Types of computer languages

5.1 Introduction

5.2 The Computer Organization - Intel PC

5.3 Instruction Format

5.4 Addressing Mode

5.5 DEBUG program

Introduction

Levels of Programming Languages

1) Machine Language
   - Consists of individual **instructions** that will be executed by the CPU one at a time

2) Assembly Language (Low Level Language)
   - Designed for a specific family of processors (different processor groups/family has different Assembly Language)
   - Consists of symbolic instructions directly related to machine language instructions one-for-one and are assembled into machine language.

3) High Level Languages
   - e.g. : C, C++ and Vbasic
   - Designed to eliminate the technicalities of a particular computer.
   - Statements compiled in a high level language typically generate many low-level instructions.

Advantages of Assembly Language

1. Shows how program interfaces with the processor, operating system, and BIOS.
2. Shows how data is represented and stored in memory and on external devices.
3. Clarifies how processor accesses and executes instructions and how instructions access and process data.
4. Clarifies how a program accesses external devices.

Reasons for using Assembly Language

1. A program written in Assembly Language requires considerably less memory and execution time than one written in a high –level language.
2. Assembly Language gives a programmer the ability to perform highly technical tasks that would be difficult, if not impossible in a high-level language.
3. Although most software specialists develop new applications in high-level languages, which are easier to write and maintain, a common practice is to recode in assembly language those sections that are time-critical.
4. Resident programs (that reside in memory while other program execute) and interrupt service routines (that handle input and output) are almost always develop in Assembly Language.

The Computer Organization - INTEL PC

In this course, only INTEL assembly language will be learnt. Below is a brief description of the development of a few INTEL model.

(i) 8088

- Has 16-bit registers and 8-bit data bus
- Able to address up to 1 MB of internal memory
- Although registers can store up to 16-bits at a time but the data bus is only able to transfer 8 bit data at one time

(ii) 8086

- Is similar to 8088 but has a 16-bit data bus and runs faster.

(iii) 80286

- Runs faster than 8086 and 8088
- Can address up to 16 MB of internal memory
- *multitasking* => more than 1 task can be ran simultaneously

(iv) 80386

- has 32-bit registers and 32-bit data bus
- can address up to 4 billion bytes. of memory
- support "*virtual mode*", whereby it can swap portions of memory onto disk: in this way, programs running concurrently have space to operate.

(v) 80486

- has 32-bit registers and 32-bit data bus
- the presence of CACHE

(vi) Pentium

- has 32-bit registers, 64-bit data bus
- has separate caches for data and instruction
- the processor can decode and execute more than one
- instruction in one clock cycle (pipeline)

(vii) Pentium II & III

- has different paths to the cache and main memory

In performing its task, the processor (CPU) is partitioned into two logical units:

1) An Execution Unit (EU)
2) A Bus Interface Unit (BIU)

EU

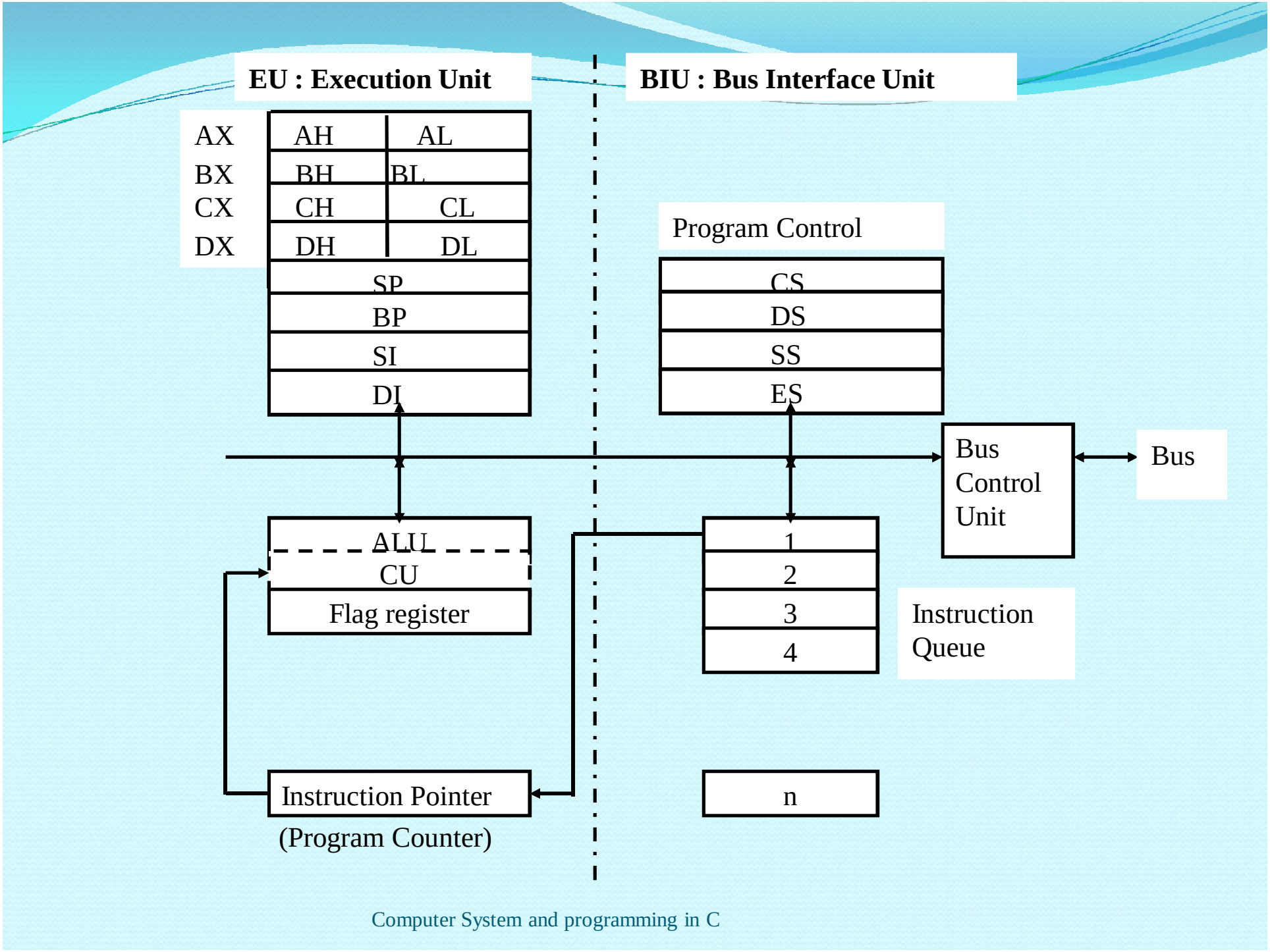
- EU is responsible for **program execution**
- Contains of an Arithmetic Logic Unit (ALU), a Control Unit (CU) and a number of registers

BIU

- **Delivers data and instructions** to the EU.
- manage the bus control unit, segment registers and instruction queue.
- The BIU controls the buses that transfer the data to the EU, to memory and to external input/output devices, whereas the segment registers control memory addressing.

EU and BIU work in parallel, with the BIU keeping one step ahead. The EU will notify the BIU when it needs to data in memory or an I/O device or obtain instruction from the BIU instruction queue.

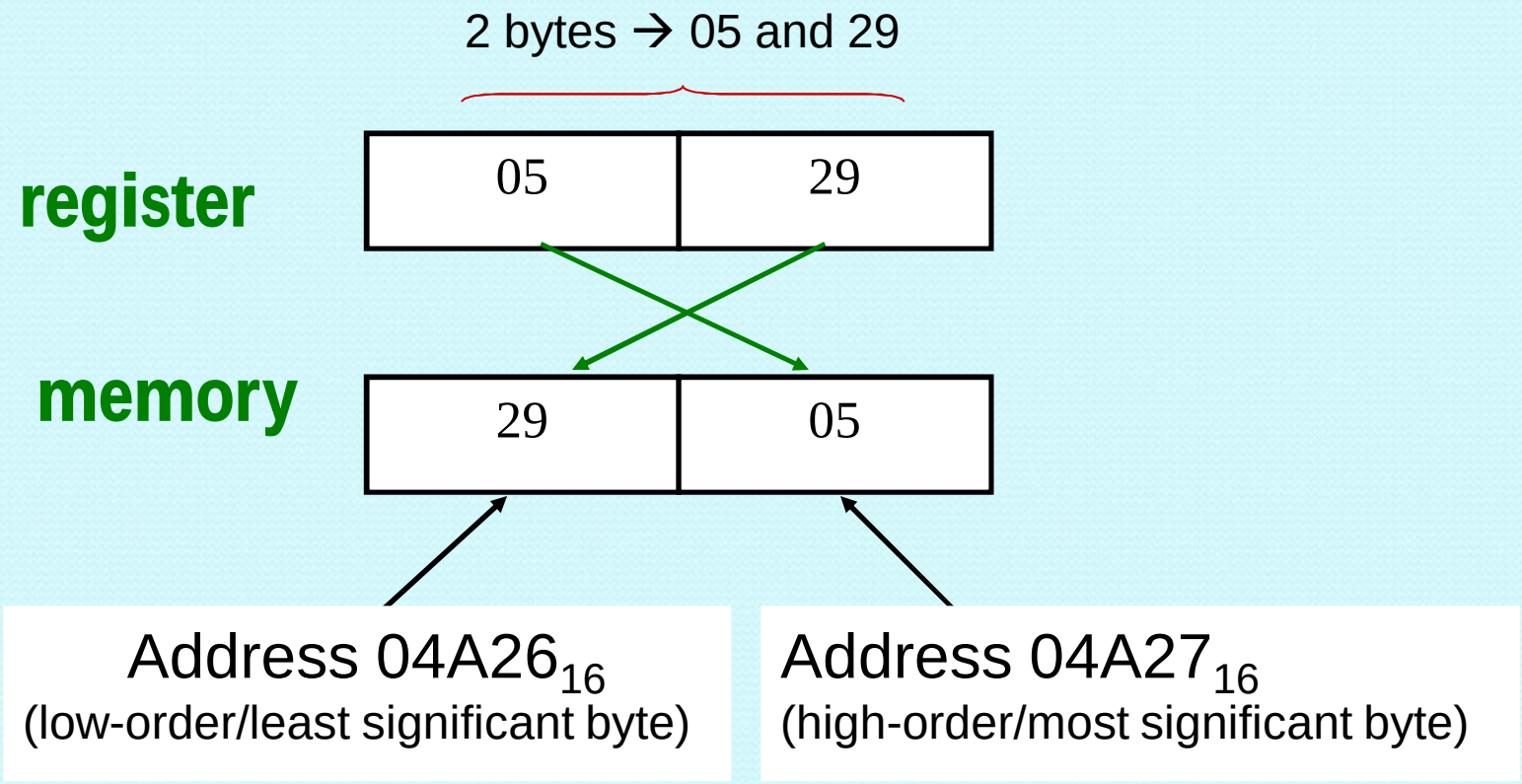
When EU executes an instruction, BIU will fetch the next instruction from the memory and insert it into to instruction queue.


EU : Execution Unit
BIU : Bus Interface Unit
AX
AH
AL
BX
BH
BL
CX
CH
CL
DX
DH
DL
SP
BP
SI
DI
Program Control
CS
DS
SS
ES
Bus Control Unit
Bus
ALU
CU
Flag register
1
2
3
4
Instruction Queue
Instruction Pointer
(Program Counter)
n

Addressing Data in Memory

- Intel Personal Computer (PC) addresses its memory according to bytes. (Every byte has a unique address beginning with 0)
- Depending to the model of a PC, CPU can access 1 or more bytes at a time
- Processor (CPU) keeps data in memory in reverse byte sequence (*reverse-byte sequence: low order byte in the low memory address and high-order byte in the high memory address*)

Example : consider value 0529_{16} (0529H)



- When the processor takes data (a word or 2 bytes), it will re-reverse the byte to its actual order 0529_{16}

Segment And Addressing

- Segments are special areas in the memory that is defined in a program, containing the code, data, and stack.
- The segment position in the memory is not fixed and can be determined by the programmer
- 3 main segments for the programming process:

(i) Code Segment (CS)

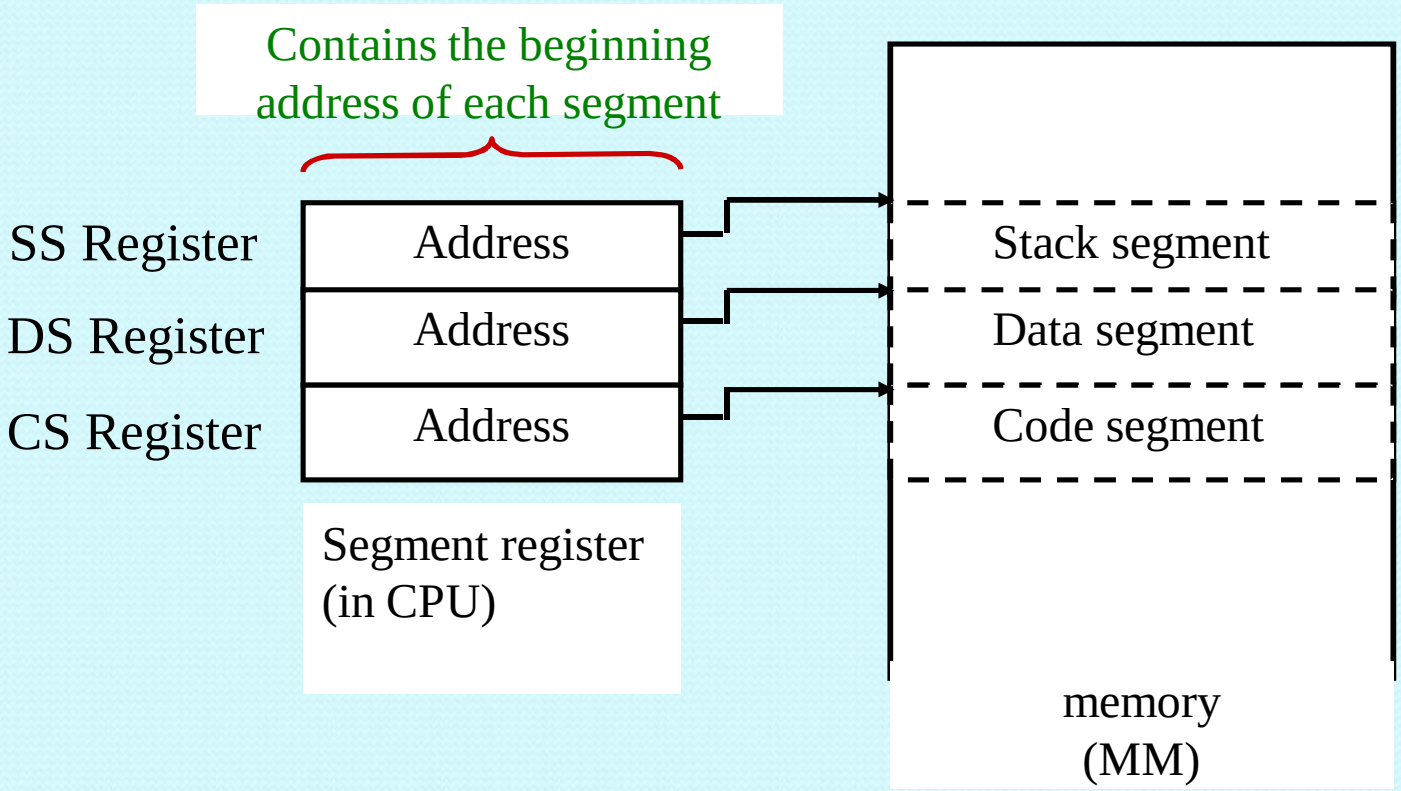
- Contains the machine instructions that are to execute.
- Typically, the first executable instruction is at the start of this segment, and the operating system links to that location to begin program execution.
- CS register will hold the beginning address of this segment

(ii) Data Segment (DS)

- Contains program's defined data, constants and works areas.
- DS register is used to store the starting address of the DS

(iii) Stack Segmen (SS)

- Contains any data or address that the program needs to save temporarily or for used by your own "called"subroutines.
- SS register is used to hold the starting address of this segment


Contains the beginning address of each segment
SS Register
DS Register
CS Register
Address
Address
Address
Segment register (in CPU)
Stack segment
Data segment
Code segment
memory (MM)

Segment Offsets

- Within a program, all memory locations within a segment are relative to the segment's starting address.
- The distance in bytes from the segment address to another location within the segment is expressed as an offset (or displacement).
- Thus the first byte of the code segment is at offset 00, the second byte is at offset 01 and so forth.
- To reference any memory location in a segment (the actual address), the processor combines the segment address in a segment register with the offset value of that location. → actual address = segment address + offset

Eg:

A starting address of data segment is 038E0H, so the value in DS register is 038E0H. An instruction references a location with an offset of 0032H bytes from the start of the data segment.

$\Rightarrow$ the actual address = DS segment address + offset

= 038E0H + 0032H

= 03912H

Registers

- Registers are used to control instructions being executed, to handle addressing of memory, and to provide arithmetic capability
- Registers of Intel Processors can be categorized into:
  1. Segment register
  2. Pointer register
  3. General purpose register
  4. Index register
  5. Flag register

i) Segment register

There are 6 segment registers :

(a) CS register

- Contains the starting address of program's code segment.
- The content of the CS register is added with the content in the Instruction Pointer (IP) register to obtain the address of the instruction that is to be fetched for execution.

(**Note**: common name for IP is PC (Program Counter))

(b) DS register

- Contains the starting address of a program's data segment.
- The address in DS register will be added with the value in the address field (in instruction format) to obtain the real address of the data in data segment.

(c) SS Register

- Contains the starting address of the stack segment.
- The content in this register will be added with the content in the Stack Pointer (SP) register to obtain the required word.

(d) ES (Extra Segment) Register

- Used by some string (character data) operations to handle memory addressing
- ES register is associated with the Data Index (DI) register.

(e) FS and GS Registers

- Additional extra segment registers introduced in 80386 for handling storage requirement.

(ii) Pointer Registers

- There are 3 pointer registers in an Intel PC :

(a) Instruction Pointer register

- The 16-bit IP register contains the offset address or displacement for the next instruction that will be executed by the CPU
- The value in the IP register will be added into the value in the CS register to obtain the real address of an instruction

Example :

The content in CS register = 39B40H

The content in IP register = 514H

next instruction address: 39B40H

\+ 514H

3A054H

- Intel 80386 introduced 32-bit IP, known as EIP (Extended IP)

(b) Stack Pointer Register (Stack Pointer (SP))

- The 16-bit SP register stores the displacement value that will be combined with the value in the SS register to obtain the required word in the stack
- Intel 80386 introduced 32-bit SP, known as ESP (*Extended SP*)

Example:

Value in register SS = 4BB30H
Value in register SP = + 412H
4BF42H

(c) Base Pointer Register

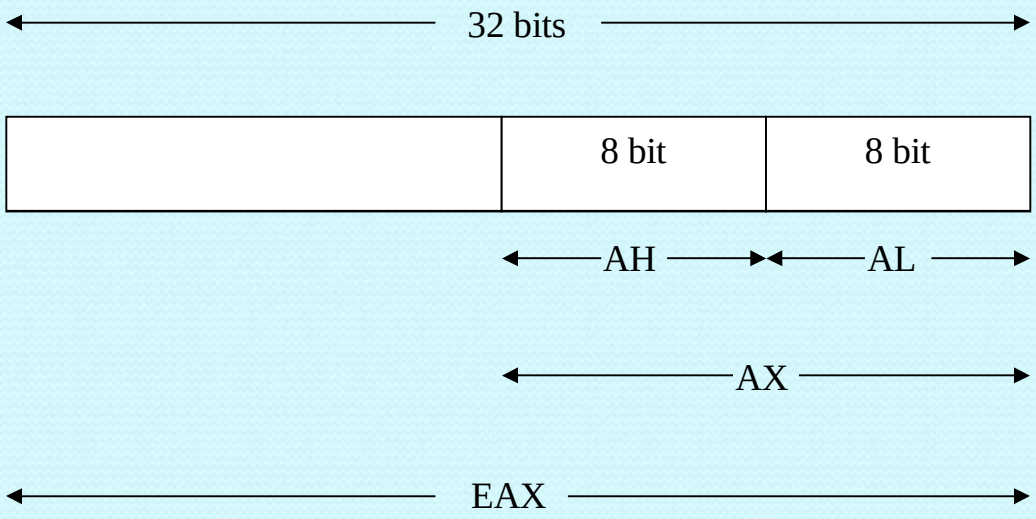
- The 16-bit BP register facilitates referencing parameters, which are data and addresses that a program passes via a stack
- The processor combines the address in SS with the offset in BP

(iii) General Purpose Registers

There are 4 general-purpose registers, AX, BX, CX, DX:

(a) AX register

- Acts as the **accumulator** and is used in operations that involve input/output and arithmetic
- The diagram below shows the **AX** register with the number of bits.



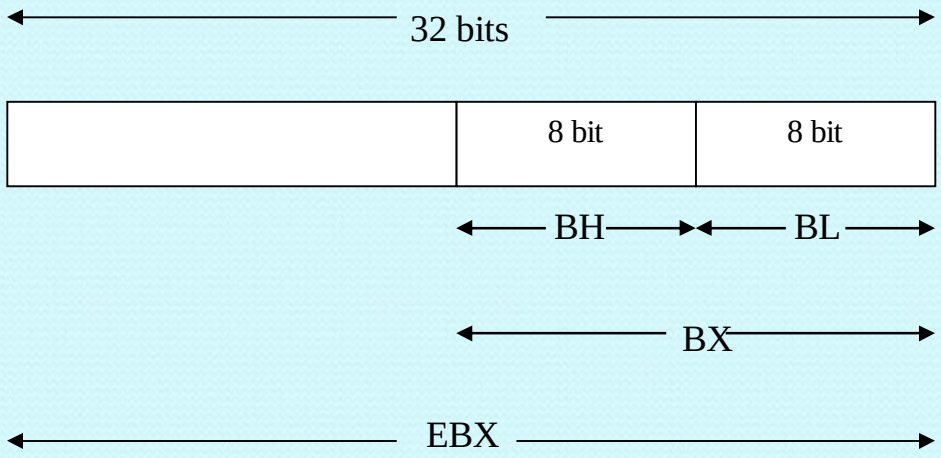
EAX : 32 bit
AX : 16 bit (rightmost 16-bit portion of EAX)
AH : 8 bit => leftmost 8 bits of AX (high portion)
AL : 8 bit => rightmost 8 bit of AX (low portion)

(b) BX Register

o Known as the **base register** since it is the only this general purpose register that can be used as an index to extend addressing.

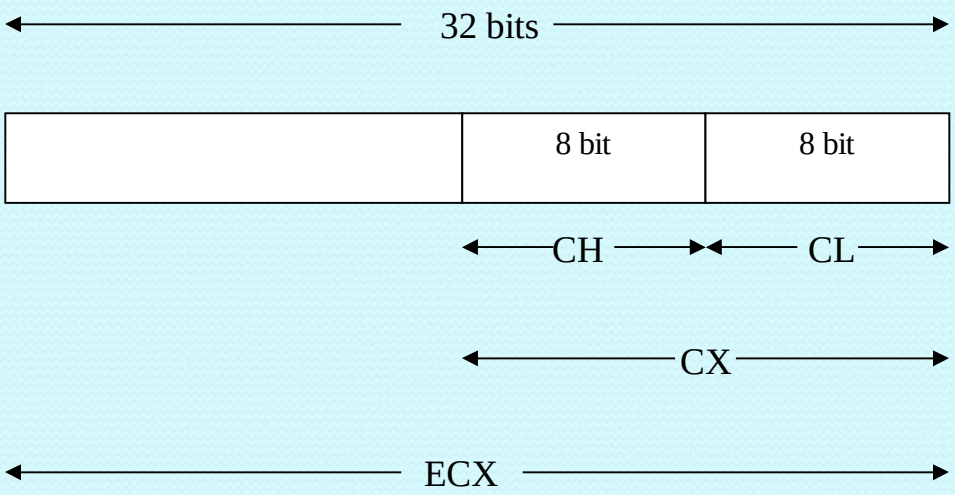
o This register also can be used for computations

o **BX** can also be combined with DI and SI register as a base registers for special addressing like AX, BX is also consists of **EBX**, BH and BL



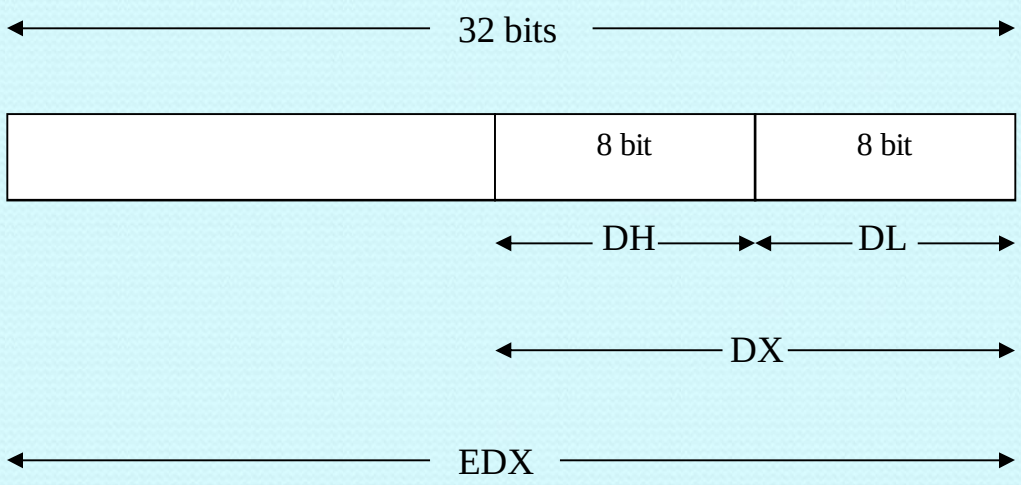
(c) CX Register

- known as **count** register
- may contain a value to control the number of times a loops is repeated or a value to shift bits left or right
- **CX** can also be used for many computations
- Number of bits and fractions of the register is like below :



(d) DX Register

- Known as **data** register
- Some I/O operations require its use
- Multiply and divide operations that involve large values assume the use of DX and AX together as a pair to hold the data or result of operation.
- Number of bits and the fractions of the register is as below :



(iv) Index Register

There are 2 index registers, SI and DI

(a) SI Register

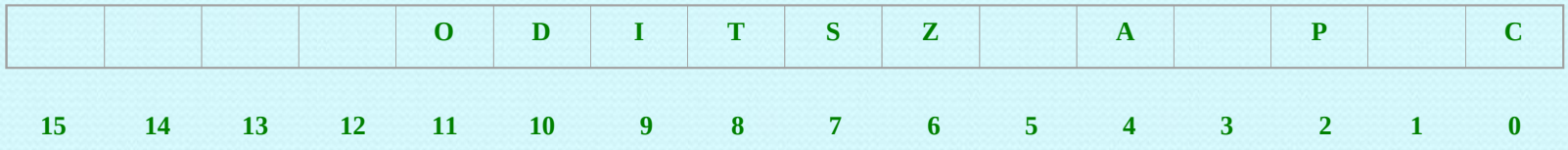
- Needed in operations that involve string (character) and is always usually associated with the DS register
- SI : 16 bit
- ESI : 32 bit (80286 and above)

(b) DI Register

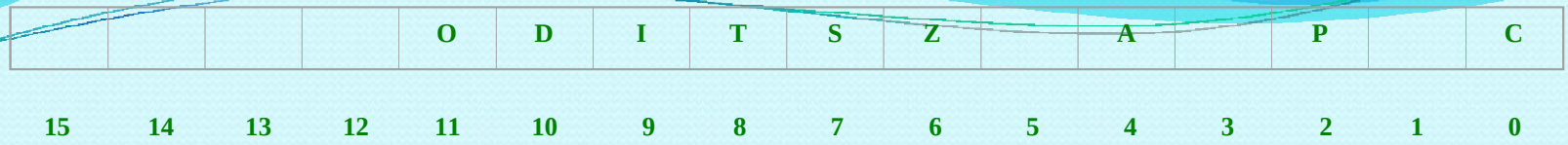
- Also used in operations that involve string (character) and it is associated with the ES register
- DI : 16 bit
- EDI : 32 bit (80386 and above)

(v) FLAG Register

- Flags register contains bits that show the status of some activities
- Instructions that involve comparison and arithmetic will change the flag status where some instruction will refer to the value of a specific bit in the flag for next subsequent action



- 9 of its 16 bits indicate the current status of the computer and the results of processing
- the above diagram shows the stated 9 bits



OF (*overflow*): indicate overflow of a high-order (leftmost) bit following arithmetic

DF (*direction*): Determines left or right direction for moving or comparing string (character) data

IF (*interrupt*): indicates that all external interrupts such as keyboard entry are to be processed or ignored

TF (*trap*): permits operation of the processor in single-step mode. Usually used in "debugging" process

SF (*sign*): contains the resulting sign of an arithmetic operation (0 = +ve, 1 = -ve)

ZF (*zero*): indicates the result of an arithmetic or comparison operation (0 = non zero; 1 = zero result)

AF (*auxillary carry*): contains a carry out of bit 3 into bit 4 in an arithmetic operation, for specialized arithmetic

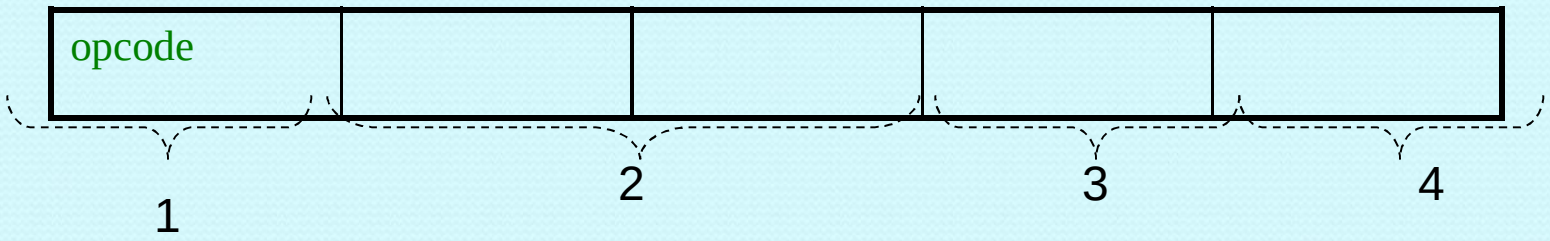
PF (*parity*): indicates the number of 1-bits that result from an operation. An even number of bits causes so-called even parity and an odd number causes odd parity

CF (*parity*): contains carries from a high-order (leftmost) bit following an arithmetic operation; also, contains the content of the last bit of a shift or rotate operation.

Instruction Format

- The operation of CPU is determined by the instructions it executes (machine or computer instructions)
- CPU's **instruction set** – the collection of different instructions that CPU can execute
- Each instruction must contain the information required by CPU for execution :-

1. Operation code (opcode) -- specifies the operation to be performed (eg: ADD, I/O) →*do this*
2. Source operand reference -- the operation may involve one or more source operands (input for the operation) → *to this*
3. Result operand reference -- the operation may produce a result → *put the answer here*
4. Next instruction reference -- to tell the CPU where to fetch the next instruction after the execution of this instruction is complete → *do this when you have done that*



- Operands (source & result) can be in one of the 3 areas:-
  - Main or Virtual Memory
  - CPU register
  - I/O device
- It is not efficient to put all the information required by CPU in a machine instruction
- Each instruction is represented by sequence of bits & is divided into 2 fields; opcode & address

opcode	address

- Processing become faster if all information required by CPU in one instruction or one instruction format

opcode	address for Operand 1	address for Operand 2	address for Result	address for Next instruction

- Problems → instruction become long (takes a few words in main memory to store 1 instruction)
- Solution → provide a few instruction formats (format instruction); 1, 2, 3 and addressing mode.
- Instruction format with 2 address is always used; INTEL processors