006149	006150	006151	006152	006153	006154	006155	006156	006157	006158	006159
0181	006160	006161	006162	006163	006164	006165	006166	006167	006168	006169	006170	006171	006172	006173	006174	006175
0182	006176	006177	006178	006179	006180	006181	006182	006183	006184	006185	006186	006187	006188	006189	006190	006191
0183	006192	006193	006194	006195	006196	006197	006198	006199	00200	006201	006202	006203	006204	006205	006206	006207
0184	006208	006209	006210	006211	006212	006213	006214	006215	006216	006217	006218	006219	006220	006221	006222	006223
0185	006224	006225	006226	006227	006228	006229	006230	006231	006232	006233	006234	006235	006236	006237	006238	006239
0186	006240	006241	006242	006243	006244	006245	006246	006247	006248	006249	006250	006251	006252	006253	006254	006255
0187	006256	006257	006258	006259	006260	006261	006262	006263	006264	006265	006266	006267	006268	006269	006270	006271
0188	006272	006273	006274	006275	006276	006277	006278	006279	006280	006281	006282	006283	006284	006285	006286	006287
0189	006288	006289	006290	006291	006292	006293	006294	006295	006296	006297	006298	006299	006300	006301	006302	006303
018A	006304	006305	006306	006307	006308	006309	006310	006311	006312	006313	006314	006315	006316	006317	006318	006319
018B	006320	006321	006322	006323	006324	006325	006326	006327	006328	006329	006330	006331	006332	006333	006334	006335
018C	006336	006337	006338	006339	006340	006341	006342	006343	006344	006345	006346	006347	006348	006349	006350	006351
018D	006352	006353	006354	006355	006356	006357	006358	006359	006360	006361	006362	006363	006364	006365	006366	006367
018E	006368	006369	006370	006371	006372	006373	006374	006375	006376	006377	006378	006379	006380	006381	006382	006383
018F	006384	006385	006386	006387	006388	006389	006390	006391	006392	006393	006394	006395	006396	006397	006398	006399
0190	006400	006401	006402	006403	006404	006405	006406	006407	006408	006409	006410	006411	006412	006413	006414	006415
0191	006416	006417	006418	006419	006420	006421	006422	006423	006424	006425	006426	006427	006428	006429	006430	006431
0192	006432	006433	006434	006435	006436	006437	006438	006439	006440	006441	006442	006443	006444	006445	006446	006447
0193	006448	006449	006450	006451	006452	006453	006454	006455	006456	006457	006458	006459	006460	006461	006462	006463
0194	006464	006465	006466	006467	006468	006469	006470	006471	006472	006473	006474	006475	006476	006477	006478	006479
0195	006480	006481	006482	006483	006484	006485	006486	006487	006488	006489	006490	006491	006492	006493	006494	006495
0196	006496	006497	006498	006499	006500	006501	006502	006503	006504	006505	006506	006507	006508	006509	006510	006511
0197	006512	006513	006514	006515	006516	006517	006518	006519	006520	006521	006522	006523	006524	006525	006526	006527
0198	006528	006529	006530	006531	006532	006533	006534	006535	006536	006537	006538	006539	006540	006541	006542	006543
0199	006544	006545	006546	006547	006548	006549	006550	006551	006552	006553	006554	006555	006556	006557	006558	006559
019A	006560	006561	006562	006563	006564	006565	006566	006567	006568	006569	006570	006571	007572	006573	006574	006575
019B	006576	006577	006578	006579	006580	006581	006582	006583	006584	006585	006586	006587	006588	006589	006590	006591
019C	006592	006593	006594	006595	006596	006597	006598	006599	006600	006601	006602	006603	006604	006605	006606	006607
019D	006608	006609	006610	006611	006612	006613	006614	006615	006616	006617	006618	006619	006620	006621	006622	006623
019E	006624	006625	006626	006627	006628	006629	006630	006631	006632	006633	006634	006635	006636	006637	006638	006639
019F	006640	006641	006642	006643	006644	006645	006646	006647	006648	006649	006650	006651	006652	006653	006654	006655
01A0	006656	006657	006658	006659	006660	006661	006662	006663	006664	006665	006666	006667	006668	006669	006670	006671
01A1	006672	006673	006674	006675	006676	006677	006678	006679	006680	006681	006682	006683	006684	006685	006686	006687
01A2	006688	006689	006690	006691	006692	006693	006694	006695	006696	006697	006698	006699	006700	006701	006702	006703
01A3	006704	006705	006706	006707	006708	006709	006710	006711	006712	006713	006714	006715	006716	006717	006718	006719
01A4	006720	006721	006722	006723	006724	006725	006726	006727	006728	006729	006730	006731	006732	006733	006734	006735
01A5	006736	006737	006738	006739	006740	006741	006742	006743	006744	006745	006746	006747	006748	006749	006750	006751
01A6	006752	006753	006754	006755	006756	006757	006758	006759	006760	006761	006762	006763	006764	006765	006766	006767
01A7	006768	006769	006770	006771	006772	006773	006774	006775	006776	006777	006778	006779	006780	006781	060872	006783

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01A8	006784	006785	006786	006787	006788	006789	006790	006791	006792	006793	006794	006795	006796	006797	006798	006799
01A9	006800	006801	006802	006803	006804	006805	006806	006807	006808	006809	006810	006811	006812	006813	006814	006815
01AA	006816	006817	006818	006819	006820	006821	006822	006823	006824	006825	006826	006827	006828	006829	006830	006831
01AB	006832	006833	006834	006835	006836	006837	006838	006839	006840	006841	006842	006843	006844	006845	006846	006847
01AC	006848	006849	006850	006851	006852	006853	006854	006855	006856	006857	006858	006859	006860	006861	006862	006863
01AD	006864	006865	006866	006867	006868	006869	006870	006871	006872	006873	006874	006875	006876	006877	006878	006879
01AE	006880	006881	006882	006883	006884	006885	006886	006887	006888	006889	006890	006891	006892	006893	006894	006895
01AF	006896	006897	006898	006899	006900	006901	006902	006903	006904	006905	006906	006907	006908	006909	006910	006911
01B0	006912	006913	006914	006915	006916	006917	006918	006919	006920	006921	006922	006923	006924	006925	006926	006927
01B1	006928	006929	006930	006931	006932	006933	006934	006935	006936	006937	006938	006939	006940	006941	006942	006943
01B2	006944	006945	006946	006947	006948	006849	006950	006951	006952	006953	006954	006955	006956	006957	006958	006959
01B3	006960	006961	006962	006963	006964	006965	006966	006967	006968	006969	006970	006971	006972	006973	006974	006975
01B4	006976	006977	006978	006979	006980	006981	006982	006983	006984	006985	006986	006987	006988	006989	006990	006991
01B5	006992	006993	006994	006995	006996	006997	006998	006999	007000	007001	007002	007003	007004	007005	007006	007007
01B6	007008	007009	007010	007011	007012	007013	007014	007015	007016	007017	007018	007019	007020	007021	007022	007023
01B7	007024	007025	007026	007027	007028	007029	007030	007031	007032	007033	007034	007035	007036	007037	007038	007039
01B8	007040	007041	007042	007043	007044	007045	007046	007047	007048	007049	007050	007051	007052	007053	007054	007055
01B9	007056	007057	007058	007059	007060	007061	007062	007063	007064	007065	007066	007067	007068	007069	007070	007071
01BA	007072	007073	007074	007075	007076	007077	007078	007079	007080	007081	007082	007083	007084	007085	007086	007087
01BB	007088	007089	007090	007091	007092	007093	007094	007095	007096	008097	007098	007099	007100	007101	007102	007103
01BC	007104	007105	007106	007107	007108	007109	007110	007111	007112	007113	007114	007115	007116	007117	007118	007119
01BD	007120	007121	007122	007123	007124	007125	007126	007127	007128	007129	007130	007131	007132	007133	007134	007135
01BE	007136	007137	007138	007139	007140	007141	007142	007143	007144	007145	007146	007147	007148	007149	007150	007151
01BF	007152	007153	007154	007155	007156	007157	007153	007159	007160	007161	007162	007163	007164	007165	007166	007167
01C0	007168	007169	007170	007171	007172	007173	007174	007175	007176	007177	007178	007179	007180	007181	007182	007183
01C1	007184	007185	007186	007187	007188	007189	007190	007191	007192	007193	007194	007195	007196	007197	007198	007199
01C2	007200	007201	007203	007203	007204	007205	007206	007207	007208	007209	007210	007211	007212	007213	007214	007215
01C3	007216	007217	007218	007219	007220	007221	007222	007223	007224	007225	007226	007227	007228	007229	007230	007231
01C4	007232	007333	007234	007235	007236	007237	007238	007239	007240	007241	007242	007243	007244	007245	007246	007247
01C5	007248	007249	007250	007251	007252	007253	007254	007255	007256	007257	007258	007259	007260	007261	007262	007263
01C6	007264	007265	007266	007267	007268	007269	007270	007271	007272	007273	007274	007275	007276	007277	007278	007279
01C7	007280	007281	007282	007283	007284	007285	007286	007287	007288	007289	007290	007291	007292	007293	007294	007295
01C8	007296	007297	007298	007299	007300	007301	007302	007303	007304	007305	007306	007307	007308	007309	007310	007311
01C9	007312	007313	007314	007315	007316	007317	007318	007319	007320	007321	007322	007323	007324	007325	007326	007327
01CA	007328	007329	007330	007331	007332	007333	007334	007335	007336	007337	007338	007339	007340	007341	007342	007343
01CB	007344	007345	007346	007347	007348	007349	007350	007351	007352	007353	007354	007355	007356	007357	007358	007359
01CC	007360	007361	007362	007363	007364	007365	007366	007367	007368	007369	007370	007371	007372	007373	007374	007375
01CD	007376	007377	007378	007379	007380	007381	007382	007383	007384	007385	007386	007387	007388	007389	007390	007391
01CE	007392	007393	007394	007395	007396	007397	007398	007399	007300	007401	007402	007403	007404	007405	007406	007407
01CF	007408	007409	007410	007411	007412	007413	007414	007415	007416	007417	007418	007419	007420	007421	007422	007423
01D0	007424	007425	007426	007427	007428	007429	007430	007431	007432	007433	007434	007435	007436	007437	007438	007439
01D1	007440	007441	007442	007443	007444	007445	007446	007447	007448	007449	007450	007451	007452	007453	007454	007455
01D2	007456	007457	007458	007459	007460	007461	007462	007463	007464	007465	007466	007467	007468	007469	007470	007471
01D3	007472	007473	007474	007475	007476	007477	007478	007479	007480	007481	007482	007483	007484	007485	007486	007487
01D4	007488	007489	007490	007491	007492	007493	007494	007495	007496	007497	007498	007499	007500	007501	007502	007503
01D5	007504	007505	007506	007507	007508	007509	007510	007511	007512	007513	007514	007515	007516	007517	007518	007519
01D6	007520	007521	007522	007523	007524	007525	007526	007527	007528	007529	007530	007531	007532	007533	007534	007535
01D7	007536	007537	007538	007539	007540	007541	007542	007543	007544	007545	007546	007547	007548	007549	007550	007551
01D8	007552	007553	007554	007555	007556	007557	007558	007559	007560	007561	007562	007563	007564	007565	007566	007567
01D9	007568	007569	007570	007571	007572	007573	007574	007575	007576	007577	007578	007579	007580	007581	007582	007583
01DA	007584	007585	007586	007587	007588	007589	007590	007591	007592	007593	007594	007595	007596	007597	007598	007599
01DB	007600	007601	007602	007603	007604	007605	007606	007607	007608	007609	007610	007611	007612	007613	007614	007615
01DC	007616	007617	007618	007619	007620	007621	007622	007623	007624	007625	007626	007627	007628	007629	007630	007631
01DD	007632	007633	007634	007635	007636	007637	007638	007639	007640	007641	007642	007643	007644	007645	007646	007647
01DE	007648	007649	007650	007651	007652	007653	007654	007655	006656	007657	007658	007659	007660	007661	007662	007663
01DF	007664	007665	007666	007667	007668	006779	007670	007671	007672	007673	007674	007675	007676	007677	007678	007679

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01E0	007680	007681	007682	007683	007684	007685	007686	007687	007688	007689	007690	007691	007692	007693	007694	007695
01E1	007696	007697	007698	007699	007700	007701	007702	007703	007704	007705	007706	007707	007708	007709	007710	007711
01E2	007712	007713	007714	007715	007716	007717	007718	007719	007720	007721	007722	007723	007724	007725	007726	007727
01E3	007728	007729	007730	007731	007732	007733	007734	007735	007736	007737	007738	007739	007740	007741	007742	007743
01E4	007744	007745	007746	007747	007748	007749	007750	007751	007752	007753	007754	007755	007756	007757	007758	007759
01E5	007760	007761	007762	007763	007764	007765	007766	007767	007768	007769	007770	007771	007772	007773	007774	007775
01E6	007776	007777	007778	007779	007780	007781	007782	007783	007784	007785	007786	007787	007788	007789	007790	007791
01E7	007792	007793	007794	007795	007796	007797	007798	007799	007800	007801	007802	007803	007804	007805	007806	007807
01E8	007808	007809	007810	007811	007812	007813	007814	007815	007816	007817	007818	007819	007820	007821	007822	007823
01E9	007824	007825	007826	007827	007828	007829	007830	007831	007832	007833	007834	007835	007836	007837	007837	007839
01EA	007840	007841	007842	007843	007844	007845	007846	007847	007848	007849	007850	007851	007852	007853	007854	007855
01EB	007856	007857	007858	007859	007860	007861	007862	007863	007864	007865	007866	007867	007868	007869	007870	007871
01EC	007872	007873	007874	007875	007876	007877	007878	007879	007880	007881	007882	007883	007884	007885	007886	007887
01ED	007888	007889	007890	007891	007892	007893	007894	007895	007896	007897	007898	007899	007900	007901	007902	007903
01EE	007904	007905	007906	007907	007908	007909	007910	007911	007912	007913	007914	007915	007916	007917	007918	007919
01EF	007920	007921	007922	007923	007924	007925	007926	007927	007928	007929	007930	007931	007932	007933	007934	007935
01F0	007936	007937	007938	007939	007940	007941	007942	007943	007944	007945	007946	007947	007948	007949	007950	007951
01F1	007952	007953	007954	007955	007956	007957	007958	007959	007960	007961	007962	007963	007964	007965	007966	007967
01F2	007968	007969	007970	007971	007972	007973	007974	007975	007976	007977	007978	007979	007980	007981	007982	007983
01F3	007984	007985	007986	007987	007988	007989	007990	007991	007992	007993	007994	007995	007996	007997	007998	007999
01F4	008000	008001	008002	008003	008004	008005	008006	008007	008008	008009	008010	008011	008012	008013	008014	008015
01F5	008016	008017	008018	008019	008020	008021	008022	008023	008024	008025	008026	008027	008028	008029	008030	008031
01F6	008032	008033	008034	008035	008036	008037	008038	008039	008040	008041	008042	008043	008044	008045	008046	008047
01F7	008048	008049	008050	008051	008052	008053	008054	008055	008056	008057	008058	008059	008060	008061	008062	008063
01F8	008064	008065	008066	008067	008068	008069	008070	008071	008072	008073	008074	008075	008076	008077	008078	008079
01F9	008080	008081	008082	008083	008084	008085	008086	008087	008088	008089	008090	008091	008092	008093	008094	008095
01FA	008096	008097	008098	008099	008100	008101	008102	008103	008104	008105	008106	008107	008108	008109	008110	008111
01FB	008112	008113	008114	008115	008116	008117	008118	008119	008120	008121	008122	008123	008124	008125	008126	008127
01FC	008128	008129	008130	008131	008132	008133	008134	008135	008136	008137	008138	008139	008140	008141	008142	008143
01FD	008144	008145	008146	008147	008148	008149	008150	008151	008152	008153	008154	008155	008156	008157	008158	008159
01FE	008160	008161	008162	008163	008164	008165	008166	008167	008168	008169	008170	008171	008172	008173	008174	008175
01FF	008176	008177	008178	008179	008180	008181	008182	008183	008184	008185	008186	008187	008188	008189	008190	008191

Conversion Table: Hexadecimal and Decimal Integers

Halfword								Halfword							
Byte				Byte				Byte				Byte			
0123		4567		0123		4567		0123		4567		0123		4567	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268.435.456	1	16.777.216	1	1.048.576	1	65.536	1	4.096	1	256	1	16	1	1
2	536.870.912	2	33.554.432	2	2.097.152	2	131.072	2	8.192	2	512	2	32	2	2
3	605.306.368	3	50.331.648	3	3.145.723	3	196.608	3	12.288	3	768	3	48	3	3
4	1.073.741.824	4	67.108.864	4	4.194.304	4	262.144	4	16.384	4	1.024	4	64	4	4
5	1.342.177.280	5	83.886.080	5	5.242.880	5	327.680	5	20.480	5	1.280	5	80	5	5
6	1.610.612.736	6	100.663.296	6	6.291.456	6	393.216	6	24.576	6	1.536	6	96	6	6
7	1.879.048.192	7	117.440.512	7	7.340.032	7	458.752	7	28.672	7	1.792	7	112	7	7
8	2.147.483.648	8	134.217.728	8	8.388.608	8	524.288	8	32.768	8	2.048	8	128	8	8
9	2.415.919.104	9	150.994.944	9	9.437.184	9	589.824	9	36.864	9	2.304	9	144	9	9
A	2.684.354.560	A	167.772.160	A	10.485.760	A	655.360	A	40.960	A	2.560	A	160	A	10
B	2.952.790.016	B	184.549.376	B	11.534.336	B	720.896	B	45.056	A	2.816	B	176	B	11
C	3.221.225.472	C	201.326.592	C	12.582.912	C	786.432	C	49.152	C	3.072	C	192	C	12
D	3.489.660.928	D	218.103.803	D	13.631.488	D	851.968	D	53.248	D	3.328	D	208	D	13
E	3.758.096.384	E	234.881.024	E	14.680.064	E	917.504	E	57.344	E	3.584	E	224	E	14
F	4.026.531.840	F	251.658.240	F	15.728.640	F	983.040	F	61.440	F	3.840	F	240	F	15
8		7		6		5		4		3		2		1	

Powers of 16 Table

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10 = A
17 592 186 044 416	11 = B
281 474 976 710 656	12 = C
4 503 599 627 370 496	13 = D
72 057 594 037 927 936	14 = E
1 152 921 504 606 846 976	15 = F

Decimal valves

Hexadecimal to Decimal Conversion :

1. Commencing with the rightmost position of the hexadecimal, locate and record the equivalent decimal number from the corresponding column of the table.
2. Repeat step 1 for the adjacent hexadecimal positions to the left to obtain the equivalent decimal numbers.
3. Add the numbers selected from the table to form the decimal equivalent of the hexadecimal number.

Example :

Conversion of the hexadecimal number D34

4	≙	4
3	≙	48
D	≙	3328
Decimal		3380

Integer numbers outside the range of the table are converted as follows :

Hexadecimal to decimal :

Successive cumulative multiplication from left to right, adding units position.

Example:

$D34_{(16)} = 3380_{(10)}$

D = 13

$\times 16$

208

3 = + 3

211

$\times 16$

3376

4 = + 4

3380

Decimal to Hexadecimal Conversion :

1. Select from the table the highest decimal number that is equal to or less than the number to be converted.
2. Record the equivalent hexadecimal digit, noting its significance.
3. Subtract the selected decimal number from the number to be converted.
4. Repeat steps 1 to 3 to convert each remainder obtained.
5. When the remainder is zero, combine the terms to form the hexadecimal equivalent of the decimal number.

Example :

Conversion of the decimal number 3380

3380		
- 3328	≙	D
52		
- 48	≙	3
4		
- 4	≙	4
0		D34

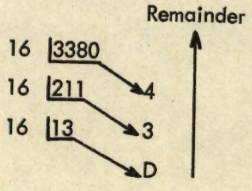
Integer numbers outside the range of the table are converted as follows :

Decimal to hexadecimal :

Divide by 16 and collect the remainder in reverse order.

Example :

$3380_{(10)} = D34_{(16)}$



Conversion Table: Hexadecimal and Decimal Fractions

Byte				Haltwort	Byte			
Bits 0123			4567		0123			4567
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	
0.0	0.0000	0.00	0.0000 0000	0.000	0.0000 0000 0000	0.0000	0.0000 0000 0000 0000	
0.1	0.0625	0.01	0.0039 0625	0,001	0,0002 4414 0625	0.0001	0.0000 1525 8789 0625	
0,2	0.1250	0.02	0.0078 1250	0.002	0.0004 8828 1250	0.0002	0.0000 3051 7578 1250	
0.3	0.1875	0.03	0.0117 1875	0.003	0.0007 3242 1875	0.0003	0.0000 4577 6367 1875	
0.4	0.2500	0.04	0.0156 2500	0.004	0.0009 7656 2500	0.0004	0.0000 6108 5156 2500	
0.5	0.3125	0.05	0.0195 3125	0.005	0.0012 2070 3125	0.0005	0.0000 7629 3945 3125	
0.6	0.3750	0.06	0.0234 3750	0.006	0.0014 6184 3750	0.0006	0.0000 9155 2734 3750	
0.7	0.4375	0.07	0.0273 4375	0.007	0.0017 0898 4375	0.0007	0.0001 0681 1523 4375	
0.8	0.5000	0.08	0.0312 5000	0.008	0.0019 5312 5000	0.0008	0.0001 2207 0312 5000	
0.9	0.5625	0.09	0.0351 5625	0.009	0.0021 9726 5625	0.0009	0.0001 3732 9101 5625	
0.A	0.6250	0.0A	0.0390 6250	0.00A	0.0024 4140 6250	0.000A	0.0001 5258 7890 6250	
0.B	0.6875	0.0B	0.0429 6875	0.00B	0.0026 8554 6875	0.000B	0.0001 6784 6679 6875	
0.C	0.7500	0.0C	0.0468 7500	0.00C	0.0029 2968 7500	0.000C	0.0001 8310 5468 7500	
0.D	0.8125	0.0D	0.0507 5125	0.00D	0.0031 7382 8125	0.000D	0.0001 9836 4257 8125	
0.E	0.8750	0.0E	0.0546 8750	0.00E	0.0034 1796 8750	0.000E	0.0002 1362 3046 8750	
0.F	0.9375	0.0F	0.0585 9375	0.00F	0.0036 6210 9375	0,000F	0.0002 2888 1835 9372	
	1		2		3		4	

Conversion of 0.ABC Hexadecimal to Decimal :

Find 0.A in column 1 = Ø.625Ø

Find 0.0B in column 2 = Ø.Ø429 6875

Find 0.00C in column 3 = Ø.ØØ29 2968 75ØØ

$0.ABC_{(16)}$ equals the total Ø.6708 9843 75ØØ$_{(1Ø)}$

Conversion of Decimal Ø.13 to Hexadecimal :

1. Find Ø.1250 next lowest to	Ø.13ØØ	
and subtract	- Ø.125Ø	≙ $Ø.2_{(16)}$
2. Find Ø.ØØ39 Ø625 next lowest to	Ø.ØØ5Ø ØØØØ	
	- Ø.ØØ39 Ø625	≙ $Ø.Ø1_{(16)}$
3. Find Ø.ØØØ9 7656 25ØØ	Ø.ØØ1Ø 9375 ØØØØ	
	- Ø.ØØØ9 7656 25ØØ	≙ $Ø.ØØ4_{(16)}$
4. Find Ø.ØØØ1 Ø681 1523 4375	Ø.ØØØ1 1718 75ØØ ØØØØ	
	- Ø.ØØØ1 Ø681 1523 4375	≙ $Ø.ØØØ7_{(16)}$
	Ø.ØØØØ 1Ø37 5976 5625	$0.2147_{(16)}$

Ø.13 decimal is approximately equal to Ø.2147 hexadecimal.

Fractions outside the range of the table are converted as follows :

Hexadecimal fraction to decimal :

To convert a hexadecimal fraction to its decimal equivalent, use the same technique as for integers. Divide the result by 16^n (n being the number of fraction positions).

Example :

$Ø.8A7_{(16)} = Ø.54Ø771_{(10)}$

$8A7_{(16)} = 2215_{(10)}$

$16^3 = 4Ø96$

```
          Ø.54Ø771
4Ø96 ⌈2215.ØØØØØØ
```

Decimal fraction to hexadecimal :

Collect the integer parts of the products in the order of calculation.

Example :

$Ø.54Ø8_{(1Ø)} = Ø.8A7_{(16)}$

```
          Ø.54Ø8
        x 16
8 <-----8.6528
        x 16
A <-----1Ø.4448
        x 16
7 <-----7.1168
```