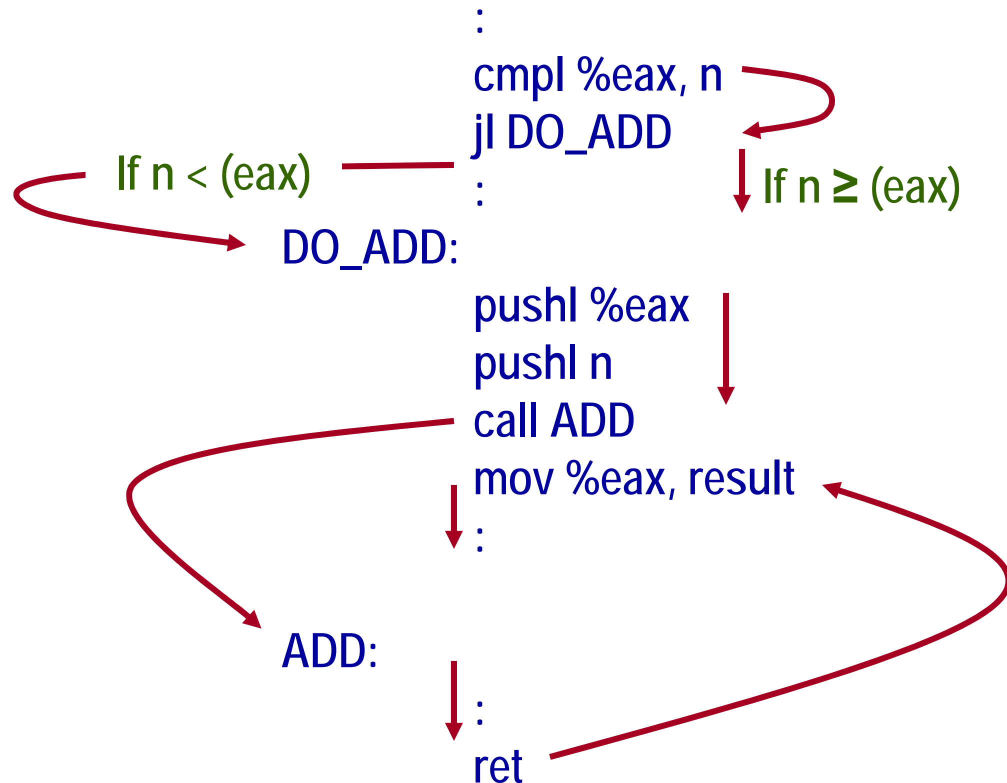
Anirban Lahiri & Prashant Agrawal
Department of Computer Science & Engineering
IIT Kharagpur

Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

Program Control

- Program control means the execution flow in a program



Program Control

- Change in flow can occur
  - Unconditionally
  - Based on some conditions
- Program control instructions
  - Jump group instructions

    E.g. JMP, JL, JGE, etc
  - Call group instructions

    E.g. CALL, RET

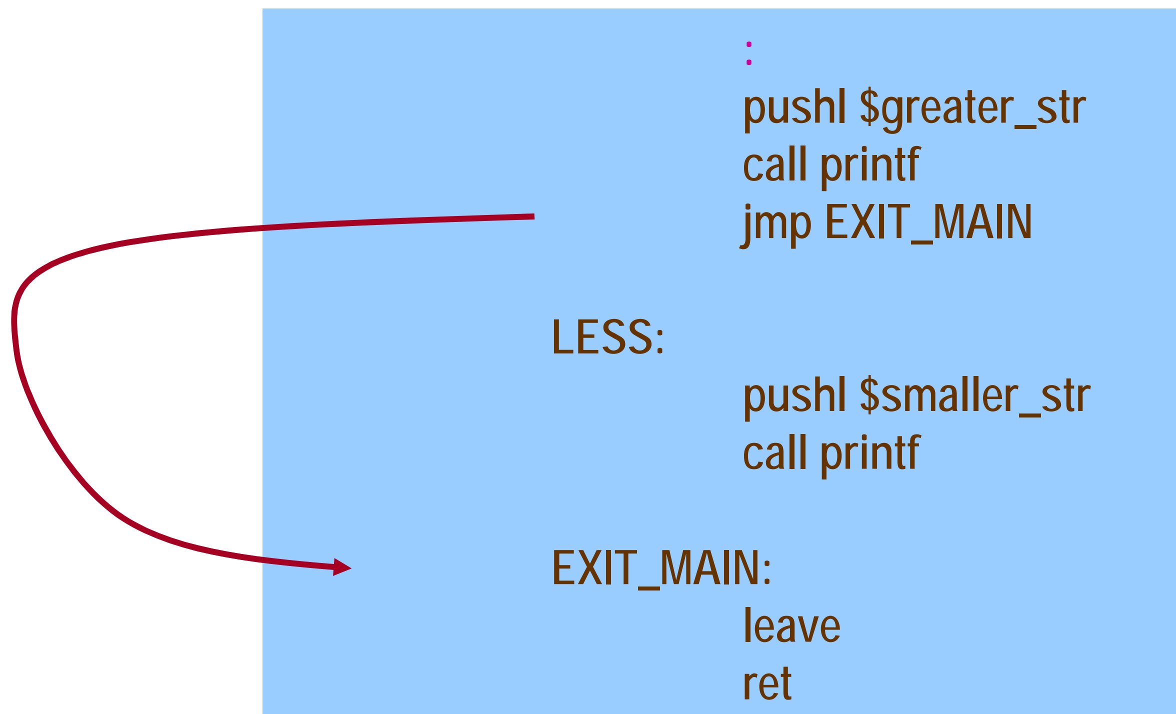
```
        :
        cmpl src, %eax
        jl LESS

        pushl $greater_str
        call printf
        jmp EXIT_MAIN

LESS:
        pushl $smaller_str
        call printf

EXIT_MAIN:
        leave
        ret
```

Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

Jump Instructions

- Jump instructions allow the programmer to skip sections of a program and branch to any part of the memory for the next instruction



Unconditional Jump

- Unconditional jump instructions do not test any condition(s) before transferring the control to the indicated portion of the program
- 3 types of unconditional jump instructions
  - Short jump
  - Near jump
  - Far jump

Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

Short Jump

- Short jump is a 2-byte instruction
  - Opcode & 8-bit displacement

- Jumps to memory location within +127 and -128 bytes from the address following the jump instruction
- Short jumps are also called relative jumps
- The displacement is sign extended and added to EIP to generate the jump address within the code segment

Short Jump

Line	Address	Machine code	Label	Instruction
1				.text
2				.globl main
3				
4			main:	
5	0000	31 DB		xor %ebx, %ebx
6				
7	0002	B8 01000000	START:	movl $1, %eax
8	0007	01 D8		add %ebx, %eax
9	0009	EB 05		jmp NEXT
10	000b	B9 04000000		movl $4, %ecx
11				
12	0010	89 C3	NEXT:	mov %eax, %ebx
13	0012	EB EE		jmp START
14				
15	0014	C3 8D7600		ret

EIP:	00 00 00 0B
Disp(NEXT):	+ 00 00 00 05
Jump Add:	00 00 00 10

For " jmp START"

EIP = EIP – 0x00000012

2's Comp of 0x00000012 is 0xFFFFFFEE

EIP:	00 00 00 14
Disp(START):	+ FF FF FF EE
Jump Add:	00 00 00 02

Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

Near Jump

- Near jump is a 3/5-byte instruction
  - Opcode & 16/32-bit displacement

- Jumps to memory location within ±32K/ ±2G bytes from the address following the jump instruction
- For 80386 and above processors
  - Protected mode
    - Segment is 4GB large
    - Displacement in JMP instruction is 32-bit
    - JMP instruction size is 5-byte
    - Address range is ±2GB

Near Jump

- Real mode
  - Segment is 64K large
  - Displacement is 16-bit
  - JMP instruction size is 3-byte
  - Address range is ±32K

Near Jump

```
1                                       .text
2                                       .globl main
3
4                                       main:
5  0000  31  DB                         xor %ebx, %ebx
6
7  0002  B8 01000000         START:     movl $1, %eax
8  0007  01  D8                         add %ebx, %eax
9  0009  E9 F2010000                    jmp NEXT
10 000e B9 04000000                     movl $4, %ecx
11
12 0013 00000000                        .org 0x0200
12      00000000
12      00000000
12      00000000
12      00000000
13
14 0200  89  C3              NEXT:      mov %eax, %ebx
15 0202  E9 FBFDFFFF                    jmp START
16
17 0207  C3 8D7600                      ret
```

EIP: 00 00 00 0E

Disp(NEXT): + 00 00 01 F2

Jump Add: 00 00 02 00

For " jmp START"

EIP = EIP – 0x00000205

2's Comp of 0x00000205 is 0xFFFFFDFB

EIP: 00 00 02 07

Disp(START): + FF FF FD FB

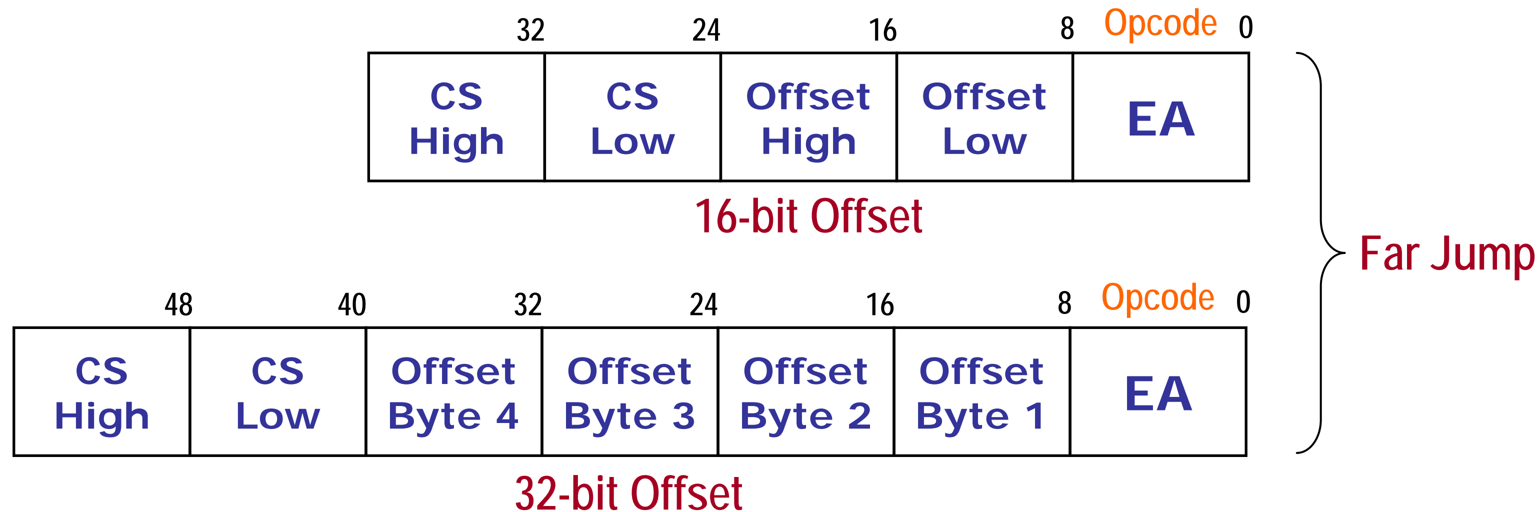
Jump Add: 00 00 00 02

Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

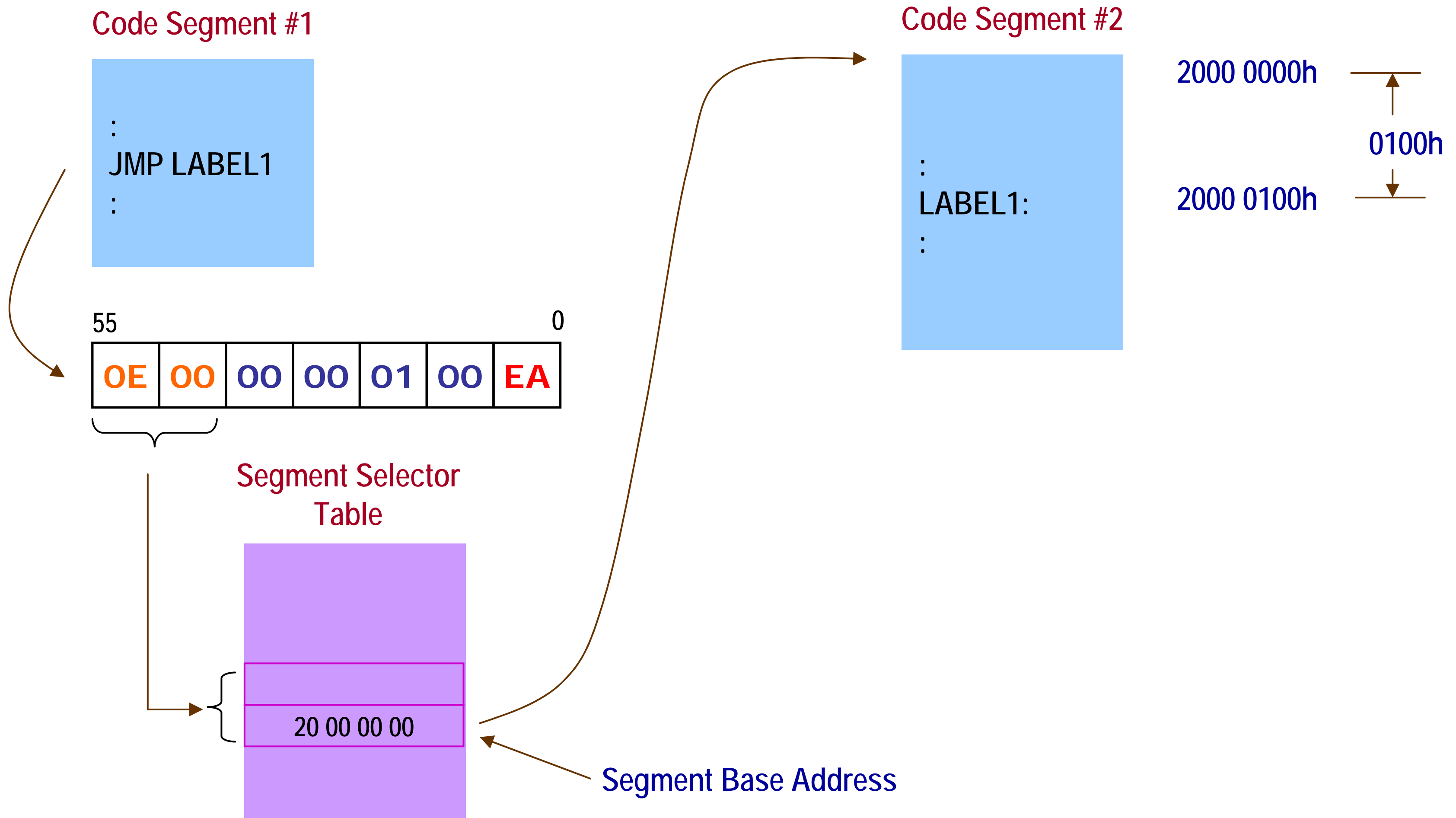
Far Jump

- Far jump is a 5/7-byte instruction



- Far jump allows a jump to any memory location within the real memory system
- It is often called inter-segment jump

Far Jump



Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

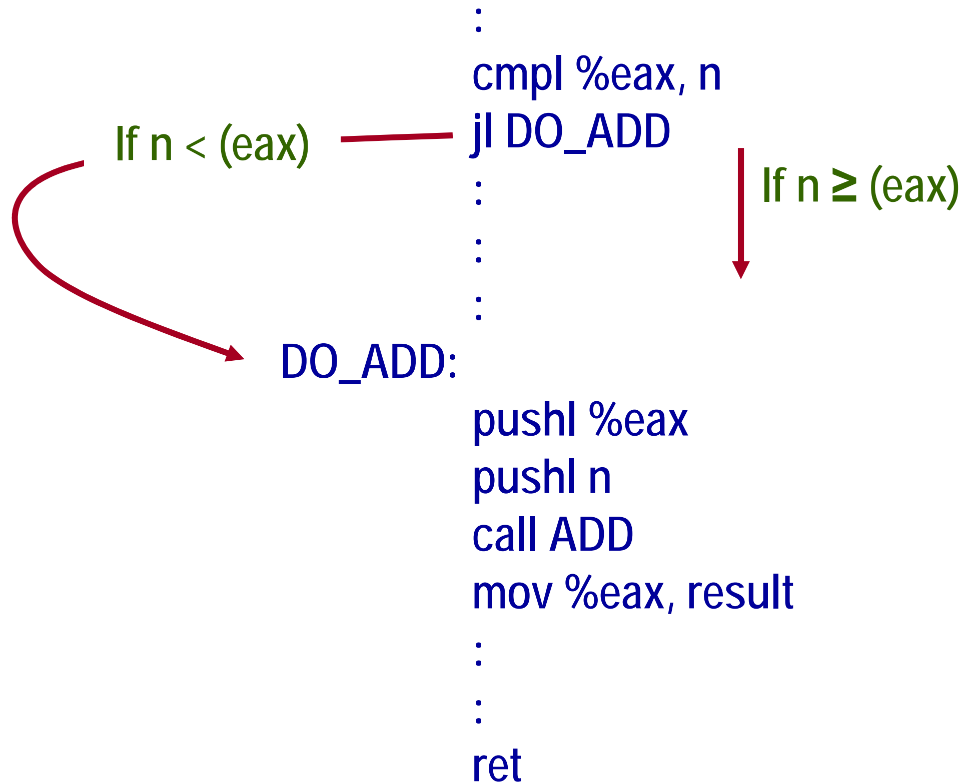
Conditional Jump

- Conditional jump instructions test the following bits:
  - Sign (S), Zero (Z), Carry (C), Parity (P) & Overflow (O)
  - If condition under test is true, a branch to the label associated with the jump instruction occurs
  - If condition is false, the next sequential step in the program executes
- In 8086 through the 80286, conditional jumps are always short jump
  - Range of jump is -128 bytes to +127 bytes

Conditional Jump

- In 80386 and above, conditional jumps are either short or near jumps
  - Range is either -128bytes to +127 bytes or ±2GB
- Different instructions used for signed & unsigned numbers
  - Signed number – JG, JL, JGE, etc
  - Unsigned number – JA, JB, JAE, etc
- JCXZ and JEXCZ are the only instructions that test a register (CX/ECX) instead of flag bit (s)

Conditional Jump



Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

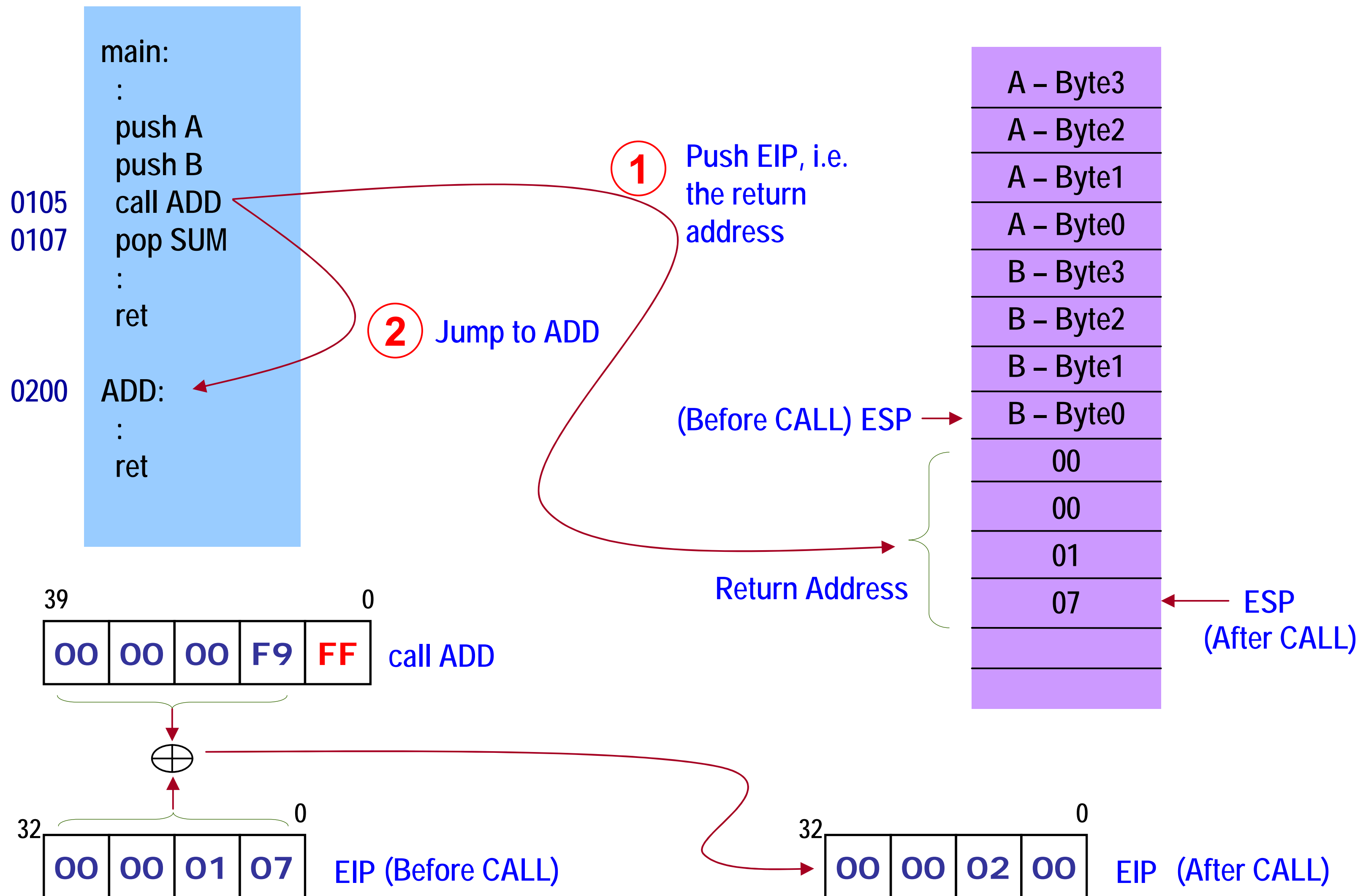
Procedures

- A procedure is an independent unit of code which
  - Usually performs a single task
  - Can be reused
- Advantages
  - Saves memory
  - Eases software development
- Disadvantage
  - Involves overhead in linking to the called procedure and returning from it
- Instructions involved – CALL & RET

CALL Instruction

- Call instruction performs following 2 functions
  - Pushes the address of the next instruction (return address) on the stack
  - Makes an unconditional jump to the called procedure
- Difference with JMP instruction
  - JMP doesn't pushes the return address but CALL does

CALL Instruction



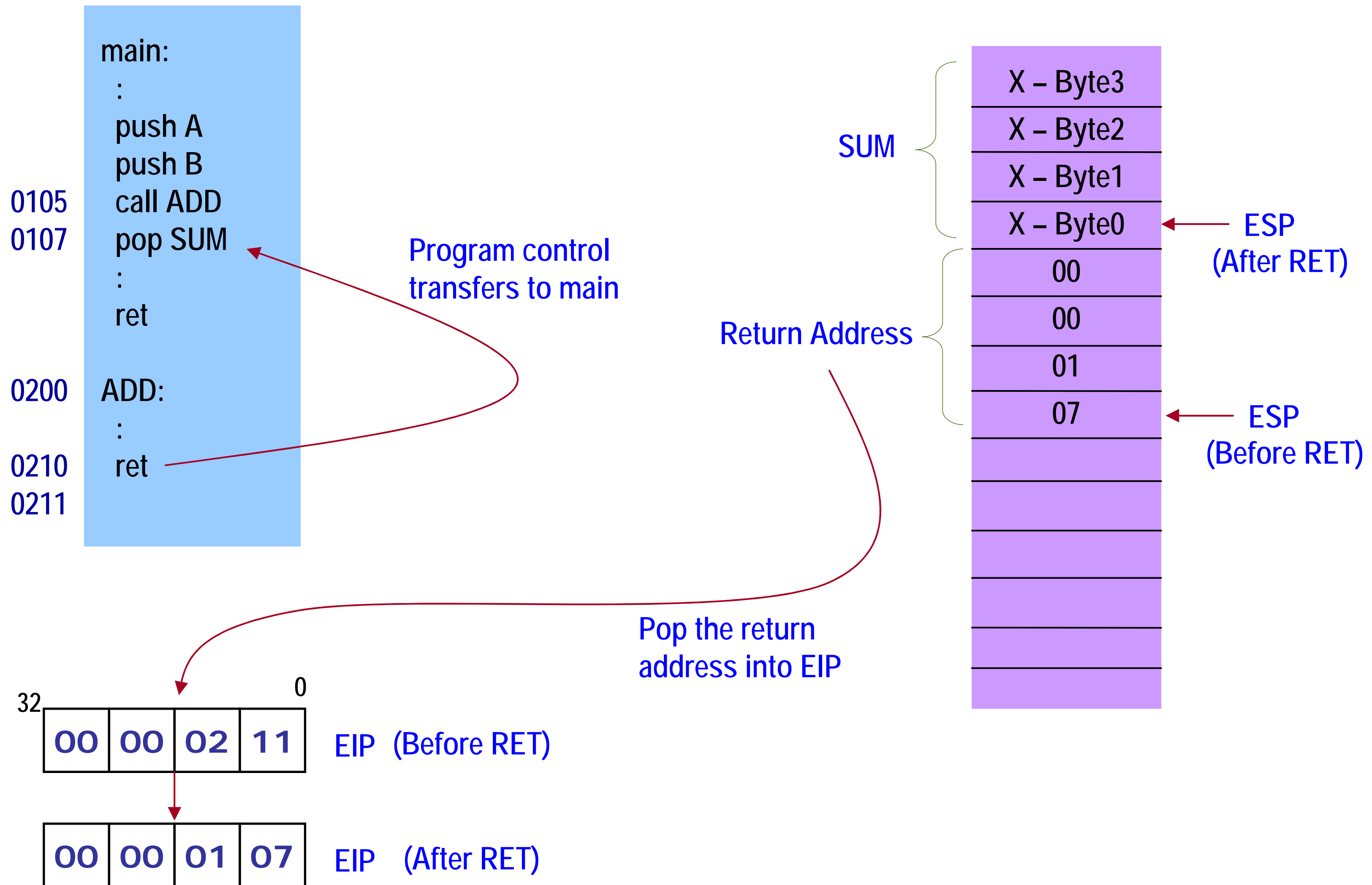
Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

RET Instruction

- RET instruction pops off an address and jumps to that address
  - When using CALL & RET, it is very important to manage the stack correctly so that the right number is popped off by the RET instruction

RET Instruction



Outline

- Program Control
- Jump Instructions
  - Unconditional jump
    - Short jump
    - Near jump
    - Far jump
  - Conditional jump
- Procedures
  - Call instruction
  - Return instruction
  - Parameter Passing

Parameter Passing

- Calling and called procedures must agree on how to pass data between them
- This is known as calling conventions
- The calling conventions can differ from compiler to compiler
- Parameter can be passed by
  - Using stack
  - Without using stack

Parameter Passing Using Stack

- Parameters are pushed onto the stack before the CALL instruction
- In the subprogram, the parameters may or may not be popped off

Parameter Passing Using Stack

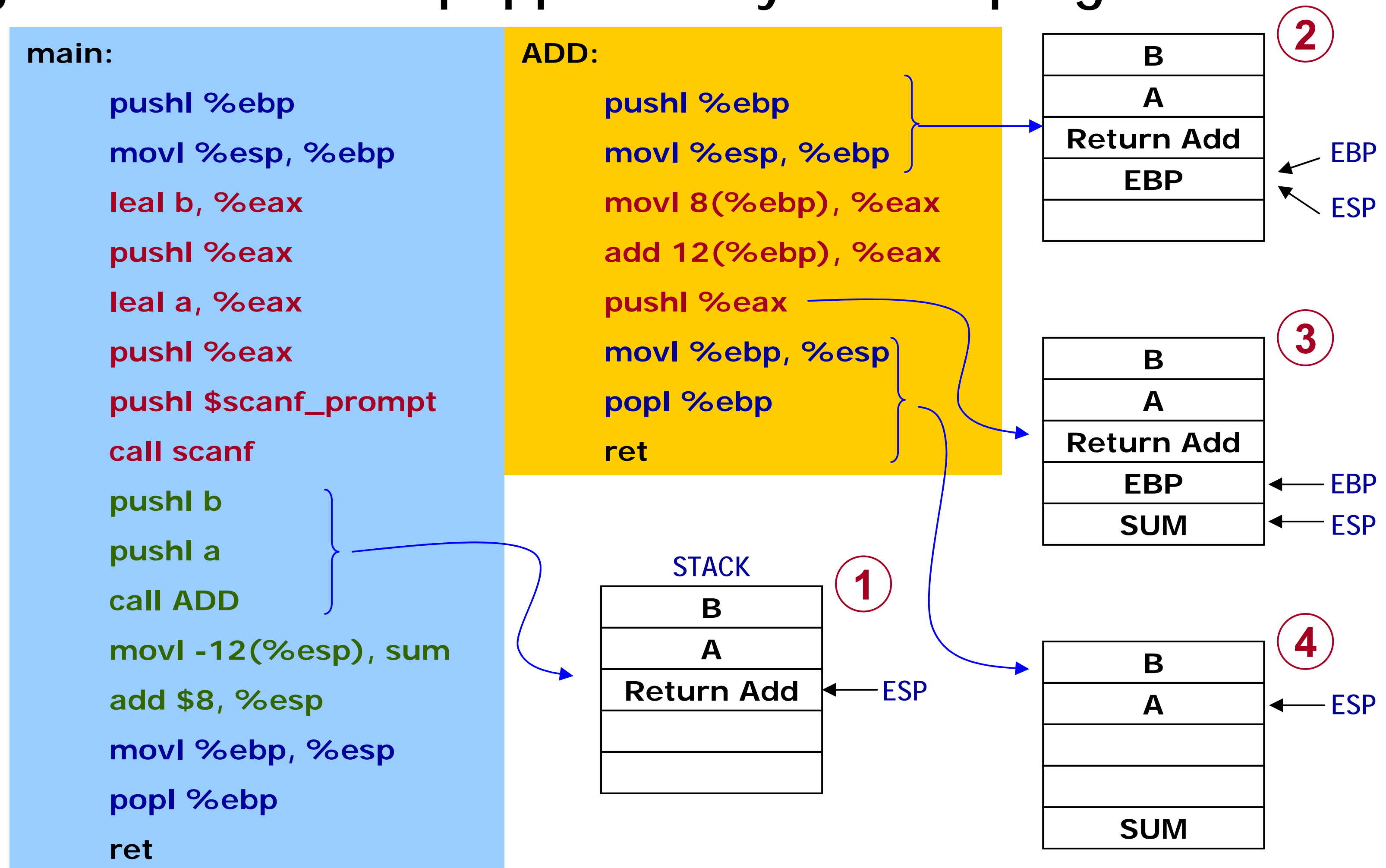
E.g. Parameters popped off by the subprogram

```
main:
        pushl %ebp
        movl %esp, %ebp
        leal b, %eax
        pushl %eax
        leal a, %eax
        pushl %eax
        pushl $scanf_prompt
        call scanf
        pushl b
        pushl a
        call ADD
        pushl $printf_prompt
        call printf
        movl %ebp, %esp
        popl %ebp
        ret
```

```
ADD:
        popl saveret
        popl %eax
        popl %ebx
        add %ebx, %eax
        pushl %eax
        pushl saveret
        ret
```

Parameter Passing Using Stack

E.g. Parameters not popped off by the subprogram



Parameter Passing Without Using Stack

```
main:
        :
        movl %ecx, param1
        movl %edx, param2
        call add
        movl %eax, sum
        :
        ret
```

```
# operand1 in param1
# operand2 in param2
# sum is returned in EAX

ADD:
        movl param1, %eax
        movl param2, %ebx
        add %ebx, %eax
        ret
```

Thanks

Lecture material available at http://10.5.18.66/~pagrawal