

Microprocessor Technologies (102045610)

MODULE 6

Advanced Microprocessors

Outline

8086 logical block diagram

Segmentation

Minimum and maximum mode

80286/80386/80486: Overview and architecture

Basic overview of Pentium and Multicore Processors

Introduction to 8086

- 8086 is a 16-bit processor, which implies that
 - 16-bit data bus
 - 16-bit ALU
 - 16-bit registers
- 8086 has a 20-bit address bus memory can access up to 2^{20} lo ($2^{20}=1048576$ bytes =1 MB).
- It can support up to 64K I/O ports. (2^{16} I/O ports $\rightarrow 2^{16}=65536$).
- 8086 has 256 vectored interrupt.
- 8086 contains powerful instruction set, that also supports multiply and divide operation.

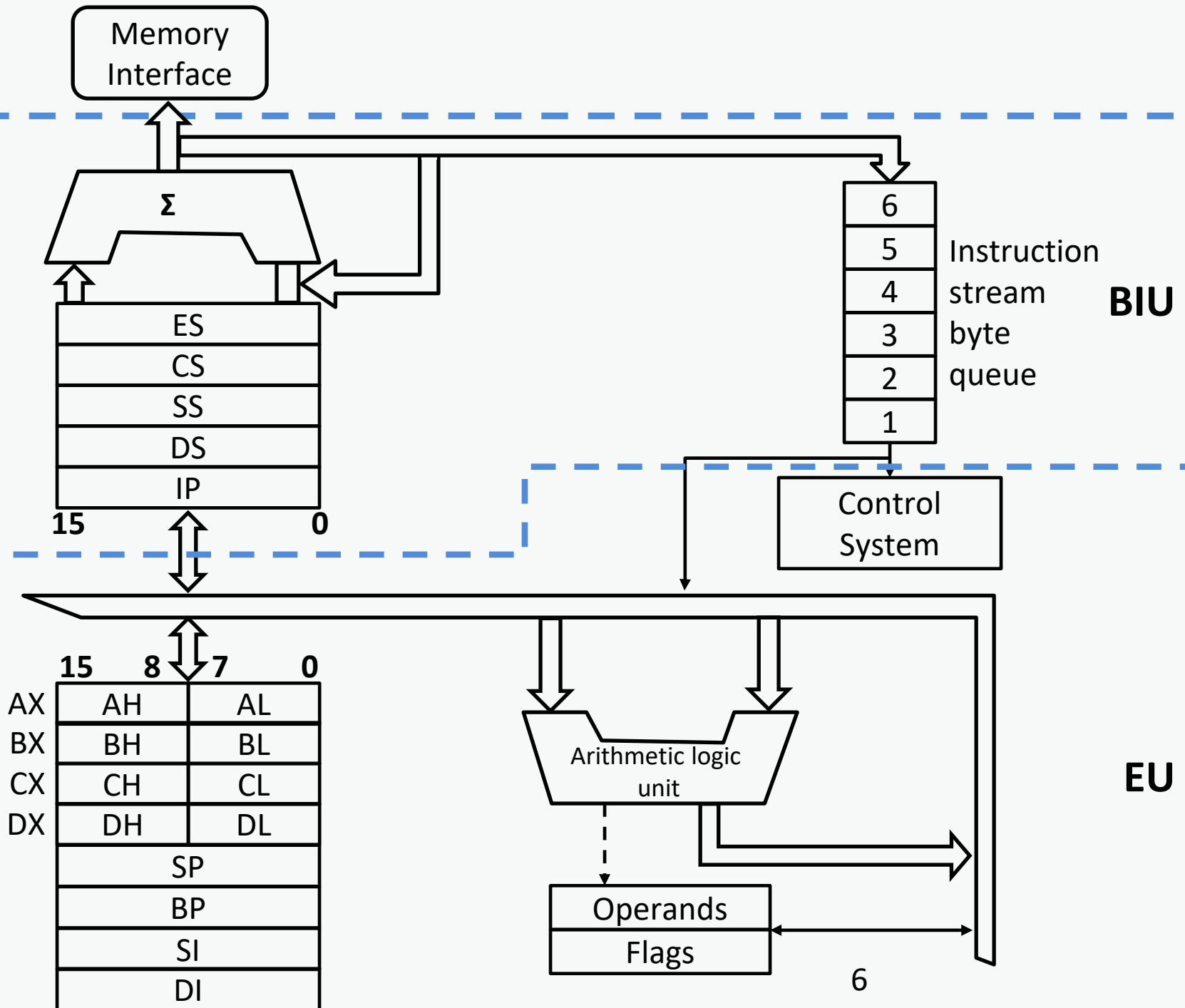
Introduction to 8086

- 8086 can operate in two modes:
 - Minimum mode:** A system with only one processor i.e. 8086
 - Maximum mode:** A system with multiple processors.
 - e.g. 8086 + math co-processor (8087), 8086 + I/O processor (8089)
- 8086 uses **memory segmentation**. Segmentation means dividing memory into logical components.
- In 8086 memory is divided into **16 segments** of capacity **2^{16}** bytes each and used as code, stack, data and extra segment respectively.

8086 Architecture

Block Diagram

8086 Architecture



8086 Architecture

- In 8086 CPU is divided into two independent functional units:

1. BIU (Bus Interface Unit)

2. EU (Execution Unit)

- Dividing the work between these two units speeds up the processing.

1) BIU (Bus Interface Unit)

Components of BIU

- **Instruction queue**
 - It holds the instruction bytes of the next instruction to be executed by EU
- **Segment Registers**
 - Four 16-bit register that provides powerful memory management mechanism
 - ES (extra segment), CS (code segment), SS (stack segment), DS (data segment).
 - The size of each register is 64kb.
- **Instruction pointer (IP)**
 - Register that holds 16-bit address or offset of next code byte within code segment
- **Address Generation and bus control**
 - Generation of 20-bit physical address

Task of BIU

1. Fetch instructions from memory.
2. Read/write instructions to/from the memory.
3. Input/output of data to/from peripheral ports.
4. Address generation for memory reference.
5. Queuing instructions.

Thus, BIU handles all transfer of data and address.

2) EU (Execution Unit)

Components of EU

- **ALU (Arithmetic logic Unit)**
 - Contains 16-bit ALU, that performs add, subtract, increment, decrement, compliment, shift binary numbers, AND, OR, XOR etc.
- **CU (Control Unit)**
 - Directs internal operation
- **Flag Register**
 - 16-bit flag register
 - EU contains 9 active flags
- **General Purpose Registers (GPR)**
 - EU has 4 general purpose 16-bit register i.e. AX, BX, CX, DX
 - Each register is the combination of two 8-bit register
 - AH, AL, BH, BL, CH, CL, DH, DL where 'L' means Lower byte and 'H' means higher byte.
- **Index Register**
 - They are 16-bit Registers
 - SI (source index) and DI (destination index).
 - Both the registers are used for string related operation and for moving block of memory from one location to the other.
- **Pointers**
 - They are 16-bit Register
 - SP (stack pointer) and BP (base pointer)
 - BP is used when we need to pass parameter through stack

- SP is always points to the top of the stack. Used for sequential access of stack segment.

- **Decoder (instruction decoder)**

- Translates the instruction fetched from memory into series of action which EU carries out

Task of EU (Execution Unit)

1. Decodes the instruction.
2. Executes decoded instructions.
3. Tells BIU from where to fetch the instruction.
4. EU takes care of performing operation on the data.
5. EU is also known as **execution heart** of the processor.

Segment Register in 8086

Segment Register in 8086

1. **Code Segment (CS):** Stores **executable** program.
2. **Data Segment (DS):** Contains **data** used by a program. Data can be accessed from this by an offset address.
3. **Stack Segment (SS):** Defines an area of memory used for the **stack**.
4. **Extra Segment (ES):** ES an additional **data** segment.

Segmentation in 8086

Segmentation in 8086

What is Segment?

An area in memory.

What is Segmentation?

The process of dividing memory into segments of various sizes is called **Segmentation**.

What is need of segmentation in 8086?

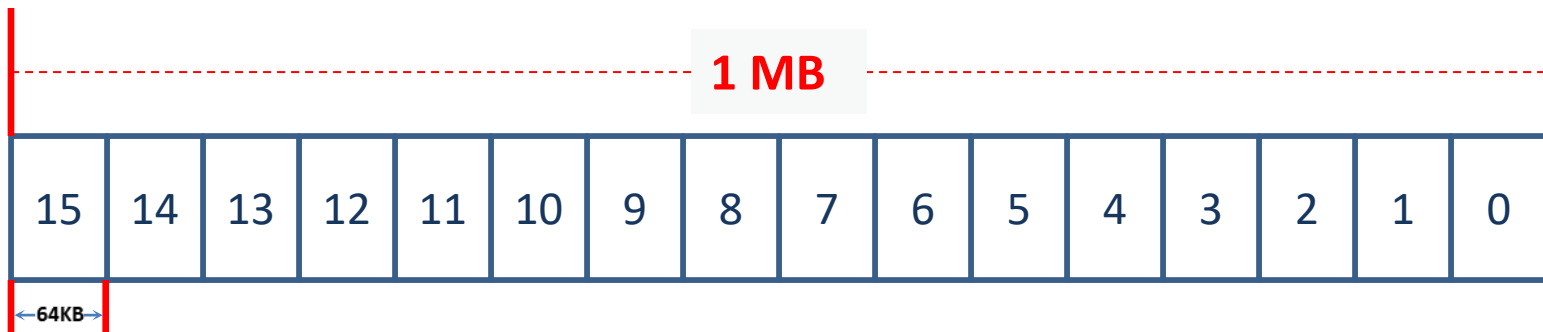
What is the need of segmentation in 8086?

Memory is huge collection of bytes.

In order to organize these bytes in an efficient manner segmentation is used.

E.g. No. of segments = $\frac{\text{Total memory available}}{\text{size of each segment}}$

$$\text{No. of segments} = \frac{1 \text{ MB}}{64 \text{ KB}} = \frac{1024 \text{ KB}}{64 \text{ KB}} = 16 \text{ segments}$$



Segmentation in 8086

- Intel 8086 has **20** lines address bus.
- With 20 address lines, the memory that can be addressed is **2^{20}** bytes.

$$2^{20} = \mathbf{1,048,576} \text{ bytes}$$

$$1 \text{ MB} = \mathbf{1111 \ 1111 \ 1111 \ 1111 \ 1111}$$

$$= \mathbf{FFFFFF \ H}$$

Segmentation in 8086

How many segments can be accessed at a time in 8086?

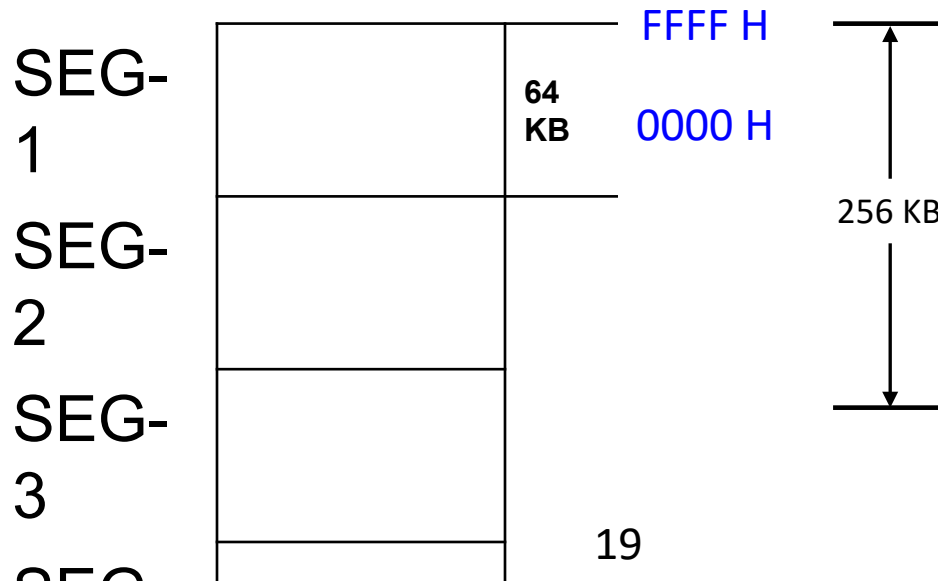
In 8086, at a time only **4 segments** can be accessed.

i.e. $64 \text{ KB} * 4 = 256 \text{ KB}$ of memory can be accessed at a time.

In 8086, memory address is ranging from **00000 H to FFFFF H**.

Size of each **Segment Register** is of **16-bit**.

$2^{16} = 65535 \text{ bytes} = 64 \text{ KB}$ [size of each segment]



Segmentation in 8086

How to calculate **physical address** from **segment address**?

- **Segment Registers** are used to hold the **upper 16-bit** of the **starting address** for each of the segment.
- The 16-bit address is starting address of the **segment** from where the **BIU** is currently fetching instruction code bytes.
- The **BIU** always inserts **zero(0)** for the **LSB(Least Significant Bit)** to generate **20-bit address**.

Segmentation in 8086

How a **20-bit physical** address can be obtained, if data bus is of 16-bit?

- 20-bit address is known as **Physical Address (PA)** of memory.
- $PA = \text{Base Address} : \text{Offset}$
- **Offset** is the **displacement** of the memory location from the starting location of the segment.

Segmentation in 8086

E.g. Base address DS=2222 H

Step-1: Convert DS 16-bit address to 20-bit address

- the BIU appends 0 H to the LSB of the base address.

22220 H

Step-2: Retrieve offset address

- Assuming offset address = 008F

H PA = Base Address :

Offset

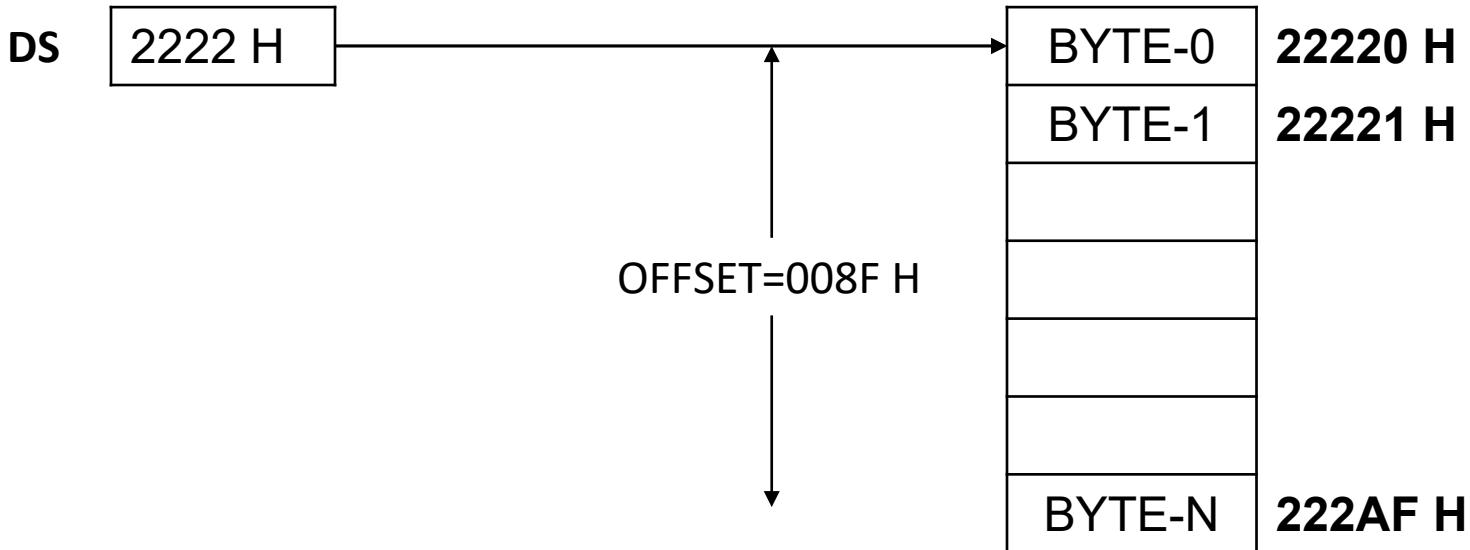
PA = 2222 H: 008F H

Segmentation in 8086

Step-3: To calculate the effective address

Physical Address = Starting Address of Segment(20-bit) + Offset

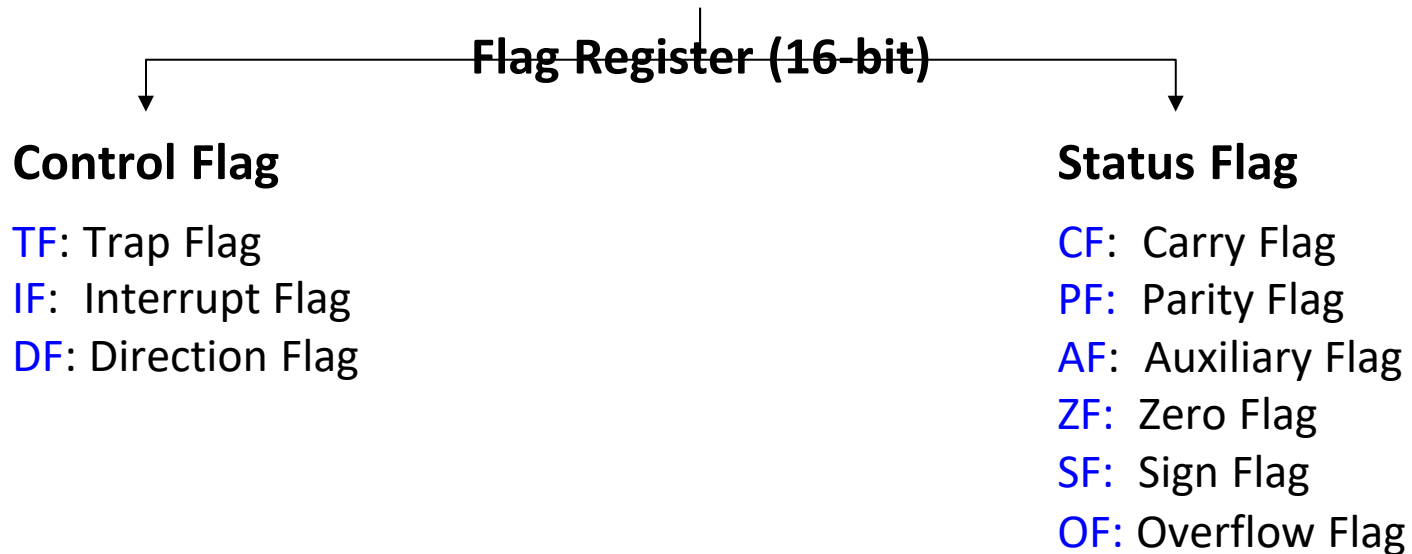
```
EA =      2 2 2 2 0 H
OFFSET= +  0 0 8 F H
          2 2 2 A F H
```



8086 Flag Register

8086 Flag Register

- The 16-bit flag register of 8086 contains 9 flags (6 conditional & 3 control flags), other 7 flags are undefined.



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF
Control Flag								Status Flag							

U- Undefined

TF- Trap Flag

IF- Interrupt Flag

DF- Direction Flag

OF- Overflow Flag

CF- Carry Flag

PF- Parity Flag

AF- Auxiliary Flag








































ZF- Zero Flag

SF- Sign Flag

8086 Flag Register

- **Carry Flag (CF):** Set(1) if arithmetic operation results in carry; otherwise reset(0).
- **Auxiliary Flag (AF):** If an operation performed in ALU generates a carry/borrow from lower nibble (i.e. $D_0 - D_3$) to upper nibble (i.e. $D_4 - D_7$), the AF flag is set i.e. carry given by D_3 bit to D_4 is AF flag. This is not a general-purpose flag, it is used internally by the processor to perform Binary to BCD conversion.
- **Parity Flag (PF):** This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity Flag is reset.
- **Zero Flag (ZF):** It is set(1), if the result of arithmetic or logical operation is zero else it is reset(0).
- **Sign Flag (SF):** In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set(1).

8086 pin diagram

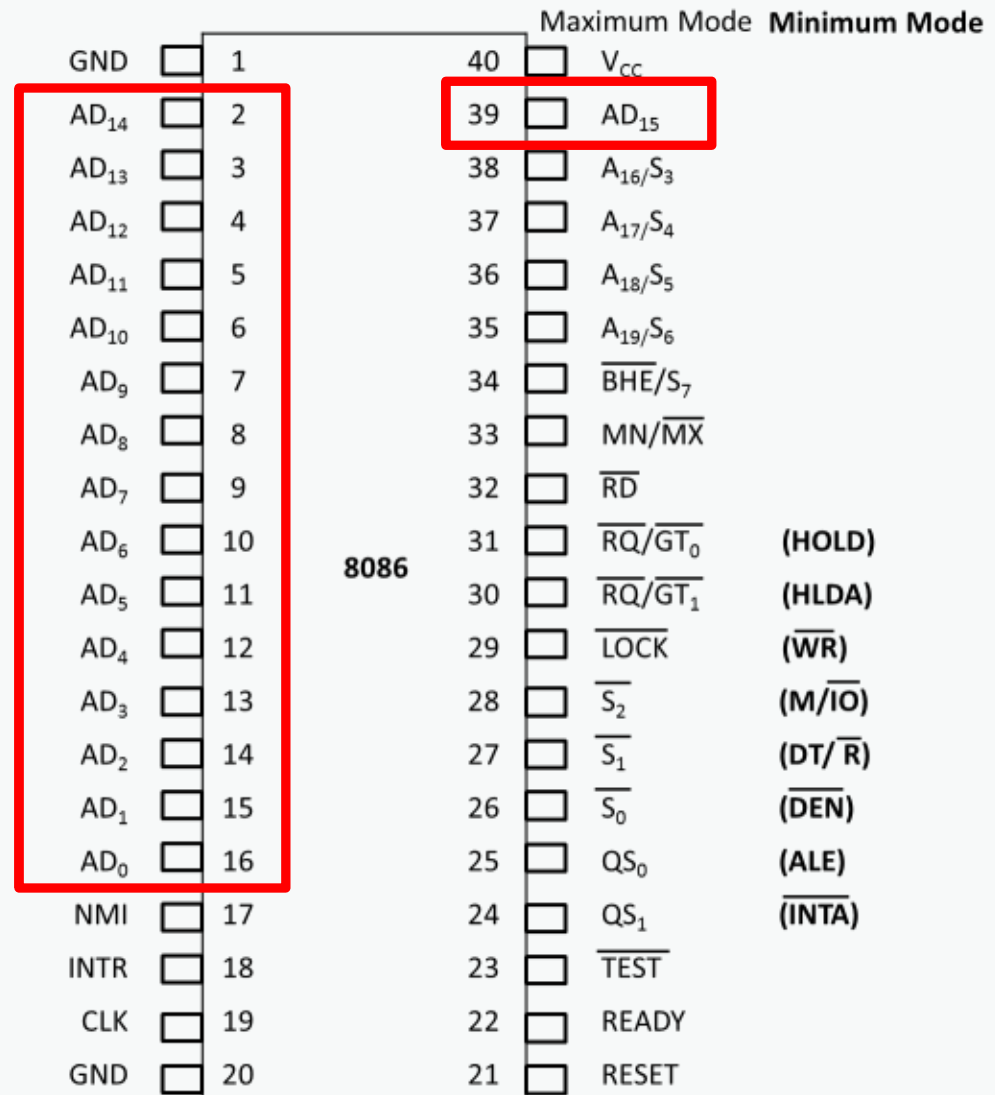
			Maximum Mode		Minimum Mode
GND		1	40		V _{CC}
AD ₁₄		2	39		AD ₁₅
AD ₁₃		3	38		A ₁₆ /S ₃
AD ₁₂		4	37		A ₁₇ /S ₄
AD ₁₁		5	36		A ₁₈ /S ₅
AD ₁₀		6	35		A ₁₉ /S ₆
AD ₉		7	34		$\overline{\text{BHE}}/\text{S}_7$
AD ₈		8	33		MN/ $\overline{\text{MX}}$
AD ₇		9	32		$\overline{\text{RD}}$
AD ₆		10	31		$\overline{\text{RQ}}/\overline{\text{GT}}_0$
					(HOLD)
AD ₅		11	30		$\overline{\text{RQ}}/\overline{\text{GT}}_1$
					(HLDA)
AD ₄		12	29		$\overline{\text{LOCK}}$
AD ₃		13	28		S ₂
AD ₂		14	27		S ₁
AD ₁		15	26		S ₀
AD ₀		16	25		$\overline{\text{QS}}_0$
NMI		17	24		QS ₁
INTR		18	23		TEST
CLK		19	22		READY

8086

29

Address and Data pins: AD_0 - AD_{15} (bidirectional)

- These lines are **bidirectional** **multiple** bus.
- AD_0 - AD_7 carry lower byte of address and order data carry higher byte of data.
- When bus enabled, Address bus will get enabled. **ALE=1**, etc



Address and Status pins: $A_{16}/S_3 - A_{19}/S_6$

- Lines are multiplexed unidirectional address and status bus.
- During T_1 , they carry higher order address.
- In the remaining clock cycles, they carry status signals.
- S_5 gives the status of controls shared system Interrupt Flag (IF)
- S_6 goes low, when the 8086 bus segment register. the
- S_3 and S_4 indicates

		Maximum Mode	Minimum Mode
GND	1	40	V_{CC}
AD_{14}	2	39	AD_{15}
AD_{13}	3	38	A_{16}/S_3
AD_{12}	4	37	A_{17}/S_4
AD_{11}	5	36	A_{18}/S_5
AD_{10}	6	35	A_{19}/S_6
AD_9	7	34	BHE/S_7
AD_8	8	33	MN/\overline{MX}
AD_7	9	32	\overline{RD}
AD_6	10	31	$\overline{RQ}/\overline{GT_0}$ (HOLD)
AD_5	11	30	$\overline{RQ}/\overline{GT_1}$ (HLDA)
AD_4	12	29	\overline{LOCK} (\overline{WR})
AD_3	13	28	$\overline{S_2}$ (M/ \overline{IO})
AD_2	14	27	$\overline{S_1}$ (DT/ \overline{R})
AD_1	15	26	$\overline{S_0}$ (\overline{DEN})
AD_0	16	25	QS_0 (ALE)
NMI	17	24	QS_1 (\overline{INTA})
INTR	18	23	\overline{TEST}
CLK	19	22	READY
GND	20	21	RESET

Status pins

Below table indicates, which segment will be accessed on the basis of S_4 & S_3 bit value.

S_4	S_3	Register
0	0	ES
0	1	SS
1	0	CS
1	1	DS

BHE/S₇

- **BHE** stands for **Bus Enable** is **High** active low output signal.
- **BHE** signal is used to indicate the transfer over higher order data (D₈ - D₁₅).
- **8-bit** I/O devices use this signal.
- **S₇** is reserved for future development.

				Maximum Mode	Minimum Mode
GND	1	40	V _{CC}		
AD ₁₄	2	39	AD ₁₅		
AD ₁₃	3	38	A ₁₆ /S ₃		
AD ₁₂	4	37	A ₁₇ /S ₄		
AD ₁₁	5	36	A ₁₈ /S ₅		
AD ₁₀	6	35	A ₁₉ /S ₆		
AD ₉	7	34	BHE/S₇		
AD ₈	8	33	MN/MX		
AD ₇	9	32	RD		
AD ₆	10	31	RQ/GT ₀	(HOLD)	
AD ₅	11	30	RQ/GT ₁	(HLDA)	
AD ₄	12	29	LOCK	(WR)	
AD ₃	13	28	S ₂	(M/I _O)	
AD ₂	14	27	S ₁	(DT/R)	
AD ₁	15	26	S ₀	(DEN)	
AD ₀	16	25	QS ₀	(ALE)	
NMI	17	24	QS ₁	(INTA)	
INTR	18	23	TEST		
CLK	19	22	READY		
GND	20	21	RESET		

Interrupt Pins

NMI

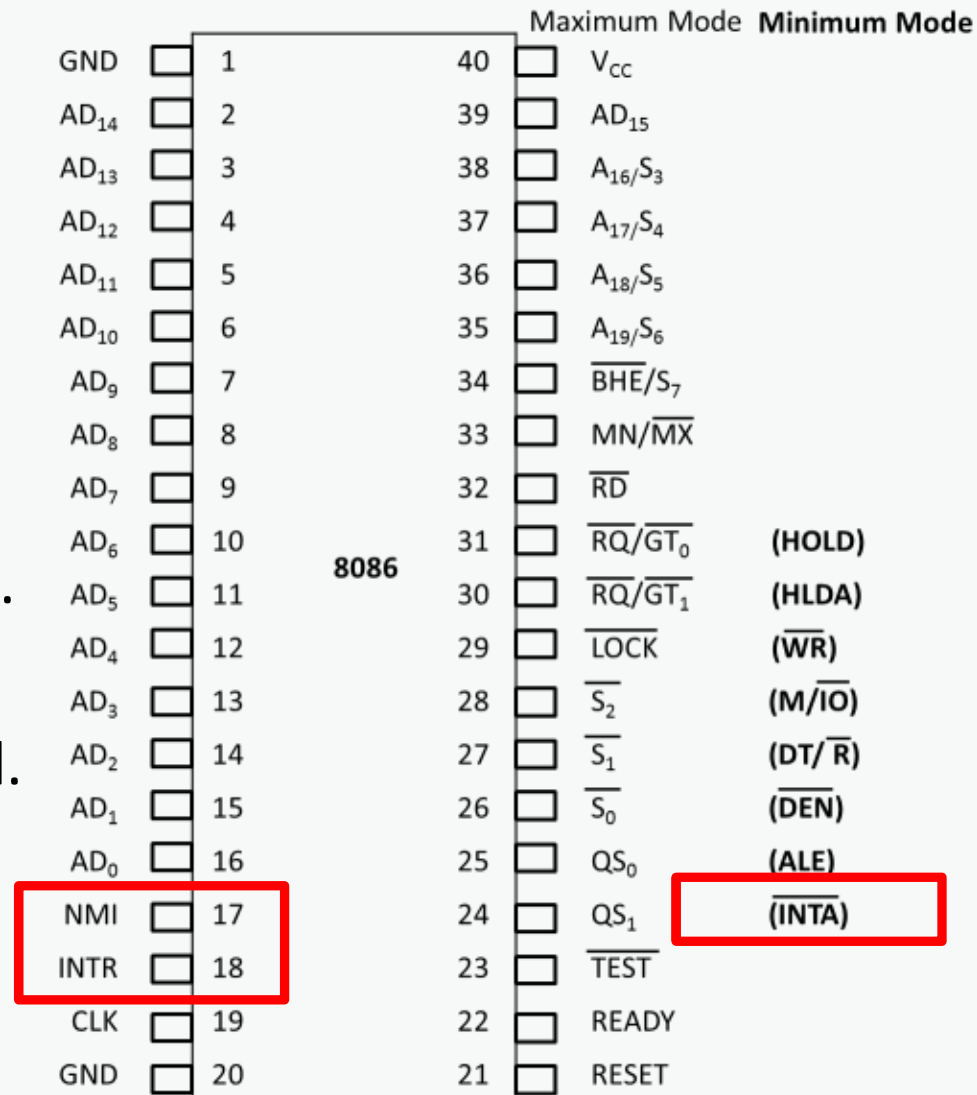
- It is an active high input signal.
- It is a **non-maskable** interrupt signal.

INTR

- It is an active high input signal
- It is an interrupt **request** signal.

INTA

- It is an active low output signal.
- This is an interrupt **acknowledge** signal.



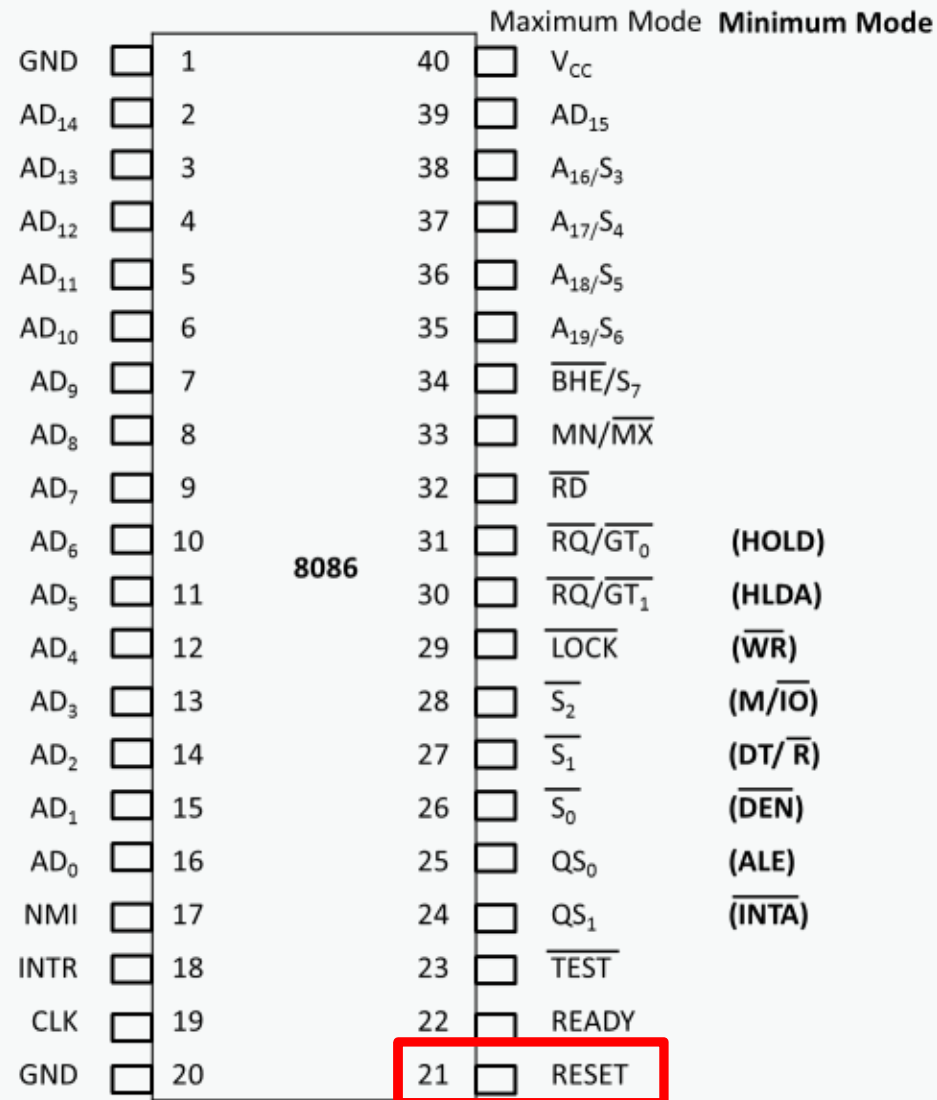
Clock Pin: CLK

Generates clock signals that synchronizes the operation of processor.

				Maximum Mode	Minimum Mode
GND	<input type="checkbox"/>	1	40	<input type="checkbox"/> V_{CC}	
AD ₁₄	<input type="checkbox"/>	2	39	<input type="checkbox"/> AD ₁₅	
AD ₁₃	<input type="checkbox"/>	3	38	<input type="checkbox"/> A ₁₆ /S ₃	
AD ₁₂	<input type="checkbox"/>	4	37	<input type="checkbox"/> A ₁₇ /S ₄	
AD ₁₁	<input type="checkbox"/>	5	36	<input type="checkbox"/> A ₁₈ /S ₅	
AD ₁₀	<input type="checkbox"/>	6	35	<input type="checkbox"/> A ₁₉ /S ₆	
AD ₉	<input type="checkbox"/>	7	34	<input type="checkbox"/> \overline{BHE}/S_7	
AD ₈	<input type="checkbox"/>	8	33	<input type="checkbox"/> MN/ \overline{MX}	
AD ₇	<input type="checkbox"/>	9	32	<input type="checkbox"/> \overline{RD}	
AD ₆	<input type="checkbox"/>	10	31	<input type="checkbox"/> $\overline{RQ}/\overline{GT_0}$	(HOLD)
AD ₅	<input type="checkbox"/>	11	30	<input type="checkbox"/> $\overline{RQ}/\overline{GT_1}$	(HLDA)
AD ₄	<input type="checkbox"/>	12	29	<input type="checkbox"/> \overline{LOCK}	(\overline{WR})
AD ₃	<input type="checkbox"/>	13	28	<input type="checkbox"/> $\overline{S_2}$	(M/ \overline{IO})
AD ₂	<input type="checkbox"/>	14	27	<input type="checkbox"/> $\overline{S_1}$	(DT/ \overline{R})
AD ₁	<input type="checkbox"/>	15	26	<input type="checkbox"/> $\overline{S_0}$	(\overline{DEN})
AD ₀	<input type="checkbox"/>	16	25	<input type="checkbox"/> QS ₀	(ALE)
NMI	<input type="checkbox"/>	17	24	<input type="checkbox"/> QS ₁	(\overline{INTA})
INTR	<input type="checkbox"/>	18	23	<input type="checkbox"/> \overline{TEST}	
CLK	<input type="checkbox"/>	19	22	<input type="checkbox"/> READY	
GND	<input type="checkbox"/>	20	21	<input type="checkbox"/> RESET	

Clock Pin: RESET

- Active **high** input signal. **high**, microprocessor enters into **reset state** and terminates all the activities of processor.
- Processor requires **4 clock cycles** to reset.
- Thus, RESET signal must be **1** for at least **4 clock cycles**.



Clock Pin: **READY**

- Active high input signal is an I/O devices or memory from
- When high, it indicates that device is ready to transfer data.
- When low, microprocessor is in wait state.

				Maximum Mode	Minimum Mode
GND	<input type="checkbox"/>	1	40	<input type="checkbox"/> V_{CC}	
AD ₁₄	<input type="checkbox"/>	2	39	<input type="checkbox"/> AD ₁₅	
AD ₁₃	<input type="checkbox"/>	3	38	<input type="checkbox"/> A ₁₆ /S ₃	
AD ₁₂	<input type="checkbox"/>	4	37	<input type="checkbox"/> A ₁₇ /S ₄	
AD ₁₁	<input type="checkbox"/>	5	36	<input type="checkbox"/> A ₁₈ /S ₅	
AD ₁₀	<input type="checkbox"/>	6	35	<input type="checkbox"/> A ₁₉ /S ₆	
AD ₉	<input type="checkbox"/>	7	34	<input type="checkbox"/> \overline{BHE}/S_7	
AD ₈	<input type="checkbox"/>	8	33	<input type="checkbox"/> MN/ \overline{MX}	
AD ₇	<input type="checkbox"/>	9	32	<input type="checkbox"/> \overline{RD}	
AD ₆	<input type="checkbox"/>	10	31	<input type="checkbox"/> $\overline{RQ}/\overline{GT_0}$	(HOLD)
AD ₅	<input type="checkbox"/>	11	30	<input type="checkbox"/> $\overline{RQ}/\overline{GT_1}$	(HLDA)
AD ₄	<input type="checkbox"/>	12	29	<input type="checkbox"/> \overline{LOCK}	(\overline{WR})
AD ₃	<input type="checkbox"/>	13	28	<input type="checkbox"/> $\overline{S_2}$	(M/ \overline{IO})
AD ₂	<input type="checkbox"/>	14	27	<input type="checkbox"/> $\overline{S_1}$	(DT/ \overline{R})
AD ₁	<input type="checkbox"/>	15	26	<input type="checkbox"/> $\overline{S_0}$	(\overline{DEN})
AD ₀	<input type="checkbox"/>	16	25	<input type="checkbox"/> QS ₀	(ALE)
NMI	<input type="checkbox"/>	17	24	<input type="checkbox"/> QS ₁	(\overline{INTA})
INTR	<input type="checkbox"/>	18	23	<input type="checkbox"/> \overline{TEST}	
CLK	<input type="checkbox"/>	19	22	<input type="checkbox"/> READY	
GND	<input type="checkbox"/>	20	21	<input type="checkbox"/> RESET	

Control Pin: **TEST**

- Active **low** input signal.
- It is used to test the **status** of **math coprocessor 8087**.
- The **BUSY** pin of 8087 is connected to this pin of 8086.
- If **low**, execution continues, else microprocessor is in idle state.

				Maximum Mode	Minimum Mode
GND	<input type="checkbox"/>	1	40	<input type="checkbox"/> V_{CC}	
AD ₁₄	<input type="checkbox"/>	2	39	<input type="checkbox"/> AD ₁₅	
AD ₁₃	<input type="checkbox"/>	3	38	<input type="checkbox"/> A ₁₆ /S ₃	
AD ₁₂	<input type="checkbox"/>	4	37	<input type="checkbox"/> A ₁₇ /S ₄	
AD ₁₁	<input type="checkbox"/>	5	36	<input type="checkbox"/> A ₁₈ /S ₅	
AD ₁₀	<input type="checkbox"/>	6	35	<input type="checkbox"/> A ₁₉ /S ₆	
AD ₉	<input type="checkbox"/>	7	34	<input type="checkbox"/> \overline{BHE}/S_7	
AD ₈	<input type="checkbox"/>	8	33	<input type="checkbox"/> MN/ \overline{MX}	
AD ₇	<input type="checkbox"/>	9	32	<input type="checkbox"/> \overline{RD}	
AD ₆	<input type="checkbox"/>	10	31	<input type="checkbox"/> $\overline{RQ}/\overline{GT_0}$	(HOLD)
AD ₅	<input type="checkbox"/>	11	30	<input type="checkbox"/> $\overline{RQ}/\overline{GT_1}$	(HLDA)
AD ₄	<input type="checkbox"/>	12	29	<input type="checkbox"/> \overline{LOCK}	(\overline{WR})
AD ₃	<input type="checkbox"/>	13	28	<input type="checkbox"/> $\overline{S_2}$	(M/ \overline{IO})
AD ₂	<input type="checkbox"/>	14	27	<input type="checkbox"/> $\overline{S_1}$	(DT/ \overline{R})
AD ₁	<input type="checkbox"/>	15	26	<input type="checkbox"/> $\overline{S_0}$	(\overline{DEN})
AD ₀	<input type="checkbox"/>	16	25	<input type="checkbox"/> QS ₀	(ALE)
NMI	<input type="checkbox"/>	17	24	<input type="checkbox"/> QS ₁	(\overline{INTA})
INTR	<input type="checkbox"/>	18	23	<input type="checkbox"/> TEST	
CLK	<input type="checkbox"/>	19	22	<input type="checkbox"/> READY	
GND	<input type="checkbox"/>	20	21	<input type="checkbox"/> RESET	

Control Pin: $\overline{MN}/\overline{MX}$

8086 works in two

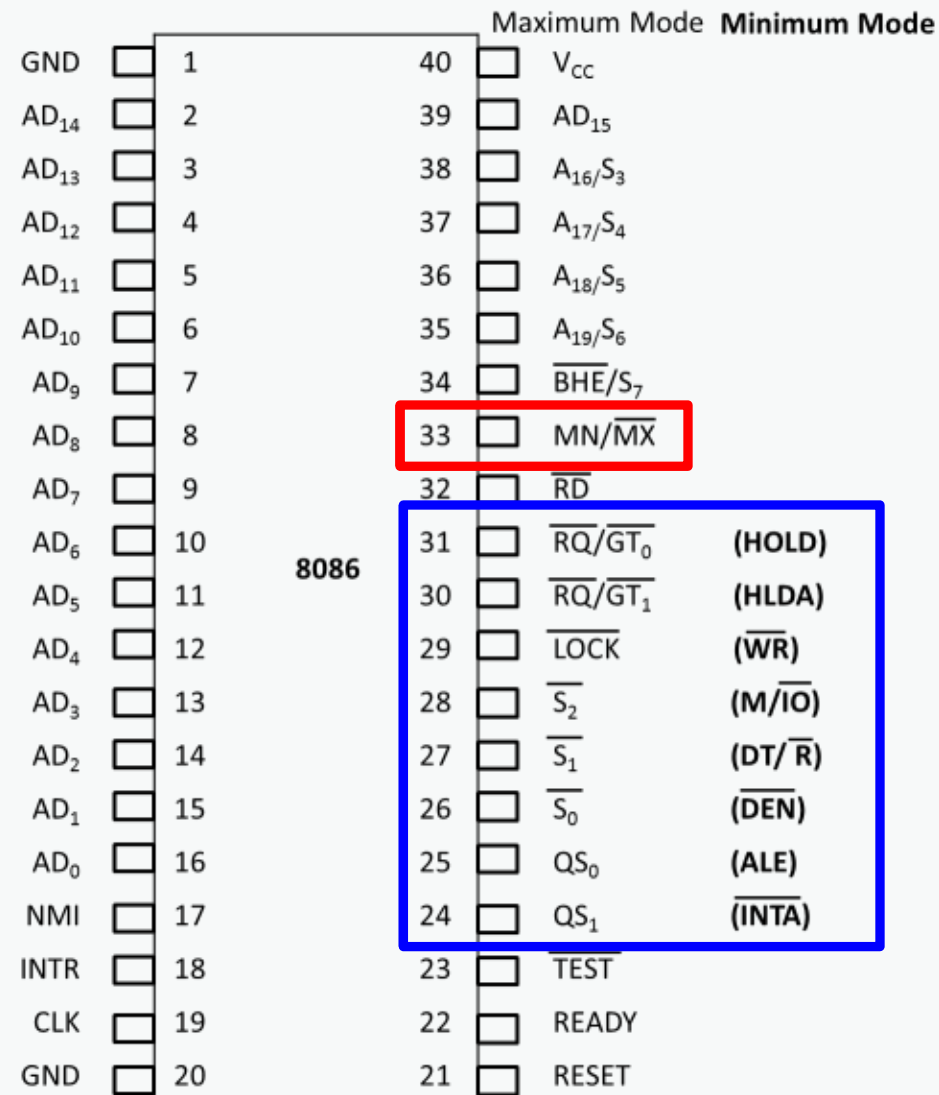
1. **Minimum** Mode (Active high

i/p signal)

2. **Maximum** Mode (Active low

i/p signal)

- Pins from **24** to **31**
- One set of signals is issued when CPU operates in **minimum** mode, while other is issued when CPU operates in **maximum** mode.



Mode Multiplexed pins: $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Indication
0	0	0	Interrupt acknowledgement
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	HALT
1	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

				Maximum Mode	Minimum Mode
GND	<input type="checkbox"/>	1	40	<input type="checkbox"/> V_{CC}	
AD ₁₄	<input type="checkbox"/>	2	39	<input type="checkbox"/> AD ₁₅	
AD ₁₃	<input type="checkbox"/>	3	38	<input type="checkbox"/> A ₁₆ /S ₃	
AD ₁₂	<input type="checkbox"/>	4	37	<input type="checkbox"/> A ₁₇ /S ₄	
AD ₁₁	<input type="checkbox"/>	5	36	<input type="checkbox"/> A ₁₈ /S ₅	
AD ₁₀	<input type="checkbox"/>	6	35	<input type="checkbox"/> A ₁₉ /S ₆	
AD ₉	<input type="checkbox"/>	7	34	<input type="checkbox"/> \overline{BHE}/S_7	
AD ₈	<input type="checkbox"/>	8	33	<input type="checkbox"/> MN/ \overline{MX}	
AD ₇	<input type="checkbox"/>	9	32	<input type="checkbox"/> \overline{RD}	
AD ₆	<input type="checkbox"/>	10	31	<input type="checkbox"/> $\overline{RQ}/\overline{GT_0}$	(HOLD)
AD ₅	<input type="checkbox"/>	11	30	<input type="checkbox"/> $\overline{RQ}/\overline{GT_1}$	(HLDA)
AD ₄	<input type="checkbox"/>	12	29	<input type="checkbox"/> \overline{LOCK}	(\overline{WR})
AD ₃	<input type="checkbox"/>	13	28	<input type="checkbox"/> $\overline{S_2}$	(M/ \overline{IO})
AD ₂	<input type="checkbox"/>	14	27	<input type="checkbox"/> $\overline{S_1}$	(DT/ \overline{R})
AD ₁	<input type="checkbox"/>	15	26	<input type="checkbox"/> $\overline{S_0}$	(\overline{DEN})
AD ₀	<input type="checkbox"/>	16	25	<input type="checkbox"/> QS ₀	(ALE)
NMI	<input type="checkbox"/>	17	24	<input type="checkbox"/> QS ₁	(\overline{INTA})
INTR	<input type="checkbox"/>	18	23	<input type="checkbox"/> \overline{TEST}	
CLK	<input type="checkbox"/>	19	22	<input type="checkbox"/> READY	
GND	<input type="checkbox"/>	20	21	<input type="checkbox"/> RESET	

8086

Mode Multiplexed pins

DEN

- Active **low output** signal.
- This is a **Data Enable signal**, which is used to **enable** the **transceiver**.

DT/R

- Data **Transmit/Receive** signal.
- When **high**, data is transmitted **out** else when low, data is received **in**.

				Maximum Mode	Minimum Mode
GND	<input type="checkbox"/>	1	40	<input type="checkbox"/> V _{CC}	
AD ₁₄	<input type="checkbox"/>	2	39	<input type="checkbox"/> AD ₁₅	
AD ₁₃	<input type="checkbox"/>	3	38	<input type="checkbox"/> A ₁₆ /S ₃	
AD ₁₂	<input type="checkbox"/>	4	37	<input type="checkbox"/> A ₁₇ /S ₄	
AD ₁₁	<input type="checkbox"/>	5	36	<input type="checkbox"/> A ₁₈ /S ₅	
AD ₁₀	<input type="checkbox"/>	6	35	<input type="checkbox"/> A ₁₉ /S ₆	
AD ₉	<input type="checkbox"/>	7	34	<input type="checkbox"/> BHE/S ₇	
AD ₈	<input type="checkbox"/>	8	33	<input type="checkbox"/> MN/ $\overline{\text{MX}}$	
AD ₇	<input type="checkbox"/>	9	32	<input type="checkbox"/> $\overline{\text{RD}}$	
AD ₆	<input type="checkbox"/>	10	31	<input type="checkbox"/> $\overline{\text{RQ}}/\overline{\text{GT}}_0$	(HOLD)
AD ₅	<input type="checkbox"/>	11	30	<input type="checkbox"/> $\overline{\text{RQ}}/\overline{\text{GT}}_1$	(HLDA)
AD ₄	<input type="checkbox"/>	12	29	<input type="checkbox"/> $\overline{\text{LOCK}}$	($\overline{\text{WR}}$)
AD ₃	<input type="checkbox"/>	13	28	<input type="checkbox"/> $\overline{\text{S}}_2$	(M/ $\overline{\text{IO}}$)
AD ₂	<input type="checkbox"/>	14	27	<input type="checkbox"/> $\overline{\text{S}}_1$	(DT/ $\overline{\text{R}}$)
AD ₁	<input type="checkbox"/>	15	26	<input type="checkbox"/> $\overline{\text{S}}_0$	($\overline{\text{DEN}}$)
AD ₀	<input type="checkbox"/>	16	25	<input type="checkbox"/> QS ₀	(ALE)
NMI	<input type="checkbox"/>	17	24	<input type="checkbox"/> QS ₁	($\overline{\text{INTA}}$)
INTR	<input type="checkbox"/>	18	23	<input type="checkbox"/> $\overline{\text{TEST}}$	
CLK	<input type="checkbox"/>	19	22	<input type="checkbox"/> READY	
GND	<input type="checkbox"/>	20	21	<input type="checkbox"/> RESET	

Mode Multiplexed pins: M/\overline{IO}

- This signal is issued by microprocessor to memory distinguish from access. I/O
- When accessed, high, else memory when low, I/O devices are accessed.

				Maximum Mode	Minimum Mode
GND	<input type="checkbox"/>	1	40	<input type="checkbox"/> V_{CC}	
AD_{14}	<input type="checkbox"/>	2	39	<input type="checkbox"/> AD_{15}	
AD_{13}	<input type="checkbox"/>	3	38	<input type="checkbox"/> A_{16}/S_3	
AD_{12}	<input type="checkbox"/>	4	37	<input type="checkbox"/> A_{17}/S_4	
AD_{11}	<input type="checkbox"/>	5	36	<input type="checkbox"/> A_{18}/S_5	
AD_{10}	<input type="checkbox"/>	6	35	<input type="checkbox"/> A_{19}/S_6	
AD_9	<input type="checkbox"/>	7	34	<input type="checkbox"/> \overline{BHE}/S_7	
AD_8	<input type="checkbox"/>	8	33	<input type="checkbox"/> MN/\overline{MX}	
AD_7	<input type="checkbox"/>	9	32	<input type="checkbox"/> \overline{RD}	
AD_6	<input type="checkbox"/>	10	31	<input type="checkbox"/> $\overline{RQ}/\overline{GT_0}$	(HOLD)
AD_5	<input type="checkbox"/>	11	30	<input type="checkbox"/> $\overline{RQ}/\overline{GT_1}$	(HLDA)
AD_4	<input type="checkbox"/>	12	29	<input type="checkbox"/> \overline{LOCK}	(\overline{WR})
AD_3	<input type="checkbox"/>	13	28	<input type="checkbox"/> $\overline{S_2}$	(M/\overline{IO})
AD_2	<input type="checkbox"/>	14	27	<input type="checkbox"/> $\overline{S_1}$	(DT/\overline{R})
AD_1	<input type="checkbox"/>	15	26	<input type="checkbox"/> $\overline{S_0}$	(\overline{DEN})
AD_0	<input type="checkbox"/>	16	25	<input type="checkbox"/> QS_0	(ALE)
NMI	<input type="checkbox"/>	17	24	<input type="checkbox"/> QS_1	(\overline{INTA})
INTR	<input type="checkbox"/>	18	23	<input type="checkbox"/> \overline{TEST}	
CLK	<input type="checkbox"/>	19	22	<input type="checkbox"/> READY	
GND	<input type="checkbox"/>	20	21	<input type="checkbox"/> RESET	

Mode Multiplexed pins: QS_1 and QS_0

QS_1 QS_0 Indication

0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

				Maximum Mode	Minimum Mode
GND	<input type="checkbox"/>	1	40	<input type="checkbox"/> V_{CC}	
AD_{14}	<input type="checkbox"/>	2	39	<input type="checkbox"/> AD_{15}	
AD_{13}	<input type="checkbox"/>	3	38	<input type="checkbox"/> A_{16}/S_3	
AD_{12}	<input type="checkbox"/>	4	37	<input type="checkbox"/> A_{17}/S_4	
AD_{11}	<input type="checkbox"/>	5	36	<input type="checkbox"/> A_{18}/S_5	
AD_{10}	<input type="checkbox"/>	6	35	<input type="checkbox"/> A_{19}/S_6	
AD_9	<input type="checkbox"/>	7	34	<input type="checkbox"/> \overline{BHE}/S_7	
AD_8	<input type="checkbox"/>	8	33	<input type="checkbox"/> MN/\overline{MX}	
AD_7	<input type="checkbox"/>	9	32	<input type="checkbox"/> \overline{RD}	
AD_6	<input type="checkbox"/>	10	31	<input type="checkbox"/> $\overline{RQ}/\overline{GT_0}$	(HOLD)
AD_5	<input type="checkbox"/>	11	30	<input type="checkbox"/> $\overline{RQ}/\overline{GT_1}$	(HLDA)
AD_4	<input type="checkbox"/>	12	29	<input type="checkbox"/> \overline{LOCK}	(\overline{WR})
AD_3	<input type="checkbox"/>	13	28	<input type="checkbox"/> $\overline{S_2}$	(M/\overline{IO})
AD_2	<input type="checkbox"/>	14	27	<input type="checkbox"/> $\overline{S_1}$	(DT/\overline{R})
AD_1	<input type="checkbox"/>	15	26	<input type="checkbox"/> $\overline{S_0}$	(\overline{DEN})
AD_0	<input type="checkbox"/>	16	25	<input type="checkbox"/> QS_0	(ALE)
NMI	<input type="checkbox"/>	17	24	<input type="checkbox"/> QS_1	(\overline{INTA})
INTR	<input type="checkbox"/>	18	23	<input type="checkbox"/> TEST	
CLK	<input type="checkbox"/>	19	22	<input type="checkbox"/> READY	
GND	<input type="checkbox"/>	20	21	<input type="checkbox"/> RESET	

8086

Mode Multiplexed pins

LOCK

- Active **low** output signal.
- This signal indicates that other processors should not ask CPU (8086) to hand over the **system bus**.
- This pin is activated by using **LOCK** prefix to any instruction.

WR

- Active **low** output signal.
- Used to write data in memory or output signal, depending on

				Maximum Mode	Minimum Mode
GND	<input type="checkbox"/>	1	40	<input type="checkbox"/>	V _{CC}
AD ₁₄	<input type="checkbox"/>	2	39	<input type="checkbox"/>	AD ₁₅
AD ₁₃	<input type="checkbox"/>	3	38	<input type="checkbox"/>	A ₁₆ /S ₃
AD ₁₂	<input type="checkbox"/>	4	37	<input type="checkbox"/>	A ₁₇ /S ₄
AD ₁₁	<input type="checkbox"/>	5	36	<input type="checkbox"/>	A ₁₈ /S ₅
AD ₁₀	<input type="checkbox"/>	6	35	<input type="checkbox"/>	A ₁₉ /S ₆
AD ₉	<input type="checkbox"/>	7	34	<input type="checkbox"/>	$\overline{\text{BHE}}/\text{S}_7$
AD ₈	<input type="checkbox"/>	8	33	<input type="checkbox"/>	MN/ $\overline{\text{MX}}$
AD ₇	<input type="checkbox"/>	9	32	<input type="checkbox"/>	$\overline{\text{RD}}$
AD ₆	<input type="checkbox"/>	10	31	<input type="checkbox"/>	$\overline{\text{RQ}}/\overline{\text{GT}}_0$ (HOLD)
AD ₅	<input type="checkbox"/>	11	30	<input type="checkbox"/>	$\overline{\text{RQ}}/\overline{\text{GT}}_1$ (HLDA)
AD ₄	<input type="checkbox"/>	12	29	<input type="checkbox"/>	$\overline{\text{LOCK}}$ ($\overline{\text{WR}}$)
AD ₃	<input type="checkbox"/>	13	28	<input type="checkbox"/>	$\overline{\text{S}}_2$ (M/ $\overline{\text{IO}}$)
AD ₂	<input type="checkbox"/>	14	27	<input type="checkbox"/>	$\overline{\text{S}}_1$ (DT/ $\overline{\text{R}}$)
AD ₁	<input type="checkbox"/>	15	26	<input type="checkbox"/>	$\overline{\text{S}}_0$ ($\overline{\text{DEN}}$)
AD ₀	<input type="checkbox"/>	16	25	<input type="checkbox"/>	QS ₀ (ALE)
NMI	<input type="checkbox"/>	17	24	<input type="checkbox"/>	QS ₁ ($\overline{\text{INTA}}$)
INTR	<input type="checkbox"/>	18	23	<input type="checkbox"/>	$\overline{\text{TEST}}$
CLK	<input type="checkbox"/>	19	22	<input type="checkbox"/>	READY
GND	<input type="checkbox"/>	20	21	<input type="checkbox"/>	RESET

Mode Multiplexed pins

HOLD

- Active **high** input signal.
- When **DMA controller** needs to use address/data bus, it sends a request to the CPU through this pin.
- When microprocessor receives **HOLD** signal, it issues **HLDA** signal to the DMA controller.
- Active **high** **HLDA** Acknowledgement signal.
- It is issued by **8086** after receiving **HOLD** signal.

				Maximum Mode	Minimum Mode
GND	1	40	V _{CC}		
AD ₁₄	2	39	AD ₁₅		
AD ₁₃	3	38	A ₁₆ /S ₃		
AD ₁₂	4	37	A ₁₇ /S ₄		
AD ₁₁	5	36	A ₁₈ /S ₅		
AD ₁₀	6	35	A ₁₉ /S ₆		
AD ₉	7	34	$\overline{\text{BHE}}/\text{S}_7$		
AD ₈	8	33	MN/ $\overline{\text{MX}}$		
AD ₇	9	32	$\overline{\text{RD}}$		
AD ₆	10	31	$\overline{\text{RQ}}/\overline{\text{GT}}_0$		(HOLD)
AD ₅	11	30	$\overline{\text{RQ}}/\overline{\text{GT}}_1$		(HLDA)
AD ₄	12	29	$\overline{\text{LOCK}}$		($\overline{\text{WR}}$)
AD ₃	13	28	$\overline{\text{S}}_2$		(M/ $\overline{\text{IO}}$)
AD ₂	14	27	$\overline{\text{S}}_1$		(DT/ $\overline{\text{R}}$)
AD ₁	15	26	$\overline{\text{S}}_0$		($\overline{\text{DEN}}$)
AD ₀	16	25	QS ₀		(ALE)
NMI	17	24	QS ₁		($\overline{\text{INTA}}$)
INTR	18	23	$\overline{\text{TEST}}$		
CLK	19	22	READY		
GND	20	21	RESET		

Mode Multiplexed pins: $\overline{\text{RQ}}/\overline{\text{GT}}_0$ and $\overline{\text{RQ}}/\overline{\text{GT}}_1$

- Request/Grant bi-directional pins.
- Other processors request the CPU(8086) through these lines to release the system bus.
- After receiving the request, CPU sends acknowledge signal through the same lines.
- $\overline{\text{RQ}}/\overline{\text{GT}}_0$ has higher priority than $\overline{\text{RQ}}/\overline{\text{GT}}_1$

		Maximum Mode		Minimum Mode	
GND	1	40	V_{CC}		
AD ₁₄	2	39	AD ₁₅		
AD ₁₃	3	38	A ₁₆ /S ₃		
AD ₁₂	4	37	A ₁₇ /S ₄		
AD ₁₁	5	36	A ₁₈ /S ₅		
AD ₁₀	6	35	A ₁₉ /S ₆		
AD ₉	7	34	$\overline{\text{BHE}}/\text{S}_7$		
AD ₈	8	33	MN/ $\overline{\text{MX}}$		
AD ₇	9	32	$\overline{\text{RD}}$		
AD ₆	10	31	$\overline{\text{RQ}}/\overline{\text{GT}}_0$	(HOLD)	
AD ₅	11	30	$\overline{\text{RQ}}/\overline{\text{GT}}_1$	(HLDA)	
AD ₄	12	29	LOCK	($\overline{\text{WR}}$)	
AD ₃	13	28	$\overline{\text{S}}_2$	(M/ $\overline{\text{IO}}$)	
AD ₂	14	27	$\overline{\text{S}}_1$	(DT/ $\overline{\text{R}}$)	
AD ₁	15	26	$\overline{\text{S}}_0$	($\overline{\text{DEN}}$)	
AD ₀	16	25	QS ₀	(ALE)	
NMI	17	24	QS ₁	($\overline{\text{INTA}}$)	
INTR	18	23	$\overline{\text{TEST}}$		
CLK	19	22	READY		
GND	20	21	RESET		

80286

Introduction:80286

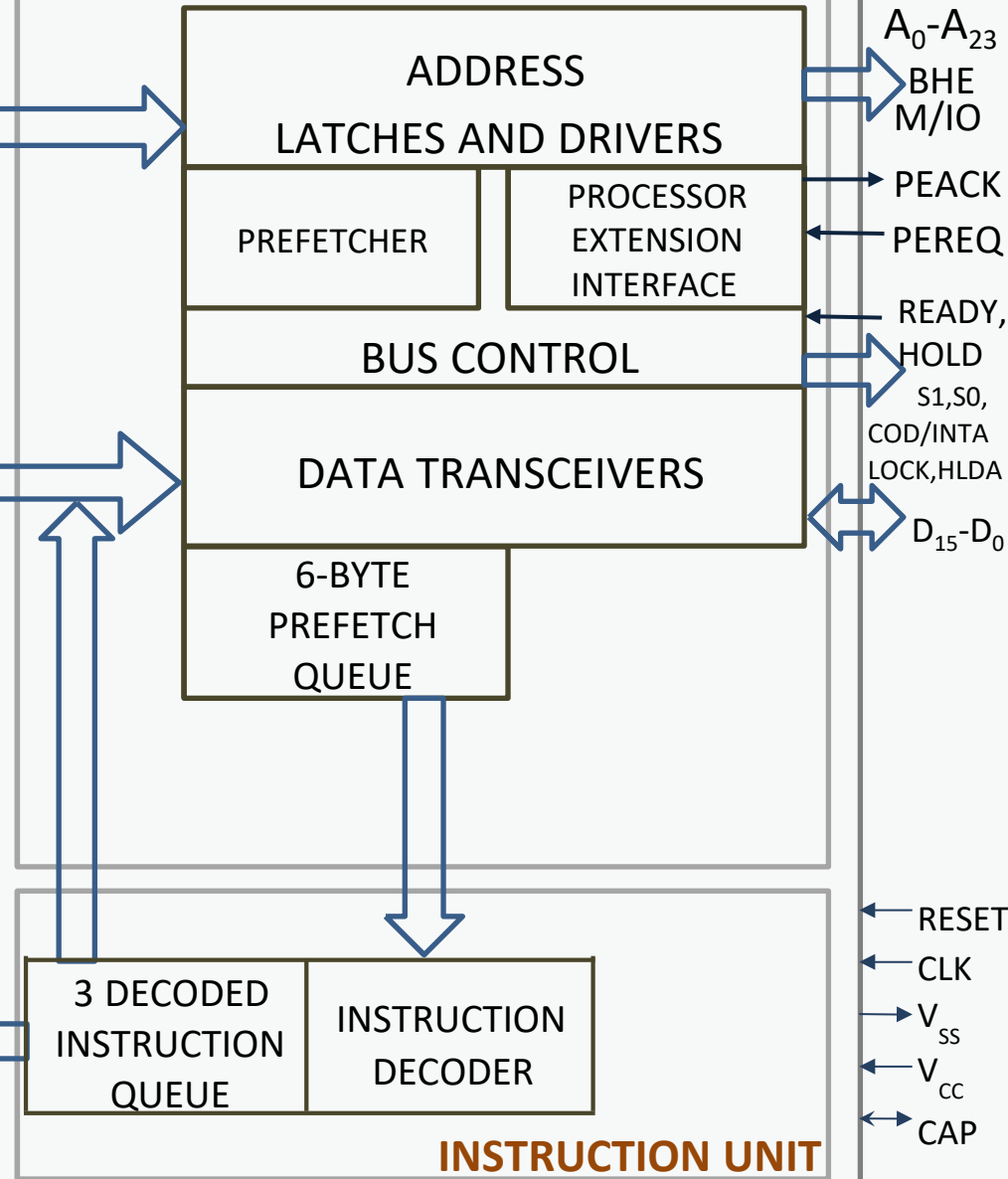
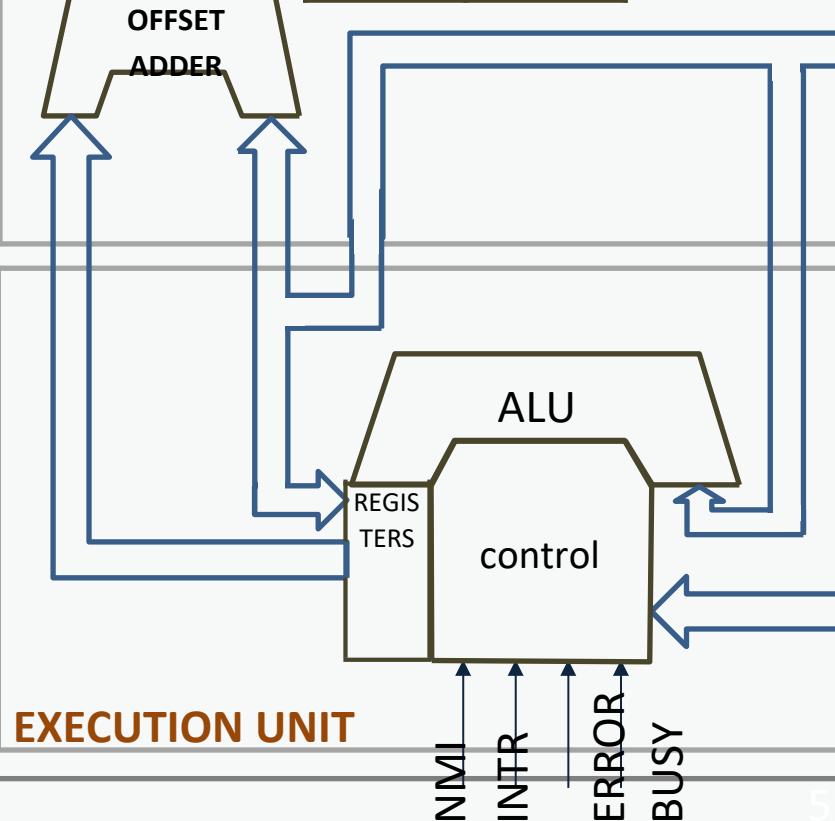
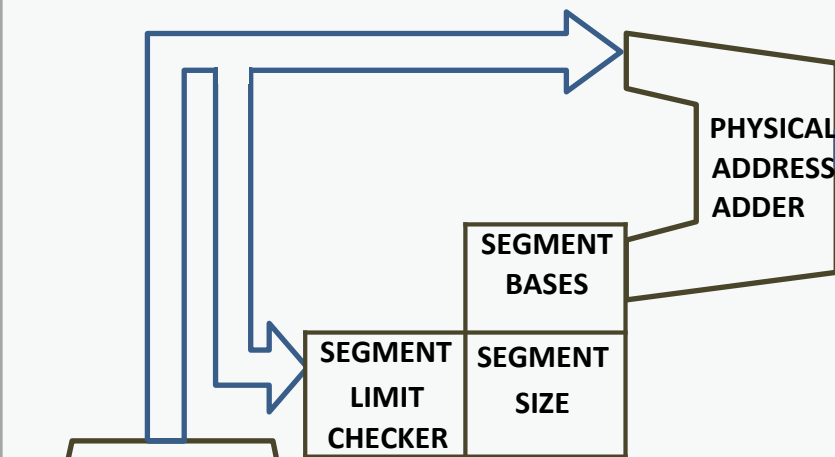
- The Intel 80286 had a 24-bit address bus and was able to address up to 16 MB of RAM compared to 1 MB of its predecessor (8086).
- It was designed for multi-user systems with applications, including multitasking communications and control. real-time process
- 80286 is the advanced with memory microprocessors management and protection abilities.

Introduction:80286

- 80286 have two operating modes namely real address mode and virtual address mode.
- In real address mode, it can address up to 1MB of physical memory address like 8086.
- In virtual address mode, it can address up to 16 MB of physical memory address space and 1 GB of virtual memory address space.
- The performance of 80286 is five times faster than 8086.

ADDRESS UNIT

BUS UNIT



80286 Architecture

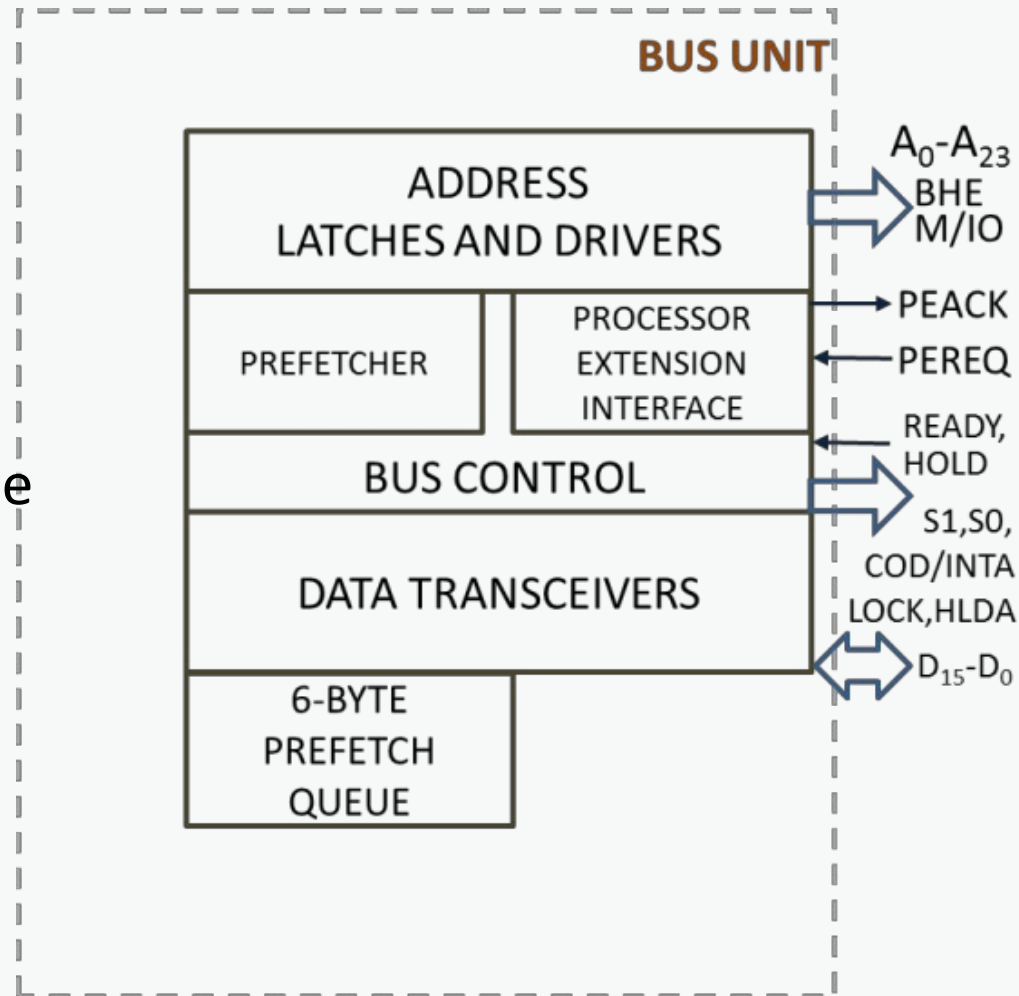
80286 Architecture contains 4 separate processing units

1. Bus Unit (BU)
2. Instruction Unit (IU)
3. Address Unit (AU)
4. Execution Unit (EU)

Bus Unit (BU)

Components:

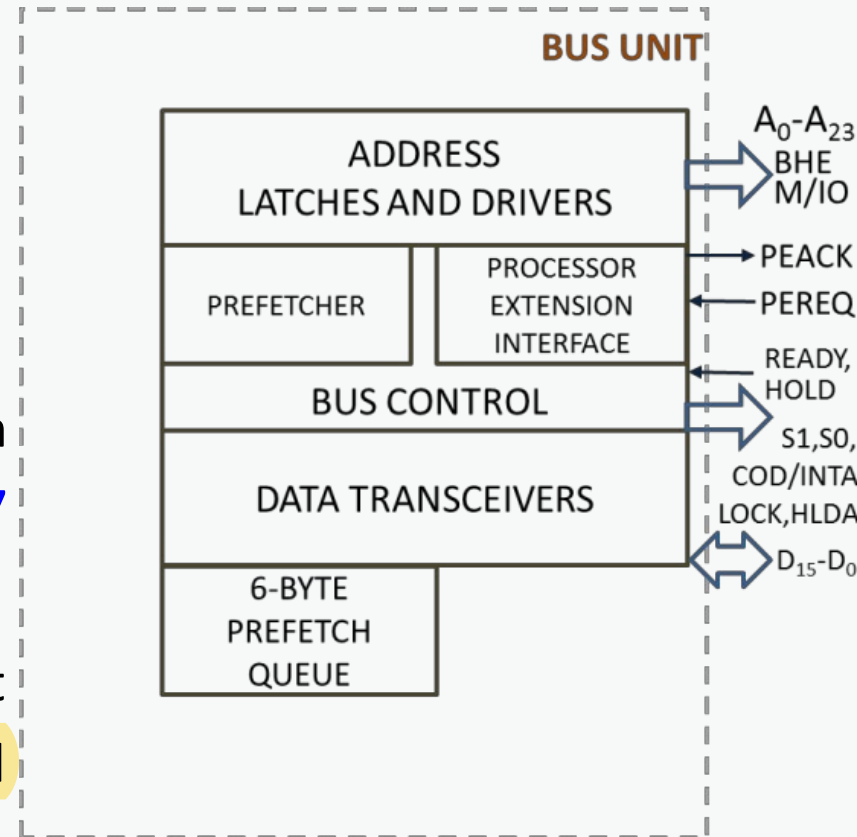
1. Address Latches & Drivers
2. Bus Control
3. Instruction Pre-fetcher
4. Processor Extension Interface
5. Data Transceivers
6. 6-byte Instruction Queue



Bus Unit (BU)

Functions:

- Perform all memory, I/O write operations. read and
- Pre-fetch the instruction bytes.
- To control transfer of data to and from processor extension devices like 80287 math co-processor.
- Whenever BU is not using the buses, it pre-fetches the instruction bytes and put them in 6 byte pre-fetch queue.
- Take care of communication between CPU and a Co-processor.



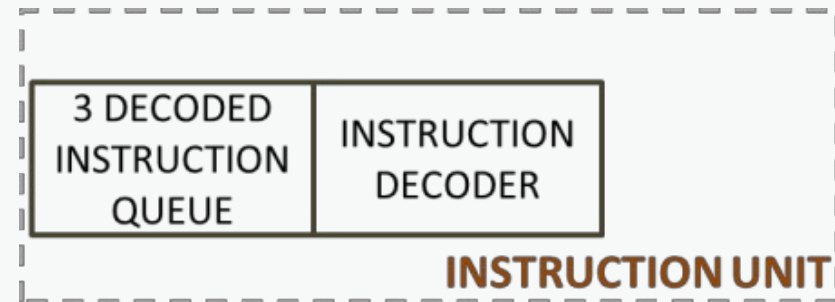
Instruction Unit (IU)

Components:

1. 3 **decoded** instruction queue
2. Instruction decoder

Functions :

- It fully decodes up to 3 pre-fetched instructions and holds them in a **queue**, so that Execution Unit (**EU**) can access them.
- It helps processor to speed up by **pipelining** the instructions.



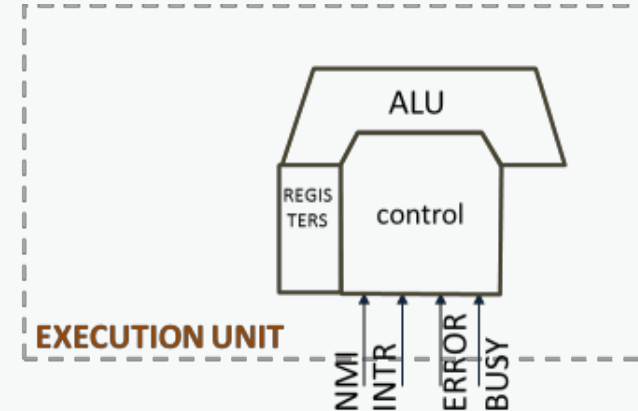
Execution Unit (EU)

Components:

- It includes **ALU**, **registers** and **control unit**.
- Registers are general purpose, index, pointer, flag registers and Machine Status Word (**MSW**).

Functions:

- To sequentially **execute** instructions received from the instruction unit.
- **ALU** result is either stored in registers or sent back over data bus.



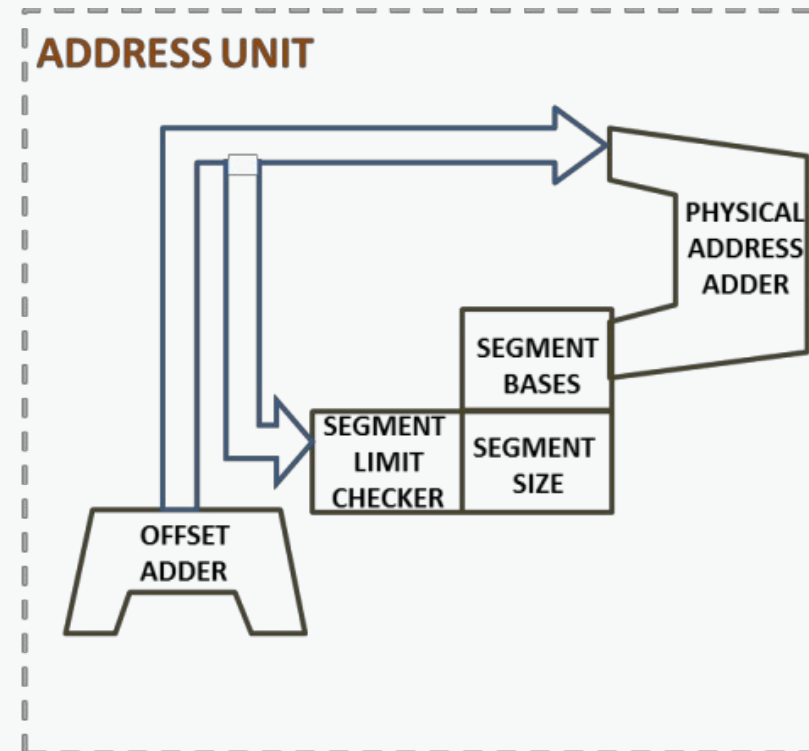
Address Unit

Components:

1. Segment Bases
2. Segment Limit Checker
3. Segment Size
4. Offset address
5. Physical address adder

Functions:

- It computes **physical address** that will be sent out to the **memory** or **I/O** by Bus Unit (BU).
- 80286 operate in two different modes
 1. **Real** address mode
 2. **Protected** Virtual Address Mode.



Register Organization of 80286

80286 Register Organization

1. Eight 16-bit general purpose registers (AX, BX, CX, DX, SP, BP, SI, DI).
2. Four 16-bit segment registers (CS, SS, DS, ES).
3. 16-bit Instruction Pointer (IP).
4. 16-bit Flag Register.

Additionally,

5. One new 16-bit Machine Status Word (MSW) register.

Flag Register:

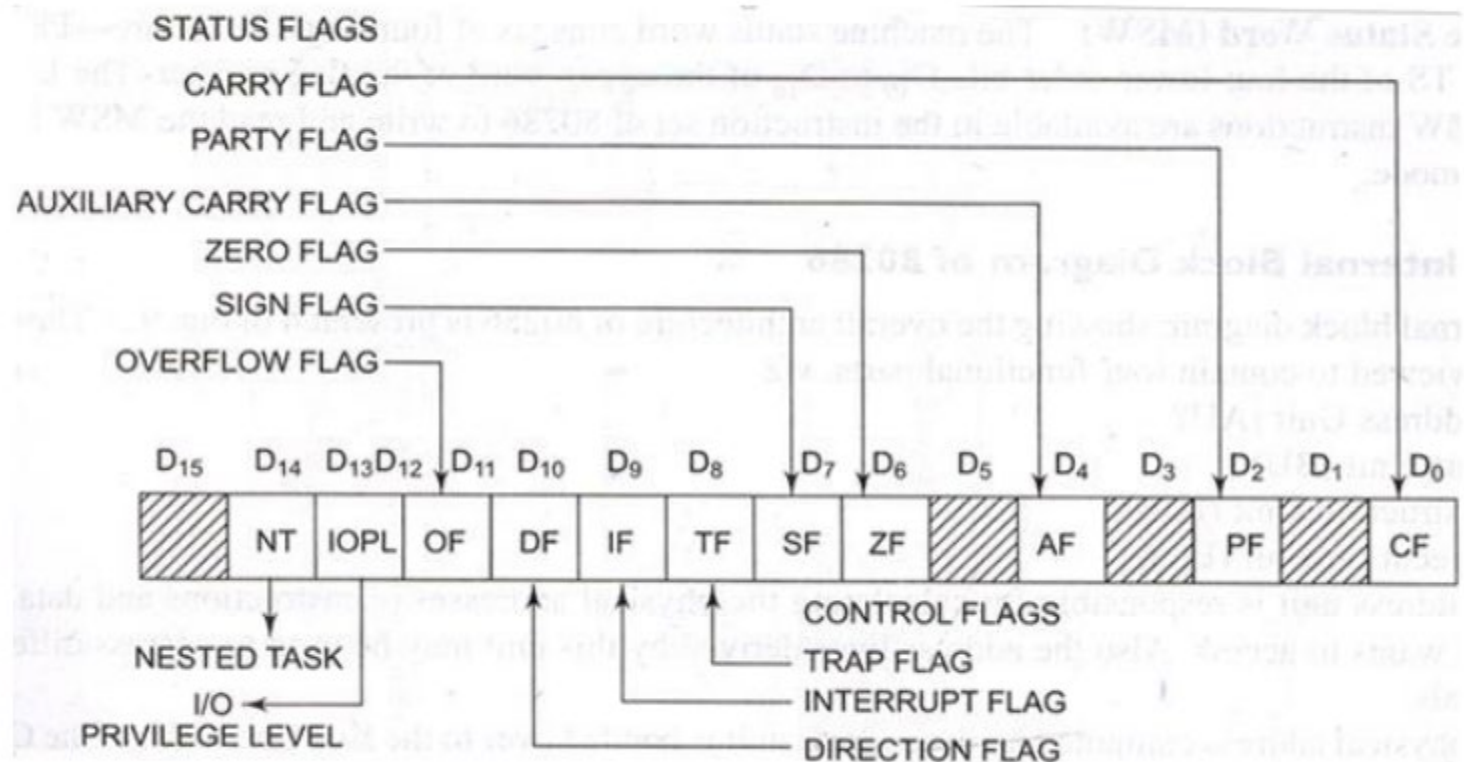


Figure 4.2: 80286 Flag Register

MSW(Machine Status Word):

- There are four Processor Status bit in 80286:-



Figure 4.3: Machine Status Word

1) PE - Protection Enable

- The PE bit is set to enable the Protected Mode. If PE is reset, the processor operates again in Real Mode.

2) MP - Monitor Processor Extension

- The MP extension allows WAIT instructions to cause a processor extension not present exception (Number 7).

3) EM - Processor Extension Emulator

- The EMulate Processor Extension cause a Processor extension not present exception (Number 7) on ESC instruction to allow emulating a processor extension.

4) TS – Task Switch

- TS is automatically set whenever a task switch operation is performed.
- Task switched indicates the next instruction using processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.

80286: Real Address Mode

Real Mode

- **Address Unit** computes the address with **segment** base and **offset** like 8086.
- Maximum physical space allowed in this mode is **1MB**.
- When 80286 get **RESET**, it always starts execution in **real mode**.

Task of 80286 in Real Mode

- **Initializes** Instruction Pointer(**IP**) and other registers of 80286.
- Initializes the **peripheral**.
- Enable **Interrupts**.
- Set up **descriptor table**.
- Prepares for entering in **PVAM**(Protected Virtual Address Mode).

80286: Protected Virtual Address Mode

- 286 is the 1st processor to support the concept of **Virtual Memory** and **Memory Management**.
- Here, the address unit acts as Memory Management Unit (**MMU**).
- All **24** address lines are used and can access up to **16MB** of physical memory.
- If descriptor table scheme is used it can address up to **1GB** of virtual memory.

What is the role of MMU?

- Memory Management Unit (MMU) translates the virtual memory address into the physical memory address.
- Virtual memory can be many times larger than physical memory.
- Only programs that are currently required brought from the secondary storage such as a hard disk to the physical memory (RAM) for execution.
- This is desirable as a microprocessor is supposed to store large programs and data can't be accommodated in the physical memory space.

What is the role of MMU?

- The **hard disk** is in the virtual or logical address space but not in the physical address space.
- Faster memory such as **RAM** is used as the physical memory.
- Before microprocessor **executes** a program, it checks whether the program is **available** in physical memory (**RAM**) or not.
- If program is not available in the physical memory, it is brought from the **secondary memory** to the **physical memory**.
- If available space is **inadequate** in the physical memory, some less important/unused program can be **swapped back** to the secondary memory to create space.

80286: Protected Virtual Address Mode

- The complete virtual memory is mapped with the 16MB of physical memory.
- If a program is larger than 16MB, it is stored in the hard disk and will be executed by swapping as per sequence of execution.
- The huge programs are divided in smaller segments or pages arranged in appropriate sequence.

80286 Modes

Real Address Mode	Protected Virtual Address Mode
Can only address up to 1MB of System Memory and act as 8086 Virtual Memory.	Can address up to 16MB of System Memory. Supports the concept of Virtual Memory.
Real mode provides no support for memory protection, multitasking, or code privilege levels.	Protected mode provides support for memory protection, multitasking, or code privilege levels.
Initially every processor is in Real Mode i.e MSW PE-bit to 0	Microprocessor will switch to PVAM mode by setting MSW PE-bit to 1

PVAM: Descriptor

- This smaller segments or pages have been associated with data structure called a *Descriptor*.
- It contains *information* of program segment or pages.
- The data structure *Descriptor* is essentially one such identifier of particular program or segment.
- The set of such descriptor arranged in a proper sequence describes the *complete program*.

Real Address Mode

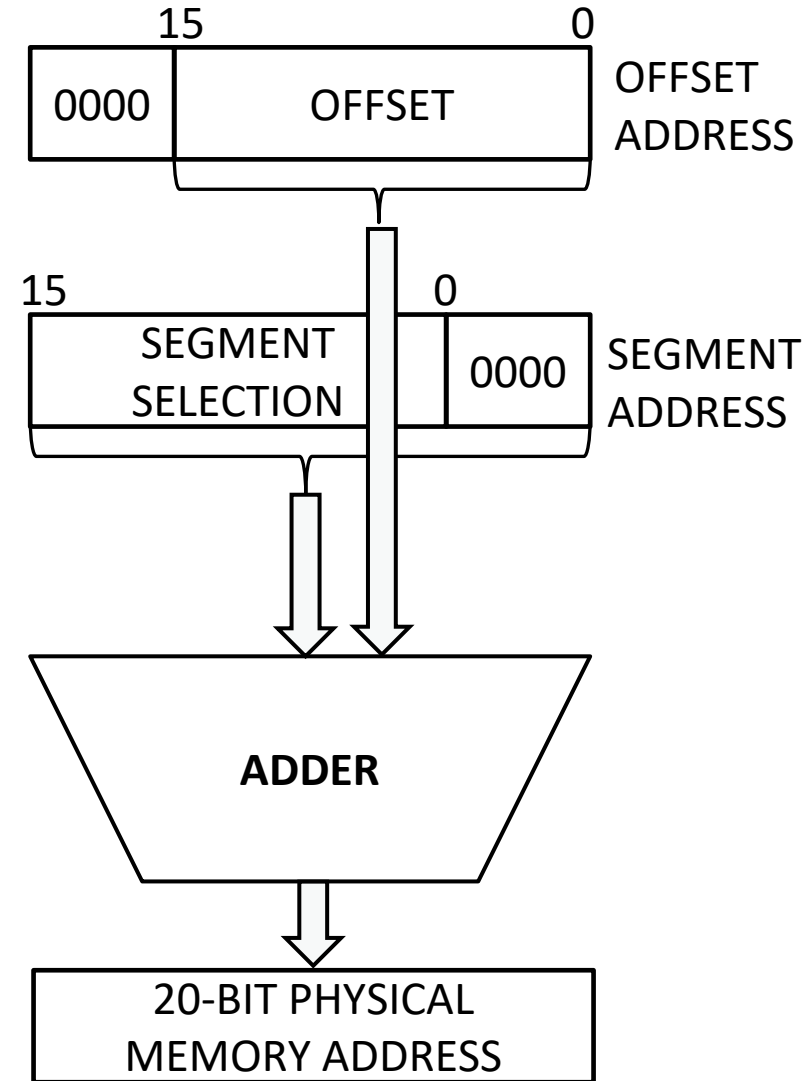
Address Calculation

Total 1MB of Memory, divided among 16-segments with each of size 64kb.

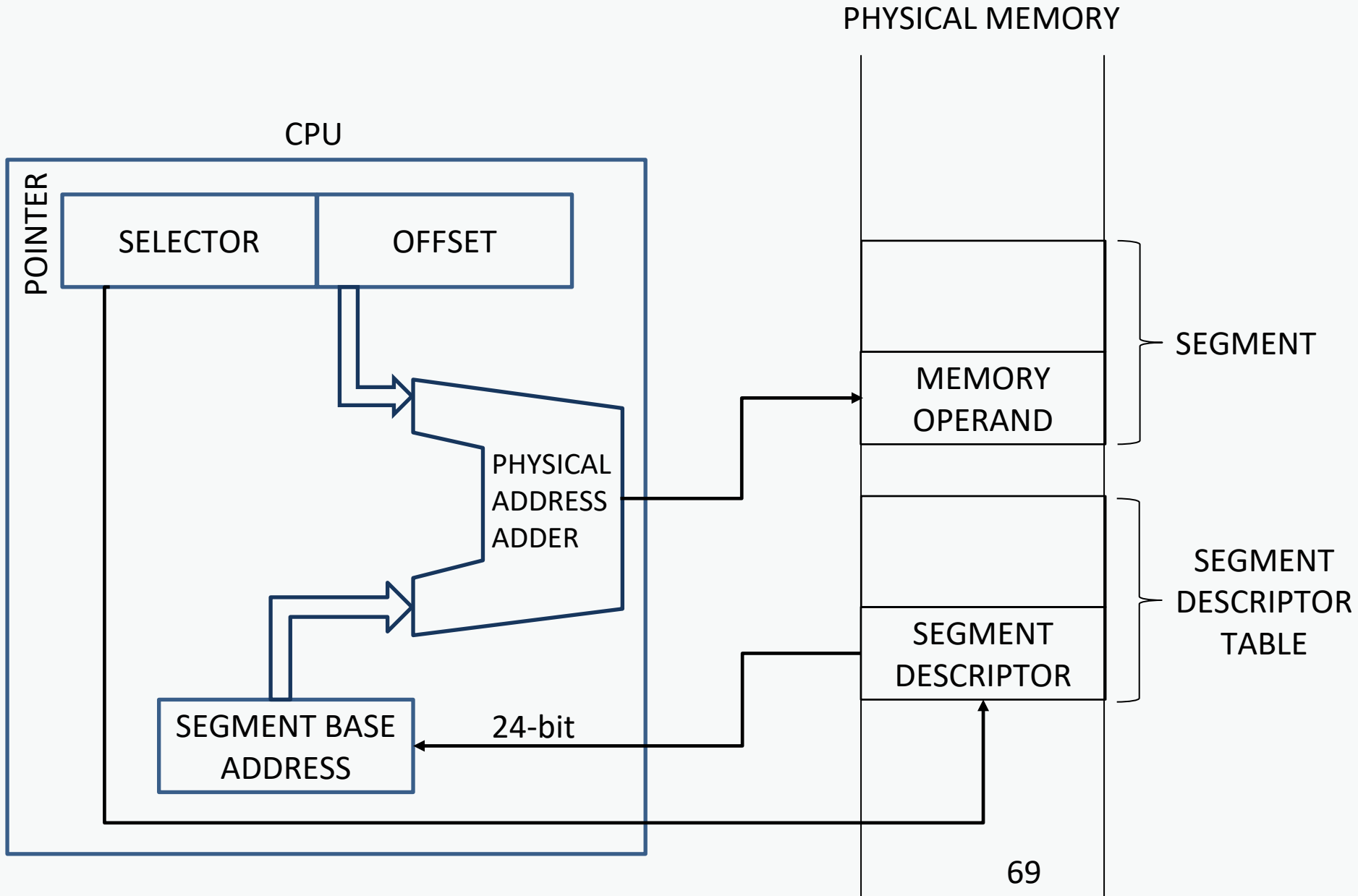
80286 reserves two fixed areas for

- i. System Initialization
- ii. IVT (Interrupt Vector Table)

IVT-1KB of Starting address	00000H – 003FFH
System Initialization	FFFF0H – FFFFFH



Physical Address calculation in PVAM



PVAM: Descriptor Table

- The set of descriptor is known as *Descriptor Table*.
- All the sets of **descriptor/Descriptor Table** are prepared and managed by **Operating System**.
- Descriptor carry all the relevant information regarding a **segment** and their **access rights**.

Types of Descriptor Table

Data Segment Descriptor

Used for Data Segment

Code Segment Descriptor

Used for Code Segment

System Segment

Used for System Programs

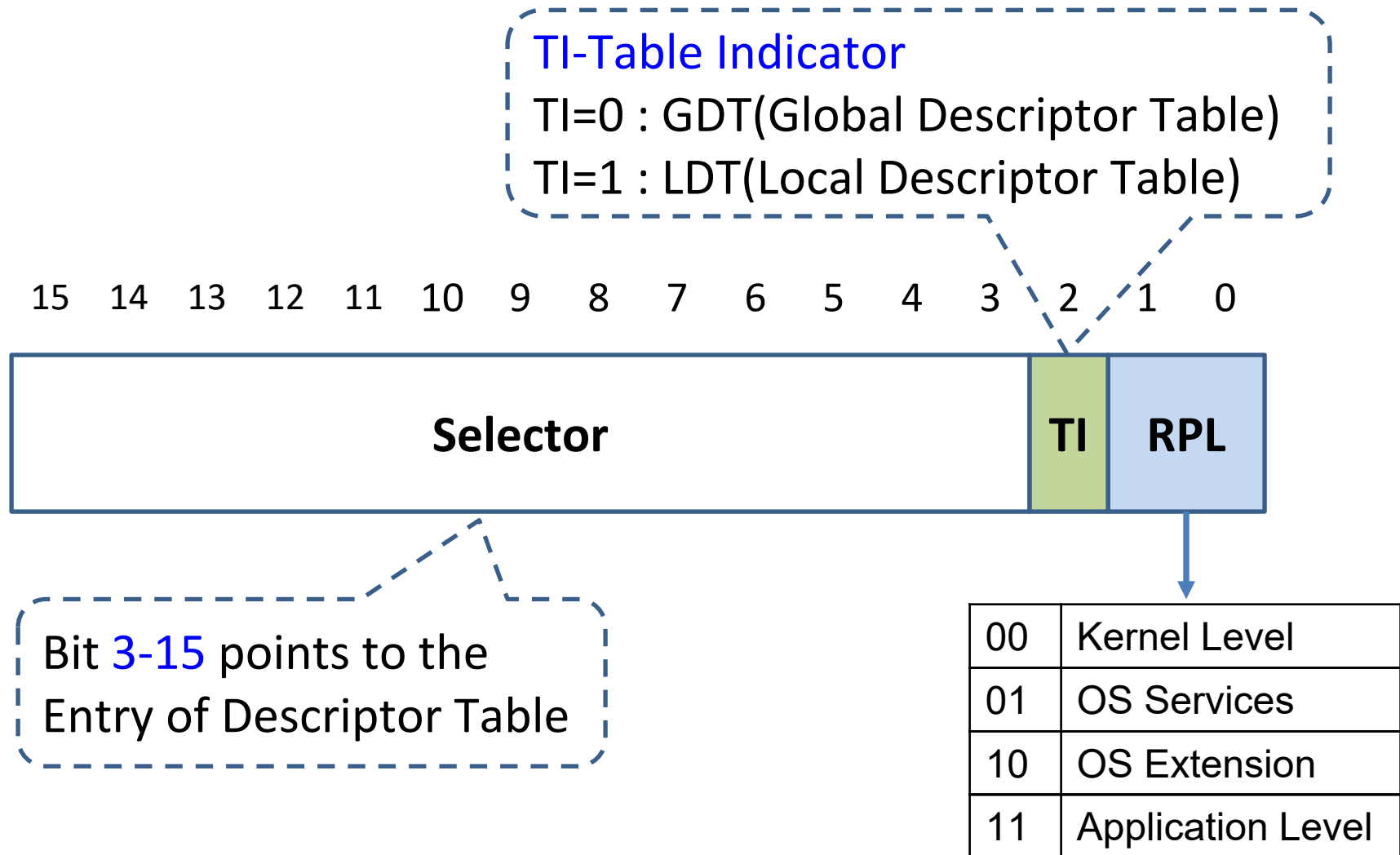
Descriptor **Gate** Segment

Used for Subroutine and
ISR

Descriptor

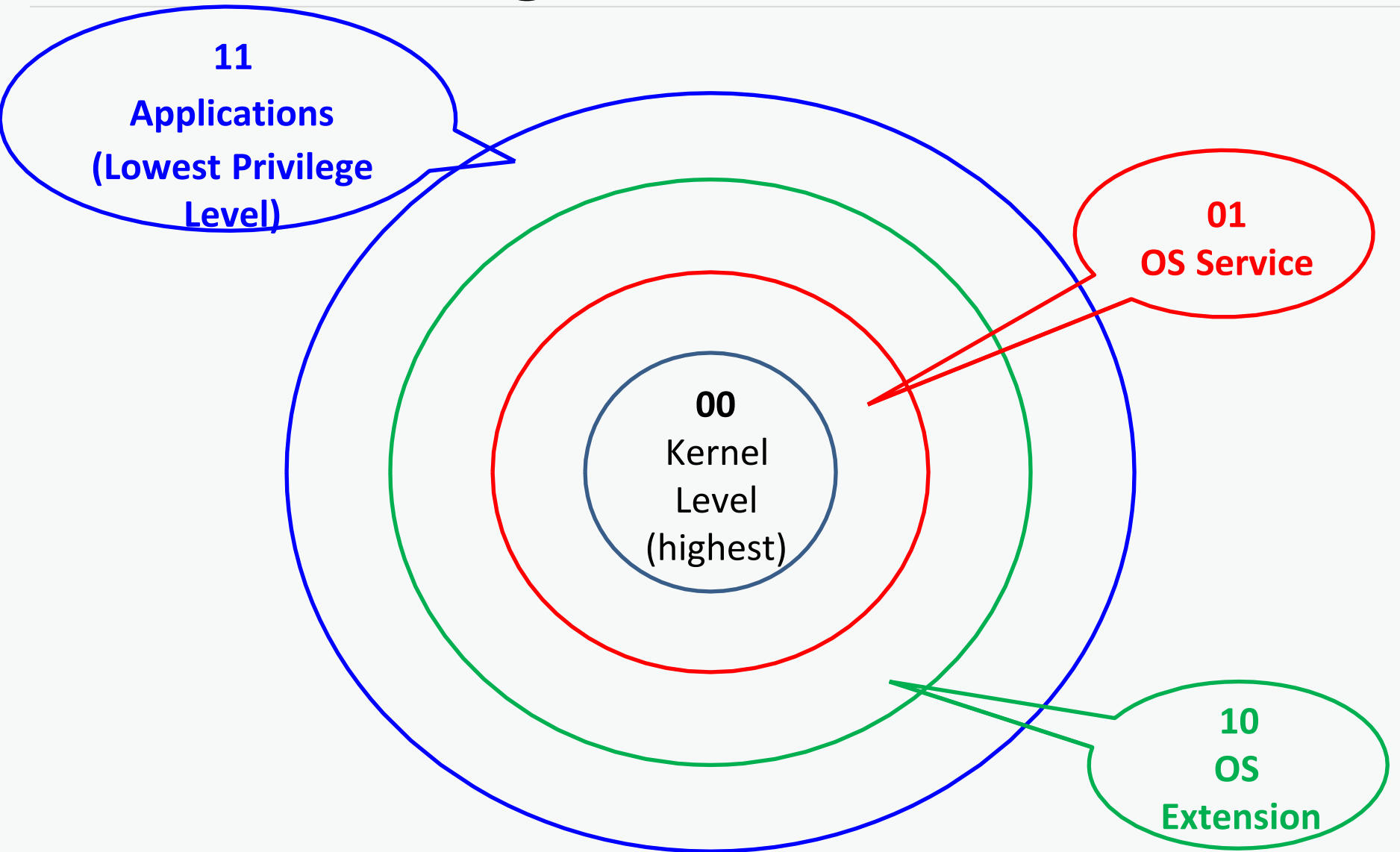
Memory Access in GDT and LDT

PVAM : Selector



80286 Privilege Level

80286 Privilege Level



Descriptor Table

GDT and
LDT

Descriptor Table: GDT,LDT

- A **segment** cannot be accessed, if its descriptor does not exist in either **LDT** (Local Descriptor Table) or **GDT** (Global Descriptor Table).
- Set of **descriptor** arranged in a proper sequence describes the complete program.
- Each Descriptor is **8-byte** long.

Descriptor Table

The descriptor is a block of contiguous memory location containing information of a segment, like

- i. Segment **base address**
- ii. Segment **limit**
- iii. Segment **type**
- iv. **Privilege** level – prevents unauthorized access
- v. Segment **availability** in physical memory
- vi. Descriptor **type**
- vii. Segment use by another **task**

Descriptor Table

The **GDT** contains information about segments that are global in nature, that is, available to all programs and normally used most heavily by the operating system.

The **LDT** contains descriptors that are application specific.

A global descriptor is also known as **System Descriptor**, and local descriptor is known as **Application Descriptor**.

The global descriptor table's base address is stored in **GDTR**.

The local descriptor table's base address is stored in **LDTR**.

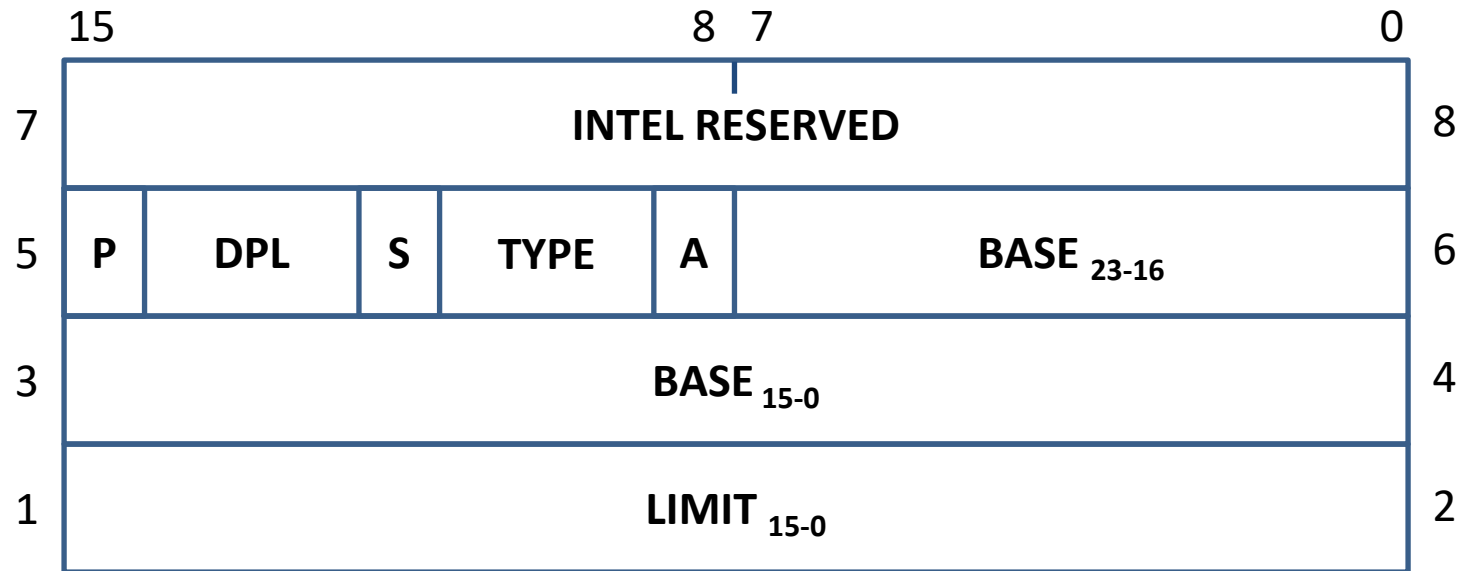
Differentiate GDT and LDT

LDT (Local Descriptor Table), acts similar to **GDT**, which also saves segments descriptor.

The main differences between **GDT** and **LDT** is:

1. **GDT** have **only one copy** in system while **LDT** can have many.
2. **GDT** may not changed during execution, while **LDT** often changes when **task switches**.
3. Entry of **LDT** is saved in **GDT**.
4. Entries in **GDT** and **LDT** have the **same structure**.

Data or Code Segment Descriptor



Data or Code Segment Descriptor

7	6	5	4	3	2	1	0
P	DPL		S	E	ED/C	R/W	A

TYPE

P=0: Descriptor is **undefined**, no mapping to physical memory exists.

P=1: Segment is mapped into physical memory.

Data or Code Segment Descriptor

7	6	5	4	3	2	1	0
P	DPL		S	E	ED/C	R/W	A

Sets the Descriptor Privilege Level (DPL) necessary for protection

S=0: System descriptor(GDT)

S=1: Application Descriptor(LDT)

Data or Code Segment Descriptor

				TYPE			
7	6	5	4	3	2	1	0
P	DPL		S	E	ED/C	R/W	A

Executable
E=0: Data Segment
E=1: Code Segment

Data or Code Segment Descriptor

				TYPE			
7	6	5	4	3	2	1	0
P	DPL		S	E	ED/C	R/W	A

Expansion Direction: Data Segment {when E=0}

ED=0 Segment expands upward (Data segment)

ED=1 Segment expands downward (Stack Segment)

Conforming: Code Segment {when E=1}

C=0 Ignore DPL

C=1 Code segment will only be executed when
 $CPL > DPL$

Data /Code Segment Descriptor

7	6	5	4	3	2	1	0
P	DPL		S	E	ED/C	R/W	A

Read : Code Segment

R=0 Code Segment execute only,
not readable

R=1 Code Segment both
executable & readable

Write : Data Segment

W=0 Data segment not writable

W=1 Data segment writable

Data /Code Segment Descriptor

7	6	5	4	3	2	1	0
P	DPL		S	E	ED/C	R/W	A

Accessed

A=0 Segment not accessed

A=1 Segment has been accessed

80386

Introduction : 80386

Enhanced version of 80286 microprocessor.

32-bit processor

i.e 32-bit data bus, 32-bit registers, 32-bit address bus.

Addressable physical memory is $2^{32} = 4\text{GB}$.

80386 CPU supports 16KB of segments, thus total virtual memory space = $4\text{GB} \times 16\text{KB} = 64\text{TB}$.

The 80386 memory manager is similar to the 80286, except the physical addresses generated by the MMU are 32-bit wide instead of 24-bit.

The concept of paging was introduced in 80386.

Introduction : 80386

386 is capable of performing more than 5 Million Instructions Per Second (MIPS).

80386 support 3 operating modes:

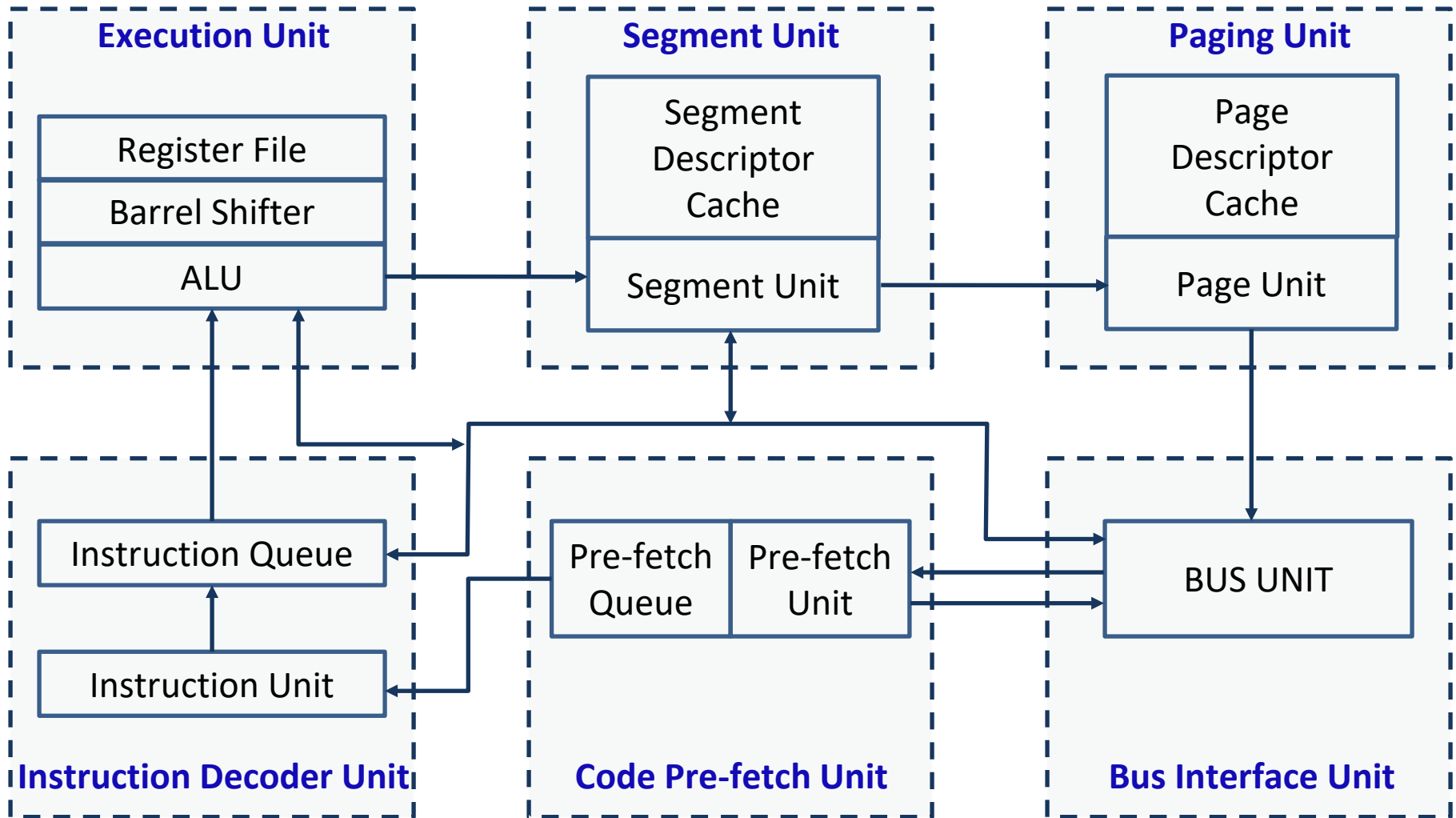
1. Real Mode (default)
2. Protected Virtual Address Mode (PVAM)
3. Virtual Mode

The memory management section of 80386 supports virtual memory, paging and four levels of protection.

The 80386 includes special hardware for task switching.

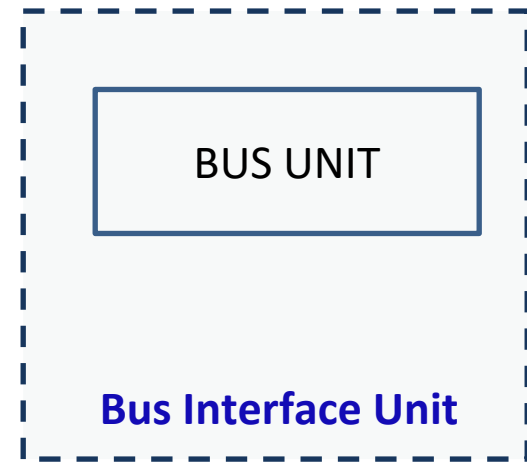
80386 Architecture

80386 Architecture



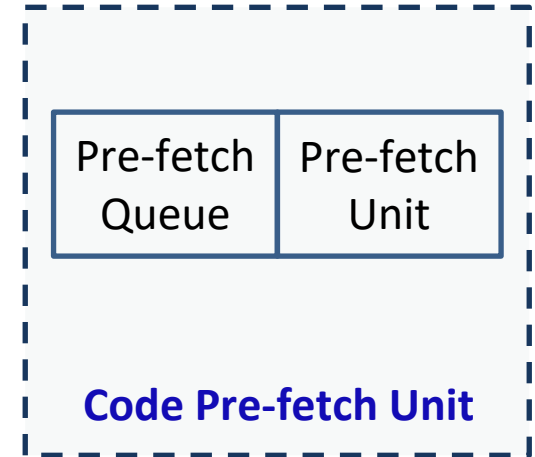
Bus Interface Unit

The **Bus Interface Unit** connects 80386 with **memory** and **I/O**. Based on internal requests for fetching instructions and transferring data from the code pre-fetch unit, 80386 generates the **address, data and control signals** for the current **bus cycles**.



Code Pre-fetch Unit

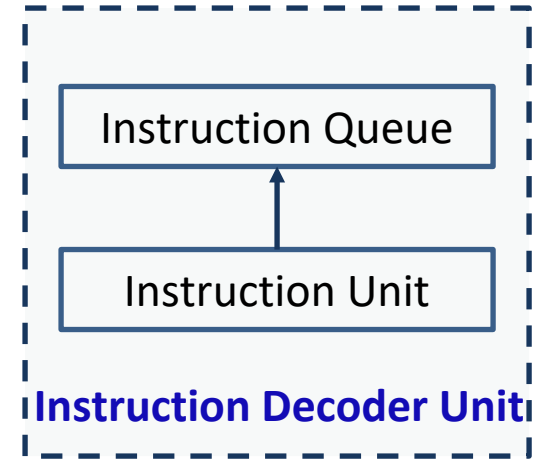
The **code pre-fetch unit** pre-fetches instructions when the bus interface unit is not executing the bus cycles. It then stores them in a **16-byte** instruction queue for decoding by the instruction decode unit.



Instruction Decoder Unit

The **Instruction Decoder Unit** translates instructions from the pre-fetch queue into micro-codes.

The decoded instructions are then stored in an instruction queue (**FIFO**) for **processing** by the execution unit.



Execution unit

Execution unit has 8 General purpose registers which are either used for handling data or calculating offset addresses.

The **execution unit** processes the instructions from the **instruction queue**.

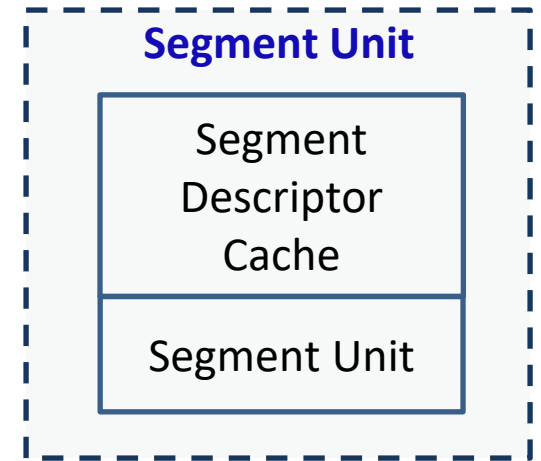
It contains a **control unit**, a **data unit** and a **protection test unit**.

The **barrel shifter** increases the speed of all shift and rotate operations.



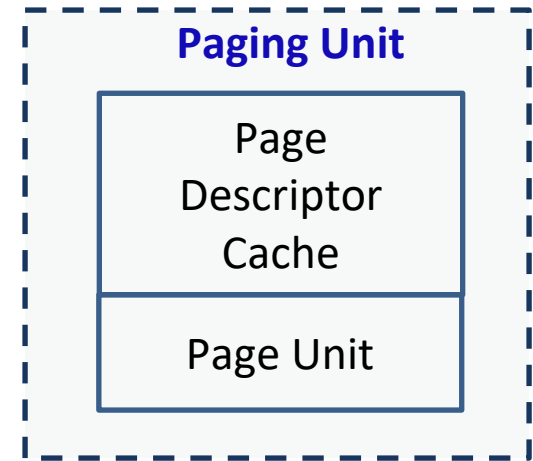
Segment Unit

The **segment unit** calculates and translates the logical address into linear addresses at the request of the execution unit.

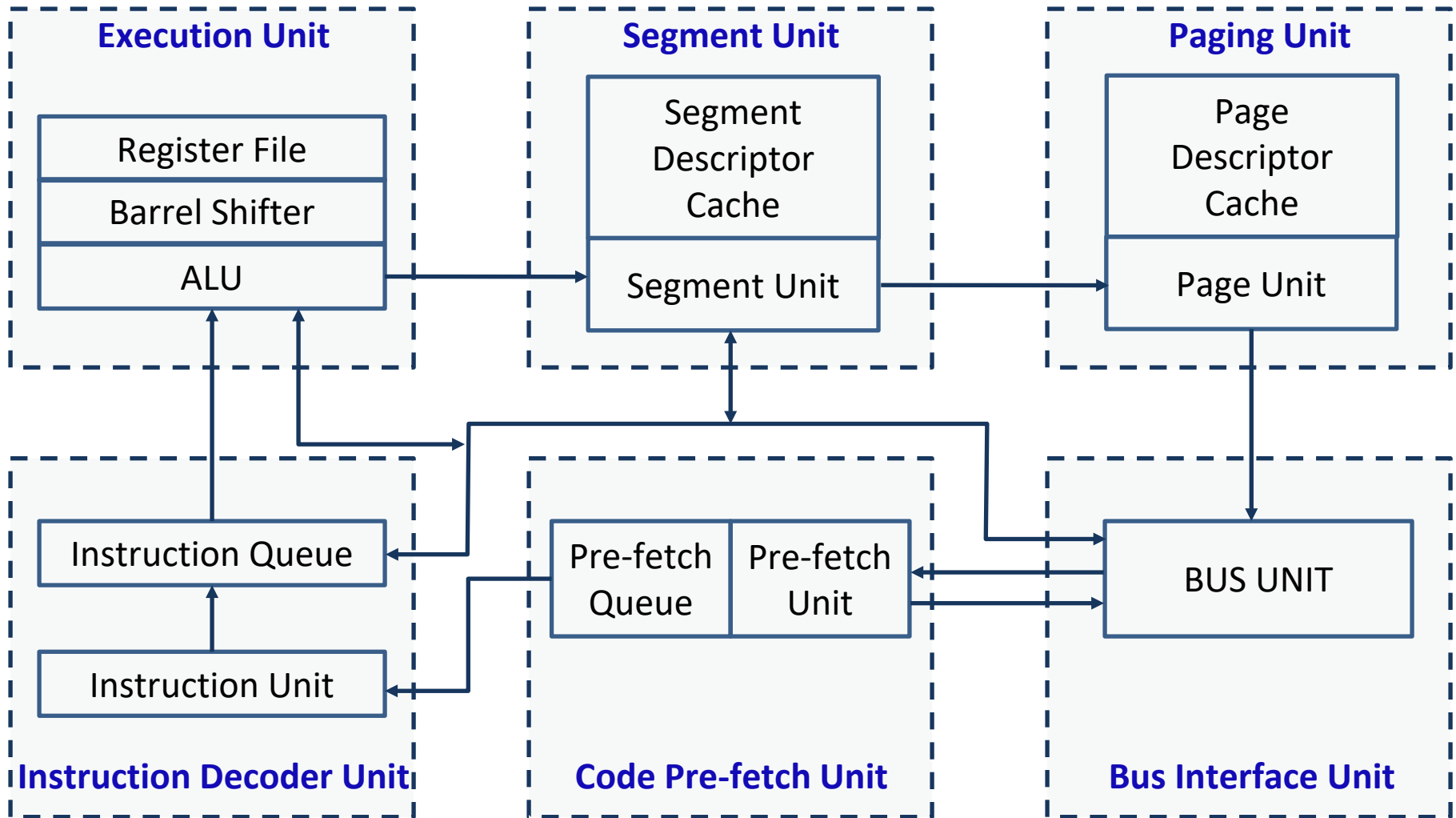


Paging Unit

The translated **linear address** is sent to the **Paging Unit**.
Upon enabling the paging mechanism, the 80386 translates these **linear addresses** into **physical addresses**.
If paging is not enabled, the physical address is identical to the linear address and no translation is necessary.

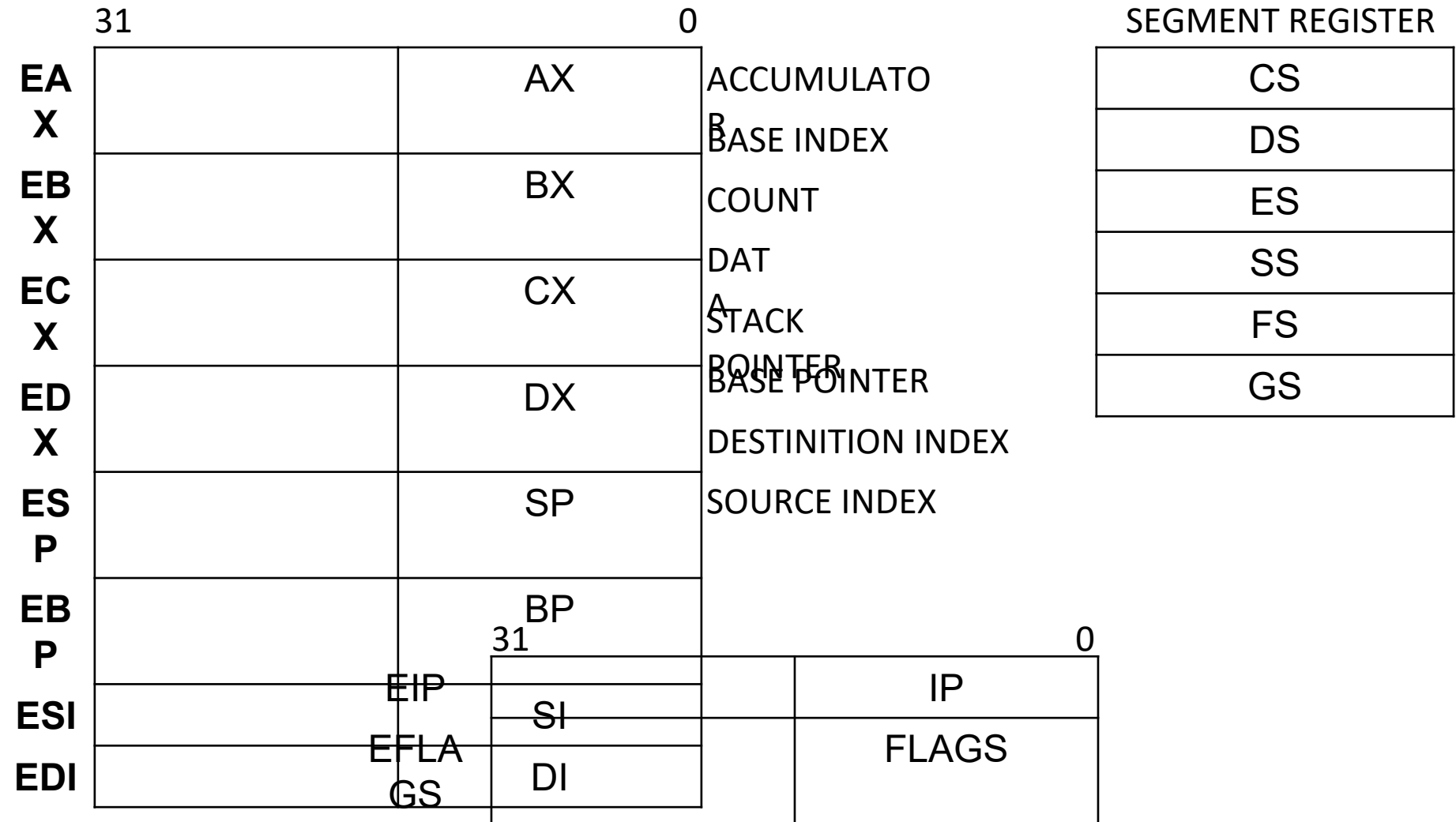


80386 Architecture



80386 Register Organization

80386 Register Organization



80386 Register Organization

Flag Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	NT	IOPL		OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
														VM	RF

Virtual Mode Enable

When VM=1; 386 will switch from Protected to virtual mode

80386 Register Organization

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	NT	IOPL		OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
														VM	RF

Resume Flag

- It is used in conjunction with the debug register breakpoints.
- When RF is set(1), it causes any debug fault to be ignored on the next instruction.
- RF is then automatically reset at the successful completion of every instruction.

Operating Modes of 80386

Operating Modes of 80386

1. Real Address Mode
2. Protected Virtual Address Mode
3. Virtual Mode

80386: Real Address Mode

When 80386 resets, the initial operating mode is [Real Address Mode](#).

In real address mode, 80386 works as a [fast 8086](#) with 32-bit registers and data types.

Real-address mode is in effect after a signal on the [RESET pin](#).

Even if the system is going to be used in protected mode, the start-up program will execute in real mode temporarily while initializing for protected mode.

The [addressing techniques](#), [memory size](#), [interrupt handling](#) in this mode of 80386 are similar to the [real addressing mode](#) of 80286.

In real address mode, the default operand size is 16-bit but 32-bit operands and addressing modes may be used with the help of [override prefixed instructions](#).

Maximum physical memory = [1MB](#).

80386 :PVAM

MMU operates similar to 80286.

Virtual addresses are represented with a selector component and an offset component.

The selector component is used to index a descriptor in a descriptor table.

The descriptor contains the 32-bit physical base address for the segment.

The offset part of the virtual address is added to the base address to produce the actual physical address.

The offset part of a virtual address can be 16-bit or 32-bit, so segment can be as large as 4GB.

Hence the virtual memory size is 64TB.

Operating Modes of 80386 :PVAM

Advantage of segmentation of memory: Segments corresponds to code and data structures in the program. Hence segmentation is useful.

Limitation of segmentation of memory: If we need only a part of memory, even then we have to swap the whole segment content. This will **increase** the time for execution.

80386:Virtual Mode

Virtual Mode (Paged Mode)

In virtual mode, 8086 can address 1MB of physical memory that may be anywhere in the 4Gbytes address space of the protected mode of 80386.

In this mode, instead of segments, 4KB of fixed page length are used.

80386:Virtual Mode

Total memory available for paging = **1MB = 1024KB**

Size of one page = **4KB**

How many pages can be addressable with 1MB of memory?

Total pages= $1024\text{KB}/4\text{KB} = 256 \text{ pages}(4\text{KB each})$

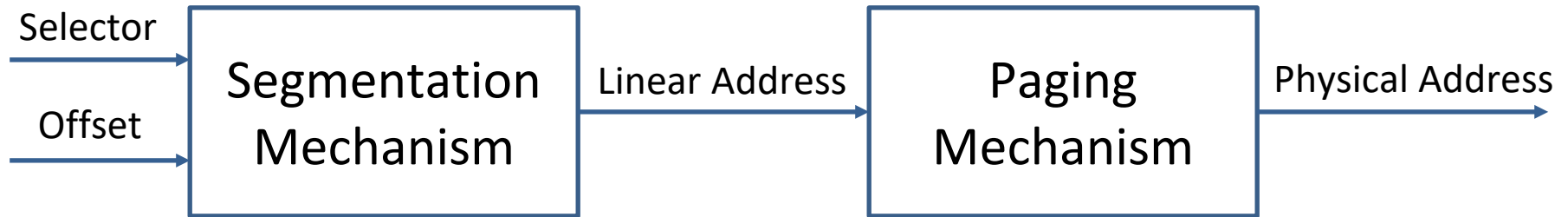
Paged Mode

Limitation : Pages do not correspond to the logical structure of the program.

Advantage : Pages can be quickly swapped.

80386 Memory Access in Virtual Mode

Memory Access in Virtual Mode



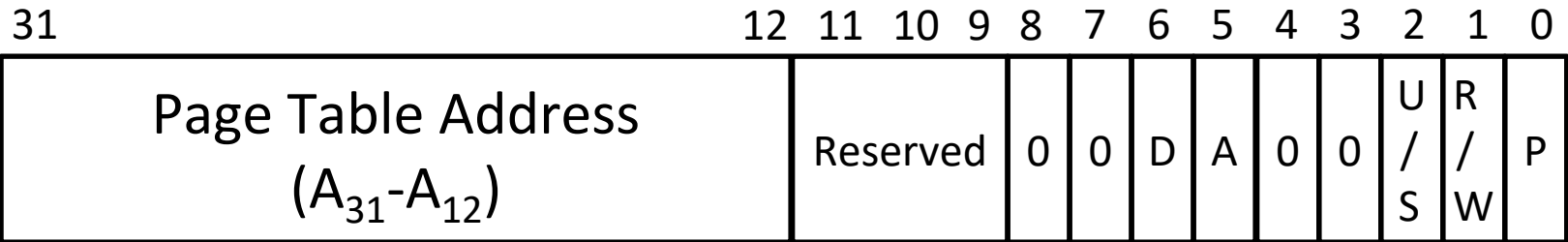
Operating Modes of 80386:VM

Page Directory Entry

Each directory entry is of 4 byte

D: Dirty bit

Dirty bit is set before any write operation to the
Dirty bit are undefined for page directory entries



P: Present bit

- **P=0**, indicates entry **cannot** be used for address translation.
- **P=1**, indicates entry **can** be used for address translation.
P-bit of currently executed page is always **high**.

the upper **20-bit** ($A_{31}-A_{21}$) are used
to select **1024** page table entries.
Page Tables can be **shared** between
the task.

112

U/S	R/W	Permitted for Level 3	Permitted for Level 2,1 or 0
0	0	None	Read/Write
0	1	None	Read/Write

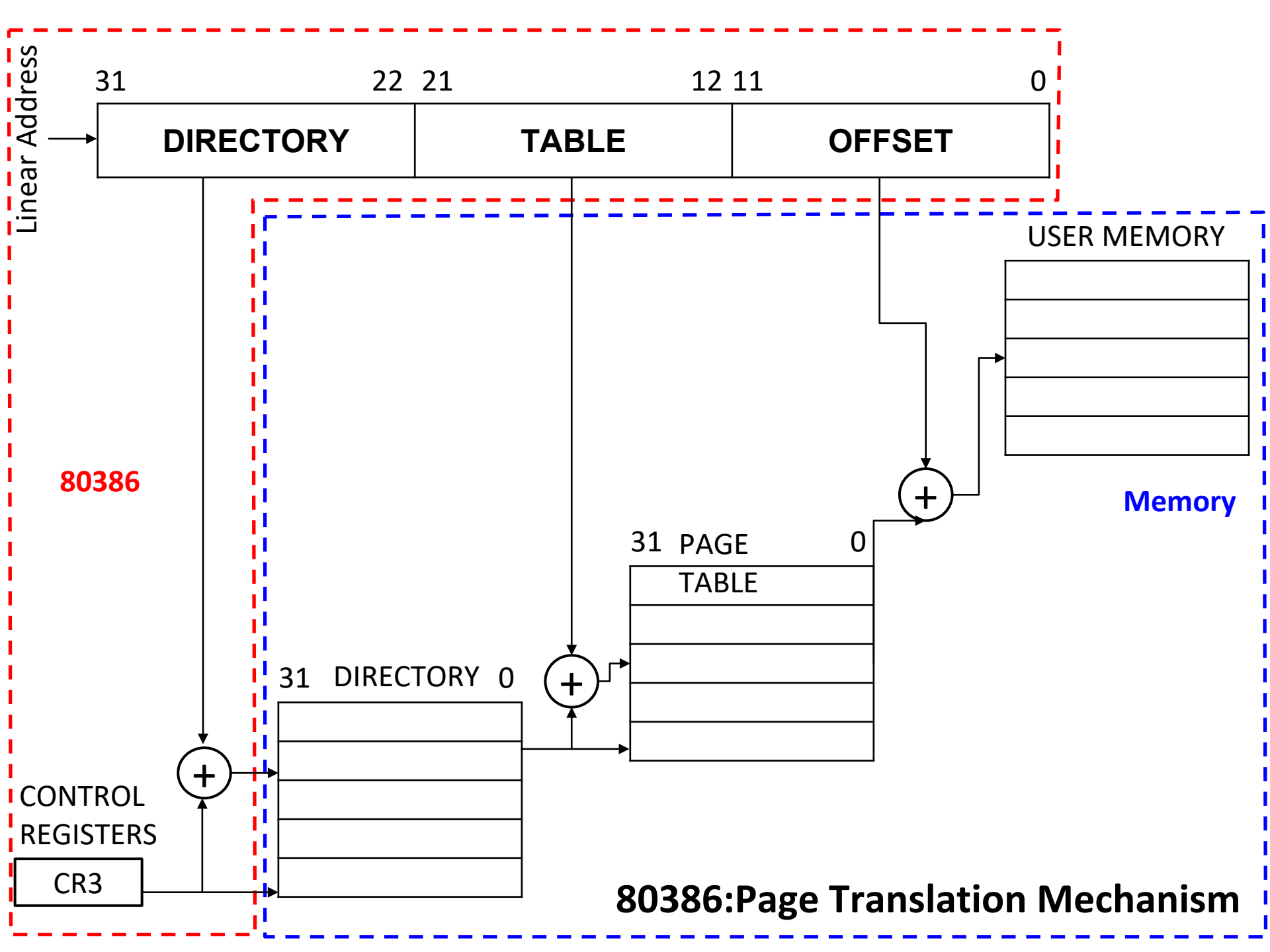
Page Translation Mechanism in 80386

Page Translation Mechanism

Format of Linear Address

A **linear address** refers indirectly to a **physical address** by specifying a **page table**, a **page** within that table, and an **offset** for that page





80386:Page Translation Mechanism

CR3: Enables processor to translate **linear addresses** into **physical addresses** by locating the **page directory** and **page tables** for the current task.

Here processor converts the **DIRECTORY**, **TABLE**, and **OFFSET** fields of a linear address into the physical address by consulting two levels.

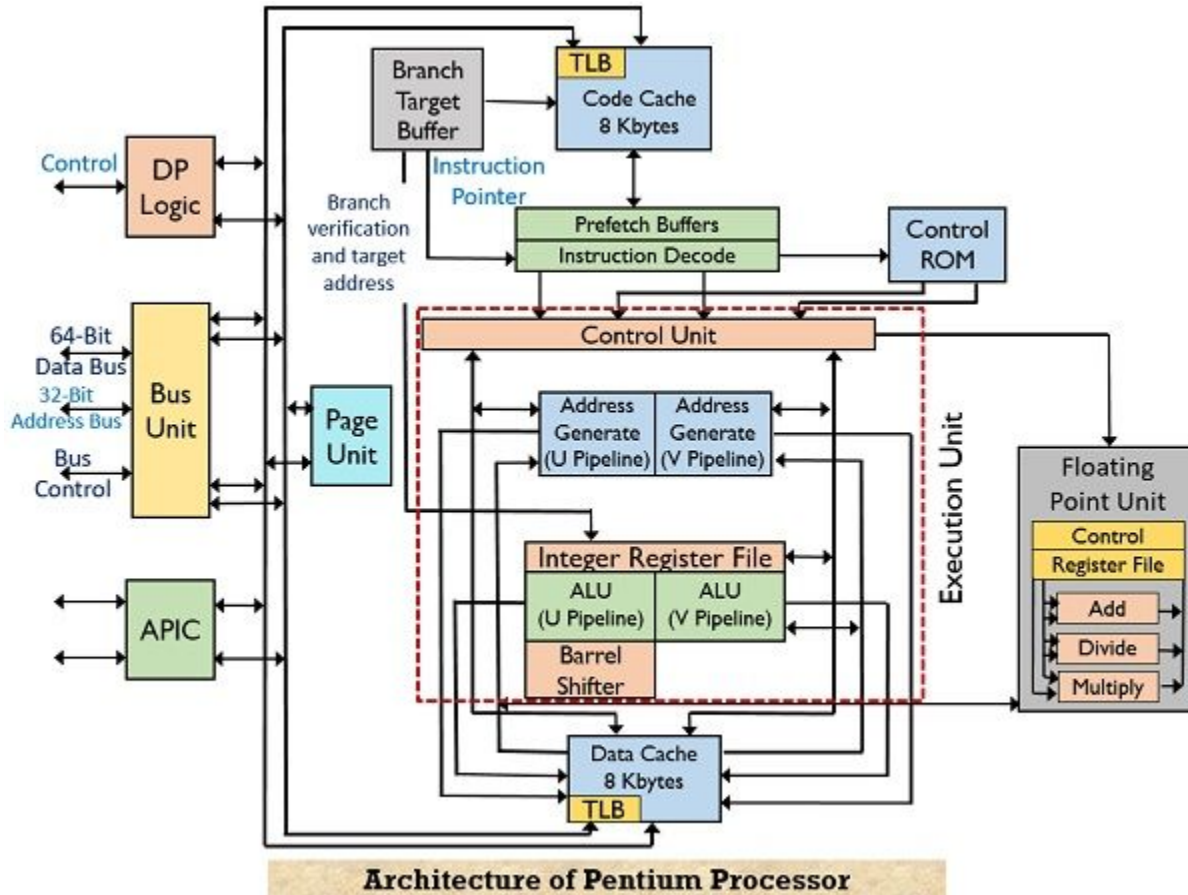
Comparison of different Processors

Parameter	8085	8086	80286	80386	80486	Pentium
N-bit Processor (data bus)	8-bit	16-bit	16-bit	32-bit	32-bit	64-bit
Address bus	16-bit	20-bit	24-bit	32-bit	32-bit	32-bit
Addressable Memory	64KB	1MB	16MB	4GB	4GB	4GB
Introduced in Year	1976	1978	1982	1985	1989	1993
Support Virtual Memory	NO	NO	YES	YES	YES	YES
Support Segmentation	NO	YES	YES	YES	YES	YES
Supports Paging	NO	NO	NO	YES	YES	YES
Operating Modes	1	2	3	3	3	3
Support Cache Memory	NO	NO	NO	NO	YES	YES
Contains on-chip FPU	NO	NO	NO	NO	YES	YES
Supports Instruction Queue	NO	YES	YES	YES	YES	YES

Features of Pentium Processor

- Separate instruction and Data caches.
- Dual integer pipelines i.e. U-pipeline and V-Pipeline.
- Branch prediction using the branch target buffer (BTB).
- Pipelined floating point unit.
- 64-bit external data bus.
- Even-parity checking is implemented for data bus, caches and TLBs.

Architecture of Pentium processor



The various functional units are as follows:

1. Bus unit
2. Paging unit
3. Control ROM
4. Prefetch buffer
5. Execution unit with two integer pipeline (U-pipe and V-pipe)
6. Code cache
7. Data cache
8. Instruction decode
9. Branch target buffer
10. Dual processing logic
11. Advanced programmable interrupt controller

Superscalar Execution

- It supports superscalar pipeline architecture.
- The Pentium processor sends two instructions in parallel to the two independent integer pipeline known as U and V pipelines for execution of multiple instructions concurrently.
- Thus, **processor capable of parallel instruction execution of multiple instructions is known as Superscalar Machine.**

-
- Each of these pipeline i.e. U and V, has 65 stages of execution i.e.
 - Prefetch
 - First Decode
 - Second Decode
 - Execute
 - Write Back

1. **Pre-fetch (PF):** In this stage, instructions are prefetched by prefetch buffer through U and V pipeline from the on-chip instruction cache.
2. **First Decode (D1):** In this stage, two decoders decode the instructions to generate a control word and try to pair them together so they can run in parallel.
3. **Second Decode (D2):** In this stage, the CPU decodes the control word and calculates the address of memory operand.
4. **Execute (EX):** The instruction is executed in ALU and data cache is accessed at this stage. For both ALU and data cache access requires more than one clock.
5. **Write Back (WB):** In this stage, the CPU stores result and update the flags.

Separate Code and Data Caches:

- Pentium has two separate 8KB caches for code and data as the superscalar design and branch prediction need more bandwidth than a unified cache.
- The data cache has a dedicated TLB to translate linear address into physical address used by data cache.
- The code cache is responsible for getting raw instructions into execution units of the Pentium processor and hence instructions are fetched from the code cache.

Advantages of having Separate Code and Data Caches

- Separate caches efficiently execute the branch prediction.
- Caches raise system performance i.e. an internal read request is performed more quickly than a bus cycle to memory.
- Separate caches also reduce the processor's use of the external bus when the same location are accessed multiple times.
- Separate caches for instructions and data allow simultaneous cache look-up.
- Up to two data references and up to 32 bytes of raw op-codes can be accessed in one clock.