

EEE 410 – Microprocessors I

Fall 05/06 – Lecture Notes # 2

Outline of the Lecture

- Brief History of 80x86 Family of Microprocessors
- Pipelining and Registers
- Introduction to Assembly Programming

✚ BRIEF HISTORY OF 80X86 FAMILY OF MICROPROCESSORS

Evolution from 8080/8085 to 8086

- Intel introduced 8086 microprocessor in 1978. This 16-bit microprocessor was a major improvement over the previous generation of 8080/8085 series of microprocessors.

8086	8080/8085
1 megabyte (20-bit add. bus)	Memory of 64 kilobyte (16-bit add. bus)
16-bit Data bus	8-bit data bus
Pipelined processor (first single-chip μ pr.)	Non-pipelined μ pr

- In a system with pipelining, the data and the address bus are busy transferring data while the CPU is processing information.

Evolution from 8086 to 8088

- 8086 was with 16-bit data bus internally and externally. All registers and the data bus carrying data in/out of the CPU were 16-bit.
 - That time all the peripherals were designed around 8-bit microprocessor.
 - It was expensive to built PCB with 16-bit data bus.
- So Intel introduced **8088** which was;
 - Identical to 8086 internally, but externally 8-bit data bus instead of 16-bit.
 - It had 1 megabyte of memory like 8086.
- IBMs decision to pick up 8088 as their choice of microprocessor in designing the IBM PC.
 - 8088-based IBM PC was enormous success, because IBM and Microsoft made it an open system.
 - This enabled the cloning of this system and resulted a huge growth in both hardware and software designs based on IBM PC.
 - In contrast IBMs main competitor Apple computer introduced a closed system and blocked all attempts of cloning.

Other microprocessors: the 80286, 80386, and 80486

- **80286**: Intel introduced 80286 in 1982.
 - With 16-bit internal and external data bus.
 - 24-bit address bus ($2^{24} = 16$ megabyte)
 - **virtual memory**: a way of fooling the microprocessor into thinking that it has access to unlimited memory by swapping data between disk storage and RAM.
 - **Real mode** (faster operation with maximum of 1 Mbytes of memory) vs. **Protected mode** protecting the operating system for accidental or deliberate destruction of the user. Protected mode is slower but can use 16 Mbytes of memory.

- **80386**: introduced in 1985 also known as (80386DX)
 - With 32-bit internal and external data bus.
 - 32-bit address bus ($2^{32} = 4$ gigabyte-physical memory). With virtual memory 64 terabytes(2^{46}).
 - 80386SX was later introduced with the same internal structure with 16-bit external data bus and 24-bit address bus. 80386SX was much cheaper.

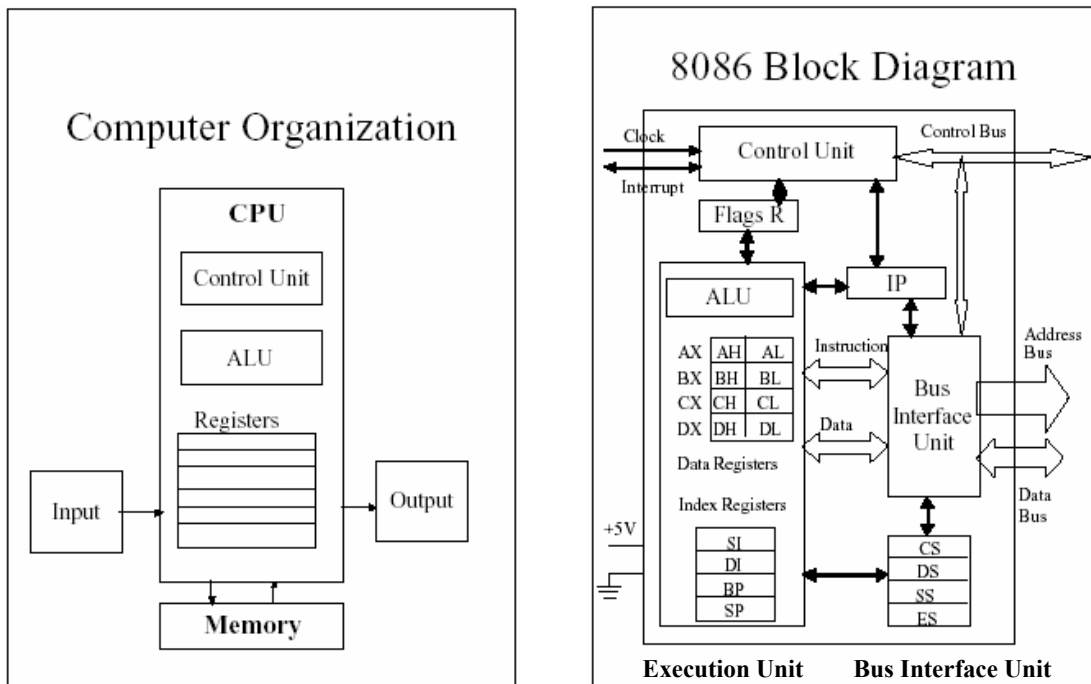
- All microprocessors discussed so far were general-purpose microprocessors and could not handle mathematical operations rapidly. For this reason, **8087**, **80287**, **80387** numeric data processing chips called **math co-processors** were used.

- **80486**: introduced in 1989 with 32-bit internal-external data bus and 32-bit address bus.
 - built in math co-processor in a single chip.
 - Introduction of **cache memory** (Static RAM with very fast access time)

Table 1: Evolution of Intel's Microprocessors

Product	8080	8085	8086	8088	80286	80386	80486
Year Introduced	1974	1976	1978	1979	1982	1985	1989
Clock rate (MHz)	2-3	3-8	5-10	5-8	6-16	16-33	25-50
No. transistors	4500	6500	29,000	29,000	130,000	275,000	1.2 million
Physical memory	64K	64K	1M	1M	16M	4G	4G
Internal data bus	8	8	16	16	16	32	32
External data bus	8	8	16	8	16	32	32
Address bus	16	16	20	20	24	32	32
Data type (bits)	8	8	8,16	8,16	8,16	8,16,32	8,16,32

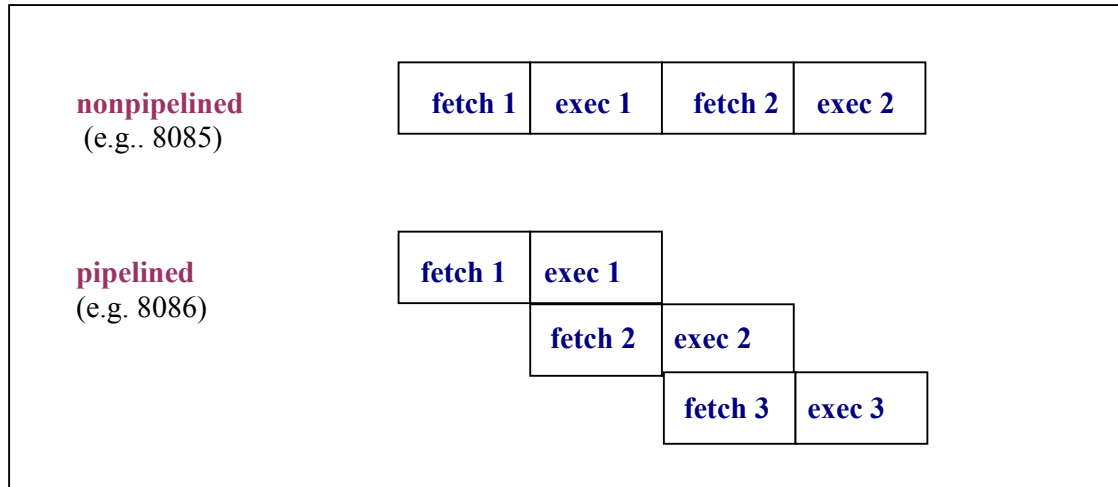
🔗 **THE 8086 INTERNAL ORGANIZATION, PIPELINING AND REGISTERS**



Pipelining

- In the 8085 microprocessor, the CPU could either fetch or execute at a given time. CPU had to fetch an instruction from the memory, then execute it, then fetch again and execute it and so on..
- Pipelining is the simplest form to allow the CPU to *fetch* and *execute* at the same time. Note that the fetch and execute times can be different.

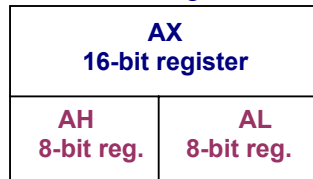
Pipelined vs. Nonpipelined Execution



- Intel implemented the concept of pipelining by splitting the internal structure of 8088/86 into two sections.
 - the *execution unit* (EU)
 - the *bus interface unit* (BIU)
 - These two sections work simultaneously. BIU accesses memory and peripherals while the EU executes the instructions previously fetched.
 - It only works if BIU keeps ahead of EU. Thus BIU has a buffer of *queue*. (8088 has 4 byte, and 8088 has 6 bytes).
 - If the execution of any instruction takes too long, the BIU is filled to its maximum capacity and busses will stay idle. It starts to fetch again whenever there is 2-byte room in the queue.
 - When there is a jump instruction, the microprocessor must flush out the queue. When a jump instruction is executed BIU starts to fetch information from the new location in the memory. In this situation EU must wait until the BIU starts to fetch the new instruction. This is known as *branch penalty*.

Registers of 8086 Microprocessor

- In the CPU, registers are used store information temporarily. The information can be one or two bytes of data, or the address of data.
- In 8088/8086 general-purpose registers can be accessed as either 16-bit or 8-bit registers. All other registers can be accessed as full 16-bit registers.

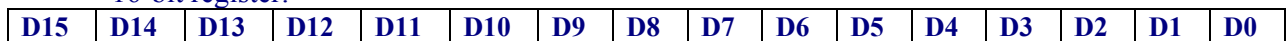


- The bits of the registers are numbered in descending order:

8-bit register:



16-bit register:



- Different registers are used for different functions. Registers will be explained later within the context of instructions and their applications.
- The first letter of each general register indicates its use.
 - AX is used for the *accumulator*.
 - BX is used for *base addressing register*.
 - CX is used for *counter loop operations*.
 - DX is used to point out *data in I/O operations*.

Registers of 8086

Category	Bits	Register Names
General	16	AX, BX, CX, DX
	8	AH, AL, BH, BL, CH, CL, DH, DL
Pointer	16	SP (stack pointer), BP (base pointer)
Index	16	SI (source index), DI (destination index)
Segment	16	CS (code segment), DS (data segment) SS (stack segment), ES (extra segment)
Instruction	16	IP (instruction pointer)
Flag	16	FR (flag register)

- Note: the general registers can be accessed as full 16 bits (such as AX), or as the high byte only (AH) or low byte only (AL). The others are not!!

INTRODUCTION TO ASSEMBLY PROGRAMMING

Machine Language:

- Programs consist of 0s and 1s are called machine language.
- *Assembly Languages* provided *mnemonics* for machine code instructions.
- *Mnemonics* refer to codes and abbreviations to make it easier for the users to remember.

Low / High level languages:

- Assembly Language is a low-level language. Deals directly with the internal structure of CPU. *Assembler* translates Assembly language program into machine code.
- In high-level languages, Pascal, Basic, C; the programmer does not have to be concerned with internal details of the CPU. *Compilers* translate the program into machine code.

Assembly Language programming:

- Assembly Language program consists of series of lines of Assembly language *instructions*.
- *Instruction* consists of a mnemonic and two operands.

MOV instruction

MOV destination, source; copy source operand to destination

mnemonic operands

Example: (8-bit)

```
MOV CL,55H      ;move 55H into register CL
MOV DL,CL       ;move/copy the contents of CL into DL (now DL=CL=55H)
MOV BH,DL       ;move/copy the contents of DL into BH (now DL=BH=55H)
MOV AH,BH       ;move/copy the contents of BH into AH (now AH=BH=55H)
```

Example: (16-bit)

```
MOV CX,468FH    ;move 468FH into CX (now CH =46 , CL=8F)
MOV AX,CX       ;move/copy the contents of CX into AX (now AX=CX=468FH)
MOV BX,AX       ;now BX=AX=468FH
MOV DX,BX       ;now DX=BX=468FH
MOV DI,AX       ;now DI=AX=468FH
MOV SI,DI       ;now SI=DI=468FH
MOV DS,SI       ;now DS=SI=468FH
MOV BP,DS       ;now BP=DS=468FH
```

- Data can be moved among all registers except the *flag* register. There are other ways to load the flag registers. To be studied later.
- Source and destination registers have to *match in size*.
- Data can be moved among all registers (except flag reg.) but data can be moved *directly* into *nonsegment* registers only. You can't move data segment registers directly.

Examples:

```
MOV BX,14AFH    ;move 14AFH into BX           (legal)
MOV SI,2345H    ;move 2345H into SI           (legal)
MOV DI,2233H    ;move 2233H into DI           (legal)
MOV CS,2A3FH    ;move 2A3FH into CS           (illegal)
MOV DS,CS       ;move the content of CS into DS (legal)
MOV FR,BX       ;move the content of BX into FR (illegal)
MOV DS,14AFH    ;move 14AFH into DS           (illegal)
```

➤ *Important points:*

- Values cannot be loaded directly into (CS,DS,SS and ES)

```
MOV AX,1234H ; load 1234H into AX
```

```
MOV SS,AX ;load the value in AX into SS
```

- Sizes of the values:

```
MOV BX,2H ; BX=0002H, BL:02H, BH:00H
```

```
MOV AL,123H ; illegal (larger than 1 byte)
```

```
MOV AX,3AFF21H ; illegal (larger than 2 bytes)
```