CSE-3103: Microprocessor and Microcontroller

Dept. of Computer Science and Engineering
University of Dhaka

Prof. Sazzad M.S. Imran, PhD
Dept. of Electrical and Electronic Engineering
sazzadmsi.webnode.com

Unconditional Jumps

Jump (JMP) allows programmer →

skip sections of program,
branch to any part of memory for next instruction.

3 types of unconditional jump instructions →

(i) Short jump →

jumps or branches to memory locations within +127 and -128 bytes.

(ii) Near jump →

branch or jump within ±32K bytes or
anywhere in current code segment.

(iii) Far jump:

jump to any memory location within real memory system.

Short and near jumps = intrasegment jumps,
Far jumps = intersegment jumps.

Unconditional Jumps

Short Jump →

Short jumps = relative jumps.

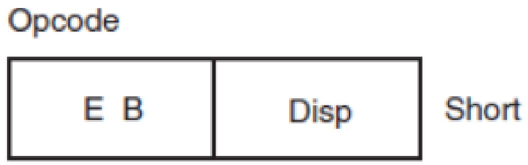
Can be moved to any location in current code segment.

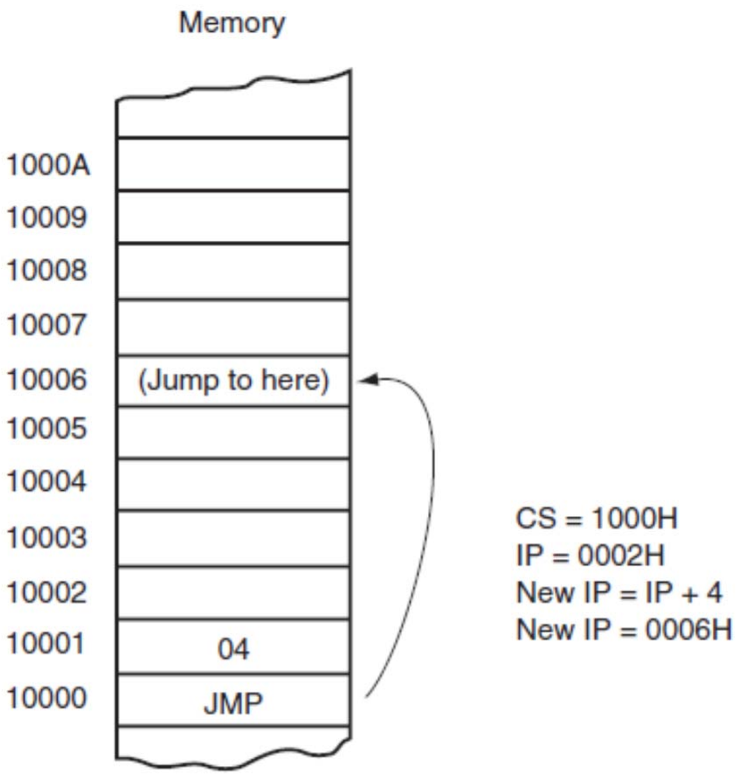
Jump address is not stored with opcode.

Distance or displacement follows opcode.

Displacement value = between +127 and -128.

Jump address = displacement is sign extended + IP.





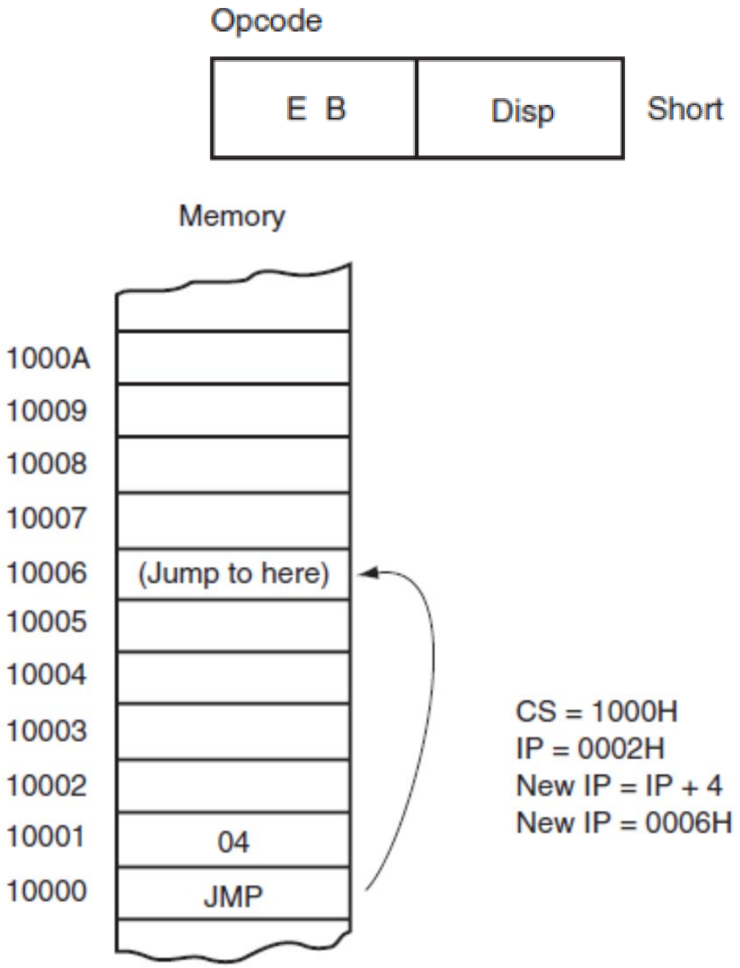
Unconditional Jumps

Short Jump →

```
                XOR     BX, BX
  START:        MOV     AX, 1
                ADD     AX, BX
                JMP     SHORT  NEXT

  ; <skipped memory locations>

  NEXT:         MOV     BX, AX
                JMP     START
```



CS = 1000H
IP = 0002H
New IP = IP + 4
New IP = 0006H

Unconditional Jumps

Near Jump →

passes control to instruction in current code segment located within ±32K bytes.

3-byte instruction =

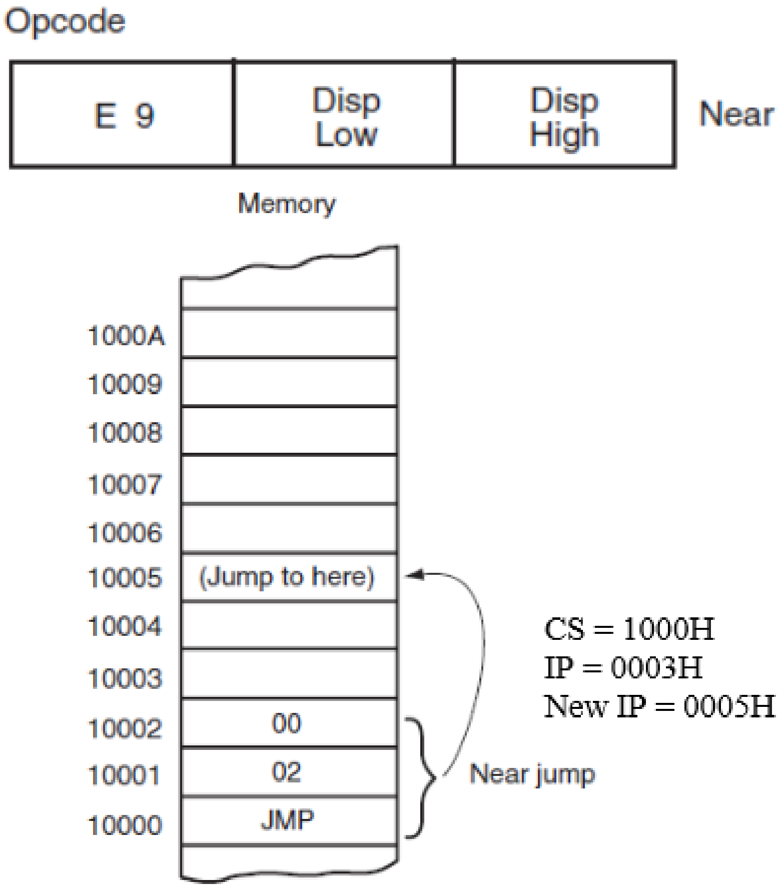
opcode + signed 16-bit displacement.

signed displacement + IP = jump address.

can jump to any memory location within current code segment.

relocatable and relative jump →

if code segment moves to new location, distance between jump instruction and operand address remains same.



CS = 1000H
IP = 0003H
New IP = 0005H

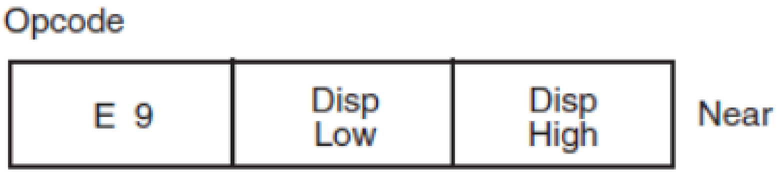
Unconditional Jumps

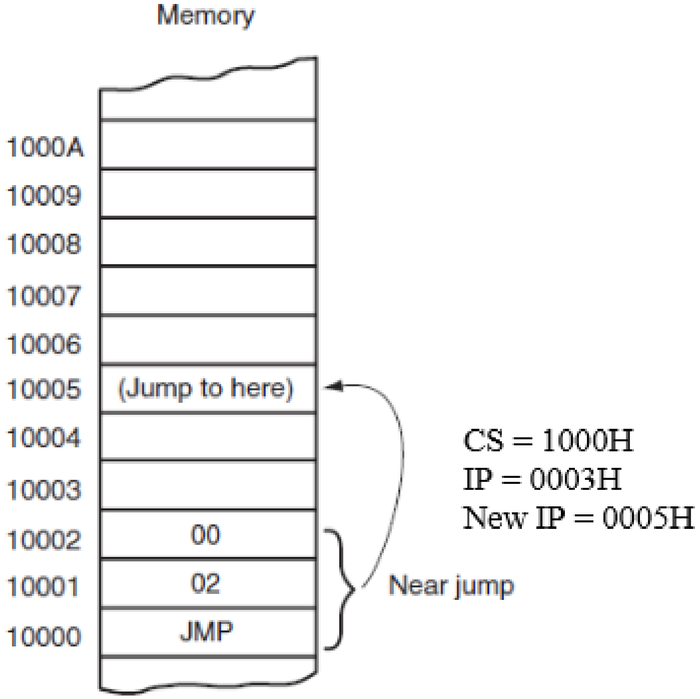
Near Jump →

```
            XOR     BX, BX
 START:     MOV     AX, 1
            ADD     AX, BX
            JMP     NEXT

 ; <skipped memory locations>

 NEXT:      MOV     BX, AX
            JMP     START
```





Unconditional Jumps

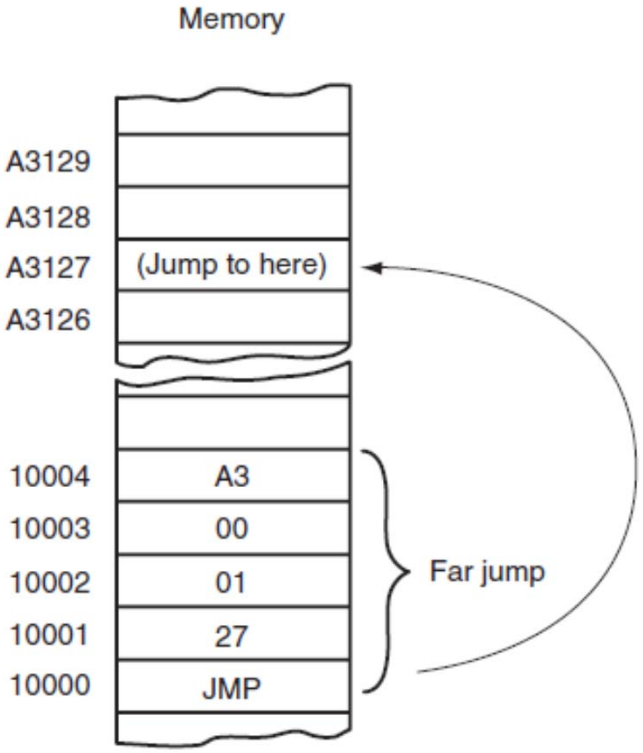
Far Jump →

instruction obtains new segment and offset address to accomplish jump.
bytes 2 and 3 = new offset address;
bytes 4 and 5 = new segment address.
instruction appears with FAR PTR directive.
obtained by defining label as far label.

```
                EXTRN   UP:FAR
                XOR     BX, BX
START:          ADD     AX, 1
                JMP     NEXT

; <skipped memory locations>

NEXT:           MOV     BX, AX
                JMP     FAR  PTR  START
                JMP     UP
```



Opcode

E A	IP Low	IP High	CS Low	CS High	Far

Conditional Jumps

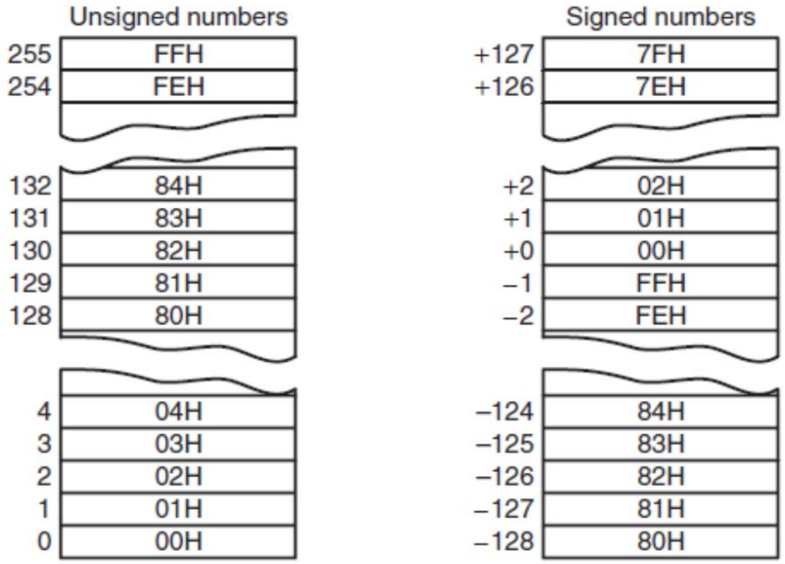
Conditional jump instructions = short jumps.
Range of jump = +127 bytes to -128 bytes.

Conditional jump instructions test flag bits →
sign (S), zero (Z), carry (C), parity (P), overflow (O).

Test condition is true =
branch to label associated with jump occurs.
Test condition is false =
next sequential step in program executes.

2 sets of conditional jump instructions for comparison.
When signed numbers are compared,
JG, JL, JGE, JLE, JE, JNE instructions.

When unsigned numbers are compared,
JA, JB, JAE, JBE, JE, JNE instructions.



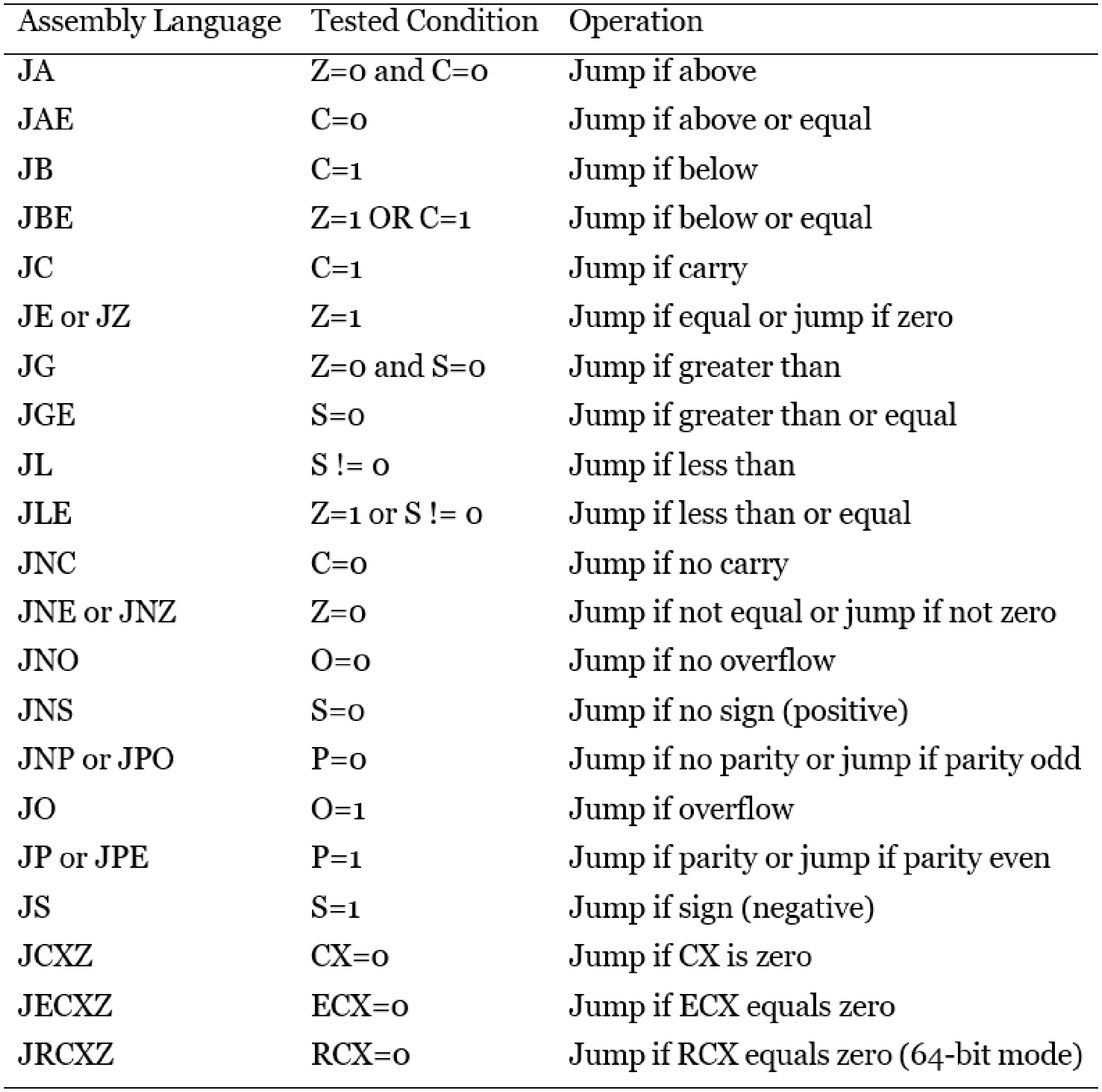
Unsigned numbers

255	FFH
254	FEH
132	84H
131	83H
130	82H
129	81H
128	80H
4	04H
3	03H
2	02H
1	01H
0	00H

Signed numbers

+127	7FH
+126	7EH
+2	02H
+1	01H
+0	00H
−1	FFH
−2	FEH
−124	84H
−125	83H
−126	82H
−127	81H
−128	80H

Conditional Jumps



Assembly Language	Tested Condition	Operation
JA	Z=0 and C=0	Jump if above
JAE	C=0	Jump if above or equal
JB	C=1	Jump if below
JBE	Z=1 OR C=1	Jump if below or equal
JC	C=1	Jump if carry
JE or JZ	Z=1	Jump if equal or jump if zero
JG	Z=0 and S=0	Jump if greater than
JGE	S=0	Jump if greater than or equal
JL	S != 0	Jump if less than
JLE	Z=1 or S != 0	Jump if less than or equal
JNC	C=0	Jump if no carry
JNE or JNZ	Z=0	Jump if not equal or jump if not zero
JNO	O=0	Jump if no overflow
JNS	S=0	Jump if no sign (positive)
JNP or JPO	P=0	Jump if no parity or jump if parity odd
JO	O=1	Jump if overflow
JP or JPE	P=1	Jump if parity or jump if parity even
JS	S=1	Jump if sign (negative)
JCXZ	CX=0	Jump if CX is zero
JECXZ	ECX=0	Jump if ECX equals zero
JRCXZ	RCX=0	Jump if RCX equals zero (64-bit mode)

Conditional Jumps

```
; SCASB searches a table of 100 bytes for 0AH
; address of TABLE is assumed to be in ES:DI

          MOV    CX, 100                  ; load counter
          MOV    AL, 0AH                  ; load AL with 0AH
          CLD                             ; auto-increment
          REPNE  SCASB                    ; search for 0AH
                                          ; repeat SCASB until CX=0
          JCXZ   NOT_FOUND                ; if not found
          STC                             ; set carry if found
NOT_FOUND
```

CALL Instruction

CALL transfers flow of program to procedure.
CALL saves return address on stack.
Return address = instruction that immediately follows CALL.
RET = returns control to Return address.

Near CALL →

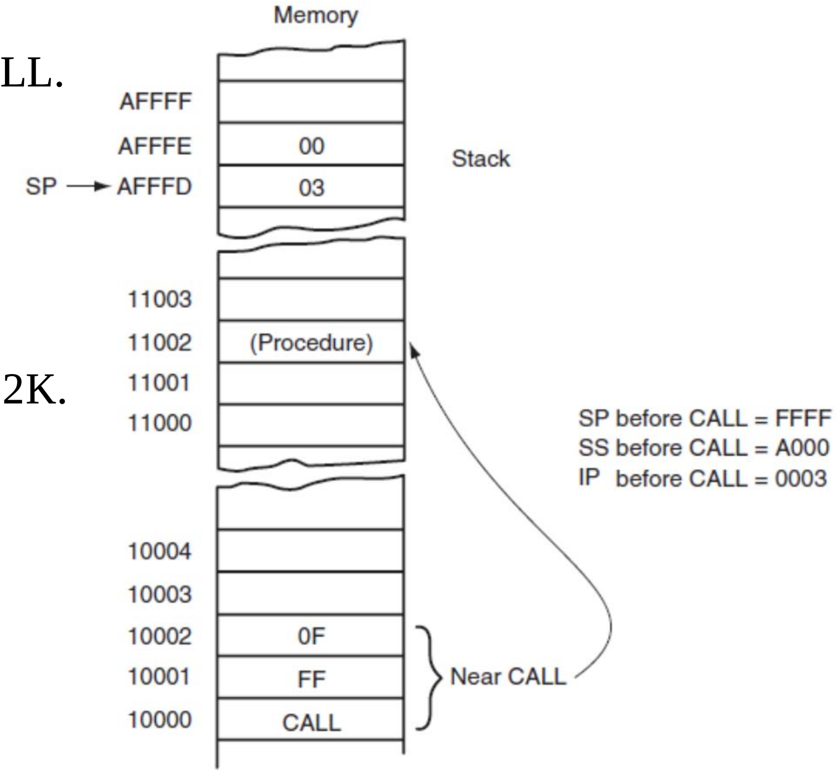
Near CALL = 3 bytes long;
1st byte = opcode,
2nd + 3rd bytes = displacement, or distance of ±32K.

When near CALL executes,
i) pushes offset address of next instruction onto stack.
ii) IP + displacement →

transfer control to procedure.

iii) RET →

pops offset address from stack (saved in step-i).
program control passes to instruction following CALL.



CALL Instruction

Far CALL →

Call procedure stored in any memory location.

Far CALL = 5-byte instruction =

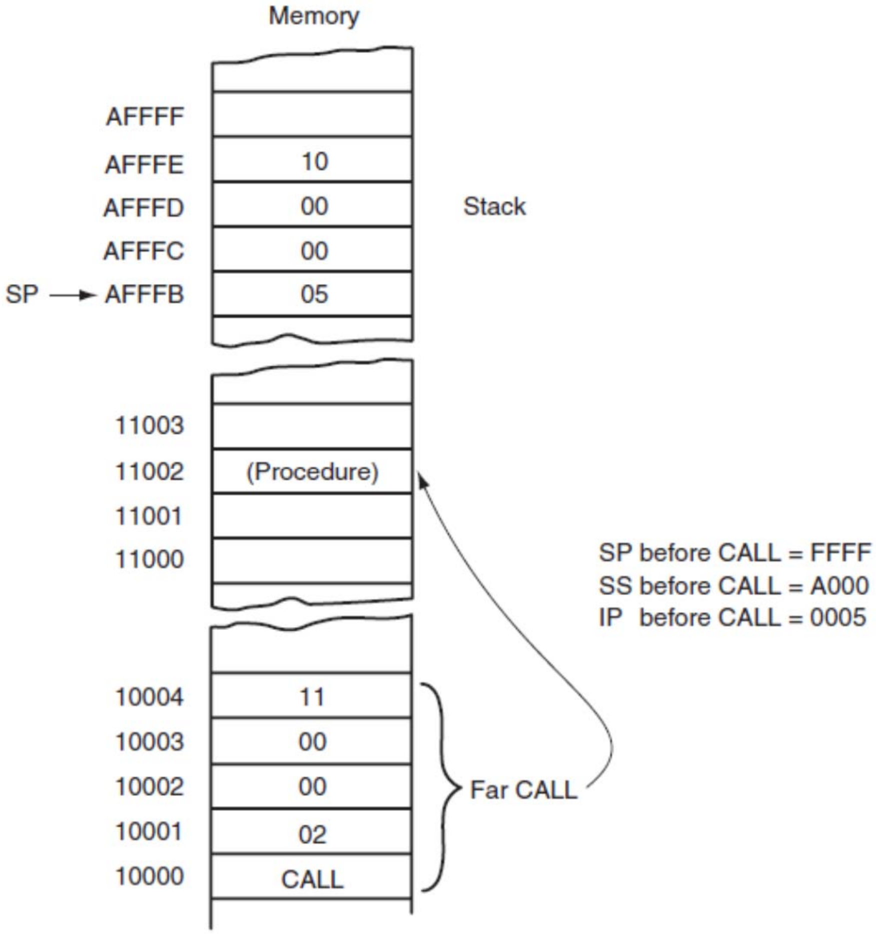
Opcode +

Bytes 2 and 3 = new contents of IP +

Bytes 4 and 5 = new contents for CS.

Far CALL pushes contents of both IP and CS on stack.

It then jumps to address indicated by bytes 2 through 5.


Memory
AFFFF
AFFFE
10
AFFFD
00
Stack
AFFFC
00
SP
AFFFB
05
11003
11002
(Procedure)
11001
11000
SP before CALL = FFFF
SS before CALL = A000
IP before CALL = 0005
10004
11
10003
00
10002
00
Far CALL
10001
02
10000
CALL