Chapter 3 Addressing Modes

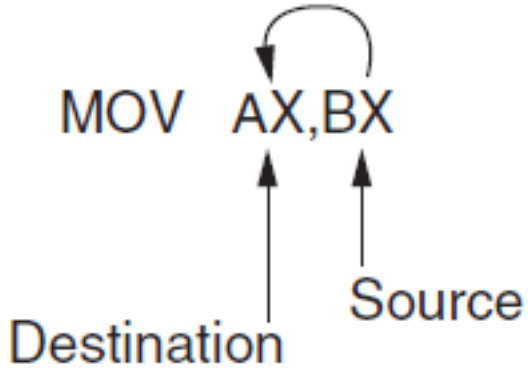
Assoc. Prof. Dr. Gazi Erkan BOSTANCI

Slides are mainly based on The Intel Microprocessors by Barry B. Brey, 2008

- Efficient software development for the MP requires a complete familiarity with the addressing modes employed by each instruction.
- MOV (move data) instruction will be used to describe the data addressing modes.
- CALL and JUMP instructions will be used to describe program memory addressing modes.
- PUSH and POP instructions will be used to describe the operation of the stack memory.

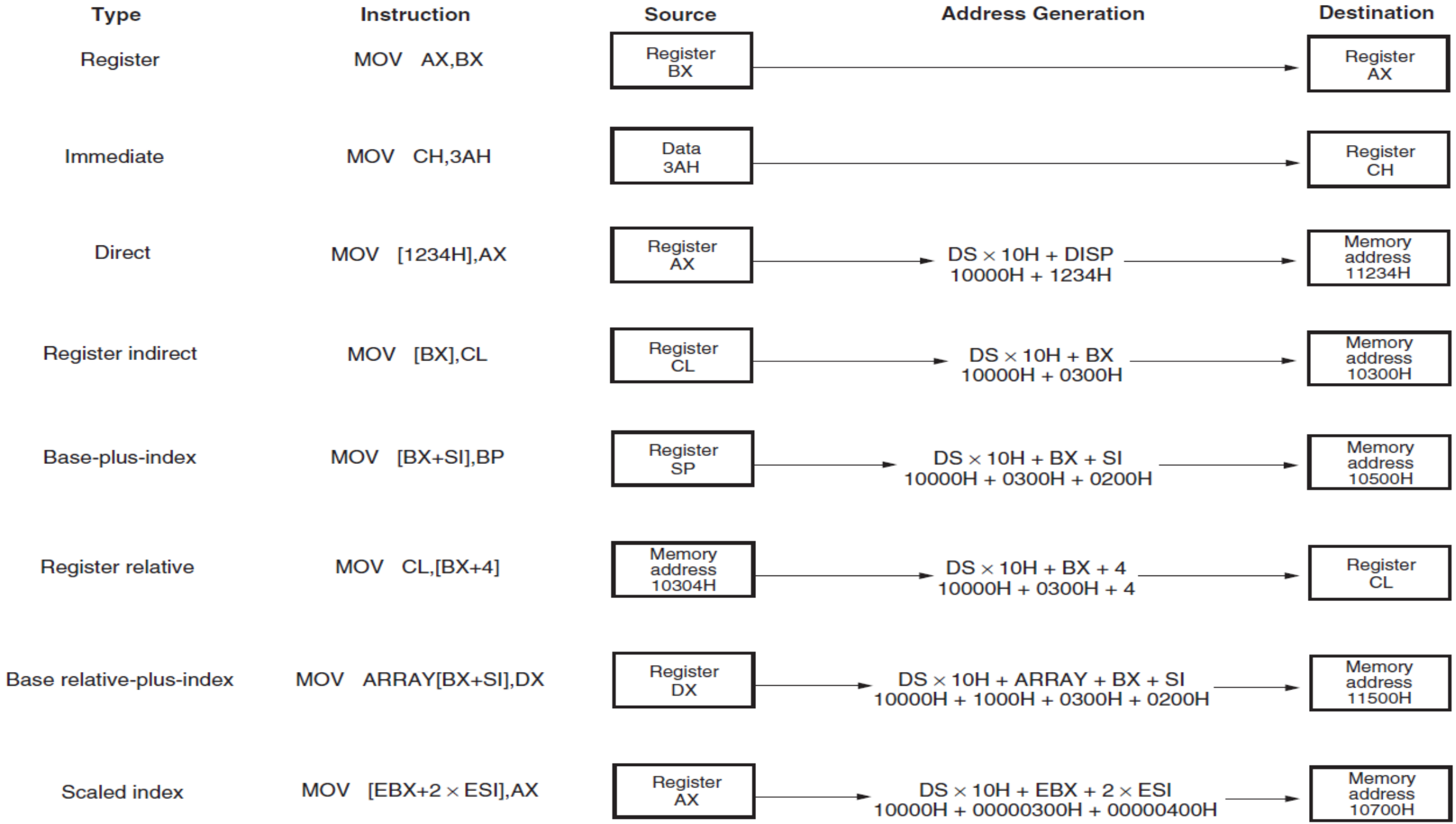
Data Addressing Modes

- Because the MOV instruction is a very common and flexible instruction, it provides a basis for the explanation of the data-addressing modes.
- Figure below illustrates the MOV instruction and defines the direction of data flow.



- The **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.)
- This direction of flow, which is applied to all instructions, is awkward at first.
  - We naturally assume that things move from left to right, whereas here they move from right to left.

- The MOV AX, BX instruction transfers the word contents of the source register (BX) into the destination register (AX). The source never changes, but the destination always changes.
- It is crucial to remember that a MOV instruction always *copies* the source data into the destination. The MOV never actually picks up the data and moves it.
- Also, note the flag register remains unaffected by most data transfer instructions. The source and destination are often called **operands**.
- Figure below shows all possible variations of the data-addressing modes using the MOV instruction. This illustration helps to show how each data-addressing mode is formulated with the MOV instruction and also serves as a reference on data-addressing modes.



Notes: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, and DS = 1000H

- **Register addressing:** Register addressing transfers a copy of a byte or word from the source register or contents of a memory location to the destination register or memory location.
  - Example: The MOV CX, DX instruction copies the word-sized contents of register DX into register CX.

Assembly Language	*Size*	*Operation*
MOV AL,BL	8 bits	Copies BL into AL
MOV CH,CL	8 bits	Copies CL into CH
MOV R8B,CL	8 bits	Copies CL to the byte portion of R8 (64-bit mode)
MOV R8B,CH	8 bits	Not allowed
MOV AX,CX	16 bits	Copies CX into AX
MOV SP,BP	16 bits	Copies BP into SP
MOV DS,AX	16 bits	Copies AX into DS
MOV BP,R10W	16 bits	Copies R10 into BP (64-bit mode)
MOV SI,DI	16 bits	Copies DI into SI
MOV BX,ES	16 bits	Copies ES into BX
MOV ECX,EBX	32 bits	Copies EBX into ECX
MOV ESP,EDX	32 bits	Copies EDX into ESP
MOV EDX,R9D	32 bits	Copies R9 into EDX (64-bit mode)
MOV RAX,RDX	64 bits	Copies RDX into RAX
MOV DS,CX	16 bits	Copies CX into DS
MOV ES,DS	—	Not allowed (segment-to-segment)
MOV BL,DX	—	Not allowed (mixed sizes)
MOV CS,AX	—	Not allowed (the code segment register may not be the destination register)

- **Immediate addressing:** Immediate addressing transfers the source, an immediate byte, word, doubleword, or quadword of data, into the destination register or memory location.
  - Example: The MOV AL, 22H instruction copies a byte-sized 22H into register AL

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

*Note: This is not an error. The ASCII characters are stored as BA, so exercise care when using word-sized pairs of ASCII characters.

- **Direct addressing:** Direct addressing moves a byte or word between a memory location and a register. The instruction set does not support a memory-to memory transfer, except with the MOVS instruction.
  - Example: The MOV CX, LIST instruction copies the word-sized contents of memory location LIST into register CX.

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

- **Register indirect addressing:** Register indirect addressing transfers a byte or word between a register and a memory location addressed by an index or base register. The index and base registers are BP, BX, DI, and SI.
  - Example: The MOV AX, [BX] instruction copies the word-sized data from the data segment offset address indexed by BX into register AX.

Assembly Language	*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 (64-bit mode)

*Note: Data addressed by BP or EBP are by default in the stack segment, while other indirect addressed instructions use the data segment by default.

- **Base-plus-index addressing:** Base-plus-index addressing transfers a byte or word between a register and the memory location addressed by a base register (BP or BX) plus an index register (DI or SI).
  - Example: The MOV [BX+DI], CL instruction copies the byte-sized contents of register CL into the data segment memory location addressed by BX plus DI.

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

- **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 plus a displacement.
  - Example: MOV AX,[BX+4] or MOV AX, ARRAY[BX].
  - The first instruction loads AX from the data segment address formed by BX plus 4. The second instruction loads AX from the data segment memory location in ARRAY plus the contents of BX.

Assembly Language	*Size*	*Operation*
MOV AX,[DI+100H]	16 bits	Copies the word contents of the data segment memory location addressed by DI plus 100H into AX
MOV ARRAY[SI],BL	8 bits	Copies BL into the data segment memory location addressed by ARRAY plus SI
MOV LIST[SI+2],CL	8 bits	Copies CL into the data segment memory location addressed by the sum of LIST, SI, and 2
MOV DI,SET_IT[BX]	16 bits	Copies the word contents of the data segment memory location addressed by SET_IT plus BX into DI

- **Base relative-plus-index addressing** Base relative-plus-index addressing transfers a byte or word between a register and the memory location addressed by a base and an index register plus a displacement.
  - Example: MOV AX, ARRAY[BX+DI] or MOV AX, [BX+DI+4].
  - These instructions load AX from a data segment memory location. The first instruction uses an address formed by adding ARRAY, BX, and DI and the second by adding BX, DI, and 4.

Assembly Language	*Size*	*Operation*
MOV DH,[BX+DI+20H]	8 bits	Copies the byte contents of the data segment memory location addressed by the sum of BX, DI and 20H into DH
MOV AX,FILE[BX+DI]	16 bits	Copies the word contents of the data segment memory location addressed by the sum of FILE, BX and DI into AX
MOV LIST[BP+DI],CL	8 bits	Copies CL into the stack segment memory location addressed by the sum of LIST, BP, and DI
MOV LIST[BP+SI+4],DH	8 bits	Copies DH into the stack segment memory location addressed by the sum of LIST, BP, SI, and 4

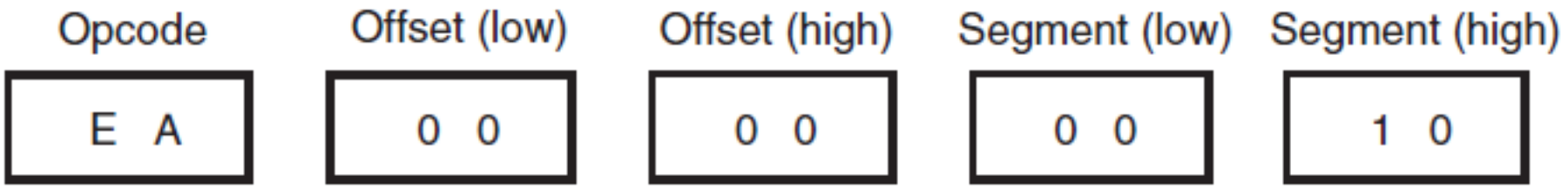
Program Memory Addressing Modes

- Program memory-addressing modes, used with the JMP (jump) and CALL instructions, consist of three distinct forms:
  - direct,
  - relative,
  - and indirect.
- This section introduces these three addressing forms, using the JMP instruction to illustrate their operation.

Direct Program Memory Addressing

- Direct program memory addressing is what many early microprocessors used for all jumps and calls.
- Direct program memory addressing is also used in high-level languages, such as the GOTO instruction.
- The microprocessor uses this form of addressing, but not as often as relative and indirect program memory addressing are used.

- The instructions for direct program memory addressing store the address with the opcode. For example, if a program jumps to memory location 10000H for the next instruction, the address (10000H) is stored following the opcode in the memory.
  - Figure shows the direct intersegment JMP instruction and the 4 bytes required to store the address 10000H.
  - This JMP instruction loads CS with 1000H and IP with 0000H to jump to memory location 10000H for the next instruction.

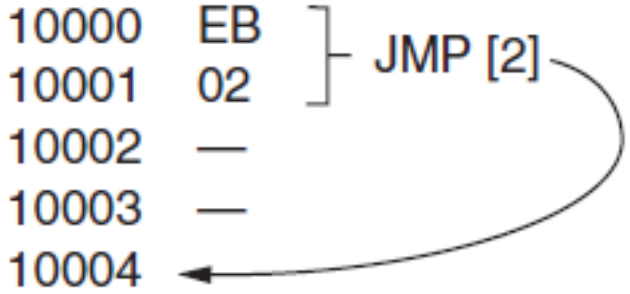


- An **intersegment jump** is a jump to any memory location within the entire memory system.) The direct jump is often called a *far jump* because it can jump to any memory location for the next instruction.

Relative Program Memory Addressing

- Relative program memory addressing is not available in all early microprocessors, but it is available to this family of microprocessors. The term *relative* means “relative to the instruction pointer (IP)”.
- For example, 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 the instruction pointer. This develops the address of the next program instruction.

- An example of the relative JMP instruction is shown below.



- Note that the JMP instruction is a 1-byte instruction, with a 1-byte or a 2-byte displacement that adds to the instruction pointer. A 1-byte displacement is used in **short** jumps, and a 2-byte displacement is used with **near** jumps and calls. Both types are considered to be intrasegment jumps.
- An **intrasegment jump** is a jump anywhere within the current code segment.

Indirect Program Memory Addressing

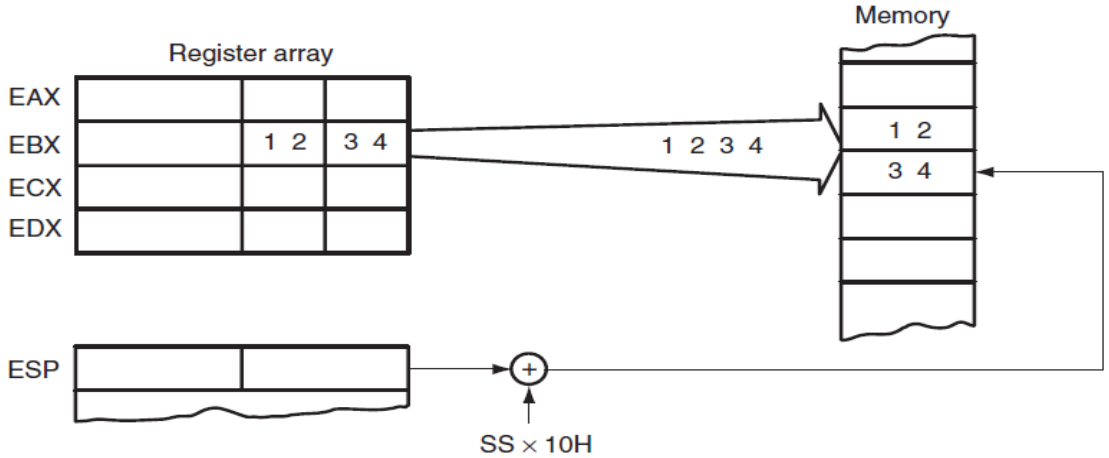
- The microprocessor allows several forms of program indirect memory addressing for the JMP and CALL instructions.

Assembly Language	*Operation*
JMP AX	Jumps to the current code segment location addressed by the contents of AX
JMP CX	Jumps to the current code segment location addressed by the contents of CX
JMP NEAR PTR[BX]	Jumps to the current code segment location addressed by the contents of the data segment location addressed by BX
JMP NEAR PTR[DI+2]	Jumps to the current code segment location addressed by the contents of the data segment memory location addressed by DI plus 2
JMP TABLE[BX]	Jumps to the current code segment location addressed by the contents of the data segment memory location address by TABLE plus BX

Stack Memory Addressing Modes

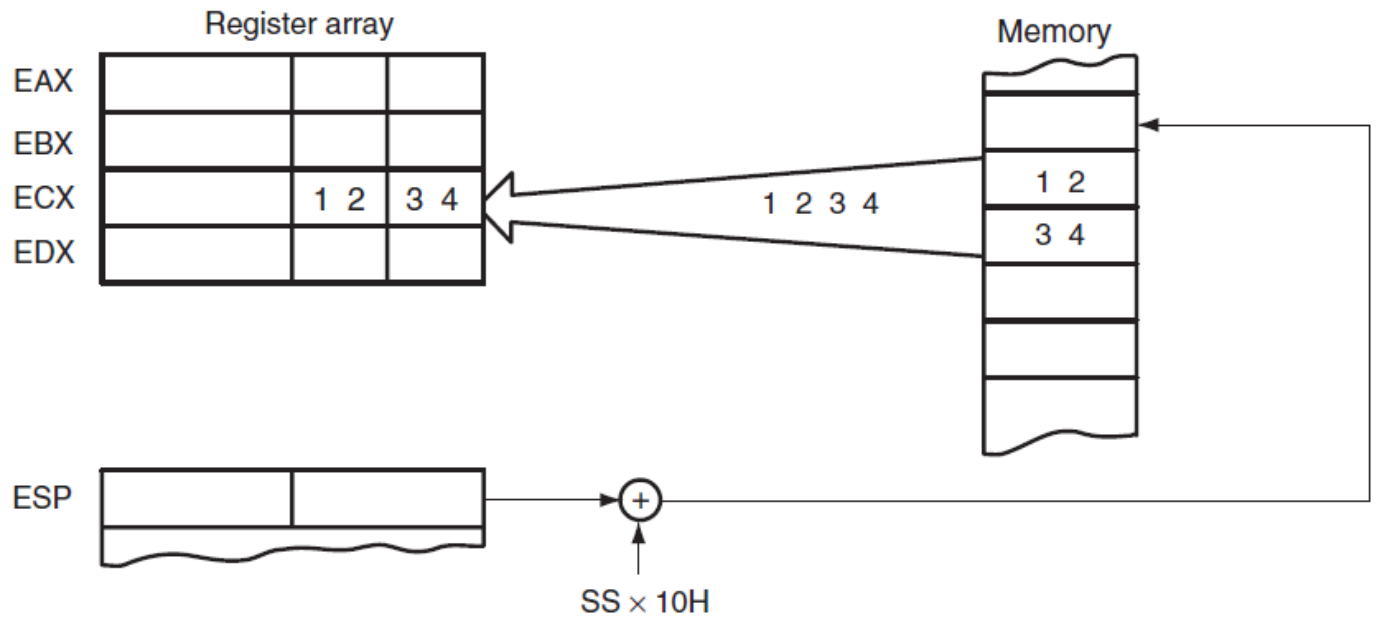
- The stack plays an important role in all microprocessors. It holds data temporarily and stores the return addresses used by procedures. The stack memory is an LIFO (**last-in, first-out**) memory, which describes the way that data are stored and removed from the stack.
- Data are placed onto the stack with a **PUSH instruction** and removed with a **POP instruction**.
- The CALL instruction also uses the stack to hold the return address for procedures and a RET (return) instruction to remove the return address from the stack.

- The stack memory is maintained by two registers: the stack pointer (SP) and the stack segment register (SS). Whenever a word of data is pushed onto the 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 the next word of data is stored in the next available stack memorv location.



- The SP register always points to an area of memory located within the stack segment. The SP register adds to SS x 10h to form the stack memory address in the real mode.

- Whenever data are popped from the stack, the low-order 8 bits are removed from the location addressed by SP. The high-order 8 bits are removed from the location addressed by SP+1. The SP register is then incremented by 2.



- Note that **PUSH** and **POP** store or retrieve **words of data**—never bytes—in the 8086 microprocessors.

Assembly Language	*Operation*
POPF	Removes a word from the stack and places it into the flag register
POPFD	Removes a doubleword from the stack and places it into the EFLAG register
PUSHF	Copies the flag register to the stack
PUSHFD	Copies the EFLAG register to the stack
PUSH AX	Copies the AX register to the stack
POP BX	Removes a word from the stack and places it into the BX register
PUSH DS	Copies the DS register to the stack
PUSH 1234H	Copies a word-sized 1234H to the stack
POP CS	This instruction is illegal
PUSH WORD PTR[BX]	Copies the word contents of the data segment memory location addressed by BX onto the stack
PUSHA	Copies AX, CX, DX, BX, SP, BP, DI, and SI to the stack
POPA	Removes the word contents for the following registers from the stack: SI, DI, BP, SP, BX, DX, CX, and AX

- Data may be popped off the stack into any register or any segment register except CS. The reason that data may not be popped from the stack into CS is that this only changes part of the address of the next instruction.

- Example Swap operation using stack

```
mov ax, 1000h
mov bx, 2000h
push ax; 1000h to stack
push bx; 2000h to stack
pop ax; 2000h to ax
pop bx; 1000h to bx
```