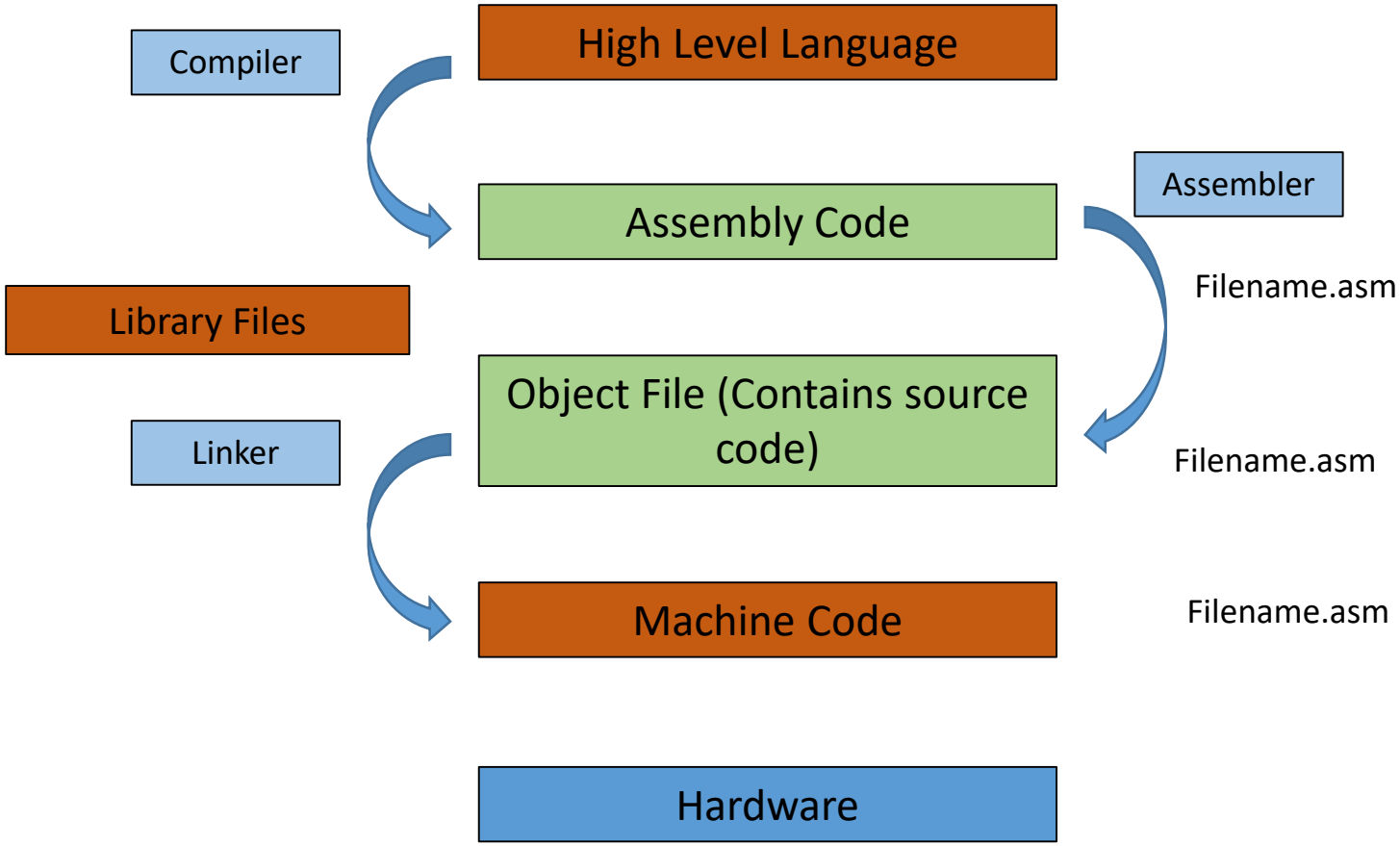
Computer Organization & Assembly Language

Lab-1

Convert Assembly Code to executable code



Why we learn Assembly Language?

- Better/deep understanding of software and hardware interaction.
- Optimization of processing time
- Embedded programming

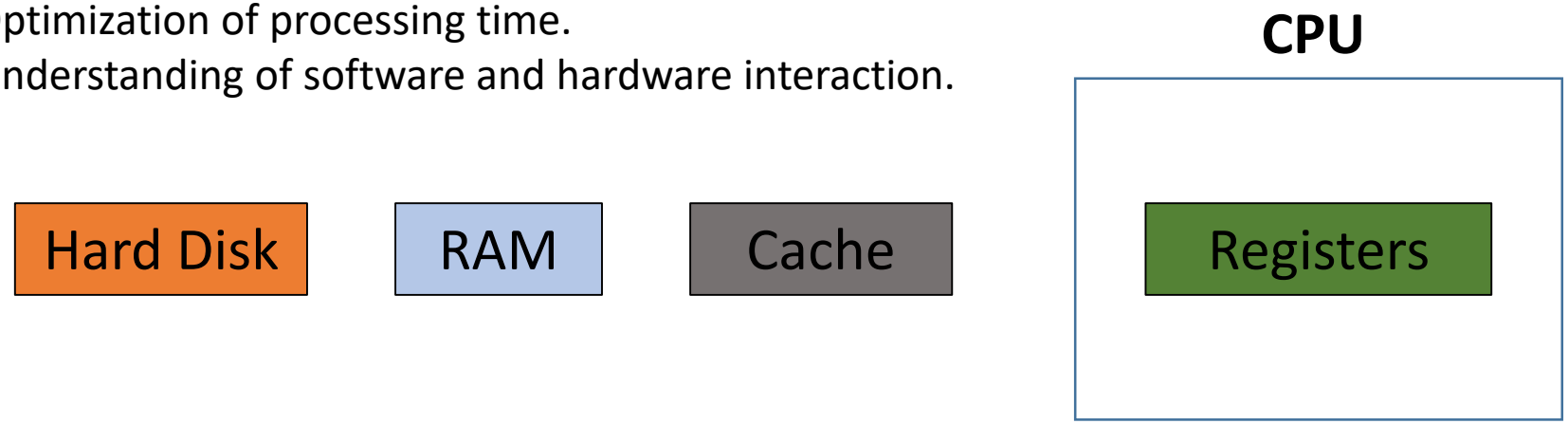
What is Assembly Language?

- Computer programming language which is used to make programs.
  - Program is a set of instructions.
- It’s a Low level language (means close to Machine)
- Use Mnemonics/Keywords (such as ADD, SUB or MOV etc.)
- Designed by David John Wheeler in 1940. He was a computer scientist and a professor at the University of Cambridge.

Registers:

Why we use Registers?

- Optimization of processing time.
- Understanding of software and hardware interaction.



What is Registers?

- Fastest Storage area/location (Quickly accessible by CPU as they are built into CPU".
- Origin in 1971 in Inter 4004 CPU

Type of Registers:

There are 14 types of Registers.

1. **Accumulator.** ← Input/Output, Operations, a, ax, eax, rax
2. **Base.** ← Holds address of data, b, bx, ebx, rbx
3. **Counter.** ← Count, used in loop, cx, ecx, rcx
4. **Data.** ← Holds data for output, d, dx, edx, rdx
5. Code Segment. ← Holds address of code segment
6. Data Segment. ← Holds address of data segment
7. Stack Segment. ← Holds address of stack segment
8. Extra Segment. ← Holds address of data segment
9. Source Index. ← Points the source operand
10. Destination Index. ← Points the destination operand
11. **Instruction Pointer.** ← Holds the next instruction
12. **Stack Pointer.** ← Point current top of stack
13. **Flag Register.** ← Holds current status of the program
14. Base Pointer. ← Base of the top of stack

General Purpose Registers

Type of Registers:

General Purpose Registers.

1. **Accumulator.** ← Input/Output, Operations, a, ax, eax, rax
2. **Base.** ← Holds address of data, b, bx, ebx, rbx
3. **Counter.** ← Count, used in loop, cx, ecx, rcx
4. **Data.** ← Holds data for output, d, dx, edx, rdx

} General Purpose Registers

The four general purpose registers are the AX, BX, CX, and DX registers.

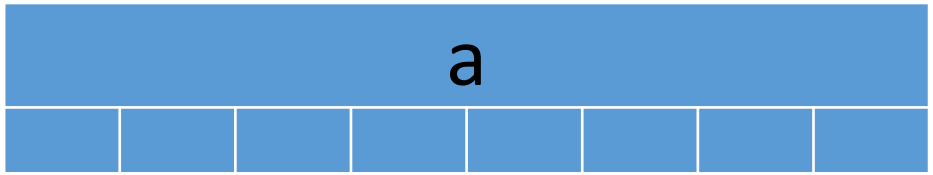
AX - accumulator, and preferred for most operations.
BX - base register, typically used to hold the address of a procedure or variable.
CX - count register, typically used for looping.
DX - data register, typically used for multiplication and division.

All of the general purpose registers can be treated as a 32 bit quantity or as two 16 bits or as four of 8 bits quantities.

Type of Registers:

1. **Accumulator**.

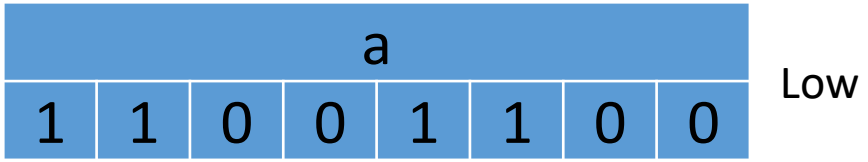
Input/Output, Operations, a, ax, eax, rax



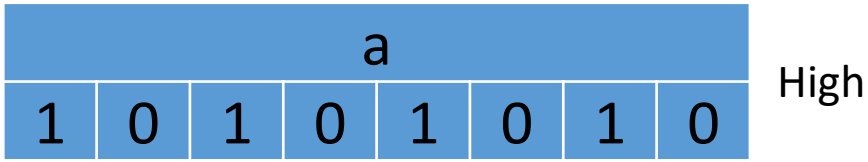
a for accumulator (**8 bits**)

If signal of **16** bits receives,
10101010 11001100 it will be divided in to two parts (low and high)

High	**Low**
10101010	11001100



Low

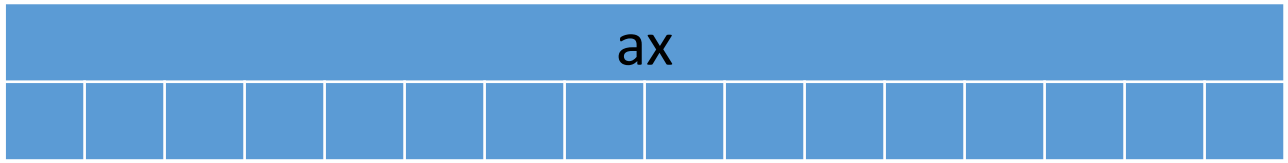


High

Type of Registers:

1. **Accumulator.**

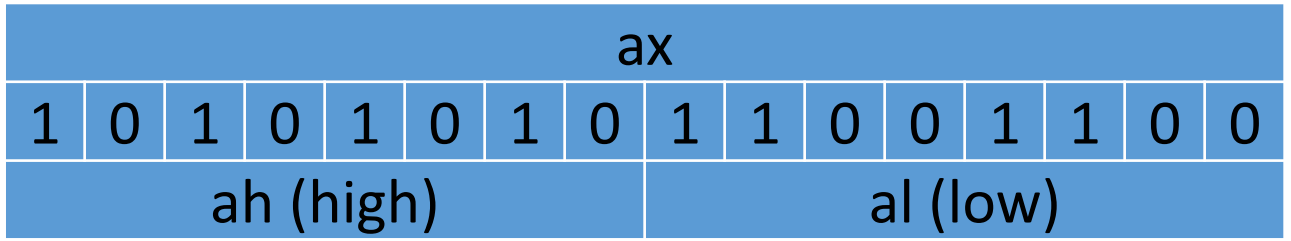
Input/Output, Operations, a, ax, eax, rax



ax for accumulator extended (**16 bits**)

If signal of **16** bits receives,
10101010 11001100 it will be directly go into accumulator at once.

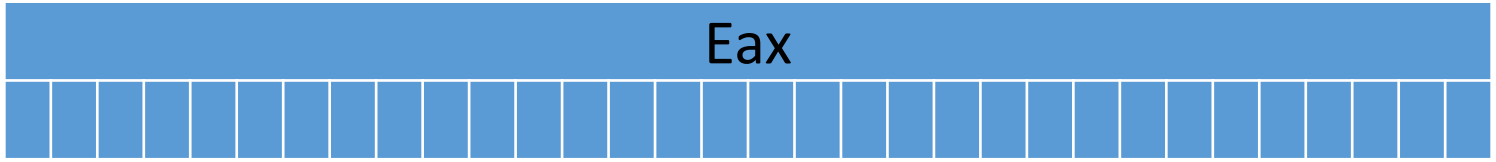
High	**Low**
10101010	11001100



Type of Registers:

1. **Accumulator**.

Input/Output, Operations, a, ax, eax, rax



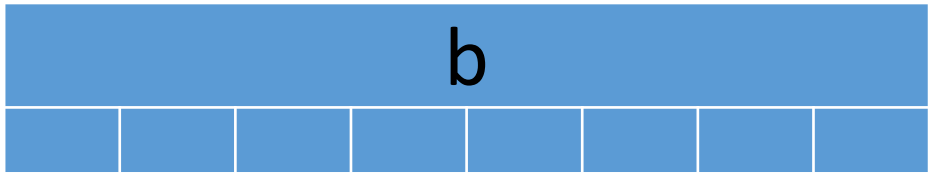
Eax for accumulator extended (**32 bits**)

Rax for Rich accumulator extended (**64 bits**)

X- Extended to **16 bits**
E- Extended to **32 bits**
R- Rich Register for **64 bits**

Type of Registers:

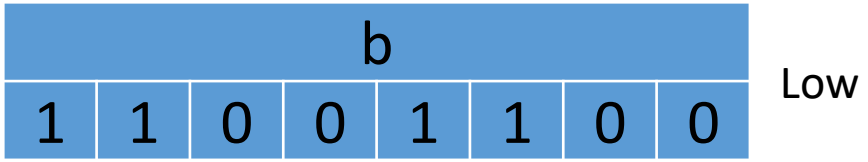
2. Base.
Holds address of data available in RAM, b, bx, ebx, rbx



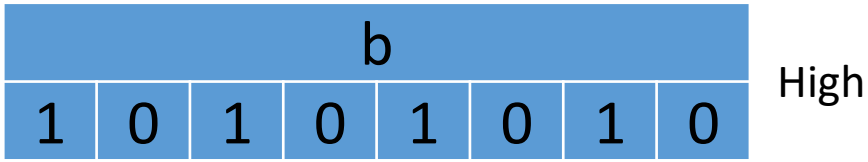
a for accumulator (**8 bits**)

If signal of **16** bits receives,
10101010 11001100 it will be divided in to two parts (low and high)

High	**Low**
10101010	11001100



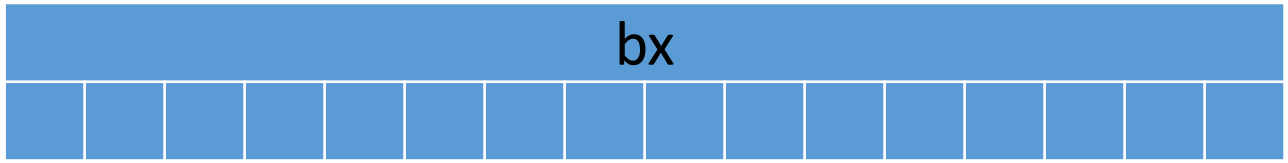
Low



High

Type of Registers:

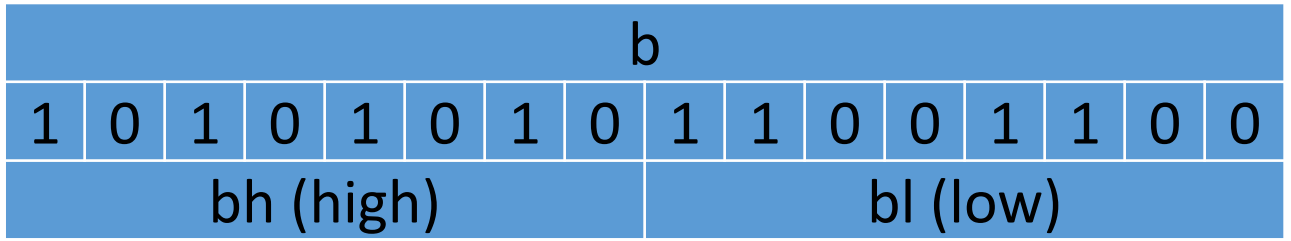
2. Base.
Holds address of data available in RAM, b, bx, ebx, rbx



ax for accumulator extended (**16 bits**)

If signal of **16** bits receives,
10101010 11001100 it will be directly go into accumulator at once.

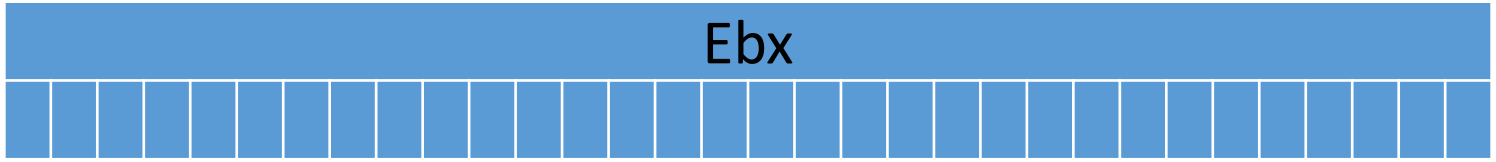
High	**Low**
10101010	11001100



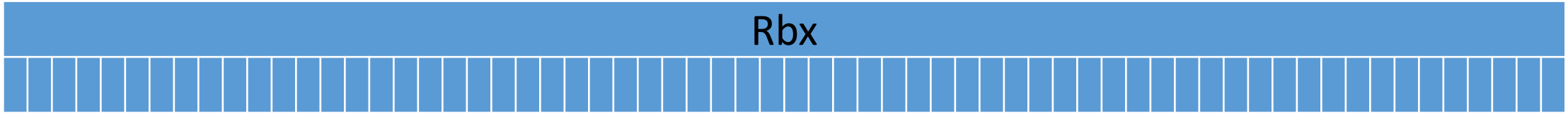
Type of Registers:

2. Base.

Holds address of data available in RAM, b, bx, ebx, rbx



Ebx for accumulator extended (**32 bits**)



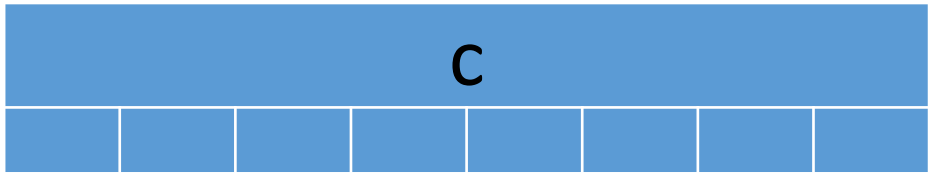
Rbx for Rich accumulator extended (**64 bits**)

X- Extended to **16 bits**
E- Extended to **32 bits**
R- Rich Register for **64 bits**

Type of Registers:

3. Counter.

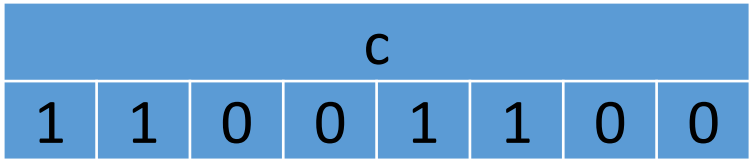
Count, used in loop, cx, ecx, rcx



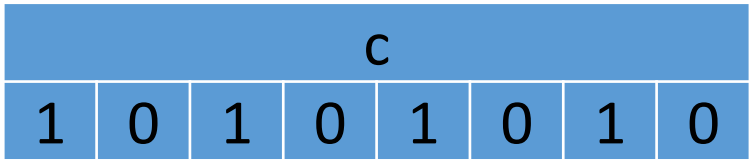
a for accumulator (**8 bits**)

If signal of **16** bits receives,
10101010 11001100 it will be divided in to two parts (low and high)

High	**Low**
10101010	11001100



Low

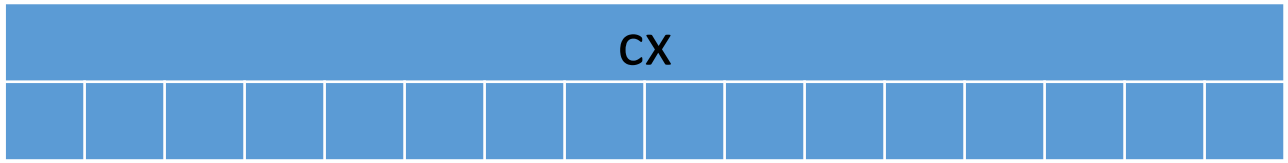


High

Type of Registers:

3. Counter.

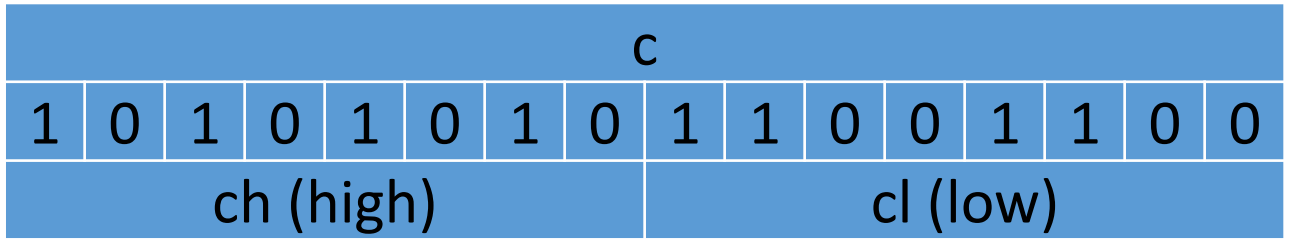
Count, used in loop, cx, ecx, rcx



ax for accumulator extended (**16 bits**)

If signal of **16** bits receives,
10101010 11001100 it will be directly go into accumulator at once.

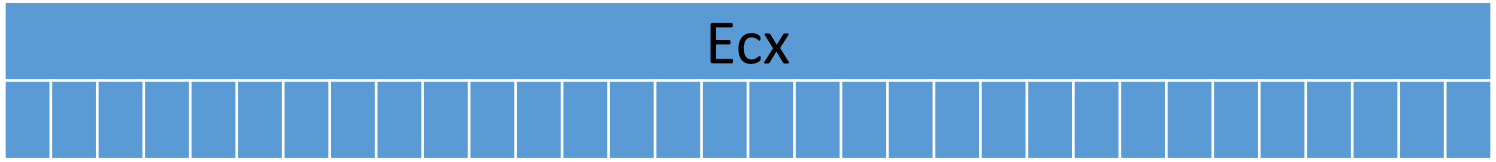
High	**Low**
10101010	11001100



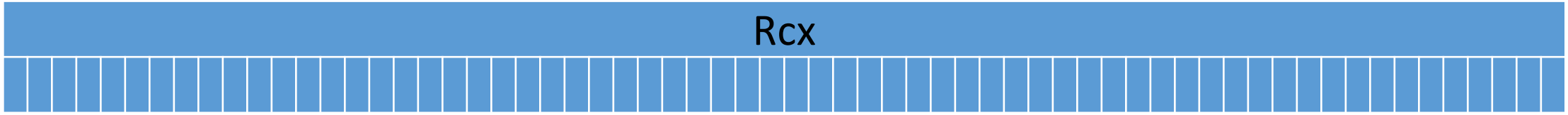
Type of Registers:

3. Counter.

Count, used in loop, cx, ecx, rcx



Ecx for accumulator extended (**32 bits**)



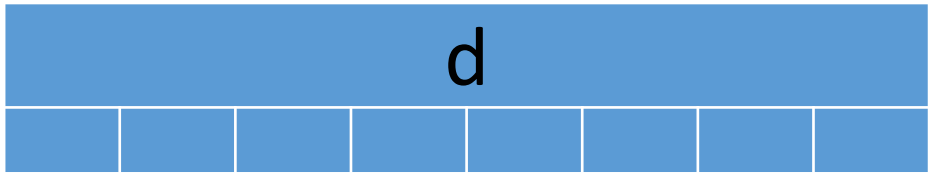
Rcx for Rich accumulator extended (**64 bits**)

X- Extended to **16 bits**
E- Extended to **32 bits**
R- Rich Register for **64 bits**

Type of Registers:

4. **Data.** (most important register)

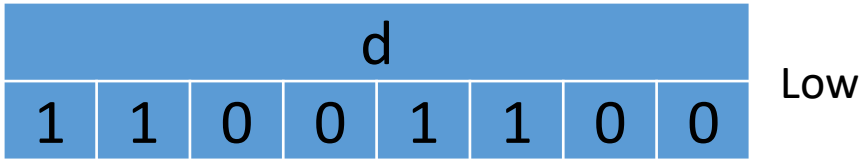
Holds data for output, d, dx, edx, rdx



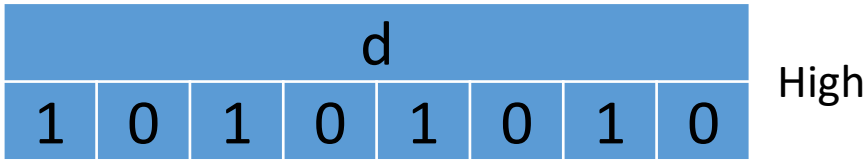
a for accumulator (**8 bits**)

If signal of **16** bits receives,
10101010 11001100 it will be divided in to two parts (low and high)

High	**Low**
10101010	11001100



Low

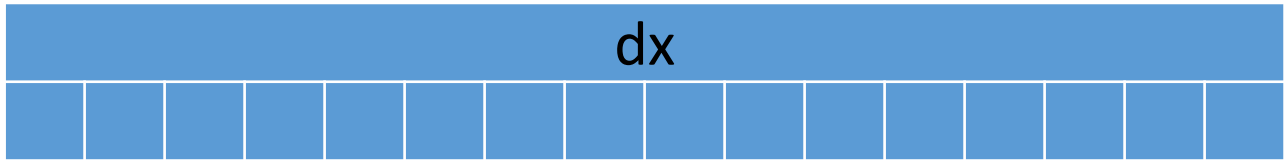


High

Type of Registers:

4. **Data.** (most important register)

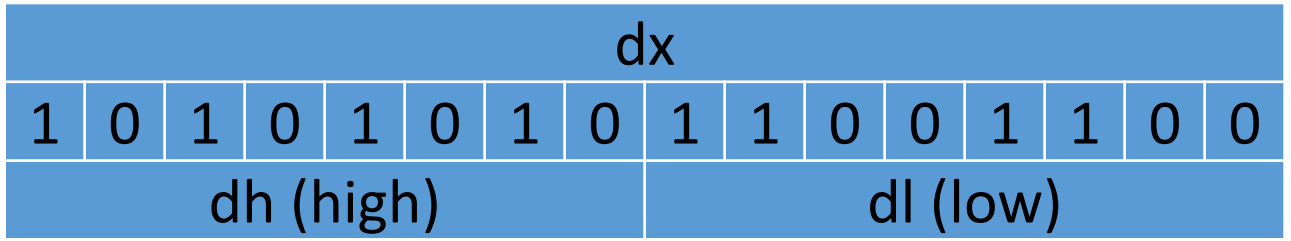
Holds data for output, d, dx, edx, rdx



ax for accumulator extended (**16 bits**)

If signal of **16** bits receives,
10101010 11001100 it will be directly go into accumulator at once.

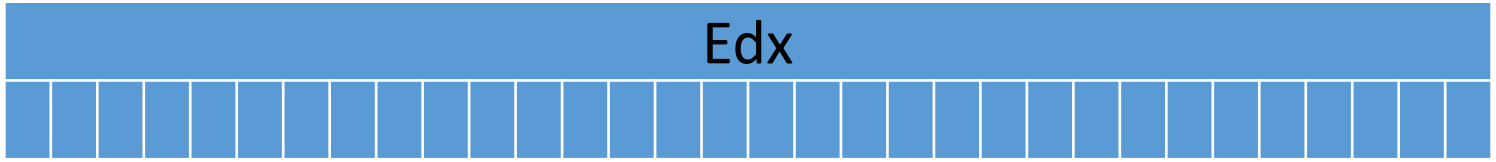
High	**Low**
10101010	11001100



Type of Registers:

4. **Data.** (most important register)

Holds data for output, d, dx, edx, rdx



Edx for accumulator extended (**32 bits**)

Rdx for Rich accumulator extended (**64 bits**)

X- Extended to **16 bits**
E- Extended to **32 bits**
R- Rich Register for **64 bits**

Type of Registers:

Segment Registers.

5. Code Segment. ← Holds address of code segment
6. Data Segment. ← Holds address of data segment
7. Stack Segment. ← Holds address of stack segment
8. Extra Segment. ← Holds address of data segment

} Segment Registers

The CPU contains four segment registers, used as base locations for program instructions, data, or the stack. In fact, all references to memory on the IBM PC involve a segment register as a base location.

The registers are:

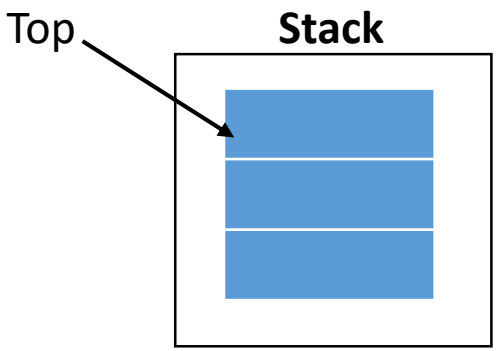
CS – Code Segment, base location of program code
DS – Data Segment, base location for variables
SS – Stack Segment. Base location of the stack
ES – Extra Segment. Additional base location for variables in memory.

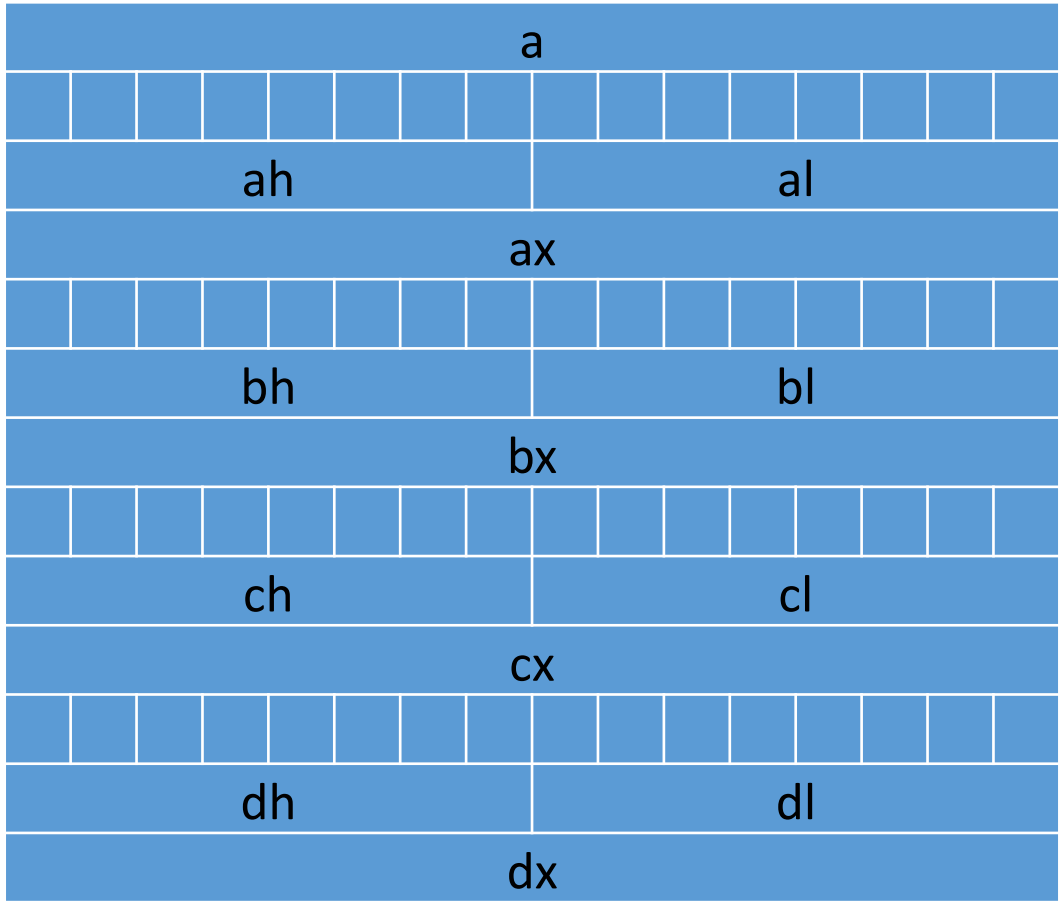
All of the segment registers can not divided.

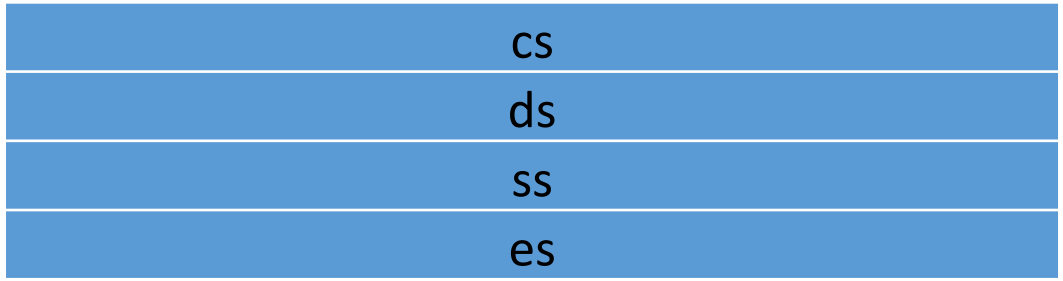
Type of Registers:

Segment Registers.

5. Code Segment. ← Holds address of code segment
6. Data Segment. ← Holds address of data segment
7. Stack Segment. ← Holds address of stack segment
8. Extra Segment. ← Holds address of data segment







Type of Registers:

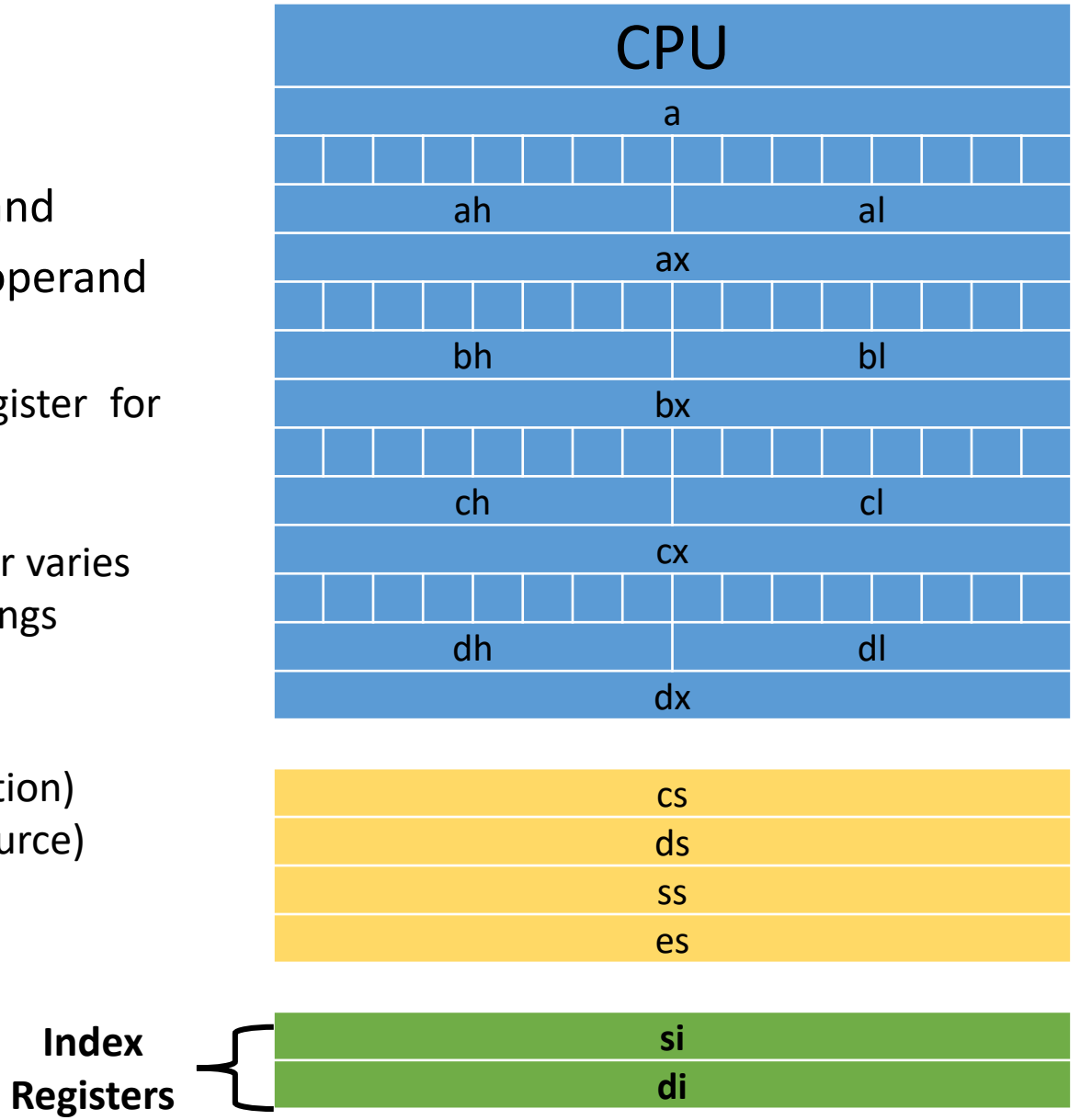
Index Registers.

9. Source Index. ← Points the source operand
10. Destination Index. ← Points the destination operand

The index registers contain offsets from a segment register for information we are interested about

SI – Source Index, used for copying strings, segment register varies
DI – Destination Index, used for destination for copying strings

```
Add     dl, bl     (dl is destination and bl is source)
Add     3, bl      (3 (constant) is source and bl is destination)
Add     dl, 3       (dl is destination and 3 (constant) is source)
```

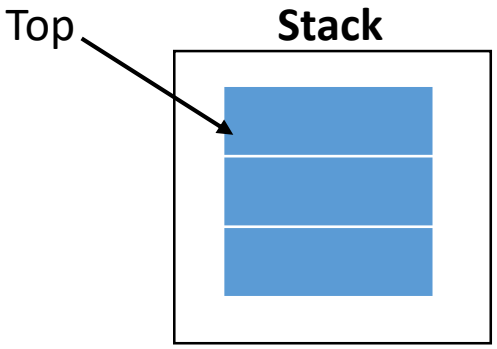


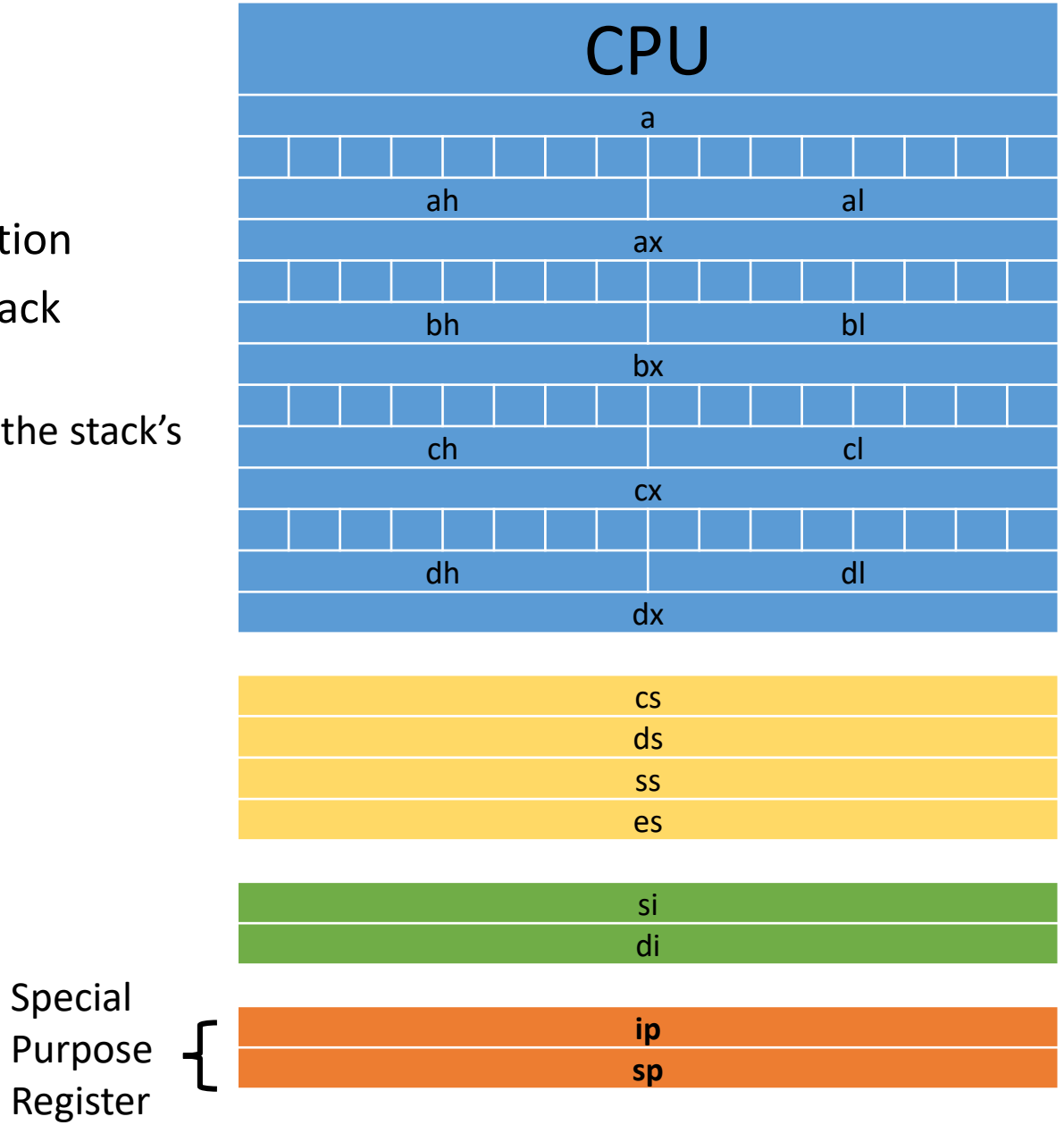
Type of Registers:

Special Purpose Registers.

11. **Instruction Pointer.** ← Holds the next instruction
12. **Stack Pointer.** ← Point current top of stack

Stack Pointer, offset from SS register as to the location of the stack's top

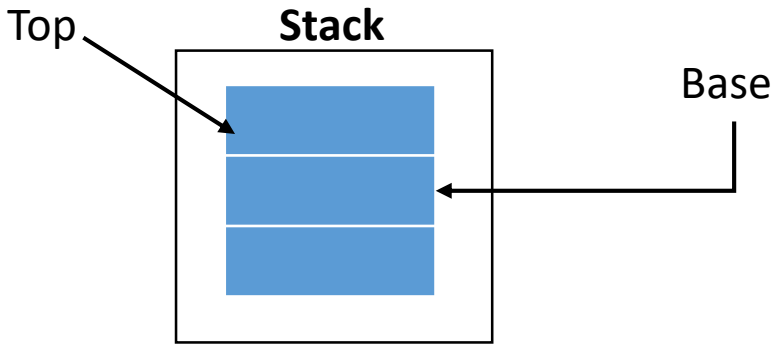


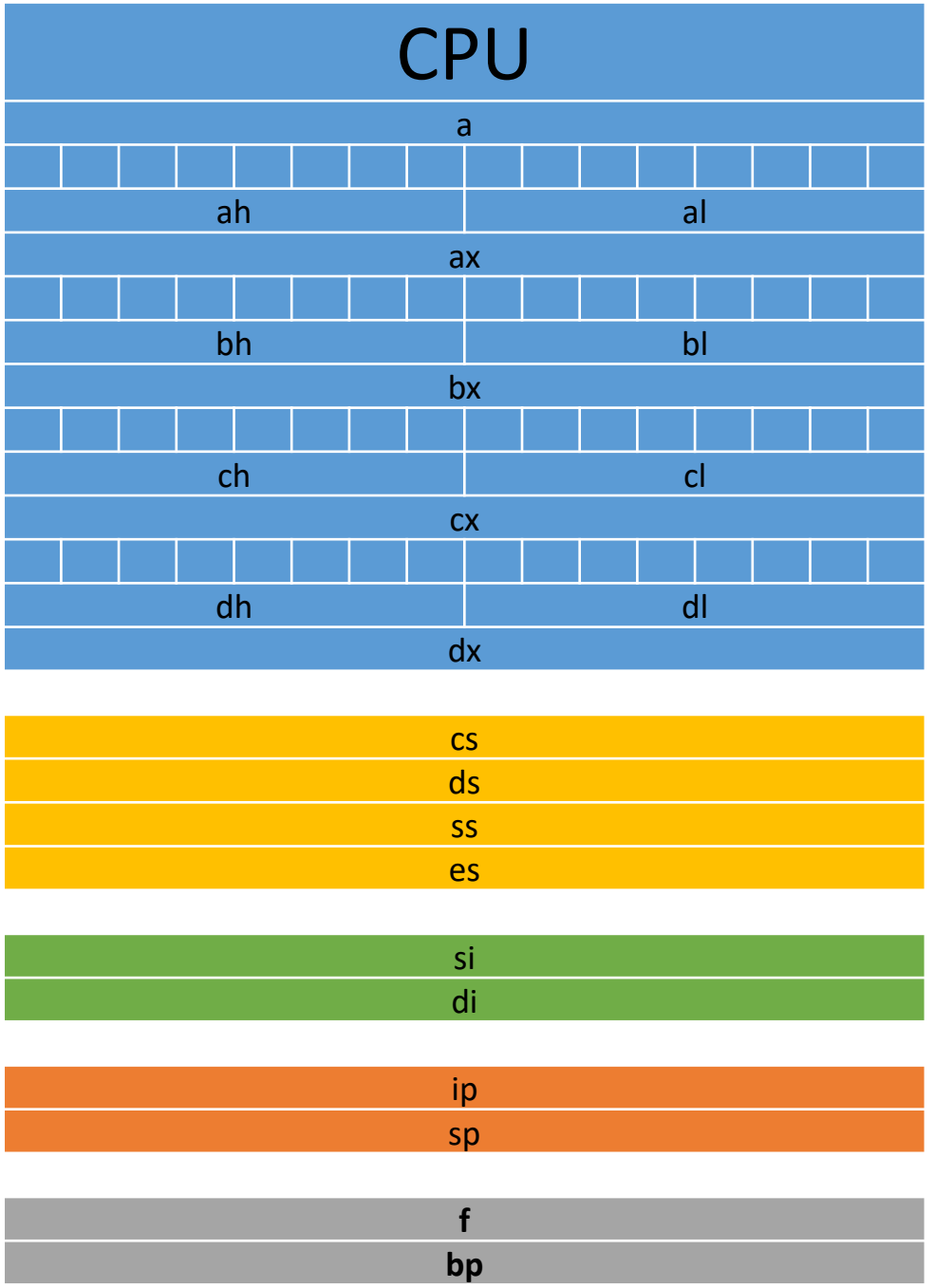


Type of Registers:

13. **Flag Register.** ← Holds current status of the program
14. Base Pointer. ← Base of the top of stack

Base Pointer, offset from SS register to locate variables on the stack





Type of Registers:

There are 14 types of Registers.

1. **Accumulator.** ← Input/Output, Operations, a, ax, eax, rax
2. **Base.** ← Holds address of data, b, bx, ebx, rbx
3. **Counter.** ← Count, used in loop, cx, ecx, rcx
4. **Data.** ← Holds data for output, d, dx, edx, rdx
5. Code Segment. ← Holds address of code segment
6. Data Segment. ← Holds address of data segment
7. Stack Segment. ← Holds address of stack segment
8. Extra Segment. ← Holds address of data segment
9. Source Index. ← Points the source operand
10. Destination Index. ← Points the destination operand
11. **Instruction Pointer.** ← Holds the next instruction
12. **Stack Pointer.** ← Point current top of stack
13. **Flag Register.** ← Holds current status of the program
14. Base Pointer. ← Base of the top of stack

General Purpose Registers

Segment Registers

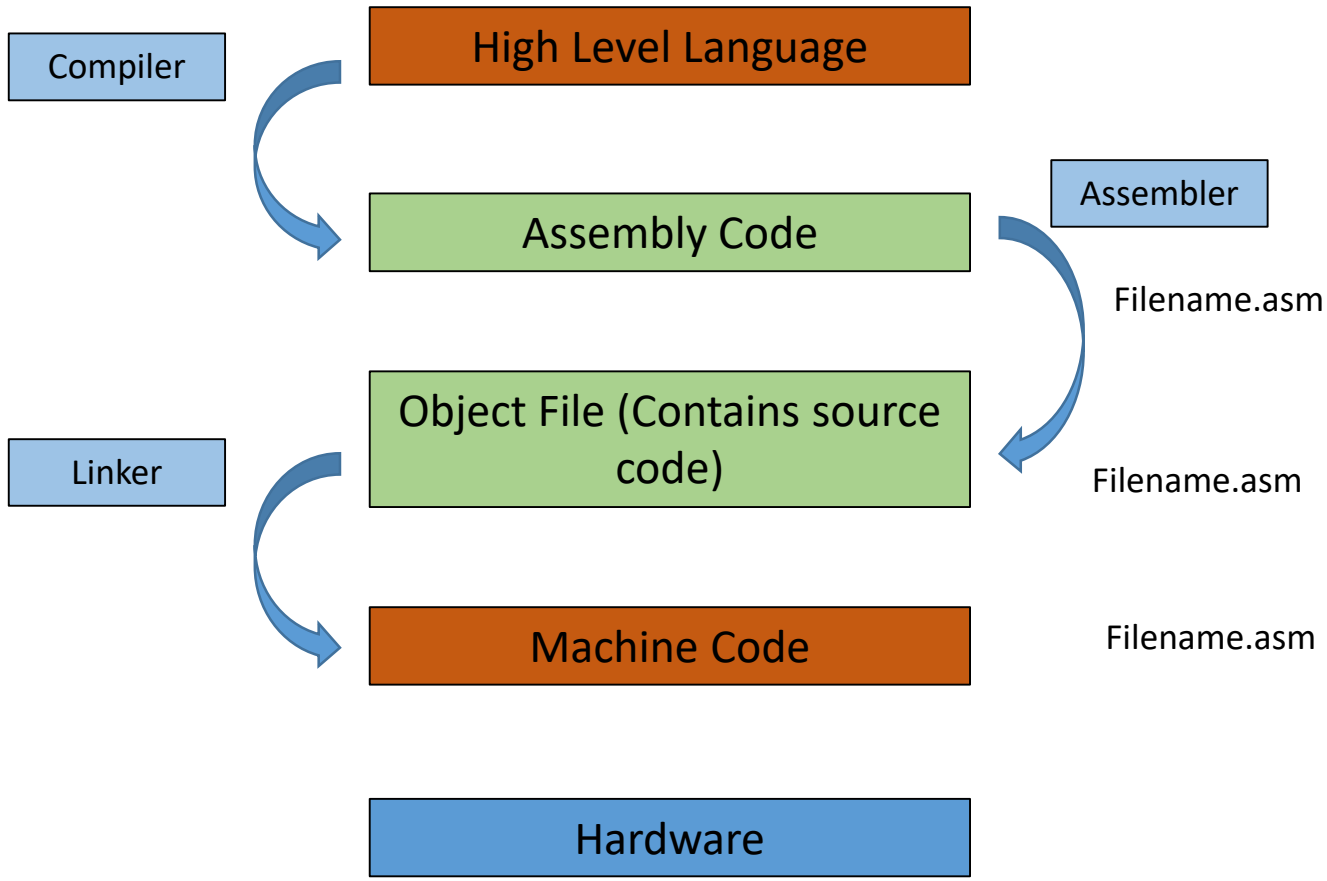
Index Registers

Special Purpose Registers

DosBox, MASM, Link and DosBox Commands

- What is DosBox?
  - DosBox is an Emulator, designed in July 2002 by Peter Veenstra.
- Why DosBox?
  - Freeware
  - Light, simple and easy to use.
  - Run on everyone environment such as Windows, MAC, Linux and Android.
  - Help of learn debugging.
  - Help to grip on Syntax.

Convert Assembly Code to executable code



DosBox Commands

- Edit Filename.asm (to create new file if not exists/open existing file)
- MASM Filename.asm; (to convert into object file using MASM assembler)
- LINK Filename.obj; (to convert object file into execution file using linker)
- To execute the exe file you just created,
  - Filename.exe (it will execute)

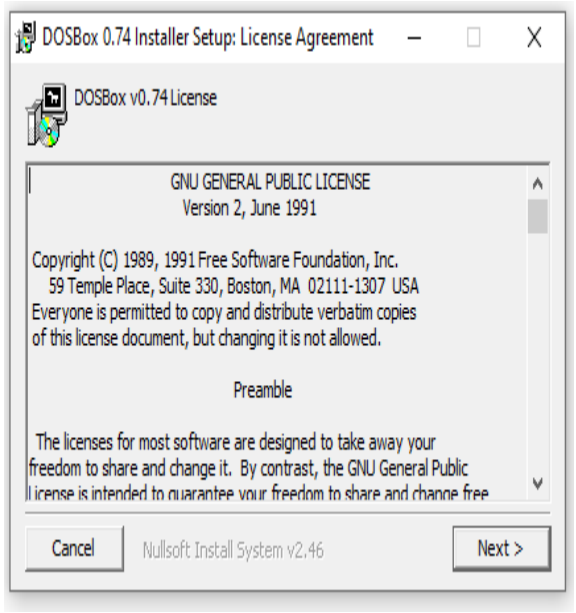
- NOTE: (Semicolon is mandatory while converting via assembler and linker only)

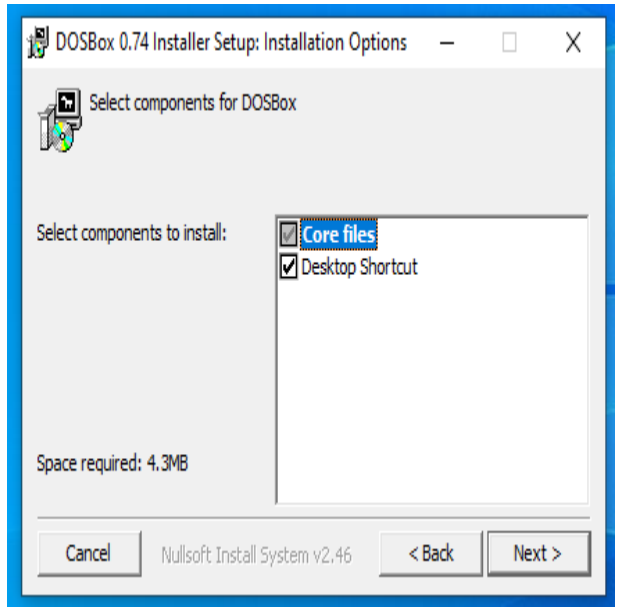
DosBox Commands

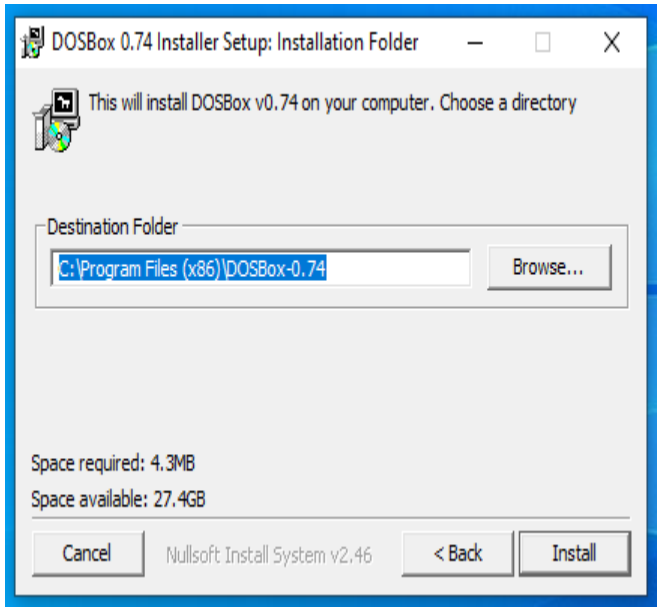
- Copy the DosBox With MP folder from

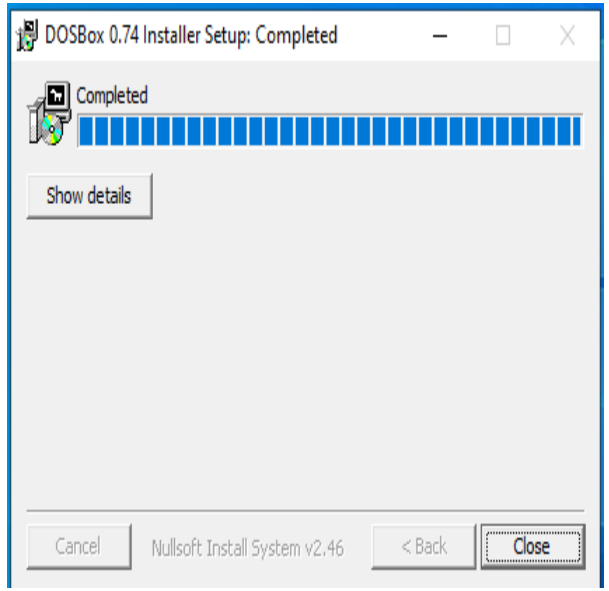
\\192.168.100.200\Teacher's Data\Qazi Shahab Azam\Spring 2022\CAO-Lab

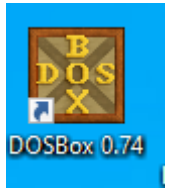
Onto your PC and Run the setup file viz “DOSBox0.74-win32-installer”





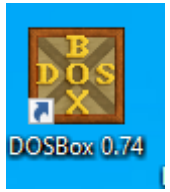




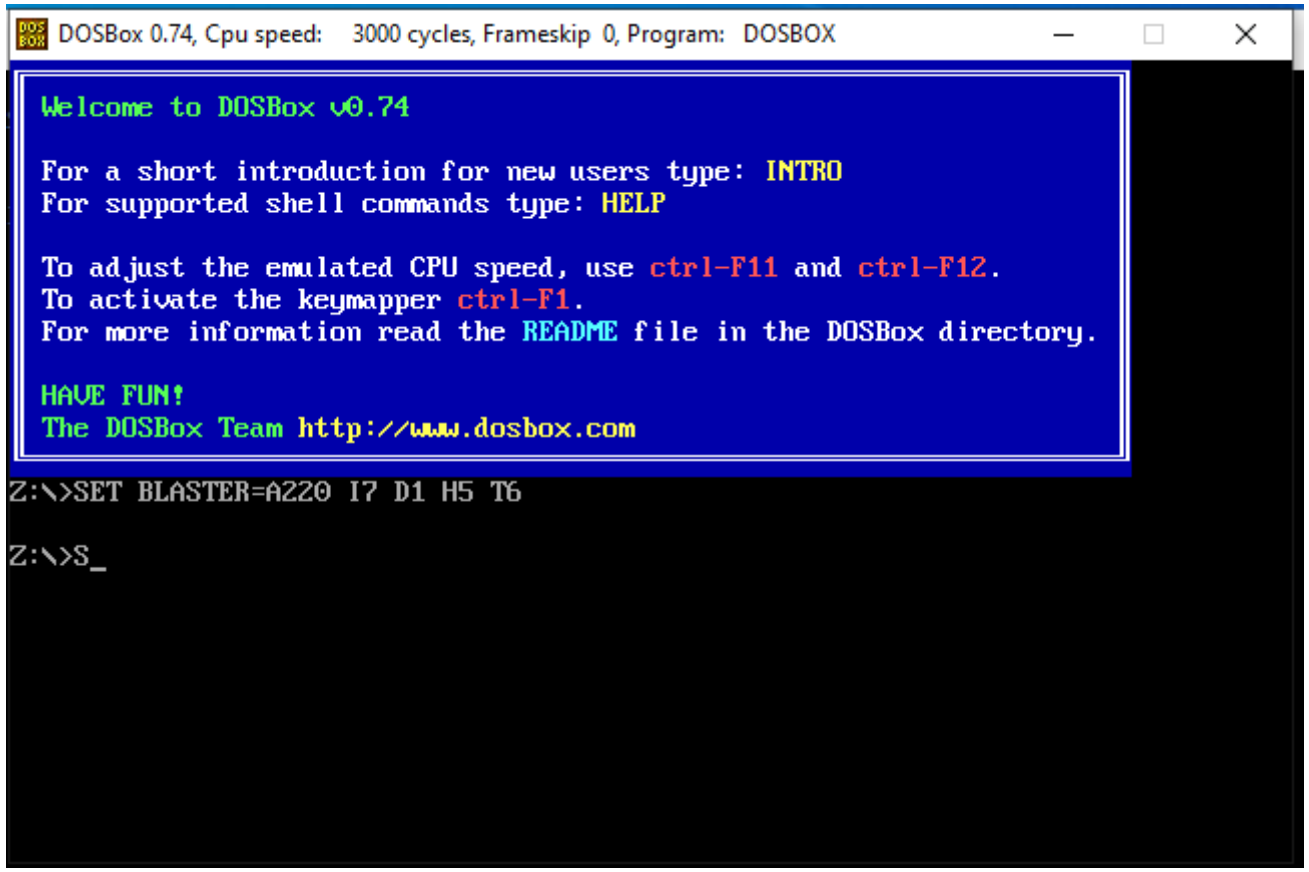


After Installation, this icon will appear on your desktop.

DosBox Commands

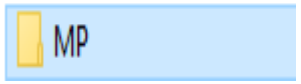


Double click on this Icon.

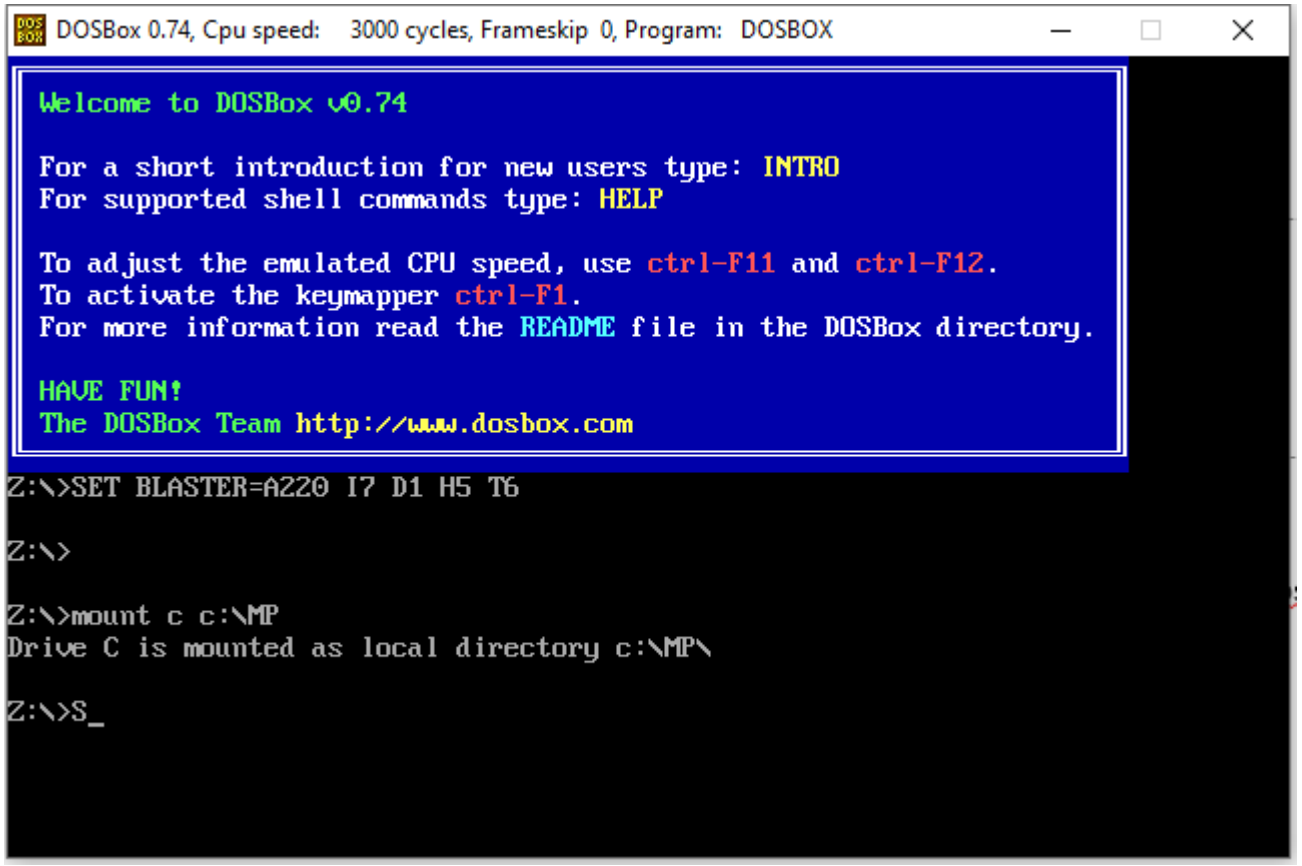


The virtual dosbox will appear on Z drive.

DosBox Commands



Copy this folder simply on C drive.



After copying the MP folder at C drive,
Type the command
mount c c:\MP on Z: drive
This command will mount the C drive for use