

8086 Microprocessor

History

1959: first IC's

1971: μ P 4004 4Bit Intel (~2000 Tr.)

1972: μ P 8008 8Bit Intel (~5000 Tr.)

1974: μ P 8080 8Bit Intel (~6000 Tr.)

1974: μ P 6800 8Bit Motorola

1975: μ P Z80 8Bit Zilog

1976: μ P 9900 16Bit Texas Instruments

1978: μ P 8086 16Bit Intel (50.000 Tr.)

1982: μ P 8286 16Bit Intel (100.000 Tr.)

1985: μ P 80386 16Bit Intel (300.000 Tr.)

1989: μ P 80486 16Bit Intel (10^6 Tr.)

1993: μ P 80586 16Bit Intel ($5 \cdot 10^6$ Tr.)

Name	Date	Transistors	Microns	Clock speed	Data width	MIPS
8080	1974	6,000	6	2 MHz	8 bits	0.64
8088	1979	29,000	3	5 MHz	16 bits 8-bit bus	0.33
80286	1982	134,000	1.5	6 MHz	16 bits	1
80386	1985	275,000	1.5	16 MHz	32 bits	5
80486	1989	1,200,000	1	25 MHz	32 bits	20
Pentium	1993	3,100,000	0.8	60 MHz	32 bits 64-bit bus	100
Pentium II	1997	7,500,000	0.35	233 MHz	32 bits 64-bit bus	~300
Pentium III	1999	9,500,000	0.25	450 MHz	32 bits 64-bit bus	~510
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits 64-bit bus	~1,700

...

History of Intel's Microprocessors

- Intel 4004
 - 1971, 4-bit
- Intel 8008
 - 1972, 8-bit
 - Originally designed for Datapoint Corp. as a CRT display controller
- Intel 8080
 - 1974, April - Altair 8800, 1975, MITS(256 bytes of Mem, \$395)
 - Apple II -- Steve Jobs and Steve Wozniak 1976, Apple 사 설립
 - Bill Gates and a fellow student : BASIC, 1975 --> Microsoft
- Intel 8086/8088
 - 1978, 16 bit: 8088, 1979, 8-bit external bus
 - IBM PC ; 1981
 - 29,000 Trs
- Intel 80286
 - 1982, 16-bit architecture
 - 24-bit addressing, memory protection and virtual memory
 - 16 MB of physical MEM and 1 GB of virtual mem
 - 130,000 Trs onto a single chip
 - IBM PC/AT in 1984, IBM PS/2 Model 50 and 60

- Intel 80386
 - 1985, 32 bits
 - 3~5 MIPS (7 MIPS on the 25 MHz chip)
 - memory paging and enhanced I/O permission features
 - 4GB programming model
- Intel 80486
 - 1989 Spring COMDEX show -> 1990 June : actual release
 - 1,200,000 Trs
 - 386+387+8K data and instruction cache, paging and MMU
- Pentium
 - 1993
 - 110 MIPS on 66 Mhz Chip
 - 16 KB on-chip cache and 64 bit data bus
 - superscalar technology (two instructions/clock)
 - 3.1 million transistors

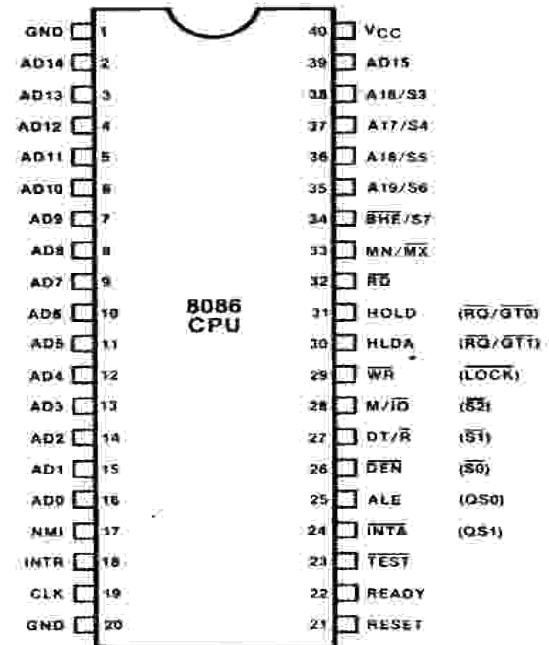
- Pentium Pro
 - 1995, Superscalar(three-way issue)
 - 5.5 million Trs in the CPU core + 15.5 million Trs in the secondary cache
8K data, 8K instr cache
 - 256 KB SRAM secondary cache
 - 200 SPECint92 at 133 MHz
 - 2.9 V, 0.6 micron BICMOS
- Pentium II
 - Pentium Pro + MMX, 1997
 - 233, 266, upto 450 MHz
 - 7.5 million Trs in CPU
 - 512KB in secondary cache
- Pentium III
 - 1999
 - Pentium Pro + MMX + Internet Streaming SIMD Instructions
 - 0.25 micron, 9.5 million Trs
 - 600 MHz, 550 MHz,...
 - 32 K(16K/16K) non-blocking level 1 cache

8086 Microprocessor

Common Signals		
Name	Function	Type
AD15-AD0	Address/Data Bus	Bidirectional, 3-State
A19/S6-A16/S3	Address/Status	Output, 3-State
BHE/S7	Bus High Enable/Status	Output, 3-State
MN/MX	Minimum/Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output, 3-State
$\overline{TEST}$	Wait On Test Control	Input
READY	Wait State Control	Input
RESET	System Reset	Input
NMI	Non-Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
VCC	+5V	Input
GND	Ground	

Minimum Mode Signals (MN/MX = VCC)		
Name	Function	Type
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{WR}$	Write Control	Output, 3-State
M/ $\overline{IO}$	Memory/I/O Control	Output, 3-State
DT/ $\overline{R}$	Data Transmit/Receive	Output, 3-State
$\overline{DEN}$	Data Enable	Output, 3-State
ALE	Address Latch Enable	Output
INTA	Interrupt Acknowledge	Output

Maximum Mode Signals (MN/MX = GND)		
Name	Function	Type
$\overline{RQ}/\overline{GT1}, \overline{D}$	Request/Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3-State
S2-S0	Bus Cycle Status	Output, 3-State
QS1, QS0	Instruction Queue Status	Output

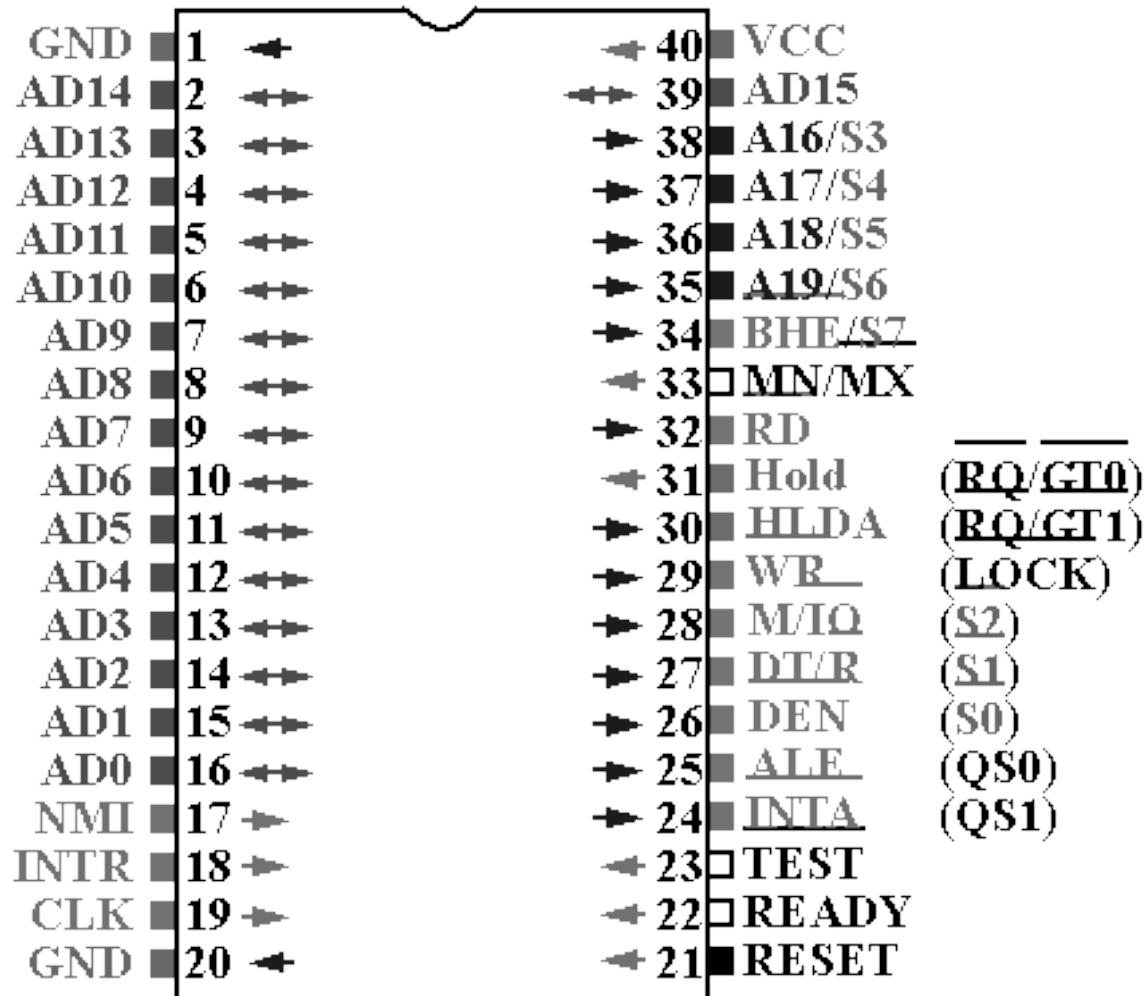


MAXIMUM MODE PIN FUNCTIONS (e.g., LOCK) ARE SHOWN IN PARENTHESES

Figure 4-1. 8086 Pin Definitions

8086 CPU

MIN MODE (MAX MODE)



16-bit Arithmetic Logic Unit

- 16-bit data bus (8088 has 8-bit data bus)

- 20-bit address bus - $2^{20} = 1,048,576 = 1 \text{ meg}$

8086 Architecture

- The 8086 has two parts, the **Bus Interface Unit (BIU)** and the **Execution Unit (EU)**.
- The BIU fetches instructions, reads and writes data, and computes the 20-bit address.
- The EU decodes and executes the instructions using the 16-bit ALU.
- The BIU contains the following registers:

IP - the Instruction Pointer
CS - the Code Segment Register
DS - the Data Segment Register
SS - the Stack Segment Register
ES - the Extra Segment Register

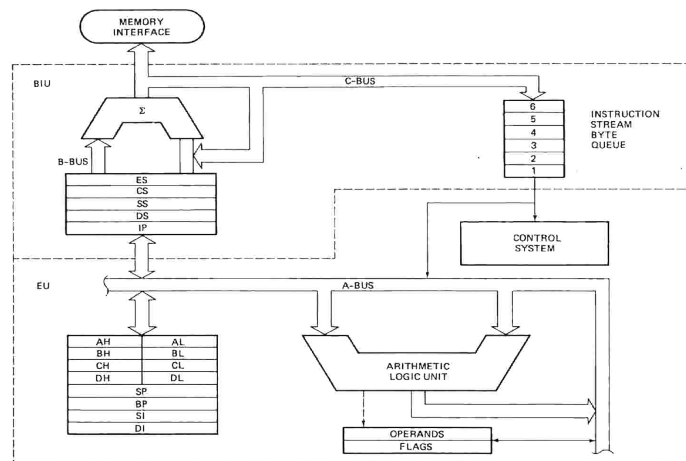


FIGURE 2-7 8086 internal block diagram. (Intel Corp.)

The BIU fetches instructions using the CS and IP, written CS:IP, to construct the 20-bit address. Data is fetched using a segment register (usually the DS) and an effective address (EA) computed by the EU depending on the addressing mode.

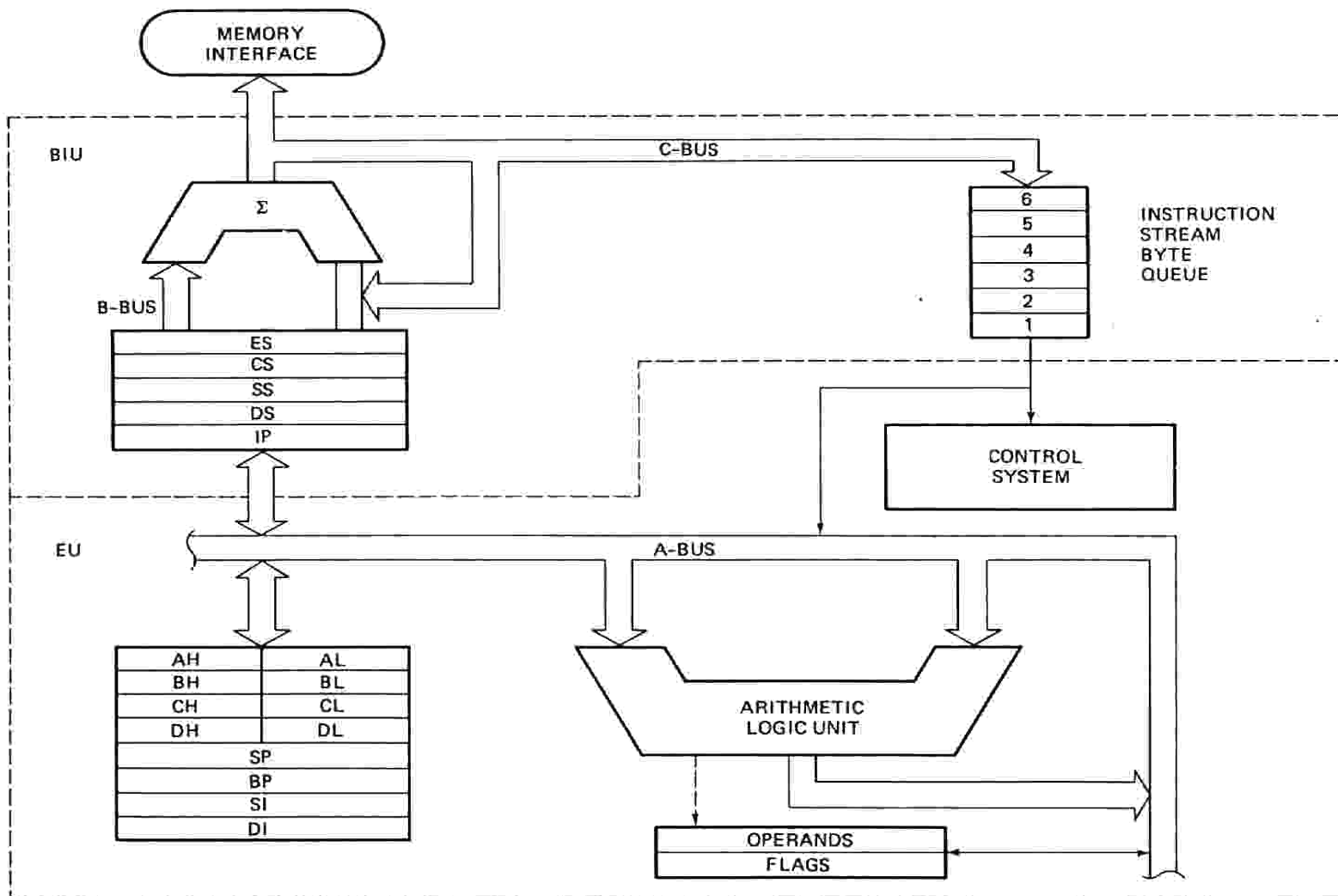
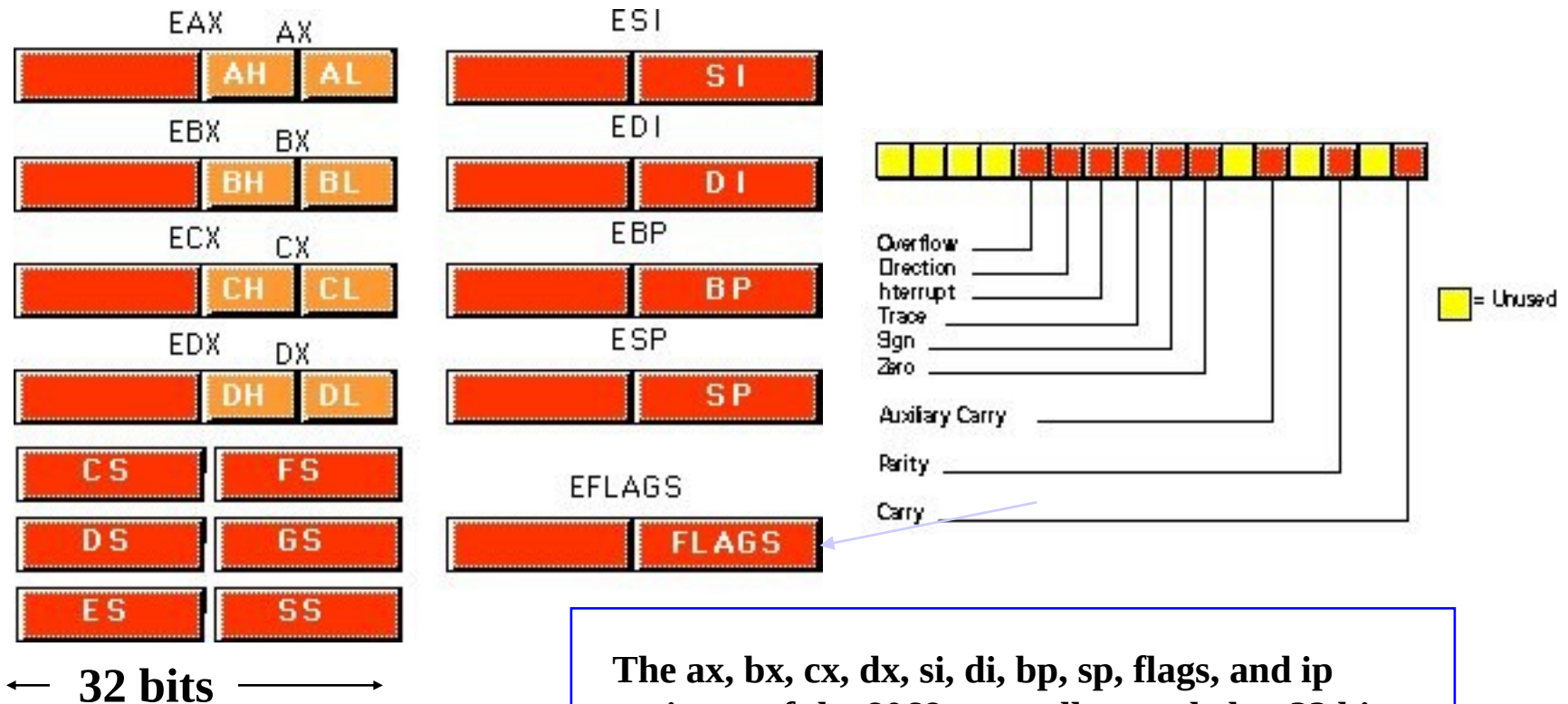


FIGURE 2-7 8086 internal block diagram. (Intel Corp.)

80386/80486/Pentium Registers



The ax, bx, cx, dx, si, di, bp, sp, flags, and ip registers of the 8068 were all extended to 32 bits.

The 80386 calls these new 32 bit versions EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EFLAGS, and EIP

Memory segments can be as large as 4GB

8086 Programmer's Model

BIU registers
(20 bit adder)

ES
CS
SS
DS
IP

Extra Segment
Code Segment
Stack Segment
Data Segment
Instruction Pointer

AX
BX
CX
DX

AH	AL
BH	BL
CH	CL
DH	DL
SP	
BP	
SI	
DI	
FLAGS	

Accumulator
Base Register
Count Register
Data Register
Stack Pointer
Base Pointer
Source Index Register
Destination Index Register

8086/88 internal registers 16 bits (2 bytes each)

AX	AH	AL	Accumulator	} Data group
BX	BH	BL	Base	
CX	CH	CL	Count	
DX	DH	DL	Data	

AX, BX, CX and DX are two bytes wide and each byte can be accessed separately

SP	Stack pointer	} Pointer and index group
BP	Base pointer	
SI	Source index	
DI	Destination index	
IP	Instruction pointer	

These registers are used as memory pointers.

Flags _H	Flags _L	Status and control flags
--------------------	--------------------	--------------------------

Flags will be discussed later

ES	Extra	} Segment group
CS	Code	
DS	Data	
SS	Stack	

Segment registers are used as base address for a segment in the 1 M byte of memory

The 8086/8088 Microprocessors: Registers

- Registers

- Registers are in the CPU and are referred to by specific names
- Data registers
 - Hold data for an operation to be performed
 - There are 4 data registers (AX, BX, CX, DX)
- Address registers
 - Hold the address of an instruction or data element
 - Segment registers (CS, DS, ES, SS)
 - Pointer registers (SP, BP, IP)
 - Index registers (SI, DI)
- Status register
 - Keeps the current status of the processor
 - On an IBM PC the status register is called the FLAGS register
- In total there are fourteen 16-bit registers in an 8086/8088

Data Registers: AX, BX, CX, DX

- Instructions execute faster if the data is in a register
- AX, BX, CX, DX are the data registers
- Low and High bytes of the data registers can be accessed separately
 - AH, BH, CH, DH are the high bytes
 - AL, BL, CL, and DL are the low bytes
- Data Registers are general purpose registers but they also perform special functions
- AX
 - Accumulator Register
 - Preferred register to use in arithmetic, logic and data transfer instructions because it generates the shortest Machine Language Code
 - Must be used in multiplication and division operations
 - Must also be used in I/O operations

- **BX**
 - Base Register
 - Also serves as an address register
 - Used in array operations
 - Used in Table Lookup operations (XLAT)
- **CX**
 - Count register
 - Used as a loop counter
 - Used in shift and rotate operations
- **DX**
 - Data register
 - Used in multiplication and division
 - Also used in I/O operations

Pointer and Index Registers

- Contain the offset addresses of memory locations
- Can also be used in arithmetic and other operations
- **SP: Stack pointer**
 - Used with SS to access the stack segment
- **BP: Base Pointer**
 - Primarily used to access data on the stack
 - Can be used to access data in other segments
- **SI: Source Index register**
 - is required for some string operations
 - When string operations are performed, the SI register points to memory locations in the data segment which is addressed by the DS register. Thus, SI is associated with the DS in string operations.

- **DI: Destination Index register**
 - is also required for some string operations.
 - When string operations are performed, the DI register points to memory locations in the data segment which is addressed by the ES register. Thus, DI is associated with the ES in string operations.
- The SI and the DI registers may also be used to access data stored in arrays

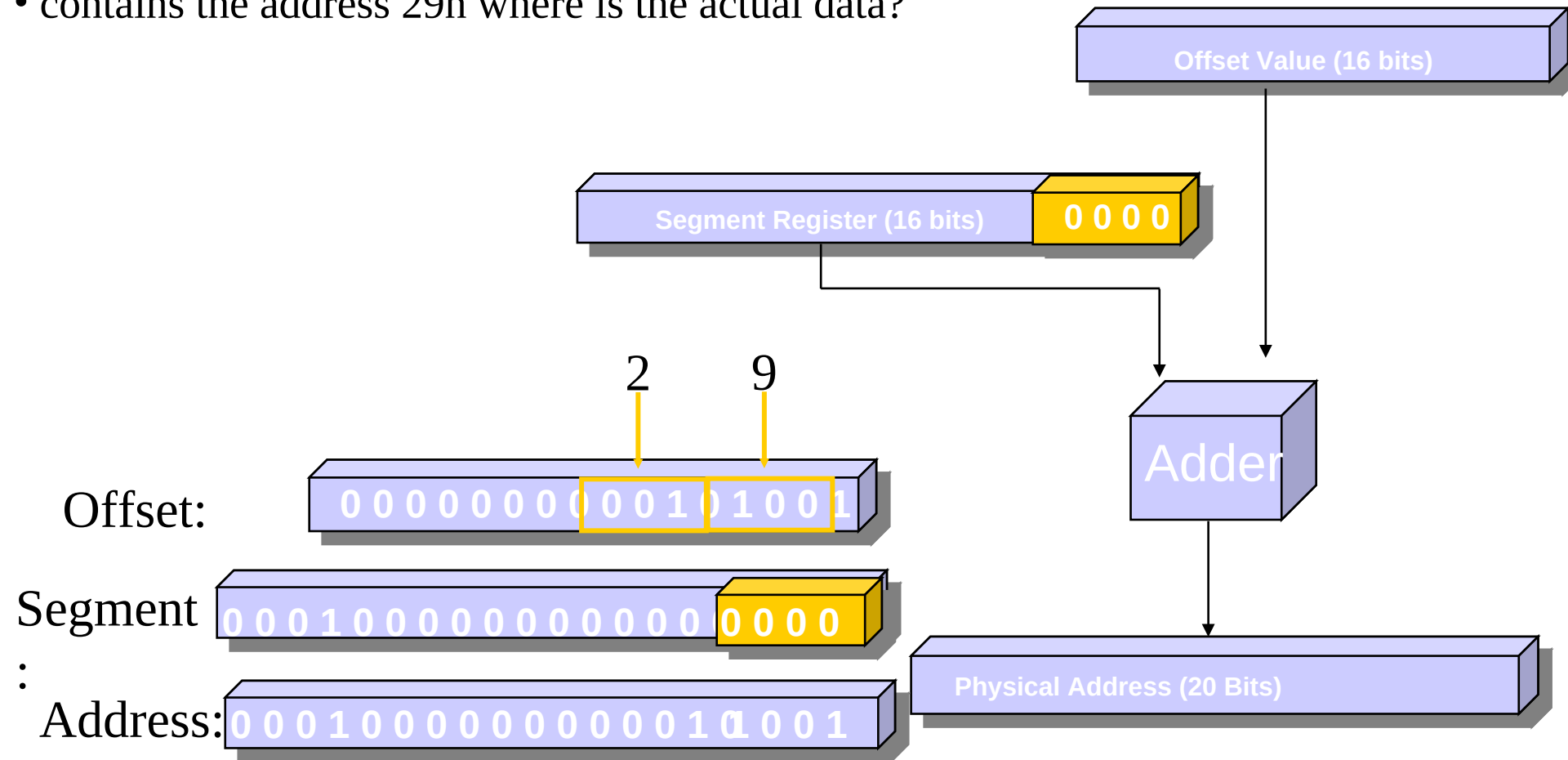
Segment Registers - CS, DS, SS and ES

- Are Address registers
- Store the memory addresses of instructions and data
- **Memory Organization**
 - Each byte in memory has a 20 bit address starting with 0 to $2^{20}-1$ or 1 meg of addressable memory
 - Addresses are expressed as 5 hex digits from 00000 - FFFFF
 - Problem: But 20 bit addresses are **TOO BIG** to fit in 16 bit registers!
 - Solution: Memory Segment
 - Block of 64K (65,536) consecutive memory bytes
 - A segment number is a 16 bit number
 - Segment numbers range from 0000 to FFFF
 - Within a segment, a particular memory location is specified with an offset
 - An offset also ranges from 0000 to FFFF

Memory Address Generation

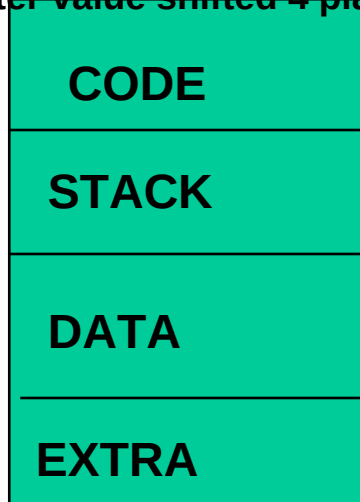
- The BIU has a dedicated adder for determining physical memory addresses

- If the data segment starts at location 1000h and a data reference
- contains the address 29h where is the actual data?



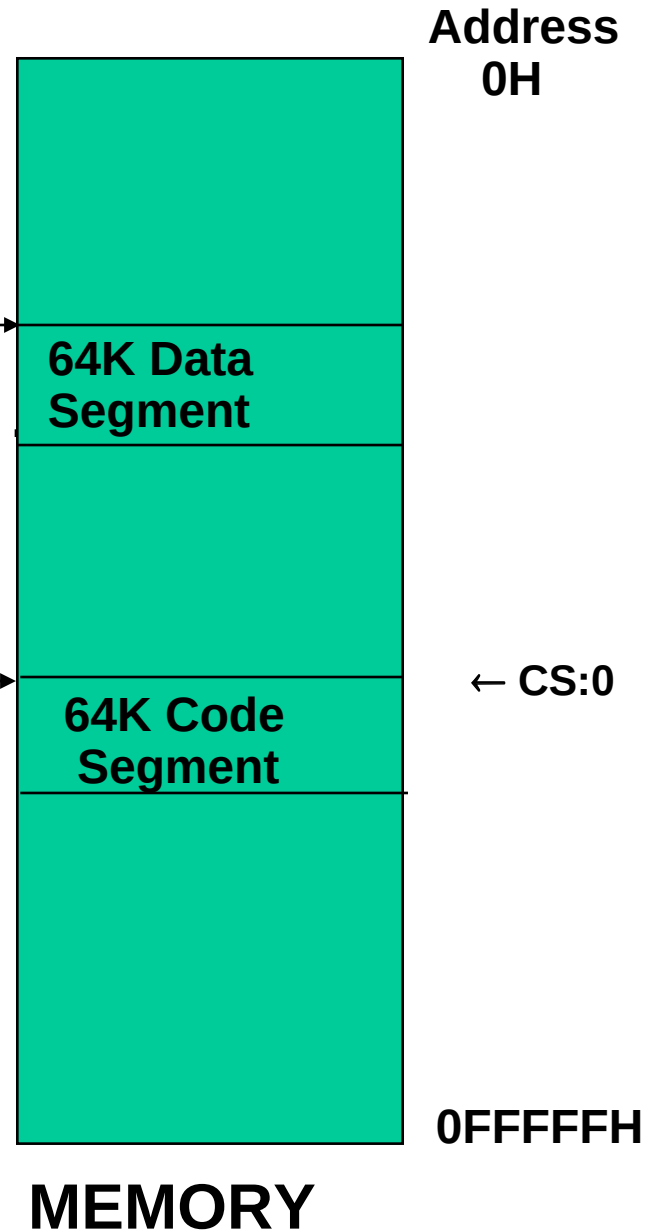
Segments

Segment Starting address is segment register value shifted 4 places to the left.

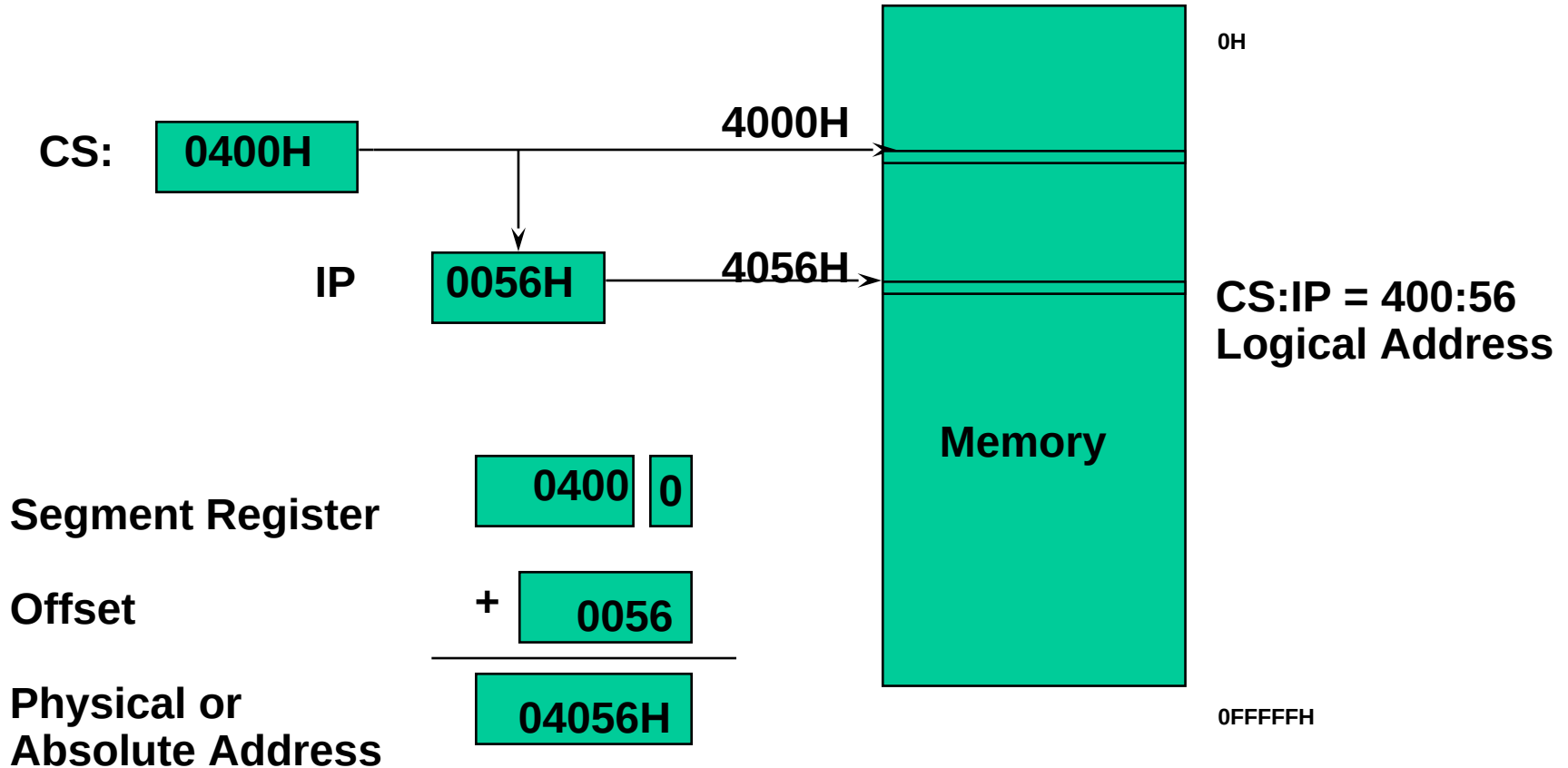


**Segment
Registers**

Segments are $\leq 64K$,
can overlap, start at an address
that ends in 0H.



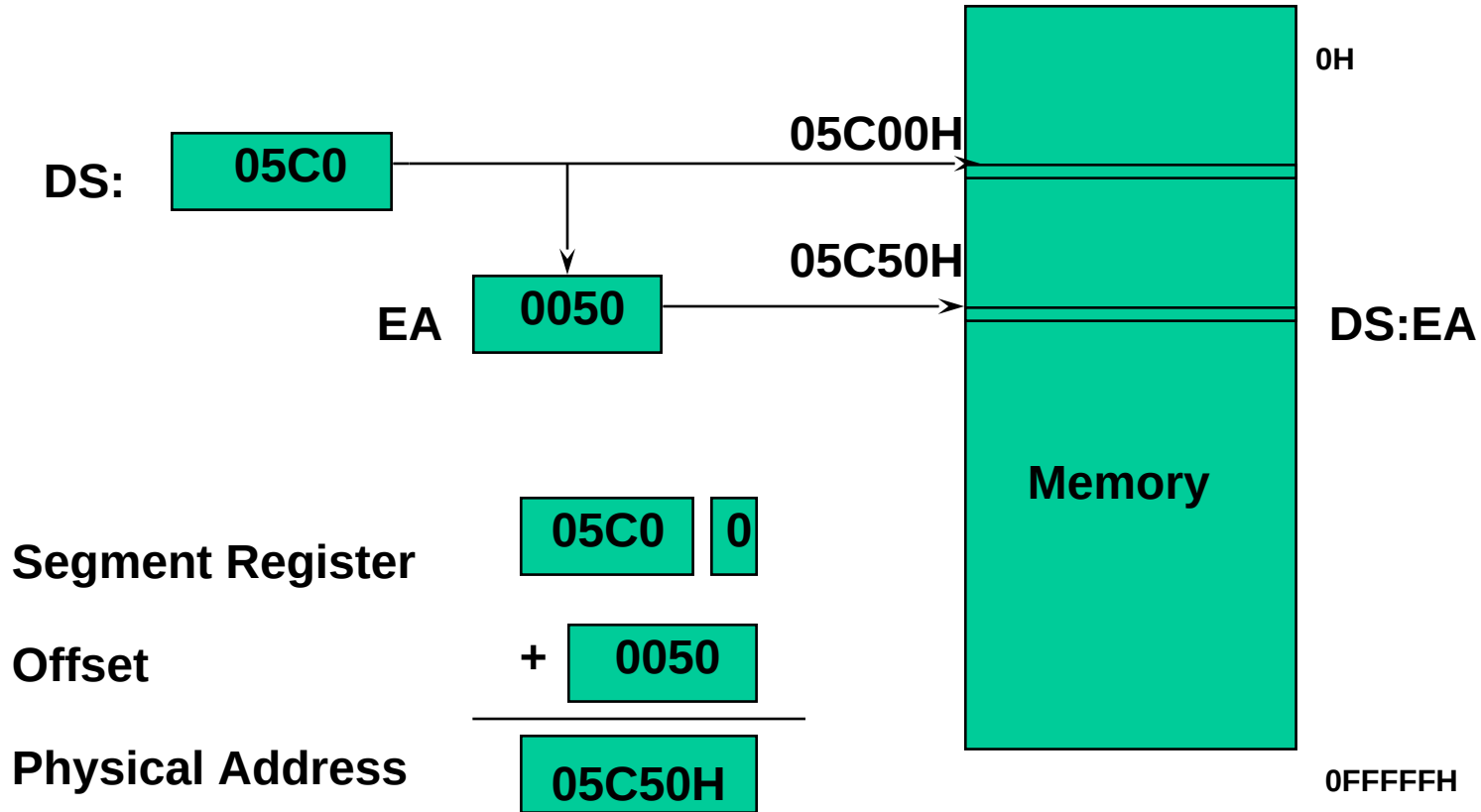
The Code Segment



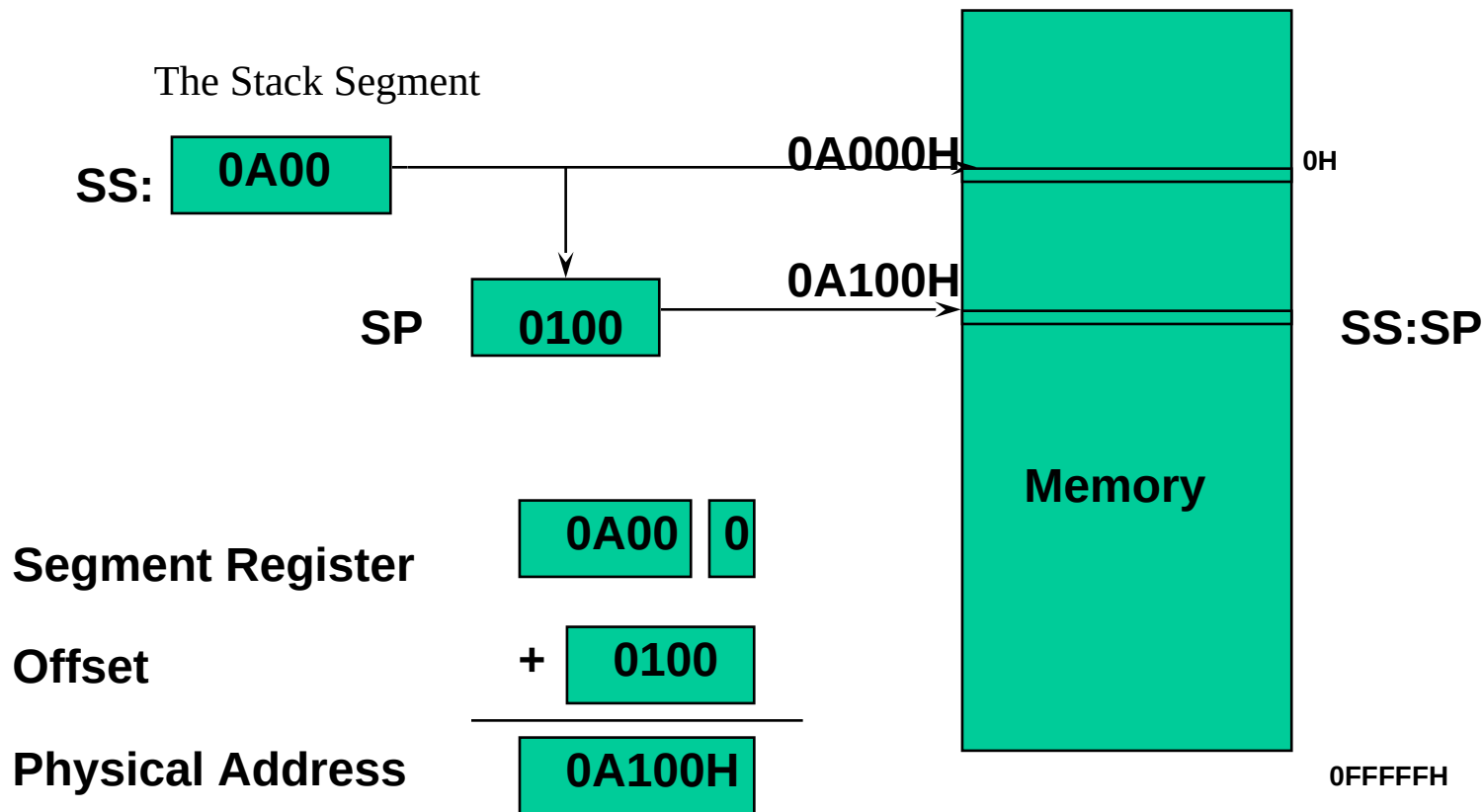
The offset is the distance in bytes from the start of the segment.
The offset is given by the IP for the Code Segment.
Instructions are always fetched with using the CS register.

The physical address is also called the absolute address.

The Data Segment



Data is usually fetched with respect to the DS register.
The effective address (EA) is the offset.
The EA depends on the addressing mode.



The offset is given by the SP register.

The stack is always referenced with respect to the stack segment register.

The stack grows toward decreasing memory locations.

The SP points to the last or top item on the stack.

- Logical Address is specified as **segment:offset**
- Physical address is obtained by shifting the segment address 4 bits to the left and adding the offset address
- Thus the physical address of the logical address A4FB:4872 is $A4FB0 + \underline{4872} = A9822$

Example

- If DS=7FA2H and the offset is 438EH

a) Calculate the physical address

$$7FA20 + 438E = 83DAE$$

b) calculate the lower range

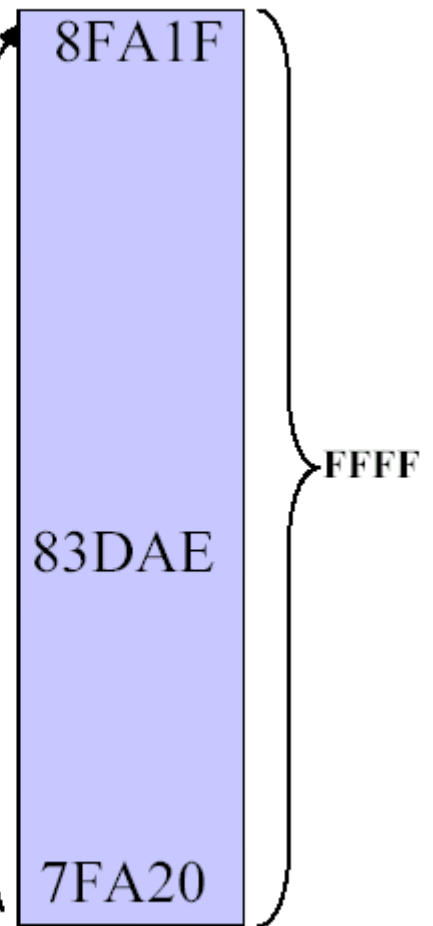
$$7FA20 + 0000 = 7FA20$$

c) Calculate the upper range of the data segment

$$7FA20 + FFFF = 8FA1F$$

d) Show the logical Address

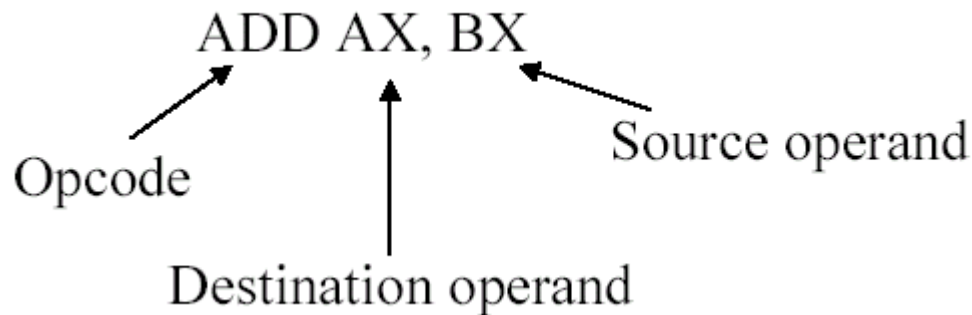
7FA2:438E



	LOGICAL ADDRESS		PHYSICAL ADDRESS	
Segment Number	Beginning Address	Last Address	Beginning Address	Last Address
0000	0000:0000	0000:FFFF	00000	0FFFF
0001	0001:0000	0001:FFFF	00010	1000F
0002	0002:0000	0002:FFFF	00020	1001F
0003	0003:0000	0003:FFFF	00030	1002F
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
FFFF	FFFF:0000	FFFF:FFFF	FFFF0	0FFEF

Software

- The sequence of commands used to tell a microcomputer what to do is called a program,
- Each command in a program is called an instruction
- 8088 understands and performs operations for 117 basic instructions
- The native language of the IBM PC is the machine language of the 8088
- A program written in machine language is referred to as machine code
- In 8088 assembly language, each of the operations is described by alphanumeric symbols instead of just 0s or 1s.



Instructions

`LABEL: INSTRUCTION ; COMMENT`


Address identifier


Does not generate any machine code

Ex. `START: MOV AX,BX ; copy BX into AX`

- There is a one-to-one relationship between assembly and machine language instructions
- What is found is that a compiled machine code implementation of a program written in a high-level language results in inefficient code
 - More machine language instructions than an assembled version of an equivalent handwritten assembly language program
- Two key benefits of assembly language programming
 - It takes up less memory
 - It executes much faster

Converting Assembly Language Instructions to Machine Code

- An instruction can be coded with 1 to 6 bytes
- Byte 1 contains three kinds of information
 - Opcode field (6 bits) specifies the operation such as add, subtract, or move
 - Register Direction Bit (D bit)
 - Tells the register operand in REG field in byte 2 is source or destination operand
 - 1: destination
 - 0: source
 - Data Size Bit (W bit)
 - Specifies whether the operation will be performed on 8-bit or 16-bit data
 - 0: 8 bits
 - 1: 16 bits
- Byte 2 has two fields
 - Mode field (MOD)
 - Register field (REG)
 - Register/memory field (R/M field)

Continued

- REG field is used to identify the register for the first operand

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

- 2-bit MOD field and 3-bit R/M field together specify the second operand

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

(a)

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

(b)

Examples

- MOV BL,AL
- Opcode for MOV = 100010
- We'll encode AL so
 - D = 0 (AL source operand)
- W bit = 0 (8-bits)
- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011

opcode	D	W	MOD	REG	R/M
--------	---	---	-----	-----	-----

MOV AL,BL = 10001000 11000011 = 88 C3h

ADD AX,[SI] = 00000011 00000100 = 03 04 h

ADD [BX],[DI] + 1234h, AX = 00000001 10000001 ____ h
= 01 81 34 12 h

MOV Instruction

- MOV destination,source
 - 8 bit moves
 - MOV CL,55h
 - MOV DL,CL
 - MOV BH,CL
 - Etc.
 - 16 bit moves
 - MOV CX,468Fh
 - MOV AX,CX
 - MOV BP,DI
 - Etc.

MOV Instruction

- Data can be moved among all registers but data cannot be moved directly into the segment registers (CS,DS,ES,SS).
 - To load as such, first load a value into a nonsegment register and then move it to the segment register

```
MOV AX,2345h
```

```
MOV DS,AX
```

- Moving a value that is too large into a register will cause an error

```
MOV BL,7F2h      ; illegal
```

```
MOV AX,2FE456h   ; illegal
```

- If a value less than FFh is moved into a 16 bit register. The rest of the bits are assumed to be all zeros.

```
MOV BX,5          ; BX = 0005 with BH = 00 and BL = 05
```

ADD Instruction

- ADD destination,source
- The ADD instruction tells the CPU to add the source and destination operands and put out the results in the destination

MOV AL,25H

MOV BL,34h

ADD AL,BL ; (AL should read 59h once the instruction is executed)

MOV DH,25H

ADD DH,34h ; (AL should read 59h once the instruction is executed)

Immediate operand



- Assume that a program is written to add 5 bytes of data 25h, 12h, 15h, 1Fh, and 2Bh.
- One way to do it

```
MOV AL,00h
ADD AL, 25h
ADD AL, 12h
ADD AL,15h
ADD AL,1Fh
ADD AL,2Bh
```
- Data and code are mixed in the instructions here
- The problem with it is if the data changes, the code must be searched for every place the data is included and data retyped.
- It is a good idea then to set aside an area of memory strictly for data

Data Segment

- The data is first placed in the memory locations
 - DS:0200 = 25h
 - DS:0201 = 12h
 - DS:0202 = 15h
 - DS:0203 = 1Fh
 - DS:0204 = 2Bh
- Then the program is written as

```
MOV AL,0
ADD AL,[0200] ; bracket means add the contents of DS:0200 to AL
ADD AL,[0201]
ADD AL,[0202]
ADD AL,[0203]
ADD AL,[0204]
```
- If the data is stored at a different offset address, say 450 h, the program need to be rewritten

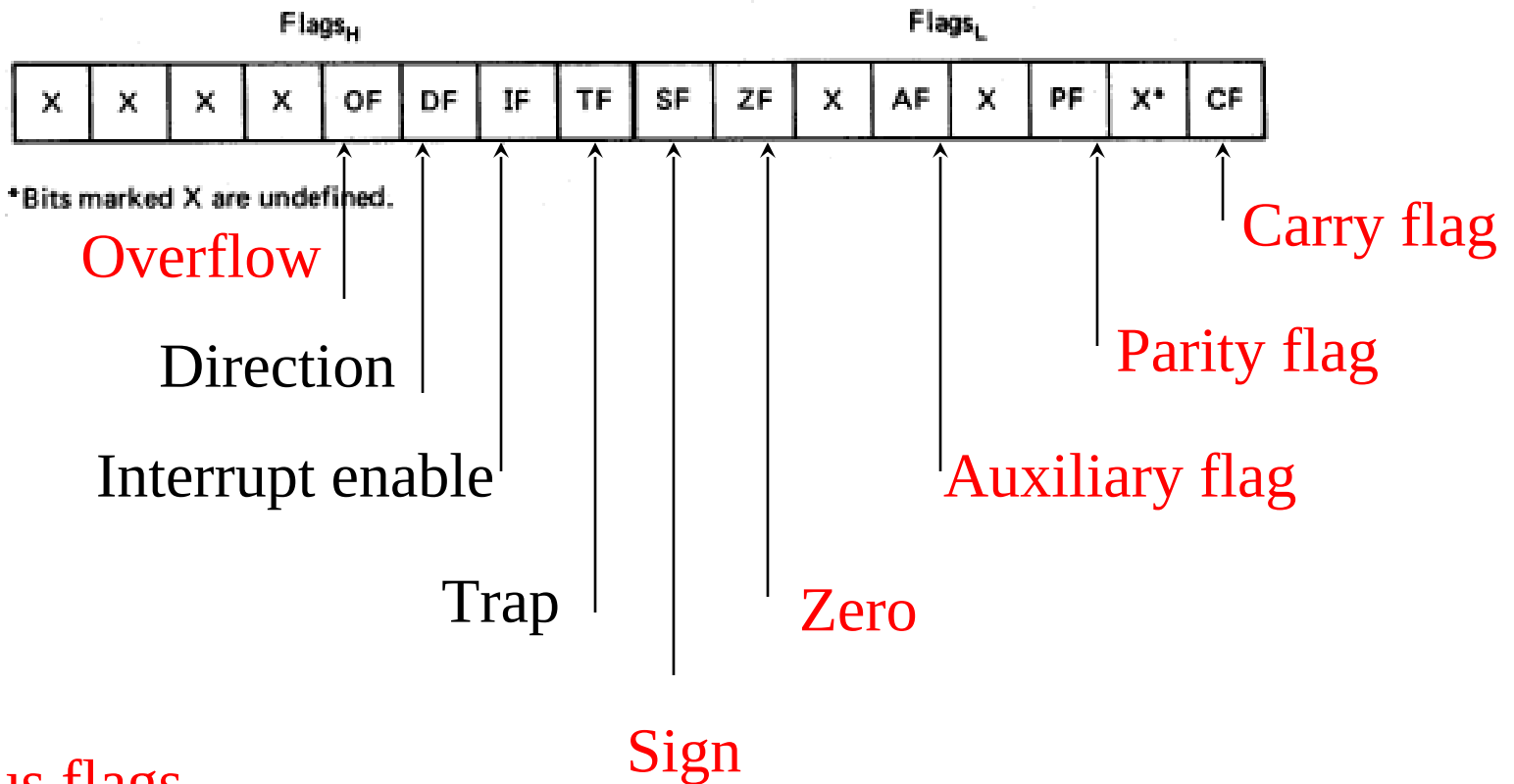
Data Segment

- The term pointer is used for a register holding an offset address
- Use BX as a pointer

```
MOV AL,0  
MOV BX,0200h  
ADD AL,[BX]  
INC BX  
ADD AL,[BX]  
INC BX  
ADD AL,[BX]  
INC BX  
ADD AL,[BX]  
INC BX  
ADD AL,[BX]
```

- If the offset address of data is to be changed, only one instruction will need to be modified

Flags in flag register



6 are status flags

3 are control flag

Flag Register

- Conditional flags:
 - They are set according to some results of arithmetic operation. You do not need to alter the value yourself.
- Control flags:
 - Used to control some operations of the MPU. These flags are to be set by you in order to achieve some specific purposes.

Flag					O	D	I	T	S	Z		A		P		C
Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- CF (carry) Contains carry from leftmost bit following arithmetic, also contains last bit from a shift or rotate operation.

Flag Register

- OF (overflow) Indicates overflow of the leftmost bit during arithmetic.
- DF (direction) Indicates left or right for moving or comparing string data.
- IF (interrupt) Indicates whether external interrupts are being processed or ignored.
- TF (trap) Permits operation of the processor in single step mode.

- SF (sign) Contains the resulting sign of an arithmetic operation (1=negative)
- ZF (zero) Indicates when the result of arithmetic or a comparison is zero. (1=yes)
- AF (auxiliary carry) Contains carry out of bit 3 into bit 4 for specialized arithmetic.
- PF (parity) Indicates the number of 1 bits that result from an operation.



- Six of the flags are status indicators reflecting properties of the last arithmetic or logical instruction.
- For example, if register AL = 7Fh and the instruction ADD AL,1 is executed then the following happen
 - AL = 80h
 - CF = 0; there is no carry out of bit 7
 - PF = 0; 80h has an odd number of ones
 - AF = 1; there is a carry out of bit 3 into bit 4
 - ZF = 0; the result is not zero
 - SF = 1; bit seven is one
 - OF = 1; the sign bit has changed
- Can be used to transfer program control to a new memory location

TF, IF, and DF

- Three of the flags can be set or reset directly by the programmer and are used to control the operation of the microprocessor, these are TF, IF, and DF.
- When TF (Trap Flag) is set, control is passed to special address after each instruction is executed. Normally a program to display all the registers and flags is stored there. Single-stepping mode.
- When IF (Interrupt Flag) is set, external interrupt requests on the 8086's interrupt line INTR will be enabled.
 - For example a printer may spend several seconds printing a page of text from its internal buffer
 - When it is ready for new data, the printer control circuit drives the 8086's INTR input line
 - The processor then suspends whatever it is doing and begins running the printer *interrupt service routine* (ISR)
 - When the routine has finished via a IRET (interrupt return) instruction control is transferred back to the original instruction in the main program that was executing when the interrupt occurred
 - Hardware and software interrupts
- Direction Flag (DF) is used with block move instructions.
 - DF = 1 then the block memory pointer will automatically decrement
 - DF = 0, then the block memory pointer will automatically increment