UNIT II

8086 Software Aspects

UNIT II (OverView)

- **8086 Software Aspects**

8086 Architecture

- **Instruction Set and Assembler Directives**
- **Addressing Modes**
- **ALP**
- **Procedures and Macros**
- **Interrupts and ISR**
- **Tutorial**

Text Book

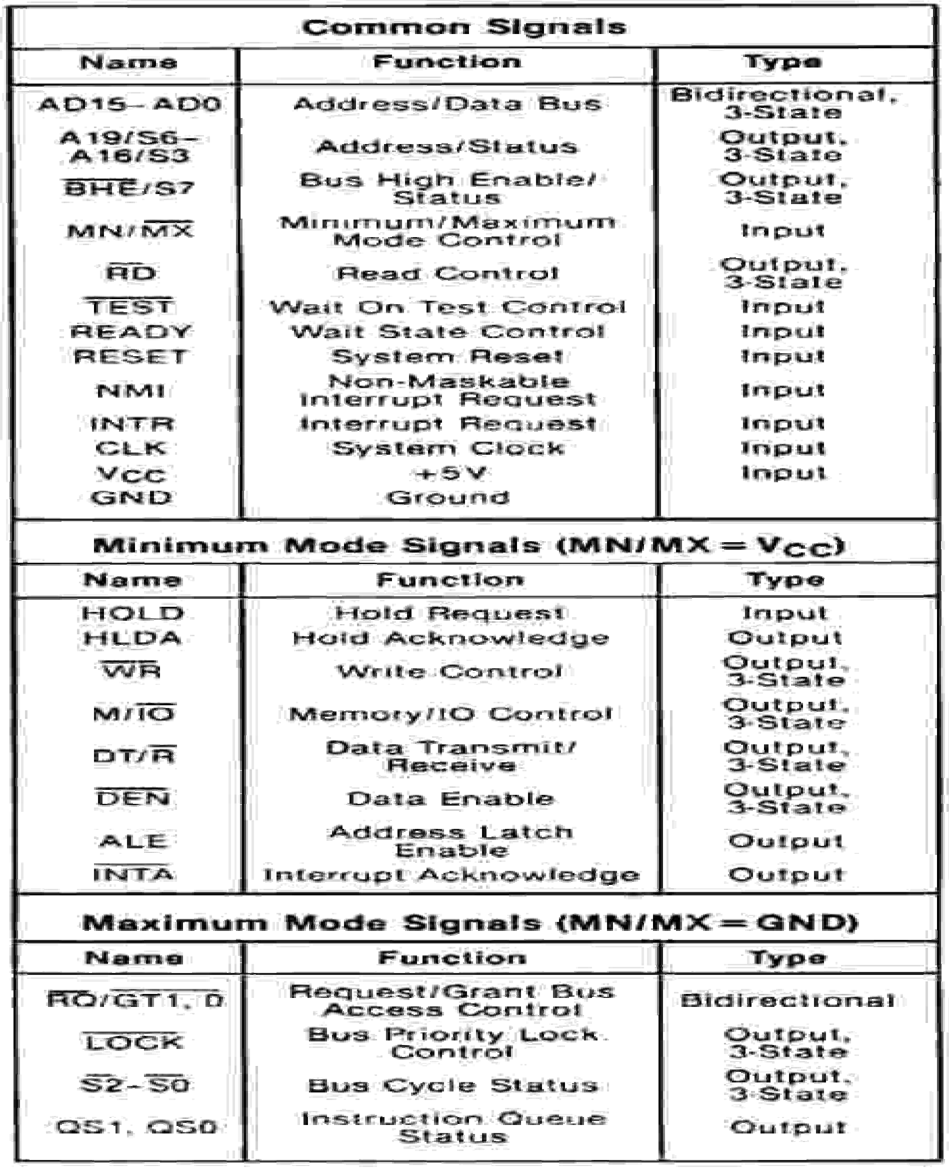
A.K. Ray & K.M.Bhurchandi,

“Advanced Microprocessors and peripherals- Architectures, Programming and Interfacing”,

TMH, 2002 reprint.

(UNITS 2 to 5:– Chapters 1-6, 7.1-7.3, 8, 16)

8086 Microprocessor



Common Signals

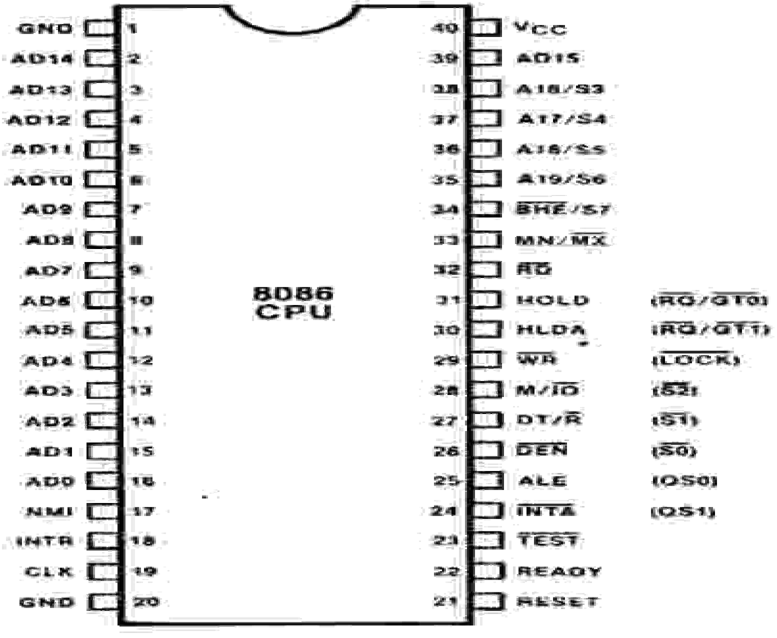
Name	Function	Type
AD15–AD0	Address/Data Bus	Bidirectional, 3-State
A19/S6–A16/S3	Address/Status	Output, 3-State
$\overline{BHE}$/S7	Bus High Enable/Status	Output, 3-State
MN/$\overline{MX}$	Minimum/Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output, 3-State
$\overline{TEST}$	Wait On Test 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	+5 V	Input
GND	Ground	

Minimum Mode Signals (MN/$\overline{MX}$ = V_{CC})

Name	Function	Type
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{WR}$	Write Control	Output, 3-State
M/$\overline{IO}$	Memory/IO 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
$\overline{INTA}$	Interrupt Acknowledge	Output

Maximum Mode Signals (MN/$\overline{MX}$ = GND)

Name	Function	Type
$\overline{RQ}/\overline{GT1}$, 0	Request/Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3-State
$\overline{S2}$–$\overline{S0}$	Bus Cycle Status	Output, 3-State
QS1, QS0	Instruction Queue Status	Output



MAXIMUM MODE PIN FUNCTIONS (e.g. $\overline{LOCK}$) ARE SHOWN IN PARENTHESES

Figure 4-1 . 8086 Pin Definitions

8086 Architecture- Functional Block Diagram

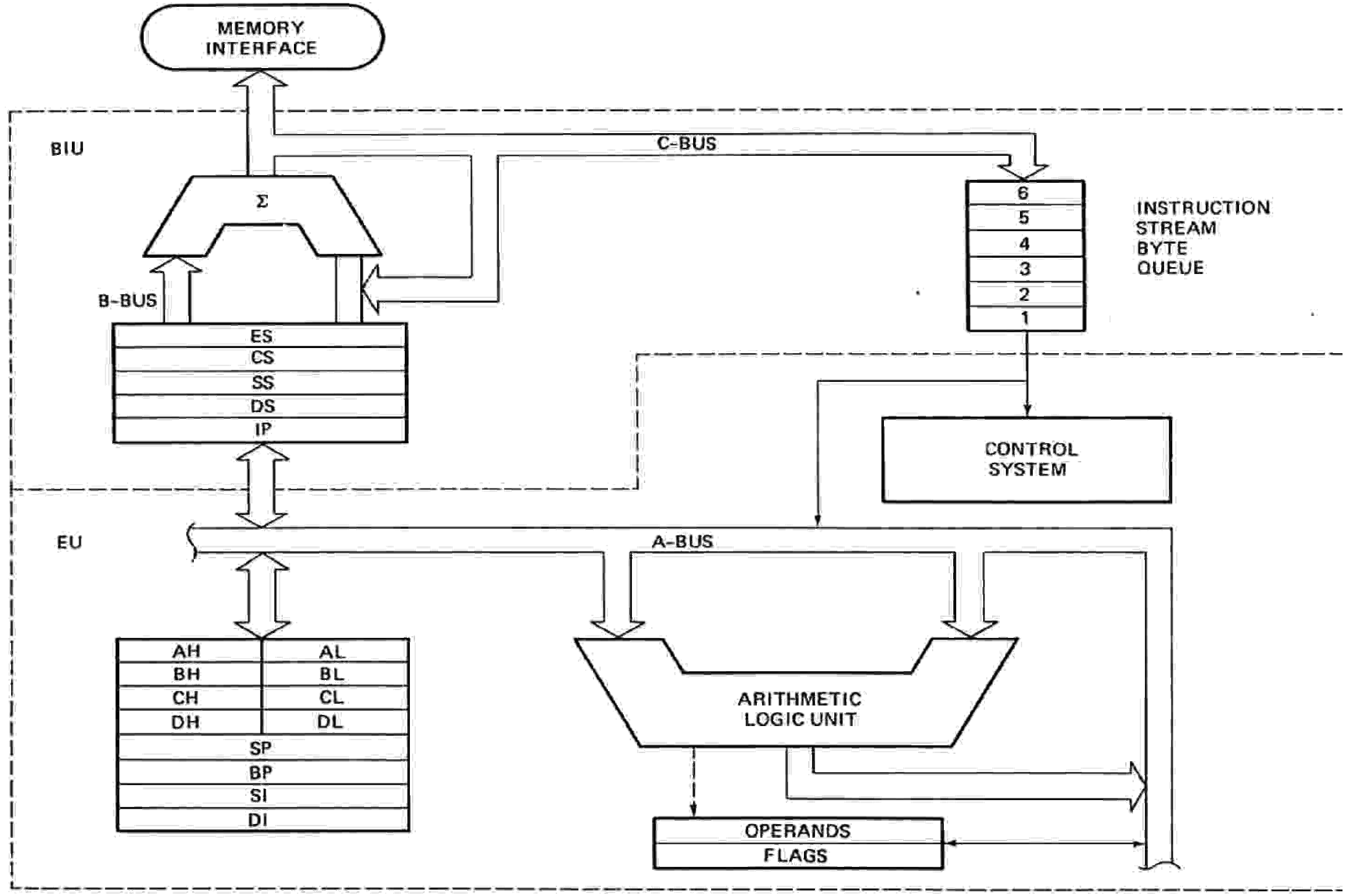
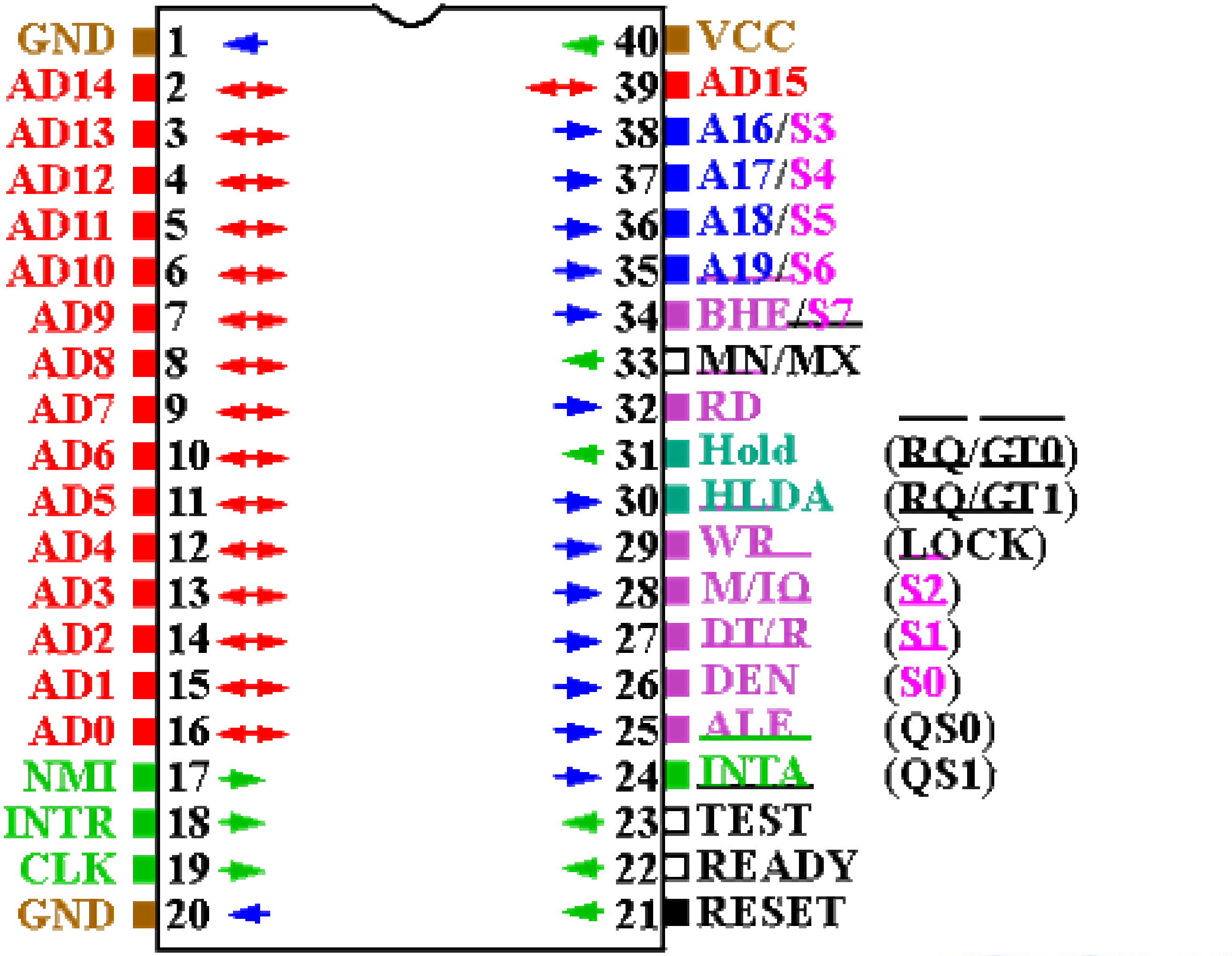


FIGURE 2-7 8086 internal block diagram. (*Intel Corp.*)

8086 CPU



MIN MODE (MAX MODE)

			MIN MODE	(MAX MODE)
GND	1	40	VCC	
AD14	2	39	AD15	
AD13	3	38	A16/S3	
AD12	4	37	A17/S4	
AD11	5	36	A18/S5	
AD10	6	35	A19/S6	
AD9	7	34	BHE/S7	
AD8	8	33	MN/MX	
AD7	9	32	RD	
AD6	10	31	Hold	(RQ/GT0)
AD5	11	30	HLDA	(RQ/GT1)
AD4	12	29	WR	(LOCK)
AD3	13	28	M/IO	(S2)
AD2	14	27	DT/R	(S1)
AD1	15	26	DEN	(S0)
AD0	16	25	ALE	(QS0)
NMI	17	24	INTA	(QS1)
INTR	18	23	TEST	
CLK	19	22	READY	
GND	20	21	RESET	

8086 Architecture

- It has two parts
  - **BIU (Bus Interface Unit)**

    Fetches Instructions, reads and writes data and computes 20 bit address

  - **EU (Execution Unit)**

    Decodes and executes the instructions in 16-bit ALU

8086 Architecture

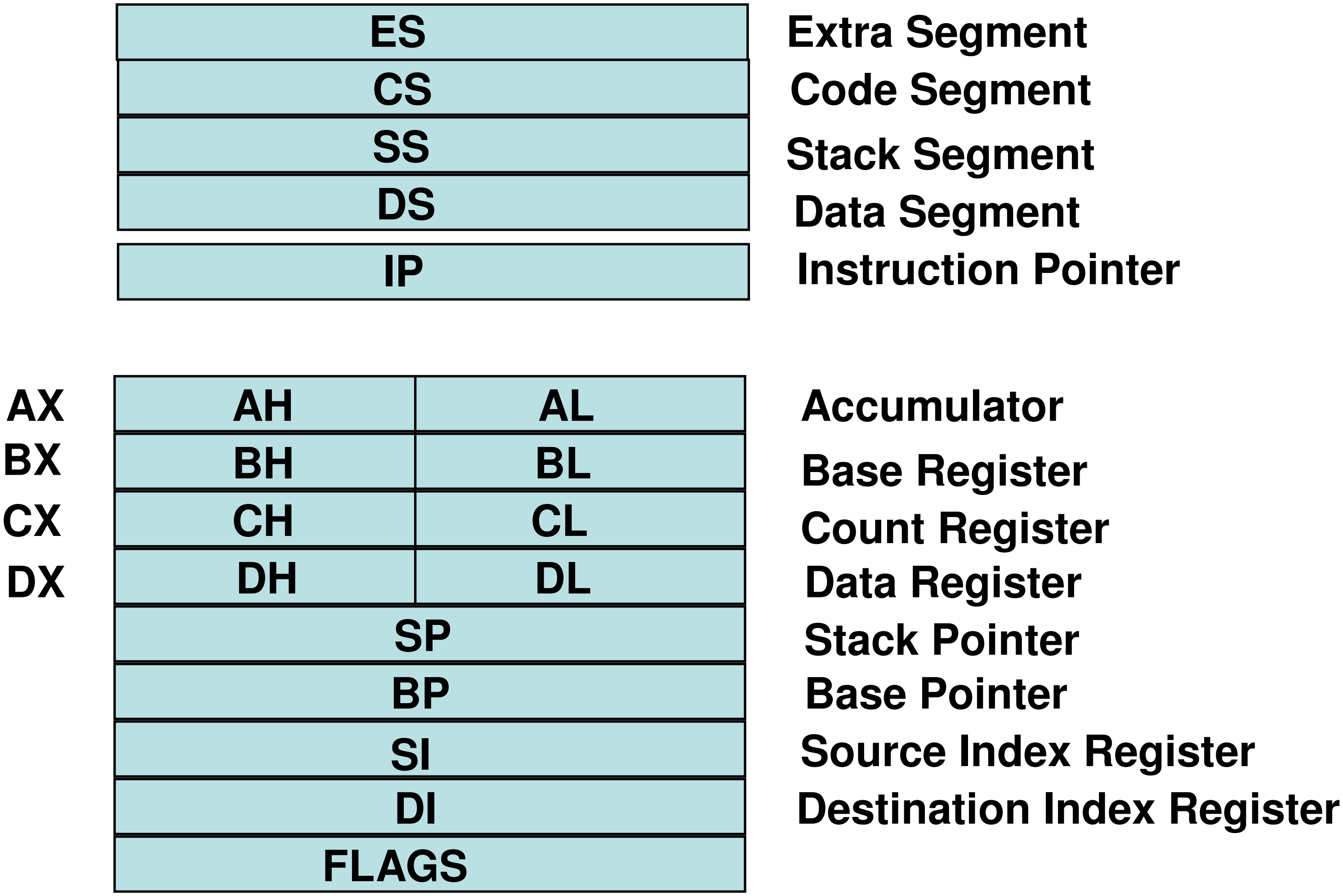
The BIU fetches instructions using the CS and IP, CS:IP, to provide the 20-bit address.

Data is fetched using a segment register (usually the DS)

An effective address (EA) computed by the EU (depending on the addressing mode.)

v1

8086 Programmer's Model



BIU registers
(20 bit adder)

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

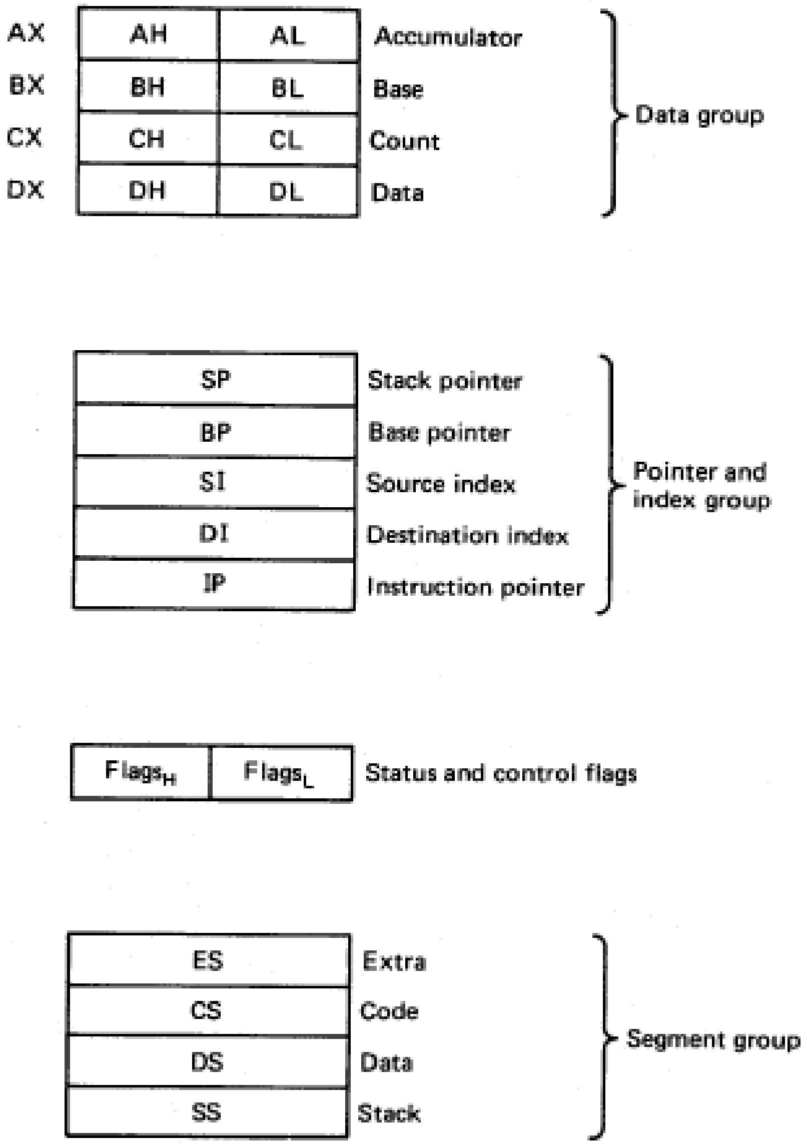
EU registers

AX	AH	AL	Accumulator
BX	BH	BL	Base Register
CX	CH	CL	Count Register
DX	DH	DL	Data Register
	SP		Stack Pointer
	BP		Base Pointer
	SI		Source Index Register
	DI		Destination Index Register
	FLAGS		

Slide 9

v1 vijay, 1/27/2009

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	

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

Flags_H	Flags_L	Status and control flags

ES	Extra	Segment group
CS	Code	
DS	Data	
SS	Stack	

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

These registers are used as memory *pointers*.

Flags will be discussed later

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

- 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 & DI: Destination Index register**
  - is required for some 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 1M 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

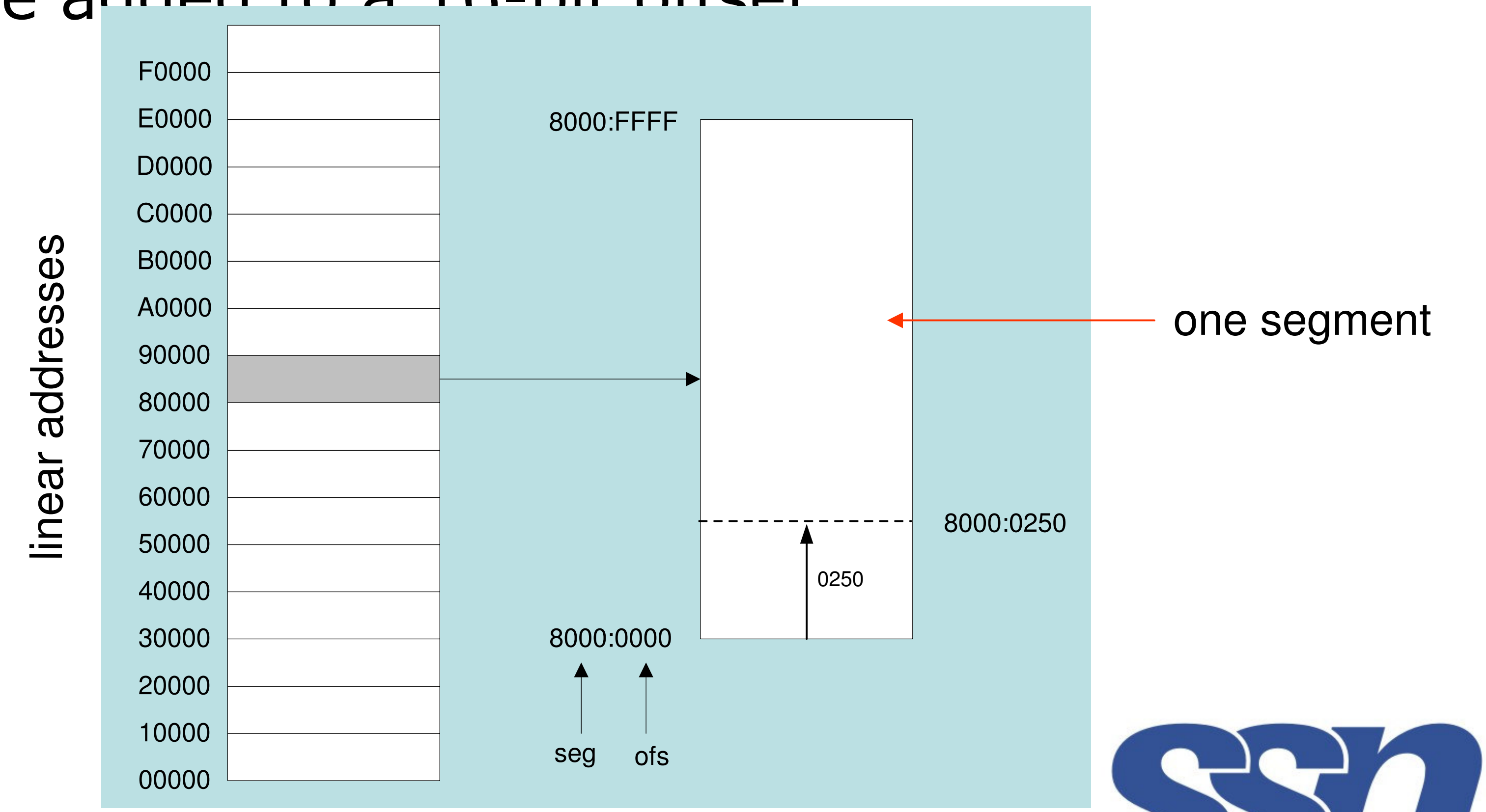
Segmented Memory Architecture

(real mode)

- A segment addresses 64K of memory
- A segment register contains the starting location of a segment
  - the absolute location of a segment can be obtained by appending a hexadecimal zero
- An offset is the distance from the beginning of a segment to a particular instruction or variable

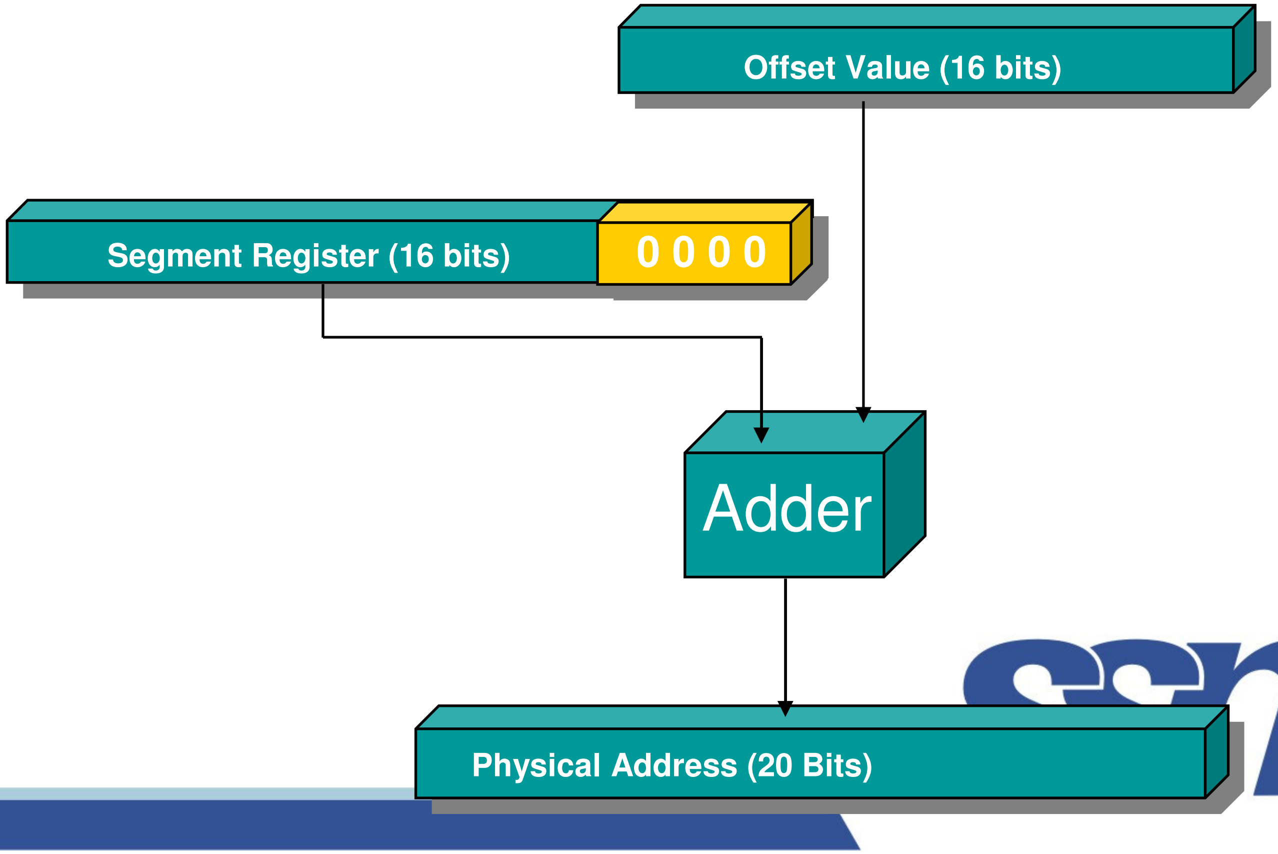
Segmented Memory

Segmented memory addressing: absolute (linear) address is a combination of a 16-bit segment value added to a 16-bit offset



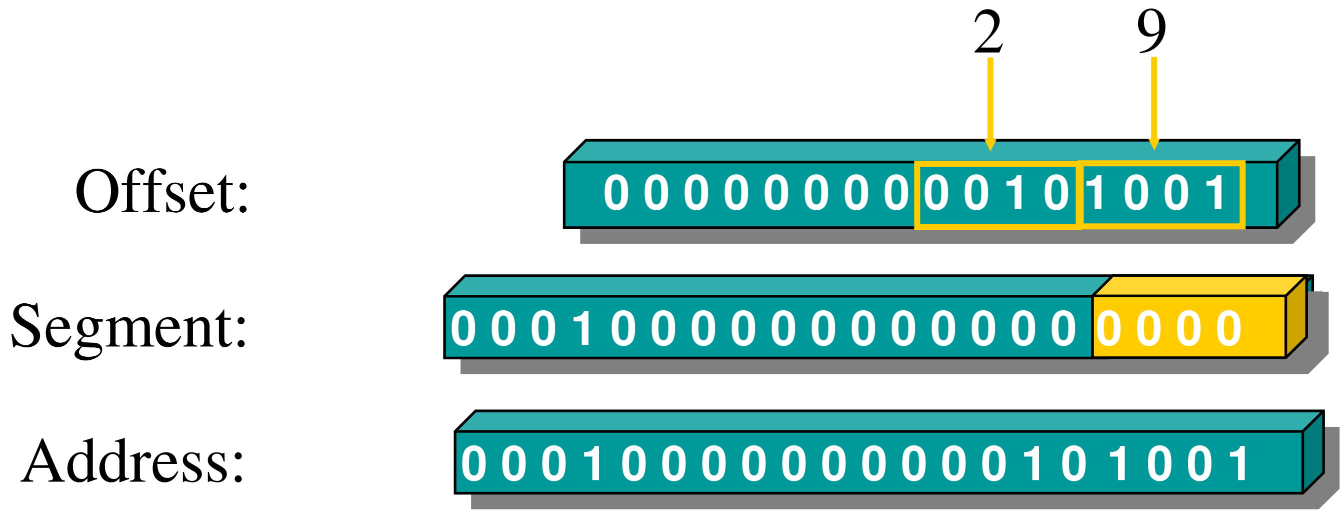
Memory Address Generation

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



Example Address Calculation

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



Segment:Offset Address

- 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

$$\begin{array}{r} \text{A4FB0} \\ +\ \underline{4872} \\ \text{A9822} \end{array}$$

Example

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

  a) Calculate the physical address

  7FA20 + 438E = 83DAE

  b) calculate the lower range

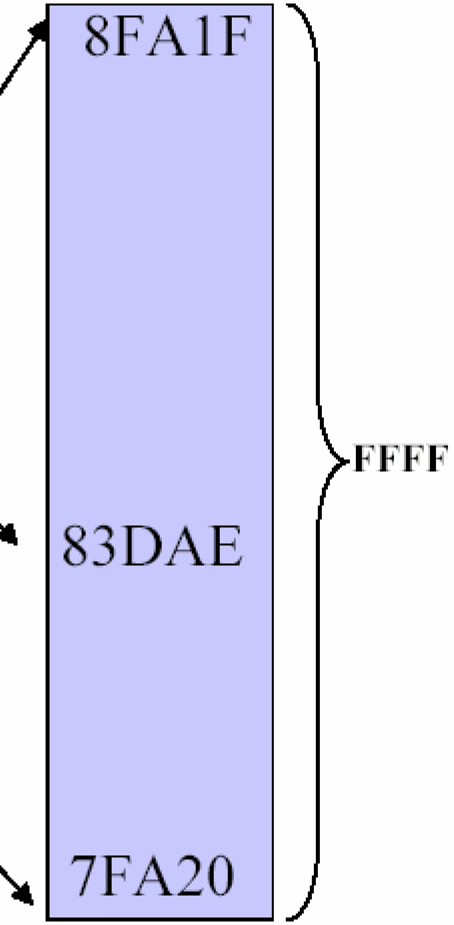
  7FA20 + 0000 = 7FA20

  c) Calculate the upper range of the data segment

  7FA20 + FFFF = 8FA1F

  d) Show the logical Address

  7FA2:438E



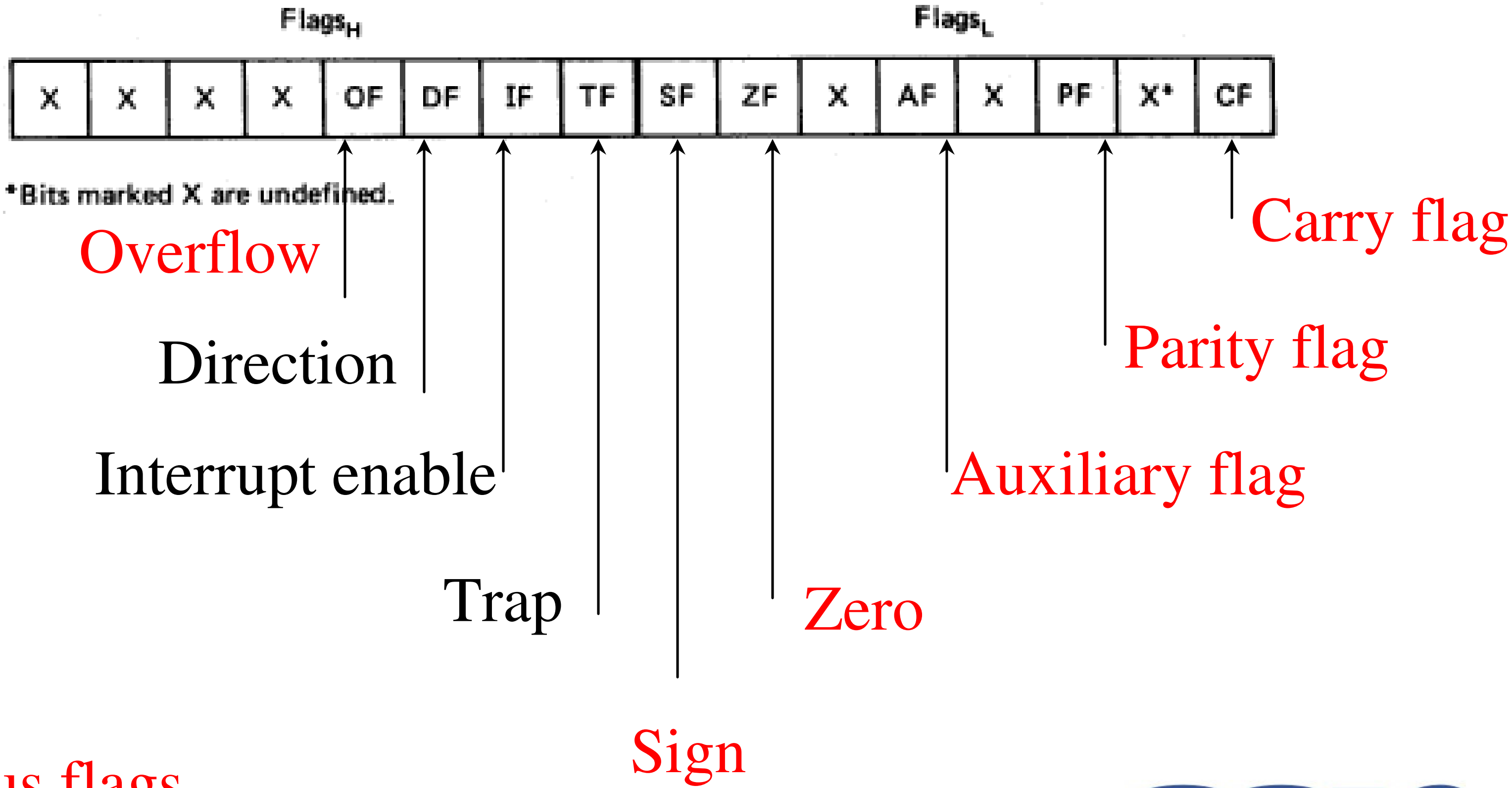
Your turn . . .

What linear address corresponds to the segment/offset address 028F:0030?

028F0 + 0030 = 02920

Always use hexadecimal notation for addresses.

Flags



Flags$_H$								Flags$_L$							
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X*	CF

*Bits marked X are undefined.

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

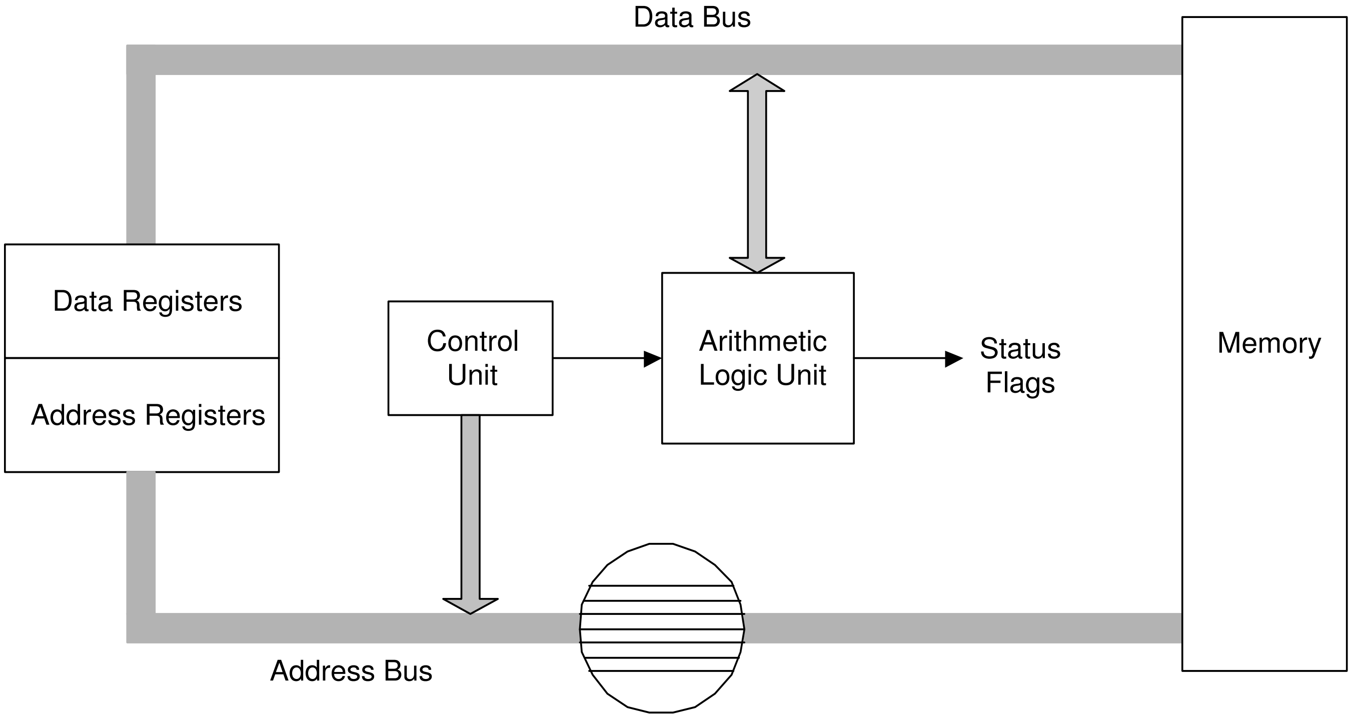
Flag Register

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

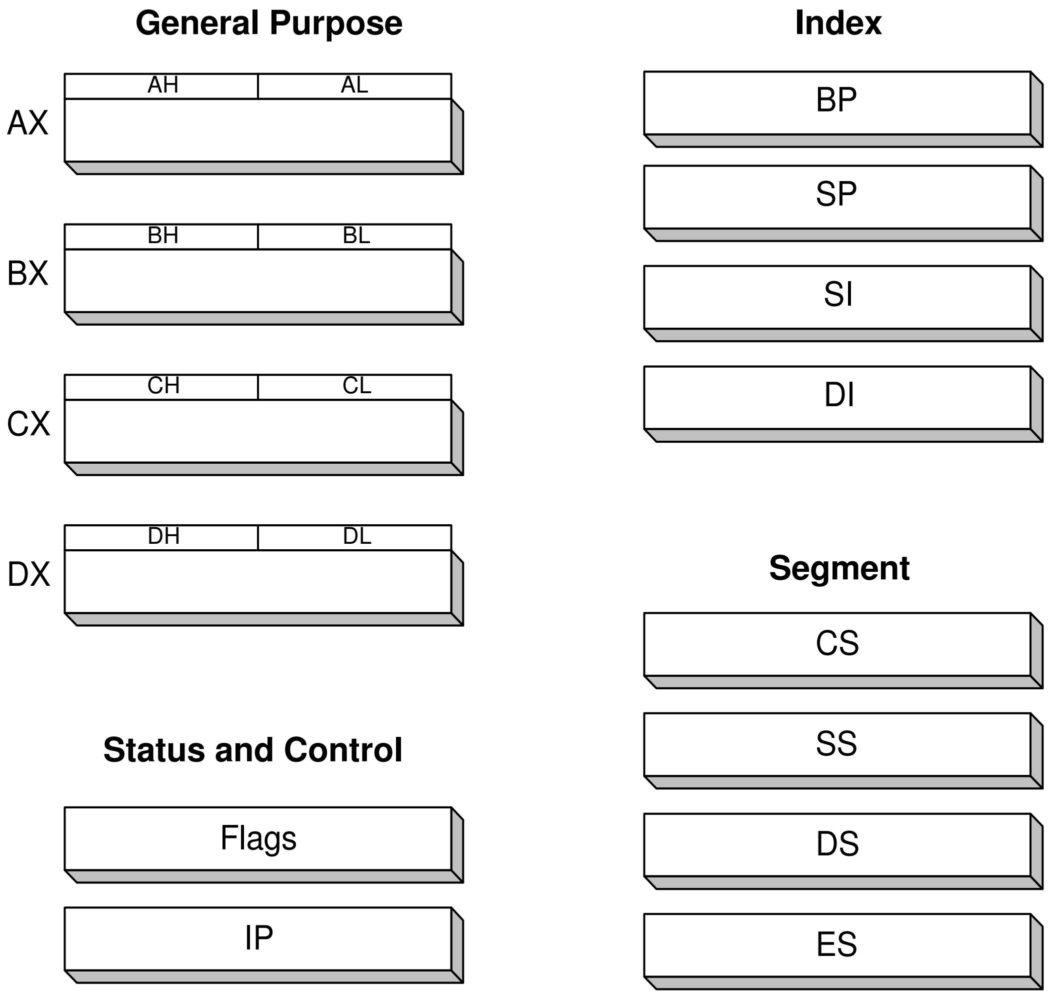
Flag Register Contd.,

- **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.

Simplified CPU Design



Intel 16-bit Registers



General Purpose

AX: AH | AL

BX: BH | BL

CX: CH | CL

DX: DH | DL

Status and Control

Flags

IP

Index

BP

SP

SI

DI

Segment

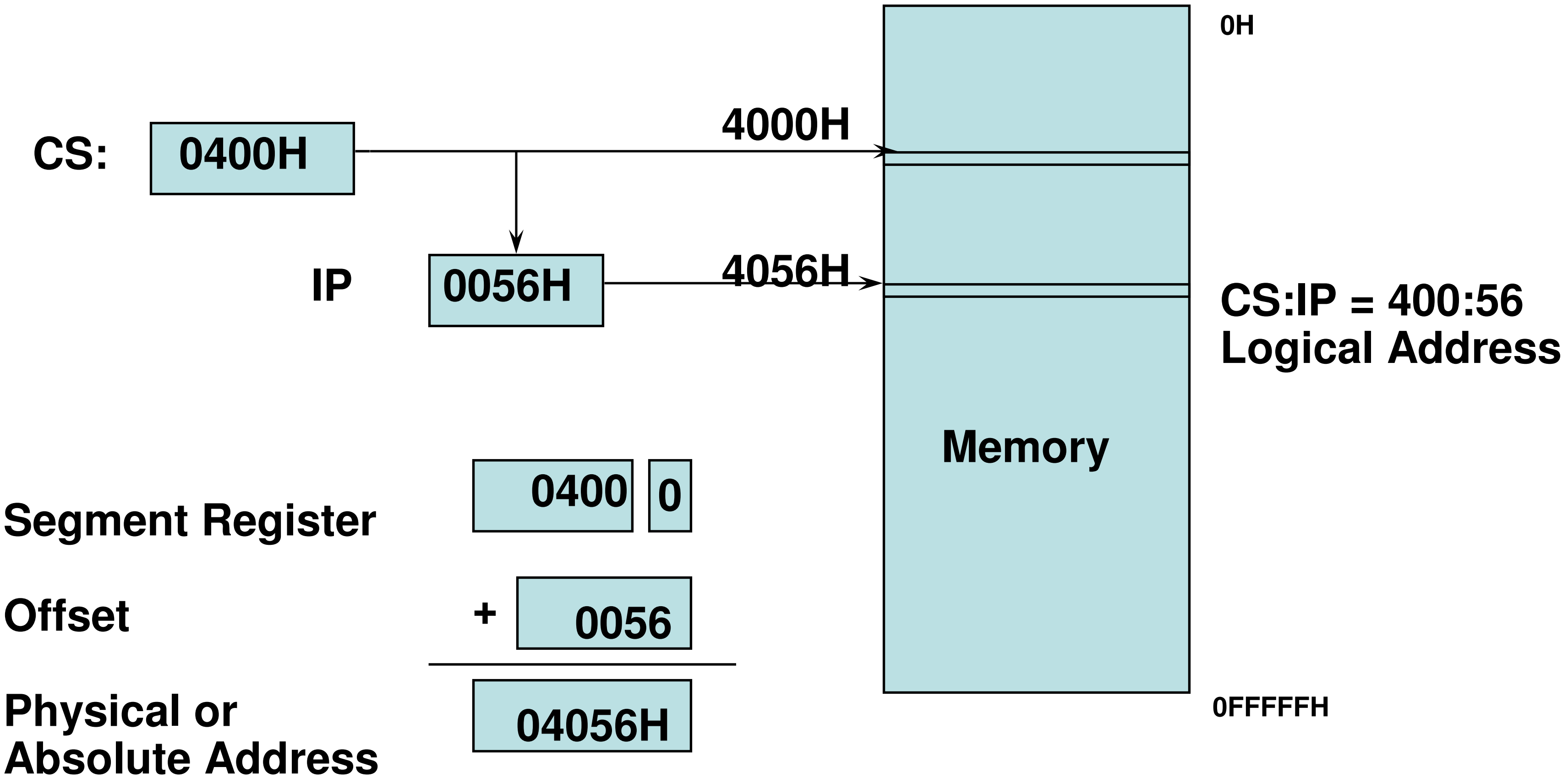
CS

SS

DS

ES

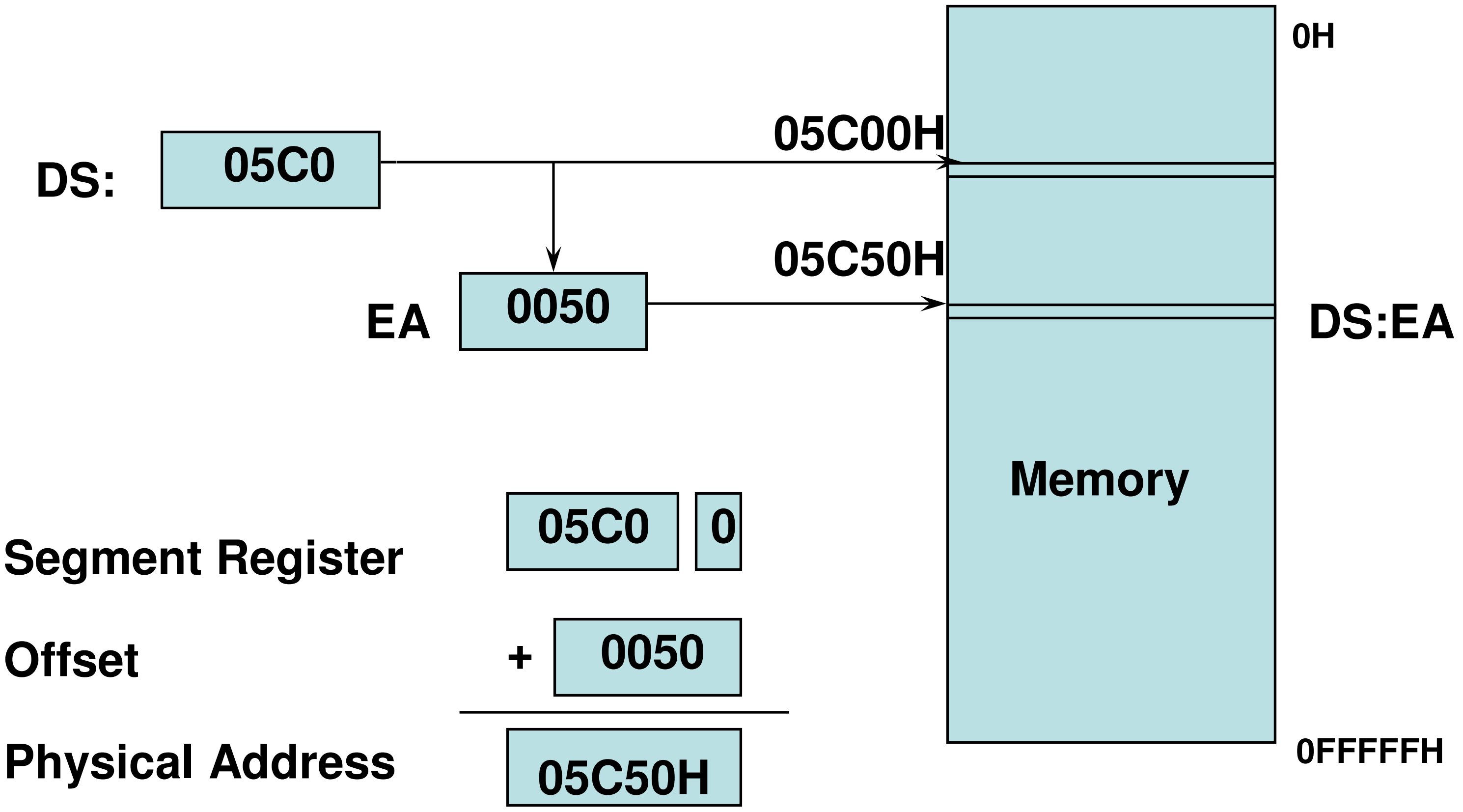
The Code Segment



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.

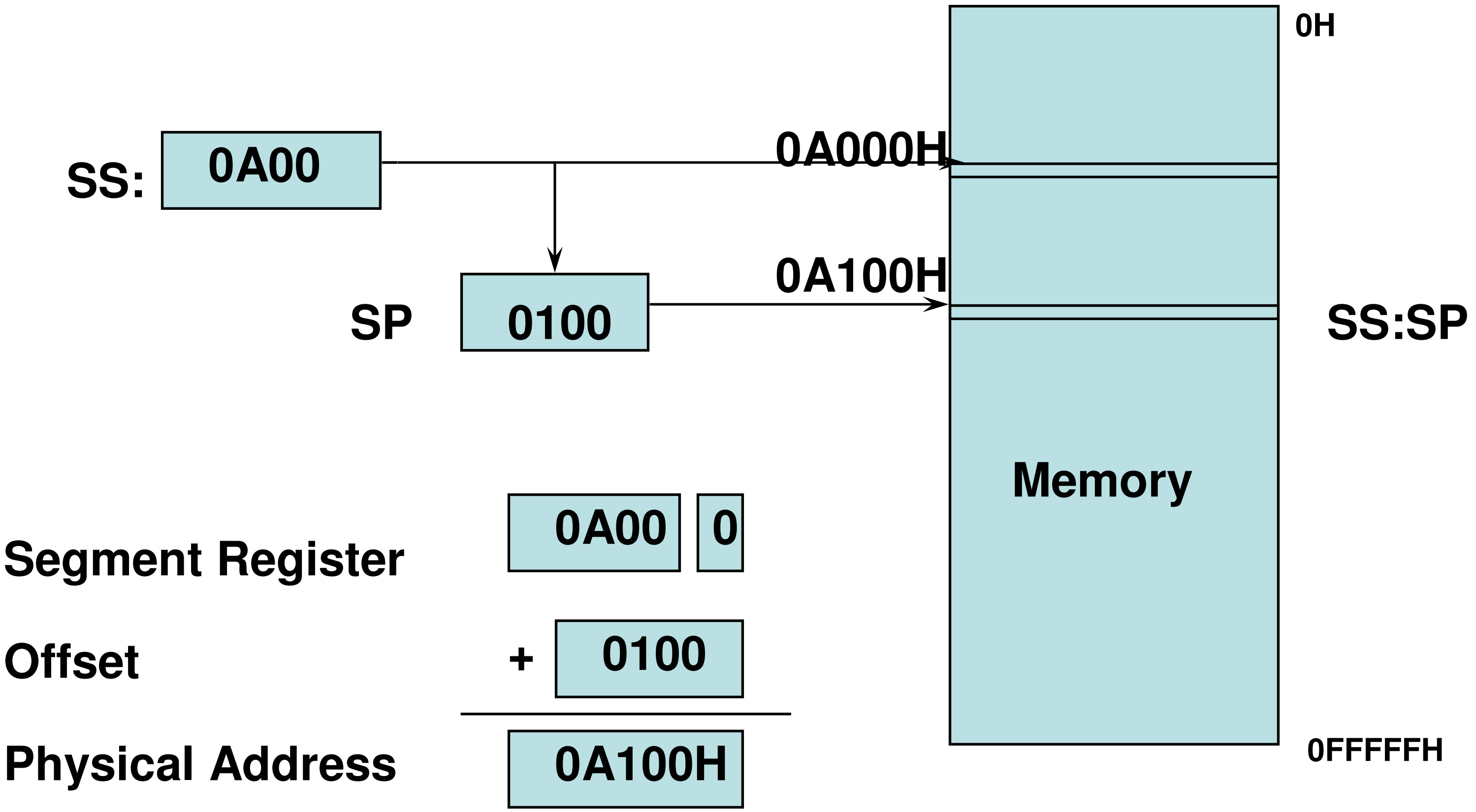
The physical address is also called the absolute address.

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 Stack Segment

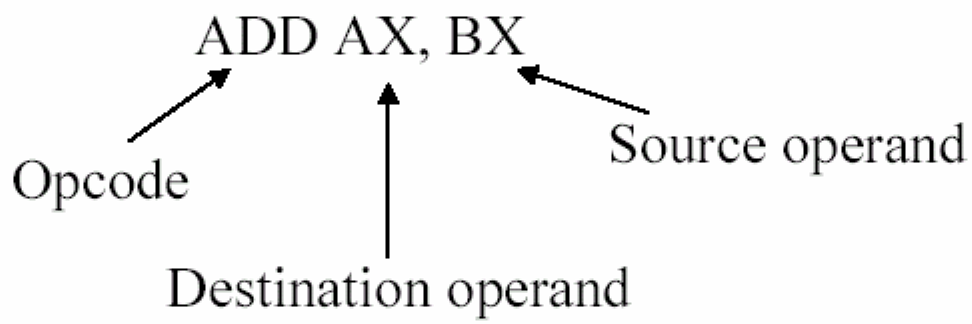


The offset is given by the SP register.
The stack is always referenced with respect to the stack segment register.
The stack grows toward decreasing memory locations.
The SP points to the last or top item on the stack.

PUSH - pre-decrement the SP
POP - post-increment the SP

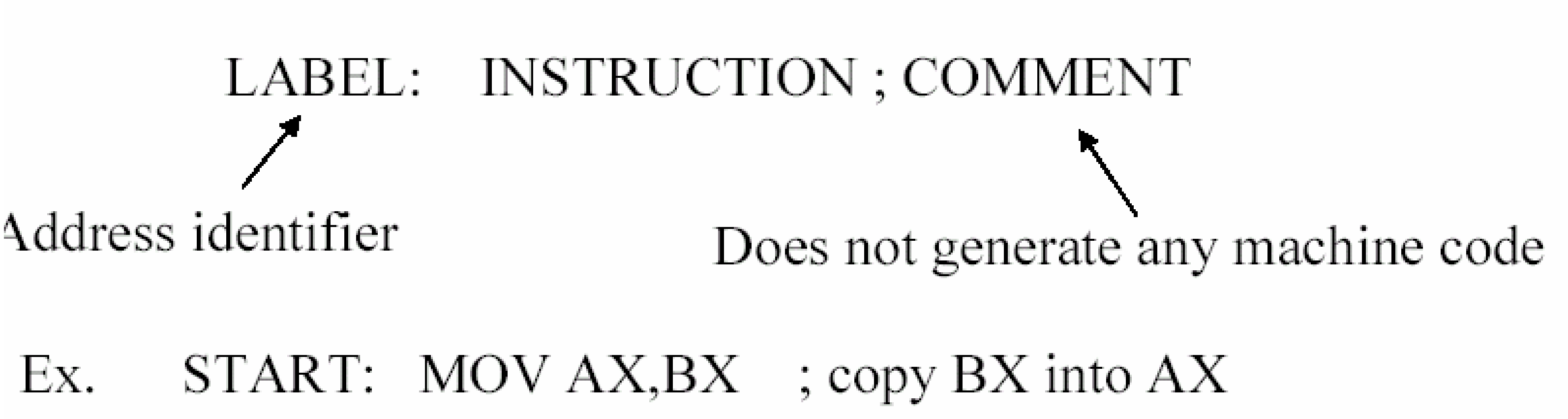
Software

- 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.



ADD AX, BX

Instructions



LABEL: INSTRUCTION ; COMMENT

Address identifier

Does not generate any machine code

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

Machine Language Instruction Formats

- One Byte Instruction
- Register to Register
- Register to/from Memory with no Displacement
- Register to/from Memory with Displacement
- Immediate Operand to Register
- Immediate Operand to memory with 16-bit Displacement.

Addressing Modes in 8086

- **Immediate**
- **Direct**
- **Register**
- **Register Indirect**
- **Indexed**
- **Register Relative**
- **Based Indexed**
- **Relative Based Indexed**
- **Intrasegment Direct Mode**
- **Intrasegment Indirect Mode**
- **Intersegment Direct**
- **Intersegment Indirect**

Instruction Sets of 8086

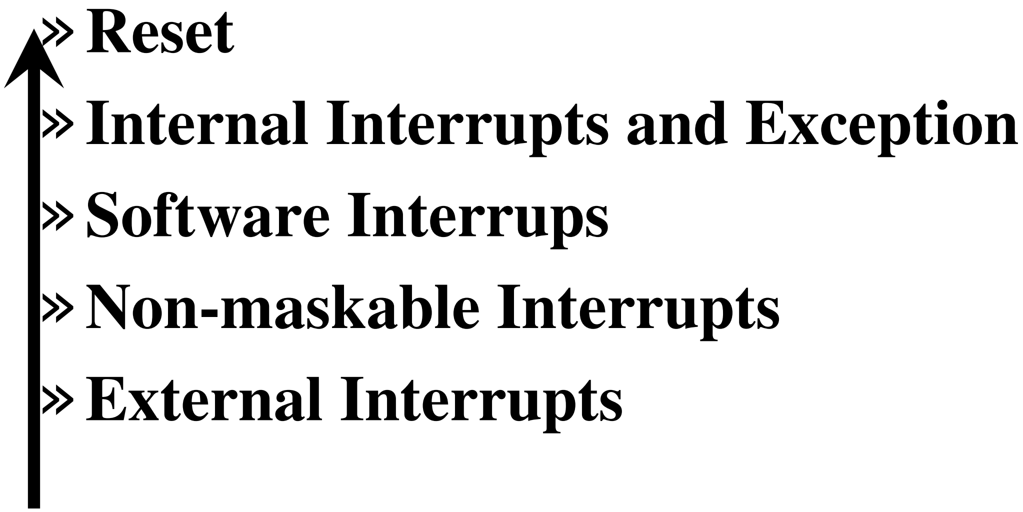
- **Data Copy/Transfer Instructions**
- **Arithmetic and Logical Instructions**
- **Branch Instructions**
- **Loop Instructions**
- **Machine control Instructions**
- **Flag manipulation Instructions**
- **Shift & Rotate Instructions**

Interrupt Types

- Up to 256 interrupts
- Divided into five groups:
  - 1. External Hardware interrupts.
  - 2. Nonmaskable.
  - 3. Software interrupt.
  - 4. Internal interrupt.
  - 5. Reset.

Priority

- Increasing priority is given as below



  - **Reset**
  - **Internal Interrupts and Exception**
  - **Software Interrups**
  - **Non-maskable Interrupts**
  - **External Interrupts**

Interrupt Instructions

CLI & STI:

- Manipulate the interrupt flag through software
- STI enables the external interrupt request (INTR)
- CLI disables the external interrupt request (INTR)

INT n:

- Software interrupt instruction.
- Program will transfer the the subroutine pointed to by vectors.

Interrupt Instructions Cont

.

- **IRET**

– Interrupt Return

– Instruction is at the end of each service subroutine

– Causes old value of flags to be popped

– CS & IP are popped

- **INTO**

– Interrupt-on-overflow

– Must be included after arithmetic instructions

– Type 4 interrupt

– IP at 0010H & CS = 00012H

Interrupt Instructions Cont

.

- **HLT & Wait:**

– **Halt:** MPU suspend operations & initiate idle states

– **Wait**: MPU check the logic level of TEST input prior to going to idle state only if TEST = 1 MPU will go idle state.

While in idle state, MPU checks the login level at TEST it goes to zero execution resumes.

Enabling/Disabling of Interrupts

Interrupt-enable Flag bit

IF: affects only the external hardware interrupts.

External input = INTR

During execution of interrupt, the MPU clears IF

Software Interrupts

- Implements up to 256 software interrupts.
- Service subroutines are initiated in response to the execution of a software interrupts instructions.
- Not an event in external hardware.
- INT n initiate a software interrupt
- Software interrupts are of a higher priority than external interrupts.

Software Interrupts

- On interrupts:

1. Old flags are saved on the stack.
2. Then IF & TF are cleared.
3. Old CS & IP pushed onto the stack.
4. New CS & IP read from memory
5. Program execution resumes at CSnew:IPnew

Internal Interrupts Functions

•**Divide Error:**
– IDIV or DIV larger than specified destination
– **Type 0 interrupt**

Single Step:
– TF is set, the single step operation is enabled
– Software control
– **Type 1 interrupt**
– Program can be executed one instruction at a time-debugging
– Could include WAIT inside the subroutine

Internal Interrupts Functions

NMI:

1. Cannot be masked out with the interrupt flag

→

NMI input is positive edge-triggered

0 1 on NMI input, NMI flip-flop MPU is set

- Must be active two consecutive clock cycles
- Current flags, CS & IP pushes into stack
- **Type 2 interrupt**
- NMI type of interrupts must respond to immediately (power failure & memory-read error)

Internal Interrupts Functions

Breakpoint:
– CC instructions
– Cause execution to stop at a certain location
- Type 3 interrupt

Overflow Error:
– Results of any arithmetic operation
– Not like Divide error, transfer is not automatic
– INT 0 after arithmetic instruction
– **Type 4 interrupt**
– Could cause a message to be displayed

Reset Interrupt

- Reset:

  hardware means of initializing the
  happens at power up, but PC allow time for
  This will synchronize the Reset with the
  MPU when it sees RESET,

  terminates operations, Bus = Z state

Memory Interfacing

- Two Types of Interfacing
  - I/O Mapped or accumulator I/O or Peripheral I/O or Isolated I/O
  - Memory Mapped I/O

Memory Decoding

- **In Practice all memory locations are not decoded.**
- **Unused address lines are decoded to generate chip select.**
- **The are two types of address decoding**
  - **Fully address decoding**
  - **Partial address decoding**

Fully Address Decoding

- **All unused lines are decoded to generate the chip select.**
- **All unused address values have particular value either 1 or 0.**
- **Each location has a fixed address.**
- **During read , write , output other locations are not affected.**
- **It does not reduce the size of the memory**
- **Expansive becoz hardware is complicated.**

Partial Address Decoding

- **All unused lines are not decoded to generate chip select.**
- **The value of undecoded address bit is don′t care.**
- **Address location are not fixed**

- **During read , write ,output other locations or its imaginary locations is affected. this type of memory is called fold-back memory.**
- **It reduces the size of memory modules.**
- **Inexpensive as hardware is simple.**

Semi Conductor Memory Interfacing

- **SemiConductor Moemory are of two types**
  - **RAM and ROM**
  - **RAM are of two types**
    - **Static RAM**
    - **Dynamic RAM**

Static RAM Interfacing

- **General Procedure for static Memory Interfacing**
  - **Arrange the available memory chips to obtain 16 bit data**
  - **The upper 8-bit bank are called ‛odd address memory Bank′ and the lower 8-bit are called ‛even address memory bank′**
  - **Connect available memory address lines of memory chips with those of mp and also to memory read and write input to corresponding processor control signals.**
  - **Connect the 16bit data bus of the memory bank with that of the microprocessor 8086.**
  - **The remainng address lines of the mp, BHE′ and A0 are used for Decoding the required chips select signals for the odd and Even memory banks.**
  - **The CS′ of memory is derived from O/P of the decoding circuit.**

Dynamic RAM Interfacing

- Whenever Large capacity memory is required in a microcomputer system then the memory subsystem is designed using Dynamic RAM.
- Advantage
  - Higher packing density
  - Lower cost
  - Less power consumption

Modular Programming

Modularity

- A system is modular when it is divided into subsystems (called modules) with good properties
  - Modules have distinct functional groupings
  - Hierarchy supports views at different granularity and scale
  - Separation of concerns among modules
  - Reusability of some modules

Reasons for breaking a program

- **Modules are easier to comprehend.**
- **Different modules can be assigned to different programmer.**
- **Debugging and testing can be done in orderly fashion**
- **Documentation is easily done**
- **Modifications may be localized**
- **Frequently used tasks can be programme into modules that are stored in libraries and used by several progrms.**

Constructing a program

- **Object modules some of which may be grouped into libraries , must be linked together to form a load module before the program can be executed.**
  - **Linker**
    - **Prints the memory map and indicates where the linked object module will be loaded.**
    - **Find the object modules**
    - **Construct the load module by assigning the position of all of the segments in all of the object modules being linked.**
    - **Fill in all the offsets that could not be determined by the assembler.**
    - **Fill in all segment address.**
    - **Load th e program for execution.**

Segment Combination

- Assembler provides a means of regulating the way segments in different object modules are organized by the linker.
- It has the form
  - Segment-name SEGMENT Combine-type.

Possible Combine types

- **PUBLIC**
  - **Different object modules are concatenated into single segment in the load module.**
- **COMMON**
  - **They are overlaid so that they have same beg., address.**
- **STACK**
  - **They become one segment whose length is sum of lengths of individually specified segments.**
- **AT**
  - **Is followed by an expression that evalutes to a constant which is to be the segment address.**
- **MEMORY**
  - **Causes the segment to be placed at the last of the load module.**

32bit/64bit processors Introduction

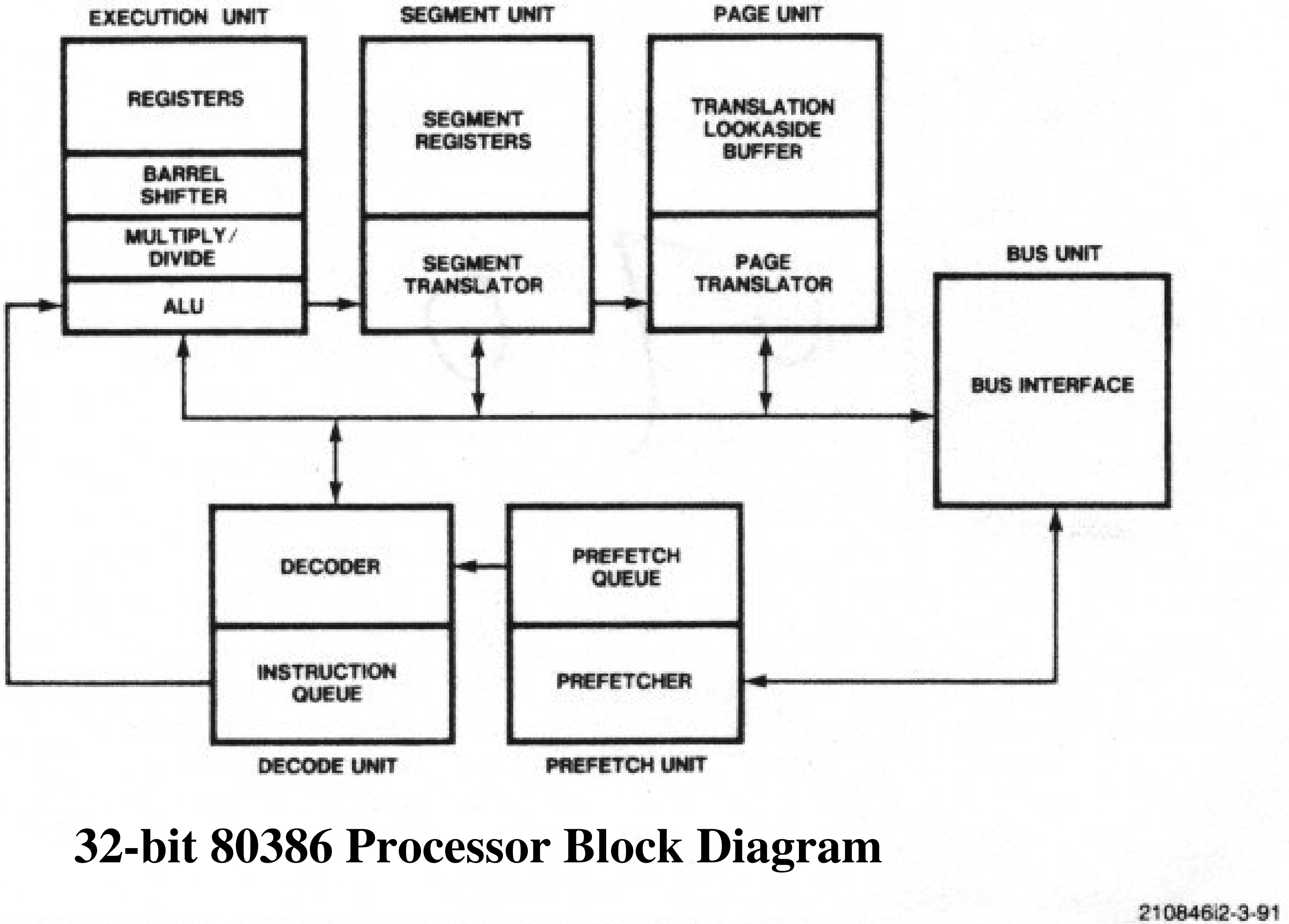
Features of 32 bit processor

1. **32-bit general and offset registers**
2. **16-byte prefetch queue**
3. **Memory Management Unit with a Segmentation Unit and a Paging Unit**
4. **32-bit Address and Data Bus**
5. **4-Gbyte Physical address space**
6. **64-Tbyte virtual address space**
7. **64K 8-, 16-, or 32-bit ports**
8. **Implementation of real, protected and virtual 8086 modes**

Real Mode and Protected Mode Operation

80286 ~ Pentium operated in

1) **Real Mode (run in DOS, Like very first 8086)**
2) **Protected Mode (run in Window OS) support for**
   - *** Multitasking**
   - *** Virtual memory addressing**
   - *** Memory management**
   - *** Protection**
   - *** Control cache**
3) **Virtual Mode**



32-bit 80386 Processor Block Diagram

Figure 2-3. Intel386™ DX CPU Block Diagram

Architecture of 80X86

- Internal Architecture consists of six functional units.

  BUS Interface Unit

  Code PREFETCH Unit

  Instruction DECODE Unit

  EXECUTION Unit

  SEGMENTation Unit

  PAGing Unit

BUS: Responsible for memory access, I/O access, Co processor interface and address relocation.

PAGE: Translates linear address generated by the segmentation or code pre fetch unit into physical address and gives to BUS interface Unit.

SEGMENT : translates the logical address to linear address and provides segment level protection.

EXECUTION: Executes instruction with the help of control, Data and Protection Unit

REGISTERS

- General Purpose Registers
- Segment registers
- Index, Pointer and Base Registers
- Flag registers
- System Address Registers
- Control Registers
- Debug Registers.
- Test Registers

General Purpose Registers

4 Data Register :

EAX, EBX, ECX, EDX

5 Pointer/Index Register

*** ESP : Stack Pointer**

*** EBP : Base Pointer**

*** ESI : Source Index**

*** EDI : Destination Index**

*** EIP : Instruction Pointer**

- **AX, BX, CX, DX, BP, SI and DI**

*** Accumulator(AX)**

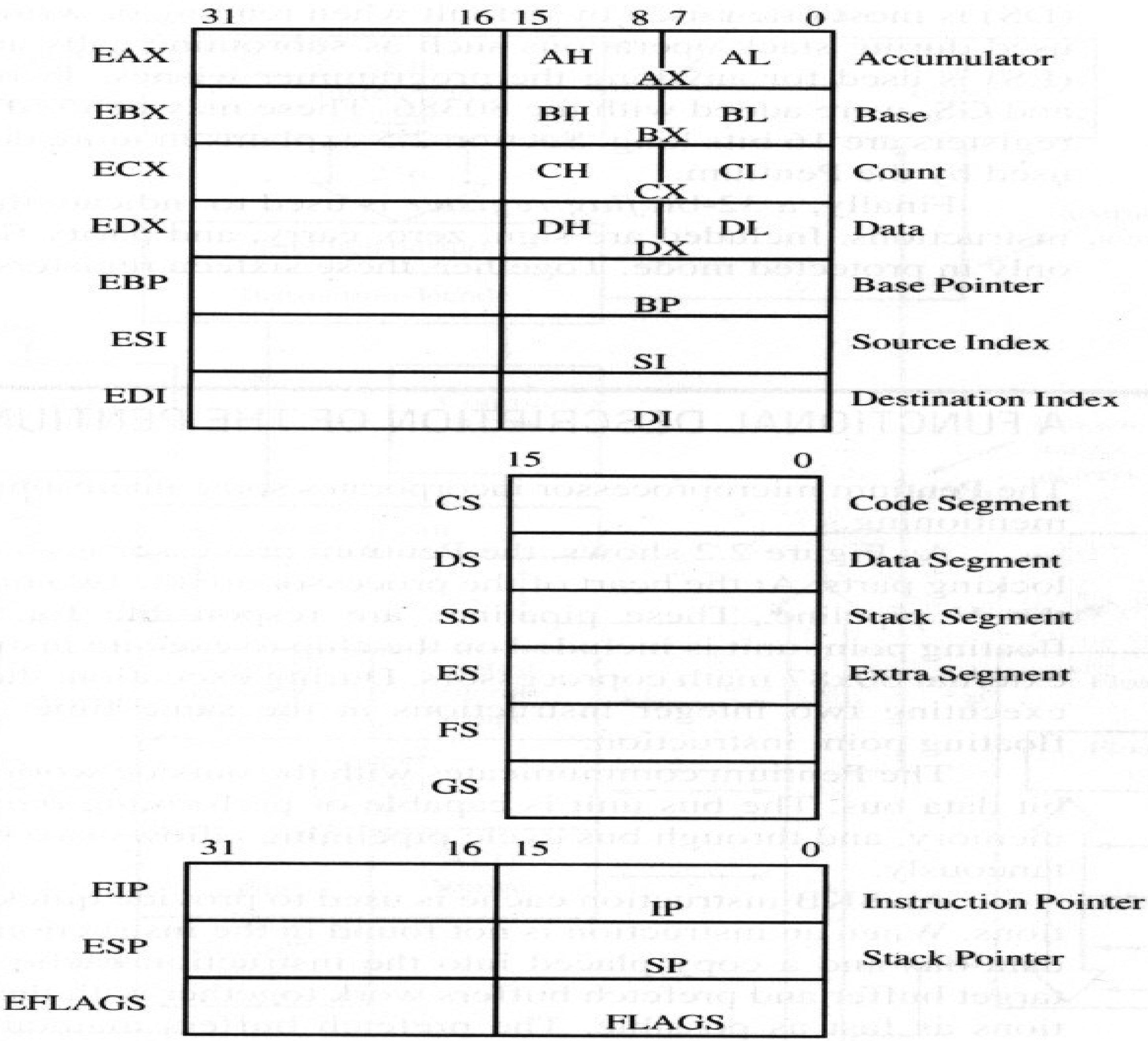
*** Count Register (CX)**

*** Data Register (DX)**

*** Source Index (SI) & Destination Index (DI)**

Segment Registers

- Supports six segments of 64kbytes of memory and addressed by 16 – bits.
  - CS (Code Segment)
  - DS (Data Segment)
  - ES (Extra Segment)
  - SS (Stack Segment)
  - FS and GS (Used as general data segment registers)


31
16
15
8
7
0
EAX
AH
AL
AX
Accumulator
EBX
BH
BL
BX
Base
ECX
CH
CL
CX
Count
EDX
DH
DL
DX
Data
EBP
BP
Base Pointer
ESI
SI
Source Index
EDI
DI
Destination Index
15
0
CS
Code Segment
DS
Data Segment
SS
Stack Segment
ES
Extra Segment
FS
GS
31
16
15
0
EIP
IP
Instruction Pointer
ESP
SP
Stack Pointer
EFLAGS
FLAGS

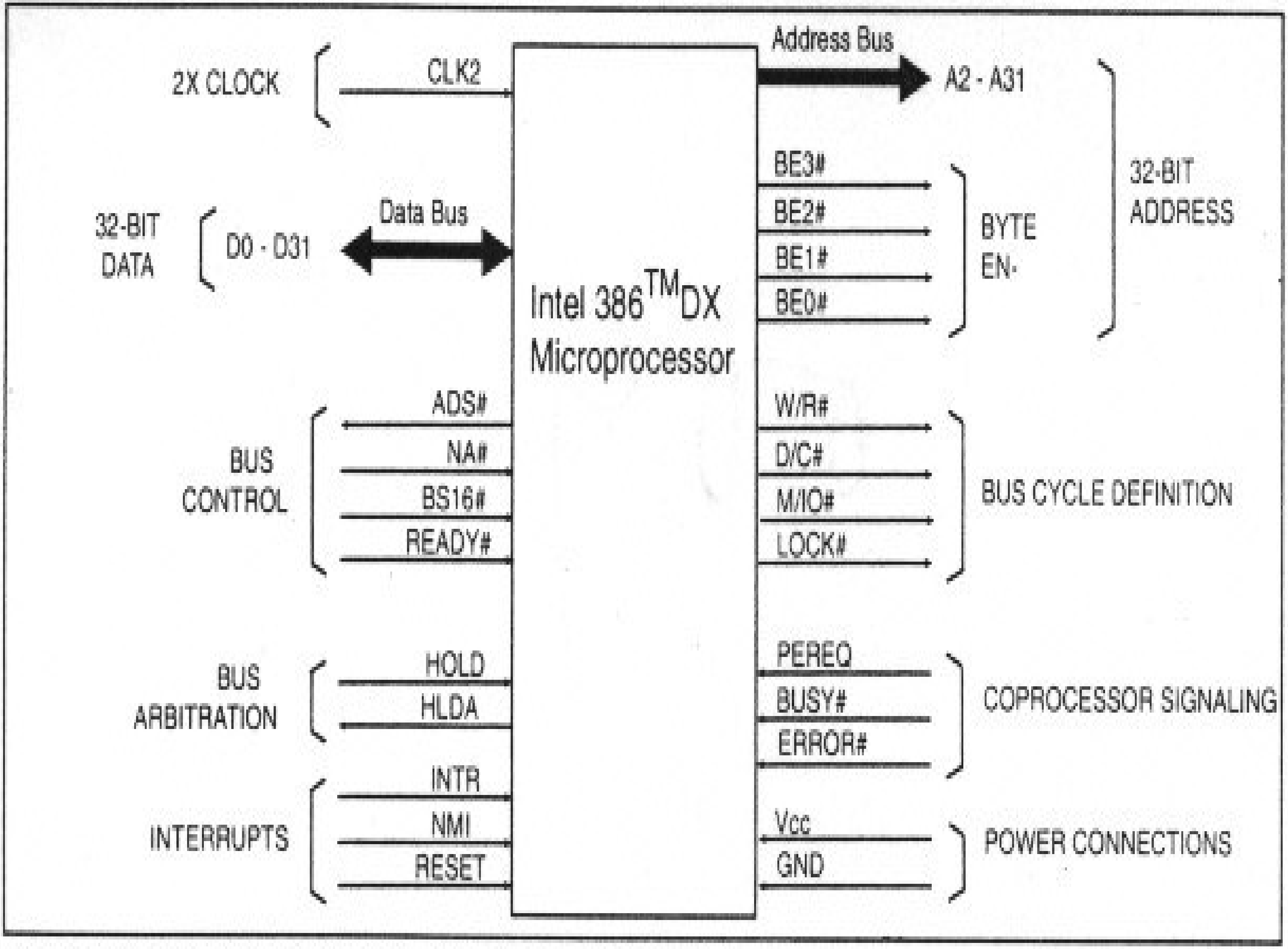
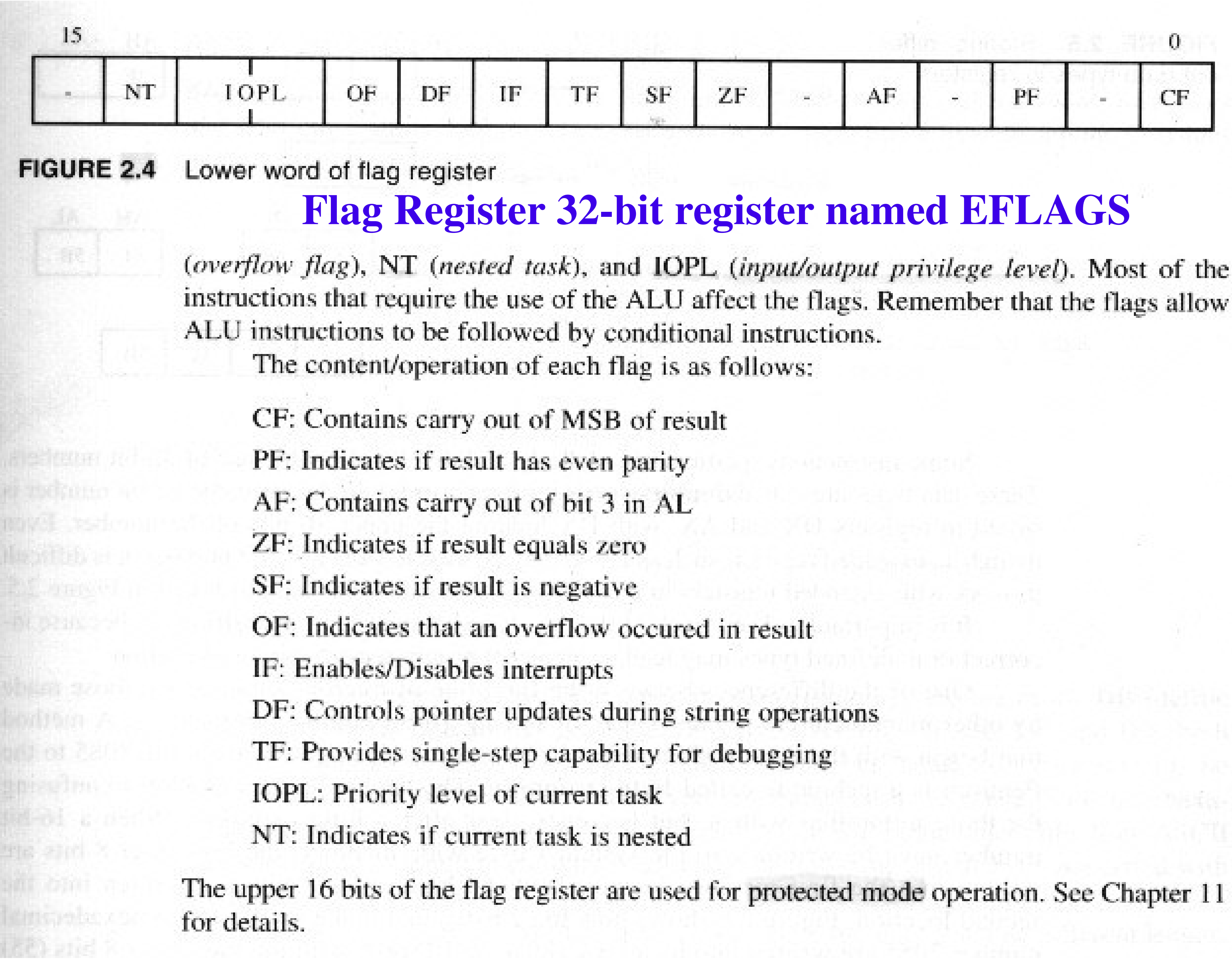


Figure 13-2. 80386 Block Diagram (# indicates active low)

(Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1992)

Flag Register 32-bit register named EFLAGS



15														0
-	NT	IOPL	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF

FIGURE 2.4 Lower word of flag register

(*overflow flag*), NT (*nested task*), and IOPL (*input/output privilege level*). Most of the instructions that require the use of the ALU affect the flags. Remember that the flags allow ALU instructions to be followed by conditional instructions.

The content/operation of each flag is as follows:

CF: Contains carry out of MSB of result

PF: Indicates if result has even parity

AF: Contains carry out of bit 3 in AL

ZF: Indicates if result equals zero

SF: Indicates if result is negative

OF: Indicates that an overflow occured in result

IF: Enables/Disables interrupts

DF: Controls pointer updates during string operations

TF: Provides single-step capability for debugging

IOPL: Priority level of current task

NT: Indicates if current task is nested

The upper 16 bits of the flag register are used for protected mode operation. See Chapter 11 for details.

System Address Registers

- Four special registers are defined to reference the tables are segments supported by 80x86 protection Model.
  - **GDT (Global Descriptor Table)**
  - **IDT (Interrupt Descriptor Table)**
  - **LDT (Local Descriptor Table)**
  - **TSS (Task State Segment**)

Other Registers

Control Registers (3)

CR0,CR2,CR3 Holdsthe m/c state

Debug Registers (6)

DR0-3	specify 4 Linear break points
DR7	To set the break point
DR6	Display the current state of break point

Test Registers (2)

TR6	Command test register
TR7	Data register which contains the data of TLB(Transition Lookaside Buffer) test

Addressing Modes

Immediate : e.g. MOV CX,1024

Register Addressing : e.g. ADD AL,BL

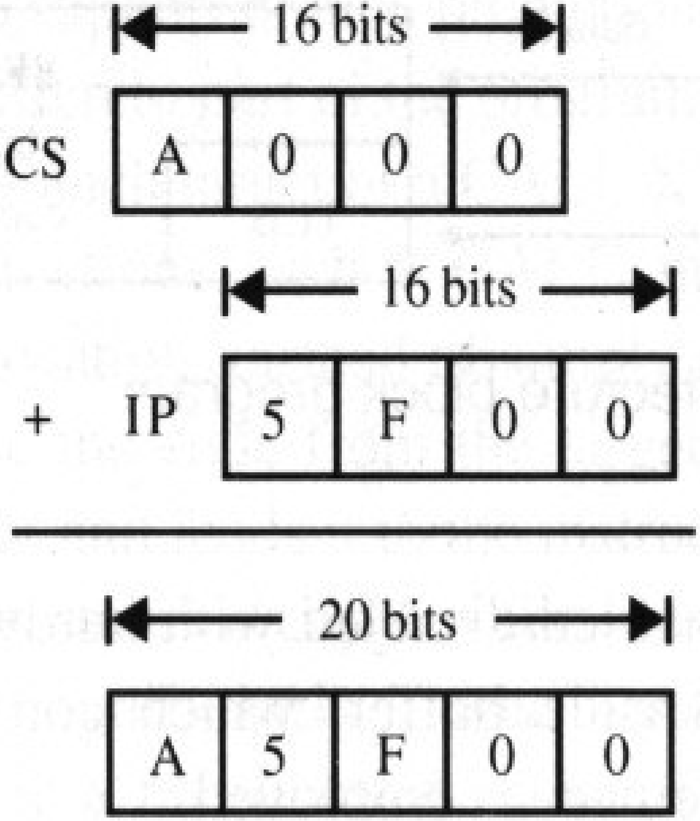
Memory Addressing :

* **Direct : e.g. MOV AX,[3000]**
* **Register Indirect : e.g. MOV BX,[SI]**
* **Based : e.g. MOV AX,[BX+4]**
* **Indexed : e.g. MOV [DI-8],BL**
* **Based Indexed : e.g. MOV [BP+SI],AH**
* **Based Indexed with Displacement :**
  e.g. MOV CL,[BX+DI+2080]
* **String Addressing :**
  e.g. MOVSB (use SI & DI as pointer)
* **Port Addressing :**
  e.g. IN AL,40
  OUT 80,AL

Instruction Types

1. **Data transfer instruction (MOV)**
2. **Arithmetic instruction (ADD)**
3. **Bit manipulation instruction (AND)**
4. **String instruction (CMPS)**
5. **Program transfer instruction (CALL)**
6. **Processor control instruction (CLC)**

FIGURE 2.3 Generating a 20-bit address in the Pentium's real mode


16 bits
CS
A
0
0
0
16 bits
+
IP
5
F
0
0
20 bits
A
5
F
0
0