PART 2 : 80386 Processor, Architecture and Bus Cycles

1.21 Introduction :

The 80386DX is a true 32 bit microprocessor. It is a logical extension of the Intel 80286. The 80386 is software compatible with Intel 8086, 80186 and 80286. It consists of separate 32 bit internal registers, a 32 bit address and data bus. It has a 32 bit physical address and can access upto (2^{32} bytes) i.e. 4 GB of main memory and (2^{4} bytes) i.e. 64 Tetrabytes of virtual memory. It can be operated from a 12.5 MHz, 16 MHz, 20 MHz, 25 MHz or 33 MHz clock. It gives us the following facilities.

(1) Multitasking support.
(2) Memory management.
(3) Pipelined architecture.
(4) Address translation cache.
(5) High-speed bus interface.
(6) Expanded instruction set.
(7) Page Translation.

It can operate in three different modes. These modes are in the order of their increased complexity. They are

(i) Real Address mode.

(ii) Virtual 8086 mode.

(iii) Protected Virtual Address mode.

The Real address mode is also referred to as real mode. It function is to initialize the 80386DX for protected mode operation.

The protected virtual address mode is referred as the protected mode. It provides paging, virtual addressing, multilevel protection, multitasking and debugging capabilities. The virtual mode allows the execution of 8086 application, taking an advantage of 80386 protection mechanism.

The 80386 has an 8086/8088 compatibility in which half of the processor shuts down and the other half becomes an embedded 8086. This allows us to run a fast 8086.

1.22 Features of 80386 :

(1) It is a 32-bit processor. It has 32-bit ALU which allows to process 32-bit data at a time.

(2) Address bus is also 32-bit. Therefore it can access 4 GB (2^{32}) physical memory. It can also access 64 Terabyte of (2^{46}) virtual memory.

(3) It has pipelined architecture which allows simultaneous instruction fetching, decoding, execution and memory management. Because of instruction pipelining, higher bus bandwidth and on-chip address translation mechanism, the average execution time has been significantly reduced.

(4) It allows user to switch between different operating systems such as DOS and UNIX.

(5) The 80386DX can operate in real mode, protected mode or virtual 8086 mode.

(6) It is compatible with 8086, 8088, 80186, 80286 architectures i.e. any code runs under these microprocessors will also run in 80386DX processor.

(7) It has different data types like bit, byte, word, double-word, Quadword, Tenbytes in integer (signed and unsigned form)

(8) The 80386DX has separate pins for its address and data bus lines. This results in higher performance and easier hardware design.

(9) The prefetch unit permits 80386DX to prefetch upto 16 bytes of instruction code. Therefore, the fetch time of most of the instruction is hidden. This increases the performance.

(10) It contains dedicated hardware for performing high-speed address calculation, logic-to-linear address translation and protection checks.

(11) It contains the Translation Lookaside Buffer (TLB) that stores recently used page directory and page table entries. This buffer consists of 32 sets of table entries which allows direct access of 128 k bytes of paged memory.

(12) Testability features include a self-test and direct access to the page translation cache. Four new break point traps on code execution or data accesses for powerful debugging.

1.23 Architecture of 80386DX :

➤➤➤ **[University Exam – May 2000, Dec. 2005 !!!]**

The internal architecture of the 80386 consists of six functional units :

(1) Bus Interface unit. (2) Code Fetch unit.
(3) Instruction Decode unit. (4) Execution unit.
(5) Segmentation unit. (6) Paging unit.

All of these functional units of the 80386DX operate in parallel. This parallel operation is called as **pipelined processing** as fetching an instruction from the memory, decoding the instruction, executing the instruction, memory management and bus access several instructions is simultaneously unit.

Fig. 1.23.1 shows the architecture of 80386DX.

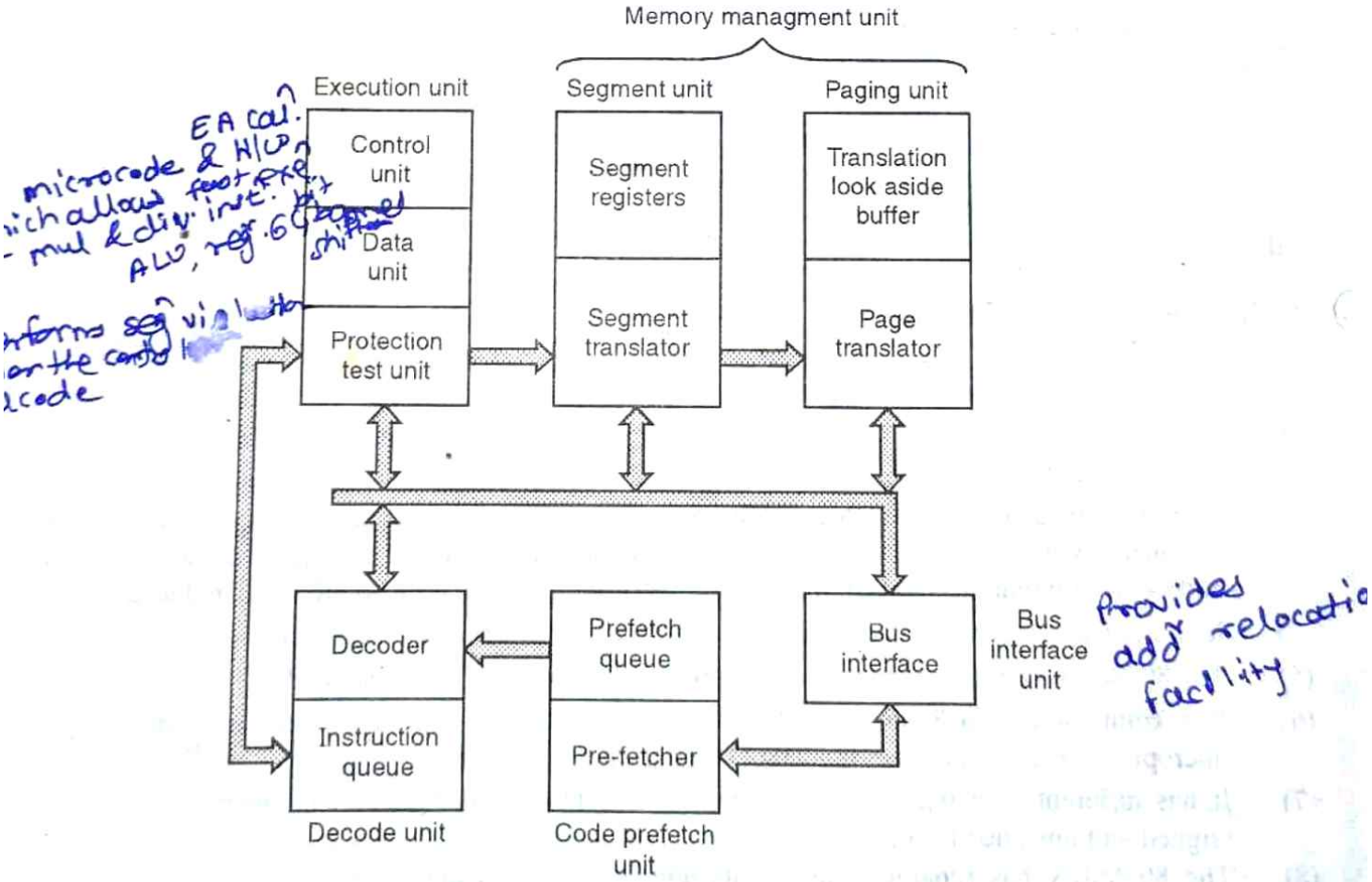


Fig. 1.23.1 : Architecture of 80386DX processor.

These 6 units are completely independent of each other and they share the work of the CPU. Such a work division speeds up the processing and reduces the processing time.

1.24 Bus Interface Unit (BIU) :

This unit is responsible for interfacing the 80386DX processor with external memory and I/O devices. It provides a 32 bit bi-directional data bus and a 32 bit address bus. The tasks of the BIU are

(1) It accepts internal requests for fetching instructions and transferring data from the code prefetch unit. It then generates the address, data and control signals for the current bus cycles.

(2) It reads data from memory and I/O devices.

(3) It also writes data from memory and I/O devices.

(4) It controls the interface to external bus masters and co-processors.

(5) It provides address relocation facility.

1.25 The Code Prefetch Unit :

The code prefetch unit fetches instructions from the memory when the bus interface unit is not executing bus cycles. The prefetched instruction is then stored in 16 byte prefetch queue.

The Prefetched Queue :

- The code prefetch unit is supposed to fetch an instruction.
- It does this when the BIU is not performing bus cycles to execute an instruction. The prefetch Queue fetches upto 16 instruction bytes for the next instruction.
- These bytes are called as prefetched bytes and they are stored in a first-in-first-out (FIFO) register set, which is called as **"Prefetched Queue"**.

Significance of Prefetched Queue :

To undestand the significance of prefetched Queue refer Fig. 1.25.1.

- The instruction corresponding to memory location 2056 7700 H, the BIU fetches next sixteen instruction bytes from locations numbered as 1 to 16.
- The prefetcher fetches the instructions in the order in which they appear in the memory. It reads one double word at a time, not caring whether it has a complete instruction or has pieces of two instructions with each access.
- These instruction bytes are stored on the first-in-first-out (FIFO) basis.
- When the code prefetch unit completes the fetching of an instruction, it gives it to the instruction decode unit for execution. The Prefetch Queue always holds the instruction bytes of the next instruction to be executed by the instruction decode unit.
- The code prefetches are given a lower priority than the data transfers. In case, a memory access is without wait state, then the prefetch activity never delays execution. Due to prefetch activity the processor spends zero time waiting for the next instruction.

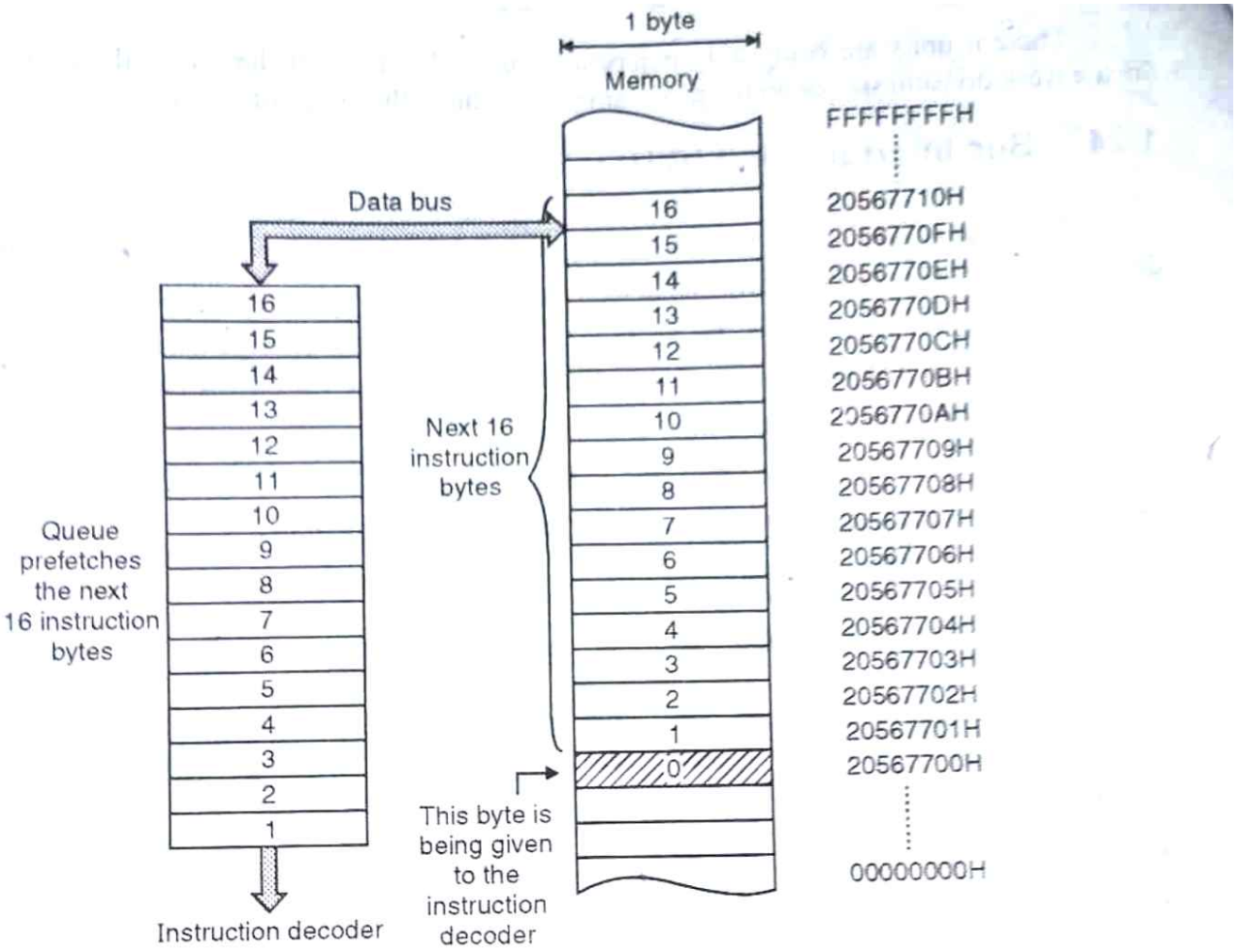


Fig. 1.25.1 : Significance of Queue

Behaviour of Prefetch Queue in JUMP and CALL instruction :

- The 80386DX BIU will not initialise a fetch unless and until there are 4 empty bytes in its queue. The BIU normally obtains 4 instruction bytes per fetch.
- If a Jump (JMP) or CALL instruction appears in the main program. the all the existing instruction bytes in the Queue are flushed out and the Queue is mode empty.
- Then the Queue is reloaded with the new instruction bytes which correspond to the new locations mentioned in the JMP or CALL instructions.

Why is the Queue only 16 byte long ?

The longest instruction, in the instruction set of the 80386 is 16 byte long. Hence, with a 16 byte long queue it is possible to prefetch even the longest instruction in the instruction set.

1.26 Instruction Decode Unit :

- This unit takes an instruction from the 16-byte prefetch queue and converts it into microcode.
- This microcode is then stored into the instruction queue.
- If the instruction needs any immediate data or offset, then it is taken from the prefetch queue and stored in the instruction queue.

1.27 Execution Unit :

The execution unit executes each instruction received from an instruction decoded queue. The Execution unit consists of

(i) The control unit. (ii) The data unit. (iii) The protection test unit.

1.27.1 The Control Unit :

- The control unit is used to perform control function of an instruction and effective address calculation.
- It contains the microcode and special hardware, which allows the fast execution of multiply and divide instructions.

1.27.2 The Data Unit :

- The data unit consists of an ALU, eight general purpose registers and 64 bit barrel shifter.
- It performs the operations requested by the control unit. The barrel shifter performs multiple bit shifts in one clock.

1.27.3 The Protection Test Unit :

- It performs the checks for segmentation violations under the control of microcode.
- The execution partially supports pipelining. The execution unit overlaps the execution of any memory reference instruction with the previous instruction.

1.28 Segmentation Unit :

- The segmentation unit translates the logical addresses into linear addresses at the request of the execution unit.
- The translated linear address is given to the paging unit.
- The linear address is of 32 bit. It is formed when the segment unit adds the effective address to the segment base.
- There are six segment registers in the segmentation unit.

1.29 Paging Unit :

- When the 80386DX paging mechanism is enabled, it translates the linear address to physical address.
- In case, if paging is not enabled, the physical address is identical to the linear address and no translation is necessary.
- The paging unit gives physical addresses to the BIU for memory or I/O access.

1.30 80386 Registers :

The 80386DX register set can be categorized according to their usage.

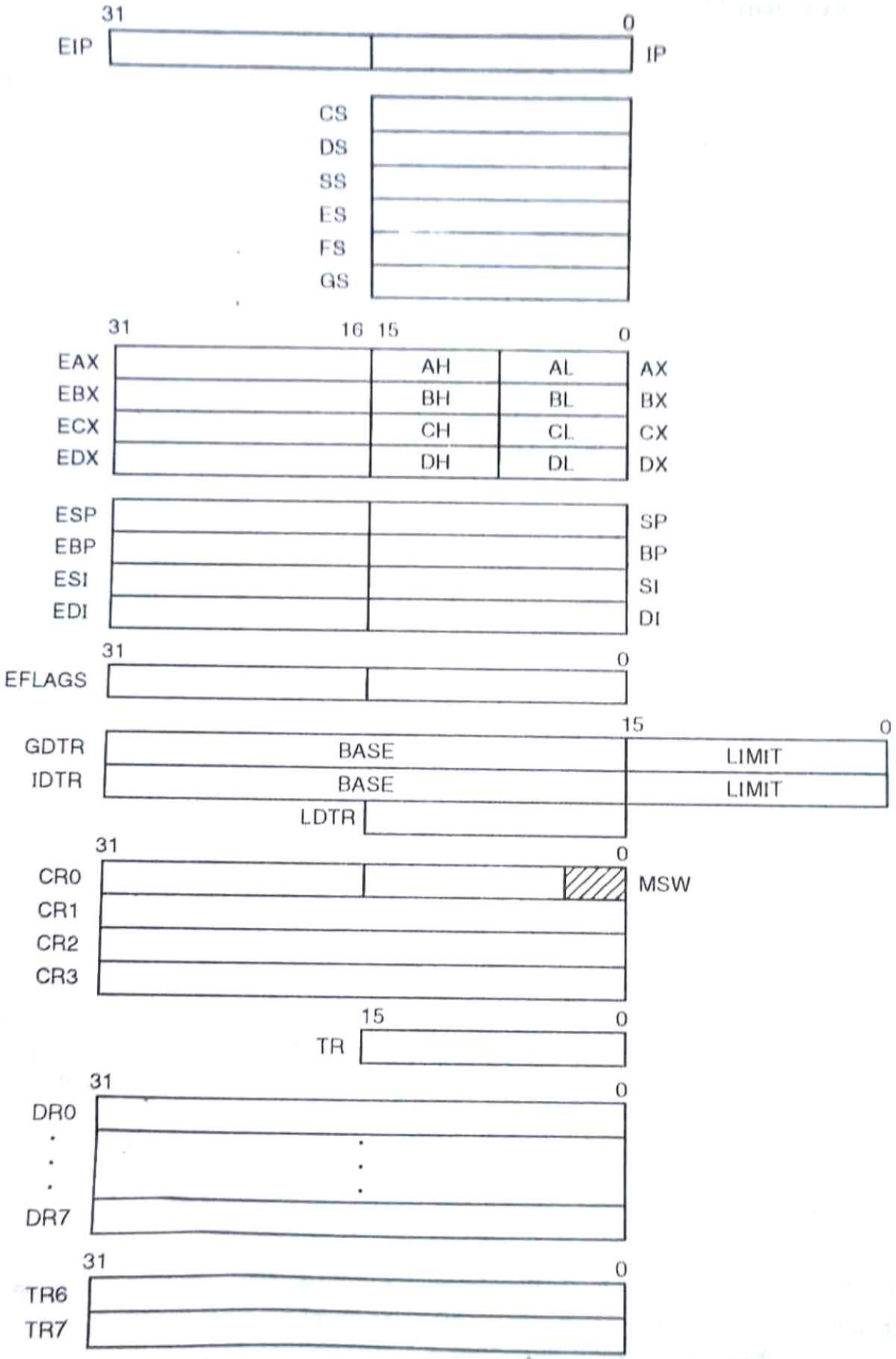
(i) General purpose registers. (ii) Segment registers.

(iii) Index, pointer and base registers. (iv) Flag registers.

(v) System address registers. (vi) Control registers.

(vii) Debug Registers.

Fig. 1.30.1 shows the complete register set of the 80386DX.


31
0
EIP
IP
CS
DS
SS
ES
FS
GS
31
16 15
0
EAX
AH
AL
AX
EBX
BH
BL
BX
ECX
CH
CL
CX
EDX
DH
DL
DX
ESP
SP
EBP
BP
ESI
SI
EDI
DI
31
0
EFLAGS
15
0
GDTR
BASE
LIMIT
IDTR
BASE
LIMIT
LDTR
31
0
CR0
MSW
CR1
CR2
CR3
15
0
TR
31
0
DR0
DR7
31
0
TR6
TR7

80386DX

1.30.1 General Purpose Registers :

- The general purpose registers are used in any manner that the programmer wishes.

Fig. 1.30.2 shows a general purpose registers.

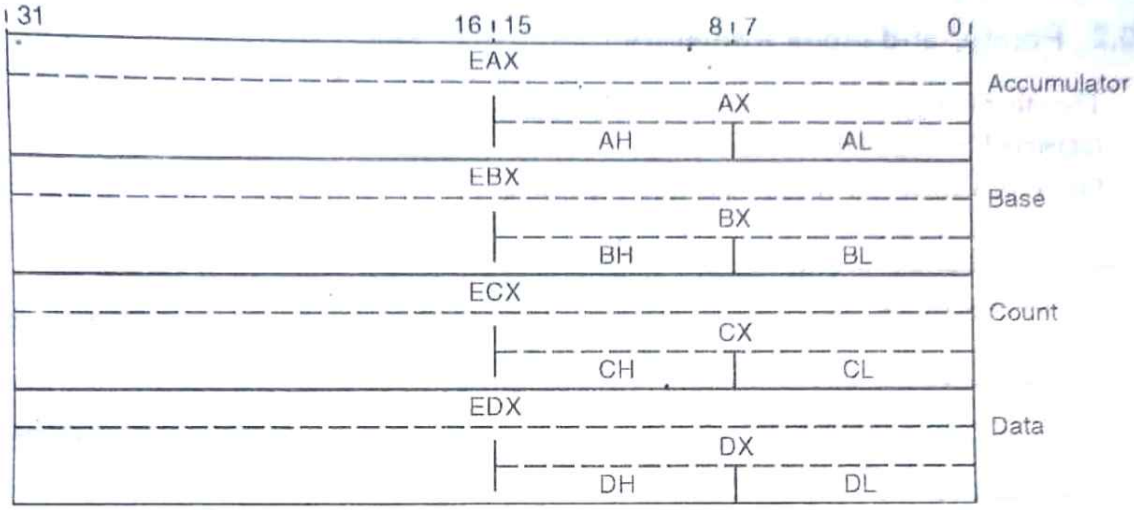


Fig. 1.30.2 : General purpose registers

- These registers are used for the temporary storage of data that is frequently used.
- The advantage of storing the data in internal register is that during the program execution, the data can be accessed must faster.
- The 80386DX has eight general purpose registers EAX, EBX, ECX, EDX, ESP, EBP, ESI and EDI.
- As earlier in 8086 the programmer can refer to the LSB of register A as AL and MSB of register us AH. The complete word (16 bit) as AX. The new register EAX (extended AX) refers to 32 bit.
- There is no way in the 80386DX to access the upper word of an extended register EAX i.e. bits 16 through 31.
- Any of the general purpose registers can be used as source or destination of an operand during the arithmetic or logical operations.

The primary functions of the general purpose registers include :

AX (accumulator) :

Often holds the temporary result after an arithmetic and logical operation.

BX (base) :

Often holds the base (offset) address of data located in the memory and also base address of a table of data referenced by translate instruction (XLAT).

CX (count) :

Contains the count for certain instructions such as shift count (CL) for shift and rotate instructions. The count in CX register is also used for Loop instructions.

DX (data) :

- It holds the MSB of product after 16 or 32 bit multiplication. The MSB of the dividend before division is also stored in DX.
- It also holds the I/O port addresses for variable port I/O instructions.

1.30.2 Pointer and Index Registers :

- The other four general purpose registers are the two pointer registers ESP and EBP and two index registers ESI and EDI.
- Fig. 1.30.3 shows the pointer and index registers.

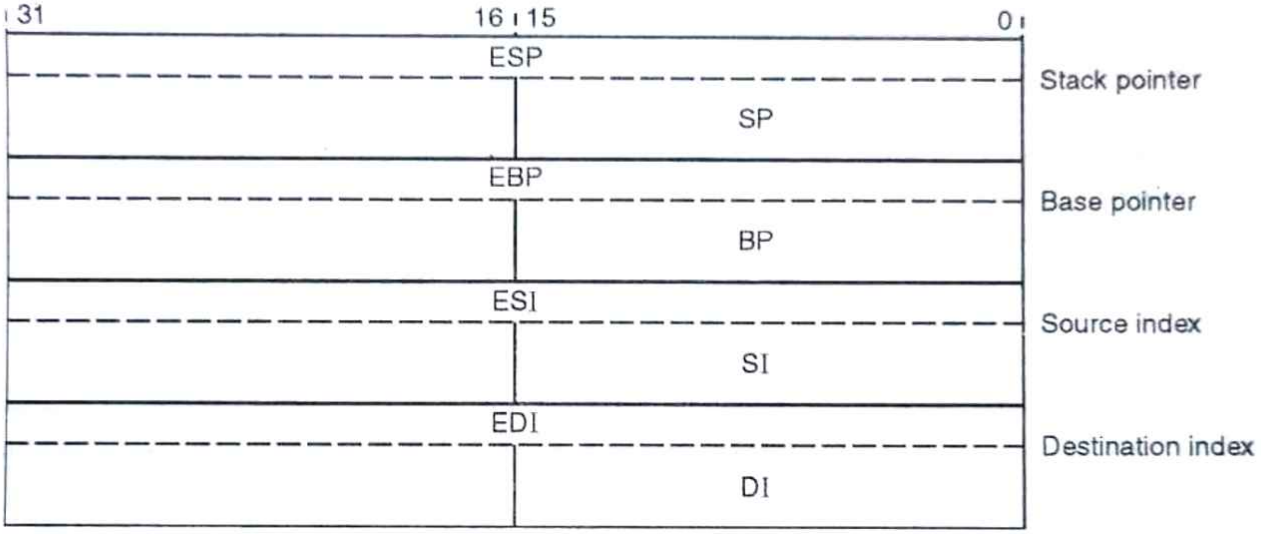


Fig. 1.30.3 : Pointer and index registers

- They are mainly used to hold the 16 bit offset of the data word relative to the segment registers.

Source Index (ESI) Register :

- This register is used for holding the offset of data word in the data segment.
- These offset values can be incremented or decremented when accessing a block of data.

Base Pointer (EBP) Register :

- It can be used to access storage locations within the stack segment of memory.
- If BP is used as an index instead of base, it serves as offset with respect to any segment register.

Stack Pointer (ESP) register :

- The SP is used to holding an offset relative to the stack segment.
- It used for stack related operations i.e. PUSH, POP, CALL, RET.

Destination Index Register (EDI) :

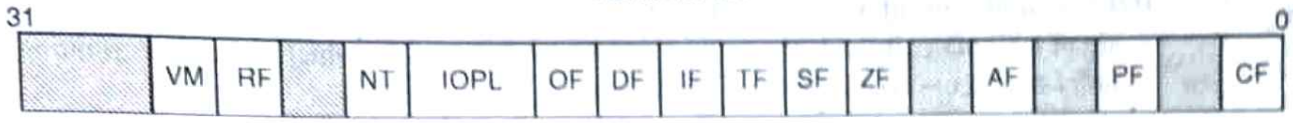
- It is used for holding the offset of data word in the data segment or extra segment.
- The extended source index register (ESI) and the extended destination index register (EDI) are used for string related operations.

eg : If we want to move a block of data from memory to memory then SI can be used to point the source memory address and DI can be used to point the destination memory address.

1.30.3 Flag Register (EFLAG) :

➤➤➤ [University Exam – Dec. 2003 !!!]

Table 1.30.1



Shaded portion : Reserved

- Flag register is a part of the EU.
- It consisting of 32 flip flops.
- A flag is a flip flop. It indicates some condition produced by the execution of an instruction.

eg : If zero flag (ZF) is set, if the result of execution of an instruction is zero.

- A flag can control certain operations of the CPU.

Table 1.30.1 shows the 32 bit extended flag register of the 80386DX.

These flags can be categorized as :

(1) Status flags : These flags are also called as the conditional flags. These flags indicate some condition produced after the execution of an instruction.

The six conditional flags are

(i) Carry Flag (CF) : This bit is set by arithmetic instructions that generate a carry or borrow.

(ii) Parity flag (PF) : This bit is set by instructions, if the LSB of the 8 bits of destination operand contain an even number of 1's.

(iii) Auxiliary carry flag (AF) : This bit is set by the instructions if there is a carry or borrow after nibble addition or subtraction. The programmer cannot directly access this bit.

(iv) Zero Flag (ZF) : This bit is set to 1, if the result of an operation is zero.

(v) Sign Flag (SF) : The signed numbers can be representated with combination of sign and magnitude. The MSB of number, indicates sign of number. The MS?B of result is copied into SF. If SF = 1 the result of execution of an instruction is negative number.

(vi) Overflow flag (OF) : This flag is set, to indicate that the signed result is out of range. If result is not out of range, OF remains reset.

(2) Control flags : These flags are used to control certain operations of the processor.

The three control flags are

(i) Trap flag (TF) : When the trap flag is set, it enables trapping through the on chip debugging features. The 80386DX goes into the single stepping mode. Here the 80386DX executes one instruction at a time.

(ii) Interrupt Flag (IF) : This bit controls the operation of INTR (interrupt request) input pin of the processor. If IF = 1, the INTR pin is enabled and if IF = 0, the INTR pin is disabled. The state of this flag is controlled by STI and CLI instructions.

(iii) Direction Flag (DF) : It controls the direction of string operations. If DF = 1, then the SI and DI registers are decremented by 1 and if the DF = 0 then SI and DI registers are incremented by 1 after each operation in the string instructions.

(3) **System Flags :** These flags reflect the current status of the machine. They are normally used by the operating system rather than different applications programs.

The system flags are

(i) **IOPL (Input/Output Priviledge Level) Flags :** These 2 bits are used in the protected mode of 80386DX. It holds the priviledge level from 0 to 3, at which the code is running in order to execute any I/O related instructions.

(ii) **Nested Task flag (NT) :** This is also used in protected mode. This bit is set, when one tasks invokes another task.

(iii) **Resume flag (RF) :** This bit allows selective masking of some exceptions, while a code is debugged.

(iv) **VM (Virtual 8086 mode flag) :** This flag indicates the operating mode of the 80386. This flag is set, when the 80386DX switches from the protected mode to virtual 8086 mode.

1.30.4 Segment Registers :

- The 80386 has a 1M-byte address space in the real mode.
- The entire memory cannot be accessed at a time, hence the memory is accessed into six memory blocks called segments.
- A segment is an independent block of memory consisting of 64 KB memory.
- These segments can be addressed by 16 bit registers.

They are

1. The code segment (CS) register.
2. The stack segment (SS) register.
3. The data segment (DS) register.
4. The extra segment (ES) register.
5. The general data segment (GS) register.
6. The general data segment (FS) register.

- The purpose of all these segment registers can be explained as follows.
- The memory accessible is 1 MB. i.e. The number of address lines are 20. So the 80386DX. BIU will send out a 20 bit address to access one of 1, 048, 576 or 1 MB memory locations.
- The six segment registers actually contain the upper 16 bits of the starting addresses of the six memory segments of 64 KB each, with which the 80386 is working at that instant of time.
- The concept can be understood by looking into the Fig. 1.30.4.
- The segment overlapping usually takes place for small programs which do not need 64 KB in each segment.
- The segments can be adjacent, disjoint, partially overlapping or fully overlapping.
- In the user's program there can be N number of segments. But 80386 can deal with only six of them at a given time, because it has six segment registers.
- Whenever the segment orientation is to be changed, we have to change the base addresses and load the upper 16 bits into corresponding segment registers.

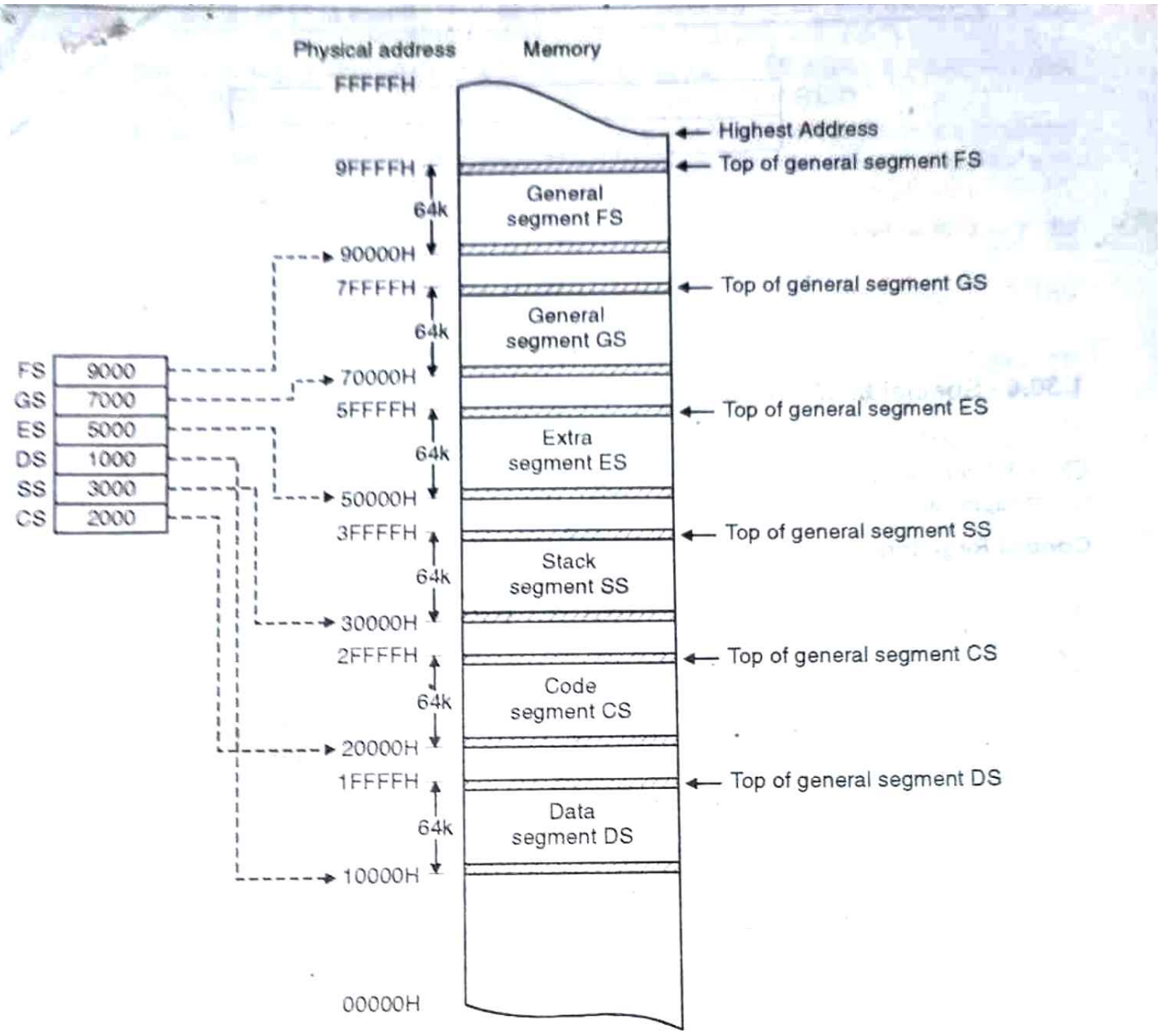


Fig. 1.30.4 : Segment Registers within the memory of 80386

1.30.5 System Address Registers :

- The system address registers are associated with the protected mode operation of the 80386DX.
- There are four system address registers.

(i) **Task register (TR) :** It points to the Task state stable.

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

(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. 1.30.5 shows the protected mode registers.

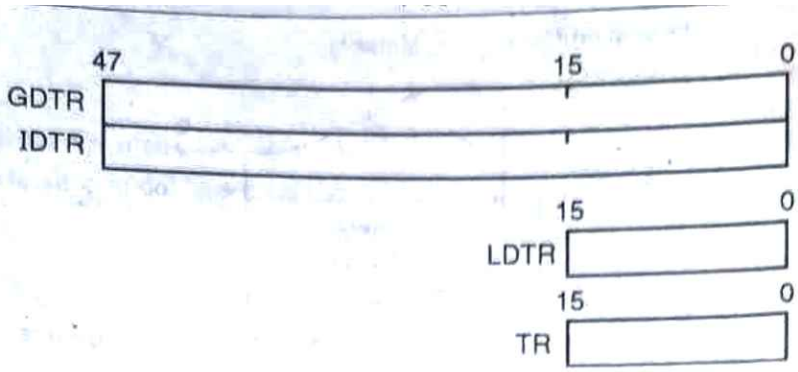


Fig. 1.30.5 : Protected Mode Registers

1.30.6 Special 80386 Registers :

80386DX includes a new set of registers as control, test and debug registers. The control registers CR0-CR3 control various features. DR0-DR7 facilitates debugging and registers TR6 and TR7 are used to test paging and caching.

Control Registers :

➤➤➤ **[University Exam – May 2002, May 2004 !!!]**

Fig. 1.30.6 shows the control register structure of the 80386.

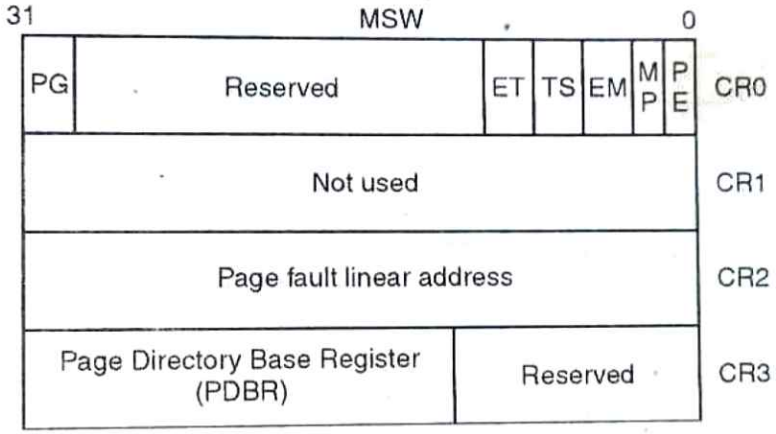


Fig. 1.30.6 : Control registers

- There are four control registers : CR0, CR1, CR2 and CR3.
- These registers define the machine status which affects all the tasks in the systems.
- **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.

- **Control register CR1 :** It is not used in the 80386DX, but is reserved for other processors.
- **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.
- **Control register CR3 :** It holds the base address of the page directory. It is also called as the **Page Directory Base Register (PDBR).**

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. 1.30.7 shows the debug registers.

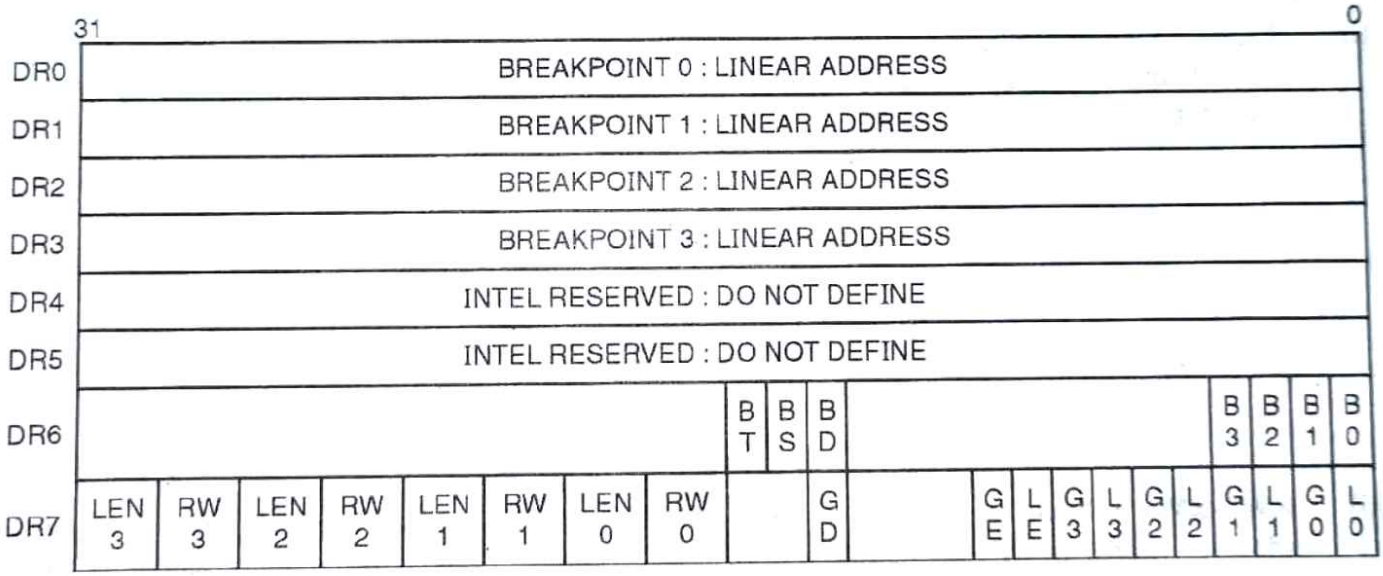


Fig. 1.30.7 : Debug Registers of 80386

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.

Debug Registers 4 and 5 :

These registers are reserved by INTEL.

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.

eg : 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.

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 1.30.2 shows the different sizes of the breakpoints.

Table 1.30.2

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 1.30.3.

Table 1.30.3

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 pit 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.

(v) **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.

Test Registers :

Two test registers (TR6-TR7) are current 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 holds most commonly used page table address translations and reduces number of memory reads required for looking up page translation address, in the page translation table.
- The TLB holds the most common 32 entries from the page table and is tested with the TR6 and TR7 registers.
- TR6 holds the tag field (linear address) of the TLB, and TR7 holds the physical address of the TLB.

Fig. 1.30.8 shows the bit pattern of the TR6 and TR7.

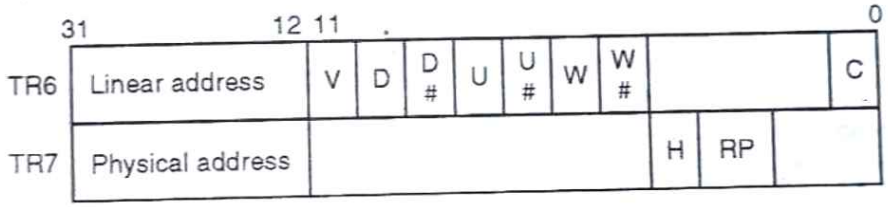


Fig. 1.30.8 : Test Registers

The bits found in TR6 and TR7 indicate the following conditions.

(i) **C :** This is a command bit. If this bit is 0, it indicates a write to the TLB is to be performed. If this bit is set, it indicates immediate look up for the TLB.

(ii) **W :** It indicates that the area accessed by the TLB entry is writable.

(iii) **W # or $\overline{W}$:** It indicates that the area accessed by TLB entry is not writable.

(iv) **U # :** It indicates that bit is not a user bit.

(v) **U :** It indicates that it is a user bit.

(vi) **D # :** It indicates that the entry in the TLB is not dirty.

(vii) **D :** It indicates that entry in the TLB is invalid or dirty.

(viii) **V :** It indicates that the entry in the TLB is valid entry.

(ix) **Linear address :** (Bits 12 to 31) of the TR6 serve as the upper 20 bits of linear address to be used for TLB reference.

(x) **RP (Replacement pointer) :** This field selects block of TLB to be written.

(xi) **H (Pointer Location) :** If this bit is set, RP field determines which is cache set to write to. If this bit is cleared the set is determined with an internal algorithm.

(xii) **Physical address :** This field contains the physical address to be written into the TLB or the results of a valid TLB.

1.31 Pin Diagram of the 80386 Microprocessor :

The 80386DX signals are divided into four groups

(i) Memory I/O interface signals.
(ii) Interrupt interface signals.
(iii) DMA interface signals.
(iv) Coprocessor interface signals.

Fig. 1.31.1 shows pin diagram.

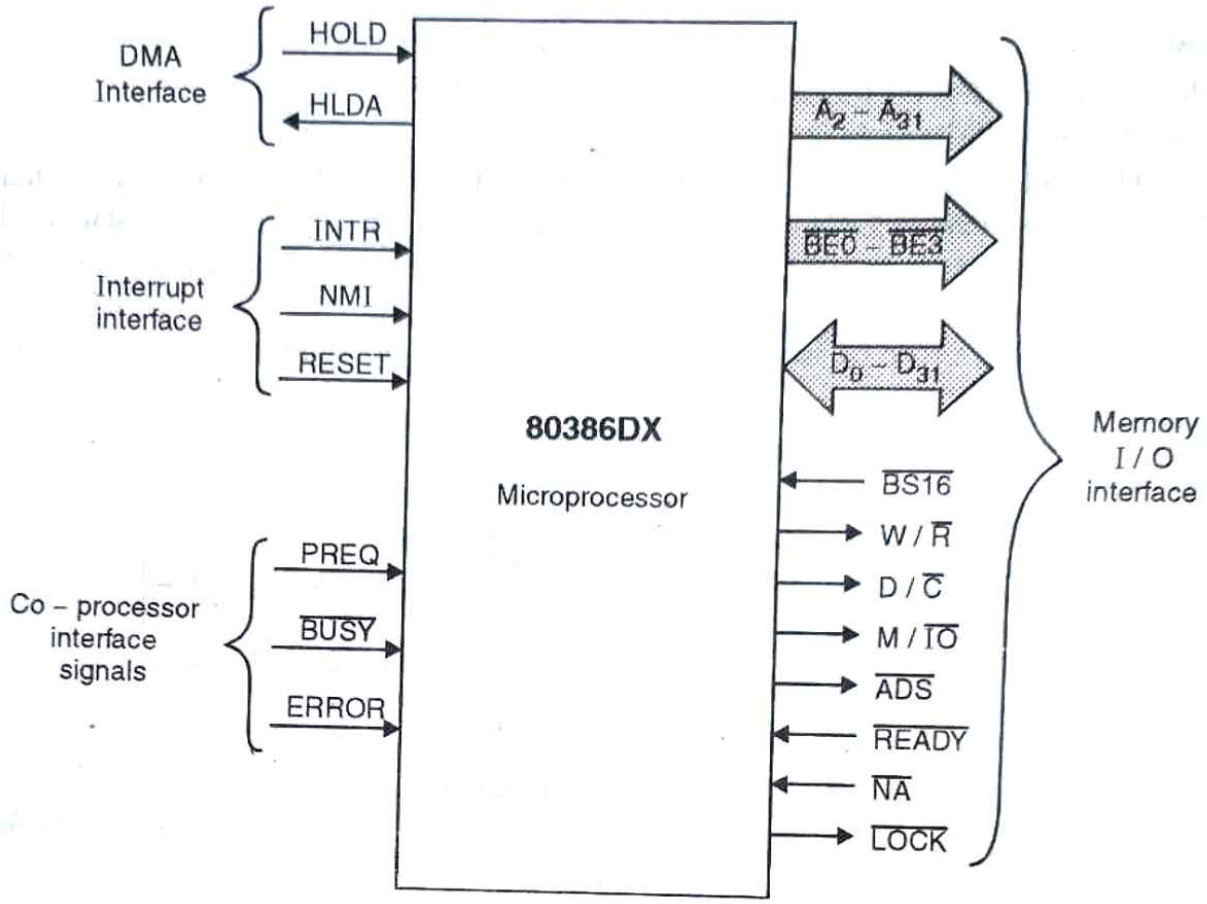


Fig. 1.31.1 : Pin diagram of the 80386DX

1.31.1 Memory I/O Interface Signals :

It includes data bus, separate address bus, five status signals and three bus control signals.

Data Bus :

It consists of 32 pins (D_{31}-D_0). These lines can be used to transfer 8, 16, 24 or 32 bit data at one time.

Address Bus :

The address generated by 80386DX are of 32 bit. The higher 30 bits of address are sent on the A_{31}-A_2. The address lines A_1 and A_0 are encoded in the bus enable ($\overline{BE3}$-$\overline{BE0}$) to select any or all of four bytes in a 32 bit wide memory location.

Bus Enable Signals ($\overline{BE3}$-$\overline{BE0}$) :

➤➤➤ **[University Exam – May 2002, Dec. 2002, May 2003 !!!]**

These signals 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.

Status Signals :

They decide the type of bus cycle to be performed. The bus status signals are

(i) Address data strobe ($\overline{ADS}$)

(ii) Write/Read ($W/\overline{R}$)

(iii) Memory/I/O ($M/\overline{IO}$)

(iv) Data control ($\overline{D}/\overline{C}$)

(v) LOCK ($\overline{LOCK}$)

(i) **Address data strobe ($\overline{ADS}$) :** 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) **Write/Read ($W/\overline{R}$) :** It decides the type of operation $M/\overline{IO}$ that will occur during the bus cycle. When a logic 0 is there at this pin, it indicates that the data is to be read from the memory or I/O. A logic 1 on this pin indicates that data is to be written on the memory or the I/O device.

(iii) **Memory/I/O ($M/\overline{IO}$) :** It identifies whether the current bus cycle is a memory cycle or an I/O cycle. Memory/I/O selects a memory device when a logic 1 is present at this pin or it selects an I/O device when logic 0. During the I/O operation, the address bus contains a 16 bit I/O address on address lines A_{15}-A_2.

(iv) **Data/Control ($D/\overline{C}$) :** This signal indicates whether the current bus cycle is a data cycle or a control cycle. This signal indicates that the data bus contains data for or from memory or I/O when this signal is at logic 1. If $D/\overline{C}$ is at logic 0, the microprocessor is halted or executes an interrupt acknowledge signal.

(v) **LOCK ($\overline{LOCK}$) :** This signal is used in multiprocessor systems. This signal ensures that the 80386DX, has uninterrupted control of the system bus and the shared resources. It is used when DMA accesses are done.

Table 1.31.1 shows the possible bus cycles and their definition signals.

Table 1.31.1

$M/\overline{IO}$	$D/\overline{C}$	$W/\overline{R}$	Type of Bus Cycle
0	0	0	Interrupt Acknowledge
0	0	1	Bus Idle
0	1	0	I/O data read
0	1	1	I/O data write
1	0	0	Halt/Shut down
1	1	0	Memory data read
1	1	1	Memory data write

Bus Control Signals :

They allow the external logic to control the bus cycles. They are

(i) $\overline{READY}$

(ii) Next address ($\overline{NA}$)

(iii) Bus Size 16 ($\overline{BS16}$)

(i) **READY :** ➤➤➤ [University Exam – Dec. 200

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

(ii) **Next Address ($\overline{NA}$) :** ➤➤➤ [University Exam – Dec. 2002, Dec. 2003 !!!]

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

(iii) **Bus Size 16 ($\overline{BS16}$) :** ➤➤➤ [University Exam – Dec. 2002, Dec. 2003 !!!]

It selects either a 32 bit data bus if it is at logic 1 or 16 bit data bus if it is at logic 0.

1.31.2 Interrupt Interface Signals (INTR) :

The three interrupt interface signals are

(i) Interrupt request (INTR)

(ii) Non-maskable interrupt (NMI)

(iii) RESET

(i) Interrupt request (INTR) : An interrupt request is used by the external circuitry to request an interrupt.

(ii) 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.

(iii) Reset : It initializes the 80386, cause it to begin executing software at memory location FFFFFFF0H. The 80386 is reset to the real mode.

1.31.3 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.

1.31.4 Coprocessor Interface Signals :

➤➤➤ [University Exam – May 2001 !!!]

These signals are required to interface the 80386DX with a coprocessor 80287 or 80387. The signals used for coprocessor interface are

(i) $\overline{BUSY}$

(ii) Error ($\overline{ERROR}$)

(iii) Coprocessor Request ($\overline{PEREQ}$)

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.

(ii) **Error ($\overline{\text{ERROR}}$) :**

➤➤➤ [University Exam – Dec. 2002 !!!]

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

(iii) **Coprocessor Request ($\overline{\text{PEREQ}}$) :**

➤➤➤ [University Exam – Dec. 2002 !!!]

The co-processor can not transfer data over the data bus by itself. Whenever the co-processor needs to read or write data from the memory, it indicates 80386 to initiate data transfer by making PEREQ signal high.

1.32 Bus Cycles :

➤➤➤ [University Exam – Dec. 2002, May 2003, Dec. 2003 !!!]

The operations of the 80386DX both internal and external bus operations are synchronized by the clock signal. There are seven types of bus cycles/operations.

(1) Interrupt acknowledge
(2) Memory read
(3) Memory write.
(4) I/O read
(5) I/O write.
(6) Instruction fetch.
(7) Halt or Shut Down.

1.32.1 Bus Cycle :

A bus cycle is the activity performed by the microprocessor in order to access information from memory or I/O devices. The 80386DX can performed bus cycles with either two types of timing :

(i) Non-pipelined bus cycles or

(ii) Pipelined bus cycles.

Before looking at the bus cycles of the 80386DX, let us see the relationship between the timing of 80386DX's CLK2 input and its bus cycle states. The shortest time unit of bus activity is called as *bus state*. A bus state 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. 1.32.1 shows this relationship.

CLK2 provides the fundamental timing for the Intel 80386DX. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock consist 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 equals 40 MHz and each clock cycle has a duration of 25 ns. In Fig. 1.32.1 the two phases ($\phi 1$ and $\phi 2$) of the processor cycle are identified as one processor clock period. Therefore, the processor clock period is 50 ns (minimum).

The clock signal 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.

1.32.7 Halt/Shutdown Cycle :

➤➤➤ [University Exam – May 2000, Dec. 2000 !!!]

In response to a HLT, the 80386DX goes into a halt condition. This condition occurs when a double fault is being processed. A protection fault is being found. This condition is retained until

(1) NMI goes high (2) RESET goes high

The Halt / Shutdown cycle is activated by initiating the $\overline{ADS}$ signal and the bus status signals. The M/$\overline{IO}$. W/$\overline{R}$ are driven high and D/$\overline{C}$ is driven low to indicate a halt cycle. All the addresses are set to logic 0. $\overline{BE0}$ is active for a shutdown condition and $\overline{BE2}$ is active for a halt condition. These signals are used by the external devices to identify the halt and shutdown cycles.

1.32.8 Bus Lock :

As more than one processor shares the system memory and I/O devices through a common system bus, extra logic must be added to ensure that only one processor has to access the system bus at a time. When one processor starts executing program, the other should not be serviced. Lock instruction can be used to achieve this. The other processor can be restricted by using a technique called as Mutual exclusion. Here, a binary flag called semaphore is stored in the shared memory to indicate where the shared memory is free to be accessed. Testing the semaphore and setting it is a critical operation and should be performed by one processor at a time. This technique can be used where multiprocessor systems are supported.

In case of 80386DX the XCHG instruction along with the LOCK prefix can be used to set or reset semaphore.

1.32.9 Control Input $\overline{BS16}$:

➤➤➤ [University Exam – May 2001, May 2002, May 2003, Dec. 2003, May 2004 !!!]

The $\overline{BS16}$ specifies the bus size for each bus cycle. This signal is sampled at the beginning of ϕ_2 of the T state in which the $\overline{ADS}$ is not active.

Whenever $\overline{BS16}$ = 0, the 80386 performs 16 bit data transfer. For this it uses the data lines D_{15}-D_0 rather than the entire 32 bit bus. D_{31}-D_0. For the data transfer two or three bus cycles are required. These transfers are 16 bit misaligned (odd addressed) transfers Fig. 1.32.11 shows the transfer. The $\overline{NA}$ and $\overline{BS16}$ signals are never low at the same time. This is done for address pipelining. Both the signals can never be low at the same time because they are sampled simultaneously by the DX 80386 processor.

Fig 1.32.12 shows data transfer for 32 bit i.e. when $\overline{BS16}$ = 1.

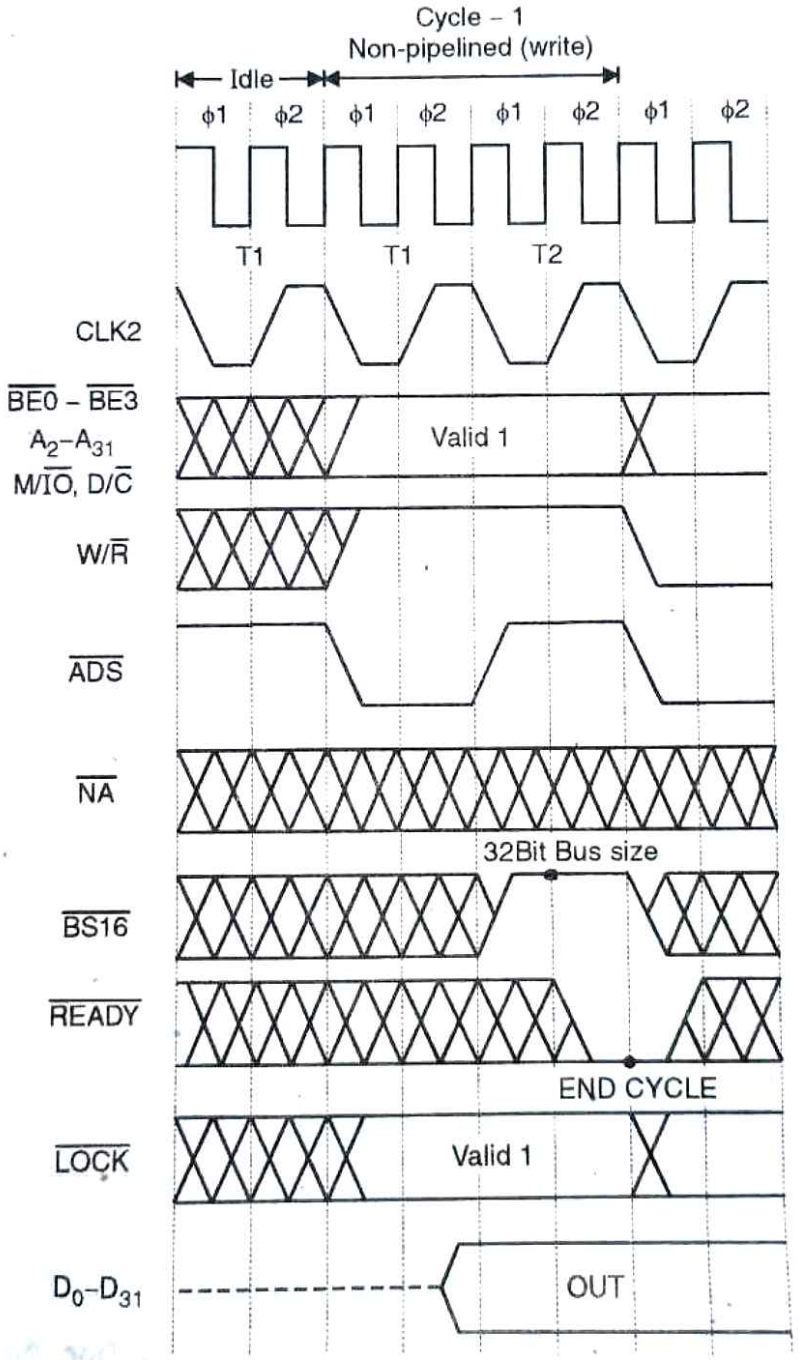


Fig. 1.32.11 : 16 Bit Transfer.

PART 2 : Programmers Model and Addressing Modes of 80386

3.5 Programmer's Model of 80386 :

➤➤➤ [University Exam – Dec. 2000, Dec. 2002 !!!]

The 80386 programmer's model is a picture of the registers available to the programmer. These registers are used to hold the numbers and addresses, as well as indicate status and acts as controls.

Data Registers :

(32 bits for each EAX, EBX, ECX and EDX).

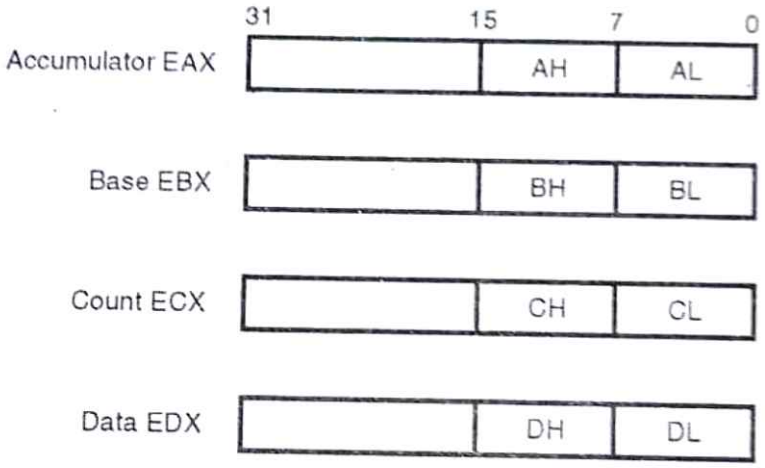


Fig. 3.30 : Data Registers of 80386 DX

- Data registers can be accessed on a double word basis by names EAX, EBX, ECX and EDX.
- Individual bytes of these registers can be accessed by names AH, AL, BH, BL, CH, CL, DH and DL.
- Individual LSB of words of these registers can be accessed by names AX, BX, CX and DX.
- However there is no convention, to access only the MSB of word (i.e. bits 15 to 31) of these registers.

Address registers :

- The segment registers are CS, DS, ES, SS, FS and GS. The offset can be stored in IP, SP, BX or BP, SI or DI.
- For specifying an address both a segment and an offset is required.
- The CS : IP pair gives the address of the next instruction to be executed in the program sequence.
- The SS : SP pair gives the address of the top of stack, a temporary storage area often automatically used by the computer.
- The SS : BP pair is used as a pointer into the stack.
- The DS : SI pair is used as source pointer and ES : DI pair is used as destination pointer.
- The GS : DI or FS : DI pair can also used as destination pointer.
- The DS : SI and ES : DI or GS : DI or FS : DI are used as general purpose pointers for copying and data processing.

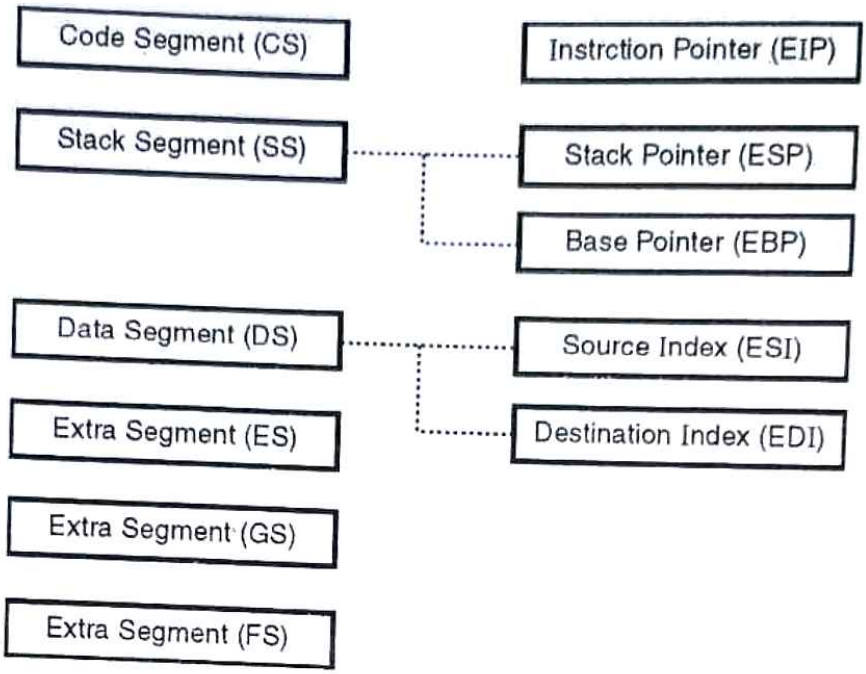
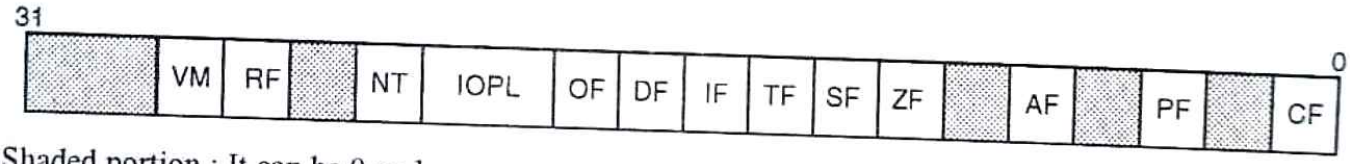


Fig. 3.31 : Address registers of the 80386

- The dotted lines above show common default couplings. These defaults can be overridden by a notation such as DS : [BP], where DS is used in place of default SS.
- The EBX data register not shown is also used as a pointer in the data segment (by default).
- The addressing scheme of the 80386 is rather inflexible.

Flag Register :



Shaded portion : It can be 0 or 1

CF : Carry out (error for unsigned numbers)
PF : Parity of last operation.
AF : Auxilary carry (used in BCD arithmetic).
SF : Sign bit from last operation.
TF : Trap flag (do single instruction)
IF : Interrupt Enable.
DF : Direction Flag
OF : Overflow (error for signed numbers)
IOPL : Input/Output Priviledge level.
RF : Resume flag
VM : Virtual Mode Flag.

NT – Nested Flag

3.6 Addressing Modes :

➤➤➤ **[University Exam – Dec. 2000 !!!]**

This will be out first step towards programming. Before we study the instruction set, it is necessary to know how 80386 accesses instruction operands in different ways.

- When 80386 executes an instruction, it performs specific function on data. The data is normally referred as operands. Operands may be contained in registers, within the instruction itself, in memory or in I/O ports.
- To access these different types of operands, 8086 has to address memory or I/O. The address memory or I/O. The address of memory and I/Os can be calculated in several different ways, normally referred as Addressing modes. These addressing modes greatly extend flexibility and convenience of the instruction set.

The addressing modes of the 80386 can be categorized as follows :

(1) Register addressing mode.
(2) Immediate addressing mode.
(3) Memory addressing mode.

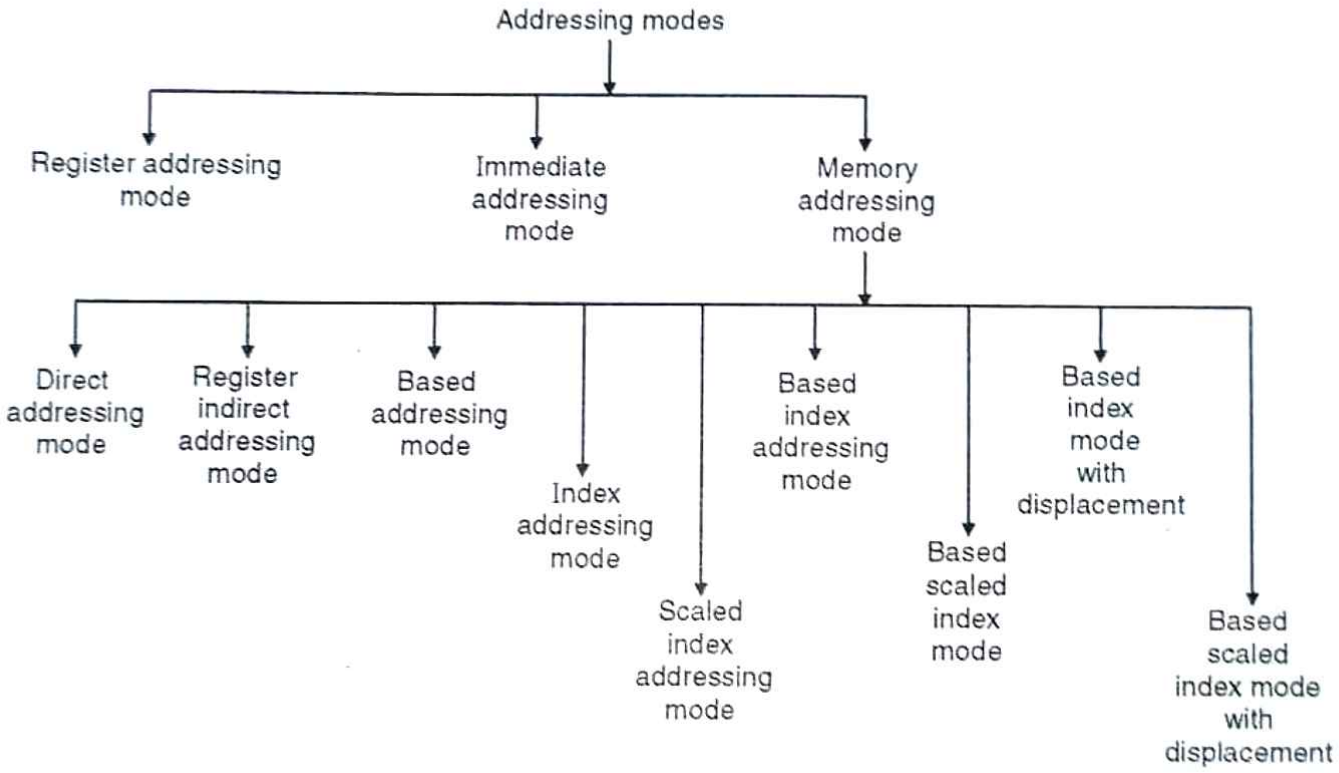


Fig. 3.32 : Addressing modes of 80386

3.6.1 Register Addressing Mode :

- In this mode of addressing, data is in the registers and the instruction specifies the particular registers.
- 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.
- 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 EAX register.

3.6.2 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. 3.33 shows the format of instruction encoded with immediate operand.

(a) Instruction encoded with an immediate operand **(b) Immediate addressing**

Fig. 3.33

- 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.
EA = 20305060H

3.6.3 Memory Addressing Modes :

In the previous addressing modes, EU was having a direct access to register and immediate data (operands). In this addressing, memory operands must be transferred to and from the CPU **over the BUS.** Whenever the Execution unit needs to read or write a memory operand, it must pass an offset value to the BIU.

Effective Address :

The offset calculated for a memory operand is called as the operand's effective address or EA.

It is an unsigned 32 bit number that expresses the operand's distance in bytes from the beginning of the segment in which it resides.

The effective address is computed by adding any combination of the following four components.

- **Displacement :** 8 or 32 bit immediate data following the instruction. 16 bit displacements can be used by inserting an address prefix before instruction.
- **Base :** The contents of any general purpose register can be used as base.
- **Index :** The contents of any general purpose register can be used an index register.

Note : ESP cannot be used as an index register.

The elements of an array or a string of characters can be accessed via the index register.

- **Scale :** The index register's contents can be multiplied (scaled) by a factor of 1, 2, 4 or 8. Scaled index mode is efficient for accessing arrays or structures.

Thus,

$$EA = \text{Base register} + (\text{Index Register} \times \text{Scaling factor}) + \text{Displacement}$$

Once, we get EA, we can calculate the PA (physical address) as,

PA = segment : Offset

⇓ ⇓

PA = Segment register : EA

$$PA = \begin{Bmatrix} CS \\ DS \\ ES \\ SS \\ FS \\ GS \end{Bmatrix} : \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{Bmatrix} + \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} \times \begin{Bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{Bmatrix} \right] + \{8, 16 \text{ or } 32 \text{ bit displacement}\}$$

The different memory addressing modes available are :

1. Direct addressing mode
2. Register indirect addressing mode
3. Based mode
4. Index mode
5. Scaled index mode
6. Based index mode
7. Based scaled index mode
8. Based index mode with displacement
9. Based scaled mode with displacement.

1. Direct addressing mode :

In this mode, the effective address is taken from the displacement field of the instruction. The effective address is used as 8, 16 or 32 bit displacement from the current value of the data segment register.

eg : MOV AX, [1897 H]

Here EA = 1897 H

PA = DS + 1897 H

2. Register indirect addressing mode :

In this mode a base or index register contains the operands effective address.

EA = {base register}/{index register}

PA = segment : offset

PA = segment register : EA

$$PA = \begin{Bmatrix} CS \\ DS \\ SS \\ ES \\ FS \\ GS \end{Bmatrix} : \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESP \\ ESI \\ EDI \end{Bmatrix} \text{ or } \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix}$$

eg : MOV EBX, [ECX]

EA = ECX

PA = DS + ECX

3. Based mode :

In this mode the contents of a base register are added to displacement, in order to obtain the operand's effective address.

EA = {base register} + {8, 16 or 32 bit displacement}

$$PA = \begin{Bmatrix} CS \\ DS \\ SS \\ ES \\ FS \\ GS \end{Bmatrix} : \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESP \\ ESI \\ EDI \end{Bmatrix} + \{8, 16 \text{ or } 32 \text{ bit displacement}\}$$

eg : MOV ESI, [EAX + 23H]

EA = EAX + 23H

PA = DS + EAX + 23H

4. Index mode :

In this addressing mode, an index register's contents are added to a displacement to obtain the operand's effective address.

$$PA = \begin{Bmatrix} CS \\ DS \\ ES \\ SS \\ FS \\ GS \end{Bmatrix} : \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} + \{8, 16 \text{ or } 32 \text{ bit displacement}\}$$

PA = DS + EDI + Offset of count

eg. : SUB COUNT [EDI], EAX

EA = EDI + Offset of COUNT

5. Scaled index mode :

In this mode the contents of an index register are multiplied by a scaling factor of (1, 2, 4 or 8) which is then added to the displacement to obtain the operand's effective address.

EA = (Index register × Scaling factor) + {8, 16 or 32 bit displacement}

$$PA = \begin{Bmatrix} CS \\ DS \\ ES \\ SS \\ FS \\ GS \end{Bmatrix} : \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} \times \begin{Bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{Bmatrix} \right] + \{8, 16 \text{ or } 32 \text{ bit displacement}\}$$

eg. : MOV [ESI * 8], ECX

EA = ESI * 8

PA = DS + (ESI * 8)

Based index mode :

In this mode the contents of a base register are added to the contents of an index register to compute the operand's effective address.

EA = {base register} + {index register}

PA = segment : EA

$$PA = \begin{Bmatrix} CS \\ DS \\ ES \\ SS \\ FS \\ GS \end{Bmatrix} : \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESP \\ ESI \\ EDI \end{Bmatrix} + \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix}$$

eg. : MOV ESI, [ECX] [EBX]

EA = ECX + EBX

PA = DS + ECX + EBX

7. Based scaled index mode :

In this mode, the contents of an index register are multiplied by a scaling factor and the result is added to base register to compute the operand's effective address.

EA = {base register} + {Index register × Scaling factor}

$$PA = \begin{Bmatrix} CS \\ DS \\ ES \\ SS \\ GS \\ FS \end{Bmatrix} : \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESP \\ ESI \\ EDI \end{Bmatrix} + \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} \times \begin{Bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{Bmatrix} \right]$$

eg. : MOV ECX, [EDI * 4] [ESP]

EA = (EDI * 4) + ESP

PA = DS + ESP + (EDI * 4)

8. Based index mode with displacement :

In this mode the operands effective address is obtained by adding the contents of base register and index register with a displacement.

EA = {base register} + {index register} + {8, 16 or 32 bit displacement}

$$PA = \begin{Bmatrix} CS \\ DS \\ GS \\ FS \\ ES \\ SS \end{Bmatrix} : \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{Bmatrix} + \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} + \{8, 16 \text{ or } 32 \text{ bit displacement}\}$$

e.g. : MOV [EBX] [EBP + 12345678 H], EDI

EA = EBX + EBP + 12345678 H

PA = DS + EBX + EBP + 12345678 H

9. Based scaled index mode with displacement :

In this mode, the operands effective address is computed by multiplying the index register with a scaling factor and result is added to base register and also a displacement is added.

EA = {base register} + {index register * scaling factor} + {8, 16 or 32 bit displacement}

$$PA = \begin{Bmatrix} CS \\ DS \\ ES \\ SS \\ GS \\ FS \end{Bmatrix} : \begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{Bmatrix} + \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} \times \begin{Bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{Bmatrix} \right] + \{8, 16 \text{ or } 32 \text{ bit displacement}\}$$

e.g. : MOV [EBX * 8] [ECX + 5678 H], ECX

EA = (EBX * 8) + ECX + 5678 H

PA = DS + (EBX * 8) + ECX + 5678 H

Review Questions

Q. 1 Draw the programmers model of 8086 microprocessor and label it neatly.

Q. 2 Write the addressing modes of 8086 with suitable example. Mention the addressing modes which are not supported by 8085 ? What do you mean by segment override prefix.

Q. 3 "Based indexed" addressing mode of 8086 is useful for an array element access explain.

Q. 4 Explain the following addressing modes :

a) Direct addressing

b) Immediate addressing

c) Register indirect addressing

d) Based addressing

e) Indexed addressing

Explain with suitable examples.

Q. 5 For the following instruction compute the address of memory operand for 8086 :

i) MOV AX, [BX]

ii) MOV AL, [BP + SI]

Assume CS : 0100 H, DS : 0200 H, SS : 0400 H, ES : 0030 H, BP : 0010 H,

BX : 0020 H,

SI : 0030 H, SP = 0040 H. Clearly show computations.

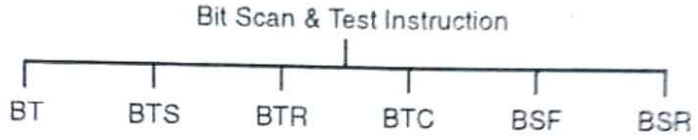
PART 2 : Instruction Set of 80386

The 80386 is compatible with the 8086/80186/80188 processors. Its instruction set includes all the 8086/80186/80286 instructions .These instructions can operate on 32 bit data words and 32 bit offsets . It includes of several new instructions which are discussed below.

1. Bit Scan and Test Instructions :

These instructions operate on one single bit within a register or memory location . If a register is specified , it can be either a 16 or 32 bit string .

The instructions in this group include:



BT(Bit Test)

Mnemonic : BT

Algorithm : CF = 0 if bit =0 OR if CF = 1 if bit =1

Operation : This instruction tests the status the specified bit in the instruction. The status of that bit is copied to the carry flag.

Example : BT EAX , 05H

This instruction copies the bit 5 of the EAX register to the carry flag.

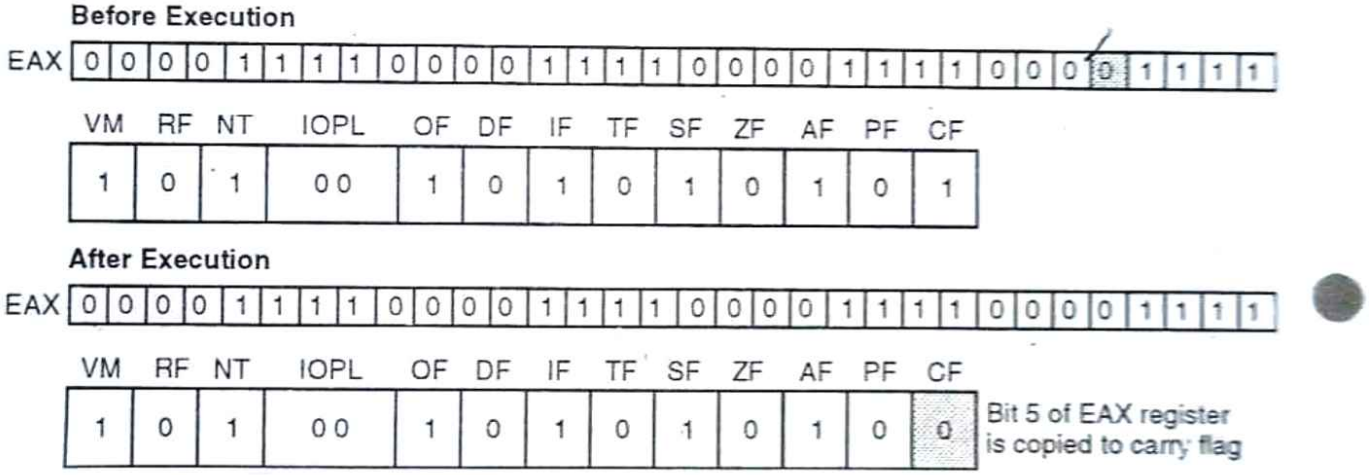


Fig. 4.93

BTS(Bit Test and Set)

Mnemonic : BTS

Algorithm : CF = 0 if bit =0 and bit =1 OR if CF = 1 if bit =1

Operation : This instruction tests the status the specified bit in the instruction . The status of that bit is copied to the carry flag. Then the specified bit is set.

Example : BTS EAX , 05

This instruction copies the bit 5 of the EAX register to the carry flag and then bit 5 of the EAX register is set.

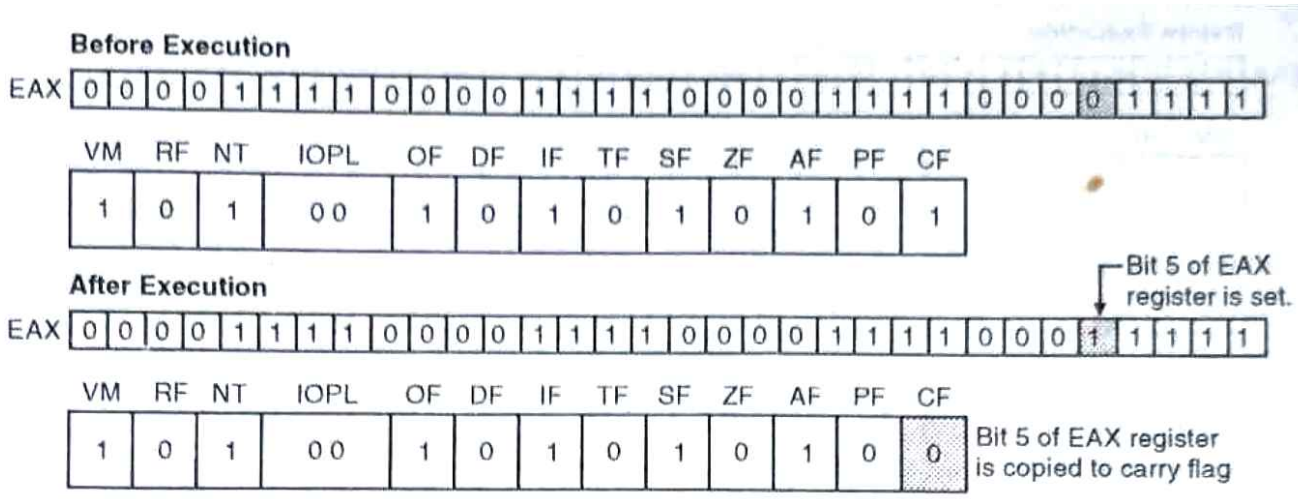


Fig. 4.94

BTR(Bit Test and Reset)

Mnemonic : BTR

Algorithm : CF = 0 if bit =0 OR if CF = 1 if bit =1and bit =0

Operation : This instruction tests the status the specified bit in the instruction . The status of that bit is copied to the carry flag. Then the specified bit is reset.

Example : BTR EAX , 05

This instruction copies the bit 5 of the EAX register to the carry flag and then bit 5 of the EAX register is reset i.e. bit5=0 .

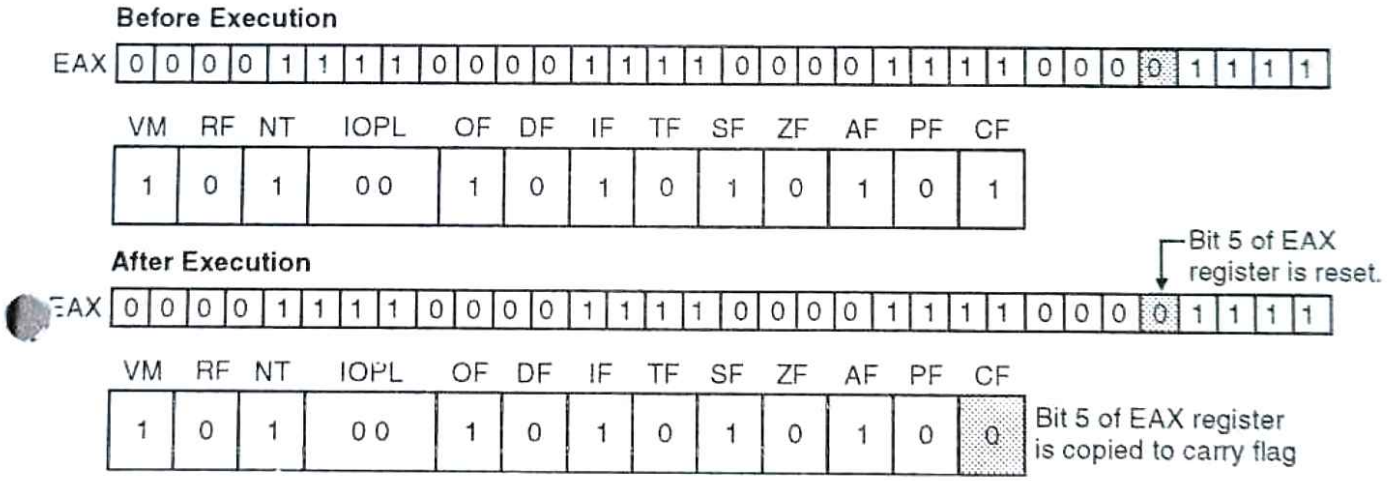


Fig. 4.95

BTC(Bit Test and Complement)

Mnemonic : BTC

Algorithm : CF = 0 if bit =0 then bit=1 OR if CF = 1 if bit =1then bit =0

Operation : This instruction tests the status the specified bit in the instruction . The status of that bit is copied to the carry flag. Then the specified bit is complemented.

Example : BTR EAX , 05

This instruction copies the bit 5 of the EAX register to the carry flag and then bit 5 of the EAX register is complemented i.e. bit5 = 0 then bit5=1 and reversely .

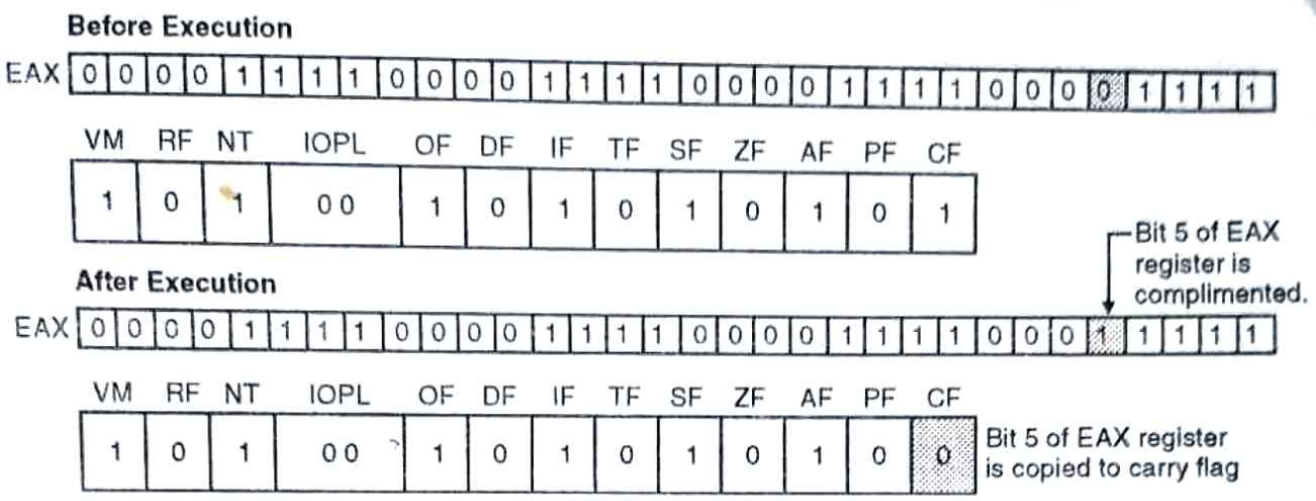


Fig. 4.96

BSF(Bit Scan Forward)

Mnemonic : BSF

Algorithm : If bits are scanned are zero then ZF = 1

else ZF=0

Operation : This instruction scans the bits in case of second word /double word operand starting with the least significant bit till a nonzero bit is found . If all the bits scanned are zero, then zero flag is set otherwise zero flag is reset to 0 . The destination operand is loaded with the bit index of the first set bit .

Example : BSF EAX , ESI

This instruction scans ESI from bit 0 till a nonzero bit i.e. till a first 1 is found., the first bit which is nonzero will be stored in the EAX register. If a 1 is not found in the ESI register then ZF = 1

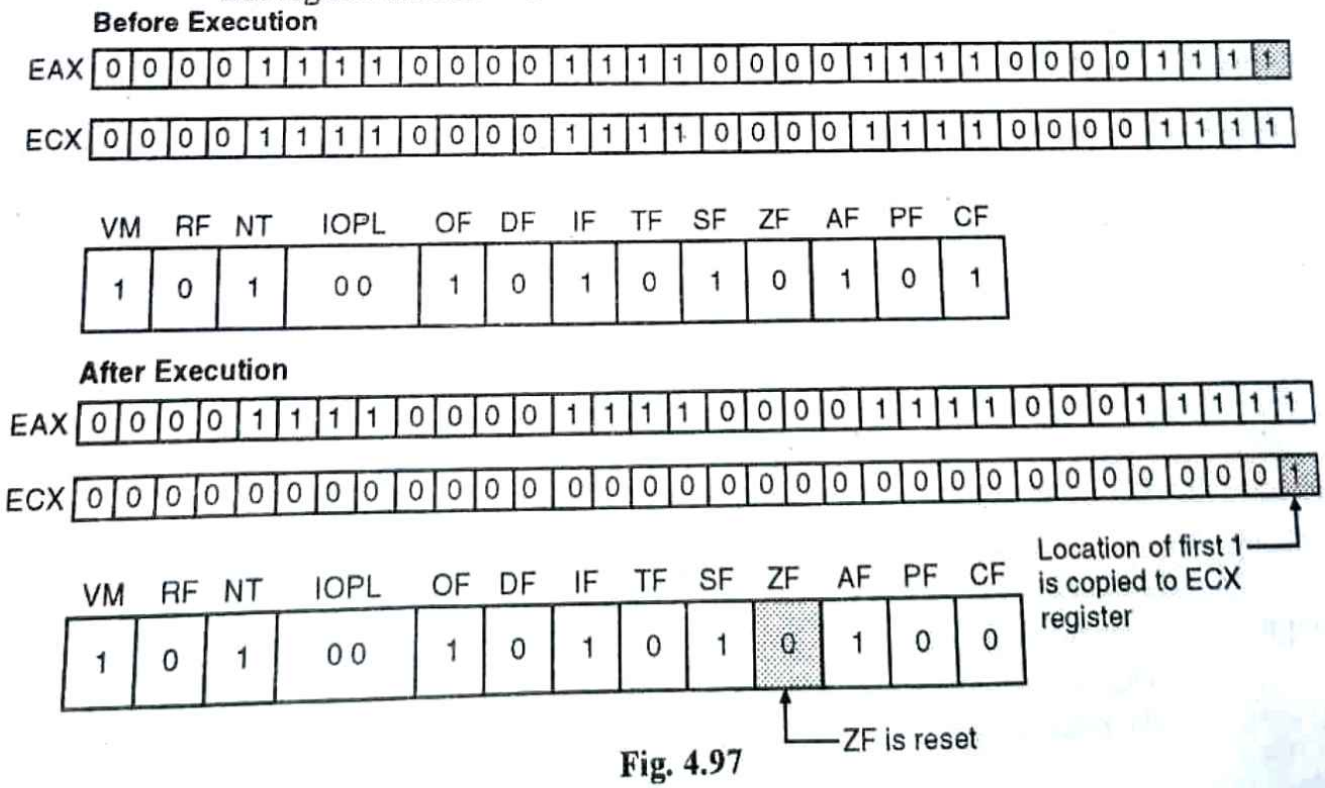


Fig. 4.97

BSR(Bit Scan Reverse)

Mnemonic : BSR

Algorithm : If bits are scanned are zero then ZF = 1 else ZF = 0

Operation : This instruction scans the bits in case of second word /double word operand starting with the most significant bit till a nonzero bit is found . If all the bits scanned are zero, then zero flag is set otherwise zero flag is reset to 0 . The destination operand is loaded with the bit index of the first set bit .

Example : BSF EAX , ESI

This instruction scans ESI from bit 31 till a nonzero bit i.e. till a first 1 is found, the first bit which is nonzero will be stored in the EAX register. If a 1 is not found in the ESI register then ZF=1

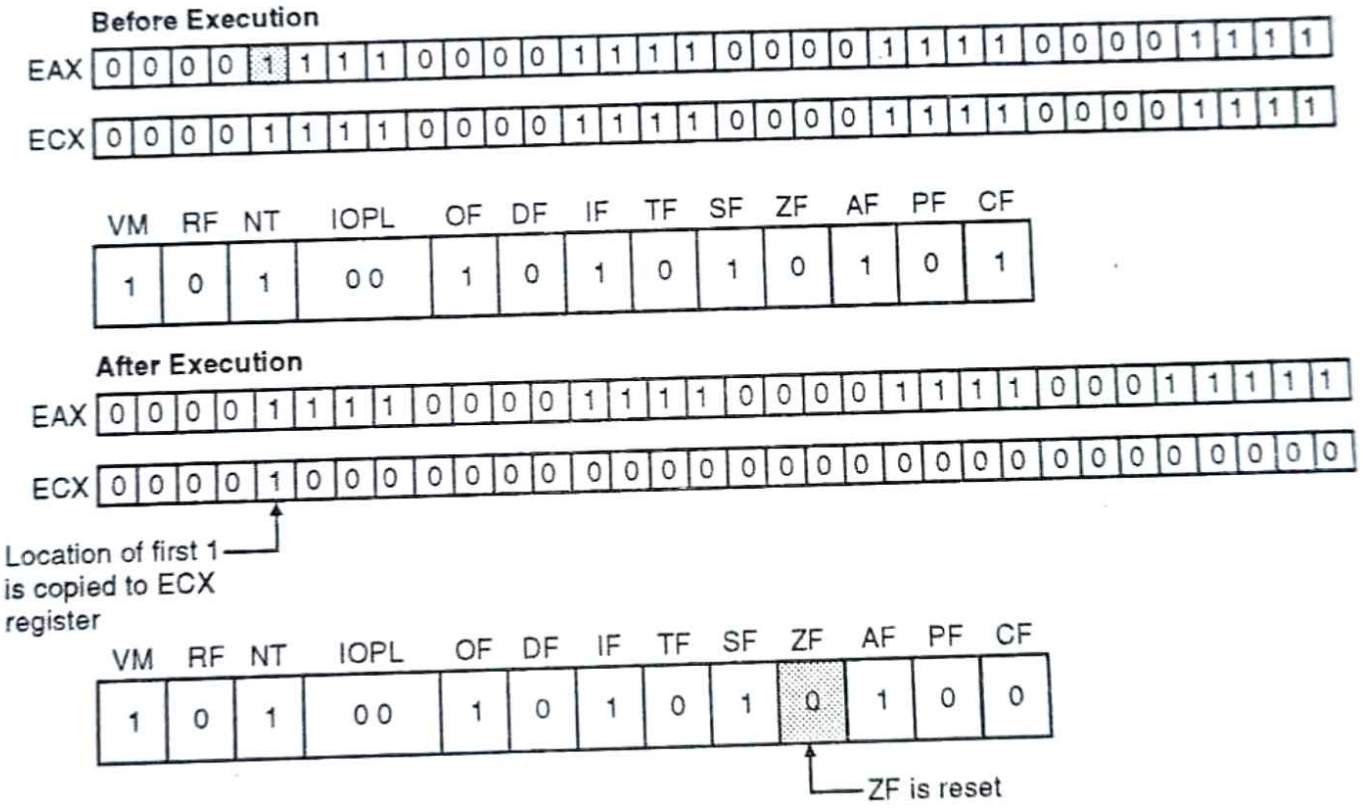
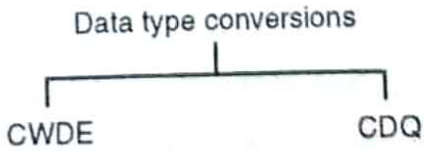


Fig. 4.98

2. Data Type Conversions :

These instructions operate for operand size conversion .

The instructions in this group include:



CWDE(Convert word to Double word)

Mnemonic : CWDE

Algorithm : If MSB bit of AX=1 then

- EAX = FFFFFFFF H

Else

- EAX = 00000000 H

Operation : This instruction copies the sign bit of AX to all the bits in the EAX register.

EAX is then a signed extension of AX register .

Example : AX=0F0F H

CWDE : This instruction will convert signed bit in AX to signed double word in the EAX register .

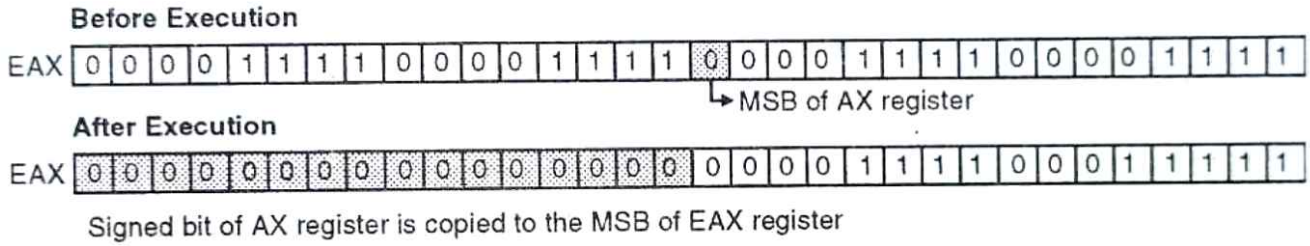


Fig. 4.99

CDQ(Convert Double word to Quad word)

Mnemonic : CDQ

Algorithm : If MSB bit of EAX=1 then

EDX = FFFFFFFF H

Else

EDX = 00000000 H

Operation : This instruction copies the sign bit of EAX to all the bits in the EDX register.

EDX is then a signed extension of EAX register .

Example : EAX= 0F0F0F0F H

```
CDQ        ; this instruction will convert signed bit in EAX to signed Quad word in
             the EDX: EAX register pair.
```

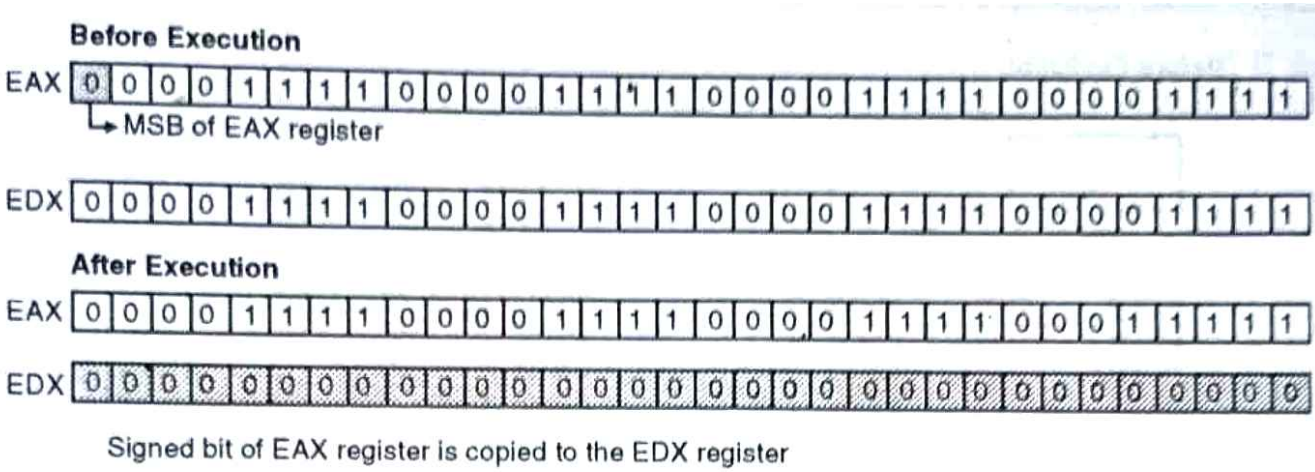
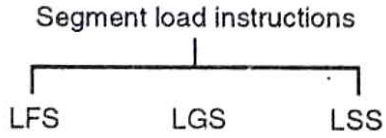


Fig. 4.100

3. Segment Load Instructions :

These instructions manipulate the addresses of the variables , rather than the contents or values of the variables . They are mainly used for list processing based variables and string instructions .

The instructions in this group include :



LFS(Load pointer with FS)
(Load register and FS with words from the memory)

Mnemonic : LFS register ,source

Algorithm : Register = First word , FS= second word

Operation : Register ←Source, FS ←(Source +2)

This instruction is a 2 byte instruction. FS is used as a segment register for memory.

This instruction copies a word from two memory locations into register specified in the instruction . It then copies a word from next two memory locations into the FS register.

Example : LFS BX , [1100]

This instruction copies a word from the memory locations at offset of 1100 H and 1101H into the BX register and copies a word from next two memory locations with the offset of 1102 H and 1103 H into the FS register.

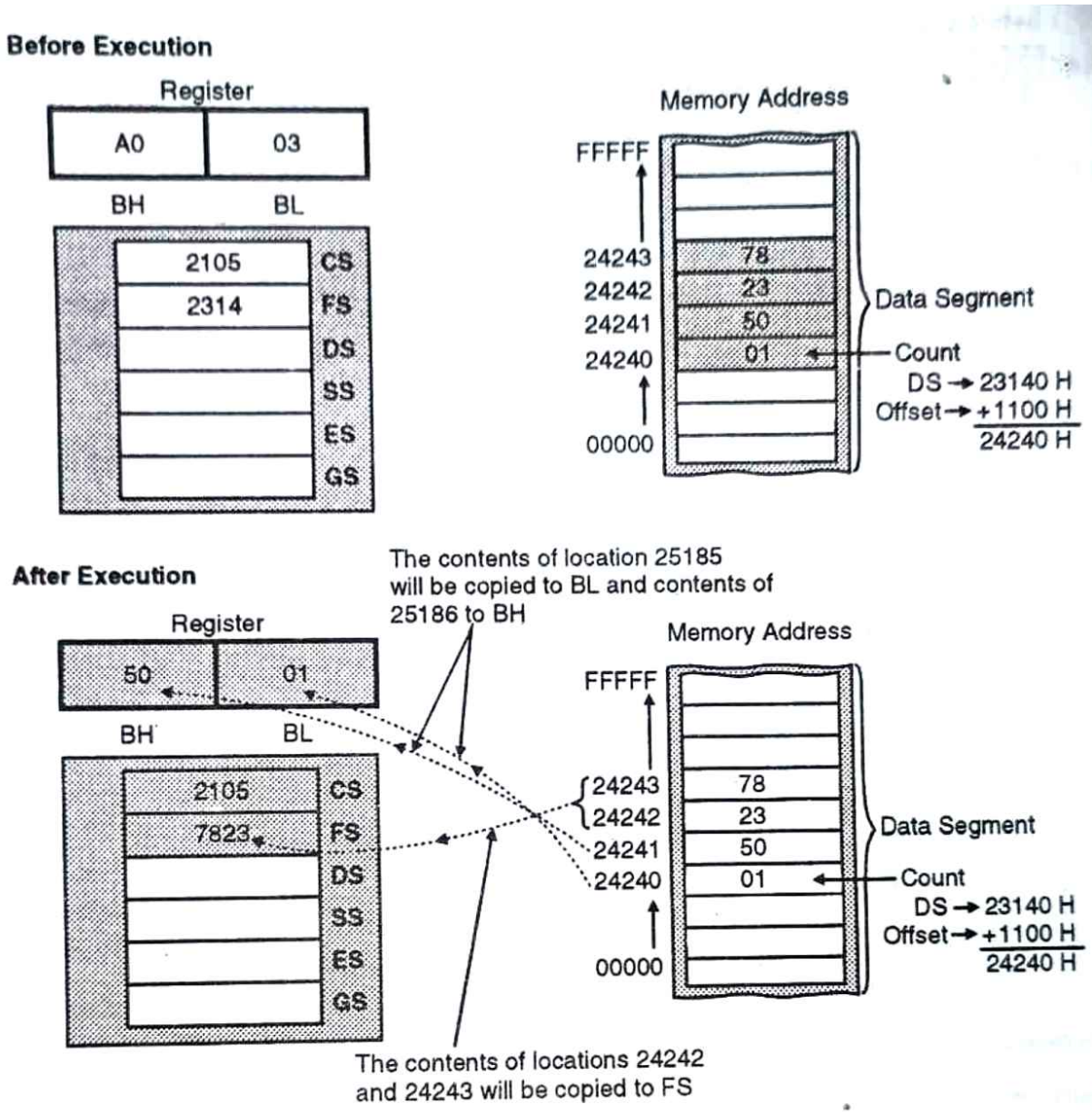


Fig. 4.101

LGS(Load pointer with GS)
(Load register and GS with words from the memory)

Mnemonic : LGS register ,source

Algorithm : Register = First word , FS= second word

Operation : Register ←Source, GS ←(Source +2)

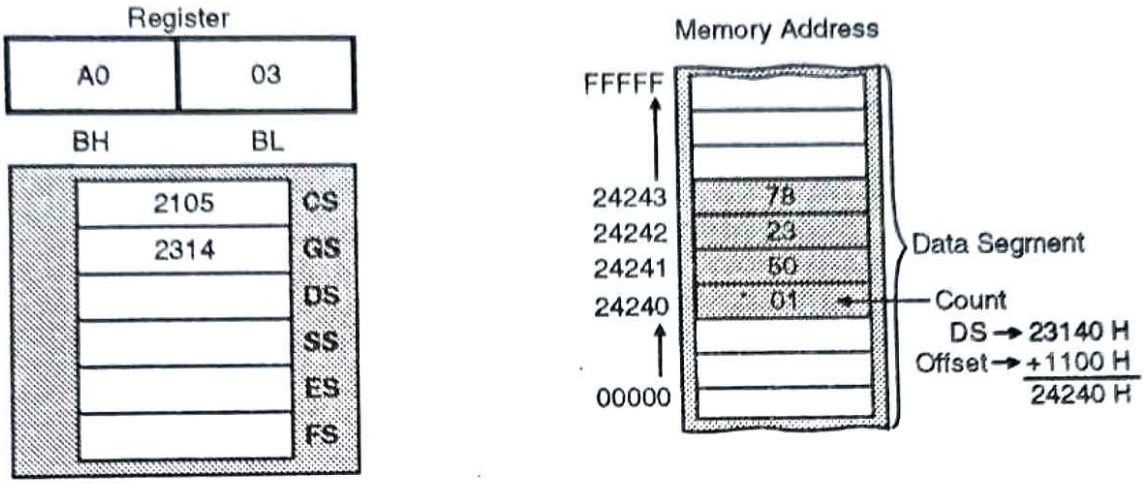
This instruction is a 2 byte instruction. FS is used as a segment register for memory.

This instruction copies a word from two memory locations into register specified in the instruction . It then copies a word from next two memory locations into the GS register.

Example : LGS BX , [1100]

This instruction copies a word from the memory locations at offset of 1100 H and 1101H into the BX register and copies a word from next two memory locations with the offset of 1102 H and 1103 H into the GS register.

Before Execution



After Execution

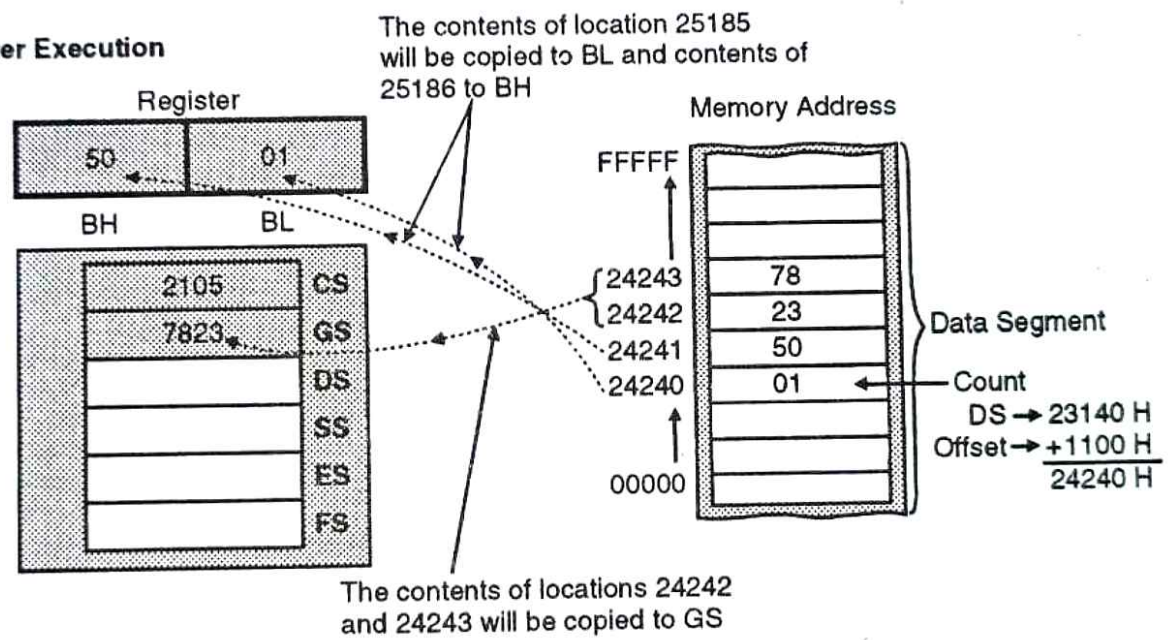


Fig. 4.102

LSS(Load pointer with SS)
(Load register and GS with words from the memory)

Mnemonic : LSS register ,source

Algorithm : Register = First word , FS= second word

Operation : Register ←Source, SS ←(Source +2)

This instruction is a 2 byte instruction. FS is used as a segment register for memory.

This instruction copies a word from two memory locations into register specified in the instruction . It then copies a word from next two memory locations into the SS register.

Example : LSS BX , [1100]

This instruction copies a word from the memory locations at offset of 1100 H and 1101H into the BX register and copies a word from next two memory locations with the offset of 1102 H and 1103 H into the SS register.

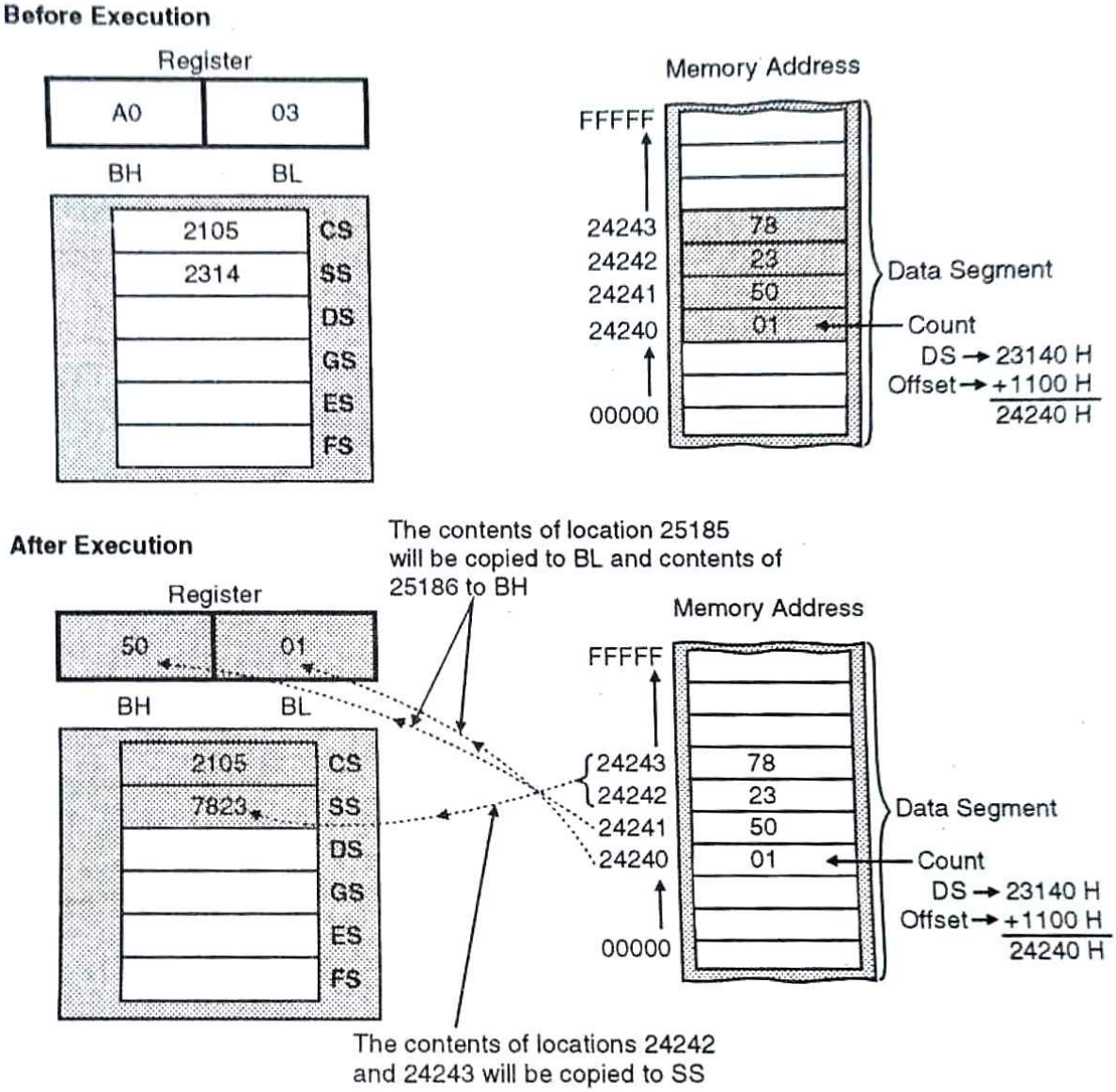


Fig. 4.103

4. Move and Expand Group

These instructions are used to copy data into and out of the control registers, debug registers, and the test registers.The additional instructions in this group are:

MOVSX(Move byte or word or double word with sign extension)

Mnemonic : MOVSX destination , source

Algorithm : Destination = Source

Operation : Destination ←Source

This instruction copies the contents of source register to the desired destination location.

It can be used to transfer 8/16/32 bit contents from source to destination.

The source can be a byte or word or double word of data in register or memory .

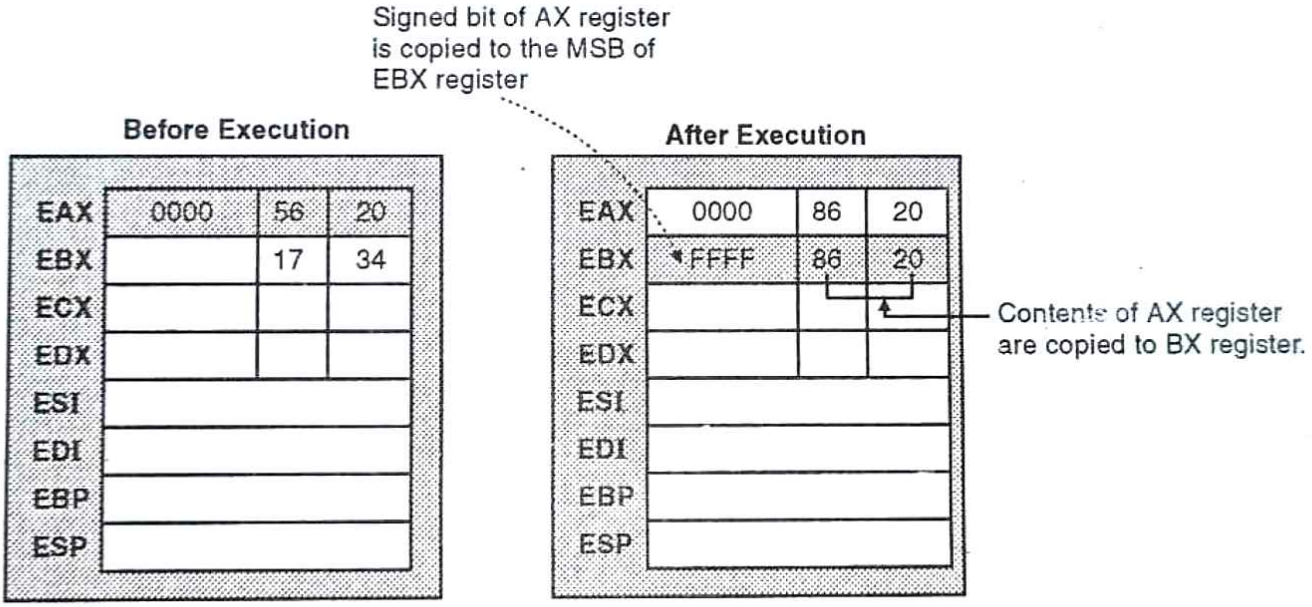
The destination can be either a 16/32 bit register.

If the source is 8 bit and desination is 16 bit then the signed bit of MSB of the 8 bit number is copied to the MSB of the 16 bit number.

If the source is 16 bit and desination is 32 bit then the signed bit of MSB of the 16 bit number is copied to the MSB of the 32 bit number.

Example : MOVSX EBX , AX

This instruction copies a word from the AX register to the BX register . The signed bit of AX is copied to the MSB of EBX register.



4

Fig. 4.104

MOVZX(Move byte or word or double word with zero extended)

Mnemonic : MOVZX destination , source

Algorithm : Destination = Source

Operation : Destination ←Source

This instruction copies the contents of source register to the desired destination location.

It can be used to transfer 8/16/32 bit contents from source to destination.

The source can be a byte or word or double word of data in register or memory .

The destination can be either a 16/32 bit register.

If the source is 8 bit and desination is 16 bit then the zeros are copied to the MSB of the 16 bit number.

If the source is 16 bit and desination is 32 bit then zeros are copied to the MSB of 32 bit number.

Example : MOVZX EBX , CX

This instruction copies a word from the CX register to the BX register . Zero are copied to the MSB of EBX register .

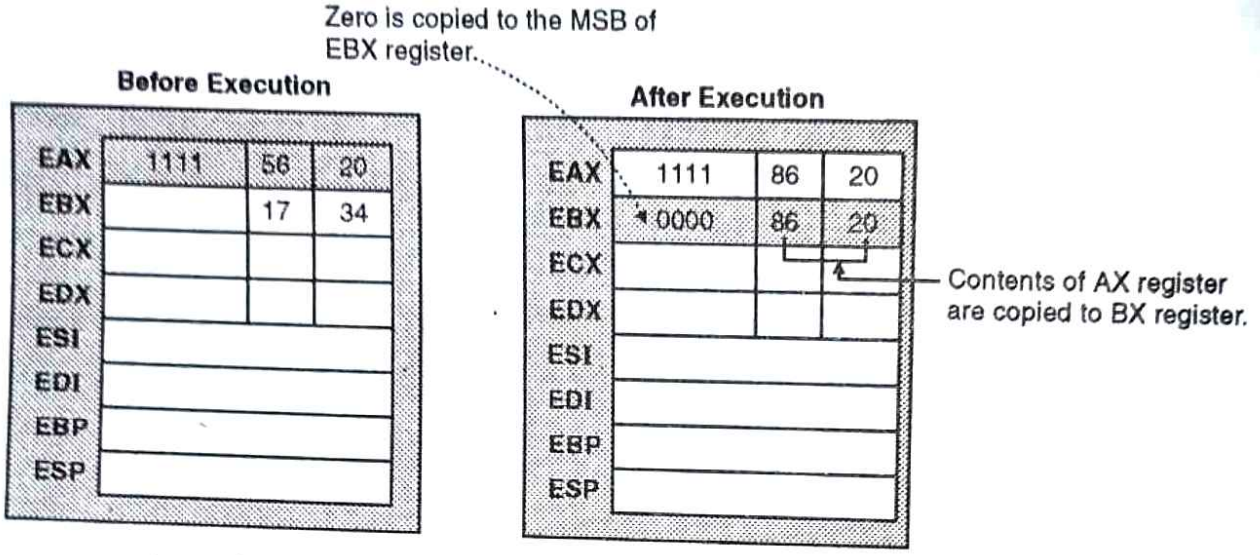


Fig. 4.105

5. Conditional Byte Set :

SETcc(SET Byte on conditional code)

Mnemonic : SETcc

Algorithm : if condition

Byte = FF H

Else

Byte=00 H

Operation : This instruction is used to set a byte of data if the given condition is met, otherwise byte is not set.

Example : STB, STO , SETNO,SETP etc.

6. Shifts Between Words :

SHLD(Shift Left Double)

Mnemonic : SHLD destination , source, count

Algorithm : Shift left from one operand to another .

Operation : This instruction is used to shift specified number of bits left from one operand to another.

Example : SHLD EAX , ECX , 3 ; This instruction shifts the upper 3 its from ECX into the lower 3 bits of EAX register.

SHRD(Shift Right Double)

Mnemonic : SHRD destination , source, count

Algorithm : Shift right from one operand to another .

Operation : This instruction is used to shift specified number of bits right from one operand to another.

Example : SHLD EAX , ECX , 3; This instruction shifts the lower 3 its from ECX into the upper 3 bits of EAX register.