the address bus down to 24 bits. The operating frequencies of the 80386 and its various versions are 16, 20, 25 and 33 MHz.

The Intel 80486 was introduced in 1989. It has an 8 KB cache memory which is shared for data and instruction. It has an internal floating point unit (FPU), i.e., FPU was integrated on the same chip. Its different versions operate at clock frequencies from 25 to 100 MHz (Intel 80486 DX4).

Pentium was introduced in 1993. It retains the 32-bit address bus of the 80486 but doubles the data bus to 64 bits. It has two 8 KB cache memories one for instruction and one for data. Dual pipeline which is also known as *superscalar architecture* was adopted in this microprocessor. At present the Pentium architecture includes frequencies upto 1.75 GHz, 20-stage pipeline and 3-level cache memory.

19.1 8086 MICROPROCESSOR

The 8086 consists of two main sections, the bus interface unit (BIU) and the execution unit (EU). The EU contains the ALU, flags and general purpose registers. The EU carries out all the arithmetic and logical operations. All the registers in the 8086 are 16 bits wide, although some can be used as two 8-bit registers. The BIU controls the address, data and control buses. When the EU is decoding an instruction or executing instructions inside the microprocessor, the BIU pre-fetches instructions from memory and stores them in the instruction queue for faster processing. The block diagram of 8086 is shown in Figure 19.1.

19.1.1 Bus Interface Unit (BIU)

The bus interface unit (BIU) consists of the following:

Queue

Microprocessor 8086 consists of a first in first out (FIFO) registers set, arranged like a pipe and called *queue*. The BIU continuously fetch operations from the memory while the processor is executing the current instruction. The BIU unit stores the fetched bytes in the queue and the EU will read these bytes from the queue as and when it requires. The memory interface is usually much slower than the processor execution time, so this decouples the memory cycle time from the execution time.

Segment registers

8086 consists of four 16-bit segment registers the code segment (CS), data segment (DS), stack segment (SS) and extra segment (ES). These registers are used with the 16-bit base registers to generate the 20-bit physical address required to allow the 8086/8088 to address 1 MB of memory. They are changed under program control to point to different segments, as a program executes. The segmented architecture was used in the 8086 to keep compatibility with earlier processors such as the 8085. It is one of the most significant elements of the intel architecture.

Code segment (CS): Code segment (CS) is a 16-bit register containing address of 64 KB segment which stores microprocessor instructions. The microprocessor uses CS segment for all

Scanned by CamScanner

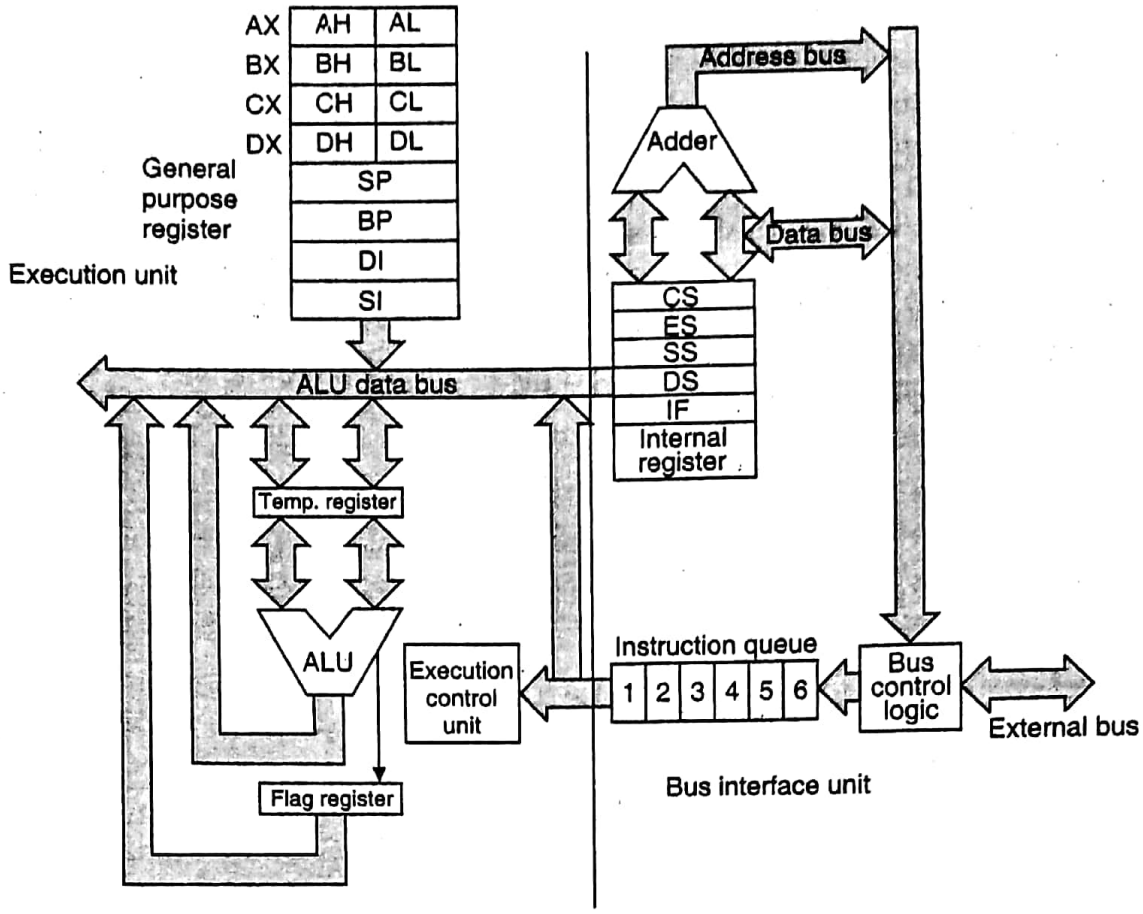


Figure 19.1 Block diagram of 8086.

accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

Stack segment (SS): Stack segment (SS) is a 16-bit register containing address of 64 KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

Data segment (DS): Data segment (DS) is a 16-bit register containing address of 64 KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX and DX) and index register (SI, DI) are located in the data segment. DS register can be changed directly using POP and LDS instructions.

Extra segment (ES): Extra segment (ES) is a 16-bit register containing address of 64 KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It is possible to change the default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.

Scanned by CamScanner

Instruction pointer (IP) and address summation

The IP contains the offset or logical address of the next byte to be read from the code segment. In fact it shows the distance of the current location, in bytes, from the base address given by the current code segment (CS) register. Figure 19.2 shows how this is done. The contents of the CS are shifted left by four bits. Bit 15 moves to bit 19 position. The lowest four bits are filled with zeroes or the CS register value is multiplied by decimal 16 or hexadecimal 10H. The resulting value is added to the instruction pointer contents to make up a 20 bit physical address. The CS makes up a segment base address and the IP is looked as an offset into this segment. This segmented model also applies to all the other general registers and segment registers in the 8086 device. For example, the SS and SP are combined in the same way to address the stack area in the physical memory.

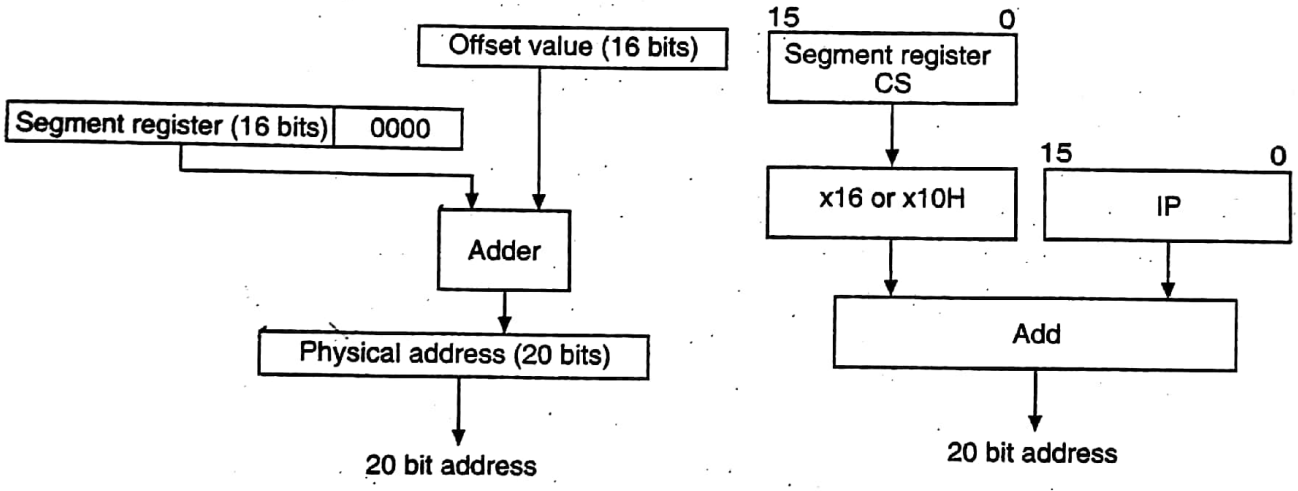


Figure 19.2 Generation of 20 bit physical address.

19.1.2 Execution Unit

The execution unit consists of four 16-bit general purpose data registers which can be used as eight 8-bit data registers, four 16-bit pointer and base registers and one 16-bit flag register.

General purpose data registers

Microprocessor 8086 consists of four 16-bit data registers AX, BX, CX and DX. Each of these registers can be divided into two parts as higher and lower part to store 8-bit data. These are shown in Table 19.1.

Table 19.1 Data Register of 8086

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Scanned by CamScanner

Apart from being general purpose registers, these registers also perform some specific functions. These functions are as follows:

Accumulator register (AX): It consists of two 8 bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

Base register (BX): It consists of two 8 bit registers BL and BH which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

Count register (CX): It consists of two 8 bit registers CL and CH which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word and CH contains the high-order byte. Count register can be used as a counter in string manipulation and shift/rotate instructions.

Data register (DX): It consists of two 8 bit registers DL and DH which can be combined together and used as a 16 bit register DX. When combined, DL register contains the low-order byte of the word and DH contains the high-order byte. It is used in DIV and MUL instructions and also in indirect IO addressing. In integer 32 bit multiply and divide instructions, the DX register contains high-order word of the initial or resulting number.

Pointers and base registers

Microprocessor 8086 consists of four 16 bit pointer and base registers. These are SI, DI, SP and BP. These registers are used to hold offset or the logical addresses within a segment.

Stack pointer (SP): It is a 16-bit register pointing to program stack.

Base pointer (BP): It is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source index (SI): It is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

Destination index (DI): It is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Flag register

Microprocessor 8086 consists of one 16-bit flag register. The flag register is a set of 16 independent flip-flops. Out of these 16 flip-flops, 6 flip-flops are used to indicate some data conditions and 3 flip-flops are used to controlled some machine control operations and the remaining 6 flip-flops are reserved for upcoming microprocessors. The format of this flag register is shown in Figure 19.3.

Scanned by CamScanner

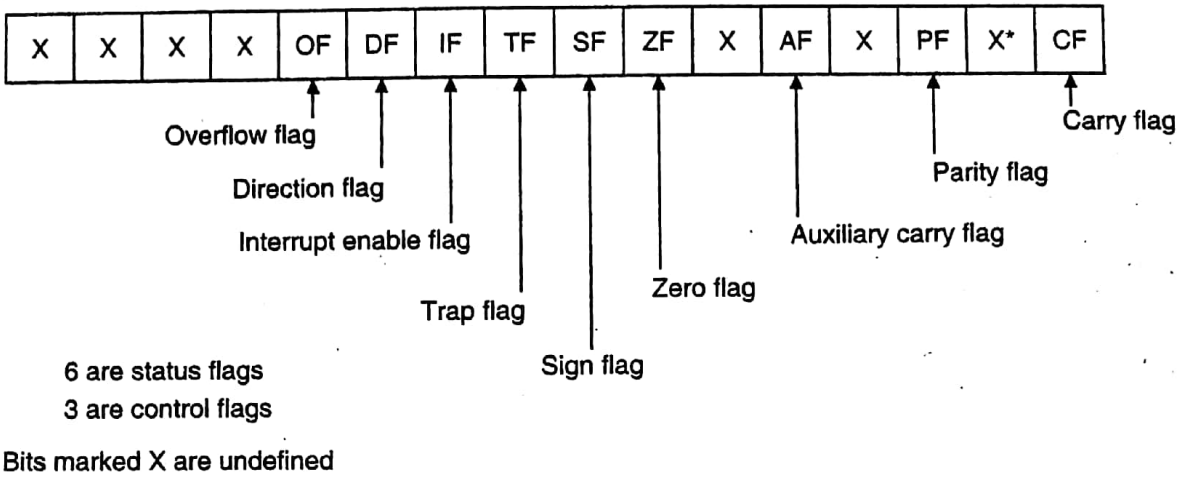


Figure 19.3 Flag register format.

Status flags: The various status flags are as follows:

Zero flag (Z) The zero flag indicates whether the result of a mathematical or logical operation is zero or non-zero. For a zero result this bit will be set otherwise this bit will be clear.

Carry flag (CY) This flag is set when an arithmetic carry or borrow has been generated out of the most significant ALU bit position during an addition or subtraction operation. When used after an arithmetic operation it could be considered to be the unsigned equivalent of the overflow flag.

Sign flag (S) This flag indicates whether the result of a mathematical operation is negative or positive. If the result is positive, then this bit will be clear. Actually this bit represents the status of the D_7 bit of the last result.

Parity flag (P) Parity flag indicates whether the number of set bits is odd or even in the binary representation of the result of the last operation. If the current result is of even parity, this bit will be set. If the result is of odd parity, this bit will be zero. This flag indicates whether the current result is of even parity (1) or of odd parity (0).

Auxiliary carry flag (AC) It shows carry propagation from D_3 position to D_4 position. This flag is used to convert the binary result into a BCD result. To understand it better, consider the example:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1
1	0	1	1	0	1	1	1

Overflow flag This flag is used in signed arithmetic operation. If the signed result is of more bits, then the destination operand will be set; otherwise it will be cleared. To understand better, consider the example:

Scanned by CamScanner

Overflow ←

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	0	1	1	0	0
1	1	1	0	1	0	1	1
1	0	1	1	0	1	1	1

Control flags: The various control flags are as follows:

Trap flag (TF) If this flag is set, a single step interrupt occurs after the execution of the next instruction. This flag is used for single step debugging. This flag is set or cleared by software means.

Interrupt enable flag (IF) This flag is used to mask or unmask the maskable interrupt. When this flag is set maskable interrupts will cause the microprocessor to transfer its control to an interrupt vector location specified location. This flag is set or cleared by software means.

Direction flag (DF) This flag is used in string related operations. It causes string instructions to auto-decrement the appropriate index register (SI or DI) when set. If it is cleared, the index registers will be in auto-increment mode.

19.2 PIN CONFIGURATION OF 8086

The 8086 microprocessor is a 40-pin IC which operates on +5 V power supply and three clock rates: 5, 8 and 10 MHz. The 8086 operates in both single processor and multiple processor configurations to achieve high performance levels. During single processor mode, which is known as *minimum mode*, eight of its pins, from pin number 24 to 32 are having different definition as that of multiple processor mode, known as *maximum mode*. The pin diagram of 8086 is shown in Figure 19.4.

19.2.1 Pin Details of 8086 (Common to Both Minimum and Maximum Mode)

Address data bus (AD_0 – AD_{15})

These lines are multiplexed address/data lines. During the T_1 state of every machine cycle these lines carry the address and for the rest of the T-states (T_2, T_3, T_W, T_4) these lines carry the data. The A_0 line along with $\overline{BHE}$ defines whether the microprocessor will access the lower byte or the higher byte or the word.

Address and status lines (A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3)

During T_1-state these lines behave as the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T_2, T_3, T_W and T_4. The statuses of the interrupt enable flag bit (S_5) is updated at the beginning of each CLK cycle. S_4 and S_3 are encoded as shown in Table 19.2.

Scanned by CamScanner

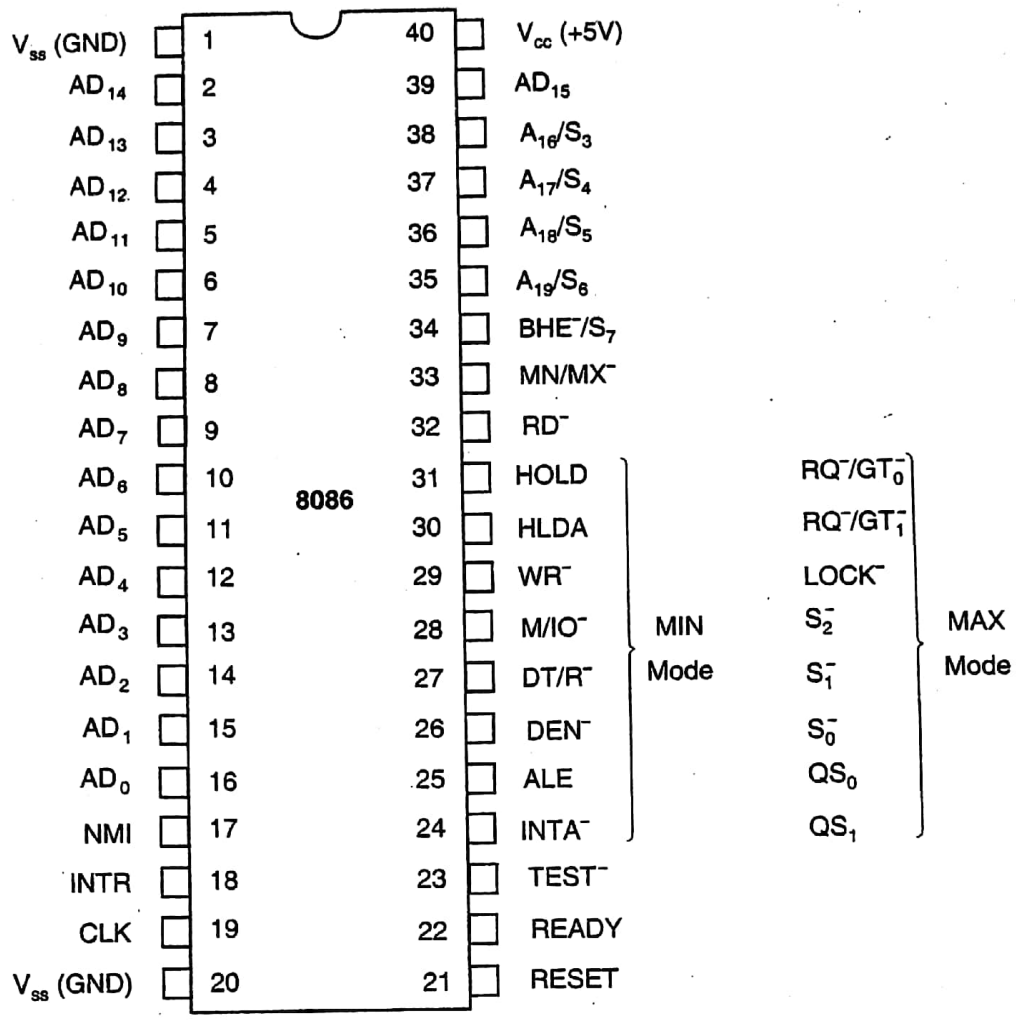


Figure 19.4 Pin configuration of 8086.

Table 19.2 Encoding of S_4 and S_3

S_4	S_3	Characteristics
0	0	Alternate data
0	1	Stack
1	0	Code or none
1	1	Data

This information indicates which relocation register is presently being used for data accessing.

Bus high enable/Status (BHE^-/S_7)

During T_1 state of every machine cycle the bus high enable signal (BHE^-) should be used to enable data into the most significant half of the data bus, pins $D_{15} - D_8$. Eight bit oriented devices tied to the upper half of the bus would normally use BHE^- to enable chip select functions. BHE^- is LOW during T_1 for read, write and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available

Scanned by CamScanner

during T_2, T_3, and T_4. The signal is active LOW and floats to 3-state OFF in ''hold''. It is LOW during T_1 for the first interrupt acknowledge cycle. Encoding of $\overline{BHE}$ and A_0 is shown in Table 19.3.

Table 19.3 Encoding of $\overline{BHE}$ and A_0

$\overline{BHE}$	A_0	Operation
0	0	Word will be accessed
0	1	Upper byte or odd byte will be accessed
1	0	Lower byte or even byte will be accessed
1	1	None

Read ($\overline{RD}$)

It is an active low output signal. It indicates that the microprocessor is performing a memory or I/O read operation. This signal is used to read devices which reside on the 8086 local bus. $\overline{RD}$ is active LOW during T_2, T_3 and T_W of any read machine cycle and is guaranteed to remain HIGH in other T states.

Ready

This signal is used to synchronize the slower peripherals. This signal is active HIGH input signal. When a peripheral device is ready to receive the data it will send the READY signal to the microprocessor. On receiving this signal, microprocessor releases the data and enters into the ''wait'' state till the next READY signal. The READY signal from memory/IO is synchronized by the 8284A clock generator to form READY.

Interrupt request (INTR)

This signal is active HIGH. INTR is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. It can be internally masked by software, resetting the interrupt enable bit.

$\overline{Test}$

TEST input signal is examined by the ''wait'' instruction. If the TEST input is LOW execution continues, otherwise the processor waits in an ''idle'' state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

Non-maskable interrupt (NMI)

It is the non-maskable interrupt. This interrupt cannot be masked. It is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector look-up table located in the system memory. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.

Reset

RESET causes the processor to immediately terminate its current operation. The signal must be active HIGH for at least four clock cycles.

Scanned by CamScanner

Clock (CLK)

It provides the basic timing for the microprocessor and the bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.

Minimum/Maximum (MN/$\overline{MX}$)

This signal indicates that in which mode the processor is to operate. If this pin is high it means that the microprocessor will be in single processor mode and if it is low then the microprocessor will be in multiprocessor operation mode.

V_{CC}

This pin is connected to the +5 V power supply.

GND

It is the GROUND pin.

19.2.2 Pin Details of 8086 (Minimum Mode)

The following pin function descriptions are for the 8086 in minimum mode, (i.e., MN/MX = V_{CC}).

Status line (M/$\overline{IO}$)

It is an output signal. It is used to distinguish whether microprocessor is going to access a memory or an IO. If M/IO^- becomes zero, it means it will be an IO operation otherwise a memory operation. M/IO^- floats to 3-state OFF in local bus "hold acknowledge".

Write ($\overline{WR}$)

It is an active low output signal. It indicates that the microprocessor is performing a write operation either from memory or I/O, depending on the status of the M/IO^- signal. WR^- is active for T_2, T_3 and T_W of any write machine cycle.

Interrupt acknowledge ($\overline{INTA}$)

It is an active low output signal. Microprocessor sends this signal in response to an interrupt request signal. It is active LOW during T_2, T_3 and T_W of each interrupt acknowledge cycle.

Address latch enable (ALE)

It is an active high output signal. It is provided by the processor to latch the address into the 8282/8283 address latch. It is HIGH pulse active during T_1 of any machine cycle. The T_1 state is sufficient enough to latch the address on the multiplexed $AD_0 - AD_{15}$ lines. Note that ALE is never floated.

Data transmit/Receive (DT/$\overline{R}$)

It is used to control the direction of data flow through the transceiver. If it is high, then data will be transmitted otherwise data will come to the microprocessor.

Scanned by CamScanner

Data enable (DEN^-)

It is an active low output signal. It provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. DEN^- is active during each memory and I/O access and for $INTA^-$ cycles. For a read or $INTA^-$ cycle it is active from the middle of T_2 until the middle of T_4, while for a write cycle it is active from the beginning of T_2 until the middle of T_4.

Hold request (HOLD)

It is an active high input signal. This pin is used by external devices to gain control of the buses. When the HOLD signal is activated by an external device, the microprocessor suspends current execution and stops using the buses. This would allow external devices to control the buses. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot, otherwise, guarantee the setup time.

Hold acknowledge (HLDA)

It indicates that the microprocessor has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The microprocessor takes the buses one half clock cycle after HLDA goes low. With the issuance of HLDA the processor will float the local bus and control lines.

19.2.3 Pin Details of 8086S (Maximum Mode)

The following pin function descriptions are for the 8086/8288 system in the maximum mode:

Status (S_2^-, S_1^-, S_0^-)

These are three status output signals. These are active during T_4, T_1 and T_2 states and is returned to the passive state (1, 1, 1) during T_3 or during T_W when READY is high. This status is used by the 8288 bus controller to generate all memory and I/O access control signals in maximum mode configuration. Any change by S_2, S_1, or S_0 during T_4 is used to indicate the beginning of a bus cycle and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These status lines are encoded as shown in Table 19.4.

Table 19.4 Encoding of S_2, S_1 and S_0

S_2^-	S_1^-	S_0^-	Operation
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

I/O request/Grant (RQ/GT_0, RQ/GT_1)

These signals are same as that of HOLD and HLDA in minimum configuration. These pins are used by other local bus masters to force the processor to release the local bus at the end of the

Scanned by CamScanner

processor's current machine cycle. Each pin is bidirectional with RQ/GT_0 having higher priority than RQ/GT_1.

The request/grant sequence is as follows:

1. A pulse of 1 CLK wide from another local bus master indicates a local bus request (hold) to the 8086 (pulse 1).
2. During a T_4 or T_1 clock cycle, a pulse 1 CLK wide from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus to float and it will enter the "hold acknowledge" state at the next CLK. The microprocessor's bus interface unit is disconnected logically from the local bus during "hold acknowledge".
3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK.

Each master–master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active low.

If the request is made while the microprocessor is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met:

Request occurs on or before T_2.

1. Current cycle is not the low byte of a word (on an odd address).
2. Current cycle is not the first acknowledge of an interrupt acknowledge sequence.
3. A locked instruction is not currently executing.

If the local bus is idle when the request is made, the two possible events will follow:

1. Local bus will be released during the next clock.
2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.

$\overline{LOCK}$

It is an active low output signal. If this signal is active, the other bus masters will not be allowed to take control over the system buses. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction.

Queue status (QS_1, QS_0)

The queue status is valid during the CLK cycle after which the queue operation is performed. QS_1 and QS_0 provide status to allow external tracking of the internal 8086 instruction queue. Table 19.5 shows the encoding of the queue status signals.

Table 19.5 Encoding of QS_1 and QS_0

QS_1	QS_0	Characteristics
0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

Scanned by CamScanner

19.3 MEMORY ORGANIZATION

The 8086 microprocessor provides a 20-bit address to memory. The memory is organized as a linear array of upto 1 MB, addressed from 00000H to FFFFFH. Though microprocessor 8086 is a 16-bit microprocessor still its memory is of 8-bit wide. This is because of two reasons:

1. Most of our IO devices are of 8-bit wide.
2. There are many instructions which are of 1 byte. The other instructions are from 2 to 7 bytes. So, by being able to access individual bytes, these odd lengthed instructions can be handled.

The memory is logically divided into code, data, extra and stack segments each of 64 KB. These four segments can partially or fully overlap with each other. These four segments are shown in Figure 19.5.

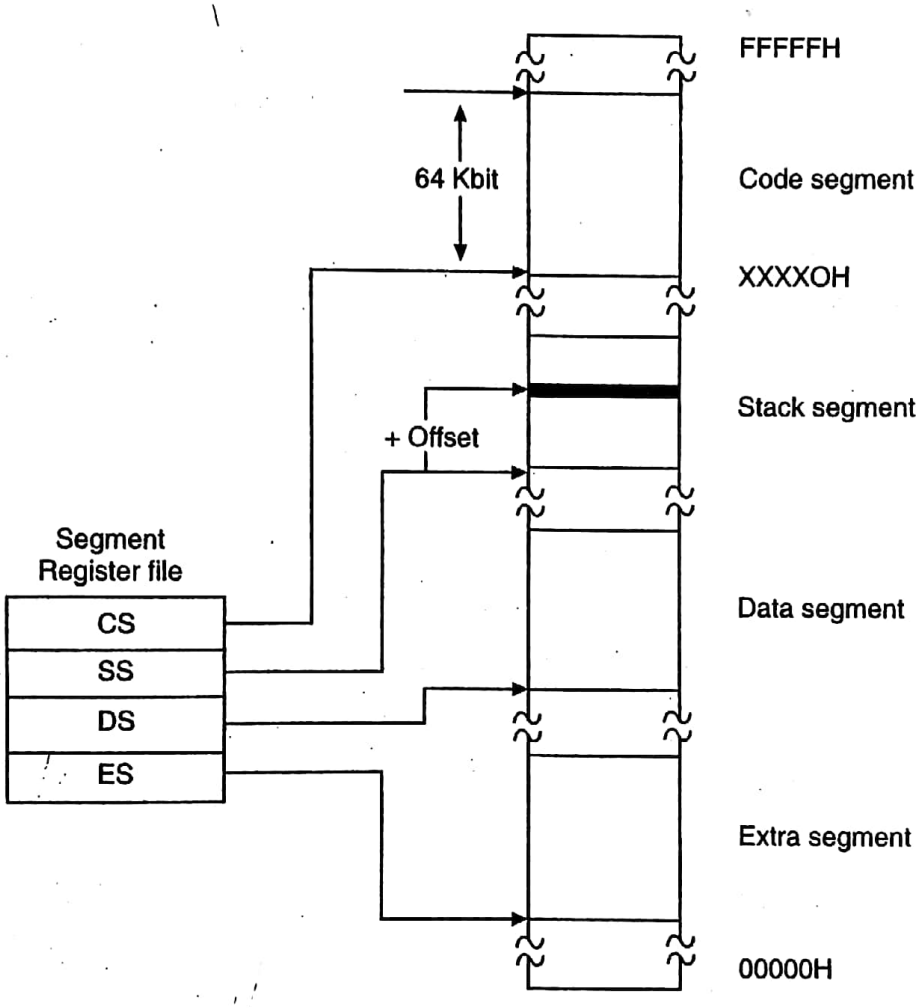


Figure 19.5 The segmented memory.

All memory references are made relative to base addresses contained in the segment registers. The segment types were chosen, based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the specific rules of Table 19.1. All information in one segment type shares the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster and more structured.

Scanned by CamScanner

Flynn's Taxonomy

"66" Michael Flynn's classification of comp. archi based on concurrent instruction & data stream.

↳ defined in 1966 by Flynn,

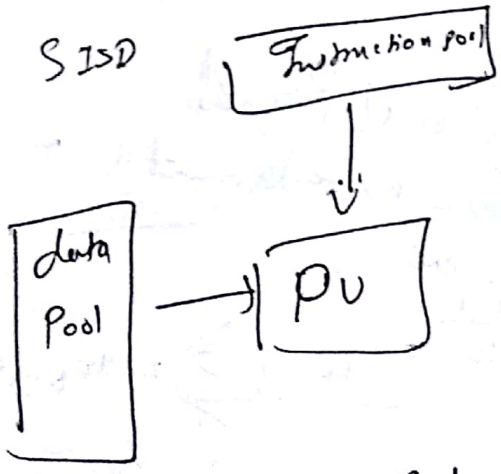
↳ classification is based on the notion of Stream of information: (instr & data)

↳ Does not clearly classify all models in use today

↳ There are Combinations also

→ SISD ⇒ Seq. Comp., no ||ism.

e.g. traditional single processor machines like PC (old one's)



Classical Von Neumann archi

Scanned by CamScanner

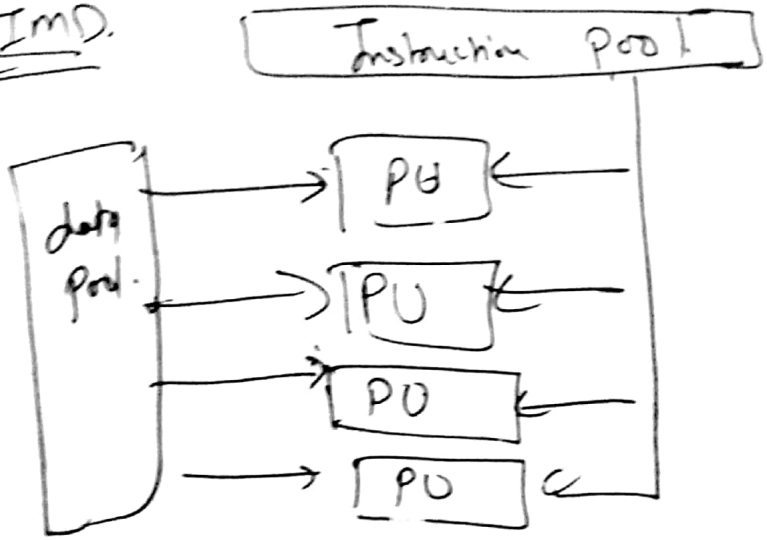
SIMD ⇒ Exploits multiple data streams against a single instruction stream.

"Naturally Parallelized".

e.g. array of processors or G.P.U

Typical for splitting large data sets.

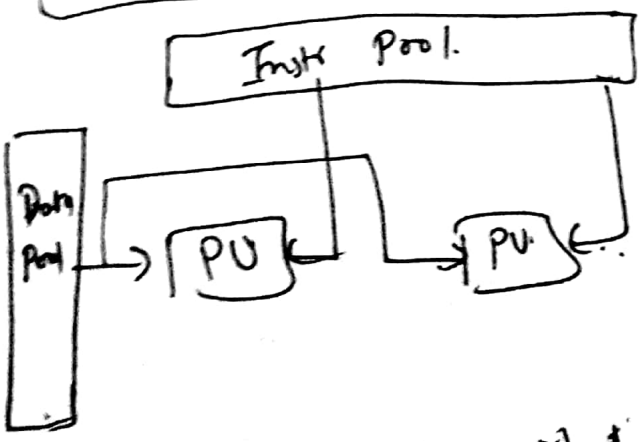
SIMD.



MISD ⇒ uncommon archi generally used for fault tolerance.

Mech ~~⇒ Seq~~ ⇒ Process data & pass onto next PU.

e.g. ~~Space Shuttle, Flight Control Comp.~~



MISD

~~Exam~~ Non existent.

Scanned by CamScanner

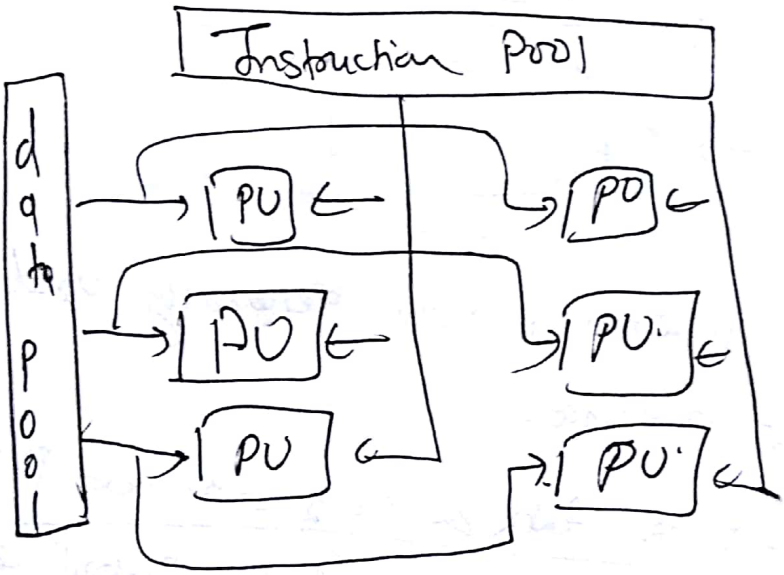
MIMD

Multiple Instruction, Multiple data Streams.

↳ multiple processors Simultaneously executing different instructions on different data.

↳ May send results to Control location.

↳ Shared memory (UMA) or (NUMA) memory space.
↳ Distributed memory system's.



most Common and General II/O machine

Shared memory Systems

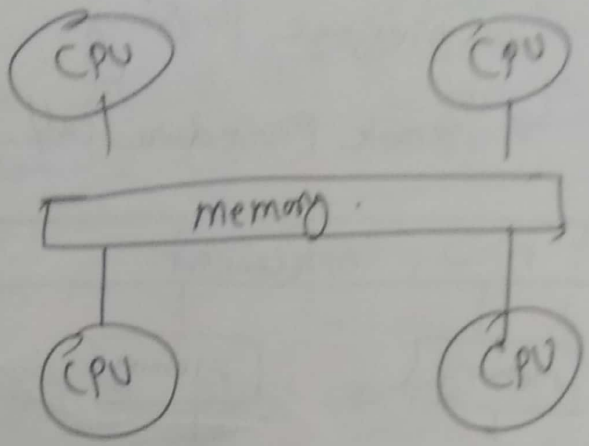
↳ All cpu share access to same address space. e.g pc with more than 1 processors
↳ multiple Shared variable for data exchange.

Scanned by CamScanner

Shared memory

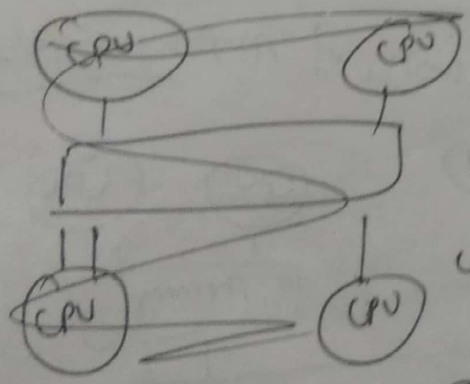
- UMA
- NUMA.

Symmetric multi processors (SMPs)

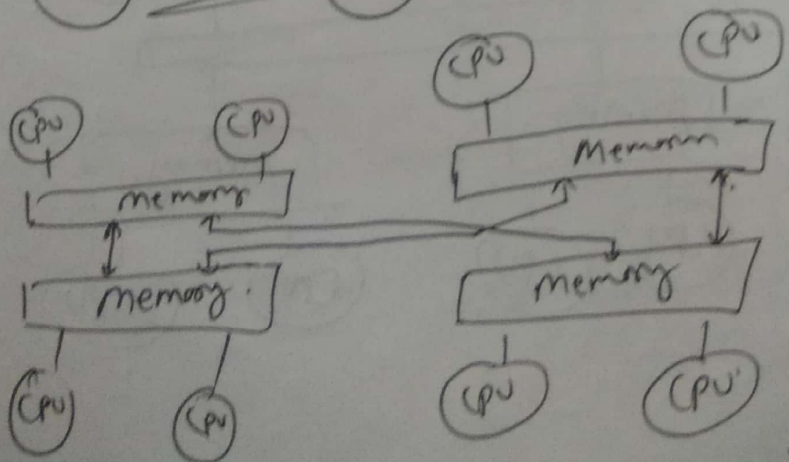


Same physical addr to all processors

NUMA



⇒ Some memory is closer to a certain processor than other memory.
→ Still addressable to all processors
↳ The access time might vary strongly.



e.g. SGI origin with 32 processors

Scanned by CamScanner

Distributed memory Making

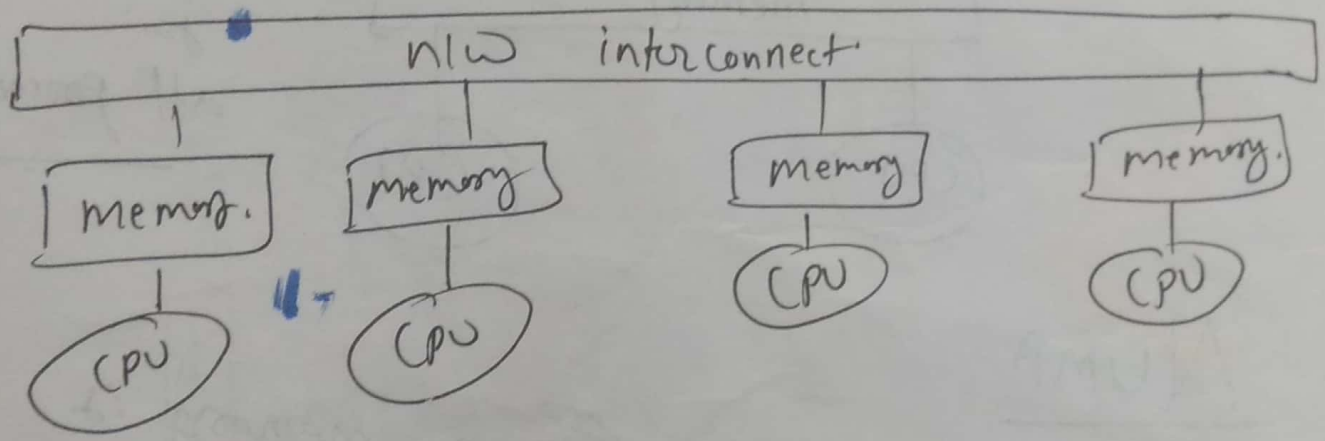
→ Each processor has its own address space.

→ Comm^n b/w processes by explicit data exchange.

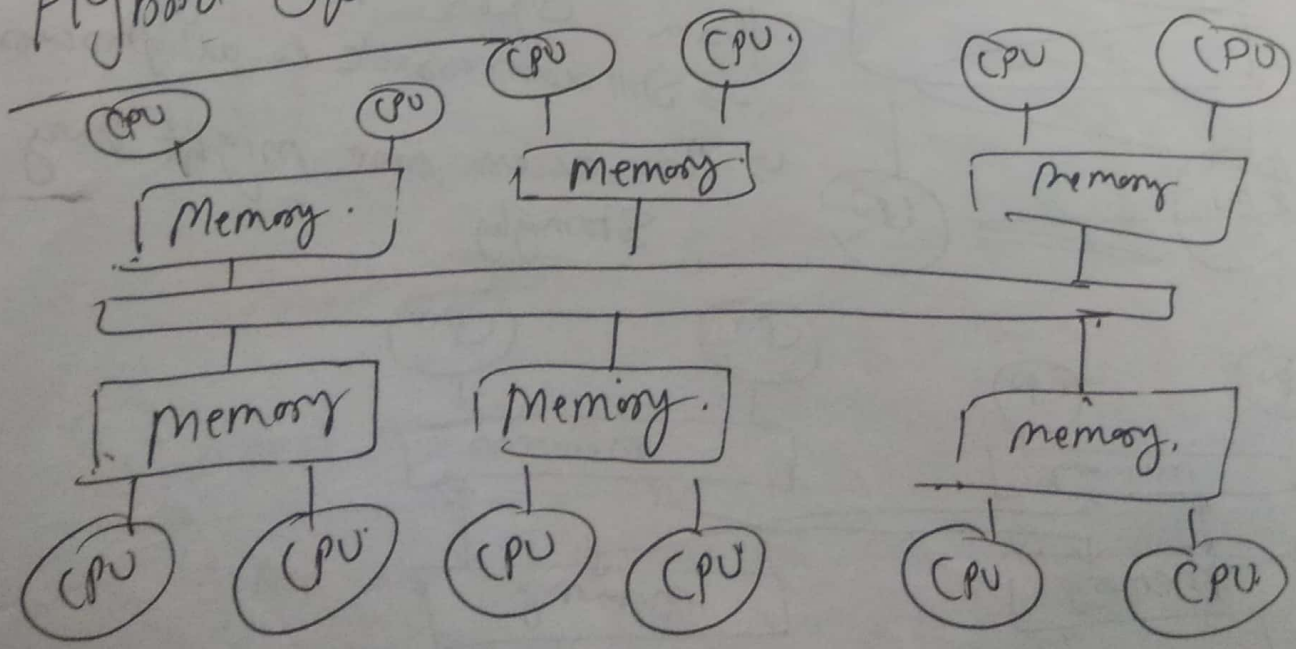
↳ Sockets

↳ Message Passing.

↳ Remote Procedure call



Hybrid Systems :- Cluster of m-processor nodes

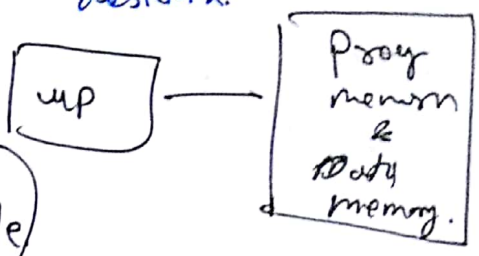


Scanned by CamScanner

Computer Architecture

Von Neumann Archi :-

47. desktop 2.

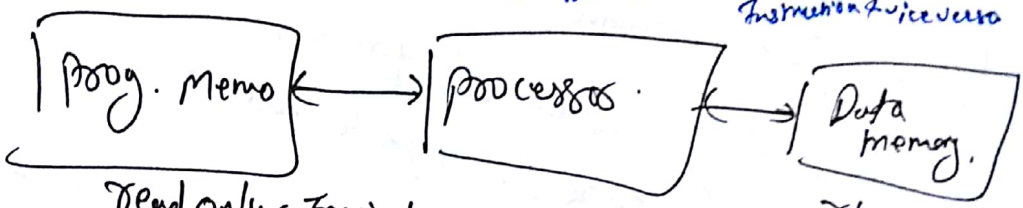


↳ Pipelining is not possible (the instruction)

Harvard Archi →

PDA mobile phones etc.

1. allowed 1st access to both.
2. DGI can be accessed the same way.
3. expensive
4. free data memory can be use for Instruction & vice versa



Read only (Typical assumption)

R/W (CT. Assumption)

⇒ 1st access to data & Instr.

RISC & CISC

Comp. arthi ⇒ Set of rules & methods that describe

↳ functionality.
↳ organization &
↳ implementation of

Computer System

Table 4.4 Bit Combinations of Registers and Register Pair

Register	Code
B	000
C	001
D	010
E	011
H	100
L	101
M	110
A	111

Register pair	Code
BC	00
DE	01
HL	10
AF or SP	11

4.2 ADDRESSING MODES OF MP 8085

Every instruction consists of two parts, i.e., what operation is to be performed (called *operation code* or *opcode*), and on what this operation is to be performed (called *operands*). For example, in case of A + B, the + sign is the opcode, and A and B are the operands.

Each instruction performs an operation on the specified data called *operand*. An operand must be specified for an instruction to be executed. The operand may be in the general purpose register, accumulator or in a memory location. The way in which the operand is specified in an instruction is called *addressing mode*. Various addressing modes used in microprocessor 8085 are:

1. Immediate addressing
2. Register addressing
3. Direct addressing
4. Register indirect addressing
5. Implicit addressing

4.2.1 Immediate Addressing

In immediate addressing mode, the operands are specified within the instruction itself. The instruction format of instructions with immediate addressing mode is shown in Figure 4.1.

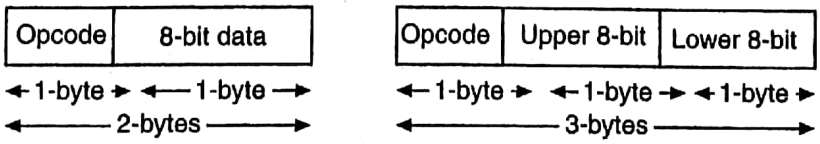


Figure 4.1 Instruction format of the instructions with immediate addressing mode.

Examples of immediate addressing mode

MVI A, 05H : Move 05H in the accumulator.
ADI 06H : Add 06H to the contents of the accumulator.
LXI H, 2500H : Load HL pair with 2500H.

Scanned by CamScanner

When microprocessor executes the MVI A,05H instruction, the processor fetches the first instruction byte and determines that it must fetch one more byte from the next memory location. Similarly, when microprocessor executes the LXI H, 2500H instruction, the processor fetches the first instruction byte and determines that it must fetch two more bytes from the next two memory locations.

4.2.2 Register Addressing

In register addressing mode, the operands are in the general purpose registers. We specify the register in our instruction. The instruction format of instructions with immediate addressing mode is shown in Figure 4.2.

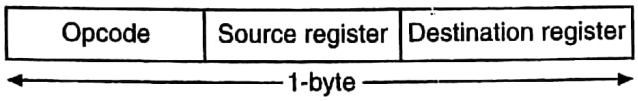


Figure 4.2 Instruction format of the instructions with register addressing mode.

Most of the instructions that use register addressing deal with 8-bit values. However, a few of these instructions deal with 16-bit register pairs. For example, the SPHL instruction moves the contents of the H and L registers to SP.

Examples of register addressing mode are as follows:

MOV A, B: This instruction moves the contents of register B to register A or accumulator.

ADD B : Add the contents of register B to accumulator.

4.2.3 Direct Addressing

The direct addressing mode is used when the operand is available/required at some memory or an IO location. In this addressing mode, the address of the operand (data) is given in the instruction itself. The instruction format of direct addressing mode is shown in Figure 4.3.

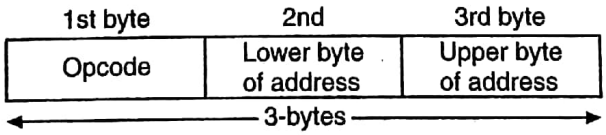


Figure 4.3 Instruction format of the instructions with direct addressing mode.

Instructions that include a direct address require three-bytes of storage, one for the instruction code, and two for the 16-bit address in case of memory related operations and require two-bytes, for the instruction code, and other for 8-bit IO address.

Examples of direct addressing mode are as follows:

STA 2400H : Store the content of the accumulator to memory location 2400H.

IN 02H : Input the data from input port 02H in accumulator.

Scanned by CamScanner

4.2.4 Register Indirect Addressing

This addressing mode is used only in concern with memory. In this mode of addressing, the address of the operand (data) is specified by a register pair. The instruction format of direct addressing mode is shown in Figure 4.4:

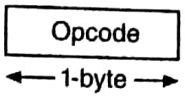


Figure 4.4 Instruction format of the instructions with register indirect addressing mode.

Example of indirect addressing mode are as follows:

```
LXI H, 2500H : Load HL pair with 2500H.
MOV A, M     : Move the contents of memory to the accumulator.
HLT          : Stop the program.
```

In this program, MOV A, M is the example of register indirect addressing because the address of the memory location M is specified by HL pair.

STAX B

This is another example of register indirect addressing.

Here, the instruction MOV A, M moves the contents of the memory address stored in the H and L register pair to accumulator. The instruction STAX B stores the accumulator in the memory location specified by the address in the B and C register pair.

4.2.5 Implicit Addressing

There are certain instructions which operate on the content or the accumulator directly; these instructions do not require specifying the operands. The instruction format of implied addressing mode is shown in Figure 4.5.

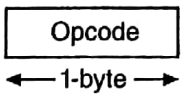


Figure 4.5 Instruction format of the instructions with implicit addressing mode.

Examples of implicit addressing mode are as follows:

```
CMA : Compliment the contents of the accumulator.
RAR : Rotate the contents of the accumulator right by one bit.
RAL : Rotate the contents of the accumulator left by one bit.
```

4.3 TIMING EFFECTS OF ADDRESSING MODES

The execution time of both an instruction and the memory space required depend on the addressing modes used. As an example, the instructions having register or implicit addressing will be executed much faster as compared to the instructions employing other addressing

Scanned by CamScanner

modes. This is because both these addressing modes deal directly with the processor's hardware or with data already present in the hardware registers. Also in these addressing modes, the instruction is always of one-byte and is fetched in a single machine cycle. Similarly, register indirect addressing mode is faster than direct addressing mode.

As most of the time of the execution of an instruction is wasted in getting a data from the memory or on how many machine cycles are required to execute that instruction. Also the number of machine cycles depends on the number of bytes required to represent that instruction. Hence, as the number of bytes required representing an instruction increases, the execution time will also increase.

For example, a three-byte instruction requires more time for execution as compared to a two-byte or a one-byte instruction. So, we can say that the number of memory accesses required is the most important factor in determining execution timing. For example, the CALL instruction, which is a three-byte instruction requires five machine cycles, three to access the complete instruction and two more to push the contents of the program counter onto the stack. Now the same CALL operation can be performed by one-byte RSTn instruction. This instruction will be executed in just three memory cycles.

4.4 INSTRUCTION SET CLASSIFICATION

An instruction is a binary bit pattern which performs a specific function in a system. The entire group of instructions of a system is called the instruction set. Instruction set determines what functions the microprocessor can perform with a single instruction. The instructions in microprocessor 8085 can be classified into five functional categories:

1. data transfer (copy) operations
2. arithmetic operations
3. logical operations
4. branching operations and
5. machine-control operations.

4.4.1 Data Transfer (Copy) Operations

This group of instructions copy data from a location called a *source* to another location called a *destination*, without modifying the contents of the source. In true sense, these instructions are not the data transfer instructions but data copy instruction because the source is not modified.

A data can be available at:

1. Immediately available (Im)
2. available at some memory (M)
3. available in some register (R)
4. available at some IO (IO).

And a data can be required at:

1. some memory
2. in some register
3. at some IO.

Scanned by CamScanner

Table 2.2 *Addressing Modes and the Corresponding MOD, REG and R/M Fields*

Operands	*Memory Operands*			*Register Operands*	
	No Displacement	*Displacement 8-bit*	*Displacement 16-bit*		
MOD	*00*	*01*	*10*	*11*	
R/M				*W = 0*	*W = 1*
000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16	AL	AX
001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16	CL	CX
010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16	DL	DX
011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16	BL	BX
100	(SI)	(SI) + D8	(SI) + D16	AH	SP
101	(DI)	(DI) + D8	(DI) + D16	CH	BP
110	D16	(BP) + D8	(BP) + D16	DH	SI
111	(BX)	(BX) + D8	(BX) + D16	BH	DI

Note: 1. D8 and D16 represent 8 and 16 bit displacements respectively.
2. The default segment for the addressing modes using BP and SP is SS. For all other addressing modes the default segments are DS or ES.

DS is the default data segment register when a data is to be referred as an operand. CS is the default code segment register for storing program codes (executable codes). SS is the default segment register for the stack data accesses and operations. ES is the default segment register for the destination data storage. All the segments available (defined in a particular program) can be read or written as data segments by newly defining the data segment as required. There is no physical difference in the memory structure or no physical separation between the segment areas. They may or may not overlap with each other. Chapter 3 on 'Assembly Language Programming' explains the coding procedure of the instructions with suitable examples.

2.2 ADDRESSING MODES OF 8086

Addressing mode indicates a way of locating data or operands. Depending upon the data types used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes, or some instruction may not belong to any of the addressing modes. Thus the addressing modes describe the types of operands and the way they are accessed for executing an instruction. Here, we will present the addressing modes of the instructions depending upon their types. According to the flow of instruction execution, the instructions may be categorised as (i) Sequential control flow instructions and (ii) Control transfer instructions.

Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example, the arithmetic, logical, data transfer and processor control instructions are sequential control flow instructions. *The control transfer instructions, on the other hand, transfer control to some predefined address or the address somehow specified in the instruction, after their execution.* For example, INT, CALL, RET and JUMP instructions fall under this category.

The addressing modes for sequential and control transfer instructions are explained as follows:

1. Immediate In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

Scanned by CamScanner

Example 2.1

```
MOV AX, 0005H
MOV BL, 06H
```

In the above examples 0005H and 06H are the immediate data. The immediate data may be 8-bit or 16-bit in size.

2. Direct In the direct addressing mode, a 16-bit memory address (offset) or an IO address is directly specified in the instruction as a part of it.

Example 2.2

```
MOV AX, [5000H]
IN 80H
```

Here, data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the offset address and content of DS as segment address. The effective address, here, is 10H*DS+5000H. In the second instruction 80H is IO address.

3. Register In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

Example 2.3

```
MOV BX, AX.
ADC AL, BL
```

The operands in these instructions are provided in registers BX, AX and AL, BL respectively.

4. Register Indirect Sometimes, the address of the memory location which contains data or operand is determined in an indirect way, using the offset registers. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI register. The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

Example 2.4

```
MOV AX, [BX]
```

Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H*DS+[BX].

5. Indexed In this addressing mode, offset of the operand is stored in one of the index registers. DS is the default segment for index registers SI and DI. In case of string instructions DS and ES are default segments for SI and DI respectively. This mode is a special case of the above discussed register indirect addressing mode.

Example 2.5

```
MOV AX, [SI]
MOV CX, [DI]
```

Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10H*DS+[SI]. The content of address 10H*DS+[SI] will be transferred into register CX.

6. Register Relative In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment. The example given below explains this mode.

Scanned by CamScanner

Example 2.6

```
MOV AX, 50H[BX]
MOV 10H[SI], DX
```

Here, the effective address is given as 10H*DS+50H+[BX] and 10H*DS+10H+[SI] respectively.

7. Based Indexed The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Example 2.7

```
MOV AX, [BX] [SI]
MOV [BX] [DI], AX
```

Here, BX is the base register and SI is the index register. The effective address is computed as 10H*DS+[BX]+[SI].

8. Relative Based Indexed The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any one of the base registers (BX or BP) and any one of the index registers, in a default segment.

Example 2.8

```
MOV AX,50H [BX][SI]
ADD 50H [BX] [SI], BP
```

Here, 50H is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as 10H*DS+[BX]+[SI]+50H. The second instruction adds content of B with memory location of which offset is given by adding 50H of content of BX and SI. The result is stered in the memory location.

For the control transfer instructions, the addressing modes depend upon whether the destination location is within the same segment or in a different one. It also depends upon the method of passing the destination address to the processor. Basically, there are two addressing modes for the control transfer instructions, viz. intersegment and intrasegment addressing modes.

If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called intersegment mode. If the destination location lies in the same segment, the mode is called intrasegment mode.

Figure 2.1 shows the modes for control transfer instructions.

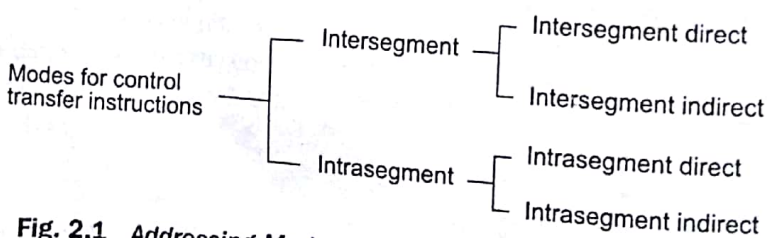


Fig. 2.1 *Addressing Modes for Control Transfer Instructions*

9. Intrasegment Direct Mode In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer IP.

Scanned by CamScanner

The effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP. In case of jump instruction, if the signed displacement (d) is of 8 bits (i.e. $-128 < d < +127$), we term it as *short jump* and if it is of 16 bits (i.e. $-32768 < d < +32767$), it is termed as *long jump*.

Example 2.9

```
JMP SHORT LABEL; LABEL lies within -128 TO +127 from the current IP
content.
```

Thus SHORT LABEL is 8-bit signed displacement.

A 16-bit target address of a label indicates that it lies within – 32768 to + 32767. But a problem arises when one requires a forward jump at a relative address greater than 32767 or backward jump at relative address – 32768; in the same segment. Suppose current contents of IP are 5000H then a forward jump may be allowed at all the displacement DISP so that IP + DISP = FFFFH or DISP = FFFF – 5000 = AFFFH. Thus forward jumps may be allowed for all 16-bit displacement values from 0000H to AFFFH. If displacement exceeds AFFFH i.e. from B000H to FFFFH, then all such jumps will be treated as backward jumps. All such jumps are called NEAR PTR jumps and coded as below.

```
JMP NEAR PTR LABEL
```

10. Intrasegment Indirect Mode In this mode, the displacement to which the control is to be transferred, is in the same segment in which the control transfer instruction lies, but it is passed to the instruction indirectly. Here, the branch address is found as the content of a register or a memory location. This addressing mode may be used in unconditional branch instructions.

Example 2.10

```
JMP [BX]; Jump to effective address stored in BX.
                        JMP [ BX + 5000H ]
```

11. Intersegment Direct In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.

Example 2.11

```
JPM 5000H : 2000H;
Jump to effective address 2000H in segment 5000H.
```

12. Intersegment Indirect In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes, i.e. IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

Example 2.12

```
JMP [2000H];
```

Jump to an address in the other segment specified at effective address 2000H in DS, that points to the memory block as said above.

Forming the Effective Addresses The following examples explain forming of the effective addresses in the different modes.

Scanned by CamScanner

Example 2.13

The contents of different registers are given below. Form effective addresses for different addressing modes.

Offset (displacement) = 5000H

[AX]-1000H, [BX]-2000H, [SI]-3000H, [DI]-4000H, [BP]-5000H,

[SP]-6000H, [CS]-0000H, [DS]-1000H, [SS]-2000H, [IP]-7000H.

Shifting a number four times is equivalent to multiplying it by 16_D or 10_H

(i) Direct addressing mode

```
MOV AX, [5000H]
```

```
DS:OFFSET ⇔  1000H: 5000H
10H* DS ⇔     10000
Offset ⇔    + 5000
            -------
            15000H - Effective address
```

(ii) Register indirect

```
MOV AX, [BX]
```

```
DS:BX ⇔   1000H:2000H
10H*DS ⇔    10000
[BX] ⇔    + 2000
          --------
          12000H - Effective address
```

(iii) Register relative

```
MOV AX, 5000 [BX]
```

```
DS: [5000 + BX]
10H*DS ⇔    10000
Offset ⇔  + 5000
[BX] ⇔    + 2000
          --------
          17000H - Effective address
```

(iv) Based indexed

```
MOV AX, [BX] [SI]
```

```
DS:[BX + SI]
10H*DS ⇔   10000
[BX] ⇔   + 2000
[SI] ⇔   + 3000
         --------
         15000H - Effective address
```

(v) Relative based indexed

```
MOV AX, 5000 [BX] [SI]
```

```
DS: [BX + SI + 5000]
10H*DS ⇔    10000
[BX] ⇔    + 2000
[SI] ⇔    + 3000
Offset ⇔  + 5000
          --------
          1A000 - effective address
```

Below, we present examples of address formation in control transfer instructions.

Scanned by CamScanner

Instruction Set with Machine Cycles and Flag Status (see notes at end of table)

Instruction	Code[1]	B/M/T[2]	Machine[3] Cycles	Flags[4] S D_7	Z D_6	AC D_4	P D_2	CY D_0
ACI DATA : Add 8-bit and CY to A	CE data	2/2/7	F R	✓	✓	✓	✓	✓
ADC REG : Add Reg. and CY to A	1000 1SSS	1/1/4	F	✓	✓	✓	✓	✓
ADC M : Add Mem. and CY to A	8E	1/2/7	F R	✓	✓	✓	✓	✓
ADD REG : Add Reg. to A	1000 0SSS	1/1/4	F	✓	✓	✓	✓	✓
ADD M : Add Mem. to A	86	1/2/7	F R	✓	✓	✓	✓	✓
ADI DATA : ADD 8-BIT TO A	C6 DATA	2/2/7	F R	✓	✓	✓	✓	✓
ANA REG : AND Reg. with A	1010 0SSS	1/1/4	F	✓	✓	1	✓	0
ANA M : AND Mem. with A	A6	1/2/7	F R	✓	✓	1	✓	0
ANI DATA : AND 8-bit with A	E6 data	2/2/7	F R	✓	✓	1	✓	0
CALL ADDR : Call Unconditional	CD addr	3/5/18	S R R W W					
CC ADDR : Call On CY	DC addr	3/5/9–18	S R R W W					
CM ADDR : Call On Minus	FC addr	3/5/9–18	S R R W W					
CMA : Complement A	2F	1/1/4	F					
CMC : Complement CY	3F	1/1/4	F					✓
CMP REG : Compare Reg. with A	1011 1SSS	1/1/4	F	✓	✓	✓	✓	✓
CMP M : Compare Mem. with A	BE	1/2/7	F R	✓	✓	✓	✓	✓
CNC ADDR : Call On No CY	D4 addr	3/5/9–18	S R R W W					
CNZ ADDR : Call On No Zero	C4 addr	3/5/9–18	S R R W W					
CP ADDR : Call On Positive	F4 addr	3/5/9–18	S R R W W					
CPE ADDR : Call On Parity Even	EC addr	3/5/9–18	S R R W W					
CPI DATA : Compare 8-bit with A	FE data	2/2/7	F R	✓	✓	✓	✓	✓
CPO ADDR : Call On Parity Odd	E4 addr	3/5/9–18	S R R W W					
CZ ADDR : Call On Zero	CC addr	3/5/9–18	S R R W W					

Scanned by CamScanner

DAA	: Decimal-Adjust A	27	1/1/4	F	✓	✓	✓	✓	✓	✓
DAD Rp	: Add Reg. Pair to HL	00Rp 1001	1/3/10	F B B		✓	✓	✓	✓	✓
DCR REG	: Decrement Reg.	00SS S101	1/1/4	F	✓	✓	✓	✓	✓	✓
DCR M	: Decrement Mem. Contents	35	1/3/10	F R W	✓	✓	✓	✓	✓	✓
DCX Rp	: Decrement Reg. Pair	00Rp 1011	1/1/6	S						
DI	: Disable Interrupt	F3	1/1/4	F						
EI	: Enable Interrupt	FB	1/1/4	F B						
HLT	: Halt	76	1/2/5	F B						
IN PORT	: Input from 8-bit Port	DB data	2/3/10	F R I						
INR REG	: Increment Reg.	00SS S100	1/1/4	F	✓	✓	✓	✓	✓	✓
INR M	: Increment Mem. Contents	34	1/3/10	F R W	✓	✓	✓	✓	✓	✓
INX Rp	: Increment Reg. Pair	00Rp 0011	1/1/6	S						
JC ADDR	: Jump On Carry	DA addr	3/3/7–10	F R R						
JM ADDR	: Jump On Minus	FA addr	3/3/7–10	F R R						
JMP ADDR	: Unconditional Jump	C3 addr	3/3/10	F R R						
JNC ADDR	: Jump On No Carry	D2 addr	3/3/7–10	F R R						
JNZ ADDR	: Jump On No Zero	C2 addr	3/3/7–10	F R R						
JP ADDR	: Jump On Positive	F2 addr	3/3/7–10	F R R						
JPE ADDR	: Jump On Parity Even	EA addr	3/3/7–10	F R R						
JPO ADDR	: Jump On Parity Odd	E2 addr	3/3/7–10	F R R						
JZ ADDR	: Jump On Zero	CA addr	3/3/7–10	F R R						
LDA ADDR	: Load A Direct	3A addr	3/4/13	F R R R						
LDAX Rp	: Load A from M; memory address is in BC/DE	000X 1010	1/2/7	F R						

Codes[1]

DDD = Binary digits identifying a destination register
SSS = Binary digits identifying a source register
B = 000, C = 001, D = 010, Memory = 110
E = 011, H = 100, L = 101, A = 111
Rp = Register Pair BC = 00, HL = 10
DE = 01, SP = 11

B/M/T[2]

B = Bytes
M = Machine cycles
T = T-states

Machine Cycles[3]

F = Fetch with 4 T-states
S = Fetch with 6 T-states
R = Memory Read
I = I/O Read
W = Memory Write
O = I/O Write
B = Bus Idle

Flags[4]

✓ = Flag is modified according to result
0 = Flag is cleared
1 = Flag is set
Blank = No change in flag, contains its previous state

S = Sign
Z = Zero
AC = Auxiliary Carry
P = Parity
CY = Carry

Scanned by CamScanner

Instruction			Code[1]	B/M/T[2]	Machine[3] Cycles	Flags[4] S D_7	Z D_6	AC D_4	P D_2	CY D_0
LHLD	*ADDR*	*Load HL Direct*	2A addr	3/5/16	F R R R R					
LXI	*Rp, 16-bit*	*Load 16-bit in Reg. Pair*	00Rp 0001 16-bit	3/3/10	F R R					
MOV	*R1,R2*	*Move from Reg. R_2 to Reg. R_1*	01DD DSSS	1/1/4	F					
MOV	*M,R*	*Move from Reg. to Mem.*	0111 0SSS	1/2/7	F W					
MOV	*R,M*	*Move from Mem. to Reg.*	01DD D110	1/2/7	F R					
MVI	*R,DATA*	*Load 8-bit in Reg.*	00DD D110 data	2/2/7	F R					
MVI	*M,DATA*	*Load 8-bit in Mem.*	36 data	2/3/10	F R W					
NOP		*No Operation*	00	1/1/4	F					
ORA	*R*	*OR Reg. with A*	1011 0SSS	1/1/4	F	✓	✓	0	✓	0
ORA	*M*	*OR Mem. Contents with A*	B6	1/2/7	F R	✓	✓	0	✓	0
ORI	*DATA*	*OR 8-bit with A*	F6 data	2/2/7	F R	✓	✓	0	✓	0
OUT	PORT	Output to 8-bit Port	D3 data	2/3/10	F R O					
PCHL		Move HL to Program Counter	E9	1/1/6	S					
POP	Rp	Pop Reg. Pair	11Rp 0001	1/3/10	F R R					
PUSH	Rp	Push Reg. Pair	11Rp 0101	1/3/12	S W W					
RAL		Rotate A Left through CY	17	1/1/4	F					✓
RAR		Rotate A Right through CY	1F	1/1/4	F					✓
RC		Return On Carry	D8	1/3/6–12	S R R					
RET		Return	C9	1/3/10	F R R					
RIM		Read Interrupt Mask	20	1/1/4	F					
RLC		Rotate A Left	07	1/1/4	F					✓
RM		Return On Minus	F8	1/3/6–12	S R R					
RNC		Return On No Carry	D0	1/3/6–12	S R R					
RNZ		Return On No Zero	C0	1/3/6–12	S R R					
RP		Return On Positive	F0	1/3/6–12	S R R					

Scanned by CamScanner

RNC		: Return On No Carry	D0	1/3/6–12	S R R						
RNZ		: Return On No Zero	C0	1/3/6–12	S R R						
RP		: Return On Positive	F0	1/3/6–12	S R R						
RPE		: Return On Parity Even	E8	1/3/6–12	S R R						
RPO		: Return On Parity Odd	E0	1/3/6–12	S R R						
RRC		: Rotate A to Right	0F	1/1/4	F						✓
RST	N	: Restart	11XX X111	1/3/12	S W W						
RZ		: Return On Zero	C8	1/3/6–12	S R R						
SBB	R	: Subtract Reg. from A with Borrow	1001 1SSS	1/1/4	F	✓	✓	✓	✓	✓	
SBB	M	: Subtract Mem. Contents from A with Borrow	9E	1/2/7	F R	✓	✓	✓	✓	✓	
SBI	DATA	: Subtract 8-bit from A	DE data	2/2/7	F R	✓	✓	✓	✓	✓	
SHLD	ADDR	: Store HL Direct	22 addr	3/5/16	F R R W W						
SIM		: Set Interrupt Mask	30	1/1/4	F						
SPHL		: Move HL to Stack Pointer	F9	1/1/6	S						
STA	ADDR	: Store A Direct	32 addr	3/4/13	F R R W						
STAX	Rp	: Store A in M, memory address is in BC/DE	000X 0010	1/2/7	F W						
STC		: Set Carry	37	1/1/4	F						1
SUB	R	: Subtract Reg. from A	1001 0SSS	1/1/4	F	✓	✓	✓	✓	✓	
SUB	M	: Subtract Mem. from A	96	1/2/7	F R	✓	✓	✓	✓	✓	
SUI	DATA	: Subtract 8-bit from A	D6 data	2/2/7	F R	✓	✓	✓	✓	✓	
XCHG		: Exchange DE with HL	EB	1/1/4	F						
XRA	R	: Exclusive OR Reg. with A	1010 1SSS	1/1/4	F	✓	✓	0	✓	0	
XRA	M	: Exclusive OR Mem. with A	AE	1/2/7	F R	✓	✓	0	✓	0	
XRI	DATA	: Exclusive OR 8-bit with A	EE data	2/2/7	F R	✓	✓	0	✓	0	
XTHL		: Exchange Stack with HL	E3	1/4/16	F R R W W						

Codes[1]

DDD = Binary digits identifying a destination register
SSS = Binary digits identifying a source register
B = 000, C = 001, D = 010, Memory = 110
E = 001, H = 100, L = 101, A = 111
Rp = Register Pair: BC = 00, HL = 10; DE = 01, SP = 11

B/M/T[2]

B = Bytes
M = Machine cycles
T = T-states

Machine Cycles[3]

F = Fetch with 4 T-states
S = Fetch with 6 T-states
R = Memory Read
I = I/O Read
W = Memory Write
O = I/O Write
B = Bus Idle

S = Sign
Z = Zero
AC = Auxiliary Carry
P = Parity
CY = Carry

Flags[4]

✓ = Flag is modified according to result
0 = Flag is cleared
1 = Flag is set
Blank = No change in flag, remains in previous state

Scanned by CamScanner