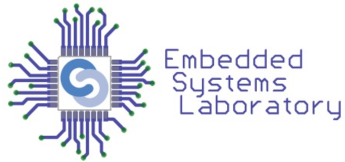
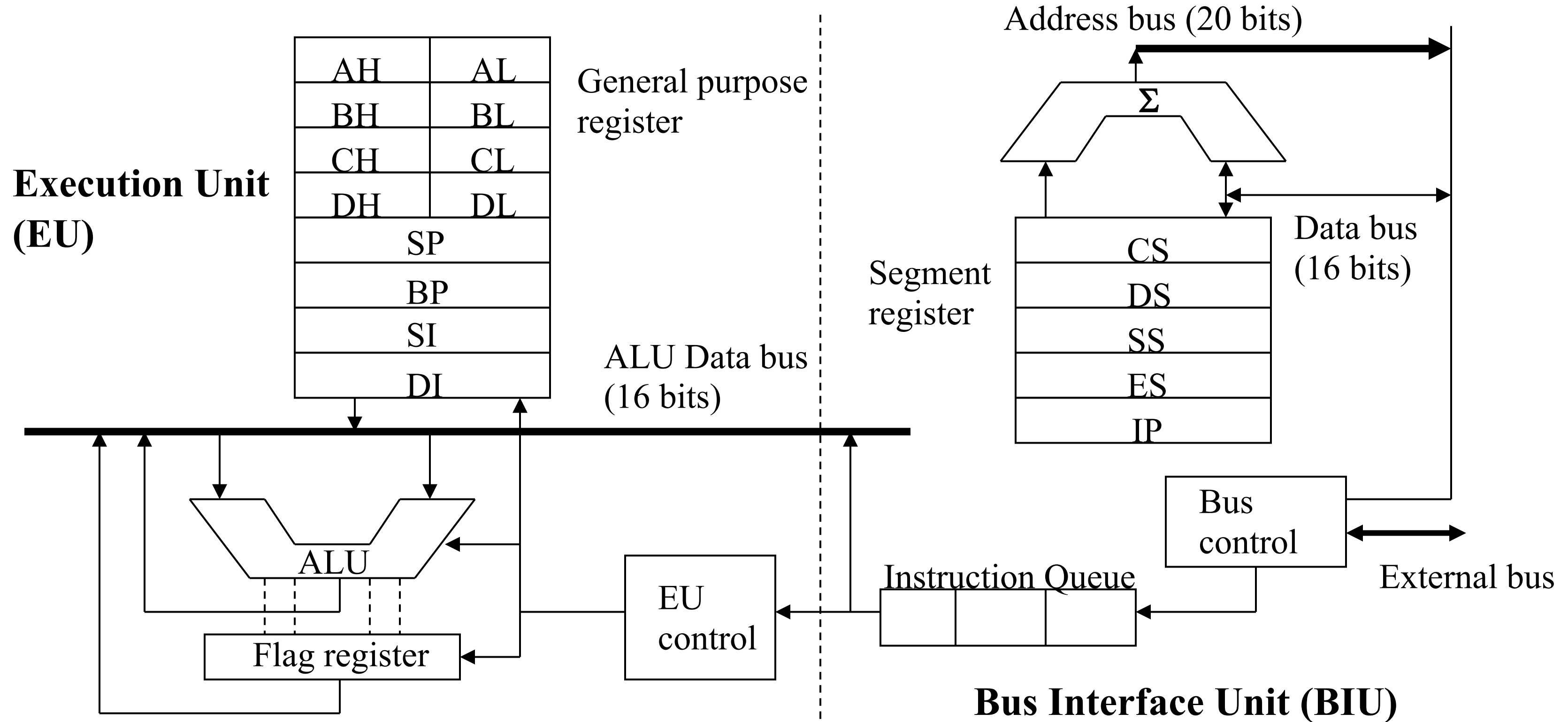


PROIECTAREA CU MICROPROCESOARE

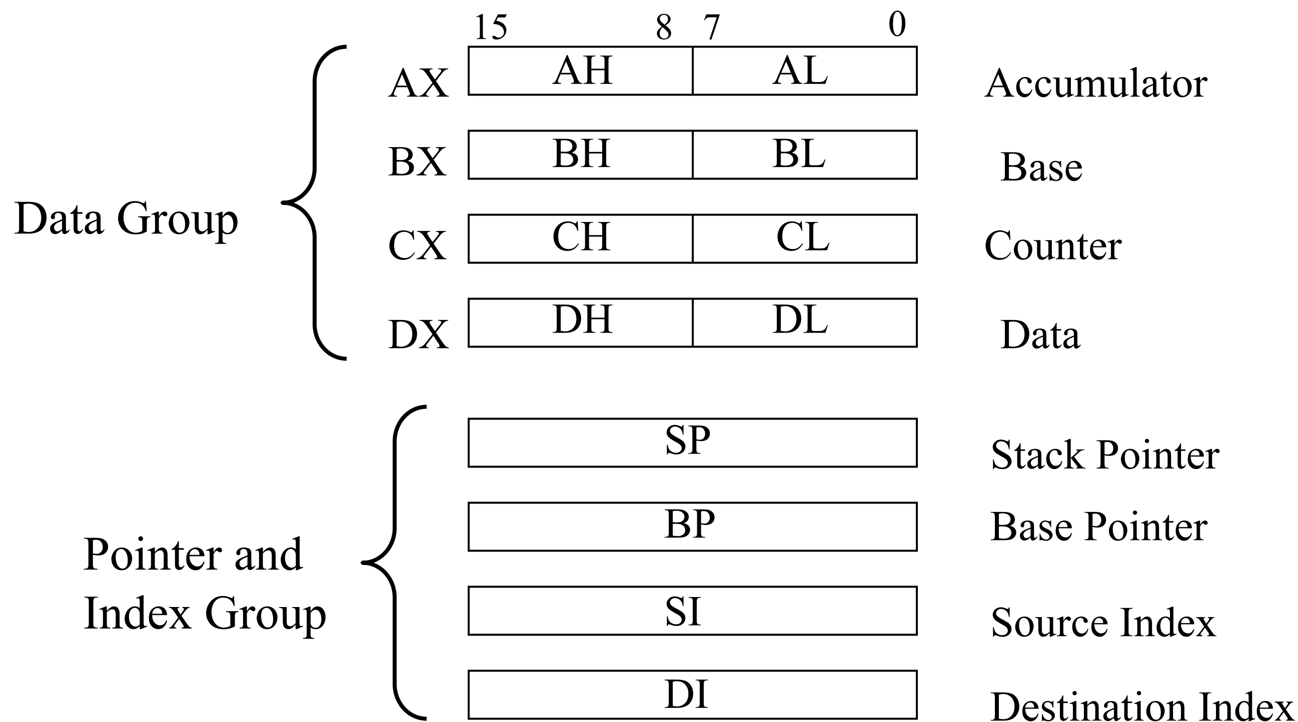
ARHITECTURA X86

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

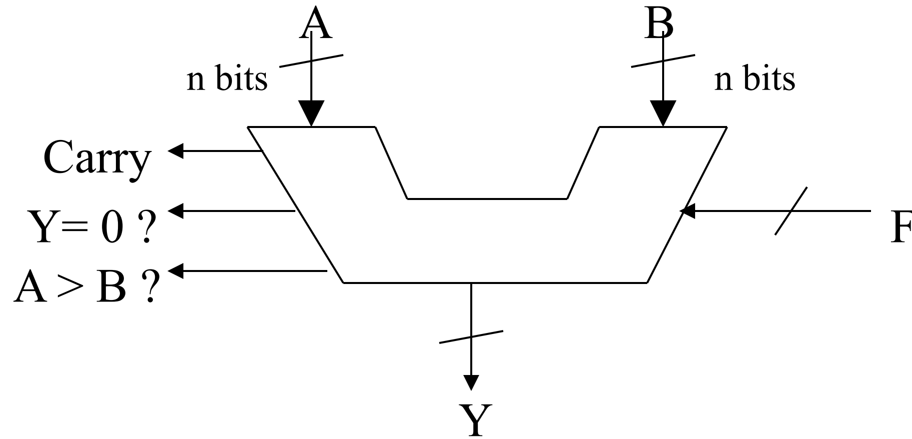
Organization of 8086



General Purpose Registers



Arithmetic Logic Unit (ALU)

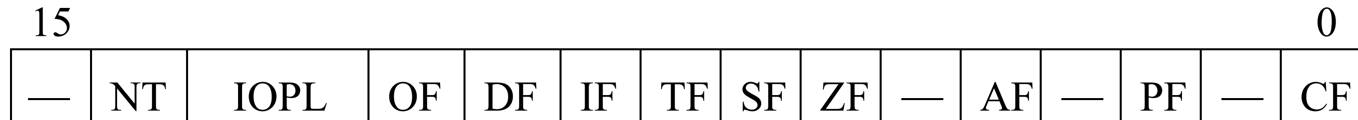


F	Y
0 0 0	A + B
0 0 1	A - B
0 1 0	A - 1
0 1 1	A ***and*** B
1 0 0	A ***or*** B
1 0 1	***not*** A
• • •	• • •

- ➢ Signal F controls which function will be conducted by ALU.
- ➢ Signal F is generated according to the current instruction.
- ➢ Basic arithmetic operations: *addition, subtraction, etc.*
- ➢ Basic logic operations: *and, or, xor, shifting, etc.*

Flag Register

- Flag register contains information reflecting the current status of a microprocessor. It also contains information which controls the operation of the microprocessor.



- Control Flags

IF:	Interrupt enable flag
DF:	Direction flag
TF:	Trap flag

- Status Flags

CF:	Carry flag
PF:	Parity flag
AF:	Auxiliary carry flag
ZF:	Zero flag
SF:	Sign flag
OF:	Overflow flag
NT:	Nested task flag
IOPL:	Input/output privilege level

Instruction Machine Codes

- Instruction machine codes are binary numbers
  - ***For Example:***

1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 ⟹ MOV AL, BL

(1 0 0 0 1 0: MOV; 1 1: Register mode)

- Machine code structure

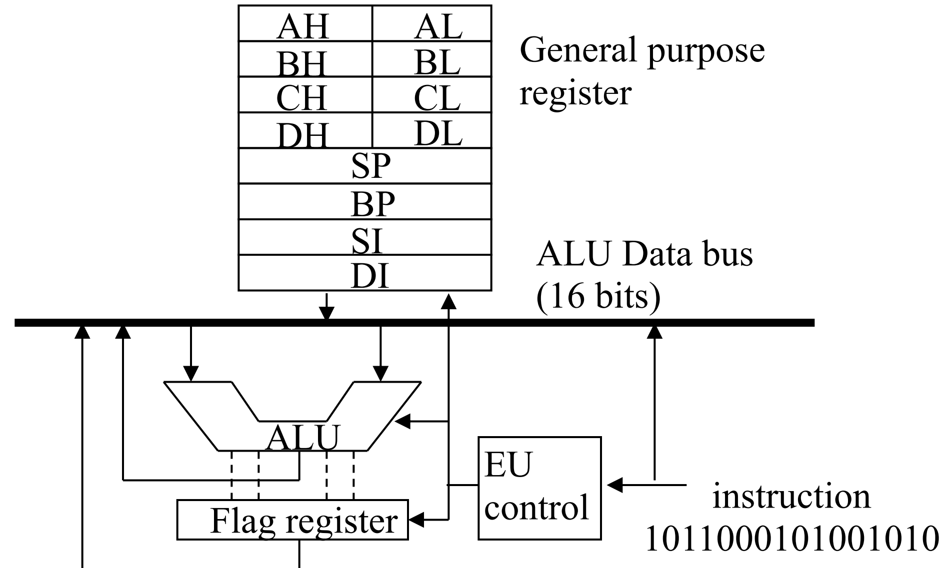
Opcode	Mode	Operand1	Operand2

  - Some instructions do not have operands, or have only one operand
  - Opcode tells what operation is to be performed.
    (EU control logic generates ALU control signals according to Opcode)
  - Mode indicates the type of a instruction: *Register type, or Memory type*
  - Operands tell what data should be used in the operation. Operands can be addresses telling where to get data (or where to store results)

EU Operation

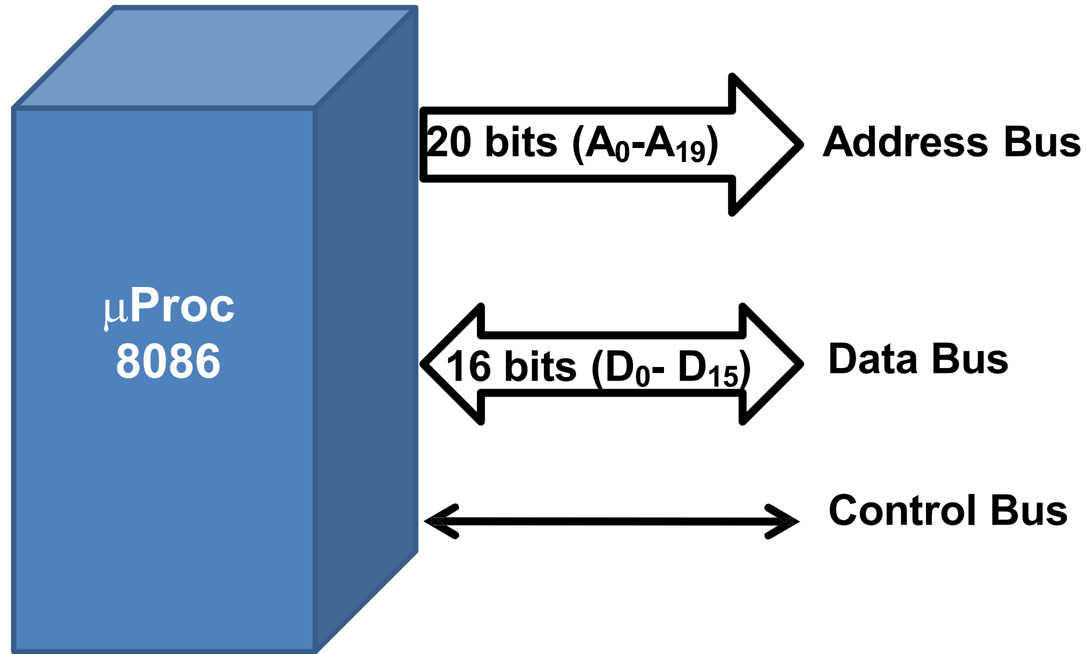
1. Fetch an instruction from instruction queue
2. According to the instruction, EU control logic generates control signals.

   (*This process is also referred to as instruction decoding)*
3. Depending on the control signal, EU performs one of the following operations:
   - An arithmetic operation
   - A logic operation
   - Storing data into a register
   - Moving data from a register
   - Changing flag register



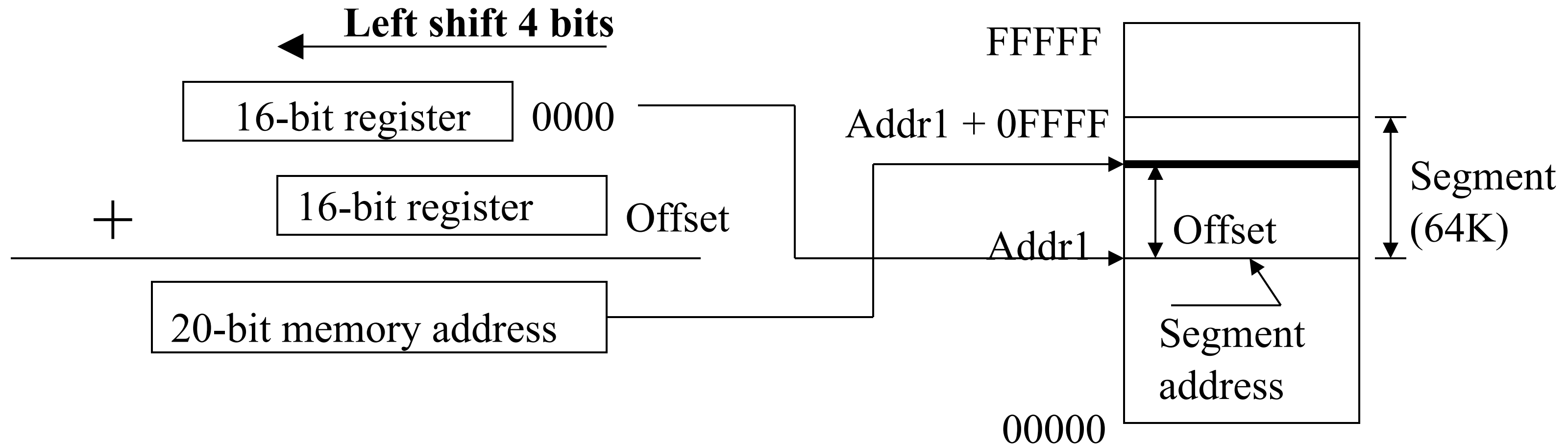
Pointers and Index Registers:

8086-based Systems can access ***2^{20}= 1M*** *memory locations at most*



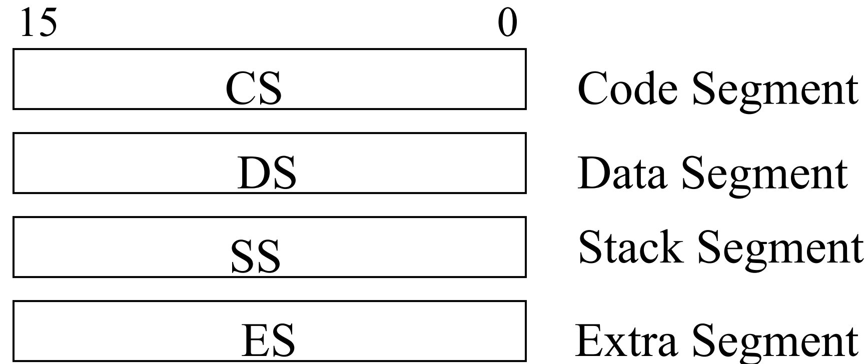
Generating Memory Addresses

- How can a 16-bit microprocessor generate 20-bit memory addresses?



Memory Segmentation

- A segment is a 64KB block of memory starting from any 16-byte boundary
  - For example: 00000, 00010, 00020, 20000, 8CE90, and E0840 are all valid segment addresses
  - The requirement of starting from 16-byte boundary is due to the 4-bit left shifting
- Segment registers in BIU



Segmentation

CS = 1000H
DS = 2000H
SS = 3000H
What will be the actual addresses in memory?

Data Segment and Code Segment can have a complete overlapping. In addition Stack Segment and Extra Segment can have an overlapping.

Code Segment:

20-bit **start** address = CS x10h +0000H
= 10000h

20-bit **end** address = CS x10H +FFFFH
= 1FFFFH

Data Segment :

20-bit **start** address = DS x10h +0000H
= 20000H

20-bit **end** address = DS x10H +FFFFH
= 2FFFFH

Stack Segment :

20-bit **start** address = SS x10h +0000H
= 30000H

20-bit **end** address = SS x10H +FFFFH
= 3FFFFH

The trouble with segments

It is well-known that programming with segmented architectures is really a pain

- In the 8086 you constantly have to make sure segment registers are set up correctly
- What happens if you have data/code that’s more than 64KiB?
- You must then switch back and forth between selector values, which can be really awkward
- Something that can cause complexity also is that two different (selector, offset) pairs can reference the same address
- Example: (a,b) and (a-1, b+16)

There is an interesting on-line article on the topic:

http://world.std.com/~swmcd/steven/rants/pc.html

Why did segmentation survive?

If you code and your data are <64KiB, segments are great

- Otherwise, they are a pain
- Given the horror of segmented programming, one may wonder how come it stuck?
- From the linked article: "*Under normal circumstances, a design so twisted and flawed as the 8086 would have simply been ignored by the market and faded away.*"

But in 1980, Intel was lucky that IBM picked it for the PC!

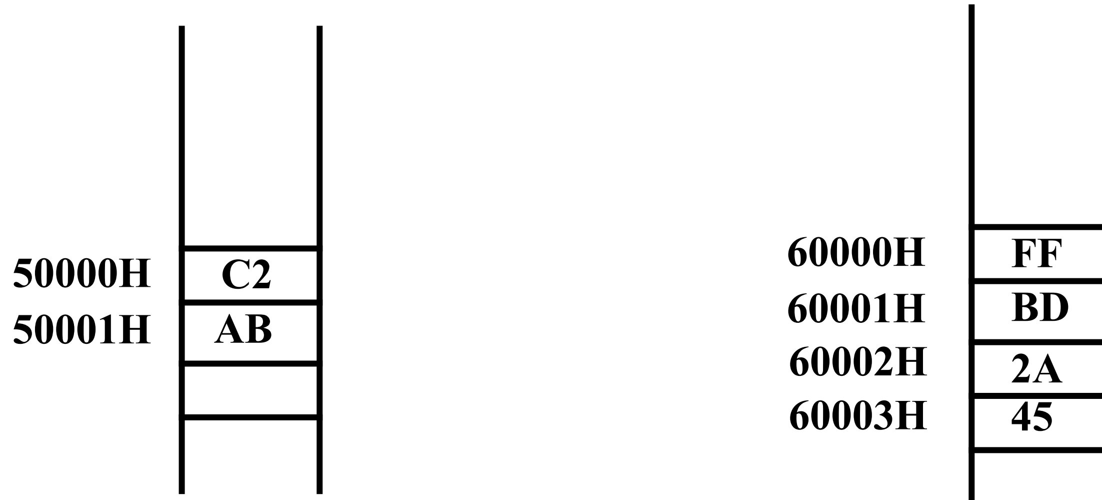
- Not to criticize IBM or anything, but they were also the reason why we got stuck with FORTRAN for so many years :/
- Big companies making "wrong" decisions has impact

Memory storage

Lower byte of word is stored at lower address

The word **ABC2H** stored in the memory starting at 20-bit address 50000H

The double word **452ABDFF** stored in the memory starting at 20-bit address 60000H



Memory Address Calculation

- Segment addresses must be stored in segment registers
- Offset is derived from the combination of pointer registers, the Instruction Pointer (IP), and immediate values
- Examples

	Segment address	0000
+	Offset	
	Memory address	

CS	3	4	8	A	0
IP +		4	2	1	4
Instruction address	3	8	A	B	4

SS	5	0	0	0	0
SP +		F	F	E	0
Stack address	5	F	F	E	0

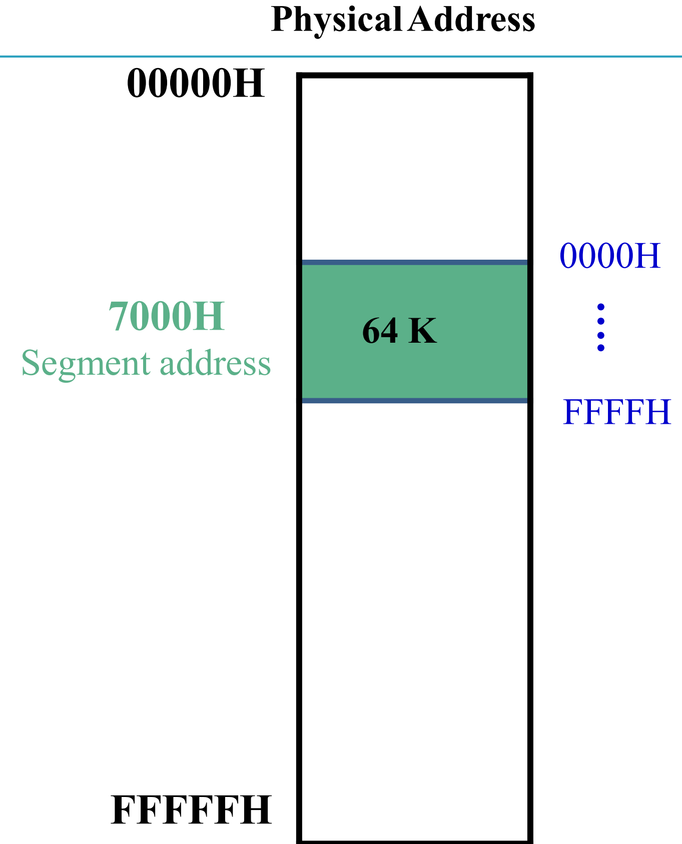
DS	1	2	3	4	0
DI +		0	0	2	2
Data address	1	2	3	6	2

The IP Register

The Instruction Pointer register (IP) contains the offset address of the next sequential instruction to be executed. Thus, the IP register cannot be directly modified.

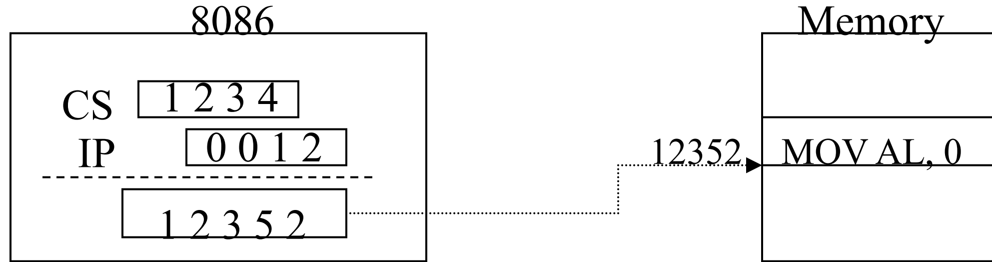
These register descriptions have slowly been introducing us to a new way of addressing memory, called:
segment-offset addressing.

The segment register is used to point to the beginning of any one of the 64K sixteen-byte boundaries



Fetching Instructions

- Where to fetch the next instruction?



- Update IP
  - After an instruction is fetched, Register IP is updated as follows:

$$IP = IP + \text{Length of the fetched instruction}$$

  - For Example: the length of **MOV AL, 0** is 2 bytes. After fetching this instruction, the IP is updated to 0014

Accessing Data Memory

- There is a number of methods to generate the memory address when accessing data memory. These methods are referred to as **Addressing Modes**
- Examples:
  - *Direct addressing:* **MOV AL, [0300H]**

DS	1	2	3	4	0	*(assume DS=1234H)*
		0	3	0	0	
Memory address	1	2	6	4	0	

  - *Register indirect addressing:* **MOV AL, [SI]**

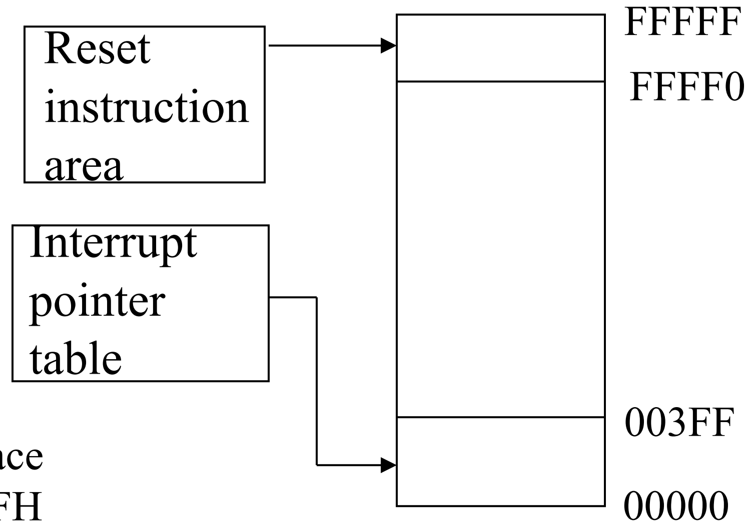
DS	1	2	3	4	0	*(assume DS=1234H)*
		0	3	1	0	*(assume SI=0310H)*
Memory address	1	2	6	5	0	

Reserved Memory Locations

- Some memory locations are reserved for special purposes. Programs should not be loaded in these areas
  - Locations from FFFF0H to FFFFFH are used for system reset code
  - Locations from 00000H to 003FFH are used for the interrupt pointer table
    - It has 256 table entries
    - Each table entry is 4 bytes

$256 \times 4 = 1024$ = memory addressing space

From 00000H to 003FFH



Interrupts

- An interrupt is an event that occurs while the processor is executing a program
- The interrupt temporarily suspends execution of the program and switch the processor to executing a special routine (interrupt service routine)
- When the execution of interrupt service routine is complete, the processor resumes the execution of the original program
- Interrupt classification

Hardware Interrupts	Software Interrupts
— Caused by activating the processor’s interrupt control signals (NMI, INTR)	— Caused by the execution of an INT instruction — Caused by an event which is generated by the execution of a program, such as division by zero

- 8088 can have 256 interrupts

Minimum and Maximum Operation modes

- Intel 8088 (8086) has two operation modes:

Minimum Mode	***Maximum Mode***
— 8088 generates control signals for memory and I/O operations	— It needs 8288 bus controller to generate control signals for memory and I/O operations
— Some functions are not available in minimum mode	— It allows the use of 8087 coprocessor; it also provides other functions
— Compatible with 8085-based systems	