May 2017

Chapter 2 : Applications Instruction Set
[Total Marks - 06]

Q. 2(c) Explain the following instructions, mention flags affected : (6 Marks)

(I) CWD (II) BT (III) LAHF

Ans. :

(I) CWD

Description :	Convert word to double word
Mnemonic :	CWD
Algorithm :	If MSB bit of AX = 1 then, DX = 655 35 (FFFF H) else DX = 0
Addressing mode :	Implied Addressing mode
Clock cycles :	2
Operation :	This instruction copies the sign bit of word in AX to all the bits of DX register. DX is sign extension of AX then. It must be done before signed word in AX can be divided by another signed word with IDIV instruction.
Flags :	No flags are affected
Example :	CWD
Real address mode exceptions :	None
Protected mode exceptions :	None

(II) BT

Description :	Bit test
Mnemonic :	BT

Algorithm :	CF = 0 if bit = 0 OR if CF = 1 if bit = 1
Addressing mode :	–
Clock cycles :	3-13
Operation :	This instruction tests the status the specified bit in the instruction. The status of that bit is copied to the carry flag.
Flags :	CF is set to the value of selected bit and OF, ZF, SF, AF and PF are undefined.
Example :	BT EAX, 05 ; This instruction copies the bit 5 of the EAX register to the carry flag.
Real address mode exceptions :	If the offset of the word operand is exceeding the segment limit i.e. 0FFFFH then exception INT 13 will be generated.
Protected mode exceptions :	In case if the operand cannot be used because of violation in access rights or violation in the segment limit then also exception 13 is generated.

(iii) LAHF

Description :	Copy lower byte of flag register to AH.
Mnemonic :	LAHF
Algorithm :	AH = flag register's lower byte
Addressing mode :	Implied Addressing mode
Clock cycles :	2
Operation :	AH ← Lower byte of flag register The lower byte of flag register (SF, ZF, AF, PF and CF) is copied to the AH register
Flags :	No flags are affected.
Example :	LAHF
Real mode exceptions :	None
Protected mode exceptions :	None

[Total Marks - 08]

Q. 1(a) What is the use of following instructions ? (i) Wait (ii) Lock (2 Marks)

Ans. :

(i) **Wait :** This instruction causes the processor to wait till an interrupt is recognized

(ii) **Lock :** Active the bus lock signal. Many multiprocessor systems contain several microprocessors. Each microprocessor has its own local buses and memory. The individual microprocessors are connected together by a shared system bus so that each can access system resources such as disk drives or memory. Each microprocessor takes control of the system bus.

Only when it needs to access some resource. Lock prefix allows a microprocessor to make sure that another processor does not take control of the system bus. While it is in the middle of a critical instruction which uses the system bus when an instruction with lock prefix executes the pentium will assert its bus lock signal output. This signal is connected to an external bus controller, which then prevents any other processor from taking over the system bus.

Q. 2(a) What is the use of Direction Flag ? (2 Marks)

Ans. :

Direction Flag (DF)

It is used to control the direction of string operations. If DF = 1, then the ESI and EDI registers are decremented by 1 and if the DF = 0 then ESI and EDI registers are incremented by 1 after each operation in the string instructions.

Q. 2(b) Draw and explain the system address and system segment registers. (4 Marks)

Ans. :

There are **four memory management registers.**

(i) **Task Register (TR) :** It points to the Task state descriptor (TSS).

(ii) **The Global Descriptor Table Register (GDTR) :** It points to the global descriptor table (GDT).

(iii) **The Interrupt Descriptor Table Register (IDTR) :** It points to the Interrupt Descriptor Table (IDT) .

(iv) **The Local Descriptor Table Register (LDTR) :** It points to the local Descriptor Table (LDT).

Fig. 1-Q. 2(b) shows the memory management registers.

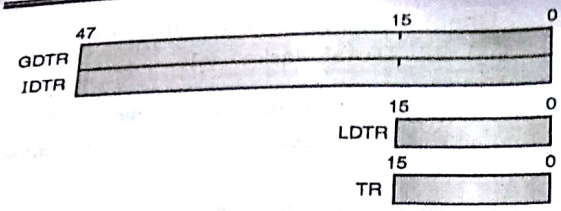


Fig. 1-Q. 2(b) : Memory management registers

The system address and segment registers are used to hold the addresses of their respective descriptors and their segments. The GDTR and IDTR are called system address registers while LDTR and TR are called system segment registers.

Chapter 4 : Memory Management [Total Marks - 10]

Q. 1(b) Explain segment address translation in detail. (4 Marks)

Ans. :

Segmentation provides both memory management as well as protection. Segmentation is a process of converting **logical address to a linear address.**

All information related to segments is stored in an segment descriptor (8 byte). The upper 13 bits (32-47) within a selector are used as an index to point one of the descriptor tables (GDT or LDT). The 13 bit index part is multiplied by 8, as each descriptor requires 8 bytes in the descriptor tables.

The descriptor in the descriptor table contains segment limit, base address and access rights byte. The two descriptor tables used are global and local descriptor table. GDT can be used by all programs to reference segments in the memory. LDT is set up in the system for individual task or closely related group of tasks. The TI bit decides which bit should be referred by selector if TI is 1, it refers to descriptor in current LDT otherwise it refers to descriptor in the GDT.

These tables have associated registers (GDTR, LDTR) that store a 32-bit linear base address and a 16 bit limit for each table. The segment limit can be examined by the LSL instruction. The segment limit of the segment specified by the source operand is loaded into the destination register. The instructions LGDT and LLDT are used to load base and limit of descriptor tables into their respective registers. And finally the linear address can be obtained by summing the base address from segment descriptor and the offset from selector.

The 32 bit base address from the segment is added to the 32 bit offset to create the linear address. If paging is not used then the 32 bit linear address is same as the 32 bit physical address. If paging is enabled then the paging hardware converts the linear address to physical address.

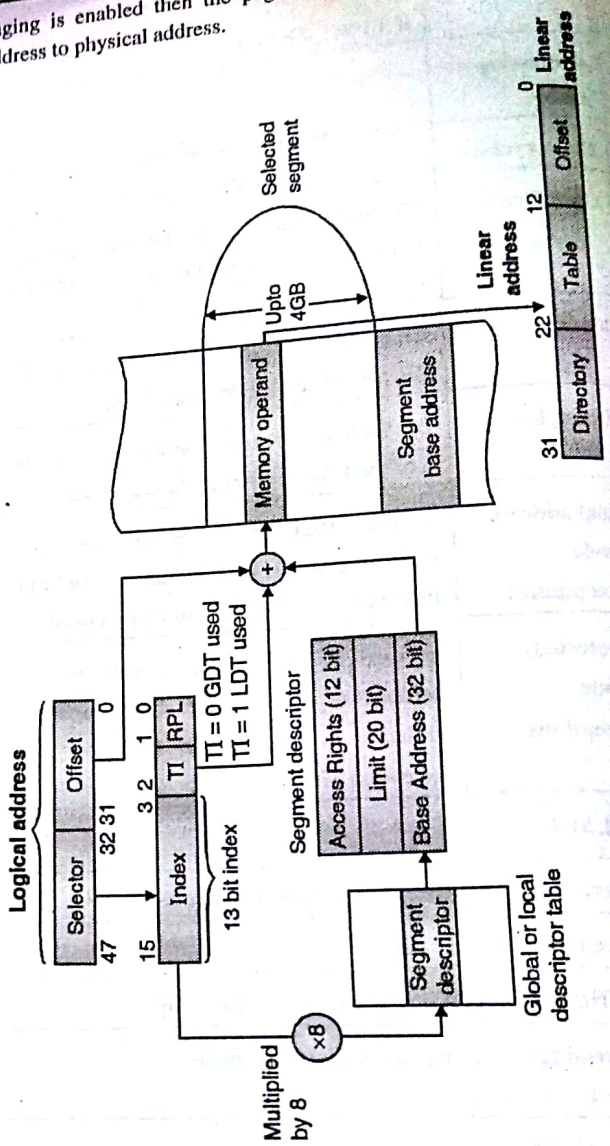


Fig. 1-Q. 1(b) : Segment Translation Process (Logical address to linear address)

The first two bits in the selector provide the privilege level of the program that requests access to a segment and in the segment descriptor DPL bits provide the privilege level of the segment. During the execution of program if an attempt is made to access the segment, then the memory management unit compares the privilege level of the selector with the privilege level of the segment descriptor. Access is allowed only if the selector has same or greater privilege level. If the selector has lower privilege level then an exception is generated and indicates privilege violation.

The data structures needed for segment translation are :

1. Segment selectors
2. Segment registers
3. Descriptors
4. Descriptors tables

Q. 1(c) Draw and explain segment descriptor.

Ans. : Segment descriptor **(6 Marks)**

In the 80386 memory segmentation it is not possible to use a 16-bit segment register to represent all the information related to segment. When multiple privilege levels and intertask protection are required a special structure called a segment descriptor is used.

A segment descriptor defines the memory segments location, access rights and length. Segments are areas of memory defined by a programmer and can be a code, data or stack segment. In 80386, segments need not be all the same size and paragraph aligned. And segments need not be exactly 64 KB long, but we can define them to be any length from 1 byte to 4 GB.

A segment with only 1 byte implies only valid offset 0. But a segment of 4 GB spans the entire address space : from 0 to FFFFFFFF H on the 80386. Segments are also assigned to have attributes viz. privilege level, segment type etc. For each segment of memory, one descriptor must be defined.

Size : The 80386 descriptors are of 8 bytes. These 8 bytes comprise the 32 bit segment base address, 20 bit segment limit and the access rights bits and another attribute bytes. The format of prototypical segment descriptor is shown in Fig. 1-Q.1(c).

Base address : The lower base address is specified in bits 15-0, 16 to 39 and the upper 8 bits are specified in bits 56 to 63. The processor combines the three parts of the base address of form a 32 bit base address.

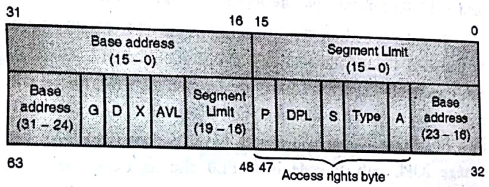


Fig. 1-Q.1(c) : 80386 descriptor structure (code, data, stack descriptors)

Limit : The 20 bit limit field combines the lower 16 bit of segments limit are specified in bits 0-15 and the remaining 4 bits are specified in 48-51. And so as, 2^{20} = 1 MB, maximum number of segments we can have is limited to 1MB.

The remaining bits 40-55 (Attribute bits) are described below :

Offset	Bit	Description
40	A (Accessed)	80386 sets this bit to 1 whenever a memory reference to a segment defined by this descriptor is made.
41-43	TYPE	This 3 bit field describes the type of segment 000 - Data segment, read only 001 - Data segment, write/read. 010 - Stack segment, read only. 011 - Stack segment, read/write. 100 - Code segment, execute only 101 - Code segment, execute/read 110 - Code segment, execute only, conforming 111 - Code segment, execute/read, conforming
44	S (System)	0 - Specifies system segment descriptor 1 - Specifies non system segment descriptor e.g. code, data or stack.
45-46	DPL (Descriptor Privilege level)	This bit indicates privilege level that is associated with memory space. 0 is most privileged level while 3 is the least.
47	P (Present)	If set to 0, indicates that the address range that is specified by this descriptor is temporarily not present.
52	AVL/U bit	Available for users. Programmers can use this bit if they wish to do so.
53	X (Reserved)	Bit reserved by Intel.
54	D (Default size)	0 - Operands within these segments are assumed to be 16 bits. 1 - Operands assumed to be 32 bits.
55	G (Granularity bit)	0 - Unit of limit field is 1 byte. 1 - Unit of limit field is of 4096 bytes.

The Granularity bits allow us to get a segment larger than 1 MB. If this bit is set, minimum size of a segment is 4 KB if limit field is 0.

The **two types of descriptors** that the 80386 uses are :

(i) System segment descriptors

(ii) Non-system segment descriptors

A system descriptor defines :

(i) 32 bit base address

(ii) 20 bit segment limit

(iii) Granularity of limit field
(iv) Whether segment can be accessed
(v) Type of segment
(vi) 2 bits privilege level of the segment
(vii) Whether segment is physically present
(viii) Size of operands
(ix) AVL/U bit
(x) Default size

A segment descriptor's function is to

(i) Describe a segment
(ii) Find the base address of the segment
(iii) Find the type of segment
(iv) Find the privilege level of the segment
(v) Find the size of the segment

The programmer creates the segment descriptor. Every segment must have a segment descriptor. It is used for intertask protection.

Chapter 6 : Multitasking [Total Marks - 02]

Q. 3(a) List the registers and data structures that are used in multitasking. (2 Marks)

Ans. :

The 80386 processor supports **task management data structures** that are used for multitasking in the protected mode operation. These task management data structures are :

(i) Task State Segment (TSS)
(ii) Task Register (TR)
(iii) Task State Segment Descriptor
(iv) Task Gate
(v) Task gate descriptor

The 80386 microprocessor can switch from one task to another with the help of these task management data structures.

When the processor switches from one task to another the conditions of the current task are saved so that they can be retrieved after sometime.

Chapter 7 : Input-Output [Total Marks - 06]

Q. 3(b) Differentiate between memory mapped I/O and I/O mapped I/O. (4 Marks)

Ans. :

Sr. No.	Memory mapped I/O	I/O mapped I/O
1)	A memory address is allotted to an I/O device.	An I/O address is given to an I/O device.
2)	Any memory related instruction will be able to access this I/O device. eg : MOV BX, [3500 H]	Only IN and OUT instructions can access such devices.
3)	$\overline{MRDC}$ or $\overline{MRD}$ and $\overline{MWR}$ or $\overline{MWRC}$ are given to $\overline{RD}$ and $\overline{WR}$ of I/O device	$\overline{IORC}$ / $\overline{IORD}$ and $\overline{IOWC}$ / $\overline{IOWR}$ are given to $\overline{RD}$ and $\overline{WR}$ of I/O device
4)	The data from I/O device can also be given to ALU and result given back to I/O device using single instruction eg : ADD [3500 H], CX	ALU operations are not possible directly on I/O data. They are to be first brought into accumulator.

Q. 4(a) Write the two mechanisms that provide protection for I/O functions. (2 Marks)

Ans. :

I/O protection can be achieved using (i) an I/O privilege level (IOPL in EFLAGS) and using (ii) an I/O permission bitmap.

I/O Privilege Level

The I/O sensitive instructions are IN, INS, OUT, OUTS, CLI and STI. These instructions are sensitive to IOPL field.

In order to execute these instructions the CPL ≤ IOPL should be satisfied. i.e. for allowing the I/O operation the CPL should be less than the IOPL. If the CPL is greater than the IOPL then it will result in a general protection violation exception. Each task has its own EFLAG's register and so can have its own IOPL. We can change IOPL only if code is at PL0 else an exception 13 is generated.

I/O Permission Bitmap

This method for the protection of I/O ports makes the use of permission bits stored in the I/O bitmap. The I/O permission bitmap is present on the top portion of the Task state segment (TSS). The size of the map and its location in the TSS is variable. If the I/O permission check fails then the processor consults the I/O permission bitmap. If the I/O permission bit that is related to an I/O address is 0 then the I/O instruction referring this address is allowed. And if this bit is 1 the I/O instruction is not allowed.

If an instruction referring this address is executed a general protection fault is generated.

Chapter 8 : Exceptions and Interrupts [Total Marks - 16]

Q. 3(c) Explain what happens when an interrupt calls a procedure as an interrupt handler. (6 Marks)

Ans. :

Step I : Whenever an INT instruction is sensed, the 80386 microprocessor will first push the flags on the Stack .The flags are pushed so that they will provide us the internal status of the 80386 processor when it was interrupted.

Step II : The processor will clear the trace flag and the interrupt enable flag.

Step III : Then the contents of CS and IP are pushed onto the Stack.

Step IV : The interrupt number is multiplied by 4 to get the address within the interrupt vector table that contains the interrupt service routine address. All the ISR addresses are of 4 bytes. These addresses are in the Standard CS : IP format.

Step V : Once the vector address is computed, the 80386 microprocessor reads the ISR address from the table. It then places the ISR address onto the CS : IP to begin the program execution from this address.

Step VI : After completion of the ISR routine the program execution has to be continued from the place where the program was interrupted. So, the values of the CS : IP and flags which specify the status of the 80386 microprocessor have to be restored. i.e. this information is to be popped from the Stack using IRET instruction.

Q. 4(b) What is IDT and how to locate IDT ? (4 Marks)

Ans. :

IDT contains group of descriptors that define interrupts or exception handling routines.

Function :

Its function is to hold the segment descriptors that define the interrupt or exception handling routines.

For 80386 to work in protected mode, at least one IDT needs to be defined. The interrupt trap or task gate descriptors can be present in the IDT. It can reside anywhere in the physical memory. Its address is stored in the IDTR. In the IDT there are 256 gate descriptors.

It is same as the Interrupt Vector Table (IVT) in 8086 system. And so we need mechanism for processor to be able to locate them and register IDTR is used. Each of the for it registers store the 32-bit linear address of the base of the descriptor table respectively.

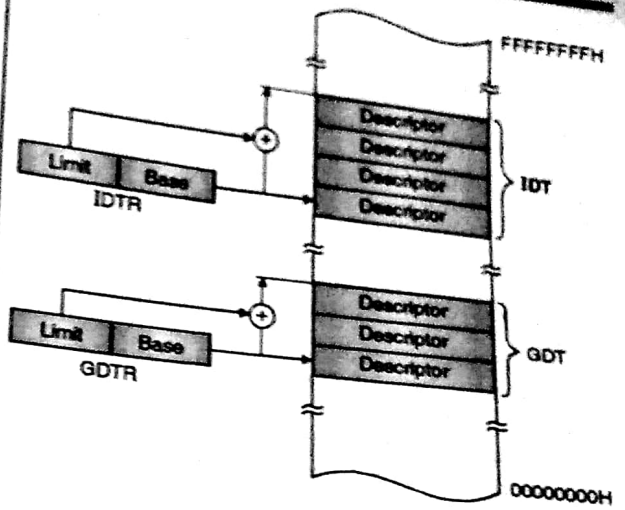


Fig. 1-Q. 4(b) : Descriptor tables and descriptors

The interrupt descriptor table :

(i) Holds the segments descriptors (interrupt descriptors, trap descriptors or task gate descriptors) for defining the interrupt or exception handling routines.

(ii) It maintains the interrupt service routines.

(iii) It contains 256 gate descriptors, one for each interrupt vector.

(iv) The user programs can never select a descriptor in the IDT like the GDT or the LDT.

(v) The descriptors used in IDT are :

(a) Trap gate descriptor

(b) Interrupt gate descriptor

(c) Task gate descriptor

Q. 4(c) Explain the different exception conditions Faults, Traps and Aborts. (6 Marks)

Ans. :

Exceptions are generated by internal events while interrupts are generated by external events.

In protected mode of the 80386 processor, more internal conditions can initiate an internal interrupt or exception. Exceptions are divided into three groups depending on the way they are reported, and whether or not restart of the instruction causing an exception is supported.

These groups are as follows :

(i) Faults :

These are exceptions that are detected and serviced before the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present in physical memory. The operating system would fetch the page or segment from disk,

and then the 80386 would restart the instruction. E.g. INT0, INT5, INT6. The 80386 will execute the instruction that created the fault after completion of the exception handling software.

(ii) Traps :

These are exceptions that are reported immediately after the execution of the instruction that caused the problem. User defined interrupts are the examples of this group of interrupts. E.g. INT2, INT3, INT4. The instruction that caused the trap cannot be restarted.

(iii) Aborts :

These are exceptions, which do not permit the precise location of the instruction causing an exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in the system table. The program with aborts must be terminated. E.g. INT8, INT9.

Chapter 9 : Initialization of 80386DX and Debugging [Total Marks - 18]

Q. 5(a) Write short note on "Task Switch Breakpoint". (3 Marks)

Ans. :

Task Switch Breakpoint

Task switch breakpoint is a debug exception. It is observed to happen after 80386 switches to a task and if the bit of the new TSS is set. After the control is transferred to the new task the exception occurs before the execution of the first instruction of the new task. By observing the BT of DR6 debug register the exception handler can detect the condition. The T bit of new TSS should not be set if the debug exception handler is a task, otherwise it may cause the 80386DX processor to enter into an infinite loop.

Q. 5(c) Explain various debugging features of 80386. (6 Marks)

Ans. :

The 80386 architecture features that support debugging are :

1. Debug control register.
2. Debug status register.
3. T (Trap bit) of TSS
4. RF (Resume flag bit) of E Flags register
5. Reserved debug interrupt vector.
6. Four debug address registers.
7. Allows task switches monitoring.
8. Single step (TF) flag.
9. Allows the users to select the debug conditions related to the 80386 debug registers.
10. It helps the debugger to determine the condition that created the debug condition.
11. It allows the user to monitor the program flow by indicating whether the 80386 should generate the debug exception along with the execution of the instruction.
12. When an event that needs to be executed by the debugger, the processor invokes the task or procedure automatically.
13. For the same condition that caused a debug instruction, it allows the processor to restart the instruction without causing another debug exception.
14. It allows the user to provide four addresses that will be monitored by the 80386 processor.

Q. 6(a) Write short note on "General Detect Fault". (3 Marks)

Ans. : General Detect fault :

General Detect fault is a debug except that occurs whenever an attempt is made to use the debug registers while the IC-386 emulator is also using them.

General Detect fault is a protection mechanism that assures that ICE-386 (in-circuit emulator 80386) has a complete control over the debug registers whenever needed. If an ICE-386 is using the debug registers and if the software debugger wishes to use the debug registers, it cannot run. By observing the BD of DR6 debug register, the exception handler can detect this condition.

Q. 6(c) Explain, how test registers are used in testing TLB ? (6 Marks)

Ans. :

Test Registers

Two test registers (TR6-TR7) are currently defined. They are used to test the Translation Look-Aside Buffer (TLB). The TLB is used with the paging unit within the 80386. The TLB is responsible for holding the most commonly used page table address translations. This decreases the number of memory reads that are needed for seeing the looking up page translation address, in the page translation table.

From the page table, the TLB generally holds the most common 32 entries. The TLB is tested with the help of the test registers TR6 and TR7. TR6 holds the tag field (linear address) of the TLB, and TR7 holds the physical address of the TLB.

The registers TR6 and TR7 can be accessed by the different MOV instruction variants . The MOV instructions can be used in the real-address as well as the protected mode of 80386. In protected mode the test registers are at privilege level 0. MOV instructions to access the test registers need to be executed at privilege level 0 otherwise an exception is generated.

Fig. 1-Q.6(c) shows the bit pattern of the TR6 and TR7.

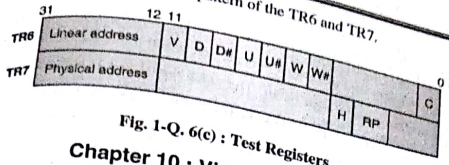


Fig. 1-Q. 6(c) : Test Registers

Chapter 10 : Virtual 8086 Mode [Total Marks - 08]

Q. 5(b) Write short note on "Protection within a V86 task".

Ans. : **(4 Marks)**

When the 8086 programs are executed they do not use the descriptors and the different protection mechanisms that they provide. For protecting the system software that executes a V86 task from the 8086 program one of the following methods can be used.

1. The U/$\overline{S}$ bit of the page table entry and page directory entry are used to protect the virtual machine monitor in V86 task space. The CPL is 3, when the 80386 DX processor operates in V86 mode. This limits the ability of processor to execute some instructions. An attempt to execute a privileged instruction in VM86 mode will cause an exception 13. Thus, an 8086 program has user privileges. If the virtual machine monitor have pages with supervisory privilege levels then the 8086 program will not be able to access them.
2. Virtual mode addressing supports 8086 addressing modes. There are no descriptor tables, segment selectors, RPL fields. Hence reserve first megabyte + 64 kB of every tasks linear space for an 8086 program as the 8086 task cannot generate addresses beyond that range.

Q. 6(b) Which bit of EFLAGs indicates V86 mode ? Explain, how hardware and software cooperate with each other to emulate V86 mode ?

Ans. : **(4 Marks)**

The Virtual 8086 mode is the last of the three main operating modes of the 80386 microprocessor. The virtual 8086 mode allows the execution of 8086 applications considering the mechanism of 80386 protection when the VM bit of Eflag register is set, the 80386 will enter the virtual mode.

The 80386 processor checks or two conditions when the VM bit is set. They are.

(i) Whether to execute 8086 programs while loading the segment registers.

(ii) To find the instructions sensitive to IOPL.

The emulation of 8086 processor is task oriented in virtual mode. The emulation of 8086 is done on a task by task basis in the virtual mode. The program or programs that are defined as a part of the task can freely transfer control into and out of various segments. They can also make intersegment procedure calls. A VM86 program is incapable of executing a FAR CALL instruction to a 32 bit 80386 procedure and has procedure return to it.

The paging hardware allows the execution of several virtual mode tasks and provides protection. Multiple 8086 programs can be simultaneously executed on **virtual machines**. A virtual machine consists of hardware and software that is required to perform a particular task. Fig. 1-Q. 6(b) shows the concept of virtual machines running on 80386.

Each virtual machine has its own 1 MB addressing space and set of processor registers. The 1 MB addressing space can be located anywhere in the memory through use of 80386's paging mechanism. The 80386 processor treats the segment registers exactly similar to that in real mode. Hence the address range for virtual 8086 mode task is 1 Mbyte.

The segment and offset registers in combination give the linear address instead of physical address with the help of the page translation mechanism. Thus, the physical address can be anywhere in the 4 GB memory that is addressable by 80386. In VM86 mode, 80386 provides mechanism to selectively trap and manage input / output and interrupt activity. Using software it is possible to determine the IO privilege level that controls the input / output transfer. The I/O port permission map can be used to control the access to I/O ports.

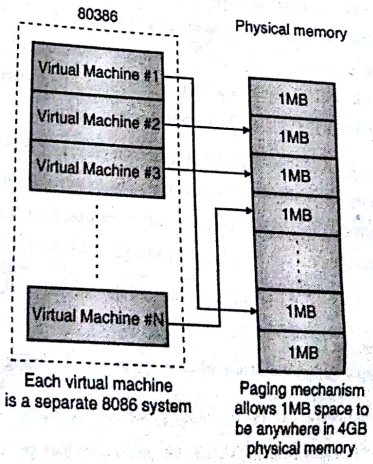


Fig. 1-Q. 6(b) : Concept of virtual machines running on 80386

Chapter 11 : 80386 DX Signals
[Total Marks - 20]

Q. 7(a) Explain following signals (3 Marks)
(I) ADS# (II) READY# (III) NA#

Ans. :

(I) ADS#

A low on this pin indicates that the 80386 has issued a valid memory/IO address. This valid address is present on the address bus.

(II) READY#

It is used to synchronize the microprocessor with slower peripherals. It controls the number of wait states inserted into the timing to lengthen memory access. A low on this indicates to the processor that it is ready for next data transfer.

(III) NA#

Next address causes the 80386 to output the address of the next instruction or data in the current bus cycle. This pin is used for pipelining the address.

Q. 7(b) Write note on CLK2 and internal processor clock. (4 Marks)

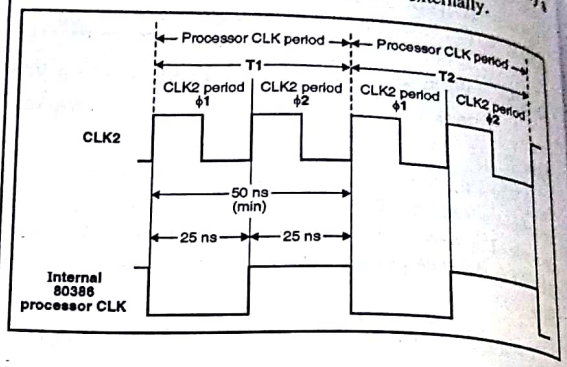
Ans. :

A bus cycle is the activity performed by the microprocessor in order to access data from memory or I/O devices. A bus state is defined as the shortest time of the activity of the processor bus. It is one processor clock period in duration. The internal processor clock (PCLK) signal is at half the frequency of the external clock input signal. Fig. 1-Q. 7(b) shows this relationship.

CLK2 input of the 80386 processor provides the fundamental timing. It is divided by two internally to generate the internal processor clock that is used for instruction execution. The internal clock comprises of two phases $\phi 1$ and $\phi 2$.

Each CLK2 period is a phase of the internal clock, e.g. In a 20 MHz 80386DX system, CLK2 frequency equals 40 MHz. Each clock cycle has a duration of 25 ns $\left(\frac{1}{40\text{ MHz}}\right)$. In Fig. 1-Q. 7(b) the two phases ($\phi 1$ and $\phi 2$) of the processor cycle contribute as one processor clock period. Hence, the processor clock period is 50 ns (minimum).

As shown in Fig. 1-Q. 7(b) each bus cycle has two states T1 and T2. Each state has two phases or two clock cycles. In first state T1 address and bus status pins are active and in the second state T2 the data transfer will occur. The clock signal that is applied to the CLK2 input of 80386DX is twice the frequency rating of the microprocessor. Therefore, CLK2 of an 80386DX-16 is driven by a 32 MHz signal. This signal must be generated externally.



p(6.1)**Fig. 1-Q. 7(b) : Processor clock periods (bus states)**

Q. 8(a) Explain following signals BE0# through BE3#. (3 Marks)

Ans. :

BE0# through BE3# signals

These signals function is to select the access of a byte, word or double word of data. These signals are generated internally by the microprocessor from the address bits A_1 and A_0.

Q. 8(b) Explain following signals (4 Marks)
(i) $\overline{\text{PEREQ}}$ (ii) BUSY# (iii) ERROR#

Ans. :

(i) $\overline{\text{PEREQ}}$

The data cannot be transferred over the data bus with the help of a co-processor. Hence the $\overline{\text{PEREQ}}$ signal is activated whenever the co-processor wishes to read or write data from the memory.

(ii) BUSY#

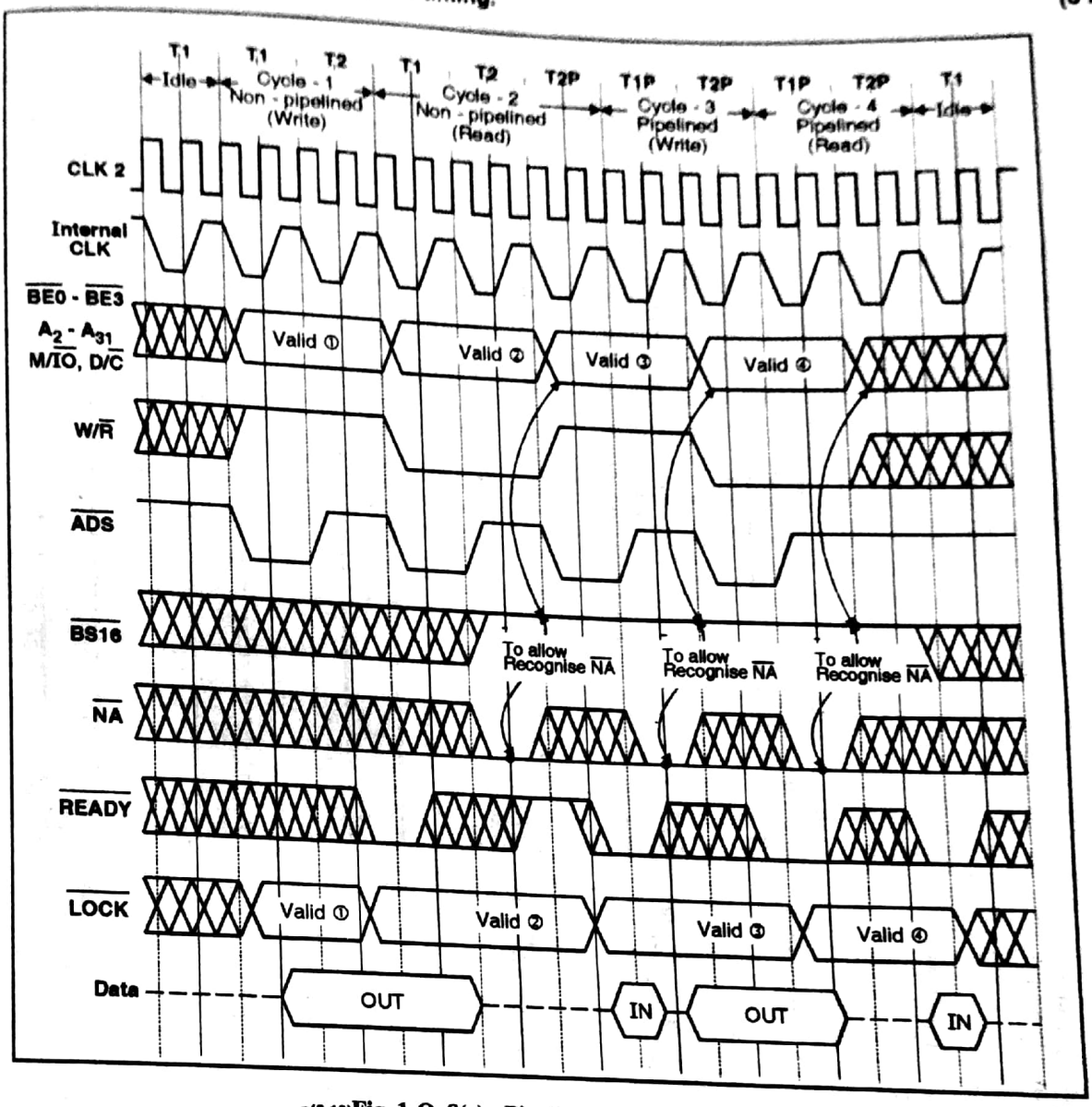
Busy is an input used by WAIT or FWAIT instruction of a co-processor. It indicates 80386 that the co-processor is computing some numerical calculation and is busy.

(iii) ERROR#

It indicates to the microprocessor 80386DX that an error is detected by the co-processor.

Q. 8(c) **Draw read cycle with pipelined address timing.** **(6 Marks)**

Ans. :



p(6.10)**Fig. 1-Q. 8(c) : Pipelined read and write cycles**

Chapter 12 : 80387 Numeric Co-Processor [Total marks-06]

Q. 7(c) **Which data types are supported by 80387 ?** **(6 Marks)**

Ans. : The 80387 can operate on memory operands of seven different data types :

Word Integer	Short Integer	Long Integer	Packed BCD
Short Real	Long Real	Temporary Real	

The number of bytes required, format and approximate range of all data types are shown in Fig. 1-Q. 7(c).

The LSB of the number will be stored at the lowest address.

The signed integer numbers or the BCD numbers are called as **fixed point numbers.** This is because these numbers do not carry any information regarding the decimal point of the number.

The decimal point is assumed to be to the right of the LBS, inorder that all the numbers are expressed as whole numbers with no fractional part.

A record of the decimal point has to be kept if the numbers are to be expressed as fractional numbers. This format is capable of representing a number in the integer as well as fractional part. The numbers that are represented in such data formats are called as **real numbers** or floating point numbers.

A **real format** is divided into three fields :

i) Sign ii) Exponent iii) Mantissa

Real number = sign $2^{\text{exponent}} \times$ mantissa

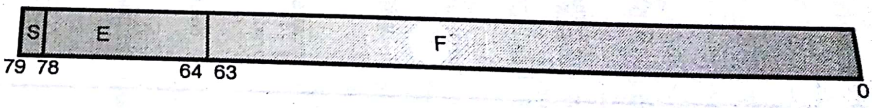
Data Type	Bytes	Approximate Range (decimal)	Format
Word Integer	2	– 32,768 to 32,767	
Short Integer	4	$\approx -2 \times 10^{9}$ to 2×10^{9}	
Long Integer	8	$\approx -9 \times 10^{18}$ to 9×10^{18}	
Packed BCD	10	$-10^{18}+1$ to $9 \times 10^{18}-1$	
Short Real	4	$\approx \pm 1 \times 10^{-38}+1$ to 3×10^{38}	
Long Real	8	$\approx \pm 1 \times 10^{-308}$ to $\pm 10^{308}$	
Temporary Real	10	$\approx \pm 10 \times 10^{-4932}$ to $\pm 10^{4923}$	

Fig. 1-Q. 7(c) : Data format for NDP

The three real data types are :

i) Short real (32 bits)
ii) Long real (64 bits)
iii) Temporary real 80 bits.

Normalization

In order to convert any numbers to a real number, we have to shift the decimal point to the right of the most significant non zero digit. Then we have to multiply the number with base N where N :- number of digits shifted to adjust the decimal point. The process of shifting the decimal point to the right of the most significant non zero digit is called as Normalization.

Integer :

Under this, we have 16 bits (word integer), 32 bits (short integer) and 64 bits (long integer). Negative integers are coded in 2's complement form.

Packed BCD

In this, decimal number is stored in 10 bytes. The entire most significant byte is dedicated to sign. The most significant bit of this byte indicates whether a decimal number is positive (0) or negative (1). The remaining 9 bytes represent the magnitude with two BCD digits packed into each byte. Therefore, valid range of values in packed BCD format is $10^{18} + 1$ to $10^{18} - 1$.

Real data types (floating point types)

The real data types, also called floating point types, can represent operands which may vary from extremely small to extremely large values, and retain a constant number of significant digits during calculations. Three types are available :

Short Real (32 bits) [single precision]

Long Real (64 bits) [double precision]

Temporary Real (80 bits) [extended precision].

Dec. 2017

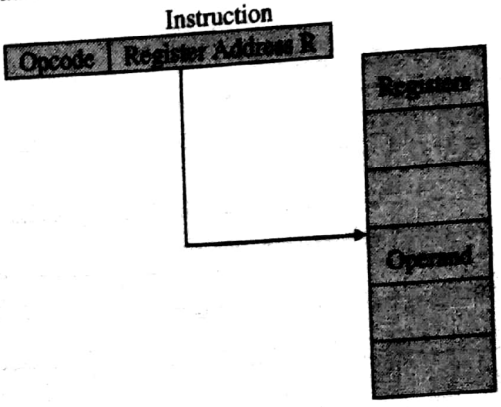
Chapter 1 : 80386DX Basic Programming Model [Total Marks - 02]

Q. 1(a) Explain immediate and register addressing mode with an examples. (2 Marks)

Ans. : Register Addressing Mode

In this mode of addressing, data is in the registers and the instruction specifies the particular registers as shown in Fig. 1-Q. 1(a).

This addressing mode is normally preferred because the instructions are compact and fastest executing of all instruction forms. The reason why it is fastest executing is just because, all the registers reside on chip, therefore data transfer is within the chip and external bus is not at all required.



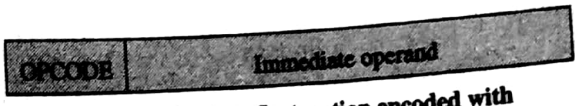
p(7.4)**Fig. 1-Q. 1(a) : Register addressing**

Registers may be used as source operands, destination operands or both. The registers may be 8 bit, 16 bit or 32 bit.

eg : MOV EAX, EDX. This instruction will copy the contents of EDX register to the AX register.

Immediate Addressing Mode

Immediate operand is nothing but **constant data** contained in an instruction. If the source operand is a part of the instruction instead of register or memory, it is referred as immediate addressing mode. Fig. 2-Q. 1(a) shows the format of instruction encoded with immediate operand.



p(7.5)**Fig. 2-Q. 1(a) : Instruction encoded with an immediate operand**

The immediate data may be 8 bits, 16 bits or 32 bits in length. Immediate operand can be accessed quickly because they are available directly in the instruction queue like a register and hence there is no need of external bus and bus cycles to obtain the data. The immediate operands only serve as source operands. They have constant values.

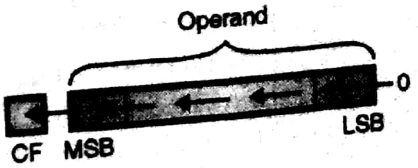
eg : MOV ECX, 20305060H. This instruction copies 20305060H in the EAX register.

Chapter 2 : Applications Instruction Set [Total Marks - 08]

Q. 1(b) Explain with example SHL and ROL instructions. (4 Marks)

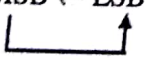
Ans. :

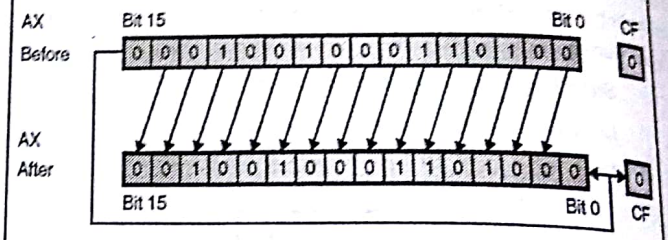
SHL/SAL - Shift Logical/ Arithmetic Left Byte or Word or Double Word

Description :	Shift left
Mnemonic :	SAL/SHL destination, count.
Algorithm :	Shift all bits left, the bit that goes off is set to CF. Zero bit is inserted in the right most position.
Addressing mode :	Immediate addressing mode
Clock cycles :	3-7
Operation :	CF ← MSB ← LSB ← 0  p(7.12)**Fig. 1-Q. 1(b)**

	SAL/SHL are two mnemonics for the same instruction. This instruction shifts each bit of the specifies destination, some number of bit positions to the left. As left bit is shifted out of the LSB position, 0 is put in the LSB position. The MSB will be shifted into CF. The count can be any immediate number. In case of multiple bit shifts, CF will contain the bit most recently shifted in from the MSB. Bits shifted into CF previously will be lost. This instruction can be used to multiply an unsigned binary number by a power of 2. The destination can be a memory location or register. The count is specified in the CX register. Negative shifts are illegal.
Flags :	OF, CF, AF, PF, SF are affected.
Example :	SHL AX, 01 ; This instruction shift AX by 1 bit to the left.
Real address mode exceptions :	If the offset of the word operand is 0FFFFH then exception INT 13 will be generated.
Protected mode exceptions :	In case if a segment register is loaded or if the segment register has an incorrect effective address or if the destination is a non-writeable segment then in such conditions, a general protection exception will be generated. In case if the address that the stack segment has, is an illegal address then the exception that is generated is a stack fault exception.

ROL - Rotate Left

Description :	Rotate left
Mnemonic :	ROL destination, source
Algorithm :	Shift all bits left, the bit goes off is set to CF and the same bit is inserted to the right most position.
Addressing mode :	–
Clock cycles :	3-10
Operation :	$CF \leftarrow MSB \leftarrow LSB$  This instruction rotates all the bits in a specified word or byte to the left, by some bit positions.
	The data bit rotated out of MSB is circled back into the LSB. The data bit rotated out of MSB is also copied to CF. CL is default register used for rotate instruction when count is greater than 1. The difference between the ROL and SHL instruction is that the bits that get rotated out of the LSB position, get rotated back into the MSB. Thus, the data inside the destination operand is never lost it is circulated within itself.
Flags :	Only CF and OF are affected.
Example :	ROL AX, 01 H ; This instruction rotates the contents of AX register by 1 bit to left.



p(7.16)**Fig. 2-Q. 1(b)**

Real address mode exceptions :	If the offset of the word operand is 0FFFFH then exception INT 13 will be generated.
Protected mode exceptions :	In case if a segment register is loaded or if the segment register has an incorrect effective address or if the destination is a non-writeable segment then in such conditions, a general protection exception will be generated. In case if the address that the stack segment has, is an illegal address then the exception that is generated is a stack fault exception.

Q. 2(c)(ii) Explain the following instruction, mention flags affected : CLD **(2 Marks)**

Ans. :

CLD - Clear Direction Flag

Description :	Clear direction flag, increment SI and DI
Mnemonic :	CLD
Algorithm :	DF = 0
Addressing Mode :	Implied addressing mode
Clock cycles :	2

Operation :	It resets (DF) direction flag to zero. If the direction flag is reset, SI and / or DI will be automatically incremented during string operations. When one of the string instructions like MOVS, CMPS or SCAS executes it automatically increments the SI and / or DI registers.
Flags :	Except direction flag (DF) no other flags are affected.
Example :	CLD
Real address mode exceptions :	None
Protected mode exceptions :	None

Q. 2(c)(iii) Explain the following instruction, mention flags affected : MOVS **(2 Marks)**

Ans. :

MOVS : move string byte or string word or string double words

Description :	Move string byte or string word or string double words
Mnemonic :	MOVS destination _string, Source_string
	MOVSB → move string bytes MOVSW → move string words MOVSD → move string double words.

Algorithm :

In case of byte :	In case of word :	In case of double word:
ES : [DI] = DS : [SI]	ES :[DI] = DS : [SI]	ES:[EDI] :DS:[ESI]
If DF = 0 then	If DF = 0 then,	If DF = 0 then
* SI = SI + 1	* SI = SI + 2	* ESI = ESI + 4
* DI = DI + 1	* DI = DI + 2	* EDI = EDI + 4
else	else	else
DF = 1	DF = 1	DF = 1
* SI = SI – 1	* SI = SI – 2	* ESI = ESI – 4
* DI = DI – 1	* DI = DI – 2	* EDI = EDI – 4

Addressing mode :	String addressing mode
Clock cycles :	7
Operation :	This instruction copies a byte or a word or double word from a location in the data segment to a location in the extra segment. The offset of the source byte/word or double word in the DS must be SI (ESI) register. The offset of the destination in ES must be in DI (EDI) register.

	After byte or word or double word are moved, SI and DI are automatically adjusted to point to the next source and next destination.
Flags :	No flags are affected.
Example :	LEA SI, SOURCE_STRING ; SI points to start of source string. LEA DI, DEST_STRING ; DI points to start of CLD ; Clear direction flag to use incrementing MOVSB ; MOV (DI) ← (SI), byte transfer
Real address mode exceptions :	If the offset of the word operand is 0FFFFH then exception INT 13 will be generated.
Protected mode exceptions :	In case if a segment register is loaded or if the segment register has an incorrect effective address or if the destination is a non-writeable segment then in such conditions, a general protection exception will be generated. In case if the address that the stack segment has, is an illegal address then the exception that is generated is a stack fault exception.

Chapter 3 : Systems Architecture
[Total Marks - 10]

Q. 2(c)(i) Explain the following instruction, mention flags affected : LIDT. **(2 Marks)**

Ans. : LIDT - Load Interrupt Descriptor Table

Description :	Load value in interrupt descriptor table
Mnemonic :	LIDT source
Algorithm :	–
Addressing mode :	–
Clock cycles :	11
Operation :	This instruction loads the values in the source to the interrupt descriptor table register (IDTR). The source specifies a 6-byte memory location that contains the base address (a linear address) and the limit (size of table in bytes) of the interrupt descriptor table (IDT). The LIDT is the only instruction that directly load a linear address i.e. it not a segment-relative address and a limit in protected mode.

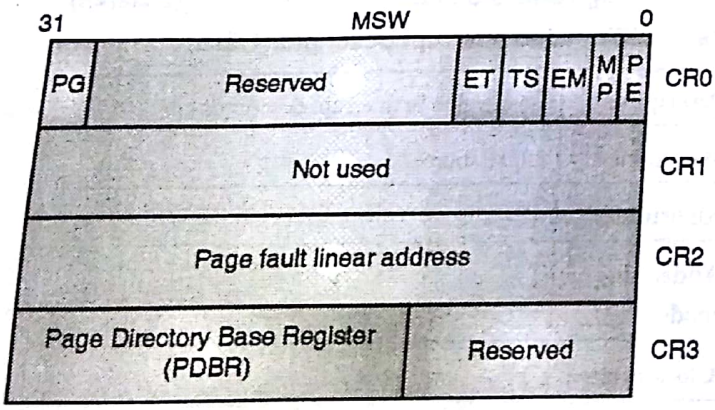
	This instruction is commonly executed in real-address mode to allow processor initialization prior to switching to protected mode.
Flags :	No flags are affected
Example :	LIDT ARR1.
Real address mode exceptions :	If the offset of the word operand is 0FFFFH then exception INT 13 will be generated.
Protected mode exceptions :	A stack fault exception is generated if the stack segment comprises an illegal address. A general protected mode exception is generated if the segments DS, CS or ES comprise an illegal memory effective operand address. If the destination is a non-writeable segment, a general protection exception is generated. If the CPL ≠ 0 then a general protection exception is generated.

Q. 1(c) Explain in detail the control registers of 80386. (6 Marks)

Ans. : Control registers of 80386

Fig. 1-Q. 1(c) shows the control register structure of the 80386. There are four control registers : CR0, CR1, CR2 and CR3.

These registers define the machine status which affects all the tasks in the systems.



p(1.23) **Fig. 1-Q. 1(c) : Control registers**

1. Control register CR0

It contains the six status bits. It gives us the **Machine Status Word (MSW).**

The six control bits are :

(i) *PG (Paging)*

It enables or disables the paging mechanism. If this bit is set then the linear address is converted to physical address.

(ii) ET (Extension Type)

This bit informs the 80386DX whether the numeric processor is an 80287 or 80387.

If ET = 0, it selects the 80287 coprocessor.

If ET = 1 it selects the 80387 processor. This is because, the 80387 uses a slightly different protocol.

(iii) TS (Task switched)

The processor sets this bit automatically each time it performs a task switch. The user has to clear this bit, the processor will never clear this bit.

(iv) EM (Emulate coprocessor)

This bit when set indicates an exception 11 (device not available) whenever a floating point instruction is fetched. We often use exception handler to emulate with software, the function of the coprocessor.

(v) MP (Math Present)

This bit is set to indicate that the arithmetic coprocessor is present in the system.

(vi) PE (Protection Enable)

It is used to select the protected mode of operation for the 80386. This bit may be cleared when the processor wants to reenter the real mode.

2. Control register CR1

It is not used in the 80386DX, but is reserved for other processors.

3. Control register CR2

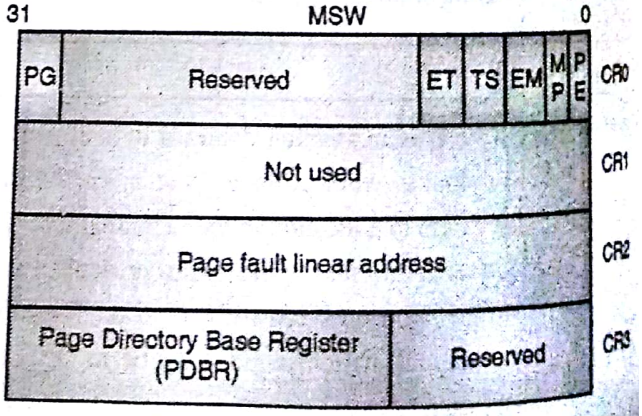
It is a read only register. It holds the linear page address of the last page accessed before a page fault interrupt. The page fault routine helps the programmer to find cause of the fault.

4. Control register CR3

It holds the base address of the page directory. It is also called as the **Page Directory Base Register (PDBR).**

Q. 2(a) Explain MSW. (2 Marks)

Ans. : MSW



p(1.23) **Fig. 1-Q. 2(a) : Control registers**

1. **Control register CR0**

It contains the six status bits. It gives us the **Machine Status Word (MSW)**. The six control bits are :

(i) PG (Paging)

It enables or disables the paging mechanism. If this bit is set then the linear address is converted to physical address.

(ii) ET (Extension Type)

This bit informs the 80386DX whether the numeric processor is an 80287 or 80387.

If ET = 0, it selects the 80287 coprocessor.

If ET = 1 it selects the 80387 processor.

This is because, the 80387 uses a slightly different protocol.

(iii) TS (Task switched)

The processor sets this bit automatically each time it performs a task switch. The user has to clear this bit, the processor will never clear this bit.

(iv) EM (Emulate coprocessor)

This bit when set indicates an exception 11 (device not available) whenever a floating point instruction is fetched. We often use exception handler to emulate with software, the function of the coprocessor.

(v) MP (Math Present)

This bit is set to indicate that the arithmetic coprocessor is present in the system.

(vi) PE (Protection Enable)

It is used to select the protected mode of operation for the 80386. This bit may be cleared when the processor wants to reenter the real mode.

Q. 3(a) What is CPL and RPL ? [Total Marks - 06]

Ans. : **(2 Marks)**

Requestor Privilege Level (RPL)

It is a two bit field.

(i) It is used to request the access the segment if requested privilege level is greater than or equal to the privilege level of the access rights byte.

(ii) They are used while protection checks to indicate whether access to a segment is allowed or not allowed.

These bits represent the four requestor privilege levels PL0 to PL3. PL0 is having the highest privilege level, then PL1, PL2 and PL3 has the lowest privilege level.

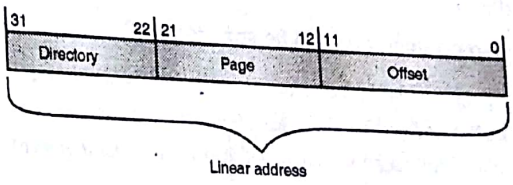
Current Privilege Level (CPL)

It is defined as the privilege level at which at task is currently being executed. The CPL of a task is stored in the processor. This privilege level is not accessible to the programmer.

Q. 2(b) Explain paging mechanism. **(4 Marks)**

Ans. :

The 80386 processor uses two levels of tables to translate the linear/ virtual address to physical address. The format of a linear address (in case of paging) is as shown in Fig. 1-Q. 2(b).



p(3.21)**Fig. 1-Q. 2(b) : Linear address format**

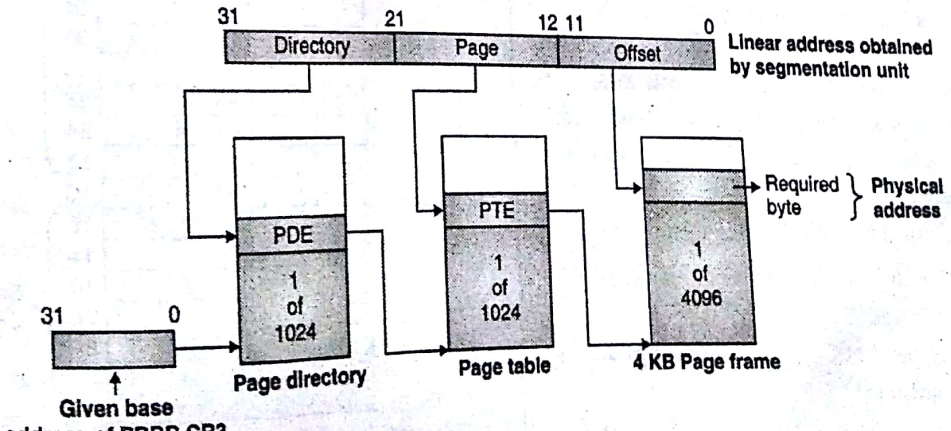


Fig. 2-Q. 2(b) : Linear to physical address translation

There are three fields in the linear address format :

1. 10-bit directory field.
2. 10-bit page field.
3. 12-bit offset field.

The 10 bit directory field is used as an index into the page directory. The 10 bit page field is used as an index to the page table determined by the page directory.

The 12 bit offset selects one of the 4096 bytes of memory from the page frame that is determined by the page table.

Function of PDBR in CR3

The physical address of the current page directory is stored in the control register (CR3). It is called as the **page directory base register (PDBR).** This register identifies the location of the page directory table in memory as shown in Fig. 2-Q. 2(b). Each entry in the page table is 4 bytes long and contains the base address of the page table and page frame.

Use of CR3 : Register CR3 gives us the base address of page directory. Then bits 22 to 31 is used as an offset into this page directory, which gives an **page directory entry**. This PDE contains base address of page table.

Then bits 12-21 are used as an offset into this page table, this gives us an **page table entry**. This PTE gives base address of 4 KB page frame. Finally bits 0-11 are used as an offset into this page frame, that give us the required byte i.e. **the physical address.**

Enabling / Disabling Paging :

Paging can be enabled or disabled using the PG bit in register CR0 (bit 31).

How paging operation can be speeded up ?

Paging can be speeded up by using a **Translation Look Aside buffer cache.** The TLB cache holds the recent 32 entries of page tables. This cache is maintained using **LRU (Last Recently Used) Algorithm.**

Whenever TLB is used, the linear address is checked if it is present in TLB i.e. a hit. And if it is not present then the earlier procedure is used to translate linear address to physical. It is mandatory to flush the complete translation look aside cache whenever the page tables are changed. The page translation cache is invisible for the application programmer, however it is visible to the system programmer.

The cache can be flushed by the following methods :

(i) Reloading CR3 with a MOV instruction. e.g. MOV CR3, EAX

(ii) By task switching to a TSS that is having a different CR3 image than the current TSS.

Q. 3(c) Draw and briefly explain Task State Segment. (6 Marks)

Ans. : Task State Segment

(i) the task and (ii) the high-performance task switching mechanism are the key elements of the multitasking software architecture in protected mode environment.

Invoking a task

A task can be invoked by two ways. A task can be invoked either directly or indirectly. We require to execute a intersegment jump or intersegment call instruction for invoking the task. In case of jump instruction is used to invoke the task no return linkage to the earlier task is available. Whereas, if a call instruction is used to switch to the new task then, back linkage information is automatically saved. The linkage information allows us to return to the instruction that follows the call instruction in the old task.

At the time of the task switching operation the contents of all the registers of 80386 processor and other data are stored for the task that is suspended on the stack. For the new task the data and information needed is collected and loaded. However this information is not loaded onto the stack. This information is stored in a special memory segment called as **Task State Segment (TSS).**

Size of TSS

TSS is a 32 bit structure. 4 GB is the maximum unit for TSS for 80386 operating in protected mode. Fig. 1-Q. 3(c) shows a 32 bit Task State Segment Structure.

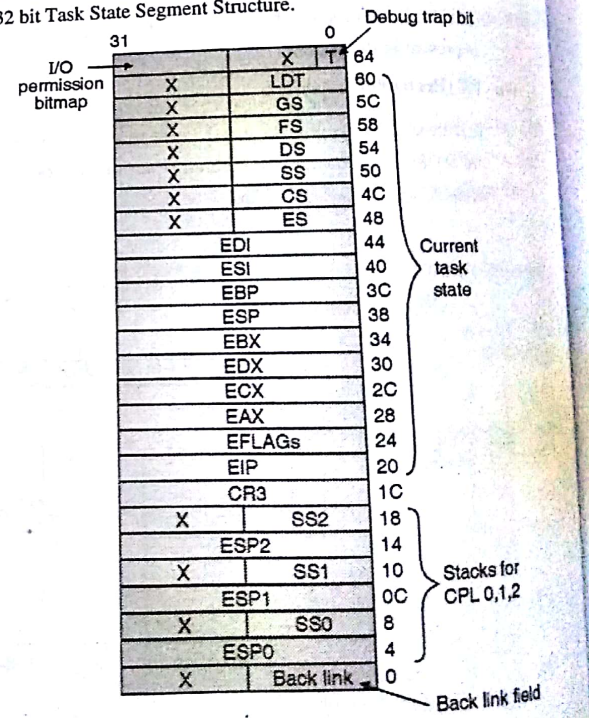


Fig. 1-Q. 3(c) : A 32 bit Task State Segment (TSS)

Contents of TSS

The task state segment comprises the storage areas for all of the 80386's 32 bit registers and the 16 bit segment selectors. It also contains additional storage for stack pointers and segment selectors for each protection level stack.

In task state segment the 80386 processor saves the various registers within the segment i.e. all 8 general purpose registers along with the segment registers, EIP, flag register, CR3 (Control Register 3) and all **privilege level stacks are stored in** the TSS.

Uses of TSS in multitasking

The main purpose of saving CR3 is that the (CR3 contains the address of page directory) page directory can be changed for each task. Similarly, we also have the LDT that can change on a per task basis.

The field marked "T" is used for enabling or disabling task switch trap. This is bit 0 at byte offset 64. If this bit is set then 80386 performs a task switch normally. And before one instruction of new task is executed, the 80386 processor generates a debug trap (exception 1). If bit is clear the trapping of task is disabled.

The I/O permission bitmap is used for I/O protection. The I/O permission bitmap is responsible for holding the 16 bit offset i.e. I/O permission bit map base address (starting address). Each I/O permission bitmap is 64 KB long and commences at the I/O permission bitmap base address.

Offset of back-link field

Finally the Backlink field is used in order to keep track of the task chain. Its offset in the TSS is 00H.

Back-link field significance

The **Back-link field** in the new task comprises of selector of the task that caused the task switch. Once the task finishes execution, it uses the Backlink field to return to the previous TSS by loading the backlink selector into the Task Register (TR).

Checks and usage by 80386

The double words ESP0, ESP1, ESP2 and ESS0, ESS1, ESS2 comprise the values for the privilege levels 0, 1, 2. On execution of the current task if an interrupt is invoked then these privilege level stacks can be addressed.

Once a task is created, the tasks LDT selector (offset 60H), PDBR (offset 1CH), T bit, the protection level stacks ESP0, ESP1, ESP2, ESS0, ESS1, ESS2 and I/O permission bit map are entered. At the time of task switching these elements do not change. However, the register portion (from offset 20H through 5CH) is modified.

As soon as 80386 starts executing the new task it reads the information that is present in the first 104 bytes of the tasks TSS. Most of data is copied as it is and the execution starts using the CS : EIP. Doesn't this sound same as executing a FAR, RET instruction and loading registers from stack.

Q. 3(b) Differentiate between memory mapped I/O and I/O mapped I/O. (4 Marks)

Ans. :

Sr. No.	Memory mapped I/O	I/O mapped I/O
1.	A memory address is allotted to an I/O device.	An I/O address is given to an I/O device.
2.	Any memory related instruction will be able to access this I/O device. eg : MOV BX, [3500 H]	Only IN and OUT instruction can access such devices.
3.	$\overline{MRDC}$ or $\overline{MRD}$ and $\overline{MWR}$ or $\overline{MWRC}$ are given to $\overline{RD}$ and $\overline{WR}$ of I/O device	$\overline{IORC}$ / $\overline{IORD}$ and $\overline{IOWC}$/ $\overline{IOWR}$ are given to $\overline{RD}$ and $\overline{WR}$ of I/O device
4.	The data from I/O device can also be given to ALU and result given back to I/O device using single instruction eg : ADD [3500 H], CX	ALU operations are not possible directly on I/O data. They are to be first brought into accumulator.

Q. 4(b) Explain any two I/O privilege instructions.

Ans. : I/O Instructions (4 Marks)

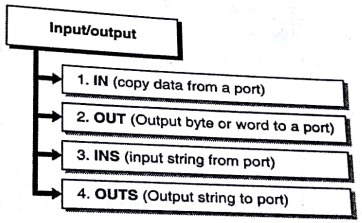


Fig. 1-Q. 4(b) : I/O Instructions

These instructions are basically related to communication with I/O devices, mapped in I/O map.

1. IN : Copy Data from a Port

Description :	Input byte, word or double word.
Operation :	This instruction will copy data from a port whose address is given in the instruction to EAX register, AX register or AL register. The data can be either 8 bit, 16 bit or 32 bit. The IN instruction has two possible formats : (i) Fixed port (ii) variable port.

	For the **Fixed port**, the port address is specified directly in the instruction. The port numbers are from 00 to FF i.e. 8 bit address is directly specified. Thus addressing mode is Direct port addressing. The port addresses F8H to FFH are reserved for I/O port addresses. Hence, they cannot be used. **E.g.** IN AL, C8 H : This instruction will copy the contents of port whose address is C8H to the AL register. For the **Variable port** IN instruction, the port address is loaded into the DX register before the IN instruction. Since DX is a 16 bit register, the port address can be any number between 0000H and FFFFH. Therefore it is possible to address up to 65, 536 ports in this mode.
Example :	MOV DX, 0FF0H; Initialize DX to point to port. IN AL, DX ; Input an 8 bit data from port 0FF0 to AL i.e. contents of port 0FF0 are copied to the AL register. The addressing mode is Indirect port addressing mode.

2. OUT Instruction - Output Byte or Word to a Port

Description :	Output byte, word or double word to immediate port [DX].
Operation :	This Instruction copies a byte from AL or a word from AX or a double word from EAX to the specified port. Load desired port address in DX. Copies contents of AL to given port address in register DX. It has two possible forms. (i) Fixed port (ii) Variable port In the **fixed port** form, the 8 bit port address is specified directly in the instruction with this form, any one of 256 possible ports can be addressed. **E.g. OUT 3B H, AL ;** **copies the contents of AL to port 3B H.** The addressing mode is Direct port addressing mode. In the **variable port** form, the contents of AL or AX or EAX will be copied to the port at an address contained in DX. The DX register should be loaded with the desired port address.
mple :	MOV DX, FFF8H OUT DX, AL

I/O Privilege Level

The I/O sensitive instructions as mentioned earlier are IN, INS, OUT, OUTS, CLI and STI. These instructions are sensitive to IOPL field. In order to execute these instructions the CPL ≤ IOPL should be satisfied. i.e. for allowing the I/O operation the CPL should be less than the IOPL. If the CPL is greater than the IOPL then it will result in a general protection violation exception.

Each task has its own EFLAG's register and so can have its own IOPL. We can change IOPL only if code is at PL0 else an exception 13 is generated.

Chapter 8 : Exceptions and Interrupts [Total Marks - 08]

Q. 4(a) When does a page fault occur ? (2 Marks)

Ans. :

This fault occurs if PG = 1 and the privilege level is incorrect or the page tables or directory contains a zero.

Q. 4(c) Explain what happens when an interrupt calls a procedure as an interrupt handler. (6 Marks)

Ans. :

Handling Interrupts and Exceptions in Protected Mode

Real mode uses a 1 KB Interrupt Vector Table (IVT) starting at address 00000H. Each 4 byte entry in the IVT consists of a CS : IP pair that specifies the address of the first instruction in the interrupt service routine (ISR). An 8 bit vector number is shifted two bits to the left to form an index into the IVT. The protected mode depends on the Interrupt Descriptor Table (IDT) to support the interrupts and exceptions.

The IDT comprises 8 byte gate descriptors for task, trap or interrupt gates. The IDT has a maximum size of 256 descriptors. The size of the IDT is controlled by a 16 bit limit value stored in the Interrupt Descriptor Table Register (IDTR) The interrupt descriptor table contains gate descriptors and not vectors. This table can be located anywhere in the memory. It is an array of 8 byte descriptors like the GDT and LDT.

There are 256 gate descriptors in interrupt descriptor table as shown in Fig. 1-Q. 4(c) However, in case of protected mode, there are 8 bytes for each interrupt, e.g. Gate-0 is located at address (IDT + 0) to (IDT + 7) and Gate-255 is located at address (IDT + 7F8) to (IDT + 7FF).

If all 256 gates are not needed for an application, the limit can be set to a value lower than 7FF H to minimize the amount of memory for the table. The protected mode interrupt descriptor table can reside anywhere in the physical address space. The location and size of the interrupt descriptor table are again defined by the contents of IDTR as shown in Fig. 1-Q. 4(c).

The Interrupt Table Descriptor Register (IDTR) is a 48 bit register. It consists of 32-bit BASE gives the starting point of the table in memory and 16-bit LIMIT gives the size of the interrupt descriptor table.

Whenever an interrupt or exception occurs, an identifier or vector number is multiplied by 8 and is added to the IDT base address that is stored in the IDTR. The instructions LIDT (Load Interrupt Descriptor Table Register) and SIDT (Store Interrupt Descriptor Table Register) are used with the IDTR.

The LIDT is a privileged instruction and is executed whenever the CPL is zero. The SIDT is a non-privileged instruction. It can be executed any time. Each instruction uses a single operand that specifies the address of the 48 bit memory word. The 6 byte word is used to change the location of the IDT or find its current address.

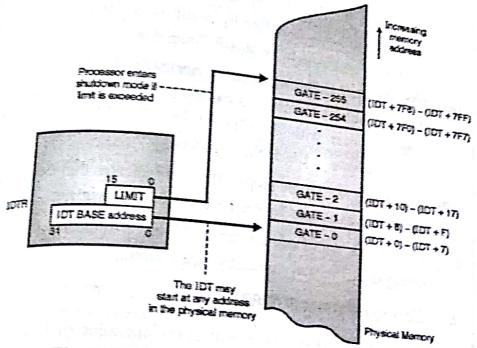


Fig. 1-Q. 4(c) : The protected mode interrupt descriptor table

Returning from an Interrupt Procedure

When the ISR is finished executing, it returns control to the interrupted code using an IRET (Interrupt Return) instruction. But before returning from interrupt it is necessary to examine the EFLAGs register that was pushed onto stack. (RET instruction cannot be used, as it doesn't check EFLAGs).

The IRET instruction increments EIP by four and then EFLAGs is restored. Then CS and EIP values that point to previous procedure is also restored and execution resumed.

Processing interrupt service routines

1. IDT stores the descriptors for interrupt service routines.
2. The three gate descriptors allowed are interrupt gate, task gate and trap gate descriptors.
3. ISRs operate like programmed procedures / subroutines.
4. Before transferring program control all the register contents used are stored.
5. ISRs are invoked by interrupts or exceptions instead of CALL instructions.
6. ISRs terminate with IRET instead of RET instructions.

Chapter 9 : Initialization of 80386DX and Debugging [Total Marks - 14]

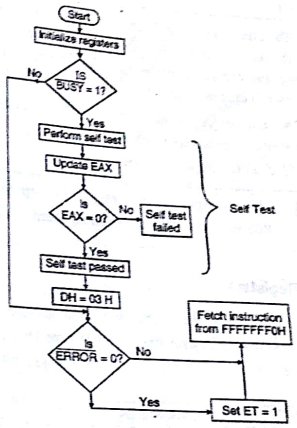
Q. 5(a) What are the contents of various registers of processor 80386 after reset ? (3 Marks)

Ans. :

Processor State After Reset

The 80386 can enter the Real Mode if there is an active low on the RESET pin or if the PE bit of CR0 register is cleared. An active low input on the RESET pin of 80386, will terminate processing, exception handling and will reset the processor.

Self Test



p(2.7) Fig. 1-Q. 5(a) : Activity after reset (Hardware activity)

After reset, the 80386 may perform **self test** if initiated. This is done by external hardware. If the $\overline{BUSY}$ pin of the 80386 is active high then self test is initiated, otherwise self test is skipped.

In case a self test is requested and the 80386 passes this test, then the register EAX will contain all 0 bits when software beings to execute. If EAX ≠ 0, then it indicates that 80386 self test is failed. In addition to EAX, the 80386 alters the DX register contents whenever it is powered up or reset. The DH register contains 3 (0000 0011). This is done in order to identify the processor as an 80386.

80386 while reset, also samples the $\overline{ERROR}$ signal. An active low on this pin indicates that an 80387 Coprocessor is connected to it. The processor also sets ET to enable

communication between the processor and the 80387. An active high on $\overline{ERROR}$, clear the ET.

It then clears the EFLAGs register including the IP. The values of remaining registers and flags are as follows :

CS Selector	0000 H
DS Selector	0000 H
ES Selector	0000 H
SS Selector	0000 H
GS Selector	0000 H
FS Selector	0000 H
IP	0000FFF0 H
EFLAGS	00000002 H
IDTR	Base =0 Limit= 03FF H

The other registers are undefined. After completing the above steps the 80386 is operating in the Real Address mode and begins fetching and executing instructions. The interrupts are disabled. After reset, it fetches its first instruction from FFFFFFF0H.

The real mode is used to initialise peripherals, enable interrupts and enter protected mode.

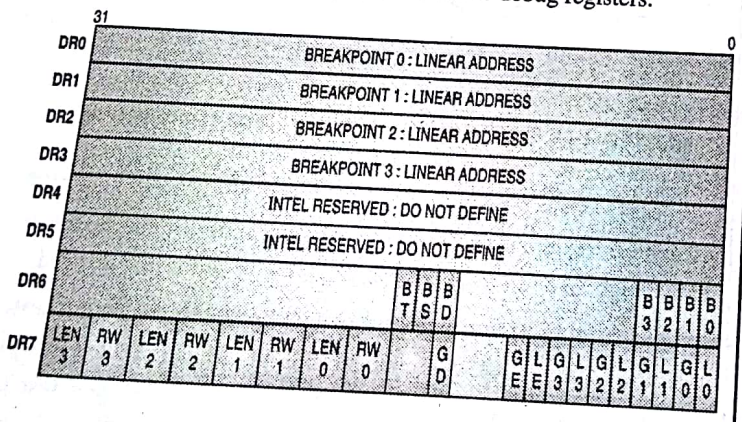
Fig. 1-Q. 5(a) shows this activity.

Q. 5(b) How many debug registers are present in 80386 ? List and draw all of them. (4 Marks)

Ans. :

Debug Registers

Debug registers allow the 80386 to provide debugging features. In order to control the debug feature DR0-DR7 debug registers are used. *Fig. 1-Q. 5(b) shows the debug registers.*



P(1.24)Fig. 1-Q. 5(b) : Debug Registers of 80386

1. Debug registers 0 through 3

The first four debug registers contain 32 bit linear breakpoint addresses. The breakpoint addresses, which may locate in an instruction or a datum are constantly compared with the addresses generated by the program.

If match occurs, 80386 will cause a type 1 interrupt to occur if directed by the debug registers DR6 and DR7. The breakpoint addresses are very useful in debugging faulty software.

2. Debug registers 4 and 5

These registers are reserved by INTEL.

3. Debug register 6 (DR6)

It is known as the **debug status register.** This bit if set indicates the condition that may have cause the last debug fault (exception 1). These bits are never cleared by the processor. This bit must be manually cleared by writing into the status register.

The control bits in DR6 and DR7 are as follows :

(i) **B3-B0 :** They indicate which of the four debug breakpoint addresses caused the debug interrupt.

e.g. : If the B0, bit is set it references linear address contained in DR0, modified by condition set by LEN0, RW0, L0, G0, LE and GE fields in DR 7 register.

(ii) **BD (Break for Debug register access) :** If set, it indicates that the debug interrupt was caused by an attempt to read the debug register with the GD bit set. The GD bit of DR7 protects access to the debug registers.

(iii) **BS (Break for Single step) :** If set, it indicates that the debug interrupt was caused by the TF bit in the flag register.

(iv) **BT (Break for Task switch) :** If set, it indicates that the debug interrupt was caused by a task switch.

4. Debug Register 7 (DR7)

This register allows us to control the operation of four linear address breakpoints. This can be done by programming. Each breakpoint can be controlled by a set of four fields. They are :

(i) **LEN3-LEN0 (Breakpoint Length)**

Each of these four length fields pertains to each of the four breakpoint addresses stored in DR3-DR0. These bits also define the size of access of the breakpoint. Table 1-Q. 5(b) shows the different sizes of the breakpoints.

Table 1-Q. 5(b)

LEN	LEN bits in DR7
00	1 byte
01	1 word (2 bytes)
10	Reserved
11	1 double word (4 bytes)

(ii) **RW3-RW0 :** Each of these four read/write fields pertains to each of the four breakpoint addresses stored in DR3-DR0. The RW field selects cause a list of different access types. They are shown in Table 2-Q. 5(b).

Table 2-Q. 5(b)

RW	RW bits in DR7
00	Instruction access
01	Data write
10	Reserved
11	Data read or write

(iii) **GD (Global Debug Access) :** If set, this bit prevents any read or write operation of a debug register by generating the debug interrupt. This bit is automatically cleared during the debug interrupt, so that debug registers can be read or changed, if required.

(iv) **GE (Global Exact) :** If set, 80386 selects a global breakpoint address for any of the four breakpoint address registers.

(v) **LE (Local Exact) :** If set, selects a local breakpoint address for any of the four breakpoint address registers.

Q. 6(b) What all initializations required to start processor in real mode after reset ? (4 Marks)

Ans. :

Software Initialization for Real Address Mode

In the Real address mode the following need to be initialized.

Interrupt Table

On reset all the interrupts are disabled. If the processor detects a NMI or an exception, the 80386 processor will try to access the interrupt table. Hence, the initialization software needs to check the following :

(a) Modify the IDTR to point to a valid interrupt table.

(b) Modify the IDTR limit =0. So, if an exception or NMI are recognized, the processor will shutdown.

(c) Assign pointers to interrupt handlers in the interrupt table.

Stack

Until the SS is loaded to point to an area in the RAM, none of the stack instructions can be used.

Returning Back to Real Mode

Whenever processor wants to return to Real mode without turning off the power or resetting the system the user can simply clear the PE bit in CR0. In case paging is enabled it should be disabled by clearing the PG bit in CR0. Also CR3 should be cleared, so that paging cache is cleared. Use segment descriptors containing the following values

Table 1-Q. 6(b)

Attribute	Value
Limit	64 K (FFFF H)
Byte Granular	G = 0
Expand up	E = 0
Writable	W = 1
Present	P = 1

Initialise segment registers as required by real mode. Clear PE bit. Flush the instruction queue by executing a far JMP to real mode code. This puts appropriate values in the access rights of CS register. Load the base and limit of the real mode interrupt vector table/Interrupt Descriptor Table (IDT) using LIDT instruction. This process makes us enter the Real mode. Fig. 1-Q. 6(b) shows a flowchart that outlines this procedure.

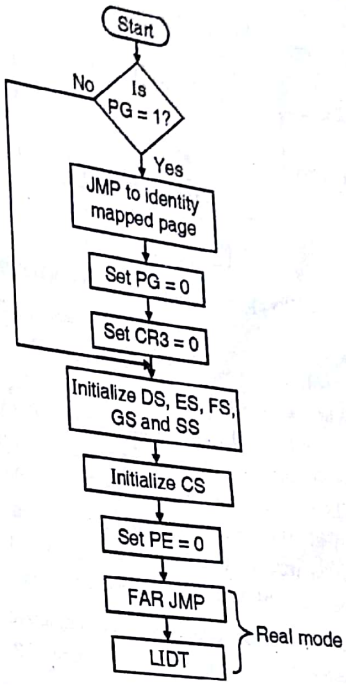


Fig. 1-Q. 6(b) : Process to Return to Real Mode

Q. 6(a) Write short note on "Instruction Address Breakpoint". (3 Marks)

Ans. :

Instruction Address Breakpoint

Instruction address breakpoint is fault. It is observed whenever the processor executes the instruction at the given address earlier. The RF flag is set and pushed onto the stack. While the next instruction is executed, the debug fault if any is ignored.

Chapter 10 : Virtual 8086 Mode [Total Marks -12]

Q. 6(c) With neat diagram explain "Entering and leaving V86 mode". (6 Marks)

Ans. :

Entering Virtual 86 Mode

The 80386 begins to operate in Virtual 8086 mode whenever VM bit in the EFLAGs register is set. This is **not possible** by PUSHF and POPF instructions. Because POPF is constructed such that it will not alter the status of VM bit. **The two ways by which we can enter the virtual 8086 mode is through (i) a task switch (ii) an interrupt return.** Fig. 1-Q. 6(c) shows entering and leaving VM86 mode.

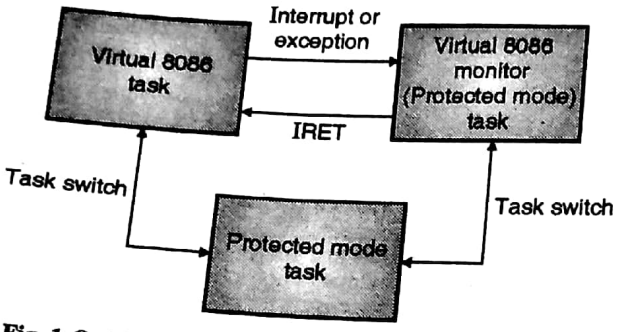


Fig. 1-Q. 6(c) : Entering and leaving virtual 8086 mode

Task switching approach is preferred, when a TSS is created, such that it will be used by virtual 8086 task, set VM bit in EFLAG register. Whenever the 80386 switches into this task it sees that VM is set and it loads all segments and interprets all addresses and instructions like the 8086. It loads the full 32 bit registers EAX through EDX and ESI through ESP. The TSS should be created with the desired values in each segment register and in the EIP. FS and GS are not required to be initialized. The privilege level 0 stack is critical and should point to PL0 stack segment.

The LDT register should also be initialized properly. Descriptor tables should not be used. The register CR3 is loaded when a VM86 task is invoked.

If paging is enabled, CR3 should contain physical address of the base of page directory. Another approach is to push the data onto the stack and issue an IRET instruction. The IRET pops EFLAGs off the stack, right after the return address and it can be made to force a transition in the VM86 mode. This is different than normal interrupt return. All the extra information is required to be duplicated before an IRET is executed. The current privilege level at the time of IRET is zero.

Leaving V86 Mode

The 80386 leaves the virtual mode when an interrupt or exception occurs.

1. A task switching from 8086 virtual task to other task caused by an interrupt or an execution loads the EFLAGs register from the TSS of the new task. If the task is a new task, then the VM bit is cleared. It then loads the segment registers as defined by new TSS and begins execution of instructions of new task according to the protected mode description.
2. The interrupt or exception which vectors to PL0, stores current setting of EFLAGs on stack, then clears the VM bit. This makes the 80386 execute the instructions in protected mode.

Fig. 1-Q. 6(c) shows how the interrupts and the exceptions cause the 80386 processor to switch between the virtual 8086 task and a virtual 8086 monitor task. The monitor task is a protected mode task that is responsible for initialization handling interrupts and exceptions, and I/O required for executing a VM86 mode task. There can be multiple VM86 tasks. However, each task needs support of a monitor task.

Q. 5(c) With neat diagram explain the process of linear address formation in V86 mode. **(6 Marks)**

Ans. : Address Generation in Virtual Mode

While operating in the virtual 8086 mode, the ability to reference the memory using 80386 addressing modes is lost. Virtual 8086 does not support scaled addressing. The addressing is similar to the real mode addressing. Virtual mode addressing only supports 8086 addressing modes.

There are no descriptor tables, no segment selectors, no gate descriptors, no descriptors and no RPL fields. All the memory segments CS, DS, ES, SS, FS and GS are exactly 64 KB long. There is no protection to the memory and all privilege information is lost in the virtual 8086 mode.

A VM86 program can access 32 bit memory. A 16 bit segment register is shifted four bits to the left and these shifted contents are added to a 16 bit offset to give 20 bit physical address.

Let DS = B000H and offset = 5F00H then the physical address will be,

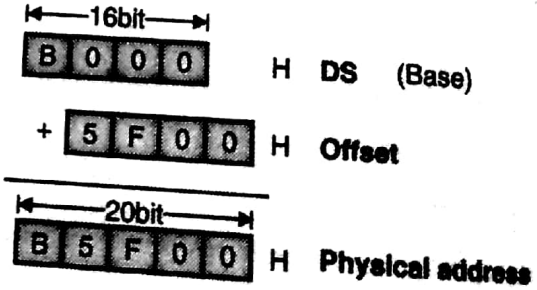


Fig. 1(a)-Q. 5(c) : Calculating the physical address in 80386 real mode

If a carry is generated out of the 20 bit effective address it is retained. Thus, 80386 accesses a larger real mode addressing space using 21 bit addresses. The actual range is from 000000H to 10FFEFH (FFFF0H plus FFFFH i.e. 1MB plus 64 KB). These 21 bits are part of the 32 bit linear address. An 8086 processor wraps the addresses around the end of a segment from FFFFH to 0000H.

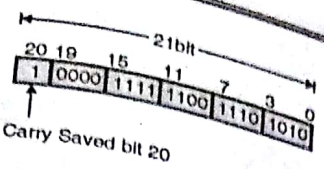


Fig. 1(b)-Q. 5(c) : 21 bit linear address in 80386's real mode if carry is generated

Virtual 8086 mode tasks use 32 bit linear addresses. If a program uses only lower 21 bits of the linear address, then linear address can be translated/paged to any physical address in 4 GB range of systems memory.

80386 can generate 32 bit effective address with the address size command prefix. The address must not exceed beyond 65535 to maintain compatibility with 80286 real mode otherwise pseudo protection fault may be generated.

Chapter 11 : 80386 DX Signals [Total Marks - 22]

Q. 7(a) Explain HOLD and HLDA signals of 80386DX.

Ans. : **(3 Marks)**

DMA Interface Signals

These pins are used in order to interface with the DMA controller. These signals are

(i) HOLD (ii) HLDA

(i) **HOLD :** A high on this pin indicates that DMA controller is requesting for bus access.

(ii) **HLDA :** The DX 80386 activates HLDA signal after completion of current bus cycle and enters into a HOLD state. In HOLD state, its local bus signals are in high impedance state.

Q. 8(a) Explain the signal :

(i) NMI

(ii) INTR

(iii) RESET **(3 Mark)**

Ans. :

(i) Non-maskable interrupt (NMI)

It indicates that the interrupt input on this pin is non-maskable. This input can be edge triggered. It causes the 80386DX to execute an ISR. Unless and until it is serviced, the 80386 will not service subsequent NMI requests.

(ii) **Interrupt request (INTR)**

An interrupt request is used by the external circuitry to request an interrupt.

(iii) **Reset**

It initializes the 80386, cause it to begin executing software at memory location. It is the default address on power ON. The 80386 is reset to the real mode.

Q. 7(b) List various bus states when address pipelining is used.

Ans. : **(4 Marks)**

Bus Cycle and Bus States

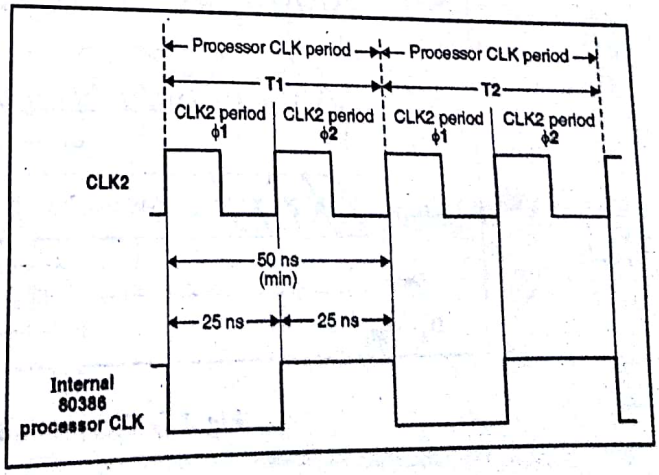
A bus cycle is the activity performed by the microprocessor in order to access data from memory or I/O devices. A bus state is defined as the shortest time of the activity of the processor bus. It is one processor clock period in duration.

The internal processor clock (PCLK) signal is at half the frequency of the external clock input signal. Fig. 1-Q. 7(b) shows this relationship.

CLK2 input of the 80386 processor provides the fundamental timing. It is divided by two internally to generate the internal processor clock that is used for instruction execution. The internal clock comprises of two phases $\phi 1$ and $\phi 2$.

Each CLK2 period is a phase of the internal clock, e.g. In a 20 MHz 80386DX system, CLK2 frequency equals 40 MHz. Each clock cycle has a duration of 25 ns $\left(\frac{1}{40\text{ MHz}}\right)$. In Fig. 1-Q. 7(b) the two phases ($\phi 1$ and $\phi 2$) of the processor cycle contribute as one processor clock period. Hence, the processor clock period is 50 ns (minimum).

As shown in Fig. 1-Q. 7(b) each bus cycle has two states T1 and T2. Each state has two phases or two clock cycles. In first state T1 address and bus status pins are active and in the second state T2 the data transfer will occur.

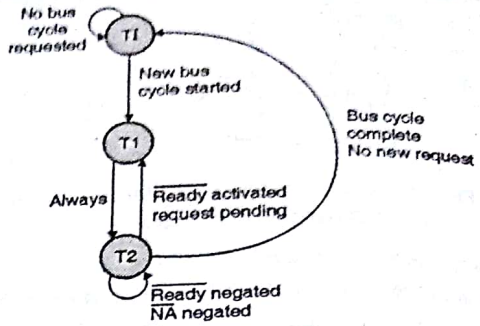


p(6.1) **Fig. 1-Q. 7(b) : Processor clock periods (bus states)**

The clock signal that is applied to the CLK2 input of 80386DX is twice the frequency rating of the microprocessor. Therefore, CLK2 of an 80386DX-16 is driven by a 32 MHz signal. This signal must be generated externally.

If the 80386 does not need any bus cycles, it remains in the T1 idle bus state. Fig. 2-Q. 7(b) shows the **bus state transition diagram**.

T1 is the **idle state** it indicates that no bus cycle is currently being processed. This cycle starts after a hardware reset.Bus states T1 and T2 are first and second bus states of a bus cycle.



p(6.2)**Fig. 2-Q. 7(b) : Bus state transition diagram (without address pipelining)**

Q. 7(c) Draw read cycle with non-pipelined address timing. **(6 Marks)**

Ans. :

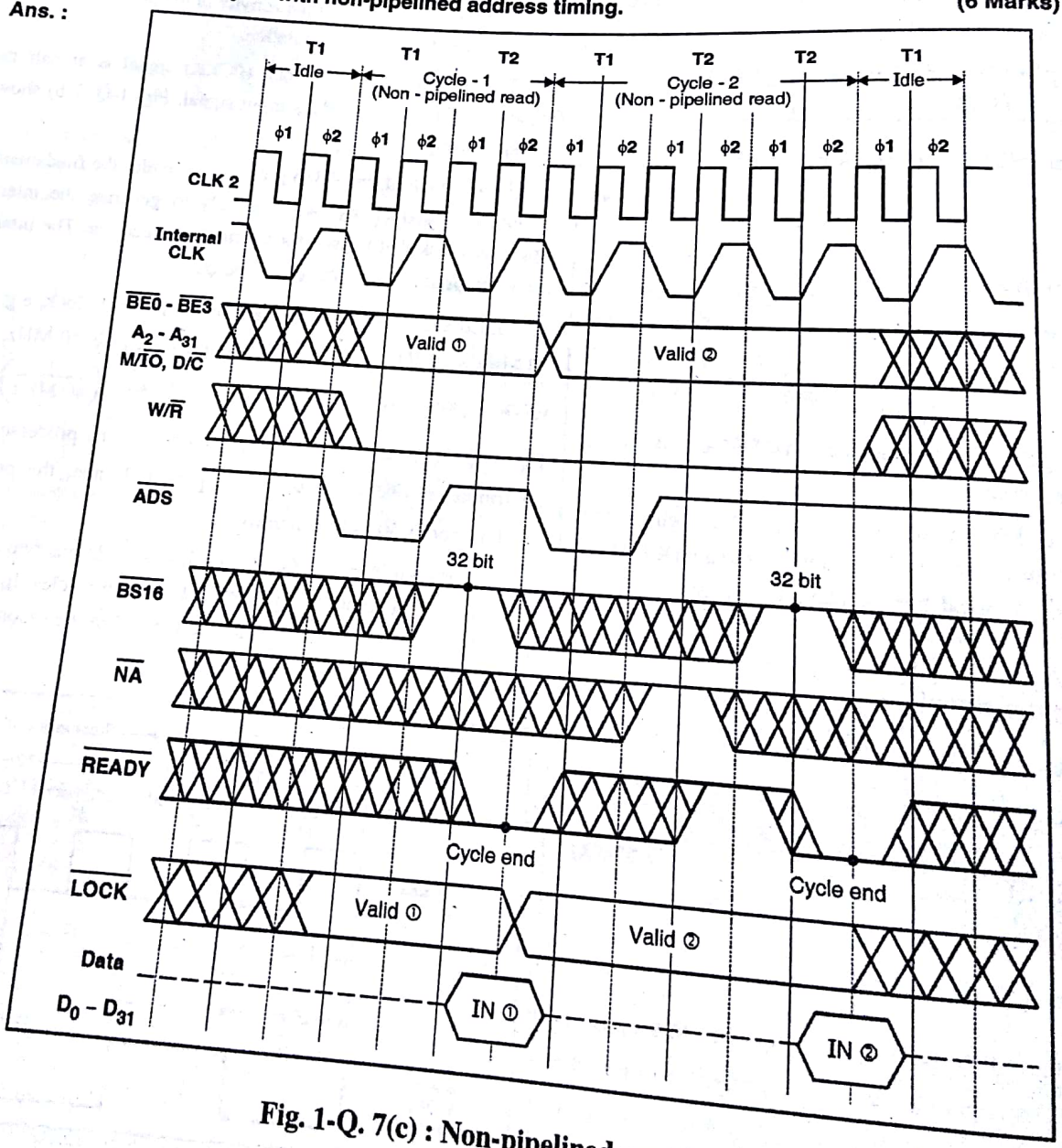


Fig. 1-Q. 7(c) : Non-pipelined read bus cycles

8(c) Draw 'write cycle with pipelined address timing'.

ns. : Pipelined Read and Write Bus Cycles

(6 Marks)

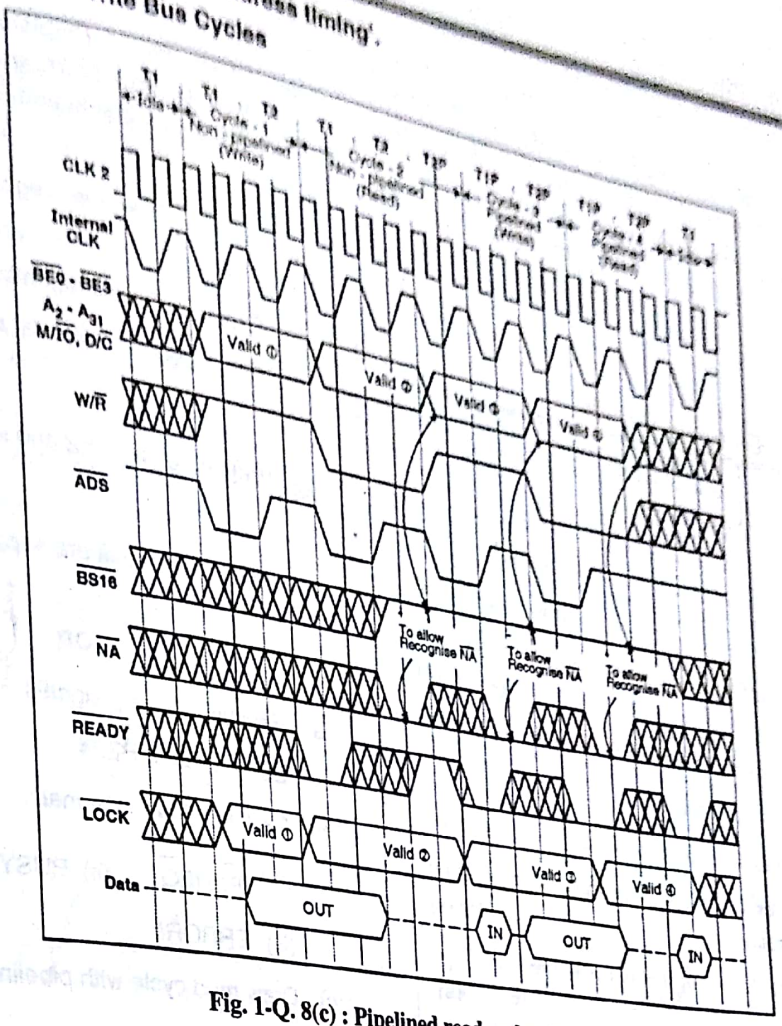


Fig. 1-Q. 8(c) : Pipelined read and write cycles

Chapter 12 : 80387 Numeric Co-processor [Total Marks - 04]

Q. 8(b) Draw and explain 80387 register stack.

(4 Marks)

Ans. : 80387 Register Stack

80387 has bits register stack, labelled as to OR ST(X); where X will vary from 0 to 7.

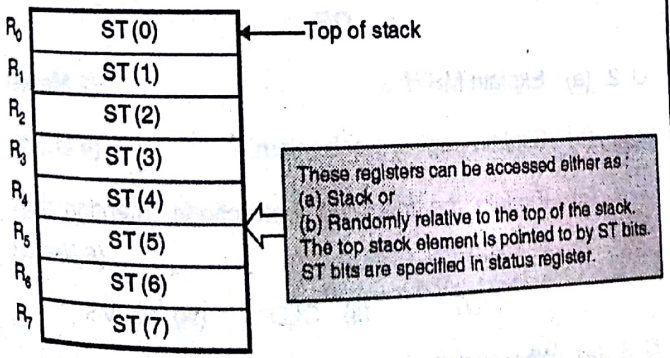
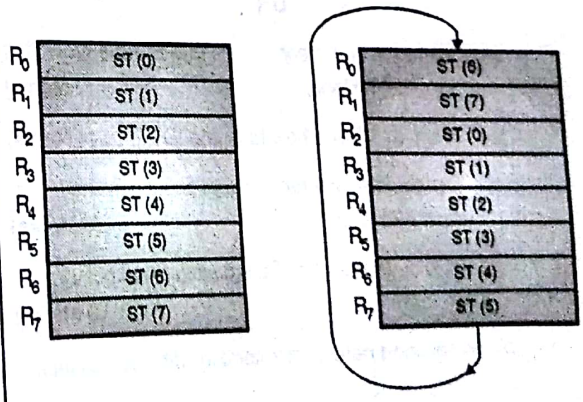


Fig. 1-Q. 8(b)

How these registers function.?

As a stack : An operand can be pushed onto or popped from the stack. The PUSH operation first decrement ST by 1 and then loads the operand into new top of the stack element. POP operation retrieves the top of stack and then increments ST by 1. The same is shown in Fig. 2-Q. 8(b).



(a) (b)

Fig. 2-Q. 8(b) : Push/pop operation on register

As a register

Register may be referenced by using an index to stack pointer. This is called ***relative stack addressing.*** For relative stack addressing the registers are considered to be in circle i.e. register 7 being next to R_0. R_2 is ST(2) in Fig. 2(a)-Q. 8(b) and it is ST(0) in Fig. 2(b)-Q. 8(b).

❑❑❑

Question Papers

May 2017

Q. 1 (a) What is the use of following instructions ?

(i) Wait (ii) Lock **(2 Marks)**

(b) Explain segment address translation in detail. **(4 Marks)**

(c) Draw and explain segment descriptor. **(6 Marks)**

OR

Q. 2 (a) What is the use of Direction Flag ? **(2 Marks)**

(b) Draw and explain the system address and system segment registers. **(4 Marks)**

(c) Explain the following instructions, mention flags affected : **(6 Marks)**

(i) CWD (ii) BT (iii) LAHF

Q. 3 (a) List the registers and data structures that are used in multitasking. **(2 Marks)**

(b) Differentiate between memory mapped I/O and I/O mapped I/O. **(4 Marks)**

(c) Explain what happens when an interrupt calls a procedure as an interrupt handler. **(6 Marks)**

OR

Q. 4 (a) Write the two mechanisms that provide protection for I/O functions. **(2 Marks)**

(b) What is IDT and how to locate IDT ? **(4 Marks)**

(c) Explain the different exception conditions-faults, traps and aborts. **(6 Marks)**

Q. 5 (a) Write short note on "Task Switch Break

Q. 6 (a) Write short note on

(b) Which bit of EFl Explain, how hard with each other to

(c) Explain, how tes TLB ?

Q. 7 (a) Explain following

(i) ADS#

(b) Write note on CL

(c) Which data type

Q. 8 (a) Explain followir BE0# through

(b) Explain followi

(i) PEREQ

(iii) ERROR#

(c) Draw read cyc

Q. 1 (a) Explain imm with an exan

(b) Explain with

(c) Explain in d

Q. 2 (a) Explain MS

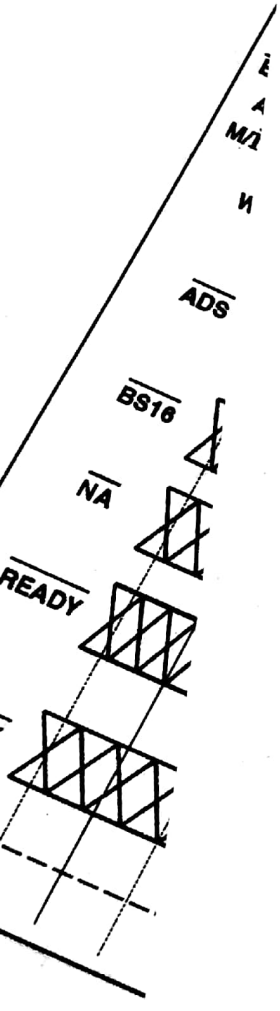


Fig. 1