

CSE 321a

Computer Organization (1)

تنظيم الحاسبات (1)



3rd year, Computer Engineering
Fall 2017

Lecture #8



Dr. Hazem Ibrahim Shehata

Dept. of Computer & Systems Engineering

Credits to Dr. Ahmed Abdul-Monem Ahmed for the slides

Administrivia

- Assignment #2:
 - Due: **Thursday, Nov. 16, 2017**
- Midterm:
 - New Date: **Sunday, Nov. 19, 2017**
 - New Time: **12:30pm - 1:45pm**
 - Location: **TBA!**
 - Coverage: lecture #1 → lecture #6

Website: <http://hshehata.github.io/courses/zu/cse321a/>

Office hours: Sunday 1:00pm – 2:00pm

Chapter 12. Instruction Sets: Characteristics and Functions (*cont.*)

Outline

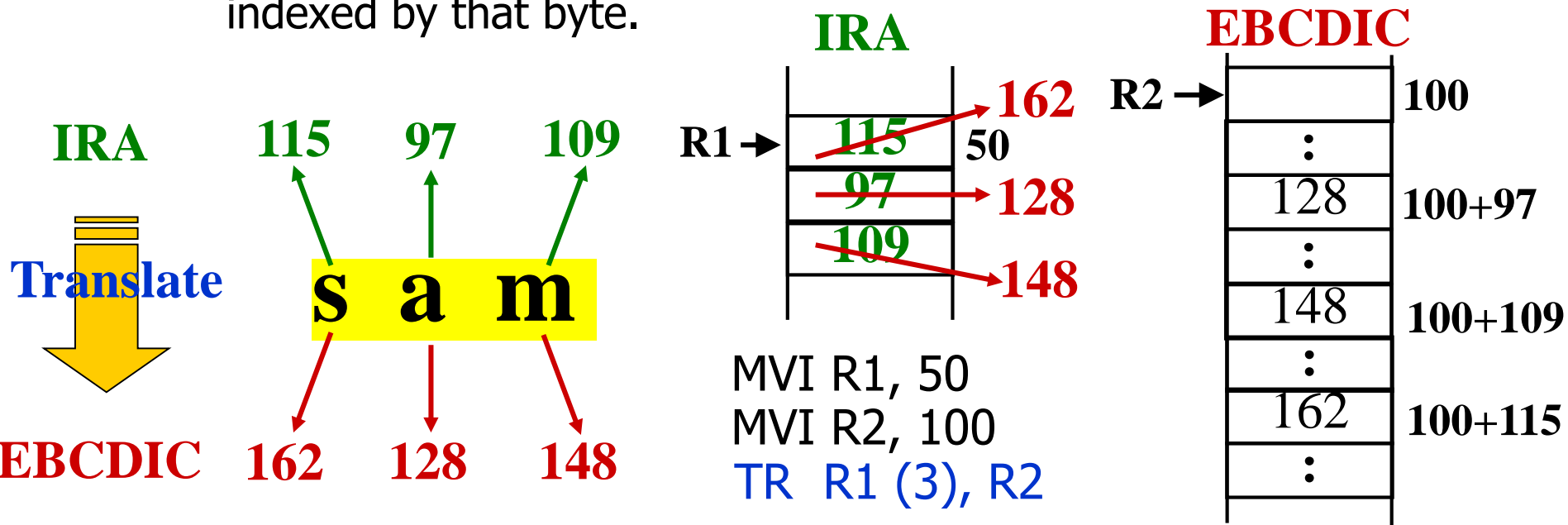
- Machine Instruction Characteristics
 - Elements of machine instructions, instruction representation, instruction types, number of addresses, instruction set design issues.
- Types of Operands
 - Addresses, numbers, characters, logical data
 - Example: x86 data types
- Types of Operations
 - Data transfer, arithmetic, logical, **conversion, input/output, system control, transfer of control.**

4. Conversion

- Instructions that change the format of data.
 - e. g., convert from binary to BCD.
- Example: Translate (TR) instruction in IBM System 390.

TR R1 (L), R2

- R1: starting address of source; L: number of bytes to be translated; R2: starting address of a table of 8-bit codes.
- Each byte of the source bytes is replaced by a table entry indexed by that byte.



5. Input/Output

- May be done using data movement instructions
 - memory-mapped i/o
- May be using specific i/o instructions
 - isolated i/o
- May be done by a separate controller
 - DMA

5. Input/Output – Common Operations

Operation Name	Description
Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
Output (write)	Transfer data from specified source to I/O port or device
Test I/O	Transfer status information from I/O system to specified destination

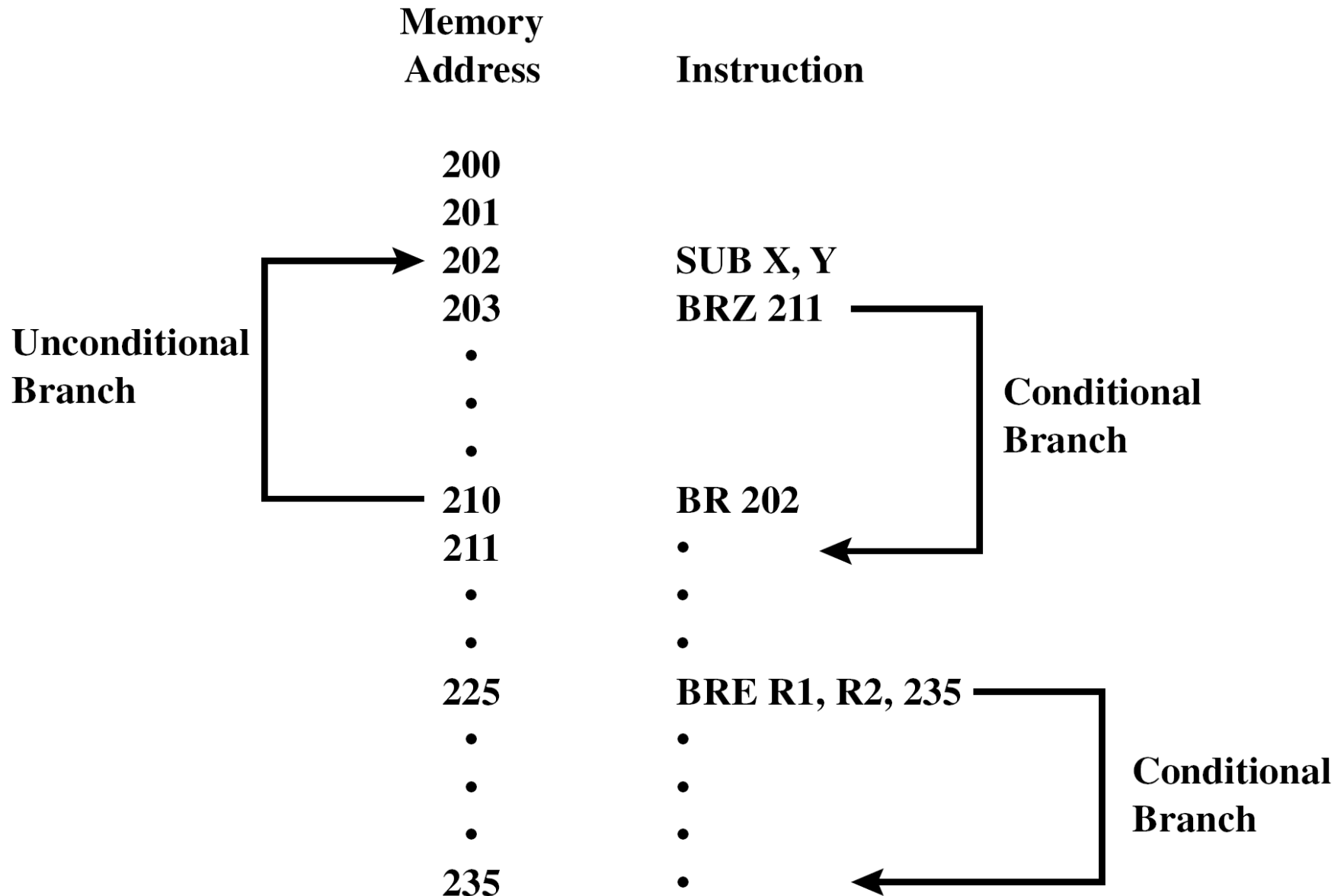
6. System Control

- Privileged instructions.
- Executed only when the processor is in a certain privileged state
 - e.g., ring 0, supervisor mode, kernel mode, ... etc.
- Reserved for operating systems use.
- Example Instructions:
 - Modify storage protection keys.
 - Access process control blocks (ID, address space, priority, ... *etc*).
 - ...

7. Transfer of Control

- Most common transfer-of-control operations:
 - branch, skip, and procedure call/return.
- Branch (jump):
 - Types: unconditional or conditional.
 - Unconditional: always take the branch (update PC to equal address specified in operand).
 - Ex.: BR X → branch to location X
 - Conditional: take the branch if a condition is met, otherwise continue.
 - Condition: test a **flag** set as a result of a recent operation, or perform a **comparison** (need three-address instruction)
 - + Ex.: BRP X → Branch to location X if result is positive.
 - + Ex.: BRE R1, R2, X → Branch to location X if [R1] equals [R2].
 - Branches can be either forward or backward.

7. Transfer of Control – Branch Instructions



7. Transfer of Control

- Skip:
 - Types: unconditional or conditional.
 - Unconditional: always skip following instn. (update PC to equal address of following instn. plus Δ).
 - Conditional: skip following instruction if condition is met, otherwise continue.
 - Typical Example: increment-and-skip-if-zero (ISZ)

+ 301	...	
+	
+ 309	ISZ	R1
+ 310	BR	301
+ 311	...	

7. Transfer of Control

- Procedure call/return:
 - Call: save address of following instruction (*i.e.*, return address) and branch to target procedure.
 - Return address can be saved: in special **register**, at **start** of called procedure, or on top of **stack**.
 - Ex.: Call X \rightarrow call the procedure at location X

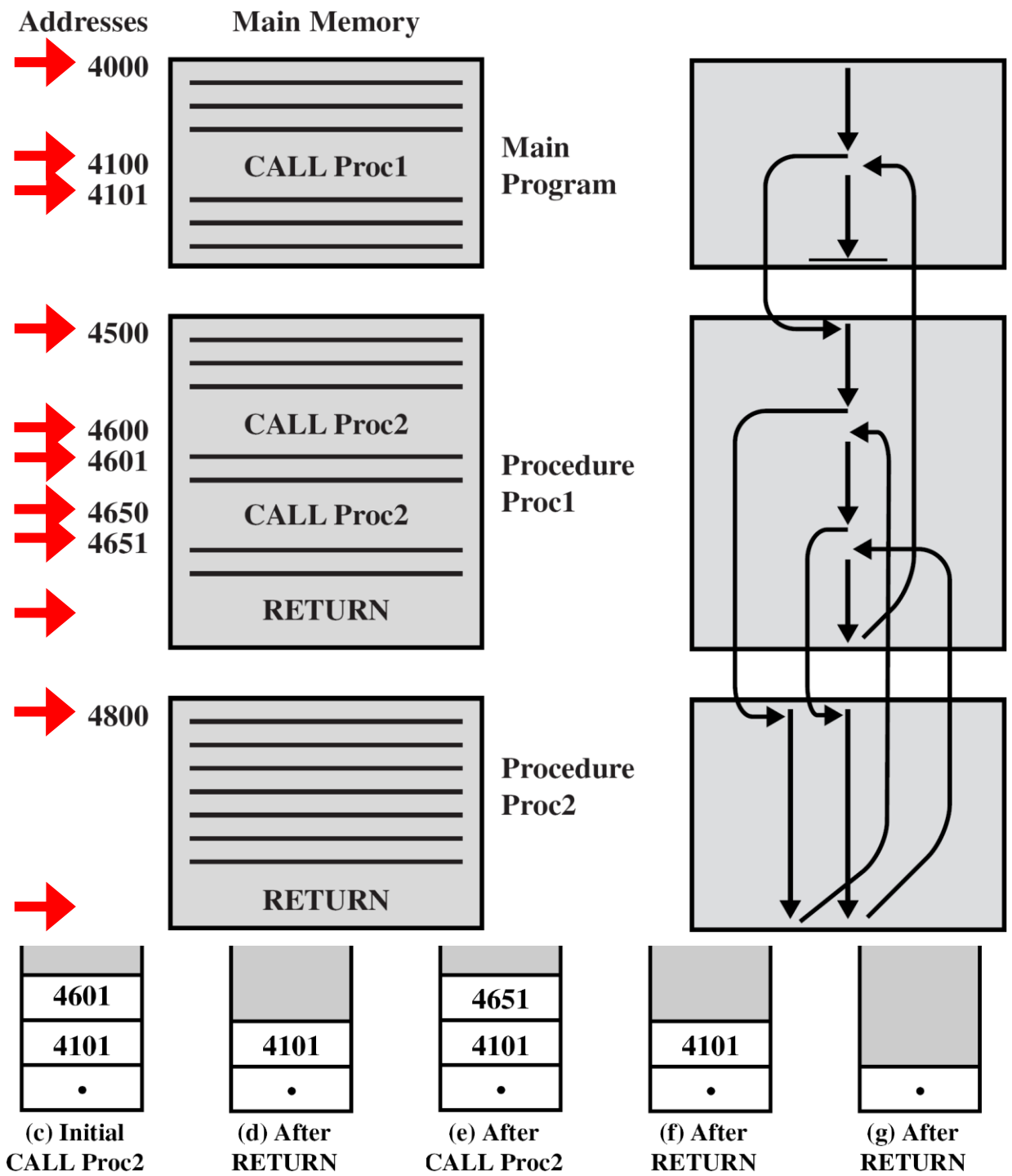
Register	Procedure Start	Stack Top
$RN \leftarrow [PC]$ $PC \leftarrow X$	$X \leftarrow [PC]$ $PC \leftarrow X+1$	Push $[PC]$ to stack $PC \leftarrow X$

- Return: Return to calling procedure.
 - Load PC with the return address
 - Ex.: Ret \rightarrow return to where you came from!

Register	Procedure Start	Stack Top
$PC \leftarrow [RN]$	$PC \leftarrow [X]$	Pop $[PC]$ from stack

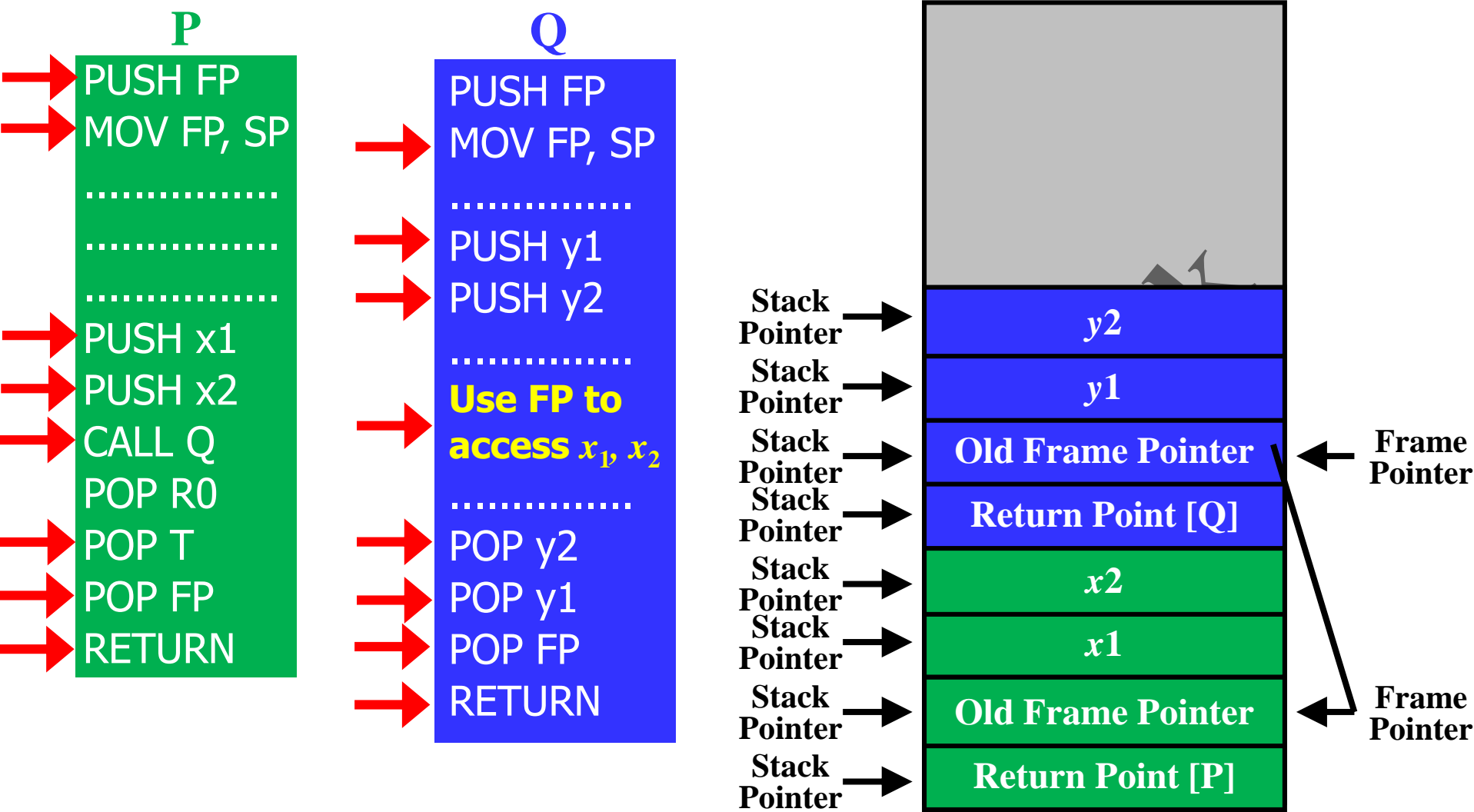
- c.f. interrupt call/return

7. Transfer of Control – Nested Procedure Calls



Stack Frame Growth – Parameter Passing

- **Stack frame**: set of (parameters + local variables + return address) necessary for a procedure call.



7. Transfer of Control – Common Op's

Operation Name	Description
Jump (branch)	Unconditional transfer; load PC with specified address
Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
Jump to Subroutine	Place current program control information in known location; jump to specified address
Return	Replace contents of PC and other register from known location
Execute	Fetch operand from specified location and execute as instruction; do not modify PC
Skip	Increment PC to skip next instruction
Skip Conditional	Test specified condition; either skip or do nothing based on condition
Halt	Stop program execution
Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
No operation	No operation is performed, but program execution is continued

Byte Order

- In what order do we read numbers that occupy more than one byte? e.g. (numbers in hex) 12345678 can be stored in 4x8-bit locations as follows:

Address

Value (1)

Value (2)

184

12

78

185

34

56

186

56

34

187

78

12

Big endian

Little endian

- The problem is called **Endian**.
- Highest significant byte in lowest address:
 - **Big-endian**.
- Least significant byte in lowest address:
 - **Little-endian**.


```
struct{
    int    a;        //0x1112_1314                word
    int    pad;      //
    double b;        //0x2122_2324_2526_2728        doubleword
    char*   c;        //0x3132_3334                word
    char    d[7];     //'A','B','C','D','E','F','G'    byte array
    short   e;        //0x5152                halfword
    int     f;        //0x6162_6364                word
} s;
```

Byte address	11	12	13	14				
00	00	01	02	03	04	05	06	07
08	21	22	23	24	25	26	27	28
	08	09	0A	0B	0C	0D	0E	0F
10	31	32	33	34	'A'	'B'	'C'	'D'
	10	11	12	13	14	15	16	17
18	'E'	'F'	'G'		51	52		
	18	19	1A	1B	1C	1D	1E	1F
20	61	62	63	64				
	20	21	22	23				

				11	12	13	14	Byte address
07	06	05	04	03	02	01	00	
21	22	23	24	25	26	27	28	08
0F	0E	0D	0C	0B	0A	09	08	
'D'	'C'	'B'	'A'	31	32	33	34	10
17	16	15	14	13	12	11	10	
		51	52		'G'	'F'	'E'	18
1F	1E	1D	1C	1B	1A	19	18	
				61	62	63	64	20
				23	22	21	20	

Standard...What Standard?

- X86 and VAX are little-endian.
- IBM System 370/390, Sun SPARC, and most RISC are big-endian.
- **Big-endian is favored for:**
 - Character-string sorting: multiple bytes can be compared in parallel.
 - Decimal/ASCII dumps: values can be printed left to right without causing confusion.
 - Consistent order: integers and character strings are stored in the same order.
- **Little-endian is favored for:**
 - 32-to-16-bit integer conversion: no need to increment address to access least-significant 2 bytes.
 - Higher-precision arithmetic: no need to find the least-significant byte and move backward.

Reading Material

- Stallings, Chapter 12:
 - Pages 425 – 431
 - Pages 447 – 450