80386 MICROPROCESSOR

Salient Features of 80386:

1) **Address Bus:**
80386 has a "32 bit" address bus.
This means it can access a total of 2^{32} = 4GB of physical memory.
The memory has an address range of 0000 0000H … FFFF FFFFH.

Memory Address	Data
0000 0000 h	8-bit
0000 0001 h	8-bit
0000 0002 h	8-bit
0000 0003 h	8-bit
---	---
FFFF FFFF h	8-bit

Though the total address bus is of 32 bits, only the higher 30 bits from A_{31} – A_2 are released by the µP. The lower 2 lines A_1 and A_0 are used internally by the µP to produce the four bank-enable signals $\overline{BE_3}$ … $\overline{BE_0}$. #Please refer Bharat Sir's Lecture Notes for this ...

2) **Data Bus:**
80386 has a "32-bit" data bus. This means 80386 can transfer 32-bit data at a time.
It also has a 32-bit ALU, which means 80386 can operate on 32-bit numbers in one cycle.
Hence 80386 is called a "32-bit µP".

32-bit data is stored in 4 consecutive locations.
To transfer 32-bit data in one operation 80386 memory is divided into 4 banks of 1 GB each. The banks are enabled by 4 bank-enable signals: $\overline{BE_3}$ … $\overline{BE_0}$ produced by the µP.

3) **Address Pipelining:**
80386 performs address pipelining, by putting address of the next machine cycle on the address bus, during T2 state of the current machine cycle. This makes the decoder delay transparent and is especially useful for interfacing slower devices as it reduces the number of wait states.

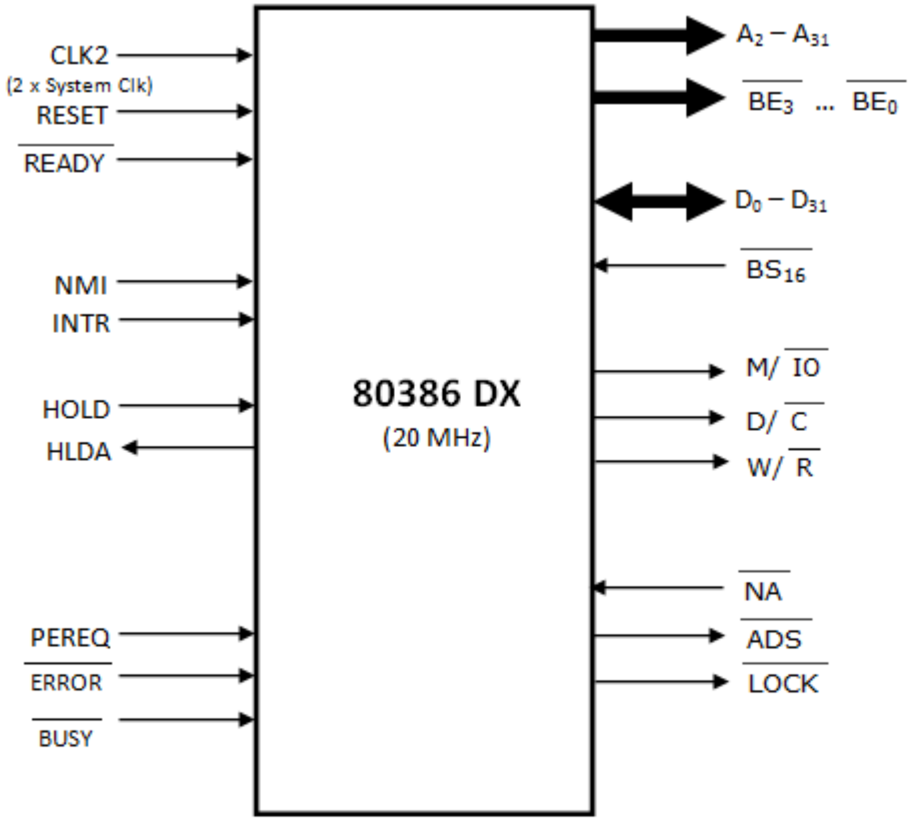
4) **Virtual Memory**:
80386 supports Virtual Memory which is implemented using Segmentation and Paging.
It can access a total Virtual Memory of 64 TB (2^{46}).

5) **Protection:**
80386 uses a protected model for accessing both memory and I/O. It uses 4 Privilege Levels.

6) **Multitasking:**
80386 allows multitasking using timesharing. Here several tasks can execute simultaneously by taking a small time slice of the µP. this gives higher system performance.

7) **I/O Addressing:**
80386 uses a 16-bit I/O address and hence can access up to 2^{16} i.e. 65536 I/O devices with address 0000 h … FFFF h.

Pin diagram of 80386 DX



There are a total of 132 pins on 80386 DX.
83 pins are shown above.
Additionally there are 20 V_{cc} pins, 21 V_{ss} and 8 NC (No Connection) pins.

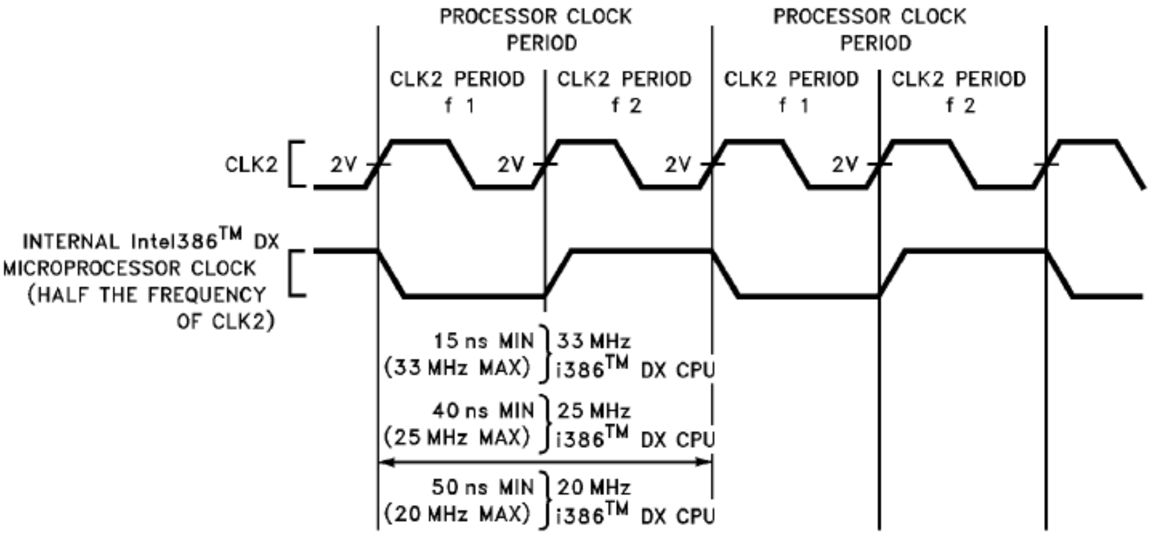
Detailed description of pins

1) CLK2 (active high input signal)

80368 has different versions working at different clock speeds as follows:

80386 DX 16, working at 16 MHz
80386 DX 20, working at 20 MHz
80386 DX 25, working at 25 MHz
80386 DX 33, working at 33 MHz

CLK2 is twice the frequency of the actual system clock. It is internally divided by two to produce the actual system clock called PCLK (Processor clock).



Henceforth each pulse of PCLK will be referred to as a T-State.

2) RESET (active high input signal)

This signal is used to reset the 80386.
On reset 80386 goes to ROM location FFFF FFF0 h, called the Reset Vector Address of 80386 and executes a BIOS program (also called Monitor Program).
The Monitor program is used to initialize the entire system.
Additionally 80386 may also execute a POST (Power-On Self Test), used to test its internal hardware. #Please refer Bharat Sir's Lecture Notes for this ...
The Self Test optional. It is only executed if during reset the $\overline{\text{BUSY}}$ pin is held low during Reset. Self Test takes approx 26 milliseconds. After the test, the result is stored in EAX register. If EAX = 00H, then the Self Test was successful, else the test was unsuccessful.

3) $\overline{\text{READY}}$ (active low input signal)

This pin is used to synchronize the µP with slower peripherals.
µP checks the $\overline{\text{READY}}$ pin during the middle of T2 state in every machine cycle.
If $\overline{\text{READY}}$ pin is = 0, that means device is ready and hence µP continues.
If $\overline{\text{READY}}$ pin is = 1, it means device is not ready hence µP inserts **"Wait States".** µP will continue to insert wait states till $\overline{\text{READY}}$ becomes 0.
This ensures that µP is synchronized with slower peripherals.

4) NMI and INTR hardware interrupt pins (both active high input signals)

	NMI	INTR
1	NMI is a non-maskable interrupt hence cannot be disabled.	INTR is a maskable interrupt. It is disabled if "IF=0" and it is enabled by if "IF=1".
2	NMI is higher priority.	INTR is lower priority.
3	NMI is edge triggered.	INTR is level triggered.
4	NMI is a vectored interrupt.	INTR is non-vectored.
5	On receiving NMI interrupt, µP performs the ISR of INT2.	On receiving INTR interrupt, µP performs two $\overline{\text{INTA}}$ cycles and obtains the vector number of the ISR to be executed.

5) HOLD and HLDA pins (both active high, HOLD is input and HLDA is output)

HOLD and HLDA are used for DMA purposes. DMA means transferring data directly between Memory and I/O without involving the µP.
To do DMA transfer, the DMA Controller requires control of the system bus.
It gives HOLD to the µP (i.e. HOLD =1).
Now µP finishes the current bus cycle (machine cycle) and releases control of the system bus, and gives HLDA.
This makes µP enter HOLD state and DMA Controller now becomes the bus master. After performing the bus operation DMA Controller makes HOLD=0, thereby returning control of the system bus back to the µP.
Advantage of DMA transfer is that it is much faster as compared to transfers performed by the µP.

6) Co-Processor interface pin: PEREQ, $\overline{\text{ERROR}}$ and BUSY.

a) PEREQ: (Peripheral Extension Request) (active high input signal)

It is issued by the Co-Processor 80387 (or even 80287).
Whenever the Co-Processor wants to perform a data transfer using the system bus, the operation must be initiated by the μP. This is because, the physical address calculation and protection checking is only performed by the μP.
Hence if 80387 makes PEREQ = 1, it requests the μP to initiate the bus operation. Now μP performs all calculations and finally places the physical address on the address bus. The data although will be transferred by 80387.

b) $\overline{\text{ERROR}}$: (Co-Processor Error) (active low input signal)

While performing numerical operations, 80387 may encounter various errors also called exceptions. These exceptions can be masked or unmasked. Whenever any of the unmasked errors occur, 80387 makes $\overline{\text{ERROR}}$ = 0.
This also causes INT 16 (Co-Processor Error Exception).

c) $\overline{\text{BUSY}}$: (Co-Processor Busy) (active low input signal)

This signal is used by 80387 to indicate 80386 whether it is busy or not.
If 80387 is busy, it makes $\overline{\text{BUSY}}$ = 0.
μP checks the $\overline{\text{BUSY}}$ pin whenever it encounters a WAIT instruction or a Co-Processor instruction.
If $\overline{\text{BUSY}}$ = 0, that means 80387 is still busy performing the previous execution hence μP enters wait states till 80387 becomes free i.e. $\overline{\text{BUSY}}$ becomes 1.

This signal is also used to indicate whether a Self Test (POST) must be performed during System RESET.
If during RESET, $\overline{\text{BUSY}}$ = 0, then μP performs a self test as explained earlier.

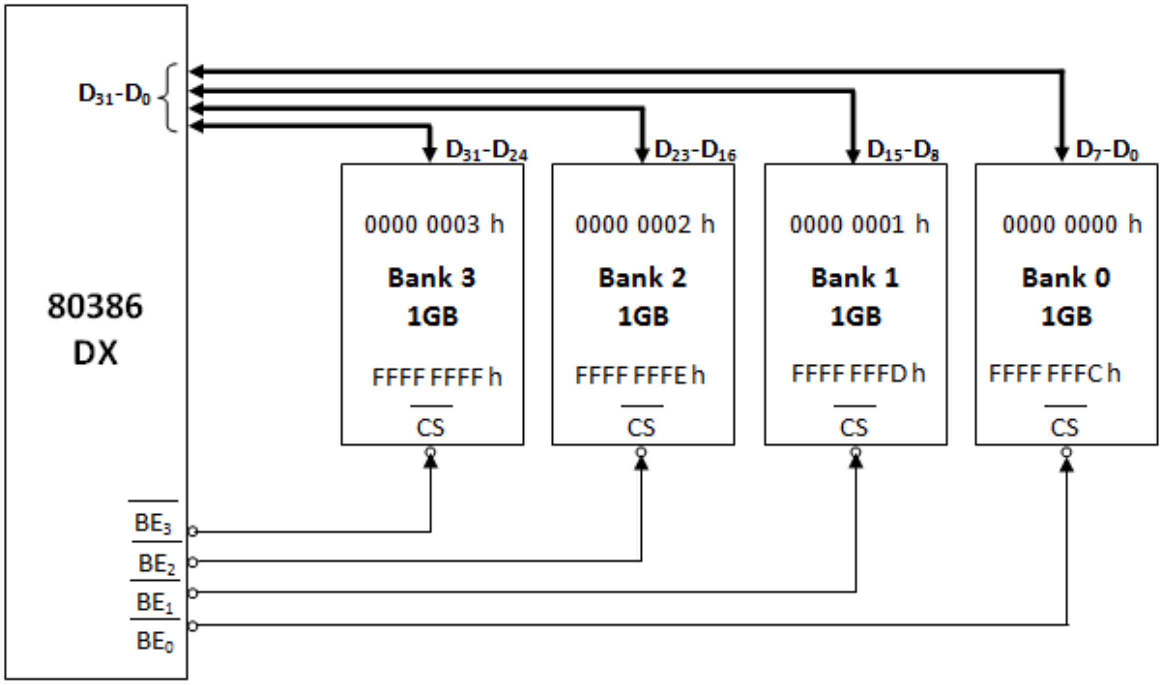
7) Address Bus (A31 – A2) (output signals)

80386 has a "32 bit" address bus.

This means it can access a total of 2^{32} = 4GB of physical memory.
The memory has an address range of 0000 0000H … FFFF FFFFH.
Though the total address bus is of 32 bits, only the higher 30 bits from A_{31} – A_2 are released by the μP. #Please refer Bharat Sir's Lecture Notes for this …
The lower 2 lines A_1 and A_0 are used internally by the μP to produce the four bank-enable signals $\overline{BE_3}$ … $\overline{BE_0}$.

8) $\overline{BE_3}$ … $\overline{BE_0}$ - Bank Enable Signals (active low output signals)

As 32 bit data has to be accessed from 4 locations, the 80386 memory is divided into 4 banks. Each bank is enabled by its respective bank enable signal. Each bank is of 1GB, making it total 4 GB. The addresses are distributed in such a way that the 4 consecutive locations carrying 32-bit data are spread across 4 different chips (banks).



The four Bank enables are produced by the μP, in the following manner:

$\overline{BE_3}$	$\overline{BE_2}$	$\overline{BE_1}$	$\overline{BE_0}$	OPERATION
1	1	1	1	No Operation, No Bank Selected
1	1	1	0	8-bit operation using Bank 0
1	1	0	1	8-bit operation using Bank 1
1	0	1	1	8-bit operation using Bank 2
0	1	1	1	8-bit operation using Bank 3
1	1	0	0	16-bit operation using Bank 1 & Bank 0
1	0	0	1	16-bit operation using Bank 2 & Bank 1
0	0	1	1	16-bit operation using Bank 3 & Bank 2
0	0	0	0	32-bit operation using all the Four Banks

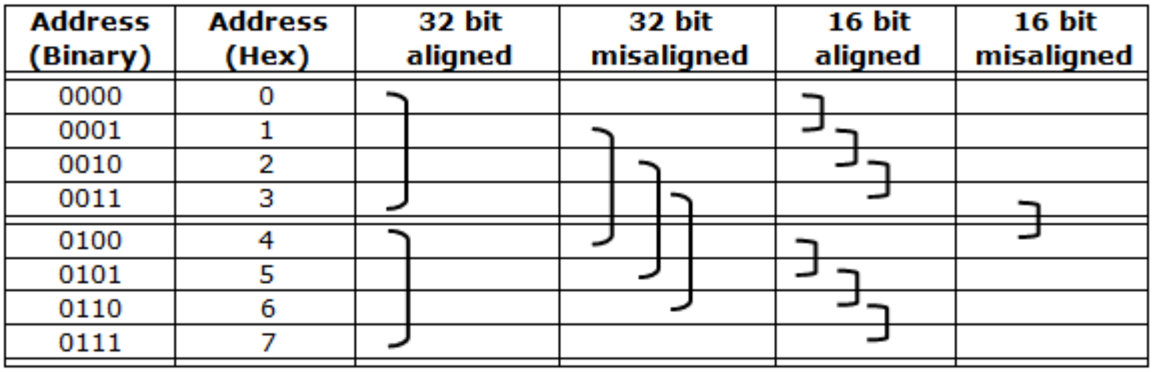
Note: Aligned and misaligned data {Important for VIVA}

Any data that can be transferred in one cycle is called aligned data. Misaligned data requires two cycles to be transferred.

32-bit operations: Any 32-bit data stored starting from a memory location which is a multiple of 4 (i.e. last two bit of address are 00) is aligned.

16-bit operations: Any 16-bit data stored starting from a memory location whose last two bits are 00 or 01 or 10 is aligned.

8-bit operations: No such issue as 8-bit data is always transferred in one cycle.



Address (Binary)	Address (Hex)	32 bit aligned	32 bit misaligned	16 bit aligned	16 bit misaligned
0000	0				
0001	1				
0010	2				
0011	3				
0100	4				
0101	5				
0110	6				
0111	7				

Another way of putting it is that an aligned data is such that it is stored in the correspondingly same location in each of the banks.

9) $D_{31} - D_0$ (32-bit data bus) (bidirectional signals)

80386 has a 32 bit data bus. It can perform 8-bit, 16-bit and 32-bit data transfers using 4 memory banks as explained earlier.

10) $\overline{\text{BS16}}$ - Dynamic Data Bus Sizing (active low input signal)

80386 allows dynamic data bus sizing. Although it has a 32-bit data bus, 80386 allows the user to choose between a 32-bit data bus and a 16-bit data bus.
A 32-bit bus gives faster performance but a 16-bit bus leads to a much simpler and cheaper circuit design. Also, 16-bit data bus was used in 8086 systems, so the 16-bit mode helps those who want to upgrade from and 8086 based system to a 80386 based system without having to entirely redesign the memory and I/O interface. ☺ **For doubts contact Bharat Sir on 98204 08217**

If $\overline{\text{BS16}}$ **= 0, the data bus is 16-bit so only $D_{15} - D_0$ are used.**

If $\overline{\textbf{BS16}}$ **= 1, the data bus is 32-bit so entire D_{31} – D_0 are used.**

11) Control Signals M/ $\overline{\text{IO}}$, D/ $\overline{\text{C}}$, W/ $\overline{\text{R}}$ (output signals)

These control signals are decoded by external bus control logic to decide which machine cycle has to be performed and produce the signals accordingly as shown.

M/ $\overline{\text{IO}}$	D/ $\overline{\text{C}}$	W/ $\overline{\text{R}}$	Machine cycle
0	0	0	INTA
0	0	1	Inactive
0	1	0	I/O Read (Data)
0	1	1	I/O Write (Data)
1	0	0	Memory Read (Code)
1	0	1	Halt
1	1	0	Memory Read (Data)
1	1	1	Memory Write (Data)

M/ $\overline{\text{IO}}$: Indicates whether it is a Memory or an I/O cycle

1 = Memory, 0 = I/O.

D/ $\overline{\text{C}}$: Indicates whether it is a Data or a Control

1 = Data, 0 = Control
Data operations are MemR, MemW, IOR and IOW
Control operations are mainly Instruction fetch (MemR - Code) and INTA.

W/ $\overline{\text{R}}$: Indicates whether it is a read or a write cycle

1 = Write, 0 = Read.

12) $\overline{\text{LOCK}}$: (active low output signal)

$\overline{\text{LOCK}}$ signals that μP is doing an instruction with a LOCK prefix.

LOCK prefix is used to prevent the μP from releasing the system bus for a bus request during an instruction. Normally, if a bus request occurs during an instruction, μP releases the bus just after finishing the current machine cycle. But once we write LOCK, μP will only release the bus after the current instruction, and will prevent external bus arbitration logic also from releasing the bus by making $\overline{\textbf{LOCK}}$ = 0.

13) $\overline{\text{ADS}}$: Address Status (active low output signal)

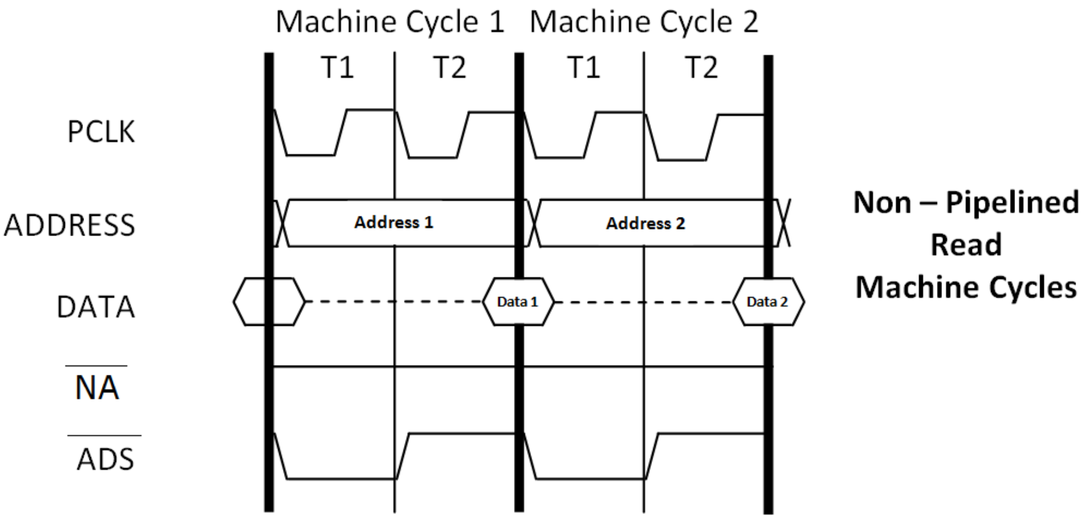
This signal is asserted (made low), when a new address is put on the address bus.

In a non pipelined cycle, the $\overline{\textbf{ADS}}$ goes low during T1 T-State of a machine cycle.

In a pipelined cycle the $\overline{\textbf{ADS}}$ goes low during T2 T-State of the previous machine cycle. It is used to signal the external circuit that μP has put a new address on the address bus.

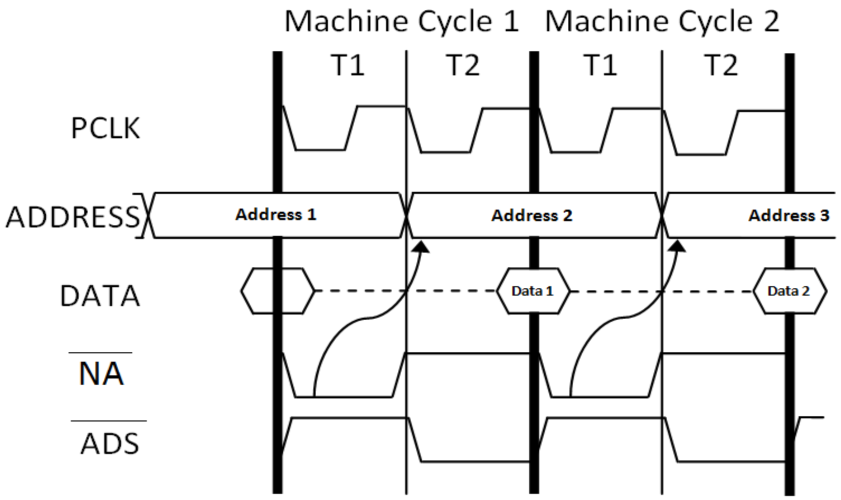
14) $\overline{\textbf{NA}}$: Next Address (used for Address Pipelining) (active low input signal)

This signal is responsible for performing address pipelining.
A machine cycle in 80386 is of 2 T-States.
In a Non-Pipelined Read cycle, 80386 sends address in the first T-State and gets the data in the second T-State. This completes the current machine cycle. In the T1 T-State of the next machine cycle, 80386 will again give the address and the process continues, as shown…



The problem here is, if the device from which μP is trying to read data, is slow, then it will not be able to send the data by T2, making READY = 1, thereby inserting extra wait states. This makes the entire system slow.

To solve this problem we use Address Pipelining. Here, the μP will start sending the address of the next machine cycle in the T2 state of the current machine cycle. This makes the decoder delay transparent and hence gives the device moiré time to get ready and start sending the data. In effect the number of wait states will get reduced and avoids making the entire system slow.



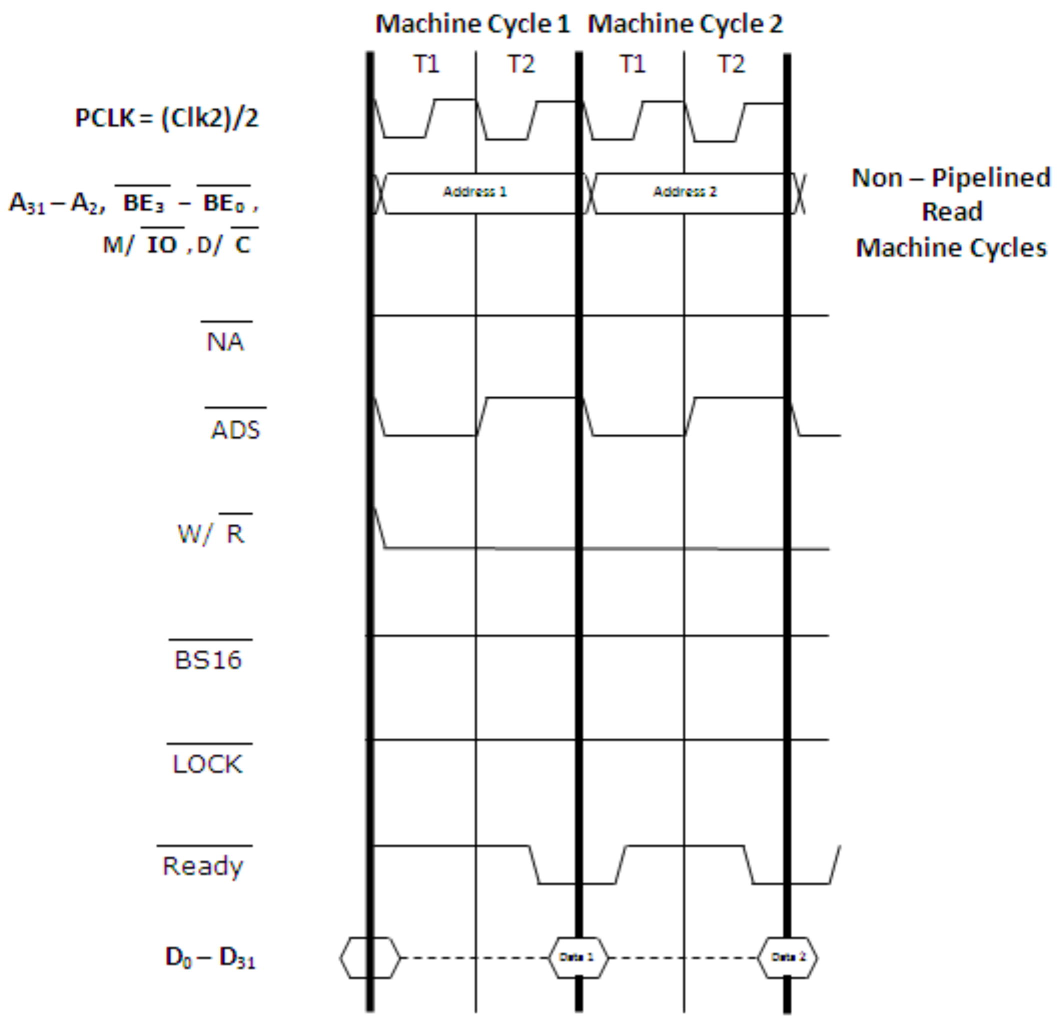
Pipelined Read Machine Cycles

Address pipelining is optional and is only performed if NA = 0.
NA is checked during T1 state of a machine cycle. If NA = 0, the next address is sent during T2 state, else the next address is sent in the next machine cycle.

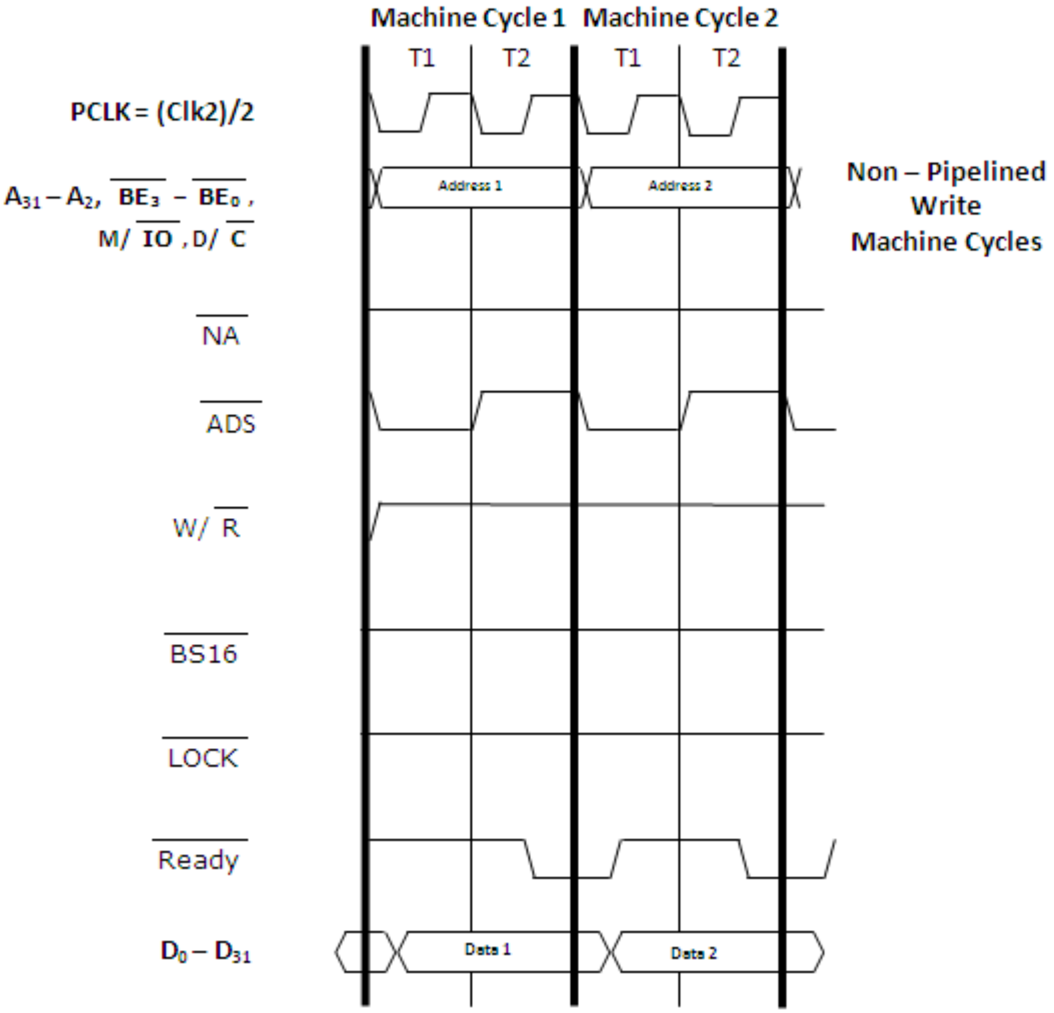
Compare 80386 SX and 80386 DX versions #Please refer Bharat Sir's Lecture Notes for this ...

	80386 DX	80386 SX
1	80386 DX has a 32 bit data bus.	80386 SX has a 16 bit data bus.
2	Due to 32 bit data bus, the execution speed is higher. Hence the name: DX – Double Execution speed.	Due to 16 bit data bus, the execution speed is lower. Hence the name: SX – Single Execution speed.
3	32-bit transfers require 4 Memory Banks.	16-bit transfers require 2 Memory Banks.
4	Has 4 Bank enable signals: $\overline{BE_3}$, $\overline{BE_2}$, $\overline{BE_1}$, $\overline{BE_0}$.	Has only 2 Bank enable signals: $\overline{BHE}$ And $\overline{BLE}$.
5	4 Bytes are fetched at once in the pipelining queue.	2 Bytes are fetched at a time in the pipelining queue.
6	Has dynamic data bus sizing of 16-bit and 32-bit data bus, using $\overline{BS16}$ signal.	No such option available as the data bus is only of 16-bits. Hence $\overline{BS16}$ signal not useful.
7	Used for high performance.	Used for low cost memory and I/O system design.
8	Comes in a 132-pin ceramic PGA (Pin Grid Array) package for higher performance.	Comes in 100 lead plastic quad flat packages (PQFP) to permit lower cost.

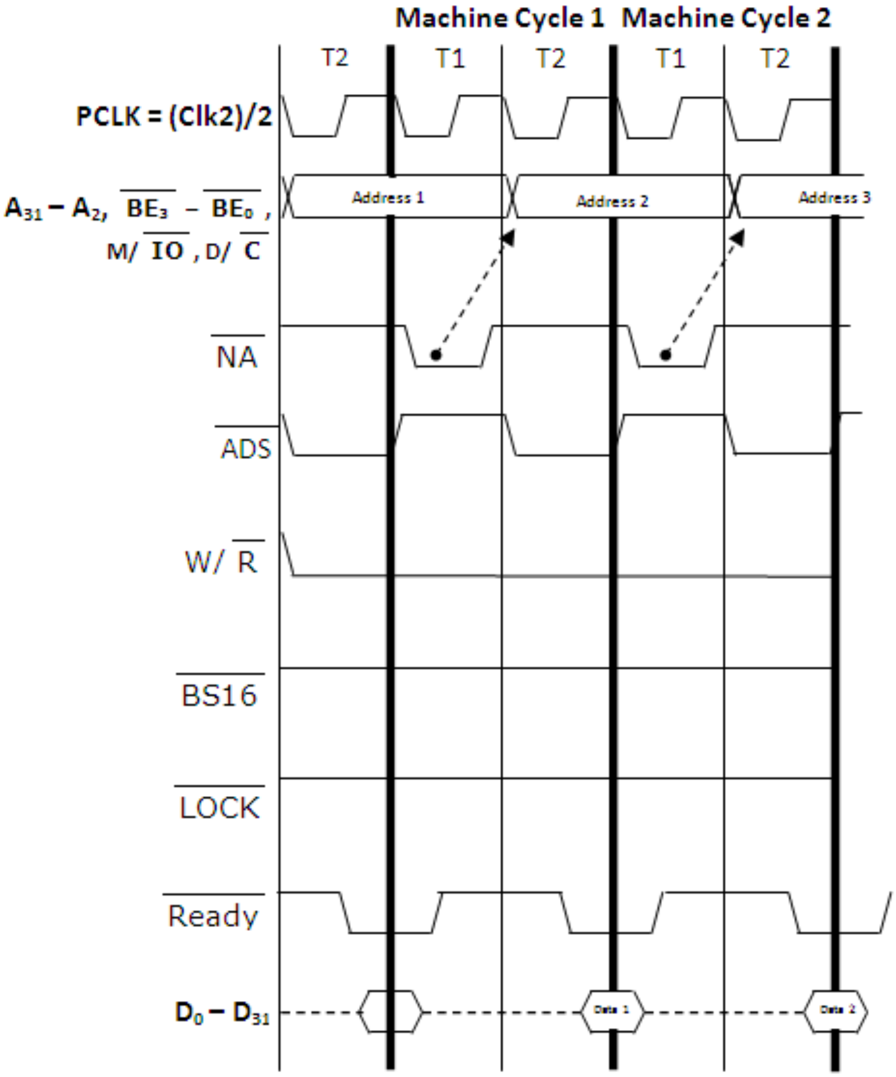
DETAILED TIMING DIAGRAMS



#Please refer Bharat Sir's Lecture Notes for this ...

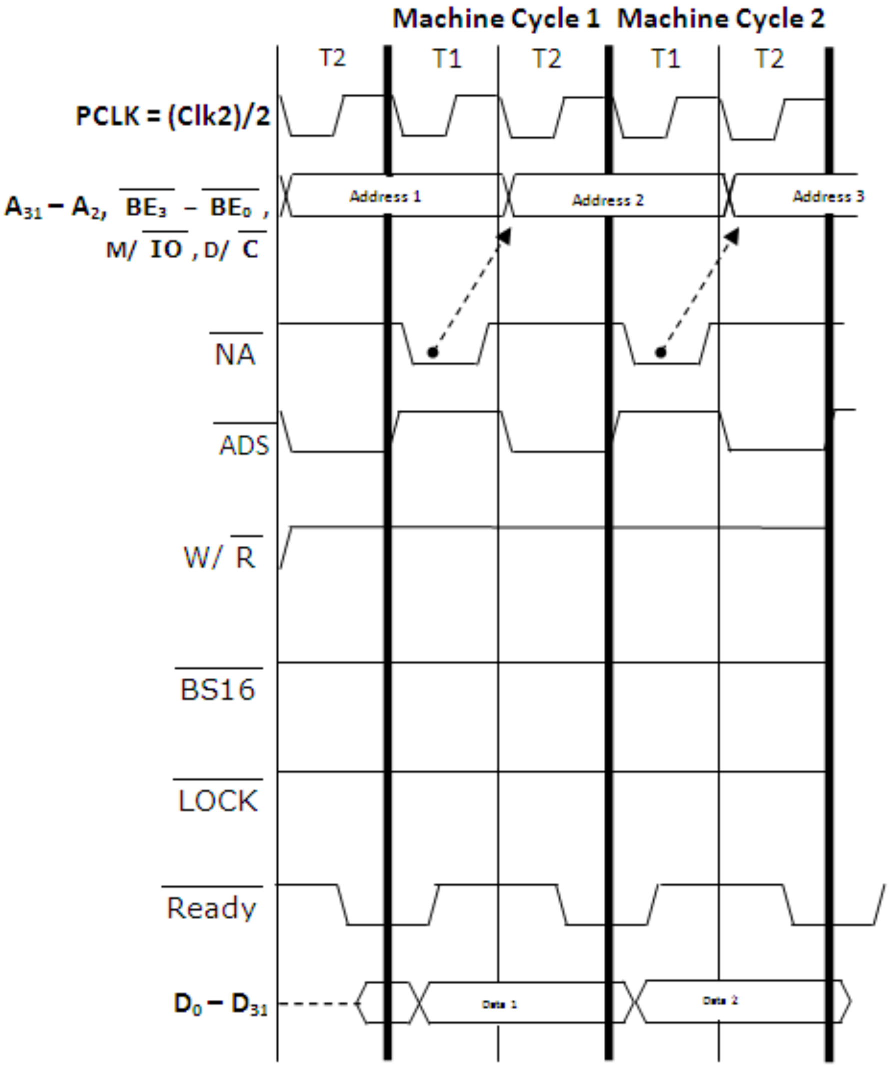


#Please refer Bharat Sir's Lecture Notes for this ...



Pipelined Read Machine Cycles

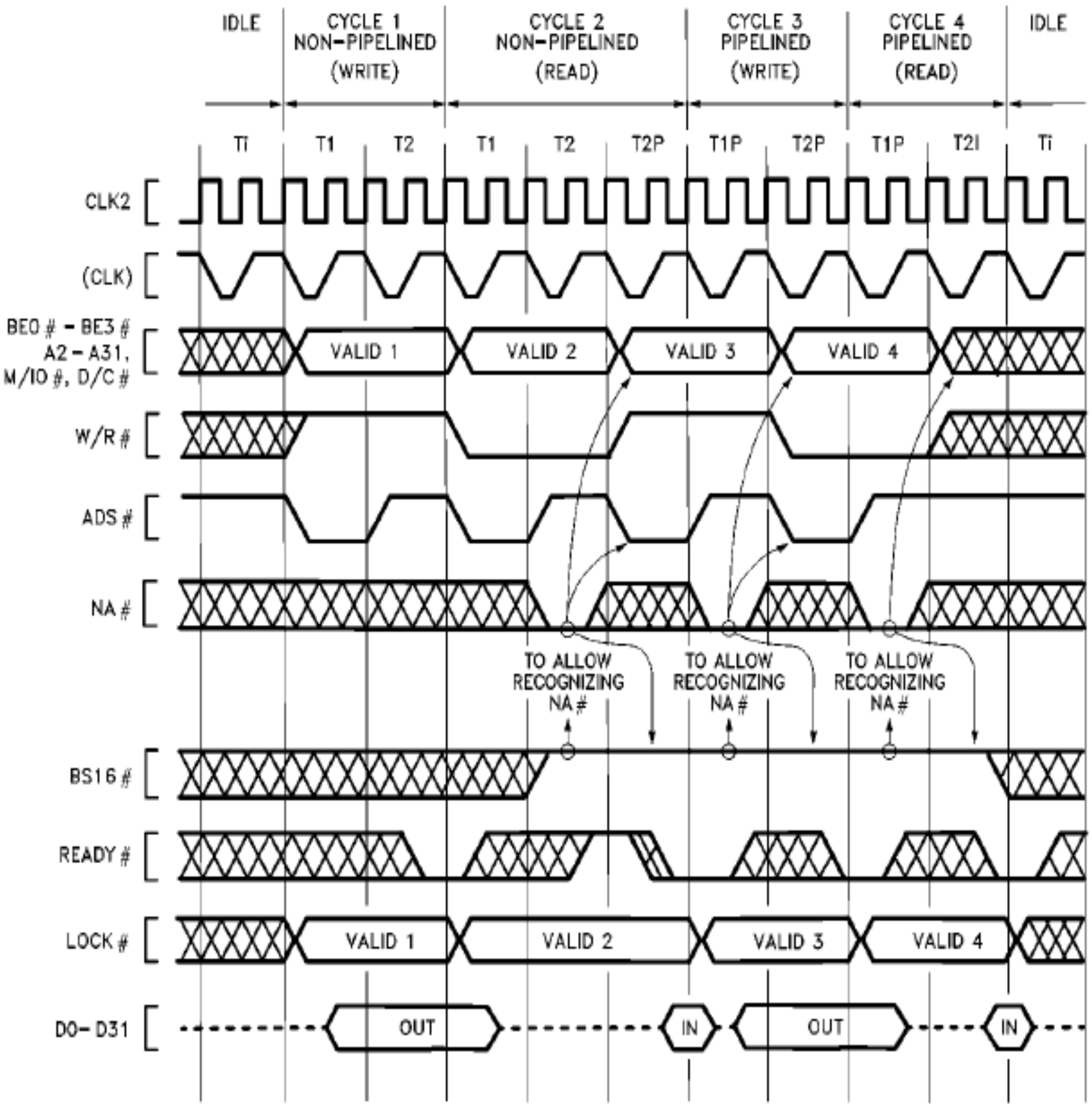
#Please refer Bharat Sir's Lecture Notes for this ...



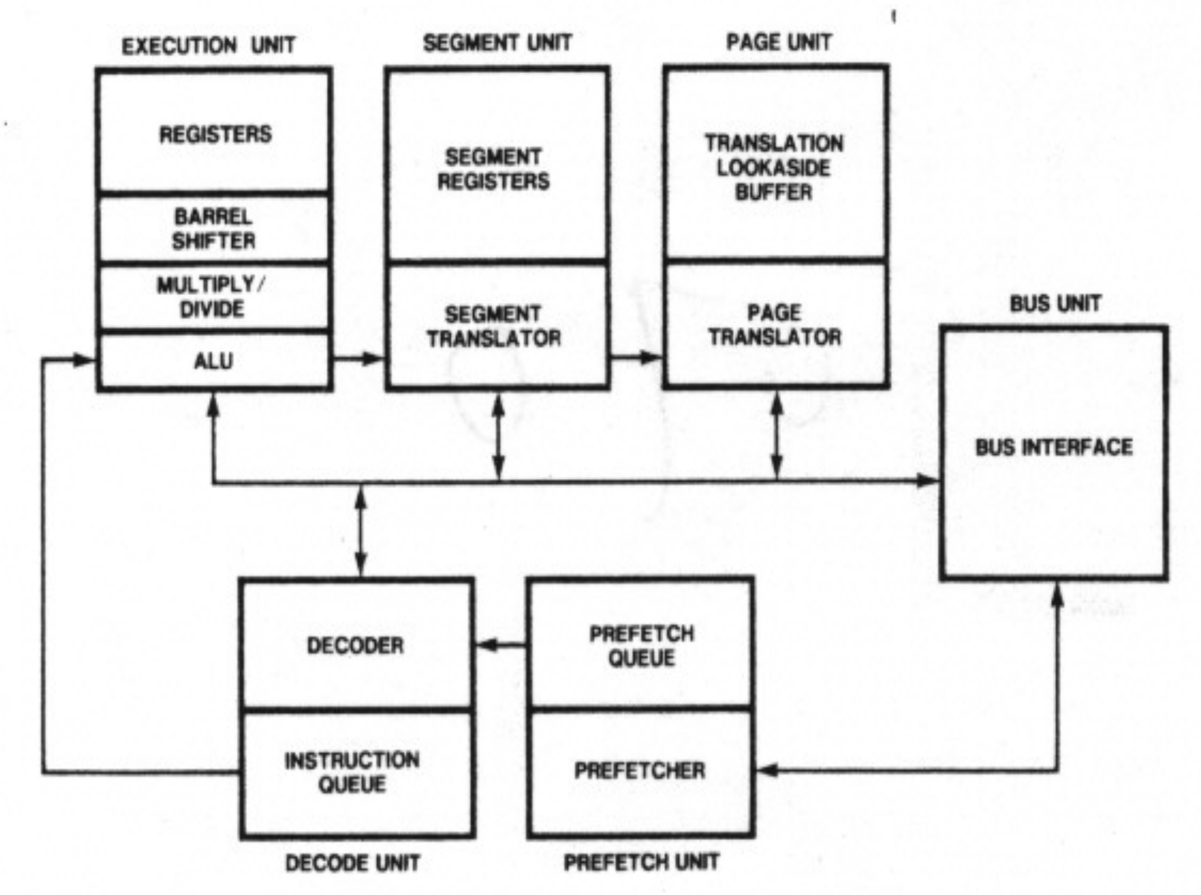
Pipelined Write Machine Cycles

#Please refer Bharat Sir's Lecture Notes for this ...

COMBINED TIMING DIAGRAMS OF PIPELINED AND NON PIPELINED READ AND WRITE CYCLES



80386 ARCHITECTURE / FUNCTIONAL BLOCK DIAGRAM



80386 architecture is divided into 5 independent units.

Bus Unit (Bus Interface Unit)

1) **The Bus unit is responsible for transferring data in and out of the µP.**
2) It is connected to the external memory and I/O devices, using the system bus.
3) It gets requests from Prefetch unit for fetching instructions and from execution unit for transferring data.
4) If both requests occur simultaneously preference is given to execution unit.

Prefetch Unit

1) The Pre-fetch unit **fetches further instructions in advance to implement pipelining.**
2) It **fetches the next 16 bytes of the program** and stores it into the **Prefetch Queue**.
3) It **refills the queue** when at least **4 bytes** are empty as 80386 has a 32 bit data bus.
4) During a **branch**, the instructions in the queue are **invalid** and hence are **discarded**.

Decode Unit

1) 80386 µP has a **separate unit for decoding instructions** called the Decode Unit.
2) It **decodes the next three instructions and keeps them ready** in the Decode Queue.
3) The decoded instructions are stored in **Micro-Coded** form.
4) During a **branch**, the instructions in the queue are **invalid** and hence are **discarded**.

Execution Unit

1) Execution Unit performs the main task of **executing instructions**.
2) Normally, execution requires Arithmetic or Logic operations performed by a **32-bit ALU**.
3) It also has dedicated circuits for **32-bit multiplication and division**.
4) A **64-bit barrel shifter** is also provided for faster shifts during multiplication and division.
5) **Operands** for the ALU can either be provided in the **instruction**, or can be taken **from memory** or could be taken from the **32-bit registers** like EAX, EBX etc.
6) Additionally there is a **32-bit Flag register** (EFLAGS) giving the **Status** of the current result.

Memory Unit

1) The Memory unit converts Virtual Address (Logical address) to Physical Address.
2) 80386 µP implements **64 Terra bytes of Virtual memory using Segmentation and Paging**. Hence the Memory Unit is sub-divided into Segmentation Unit and Paging Unit.
3) **Segmentation is compulsory, while Paging is optional.**
4) **The Segmentation Unit converts the Logical Address into a Linear Address.**
5) **The Paging Unit converts the Linear Address into a Physical Address.**
6) If Paging is not used, then the Linear Address itself is the Physical Address.