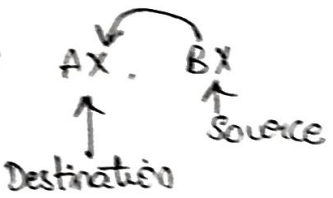
Eg: MOV AL, 22H

Data Addressing Modes

MOV instruction showing source, destination and direction of data flow



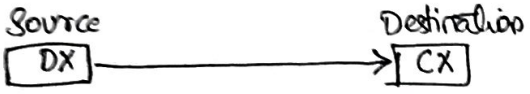
* Source is to the right and the destination is to the left, next to the opcode MOV.
* An opcode or operation code tells the microprocessor which operation to perform.
* In the above instruction, word contents of source register (BX) is transferred to destination register (AX).
* Source and destination are called as operands.

The data addressing modes are

① Register addressing

* It transfers a copy of a byte or word from source register or contents of a memory location to destination register or memory location.

Eg: MOV CX, DX



Copies word sized contents of DX to CX.

MOV ECX, EDX

Copies doubleword sized contents of EDX to ECX.

② Immediate Addressing

* It transfers the source, an immediate byte, word, doubleword, or quadword of data into destination register or memory location.

Eg: MOV AL, 22H

Copies byte sized 22H into register AL.

MOV EBX, 12345678H

Copies doubleword sized 12345678H into 32-bit wide EBX register

MOV CH, 3AH

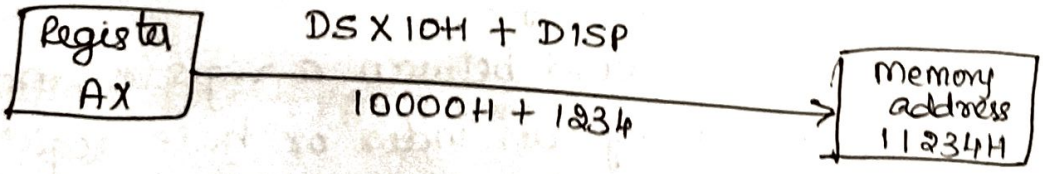
③ Direct addressing

* It moves a byte or word between memory location and a register.

Eg: MOV CX, LIST

Copies the word-sized contents of memory location LIST into register CX.

Eg: MOV [1234H], AX



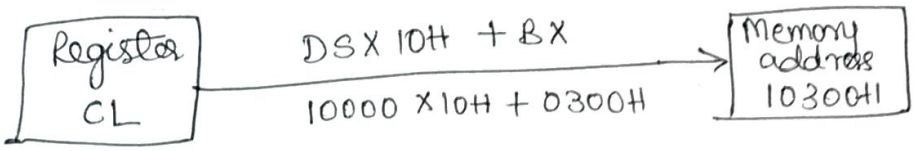
④ Register indirect addressing

* It 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

Eg: MOV AX, [BX]

Copies word-sized data from data segment offset address indexed by BX into register AX.

MOV [BX], CL



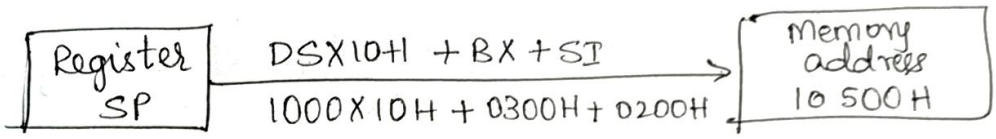
⑤ Base - plus - index addressing

* It transfess a byte or word between a register and memory location addressed by a base register (BP or BX) plus an index register (DI or SI).

Eg: MOV [BX+DI], CL

Copies byte-sized contents of register CL into data segment memory location addressed by BX plus DI.

MOV [BX + SI], SP



⑥ Register relative addressing

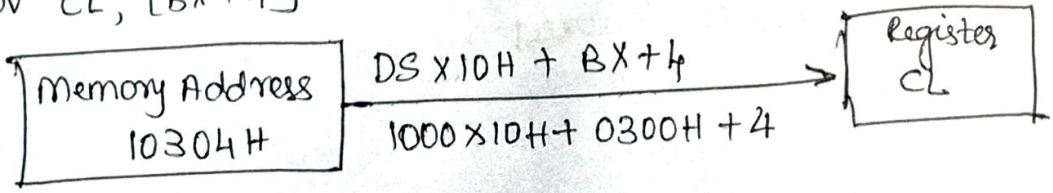
* It moves a byte or word between a register and the memory location addressed by an index or base register plus a displacement.

Eg: MOV AX, [BX + 4] or MOV AX, ARRAY [BX].

* It loads AX from data segment address formed by BX plus 4.

* loads AX from data segment memory location in ARRAY plus contents of BX.

MOV CL, [BX+4]



⑦ Base relative plus index addressing

* It transfers a byte or word between a register and memory location addressed by a base and an index register plus a displacement.

Eg: MOV AX, ARRAY [BX + DI] (or) MOV AX, [BX + DI + 4]

* ~~load AX from~~ uses an address formed by adding ARRAY, BX and DI.

* uses an address formed by adding BX, DI and 4.

```
MOV ARRAY[BX + SI], DX
```

Register DX → DS X 10H + ARRAY + BX + SI → Memory address

1000 X 10H + 1000H + 0300H + 0200H → 11500H

⑧ Scaled-index addressing

* It is available only in 80386 through Pentium 4 µp.
* The second register of a pair of registers is modified by scale factor of 2X, 4X or 8X to generate operand memory address.

Eg: MOV EDX, [EAX + 4 X EBX]

* load EDX from data segment memory location addressed by EAX plus four times EBX.

```
Mov [EBX + 2 X ESI], AX
```

Register AX → DS X 10H + EBX + 2X ESI → Memory Address

10000H + 00000300H + 2 X 00000200H → 10700H

Note:

For all the above examples consider the following values

EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, DS = 1000H

Register Addressing

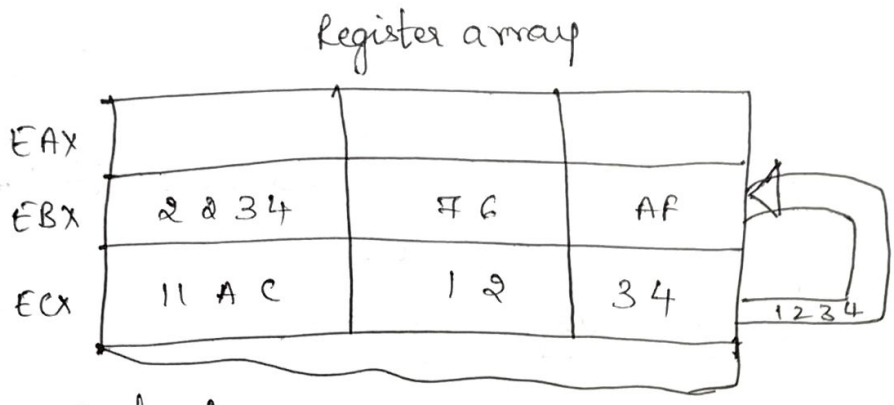
* 8-bit registers → AH, AL, BH, BL, CH, CL, DH and DL
* 16-bit registers → AX, BX, CX, DX, SP, BP, SI and DI
* 32-bit registers → EAX, EBX, ECX, EDX, ESP, EBP and EDI, ESI
* 64-bit registers → RAX, RBX, RCX, RDX, RSP, RBP, RDI, RSI and R8 through R15.
* With register addressing, some MOV instructions and the PUSH and POP instructions also used 16-bit segment registers such as CS, ES, DS, SS, FS and GS.
* The instructions should always used registers that are of same size.

Variations of register move instructions

1.	MOV AL, BL	8 bits	Copies BL to AL
2.	MOV R8B, CL		Not allowed
3.	MOV SP, BP	16 bits	Copies BP to SP.
4.	MOV BP, R10W	16 bits	Copies R10 into BP.
5.	MOV ECX, EBX	32 bits	Copies EBX into ECX.
6.	MOV EDX, R9D	32-bits	Copies R9 into EDX.
7.	MOV DS, CX	16 bits	Copies CX into DS.
8.	MOV ES, DS	-	Not allowed (Segment to Segment)
9.	MOV BL, DX	-	Not allowed (Mixed sizes)
10.	MOV CS, AX	-	Not allowed (Code Segment Register may not be destination register)

* Segment-to-segment MOV instruction is not allowed.
* Changing CS register with mov instruction is not allowed.

Operation of MOV BX, CX instruction

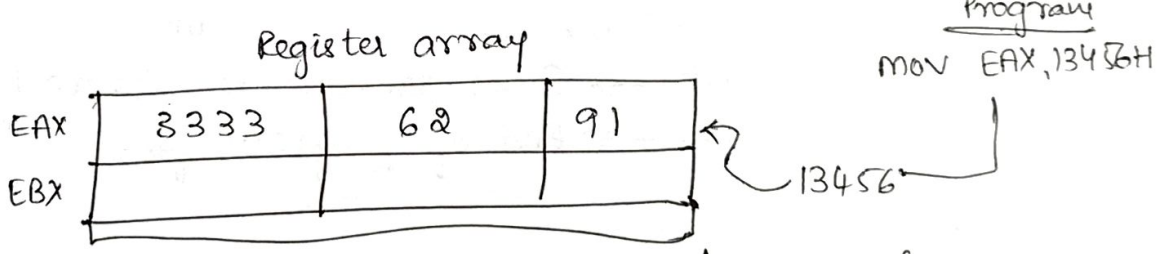


* The instruction copies 1234H from register CX into BX.
* This erases old content (76AFH) of register BX, but register CX contents remains unchanged.

Immediate Addressing

* Immediate data are constant data, whereas data transferred from a register or memory location are variable data.

Operation of MOV EAX, 13456H instruction



* This instruction copies 13456H from instruction, located in the memory immediately following the hexadecimal opcode, into register EAX.
* In symbolic assembly language, symbol # precedes immediate data in some assemblers.

  Eg: MOV AX, #3456H.

* If hexadecimal data begin with a letter, the assembler requires that the data start with a 0.

  Eg: To represent a hexadecimal F0, 0F0H is used in assembly language.

* Decimal data are represented as it is.

  Eg: 100 decimal is represented as MOV AL, 100.

* ASCII - coded characters are depicted in the immediate form if ASCII data is enclosed in apostrophes.

  Eg: MOV BH, 'A' → moves ASCII-coded letter A [41H] into BH.

* Binary data is represented by letter B.

  Eg: MOV CL, 11001110B.

Example:

- Model Tiny → Single code Segment / Segment
- Code → Start of Code Segment
- startup → starting instruction in the program.

MOV AX, 0 → place 0000H into AX
MOV BX, 0 → place 0000H into BX
MOV CX, 0 → place 0000H into CX
MOV SI, AX → copy AX into SI
MOV DI, AX → " " DI
MOV BP, AX → " " BP.

- Exit → Cause program to exit to DOS.

END. → End of program file.

Example 2

label	opcode	operand	Comment
DATA 1	DB	23H	;Define DATA1 as a byte of 23H
DATA 2	DW	1000H	;Define DATA2 as a word of 1000H
START:	MOV	AL, BL	
	MOV	BH, AL	
	MOV	CX, 200	

Examples of immediate addressing

1. MOV BL, 44 — Copies 44 decimal (2CH) into BL
2. MOV AX, 44H — Copies 0044H into AX
3. MOV CH, 100 — Copies 100 decimal (64H) into CH.
4. MOV AL, 'A' — Copies ASCII A into AL
5. MOV CL, 11001110B — Copies 11001110 binary into CL.

Direct Data Addressing

* Two basic forms of direct data addressing
  - ① Direct addressing → applies to a MOV b/w memory location and AL, AX or EAX.
  - ② Displacement addressing → Applies to almost any instruction in instruction set.
* In either case, address is formed by adding displacement to the default data segment address or an alternate segment address.

Direct addressing Example : MOV AL, DATA

↓

loads AL from data segment memory location.

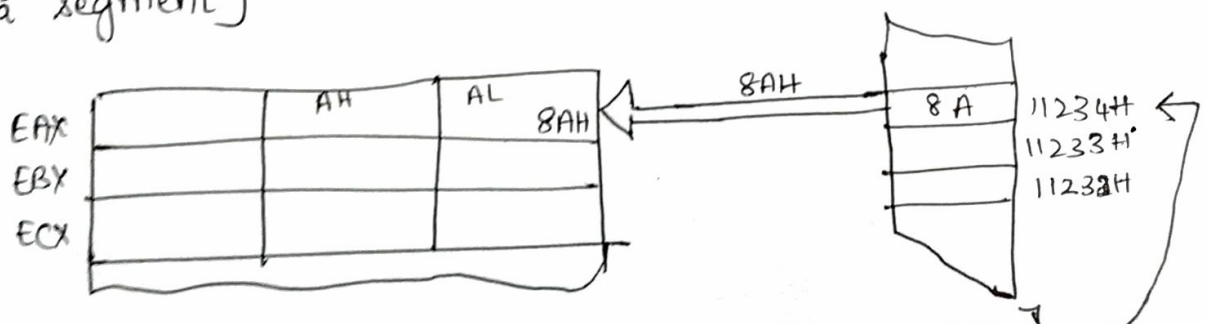
(or)

MOV AL, [1234H]

↓

absolute memory location.

[Interpreted as MOV AL, DS:[1234H] because 1234H is in data segment]



[Here DS = 1000H. Add 1234H + 1000H × 10 → 11234H]

Examples of Direct Addressing

1. MOV AL, NUMBER — Copies byte contents of data segment memory location NUMBER into AL
2. MOV NEWS, AL
3. MOV ES:[2000H], AL

Displacement Addressing

* It is almost identical to direct addressing, except that the instructions is 4 bytes wide instead of 3.

Examples: MOV AL, DS:[1234H]

MOV ES, DATA6

Register Indirect Addressing

* It 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.

Eg: If register BX contains 1000H and the MOV AX, [BX]

Refer class notes for operation (diagram) of addressing modes.

Program Memory - Addressing Modes

* Program memory addressing modes (used with JMP(jump) and CALL instructions consists of 3 distinct forms
  1. Direct
  2. Relative
  3. Indirect

Direct Program Memory Addressing

* They are used in early microprocessors for jumps and calls and also used in high level languages such as BASIC.
* The instructions for direct program memory addressing store the address with the opcode.

  Eg: If program jumps to memory location 10000H for next instruction, the address (10000H) is stored following the opcode in the memory.

Opcode	offset (low)	offset (high)	Segment (low)	Segment (high)
EA	00	00	00	10

Fig: Direct intersegment JMP instruction

* This JMP instruction loads CS with 1000H and IP with 0000H to jump to memory location 10000H for the next instruction.

  [Intersegment jump is a jump to any memory location within the entire memory system].

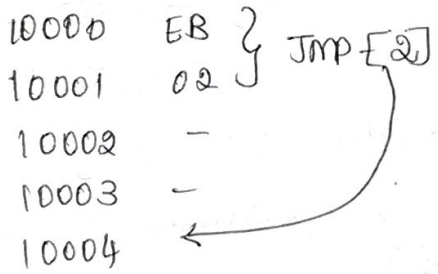
* Direct jump is called far jump because it can jump to any memory location for the next instruction.

* In real mode ⟶ far jump accesses any location within first 1M byte of memory
* In protected mode ⟶ far jump accesses a new code segment descriptor from the descriptor table allowing it to jump to any memory location in the entire 4G-byte address.
* The other instruction that uses direct program addressing is the intersegment or far CALL instruction.

Relative Program Memory Addressing

* The term relative means "relative to the instruction pointer (IP)".

Eg: If a JMP instruction skips the next 2 bytes of memory, the address in relation to the instruction pointer is a 2 that adds to instruction pointer. This forms the address of the next program instruction.



* JMP instruction is a 1-byte instruction with 1-byte or 2-byte displacement that adds to instruction pointer.
* A 1-byte displacement is used in short jumps
  2-byte displacement is used with near jumps & calls.
* Both are intrasegment jumps [intrasegment jump is a jump anywhere within the current code segment].
* 8-bit displacement (short) has a jump range blw +127 and -128 bytes from next instruction.
* 16-bit displacement (near) has a range ± 32K bytes.
* 32-bit displacement allows a range of ±2G bytes.

Indirect Program Memory Addressing

1. JMP AX → Jumps to current code segment location addressed by contents of AX

2. JMP NEAR PTR[BX] → Jumps to current code segment location addressed by contents of data segment location addressed by BX.

3. JMP NEAR PTR[DI+2] → Jumps to current code segment location addressed by contents of data segment memory location addressed by DI plus 2.

4. JMP TABLE [BX]

5. JMP ECX

6. JMP RDI

* All the above acceptable program indirect jump instructions can use 16-bit register, any relative register ([BP], [BX], [DI] or [SI]) and any relative register with a displacement.

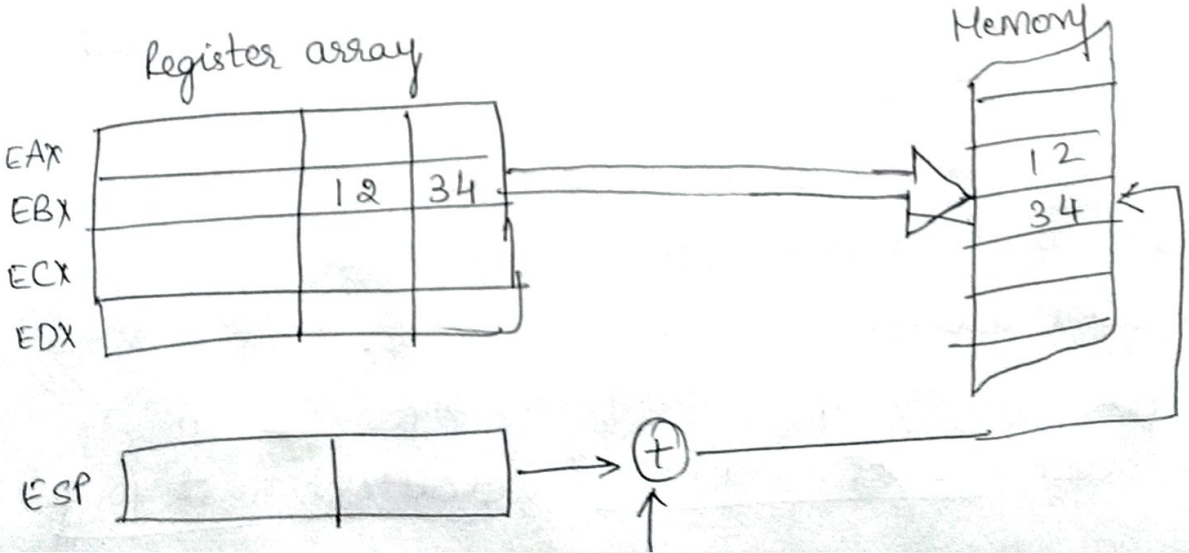
* If 16-bit register holds address of JMP, instruction, jump is near. Eg: If BX contains 1000H and JMP BX is executed, µp jumps to offset address 1000H in current code segment.

* If relative register holds the address, jump is considered to be an indirect jump.
  Eg: JMP [BX] refers to memory location within data segment at the offset address contained in BX. At this offset address is a 16-bit number that is used as the offset address in intrasegment jump. This type of jump is called indirect - indirect or double-indirect jump.

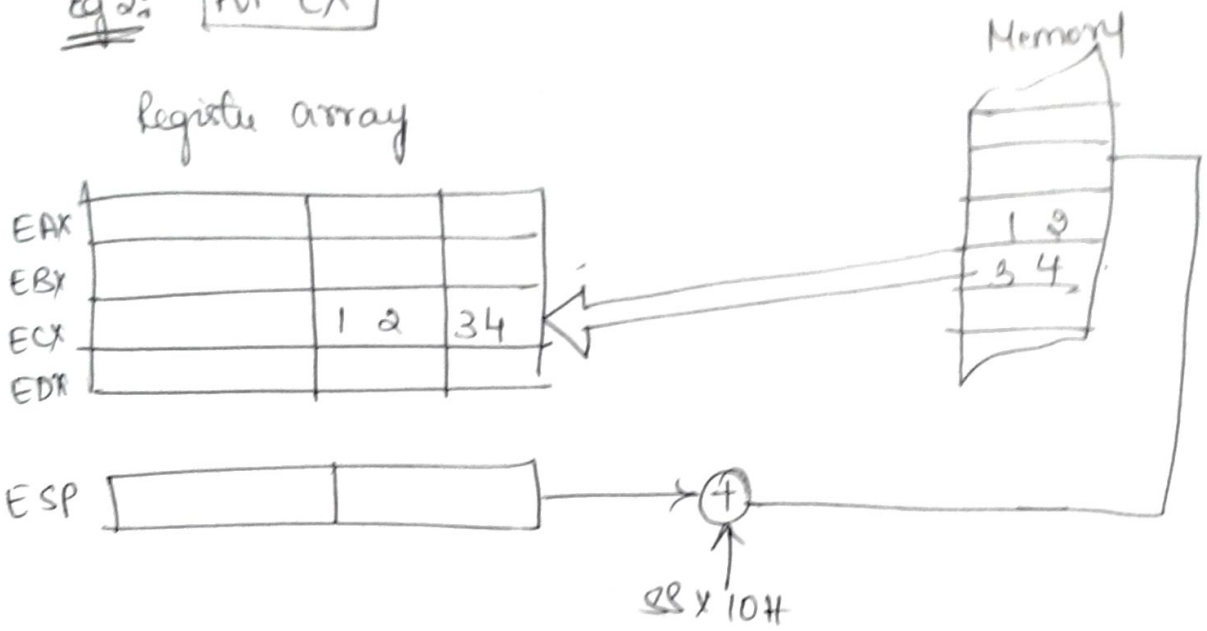
Stack Memory Addressing Mode

* Stack holds data temporarily and stores the return address used by procedures.
* Stack memory is LIFO memory.
* Data are placed onto stack with a PUSH instruction and removed with a POP instruction.
* CALL instruction uses the stack to hold return address for procedures and RET (return) instruction to remove the return address from the stack.
* Stack memory is maintained by two registers.
  1. Stack Pointer (SP or ESP)
  2. Stack Segment Register (SS)
* whenever a word of data is pushed onto stack, the high-order 8-bits are placed in the location addressed by SP-1.
* The low-order 8-bits are placed in the location addressed by SP-2.
* The SP is then decremented by 2 so that next word of data is stored in next available stack memory location.

Eg1: `PUSH BX`



Eg 2: POP CX



* SP/ESP always points to an area of memory located within stack segment.
* Real mode → SP/ESP register adds to SS X 10H to form stack memory address.
* Protected mode → SS register holds a selector that accesses a descriptor for base address of stack segment.
* In Eg:2, whenever data are popped from stack, low-order 8 bits are removed from location addressed by SP. Higher order 8-bits are removed from the location addressed by SP+1. SP register is then incremented by 2.

Examples of PUSH and POP instructions

1. POPF → Removes a word from stack and places it into flag register.
2. PUSHF → Copies flag register to the stack.
3. PUSHFD → Copies EFLAG register to stack.
4. PUSH 1234H → Copies a word-sized 1234H to the stack.

* PUSHA and POPA instructions either push or pop all of the registers, except segment registers, onto stack.

Eg: Program that pushes contents of AX, BX and CX onto stack.

```
.model TINY
.code
.startup
mov ax, 1000H
mov bx, 2000H
mov cx, 3000H
Push ax  ----> 1000H to stack
push bx  ----> 2000H to stack
push cx  ----> 3000H to stack
pop ax   ----> 3000H to ax
pop cx   ----> 2000H to cx
pop bx   ----> 1000H to bx
.exit
end
```