8086/8088 - CPU Architecture

The 8086/8088 architecture can be broadly divided into two groups:

(i) Execution Unit (EU)
(ii) Bus Interface Unit (BIU)

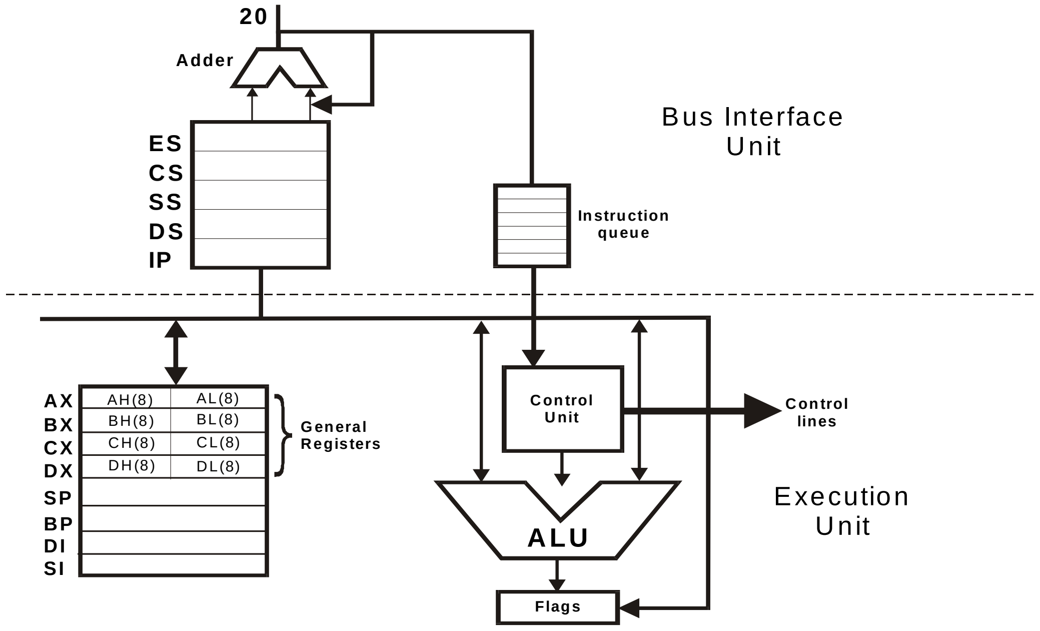

Fig 1.3 8086 Architecture

The execution unit contains the Data and Address registers, the Arithmetic and Logic Unit and the Control Unit. The Bus Interface Unit contains Bus Interface Logic, Segment registers, Memory addressing logic and a Six byte instruction object code queue (4-byte instruction object-code queue in case of 8088 microprocessor).

The execution unit and the Bus Interface unit operate asynchronously. The EU waits for the instruction object code to be fetched from the memory by the BIU.

The BIU fetches or pre-fetches the object code (16-bits at a time) and loads it into the six bytes queue. Whenever the EU is ready to execute a new instruction, it fetches the instruction object code from the front of the instruction queue and executes the instruction in specified number of clock periods.

If memory or Input/output devices must be accessed in the course of executing an instruction, then the EU informs the BIU of its needs. The BIU completes its operation code (opcode) fetch cycle, if in progress, and executes an appropriate external access machine cycle in response to the EU demand.

The BIU is independent of the EU and attempts to keep the six-bytes queue filled with instruction object codes. If two or more of these six bytes are empty, then the BIU executes instruction fetch machine cycles as long as the EU does not have an active request for the bus access pending. If the EU issues a request for the bus access while the BIU is in the middle of an instruction fetch machine cycle, then the BIU will complete the instruction fetch machine cycle before honoring the EU bus access request.

The EU does not use machine cycles; it executes instructions in some number of clock periods that are not subjected to any type of machine cycles. The only time clock periods are grouped is clock when the bus control logic wishes to access memory or I/O
devices.

Execution Unit (EU)

The execution unit consists of

General Registers

Arithmetic Logic Unit
Control unit
Flag Registers

General Registers

The CPU has eight 16-bit general registers. They are divided into two files of four registers each. They are:

(a) The data register file and
(b) The pointer and index register file

AH	**AL**	**AX**
BH	**BL**	**BX**
CH	**CL**	**CX**
DH	**DL**	**DX**

Fig. 1.4 Data Register File

AX, BX, CX and DX registers are the data registers. The upper and lower halves of the data registers are individually addressable. AX register can be addressed as AL and AH registers, BX register can be addressed as BL and BH register, CX register can be addressed as CL and CH register, DX register can be addressed as DL and DH.

The data registers can be used in most arithmetic and logic operations. Some instructions however require these registers for specific use. This implicit register usage allows a more compact instruction encoding. Fig.1.4 shows the data registers specific one. The index register file consists of the Stack Pointer (SP), the Base Pointer (BP), Source Index (SI) and Destination Index (DI) registers all are of 16-bits. They can also be used in most arithmetic and logic operations. These registers are usually used to hold offset addresses for addressing within a segment. Offset addressing reduces program size by eliminating the need for each instruction to specify frequently used addresses.

The pointer and index register files are further divided into the pointer sub-file (containing the Stack Pointer and the Base Pointer registers) and the index sub-file (containing the Source index and Destination index registers). The Pointer registers are used to access the current stack segment. The index registers are used to access the current data. (Stack segment and data segment are specific areas of memory. Their application will be explained in later chapters). Unless otherwise specified in the instruction, stack pointer registers refer to the current stack segment while index register refers to the current data segment.

The BP and SP registers are both used to point to the stack, a linear array in the memory used for subroutine parameters, subroutine return addresses, and the data temporarily saved during execution of a program

The implicit register usage is as follows:

AX Register	Word Multiplication Word Division and Word I/O Operation.
AL Register	Byte Multiplication Byte Division Byte I/O Translate, and Decimal Arithmetic
AH Register	Byte Multiplication Byte Division.
BX Register	Base Register Translate
CX Register	String Operations
CL Register	Variable Shift and Rotate
DX Register	Word Multiplication,

	Word Division, Indirect I/O.

Fig. 1.5

Most microprocessors have a single stack pointer register called the SP. 8086 / 8088 has an additional pointer into the stack called the BP register. While the SP is used similar to the stack pointer in other machine (for pointing to subroutine and interrupt return addresses), the BP register is used to hold an old stack pointer value, or it can mark a place in the subroutine stack independent of the SP register. Using the BP register to mark the stack saves the juggling of a single stack pointer to reference subroutine parameters and addresses.

SI and DI are both 16-bits wide and are used by string manipulation instructions and in building some of the more powerful 8086/8088 data structures and addressing modes. Both the SI and the DI registers have auto incrementing and auto-decrementing capabilities.

Arithmetic Logic Unit (ALU)

ALU is 16-bits wide. It can do the following 16-bits arithmetic operations

(i) Addition
(ii) Subtraction
(iii) Multiplication
(iv) Division

Arithmetic operations may be performed on four types of numbers

- Unsigned binary numbers
- Signed binary numbers (Integers)
- Unsigned packed decimal numbers
- Unsigned unpacked decimal numbers

The ALU can also perform logical operations such as

(i) NOT
(ii) AND
(iii) OR
(iv) EXCLUSIVE OR
(v) TEST

Flag Register

The Execution Unit has a 16-bit flag register which indicates some conditions affected by the execution of an instruction. Some bits of the flag register control certain operations of the EU. The flag register in the EU contains nine active flags shown in fig.1.6

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
				O	D	I	T	S	Z		AF		PF		CF

Fig, 1.6 Flag Register

Six of the nine flags are used to indicate some condition produced by an instruction. These condition flags are also called status flags of 8086/8088 microprocessor. These are the Carry flag, Parity flag, Auxiliary carry flag, Zero flag, and Sign flag. The other three Control flags are Trap Flag, Direction Flag and Interrupt flag.

Condition Flags

Carry Flag (CF)

This flag will be set to one if the addition of two 16-bit binary numbers produces a carry out of the most significant bit position or if there is a borrow to the MSB after subtraction. This flag is also affected when other arithmetic and logical instruction are executed.

Parity Flag (PF)

This flag is set, if the result of the operation has an even number of 1's (in the lower 8 bits of the result). This flag can be used to check for data transmission error.

Auxiliary Carry Flag (AF)

This flag is set, when there is a carry out of the lower nibble to the higher nibble or a borrow from the higher nibble to the lower. The auxiliary carry flag is used for decimal adjust operation. The AF flag is of significance only for byte operations during
which the lower order byte of the 16-bit word is used.

Zero Flag (Z)

This flag is set when the result of an operation is zero. The flag is reset when the
result is not zero.

Overflow Flag (O)

This flag is set, when an arithmetic overflow occurres. Overflow means that the size of the result exceeded the storage capacity of the destination, and a significant digit
has been lost.

Sign flag (S)

This flag is set, when an MSB bit of the result is high after an arithmetic operation. When this flag is set the data in assumed to be negative and when this flag is
zero it is assumed to be positive.

Control Flags

Control flags are used to control certain operations of the processor. The application of these flags are different from that of six conditional flags. The conditional flags are set or reset by the EU on the basis of the result of some arithmetic or logic operations. The control flags are deliberately set or reset with specific instructions included in the program.

Trap flag (T)

This is used for single stepping through a program. It is used for debugging the
programs. (Discusses with interrupts).

Interrupt Flag (I)

It is used to allow / prohibit the interruption of a program. When the flag set, it
enables the interrupt from INTR. When the flag is reset (0), it disables the interrupt.

Direction Flag (D)

It is used for string instructiion (Discussed with the specific instructions later in
the book). If the direction flag is set, the pointers are decremented else the pointers are
incremented.

1.5.2 Bus Interface Unit (BIU)

The BIU sends out addresses, fetches instructions from memory, reads data from memory and ports, and writes data to ports and memory. In other words the BIU handles
all transfers of data and addresses on the buses for the execution unit. The BIU has

1. An instruction queue
2. An Instruction pointer
3. Segment registers

Instruction Queue

To speed up program execution, the BIU fetches as many as 6 insturction bytes ahead of time from memory. The prefetched instruction bytes are held for the EU in a first-in-first-out group of register called a queue. The EU decodes an instruction or executes an instruction which does not require the buses. When the EU is ready for its next instruction, it simply reads the instruction from the queue in the BIU. Fetching the next instruction while the current instruction executes, is called pipelining.

Note: The 8088 microprocessor has only a 4-byte queue.

Instruction Pointer (IP)

The Instruction Pointer is a 16-bit register. This register is always used as the effective memory address, and is added to the Code segment with a displacement of four bits to obtain the physical address of the opcode. The code segment cannot be changed by the move instruction. The instruction pointer is incremented after each opcode fetch
to point to the next instruction.

Segment Registers

The 8086 / 8088 microprocessor has 20-bit address lines. All the registers in 8086 / 8088 are 16-bits in length. Hence to obtain 20-bit addresses from the available 16-bit registers, all 8086 / 8088 memory addresses are computed by summing the contents of a segment register and a effective memory address. The effective memory address is computed via a variety of addressing modes. The process of adding, to obtain 20-bit address is as follows:

The selected segment register contents are shifted-left four bits (i.e., the contents are multiplied by 16 decimal), and then added to the effective memory address to generate the actual physical address output.

Segment Register value (CS, DS, ES or SS)	x x x x x x x x x x x x x x x xH
Effective Memory Address	y y y y y y y y y y y y y y y yH
Physical Address	wwwwwwwwwwwwwwwwwwwwH

Table 1.1

The table 1.1 shows 16-bits of the segment registers CS, DS, ES or SS displaced by 4-bits to the left. The effective address is calculated depending on the type of addressing mode. The effective address is shown as "yyyyyyyyyyyyyyyy". The 20-bit physical address "wwwwwwwwwwwwwwwwwwww" is obtained after adding the segment register value and effective address. The physical address is 20-bits wide.

To understand how the segmentation is used, it is required to know the memory structure of the 8086 / 8088 microprocessor.

The memory in an 8086/8088 system is a sequence of up to 2^{20} = one million bytes. A word is any two consecutive bytes in memory (word alignment is not required). Words are stored in memory with the most significant byte at the higher memory address. These bytes are stored sequentially from byte 00000 to byte FFFFF hex.

Programs view memory space as a group of segments defined by the application. A segment is a logical unit of memory that may be up to 64K bytes long. Each segment is made up of contiguous memory locations and is an independent, separately addressable unit. Each segment is assigned a base address, which is its starting location in the memory space. All segments start on 16-bit memory boundaries. Segments may be adjacent, disjoint, partially overlapped, or fully overlapped. It is as shown in fig. 1.7

The segment registers point to the four immediately addressable segments. The four segment registers are

- Code Segment register [points to the instruction opcode]

- Data Segment register [points to the data memory]
- Stack Segment register [points to the Stack memory]
- Extra Segment register [points to the data memory]

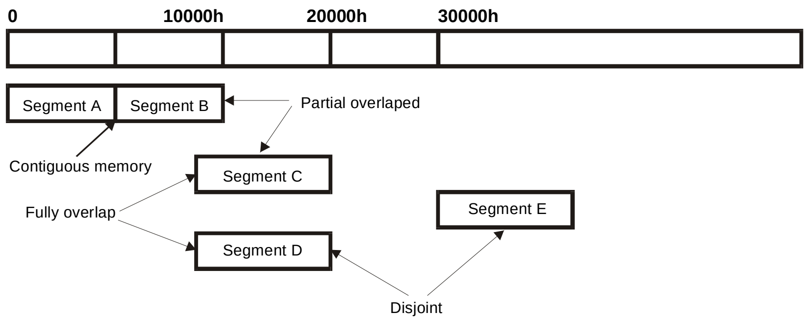

Fig 1.7

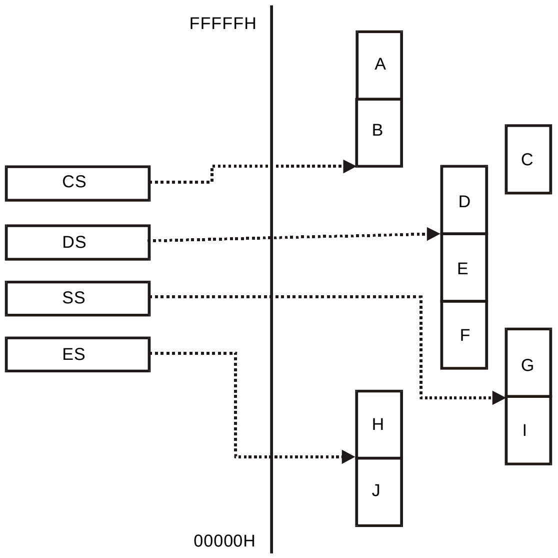

Fig-1.8

Fig 1.8 shows the segment registers pointing to the various memory segments. Since logical addresses are 16-bits wide, up to 64K (65536) bytes in a given segment can be addressed.

Each time the CPU need to generate a memory address, one of the segment registers is automatically chosen and its contents added to a logical address.

For an instruction fetch, the code segment register is automatically added to the logical address (in this case, the contents of the instruction pointer) to compute the value of the instruction address.

For stack referencing the stack segment register is automatically added to the logical address (the SP or BP register contents) to compute the value of the stack address.

For data reference operations, where either the data or extra segment register is chosen as the base, the logical address can be made up of many different types of values: it can be simply the immediate data value contained in the instruction, or it can be the sum of an immediate data value and a base register, plus an index register. Generally, the selection of the DS or ES register is made automatically, though provisions do exist to override this selection. Thus any memory location may be addressed without changing the value of the segment base register. In systems that use 64K or fewer bytes of memory for each memory area (code, stack, data and extra), the segment registers can be initialized to zero at the beginning of the program and then ignored, since zero plus a 16-bit offset yields a 16-bit address. In a system where the total amount of memory is 64K bytes or less, it is possible to set all segments equal and have fully overlapping segments.

Segment registers are also very useful for large programming tasks, which require isolation of program code from the data area, or isolation of module data from the stack information etc.

Segmentation makes it easy to build re-locatable and reentrant programs. In many cases, the task of relocating a program (relocation means having the ability to run the same program in several different areas of memory without changing addresses in the program itself) simply requires moving the program code and then adjusting the code segment register to point to the base of the new code area. Since programs can be written for the 8086 / 8088 in which all branches and jumps are relative to the instruction pointer, it does not matter what value is kept in the code segment register. Every application will define and use segments differently. The currently addressable segment override provide, a generous workspace: 64K bytes for code, 64K bytes stack and 128K bytes of data
storage.

Solved Problems

1. If a physical branch address is 5A230 H when (CS) = 5200 H, what will it be if the (CS) are changed to 7800 H.

```
          CS:        52 0 0
        Offset:        XXXX
   Physical add. 5A2 3 0 H
```

Hence Offset = Physical add - (Segment address displaced by 4-bits)

Offset = 5A230 - 52000 = 8230 H

If the CS is changed to 7800 H the Physical address will be

78000 + 8230 = 80230

2. Given that the EA of a datum is 2359 H and the DS = 490B H, what is the physical address of the datum?

```
          DS:      490B0 H
          EA:        2359 H
    Physical add.   4B409
```