MICROPROCESSOR

CSC 405

Subject Incharge

Dakshata Panchal
Assistant Professor
email: dakshatapanchal@sfit.ac.in

CSC405 Microprocessor

Module 2

Instruction Set and Programming

Contents as per syllabus

- Addressing Modes
- Instruction set – Data Transfer Instructions, String Instructions, Logical Instructions, Arithmetic Instructions, Transfer of Control Instructions, Processor Control Instructions
- Assembler Directives and Assembly Language Programming, Macros, Procedures
- Mixed Language Programming with C Language and Assembly Language.
- Programming based on DOS and BIOS Interrupts (INT 21H, INT 10H)

Addressing Modes

The different ways in which a processor can access the operands of an instruction are called **Addressing Modes.**

Operands can be located anywhere

- Memory locations
- Registers
- I/O ports
- Present in the instruction itself.

Back

Instruction

A command given by the user to perform a certain task

It is written in a specific format

Opcode Operand field(s)

Opcode is a unique hexadecimal encoding of the given instruction.

Opcode depends on the operation as well as the operand fields.

Back

Instruction

Generally all instructions have

- Operation (mnemonic)
- Destination (operand)
- Source (operand)

Mnemonic Destination, Source

Some instructions may have only destination, while some may have more than one source, or more than one destination. (details later)

Back

Instruction

- Source of data can be:
    - Immediate data (available on the instruction)
    - A register
    - Specified address (memory or I/O)

- Destination can be:
    - A register
    - Specified address (memory or I/O)

Bac
k

Addressing Modes

There are 5 fundamental addressing modes

1. Immediate addressing Mode
2. Register Addressing Mode
3. Direct Addressing Mode
4. Indirect Addressing Mode
   1. Register Indirect
   2. Register Relative
   3. Base Index
   4. Base Relative Indexed
5. Implied Addressing Mode

Back

Addressing Modes

1. Immediate addressing Mode:

- In this mode, data is given in the instruction
- Doesn't involve computation of address
- Example:

MOV AX, 0170h

AX	0000 0001 0111 0000

Back

Addressing Modes

2. Register Addressing:

- In this mode, Data is given by the Register.
- The operands on which instruction operate are stored in the specified registers
- For 16-bit operand – AX, BX, CX, DX, SI, DI, SP, BP
- For 8-bit operand – AH,AL, BH,BL, CH,CL, DH,DL
- Example:

MOV AX, BX

Back

Addressing Modes

3. Direct Addressing:

In this mode, <u>Address is given in the instruction</u>

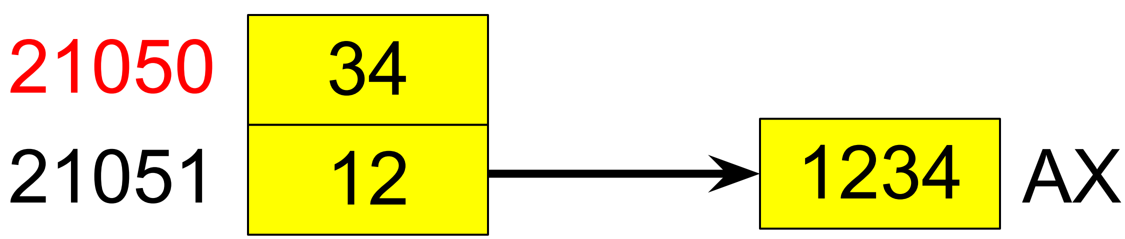
Example: MOV AX, [1050H]

(or) MOV AX, DATA

By default DS is segment address

let, DS = 2000h

EA = 2000*10 + 1050 = 21050



Back

Addressing Modes

Practice:

Assume with in data segment,

1. Take the data of location 4000 into BL
2. Take the data of location 5000 into CL
3. Add Both BL and CL, so that results comes in BL
4. Then add number “40” into BL and results in BL
5. Finally store the result in location 6000

Write the instruction?

Bac
k

Addressing Modes

Practice:

Answer:

```
MOV BL,[4000]
MOV CL,[5000]
ADD BL,CL
ADD BL,40
MOV [6000], BL
```

Back

Addressing Modes

4. Indirect Addressing:

In this mode, Address is given in Register

Types of Indirect Addressing Mode:-

1. Register Indirect Addressing
2. Register Relative Addressing
3. Base Index Addressing
4. Base Relative Indexed Addressing

Back

Addressing Modes

4(a): Register Indirect Addressing:

- Address given by the Register
- Address of the data is present in one of the base or index registers – BX, BP, SI or DI

    Offset is always 16-bits wide

    Data can be 8-bit or 16-bit

Example: MOV AL, [BX]

let BX = 1050 & DS = 2000

then EA = 2000*10 + 1050 = **21050**

Back

Addressing Modes

4(a): Register Indirect Addressing:

Suppose in the data segment there is some location, let's say 6000 and we want any register(let’s say CL)to contain the data from that location.

How can we write the instruction??

Use direct addressing mode

Or

Indirect addressing mode

Back

Addressing Modes

4(a): Register Indirect Addressing:

Using direct addressing mode

MOV CL,[6000]

CL□DS :[6000]

Or

Indirect addressing mode

MOV BX, 6000

MOV CL,[BX]

CL,□DS:[6000]

So What's the difference??

Back

Addressing Modes

To understand the difference Answer few Question?

Get the data from location 6000 which addressing mode will you use?

Ans: Direct addressing mode

MOV CL,[6000]

CLDS :[6000]

Now get the data of 100 locations starting from location 6000, so which addressing mode will you use?

Back

Addressing Modes

Ans: if you use Direct addressing mode, then

100 times you need to write the same instruction

MOV CL,[6000]

MOV CL,[6001]

MOV CL,[6002]

MOV CL,[6003] …. And so on

Is it feasible????

No!!!

Back

Addressing Modes

Ans:

But if we use indirect addressing mode, then MOV BX, 6000

MOV CL, [BX]

INC BX

CL□DS:[6000]

Back

Addressing Modes

Ans:

But if we use indirect addressing mode, then MOV BX, 6000

$\boxed{\begin{array}{l}\text{MOV CL, [BX]} \\ \text{INC BX}\end{array}}$ Put this in a loop

CL□DS:[6000]

So whenever we want to work on a series of location, we use indirect addressing mode.

Back

Addressing Modes

Ques. Take the data from 2000 location to CL register and store it at location 3000. Which addressing mode will you use to write an instruction.

Ans:

Direct Addressing Mode

MOV CL, [2000]

MOV [3000], CL

Back

Addressing Modes

Ques. Write a program to transfer a block of data from location 2000 to 3000

Ans:

```
MOV SI, 2000H
MOV DI, 3000H
MOV CL, [SI]
MOV [DI], CL
INC SI
INC DI
```

Back

Addressing Modes

Ques. Block Inversion Program

Ans:

```
MOV SI, 2000H
MOV DI, 3000H
MOV CL, [SI]
MOV [DI], CL
INC SI
DEC DI
```

Back

Addressing Modes

But we need to follow some Rule:

Rule 1:

In indirect addressing mode, address must be given by specific Register.

There are 4 general purpose register, AX, BX, CX, DX.

Out of these only **BX** register can be used to give the memory address and Only these offset Register can be used **BP**,**SI** and **DI**

Back

Addressing Modes

But we need to follow some Rule:

Rule 2:

- If you use BX, SI or DI it will be used in data segment
- If you use BP, by default it will work on stack segment

Back

Addressing Modes

4(b): Register Relative Addressing:

- Effective Address is the sum of

  16-bit offset given in a base register (BX or BP)

  + 8 or 16-bit displacement (+ or – constant)

  + Segment Register (DS or SS) * 10

- Example: MOV AL, [BX+04]

  let BX = 1050 & DS = 2000

  then EA = 2000*10 + 1050 + 4 = **21054**

Back

Addressing Modes

4(c): Base Indexed Addressing:

Index addressing:-

Used in arrays to access the nth element

Example: MOV AL, Array[3]

let   Array = {10, 23, 08, 31}

if offset of Array = 1050 and DS = 2000

then EA = 2000*10 + 1050 + 3 = 21053

Back

Addressing Modes

4(c): Base Indexed Addressing:

Effective Address is the sum of

16-bit offset given in a base register (BX or BP)

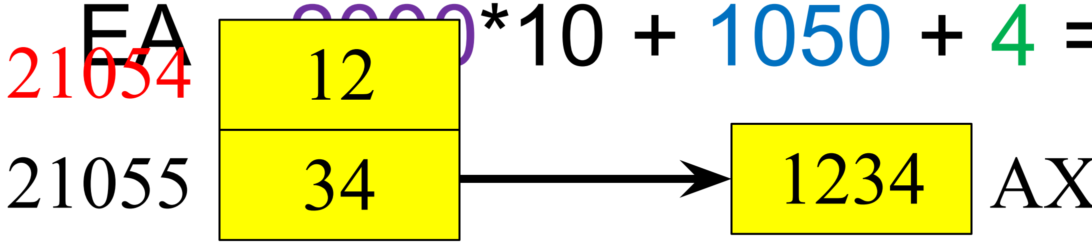
displacement in an index register (SI or DI)

Segment Register (DS or SS) * 10

Example: MOV AX, [BX+SI]

let BX = 1050 SI = 4 & DS = 2000

then EA = 2000*10 + 1050 + 4 = 21054



Back

Addressing Modes

4(d): Relative Base Indexed Addressing:

Effective Address is the sum of 16-bit offset given in a base register (BX or BP)

+index register (SI or DI)

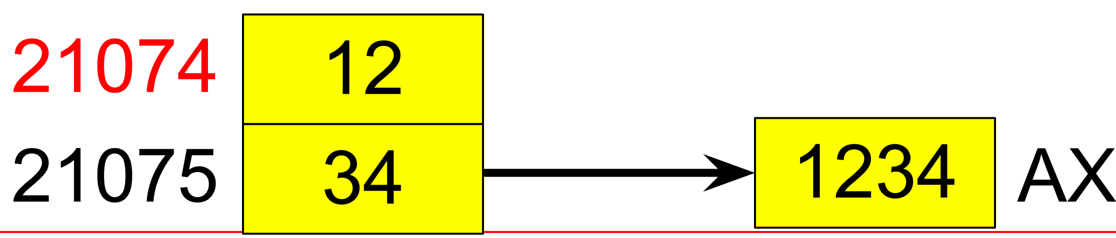
+8 or 16-bit displacement (+ or – constant)

+Segment Register (DS or SS) *10

Example: MOV AX, [BX+SI+20]

let BX = 1050 SI = 4 & DS = 2000

then EA = 2000*10 + 1050 + 4 + 20 = 21074



Back

Addressing Modes

5. Implied/ Implicit Addressing:

No operands are used to execute these instructions

Example:

NOP : No Operation

CLC : Clear Carry Flag (CF = 0)

START: Execution Starts

Back

Instruction Set

Processors execute a machine code.

- **MACHINE CODE**: The sequence of numbers that flip the switches in the computer on and off to perform a certain job of work - such as addition of numbers, branching, multiplication, etc.
- Machine specific and well documented by the implementers of the processor.
- **OPCODE**: It is a number interpreted by your machine(virtual or silicon) that represents the operation to perform.
  - E.g., **MOV BX, data**

    47H [1050] (47H is opcode for MOV into BX, 16-bits of memory)

Back

Opcode & Operands

- **OPCODE** is short for 'Operation Code'.
- An opcode is a single instruction that can be executed by the CPU.
- In machine language it is a binary or hexadecimal value such as 'B6' loaded into the instruction register.
- In assembly language mnemonic form of an opcode is a command such as MOV or ADD or JMP
- **OPERAND** is the data on which the operation is performed

Instruction Format

- General format of an assembly language instruction :

Label: **Mnemonic** **Operand, Operand** **;Comment**

- Every instruction starts on a new line
- Labels must be followed by colon (:) (It is an identifier)
- Instructions operate on implicit (zero), one, two, or three operands
- Multiple operands must be separated by a comma
- Comment is ignored by assembler

8086 Instruction Set

Instructions in 8086 are classified into the following 8 functional groups

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Bit Manipulation Instructions
4. Branch Instructions
5. String Operation Instructions
6. Processor Control Instructions
7. Iteration Control Instructions
8. Interrupt Instructions

1. Data Transfer

- MOV : R/M/D □ R/M
- XCHG : R/ M □□ R/M
- LEA : EA of operand □ R
- PUSH/POP : Stack manipulation
- PUSHA/POPA : Copy all registers □□ Stack
- PUSHF/POPF : Flag register □□ Stack
- IN/ OUT : R □□ port

Data Transfer Instructions

Instruction	Syntax	Example
Move	MOV destination,	MOV AX, BX
Pop	POP destination	POP AX

- Push decrements SP by 2

  PUSH DX : SP←SP-2; Contents of DX copied onto stack

- Pop increments SP by 2

  POP AX SP←SP+2; copy stack top contents to AX

2. Arithmetic and Logical Instructions

- Add or Subtract (ADD / SUB)
- Add/Subtract with Carry/Borrow (ADC / SBB)
- Increment / Decrement (INC / DEC)
- Compare / Negate (CMP / NEG)
- ASCII adjust (AAA/ AAS/ AAM/ AAD)
- Decimal Adjust (DAA/DAS)
- Multiplication/Division (MUL/ IMUL/ DIV/ IDIV)
- Logical (AND/ OR/ XOR/ NOT)
- Convert (CBW/ CWD)

Arithmetic Instructions

- NEG: Negate changes sign of operand (2's complement)
- CMP: Compare
  - If (dest > src): CF=0, ZF=0, SF=0
  - If (dest < src): CF=1, ZF=0, SF=1
  - If (dest = src): CF=0, ZF=1; SF=0

Instruction	Syntax	Example
Add / Subtract	ADD/SUB	ADD/SUB BX, AX
	source	
Negate	NEG destination	NEG BX

Arithmetic Instructions

Instruction	Syntax	Example
Multiply (unsigned)	MUL source	MUL BX
Integer Divide (signed)	IDIV source	IDIV CL

- Accumulator is default operand in multiply and divide
- MUL BX → DX:AX = AX*BX
- MUL BH → AX = AL*BH
- DIV BX → AX = Quotient; DX = Remainder
- DIV BL → AL = Quotient; AH = Remainder

Arithmetic Instructions

Instruction	Syntax	Example	
		Before	After
ASCII adjust for Addition	AAA	AH=00H,	AH=01H,
ASCII adjust for Division	AAD	AL=09H	AL=39H
Decimal adjust for Addition	DAA	AL=6BH	AL=71H
Decimal adjust for Subtraction	DAS	AL=2EH	AL=28H

- AAA and AAS check only lower nibble of AL. DAA and DAS check lower and higher nibble of AL.
- AAM : AL = AL/10, AH = AL mod 10
- AAD : AL = AH*10+AL, AH=00H

3. Bit manipulation Instructions

Instruction	Syntax	Example
Logical AND	AND destination, source	AND AX, F000H
Logical OR	OR destination, source	OR BX, AX
[illegible]	[illegible]	[illegible]
Logical NOT	NOT destination	NOT AL

- NOT inverts each bit in destination (1’s complement)

Bit manipulation

Instruction	Syntax	Example	
		Before	After
Convert Byte to Word	CBW	AH=??, AL=10011011	AH=11111111, AL=10011011
Convert Word to D Word	CWD	DX=?? AX=10101101 10010001	DX=11111111 11111111 AX=10101101 10010001

Bit manipulation

Instruction	Syntax	Example
Shift Logical Left	SHL destination, source	SHL BL, 1
Shift Logical Right	SHR destination, source	SHR BL, 1
Shift Arithmetic Left	SAL destination, count	SAL AL, 2
Shift Arithmetic Right	SAR destination,	SAR AL, 2
Rotate through carry	RCR destination,	RCR CL, 4

- SAL is same as SHL, LSB becomes 0
- In SHR, MSB is padded with 0. In SAR, MSB is padded with previous MSB bit

4. Branch Instructions

- CALL : Call a subroutine
- RET : Return to main procedure
- JMP : Provide an address of the next instruction
  - Unconditional Jump (JMP)
    - Always jumps to specified address
  - Conditional Jump (JZ, JNC...)
    - Checks the status of certain flag bits (depending on the condition) and jumps only if condition is satisfied.

5. String Operation Instructions

- Load
  - string/ string byte/ string word (LODS/LODSB/LODSW)
- Store
  - string/ string byte/ string word (STOS/STOSB/STOSW)
- Compare
  - string/ string byte/ string word (CMPS/CMPSB/CMPSW)

6. Processor Control Instructions

- STC : Set CF = 1
- CLC : Clear CF = 0
- CMC : Complement CF
- STD : Set DF = 1
- CLD : Clear DF = 0
- STI : Set IF = 1
- CLI : Clear IF = 0

7. iteration Control Instructions

- LOOP: Jump back to address until CX = 0
- LOOPE/ LOOPZ: Jump back to address until ZF = 1 and CF = 0
- LOOPNE/ LOOPNZ: Jump back to address until ZF = 0 and CX = 0
- JCXZ : Jump to address if CX = 0

8. Interrupt Instructions

- INT : Interrupt execution by calling a stored program
- INTO : Execute interrupt program if $OF = 1$
- IRET : Return from Interrupt to main program

III. Assembly Language Programming

Declaring Variables

- Define Byte – DB

```
SUM DB 0
```

- Define Word – DW
- Define Double Word – DD
- Define Quad Word – DQ
- Define Ten Bytes – DT
- Arrays – DUP(size)

```
LIST DB DUP(20)
```

Program Directives

- SEGMENT
- ENDS
- ASSUME
- OFFSET
- END
- PROC
- ENDP

DOS System Call: INT 21H

- Chararcter I/O
- File Operations
- Record Operations
- Directory Operations
- Disk Management
- Process Management
- Memory Management
- Network Functions
- Time and Date Functions
- Miscellaneous System Functions
- Reserved Functions

IV. Mixed Language Programming

Mixed Language Program

- Can be used with C/C++
- Called inline assembler. Used to insert short, limited assembly language code
- Need to use an **asm** block inside main function
- No need to use data definition directives

Assembly Program

```
data segment
   ..........
data ends
code segment
assume cs:code,ds:data
   start:
    mov ax,data
    mov ds,ax
    /*

    instructions

    */
code ends
end start
```

C/C++ program

```
void main()
{
  int a,b;
  int c=0;
  clrscr();
  printf("Enter 1st no : ");
  scanf("%d",&a);
  printf("Enter 2nd no : ");
  scanf("%d",&b);
  c = a + b;
  printf("Sum  = %d",c);
  getch();
}
```

Example

```
void main()
{
  int a,b;
  int c=0;
  clrscr();
  printf("Enter 1st no : ");
  scanf("%d",&a);
  printf("Enter 2nd no : ");
  scanf("%d",&b);

asm{
    mov ax,a
```

Assembly program - Examples

Addition of 2 nos

Assembly Programming

STEP 1: Define variables (numbers, strings, arrays, etc.)

```
data segment
    msg1 db 10,13, “Enter 1st no : ”
    msg2 db 10,13, “Enter 2nd no : ”
    msgres db 10,13, “result : ”
data ends
```

Assembly Programming

STEP 2: Define code segment, move data into ds

```
code segment
    assume cs:code, ds:data
    start:
        mov ax,data
        mov ds,ax
```

Assembly Programming

STEP 3: Take input

```
mov ax,data
mov ds,ax
; display msg1
mov dx,offset msg1
mov ah,09h
int 21h
```

Assembly Programming

STEP 4: add and store

```
mov ah, 00h
add al, bl
aaa
mov cx,ax
; convert decimal to ascii
add cx,3030h
```

Assembly Programming

STEP 5: display result

```
mov ah, 09h
mov dx, offset msgres
int 21h
mov dl,ch
mov ah,02h
int 21h
mov dl,cl
mov ah,02h
int 21h
```

Assembly Programming

STEP 6: Terminate program

```
    mov ah, 4ch
    mov al, 00h
    int 21h
code ends
end start
```

Questions

1. List and explain the addressing modes of 8086
2. Compare direct and indirect addressing modes of 8086, with examples.
3. List and explain the types of instructions used in 8086 with examples.

THE END!

Have a nice day!