UNIT I

Introduction to 80386

* Architecture of 80386 -

→ • Internal Architecture is divided into 3 parts:

1) Central Processing Unit -
   - It is further divided into Execution Unit and Instruction Unit
   - Execution Unit has 8 General & 8 Special purpose registers for handling data
   - Instruction unit decodes opcode bytes received from 16-byte instruction code

2) Memory Management Unit -
   - It consists of Segmentation Unit & Paging Unit
   - Segmentation unit allows use of segment & offset address for relocability
   - Paging unit organizes physical memory in terms of pages of 4kb size each.

3) Bus Interface Unit -
   - It consists of Bus Control Unit
   - Bus Control Unit has a prioritizer to resolve priority of various bus requests
   - This controls the access of bus.

★ Flag Register (80386) -

→ • Flag Register is a part of EU, consisting of 32 flip flops

1) Status Flags (Conditional flags) :

(i) Carry Flag (CF) -
It is set by arithmetic instructions that generate carry / borrow

(ii) Parity Flag (PF) -
It is set, if LSB of 8 bit of destination operand contain an even no. of 1's

(iii) Auxillary Carry Flag (AF) -
It is set, if there is carry or borrow after nibble addition / subtraction

(iv) Zero Flag (ZF) -
It is set to 1, if result of operation is 0

(v) Sign Flag (SF) -
It is set, if result of operation is negative (MSB = 1)

(vi) Overflow Flag (OF) -
It is set, if signed result is out of range

2) Control Flags :

(i) Trap Flag (TF) -
When set, it enables trapping through on chip debugging features

(ii) Interrupt Flag (IF) -
It controls the operation of INTR input pin
If IF = 1, INTR pin is enable, else, disabled

(iii) Direction Flag (DF) -
It controls the direction of string operations
If DF = 1, then ESI & EDI register decrement by 1
else, get increment by 1

3) System Flags :

(i) IOPL (IIO Privilege Level) -
The 2 bits in IOPL are used by processor to determine your access to IIO facilities

(ii) Nested Flag (NF) -
It is set when one system task invokes another task

(iii) Resume Flag (RF) -
When set, allows selective masking of some exceptions at time of debugging

(iv) VM flag -
It indicates operating mode of 80386.

When set, 80386 switches from protected mode to virtual mode

* Addressing Modes in 80386 -

→ 1) Register Addressing Mode -
- Data is stored in register & is referred using particular register
- Example : MOV AX, BX
ADD EAX, EBX

2) Immediate Addressing Mode -
- In this, immediate data is part of instruction
- Example : MOV EAX, 12345678H

3) Direct Addressing mode -
- 16 bit memory address (offset) is directly specified in instruction as part of it
- Example : MOV EAX, [5000H]

4) Register Indirect Addressing mode -
- Address of memory location containing data is determined in indirect way using offset register
- Offset address of data is in either EBX / ESI / EDI register

- Example : MOV AX, [EBX]

5) Based Index Addressing Mode -
- Contents of Base register (EBX / EBP) are added to Index register (ESI / EDI)
- Eg : MOV AX, [EBX][ESI]

6) Scaled Index Addressing mode -
- In this, index register is multiplied by 1, 2, 4 or 8
- Eg : MOV EAX, LIST[ESI * n]

* Data Types in 80386 -

→ Fundamental Data Types :

1) Bytes :
- Byte is a 8 contiguous bits starting at any logical address

2) Word :
- word is a two contiguous bytes starting at any byte address
- A word thus contains 16 bits

3) Doubleword :
- Doubleword is two contiguous words starting at any byte address

- A doubleword thus contains 32 bits

★ Instructions Set of 80386 -

Q. How IMUL is different from MUL ?

→ • MUL - Product after multiplication is always a double width product.
i.e if two 8 bit nos. are multiplied it generates 16 bit result
- It performs unsigned multiplication with AL / AX register
- Eg : MUL BH : (AX) ← (AL) x (BH)

• IMUL -
- It performs signed multiplication.
- It multiplies signed byte / word by AL / AX
- Example : IMUL BL : (AX) ← (AL) x (BL)

List of Instructions:

1) XCHG - (Exchange)
It swaps the contents of two operands (Immediate data is not allowed)
Eg : XCHG AX, BX

2) OUT -
Used for writing to an output port. Address of output port may be specified directly / indirectly
e.g OUT DX, AX

3) XLAT -
Used for code conversion using look up table
Eg : Hexadecimal keypad code is converted into 7 segment display device

4) LEA - (Load Effective Address)
It loads E.A formed by destination operand into specified source register
Eg : LEA BX, ADR

5) LAHF -
Load AH from Lower Byte of flag Register
Flag Affected : None

6) CWD - (Converted Signed Word to Double Word)
It copies sign bit of AX to all bits of DX register. Done before signed division
Flag Affected : None

7) CMP –
- It compares source operand (R/m/I) with destination operand (R/m)
- It subtracts source operand from destination operand but does not store result
- Flag Affected:
  If S = D, then ZF = 1
  If S > D, then CF = 1, SF = 1
  If S < D, then CF = 0, SF = 0
- Eg : CMP D, S

8) DIV –
It divides an unsigned word/ doubleword by 8/16 operand
Flag Affected : OF, CF, SF, ZF, AR, PF are undefined

9) AAA – ASCII Adjust AL After Addition
- It is executed after ADD instruction
- It converts result in AX into unpacked decimal digit
- If (lower 4 bits of AL) > 9 or AF = 1 then
  AL = AL + 6 and AH = AH + 1
  Sets AF = 1 & CF = 1
  else
  AF = 0 & CF = 0
  in both cases : clear high nibble of AL

Similar you can write for Subtract, multiply, divi

10) DAA - (Decimal Adjust AL after Addition)
- It is used to convert result of addition into packed BCD
- If lower nibble > 9 or AF = 1, then add 06h to AL
- If higher nibble > 9 or CF = 1, then add 60h to AL
- Flag Affected: AF, CF, PF, ZF

- Example: ADL = 53, CL = 29
ADD AL, CL : AL ← (AL) + (CL)
: AL ← 7C
DAA : AL ← 7C + 06 (as C > 9)
AL ← 82

11) Shift & Rotate Instructions:

(a) SAL / SHL -
It shifts operand word / byte bit by bit to left and inserts zeros in new LSB
Example:
Operand 1010 1100 1010 0101
SHL 1010 1001 0100 1010 ← inserted

(b) ROL and ROR -
ROL (Rotate left) rotates the byte, word, double word destination operand left by one or by no. of bit specified
ROL: MSB -> CF, LSB

Example:

Operand 1010 1111 0101 1101

1 0101 1110 1011 1011

12) Control Transfer Instruction -

(a) ENTER -

- Creates a stack frame for procedure. Dynamically allocates stacks space for procedure
- Syntax: ENTER storage, level
- Example: ENTER 16, 0

(b) LEAVE -

- Releases local variables created by ENTER instruction by restoring SP and BP to their condition before procedure stack frame was initialized
- Syntax: LEAVE; POP SP, POP BP

(c) BOUND -

- It ensures that signed array index is within limits specified by block of memory consisting of upper & lower bound
- Example: BOUND AX, mem_word
  BOUND AX, mem_limits

13) WAIT & LOCK -

ⓐ WAIT :

It is used to prevent CPU from accessing memory that may be temporarily in use by coprocessor

Syntax : WAIT

ⓑ LOCK -

Causes x86 LOCK Pin to be asserted & thereby causing external bus masters to be disabled

14) BSR - (Bit Scan Reverse)

- It scans operand from left to right for 1
- If 1 is encountered, ZF = 1 & location of it is stored in destination operand

15) BT -

Debug trap due to task switch

16) SLDT -

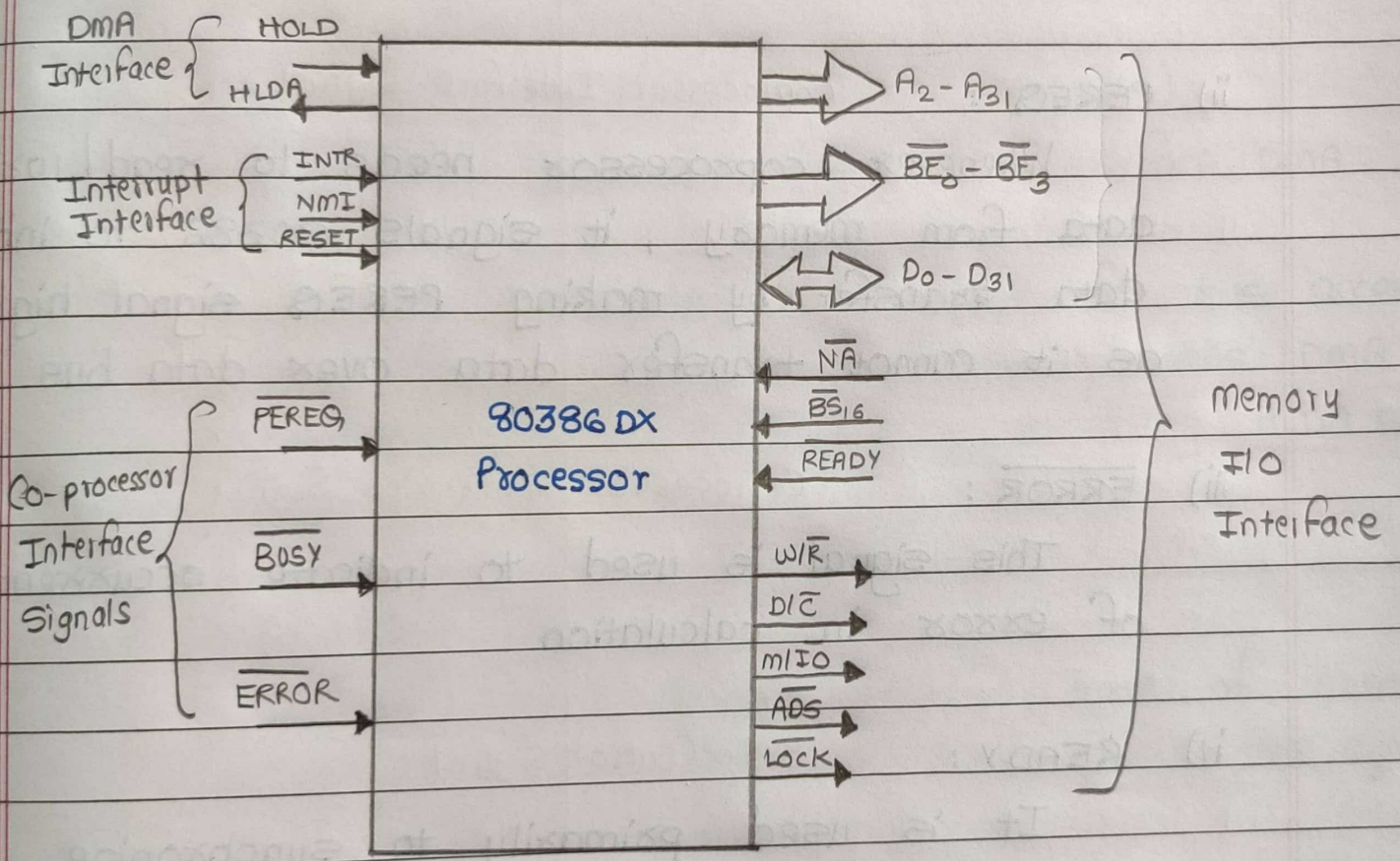
- It stores Local Descriptor Table Register (LDTR) in two-byte register / memory location indicated by effective address operand
- Flag Affected : None
- Example : SLDT BX

UNIT - II

Bus Cycles & System Architecture

* Processor State after RESET :

▪ Functional Pin Diagram of 80386 :



* Functional Pin Diagram of 80386

i) $\overline{BE_3}$ - $\overline{BE_0}$: Memory / IO interface signals

The lower 2-bits of the 32-bit address generated by 80386DX, are internally decoded and sent on four byte enable pins ($\overline{BE_3}$ - $\overline{BE_0}$)

ii) PEREQ : Coprocessor Interface signal

Whenever coprocessor needs to read/write data from memory, it signals 80386 to initiate data transfer by making PEREQ signal high, as it cannot transfer data over data bus

iii) $\overline{ERROR}$: Coprocessor Interface Signal

This signal is used to indicate occurrence of error in calculation

iv) READY : Bus Control signal

It is used primarily to synchronize slower peripherals with microprocessor
If READY signal is 0, IIO devices tell 80386DX that they are ready for next data transfer
If READY signal is 1, then processor enters wait state.

v) $\overline{NA}$: (Bus Status Signal)

External bus control logic controls this signal. It activates pipelining by making next address request low input

v) NMI:- (Interrupt Interface Signal)

This interrupt input is non-maskable and edge-triggered. A valid interrupt on this pin causes 80386 to execute interrupt service routine.

vi) HOLD & HLDA - (DMA interface signal)

- These pins are used to interface DMA controller
- DMA controller can request for bus access by asserting HOLD pin & 80386DX tells DMA that buses are available by asserting HLDA signal

Bus Status Signals

Write/ Read (W/$\bar{R}$) -

This signal decides specific type of memory or I/O operation that will occur during bus cycle

Memory / I/O (M/$\bar{IO}$) -

This signal identifies whether current bus cycle is memory or I/O cycle

Data / Control (D/$\bar{C}$) -

This signal identifies whether current bus cycle is data or control cycle

★ Difference : I/O mapped I/O vs memory mapped I/O

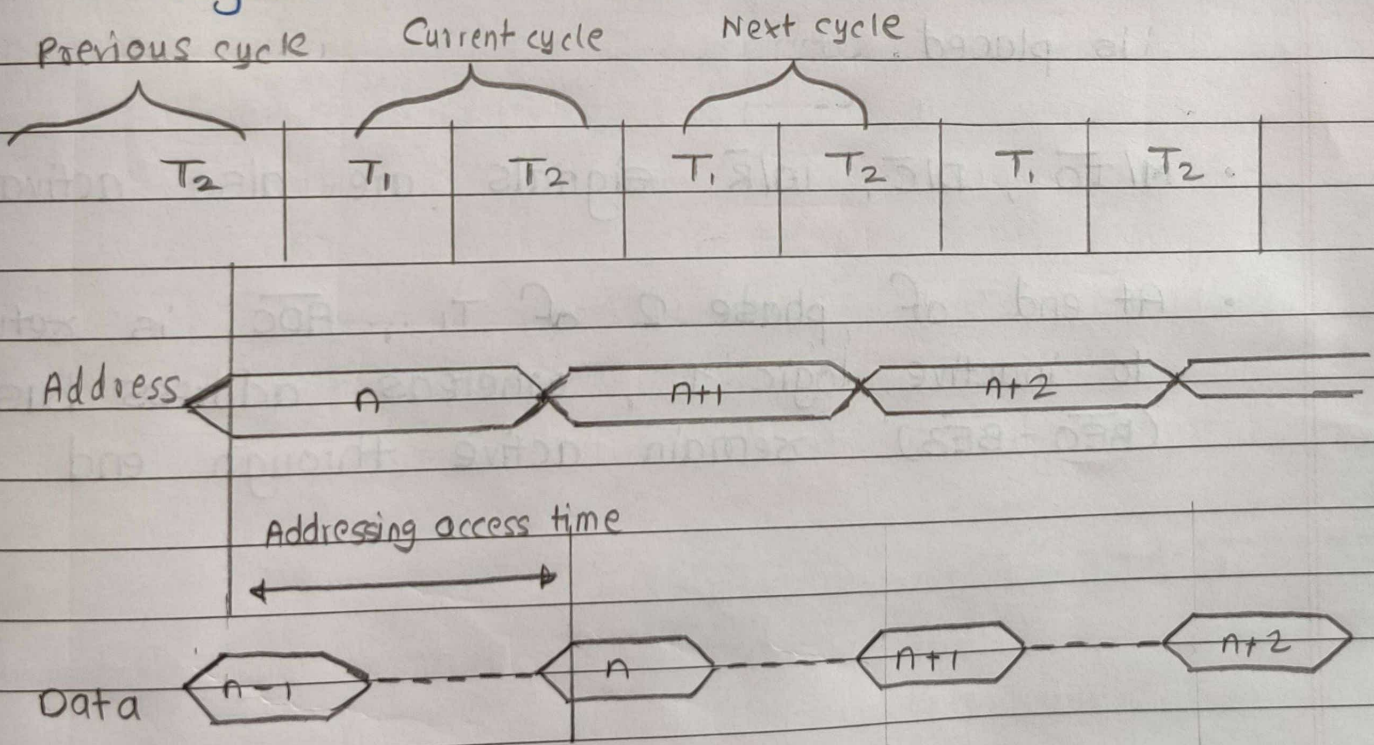
→ Memory mapped I/O	I/O mapped I/O
i) I/O devices are placed in memory address space	i) Provides seperate I/O space, distinct from physical memory
ii) All memory-related instructions can be used to access I/O device	ii) Only I/O related instructions are used to transfer data
iii) Max no. of I/O device can be connected is more due to more memory address space	iii) Max. no of I/O can be connected is less
iv) More decoding hardware is required	iv) Less decoding hardware is required
v) More flexibility	v) Less flexibility

* Basic Memory Read & Write Cycles -

▪ Pipelined Bus Cycles -

- Pipelining allows machine cycles to be overlapped. Main advantage is that it increases amount of time required for memory / IO device to respond.
- 80386DX implements pipelining by overlapping addressing of next bus cycle with data transfer of previous bus cycle

- Diagram -



▪ Non-Pipelined Read Cycle -

→ • For non-pipelined bus cycle, address phase is followed by data phase i.e T1 & T2

• 80386DX bus cycle requires only two bus states. for ex, 3 consecutive bus read cycles, each consists of two bus states

• Read operation starts at beginning of phase in T1 state, where 80386DX sends address bus & enables ($\overline{BE0}$ - $\overline{BE3}$)

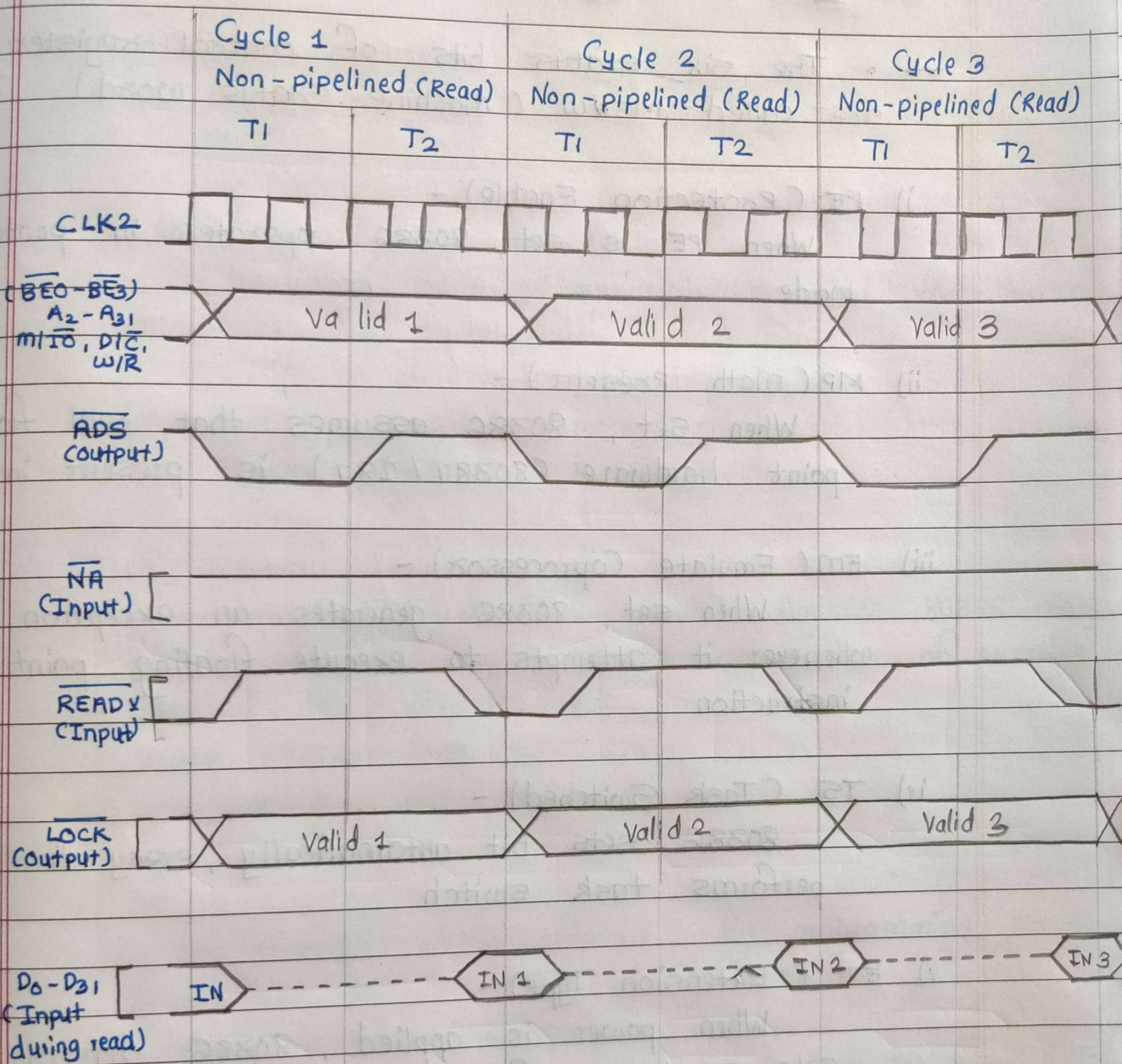
• It also activates $\overline{ADS}$ to indicate valid address is placed.

• M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$ signals are also activated

• At end of phase 2 of T_1, $\overline{ADC}$ is returned to inactive logic 1, whereas address bus, ($\overline{BE0}$ - $\overline{BE3}$) remain active through end

Diagram on Next page ☺

- Diagram –


Cycle 1
Non-pipelined (Read)
Cycle 2
Non-pipelined (Read)
Cycle 3
Non-pipelined (Read)
T1
T2
T1
T2
T1
T2
CLK2
(BE0-BE3)
A2-A31
M/IO, D/C, W/R
Valid 1
Valid 2
Valid 3
ADS (output)
NA (Input)
READY (Input)
LOCK (Output)
Valid 1
Valid 2
Valid 3
D0-D31 (Input during read)
IN
IN 1
IN 2
IN 3

* MSW in 80386 DX -

→ • The six status bits of control register are called MSW (Machine Status Word)

i) PE (Protection Enable) -
When PE is set, 80386 operates in protection mode

ii) MP (Math Present) -
When set, 80386 assumes that real floating point hardware (80387/287) is present in system

iii) EM (Emulate Coprocessor) -
When set, 80386 generates an exception whenever it attempts to execute floating point instruction

iv) TS (Task Switched) -
80386 sets bit automatically, every time it performs task switch

v) ET (Extension Type) -
When power is applied, 80386 detects & sets ET to 1, if numeric processor is 80387

vi) PG (paging) -
This bit enables/disables paging mechanism in mmu.

Date
Page

★ DEBUG registers -

→ • Debug registers allow 80386 to provide debugging feature. DR_0 to DR_7 registers are used

• $DR_0 - DR_3$:
The first four debug registers hold four linear addresses for breakpoint

• DR_4 and DR_5 :
These registers are undefined

• DR_6 :
It is also called Debug Status Register. 80386 sets status bits in this, to give information of probable causes for debug fault.
These status bits are:
B0, B1-3, BD, BS, BT.

• DR_7 :
It controls debug feature. By programming these bits, we can configure debug operation of four linear address breakpoints.
Each breakpoint is set by controlled by four fields
These are:
L0, G0, RW0, LEN0

* Test Registers -

→ • Among eight test registers ($TR_0 - TR_7$), only TR_6 & TR_7 are defined

• TR_6 :
This is TLB testing command register. By writing into this, it is possible to either initiate write directly in 80386's TLB or perform TLB lookups

• TR_7 -
This is Data testing register of TLB. When program is performing writes, entry to be stored is contained in this register.