

2. Data unit : The data unit contains the ALU, eight 32-bit general purpose registers and a 64-bit barrel shifter. The barrel shifter is used for multiple bit shifts in one clock. The entire data unit is responsible for data operations requested by the control unit.

3. Protection test unit : The protection test unit checks for segmentation violations under the control of the microcode.

The execution unit partially supports pipelining. It overlaps the execution of any memory reference instruction with the previous instruction.

Segmentation unit

The segmentation unit translates logical addresses into linear addresses at the request of the execution unit. The segmentation unit compares the effective address for the length limit specified in the segment descriptor. The segment unit adds the segment base and the effective address to generate linear address. Before calculation of linear address it also checks for access rights. The violation of access rights causes a protection exception to be generated.

Paging unit

When the 80386 paging mechanism is enabled, the paging unit translates linear addresses generated by the segmentation unit or the code prefetch unit into physical addresses. If paging unit is not enabled, the physical address is the same as the linear address, and no translation is necessary.

The paging unit gives physical address to the Bus Interface Unit to perform memory and I/O accesses.

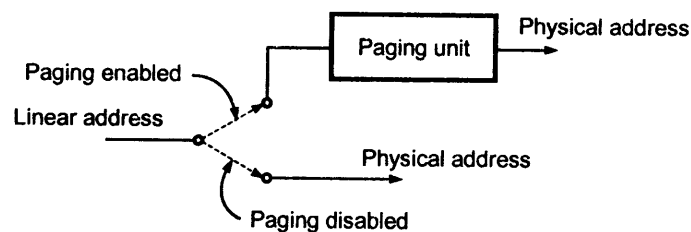


Fig. 13.3 Linear to physical address conversion

80386 Functional Units

Bus interface unit	:	Responsible for memory access, I/O access, coprocessor interface and address relocation.
Code prefetch unit	:	Responsible for instruction fetch
Instruction decode unit	:	Responsible for instruction decode
Execution unit	:	Execute instruction with the help of control, data and protection unit
Segmentation unit	:	Translates logical address to linear address and provides segment level protection.

Registers

Usually, the first item of interest to an assembly language programmer is the register set. The 80386 register set can be categorized according to their usage.

- | | |
|---------------------------------------|----------------------|
| 1. General purpose registers | 2. Segment registers |
| 3. Index, pointers and base registers | 4. Flag registers |
| 5. System address registers | 6. Control registers |
| 7. Debug registers | |

General purpose registers

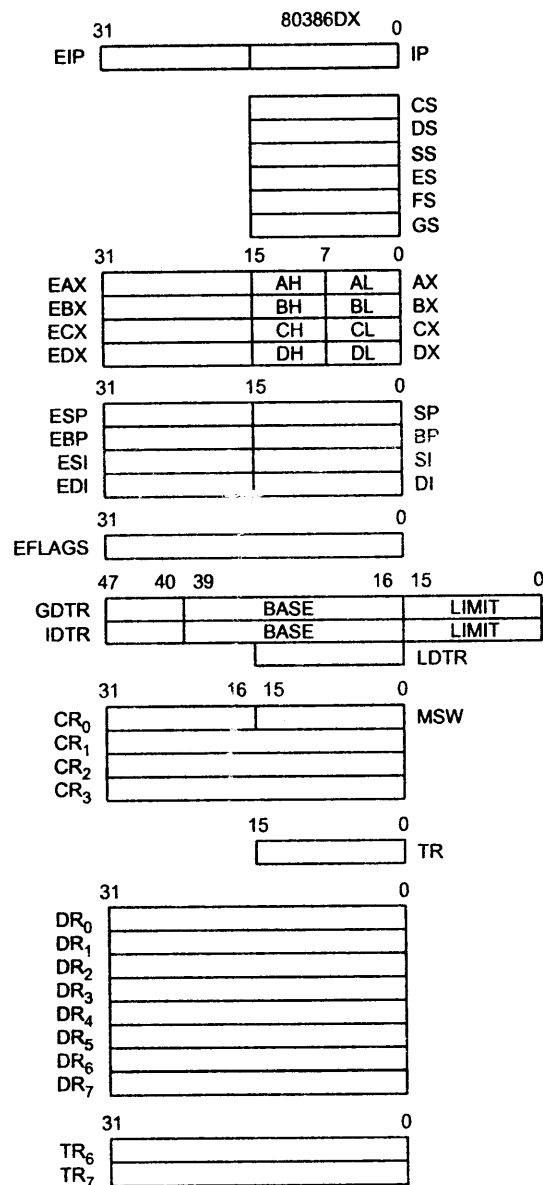


Fig. 13.4 80386 register set

The 80386 contains 32-bit general purpose register called EAX, EBX, ECX, EDX, ESP, EBP, ESI, and EDI. Fig. 13.4 shows the general purpose registers in 80386. The lower 16-bits of each of the general purpose register can be accessed individually. These 16-bit registers are accessed as AX, BX, CX, DX, SP, BP, SI, and, DI respectively. The AX, BX, CX and DX registers can be further divided into two separate bytes : Higher byte and lower byte.

For example : $AX \leftarrow AH + AL$. These bytes can be individually accessed as AH, AL, BH, BL, CH, CL, DH, and DL.

NOTE : Register name beginning with an E (For example : EAX) indicate register width is 32-bit. Register name ending with an X (for example AX) indicate a 16-bit register and register name ending with H or L (for example AH or AL) indicate it is an 8-bit register.

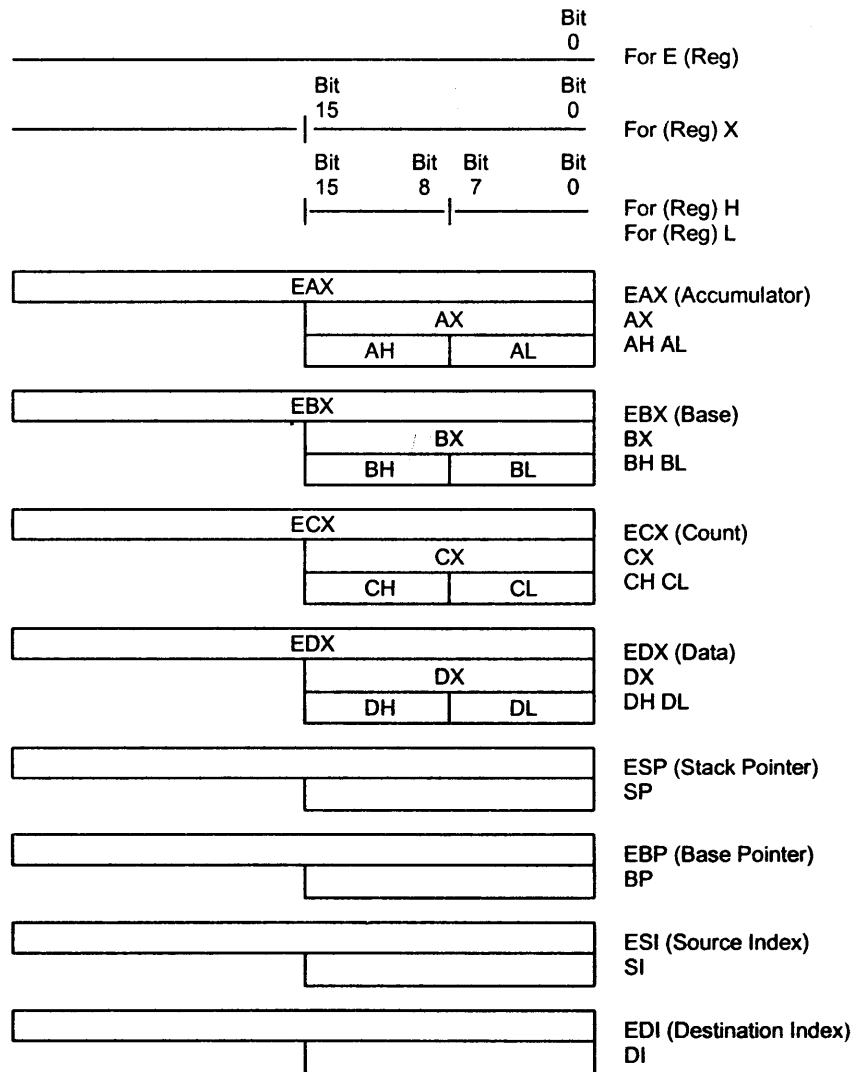


Fig. 13.5 General purpose registers

The other four general purpose registers, are the two pointer registers, ESP and EBP, and the two index registers, ESI and EDI. These registers are used to do special functions. They are used to store offset addresses of memory locations relative to the segment registers. The index registers ESI and EDI are used to store offset values to be incremented or decremented when stepping through block of data. The index registers are also used to hold offset addresses for instructions that access data stored in the data segment part of memory. Thus these registers can be combined with the values in the DS register using index addressing. The pointer register ESP and EBP are used to store offset addresses of memory locations relative to the stack segment register.

Segment registers

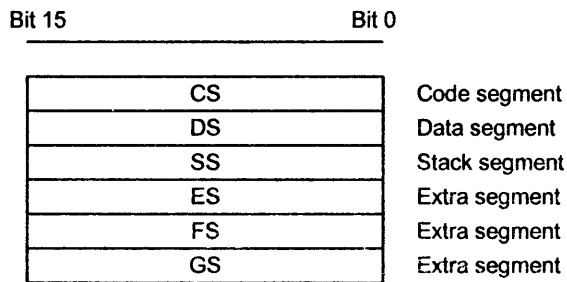


Fig. 13.6 Segment registers

The 80386 has a 1 M-byte address space in real mode. But all of this memory cannot be active at one time. The 80386 supports six simultaneously accessible memory blocks called segments. A segment represents an independently accessible block of memory consisting of 64K consecutive byte-wide storage locations. These segments are addressed by 16-bit

registers : CS, DS, ES, SS, FS and GS. Fig. 13.6 shows the segment registers.

1. The CS (Code Segment) register holds the base address of the currently active code segment.
2. The DS (Data Segment) is used to hold the address of currently active data segment.
3. The ES (Extra Segment), FS, and GS are used as general data segment registers. These registers hold the base addresses of three different memory segments. These segments are referred as to Extra Segments.
4. The base address of the currently active stack segment is contained in the SS (Stack Segment) register.

Index, pointers, and base registers

As mentioned earlier, the physical address of any memory location within a selected memory segment is obtained by adding the segment address and the offset (The contents of segment register are shifted left by 4 and the offset is added to the shifted contents of segment register to generate physical address. The offset used to calculate physical address is contained in any of the pointer, base, or index registers.

The Table 13.1 shows the segments and offset registers used with the corresponding segments

Segment register	Offset registers
CS (Code Segments)	(E) IP (Instruction Pointer)
SS (Stack Segments)	(E) SP (Stack Pointer) (E) BP (Base Pointer)
DS (Data Segments)	(E) BX (Base Register) (E) SI Source Index Register (E) DI Destination Index Register
ES, FS and GS (Extra Segments)	(E) BX (Base Register) (E) SI Source Index Register

Table 13.1 Segments and offset registers

Flag register (EFLAG)

A Flag is a flip-flop which indicates some condition produced by the execution of an instruction or controls certain operations of the EU. The EFLAG register contains thirteen flags. Fig. 13.7 shows the bit pattern of the EFLAG register.

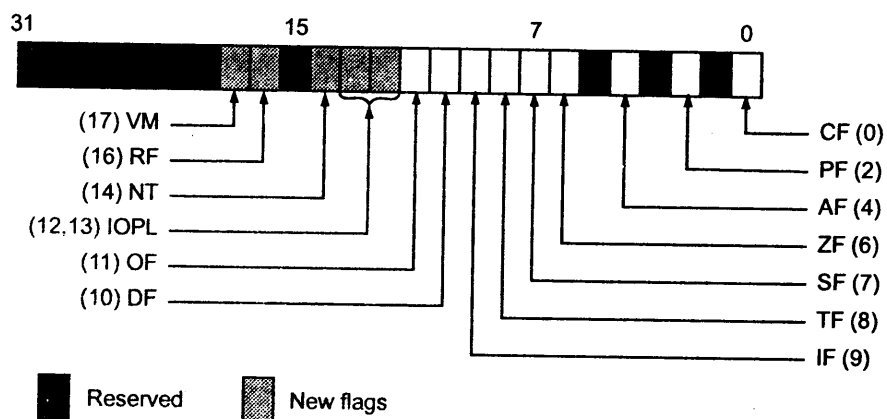


Fig. 13.7 Bit pattern of EFLAG register

These flags can be categorized in three different groups.

- Status flags** : These flags reflect the state of a particular program.
- Control flags** : These flags directly affect operation of few instructions.
- System flags** : These flags reflect the current status of the machine and which are usually used by operating system than by application programs.

Status flags : The status flags are : CF (Carry flag), PF (Parity flag) AF (Auxiliary carry flag), ZF (Zero flag), SF (Sign flag), and OF (Overflow flag). These flags indicate some condition produced by the execution of arithmetic or logical instructions. These flags provide necessary information for arithmetic and logical control decisions.

CF (Carry flag) : This bit is set by arithmetic instructions that generate either a carry or a borrow. This bit can also be set, cleared, or inverted with the STC, CLC or CMC instructions, respectively. Carry flag is also used in shift and rotate instructions to contain the bit shifted or rotated out of the register.

PF (Parity flag) : The parity bit is set by most instructions if the least significant 8-bit of the result contain even number of one's.

AF (Auxiliary carry flag) : This bit is set when there is a carry or borrow after a nibble addition or subtraction, respectively. The programmer can't access this bit directly, but this bit is internally used for BCD arithmetic.

ZF (Zero flag) : Zero flag is set to 1, if the result of an operation is zero.

SF (Sign flag) : The signed numbers are represented by combination of sign and magnitude. The most significant bit (MSB) indicates sign of the number. For negative number MSB is 1. Sign flag is set to 1, if the result of an operation is negative (MSB = 1).

OF (Overflow flag) : In 2's complemented arithmetic, most significant bit is used to represent sign and remaining bits are used to represent magnitude of a number (see Fig. 13.8). This flag is set if the result of a signed operation is too large to fit in the number of bits available (7-bits for 8-bit number) to represent it.

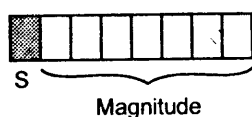


Fig. 13.8 Sign and magnitude representation

For example, if you add the 8-bit signed number 01110110 (+118 decimal) and the 8-bit signed number 00110110 (+54 decimal). The result will be 10101100 (+172 decimal), which is correct binary result. But in this case, it is too large to fit in the 7 bits allowed for the magnitude in an 8-bit signed number. The overflow flag will be set after this operation to indicate that the result of the addition has overflowed into the sign bit.

Control flags

DF (Direction flag) : The direction flag controls the direction of string operations. When the D flag is cleared these operations process strings from low memory up towards high memory. This means that offset pointers (usually SI and DI) are incremented by 1 after each operation in the string instructions when D flag is cleared. If the D flag is set, then SI and DI are decremented by 1 after each operation to process strings from high to low memory.

System flags

VM (Virtual memory) flag : This flag indicates operating mode of 80386. When VM flag is set, 80386 switches from protected mode to virtual 8086 mode.

R (Resume) flag : This flag, when set allows selective masking of some exceptions at the time of debugging.

NT (Nested flag) : This flag is set when one system task invokes another task. (i.e. nested task).

IOPL (I/O privilege level) : The two bits in the IOPL are used by the processor and the operating system to determine your application's access to I/O facilities. It holds privilege level, from 0 to 3, at which the current code is running in order to execute any I/O related instruction.

IF (Interrupt flag) : When interrupt flag is set, the 80386 recognizes and handles external hardware interrupts on its INTR pin. If the interrupt flag is cleared, 80386 ignores any inputs on this pin. The IF flag is set and cleared with the STI and CLI instructions, respectively.

TF (Trap flag) : Trap flag allows user to single-step through programs. When an 80386 detects that this flag is set, it executes one instruction and then automatically generates an internal exception 1. After servicing the exception, the processor executes the next instruction and repeats the process. This single stepping continues until program code resets this flag for debugging programs single step facility is used.

System address registers

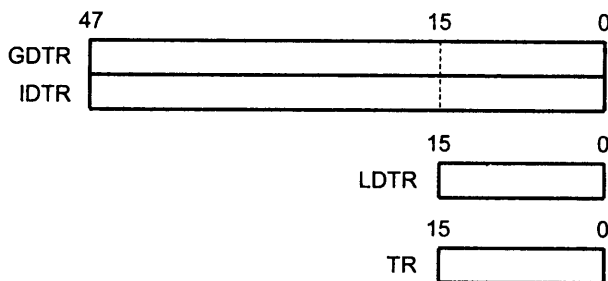


Fig. 13.9 Protected mode registers

There are four system address register : TR (Task Register), IDTR (Interrupt Descriptor Table Register), GDTR (Global Descriptor Table Register) and LDTR (Local Descriptor Table Register). Fig. 13.9 shows these special registers which are used in protected mode.

These registers hold the addresses for the four special descriptor table segments.

- The TR (Task Register) points to the Task state segment

- The IDTR (Interrupt Descriptor Table Register) points to the Interrupt Descriptor Table (IDT)
- The GDTR (Global Descriptor Table Register) points to the Global Descriptor Table (GDT)
- The LDTR (Local Descriptor Table Register) points to the local Descriptor Table (LDT)

These registers and their usage are discussed in detail in section 13.6.2.

13.1.3 Special 80386 Registers

The 80386 includes a new sets of registers for the purpose of control, debug and test. Control registers CR₀-CR₃ control various features, DR₀-DR₇ facilitate debugging, and registers TR₆ and TR₇ are used to test paging and caching.

Control Registers

There are four control registers : CR₀, CR₁, CR₂ and CR₃. Fig. 13.10 shows control registers. These registers define the machine state that affects all the tasks in the systems.

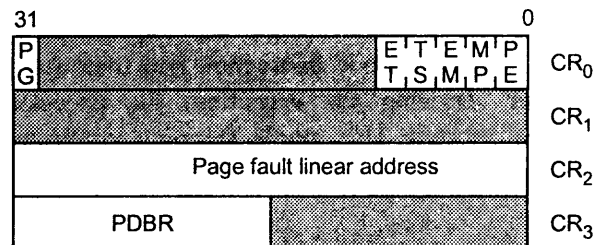


Fig. 13.10 Control registers

Control register 0 (CR₀)

The CR₀ holds the MSW (Machine Status Word). It contains six status bits : PE (Protection Enable), MP (Math Present), EM (Emulate Coprocessor), TS (Task Switched), ET (Extension Type), and PG (Paging).

PE (Protection Enable) : This bit is similar to the VM bit in EFLAGS in that it controls the 80386's mode of operation. When PE is set, it is in protection mode otherwise it operates in Real Mode.

MP (Math Present) : When this bit is set, the 80386 assumes that real floating point hardware (80287 or 80387) is present in the system. When this bit is clear, the 80386 assumes that no such coprocessor exists, and will not attempt to use real floating point hardware.

EM (Emulate Coprocessor) : When this bit is set, the 80386 will generate an exception 11 (device not available) whenever it attempts to execute a floating point instruction. Programmer can use this exception handler to emulate floating point hardware in software.

TS (Task Switched) : The 80386 sets the bit automatically every time it performs a task switch. It will never clear this bit on its own. But programmer can clear this bit using CLTS instruction.

ET (Extension Type) : When power is applied, 80386 detects whether numeric processor connected is 80287 or 80387 and sets ET to logic 1, if numeric processor is 80387. This is necessary because the 80387 uses a slightly different protocol than 80287.

PG (Paging) : This bit enables or disables paging mechanism in Memory Management Unit (MMU). If bit is set, paging is enabled. Paging mechanism is explained detail in section 13.6.2.

Control Register 1 (CR₁)

This is reserved by Intel.

Control Register 2 (CR₂)

CR₂ is read-only register. The 80386, itself writes the last 32-bit linear address of page fault routine in this register. When page fault occurs, the 80386 generates exception 14 (page fault). This address is important for writing page fault routine. The page fault routine helps programmer to find cause of the fault.

Control Register 3 (CR₃)

Control register 3 holds the physical address of the root of the two-level paging tables used when paging is enabled. It is also called **Page Directory Base Register (PDBR)**.

Debug Registers

Debug registers allow 80386 to provide debugging feature. The DR₀ to DR₇ registers are used to control debug feature. The debug registers DR₀ to DR₃ contain addresses associated with one of four breakpoints defined by certain bits in debug register 7 (DR₇) Fig. 13.11 shows debug registers. The software debugger can load breakpoint addresses in these registers to aid in debugging. (See Fig. 13.11 on next page).

Debug Registers 0 through 3

The first four debug registers (DR₀ - DR₃) hold four linear addresses for breakpoints. The addresses in these registers are compared with address of the each instruction at the time of instruction execution and if a match is found, an exception 1 (debug fault) is generated. This allows 80386 to monitor upto four different addresses in the system.

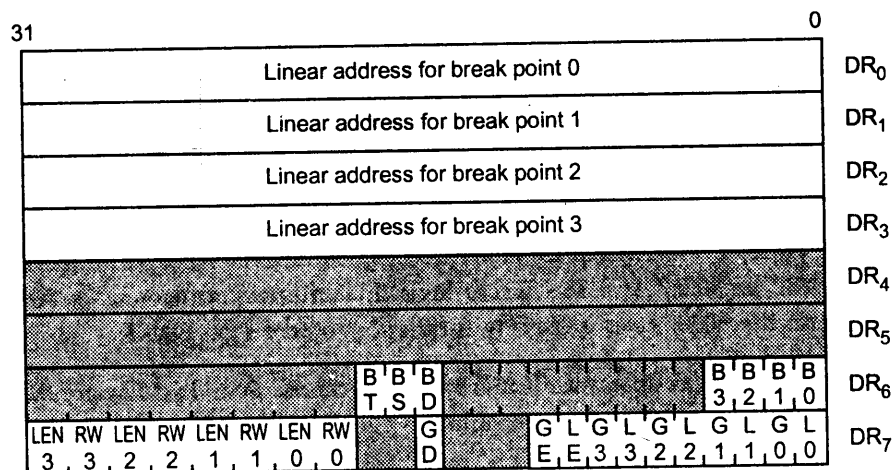


Fig. 13.11 Debug registers

Debug registers 4 and 5

Registers 4 and 5 are undefined.

Debug register 6

Debug register 6 is also called **debug status register**. The 80386 sets the appropriate bits in this register which gives information of the probable causes for the last debug fault (Exception 1). The 80386 never clears these bits. Programmer must clear these status bits by writing into DR₆.

The status bits are :

B0 (Breakpoint 0) : The 80386 sets this bit when it references the linear address contained in DR₀, according to the conditions set by the LEN₀, RW₀, L₀, G₀, LE, and GE fields in DR₇.

B1-3 (Breakpoint P-3) : These bits are similar to B₀. In each case the linear address is referred from the respective debug register. For example : B₂ refers DR₂.

BD (Break for debug register access) : The access for the debug registers can be locked by setting GD bit in DR₇. The BD bit, if set, allows to invoke exception 1 handler, if processor tries to access debug register eventhough the accessed is locked.

BS (Break for single step) : This bit is set if the 80386 has invoked exception 1 since trace bit is set (TF bit is set in EFLAGS)

BT (Break for task switch) : When a task is initiated whose trace bit is set, the 80386 invokes an exception 1 if BT bit is set.

Debug register 7

It controls the debug feature. By programming bits in this register, programmer can configure the debug operation of the four linear address breakpoints. Each breakpoint is controlled by a set of four fields. These are :

L0 (Local enable) : When this bit is set, the breakpoint address in DR₀ is monitored as long as 80386 is executing current task. When a task switch occurs, this bit is cleared by the 80386 and it must be re-enabled by writing into DR₇ required.

G0 (Global enable) : When this bit is set, the breakpoint address in DR₀ is monitored all times, regardless of task. This bit must be cleared by writing into DR₇.

RW0 (Read/Write access) : These bits decides the type of access that must occur at the address in DR₀. Table 13.2 gives the list of different access types.

RW	RW bits in register DR ₇
00	Code fetch
01	Data write
10	Reserved
11	Data Read or write

Table 13.2 RW bits

LEN0 (Breakpoint length) :

The breakpoints are further distinguished by its size. The Table 13.3 shows the different sizes of the breakpoints.

LEN	LEN bits in register DR ₇
00	1 byte
01	2 bytes, word aligned
10	Reserved
11	4 bytes, dword aligned

Table 13.3 LEN bits

There are in all four such fields (L, G, RW and LEN) for four breakpoints (B0-B3). The DR₇ contains three more bits. These are

LE (Local Exact) : The pipelined architecture of 80386 fetches, decodes next instruction before the current one completes. Due to this, 80386 may not set status bit in DR₆ at the instant breakpoint occurs. If you set local exact bit, 80386 sets, corresponding status bit at the same instant at which breakpoint occurs, when 80386 is running the current task. When a task switch occurs this bit is cleared. This bit applies to all four linear breakpoints.

GE (Global Exact) : This is similar to the LE bit. If this bit is set 80386 informs about breakpoint at the instant it occurs regardless of task.

GD (Global debug access) : When this bit is set, the 80386 denies the further access to any of the debug registers, either for reading or writing.

Test registers

Among the eight test registers (TR₀-TR₇), only two test registers (TR₆-TR₇) are currently defined.

The fig. 13.12 shows the bit pattern of test registers.

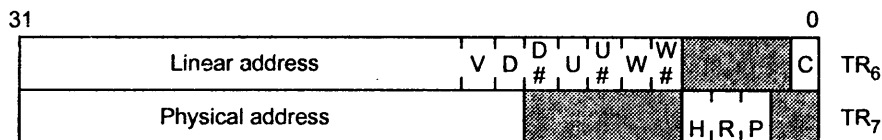


Fig. 13.12 Test registers

These registers are used to check translation lookaside buffer (TLB) of the paging unit. The TLB is explained detail in section 13.6.2.

Test Register 6 : This is the TLB testing command registers. By writing into this register, it is possible to either initiate a write directly into the 80386's TLB or to perform TLB lookups. TR₆ is divided into fields as follows :

- C : This is a command bit. When this bit is cleared, a write to the TLB is performed. If it is set, the processor performs a TLB lookup. The next 7 bits are used as tag attributes for the TLB cache, either when writing a new entry or when performing a TLB lookup.
- W̄ (bit 5) : Not writable
- W (bit 6) : Writable
- Ū (bit 7) : Not user
- U (bit 8) : User
- D̄ (bit 9) : Not dirty
- D (bit 10) : Dirty
- V (bit 11) : Valid
- Linear address (bits 12-31) : This is the tag field of the TLB. This field serves as the upper 20 bits of a linear address to be used for TLB references.

Test Register 7 : This register is the data testing register of the TLB. When a program is performing writes, the entry to be stored is contained in this register, along with cache set information.

TR₇ is divided into fields as follows :

RP : This is replacement pointer. This field indicates which set of the TLB's four-way set associative cache to write to.

H : This is pointer location. If this bit is set, the RP field determines which cache set to write to. If it is cleared, the set is determined with an internal algorithm.

Physical address (bits 12-31) : This is the data field of the TLB. This field contains either the physical address to be written into the TLB or the result of a valid TLB hit.

13.1.4 Summary of Various Registers in 80386

Registers	
General purpose	: EAX, EBX, ECX, EDX, ESP, EBP, ESI and EDI
Scratch Pad	: EAX, EBX, ECX and EDX
Index	: ESI and EDI
Pointer	: ESP and EBP
Segment	: CS, DS, SS, ES, FS and GS. Store the base addresses of the currently active segments. In Real Mode and in Protected Mode these registers are used to store selectors which point the descriptors for segments.
Flag	
Status	: CF, PF, AF, ZF, SF and OF
Control	: DF

System	:	VM, R, NT, IOPL, IF, TF
System address registers	:	TR, IDTR, GDTR and LDTR
Control registers	:	CR ₀ - CR ₃
CR ₀	:	Stores machine status word
CR ₁	:	Reserved by Intel
CR ₂	:	Stores address of page fault routine
CR ₃ (PDBR)	:	Stores address of two level paging tables
Debug	:	DR ₀ - DR ₇
DR ₀ -DR ₃	:	DR ₀₋₃ holds four linear addresses for breakpoints
DR ₄	:	Stores information of last debug fault
DR ₇	:	Decides the access for the debug register controls the debug feature
Test registers	:	TR ₀ - TR ₇
TR ₀ - TR ₅	:	Undefined
TR ₆ - TR ₇	:	Checks translation look-aside buffer of the paging unit

13.1.5 Data Types

The 80386DX is a 32-bit processor. The 80386DX's 32-bit wide data path allows memory to be read and written 32-bits at a time. Depending on the instruction used, the 80386DX interpret the data it reads from memory in different ways. The data can be interpreted as a signed value or unsigned value. Fig. 13.13 shows the data formats for signed and unsigned values.

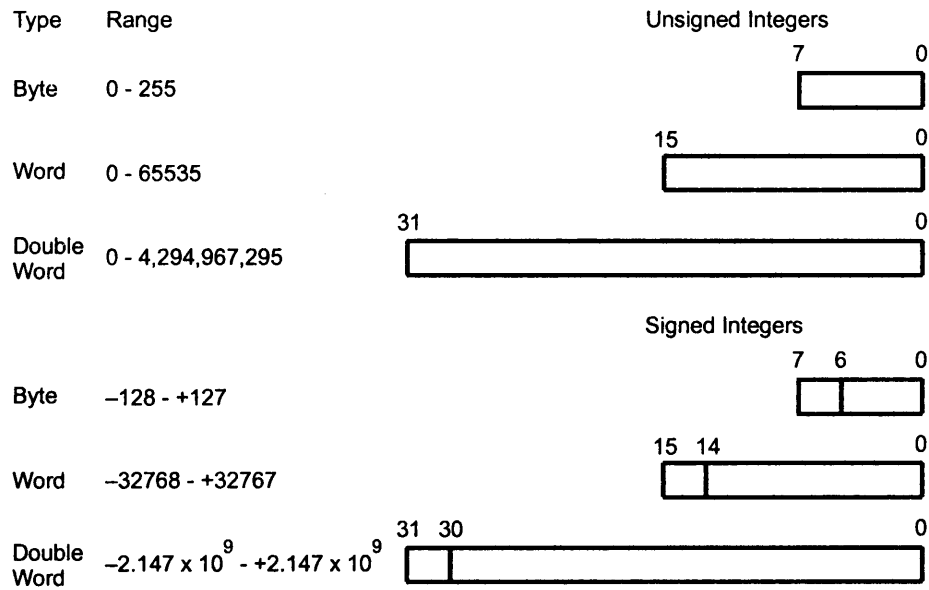


Fig. 13.13 Signed and unsigned data types

As shown in the figure the 80386DX handles signed numbers by interpreting the most significant bit of an operand as a sign bit. If the most significant bit of the operand is 0, then the number is positive. If it is 1, the number is negative.

BCD numbers : It is important to note that the 80386DX has the ability to perform four-function arithmetic on numbers that are represented in binary-coded decimal (BCD) instead of the standard binary format. The BCD numbers are more readable but they take more storage space. The 80386DX can handle two different BCD formats :

1. Unpacked BCD
2. Packed BCD

In unpacked BCD, one decimal digit (0 - 9) is stored per byte whereas in packed BCD two decimal digits are stored per byte. Thus the maximum value that can be stored in the unpacked BCD format is 9 and in packed BCD it is 99 (in one byte).

The 80386DX supports the data types which can be used as far and near pointers. Fig. 13.14 shows the datatypes supported by 80386DX. The data formats long integer, short real, long real, extended real and packed BCD are technically not supported by the 80386 at all, but rather by the 80387 numeric processor.

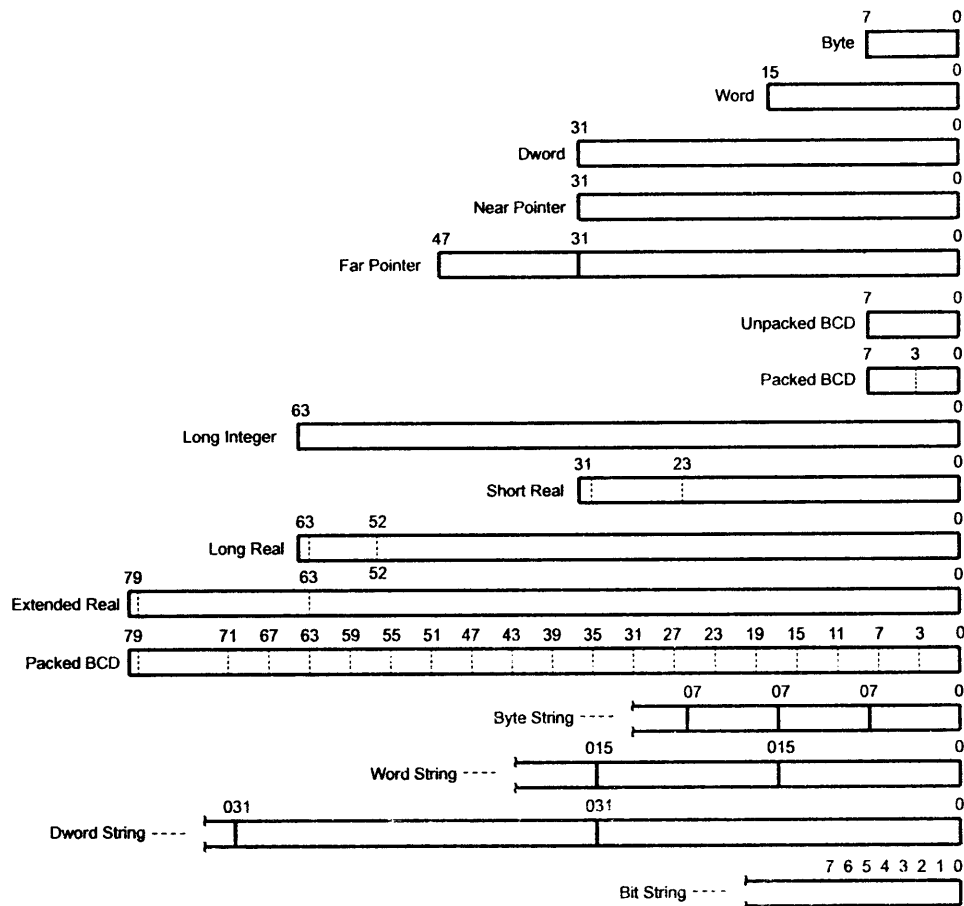


Fig. 13.14 Datatypes supported by 80386DX

BIT : The 80386DX also supports "BIT" data type. The bit data type allows a program to directly access and modify any selected bit within a bit string. The programmer can initialize bit data type with the help of DBIT mnemonic. The 80386DX assembler supports eight instructions for bit operations. These are : BT, BTC, BTS, BTR, BSF, BSR, IBTS, and XBTS.

Strings : The 80386DX supports bit string, byte string, word string, and dword strings. These strings can easily be scanned for a particular value or any deviation from a value or to match the value.

13.1.6 Addressing Modes of 80386DX

When processor executes an instruction, it performs the specified function on data, which is referred to as **operands**. The operand may be the part of instruction, may reside in one of the internal registers of the processor, may be stored in memory, or may be held at an I/O port. As a part of programming flexibility, processor provides different ways to access these operands from different locations. The different ways by which processor can access data are referred to as **addressing modes**.

The 80386DX provides a total of 11 addressing modes for instructions to specify operands. These addressing modes can be categorized in three groups :

- Register operand addressing
- Immediate operand addressing
- Memory operand addressing

Register operand addressing mode

In the register addressing mode, the operand is located in one of the 8, 16 or 32-bit general purpose registers of 80386 DX. Table 13.4 shows the list of internal general purpose registers that can be used as a source or destination operand.

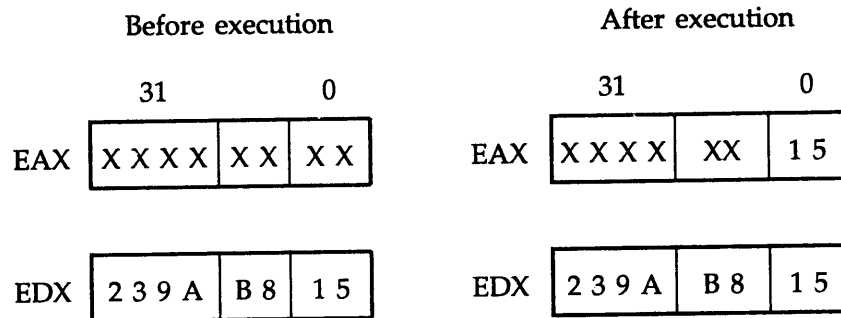
Register	Operand size		
	Bytes (Reg 8)	Word (Reg 18)	Double word (Reg 32)
Accumulator	AL, AH	AX	EAX
Base	BL, BH	BX	EBX
Count	CL, CH	CX	ECX
Data	DL, DH	DX	EDX
Stack pointer	–	SP	ESP
Base pointer	–	BP	EBP
Source index	–	SI	ESI
Destination index	–	DI	EDI
Code segment	–	CS	–
Data segment	–	DS	–
Stack segment	–	SS	–
E data segment	–	ES	–
F data segment	–	FS	–
G data segment	–	GS	–

Table 13.4 Direct addressing registers and their sizes

Examples :

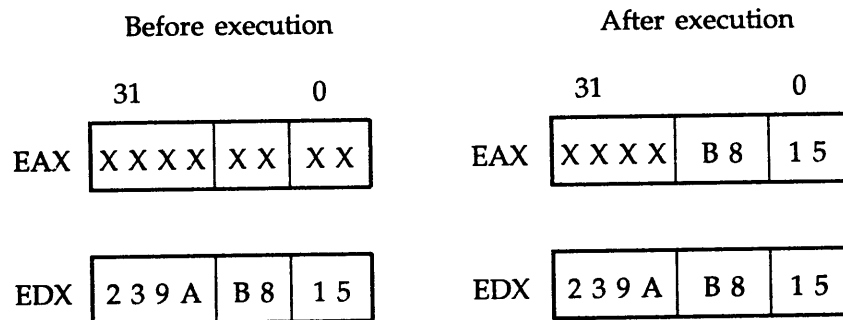
For 8-bit operand : MOV AL, DL

This instruction copies the lower byte contents of the EDX register to the lower byte of the EAX register. Both source and destination operands are the internal registers of 80386DX.



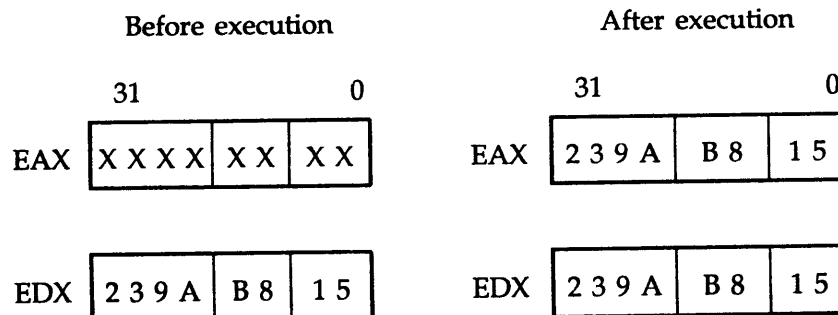
For 16-bit operand : MOV AX, DX

This instruction copies the lower word contents of EDX register to the lower word of the EAX register.



For 32-bit operand : MOV EAX, EDX

This instruction copies the contents of EDX register to the EAX register.



Immediate operand addressing mode

In the immediate operand addressing mode, the operand is a part of the instruction, as shown in the Fig. 13.15. The operand can be 8-bit, 16-bit or 32-bit.

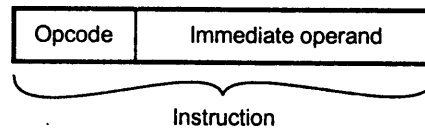


Fig. 13.15 Instruction encoded with an immediate operand

Example :

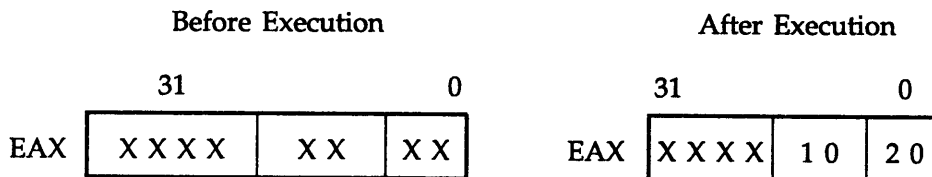
For 8-bit operand : MOV AL, 20H

This instruction copies 20H in the lower byte of EAX register.



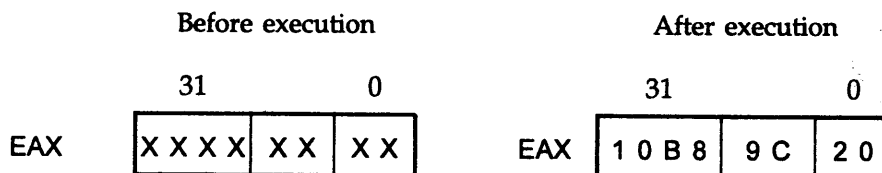
For 16-bit operand : MOV AX, 1020 H

This instruction copies 1020H in the lower word of EAX register



For 32-bit operand : MOV EAX, 10B89C20H

This instruction copies 10B89C20H in the EAX register.



Memory operand addressing modes

The remaining 9 addressing modes provide a mechanism for specifying the physical address of an operand. In 80386DX, physical address is calculated before any read or write operation.

The physical address consists of two components : The **segment base address** and an **effective address**. The effective address can be specified in a variety of ways. One way is to encode the effective address of the operand directly in the instruction. This represents direct addressing mode. The effective address can be generated with the combinations of four addressing elements : Base, Index, Scale factor and Displacement.

Where

Base : The contents of any general purpose register.

Index : The contents of any general purpose register. The index registers are used to access the elements of an array, or a string of characters.

Scale : The index register's value can be multiplied by a scale factor either 1, 2, 4, or 8. Scaled index mode is especially useful for accessing arrays or structures.

Displacement : An 8, 16 or 32-bit immediate value following the instruction.

The general formula for generating effective address is given as follows :

$$EA = \text{Base} + (\text{Index} \times \text{Scaling Factor}) + \text{Displacement}$$

The Fig. 13.16 shows the registers that can be used to hold the values of segment base, base, and index.

Physical Address = Segment Base Address + Effective Address

$$(PA) = SBA + EA$$

$$PA = SBA : \{ \text{Base} + (\text{Index} \times \text{Scale factor}) + \text{Displacement} \}$$

$$PA = \left\{ \begin{array}{c} CS \\ SS \\ DS \\ ES \\ FS \\ FS \\ GS \end{array} \right\} : \left\{ \begin{array}{c} AX \\ BX \\ CX \\ DX \\ SP \\ BP \\ SI \\ DI \end{array} \right\} + \left[\left\{ \begin{array}{c} AX \\ BX \\ CX \\ DX \\ BP \\ SI \\ DI \end{array} \right\} \times \left\{ \begin{array}{c} 1 \\ 2 \\ 4 \\ 8 \end{array} \right\} \right] + \left\{ \begin{array}{c} 8, 16 \text{ or} \\ 32 \text{ Bit} \\ \text{Displacement} \end{array} \right\}$$

Fig. 13.16 Physical address generation

Now we see the different memory operand addressing modes :

Direct mode : In this mode, the instruction is having the effective address of the operand. This effective address is used as an 8, 16 or 32 displacement from the location specified by the current value in the selected segment register is always DS.

Example : `MOV EBX, [159DH]`

$$\text{Here, } PA = DS + 159DH$$

Register indirect mode : In this mode, the base register gives the effective address of the operand.

Example : `MOV EBX, [EAX]`

$$\text{Here, } PA = DS + EAX$$

Based mode : In this mode, a base register's contents are added to a displacement to form the effective address of the operand.

Example : `MOV EBX, [EAX + 24]`

$$\text{Here, } PA = DS + EAX + 24$$

Index mode : In this mode, an index register's contents are added to a displacement to form the effective address of the operand.

Example : MOV EBX, [SI] + 159DH

Here, PA = DS + 159DH + SI

Scaled index mode : In this mode, an index register's contents are multiplied by a scaling factor and then added to displacement to form the effective address of the operand.

Example : MOV EBX, 159DH + [SI * 4]

Here, PA = DS + 159DH + (SI * 4)

Based index mode : In this mode, the contents of a base register are added to the contents of an index register to form the effective address of the operand.

Example : MOV EBX, [ESI][EAX]

Here, PA = DS + ESI + EAX

Based scaled index mode : In this mode, the contents of an index register are multiplied by a scaling factor and then added to the base register to obtain the effective address of the operand.

Example : MOV EBX, [ESI * 2][EAX]

Here, PA = DS + (ESI × 4) + EAX

Based index mode with displacement : In this mode, the contents of an index register and the base register and a displacement are all added together to form the effective address of the operand.

Example : MOV EBX, [EAX][EDI + 24]

Here, PA = DS + EAX + EDI + 24

Based scaled index mode with displacement : In this mode, the contents of an index register are multiplied by a scaling factor and result is then added to the contents of a base register and displacement to form the effective address of the operand.

Example : MOV EBX, [EAX][ESI * 4] + 24

Here, PA = DS + EAX + (ESI × 4) + 24

13.1.7 New 80386 Instructions

The 80386 instruction set includes all the 8086/80186/80286 instructions. The 80386 extends these instructions to work with 32-bit data words and 32-bit offsets. The 80386 instruction set also includes several new instructions which are discussed here with example.

Bit scan and test instructions

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

BT : Bit Test

Bit Test and put specified bit in carry flag. This instruction tests the bit specified in the instruction by setting or resetting carry flag according to status of the specified bit.

Example :

BT ECX, 4 ; Copy bit 4 of ECX to carry flag

BTS : Bit Test and Set

Like, BT instruction, this instruction tests the specified bit and then sets the specified bit.

Example :

MOV CL, 5
BTS EBX, CL ; Copy bit 5 of EBX to CF and set bit 5 of EBX

BTR : Bit Test and Reset

Like BT instruction, this instruction tests the specified bit and then resets the specified bit

Example :

BTR EDX, 3 ; Copy bit 3 of EDX to CF and reset bit 3 of EDX

BTC : Bit Test and Complement

Like BT instruction, this instruction test the specified bit and then complements the specified bit.

Example :

BTC EAX, 05 ; Copy bit 5 of EAX to CF and complement
; bit 5 of EAX

BSF Instruction : Bit Scan Forward

The BSF instruction scans the bits in the second word/doubleword operand starting with bit 0 (least significant bit) until nonzero bit is found. If all of the bits scanned are 0, the zero flag (ZF) is set to; otherwise, ZF is reset to 0. It also loads the destination operand with the bit index of the first set bit.

Example :

BSF BX, AX ; Scan AX from Bit 0 until nonzero bit, leave bit
; number in BX and zero flag set if all AX = 0

BSR : Bit Scan Reverse

The BSR instruction scans the bits in the second word/doubleword operand starting with most significant bit until nonzero bit is found. If all of the bits scanned are 0, the zero flag (ZF) is set to; otherwise, ZF is reset to 0. It also loads the destination operand with the bit index of the first set bit.

Example :

```
BSR CX, DX      ; Scan DX from most significant bit until nonzero
                ; bit and leave bit number in CX and zero flag set if
                ; all DX = 0.
```

Data type conversions**CDQ :** Converts Dword to Qword

This instruction converts the signed double word in the EAX register to a signed quadword in the EDX : EAX register pair. The CDQ operates by extending the most significant bit of the EAX register into all of the bit positions in EDX register.

CWDE : Converts Word to Dword

This instruction convert the signed byte in the AX register to a signed double word in the EAX register. The CBW operation works by extending the most significant bit of the AX register into the two most significant bytes of the EAX register.

Segment load instructions

These instructions are similar to LDS and LES instructions described in chapter 3.

LFS : This instruction loads new values into the specified register and into the FS register from four successive memory locations. The word from the first two memory location is copied into the specified register and the word from the next two memory locations is copied into the FS register.**Example :**

```
LFS X, [3483H]   ; Copy contents of memory at displacement of 3483H in
                ; DS to CL, contents of 3484H in DS to CH and copy
                ; the contents of memory at displacement of 3485H and
                ; 3486H in DS to FS register.
```

LGS : Load register and GS with words from memory. This instruction copies a word from two memory locations into the register specified in the instruction. It then copies a word from the next two memory locations into the GS register.**Examples :**

```
LGS CX, [391AH]  ; Copy contents of memory at displacement of 391AH
                ; to CH. Copy contents at displacement of 391CH and
                ; 391DH in GS to GS register.
```

LSS : Load register and SS with words from memory. This instruction copies a word from two memory locations into the register specified in the instruction. It then copies a word from the next two memory locations into the SS register.**Examples :**

```
LSS CX, [391AH]  ; Copy contents of memory at displacement of 391AH
                ; to CH. Copy contents at displacement of 391CH and
                ; 391DH to SS register.
```

Move and expand instructions**MOVSX** : Move byte or Word, Dword, with sign extension

This instruction takes byte or word as a source and sign extends the value into destination register. Source can be either byte or word of data in the register or storage location in memory, while the destination can be either 16-bit or 32-bit register.

Example :

```
MOVSX EBX, AX    ; Copy the 16-bit value in AX into EBX as lower 16-bits.
                 ; After copy, bit 15 of AX is extended into the 16 higher
                 ; order bit of EBX.
```

MOVZX : Move byte or Word, Dword, with zero extension

This instruction takes byte or word as a source and zero extends the value into destination register. Source can be either byte or word of data in the register or storage location in memory, while the destination can be either 16-bit or 32-bit register.

Example :

```
MOVZX EBX, AX    ; Copy the 16-bit value in AX into EBX as lower 16-bits.
                 ; After copy 16 higher order bit of EBX are all zeros.
```

Conditional byte set

SETxx : Set all bits in specified byte if condition xx is met. xx here can be any condition from conditional jump mnemonics. For example, SETO, SETNO, SETB, SETP, etc.

Shifts between words

SHLD-shift left double : This instruction shifts specified number of bits left from one operand into another.

Example :

```
SHLD EAX, EBX, 8 ; Shift upper 8-bits from EBX into lower 8-bits of EAX
```

SHRD - Shift Right Double :

This instruction shifts number of bits right from one operand into another.

Example :

```
SHRD EAX, EBX, 8 ; Shift lower 8-bits from EBX into upper 8-bits of EAX
```

13.1.8 Instruction Enhancements

Several new instructions of 80386 have made significant improvement over the 8086/80186/80286 versions.

1. The 80386 string instructions work with double-word operands as well as with word and byte operands.

2. Unlike 8086, destination for a 80386 conditional jump can be anywhere in the segment containing the jump instruction. In 8086, conditional jumps are short type jumps where destination address must be in the range of -128 bytes to $+127$ bytes from the address of the instruction after the jump instruction.
3. The LOOP instructions can use the CX register or the ECX register as a counter.
4. PUSHA pushes the 8 general purpose 32-bit registers on the stack, and POPA restores these registers except for the value of ESP which is ignored.
5. PUSHFD pushes the 32-bit EFLAGS register and POPFD restores it.
6. IRETD POPS the double word EIP, a double word for CS, and the EFLAGS register off the stack. The high word of the value popped for CS is discarded.
7. The IMUL instruction is now more generalized. It can perform signed multiplication on any general-purpose register and a memory location or another general-purpose register.
8. The 80386 supports all protected mode instructions supported by 80286. In addition, the 80386 instructions used to move data to/from the control register (CR₀-CR₃), the debug register (DR₀-DR₇), and the test registers (TR₀-TR₇) can be executed only in protected mode.

13.2 Pin Description of 80386DX Microprocessor

The number of changes have been made in the 80386DX hardware to make it more versatile and improve its performance. The Fig. 13.17 (a) and (b) shows functional pin diagram and pin layout of 80386DX.

As shown in the Fig. 13.17 (b), these signals are separated in four groups :

1. Memory/IO interface
2. Interrupt interface
3. DMA interface
4. Coprocessor interface

Please refer Fig. 13.7 on next page.

13.2.1 Memory/IO Interface Signals

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

Data bus : The data bus consists of 32 pins (D₃₁ - D₀). These lines are used to transfer 8, 16, 24, or 32-bit data at one time.

Address bus : The 80386DX generates 32 bit address. The higher 30 bits of address are sent on the A₃₁-A₂. The lower 2-bits, select one of four bytes of the 32-bit data bus. These two bits are internally decoded and sent on the four byte enable pins (\overline{BE}_3 - \overline{BE}_0). Each byte enable pin corresponds to one of four bytes of the 32-bit data bus.

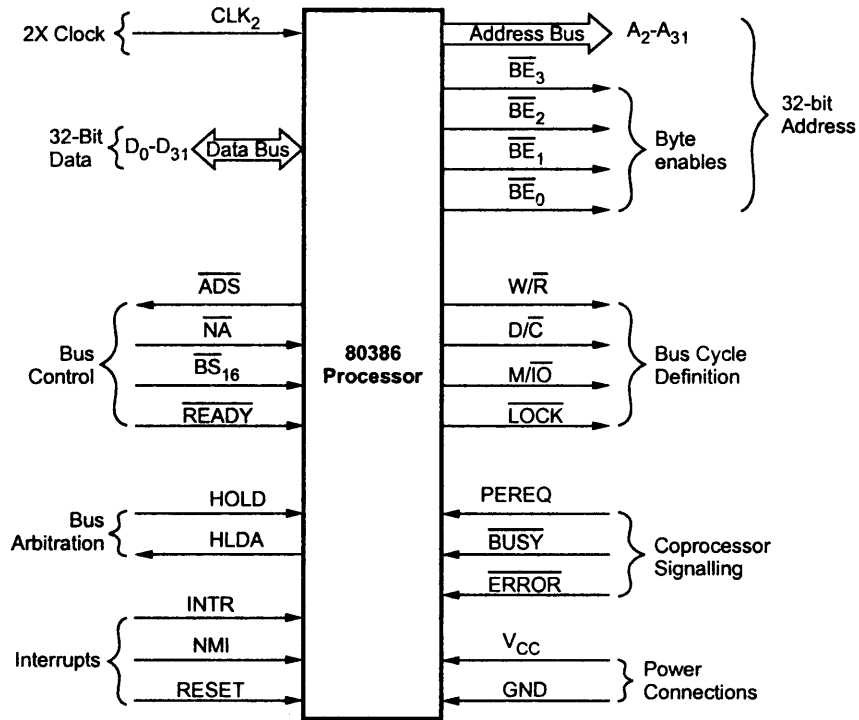


Fig. 13.17 (a) Functional pin diagram of 80386DX

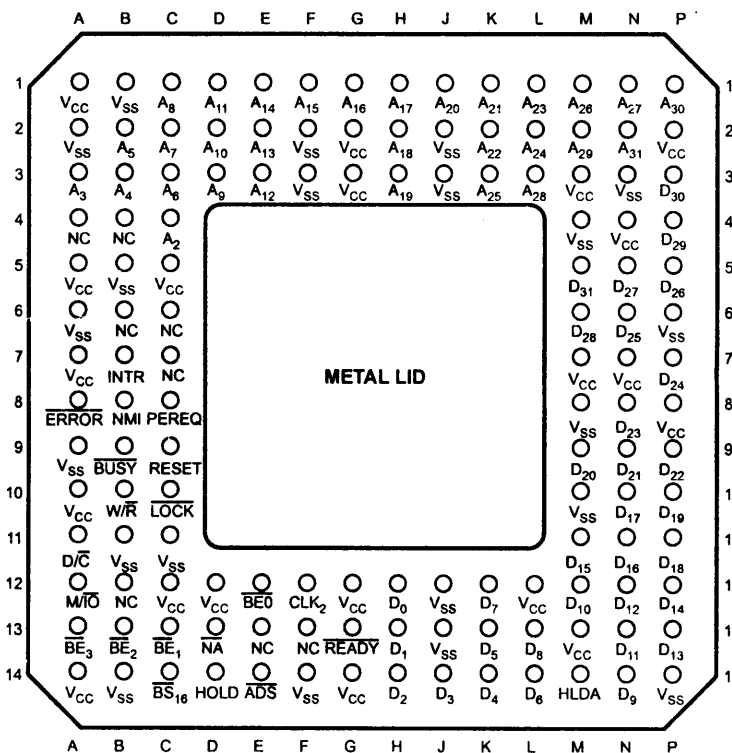


Fig. 13.17 (b) Pin layout of 80386DX

\overline{BE}_3	\overline{BE}_2	\overline{BE}_1	\overline{BE}_0	D ₃₁ - D ₂₄	D ₂₃ - D ₁₆	D ₁₅ - D ₈	D ₇ - D ₀
1	1	1	0				XXXXXXXX
1	1	0	1			XXXXXXXX	
1	0	1	1		XXXXXXXX		
0	1	1	1	XXXXXXXX			
1	1	0	0			XXXXXXXX	XXXXXXXX
1	0	0	1		XXXXXXXX	XXXXXXXX	
0	0	1	1	XXXXXXXX	XXXXXXXX		
1	0	0	0		XXXXXXXX	XXXXXXXX	XXXXXXXX
0	0	0	1	XXXXXXXX	XXXXXXXX	XXXXXXXX	
0	0	0	0	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX

Fig. 13.18 (a) Types of data transfers for various byte enable combinations

\overline{BE}_3	\overline{BE}_2	\overline{BE}_1	\overline{BE}_0	D ₃₁ - D ₂₄	D ₂₃ - D ₁₆	D ₁₅ - D ₈	D ₇ - D ₀
1	1	1	0				XXXXXXXX
1	1	0	1			XXXXXXXX	
1	0	1	1		XXXXXXXX		DDDDDDDD
0	1	1	1	XXXXXXXX		DDDDDDDD	
1	1	0	0			XXXXXXXX	XXXXXXXX
1	0	0	1		XXXXXXXX	XXXXXXXX	
0	0	1	1	XXXXXXXX	XXXXXXXX	DDDDDDDD	DDDDDDDD
1	0	0	0		XXXXXXXX	XXXXXXXX	XXXXXXXX
0	0	0	1	XXXXXXXX	XXXXXXXX	XXXXXXXX	
0	0	0	0	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX

Fig. 13.18 (b) Data transfers including duplication

Bus Enable Signal	Data Lines
\overline{BE}_0	D ₇ - D ₀
\overline{BE}_1	D ₁₅ - D ₈
\overline{BE}_2	D ₂₃ - D ₁₆
\overline{BE}_3	D ₃₁ - D ₂₄

Fig. 13.18 (a) and (b) show the data transfers for all the possible variations of the byte enable outputs with and without duplication of data. The 80386DX performs data duplication during certain types of write cycles. When the data is transferred over high-order of data bus, same data is available on the low-order of data bus. This duplication is indicated by "DDDDDDDD" in Fig. 13.18 (b) .

Bus Status Signals : The bus status signals decide the type of bus cycle to be performed. These signals are :

1. Address status
2. Write/Read
3. Memory/I/O
4. Data/Control
5. LOCK.

Address status (\overline{ADS}) : A low on this pin indicates that the valid address is present on the address bus.

Write/Read ($\overline{W/R}$) : This signal decides the specific type of memory or I/O operation that will occur during a bus cycle. A low on this pin indicates that, the data is to be read from memory or an I/O port, on the otherhand a high on this pin indicates that the data is to be written into memory or an I/O port.

Memory/I/O ($\overline{M/I\overline{O}}$) : This signal identifies whether the current bus cycle is memory cycle or I/O cycle. Logic 0 on this pin indicates that the current bus cycle is I/O cycle whereas logic 1 indicates that the current cycle is memory cycle.

Data/Control ($\overline{D/C}$) : This signal identifies whether the current bus cycle is data or control cycle. These signal is logic 0 for instruction fetch, interrupt acknowledge, and halt/shut down operations and logic 1 for memory and I/O data read and write operations.

It is important to note that using $\overline{M/I\overline{O}}$, $\overline{D/C}$ and $\overline{W/R}$, we can identify the type of bus cycle that 80386 is currently executing.

The Table 13.5 shows the list of all possible bus cycles with their definition signals.

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

Table 13.5 Bus cycle definition signals and types of bus cycles

LOCK : This signal is used in multiprocessor systems. In multiprocessor systems resources are shared, such as global memory. The **LOCK** signal is used to ensure that the 80386DX has uninterrupted control of the system bus and the shared resource. By making **LOCK** output, 0 the 80386DX can lock up the shared resource for the other masters in the system.

Bus control signals :

The three bus control signals allow external logic to control the bus cycle. These signals are 1. **READY** 2. **Next Address (NA)** 3. **Bus Size 16 (BS16)**.

READY : It is used primarily to synchronize slower peripherals with the microprocessor. This signal is produced by the microcomputer's memory or I/O subsystem. When **READY** signal is logic 0, slow memory or I/O devices tell 80386DX that they are ready for next data transfer. If ready is logic 1 then processor enters wait state since logic 1 on ready pin indicates that, the data transfer of current cycle is not yet completed.

Next Address Request (NA) : The external bus control logic control this signal. It activates pipelining by making next address request input low. Pipelining increases the address to data access time. By increasing the address to data access time, same level of performance can be obtained with slower, memory devices.

BUS Size 16 (BS16) : This signal activates 16-bit data bus operation; data is transferred on the low-order 16-bits of the data bus, and an extra cycle is provided for transfers of more than 16-bits.

13.2.2 Interrupt Interface Signals

There are three interrupt interface signals :

1. Interrupt request (INTR)
2. Nonmaskable interrupt request (NMI)
3. System reset (RESET).

INTR : The INTR input of the 80386 allows external devices to interrupt 80386 program execution. This input is sampled at the beginning of each instruction cycle. To ensure recognition of interrupt by the 80386, the INTR input must be held high until the 80386 acknowledges the interrupt by performing the interrupt acknowledge cycles. Thus it must be high at least eight CLK2 periods prior to the instruction to guarantee recognition as a valid interrupt. This specific requirement reduces the false triggering.

Nonmaskable Interrupt (NMI) : As name indicates this interrupt input is non maskable. This input is edge-triggered. A valid interrupt on this pin causes 80386 to execute interrupt service routine. The 80386 will not service subsequent NMI requests until the current request has been serviced.