Computer System Architecture

Second Lecture

Format for Assembly Language Program

There are 2 types of executable programs:

1. **.COM** program
2. **.EXE** program

.COM program

- consist of one segment that contains code, data and the Stack
- is useful as a small utility program or as a resident program (one that is installed in memory and is available while other programs run)

.EXE program

- consist of separate code, data and stack segments.
- is used for more serious programs.

Assembly language can be written by using either .COM or .EXE format.

DEBUG Program

- The DEBUG program is used for testing and debugging executable programs which include to:

1. viewing the content of the main memory (MM)
2. enter programs in memory
3. trace the execution of a program

- DEBUG also provides **a single-step mode**, which allows you to execute a program one instruction at a time, so that you can view the effect of each instruction on memory locations and registers.
- Note : - DEBUG set the Trap flag.

DEBUG Commands

- **A** :**A**ssemble symbolic instructions into machine code
- **N** :**N**ame a program
- **P** : **P**roceed or execute a set of related instruction
- **Q** :**Q**uit the debug session
- **T** :**T**race the execution of one instruction
- **U** :**U**nassembled the machine code into symbolic code
- **D** :**D**isplays content of memory locations.
- **E** :**E**nter data into memory beginning at specific location
- **R** : Displays the content of all **R**egisters
- **G** : Run executable program into memory(G means "**G**o")
- **W** :**W**rite a program onto disk

Rules of DEBUG Commands

- DEBUG does not distinguish between lowercase and uppercase letters.
- DEBUG assumes that all numbers are in hexadecimal format.
- Spaces in commands are used only to separate parameters
- Segments and offset are specified with a colon, in the form **segment: offset**

Assembly Language Example

- Assembly Language program can be written using the DEBUG command “**A**” or “**a**”. (A ⇒ Assemble)
- Example:

A 100 <enter>

xxxx : 0100	**MOV CL, 42**
xxxx : 0102	**MOV DL, 2A**
xxxx : 0104	**ADD CL, DL**
xxxx : 0106	**NOP**

To view machine code for the assembly language entered, use the “u” or “U” command. (U ⇒ Un-assemble)

-U 100, 106

xxxx : 0100	**B142**	**MOV**	**CL, 42**
xxxx : 0102	**B22A**	**MOV**	**DL, 2A**
xxxx : 0104	**00D1**	**ADD**	**CL, DL**
xxxx : 0106	**90**	**NOP**	

The machine code for the instruction entered

- To execute the above program, use the “**r” or “R”** command followed by the “**T” or “t**” command.

Data Transfer Instructions

MOV instruction

- The MOV instruction copies data from one location to another, using this format :

```
MOV  destination,source ;copy source operand to destination
```

- The data are copied from source to destination and the source operand remains unchanged. The **MOV** instruction can take one of the following five forms:

- MOV register, register
- MOV register, immediate
- MOV memory, immediate
- MOV register, memory
- MOV memory, register

Note

- **MOV instruction can't:**
  - set the value of the CS and IP registers.
  - copy value of one segment register to another segment register *(should copy to general register first)*.
  - ~~***MOV ES, DS***~~
  - copy immediate value to segment register *(should copy to general register first)*.
  - ~~***MOV DS, 100***~~
  - transfer data from memory to memory.

XCHG instruction

XCHG swap the two data items

- As in the MOV instruction, both operands cannot be located in memory. It can take one of the following forms:
  - XCHG register, register
  - XCHG register, memory
  - XCHG memory, register
- The XCHG instruction do not need a third register to hold a temporary value in order to swap two values. For example, we need three MOV instructions to perform exchange AX,DX registers.
  - **MOV CX,AX**
  - **MOV AX,DX**
  - **MOV DX,CX**

XCHG instruction

- It is also useful to swap the two bytes of 16-bit data .The following example.
- Example:
  - **MOV AL, 5**
  - **MOV AH, 2**
  - **XCHG AL, AH ; AL = 2, AH = 5**

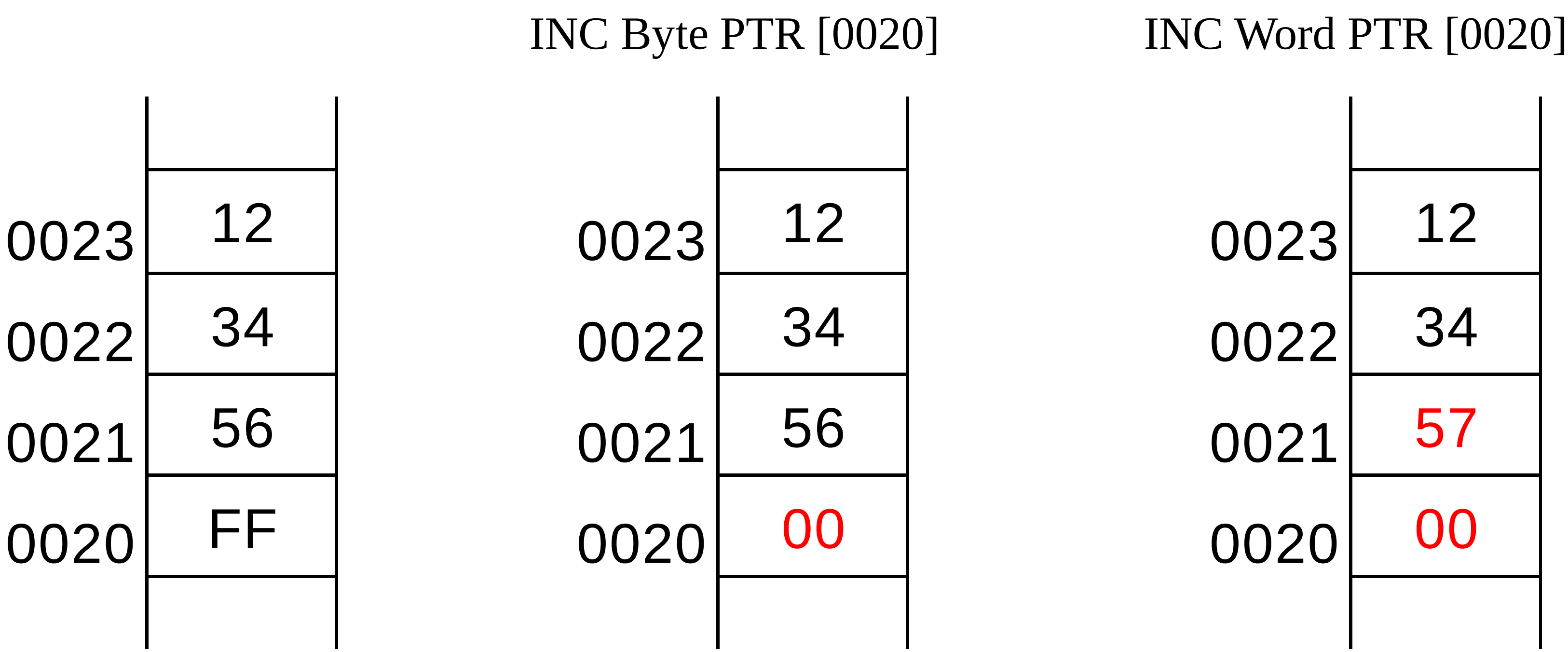
Arithmetic Instructions

INC and DEC Instructions

- Add 1, subtract 1 from destination operand
  - operand may be register or memory
- **INC** *destination*
  - *destination* ← *destination* + 1
- **DEC** *destination*
  - *destination* ← *destination* – 1
- It does not make sense to use an immediate operand such as
  - ~~**INC 55 or DEC 109.**~~

PTR Operator

PTR Operator - For some instructions, the size of the operand is not clear (INC [20H]).



ADD and SUB Instructions

- **ADD** *destination, source*
  - *destination* ← *destination* + source
- **SUB** *destination, source*
  - *destination* ← *destination* – source
- Same operand rules as for the **MOV** instruction

Flags Affected by Arithmetic

- The ALU has a number of status flags that reflect the outcome of arithmetic operations
  - based on the contents of the destination operand
- Essential flags:
  - Zero flag – destination equals zero
  - Sign flag – destination is negative
  - Carry flag – unsigned value out of range
  - Overflow flag – signed value out of range
- The **MOV** instruction never affects the flags.

Zero Flag (ZF)

Whenever the destination operand equals Zero, the Zero flag is set.

```
MOV CX,1
SUB CX,1                    ; CX = 0, ZF = 1
```

A flag is **set** when it equals 1.

A flag is **clear** when it equals 0.

Sign Flag (SF)

The Sign flag is set when the destination operand is negative. The flag is clear when the destination is positive.

```
MOV CX,0
SUB CX,1        ; CX = -1, SF = 1
ADD CX,2        ; CX = 1, SF = 0
```

Overflow Flag (OF)

The Overflow flag is set when the signed result of an operation is invalid or out of range.

Example 1

```
MOV AL,+127          ;+127 decimal number
ADD AL,1             ; OF = 1,   AL = ??

; EXAMPLE 2
MOV AL,7FH
ADD AL,1             ; OF = 1,   AL = 80H
```

The two examples are identical at the binary level because 7Fh equals +127. To determine the value of the destination operand, it is often easier to calculate in hexadecimal.

Addressing mode

- **Register addressing**:
  - E.g. **MOV BX,DX**
  - **ADD AX, BX**

  because processing data between registers involves no reference to memory, it is the fastest type of operations
- **Immediate addressing**

  The second operand contains a constant value .The first operand is never an immediate value.

  E.g. **MOV AX, 0245H ;move 0245H to AX register**

  ADD BX,25 ;add 25 to BX

Addressing mode

- **Direct memory addressing**
  - One of operand references a memory location and the other operand references a register.
  - The address of the data in memory comes immediately after the instruction.
  - MOV CX,[200H] ;Move word from memory at offset 200H
  - The omission of square brackets, as in MOV CX,200H indicates an immediate value.

Addressing mode

- Assume data segment offset begins at 200H.
- The data is placed in memory location: (all data in hexadecimal format)

```
DS:0200 = 25
DS:0201 = 12
DS:0202 = 15
DS:0203 = 1F
DS:0204 = 2B
```

-

```
MOV AL,0
ADD AL,[200] ;ADD the contents of ds:200 to AL
ADD AL,[201]  ; ADD the contents of ds:201 to AL
ADD AL,[202]  ;Add the contents of ds:202 to AL
ADD AL,[203]
ADD AL,[204]
```

Addressing mode

- **Indirect memory addressing**
  - **E.g. MOV AX, [SI]**
  - Value stored in the SI register is used as the offset address.
  - For variables containing a single element this would have little value; but for a list of items, a pointer may be increment to point to each element.

Addressing mode

- 8088/86 allows only the use of registers BX, SI, and DI as offset registers for the data segment. The term pointer is often used for a register holding an offset address. In the following example, BX is used as a pointer:

```
MOV   AL,0          ;initialize AL
MOV   BX,0200H      ;BX points to offset addr of first byte
ADD   AL,[BX]       ;add the first byte to AL
INC   BX            ;increment BX to point to the next byte
ADD   AL,[BX]       ;add the next byte to AL
INC   BX            ;increment the pointer
ADD   AL,[BX]       ;add the next byte to AL
INC   BX            ;increment the pointer
ADD   AL,[BX]       ;add the last byte to AL
```

MUL Instruction(Unsigned Multiply)

Multiplies an 8-, 16-, or 32-bit operand by either AL, AX or EAX.

MUL r/m8

MUL r/m16

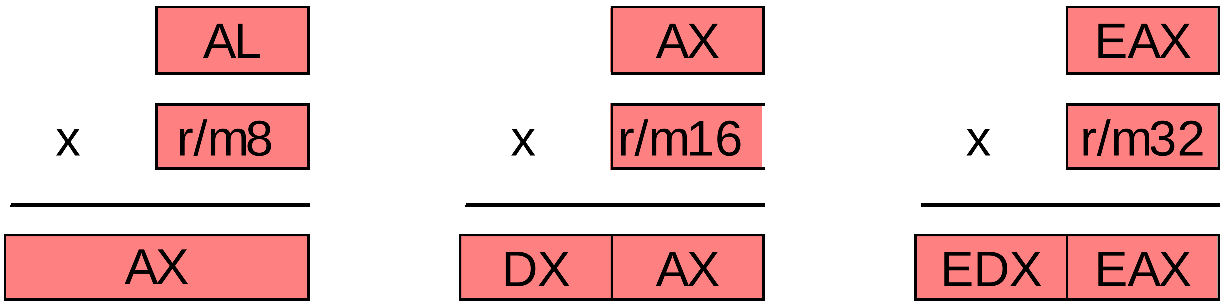
MUL r/m32

r……….register

m………memory

MUL Instruction(Unsigned Multiply)

- Note that the product is stored in a register (or group of registers) .The operand can be a register or a memory operand



MUL Examples

```
MOV   AL, 5H
MOV   BL, 10H
MUL   BL                ; AX = 0050H, CF = 0
```

(no overflow - the Carry flag is 0 because the upper half of AX is zero)

MUL Examples

100h * 2000h, using 16-bit operands:

```
MOV AX,100
MOV BX,2000
MUL  BX        ; DX:AX = 00200000h, CF=1
```

The Carry flag indicates whether or not the upper half of the product contains significant digits.

DIV Instruction(Unsigned Divide)

- Performs 8-, 16-, and 32-bit division on unsigned integers.

DIV *register/memory*

DIV Instruction

Dividend	Divisor	Quotient	Remainder
AX	r/m8	AL	AH
DX:AX	r/m16	AX	DX
EDX:EAX	r/m32	EAX	EDX

DIV Examples (8-bit Unsigned Division)

```
MOV   AX,0083H        ;dividend
MOV   BL, 2H          ;divisor
DIV   BL              ; AL = 41h, AH = 01h
```

Quotient is 41h, remainder is 1

DIV Examples

Divide 8003h by 100h, using 16-bit operands:

```
MOV DX,0          ; clear dividend, high
MOV AX,8003H      ; dividend, low
MOV CX,100H       ; divisor
DIV CX            ; AX = 0080h, DX = 3
```

Exercise . . .

EX1. For each of the following marked entries, show the values of the destination operand and the Sign, Zero, and Carry flags:

```
MOV AX,00FFH
ADD AX,1
SUB AX,1
ADD AL,1
MOV BH,6CH
ADD BH,95H
MOV AL,2
SUB AL,3
```

EX2. Show how the flags register affected by:

A) MOV BH,38H
ADD BH,2Fh

B)MOV AL,9CH
MOV DH,64H
ADD AL,DH

C) MOV AX,34F5H
ADD AX,95EB

Exercise

- EX 3. The value 2505H is stored in locations 130 and 131, and 1C04H is stored in locations 132 and 133. What is the effect of the following related instructions?
  - (a) MOV BX,[0130]
  - (b) ADD BX,[0132]
  - (c) MOV [0134],BX
- EX4. What will be the hexadecimal values of DX, AX, and the Carry flag after the following instructions execute?

```
MOV AX,1234H
MOV BX,100H
MUL BX
```

- EX5 .What will be the hexadecimal values of DX and AX after the following instructions execute

```
MOV DX,0087H
MOV AX,6000H
MOV BX,100H
DIV BX
```