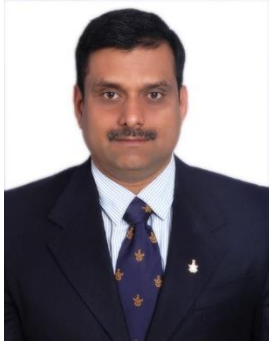
Microprocessors and Interfaces: 2021-22
Lecture 17
8086 Branching & Program Control Instructions : Part-2

By Dr. Sanjay Vidhyadharan



LOOP

- A combination of a decrement CX and the JNZ conditional jump.

- In 8086 , LOOP decrements CX.
  - if CX not equal to 0, it jumps to the address indicated by the label
  - If CX becomes 0, the next sequential instruction executes

The Loop instruction decrements CX without changing any flags

Conditional LOOPs

- LOOP instruction also has conditional forms:

1. LOOPZ/LOOPE: Loop while (ZF = 1) && (CX<> 0)
2. LOOPNZ/LOOPNE: Loop while (ZF = 0) && (CX <> 0)

```
Loop until '7' is found, ; or 5 times.
Data
v1 db 9, 8, 7, 6, 5
Code
ORG 100h
MOV CX, 5
LEA SI, v1
label1: MOV AL, [SI]
INC SI ;
CMP AL, 7
LOOPNE label1
RET
```

Conditional LOOPs

- LOOPE same as LOOPZ
- LOOPNE instruction is the same as LOOPNZ

Ex:

```
        MOV  BX, OFFSET ARRAY
        MOV CX, 100
NEXT:  INC BX
        CMP [BX], 0FFH
        LOOPNE NEXT
```

PROCEDURES

- A procedure is a group of instructions that usually performs one task.
  - subroutine, method, or **function** is an important part of any system's architecture

- A procedure is a reusable section of the software stored in memory once, used as often as necessary.
  - saves memory space and makes it easier to develop software

PROCEDURES

- Disadvantage of procedure is time it takes the computer to link to, and return from it.
  - **CALL** links to the procedure; the RET (**return**) instruction returns from the procedure
- **CALL** pushes the address of the instruction following the CALL (**return address**) on the stack.
  - the stack stores the return address when a procedure is called during a program
- RET instruction removes an address from the stack so the program returns to the instruction following the CALL.

PROCEDURES

- A procedure begins with the PROC directive and ends with the ENDP directive.
  - each directive appears with the procedure name
- PROC is followed by the type of procedure:
  - NEAR or FAR
- Procedures that are to be used by all software (global) should be written as far procedures.
- Procedures that are used by a given task (local) are normally defined as near procedures.

CALL

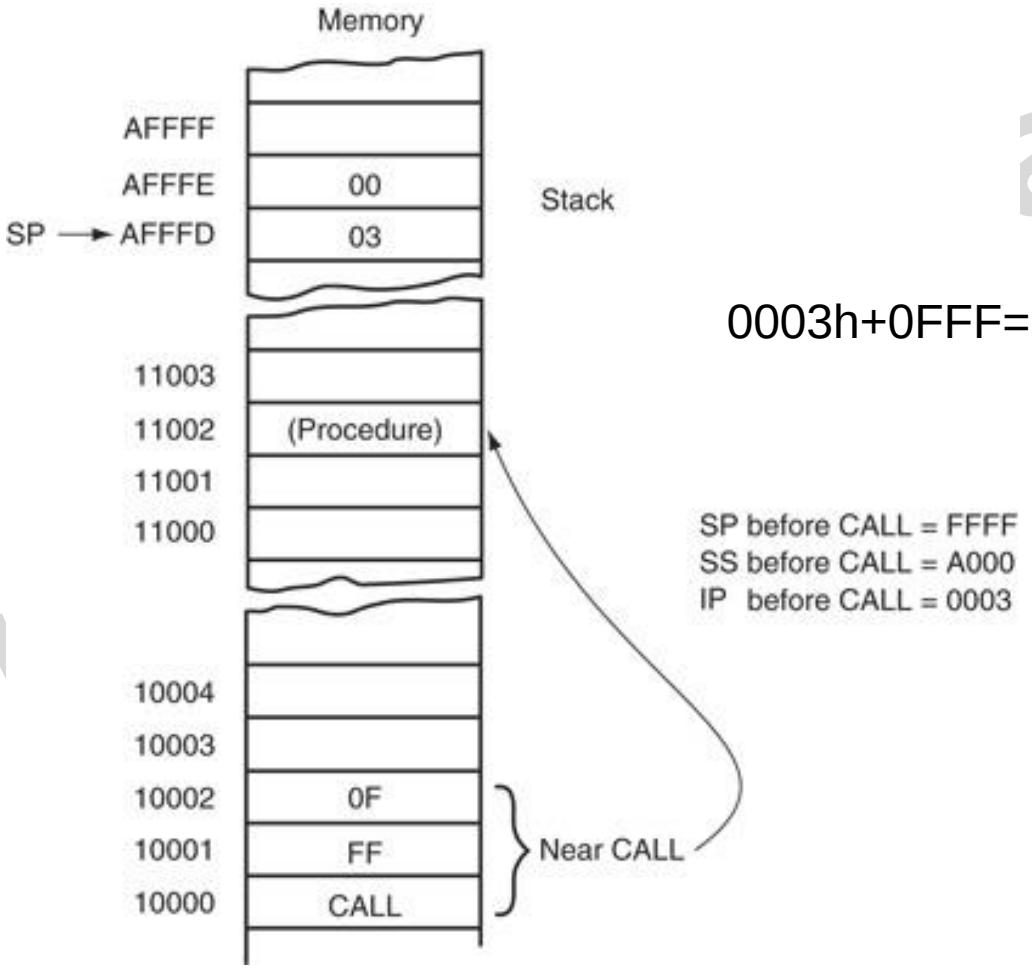
- Transfers the flow of the program to the procedure.
- CALL instruction differs from the jump instruction because a CALL saves a return address on the stack.
- The return address returns control to the instruction that immediately follows the CALL in a program when a RET instruction executes.

Near CALL

- 3 bytes long. E8 disp-low disp-high
  - the first byte contains the opcode; the second and third bytes contain the displacement
- When the near CALL executes, it first pushes the offset address of the next instruction onto the stack.
  - offset address of the next instruction appears in the instruction pointer (IP)
- It then adds displacement from bytes 2 & 3 to the IP to transfer control to the procedure.

CALL = Call:			
Direct Within Segment	11101000	disp-low	disp-high
Indirect Within Segment	11111111	mod 010 r/m	
Direct Intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
	76543210	76543210	76543210
Indirect Intersegment	11111111	mod 011 r/m	

The effect of a near CALL on the stack and the instruction pointer.



0003h+0FFF=1002

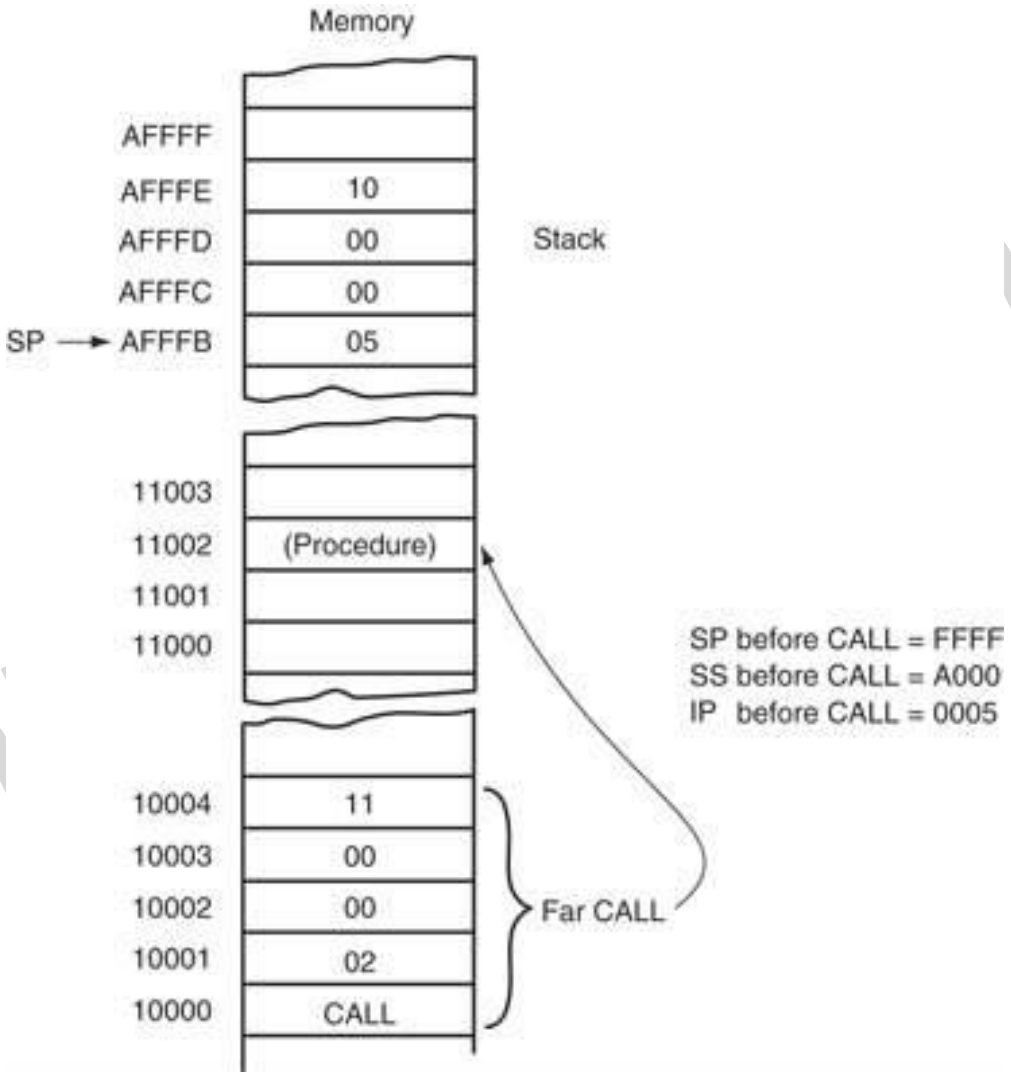
Far CALL

- 5-byte instruction contains an opcode followed by the next value for the IP and CS registers. (9A,___, ____, ____,___)
  - bytes 2 and 3 contain new contents of the IP
  - bytes 4 and 5 contain the new contents for CS
- Far CALL places the contents of both IP and CS on the stack before jumping to the address indicated by bytes 2 through 5.
- This allows far CALL to call a procedure located anywhere in the memory and return from that procedure.

CALL = Call:

Direct Within Segment	11101000	disp-low	disp-high
Indirect Within Segment	11111111	mod 010 r/m	
Direct Intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
	76543210	76543210	76543210
Indirect Intersegment	11111111	mod 011 r/m	

The effect of a far CALL instruction.



CALLs with Register Operands

- An example CALL BX, which pushes the contents of IP onto the stack.
  - then jumps to the offset address, located in register BX, in the current code segment
- Always uses a 16-bit offset address, stored in any 16-bit register except segment registers.

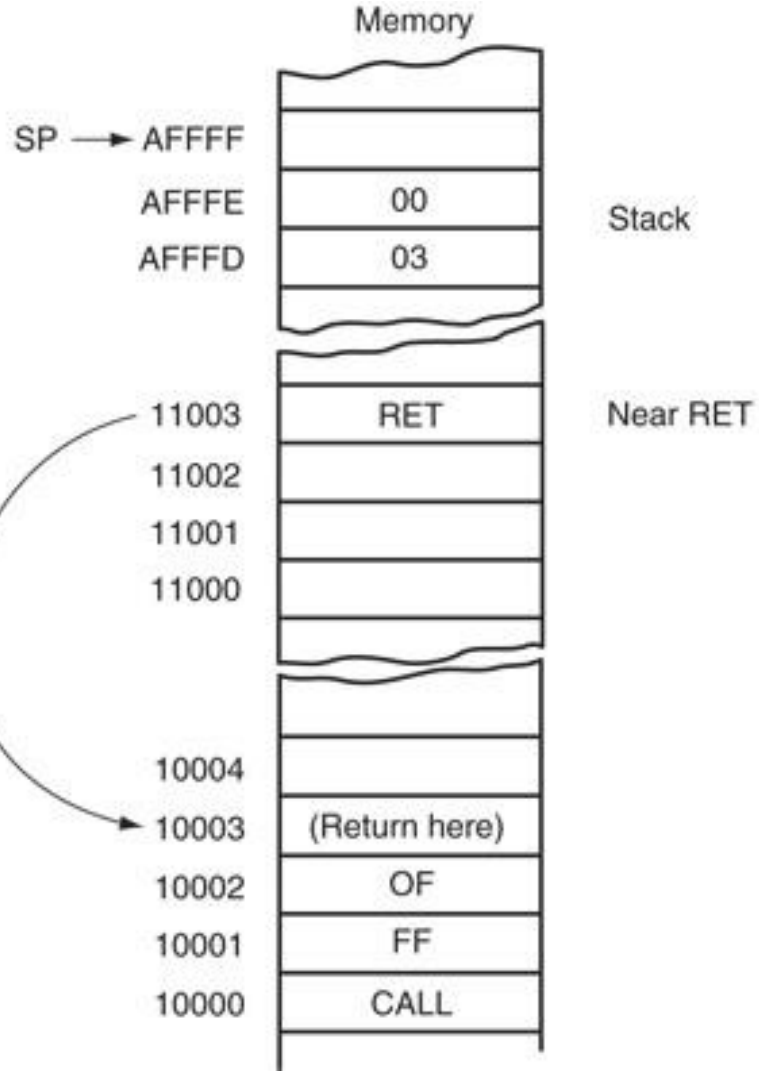
CALLs with Indirect Memory Addresses

- Particularly useful when different subroutines need to be chosen in a program.
  - selection process is often keyed with a number that addresses a CALL address in a lookup table
- Essentially the same as the indirect jump that used a lookup table for a jump address.

RET

- Removes a 16-bit number (**near return**) from the stack placing it in IP,
- or removes a 32-bit number (**far return**) and places it in IP & CS.

The effect of a near return instruction on the stack and instruction pointer.



CALL Stores the address of the instruction after call into stack (return address)

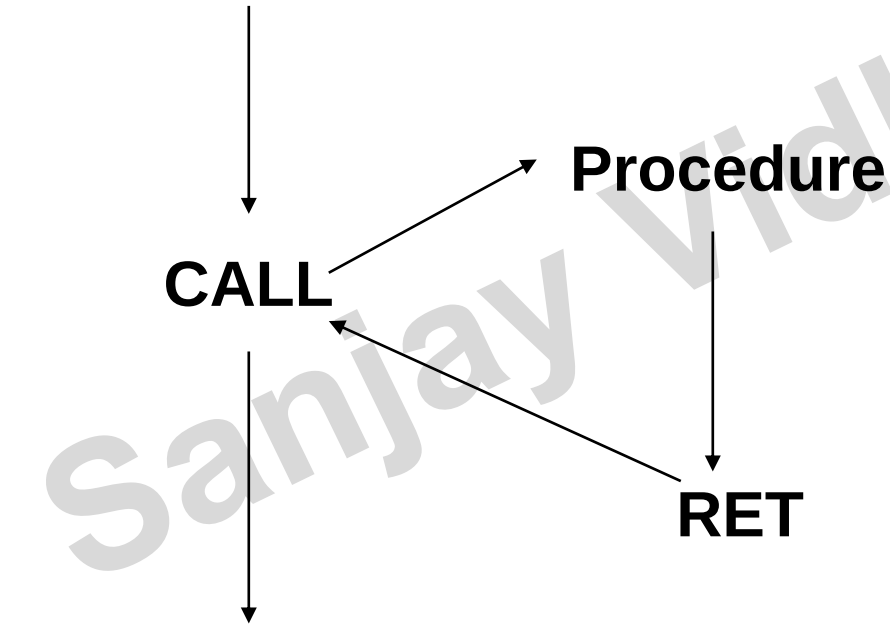
near CALL or far CALL

(IP saved) (CS and IP saved)

RET instruction retrieves the next address after CALL

Back to IP or (IP and CS)

RET instruction at the procedure end sends the execution Back to mail line program



Example:

```
ORG 100h   ; for COM file.

CALL p1

ADD AX, 1

RET          ; return to OS.

p1 PROC      ; procedure declaration.
    MOV AX, 1234h
    RET      ; return to caller.
p1 ENDP
```

Example of a Procedure

```
ORG 100h
MOV AL, 1
MOV BL, 2
CALL m2
CALL m2
CALL m2
CALL m2
RET ; return to operating system.
m2 PROC
MUL BL ; AX = AL * BL.
RET ; return to caller.
m2 ENDP
END
```

MACROS

- Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions.

- Macro is **faster** than procedure because **no CALL and RET**

```
Macro definition:

name    MACRO  [parameters,...]

        <instructions>

ENDM
```

MACROS

Unlike procedures, macros should be defined above the code that uses it, for example:

```
MyMacro    MACRO  p1, p2, p3

    MOV AX, p1
    MOV BX, p2
    MOV CX, p3

ENDM

ORG 100h

MyMacro 1, 2, 3

MyMacro 4, 5, DX

RET
```

```
MOV AX, 00001h
MOV BX, 00002h
MOV CX, 00003h
MOV AX, 00004h
MOV BX, 00005h
MOV CX, DX
```

Miscellaneous Instructions

- CMC ⟶ Complement carry flag (NOT carry flag content)
- CLC ⟶ Clear carry flag
- STC ⟶ Set carry flag

- CLI ⟶ Clear the Interrupt
- STI ⟶ Set the Interrupt

Miscellaneous Instructions

- CWD ⟶ word to double word, AX⟶DX AX
- CWDE ⟶ word to double word extended AX⟶EAX
- CDQ ⟶ double word to quad word EAX⟶EDX EAX

(Note: Instructions without operand, Implicit operand is accumulator)

Miscellaneous Instructions

CMPXCHG

CMPXCHG DST, Source

Compare destination with accumulator,

if equal source will transfer to destination,

if not equal destination will transfer to accumulator

Example CMPXCHG EDX, ECX

Compare EDX with EAX

if EDX=EAX, then ECX content will transfer to EDX (EDX←ECX)

if EDX is not =EAX, then EDX content will transfer to EAX (EAX←EDX)

Thankyou