File No. S360-01
Form A22-6943-0

IBM Systems Reference Library

IBM System/360 Model 195

Functional Characteristics

This publication describes the organization and functional characteristics of the IBM System/360 Model 195, an information-processing system designed for ultrahigh-speed, large-scale scientific applications.

System components are described, and detailed consideration is given to the functions of processor storage, central processing unit, input/output channels, and operator-control and operator-intervention portions of the system control panel. Coding and timing considerations are discussed.

The reader is assumed to have a knowledge of information-processing systems and to have read the *IBM System/360 Principles of Operation,* Form A22-6821.

First Edition (August, 1969)
Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, refer to the latest System/360 SRL Newsletter, Form N20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

This manual has been prepared by the IBM Systems Development Division, Product Publications, Dept. B98, PO Box 390, Poughkeepsie, N.Y. 12602. A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be sent to the above address.

Contents

The IBM System/360 Model 195 is an information-processing system designed for ultrahigh-speed, large-scale computer applications. Its speed and power result primarily from advanced circuit technology, a high performance buffer for processor storage accesses, buffering within the processor, very fast execution times, a high degree of concurrency in operations, and employment of exceptionally efficient algorithms, particularly in floating-point operations.

Speed in accessing storage and in executing instructions is achieved with a high-speed buffer storage and multiple, interleaved processor storage elements, by functional buffering within the processor, and by an assembly-line approach to instruction processing. All of these factors are controlled to maintain a high degree of concurrent, continuous operation in the instruction unit and in several execution units. A unique internal bus system also plays a major role.

In the Model 195, five separate units – each to a large degree autonomous – may be operating concurrently: processor storage, storage control unit and buffer storage, instruction processor, fixed-point/VFL/decimal processor, and floating-point processor. Furthermore, each of these units may be performing several functions at one time. In the floating-point processor, for example, as many as three floating-point operations may be taking place concurrently.

Because of the concurrency achieved in the Model 195, the effective time required by a given instruction is not directly related to the rate at which that instruction can be processed. For example, one normalized floating-point-add operation requires two cycles and one normalized floating-point-multiply operation requires three cycles; if the operations are logically independent, it is possible in the Model 195 to process up to two add and one multiply instructions concurrently for a total of three cycles, rather than sequentially for a total of seven.

Although central processing unit (CPU) operations are to a high degree performed in parallel, no special optimization is required in preparing programs for CPU processing. In general, System/360 coding is processed in the CPU with a high degree of efficiency. Using the interrupt mechanism as a part of the problem program logic should be avoided. Although this use of interrupts applies reasonably well to slower, serial CPU's, such use degrades higher performance CPU's. In particular, certain program interrupts that occur at the end of a particular "assembly line" are too late to act as modifiers to the beginning of that line. This situation restricts the user from taking unique, in-line action based on exceptions like floating-point overflow.

Another consequence of the high-performance design is a recommendation (not a functional requirement) that FORTRAN users arrange arrays in COMMON so that long precision data precedes short-precision data, etc. This ensures that the data does not need to be aligned at execution time. (See *IBM System/360 FORTRAN IV Language,* Form C28-6515.)

Model 195 machine cycle time is 54 nanoseconds; data flow is eight bytes (one doubleword) in parallel. The processor storage cycle time is 756 nanoseconds, and the buffer storage cycle time for successive read or successive write cycles is 54 nanoseconds. (Depending on the addressing pattern, an occasional Write followed by a Read may encounter a blank cycle.)

Monolithic circuitry is used in the Model 195. The advanced circuits have a basic delay time of less than 5 nanoseconds, compared to SLT delay times ranging from 5 to 30 nanoseconds. In packaging, densities many times that of SLT have been achieved. Boards approximately 8 by 12 inches can hold pluggable cards containing up to 4,000 circuits. Two of these boards can contain a floating-point-add execution unit for 64 bits in which both preshifting and postshifting are accomplished.

The buffer storage, also in monolithic technology, has a 54-nanosecond read cycle with an eight-byte data path. The buffer storage capacity of 32,768 bytes is packaged, using pluggable cards, on two 10- by 12-inch boards. Storage circuits lend themselves to much denser packaging techniques, with one board containing as many as 150,000 circuits.

Relationship to Other Models of IBM System/360

Because of the emphasis on high performance, the operation of the Model 195 in the following cases differs from that specified in the *IBM System/360 Principles of Operation,* Form A22-6821.

1. The quotient of a floating-point-divide operation may differ in the Model 195 from that of other models by an amount equal to one bit in the low-order fraction position. For zero remainders, the results are identical.
2. Several program interruptions that should, according to the *IBM System/360 Principles of Operation,* store a nonzero instruction-length code are imprecise in the Model 195. An imprecise interruption is one that causes an instruction-length code of zero to be stored; this code indicates that the address of the instruction causing the interruption has not been retained. When imprecise program interruptions occur, the interruption-code portion of the current PSW is used in a special way. (See the discussion of imprecise interruptions in "Instruction Processor.")
3. Because floating-point overflow and underflow cause imprecise interruptions on the Model 195, it is possible that subsequent instructions will be executed using the overflow or underflow results. For this reason, the results are made to differ from the standard System/360 results, which produce the correct fraction and a

wraparound exponent. In the Model 195, overflow produces the correct sign and the maximum fraction and exponent; underflow produces a true zero result. For those instructions that change the condition code, the code is 1 or 2 for overflow and 0 for underflow.

4. The Model 195 is capable of executing CPU stores out of sequence. Logical consistency is maintained within CPU programs, including the beginning and ending of I/O operations. However, if a CPU program is to modify a string of CCW's while they are being used by the channel, steps must be taken to arrange the CPU program so that the stores are made in sequence.

To provide a synchronization when other means are not practical, a branch instruction may be used. This particular branch instruction is a no-operation instruction for other models of System/360, but is implemented in the Model 195 in such a way that its execution is delayed until all previously decoded instructions have been completed. (See the handling of interrupts discussion in "Instruction Processor.")

System Components

Major components of the Model 195 include an IBM 2195 Processing Unit (which includes the Processor Storage), an IBM 2860 Selector Channel, and/or an IBM 2870 Multiplexer Channel. Input/output (I/O) devices are attached to the channels through control units (Figure 1). The three processor models are termed J, K, and L, depending upon the amount of processor storage available. (In this publication, main storage and processor storage are used interchangeably.)

Processing Unit Model	*Processor Storage*	*Interleave Factor*
J	1,048,576 bytes	8
K	2,097,152 bytes	16
L	4,194,304 bytes	16

Figure 2 is an outline configuration of the 2195 J, K, and L Processing Units, including processor storage.

The standard features for any IBM 2195 Processing Unit (CPU) include:

Universal Instruction Set (including the Standard Instruction Set, Floating-Point Arithmetic, Decimal Arithmetic, and storage protection)
Extended Precision Floating-Point Arithmetic
Byte-Oriented Operands
Direct Control
Protection Features (Store and Fetch Protection)
Buffer Storage
Interval Timer (9.6-kilohertz – about 104-microsecond interval)
2870 Multiplexer Channel Attachment
2860 Selector Channel Attachment
Display Console
Remote Operator Control Panel Attachment
Emergency Power-Off Control

Resolution of the interval timer is 104 microseconds. The timer is updated by decrementing bit position 28 every 104 microseconds (more precisely, at a frequency of 9.6 kilohertz). The updating takes place with minimal interference, and no backup storage for the timer is used.

The display console, similar to an IBM 2250 Display Unit Model 1, is integrated with the system console, which is a stand-alone unit. Positioning a switch connects the display console with either an I/O channel or the system console. (When connected to a channel, the display console may be used for two-way communication with the system. When the display console is connected to the system console, the communication path is from the system console to the display console.) For connection to an I/O channel, the display console requires one control unit position on a 2860 Selector Channel or on a selector subchannel of a 2870 Multiplexer Channel.

Standard on the display console are an alphameric keyboard, character generator, light pen, 8,192-byte buffer (4,096 bytes of which are reserved for maintenance purposes and contain format control data), and operator control panel with one set of controls and indicator lights.

To control another System/360 processor, a second set of controls and indicator lights may be added as an optional feature to the operator control panel.

One set of the operator control panel controls and indicators may be duplicated as a remote panel on a stand-alone operator's console (IBM 2150 Console or IBM 2250 Display Unit Model 1). Provision for this attachment is a standard feature.

A channel-to-channel adapter, an optional feature, may be installed on an IBM 2860 Selector Channel (maximum of one per selector channel), permitting program-controlled, storage-to-storage operations to take place directly between I/O channels.

A variety of control units and input/output devices is available for use with the Model 195. Descriptions of these devices appear in separate publications. Configurators for I/O devices and for system components are also available. (See *IBM System/360 Bibliography,* Form A22-6822.)

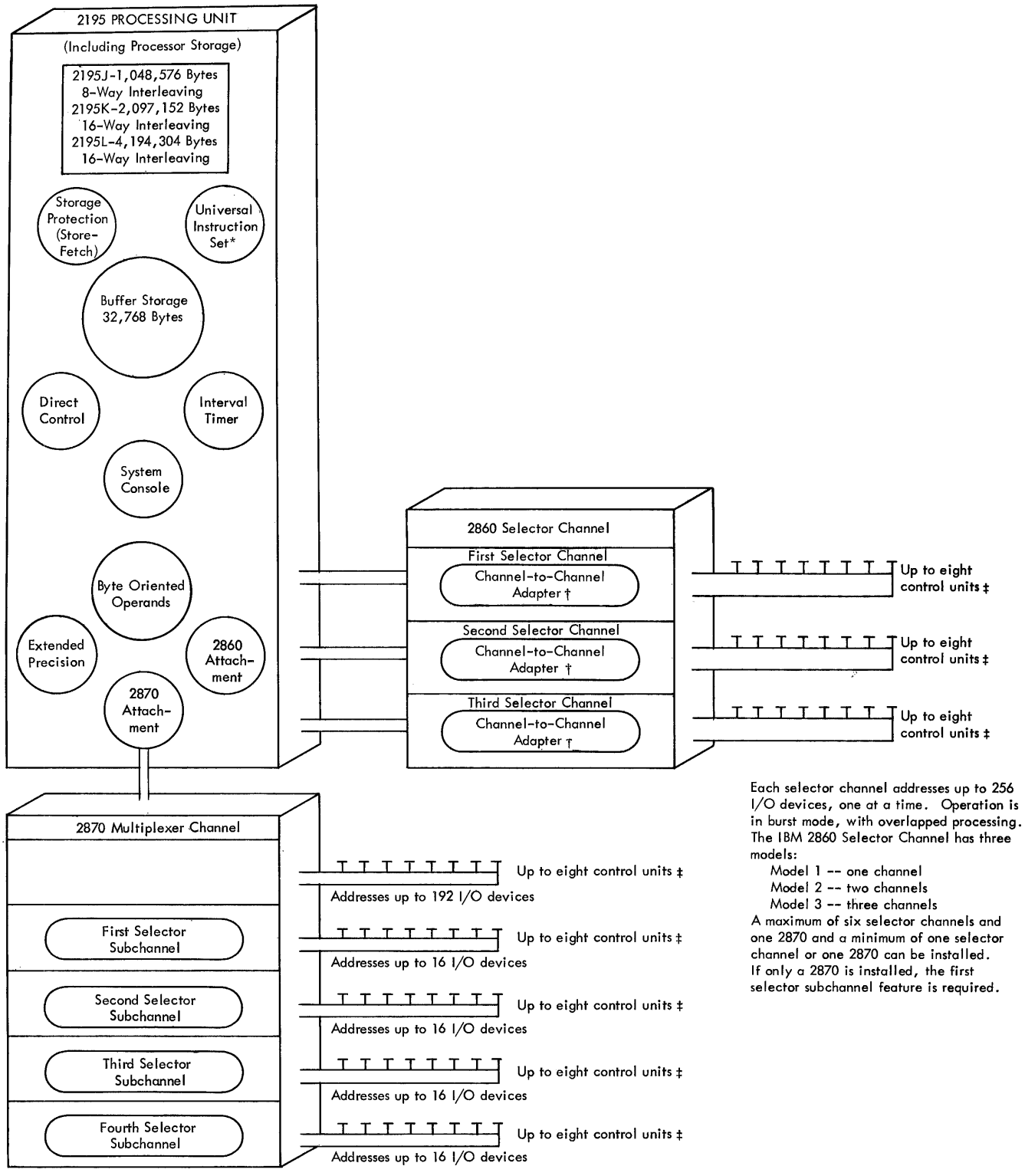
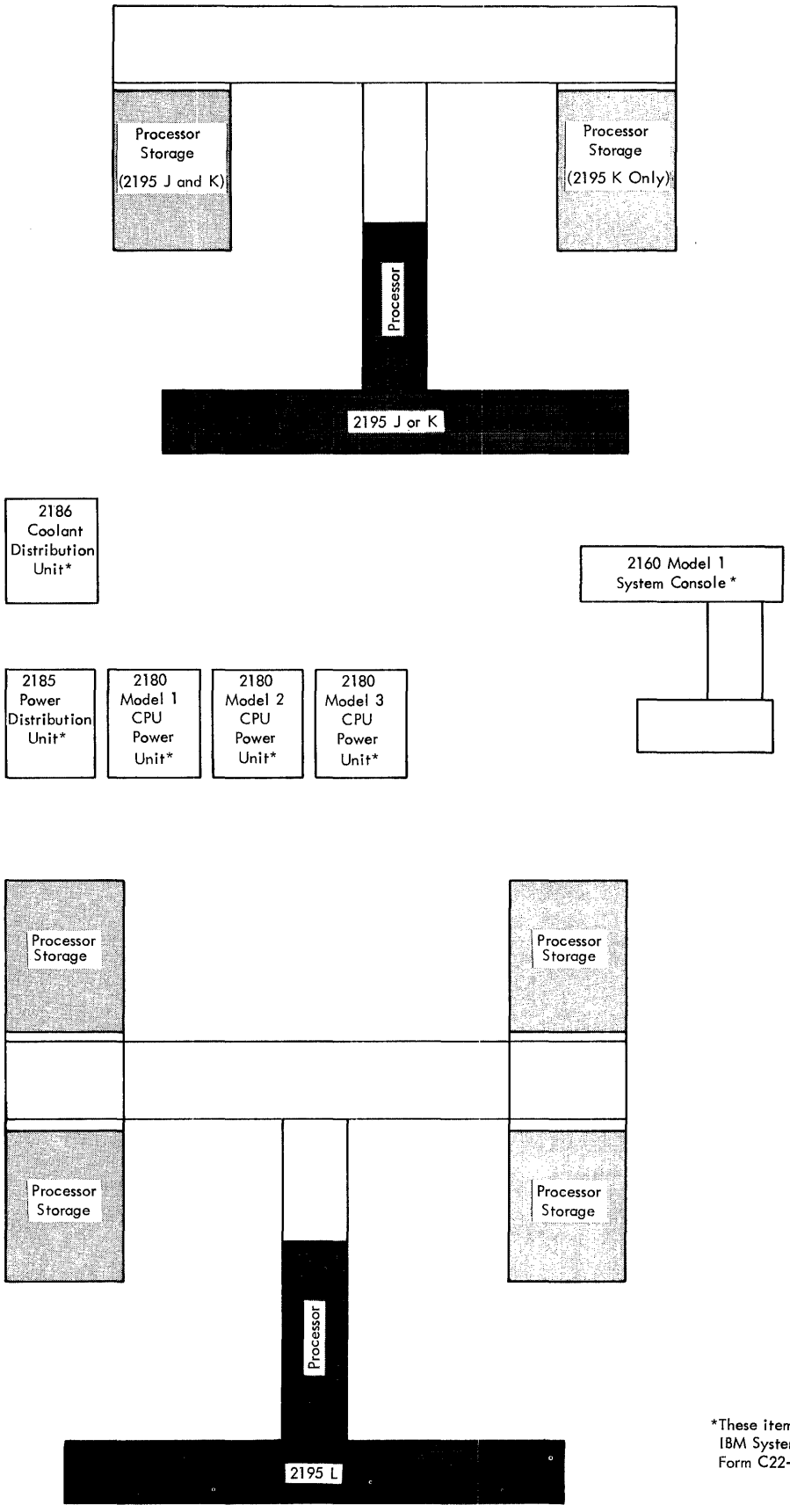


Figure 1. System/360 Model 195 Configurator



*These items may be positioned elsewhere as required. See IBM System/360 Installation Manual-Physical Planning, Form C22-6820.

Figure 2. IBM 2195 J, K, L Processing Unit and Processor Storage Configurations

The central processing complex of the System/360 Model 195 is made up of seven stand-alone units: a CPU, three CPU power supply units, a power distribution unit, a coolant distribution unit, and a system console (Figure 2). (A motor-generator set must be ordered separately and may be located in a remote area.)

CENTRAL PROCESSING UNIT

Functionally, the central processing unit consists of these major logical elements: instruction processor, fixed-point/variable-field-length (VFL)/decimal execution element, floating-point execution element, high-speed buffer storage, storage control unit, and processor storage (Figure 3). The instruction processor and the two execution elements make up the central processing element (CPE), also called the processor.

The instruction processor is the major coordinating element in the Model 195. For each instruction, it determines what must be done and issues the operation to the proper execution unit. Branching, status switching, and I/O instructions are handled by the instruction processor; other instructions are issued by the instruction processor to other processor elements for completion.

The fixed-point/VFL/decimal execution element contains the general registers, which are used also by the instruction processor. Functionally, this element operates as an independent stored-program computer; it has its own instruction stream, its own execution circuitry, and a set of operand buffers.

The floating-point execution element also operates as an independent computer. Although most of the floating-point instructions require more than one cycle of execution time, this element is capable of sustaining an execution rate of up to one instruction per cycle.

The storage control unit handles all fetching and storing of data for the CPE. It is designed to minimize conflicting requests for storage and to make the most efficient use of storage.

The high-speed buffer storage provides the principal means of reducing average access time to main storage. The most current blocks (a block is eight doublewords) of storage are maintained in the buffer storage, the operation of which is not obvious to the programmer. The first processor fetch to a block, for a storage address within that block, accesses the addressed location and initiates a transfer of the block into the buffer storage.

Subsequent accesses to that block can then be made directly from the buffer storage. Processor stores are made to both the buffer (if appropriate) and to processor storage. I/O fetches and stores are made to processor storage only.

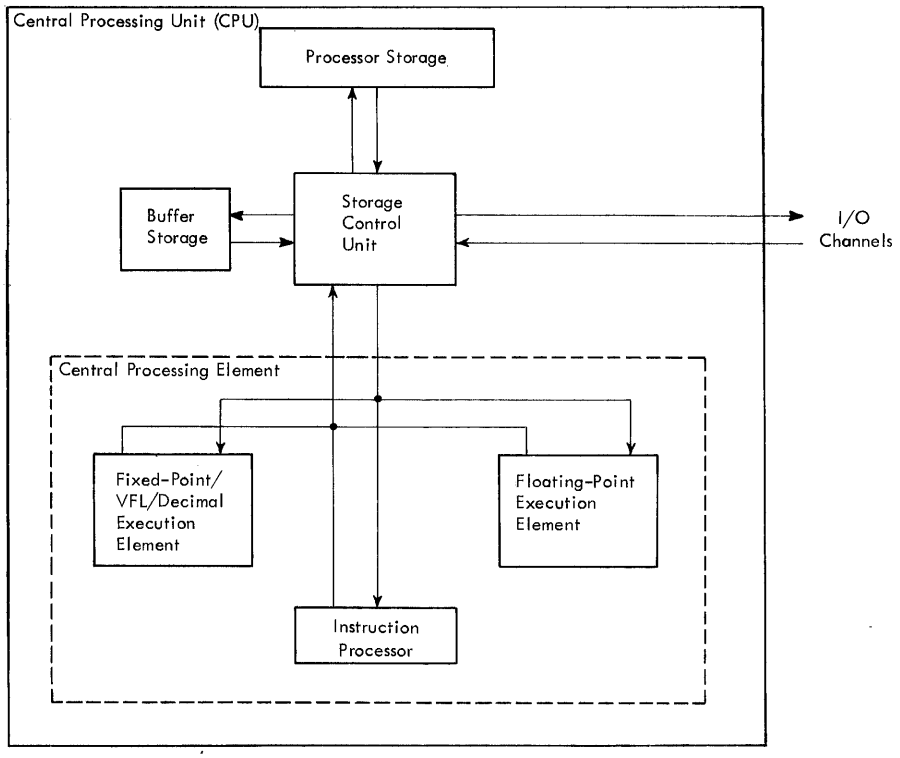


Figure 3. Model 195 Logical Elements

An I/O store to a location also currently held in the buffer storage invalidates the appropriate block in the buffer storage.

PROCESSOR STORAGE

Up to 4,194,304 bytes of processor storage are available with an individual access of eight bytes (a doubleword). Interleaving of processor storage is provided so that the addresses of successive doublewords are in successive, functionally independent units. Processor access to storage is performed in combination with the high-performance buffer storage. The effect is that average access time approaches the access time of the buffer storage. Transfers between the buffer storage and processor storage are made in blocks of eight doublewords. I/O accesses (eight bytes) to processor storage do not involve buffer storage except where necessary for control.

Function Performed	*Time (Nanoseconds)*
Access time to buffer storage	162
Access time to processor storage if not in buffer storage	810
Access time to processor storage for I/O channels	648
Cycle time for buffer storage, successive read or successive write cycles	54
Cycle time for processor storage	756

INSTRUCTION PROCESSOR

The primary functions of the instruction processor are fetching and buffering instructions from storage, fetching required operands, issuing instructions to the appropriate execution elements, handling interrupts, and executing all branching, status-switching, and I/O instructions.

The instruction processor has an instruction stack of eight doublewords, a set of three instruction-control registers, a set of temporary instruction buffer registers totaling two doublewords, a decoder, and a three-input adder for the generation of effective addresses (Figure 4). The instruction processor uses the general registers in the fixed-point/VFL/decimal execution element.

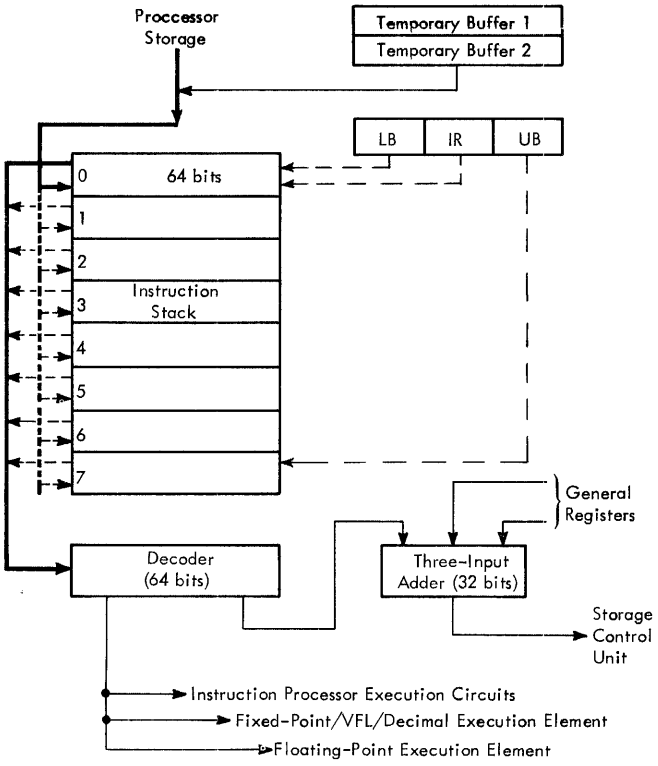


Figure 4. Instruction Processor

Instruction Fetching

Instructions fetched from storage are stored in the instruction stack of the instruction processor. An instruction stack is used for two principal reasons:

1. To minimize storage access time for instruction fetching.
2. To reduce the number of instruction fetches required while the program is executing a tight loop.

The instruction stack normally contains the current instruction doubleword, and seven doublewords of either history (instructions already decoded) or instructions to be executed. Keeping a number of doublewords ahead enables the fetching mechanism to fit instruction fetches into slack periods between data fetches and stores. The doublewords of history in the stack minimize refetching of instructions when a loop backward that can be contained in the instruction stack is detected.

The fetching mechanism operates differently under each of three conditions: initialization, normal operation, and recognition of a discontinuity. It is governed by three control registers: the instruction register (IR), the upper-bound (UB) register, and the lower-bound (LB) register. The IR points to the instruction being decoded, the UB register to the most recent doubleword brought into the stack, and the LB register to the earliest doubleword in the stack.

Initialization

Initially, the instruction stack is empty. When instruction fetching is initiated, the main-storage address of the first doubleword of instructions is set into the UB and LB registers, and part of the address of the first instruction is set into the IR. The UB register, which controls the actual fetching of doublewords of instructions, brings the first doubleword into the appropriate position of the instruction stack. At the same time, the first doubleword is brought into the decoder.

As each instruction doubleword is fetched during initialization, the UB register is incremented (a doubleword being brought into the stack for each increment) until any of three conditions occurs:

1. The address in the UB register is seven doublewords higher than that of the IR (Figure 4). Doubleword instruction fetches are made whenever it does not delay data fetching or storing.
2. A branch instruction is decoded that sets conditional mode (see "Execution of Branching Instructions").
3. A discontinuity is recognized (see "Discontinuities").

Normal Operation

During normal operation, the instruction fetching mechanism continually attempts to fetch a doubleword (Figure 4). Fetching will not take place if any of the three conditions just described is present.

When incrementing the UB register would cause the three low-order bits of that register to match the three low-order bits of the LB register, both the UB and LB registers are incremented. This incrementing of both registers causes the earliest (oldest) doubleword in the stack to be replaced with the latest doubleword just fetched. The LB and UB registers then point to a doubleword positioned one doubleword higher in the stack. This relative positioning of the LB and UB pointers (instruction stack addressing) remains constant during normal operation.

Discontinuities

A branch operation, interrupt, or store into the instruction stream may cause a disruption in fetching. (Branching operations and interrupts are discussed separately. See "Execution of Branching Instructions" and "Handling Interrupts.")

If the store instruction results in the alteration of the contents of a doubleword in the stack, the instruction fetching mechanism treats that doubleword slot as empty and fetches the altered doubleword from storage.

Because the Model 195 can execute several instructions at one time, the instruction STORE * + 4 presents a special problem. This problem is solved by making a check of the effective address of each store operation to determine whether the operation affects the instruction following the store; if the next instruction might be affected, measures are taken to preserve the logical consistency of the program.

Instruction Issuing

In addition to fetching and buffering instructions, the instruction processor fetches the required operands and issues instructions to the appropriate execution elements.

During each machine cycle, the instruction processor checks for interlocks. If there are none, the instruction selected by the instruction register is decoded. After an instruction has been decoded, the IR is incremented by the number of halfwords of the instruction just decoded, and the next instruction is then decoded. Decoding is the first of three possible stages in the issuing of the instruction.

Stage 1

During decoding, the following are determined:

1. The type of operation to be performed.
2. Whether the operation stack for the appropriate execution element can accept the operation.
3. If a storage operand is required, whether a buffer register in the appropriate execution element is available to receive the operand; or, if a store operation is specified, whether a store address register is available in the storage control unit.
4. If an effective address is required, whether the three-input adder and general registers used in generating the effective address are available.

When the results of these checks indicate that the instruction can be processed, the decoding control determines whether the instruction processor is operating in conditional mode (see "Execution of Branching Instructions"); if it is, the operation is tagged as conditional, indicating to the execution element that it is not to decode or execute the operation until signaled to do so. The operation is then issued for processing to the appropriate execution element (usually during stage 2), along with information about which buffer registers in the execution element, if any are needed, have been assigned by the instruction processor for use in the operation.

Stage 2

If address generation is required, the pertinent operand addresses are routed to the three-input adder. (Another instruction can now be processed at stage 1.) If the instruction is a store, a quick check is made of the effective address and, if this check indicates a possible store into the already fetched instruction stream, processing of the instruction at stage 1 is stopped until the processor determines whether the store is actually into the instruction stream. If it is, the processing at stage 1 remains stopped until the processor has issued a fetch to storage for the updated value of the instruction doubleword affected.

Fetches and stores can be made to operands that are not on proper boundaries; however, performance is degraded. Operands should be located on proper boundaries.

Stage 3

At this stage, the effective address of the storage operand is passed to the storage control unit. If a fetch operation is specified, the address of the buffer register to which the operand is to be issued is also specified. (During stage 3, another instruction can be processed at stage 1 and another at stage 2.)

Execution of Branching Instructions

The instruction processor executes all branching instructions. The actions taken by the instruction processor as a result of decoding a branch instruction are determined by

the type of branch instruction to be processed, the availability of circuitry for processing, and the following:

1. Whether the instruction processor is in conditional mode (see "Conditional Mode").
2. Whether the instruction processor is in loop mode (see "Loop Mode").
3. If loop mode is established, whether the current instruction is that which defined the current loop.
4. Whether the current instruction is the target of an 'execute' instruction currently being processed.

When a branch is taken, the target address of the branch normally is set into the instruction register, and the UB and LB registers and instruction stack are adjusted as required.

When a conditional branch is encountered and loop mode is not set, the instruction processor operates as though either direction could be taken. It continues to process the instructions in the instruction stack as long as conditions permit, while issuing operations to the fixed-point and floating-point execution elements on a conditional basis. These conditional operations will not be executed until after the condition code is set.

The instruction processor also makes use of two temporary buffers. Into these buffers it fetches the branch-target doubleword and the succeeding doubleword. Therefore, regardless of the outcome of the branch operation, the instruction processor will have a lead in the correct direction.

Conditional Mode

Conditional mode is established when the instruction processor executes a 'branch on condition' instruction for which the condition code is not yet determined.

When conditional mode is set, no additional instruction fetches are made beyond the first two doublewords at the target address of the branch. The instruction processor continues to decode instructions, generate addresses, and issue operations to the fixed-point and floating-point execution elements. The operations issued, however, are tagged as conditional and cannot be decoded or executed until the condition code is set and the instruction processor sends a signal to the execution element.

The instruction processor continues to decode instructions conditionally until any of the following occurs:

1. The condition code is set.
2. No instructions are available in the instruction stack.
3. The operation stack of the fixed-point or floating-point execution element is full, and the currently decoded instruction needs the filled execution element.
4. An instruction to be executed within the instruction processor is decoded, or a variable-field-length instruction is decoded. (However, a no-operation instruction or an unconditional branch may be executed during conditional mode.)

Loop Mode

Whenever a branch backward is taken to a target fewer than eight doublewords back from the current address in the instruction register, loop mode is entered and the instruction stack is reinitialized to contain the pertinent eight doublewords. The loop is then locked into the instruction stack and, as a result, can be executed repetitively without refetching the instructions. Thus, conflicts between instruction fetching and data fetching are eliminated, and branches can be executed faster.

During loop mode reinitialization, when no data fetches or stores are to be made, an instruction doubleword is fetched every cycle until the instruction stack is full. If data fetches or stores are to be made, instruction doubleword fetches take second priority.

When loop mode is entered, the branch target address is placed in one special register, and the address of the branch instruction is placed in a second special register. Subsequently, when a branch instruction is decoded during loop mode, that instruction address is compared with the address (in the second special register) of the branch instruction that initiated loop mode; if they are the same, the branch is made to the target address in the first special register. Because no time is taken to re-form the address specified in the branch instruction, one cycle is saved.

If a conditional branch instruction is processed when loop mode is already set, it is assumed that the branch will be taken; therefore, during loop mode no temporary fetches (down the no-branch path) are made for conditional branches.

Loop mode is turned off when any of the following occurs:

1. A branch out of the instruction stack is taken.
2. The instruction processor starts to decode the 32nd halfword in the instruction stack.
3. The target of the quick loop is the same as the target of the outermost loop, and the branch closing the quick loop is not taken. (If two nested loops fit in the instruction stack, the innermost loop is called the quick loop.)
4. The base register or index register of the quick-loop branch is altered.

Programming Notes: Because of item 2, a loop with 29-31 halfwords should be aligned on a doubleword boundary. If the loop has fewer than 29 halfwords, the loop is executed in loop mode regardless of boundary alignment; if it has more than 31 halfwords, it is not executed in loop mode.

Because of item 3, if the nested loops both have the same target address, loop mode will be destroyed every time an exit is made from the quick loop. To prevent loop mode from being destroyed, a no-operation instruction may be used as a dummy branch target for the outer loop.

Execution of Other Instructions

The instruction processor executes all status-switching and I/O instructions and plays a large part in the execution of multiple-operation instructions. When one of these instructions is processed, the instruction processor usually does not issue any succeeding instruction until its part in processing the first instruction is completed.

None of these instructions is executed while conditional mode is set. Some require that all instructions being executed when that instruction is decoded, be completed prior to its execution. The instructions requiring this completion of other instructions are the four I/O instructions and 'load PSW', 'supervisor call,' 'set storage key,' and 'set program mask' (except when the old and new mask bits are the same). Also, one type of 'branch on condition' instruction (a no-operation instruction) is implemented in the Model 195 in such a way that all other instructions being executed when it is decoded must be completed before its execution. See the programming note in "Handling Interrupts."

The following instructions are classed as multiple-operation instructions:

Load Multiple (LM)	Move With Offset (MVO)
Store Multiple (STM)	Pack (PACK)
Translate (TR)	Unpack (UNPK)
Translate and Test (TRT)	Edit (ED)
And (NC)	Edit and Mark (EDMK)
Or (OC)	Add Decimal (AP)
Exclusive Or (XC)	Subtract Decimal (SP)
Compare Logical (CLC)	Compare Decimal (CP)
Move Zones (MVZ)	Multiply Decimal (MP)
Move Numerics (MVN)	Divide Decimal (DP)
Move (MVC)	Zero and Add (ZAP)

These multiple-operation instructions have variable length data fields and require the issuing of more than one operation from the instruction processor to the fixed-point execution element, which shares responsibility for execution with the instruction processor. Also, each operation of a multiple-operation instruction issued to the fixed-point area contains information concerning at least one storage request.

The multiple-operation instructions are the only instructions, except 'convert to binary,' that cause operands to be fetched into the floating-point operand buffers for use in the fixed-point area. Four of the six fixed-point operand buffers are unavailable for reassignment while a multiple-operation instruction is being executed.

Usually, the instruction processor is available to issue the succeeding instruction after it has issued the last required operation to the fixed-point area. If the next instruction is in the SI format, it is not issued until the variable-field-length execution for the multiple-operation instruction is complete. If the multiple-operation instruction is a 'translate and test' (TRT) or an 'edit and mark' (EDMK) instruction, the instruction processor will be available to issue subsequent instructions only after the entire TRT or EDMK instruction has been executed.

Handling Interrupts

The Model 195 performs all interrupt functions defined for the IBM System/360. (See *IBM System/360 Principles of Operation,* Form A22-6821.) The supervisor call, external, machine check, and I/O interrupts are logically handled as defined.

The performance objectives of the Model 195, however, require some deviations in handling program exceptions. The program-exception deviations are basically those resulting from an operation that has been sent by the instruction processor to another element for execution, so that the current PSW no longer references the operation. Consequently, the interrupt-causing instruction cannot be directly identified. Such a program interrupt is called imprecise. An imprecise interrupt is identified by the storing of zero as the instruction-length code in the PSW current at the time of interrupt.

Logical accuracy is preserved in all situations where a basic machine status change is involved. For example, all instructions issued under a program mask are completed before the mask is changed to ensure that the mask stored is that which allowed the interrupt.

The instruction-length codes (ILC) for program interrupts on the Model 195 follow. The codes in this listing replace those listed for ILC on program interrupts in the *IBM System/360 Principles of Operation.*

Program Exception	*ILC*
Operation	1,2,3
Privileged Operation	1,2
Execute	2
Protection	0
Addressing	0,1,2,3
Specification	0,1,2,3
Data	0
Fixed-Point Overflow	0
Fixed-Point Divide	0
Decimal Divide	3
Decimal Overflow	3
Exponent Overflow	0
Exponent Underflow	0
Significance	0
Floating-Point Divide	0

Imprecise Interrupts

The following program exceptions always cause *imprecise* interrupts:

1. Data, fixed-point-overflow, fixed-point-divide, decimal overflow, and decimal divide exceptions signaled from the fixed-point/VFL/decimal execution element.
2. Exponent-overflow, exponent-underflow, significance, and floating-point-divide exceptions signaled from the floating-point execution element.
3. A protection exception when a protection violation is detected.

An addressing exception can produce either a precise or an imprecise program interrupt, as determined by the problem.

When an imprecise interrupt is signaled, the instruction processor ensures that all instructions that were decoded before the signal was recognized are completed before the interrupt is honored. When the interrupt is taken, the instruction address stored in the program old PSW points to the next instruction that would have been decoded, and for which an attempt would have been made to issue it, had the interrupt not occurred.

Imprecise interrupts that arise from conditional instructions (that is, instructions issued subsequent to a 'branch on condition' instruction for which the condition code is not yet determined) are noted and either activated or canceled, as appropriate, when the conditional instructions themselves are activated or canceled.

When an imprecise interrupt takes place, not just one but several exceptions may have occurred, because all decoded instructions are completed before the interrupt is taken. Also, because instructions can be executed out of sequence, the interrupt condition recognized first may not be the condition that logically should be recognized first. To account for both possibilities (an out-of-sequence detection and the occurrence of more than one type of exception, either within one or different instructions), the action taken when an imprecise interruption occurs is that each type of exception that took place is identified in bits 16-25 of the program old PSW, and bits 26-31 are set to zero. Also, the instruction-length code (bits 32-33) is set to zero.

Bit Position in Program Old PSW	*Program Exception*
16	Protection
17	Addressing
18	Not Used
19	Data
20	Fixed-Point Overflow
21	Fixed-Point Divide
22	Exponent Overflow
23	Exponent Underflow
24	Significance
25	Floating-Point Divide
26	Decimal Overflow
27	Decimal Divide

Note: For an imprecise interrupt, the types of exceptions that occurred, but not the number of exceptions of any one type that occurred, are identified in the program old PSW.

Precise Interrupts

When the program interrupt is precise, bits 28-31 of the program old PSW identify the exception causing the interrupt; the remainder of the interrupt code (bits 16-27) is all zeros; and the instruction-length code (bits 32-33) is 1, 2, or 3, as appropriate.

A logical consistency is maintained when a precise program interrupt precedes an imprecise program interrupt that logically should have taken place first. If an imprecise interrupt occurs during execution of outstanding instructions before a precisely identifiable interrupt is honored, the instruction causing the precise interrupt is not executed, the precise interrupt condition associated with this instruction is not indicated, and the address of the instruction causing the precise interrupt is placed in the instruction-address portion of the program old PSW. In effect, the instruction leading to the precise interrupt is treated as never having occurred, and a return to the program causes the original interrupting instruction to be reinitiated. (The same operation takes place when a supervisor-call interrupt is followed by an imprecise program interrupt that logically should have occurred first.)

Addressing Exceptions

An addressing exception resulting in a precise program interrupt is produced if any of the following conditions is detected:

1. Any portion of the current instruction to be decoded lies outside available storage.
2. The address generated for any of the following instructions lies outside available storage: 'read direct,' 'write direct,' 'load PSW,' 'set system mask,' 'set storage key', 'insert storage key,' and 'diagnose.'
3. Any portion of the target instruction for 'execute' lies outside available storage.

All other addressing exceptions, which are signaled after the completion of address generation leading to the fetching or storing outside of available storage, result in imprecise program interrupts.

Specification Exceptions

A specification exception resulting in a precise program interrupt is produced if any of the following conditions is detected:

1. An attempt is made to execute an instruction specified at an odd-numbered location in storage.
2. The R1 field of an instruction specifies an odd-numbered register for the pair of general registers that contains a 64-bit operand.
3. A number other than 0, 2, 4, or 6 is specified for a floating-point register.
4. The block address specified in 'set storage key' or 'insert storage key' has the four low-order bits not all zero.
5. The three low-order bits are not all zero in the address generated for 'load PSW' or 'diagnose.'
6. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.
7. The first operand field is shorter than or equal to the second operand field in decimal multiplication or division.

Programming Notes

A program may not operate correctly on the Model 195 if identification of the instruction that caused an imprecise interrupt is required. When an imprecise interrupt occurs, the program old PSW does not reference the operation that caused it.

Also, a program may not operate correctly on the Model 195 if it requires the honoring of an imprecise interrupt before some instruction later in the program is executed. When an imprecise interrupt is detected, all instructions decoded by that time are executed before the interrupt is taken. Therefore, several instructions following the instruction that caused the imprecise interrupt may be executed before the interrupt is taken. (How many of these subsequent instructions will be executed will vary, principally because the Model 195 can execute instructions concurrently and out of sequence.) It is possible, at the programmer's option, to return to the problem program but, because all decoded instructions are completed before the interrupt is taken, the instructions executed after the interrupt may have been adversely affected by the program exception.

If preciseness is a principal concern, the unwanted effects of imprecise program interrupts can usually be eliminated by testing and masking, as appropriate, and by using this 'branch on condition' instruction:

Mnemonic	*Type*	*M_1 Field*	*R_2 Field*
BCR	RR	Not zero	Zero

This branch instruction is a no-operation instruction for System/360 generally, but is implemented in the Model 195 in such a way that its execution is delayed until all previously decoded instructions have been completed.

Note: The address in the instruction counter is that of the BCR instruction, and the instruction length code is as listed at the beginning of this section. The use of this no-operation instruction degrades the performance of the Model 195. It should be used only to eliminate a problem for which there is no other reasonable solution.

Note that a program may have been naturally arranged so that the adverse effects of certain imprecise program interrupts are eliminated in advance. For example, in addition to the branch (no-operation) instruction just mentioned, execution of the following instructions is delayed until all previously decoded instructions have been completed: the four I/O instructions, 'load PSW','supervisor call', 'set storage key', 'diagnose', and 'set program mask' (except when the old and new mask bits are the same).

The execution of instructions out of sequence may present a problem in a situation other than the one concerning imprecise interrupts. Although the CPU maintains a logical consistency with respect to its own operations, including the starting and ending of I/O operations, it cannot ensure logical consistency between the CPU and asynchronous units during their operations. For example, if an I/O channel program relies on proper sequencing of stores by the CPU to ensure proper channel operation, steps must be taken in the CPU program to guarantee that the stores actually are made in that sequence. The no-operation instruction can be used to accomplish this.

Interrupt Example

The following example taken from the program controlled interrupt (PCI) appendage for dynamic buffer allocation in the basic telecommunications access method (BTAM) illustrates the dependence of an asynchronous channel program upon serial execution. The example also demonstrates use of the BCR instruction to effect serial execution.

The purpose of the PCI appendage is to maintain an uninterrupted transmission of data into main storage. The controlling factors in this transmission are the availability of buffers and the ability (of the appendage routine) to modify and chain two channel programs. Each channel program consists of the following two channel command word (CCW) chains:

Chain 1	CCW1	READ	into a buffer with data chaining (CD) and PCI flags
	CCW2	READ	into CCW3 with skip (SKIP) and suppress length indication (SLI) flags
Chain 2	CCW3	READ	into a buffer with CD and PCI flags
	CCW4	READ	into CCW1 with SKIP and SLI flags

1. In Chain 1, CCW1 is initialized with the first available buffer address.
2. The address fields for CCW2 and CCW4 initially contain the storage addresses of CCW3 and CCW1, respectively.
3. When the PCI interrupt in CCW1 occurs, the PCI appendage routine determines the address of the next available buffer and stores it into the address field of CCW3.
4. The command code in CCW2 is changed from a READ to a transfer-in-channel (TIC), and the command code in CCW4 is reset to READ as shown in Chain 2. (The first time through the channel program, CCW4 is already set to READ.)
5. When the PCI interruption in CCW3 occurs, the PCI appendage routine determines the address of the next available buffer (after the one indicated in step 3) and stores it into the address field of CCW1.
6. The command code in CCW4 is changed from a READ to a TIC, and the command code in CCW2 is reset to READ as shown in Chain 1.

Steps 3 through 6 of the preceding sequence of events continue until the input data transmission is completed. The PCI appendage instructions that accomplish the alteration of the CCWs are:

ST	available buffer address into CCW3 (or CCW1)
BCR	15, 0
MVI	into CCW2 (or CCW4), the code for a TIC command
MVI	into CCW4 (or CCW2), the code for a READ command

On the Model 195, use of the BCR instruction effects a pipeline drain to ensure that the proper buffer address is stored before the READ command is changed to a TIC command. If, as could happen when the BCR instruction is omitted, the MVI instructions were completed out of sequence (i.e., before the ST instruction), the channel could fetch a READ (into buffer) command with an old buffer address. In such a situation, the new input would overlay the old data.

Note: In this example, it is assumed that the PCI appendage instructions necessary to alter the format of the CCWs were executed before the channel fetched the "READ xx (SKIP, SLI)" command. If, in a given situation, this is not the case, then the fetching of the "READ xx (SKIP,SLI)" command is executed without transfer of data and causes a normal termination. Then, the appropriate channel program must be restarted at the expense of additional input/output time.

STORAGE CONTROL UNIT

The storage control unit (SCU) is the intermediary between main storage and the other system units. As such, it controls central processing element (CPE) access to the high-speed buffer storage backed up by the full capacity of main storage. The SCU:

1. Provides and controls data and address paths to and from main storage and the buffer storage for the central processing element, the channels, and the system console.
2. Controls the transfer of doubleword blocks of information from main storage to the high-speed buffer storage.
3. Buffers store operations pending the availability of store data.
4. Properly sequences CPE stores and fetches to the same address.
5. Provides the storage protection function.

Buffer Storage

Because of the sequential nature of most programs, a CPE fetch is likely to be followed by succeeding fetches to the same storage block. Access time for subsequent fetches is considerably reduced by placing the addressed block of main storage in the high-speed buffer storage. Block transfer is controlled by the storage control unit so that use of the buffer storage is not obvious to the programmer.

The Model 195's increased performance is due in part to the faster access to instructions and operands provided by the high-speed buffer storage. Whereas normal instruction or operand fetches from main storage require 810 nanoseconds, fetches from the buffer storage require only 162 nanoseconds.

Main storage and the high-speed buffer storage (Figure 5) are similarly arranged. In main storage, eight doublewords (64 bytes) occupy each of the 128 blocks that form a segment. The largest processor storage, in the Model 195 L, has 512 segments. The buffer storage is arranged in a like manner but with four segments. Also provided are four data directories, one for each buffer segment, with 128 locations per directory. Each location in the directory contains the main storage address of the block of data in the corresponding block of the high-speed buffer storage.

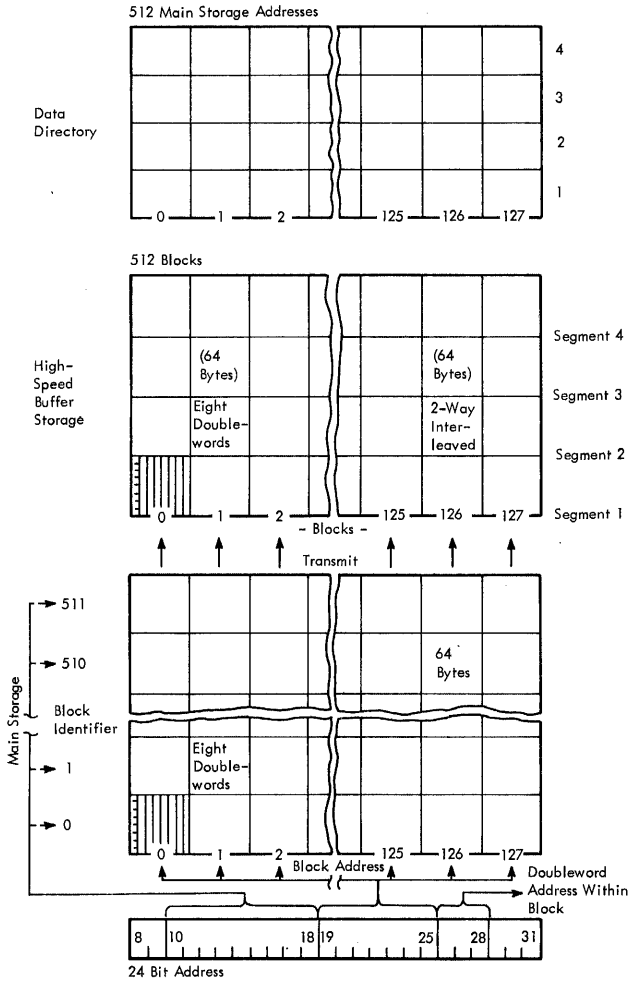


Figure 5. High Speed Buffer Storage

Buffer Storage Operation

When the central processing element performs a store operation to main storage, the main storage address is placed in a store address register to await arrival of data in a store data buffer. See Figure 6. Comparing the address of the block addressed by the store operation with the block addresses in the data directory indicates whether the location also resides in the buffer storage. If so, the store is directed to both main storage and the buffer storage; if not, only main storage is modified by the store operation.

A channel or system console store to main storage must also determine when the addressed block resides in both main storage and the buffer storage, but for a different reason. Input from the channel or system console is directed only to main storage. Therefore, when the store address is also in the buffer storage, the addressed block is invalidated in the data directory. Consequently, it must be retransmitted from main storage before a subsequent fetch to that block is allowed.

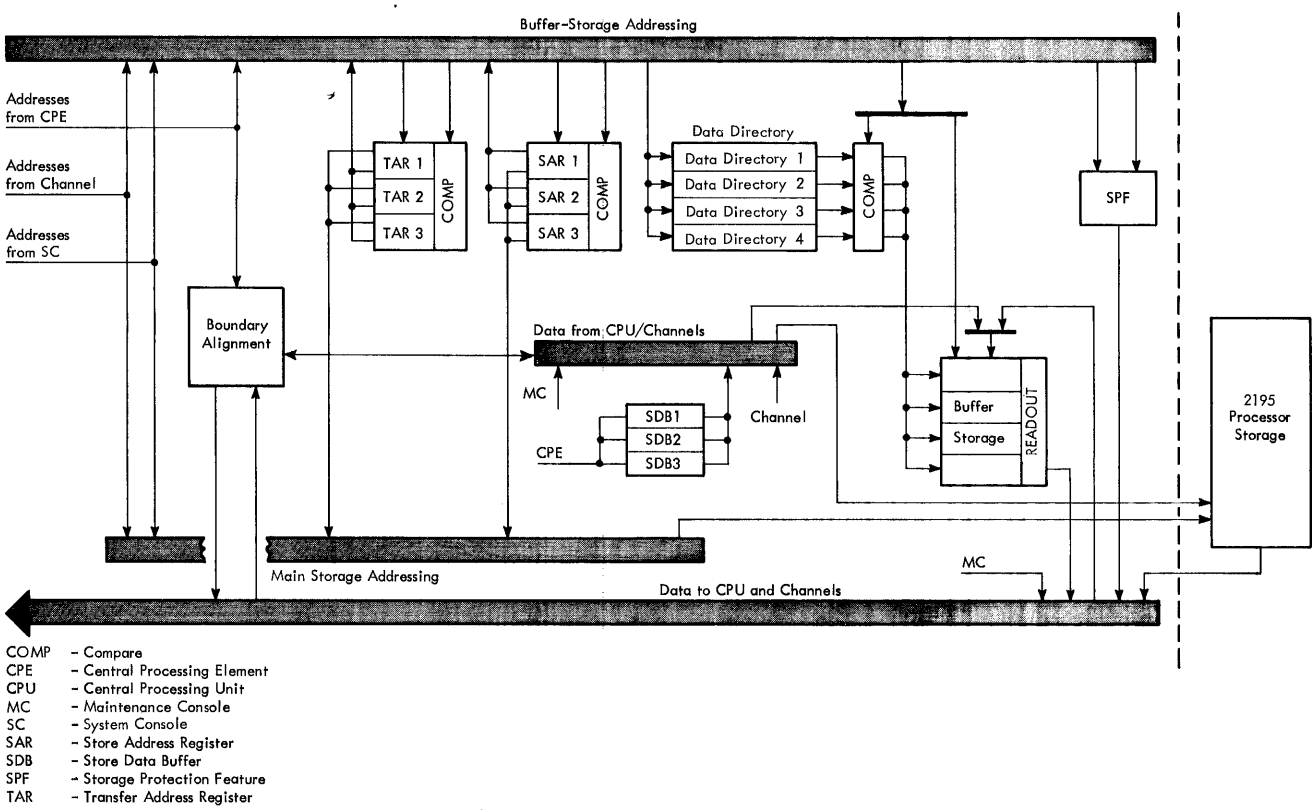


Figure 6. Storage Control Unit Data Paths

CPE fetches from main storage are usually fetches from the high-speed buffer storage. The CPE fetch address is placed in a transfer address register (Figure 6), and a comparison is made with the store address register and the directory. An equal compare with the store address register causes the fetch to be delayed until the indicated store to that address is completed.

An equal comparison of the CPE fetch address and the data directory indicates that the data to be fetched resides in the high-speed buffer storage. The fetch is then made from the buffer storage and the data placed on the bus to the CPE.

When the fetch address does not reside in the buffer storage, a block transfer to the buffer storage is called for. The addressed doubleword is fetched from main storage and placed on the bus to the CPE. This doubleword is also transmitted to the buffer storage to become the first of eight doublewords in the block transfer. Subsequent fetches to this block can then be made from the high-speed buffer storage.

The block of doublewords transmitted to the buffer storage is placed in a block location corresponding to its main storage location as determined by the block address (Figure 5), and the block's main storage address is placed in the corresponding data directory location. The block may be placed, however, in any one of four possible buffer storage segments.

Because all doublewords having the same block address are assigned to the same buffer storage location, four identical buffer segments are used to avoid conflicts. Which of the four buffer segments is used or replaced is determined by the replacement code. The replacement code is maintained to indicate the order of buffer storage segment usage. It indicates the most recently, second most recently, third most recently, and least recently accessed segment for each of the four possible blocks to be accessed.

The block transmitted from main storage replaces the least recently accessed segment block. Thus, the buffer always contains 512 blocks of main storage that have been used most recently.

Channel fetches are made only from main storage. Addresses from channels are held until the requested storage is free. Channel requests are then given highest priority to ensure against channel overrun.

FIXED-POINT/VARIABLE-FIELD-LENGTH/DECIMAL EXECUTION ELEMENT

The fixed-point/variable-field-length (VFL)/decimal execution element executes all fixed-point arithmetic, logical, and variable-field-length and decimal arithmetic operations. It consists of six major logical elements (Figure 7):

1. An operation stack (FXOS) of six positions
2. Sixteen general registers
3. Six 32-bit operand buffers (FXB)

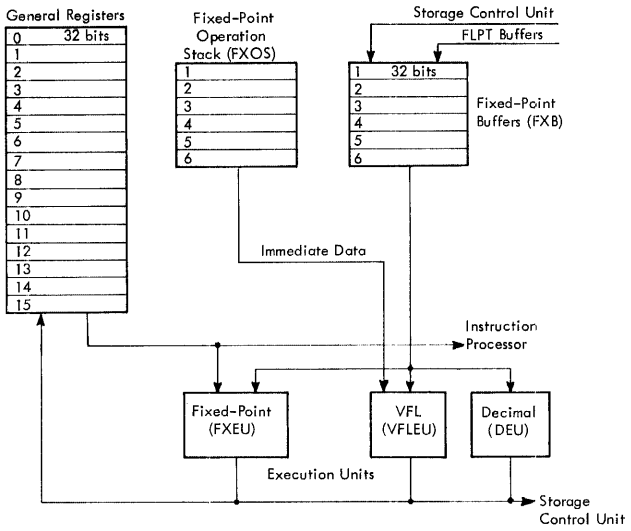


Figure 7. Fixed-Point/VFL/Decimal Execution Element

4. A fixed-point execution unit (FXEU)
5. A VFL execution unit (VFLEU)
6. A decimal execution unit (DEU)

Fetches from storage of data fields necessary for processing a fixed-point operation are initiated by the instruction processor, which also reserves in the fixed-point area the buffers that are to receive the requested operands.

The instruction processor also maintains for its own use counters that indicate whether: (1) the fixed-point operation stack has an available position, (2) which fixed-point buffers are available, and (3) which general registers are available to the instruction processor and which are being used by the fixed-point/VFL/decimal execution element.

During normal processing, operations in the FXOS are decoded serially and issued to either the fixed-point execution unit or the VFL or decimal execution unit. An operation can be executed if it has been decoded, if the data is available, and if the execution circuitry is free. When decoding is completed, the instruction processor is notified that the stack position and operand buffers assigned to that operation are free.

The execution of one operation is overlapped when possible with the decoding of the next. When a multiple-operation instruction is processed (see the discussion of multiple-operation instructions under "Instruction Processor"), decoding of the next instruction in the FXOS does not begin until the execution of the last of the multiple operations is begun.

Operations tagged as conditional are not decoded or executed until they are activated or canceled by the instruction processor. A canceled operation is decoded in one cycle, and execution of the operation consists of freeing any operand buffers previously assigned to the canceled operation without actual execution of the operation.

At any time during a fixed-point/VFL/decimal operation, the instruction processor can request a direct store into the general registers, which, because the instruction processor has priority, may delay fixed-point/VFL/decimal processing.

Fetches made during the execution of a multiple-operation instruction may require the use of operand buffers in the floating-point execution element; also, four of the six fixed-point operand buffers are unavailable for reassignment while such an instruction is being processed.

FLOATING-POINT EXECUTION ELEMENT

The floating-point execution element handles execution of the floating-point arithmetic operations, including the extended precision operations. (See "Extended Execution Unit.") Several operations can be executed at one time (maximum: two adds and one multiply or divide) if the operations are logically independent. This performance capability results largely from three significant features: (1) operand and instruction buffering, (2) multiple execution units employing extremely efficient algorithms, and (3) a common data bus, which links the several sets of execution circuitry so that the full power of the multiple execution units is realized without a reliance on programming for the special arrangement of instructions.

The floating-point area contains the following major logical elements (Figure 8):

1. An operation stack (FLOS) of eight positions.
2. Four floating-point registers (FLR).
3. Six operand buffers (FLB), which are also used by the fixed-point area when any multiple-operation instruction or the 'convert to binary' instruction is processed.
4. Three execution units: an add unit (preceded by three reservation stations) capable of performing two add operations concurrently, a multiply/divide (M/D) unit (preceded by two reservation stations), and an extended execution unit (preceded by one reservation station).

Decoding of operations in the FLOS proceeds serially. As an execution unit is selected for an operation (on the second cycle), the decoding of the next operation (on the first cycle) can begin. The FLOS issues operations subject to only one principal constraint: a reservation station of the appropriate type must be available.

The FLOS need not wait for all the operands to be available (as in the fixed-point area) before issuing the operation. Instead, the selected reservation station and controls hold the issued operation until the required operands have been collected and then engage the execution circuitry.

Because several operations may be in various stages of execution at one time, provision must be made for properly sequencing dependent operations. A system of tagging for usage of the common bus ensures proper sequencing and also facilitates fastest execution of independent operations.

The FLB and the common data bus execute all RX load operations. RR load and RR load and test operations are executed by the common data bus and special testing circuitry. Store operations are executed by the three store data buffers. Multiply and divide operations are executed

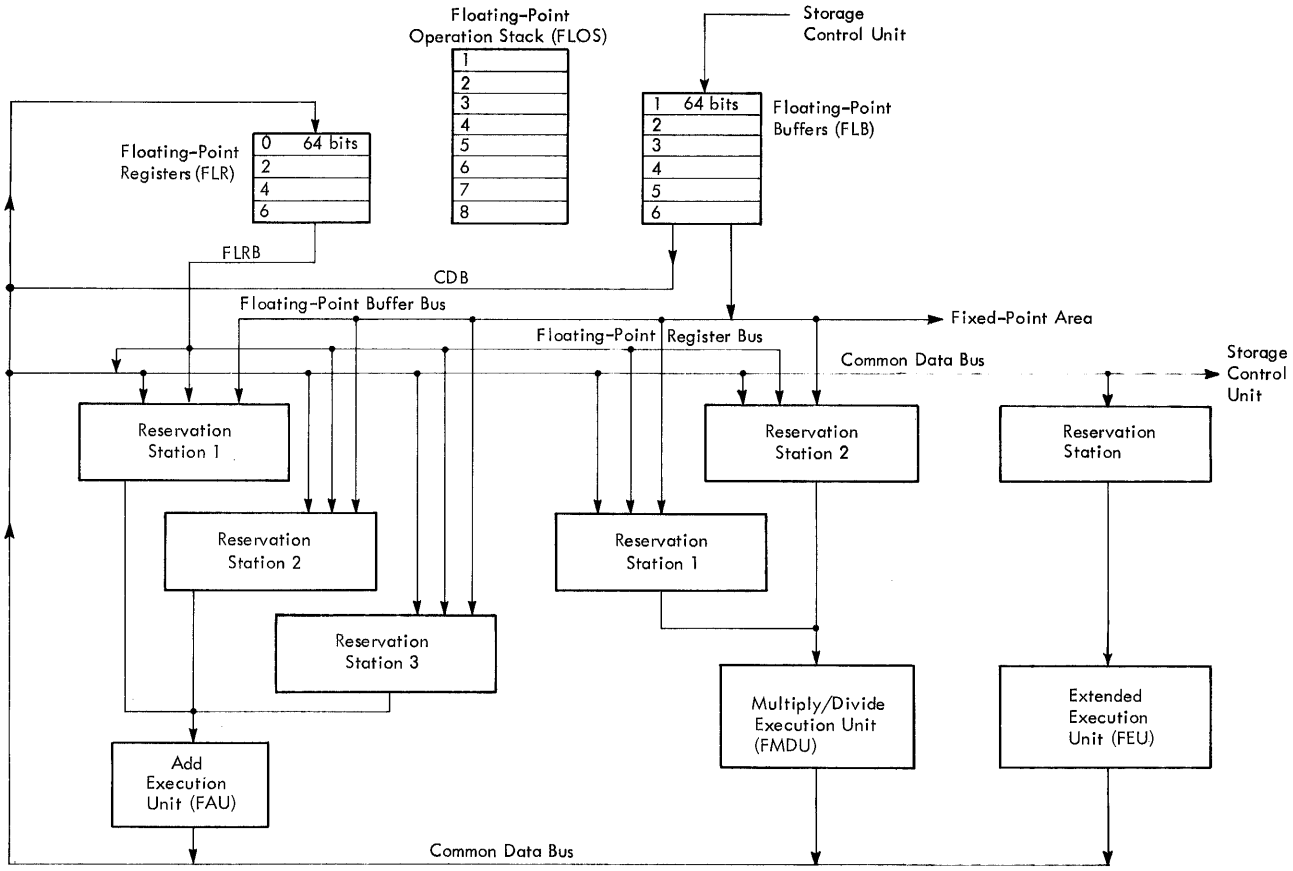


Figure 8. Floating-Point Execution Element

by an M/D unit. All other operations are executed by the add unit.

Fetches for data fields needed to process a floating-point operation are initiated by the instruction processor, which also reserves the buffers in the floating-point area that are to receive the requested operands. The instruction processor also maintains for its own use counters that indicate whether the FLOS has an available position and which floating-point buffers are available.

When the FLOS completes decoding, it signals the instruction processor that the stack position is empty. If an operation has been decoded, the related operand buffer is set free at the time it is filled; if the operand buffer is filled before the related operation is decoded, however, the buffer is set free one cycle after decoding is completed.

Operations tagged as conditional are not decoded or executed until they are activated or canceled by the instruction processor. A canceled operation is decoded in one cycle, and execution of the operation consists of freeing all operand buffers previously assigned to the canceled operation without actual execution of the operation.

Add Execution Unit

The add execution unit can begin execution if the operation has been decoded, the data is available, and another add operation of higher priority is not beginning on the same cycle. Two add operations can be executed concurrently by offsetting the start of the second operation one cycle from the start of the first. While two operations are being performed, the third reservation station may be acquiring data.

Multiply/Divide Execution Unit

The multiply or divide execution unit can begin execution if the operation has been decoded, the data is available, another multiply or divide operation of higher priority is not beginning on the same cycle, and the execution circuitry is free. The two M/D reservation stations share a single execution section; therefore, only one M/D operation may be executed at a time.

Extended Execution Unit

The extended execution unit (Figure 8) is a standard feature that provides additional logic for handling extended precision (28-digit fraction) floating-point operands. The feature includes seven instructions and additional controls for using the multiply unit. (Details of the extended-precision floating-point instructions are in *IBM System/360 Principles of Operation,* Form A22-6821.)

Instruction execution begins when the operands are in the reservation station. For an extended-precision multiply operation, priority for the use of the multiply unit is required.

Channels

In the Model 195, the IBM 2860 Selector Channel and the IBM 2870 Multiplexer Channel provide for attachment of I/O devices to the system. The channel relieves the CPU of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O operations.

Data is transferred a byte at a time between the I/O device and the channel. A standard channel-to-control-unit interface provides a uniform method of attaching control units to all channels. Data transfers between the channel and the SCU are eight bytes (one doubleword) in parallel for both selector and multiplexer channels.

2860 SELECTOR CHANNEL

The 2860 Selector Channel provides for attachment and control of I/O control units and associated devices. At least one 2860 (any model) is required if no 2870 Multiplexer Channel is attached. The 2860 is available in three models:

Model 1 – provides one selector channel
Model 2 – provides two selector channels
Model 3 – provides three selector channels

One 2870 Multiplexer Channel and six selector channels in the following combinations may be attached to the CPU:

No. of Channels Required	*Recommended Combinations*
1	One 2860-1
2	One 2860-2
3	One 2860-3
4	One 2860-3 and one 2860-1; or two 2860-2
5	One 2860-3 and one 2860-2
6	Two 2860-3

The selector channel permits data rates of up to 1.3 million bytes a second. I/O operations are overlapped with processing and, depending on the data rates and channel programming considerations, all selector channels can operate concurrently. A set of channel control and buffer registers permits each channel to operate with a minimum of interference.

Eight control units can be attached to each selector channel. Each control unit may have more than one I/O device connected to it, but only one device per channel may transfer data at any given time. A selector channel operates only in burst mode.

2870 MULTIPLEXER CHANNEL

The 2870 Multiplexer Channel provides for attachment of a wide range of low- to medium-speed I/O control units and associated devices. One 2870 can be attached to the Model 195, furnishing up to 196 subchannels, including four optional selector subchannels. If no 2860 is attached, the first selector subchannel feature is required on the 2870.

The basic 2870 Multiplexer Channel with 192 subchannels can attach eight control units and can address 192 I/O devices. The basic multiplexer channel can operate several multiplex-mode I/O devices concurrently or a single burst-mode device.

One to four selector subchannels are optional. Each selector subchannel can operate one I/O device concurrently with the basic multiplexer channel.

Each selector subchannel permits attachment of eight control units for certain devices having a data rate not exceeding 180 kilobytes (kb) a second. Regardless of the number of control units attached, a maximum of 16 I/O devices can be attached to a selector subchannel.

The maximum aggregate data rate for the multiplexer channel ranges from 110 kb to 670 kb, depending on the number of selector subchannels in operation. The first three selector subchannels may operate concurrently at up to 180 kb for each subchannel. When all four selector subchannels operate concurrently, the fourth has a maximum data rate of 100 kb.

Each selector subchannel in operation diminishes the basic multiplexer channel's maximum data rate of 110 kb; the maximum data rates for concurrent selector subchannel operations are:

Basic Multiplexer Channel	*Data Rates for Selector Subchannel*			
	1st	*2nd*	*3rd*	*4th*
110 kb				
88 kb	180 kb			
66 kb	180 kb	180 kb		
44 kb	180 kb	180 kb	180 kb	
30 kb	180 kb	180 kb	180 kb	100 kb

Note: The 180-kb maximum data rate for 2870 selector subchannels pertains to attachment of magnetic tape devices; timing factors other than data rates may preclude attachment of direct-access storage devices that have lesser data rates. Also, note that when other channels in addition to the 2870 are in operation, the total system I/O data rate must be analyzed.

CHANNEL-TO-CHANNEL ADAPTER FEATURE

A channel-to-channel adapter is available as an optional feature on the 2860. The adapter provides a path for operations to take place between two channels and synchronizes those operations. It may be used in multiple-processor or single-processor systems: in a multisystem, to achieve rapid communications between the channels of two System/360 models; in a single system, to move blocks of data from one main storage area to another.

The adapter uses one control-unit position on each of the two channels, but only one of the two connected channels requires the feature. In the Model 195, one adapter may be installed per selector channel.

When the 2870 Multiplexer Channel is connected to another channel, the channel-to-channel adapter is installed on the other channel, not on the 2870.

For restrictions on channel attachments for another IBM System/360 model used with the Model 195, refer to the Systems Reference Library (SRL) functional characteristics publication for that model.

The system control panel on the system console contains the switches, keys, and indicator lights to operate and control the system (CPU, storage, channels, on-line control units, and input/output devices). Off-line control units and I/O devices, though part of the system environment, are not considered part of the system.

System controls are divided into three classes: operator control, operator intervention, and customer engineering control. This section of the manual discusses operator control and operator intervention.

Using the control panel, the operator can perform the following system control functions:

1. Reset the system.
2. Store and display information in storage, registers, and program status word (PSW).
3. Load initial program information.

SYSTEM CONTROL FUNCTIONS

System Reset

The system-reset function resets the CPU, channels, and on-line nonshared control units and I/O devices.

The CPU is placed in the stopped state, and all pending interrupts are eliminated. All error-status indicators are reset to zero.

In general, the system is placed in such a state that processing can be initiated without machine checks occurring, except those caused by subsequent machine malfunction.

Addresses in the data directory of the high-speed buffer storage are reset to zero by a system reset. Subsequently, the contents of the high-speed buffer storage are replaced, block by block, as required by ensuing fetch requests.

The reset state for a control unit or device is described in the appropriate Systems Reference Library (SRL) publication. A system-reset signal from a CPU resets only the functions in a shared control unit or device belonging to that CPU. Any function pertaining to another CPU remains undisturbed.

The system-reset function is performed when the system-reset key is pressed, when the PSW-restart key is pressed, when initial program loading is initiated, or when a power-on sequence is performed.

Programming Note: If a system reset occurs in the middle of an operation, the contents of the PSW and of the result registers or storage locations are unpredictable. If the CPU is in the wait state when the system reset is performed, and no I/O operation is in progress, this uncertainty does not exist.

A system reset does not correct parity in registers or storage. Because a machine check occurs when information with incorrect parity is used, the incorrect information should be replaced by loading new information.

Store and Display

The store-and-display function permits manual intervention in the progress of a program. The storing and/or displaying of data may be provided by a supervisor program, proper I/O equipment, and the interrupt key.

In the absence of an appropriate supervisor program, the controls on the operator intervention panels allow direct storing and displaying of data. This is done by placing the CPU in the stopped state and subsequently storing and/or displaying information in main storage, in general and floating-point registers, and in the instruction-address part of the PSW. The stopped state is achieved when the stop key is pressed, when single instruction execution is specified and the instruction has been executed, or when a preset address is reached.

In Model 195, the transition from operating to stopped state includes completing all instructions that were decoded at the time stopped state was called for. The store-and-display function is achieved by use of the store, display, and set CAR keys, address switches, data switches, store/display/storage select switch, scan key, and CRT display switch. Once the desired intervention is completed, the CPU can be started again.

Normal stopping and starting of the CPU in itself does not cause any alteration in program execution other than in the time element involved in the transition from operating to stopped state.

Machine checks occurring during store-and-display operations do not log immediately but create a pending log condition that can be removed by a system reset or check reset. The error condition, when not disabled, forces a log-out and a subsequent machine check interrupt when the CPU is returned to the operating state.

Initial Program Loading

Initial program loading (IPL) is provided for initiation of processing when the contents of storage or the PSW are not suitable for further processing.

Initial program loading is initiated manually by selecting an input device with the load-unit switches and pressing the load key.

Pressing the load key causes a system reset, turns on the load light, turns off the manual light, and initiates a read operation from the selected input device. When reading is completed satisfactorily, a new PSW is obtained, the CPU starts operating, and the load light is turned off.

The system reset suspends all instruction processing, interrupts, and timer updating and also resets all channels, online nonshared control units, I/O devices, and the data directory. The contents of general and floating-point registers remain unchanged.

When IPL is initiated, the selected input device starts transferring data. The first 24 bytes read are placed in storage locations 0-23. Protection, program-controlled interrupt, and a possible incorrect length indication are ignored. The doubleword read into location 8 is used as the channel command word (CCW) for reading more than 24 bytes. When chaining is specified in this CCW, the operation proceeds with the CCW in location 16. Either command chaining or data chaining may be specified.

When the device provides channel end for the last operation of the chain, the I/O address is stored in bits 21-31 of the first word in storage. Bits 16-20 are made zero. Bits 0-15 remain unchanged.

The CPU subsequently fetches the doubleword in location 0 as a new PSW and proceeds under control of the new PSW. The load light is turned off. No I/O interrupt condition is generated. When the I/O operations and PSW loading are not completed satisfactorily, the CPU idles and the load light remains on.

Programming Notes: Initial program loading resembles a 'start I/O' that specifies the I/O device selected by the load-unit switches and a zero protection key. The CCW for this 'start I/O' is simulated by CPU circuitry and contains a read command, zero data address, a byte count of 24, chain-command flag on, program-controlled-interrupt flag off, chain-data flag off, and skip flag off. The CCW has a virtual address of zero.

Initial program loading reads new information into the first six words of storage. Because the remainder of the IPL program may be placed in any desired section of storage, it is possible to preserve such areas of storage as the timer and PSW locations, which may be helpful in program debugging.

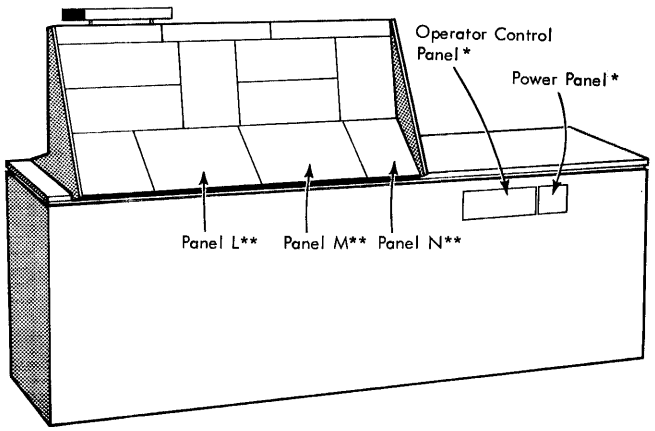
If the selected input device is a disk, the IPL information is read from track 0.

The selected input device may be a channel-to-channel adapter connecting the channels of two CPU's. After a system reset is performed and a read command is issued to this adapter by the requesting CPU, the adapter sends an attention signal to the addressed CPU, which then should issue the write command necessary to load a program into main storage of the requesting CPU.

When the PSW in location 0 has bit 14 set to 1, the CPU is in the wait state after the IPL procedure. (The manual system and load lights are off, and the wait light is on.) Interrupts that become pending during IPL are taken before instruction execution.

CONTROLS

System controls are divided into three groups: operator control, operator intervention, and customer engineering control. Figure 9 shows the location of panels used to perform the operator control and the operator intervention functions. Figure 10 shows controls and indicator lights used in operator control, and Figure 11 shows the controls and indicator lights used in operator intervention.



* Panels containing operator controls. See Figure 10.
** Panels containing operator intervention controls. See Figure 11.

Figure 9. System Console Panels

Operator Control

The operator-control section of the system console panel (Figure 9) contains controls and indicator lights required by the operator when the CPU is operating under full supervisor control. Under supervisor control, a minimum of direct manual intervention is required because the supervisor performs operations similar to store and display.

To control another System/360 processor, a second set of controls and indicator lights (an optional feature) can be provided on the operator control panel (Figure 10). One set may be duplicated as a remote panel on a stand-alone operator's console (IBM 2150 Console or IBM 2250 Display Unit Model 1). Provision for the remote panel is a standard feature.

The main functions provided by the operator controls are the control and indication of power, the indication of system status, operator-to-machine communication, and initial program loading.

The operator controls and indicator lights (Figure 10) are:

Name	*Type*
Display Power Off	Key
Display Power On	Key (backlighted)
Emergency Pull	Pull switch
Interrupt	Key
Load	Key
Load	Indicator light

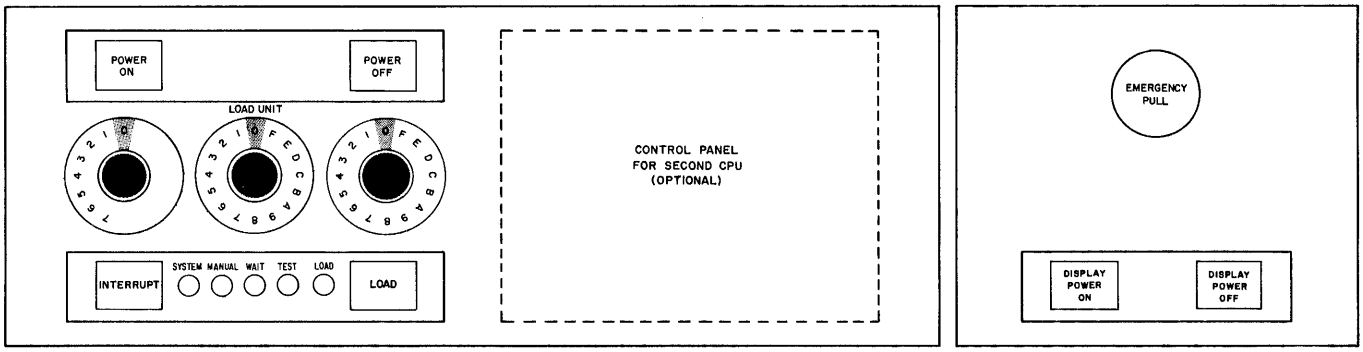


Figure 10. Operator Control Panel and Power Panel

Name	*Type*
Load Unit	Rotary switches (3)
Manual	Indicator light
Power Off (System)	Key
Power On (System)	Key (backlighted)
System	Indicator light
Test	Indicator light
Wait	Indicator light

Display Power Off

The display-power-off key on the power panel initiates the power-off sequence of the display console integrated with the system control panel. The contents of the upper 4,096 bytes of display console buffer storage (containing format control data) are preserved after a power-off to the display console.

Display Power On

The display-power-on key on the power panel initiates the power-on sequence of the display console integrated with the system control panel. While power is on the display console, the key is backlighted white. The contents of the upper 4,096 bytes of display console buffer storage (containing format control data) are preserved after a power-on to the display control.

Emergency Pull

Pulling the emergency-pull switch turns off all power, beyond the power-entry terminal, on every unit that is part of the system or that can be switched onto the system. Therefore, the switch controls the system proper and all off-line and shared control units and I/O devices. A second emergency-pull switch is on the power distribution unit.

The switch latches in the out position and can be restored to its in position by maintenance personnel only.

Interrupt

The interrupt key is pressed to request an external interrupt. The interrupt is taken when it is allowed and when the CPU is not stopped. Otherwise, the interrupt remains pending.

When the interrupt is taken, bit 25 in the interrupt-code portion of the current PSW is set to 1 to indicate that the interrupt key is the source of the external interrupt. The key is effective while power is on the system.

Load (Key)

The load key begins initial program loading. (See "Initial Program Loading.") It is effective while power is on the system.

Load (Light)

The load light is on during initial program loading. It turns on when the load key is pressed and turns off after the new PSW is successfully loaded.

Load Unit

The three load-unit switches provide the 11 rightmost I/O address bits of the device to be used for initial program loading.

The leftmost switch has eight positions, labeled 0-7. The other two switches are 16-position switches labeled hexadecimally 0-F.

Manual

The manual light is on when the CPU is in the stopped state. Several manual controls are effective only when the CPU is stopped, that is, when the manual light is on.

Power Off (System)

The power-off key initiates the power-off sequence of the system. The contents of main storage (but not the keys in storage associated with the protection features nor the contents of the high-speed buffer storage) are preserved if the CPU is in the stopped state and all I/O operations are complete. The key is effective while power is on the system.

Power On (System)

The power-on key initiates the power-on sequence of the system. As part of the power-on sequence, a system reset is performed in such a way that the system performs no instructions or I/O operations until explicitly directed. The contents of main storage are preserved.

The power-on key is backlighted white when power is on the entire system. The key is backlighted red during the power-on sequence and when any remote/local power control switch in the power system is in the local position. If there is a loss of power in some section of the processor, main storage units, or channels, the light will change from white to red. The power-on key is effective only when the emergency-pull switch is in its in position.

System

The system light is on when the CPU-cluster usage meter or customer engineering meter is running. These meters are on the display console.

The states indicated by the wait and manual lights are independent of each other; however, the state of the system light is not independent of the states of the wait and manual lights. The possible conditions when power is on are:

System Light	*Manual Light*	*Wait Light*	*CPU State*	*I/O State*
Off	Off	Off	*	*
Off	Off	On	Wait	Not working
Off	On	Off	Stopped	Not working
Off	On	On	Stopped, Wait	Not working
On	Off	Off	Running	Undetermined
On	Off	On	Wait	Working
On	On	Off	Stopped	Working
On	On	On	Stopped, Wait	Working

* Abnormal Condition

Test

The test light is on when a manual control is not in its normal position or when a maintenance function is being performed for the CPU, channels, or main storage.

Any abnormal setting of a switch on the system control panel or on any separate maintenance panel for the CPU, main storage, or channels that can affect the normal operation of a program causes the test light to go on.

The test light may be on when certain diagnostic functions are activated or when certain abnormal circuit-breaker or thermal conditions occur.

The test light is on when any of the following controls is not in its normal position:

Address compare
Address increment (BSM/beat)
Address increment (1-up store/display)
Block scan
Mach check stop
CRT display and tape operation
Disable interval timer
Enter instruction
Frequency margin range
Inhibit overlap
Inhibit replace buffer 1, 2, 3, or 4
Maintenance console test, rotary
MCW active
MCW to Chan sim
Rate
Repeat
Reverse CBR PTYS/Block DD reset
Storage reconfiguration
Storage test (store/fetch)

Wait

The wait light is on when the CPU is in the wait state. The wait state exists whenever bit position 14 of the current PSW contains a 1. The wait state can be changed to the running state only by loading a new PSW in which bit position 14 contains a 0; it cannot be changed by pressing the system reset key.

Operator Intervention

This section of the system control panel (Figure 11) contains controls required by the operator to intervene in normal programming operation. These controls are intermixed with the customer engineering controls, which are not used by the operator.

Operator intervention provides the system-reset function and the store-and-display function.

The intervention controls are:

Name	*Type*	*Panel*
Address	Key switches	M
Address Compare	Rotary switch	L
Block Scan	Key switch	L
CRT Display and Tape Operation	Rotary switch	N
Data (0-63 plus eight parity)	Key switches	M
Display	Key	M
Force Machine Check Interrupt	Key	L
PSW Restart	Key	N
Rate	Rotary switch	L
Scan	Key	N
Set CAR	Key	M
Set IC	Key	N
Set PSW	Key	N
Start	Key	L
Stop	Key	L
Store	Key	M
Store/Display/Storage Select	Rotary switch	N
System Reset	Key	N

Address (Panel M)

The 27 address switches (24 address switches and three parity switches) are locking key switches and are used to address a storage location or register, as specified by the address compare, CRT display and tape operation, or store/display/storage select switches.

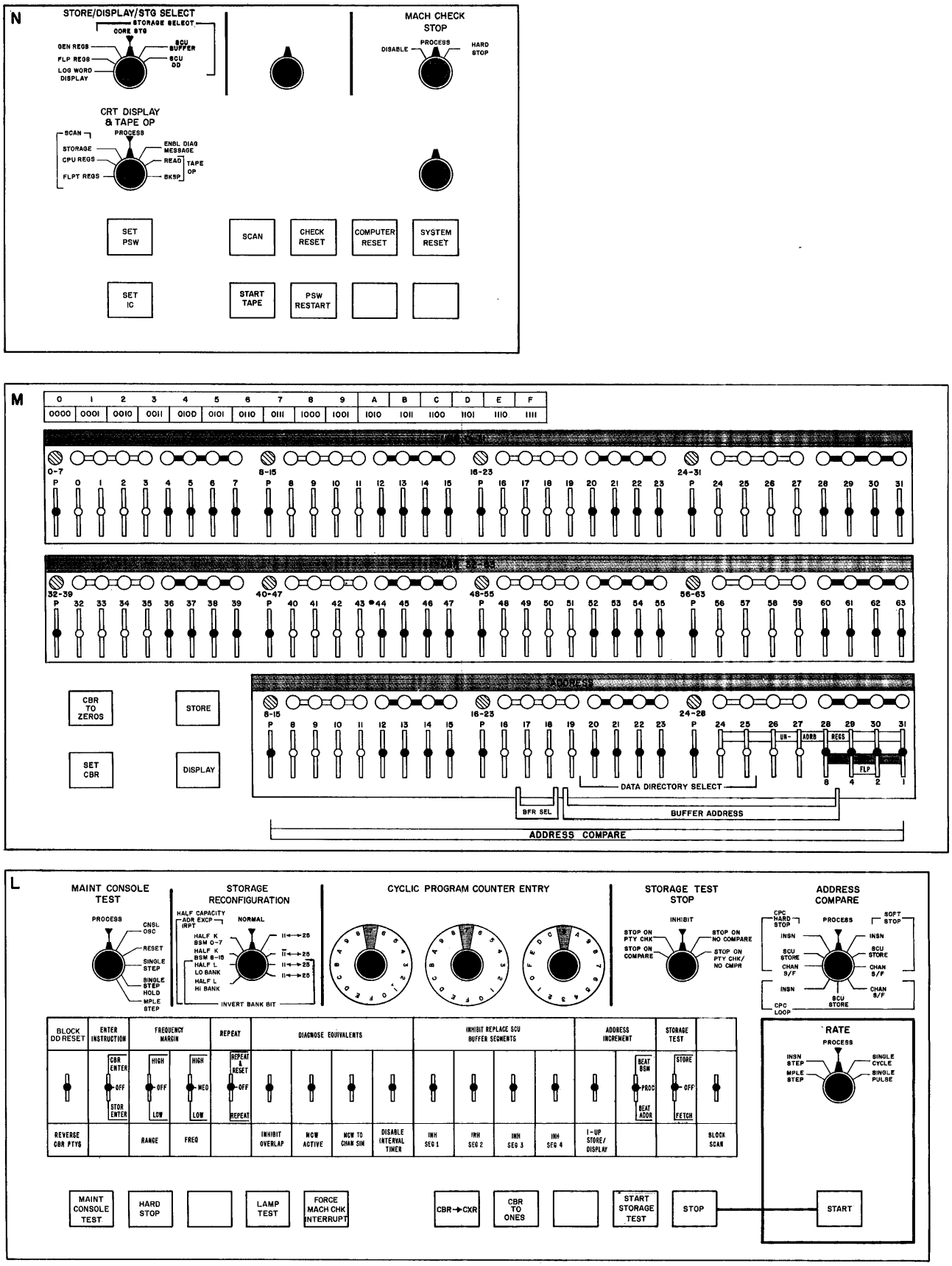


Figure 11. Operator Intervention Controls and Indicator Lights

The address switches have locking set and reset positions. The associated bit position of the console address register is set or reset according to the position of the switch when the set CAR key is pressed. (When the switch is down, the bit position is set to 1; in the center position, the bit position is set to 0.)

The console address register can also be altered by the address stepping circuitry. The contents of the console address register are indicated continuously.

The three low-order bit positions and positions 8 and 9 are not used in main-storage addressing and are not affected by the address stepping circuitry. Thus, main-storage addressing always specifies a doubleword boundary. In the performance of the address-compare and for register selection, however, the three low-order bit positions are used.

Parity for each byte is indicated by the parity indicators in the address register and is generated automatically whenever the address register is used. The three parity switches do not affect address usage; when activated, they turn the associated address-register bit on or off, but parity is automatically updated in the address register before the address is used.

Address Compare (Panel L)

The address-compare rotary switch controls synchronizing pulses, program loops, and machine stops by means of address comparisons during instruction-fetch or data-store operations. The switch has ten active positions, seven of which are for customer engineering use (channel soft stop setting, the CPC hard stop settings, and the CPC loop settings). If the switch is in other than the process position, the test light will be on.

The address-compare switch can be manipulated among the three customer settings, described below, without disrupting CPU operation, other than by causing the address-comparison stop.

Process: When the switch is in this position, a synchronizing pulse is provided when the address specified in the address register matches the instruction address. The pulse occurs when decoding of the instruction begins and may be used to synchronize an oscilloscope at the start of an instruction. This position is the normal operating position for the switch.

Insn Soft Stop: When the switch is in this position, the CPU enters the stopped state when the address specified in the address register matches the instruction address. This position may be used to control the stopping point of a program. The instruction-fetch operation, all other outstanding operations, and all pending interrupts that are allowed are completed before the CPU enters the stopped state.

SCU Store Soft-Stop: When the switch is in this position, the CPU enters the stopped state when the address specified in the address register matches a main-storage address specified in any CPE store operation or in an I/O store operation into main storage. The store operation, all other outstanding operations, and all pending interrupts that are allowed are completed before the CPU enters the stopped state.

Block Scan (Panel L)

The block scan key switch inhibits scanning of registers or main storage during a CRT-display operation.

When the switch is in the centered position, pressing the scan key initiates the normal scan function. (See "Scan".)

When the switch is in the down position, pressing the scan key immediately after setting the block scan switch causes the information displayed during the last display operation to be displayed again; the data specified by the CRT display switch and related controls is not scanned, and no data is transferred to the buffer storage of the display console.

When the switch is in the on position, the test light is on.

CRT Display and Tape Operation (Panel N)

This switch connects the display console with either an I/O channel or the system control panel. The CRT-display portion of this switch, labeled "scan," determines the type of display to be produced on the CRT when the scan key is pressed. The enable diagnostic message and tape operation portions of this switch are for customer engineering use.

When the switch is in any position other than process, the test light is on.

Process: When the switch is in this position, the display console is connected to either a 2860 channel or a 2870 selector subchannel and is under program control.

Storage: When the switch is in this position, the type of storage scan operation performed is determined by the storage select setting of the store/display/storage select switch.

CPU Regs: When the switch is in this position and the CPU is in the stopped state, pressing the scan key initiates an operation in which the contents of the CPU registers are placed, in order, into the console buffer register and then transferred to preassigned buffer storage areas of the display console; then this data and the identifications of the registers are displayed on the CRT (Figures 12 and 13).

Operating the block scan switch immediately before pressing the scan key inhibits the fetch-and-transfer operation; when the scan key is then pressed, data displayed during the last CPE-register-display operation is displayed again.

FLPT Regs: When the switch is in this position and the CPU is in the stopped state, pressing the scan key initiates an operation identical to that described for the CPU-regs position, except that the floating-point registers are scanned and displayed. Operating the block scan switch before pressing the scan key inhibits the fetch-and-transfer operation; when the scan key is then pressed, data displayed during the last floating-point display operation is displayed again (Figure 13).

```
                                PROCESSOR DISPLAY

TEMP 1  6A 00 94 76 12 22 47 20                 TEMP 2  90 44 23 00 6A 00 94 66
IBFR 0  58 20 94 02 68 00 94 78                 IBFR 4  90 44 23 00 6A 00 94 65
     1  13 22 50 20 94 02 6C 00                      5  60 00 94 EE 60 00 95 6E
     2  94 FE 47 80 90 38 23 00                      6  58 20 94 02 68 00 94 76
     3  6A 00 94 70 12 22 47 20                      7  13 22 50 20 94 02 6C 00

OP REF  50 20 94 02        LB       00 FF EE EE     IR       7F         SLT     7F
OP STO  6A 00 63           UB       00 FF F8 FB     AOC      78         SLCIR   7E

GPR 0   00 00 00 00        GPR 1    11 11 11 11     ISR      78         SLCB    F0
    2   22 22 22 22             3   33 33 33 33                         SLCX    F0
    4   44 44 44 44             5   55 55 55 55     DWCR     1F         SVR1    F0
    6   66 66 66 66             7   77 77 77 77     BYT BFR  FF         XECOR   FF
    8   88 88 88 88             9   99 99 99 99     L REG    FF       D BUFF 07 FF
    A   AA AA AA AA             B   BB BB BB BB
    C   CC CC CC CC             D   DD DD DD DD              TEMP     00 FF EE EE
    E   EE EE EE EE             F   FF FF FF FF              WR       00 00 FF EF

                                                                FXOS 1  58 A0 60
                                                                     2  1A AA 00
FXB A   AA AA AA AA        FXB B    BB BB BB BB                     3  50 A0 03
    C   CC CC CC CC             D   DD DD DD DD                     4  5A A0 10
    E   EE EE EE EE             F   FF FF FF FF                     5  18 BA 00
                                                                     6  1A BB 00

L ACC   00 00 00 00 00     R ACC   00 00 00 00 0

          LK  ADDRESS  RS  SA
TAR 1     00  3E FC E8 00  00          B CTR 02     STO KEY 01
    2     02  3E FC E8 04  16          C CTR 02     PRO KEY 01
    3     03  3E FC E8 15  1E          P CTR 00

SIM REG BC  ADDRESS  MK
    1   F2  3E FC E8 0F   SAR 1  SDB 11 11 11 11 11 11 11 11
    2   F3  3E FC E8 F0   SAR 2  SDB 22 22 22 22 22 22 22 22
    3   F4  3E FC E8 FF   SAR 3  SDB 33 33 33 33 33 33 33 33

                      P1 P2 ADDRESS    MK1 MK2
                STACK 00 00 00 00 00    00  00

CHAN SDB  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00

CHAN SBO  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
```

Note: A blank following any byte indicates correct parity; an asterisk denotes incorrect parity.

Figure 12. Sample Processor Display

```
                            FLOATING POINT DISPLAY

FLB 1  00 00 00 00 00 00 00 00 0      A1 SRC  00 00 00 00 00 00 00 00
    2  00 00 00 00 00 00 00 00 0         SNK  00 00 00 00 00 00 00 00
    3  00 00 00 00 00 00 00 00 0      A2 SRC  00 00 00 00 00 00 00 00
    4  00 00 00 00 00 00 00 00 0         SNK  00 00 00 00 00 00 00 00
    5  00 00 00 00 00 00 00 00 0      A3 SRC  00 00 00 00 00 00 00 00
    6  00 00 00 00 00 00 00 00 0         SNK  00 00 00 00 00 00 00 00

FLB 0  00 00 00 00 00 00 00 00       ADR INT 00 00 00 00 00 00 00 00 0
    2  00 00 00 00 00 00 00 00       AD RSLT 00 00 00 00 00 00 00 00
    4  00 00 00 00 00 00 00 00
    6  00 00 00 00 00 00 00 00
                                     MUL DEC 00 00 00 00 00 00 00
FLOS 0 00 00        RSLT EXP         CSA SUM 00 00 00 00 00 00 00 00 00
     1 00 00         0  PSI 00        CSA CAR 00 00 00 00 00 00 00 00 00
     2 00 00        L/R PS1 00       PA RSLT 00 00 00 00 00 00 00 00 00
     3 00 00         0  PS2 00
     4 00 00        L/R PS2 00          MD1 SRC 00 00 00 00 00 00 00 00
     5 00 00             EP 00              SNK 00 00 00 00 00 00 00 00
     6 00 00                            MD2 SRC 00 00 00 00 00 00 00 00
     7 00 00                                SNK 00 00 00 00 00 00 00 00

      EP SRC 00 00 00 00 00 00 00 00      00 00 00 00 00 00 00 00
         SNK 00 00 00 00 00 00 00 00      00 00 00 00 00 00 00 00
         SUM 00 00 00 00 00 00 00 00      00 00 00 00 00 00 00 00 0
```

Note: A blank following any byte indicates correct parity;
an asterisk denotes incorrect parity.

Figure 13. Sample Floating-Point Display

Data (Panel M)

The 72 data switches (64 data switches and eight parity switches), labeled CBR, are nonlocking key switches and are used to enter data into selected areas of the CPU or storage.

The contents of the console buffer register (CBR) are normally the output of the data switches; the contents of this register are altered by manipulation of these switches, by a storage fetch operation or by a log-word or register display.

The switches have nonlocking set and reset positions; they are in a neutral position when they are not being operated. The associated bit position of the buffer register is set or reset depending on the position to which the switch is operated. (When the switch is operated down, the bit position is *set.*) The contents of the console buffer register are indicated continuously so that any manipulation of the data switches can be seen.

Data is stored according to the contents of the console address register and the setting of the store/display/storage select switch. The store key must be pressed to initiate the store operation. Parity is automatically generated whenever the data is transferred.

Data cannot be stored into the high-speed buffer-storage data directory from the data switches.

Display (Panel M)

The display key is pressed to place data into the console buffer register, as determined by the setting of the store/display/storage select switch and by the contents of the console address register. The lights for the console buffer register continuously display the contents of that register.

When the designated location is not available, the displayed information is unpredictable.

Force Machine Check Interrupt (Panel L)

Pressing this key causes a hard stop. A log-out, CPU and check reset, and a machine check interrupt occur. If the rate switch is in the single-cycle position, the sequence stops at the beginning of the machine check interrupt.

PSW Restart (Panel N)

The PSW-restart key is pressed to initiate the following operations in sequence:

1. System reset.
2. Loading a new PSW from location 0.
3. Instruction fetching, starting at the new program location specified by the new PSW.
4. Execution of instructions as specified by the setting of the rate switch.

Rate (Panel L)

The rate rotary switch indicates in which way the instructions are to be performed. The test light is on if the rate switch is set to any position other than process.

The position of the rate switch should be changed only while the CPU is in the stopped state. Otherwise, results are unpredictable.

Process: When the switch is in this position, the system operates at normal speed after the start key is pressed. The decoding of instructions is halted by pressing the stop key.

Insn Step: When the switch is in this position, one instruction is completely executed each time the start key is pressed. The CPU automatically halts in the stopped state. When the start key is pressed, but before the one instruction is processed, interrupts that were allowed but became pending during the stopped state are processed before execution of the next instruction.

Mple Step: When the switch is in this position, an instruction is executed every 100 milliseconds for as long as the start key is pressed. The CPU automatically halts in the stopped state when the start key is released.

Single Cycle: This position is for customer engineering use.

Single Pulse: This position is for customer engineering use.

Scan (Panel N)

The scan key is pressed to produce a display on the cathode-ray tube of the display console. The display produced is determined by the settings of the CRT display and tape operation switch and the store/display/storage select switch.

Whether the display represents updated information depends on whether the block scan switch has been operated. Pressing the scan key immediately after the block scan switch has been placed in the down position causes the information displayed during the last display operation to be displayed again.

This key is effective only while the CPU is in the stopped state.

Set CAR (Panel M)

Pressing the set CAR key transfers the setting in the 24 address switches to the console address register.

Set IC (Panel N)

The set IC (instruction counter) key is pressed to enter the contents of bit positions 40-63 of the console buffer register into bit positions 40-63 (the instruction address part) of the current PSW.

This key is effective only while the CPU is in the stopped state.

Set PSW (Panel N)

The set PSW (program status word) key is pressed to enter the contents of bit positions 0-15 and 32-63 of the console buffer register into bit positions 0-15 and 32-63 of the current PSW.

The key is effective only while the CPU is in the stopped state.

Start (Panel L)

The start key is pressed to start instruction execution as specified by the setting of the rate switch.

Pressing the start key after a normal halt causes instruction processing to continue as if no halt had occurred, provided the rate switch is in the process, instruction-step, or multiple-step position.

Pressing the start key after system reset without first having introduced a new instruction address yields unpredictable results.

Pending interrupts that are allowed will be honored before the first instruction is executed.

The key is effective only while the CPU is in the stopped state.

Stop (Panel L)

The stop key is pressed to terminate machine operation without destroying system status. The CPU enters the stopped state after all previously decoded instructions have been executed, after all pending interrupts have been processed, and after any interrupts that became pending while the CPU was in the decode or stop-decode state have been processed.

When the CPU enters the stopped state, the manual light turns on. After stopped state has been entered, no interrupts are processed.

The stop key is active while power is on the system.

Store (Panel M)

The store key is pressed to store data from the console buffer register into the location specified by the setting of the store/display/storage select switch and by the contents of the console address register.

Store protection is ignored. When the location designated by the console address register and by the setting of the store/display/storage select switch is not available, no data is stored.

If data is stored into a main storage location that is also resident in the high-speed buffer storage, the buffer storage block containing this information is invalidated to maintain the integrity of storage.

Data cannot be stored into the high-speed buffer-storage data directory from the system control panel.

The store key is active only while the CPU is in the stopped state.

Store/Display/Storage Select (Panel N)

The store/display positions of this rotary switch specify the sections of the CPU that are addressed by the console address register when the store and display keys are used. Store data is set in the 72 data switches.

The storage select positions of the rotary switch specify the parts of storage affected by the console address register when the CRT display and tape operation switch is in the *storage* position.

Gen Regs: When the switch is in this position and the CPU is in the stopped state, the contents of the general register (indicated by the console address register) can be placed in the console buffer register by pressing the display key, or can be replaced by the contents of the console buffer register by pressing the store key. The contents of the general register are displayed left-justified in the console buffer register. For store operations, the data must be placed in the upper half of the console buffer register.

FLP Regs: When the switch is in this position and the CPU is in the stopped state, the contents of the floating-point register (indicated in the address register) are placed in the console buffer register by pressing the display key, or are replaced by the contents of the console buffer register by pressing the store key.

Log Word Display: When the switch is in this position and the CPU is in the stopped state, the log word that has its pseudo-address (01-8E) in the console address register is displayed in the console buffer register lights.

Core Storage: In this position, 16 doublewords of main storage are displayed. The starting doubleword address must be placed in the console address register (Figure 14).

STORAGE DISPLAY

ADDRESS	DATA
000000	00 00 00 00 00 00 00 00
000008	01 02 03 04 05 06 07 08
000010	09 0A 0B 0C 0D 0E 0F 00
000018	11 12 13 14 15 16 17 18
000020	19 1A 1B 1C 1D 1E 1F 11
000028	20 21 22 23 24 25 26 27
000030	28 29 2A 2B 2C 2D 2E 2F
000038	30 31 32 33 34 35 36 37
000040	38 39 3A 3B 3C 3D 3E 3F
000048	40 41 42 43 44 45 46 47
000050	48 49 4A 4B 4C 4D 4E 4F
000058	50 51 52 53 54 55 56 57
000060	58 59 5A 5B 5C 5D 5E 5F
000068	60 61 62 63 64 65 66 67
000070	68 69 6A 6B 6C 6D 6E 6F
000078	70 71 72 73 74 75 76 77

Note: A blank following any byte indicates correct parity; an asterisk denotes incorrect parity.

Figure 14. Sample Main Storage or Buffer Storage Display

SCU Buffer: In this position, 16 doublewords of high-speed buffer storage are displayed. Console address register bit positons 17 and 18 select the buffer storage segment, and bit positions 19-28 select the first doubleword within the segment for display. The address displayed on the screen represents the buffer storage location only (Figure 14).

SCU DD: In this position, 16 doublewords from the data directory, including the chronology array, are displayed. Bit positions 19-25 in the console address register address one of the 128 locations in the data directories and chronology array (Figure 15). For each address, reading from left to right, characters 1, 2, and 3 of the display refer to the contents of data directory 1; characters 5, 6, and 7 to data directory 2; characters 9, 10, and 11 to data directory 3; and characters 13, 14, and 15 to data directory 4. Characters 4, 8, and 12 are hexadecimal representations of special chronology array bits (denoted by C above the character). Character 16 is always F.

ADDRESS	DIRECTORY DATA 1 C	2 C	3 C	4
000000	00 00	00 00	00 00	00 0F
000040	01 25	45 66	89 09	23 4F
000080	02 35	45 66	77 89	99 0F
0000C0	03 36	44 99	22 1A	45 9F
000100	04 09	22 36	44 09	88 9F
000140	05 1A	33 15	22 16	77 8F
000180	04 25	44 36	55 89	66 7F
0001C0	05 35	55 49	77 96	55 6F
000200	06 46	66 95	99 09	88 5F
000240	07 0A	77 19	88 15	33 4F
000280	08 1A	88 15	99 26	22 3F
0002C0	09 45	99 66	44 66	88 2F
000300	01 46	88 65	23 45	67 3F
000340	09 5A	98 6A	87 99	65 4F
000380	10 56	43 56	33 99	45 9F
0003C0	08 55	21 95	63 05	87 5F

Note: A blank following any byte indicates correct parity; an asterisk denotes incorrect parity.

Figure 15. Sample Data Directory Display

System Reset (Panel N)

The system-reset key is pressed to reset on-line channels, control units, and CPU controls to their initial states. All check indicators are reset and the contents of the high-speed buffer storage data directory are cleared. The current PSW, data flow registers, keys in storage, and main storage are not reset. The CPU is placed in the stopped state, and all pending interrupts are eliminated. The reset function does not affect any off-line or shared devices.

This key is active while power is on the system.

Key Switch and Meters

The customer usage and the customer engineering (CE) meters for the CPU cluster are on the left side of the display console.

The Model 195 CPU cluster includes: CPU, processor storage, processor system console, CPU power supplies, and power and coolant distribution units.

A key switch controls which meter is to run while the system is in operation, that is, initiating, executing, or completing instructions, including I/O and assignable unit operations.

Although the Model 195 performs CPU operations in a highly parallel fashion, no elaborate optimization plan is required to prepare programs for CPU processing. For the most part, they may be written in a straightforward IBM System/360 code. If a program will benefit by some modification, however, the following suggestions may be helpful.

1. Place index loading and incrementing instructions well ahead of instructions that use them for address generation. In a loop, a convenient place for an indexing instruction such as 'add' (AR) is at the end of the loop, just before a 'branch on index low or equal'; by the time the branch is completed, the index registers will be ready for use.
2. The instructions 'load address', 'branch on count' (BCT, BCTR), 'branch on index low or equal', and 'branch on index high' use the address adder to change a general register. As suggested in item 2, make sure that the registers required are available.
3. The 'load address' instruction requires three cycles that cannot be overlapped; it is also subject to delays if registers are unavailable. Instructions such as 'add' (A, AR) require only one unoverlapped cycle and are not subject to delays if registers are unavailable. In most cases, therefore, replace the 'load address' instruction with an AR instruction. In some situations, the 'load address' instruction is preferable:
   a. When the register to be used is needed for addressing by the next instruction.
   b. When the fixed-point execution element is busy with a lengthy instruction sequence and a register is needed for addressing within the next few cycles.
   c. When the condition code must not be changed.
4. Because the Model 195 fetches and stores doublewords, align operands on doubleword boundaries for faster operations. Operands that are not aligned to doubleword boundaries can be used in fixed- and floating-point arithmetic and in variable field length (VFL) operations, but performance is affected adversely.
5. In normal coding, a condition-setting instruction immediately precedes most 'branch on condition' instructions. On the Model 195, place neutral instructions, such as those dealing with loads and stores, between the condition-setting instruction and the conditional branch.
6. Avoid storing into the next several words of the instruction stream.
7. Whenever possible, contain a loop in the instruction stack so that it is executed in loop mode. (See the discussion of loop mode in "Instruction Processor.")
8. Because all instructions that store data use the same three store address registers (SAR) and the same three store data buffers (SDB), if a fourth store is encountered before a store address register is freed, the instruction processor must wait. When possible, avoid more than three stores in a row. For example, if it is necessary to store data from six registers by using one 'store multiple' instruction, only three SAR's are required if the first address started on a doubleword boundary; four stores are required otherwise.
9. When only two registers are to be loaded, using two load (L) instructions is faster than using a 'load multiple' instruction; however, when four or more registers are to be loaded, prefer the 'load multiple' instruction. Also, the 'store multiple' instruction is usually better than repeated 'store' (ST) instructions because it requires fewer SAR's and SDB's.
10. Avoid repeated accesses to different doublewords in the same storage module; conflicts result. For example, with 16-way interleaving of processor storage, a Model 195K that is storing by column into a 16 x 16 array of doublewords is storing consecutively into the same storage module. This does not take advantage of the interleaving, nor of the buffer storage because it is a store operation.
11. Try not to use the interrupt mechanism to effect logical program branches; operation is slow because of the required program interlocks. Also, some logical program operations available through the interrupt mechanism in slower, more serial processors are not available in the Model 195 if the interrupt in question is not precise.
12. Efficiency is increased by eliminating short records. Avoid excessive use of SIO for small quantities of data, because the I/O device response time is included as a part of the instruction.

Appendix B: Timing Considerations

For other models of the IBM System/360, average times can be given for each instruction. For a parallel system like the Model 195, however, no average times are meaningful, because the amount of overlap varies from program to program.

The following information gives an appreciation of some major aspects of timing in the Model 195 but it is not intended to be comprehensive. In the discussion, "cycle" refers to a major-machine-cycle time of 54 nanoseconds.

Instruction Processor Delays

Any of the following conditions delay the instruction processor:

1. The next instruction is unavailable.
2. The system is in conditional mode, and the next instruction is an instruction to be executed by the instruction processor or is a variable-field-length instruction. (An unconditional branch or a no-operation instruction, however, can be executed in conditional mode.)
3. A general register is unavailable for the addressing of the next instruction.
4. A general register is unavailable for modification by the next instruction – a condition that applies only to an instruction-processor instruction, such as 'load address' or 'branch on index low or equal,' which changes a general register.
5. The next instruction requires an address generation, but a previous instruction will not be able to complete its address generation for another cycle.
6. The next instruction requires a fixed-point buffer register, but all fixed-point buffer registers are busy.
7. The next instruction requires a floating-point buffer register, but all floating-point buffer registers are busy.
8. The next instruction is a fixed-point operation, but the fixed-point operation stack is full.
9. The next instruction is a floating-point operation, but the floating-point operation stack is full.
10. The next instruction requires a store, but all store address registers are busy.
11. An instruction is decoded whose execution is delayed until the completion of all previously decoded instructions.

Transmission Time

Each of the following transmissions requires one cycle. In most cases, these transmissions take place concurrently with other operations, but instances may occur in which delays due to these transmissions will directly affect the timing.

1. A fixed-point or floating-point operation from the instruction processor to the fixed-point operation stack or the floating-point operation stack, respectively.
2. An activate or cancel signal from the instruction processor to the fixed-point operation stack or the floating-point operation stack.
3. A condition-code indication from an execution unit to the instruction processor.
4. A general-register-available indication from the fixed-point execution element to the instruction processor.
5. A buffer-free indication from the fixed-point execution element or the floating-point execution element to the instruction processor.
6. An operation-stack-position-free indication from the fixed-point execution element or the floating-point execution element to the instruction processor.
7. A store-address-register-free indication from the storage control unit to the instruction processor.

Branches

When loop mode is not set, the first cycle of a branch is the usual decoding in the instruction processor. The next two cycles are address generations for the target and target +1 doublewords; the two temporary fetches are initiated immediately after the address generations. Minimum time for any branch out of the instruction stack, therefore, is two cycles plus the access time.

The test for a conditional branch is normally made after the address generation. The two types of conditional branches are: those whose condition is set by the instruction processor, and those whose condition is set by the fixed-point or floating-point execution element. For the instructions 'branch on count' (BCT, BCTR), 'branch on index high,' and 'branch on index low or equal,' the condition is set by the instruction processor. For the 'branch on condition' (BC, BCR) instruction, the condition is set by the execution elements. (Masks of 0 and 15 are special cases and are detected during the decoding cycle.)

When the condition is set by the instruction processor, no further instructions are decoded until all tests have been completed. Instruction processor times (in cycles) for some of the more important branches are:

	Target in Stack		*Target Not in Stack*	
	Loop Mode	*Quick Mode*	*Loop Mode*	*Not Loop Mode*
BX, Branch	4	3	6 + access time	8 (or 2 + access time)*
BX, No Branch	6	5	5	6
BCT, Branch	4	3	5 + access time	7 (or 2 + access time)*
BCT, No Branch	5	4	4	5

* The actual time required is the longer of the two times listed.

When the condition is set by an execution element, the first three cycles of the branch are taken by the instruction processor, and the temporary fetches are made. The instruction processor then enters conditional mode until the condition code is determined.

In conditional mode, no additional instruction fetches are made. The instruction processor continues to decode instructions, generate addresses, and issue operations to the fixed-point operation stack and the floating-point operation stack; the operations are conditional and cannot be decoded or executed until an activate signal is sent by the instruction processor.

The instruction processor continues to decode instructions conditionally until any of the following conditions occurs:

1. The condition code is set.
2. No more instructions are available in the stack.
3. The fixed-point or floating-point operation stack is filled.
4. An instruction-processor or variable-field-length instruction is encountered (except for an unconditional branch or a no-operation instruction, which can be executed in conditional mode).

When the condition code is set, the instruction processor takes one cycle to make a decision. If the branch is not taken, an activate signal is sent to the fixed-point and floating-point operation stacks, and the instruction processor continues decoding instructions. If the branch is taken, a cancel signal is sent to the fixed-point and floating-point operation stacks and to the SAR's, and the instruction processor begins decoding instructions along the new path. When conditional mode is ended, instruction fetching resumes along the correct path.

When the machine is in loop mode, no temporary fetches are made for conditional branches.

An unconditional branch (BC 15 or BCR 15) takes either six cycles or two cycles plus the access time. A branch within the stack takes five cycles, and a branch closing a loop takes two cycles.

The 'branch and link' instructions (BAL, BALR) require four cycles plus the time required for access or plus the time required for the condition code to be determined, whichever is longer. The 'branch and link' instruction destroys loop mode.

A no operation (BC 0,X; BCR 0,R; BCR C,0) requires one cycle; a count without branching (BCTR R,0), three cycles; a link without branching (BALR R,0), five cycles or the time until the condition code is determined; and an 'execute,' five cycles plus the access time plus the target execution time.

Fixed-Point Execution

The following information is pertinent to fixed-point execution timing:

1. Decoding proceeds serially.
2. No conditional operation can be decoded until it has been activated or canceled.
3. Canceled operations are decoded in one cycle.
4. An active operation is not completely decoded until the cycle before its execution starts.
5. Execution can begin if the following conditions are met:
   a. The operation is decoded.
   b. The data is available.
   c. The execution circuitry is free.
6. As soon as decoding is completed for a one-cycle operation, the instruction processor is notified that the stack position is free. For operations of more than one cycle, the stack-position-free notification is delayed until the second or third cycle. Notification that the fixed-point buffers are released is given to the instruction processor during the first cycle for all instructions except 'convert to binary' and 'divide' (D), which do not release the buffers until during the last cycle.

Floating-Point Execution

In the following information, pertaining to floating-point execution timing, precision conflicts (differences in precision between overlapped floating-point operations using the same floating-point register) and RR instructions for which both registers are free may cause exceptions to items 1-6:

1. Decoding proceeds serially.
2. No conditional operation can be decoded until it has been activated or canceled.
3. Canceled operations are decoded in one cycle.
4. Operations that do not require an execution unit can be decoded in one cycle.
5. Operations that require an adder or a multiplier can be decoded in one cycle if a reservation station is available.
6. If a decode is waiting for a reservation station, it can be completed on the cycle before the result of that reservation station goes on the common data bus.
7. The test for 'load and test' (LTDR and LTER) is made during the common data bus cycle.
8. An operation in which the adder is used can begin if the following conditions are met:
   a. The operation is decoded.
   b. The data is available.
   c. Another add with higher priority is not beginning on the same cycle.
   d. The execution circuitry is free.
9. A multiply or divide can begin if the following conditions are met:
   a. The operation is decoded.
   b. The data is available.
   c. Another multiply or divide with higher priority is not beginning on the same cycle.
   d. The execution circuitry is free.
10. If more than one unit request the common data bus simultaneously, the following operations are given

priority in the order indicated: loads, short- and long-precision adds, short- and long-precision multiplies, extended-precision operations.

11. As soon as an operation has been decoded, the instruction processor is notified that the stack position is free.
12. If the operation has already been decoded, the buffer is set free as soon as the data enters it.
13. If the buffer is filled before the operation is decoded, the buffer is set free one cycle after the decoding.

Selected Execution Times

Because of the concurrency achieved in the Model 195, the effective time required by a given instruction is not directly related to the rate at which the instruction can be processed.

The following is a list, by category, of the number of cycles required by the appropriate execution element to process certain instructions. These times do not include any of the other processing times required for that instruction and do not reflect the effects of simultaneous operations or overlap. Instructions are listed by their mnemonics.

	No. of Machine Cycles
Fixed-Point Instructions	
A, AH, AL, ALR, AR, C, CH, CL, CLR, CR, IC L, LCR, LH, LNR, LPR, LR, LTR, N, NR, O, OR, S, SH, SL, SLR, SR, ST, STC, STH, X, XR	1
SLA, SLL, SRA, SRL	2
SLDA, SLDL, SRDA, SRDL	3-4
MH	7
M, MR	7-11
D, DR	36-37
CVB	17-18
CVD	17-32
Immediate Instructions	
Fetch only: CLI, TM	1
Store only: MVI	1
Fetch and Store: NI, OI, XI	2
Floating-Point Instructions	
LD, LDR, LE, LER, LTDR, LTER, STD, STE	0
AD, ADR, AE, AER, AU, AUR, AW, AWR, CD, CDR, CE, CER, HDR, HER, LCDR, LCER, LNDR, LNER, LPDR, LPER, SD, SDR, SE, SER, SU, SUR, SW, SWR	2
MD, MDR, ME, MER (normalized numbers)	3
MD, MDR, ME, MER (unnormalized numbers)	4
DE, DER	9
DD, DDR	11

The 0-cycle instructions do not require an execution unit. The 2-cycle instructions are executed in the adder. The 3-, 4-, 9-, and 12-cycle instructions are executed in the multiplier.

	No. of Machine Cycles
Extended-Precision Floating-Point Instructions	
LRDR, LRER	2
AXR, SXR	9
MXR (normalized/unnormalized)	25/29
MXDR, MXD (normalized numbers)	7/8
MXDR, MXD (unnormalized numbers)	7/8

READER'S COMMENT FORM

IBM System/360 Model 195
Functional Characteristics

A22-6943-0

- How did you use this publication?

  As a reference source ☐
  As a classroom text ☐
  As ..... ☐

- Based on your own experience, rate this publication...

	.....	.....	.....	.....	.....
As a reference source:	Very Good	Good	Fair	Poor	Very Poor
As a text:	Very Good	Good	Fair	Poor	Very Poor

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

  Reply requested ☐

  No reply required ☐

- Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . . . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 419
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . . . . .

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N.Y. 12602

ATTENTION: CUSTOMER MANUALS, DEPT. B98

fold fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM System/360 Model 195 Functional Characteristics Printed in U.S.A. A22-6943-0

A22-6943-0

IBM System/360 Model 195 Functional Characteristics Printed in U.S.A. A22-6943-0

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]