Computer Organization & Assembly Language
(Lecture 03)

Omar Bin Samin
Lecturer
Institute of Management Sciences, Peshawar.

General Knowledge

- Base Address
  - Starting Address of the segment
- Offset Address
  - Distance from the base Address
- Register name starting with ‘R’ means the register size is of 64-bit
- Register name starting with ‘E’ means the register size is of 32-bit

IA-32

- IA-32 stands for "Intel Architecture, 32-bit”
- It is also known as i386
- It is the 32-bit version of the x86 (8086 microprocessor) instruction set architecture
- First implemented in the 80386 microprocessor in 1985

Basic Program Execution Registers

- Registers are high-speed storage locations directly inside the CPU, designed to be accessed at much higher speed than conventional memory

- The registers are grouped into 3 categories:
  - Segment registers
  - General registers
  - Control registers

Segment Registers

Segment Registers

Segments are specific areas defined in a program for containing data, code and stack

1. Code Segment
2. Data Segment
3. Stack Segment
4. Extra Segment

- **Code Segment (CS)**
  - It contains all the instructions to be executed
  - A 16-bit Code Segment Register or CS Register stores the starting address of the code segment
- **Data Segment (DS)**
  - It contains variables (data)
  - A 16-bit Data Segment Register or DS Register stores the starting address of the data segment

- **Stack Segment (SS)**
  - It contains data and return addresses of procedures or subroutines
  - A 16-bit Stack Segment Register or SS Register stores the starting address of the stack

- **Extra/ Extended Segment (ES)**
  - It is used to store variables (data), incase the data segment is not sufficient to store all variables
  - A 16-bit Extra Segment Register or ES Register stores the starting address of the extra segment

General Registers

General Registers

- The general registers are further divided into the following groups:
  1. Data Registers
  2. Pointer Registers
  3. Index Registers

1. Data Registers

- Four 32-bit data registers are used for arithmetic, logical and other operations in IA-32 architecture:

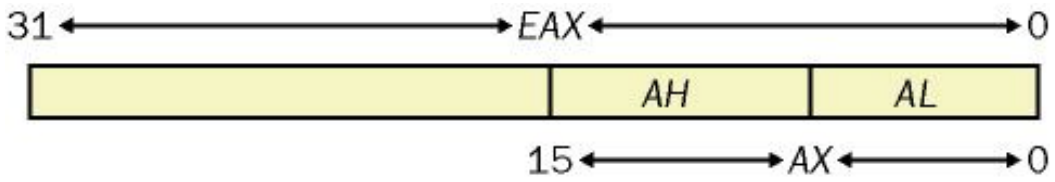
1. EAX- Extended Accumulator Register
2. EBX- Extended Base Register
3. ECX- Extended Count Register
4. EDX - Extended Data Register

- These 32-bit registers can be used in three ways:

1. As complete 32-bit data registers: **EAX, EBX, ECX** and **EDX**

2. Lower halves of the 32-bit registers can be used as four 16-bit data registers: **AX, BX, CX** and **DX**

3. Lower and higher halves of the above-mentioned four 16-bit registers can be used as eight 8-bit data registers: **AH, AL, BH, BL, CH, CL, DH**, and **DL**



Value	Bits
EAX	0–31
AX	0–15
AH	8–15
AL	0–7

32-Bit	16-Bit	8-Bit (High)	8-Bit (Low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

1. **EAX (Extended AX)**

   EAX (Extended Accumulator Register) is a 32-bit register in which intermediate arithmetic and logic results are stored

2. **EBX (Extended BX)**

   EBX (Extended Base Register) is a 32-bit register which can hold the address of a procedure or variable or can also be used as EAX helper for arithmetic and logic operations

3. **ECX (Extended CX)**

   ECX (Extended Counter Register) is a 32-bit register which acts as a counter for repeating or loop instructions or can also be used as EAX helper for arithmetic and logic operations

4. **EDX (Extended Data Register)**

   EDX (Extended Data Register) is a 32-bit register which can be used as a port number in I/O operations and it also can:

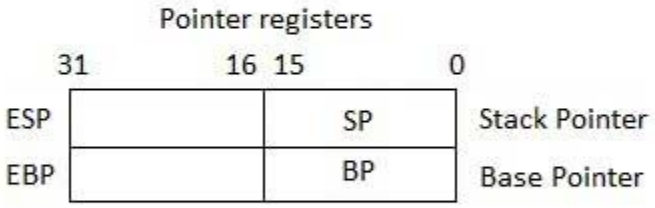
   - Holds the high 16 bits of the product in multiply
   - Holds the remainder of the divide operations

- It can also be used as EAX helper for arithmetic and logic operations

2. Pointer Registers

- Two 32-bit pointer registers used in IA-32 architecture are:

1. ESP - Extended Stack Pointer
2. EBP - Extended Base Pointer



- These 32-bit registers can be used in two ways:

1. As complete 32-bit pointer registers: **ESP** and **EBP**

2. Lower halves of the 32-bit registers can be used as two 16-bit data registers: **SP** and **BP**

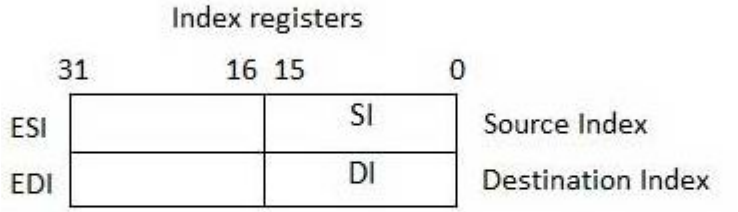
- **ESP (Extended Stack Pointer)**
  - The 32-bit ESP register is the stack pointer, used to manage push and pop operations
  - It points to the top item of the stack
  - It works with SS register (SS:SP)

- **EBP (Extended Base Pointer)**
  - The 32-bit EBP register is the base pointer, mainly used to access parameters passed to the procedures
  - It works with SS register (SS:BP)

3. Index Registers

- The 32-bit index registers, ESI and EDI, and their 16-bit rightmost portions, SI and DI, are used for indexed addressing and sometimes used in addition and subtraction

- There are two sets of index pointers
  1. ESI – Extended Source Index
  2. EDI – Extended Destination Index



- **Source Index (SI)**
  - It is used to store offset address of source operand
  - It works with DS (DS:SI)
- **Destination Index (DI)**
  - It is used to store offset address of destination operand
  - It works with ES (ES:DI)
- Both SI and DI registers are used in string movement operations

Example:

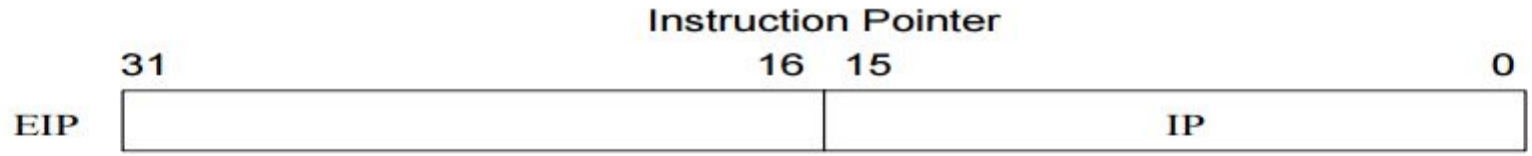
An instruction like movsb will copy a byte from the memory location pointed at/by SI, and move it to the memory location pointed at/by DI, and then increment both, so if you want to copy the byte stored at SI+1 to DI+1, it only takes a further movsb instruction

Control Registers

Control Registers

- The 16-bit and 32-bit Instruction Pointer Register (**IP)** and Flags Register (**FLAGS)** combined are considered as the control registers

- **Instruction Pointer (IP)**
  - It is used to store offset address of the next instruction to be executed
  - It works with CS (CS:IP)



- **Flags**

  Many instructions involve comparisons and mathematical calculations that changes the status of the flags, after which conditional instructions test the value of these status flags to take the control/ flow to other location

- Common flag bits are:
  - Sign Flag - SF
  - Overflow Flag - OF
  - Direction Flag - DF
  - Interrupt Flag - IF
  - Trap Flag - TF
  - Zero Flag - ZF
  - Auxiliary Flag - AF
  - Parity Flag - PF
  - Carry Flag - CF

NASM

- NASM stands for "Netwide Assembler"
- It is considered as one of the most popular assemblers for Linux
- It is an assembler for the Intel x86 architecture
- It can be used to write 16-bit, 32-bit (IA-32) and 64-bit (AMD-64) programs

NASM Setup

- Download DOSBox, NASM and AFD from the following link:

  http://theiteducation.com/wp-content/uploads/2020/05/Assably_Language_Setup.zip

- Extract the downloaded file, following 2 files will be extracted
  1. DOSBox0.74.exe
  2. AssmSoft.zip

- Install DOSBox. DOSBox is the x86 emulator with DOS or in simple words a virtual machine
- Create a folder (.e.g. COAL) in any local drive (.e.g. C) of your system and extract “AssmSoft” in it
- To program and work in this environment, create a new file with “.asm” extension in the same folder and open it in desired application (.e.g. Notepad)

- Write the following program in the created file:

```
[org 0x0100]
mov ax,2
mov bx,2
add ax, bx
mov ax,0x4c00  ; exit
int 0x21       ; OS should perform the above operation
```

- Save the file and open DOSBox

- Mount the required directory using the following command:

```
mount c c:/COAL
```

- Now this path becomes the drive of virtual machine

- To assemble, follow the given steps:

```
C:
nasm L3.asm –o L3.com
```

- To open Advance Free Debugger, use the following command:

```
afd L3.com
```

- To see addresses in memory using Window M1, use the following command:

  m1 0100

- To quit Advance Free Debugger, use the following command:

  quit

- To get listing file, use the following command:

  nasm L3.asm –l L3.lst –o L3.com

Listing File

```
1                       [org 0x0100]
2  00000000     B80200 mov ax,2
3  00000003     BB0200 mov bx,2
4  00000006     01D8      add ax, bx
5  00000008     B8004C mov ax,0x4c00
6  0000000B     CD21      int 0x21
```

- Using debugger, the address of all the segments registers can be changed/ updated to the desired address

- Syntax:

Segment Register = Desired Address

CS = 0700

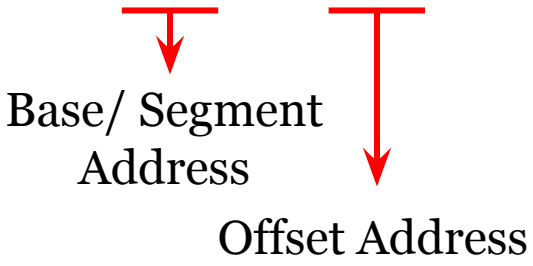
DS = 0800

SS = 0900

ES = 1000

Logical to Physical Address Conversion

- Logical Address or Virtual Address
  - An address generated by the CPU
- Physical Address
  - An address available on memory unit
- Consider **12F3:4B27** (Logical Address)



- Put ‘0’ at the end of Base/ Segment Address and add it with Offset Address

$$
\begin{array}{r}
{\scriptstyle 1} \\
12F30 \\
+\ 04B27 \\
\hline
17A57
\end{array}
$$

- **12F3:4B27** corresponds to the physical address **0x17A57**

Task

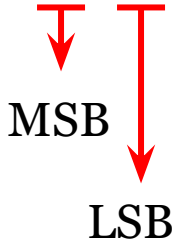
- Calculate physical addresses for the given logical addresses:
  - **020A:1BCD**
  - **001A:0021**
  - **0FE1:2F22**

Endianness

- Computers store data in memory in binary
- The order in which bytes are stored is referred to as endianness. Endianness is of two types:
- Little Endian
  - When the least significant bytes are stored prior to most significant bytes
- Big Endian
  - When the most significant bytes are stored prior to least significant bytes

Example

- For “mov ax, 2” the machine code is “B80200”
- “B8” is the machine code for “mov ax”
- Data stored in ax is “0002” but in memory it is stored as “0200”



- Intel architecture stores LSB to the lower address and MSB to higher address, hence following “Little Endian”