Lecture 3

October 21, 2013

1 8086 Internal Architecture

1.1 Main Units

- The 8086 CPU has two main units

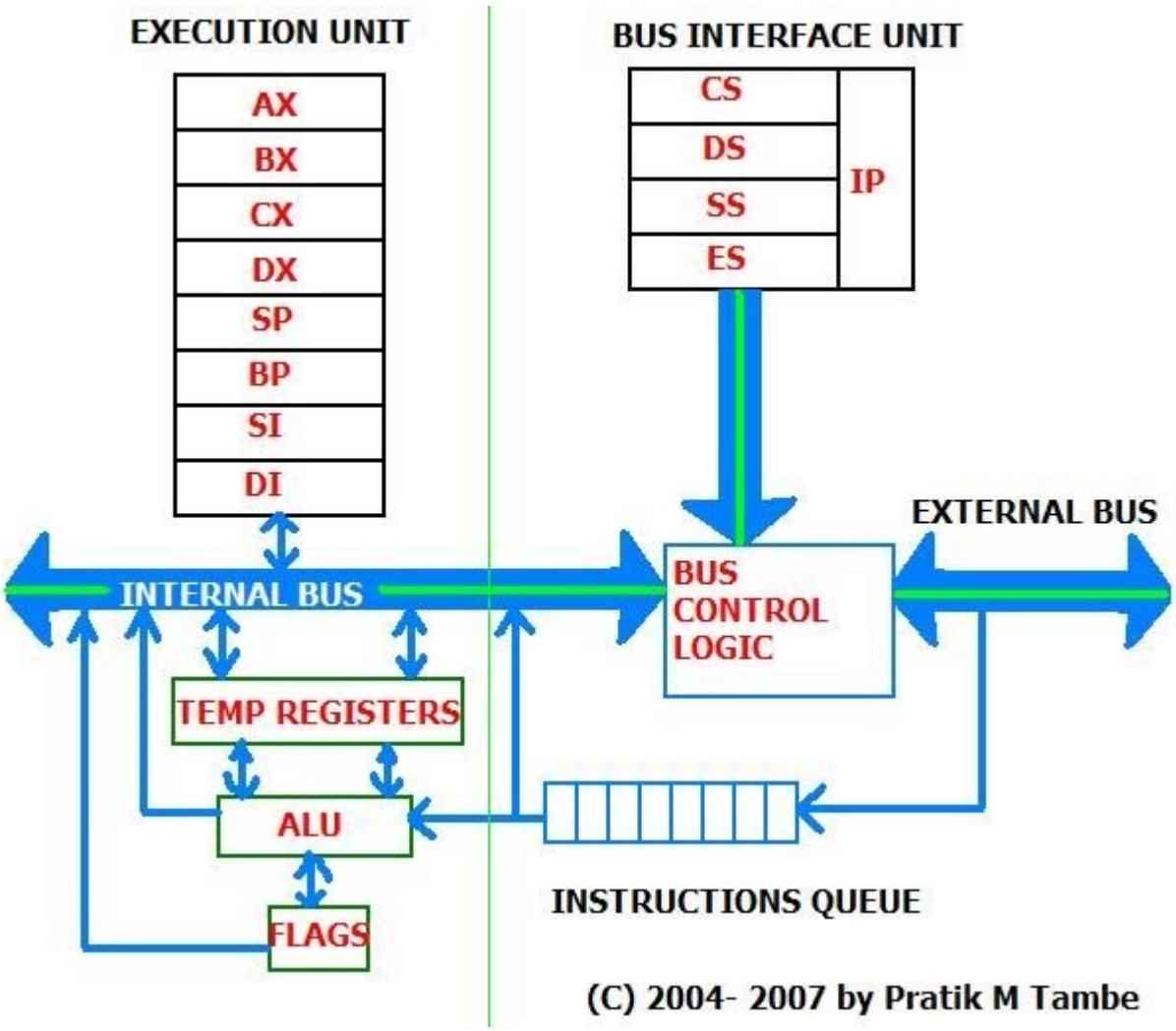

Figure 1: Intel 8086 Architecture

  - Bus Interface Unit (BIU):
    - The BIU provides H/W functions, including generation of the memory and I/O addresses for the transfer of data between the outside world outside the CPU

    * BIU reads (fetches) instructions, reads operands, and writes results.
  - Execution Unit (EU):
    * The EU receives program instruction codes and data from the BIU, executes these instructions, and store the results in the general registers.
    * By passing the data back to the BIU, data can also be stored in a memory location or written to an output device.
    * Note that the EU has no connection to the system buses. It receives and outputs all its data thru the BIU
    * executes instructions already fetched by the BIU

1.2 8086 Memory

- Even though the 8086 is considered a 16-bit processor, (it has a 16-bit data bus width) its **memory is still thought of in bytes**. This design is useful from different perspectives
  - First, it allows the processor to work on bytes as well as words. This is especially important with I/O devices such as printers, terminals, and modems, all of which are designed to transfer ASCII-encoded (7- or 8-bit) data.
  - Second, **many of the 8086's (and 8088's) operation codes are single bytes.** Other instructions may require anywhere from two to seven bytes. By being able to access individual bytes, these odd-length instructions can be handled.
- Since 8086/88 has a 20-bit address bus, it can access 1'048.576, different memory byte addresses as well as 524.288 words.
- Since, the ***physical address*** of any byte consists of five hex digits, an important question arise.
  - How to formulate a 20-bit address if all the registers are 16-bit addresses.
  - The answer of this question is ***memory segmentation***.
- Any address is composed from two parts
  - ***base (segment)*** part identified by on of the BIU segment registers (CS, DS, ES, and SS) are used to "point" at location 0 (the base address) of each segment.
  - ***offset (logical)*** address which gives the displacement from the address base of the segment to the desired location within that segment (memory block)

The physical address is 20 bits long and corresponds to the actual binary code output by the BIU on the address bus lines. The logical address is an offset from location 0 of a given segment.

- Since, segment registers are only 16 bits wide while the memory address is 20bits, the BIU takes care of this problem by appending four 0's to the low-order bits of the segment register. After that the result is added to the logical address to calculate the physical address.

Example:

	A4FB0h	Shifted segment value
+	04872h	Offset
	A9822h	(20-bit physical address)

Example

A memory location has physical address 80FD2h. In what segment does it have offset BFD2h?

Solution:
physical address = segment * 10h + offset
segment * 10h = physical address - offset
Hence:
physical address = 80FD2h
- offset = BFD2h
75000h
Thus the segment is 7500h

1.3 BIU Registers and Memory Segments

- Within the 1 MB of memory space the 8086/88 defines four 64K-byte memory blocks called the ***code segment***, ***stack segment, data segment, and extra segment***. Each of these blocks of memory is used differently by the processor.
  - The code segment holds the program instruction codes.
  - The data segment stores data for the program.
  - The extra segment is an extra data segment (often used for shared data).
  - The stack segment is used to store interrupt and subroutine return addresses.
- Code segment (CS)
  - points to a memory block where the instructions are located.
  - Next instruction is determined by CS and the EU instruction pointer (IP) register
- Data segment (DS)
  - points to a memory block where program data resides.
  - operands for most instructions are fetched from this segment.
  - Exact data address is determined by a combination of DS and an offset obtained from different sources.
- Stack segment (SS)
  - points to a memory block where stack data resides.
  - The exact address for stack data is obtained as a combination of SS and the EU stack pointer (SP) register.

- The stack is used as a temporary storage
- One of its main usages is to store the memory location for the next instruction to be executed after a program diversion due to a call or interrupting the sequential program operation.
- The SP grows toward lower address within the stack segment.

- Extra segment (ES)
  - points to an extra segment for program data
- The greatest advantage of segmented memory is that programs that reference logical addresses only can be loaded and run anywhere in memory. This is because the logical addresses always range from 00000h to 0FFFFh, independent of the code segment base. **Such programs are said to be relocatable,** meaning that they will run at any location in memory. The requirements for writing relocatable programs are that no references be made to physical addresses, and no changes to the segment registers are allowed.

- Note that the four segments need not be defined separately. Indeed, it is allowable for all four segments to completely overlap (CS = DS = ES = SS).

When two segments overlap it is certainly possible for two different logical addresses to map to the same physical address. This can have disastrous results when the data begins to overwrite the subroutine stack area, or vice versa. For this reason you must be very careful when segments are allowed to overlap.

- The maximum memory used by an 8086 program is 256 K (64K * 4) bytes.

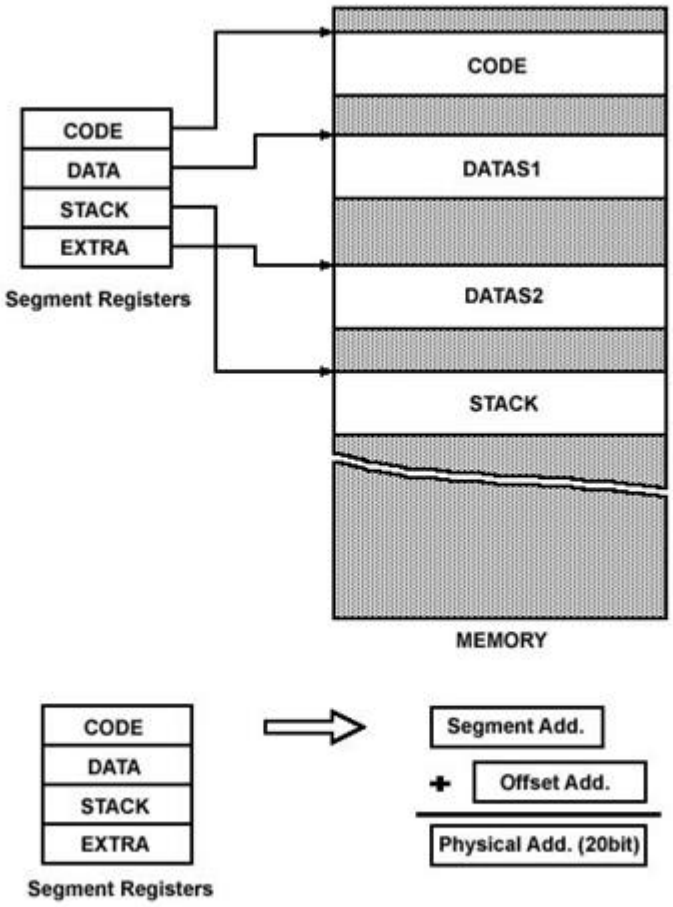

Figure 2: Memory Segments

- As a programmer, you should be aware that memory is partitioned as shown in figure 3.

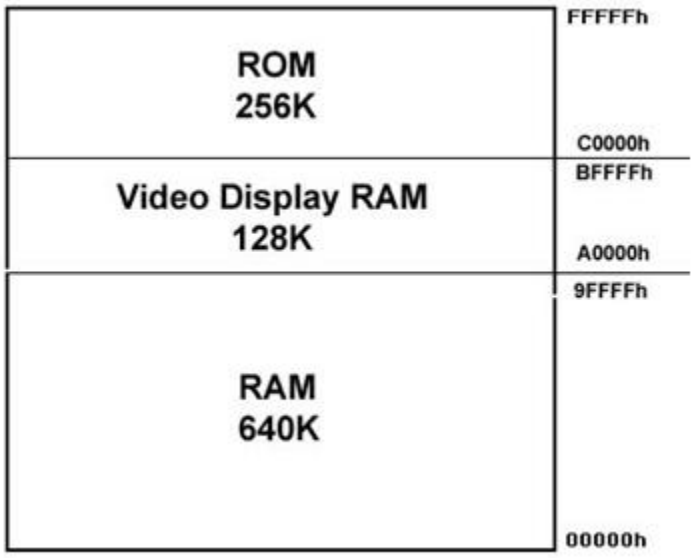

Figure 3: Memory map

- Some memory locations are marked reserved and others dedicated. The dedicated locations are used for processing specific system interrupts and the reset function. Intel has also reserved several locations for future H/W and S/W products. If you make use of these memory locations, you risk incompatibility with these future products.

1.4 Execution Unit

1.4.1 EU Registers

- The EU has nine 16-bit registers: AX, BX, CX, DX, SP, BP, SI, and DI, and the flag register.
- AX, BX, CX, and DX can be used as two 8-bit registers (AH, AL; BH, BL; CH, CL; DH, DL).
- AX(AH|AL): Accumulator
  - implied in many instructions such as I/O, multiplication and division.
- BX(BH|BL): Base register
  - general-purpose register
  - typically used for addressing 8086 memory with DS as a default segment base register for BX
- CX (CH|CL): Counter
  - SHIFT, ROTATE, and LOOP instructions use the contents of CX as a counter
  - Example: Loop ADDRESS
- DX (DH|DL): Data Register
  - used to hold the high 16-bit result (data)
  - implied role in some instructions such as multiplication and division
  - (LOW 16-bit data is contained in AX) after 16 x 16 multiplication or the high 16-bit dividend (data) before a 32/16 division and the 16-bit remainder after the division
- SP (stack pointer) and BP (base pointer) (Pointer registers)

  - used to access data in the stack segment
  - SP contents are automatically updated after PUSH and POP
  - BP contains an offset address in the current SS
- SI (source index) and DI (destination index): Index registers
  - SI and DI are used with the DS and ES, respectively, for source and destination addresses
- flag register
  - EU sets or resets these flags to reflect the results of arithmetic and logic operations
  - 8086 has six one-bit status flags
  - flags may be changed with some commands, e.g. **CLC**
- Instruction Pointer (IP) is similar to the program counter

Flag Register

- CF (carry flag) is set if there is a carry from addition or a borrow from subtraction.
- OF (overflow flag) is set if there is an arithmetic overflow
- SF (sign flag) is set if the most significant bit of the result is one; otherwise, it is zero.
- PF (parity flag) is set if the result has even parity
- ZF (zero flag) is set if the result is zero
- AF (auxiliary carry flag) is set if there is a carry/borrow due to addition/subtraction of the low nibble into the high nibble
- 8086 has three control bits in the flag register
  - the control bits can be changed by the programmer
  - Setting IF (interrupt flag) causes the 8086 to recognize external maskable interrupts; clearing IF disables these interrupts
  - Setting TF (trap flag) puts the 8086 in the single-step mode
  - Setting DF (direction flag) causes string instructions to auto-decrement; clearing DF causes string instructions to auto-increment

1.5 Fetch and Execute

- the organization of the CPU into a separate BIU and EU allows the fetch and execute cycles to overlap.
  - consider what happens when the 8086 or 8088 is first started
    1. The BIU outputs the contents of the instruction pointer register (IP) onto the address bus, causing the selected byte or word to be read into the BIU.
    2. Register IP is incremented by 1 to prepare for the next instruction fetch.
    3. Once inside the BIU, the instruction is passed to the queue. This is a first-in, first-out storage register sometimes likened to a "pipeline".
    4. Assuming that the queue is initially empty, the EU immediately draws this instruction from the queue and begins execution.
    5. While the EU is executing this instruction, the BIU proceeds to fetch a new instruction. Depending on the execution time of the first instruction, the BIU may fill the queue with several new instructions before the EU is ready to draw its next instruction.

- The described architecture is called ***a pipelined architecture***

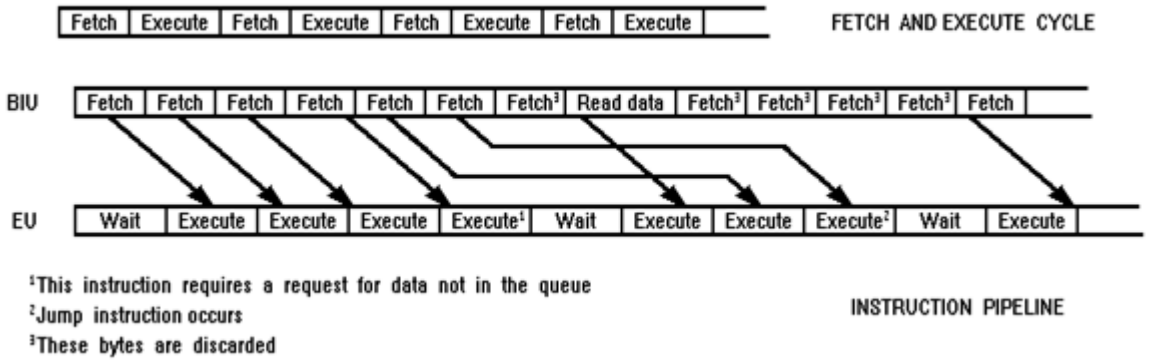

Figure 4: Instruction Pipeline

  - The advantage of this pipelined architecture is that the EU can execute instructions almost continually instead of having to wait for the BIU to fetch a new instruction.
  - There are three conditions that will cause the EU to enter a "wait" mode.
    1. an instruction requires ***access to a memory*** location not in the queue. The BIU must suspend fetching instructions and output the address of this memory location. After waiting for the memory access, the EU can resume executing instruction codes from the queue (and the BIU can resume filling the queue).
    2. instruction to be executed is a ***"jump" instruction.*** In this case control is to be transferred to a new (non-sequential) address. The queue, however, assumes that instructions will always be executed in sequence and thus will be holding the "wrong" instruction codes. The EU must wait while the instruction at the jump address is fetched. Note that any bytes presently in the queue must be discarded (they are overwritten).
    3. Buffer is full due to a slow instruction.

- 8086 reads 16 bits from memory by simultaneously reading an odd-addressed byte and an even-addressed byte. For this reason the 8086 organizes its memory into an even-addressed bank and an odd-addressed bank.
- Due to 8086 memory architecture, all words must begin at an even address. Otherwise, the CPU must perform two memory read cycles: one to fetch the low-order byte and a second to fetch the high-order byte. **This slows down the processor but is transparent to the programmer.**

References

M. RAFIQUZZAMAN, “Fundamentals of Digital Logic and Microcomputer Design,” Fifth Edition.