Data Transfer Instructions

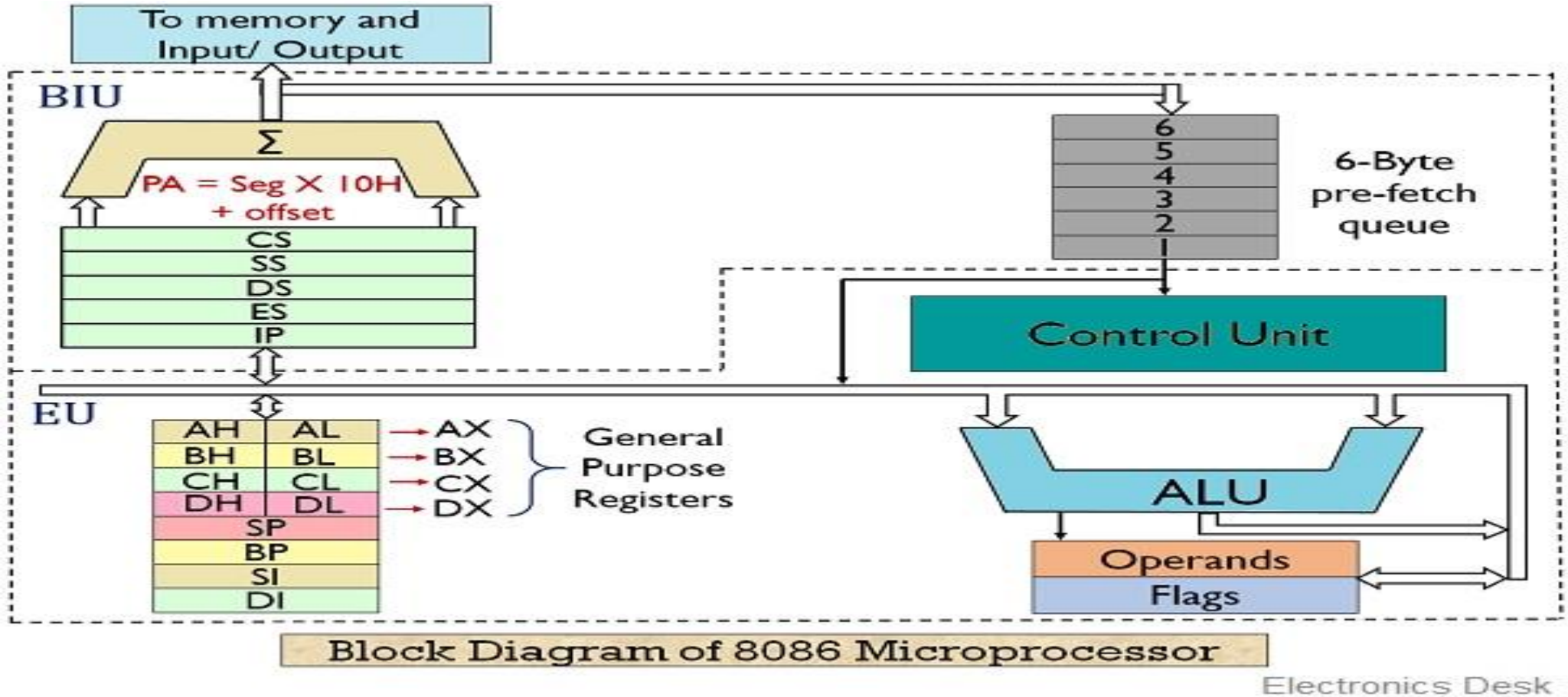

To memory and Input/ Output
BIU
Σ
PA = Seg X 10H + offset
CS
SS
DS
ES
IP
6
5
4
3
2
1
6-Byte pre-fetch queue
Control Unit
EU
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
AX
BX
CX
DX
General Purpose Registers
ALU
Operands
Flags
Block Diagram of 8086 Microprocessor
Electronics Desk

Figure (1)

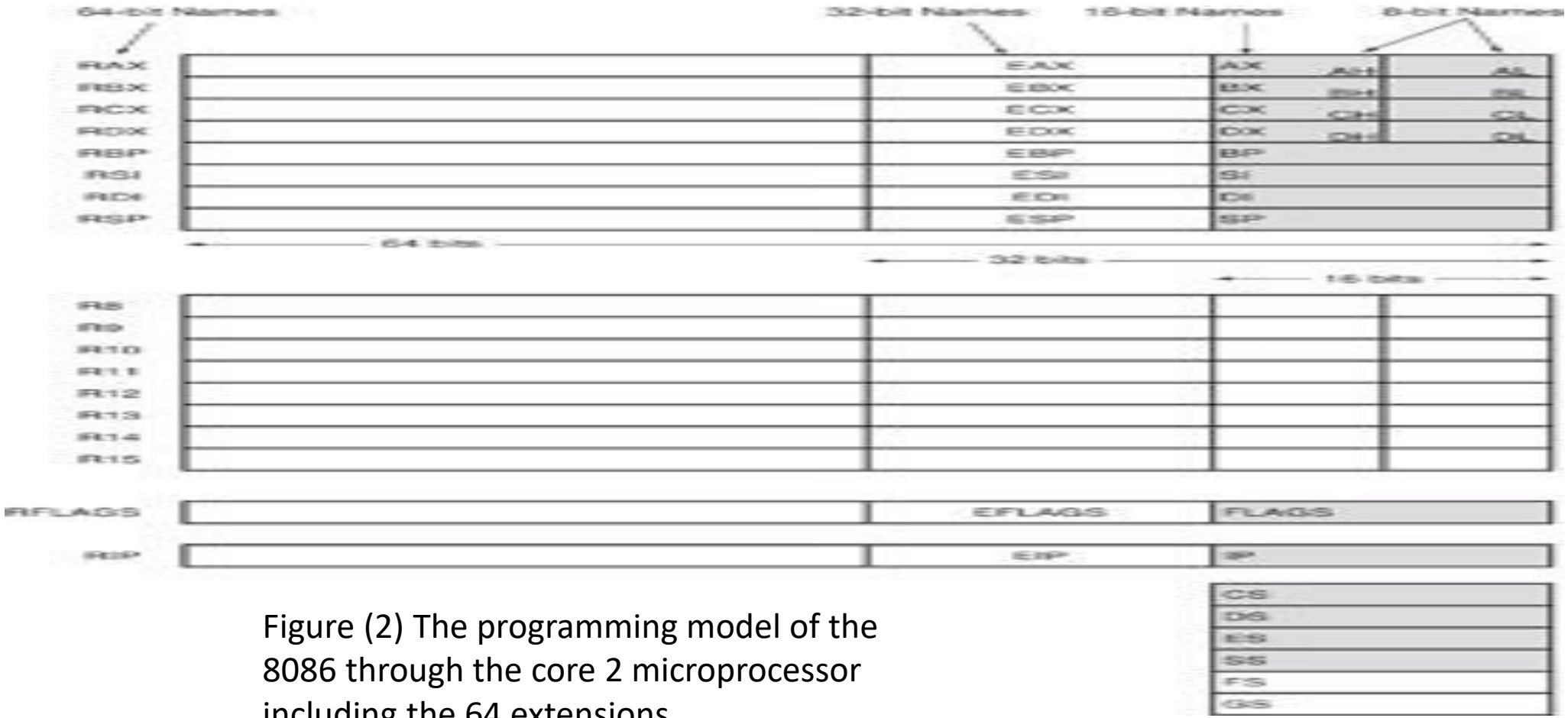
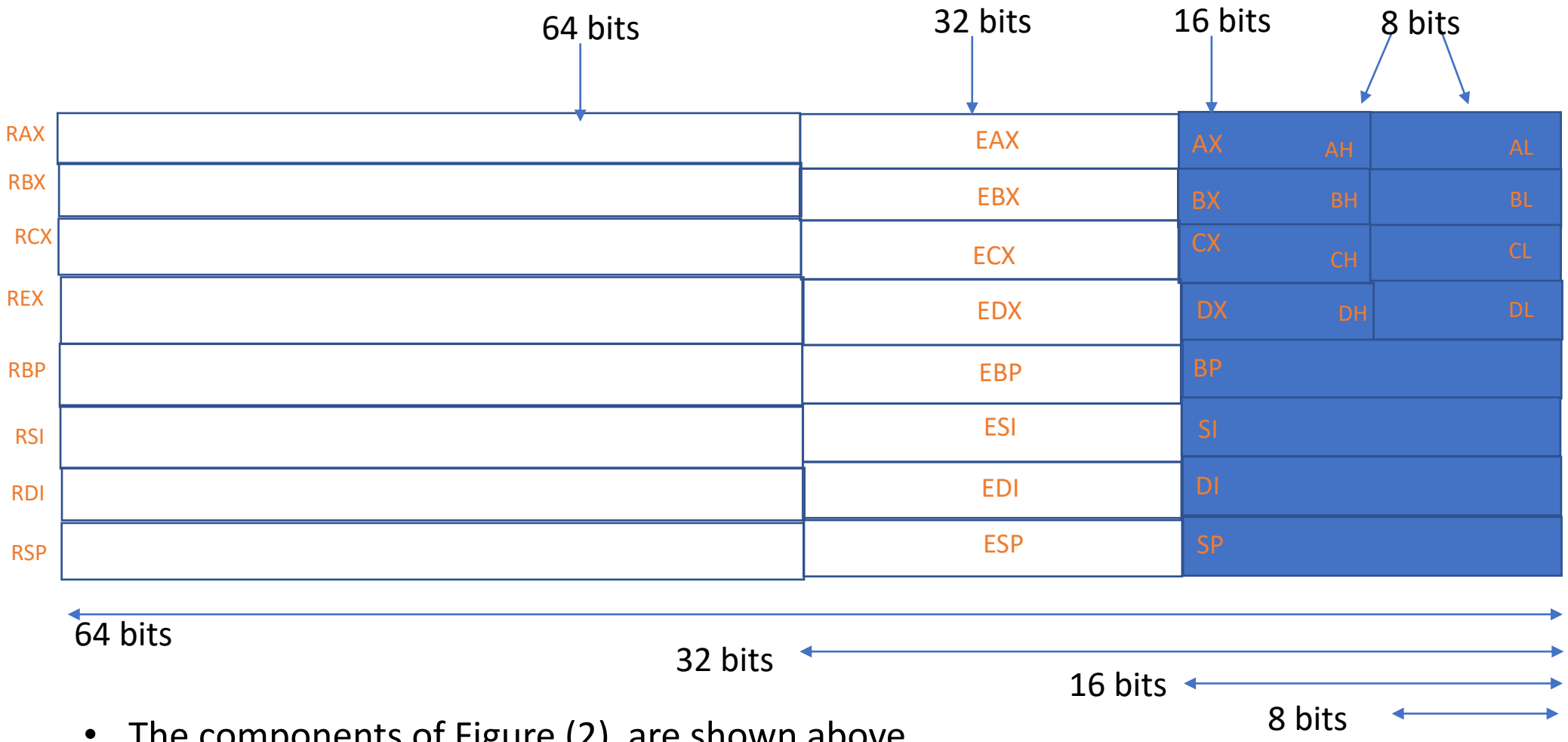
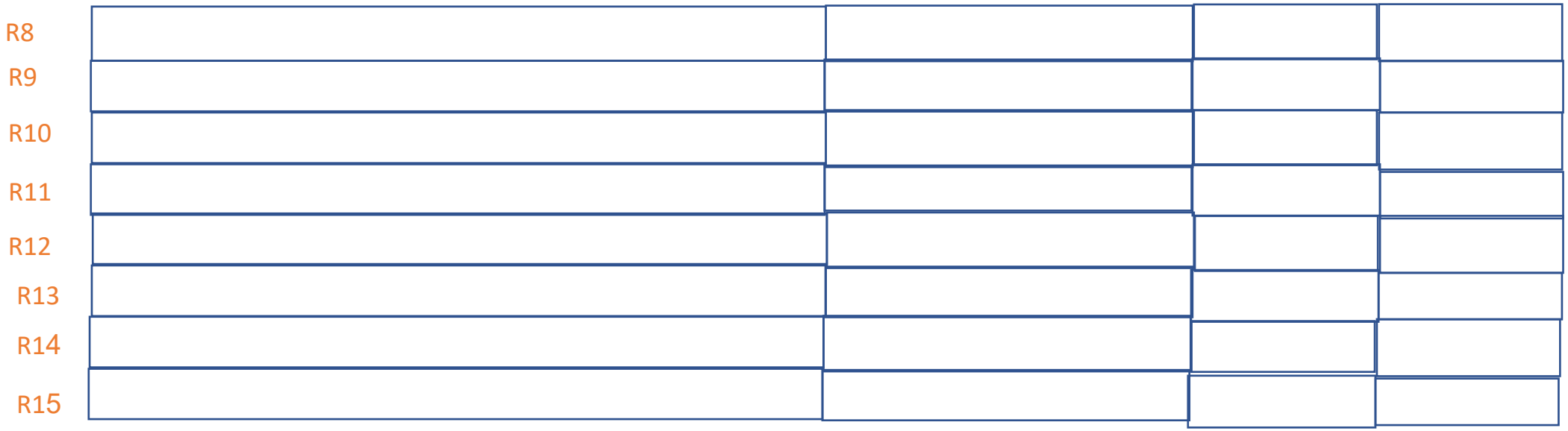


Figure (2) The programming model of the 8086 through the core 2 microprocessor including the 64 extensions



- The components of Figure (2) are shown above.



- The components of Figure (2) are shown above.

- The components of Figure (2) are shown above.

- In 80386 & above, extended 32-bit register names are: EAX, EBX, ECX, EDX, ESP, EBP, EDI, and ESI.
- 64-bit mode register names are: RAX, RBX, RCX, RDX, RSP, RBP, RDI, RSI, and R8 through R15.
- The programming model of the 8086 through 80286 contains 8-bit and 16-bit registers. The programming model of the 80386 and above contains 8-bit, 16-bit, and 32-bit extended registers as well as two additional 16-bit segment registers: FS and GS.
- The 8-bit registers are AH, AL, BH, BL, CH, CL, DH, and DL. The 16-bit registers are AX,BX, CX, DX, SP, BP, DI, and SI. The segment registers are CS, DS, ES, SS, FS, and GS. The 32-bit extended registers are EAX, EBX, ECX, EDX, ESP, EBP, EDI, and ESI. The 64-bit registers in a Pentium 4 with 64-bit extensions are RAX, RBX, RCX, RDX, RSP, RBP, RDI,RSI, and R8 through R15. In addition, the microprocessor contains an instruction pointer (IP/EIP/RIP) and flag register (FLAGS, EFLAGS, or RFLAGS).
- All real mode memory addresses are a combination of a segment address plus an offset address. The starting location of a segment is defined by the 16-bit number in the segment register that is appended with a hexadecimal zero at its rightmost end. The offset address is a 16-bit number added to the 20-bit segment address to form the real mode memory address.

- All instructions (code) are accessed by the combination of CS (segment address) plus IP or EIP (offset address).
- Data are normally referenced through a combination of the DS (data segment) and either an offset address or the contents of a register that contains the offset address. The 8086–Core2 use BX, DI, and SI as default offset registers for data if 16-bit registers are selected. The80386 and above can use the 32-bit registers EAX, EBX, ECX, EDX, EDI, and ESI as default offset registers for data. The default 16 bit segment and offset combinations (in 16-bit registers)are listed below.

Segment	Offset	Special Purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8- or 16-bit number	Data address
ES	DI for string instructions	String destination address

The default 16 bit segment and offset combinations (in 32-bit registers)are listed below.

Segment	Offset	Special Purpose
CS	EIP	Instruction address
SS	ESP or EBP	Stack address
DS	EAX, EBX, ECX, EDX, ESI, EDI, an 8- or 32-bit number	Data address
ES	EDI for string instructions	String destination address
FS	No default	General address
GS	No default	General address

Addressing mode

- An addressing mode is a method of specifying an operand. The 8086 addressing modes categorized into three types:

1.Register Addressing

2. Immediate Addressing

3. Memory Addressing

Register addressing mode

In this addressing mode, the operands may be (in 16-bit registers):

- reg16: 16-bit general registers: AX, BX, CX, DX, SI, DI, SP or BP.
- reg8 : 8-bit general registers: AH, BH, CH, DH, AL, BL, CL, or DL.
- Segment reg : segment registers: CS, DS, ES, or SS. There is an exception: CS cannot be a destination.

- In 80386 & above, extended 32-bit register names are: EAX, EBX, ECX, EDX, ESP, EBP, EDI, and ESI.
- 64-bit mode register names are: RAX, RBX, RCX, RDX, RSP, RBP, RDI, RSI, and R8 through R15.
- Important for instructions to use registers that are the same size.
  –never mix an 8-bit with a 16-bit register, an 8-or a 16-bit register with a 32-bit register
  –this is not allowed by the microprocessor and results in an error when assembled

Register Addressing

- For register addressing modes, there is no need to compute the effective address. The operand is in a register and to get the operand there is no memory access involved.

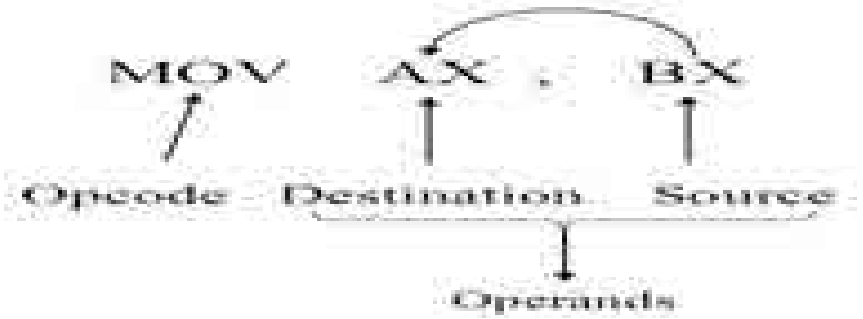


FIGURE (3) The MOV instruction showing the source, destination, and direction of

Register Addressing

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.

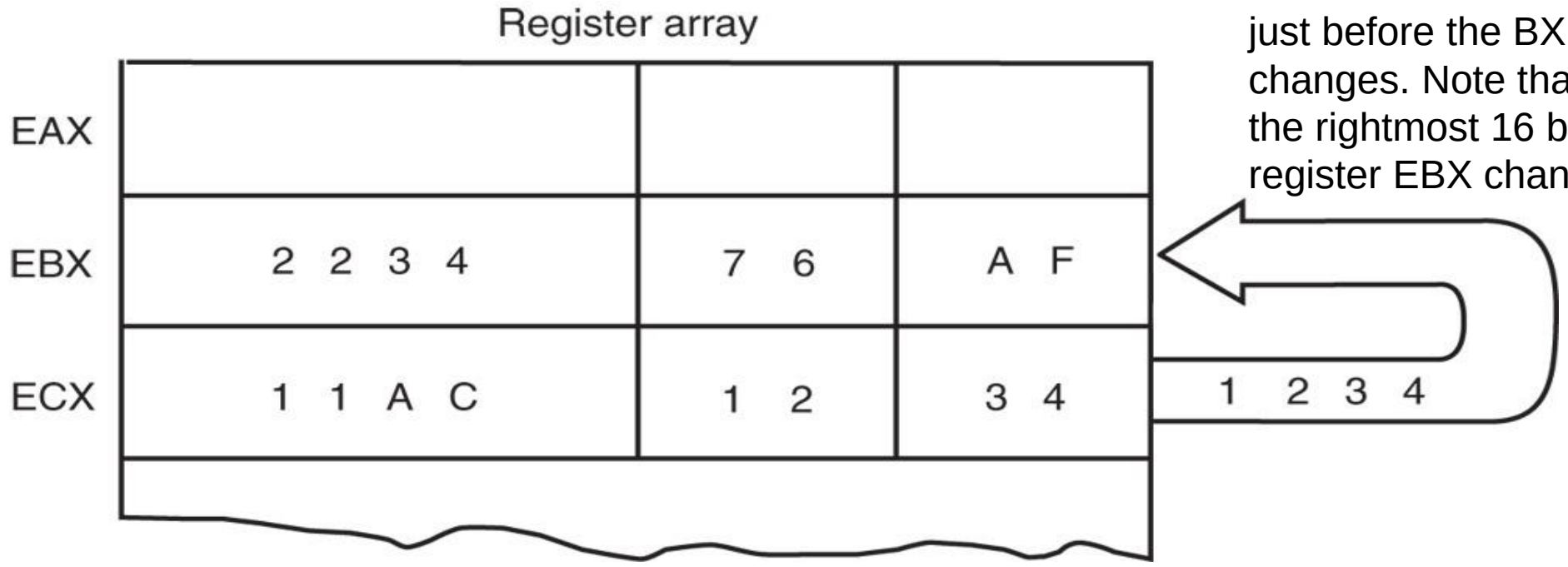


FIGURE (4) The effect of executing the MOV BX, CX instruction at the point just before the BX register changes. Note that only the rightmost 16 bits of register EBX change.

Register Addressing

- Figure (4) shows the operation of the MOV BX, CX instruction.
- The source register’s contents do not change.
  - the destination register’s contents do change
- The contents of the destination register or destination memory location change for all instructions except the CMP and TEST instructions.
- The MOV BX, CX instruction does not affect the leftmost 16 bits of register EBX.

Examples of register addressing

- MOV AX,BX copy contents of BX into AX
- MOV CL,DH ;copy contents of DH into CL
- MOV CL,CH ;copy contents of CH into CL
- MOV EAX,EBX ;copy contents of EBX into EAX
- MOV EBX,EAX ;copy contents of EAX into EBX
- MOV ECX,EAX ;copy contents of EAX into ECX
- MOV EDX,EAX ;copy contents of EAX into EDX
- MOV AX,CS ;copy CS into DS (two steps)
- MOV DS,AX
- MOV CS,AX ; copy AX into CS (causes problems)

Immediate Addressing

- Term *immediate* implies that data immediately follow the hexadecimal opcode in the memory.
  - immediate data are constant data
  - data transferred from a register or memory location are variable data
- Immediate addressing operates upon a byte or word of data.
- Figure (5) shows the operation of a MOV EAX,13456H instruction.
- The operation of the MOV EAX,13456H instruction. This instruction copies the immediate data (13456H) into EAX.

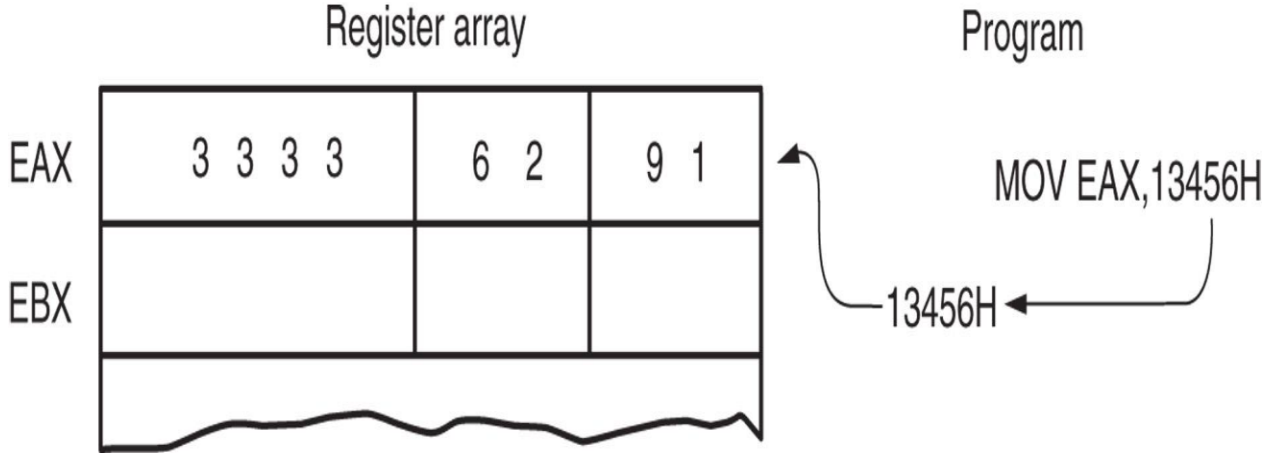


FIGURE (5) The operation of the MOV EAX,13456H instruction. This instruction copies the immediate data

Immediate Addressing

- In symbolic assembly language, the symbol # precedes immediate data in some assemblers.
  - MOV AX,#3456H instruction is an example
- Most assemblers do not use the # symbol,
  but represent immediate data as in the MOV AX,3456H instruction.
- The symbolic assembler portrays immediate data in many ways.
- The letter **H** appends hexadecimal data.
- If hexadecimal data begin with a letter, the assembler requires the data start with a **0**.
  - to represent a hexadecimal F2, 0F2H is used
    in assembly language
- Decimal data are represented as is and require no special codes or adjustments.
  - an example is the 100 decimal in the
    MOV AL,100 instruction

Immediate Addressing

- An ASCII-coded character or characters may be depicted in the immediate form if the ASCII data are enclosed in apostrophes.
  - be careful to use the apostrophe (‘) for ASCII data and not the single quotation mark (‘)
- Binary data are represented if the binary number is followed by the letter B.
  - in some assemblers, the letter Y

Immediate Addressing

- Each statement in an assembly language program consists of four parts or fields.
- The leftmost field is called the *label.*
  - used to store a symbolic name for the memory location it represents
- All labels must begin with a letter or one of the following special characters: @, $, -, or ?.
  - a label may any length from 1 to 35 characters
- The label appears in a program to identify the name of a memory location for storing data and for other purposes.
- The next field to the right is the *opcode field.*
  - designed to hold the instruction, or opcode
  - the MOV part of the move data instruction is an example of an opcode
- Right of the opcode field is the *operand field.*
  - contains information used by the opcode
  - the MOV AL,BL instruction has the opcode MOV and operands AL and BL
- The *comment field,* the final field, contains a comment about the instruction(s).
  - comments always begin with a semicolon (;)

Examples of Immediate Addressing

Assembly Language	Size	Operation
MOV BL,44	8 bits	Copies 44 decimal (2CH) into BL
MOV AX,44H	16 bits	Copies 0044H into AX
MOV SI,0	16 bits	Copies 0000H into SI
MOV CH,100	8 bits	Copies 100 decimal (64H) into CH
MOV AL,'A'	8 bits	Copies ASCII A into AL
MOV AH,1	8 bits	Not allowed in 64-bit mode, but allowed in 32-or 16-bit modes
MOV AX,'AB'	16 bits	Copies ASCII BA* into AX
MOV CL,11001110B	8 bits	Copies 11001110 binary into CL
MOV EBX,12340000H	32 bits	Copies 12340000H into EBX
MOV ESI,12	32 bits	Copies 12 decimal into ESI
MOV EAX,100B	32 bits	Copies 100 binary into EAX
MOV RCX,100H	64 bits	Copies 100H into RCX

Direct Data Addressing

- Applied to many instructions in a typical program.
- Two basic forms of direct data addressing:
  - –direct addressing, which applies to a MOV between a memory location and AL, AX, or EAX
  - –displacement addressing, which applies to almost any instruction in the instruction set
- Address is formed by adding the displacement to the default data segment address or an alternate segment address.

Direct addressing with a MOV instruction transfers data between a memory location, located within the data segment, and the AL (8-bit), AX (16-bit), or EAX (32-bit) register.

- –usually a 3-byte long instruction
- MOV AL,DATA loads AL from the data segment memory location DATA (1234H).
  - –DATA is a symbolic memory location, while1234H is the actual hexadecimal location

Direct addressed instructions using EAX, AX, and AL and RAX in 64-bit mode.

- Assembly Language Size Operation

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 data segment location WATER into EAX

MOV NEWS,AL 8 bits Copies AL into byte memory location NEWS

MOV THERE,AX 16 bits Copies AX into word memory location THERE

MOV HOME,EAX* 32 bits Copies EAX into doubleword memory location HOME

MOV ES:[2000H],AL 8 bits Copies AL into extra segment memory at offset address 2000H

MOV AL,MOUSE 8 bits Copies the contents of location MOUSE into AL ; In 64-bit mode MOUSE can be any address

MOV RAX, HERE 64 bits Copies 8 bytes from memory location HERE

Direct Data Addressing

- The operation of the MOV AL,[1234H] instruction when DS=1000HThis instruction transfers a copy contents of memory location 11234H into AL. The effective address is formed by adding1234H (the offset address) and 10000H(the data segment address of 1000H times10H) in a system operating in the real mode

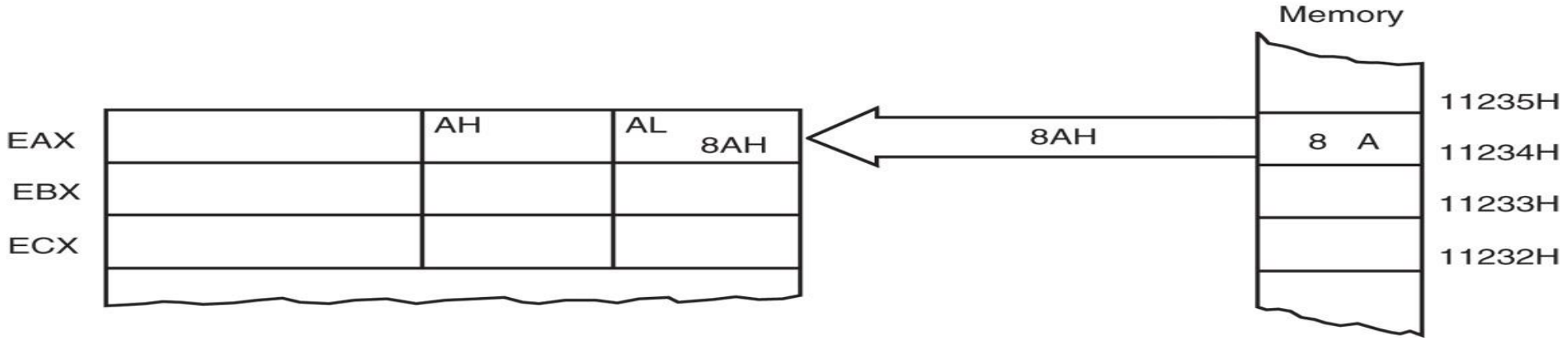


FIGURE (6) The operation of the MOV AL,[1234H] instruction when DS = 1000H

Displacement Addressing.

- Displacement addressing is almost identical to direct addressing, except that the instruction is 4 bytes wide instead of 3. In the 80386 through the Pentium 4, this instruction can be up to 7 bytes wide if both a 32-bit register and a 32-bit displacement are specified. This type of direct data addressing is much more flexible because most instructions use it.
- If the operation of the MOV CL,DS:[1234H] instruction is compared to that of the MOV AL,DS:[1234H] instruction of Figure (6), we see that both basically perform the same operation except for the destination register (CL versus AL). Another difference only becomes apparent upon examining the assembled versions of these two instructions. The MOV AL,DS:[1234H] instruction is 3 bytes long and the MOV CL,DS:[1234H] instruction is 4 bytes long.

Examples of direct data addressing using a displacement

Assembly Language	Size	Operation
MOV CH,DOG	8 bits	Copies the byte contents of data segment memory location DOG into CH
MOV CH,DS:[1000H]*	8 bits	Copies the byte contents of data segment memory offset address 1000H into CH
MOV ES,DATA6	16 bits	Copies the word contents of data segment memory location DATA6 into ES
MOV DATA7,BP	16 bits	Copies BP into data segment memory location DATA7
MOV NUMBER,SP	16 bits	Copies SP into data segment memory location NUMBER
MOV EDI,SUM1	32 bits	Copies the doubleword contents of data segment memory location SUM1 into EDI

Register Indirect Addressing

- Register indirect addressing allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI, and SI. For example, if register BX contains 1000H and the MOV AX,[BX] instruction executes, the word contents of data segment offset address 1000H are copied into register AX. If the microprocessor is operated in the real mode and , this instruction addresses a word stored at memory bytes 2000H and 2001H, and
- transfers it into register AX (see Figure (7)). Note that the contents of 2000H are moved into AL and the contents of 2001H are moved into AH. The [ ] symbols denote indirect addressing in assembly language. In addition to using the BP, BX, DI, and SI registers to indirectly address memory, the 80386 and above allow register indirect addressing with any extended register except ESP.

Register Indirect Addressing

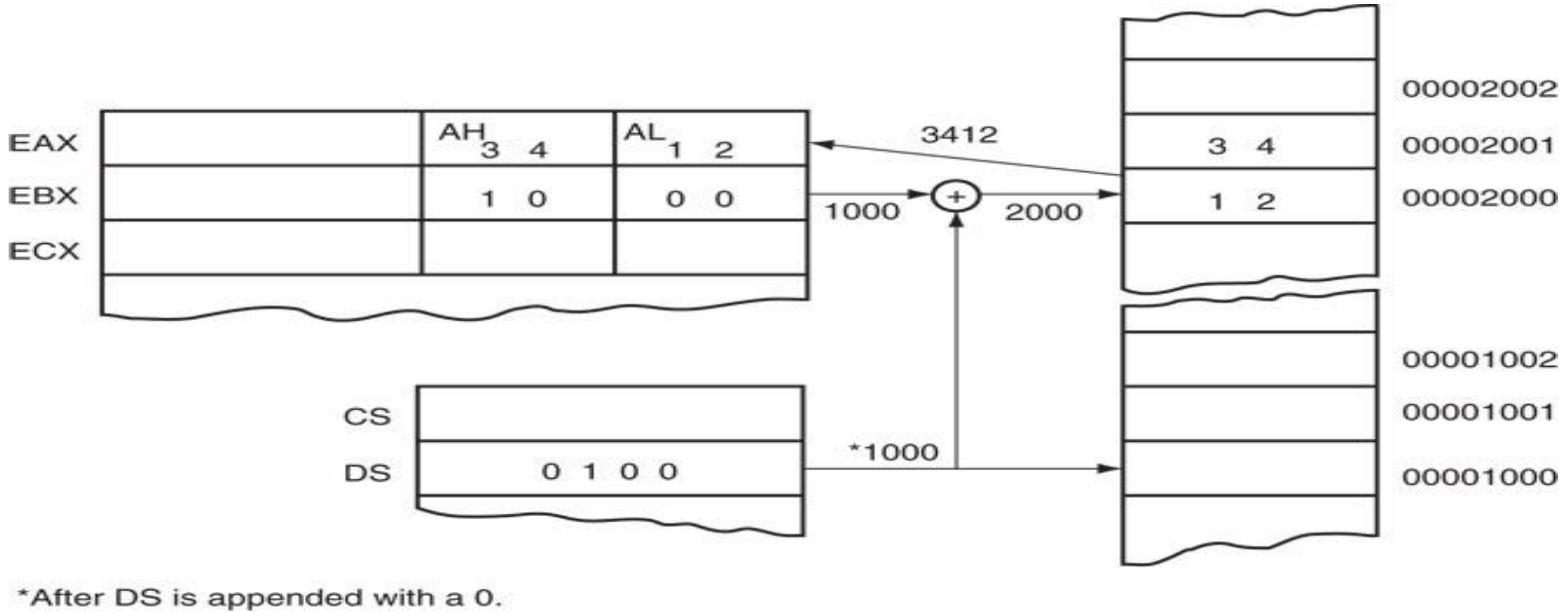


Figure (7)The operation of the MOV AX,[BX] instruction when BX =1000H and DS = 0100H. Note that this instruction is shown after the contents of memory are transferred to AX.

Examples of register indirect addressing

Assembly Language b	Size	Operation
MOV CX,[BX]	16 bits	Copies the word contents of the data segment memory location addressed by BX into CX
MOV [BP],DL*	8 bits	Copies DL into the stack segment memory location addressed by BP
MOV [DI],BH	8 bits	Copies BH into the data segment memory location addressed by DI
MOV [DI],[BX] —		Memory-to-memory transfers are not allowed except with string instructions
MOV AL,[EDX]	8 bits	Copies the byte contents of the data segment memory location addressed by EDX into AL
MOV ECX,[EBX]	32 bits	Copies the doubleword contents of the data segment memory location addressed by EBX into ECX
MOV RAX,[RDX]	64 bits	Copies the quadword contents of the memory location address by the linear address located in RDX into RAX

Register Indirect Addressing

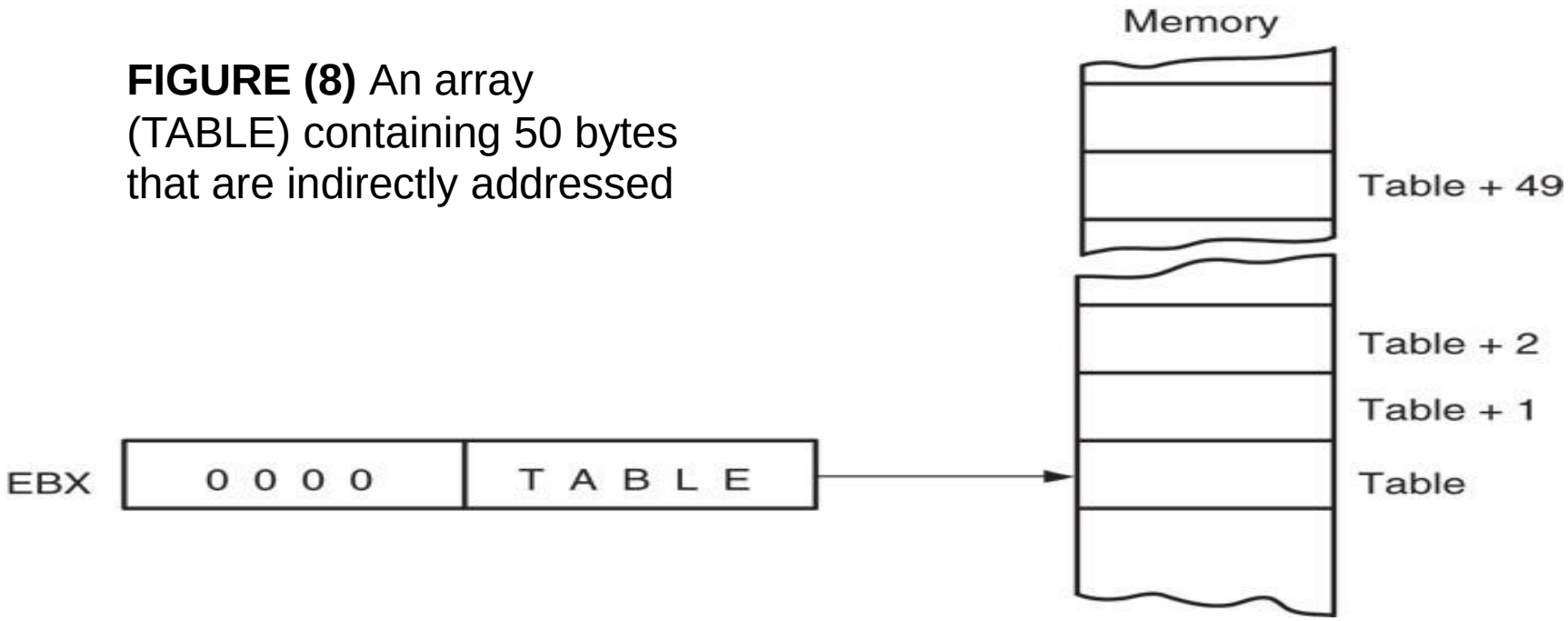
Indirect addressing often allows a program to refer to tabular data located in memory.

- Figure (8) shows the table and the BX register used to sequentially address each location in the table.
- To accomplish this task, load the starting location of the table into the BX register with a MOV immediate instruction.
- After initializing the starting address of the table, use register indirect addressing to store the 50 samples sequentially.

Register Indirect Addressing

An array (TABLE) containing 50 bytes that are indirectly addressed through register BX.

FIGURE (8) An array (TABLE) containing 50 bytes that are indirectly addressed



Base-Plus-Index Addressing

Similar to indirect addressing because it indirectly addresses memory data.

- The base register often holds the beginning location of a memory array.
  - –the index register holds the relative position of an element in the array
  - –whenever BP addresses memory data, both the stack segment register and BP generate the effective address.

Figure (9) shows how data are addressed by the MOV DX,[BX + DI] instruction when the microprocessor operates in the real mode.

Base-Plus-Index Addressing

•Figure (9) shows how data are addressed by the MOV DX,[BX + DI] instruction when the microprocessor operates in the real mode.

•

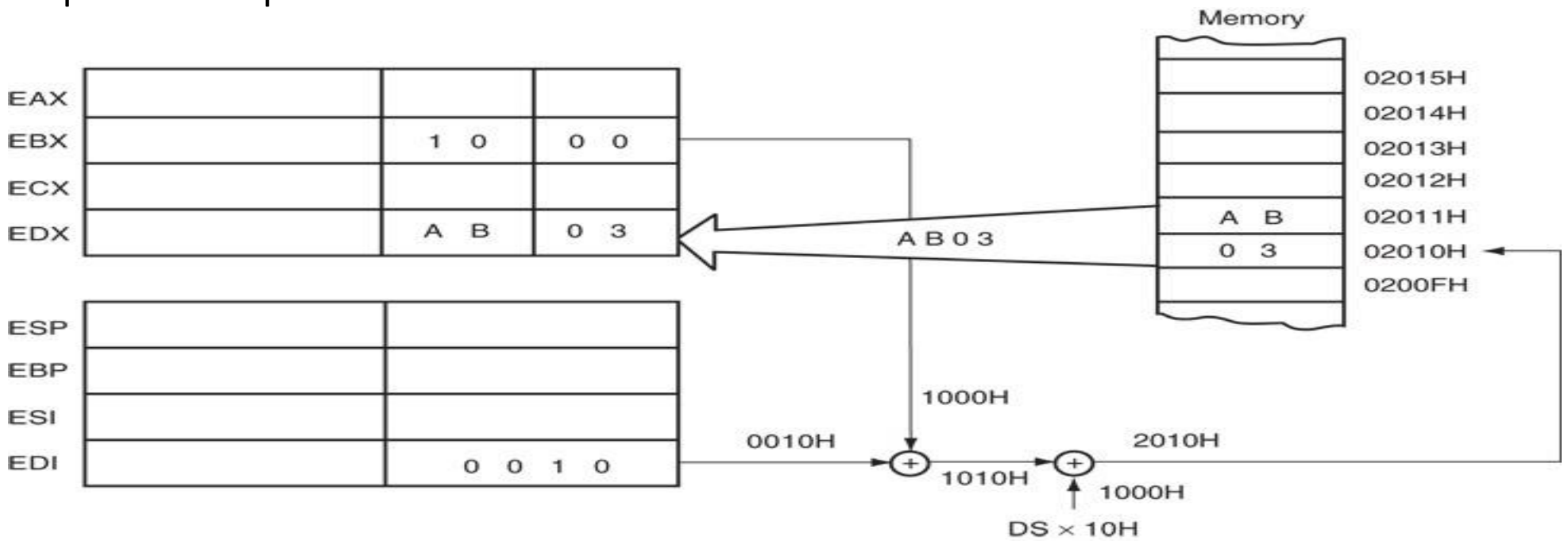


Figure (9)The operation of the MOV AX,[BX+DI] instruction when BX =1000H and DS = 0100H,DI=10H. Note that this instruction is shown after the contents of memory are transferred to AX.

Examples of base-plus-index addressing.

Assembly Language	Size	Operation
MOV CX,[BX+DI]	16 bits	Copies the word contents of the data segment memory location addressed by BX plus DI into CX
MOV CH,[BP+SI]	8 bits	Copies the byte contents of the stack segment memory location addressed by BP plus SI into CH
MOV [BX+SI],SP	16 bits	Copies SP into the data segment memory location addressed by BX plus SI
MOV [BP+DI],AH	8 bits	Copies AH into the stack segment memory location addressed by BP plus DI
MOV CL,[EDX+EDI]	8 bits	Copies the byte contents of the data segment memory location addressed by EDX plus EDI into CL
MOV [EAX+EBX],ECX	32 bits	Copies ECX into the data segment memory location addressed by EAX plus EBX

Register Relative Addressing

- Similar to base-plus-index addressing and displacement addressing.

–data in a segment of memory are addressed by adding the displacement to the contents of a base or an index register (BP, BX, DI, or SI)

- Figure (10) shows the operation of the MOV AX,[BX+1000H] instruction

Register Relative Addressing

The operation of the MOV AX, [BX+1000H] instruct•on, when BX=100H and DS=0200H .

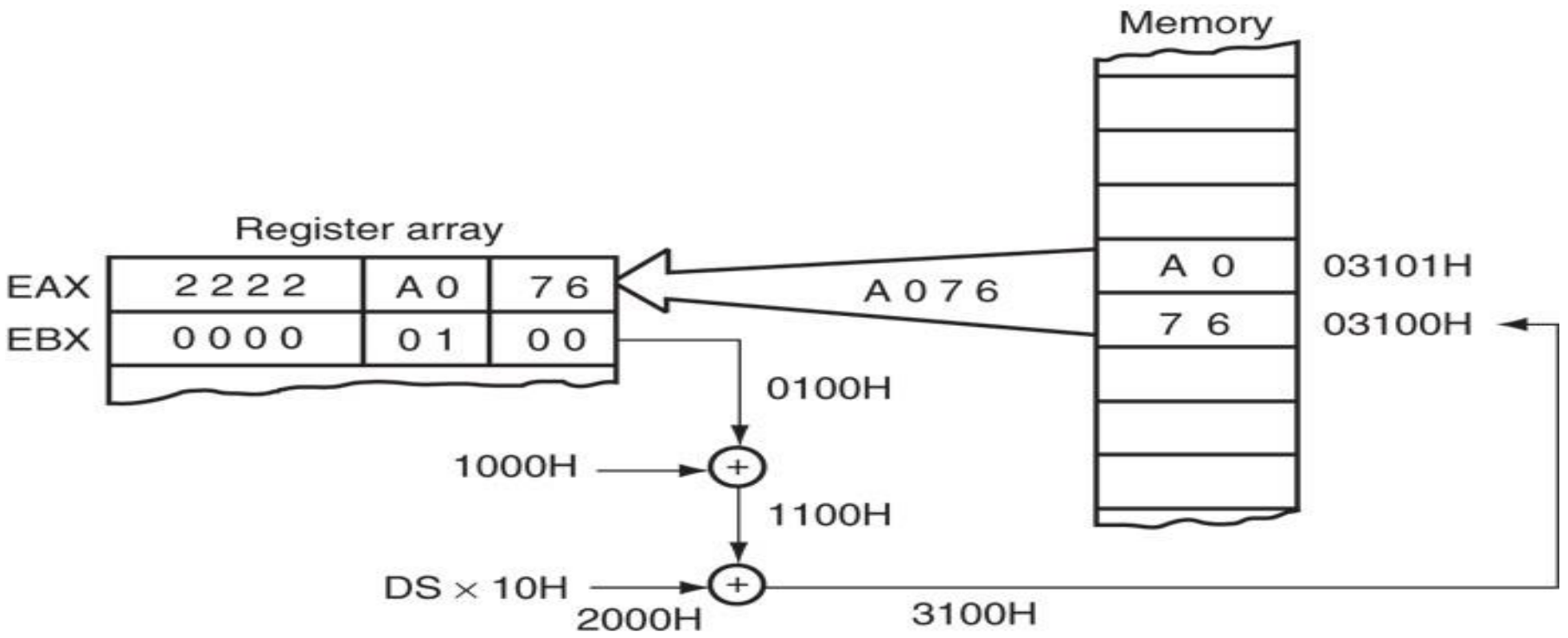


Figure (10)The operation of the MOV AX, [BX+1000H] instruction, when and DS = 0200H.BX = 0100H

Base Relative-Plus-Index Addressing

- The base relative-plus-index addressing mode is similar to base-plus-index addressing, but it adds a displacement, besides using a base register and an index register, to form the memory address. This type of addressing mode often addresses a two-dimensional array of memory data.
- Figure (11) shows how data are referenced if the instruction executed by the microprocessor is MOV AX,[BX + SI + 100H].
- –displacement of 100H adds to BX and SI to form the offset address within the data segment

Base Relative-Plus-Index Addressing

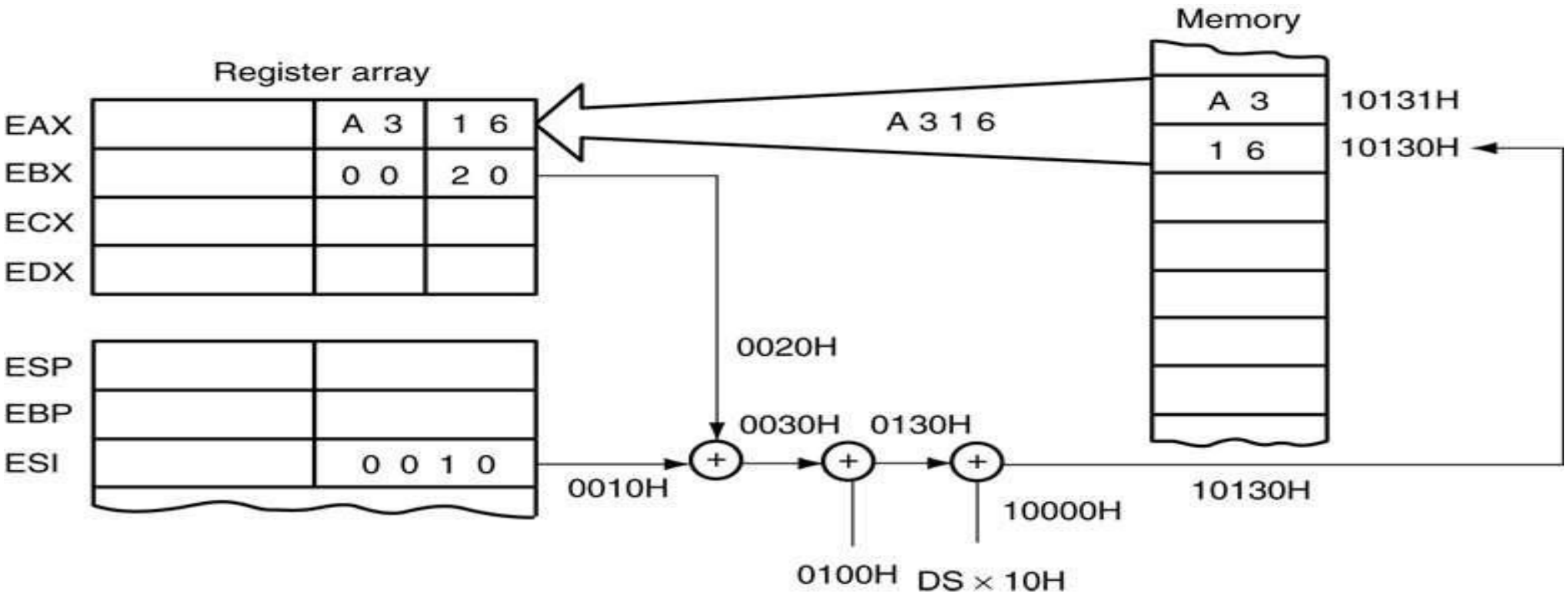


Figure (11) An example of base relative-plus-index addressing using a MOV AX,[BX+SI+100H] instruction. Note: DS=1000H

Scaled-Index Addressing

- Scaled-index addressing is the last type of data-addressing mode discussed. This data-addressing mode is unique to the 80386 through the Core2 microprocessors. Scaled-index addressing uses two 32-bit registers (a base register and an index register) to access the memory. The second register (index) is multiplied by a scaling factor. The scaling factor can be .
- A scaling factor of 1X is implied and need not be included in the assembly language instruction.
- (MOV AL,[EBX+ECX]).
- A scaling factor of 2X is used to address word-sized memory arrays,
- a scaling factor of 4X is used with doubleword-sized memory arrays, and a scaling factor of 8X is used with quadword-size memory array.

Scaled-Index Addressing

- .
- **TABLE 3–9** Examples of scaled-index addressing.

Assembly Language	Size	Operation
MOV EAX,[EBX+4*ECX]	32 bits	Copies the doubleword contents of the data segment memory location addressed by the sum of 4 times ECX plus EBX into EAX
MOV [EAX+2*EDI+100H],CX	16 bits	Copies CX into the data segment memory location addressed by the sum of EAX, 100H, and 2 times EDI
MOV AL,[EBP+2*EDI+2]	8 bits	Copies the byte contents of the stack segment memory location addressed by the sum of EBP, 2, and 2 times EDI into AL
MOV EAX,ARRAY[4*ECX]	32 bits	Copies the doubleword contents of the data segment memory location addressed by the sum of ARRAY and 4 times ECX into EAX

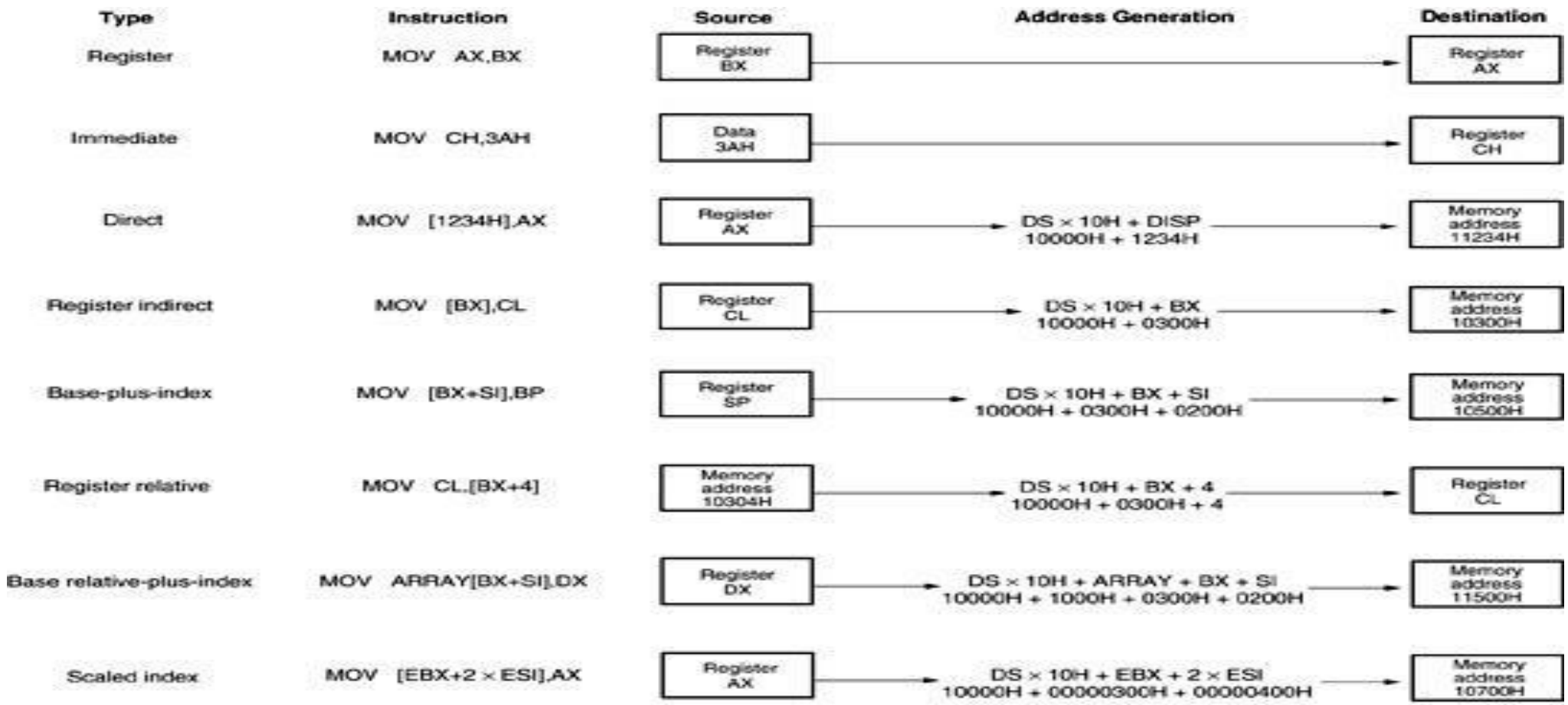


Figure (12) 80886- core2 addressing modes

- The 64-bit registers in a Pentium 4 with 64-bit extensions are RAX, RBX, RCX, RDX, RSP, RBP, RDI, RSI, and R8 through R15.
- In addition, the microprocessor contains an instruction pointer (IP/EIP/RIP) and flag register (FLAGS, EFLAGS, or RFLAGS).
- All real mode memory addresses are a combination of a segment address plus an offset address.
- The starting location of a segment is defined by the 16-bit number in the segment register that is appended with a hexadecimal zero at its rightmost end.
- The offset address is a 16-bit number added to the 20-bit segment address to form the real mode memory address.
- All instructions (code) are accessed by the combination of CS (segment ad-dress) plus IP or EIP (offset address).

- Data are normally referenced through a combination of the DS (data segment) and either an offset address or the contents of a register that contains the offset address.
- The 8086-Core2 use BX, DI, and SI as default offset registers for data if 16-bit registers are selected.
- The 80386 and above can use the 32-bit registers EAX, EBX, ECX, EDX, EDI, and ESI as default offset registers for data.
- The MOV immediate instruction transfers the byte or word that immediately follows the opcode into a register or a memory location. Immediate addressing manipulates constant data in a program. In the 80386 and above, doubleword immediate data may also be loaded into a 32-bit register or memory location.
- Direct addressing occurs in two forms in the microprocessor: (1) direct addressing and (2)displacement addressing. Both forms of addressing are identical except that direct addressing is used to transfer data between EAX, AX, or AL and memory; displacement addressing is used with any register-memory transfer. Direct addressing requires 3 bytes of memory, whereas displacement addressing requires 4 bytes. Note that some of these instructions in the 80386 and above may require additional bytes in the form of prefixes for register and operand sizes.

- Register indirect addressing allows data to be addressed at the memory location pointed to by either a base (BP and BX) or index register (DI and SI). In the 80386 and above, extended registers EAX, EBX, ECX, EDX, EBP, EDI, and ESI are used to address memory data.
- Base-plus-index addressing often addresses data in an array. The memory address for this mode is formed by adding a base register, index register, and the contents of a segment register times 10H. In the 80386 and above, the base and index registers may be any 32-bit register except EIP and ESP.
- Register relative addressing uses a base or index register, plus a displacement to access memory data.
- Base relative-plus-index addressing is useful for addressing a two-dimensional memory array. The address is formed by adding a base register, an index register, displacement, and the contents of a segment register times 10H.
- Scaled-index addressing is unique to the 80386 through the Core2. The second of two registers (index) is scaled by a factor of 2X,4X, or 8X to access words, doublewords, or quadwords in memory arrays. The MOV AX,[EBX+2*ECX] and the MOV [4*ECX],EDX are examples of scaled-index instructions.