

BCN1043

COMPUTER ARCHITECTURE & ORGANIZATION

By
Dr. Mritha Ramalingam

Faculty of Computer Systems & Software Engineering
mritha@ump.edu.my

<http://ocw.ump.edu.my/>



CAO – Chapter 4 – P1. Mritha Ramalingam

AUTHORS

- **Dr. Mohd Nizam Mohmad Kahar** (mnizam@ump.edu.my)
- **Jamaludin Sallim** (jamal@ump.edu.my)
- **Dr. Syafiq Fauzi Kamarulzaman** (syafiq29@ump.edu.my)
- **Dr. Mritha Ramalingam** (mritha@ump.edu.my)

Faculty of Computer Systems & Software Engineering

Chapter 4: Assembly Level Machine Organization

LEARNING OUTCOMES

- Understand the assembly language instruction formats, such as addresses per instruction and variable length vs. fixed length formats and how subroutine calls are handled at the assembly level.
- Able to write assembly language program segments.

CHAPTER 4

ASSEMBLY LANGUAGE

- Introduction
- The Computer Organization
- Instruction Format
- Addressing Mode
- 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)

- for a specific family of processors (different processor groups/family has different Assembly Language)
- contains mnemonic instructions related with 1-to-1 machine instructions.

3) High-Level Language

- e.g. C, C++, V-basic
- Can eliminate the technicalities
- High level statements when compiled, produces many low-level instructions



Advantages

1. How a program can interface with BIOS, processor, and operating system
2. how to represent and store data in external memory devices.
3. how a processor can access, execute instructions
4. how a program can access external devices.

Reasons to use Assembly Language

1. less memory time and execution time than high-level language.
2. perform high technical tasks
3. recoding in assembly language is time-critical.
4. Self developing of Resident programs and interrupt service routines

To perform a task, the CPU involves:

- 1) Execution Unit (EU)
- 2) Bus Interface Unit (BIU)

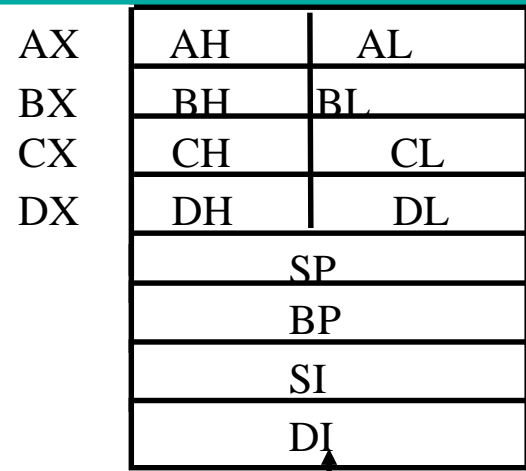
EU

- EU is responsible for **program execution**
- Includes ALU, a Control Unit and registers

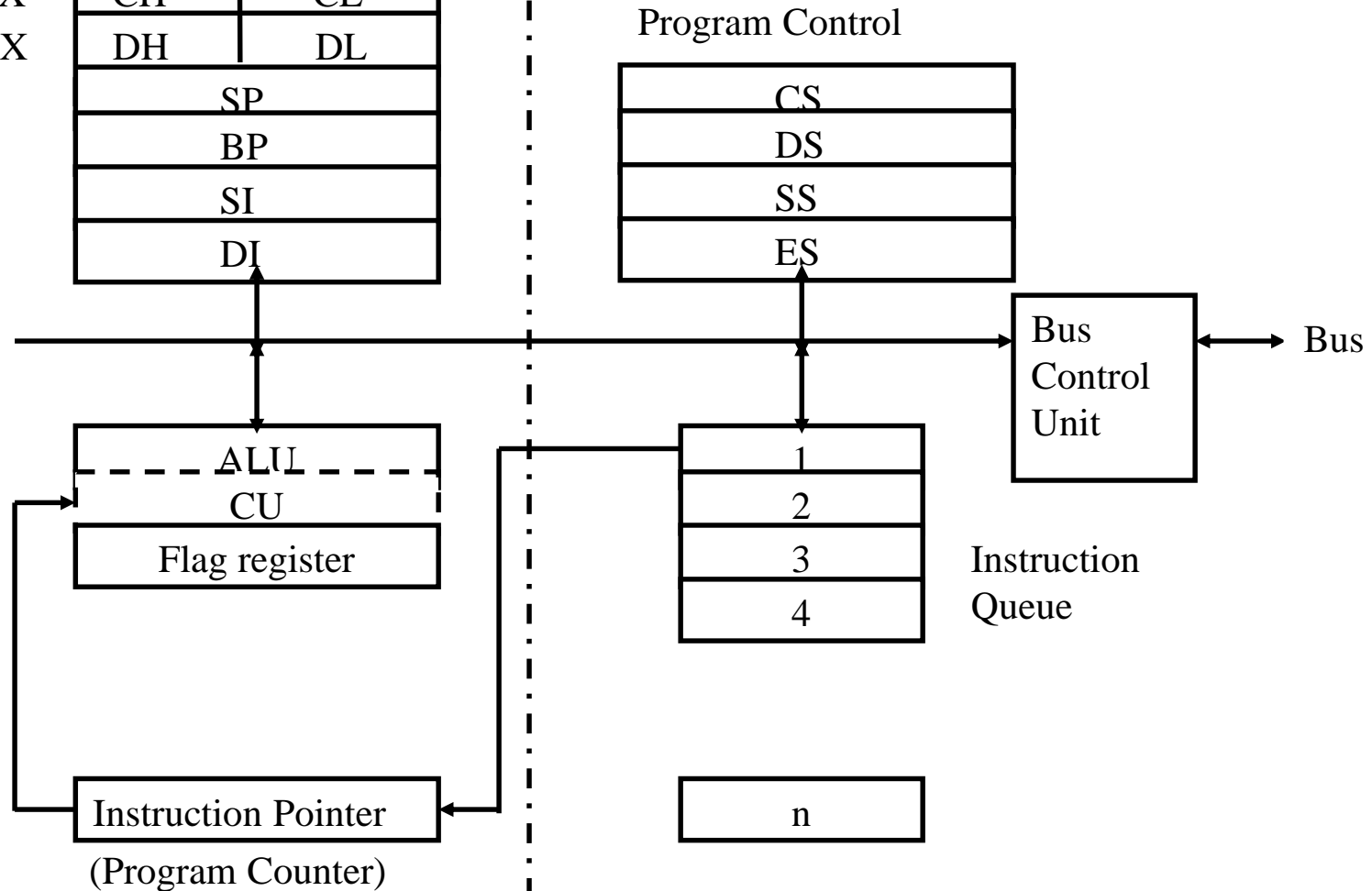
BIU

- **Delivers data and instructions** to the EU.
- manages control unit of bus, segment register and instruction queue.

EU : Execution Unit



BIU : Bus Interface Unit



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*)

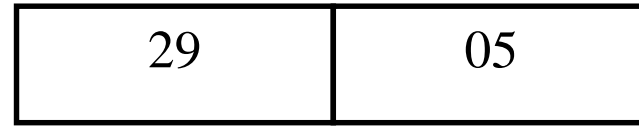
Example : consider value 0529_{16} (0529H)

2 bytes \rightarrow 05 and 29

register



memory



Address $04A26_{16}$
(low-order/least significant byte)

Address $04A27_{16}$
(high-order/most significant byte)

- 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 determined by programmer in the memory

(i) Code Segment (CS)

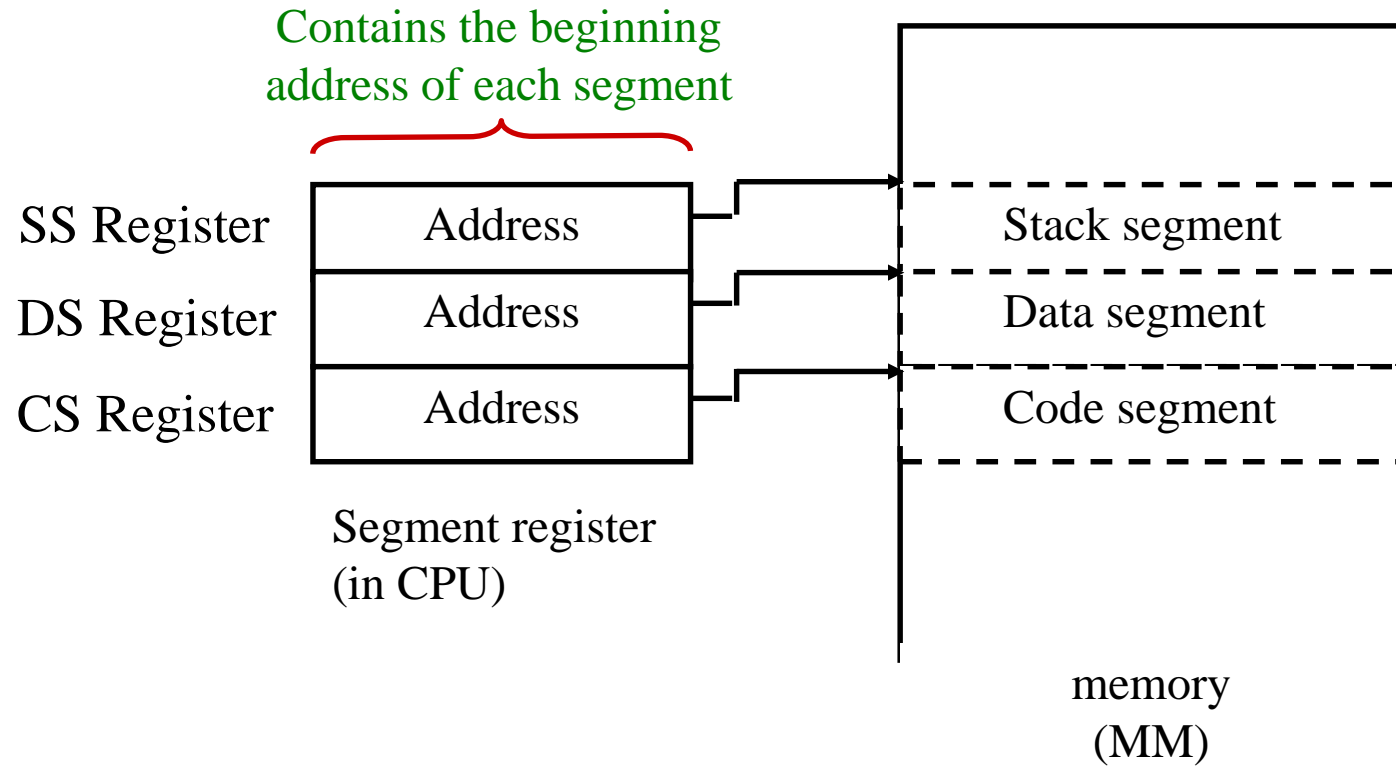
- machine **instructions** to be executed
- CS register holds the starting address of this segment

(ii) Data Segment (DS)

- Includes defined **data**, **constants** and **works areas**.
- DS register stores starting address of DS

(iii) Stack Segment (SS)

- Saves temporarily data.
- SS register holds the starting address a segment



Segment Offsets

- 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 address any memory location of a segment, the actual address is computed as
→ 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.

⇒ the actual address = DS segment address + offset

$$= 038E0H + 0032H$$

$$= 03912H$$

Registers

- Registers are used to control instructions being executed, to handle addressing of memory
- 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

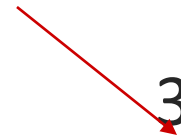
- The 16-bit IP register contains the offset address or displacement for the next instruction that will be executed by the CPU
- The value in the IP register will be added into the value in the CS register to obtain the real address of an instruction

Example :

The content in CS register = 39B40H

The content in IP register = 514H

next instruction address:

$$\begin{array}{r} 39B40H \\ + \quad 514H \\ \hline 3A054H \end{array}$$


- 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:

$$\begin{array}{r} \text{Value in register SS} = \quad 4\text{BB}30\text{H} \\ \text{Value in register SP} = \quad + \underline{412\text{H}} \\ \quad \underline{4\text{BF}42\text{H}} \end{array}$$

(c) Base Pointer Register

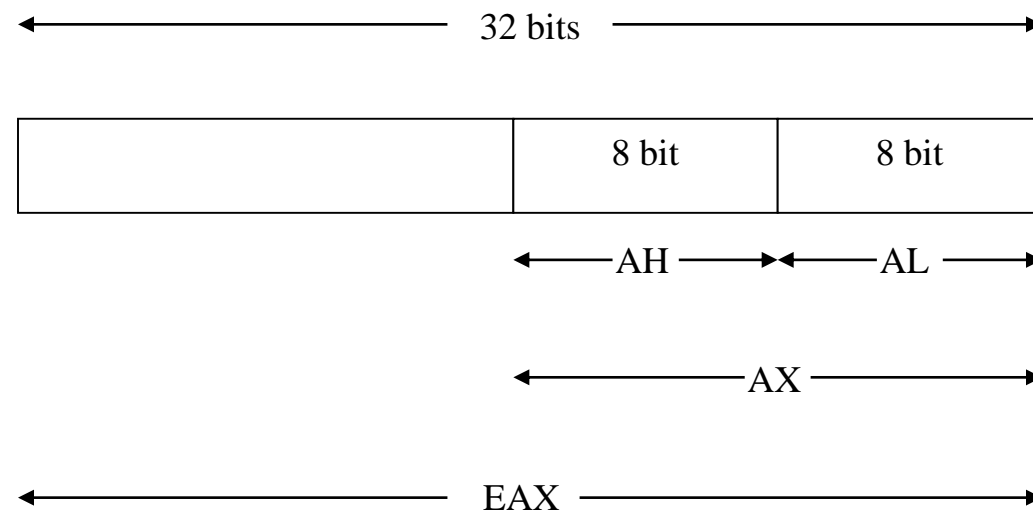
- The 16-bit BP register facilitates referencing parameters, which are data and addresses that a program passes via a stack
- The processor combines the address in SS with the offset in BP

(iii) General Purpose Registers

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

(a) AX register

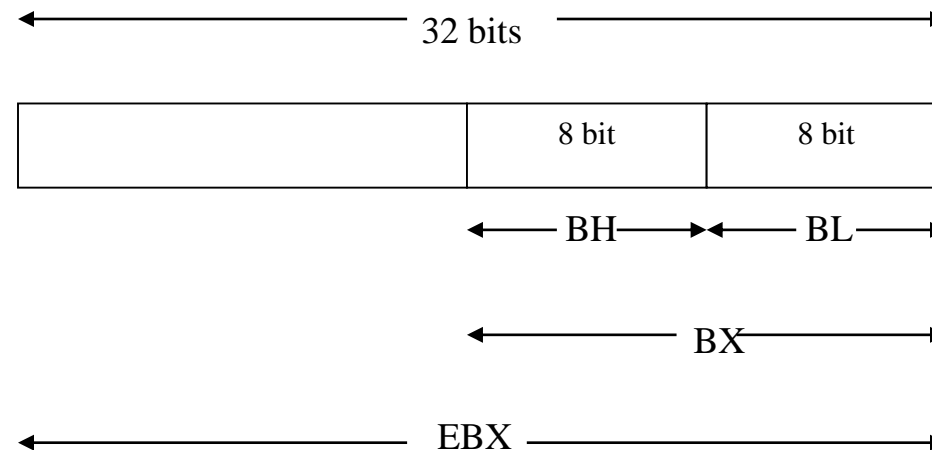
- Acts as the **accumulator** and is used in operations that involve input/output and arithmetic
- The diagram below shows the **AX** register with the number of bits.



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)

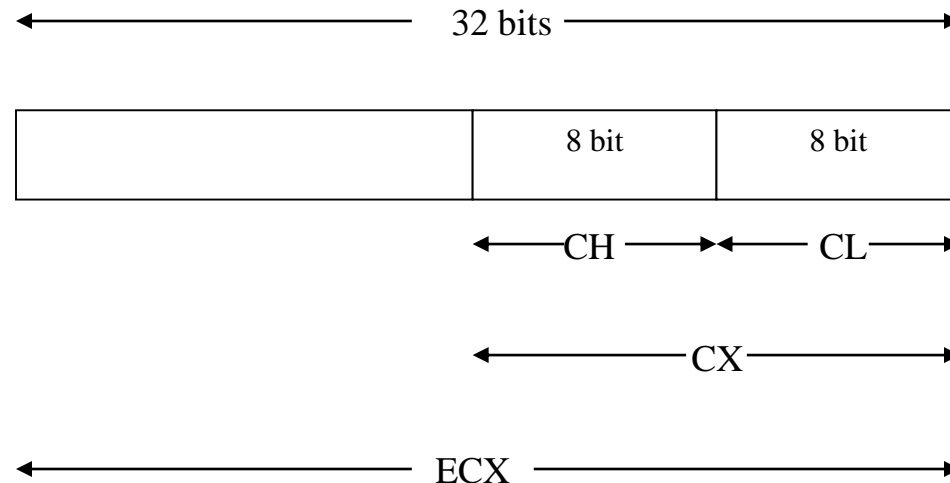
(b) BX Register

- 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



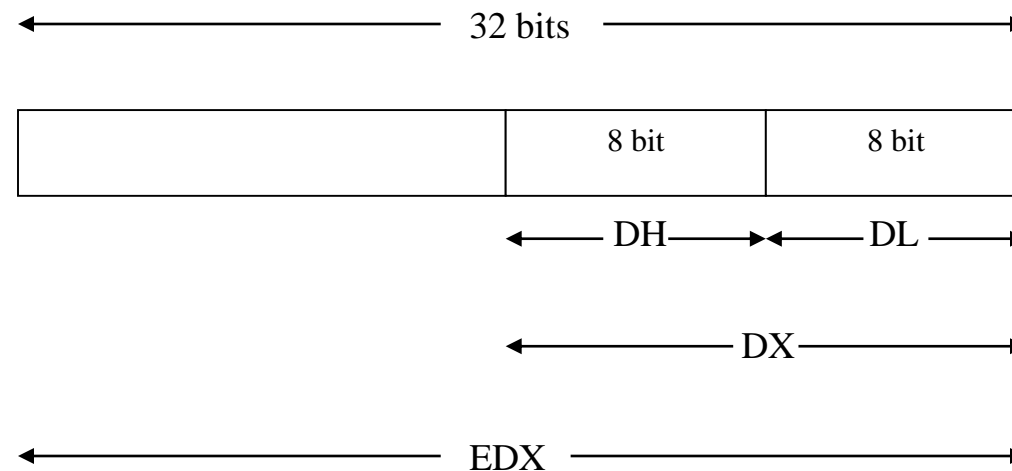
(c) CX Register

- 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

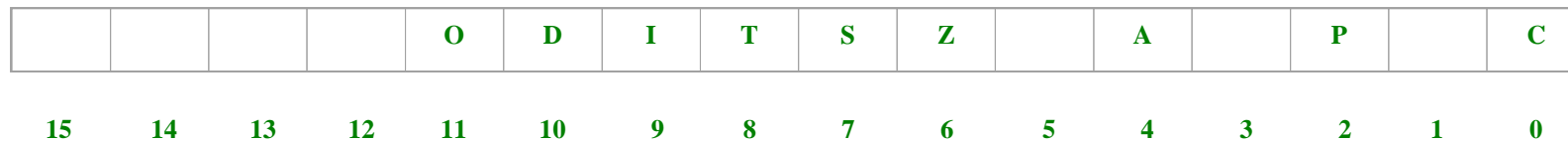
- o Needed in operations that involve string (character) and is always usually associated with the DS register
- o SI : 16 bit
- o ESI : 32 bit (80286 and above)

(b) DI Register

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

(v) FLAG Register

- o Flags register contains bits that show the status of some activities
- o 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

				O	D	I	T	S	Z		A		P		C
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

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 rotation

Will continue...