[8086 ma MVI hudaina]

Block diag memory flag reg

Page No.
Date : / /

Addressing Modes in 8086.

(i) Register Addressing Mode.
In this mode, the operand is specified in register i.e. operands are present in specific register.
Example:

MOV AH, BL , Mov CX, DX

Here,
the operand is present in register BL which is transfered to register AH.

(ii) Immediate Addressing Mode.
In this addressing mode, the operand is present in the instruction.
Example:

MOV AH, 45 H
MOV BX, 2070 H

(iii) Direct Addressing Mode.
In this addressing mode, the location of operand is specified in a instruction.
Example:

→ ADD AX, [5000 H]

Here,
data resides in memory location in the data segment whose effective address may be computed using 5000 H as the offset address and content of ~~Haeirs~~ H DS as segmented address.

$$\text{Effective Address (EA)} = DS + 5000.$$
$$= DS \times 10\,H + 5000$$
$$= 20000 + 5000 = 25000\,H$$

→ MOV AL, [1234 H]
Effective Address = DS + 1234.

(iv) Indirect Memory Addressing Mode.
In this mode, effective address of operand is taken directly from one of the base register (BX or BP) or index registe (SI or DI).
Example :
→ MOV CX, [BX]
If DS = 1000H, [BX] = 0008
Effective Address = 10000 + 0008 = DS + [BX]
= 10008 H.//

→ ADD, CX, [SI]
E.A. = DS + [SI]
= 10000 + [SI]

(v) Based Addressing Mode.
In this mode, the effective address is the sum of the displace-ment and the contents of the register BX or BP.
Example :
→ displacement value
MOV AX, 8 [BX]
16 bits

Effective Address = DS + [BX] + 0008

(vi) Indexed Addressing Mode :
In this addressing mode, the P.A. is calculated with the help of displacement value specified in a instruction and index register.
Example:
MOV AX, value [SI]
Physical Address = DS + value + [SI]

Throughput → No. of more ... time for executi
- doesnot decrease

(vii) Based Indexed Addressing Mode.
In this addressing mode, physical address is calculated with indexed register, based register and displacement value.
Example:

MOV AX, 4 [BX] [SI]

Physical Address = DS + 0004 + [BX] + [SI]

Pipelining.

clock	1	2	3	4	5	6	7
Instruction i	Fetch	Decode	Execute ~~Decode~~				
Instruction i+1		Fetch	Decode	Execute			
Instruction i+2			Fetch	Decode	Execute		
Instruction i+3				Fetch	Decode	Execute	
Instruction i+4					Fetch	Decode	Execute

Fig: Instruction Pipeline.

DOS interrupt signal function

INT21H

It is DOS interrupt signal. There are different functions of INT 21 H.

(1) MOV AH, 01
INT 21H

There two instructions together will accept one single character from Keyboard and store it in AL register.

Page No.
Date : / /

(2) MOV AH, 02 H
INT 21 H

These two instructions will display a single character stored in the DL register.
Example:
MOV DL, 41H : 41H is ascii value of 'A' so it display 'A'
MOV AH, 02H
INT 21H

(3) MOV AH, 09 H
INT 21 H

These two instructions will display an entire string stored in memory location. For this, these two condition is a must.
- ~~It~~ the offset address of the string has to be stored in DX register.
- The string must be terminated by a $.

~~Example:~~
(4) MOV AH, 4CH or MOV AX, 4C00H
INT 21 H

These two instructions will terminate the program execution.

(5) MOV AH, 0AH
INT 21H

To input a string, we use these two instructions.

EXAMPLES

MOV ARRAY[SI],BL : Copys BL into the data segment memory location addressed by ARRAY plus SI.

MOV LIST[SI+2],CL : Copys CL into the data segment memory location addressed by sum of LIST, SI and 2.

7. Base Relative plus Index Addressing Mode

The base relative plus index addressing mode is similar to the base plus index addressing mode but it adds a displacement to form a memory address.

Transfers a byte or word between a register and the memory location addressed by a base and an index register plus a displacement.

Instruction	Source	Destination
MOV[BX+SI+05], CL	Register CL	Assume DS =1000H assume BX=0300H Assume SI=0200H 10000H+0300H+0200H+05H=10505H Memory Location 10505H

EXAMPLES

MOV LIST[BP+DI],CL : Copys CL into the stack segment memory location addressed by the sum of LIST, BP and DI

MOV DH,[BX+DI+20H] : Copys the byte contents of the data segment memory location addressed by the sum of BX, DI and 20 Hint oDH

Pipelining

Pipelining is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline is divided in **stages**. Each stage completes a part of an instruction in parallel. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end.

Pipelining does not decrease the time for individual instruction execution. Instead, it increases instruction throughput. The **throughput** of the instruction pipeline is determined by how often an instruction exits the pipeline.

Because the pipe stages are hooked together, all the stages must be ready to proceed at the same time. We call the time required to move an instruction one step further in the pipeline ***a machine cycle***. The ***length*** of the machine cycle is determined by the time required for the slowest pipe stage.

The pipeline designer's *goal is to balance the length of each pipeline stage*. If the stages are perfectly balanced, then the time per instruction on the pipelined machine is equal to

Time per instruction on nonpipelined machine Number of pipe stages

Under these conditions, the speedup from pipelining equals the number of pipe stages. Usually, however, the stages will not be perfectly balanced; besides, the pipelining itself involves some overhead.

Instruction Pipeline

Any architecture can be pipelined by making each clock cycle into a pipe stage.

Clock#	1	2	3	4	5	6	7
Instruction *i*	Fetch	Decode	Execute				
Instr. *i* + 1		Fetch	Decode	Execute			
Instr. *i* + 2			Fetch	Decode	Execute		
Instr. *i* + 3				Fetch	Decode	Execute	
Instr. *i* + 4					Fetch	Decode	Execute

Table 2.1: Instruction Pipeline

Here instruction i is fetched in clock #1. After it has been fetched it is decoded in clock #2 and at the same time next instruction i. e. instr. i+1 is fetched. At Clock#3, instruction i is executed, instr. i+1 is decoded and instr. i+2 is fetched and so on.

Hence at the end of clock #3, instruction i is executed. Sometime later at the end of clock #4 instruction i+1 is executed.

If we assume that unpipelined architecture took 3 ns then this pipelined architecture will take 1 ns to finish a stage (fetch, decode and execute).

Advantages of pipelining:

⌘ The execution unit always reads the next instruction byte from the queue in BIU. This is faster than sending out an address to the memory and waiting for the next instruction byte to come.

⌘ In short pipelining eliminates the waiting time of EU and speeds up the processing. The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in queue. 8086 BIU normally obtains two instruction bytes per fetch.

QUESTIONS

1. Explain 8085 architecture with the help of its block diagram.
2. Why addressing modes are required in microprocessor? Explain the addressing modes of 8085 architecture with examples.
3. Why flags are required in microprocessor? Explain 8085 flags with suitable facts and figures.
4. Draw a neat pin diagram of 8085 microprocessor and explain it.
5. Explain 8086 architecture with the help of its EU and BIU.
6. What is the advantage of having more addressing modes in 8086 microprocessor? Explain the addressing modes of 8086 architecture with examples.
7. Why is flags? Explain 8086 flags with suitable facts and figures.
8. What is segmented memory? List out the advantages and disadvantages of segmentation.
9. What is pipeline? Explain instruction pipeline in brief.

❑❑❑