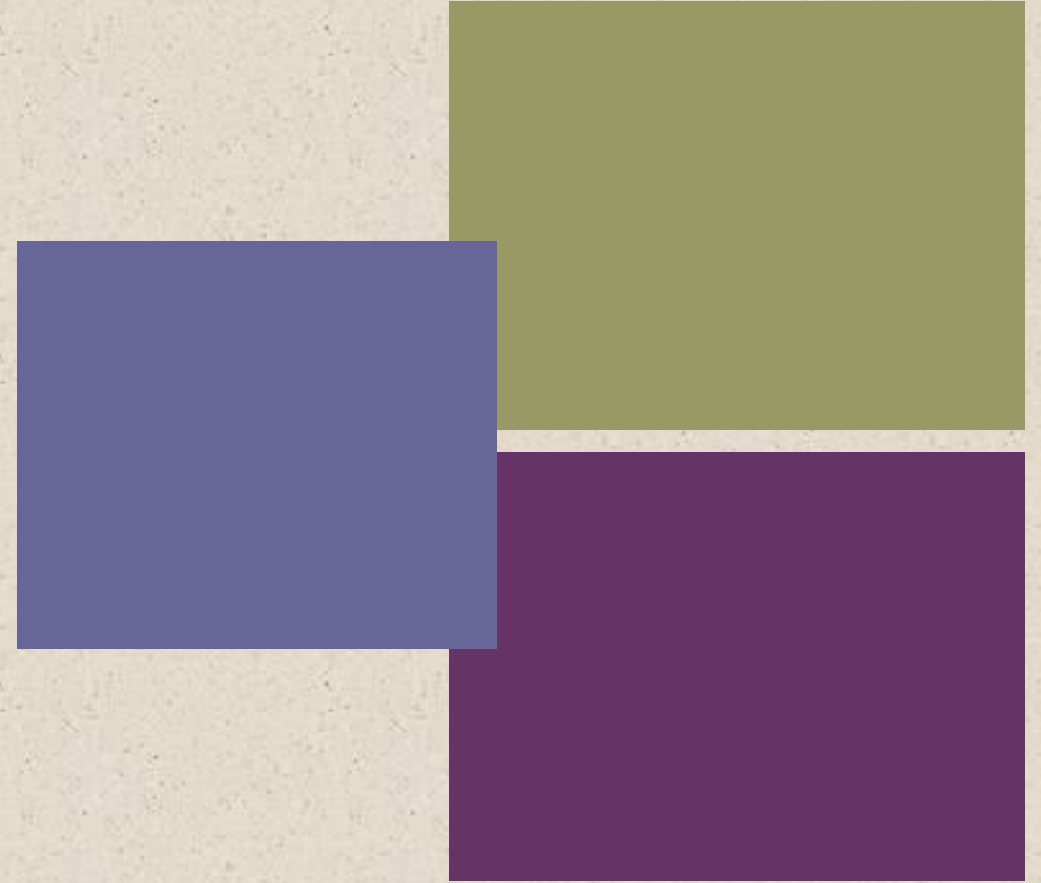


Instruction Sets: Addressing Modes



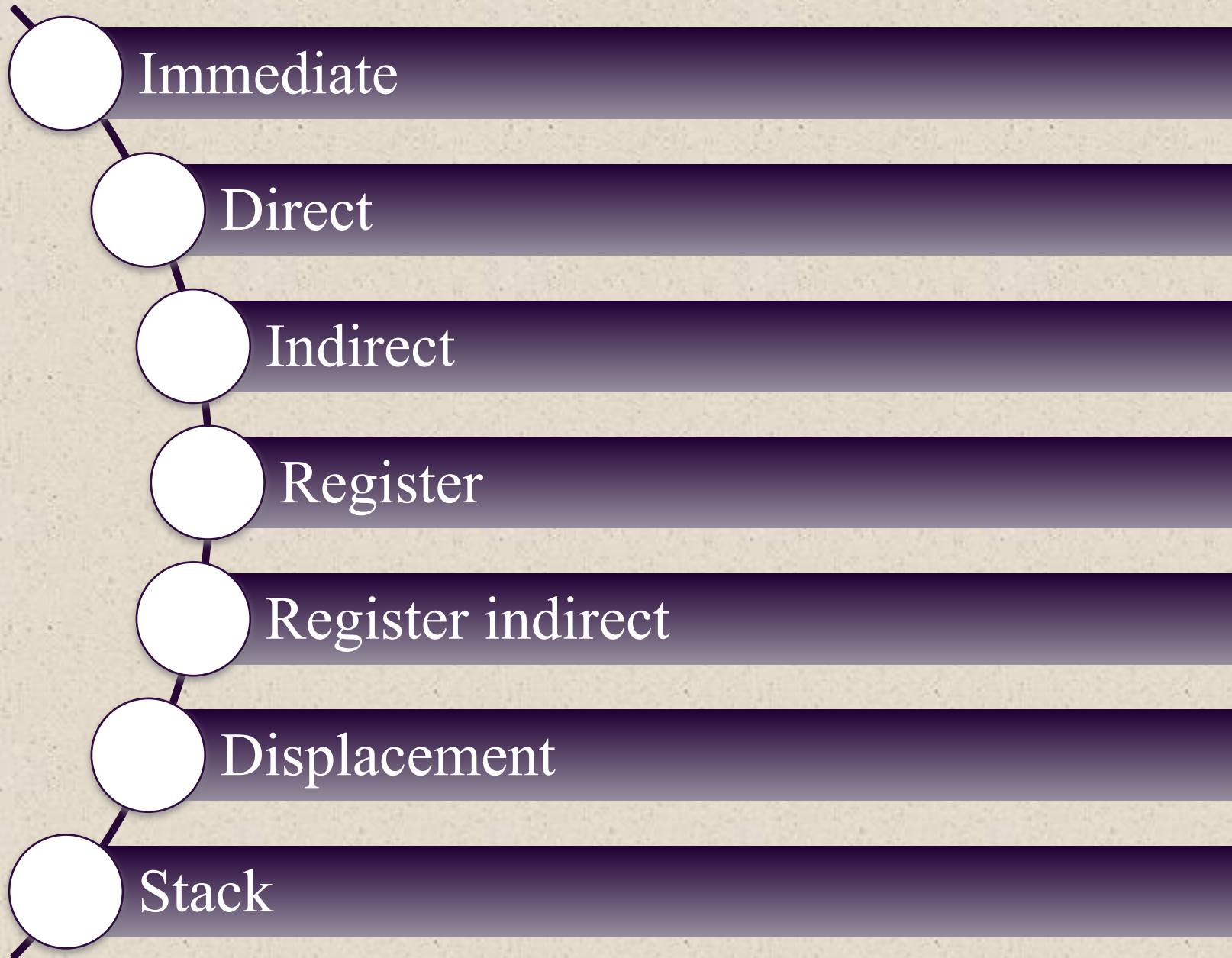
+

SANTOSH KUMAR VERMA

Mentor

GeeksforGeeks

Addressing Modes



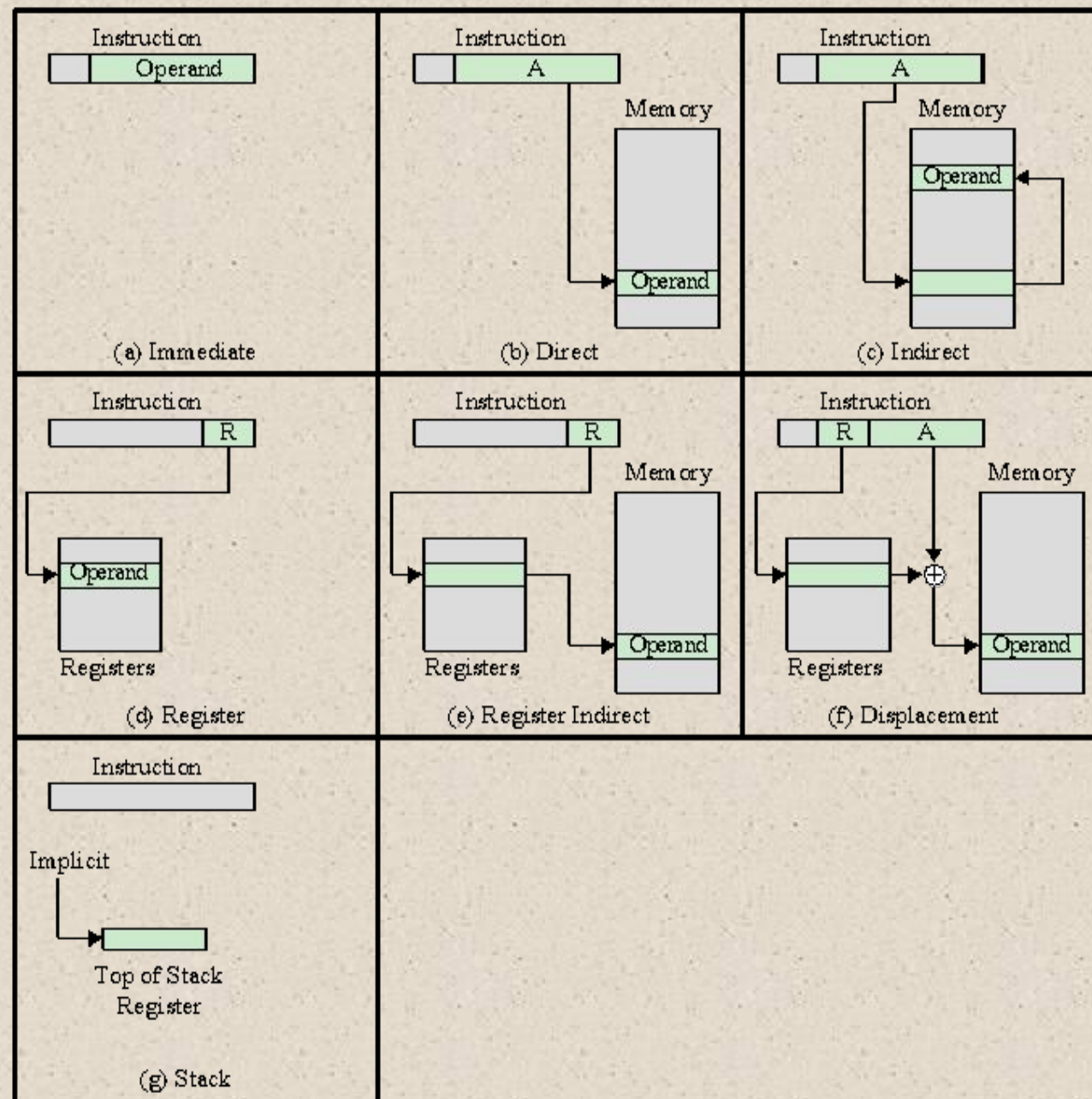


Figure 13.1 Addressing Modes


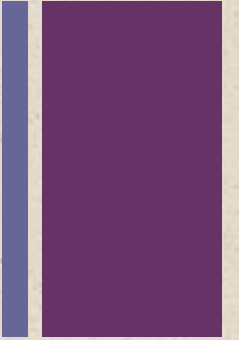


Table 13.1
Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability



Immediate Addressing



- Simplest form of addressing
- Operand = A
- This mode can be used to define and use constants or set initial values of variables
 - Typically the number will be stored in twos complement form
 - The leftmost bit of the operand field is used as a sign bit
- Advantage:
 - No memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle
- Disadvantage:
 - The size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the word length

Direct Addressing

Address field contains the effective address of the operand

Effective address (EA)
= address field (A)

Was common in earlier generations of computers

Requires only one memory reference and no special calculation

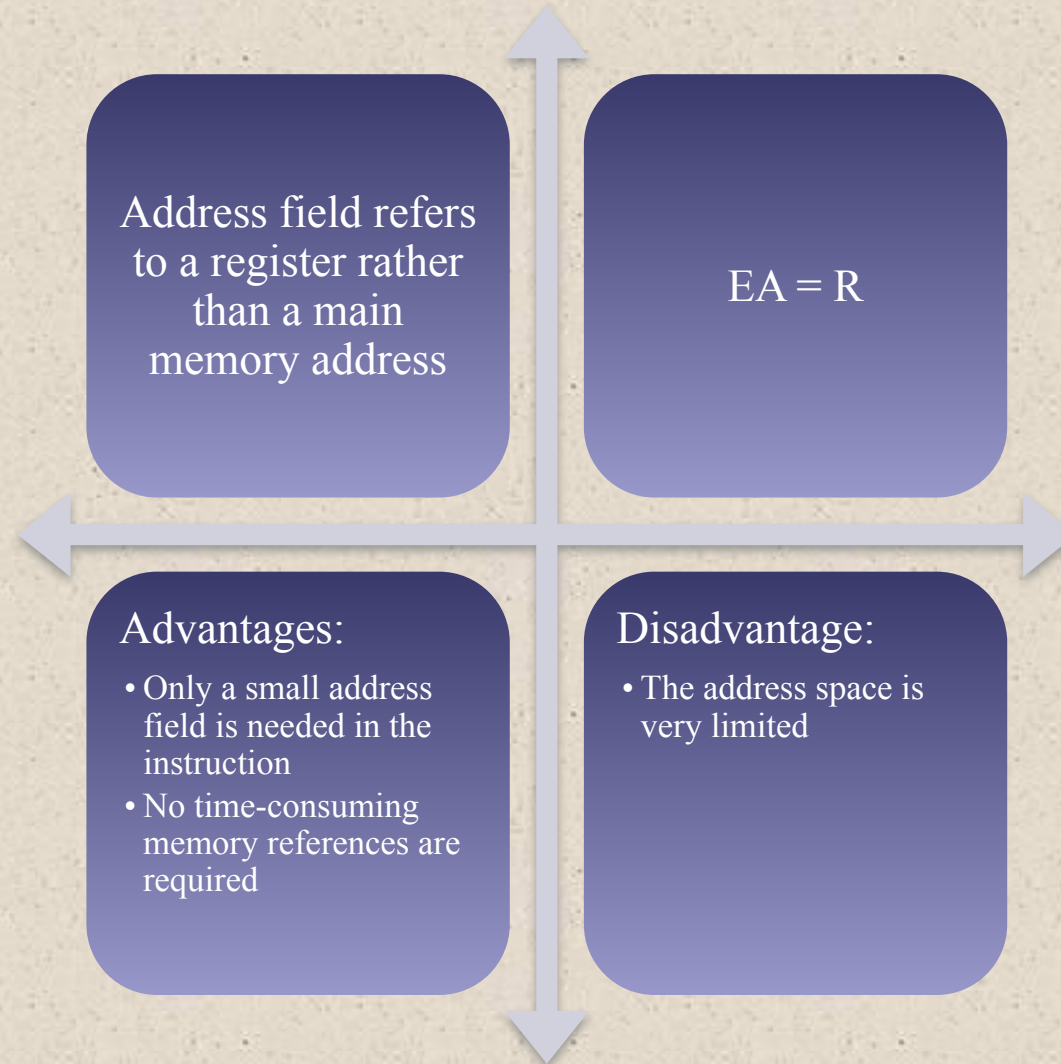
Limitation is that it provides only a limited address space



+ Indirect Addressing

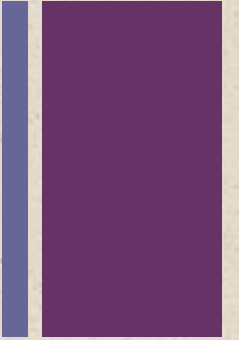
- Reference to the address of a word in memory which contains a full-length address of the operand
- $EA = (A)$
 - Parentheses are to be interpreted as meaning *contents of*
- Advantage:
 - For a word length of N an address space of 2^N is now available
- Disadvantage:
 - Instruction execution requires two memory references to fetch the operand
 - One to get its address and a second to get its value
- A rarely used variant of indirect addressing is multilevel or cascaded indirect addressing
 - $EA = (\dots (A) \dots)$
 - Disadvantage is that three or more memory references could be required to fetch an operand

Register Addressing





Register Indirect Addressing



- Analogous to indirect addressing
 - The only difference is whether the address field refers to a memory location or a register
- $EA = (R)$
- Address space limitation of the address field is overcome by having that field refer to a word-length location containing an address
- Uses one less memory reference than indirect addressing



Displacement Addressing



- Combines the capabilities of direct addressing and register indirect addressing
- $EA = A + (R)$
- Requires that the instruction have two address fields, at least one of which is explicit
 - The value contained in one address field (value = A) is used directly
 - The other address field refers to a register whose contents are added to A to produce the effective address
- Most common uses:
 - Relative addressing
 - Base-register addressing
 - Indexing

Relative Addressing

The implicitly referenced register is the program counter (PC)

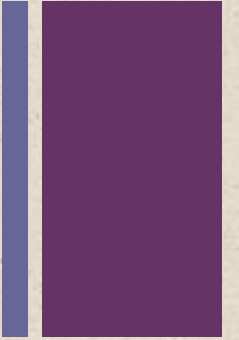
- The next instruction address is added to the address field to produce the EA
- Typically the address field is treated as a two's complement number for this operation
- Thus the effective address is a displacement relative to the address of the instruction

Exploits the concept of locality

Saves address bits in the instruction if most memory references are relatively near to the instruction being executed



Base-Register Addressing



- The referenced register contains a main memory address and the address field contains a displacement from that address
- The register reference may be explicit or implicit
- Exploits the locality of memory references
- Convenient means of implementing segmentation
- In some implementations a single segment base register is employed and is used implicitly
- In others the programmer may choose a register to hold the base address of a segment and the instruction must reference it explicitly

Indexing

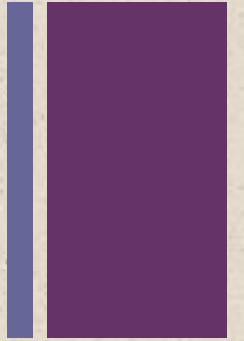
- The address field references a main memory address and the referenced register contains a positive displacement from that address
- The method of calculating the EA is the same as for base-register addressing
- An important use is to provide an efficient mechanism for performing iterative operations
- Autoindexing
 - Automatically increment or decrement the index register after each reference to it
 - $EA = A + (R)$
 - $(R) \leftarrow (R) + 1$
- Postindexing
 - Indexing is performed after the indirection
 - $EA = (A) + (R)$
- Preindexing
 - Indexing is performed before the indirection
 - $EA = (A + (R))$





Stack Addressing

- A stack is a linear array of locations
 - Sometimes referred to as a *pushdown list* or *last-in-first-out queue*
- A stack is a reserved block of locations
 - Items are appended to the top of the stack so that the block is partially filled
- Associated with the stack is a pointer whose value is the address of the top of the stack
 - The stack pointer is maintained in a register
 - Thus references to stack locations in memory are in fact register indirect addresses
- Is a form of implied addressing
- The machine instructions need not include a memory reference but implicitly operate on the top of the stack



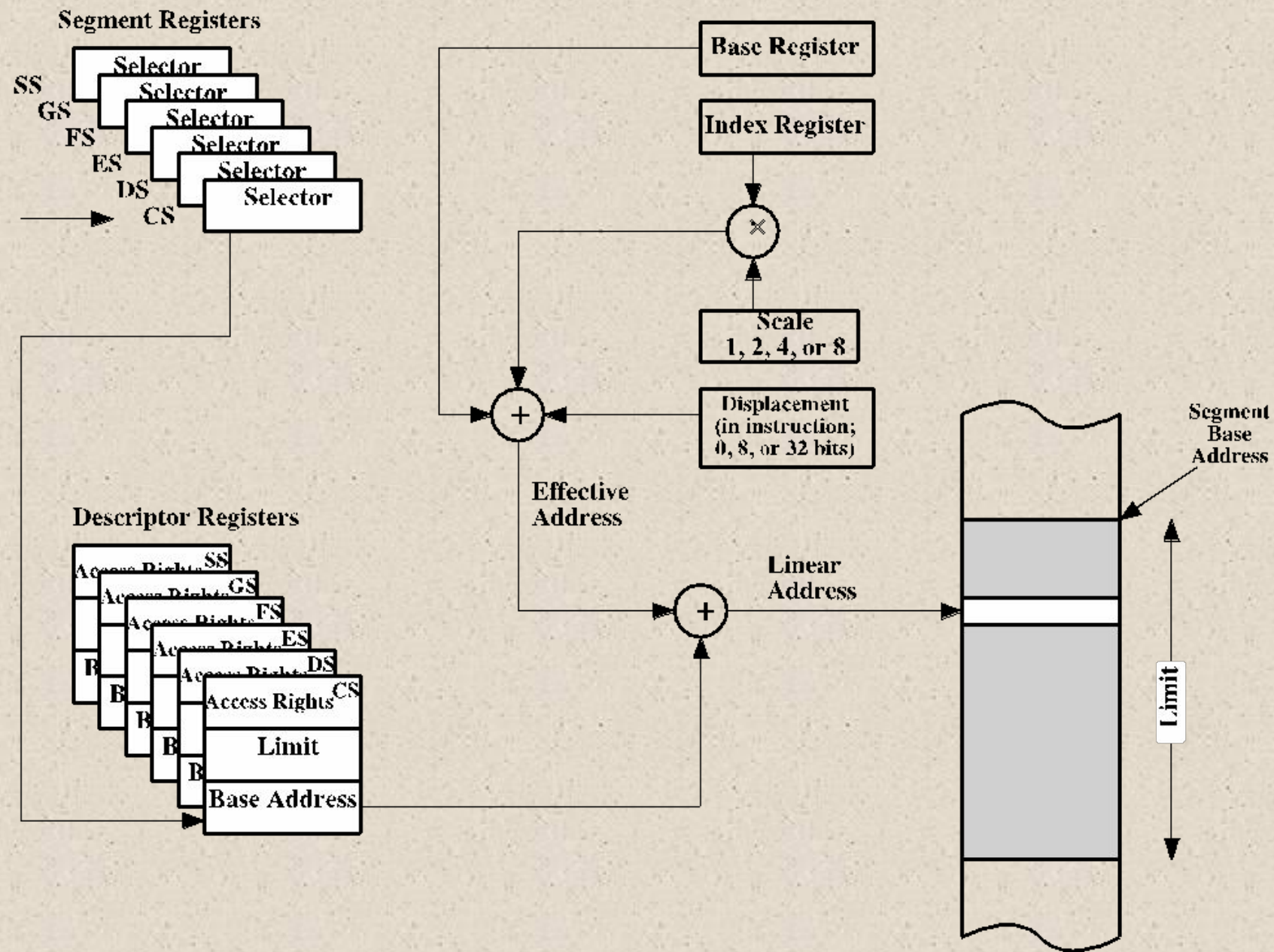


Figure 13.2 x86 Addressing Mode Calculation

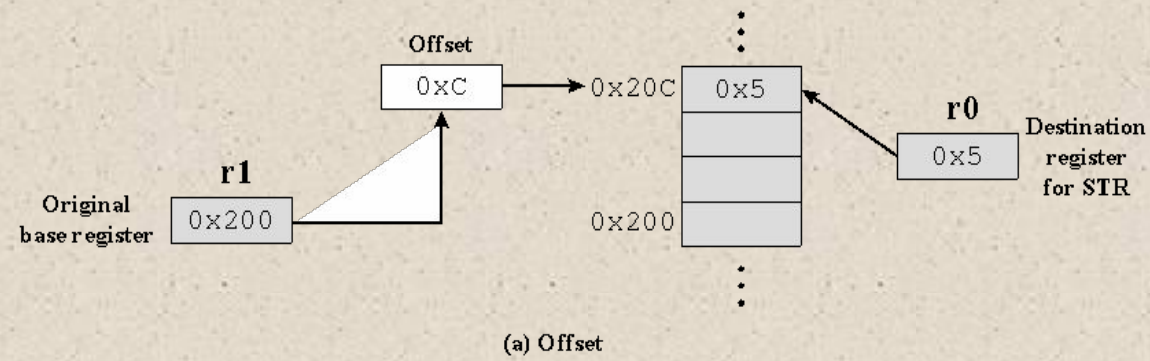
Table 13.2

x86 Addressing Modes

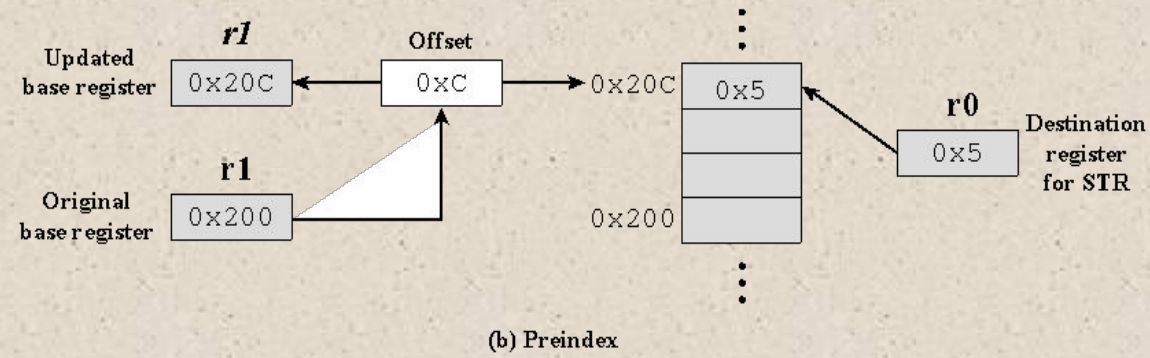
Mode	Algorithm
Immediate	$\text{Operand} = A$
Register Operand	$\text{LA} = R$
Displacement	$\text{LA} = (\text{SR}) + A$
Base	$\text{LA} = (\text{SR}) + (B)$
Base with Displacement	$\text{LA} = (\text{SR}) + (B) + A$
Scaled Index with Displacement	$\text{LA} = (\text{SR}) + (I) \times S + A$
Base with Index and Displacement	$\text{LA} = (\text{SR}) + (B) + (I) + A$
Base with Scaled Index and Displacement	$\text{LA} = (\text{SR}) + (I) \times S + (B) + A$
Relative	$\text{LA} = (\text{PC}) + A$

LA = linear address
 (X) = contents of X
 SR = segment register
 PC = program counter
 A = contents of an address field in the instruction
 R = register
 B = base register
 I = index register
 S = scaling factor

STRB r0, [r1, #12]



STRB r0, [r1, #12]!



STRB r0, [r1], #12

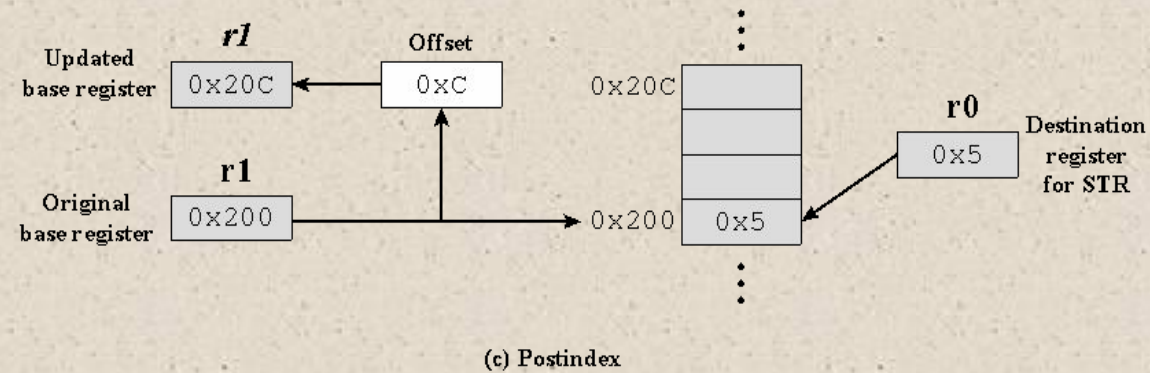


Figure 13.3 ARM Indexing Methods

8086 programmer's model

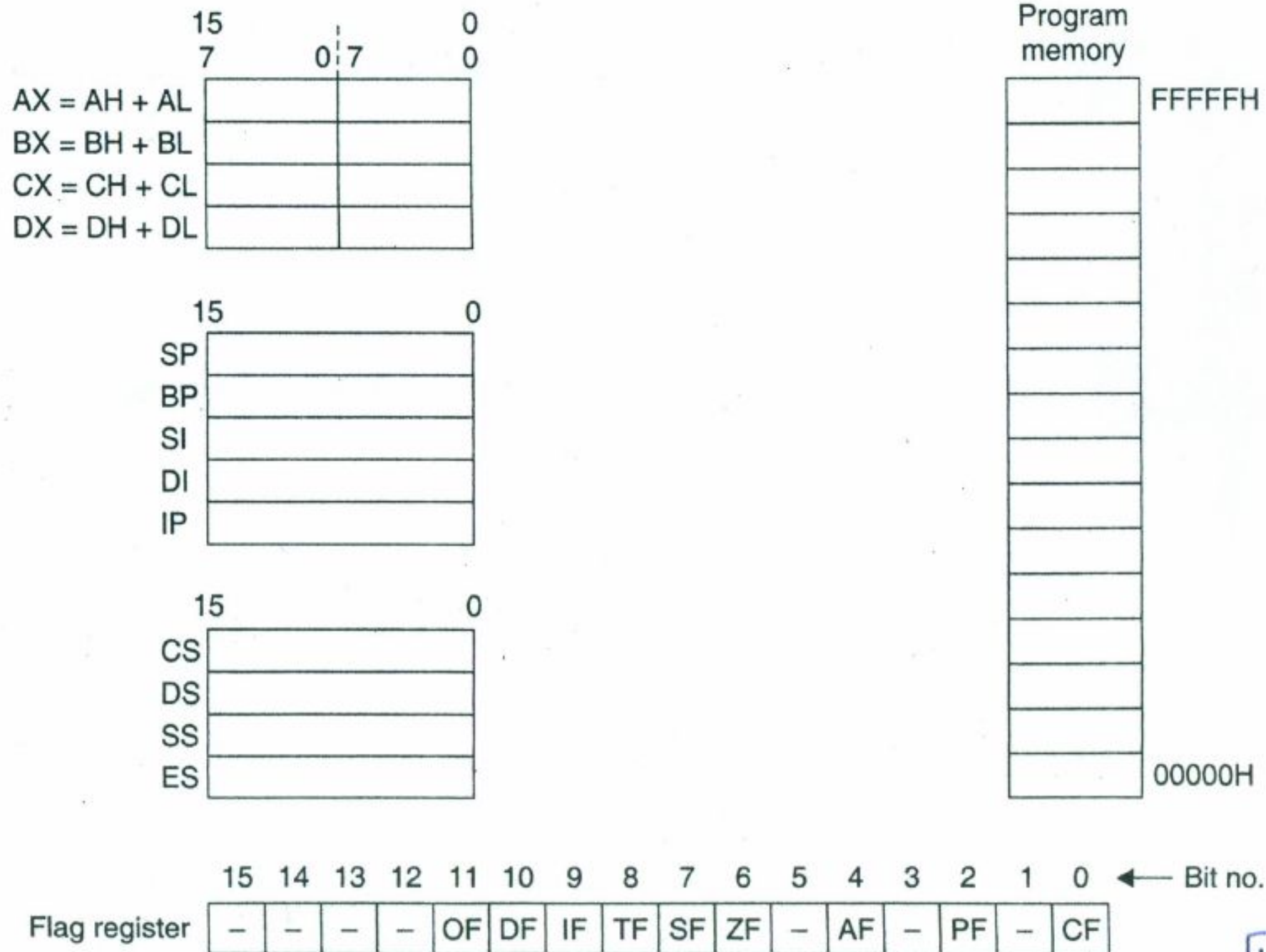


Figure 6.1 The 8086 programmer's model.

- 20 address lines
- 1 MB memory capacity
- Memory space may be divided into 4 segments each of 64 kb.
- Segment register contains information on the starting address of segment in memory.

CS – Code Segment Register

DS – Data Segment Register

ES – Extra Segment Register

SS – Stack Segment Register

Memory space of segment may overlap.

There may be more than one segment of a particular type.

AX – Primary Acc. Unique in following way

- I/O operations pass through AX (or AL).
- Instructions using AX (or AL) take less time to execute.
- Many string instructions require one of the operands to be in AX(or AL)
- AX and AL used in multiply and divide instructions.

BX – GPR – also may be used as base register for memory address calculation.

CX – GPR – also may be used to hold count in multiiteration instructions. CX contains the loop count.

DX – GPR – also may be used in I/O. Instruction [DX stores the port address], multiply and divide instructions.

SI – Source Index – helps in indexed addressing

DI – Destination Index - helps in indexed addressing

BP – Base Pointer – used for indirect addressing and base relative addressing for stack operations.

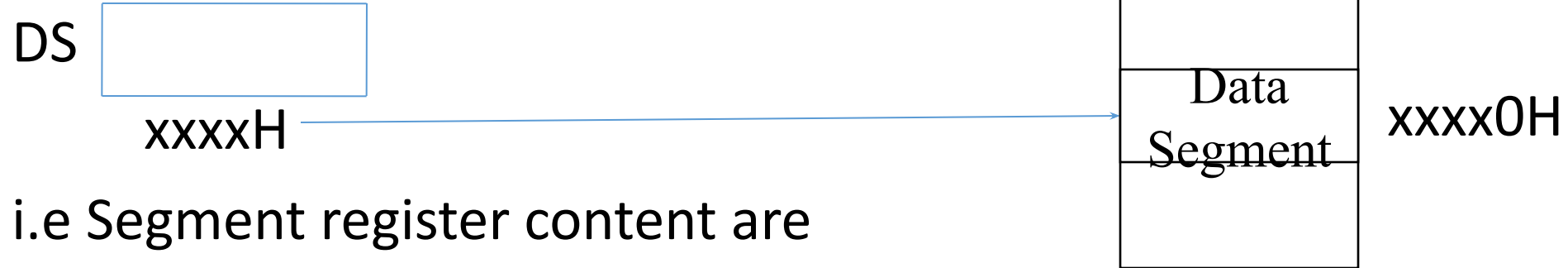
IP – Instruction Pointer – contains offset address of next instruction to be executed. Serves as program counter.

Operand addressing-

Segment Register – 16 bit

Memory Address – 20 bit

Segment register content are appended with 0000B to get starting address of segment in memory.



i.e Segment register content are

multiplied by 16(Decimal) or 10H to get starting address of segment in memory.

The starting add. of segment is added with offset to get effective memory address also called physical memory address. Offset is called logical address.

Instruction Addressing

Starting address of code segment = $(CS) \times 10H$

Offset address = (IP) i.e content of IP register

Effective address = $(CS) \times 10H + (IP)$

Operand Addressing

Addressing modes clearly specify the location of operand and how its location may be determined.

Following addressing modes are available-

- Register Addressing Mode
- Immediate Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Base Plus Index Register Addressing Mode
- Register Relative Addressing Mode
- Base plus Index Register Relative Addressing Mode
- String Addressing Mode.

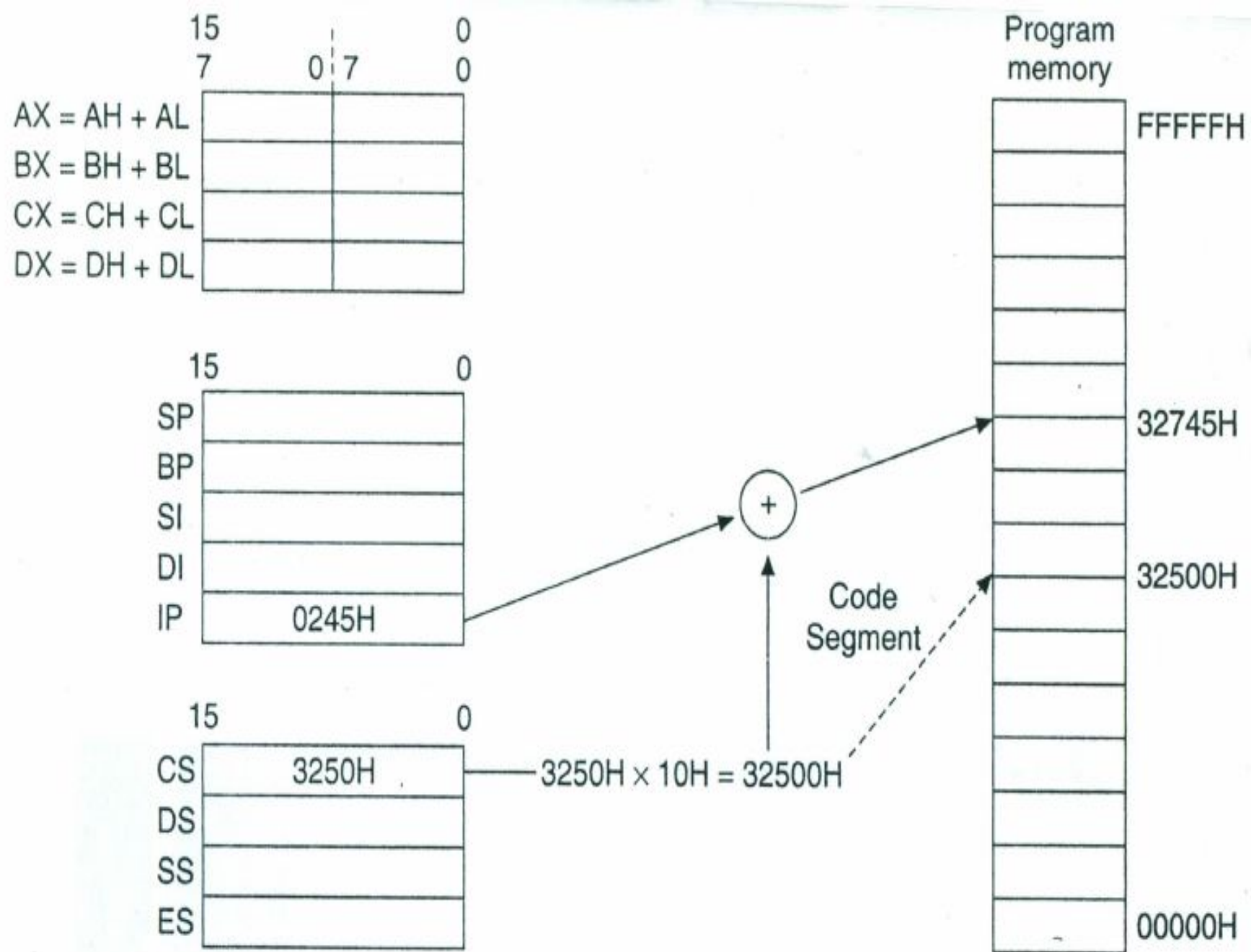


Figure 6.2 Instruction addressing.

Register Addressing-

Both destination and source operand in registers.

MOV AL, AH (AL) \square (AH)

MOV AX, DX (AX) \square (DX)

AND AL BL (AL) \square (AL) AND (BL)

Immediate Addressing

One of the operands is part of instruction as immediate data –
8/16 bit

Mov CX, 2346H (CX) \square 2346H

SUB AX , 7860H (AX) \square (AX)- 7860H

Direct Memory Addressing

16 bit offset address is part of instruction as displacement field. It is stored as 16 bit unsigned or 8 bit sign extended number, following the instruction code.

Example

MOV [3725H], DX

$((DS) * 10H + 3725H) \square (DL)$

$((DS) * 10H + 3726H) \square (DH)$

Move the content of DL and DH registers to memory location calculated from DS register and displacement.

\square OR AL, [3030H]

OR the content of AL register with the content of memory location calculated with DS register and displacement and store the result in AL register.

$(AL) \square ((DS) * 10H + 3030H) \text{ OR } (AL)$

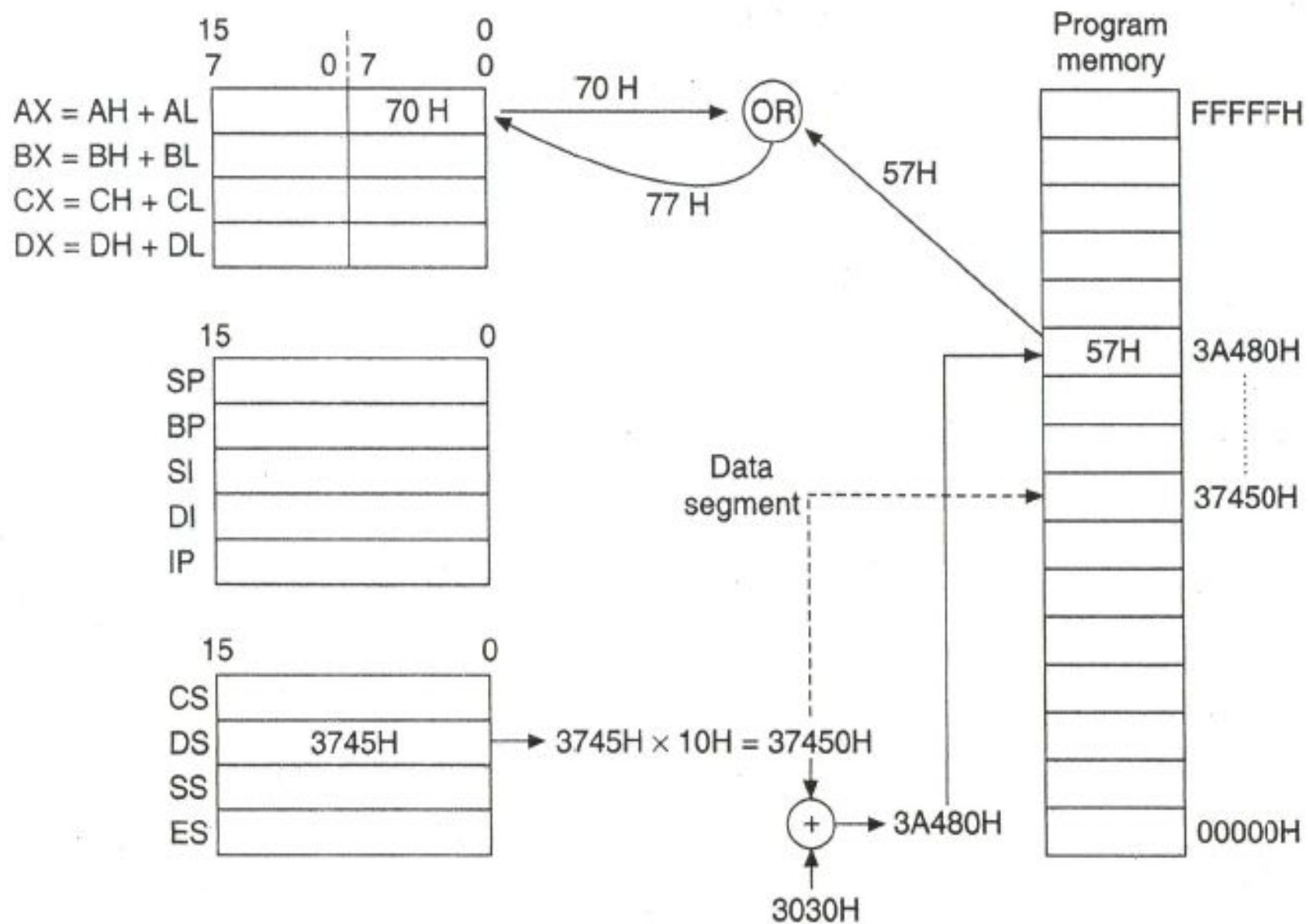


Figure 6.3 Execution of direct memory addressed instruction OR AL, (3030H).

In the figure

$$(AL) = 70H$$

$$(DS) = (3745H)$$

$$\text{Effective memory address} = 3745H * 10H + 3030H$$

$$= 37450H + 3030H$$

$$= 3A480H$$

Direct Memory Addressing Instructions - Example

The content of memory location identified are moved directly.

```
MOV  AX,  M1
M1   DW   3425H
```

(AX) \square 3425H

(AL) \square 25H, (AH) \square 34H

```
MOV  N1,  CL
      (N1)  $\square$  (CL)
N1   DB   08H
```

```
MOV  ES: [3520],  AL
[ES: 3520]  $\square$  [AL]
```

Offset

Register Indirect Addressing

Offset register is specified through pointer register(i.e BP or BX) or index register(SI or DI)

Mov AL, [BX]; offset is stored in BX

$(AL) \leftarrow ((DS) * 10H + (BX))$

XOR [DI], CL

$((DS) * 10H + (DI)) \leftarrow ((DS) * 10H + (DI)) \text{ XOR } (CL)$

Similarly MOV AL, [BX+4]

XOR [DI+6],CL

Note:- memory to memory transfer is not allowed except in case of string.

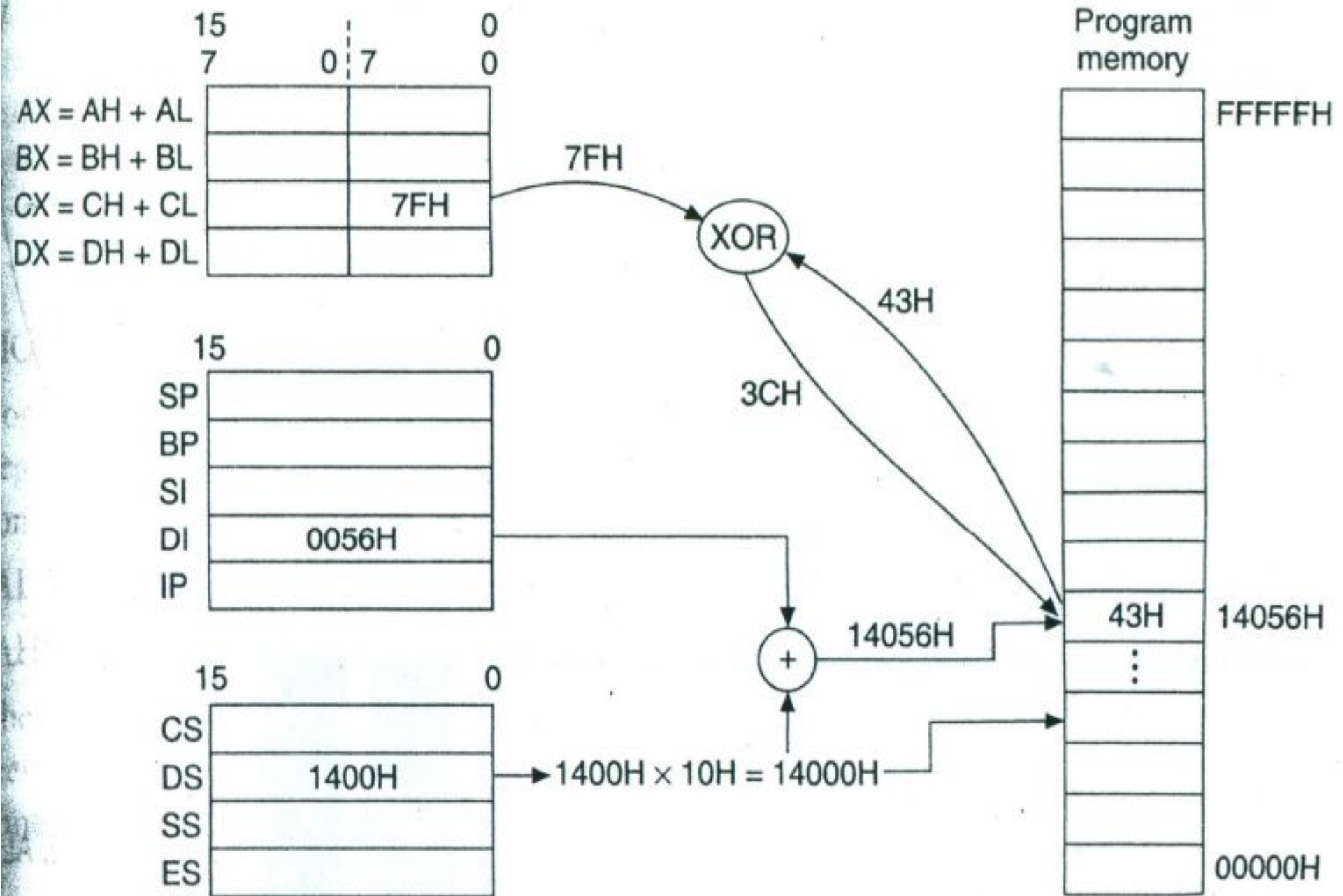


Figure 6.4 Execution of register indirect addressing mode instruction `XOR (DI), CL`.

Register Indirect Addressing

MOV[DI], 30H

; Offset in DI

Whether byte or word in to be transfered is not known

Assembler can't determine the size of data 30H

Modify to

MOV BYTE PTR [DI], 30H - or

MOV WORD PTR [DI], 30H

To find offset address of N1

MOV DI, OFFSET N1

MOV BYTE PTR[DI], 30H

N1 DB -

Examples

DIRECT

.DATA

N1 DB 5

N2 DB 7

N3 DB 8

N4 DB 0

.CODE

MOV CL, N1

ADD CL, N2

ADD CL, N3

MOV N4, CL

INDIRECT

.DATA

N1 DW 1234H

N2 DB 12H

.CODE

MOV BX, OFFSET N1

MOV [BX], 0333H

MOV BX, OFFSET N2

MOV [BX], 03H

Correct way to write is

MOV WORDPTR [BX], 0333H

OR

MOV BYTE PTR [BX], 03H

Base Plus Index Register Addressing Mode

Main utility – for addressing an array of data.

Extension of register indirect addressing.

Base register (BX) and index register (SI or DI) used to indirectly address the memory location.

MOV [BX + DI], AL -or

MOV [BX] [DI], AL

Offset address = (BX)+(DI)

BX - Base register DI - Index Register

Copy the content of AL register to the memory location calculated by taking offset at = (BX)+(DI)

Starting address of data segment = (DS)x10H

Effective memory address = ((DS)x10H +(BX)+(DI))

((DS)x10H +(BX)+(DI)) □ (AL)

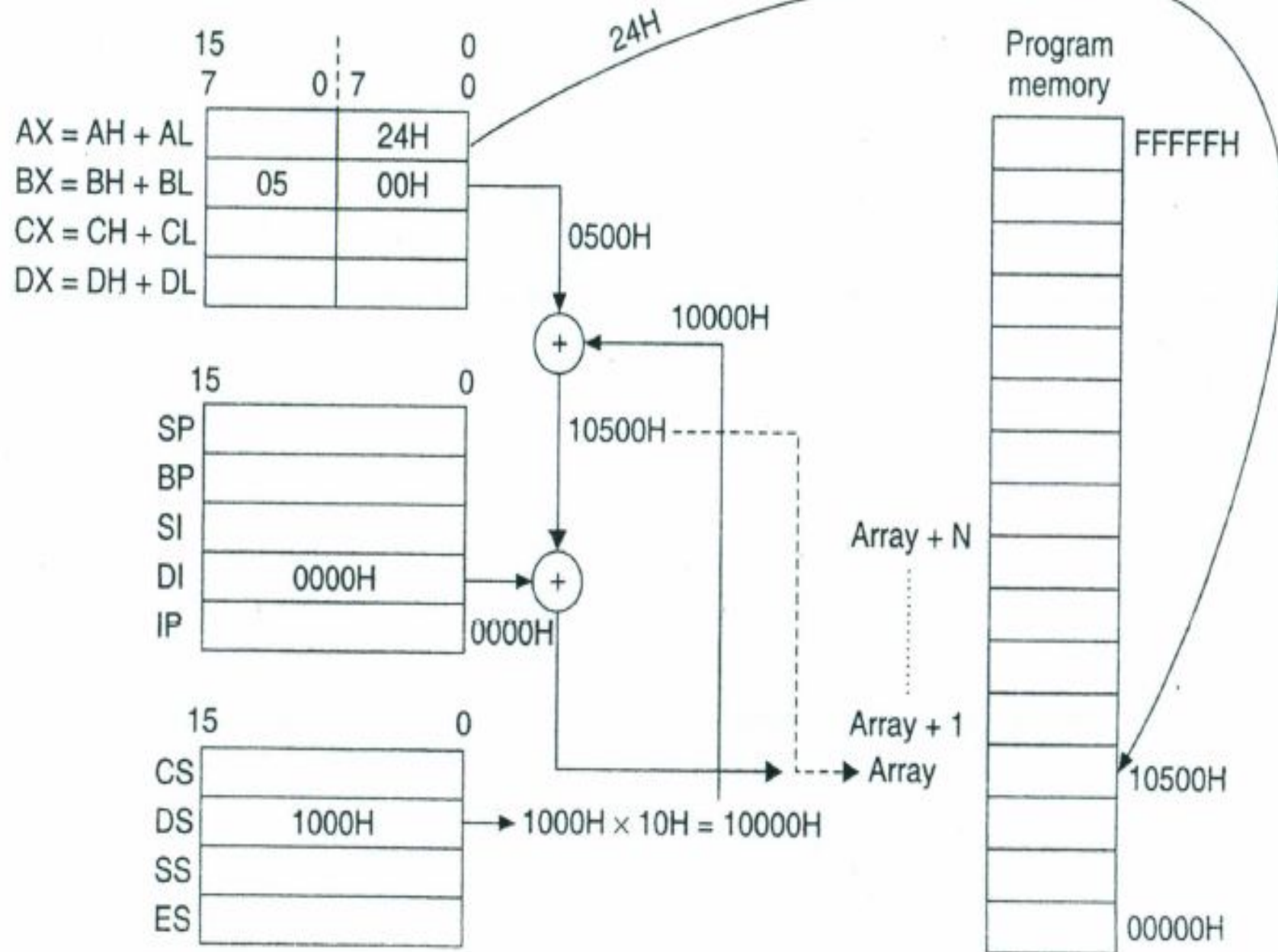
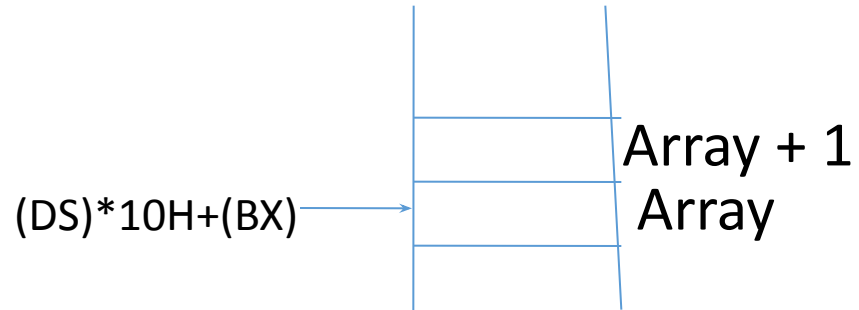


Figure 6.5 Execution of base plus index register addressing mode instruction `MOV (BX + DI), AL`.

Base register may contain address to point to base of array i.e 0th location.

The value of DI may be changed to access different locations.



Example – Array accessing

.DATA

ARRAY DB 75H, 3CH----

.CODE

MOV BX, OFFSET ARRAY

MOV DI, 00H

MOV AL, [BX+DI] (AL) = 75H

INC DI



OFFSET directive loads the offset address of a particular memory location.

NOTE – MOV BX, ARRAY will move two bytes 75 & 3CH of Array to BX

Register Relative Addressing Mode

Similar to Base Plus Index Register Addressing Mode.

Offset is calculated by using one of the register- BX, BP, DI or SI and displacement specified as 8 bit or 16 bit number specified in the instruction.

MOV AX, [DI+06] / MOV AX, [DI + 06H] / MOV AX, 06[DI]

Effective memory address = $(DS) * 10H + (DI) + 06$

(AL) $\square ((DS) * 10H + (DI) + 06)$

(AH) $\square ((DS) * 10H + (DI) + 07)$

These bytes may be part of array.

Base of array – $(DS) * 10H + (DI)$

By decreasing or increasing displacement we can access different elements of array.

AX = AH + AL
 BX = BH + BL
 CX = CH + CL
 DX = DH + DL

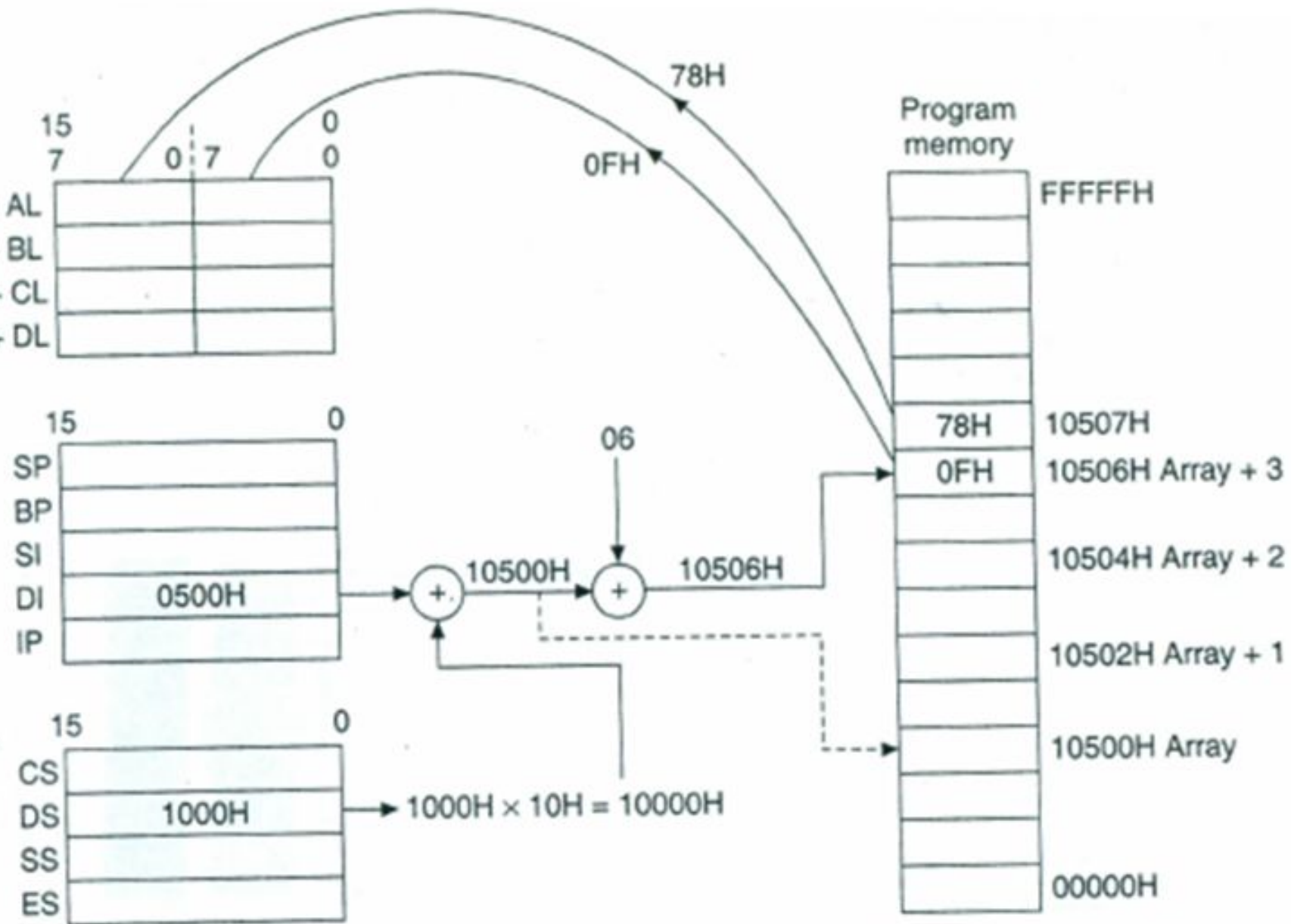


Figure 6.6 Execution of register relative addressing mode instruction `MOV AX, (DI + 06)`.

Addressing Array with Register Relative Addressing

Example

.DATA

ARRAY DB 75, 3C, 9A ----

.CODE

MOV DI, OFFSET ARRAY

MOV AL, [DI]; AL \square ARRAY [0]

MOV AL, [DI+1]; [AL] \square ARRAY[1]

MOV DI, 03

MOV AL, ARRAY[DI]

;Offset = Address of Array + [DI]

MOV ARRAY[DI], AL

Base Plus Index Register Relative Addressing Mode- (Based indexed Addressing mode)

Combination of base + index register addressing and register relative addressing.

Suited for accessing the element of two dimensional array.

For calculating offset-

Base Register – BX

Index Register – SI or DI

And displacement specified in the instruction is used.

Base Register

BX ☐ Starting memory location of array.

SI or DI ☐ Starting of row.

Displacement ☐ particular element in the row.

MOV [BX + DI + 2], CL

Or MOV [BX][DI+2], CL

MOV 2[BX][DI], CL

$((DS)*10H+(BX)+(DI)+2) \text{ } \square \text{ } (CL)$

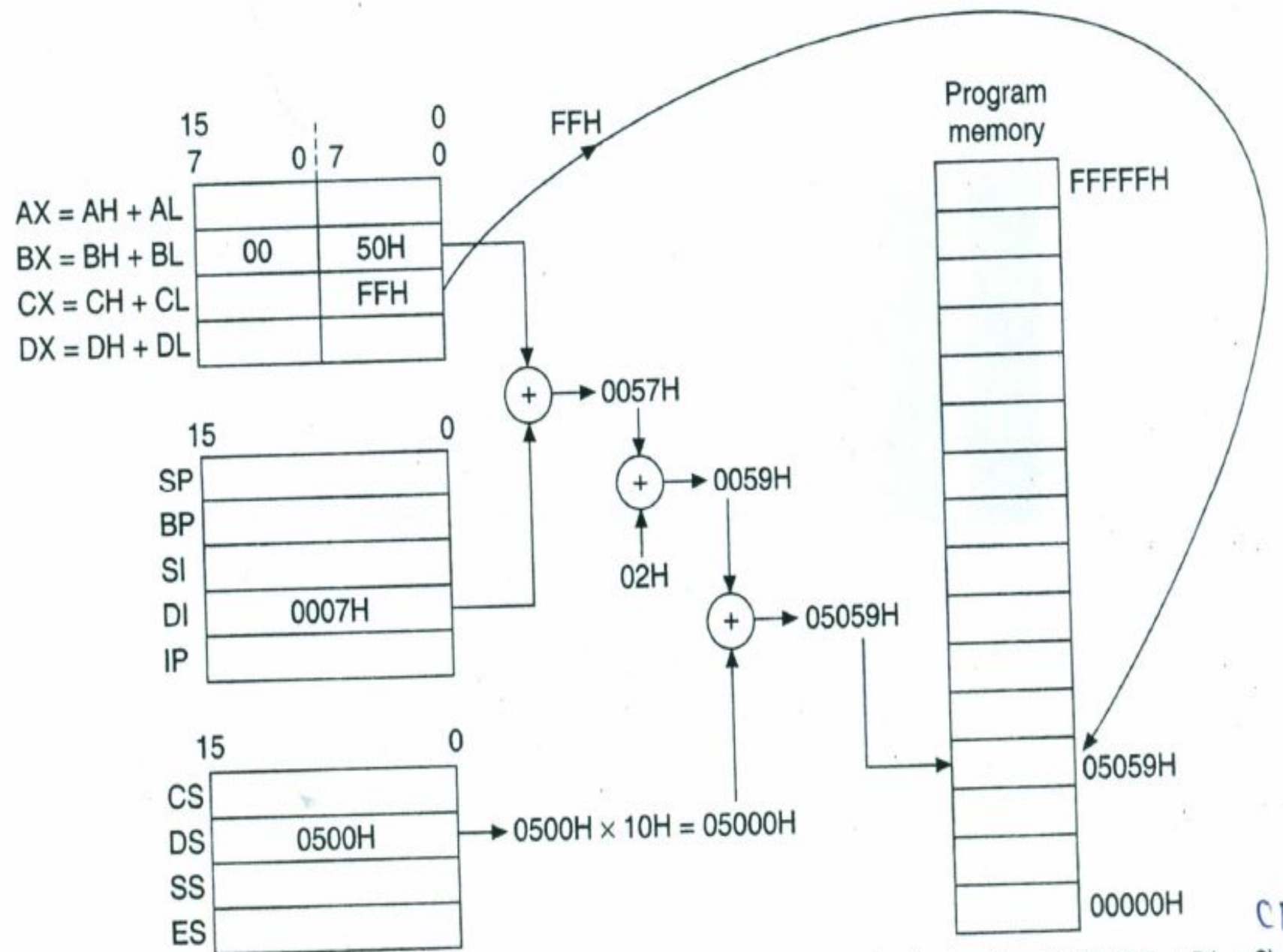


Figure 6.7 Execution of base plus index register relative addressing mode instruction `MOV (BX + DI + 2),`

Two dimensional array-

X00	X01	X02	X03
X10	X11	X12	X13
--	--	--	--
--	--	--	--

- Elements stored sequentially in memory row wise

<u>X00</u>	<u>X01</u>	<u>X02</u>	<u>X03</u>	<u>X10</u>	<u>X11</u>	<u>X12</u>	<u>X13</u>
ROW 0				ROW 1			

Let us say BX contains offset which with point to X00 i.e base of array.

To access X03, 3rd element in row 0

(DI) = 0, displacement = 03

Example

.DATA

ARRAY DB X1, X2, X3, X4, Y1, Y2, Y3,Y4

.CODE

MOV BX, OFFSET ARRAY

MOV DI, 00 ; - 0th ROW, 0th ELEMENT i.e X1

MOV AL, (BX + DI)

MOV AL , (BX+DI+3) ; (AL) □ X4

ADD DI, 04; (DI) □ 04 Row = 01, Column =00

MOV AL, (BX + DI + 3) ; (AL) □ Y4

;To access 0th element of row 1. i.e. Y1

; (DI) = Row No * No of elements in row = 1 * 4 = 4

;Displacement = 00

;To access 2nd element (Y3) of row 1

;(DI) = 4, Displacement = 2

Note – (BX) – starting address of array

(BX) + (DI) – Starting address of row.

Another representation is

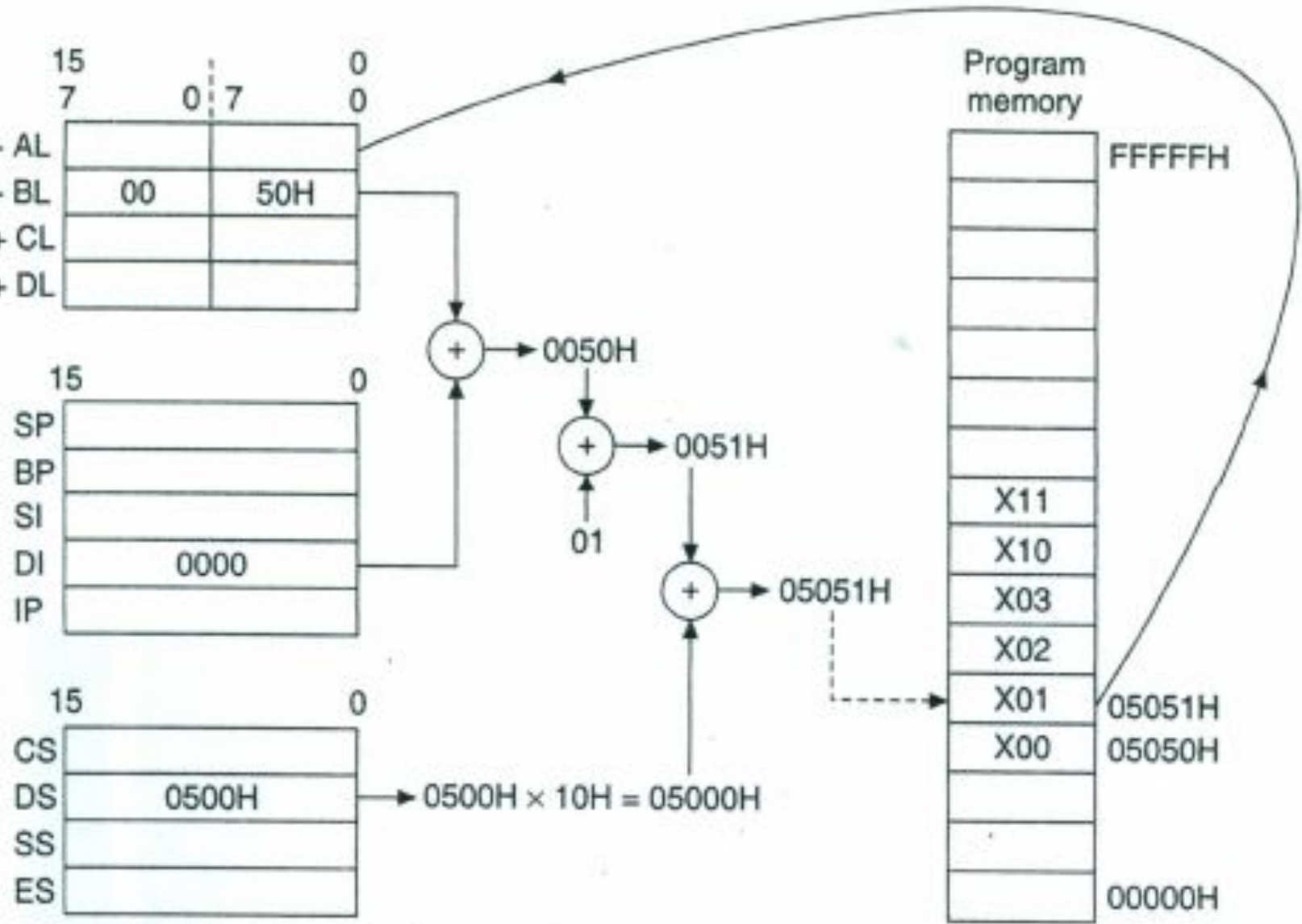
.CODE

MOV BX, 00 ; ROW NO

MOV DI, 01 ; COLUMN NO

MOV AL, ARRAY [BX+DI]

$AX = AH + AL$
 $BX = BH + BL$
 $CX = CH + CL$
 $DX = DH + DL$



Accessing a two-dimensional array using base plus index register relative addressing mode instruction `MOV AL, (BX + DI + 1)`.



Which of the following addressing modes are suitable for program relocation at run time?

- A. Absolute addressing
- B. Based addressing
- C. Relative addressing
- D. Indirect addressing





Which of the following is not a form of memory?

- A. Instruction Cache
- B. Instruction Register
- C. Instruction Opcode
- D. Translation look aside buffer





In absolute addressing mode

- A. the operand is inside the instruction
- B. the address of the operand is inside the instruction
- C. the register containing the address of the operand is specified inside the instruction
- D. the location of the operand is implicit





The most appropriate matching for the following pairs

X: Indirect addressing

Y: Immediate addressing

Z: Auto decrement addressing is

1: Loops

2: Pointers

3. Constants

A. X-3, Y-2, Z-1

B. X-1, Y-3, Z-2

C. X-2, Y-3, Z-1

D. X-3, Y-1, Z-2





Consider the following program segment. Here R1, R2 and R3 are the general purpose registers

	Instruction	Operation	Instruction size (no.of words)
	MOV R1, (3000)	$R1 \leftarrow m[3000]$	2
LOOP:	MOV R2, (R3)	$R2 \leftarrow M[R3]$	1
	ADD R2, R1	$R2 \leftarrow R1 + R2$	1
	MOV (R3), R2	$M[R3] \leftarrow R2$	1
	INC R3	$R3 \leftarrow R3 + 1$	1
	DEC R1	$R1 \leftarrow R1 - 1$	1
	BNZ LOOP	Branch on not zero	2
	HALT	Stop	1

Assume that the content of memory location 3000 is 10 and the content of the register R3 is 2000. The content of each of the memory locations from 2000 to 2010 is 100. The program is loaded from the memory location 1000. All the numbers are in decimal. Assume that the memory is word addressable. The number of memory references for accessing the data in executing the program completely is:

- A. 10
- B. 11
- C. 20
- D. 21



Consider the C struct defines below:

```
struct data {  
    int marks [100] ;  
    char grade;  
    int cnumber;  
};  
struct data student;
```

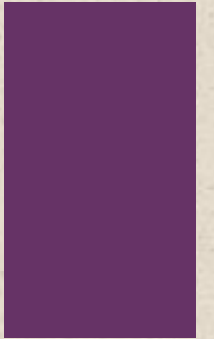
The base address of **student** is available in register R1. The field **student.grade** can be accessed efficiently using

- A. Post-increment addressing mode. (R1)+
- B. Pre-decrement addressing mode, -(R1)
- C. Register direct addressing mode, R1
- D. Index addressing mode, X(R1), where X is an offset represented in 2's complement 16-bit representation.



Consider a hypothetical processor with a instruction of type LW R1, 20(R2), which during execution reads a 32-bit word from memory and stores it in a 32-bit register R1. The effective address of the memory location is obtained by the addition of a constant 20 and the contents of register R2. Which of the following best reflects the addressing mode implemented by this instruction for the operand in memory?

- A. Immediate Addressing
- B. Base Indexed Addressing
- C. Register Indirect Scaled Addressing
- D. Register Addressing





Consider the following program segment for a hypothetical CPU having three user registers R1, R2 and R3.

Instruction; Operation ; Instruction Size (in words)

MOV R1, 5000 ; $R1 \leftarrow \text{Memory}[5000]$ 2

MOV R2, (R1) ; $R2 \leftarrow \text{Memory}[(R1)]$ 1

ADD R2, R3 ; $R2 \leftarrow R2 + R3$ 1

MOV 6000, R2 ; $\text{Memory}[6000] \leftarrow R2$ 2

HALT ; Machine halts 1

Consider that the memory is byte addressable with size 32 bits, and the program has been loaded starting from memory location 1000 (decimal). If an interrupt occurs while the CPU has been halted after executing the HALT instruction, the return address (in decimal) saved in the stack will be

- A. 1024
- B. 1020
- C. 1028
- D. 1007



The addressing mode / s, which uses the PC instead of a general-purpose register is:

- A. Indexed with offset
- B. Relative
- C. Direct
- D. Both Indexed with offset and direct





Summary

Chapter 13

- Addressing modes
 - Immediate addressing
 - Direct addressing
 - Indirect addressing
 - Register addressing
 - Register indirect addressing
 - Displacement addressing
 - Stack addressing
- Assembly language

Instruction Sets: Addressing Modes and Formats

- x86 addressing modes
- ARM addressing modes
- Instruction formats
 - Instruction length
 - Allocation of bits
 - Variable-length instructions
- X86 instruction formats
- ARM instruction formats