

Microprocessor, Microcontroller & Embedded System

Course Code: CSE-3523

Course Instructor:

Shefayatuj Johara Chowdhury

Assistant Lecturer

Dept. of CSE, IIUC

Segment 2:

8086 Microprocessor: Properties, architecture, registers, FLAGS register, physical address calculation, addressing modes, addressing Techniques of 8086 Microprocessor.


8086 Microprocessor is an enhanced version of 8085 Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20 address lines and 16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily.

It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

Features of 8086

The most prominent features of a 8086 microprocessor are as follows –

- It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It is available in 3 versions based on the frequency of operation –
 - 8086 → 5MHz
 - 8086-2 → 8MHz
 - (c)8086-1 → 10 MHz

 It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.

- Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It has 256 vectored interrupts.
- It consists of 29,000 transistors.

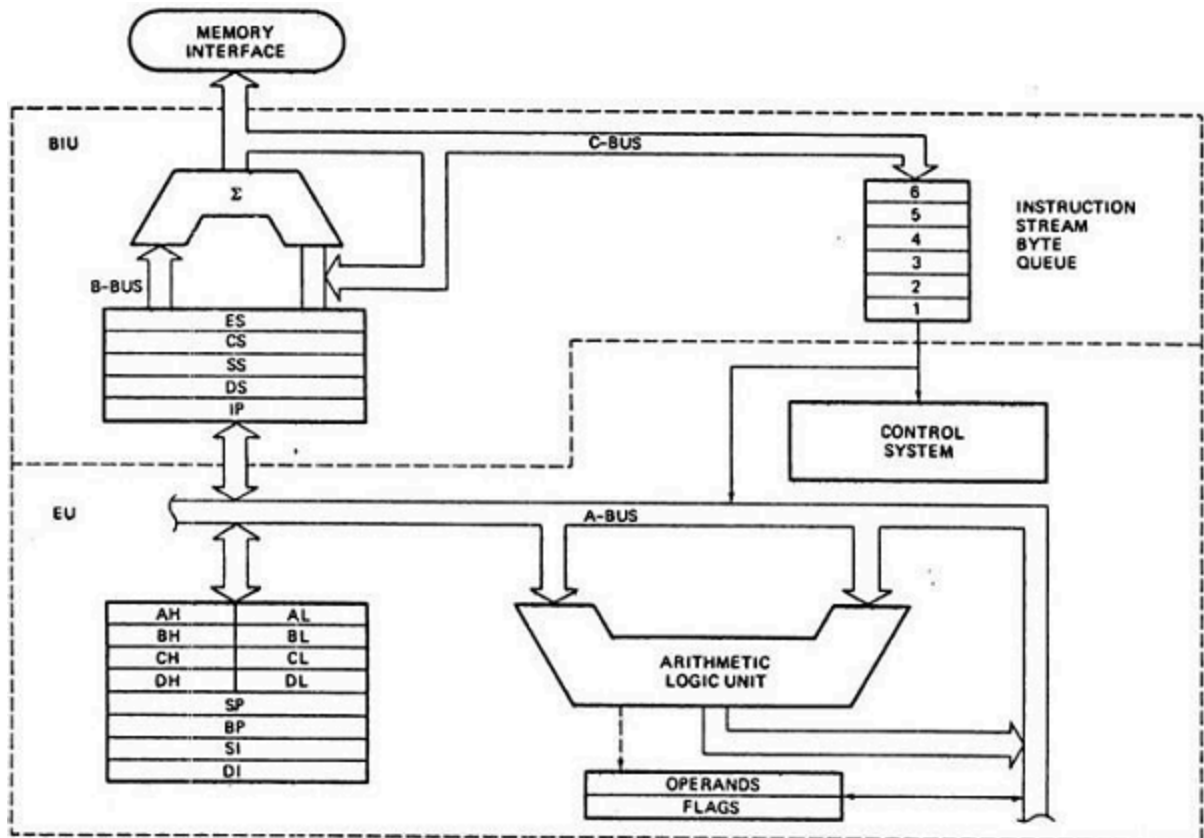
□ The microprocessor performs three main operations:

- 1. Data transfer between itself and the memory or I/O system.

- 2. Arithmetic or Logical operations.
- 3. Program flow via simple decisions.
- Although these are simple tasks, microprocessor performs any series of operations through these tasks.
- Addition, subtraction, multiplication. Division, AND, OR, NOT (Table-1-4, 25P Barry B. Brey (8th edition.))
- Decisions found in 8086 through 8086-Core 2 microprocessors are given below (Table-1-5, 26P Barry B. Brey (8th edition.)):
- Zero: tests a number for zero or not.
- Sign : Tests whether a number is positive or negative.
- Carry: Tests for carries after addition and borrows after subtraction.
- Parity: Tests a number for even or odd number of ones.
- Overflow: Tests for an invalid result

Architecture of 8086

The following diagram depicts the architecture of a 8086 Microprocessor –



8086 Microprocessor is divided into two functional units, i.e., **EU** (Execution Unit) and **BIU** (Bus Interface Unit).

EU (Execution Unit)

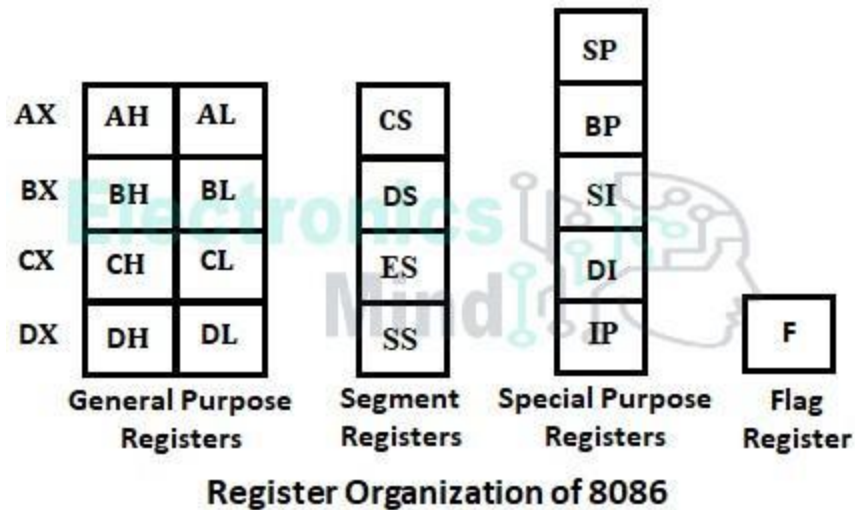
Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions. Its function is to control operations on data using the instruction decoder & ALU. EU has no direct connection with system buses as shown in the above figure, it performs operations over data through BIU.

Let us now discuss the functional parts of 8086 microprocessors.

ALU

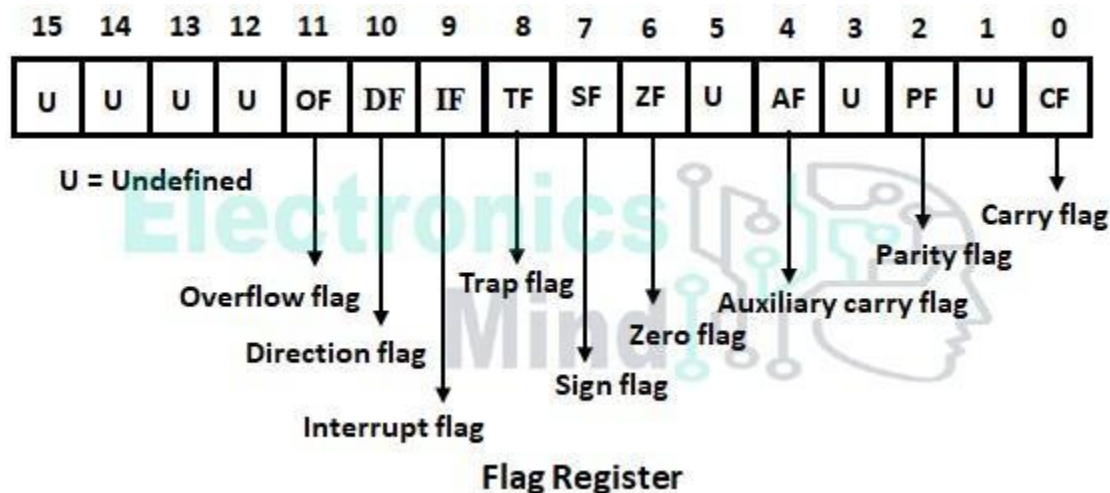
It handles all arithmetic and logical operations, like +, -, ×, /, OR, AND, NOT operations.

Register:



Flag Register

It is a 16-bit register that behaves like a flip-flop, i.e. it changes its status according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups – Conditional Flags and Control Flags.



Conditional Flags

It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags –

- **Carry flag** – This flag indicates an overflow condition for arithmetic operations.
- **Auxiliary flag** – When an operation is performed at ALU, it results in a carry/borrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.

- **Parity flag** – This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1's, then the Parity Flag is set. For odd number of 1's, the Parity Flag is reset.
- **Zero flag** – This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.
- **Sign flag** – This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.
- **Overflow flag** – This flag represents the result when the system capacity is exceeded.

Control Flags

Control flags controls the operations of the execution unit. Following is the list of control flags –

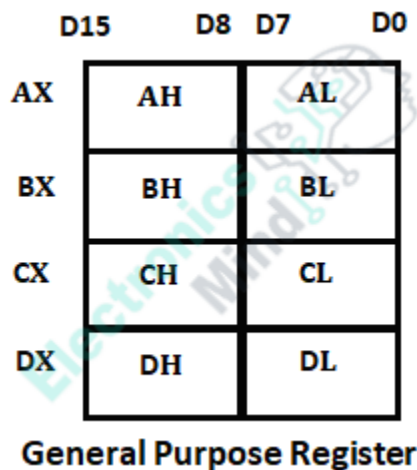
- **Trap flag** – It is used for single step control and allows the user to execute one instruction at a time for debugging. If it is set, then the program can be run in a single step mode.
- **Interrupt flag** – It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.
- **Direction flag** – It is used in string operation. As the name suggests when it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.

General purpose register

There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually to store 8-bit data and can be used in pairs to store 16bit data. The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.

- **RAX (accumulator):** Referenced as a 64 bit register (RAX), 32 bit register (EAX), 16 bit register (AX). AX's high byte is AH and low bytes is AL. It is called accumulator and used for instructions related to multiplication, division, etc.
- **RBX (Base index):** addressable in the same way as RAX. BX register holds the offset address of the memory, RBX can also address memory data.
- **RCX register** – It is referred to as counter. It is used in loop instruction to store the loop counter.

- RDX register – This register is used to hold I/O port address for I/O instruction.
- RBP register-RBP is another 32-bit register. This is base pointer register. This register is primary used in accessing the parameters passed by the stack. It's offset address relatives to stack segment.
- RDI register-This is destination index register. This is used to point destination in some string related operations. Its offset is relative to extra segment.
- RSI register- This is Source Index register. This is used to point the source in some string related operations. Its offset is relative to data segment.



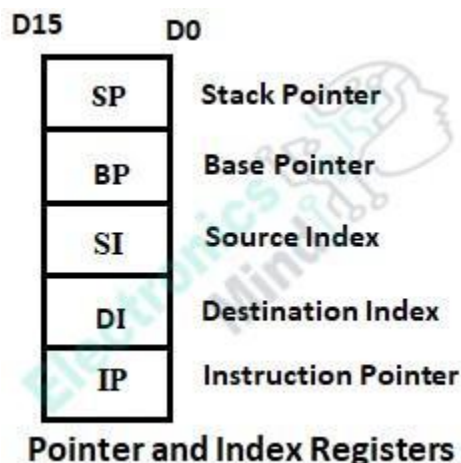
Stack pointer register

It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

Special purpose registers are following:

- RIP (Instruction Pointer): It addresses the next instruction in a section of memory defined as a code segment.
- RSP (Stack Pointer): RSP addresses an area of memory called stack.
- RFLAGS: FLAG registers indicate the condition of the microprocessor and control it's operation. Among these C (carry flag) indicates carry due to addition or borrow after subtraction.

- Parity (P) indicates even or odd parity.
- Z (Zero) indicates whether result of an arithmetic operation is 0 or non zero.
- S(sign) indicates the arithmetic sign of a number.
- Trap (T) is used for debugging. (When a system is instructed to single-step operation, it will execute one instruction and then stop. The contents of registers and memory locations can be examined; if they are correct, the system can be told to go on and execute the next instruction.)
- I(Interrupt) controls the operation of INTR (interrupt request) input pin.
- Overflow (O) indicates if an overflow occurred.

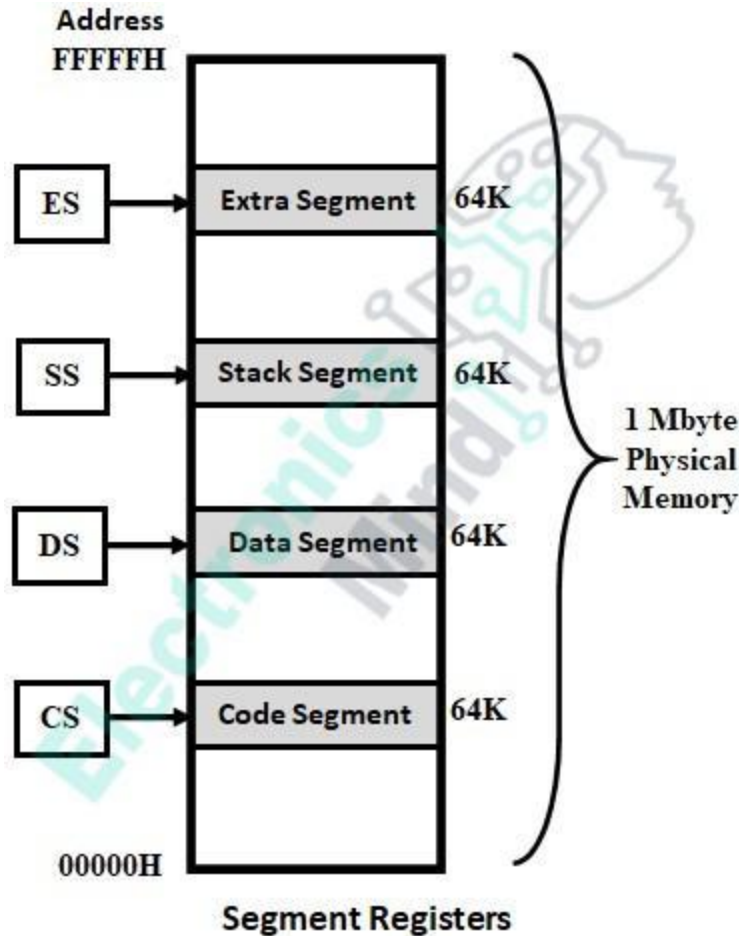


BIU (Bus Interface Unit)

BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory. EU has no direct connection with System Buses so this is possible with the BIU. EU and BIU are connected with the Internal Bus.

It has the following functional parts –

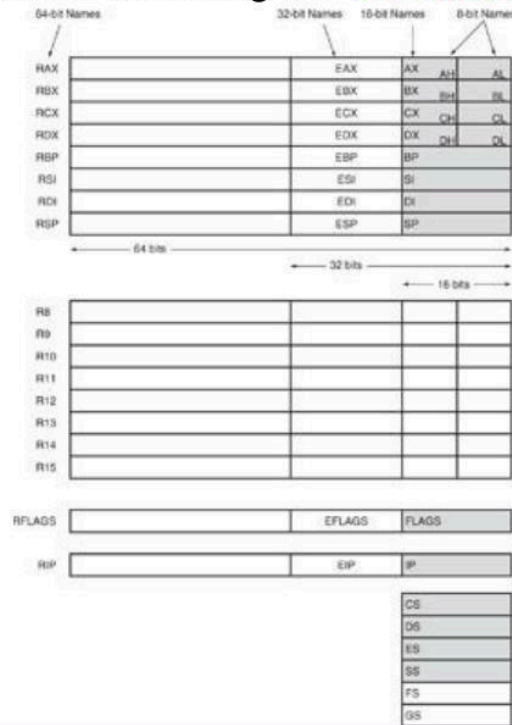
- **Instruction queue** – BIU contains the instruction queue. BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed.
- Fetching the next instruction while the current instruction executes is called **pipelining**.
- **Segment register** – BIU has 4 segment buses, i.e. CS, DS, SS& ES. It holds the addresses of instructions and data in memory, which are used by the processor to access memory locations. It also contains 1 pointer register IP, which holds the address of the next instruction to executed by the EU.
 - **CS** – It stands for Code Segment. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
 - **DS** – It stands for Data Segment. It consists of data used by the program and is accessed in the data segment by an offset address or the content of other register that holds the offset address.
 - **SS** – It stands for Stack Segment. It handles memory to store data and addresses during execution.
 - **ES** – It stands for Extra Segment. ES is additional data segment, which is used by the string to hold the extra destination data.
- **Instruction pointer** – It is a 16-bit register used to hold the address of the next instruction to be executed.



The default and specified segment registers for each memory pointer are tabulated as shown below.

Operation of Microprocessor	Memory Pointer (logical address)	Default Segment	Specified Segment
Microprocessor reading instruction from memory	IP	CS	None
Microprocessor storing stack of data (PUSH/POP)	SP, BP	SS	None
Source of string data	SI	DS	CS, ES, SS
Destination of string data	DI	ES	None
General data access	Effective address	DS	CS, ES, SS
BX used as a pointer	Effective address	DS	CS, ES, SS
BP used as a pointer	Effective address	SS	CS, ES, DS

Figure 2–1 The programming model of the 8086 through the Core2 microprocessor including the **64-bit extensions**.

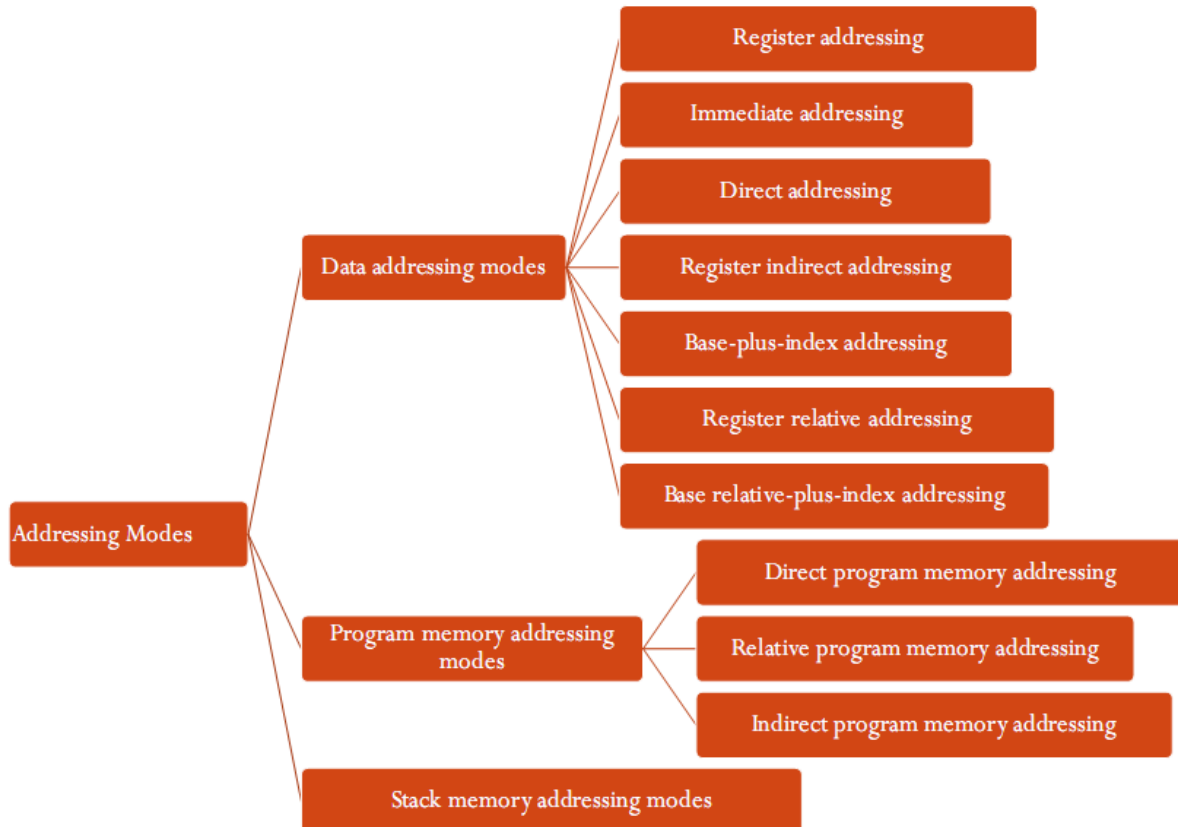


Addressing modes.

The operands of the instructions can be located either in the main memory or in the CPU registers. If the operand is placed in the main memory, then the

instruction provides the location address in the operand field. Many methods are followed to specify the operand address. The different methods/modes for specifying the operand address in the instructions are known as addressing modes.

Addressing Modes: classification



Addressing Modes: Definition and classification

Addressing Modes:

- Addressing mode provide different ways for access an address to given data to a processor.
- When 8086 executes an instruction, it performs the specified function on data. Operated data is stored in the memory location. There are various techniques to specify address of data. These techniques are called Addressing Modes.

Classification of Addressing Modes:

Data-Addressing Modes:

This mode is related to data transfer operation, that is, data is transferred either from the memory to internal registers of 8086 processors or from one register to another register.

Example: MOV AX, DX

Program Memory addressing Modes:

This mode involves program memory addresses during various operations.

Example: JMP AX, in this instruction, the code execution control jumps to the current code segment location addressed by the contents of AX register.

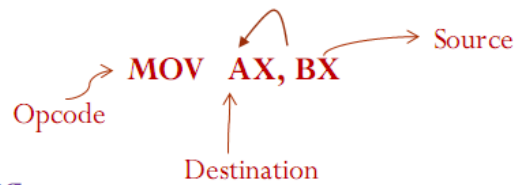
Stack memory addressing Modes:

This mode involves stack registry operations.

Example: PUSH AX, this instruction copies the contents of AX register to the stack.

Data addressing modes

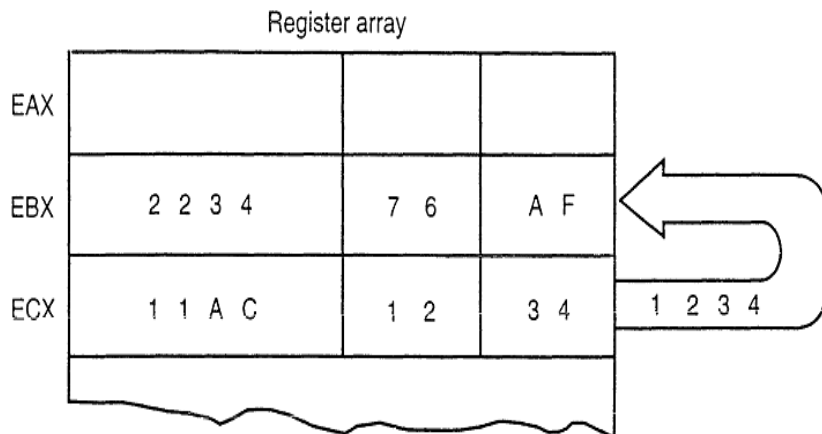
Note: We will use MOV instruction to explain all the data addressing modes.



Register addressing

- Register addressing transfers a copy of a byte or word from the source register to destination register. Register addressing is the most common form of data addressing and once the register names are learned, is the easiest to apply.
- 8-bit register names with register addressing: AH, AL, BH, BL, CH, CL, DH, DL.
- 16-bit register names: AX, BX, CX, DX, SP, BP, SI, DI, IP, CS, SS, DS and ES.
- Never mix an 8-bit register with 16-bit, it is not allowed in microprocessor.
- Code segment register (CS) is never used as destination.
- Segment to segment MOV instruction is not allowed.
- Example: MOV AL, BL ; Copies 8-bit content of BL into AL
 MOV AX, CX ; Copies 16-bit content of CX into AX
 MOV EX, DS ; Not allowed (segment to segment)
 MOV BL, DX ; Not allowed (mixed size)
 MOV CS, AX ; Not allowed (Code segment register may not be destination register)

FIGURE 3-3 The effect of executing the MOV BX, CX instruction at the point just before the BX register changes. Note that only the rightmost 16-bits of register EBX change.



Data addressing modes (Continued)

Immediate addressing

- Immediate addressing transfers the source, an immediate byte or word data, into the destination register.
- Immediate data means constant data, whereas data transferred from a register or memory location are variable data.
- Example: `MOV BL, 44` ; Copies 44 decimal (2CH) into BL
`MOV AX, 44H` ; Copies 0044H into AX
`MOV AL, 'A'` ; Copies ASCII A into AL

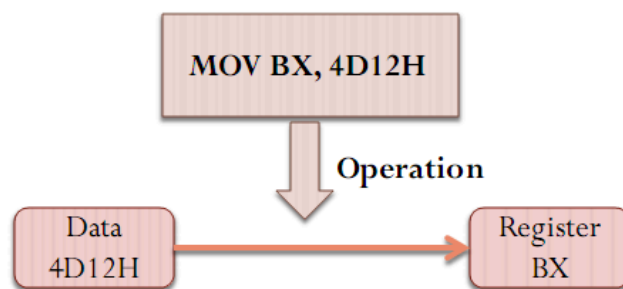
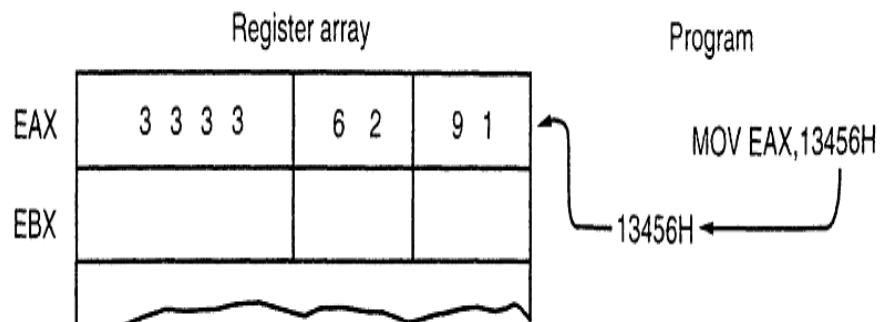


FIGURE 3-4 The operation of the `MOV EAX, 3456H` instruction. This instruction copies the immediate data (13456H) into EAX.



Data addressing modes (Continued)

Direct data addressing

- Direct addressing moves a byte or word between a data segment memory location and register.
- The instruction set does not support a memory to memory transfer except with the MOVS instruction.
- There are two basic form of direct data addressing: (1) Direct addressing (2) Displacement addressing

Direct addressing:

- Direct addressing with a MOV instruction transfers data between a memory location, located within the data segment, and the AL (8-bit) or, AX (16-bit).
- A MOV instruction using this type of addressing is usually a three byte long instruction.
- Example: MOV AL, DS:[1234H]

Displacement addressing:

- Direct addressing with a MOV instruction transfers data between a memory location, located within the data segment, and registers other than AL or AX.
- It is almost identical to direct addressing except that the instruction is four byte wide instead of three.
- Example: MOV CL, DS:[1234H]

```
0000 A0 1234 R      MOV AL, [1234H]
0003 8A 0E 1234 R      MOV CL, [1234H]
```

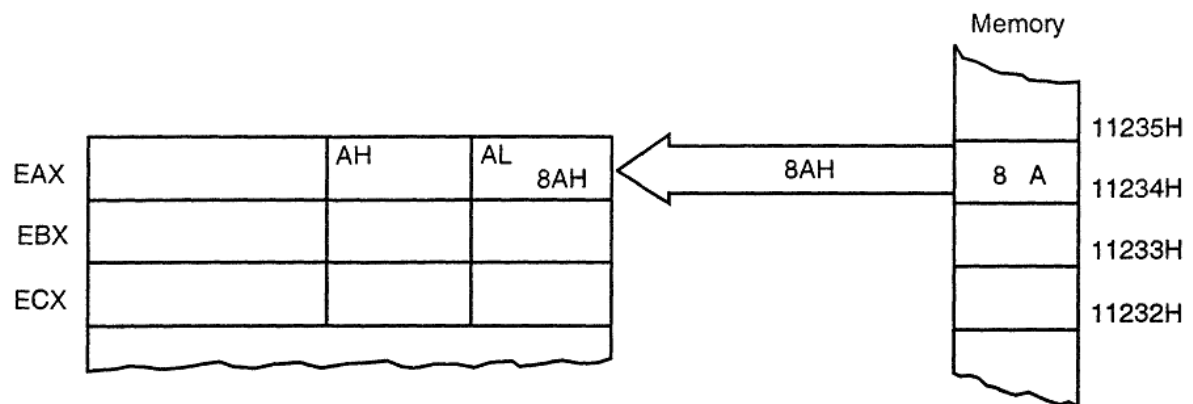


FIGURE 3–5 The operation of the MOV AL,[1234H] instruction when DS = 1000H

TABLE 3-3 Direct addressed instructions using EAX, AX and AL

<i>Assembly Language</i>	<i>Size</i>	<i>Operation</i>
MOV AL,NUMBER	8-bits	Copies the byte contents of data segment memory location NUMBER into AL
MOV AX,COW	16-bits	Copies the word contents of data segment memory location COW into AX
MOV EAX,WATER*	32-bits	Copies the doubleword contents of memory location WATER into EAX
MOV NEWS,AL	8-bits	Copies AL into data segment memory location NEWS
MOV THERE,AX	16-bits	Copies AX into data segment memory location THERE
MOV HOME,EAX*	32-bits	Copies EAX into data segment memory location HOME

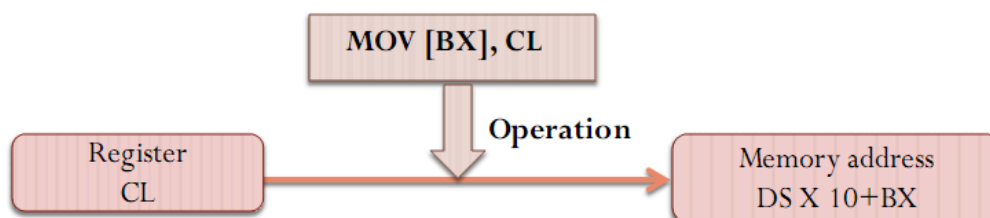
Data addressing modes (Continued)

Register Indirect addressing

- Register addressing transfers a byte or word between a register and memory location addressed by an index or base register.
- The index and base registers are BP, BX, DI and SI. These registers hold the offset address of the memory location.
- The data segment is used by default with register indirect addressing or any other addressing modes that uses BX, DI or, SI to address memory.
- If BP register addresses memory, the stack segment is used by default.
- The [] symbol denote indirect addressing in assembly language.
- Example: MOV CX, [BX] ; Copies the word contents of the data segment memory location addressed by BX into CX.

MOV [BP], DL ; Copies DL into stack segment memory location addressed by DI.

MOV [DI], [BX] ; Memory to memory transfers are not allowed except with string inst.



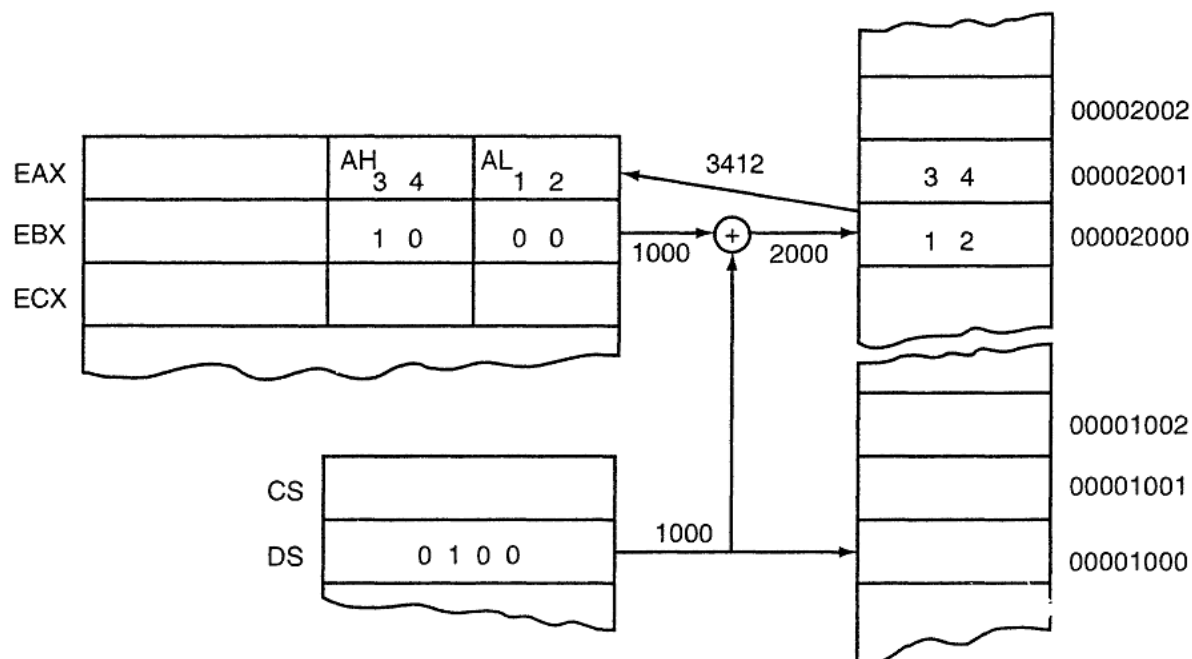


FIGURE 3-6 The operation of the `MOV AX,[BX]` instruction when `BX = 1000H` and `DS = 0100H`. Note that this instruction is shown after the contents of memory are transferred to `AX`.

TABLE 3-4 Examples of direct data addressing using a displacement

<i>Assembly Language</i>	<i>Size</i>	<i>Operation</i>
<code>MOV CH,DOG</code>	8-bits	Copies the byte contents of data segment memory location <code>DOG</code> into <code>CH</code>
<code>MOV CH,[1000H]*</code>	8-bits	Copies the byte contents of data segment offset address <code>1000H</code> into <code>CH</code>
<code>MOV ES,DATA6</code>	16-bits	Copies the word contents of data segment memory location <code>DATA6</code> into <code>ES</code>
<code>MOV DATA7,BP</code>	16-bits	Copies <code>BP</code> into data segment memory location <code>DATA7</code>
<code>MOV NUMBER,SP</code>	16-bits	Copies <code>SP</code> into data segment memory location <code>NUMBER</code>
<code>MOV DATA1,EAX</code>	32-bits	Copies <code>EAX</code> into data segment memory location <code>DATA1</code>
<code>MOV EDI,SUM1</code>	32-bits	Copies the doubleword contents of data segment memory location <code>SUM1</code> into <code>EDI</code>

*Note: This form of addressing is seldom used with most assemblers because an actual numeric offset address is rarely accessed.

Data addressing modes (Continued)

Base-Plus-Index addressing

- Base-plus-index addressing transfer a byte or word between a register and a memory location addressed by a base register (BP or BX) plus an index register (DI or SI).
- The base register often holds the beginning location of a memory array, whereas the index register holds the relative position of an element in the array.
- When BP addresses memory, stack segment is selected by default and when BX addresses memory data segment is selected by default.
- Example: `MOV CX, [BX+DI]` ; Copies the word content of the data segment memory location addressed by BX plus DI into CX.

`MOV [BP+DI], AH` ; Copies AH into stack segment memory location addressed by BP plus DI

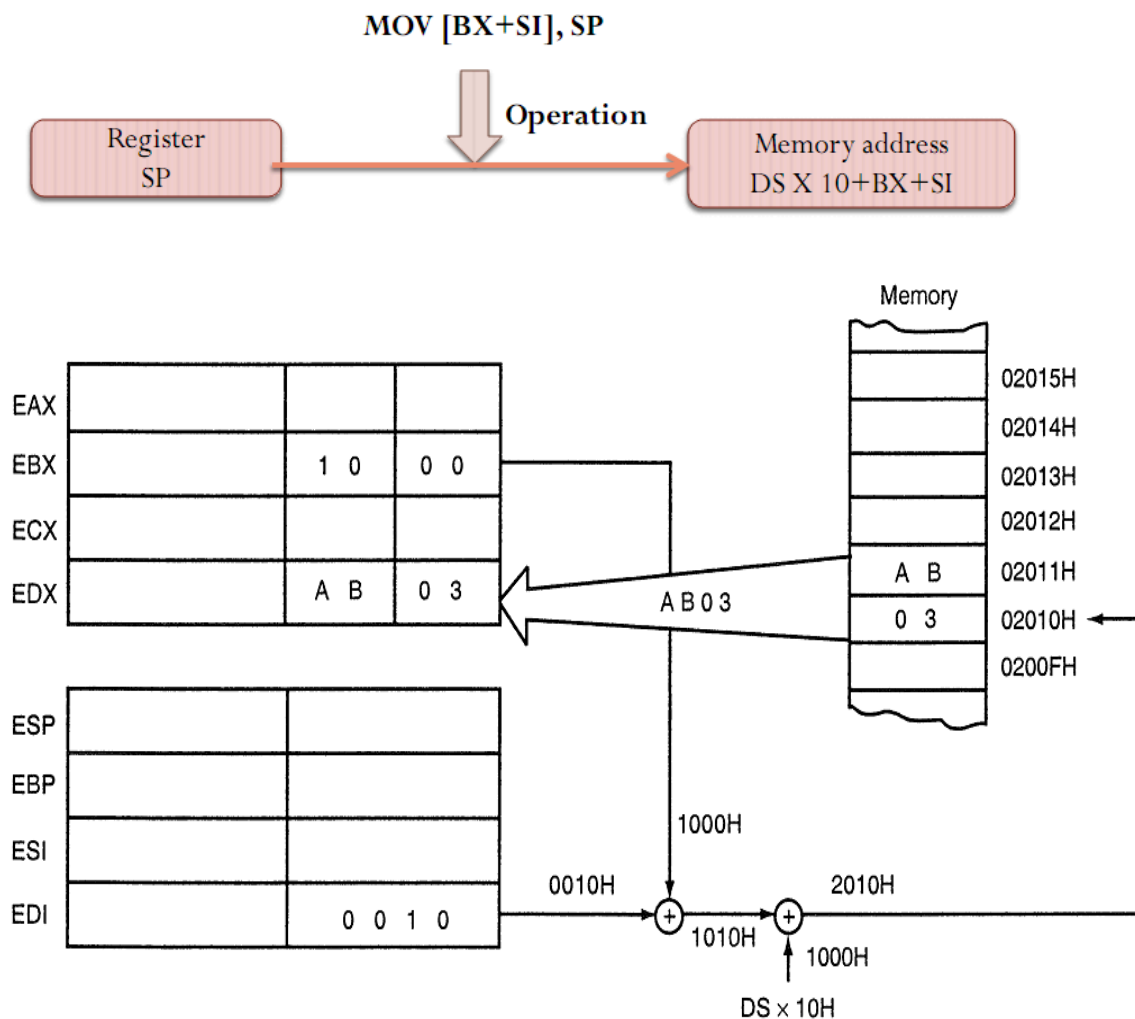


FIGURE 3-8 An example showing how the base-plus-index addressing mode functions for the `MOV DX,[BX+DI]` instruction. Notice that memory address 02010H is accessed because DS = 0100H, BX = 100H, and DI = 0010H.

Data addressing modes (Continued)

Register Relative addressing

- Register relative addressing moves a byte or word between a register and the memory location addressed by an index or base register (BP, BX, DI or SI) plus a displacement.
- Remember that BX, DI or SI addresses data segment and BP addresses the stack segment.
- Example: `MOV AX, [DI+100H]` ; Copies the word content of the data segment memory location addressed by DI plus 100H into AX.
`MOV ARRAY[SI], BL` ; Copies BL into the data segment memory location addressed by ARRAY plus SI

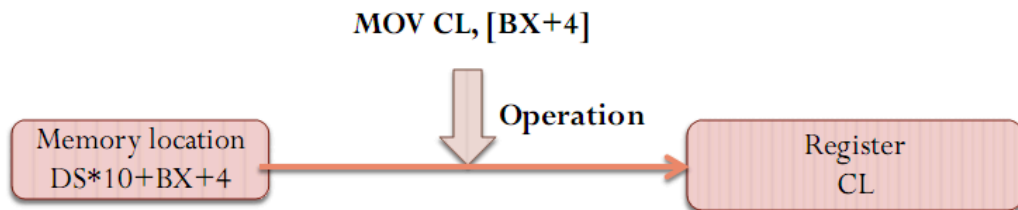
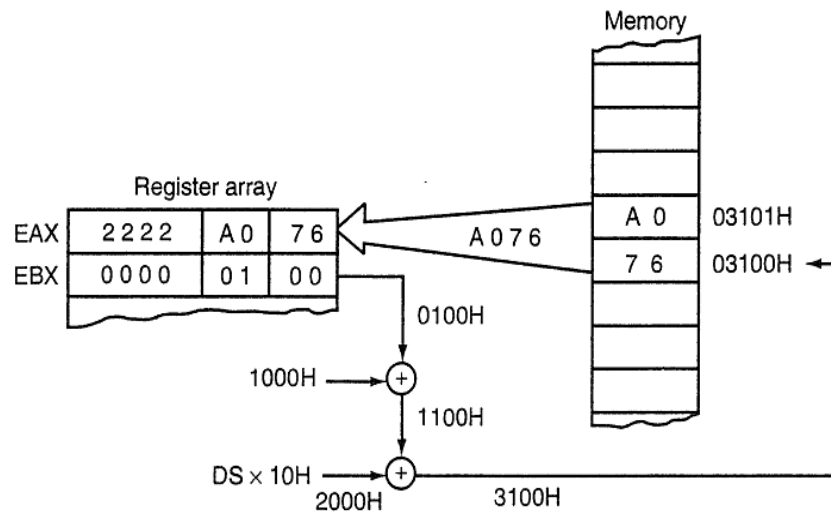


FIGURE 3-10 The operation of the `MOV AX, [BX+1000H]` instruction, when $BX = 0100H$ and $DS = 0200H$



Data addressing modes (Continued)

Base Relative plus Index addressing

- Base relative plus index addressing transfers a byte or word between a register and a memory location addressed by a base and an index register plus displacement.
- It is similar to base plus index addressing, but it adds a displacement beside using a base register and an index register.
- This type of addressing mode often addresses a two-dimensional array of memory data.
- Example: `MOV DH, [BX+DI+20H]` ; Copies the byte content of data segment memory location addressed by the sum of BX, DI and 20H into DH.

`MOV LIST[BP+DI], CL` ; Copies CL into the stack segment memory location addressed by the sum of LIST, BP, SI and 4.

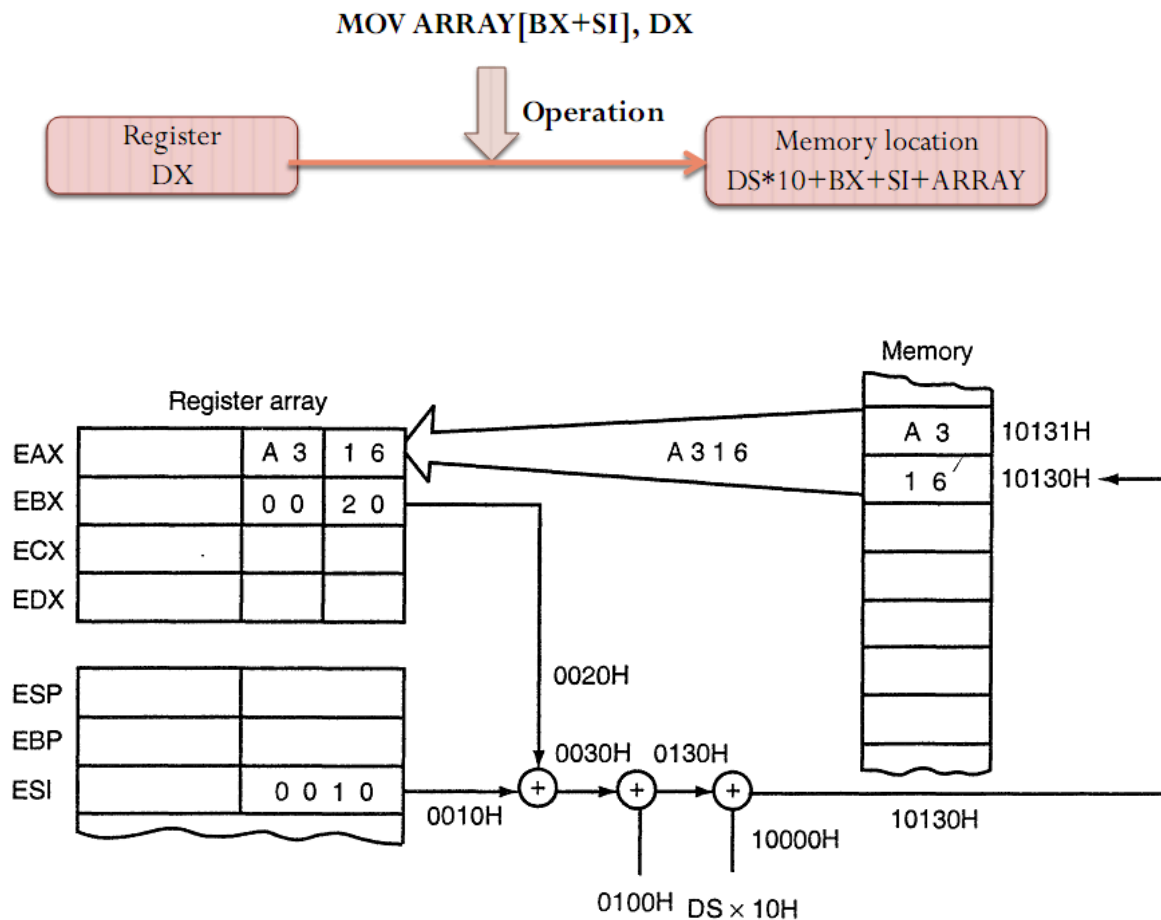


FIGURE 3–12 An example of base relative-plus-index addressing using a `MOV AX,[BX+SI+100H]` instruction. Note: DS = 1000H.

Addressing TECHNIQUES

- Real mode memory (DOS memory) exists at locations 00000H–FFFFFH, the first 1M byte of the memory system, and is present on all versions of the microprocessor. The first 1M byte of memory is called the real memory, conventional memory, or DOS memory system.
- Protected mode memory (Windows memory) exists at any location in the entire protected memory system, but is available only to the 80286–Core2, not to the earlier 8086 or 8088 microprocessors.

Real Mode Memory Addressing:

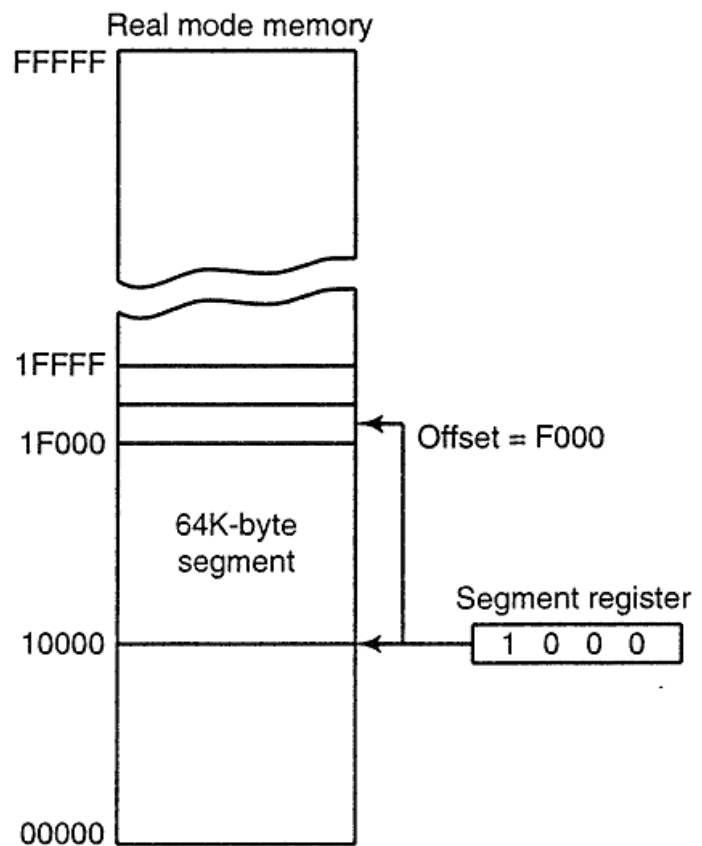
How is a 20-bit address obtained if there are only 16-bit registers?

- In the real mode,

Memory address=segment address*10+offset address

The segment address, located within one of the segment registers, defines the beginning address of any 64K-byte memory segment. The offset address selects any location within the 64K byte memory segment.

FIGURE 2-3 The real mode memory-addressing scheme, using a segment address plus an offset



- All offsets are limited to 16-bits. It means that the maximum size possible for a segment is $2^{16} = 65,535$ bytes (64 KB). The offset of the first location within the segment is 0000H. The offset of the last location in the segment is FFFF H.

<i>Segment Register</i>	<i>Starting Address</i>	<i>Ending Address</i>
2000H	20000H	2FFFFH
2001H	20010H	3000FH
21000H	21000H	30FFFFH
AB00H	AB000H	BAFFFFH
1234H	12340H	2233FH

Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

The contents of the following registers are:

CS = 1111 H

DS = 3333 H

SS = 2526 H

IP = 1232 H

SP = 1100 H

DI = 0020 H

Calculate the corresponding physical addresses for the address bytes in CS, DS and SS.

See exercise: 13, 14, 20, 21

Answer

1. CS = 1111 H

The base address of the code segment is 11110 H.

Effective address of memory is given by $11110H + 1232H = 12342H$.

2. DS = 3333 H

The base address of the data segment is 33330 H.

Effective address of memory is given by $33330H + 0020H = 33350H$.

3. SS = 2526 H

The base address of the stack segment is 25260 H.

Effective address of memory is given by $25260H + 1100H =$

26350H.

Protected Mode Memory Addressing

Protected mode memory addressing (80286 and above) allows access to data and programs located above the first 1M byte of memory, as well as within the first 1M byte of memory. Protected mode is where Windows operates. Addressing this extended section of the memory system requires a change to the segment plus an offset addressing scheme used with real mode memory addressing. The segment register contains a **selector** that selects a **descriptor** from a descriptor table. The descriptor describes the memory segment's location, length, and access rights. The **global descriptors** contain segment definitions that apply to all programs, whereas the **local descriptors** are usually unique to an application.

Descriptor Format:

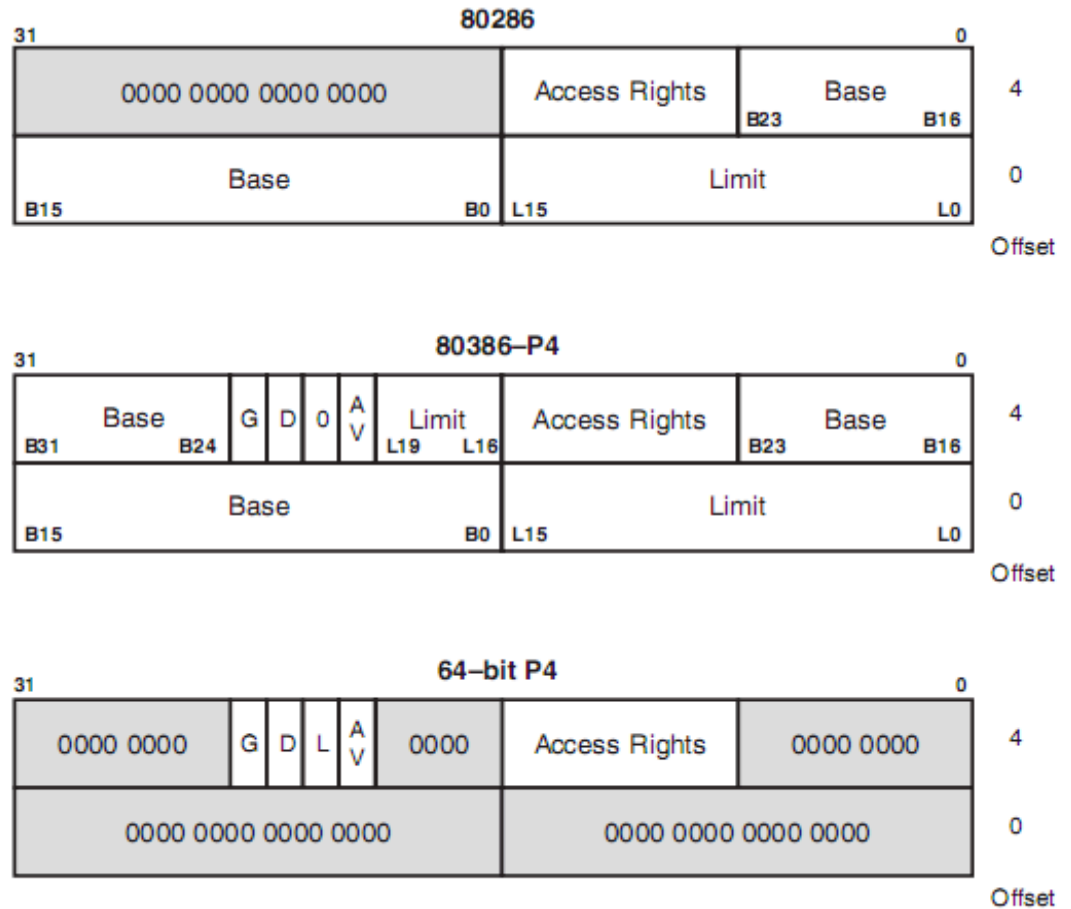


FIGURE 2-6 The 80286 through Core2 64-bit descriptors.

- The **base address** portion of the descriptor indicates the starting location of the memory segment.
- The **segment limit** contains the last offset address found in a segment. For example, if a segment begins at memory location F00000H and ends at location F000FFH, the base address is F00000H and the limit is FFH. For the 80286 microprocessor, the base address is F00000H and the limit is 00FFH.

- The **G bit**, or **granularity bit**. If G=0, the limit specifies a segment limit of 00000H to FFFFFH. If G=1, the value of the limit is multiplied by 4K bytes (appended with FFFH). The limit is then 00000FFFFH to FFFFFFFFH, if , this allows a segment length of 4K to 4G bytes in steps of 4K bytes.
- In the 64-bit descriptor, **the L bit** (probably means large, but Intel calls it the 64-bit) selects 64-bit addresses in a Pentium 4 or Core2 with 64-bit extensions when L=1 and 32-bit compatibility mode when L=0.
- The **AV bit**, in the 80386 and above descriptor, is used by some operating systems to indicate that the segment is available (AV=1) or not available (AV=0).
- The **D bit** indicates how the 80386 through the Core2 instructions access register and memory data in the protected or real mode. If D=0, the instructions are 16-bit instructions, compatible with the 8086–80286 microprocessors. This means that the instructions use 16-bit offset addresses and 16-bit register by default. If D=1, the instructions are 32-bit instructions. By default, the 32-bit instruction mode assumes that all offset addresses and all registers are 32 bits. Note that the default for register size and offset address is overridden in both the 16- and 32-bit instruction modes.

- The **access rights** byte controls access to the protected mode segment. This byte describes how the segment functions in the system.

Segment Register Format:

- Descriptors are chosen from the descriptor table by the segment register. Figure 2–8 shows how the segment register functions in the protected mode system.

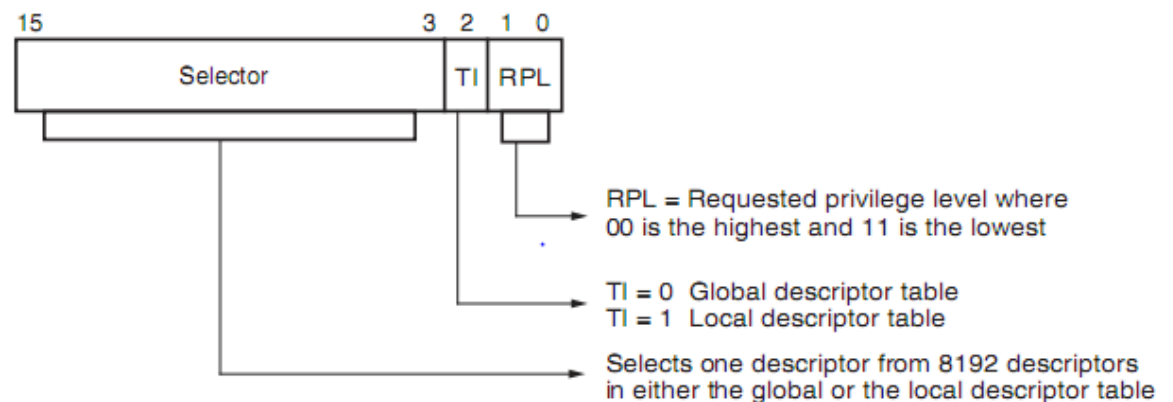


FIGURE 2–8 The contents of a segment register during protected mode operation of the 80286 through Core2 microprocessors.

- The segment register contains a 13-bit selector field, a table selector bit, and a requested privilege level field.
- The **13-bit selector** chooses one of the 8192 descriptors from the descriptor table.
- The **TI bit** selects either the global descriptor table (TI=0) or the local descriptor table (TI=1).

- The **requested privilege level (RPL)** requests the access privilege level of a memory segment. The highest privilege level is 00 and the lowest is 11. If the requested privilege level matches or is higher in priority than the privilege level set by the access rights byte, access is granted.

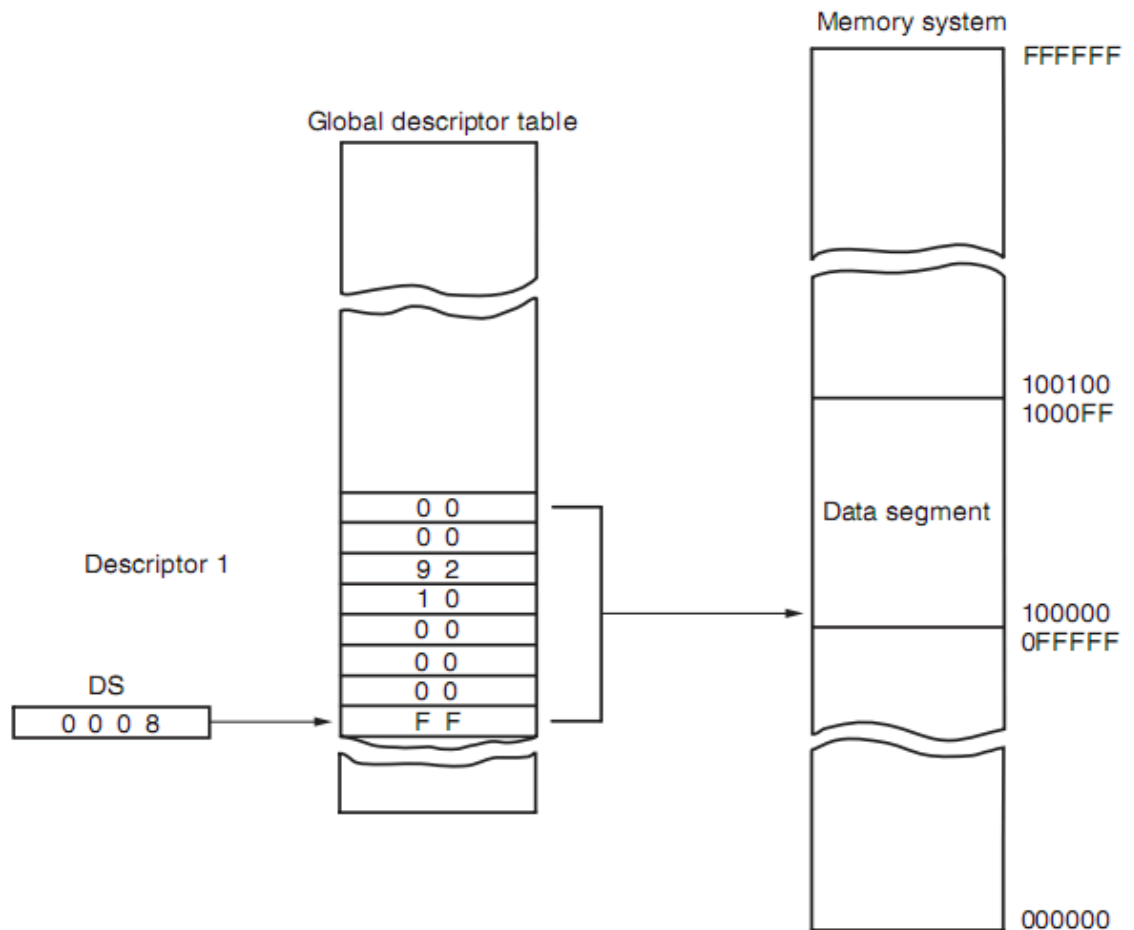


FIGURE 2–9 Using the DS register to select a description from the global descriptor table. In this example, the DS register accesses memory locations 00100000H–001000FFH as a data segment.

- Figure 2–9 shows how the segment register, containing a selector, chooses a descriptor from the global descriptor table. The entry in the global descriptor table selects a segment in the memory system.

- DS contains 0008H, which accesses the descriptor number 1 from the global descriptor table using a requested privilege level of 00.
- Descriptor number 1 contains a descriptor that defines the base address as 00100000H with a segment limit of 000FFH. This means that a value of 0008H loaded into DS causes the microprocessor to use memory locations 00100000H–001000FFH for the data segment with this example descriptor table.

Memory-addressing techniques Memory capacity from range:

- The number of things in a range can be found from the following relationship

- $(\text{Upper_Limit} - \text{Lower_Limit}) + 1$

- Example 1 - How many integers are in the range 3 to 18?

Answer $(18 - 3) + 1 = 16$

- How many addresses in the range 0x00000 to 0x7FFFF?

Answer $(0x7FFFF - 0x00000) + 1 = 0x80000H$ (This many hexadecimal numbers are in this range)

- Convert Hex to Decimal to find

$0x80000(\text{Hex}) = 524,288 (\text{Decimal})$

- So, Capacity for the given memory range is 524,288 Bytes (Because each decimal address has capacity 1 byte).

References

- **The Intel Microprocessors (8th edition) by Berry B. Brey**
- **Other web sources**
- **Israt Binteh Habib, Lecturer, Dept. of CSE IIUC**