Computer Architecture

Course Code: CSE360

Lecture 9

Instruction Sets:

Characteristics and Functions

What is an Instruction Set?

- **The complete collection of different instructions that the CPU can execute is referred to as the CPU's instruction set**
- The instruction sets are represented **by assembly codes**

Elements of an Instruction

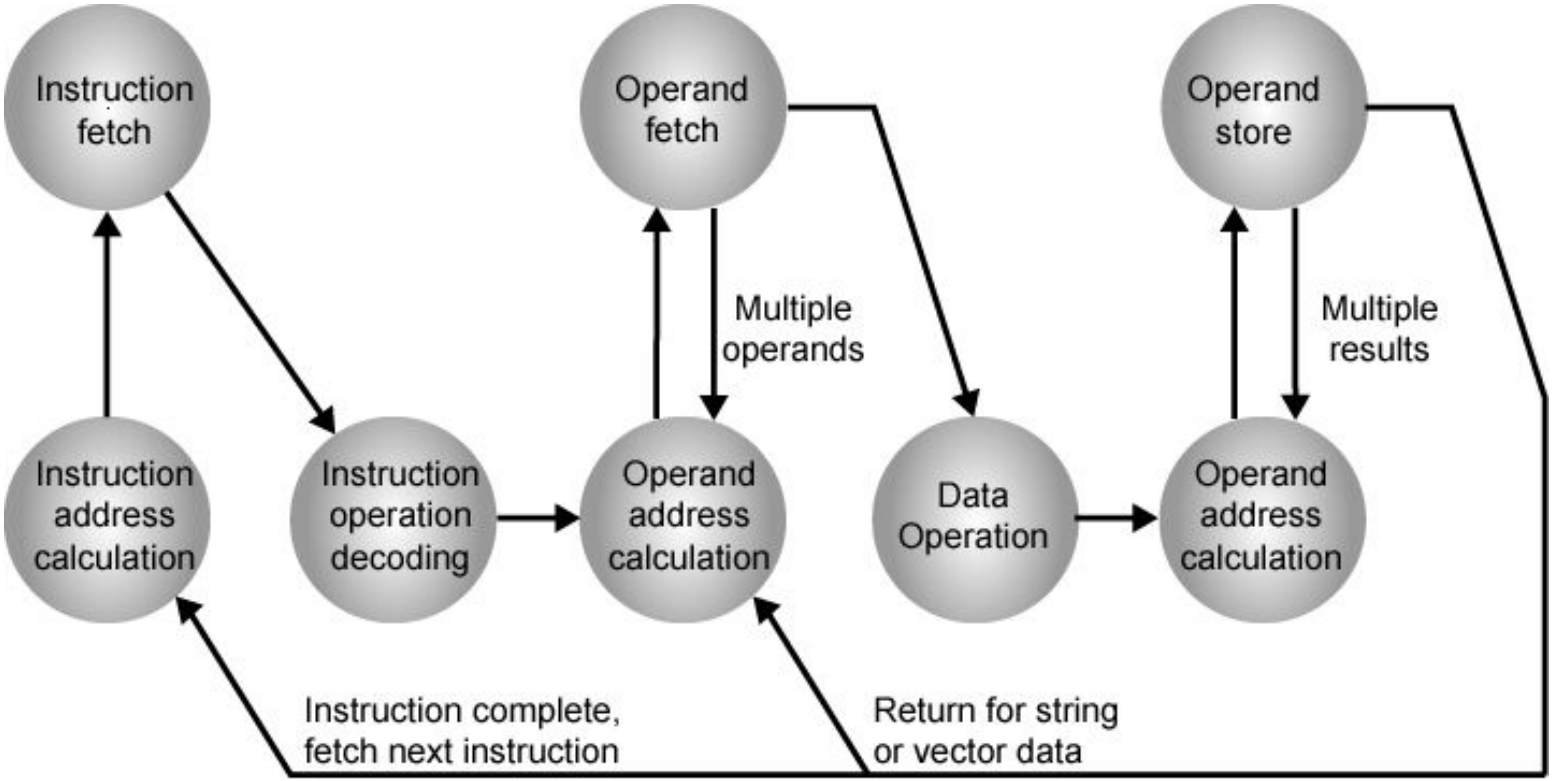
- Operation code (Op code)
  - Specifies the operation to be performed (e.g., ADD, SUB etc.)
- Source Operand reference
  - The operation may involve one or more source operands that is operands that are input for the operation
- Result Operand reference
  - The operation may produce output or result
- Next Instruction Reference
  - This tells the processor where to fetch the next instruction after the execution of this instruction is complete

Where have all the Operands Gone?

Source and result operands can be in one of three areas:

- Main memory (or virtual memory) – As with next instruction references, main or virtual memory address must be supplied
- CPU register- A processor may contain one or more registers that may be referenced by machine instructions
- I/O device- The instruction must specify the I/O module and device for the operation.

Instruction Cycle State Diagram



- *Opcodes specify arithmetic and logic operation, movement of data between two registers, register and memory, or two memory locations (ADD, SUB, MPY, DIV, LOAD, STOR)*
- *Operand references specify a register or memory location of operand data*

Instruction Cycle State

- **Instruction address calculation:** Determine the address of the next instruction to be executed.
- **Instruction fetch:** Read instruction from its memory location into the processor.
- **Instruction operation decoding:** Analyze instruction to determine type of operation to be performed and operand to be used
- **Operand address calculation:** If the operation involves reference to an operand in memory or via I/O, then determine the address of the operand.
- **Operand fetch:** Fetch the operand from memory or read it in from I/O.
- **Data operation:** Performs the operation indicated in the instruction
- **Data store:** Write the result into memory or out to I/O.

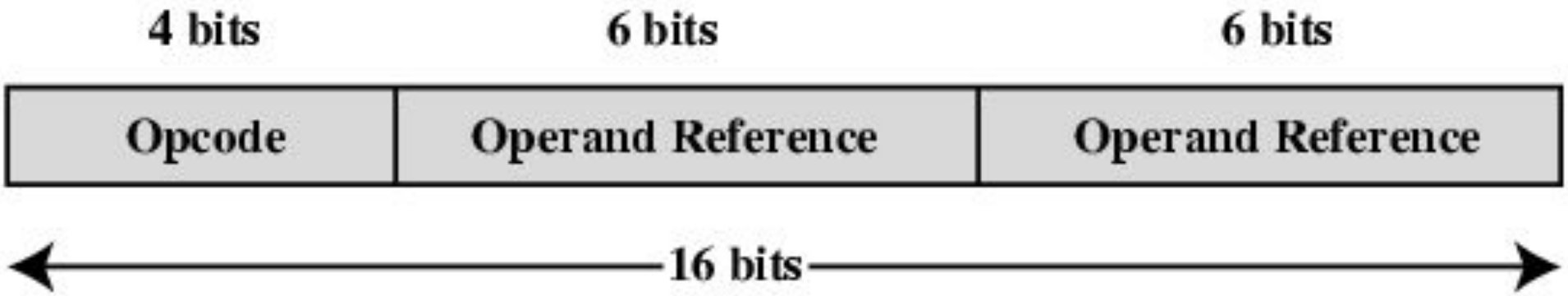
Instruction Representation

- In machine code each instruction has a unique bit pattern
- Opcodes indicate the following operation:
  - e.g. ADD, SUB, MPY, DIV, LOAD, STOR
- Operands can also be represented in this way
  - ADD R,Y

  (Add the value contained in data location Y with the content of register R; in this example, Y refers to the address of a location in memory and R refers to a particular register).

  Note that: The operation is performed on the contents of a location; not on its address

Simple Instruction Format



ADD AX, BX
0110 101101 111001

Instruction Types

- **Data processing**: Arithmetic and logical operation
- **Data storage (main memory):** Memory instructions
- **Data movement (I/O):** I/O instructions
- **Program flow control:** Test the value of a data word and status of a computation and branch instructions (Branch to a different set of instruction depending on the decision made)

Number of Addresses (a)

- An instruction may contain 4 addresses
- 3 addresses:
  - Operand 1, Operand 2, Result (destination operand)
  - a (result) = b (operand 1) + c (operand 2);
  - Fourth address may be used for next instruction

Number of Addresses (b)

- 2 addresses
  - One address doubles as operand and result
  - a = a + b
  - Reduces length of instruction
  - Requires some extra work
    - Temporary storage to hold some results

Number of Addresses (c)

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines

Number of Addresses (d)

- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push a
  - push b
  - add
  - pop c

  - c = a + b

Number of Addresses - summary

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T – 1) OP T

AC = accumulator
T = top of stack
(T – 1) = second element of stack
A, B, C = memory or register locations

Number of Addresses - Execution

Instruction		Comment
SUB	Y, A, B	Y ← A – B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y – B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC – B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) One-address instructions

Figure 10.3 Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

Common Instruction Set Operations

Type	Operation Name	Description
Data Transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

Common Instruction Set Operations

Type	Operation Name	Description
Transfer of Control	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
Input/Output	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
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Common Instruction Set Operations

Data Transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write
Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes and flags
Logical	Same as arithmetic
Conversion	Similar to arithmetic and logical. May involve special logic to perform conversion
Transfer of Control	Update program counter. For subroutine call/return, manage parameter passing and linkage
I/O	Issue command to I/O module
	If memory-mapped I/O, determine memory-mapped address

Arithmetic

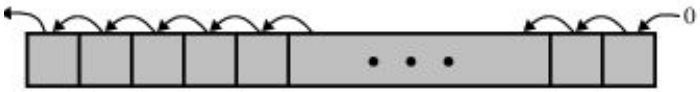
- Basic arithmetic operations:

Add, Subtract, Multiply, and Divide

- These are invariably provided for Signed Integer and often they are also provided for floating point.
- May include
  - Increment (a++) [Add 1 to the operand]
  - Decrement (a--) [Subtract 1 from the operand]
  - Negate (-a)

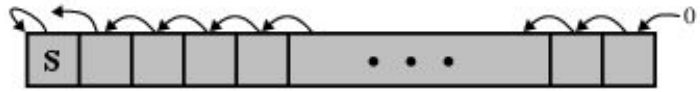
Shift and Rotate Operations

(a) Logical right shift

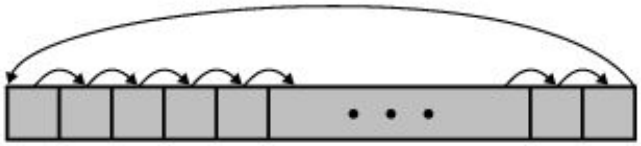


(b) Logical left shift

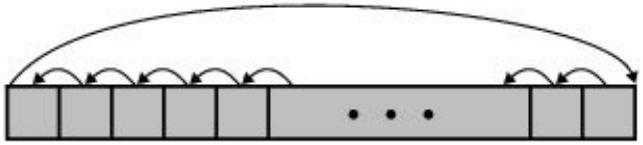
(c) Arithmetic right shift



(d) Arithmetic left shift



(e) Right rotate



(f) Left rotate

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

Logical

- Bitwise operations
- AND, OR, NOT

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P=Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Conversion

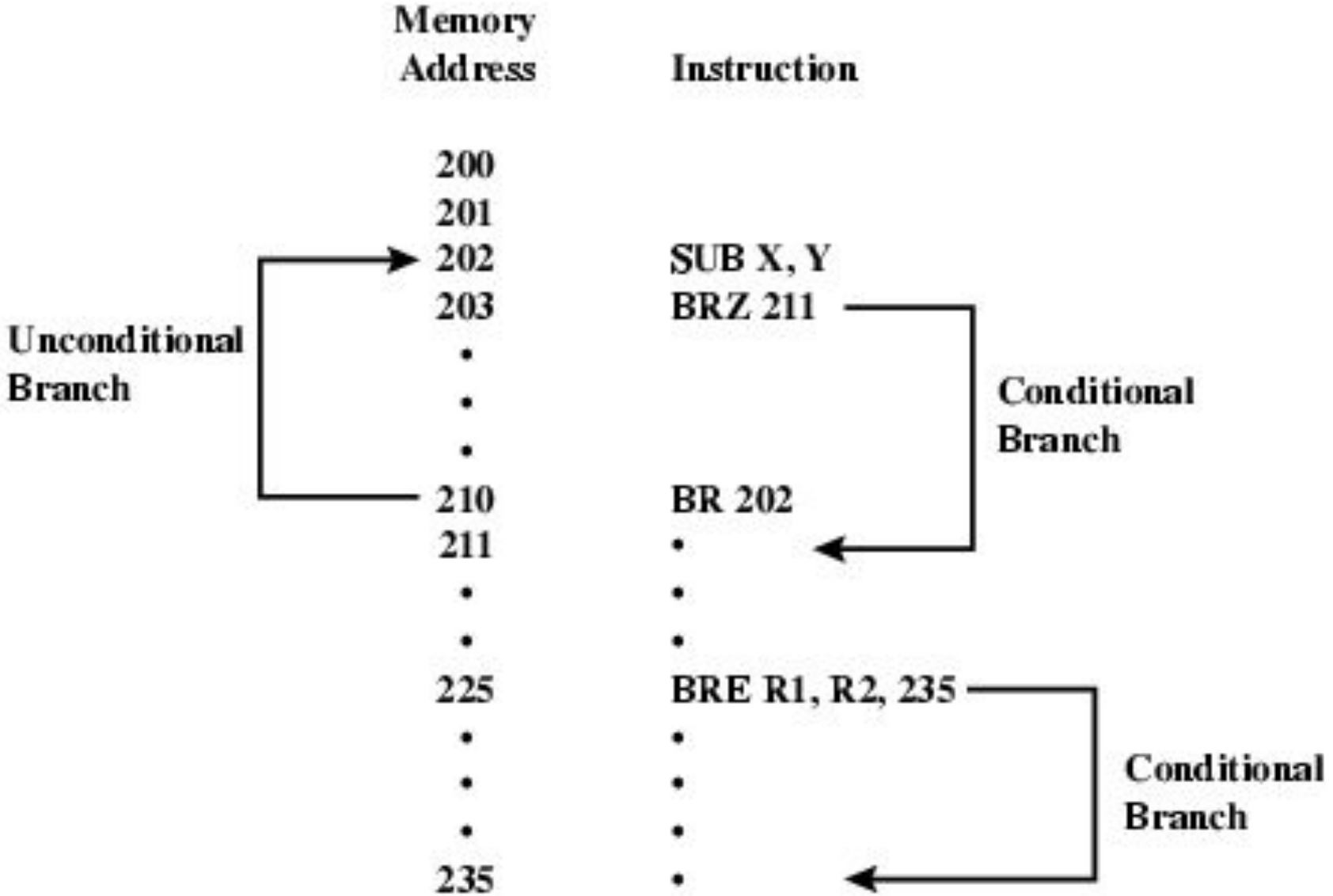
- E.g. Binary to Decimal

Transfer of Control

- Branch/Jump instruction
- Conditional Branch
  - BRP X (Branch to location X if result is positive)
  - BRN X (Branch to location X if result is Negative)
  - BRZ X (Branch to location X if result is Zero)
  - BRO X (Branch to location X if result is Overflow)
  - BRE R1, R2,X (Branch to X if contents of R1=contents of R2)

- Unconditional branch

Transfer of Control: Branch Instruction

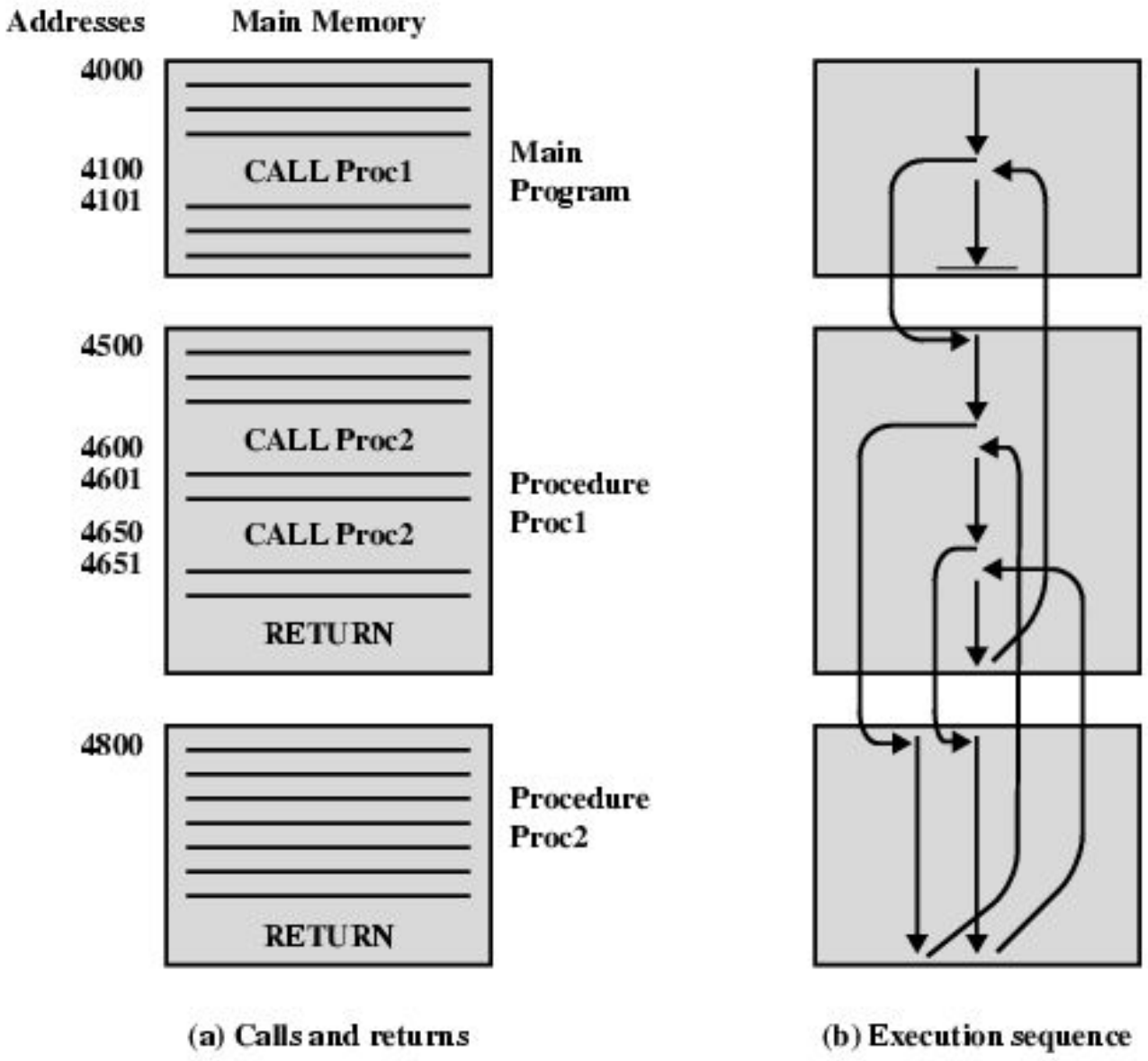


Transfer of Control

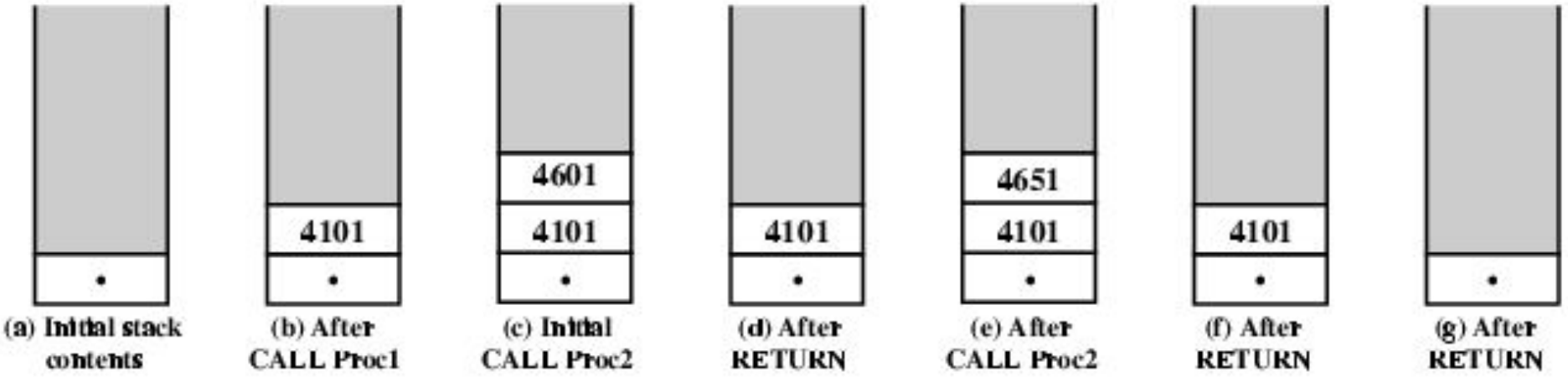
- Skip Instructions
  - ISZ - increment and skip if zero (like *continue* in C++)
  - R1 is set with negative number. R1 is incremented as the number of iteration performed. If it is not zero program branches back to zero, otherwise branch is skipped and program continues to next instruction after end of loop.

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

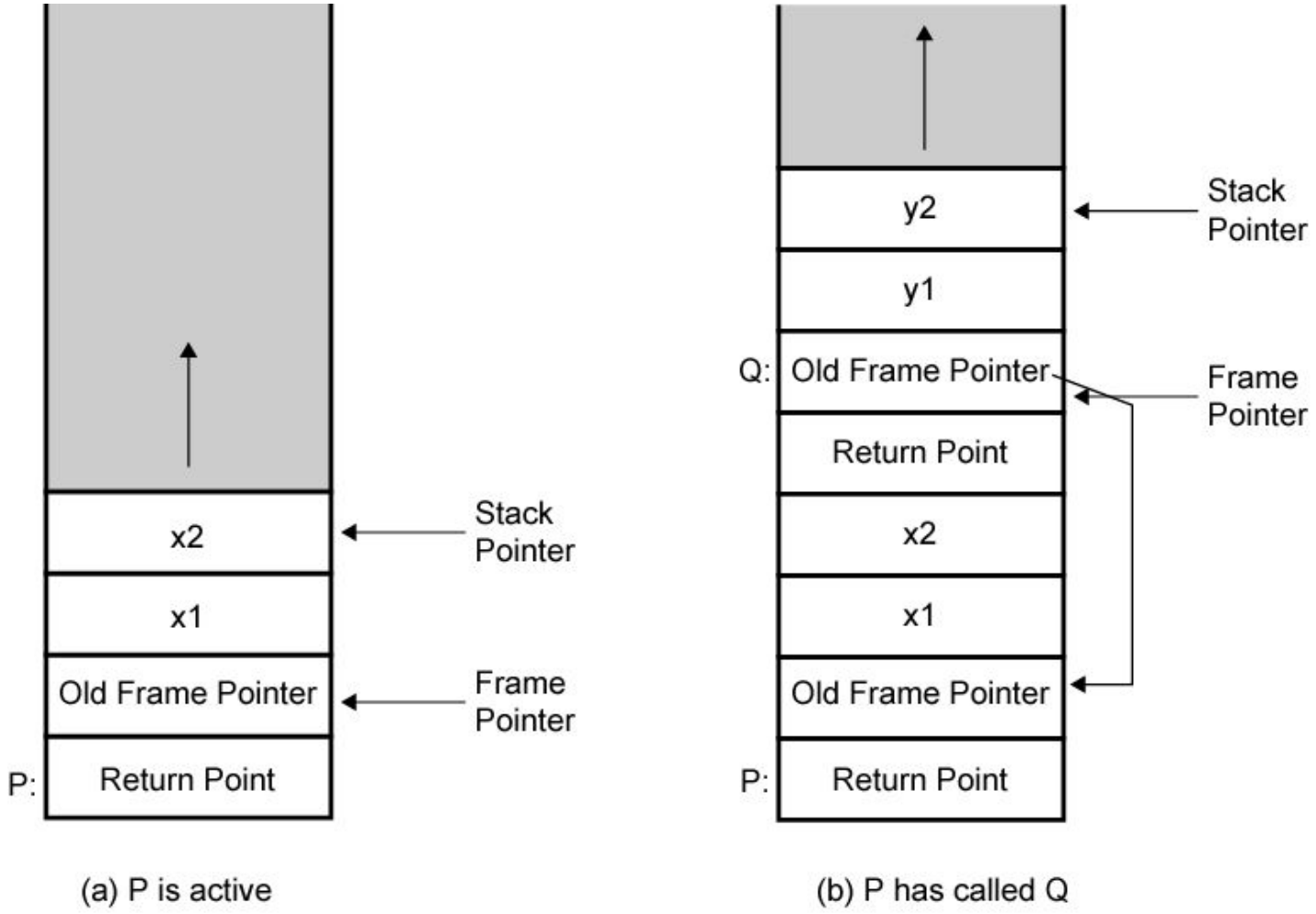
Nested Procedure Calls



Use of Stack


4601
4101
4101
4651
4101
4101
4101
(a) Initial stack contents
(b) After CALL Proc1
(c) Initial CALL Proc2
(d) After RETURN
(e) After CALL Proc2
(f) After RETURN
(g) After RETURN

Stack Frame Growth Using Sample Procedures P and Q



- When the processor executes a call, it not only stacks the return address, it stacks the parameters to be passed to the called address. The called procedure can access the parameters from stack. Upon return, return parameters can also be placed on the stack. The entire set is called stack frame.

Use of Stack for computation

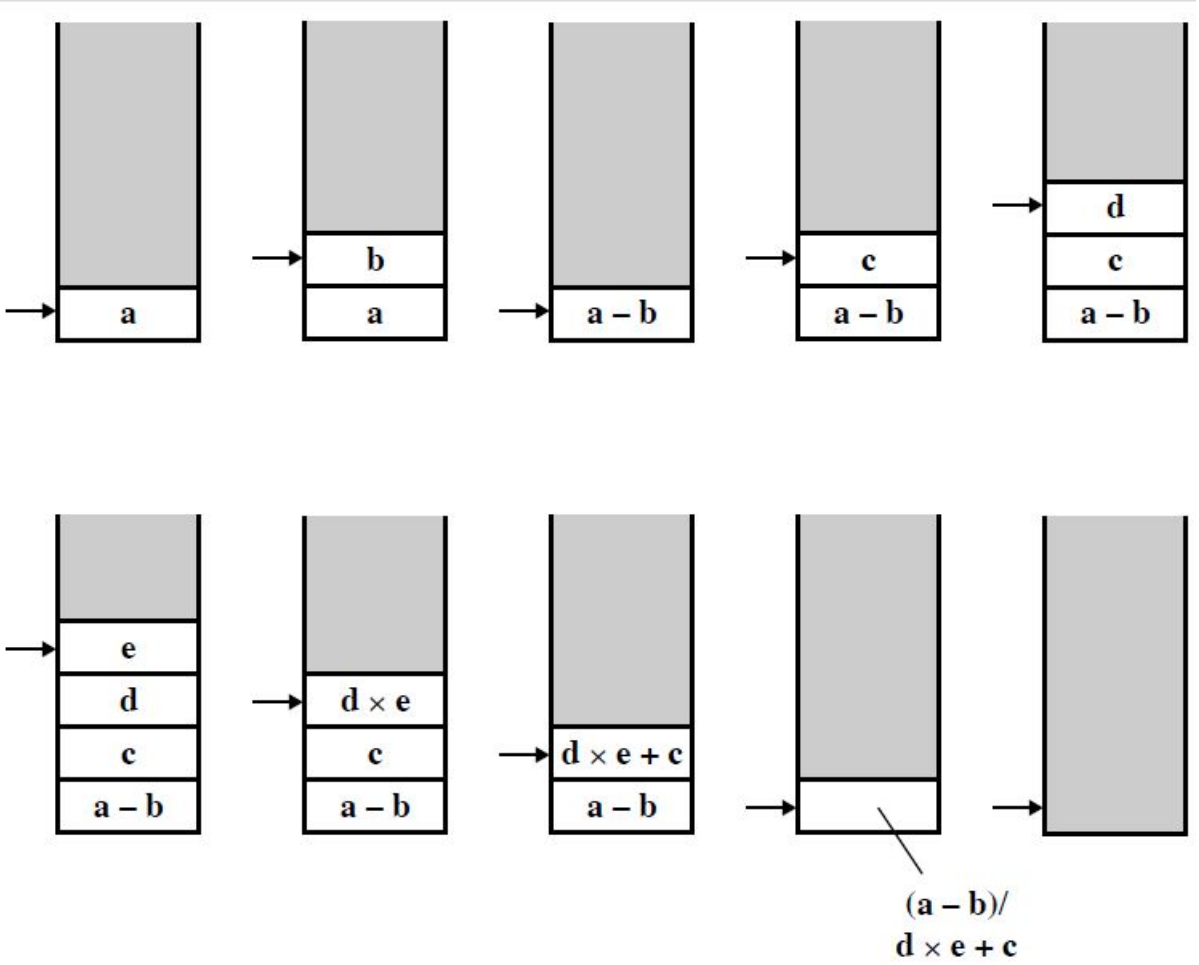


Figure 10.16 Use of Stack to Compute $f = \dfrac{a - b}{c + (d \times e)}$

Comparison of three programs

	Stack	General Registers	Single Register
	Push a Push b Subtract Push c Push d Push e Multiply Add Divide Pop f	Load R1, a Subtract R1, b Load R2, d Multiply R2, e Add R2, c Divide R1, R2 Store R1, f	Load d Multiply e Add c Store f Load a Subtract b Divide f Store f
Number of instructions	10	7	8
Memory access	10 op + 6 d	7 op + 6 d	8 op + 8 d

Figure 10.15 Comparison of Three Programs to Calculate $f = \dfrac{a-b}{c+(d \times e)}$

Expression Evaluation

- Mathematical formulas are expressed is known as infix notation
- a+(b x c) will yield different result than (a+b)xc
- To minimize the use of parentheses, operations have precedence
- Therefore, multiplication takes precedence over addition, so, a+ b x c is equivalent to a+(b x c)
- An alternative technique is known as reverse polish or postfix notation.
- a+b becomes a b+
- a + (bxc) becomes abcx+
- (a+b) x c becomes ab+cx

Infix to Postfix Notation

Input	Output	Stack (top on right)
A + B × C + (D + E) × F	empty	empty
+ B × C + (D + E) × F	A	empty
B × C + (D + E) × F	A	+
× C + (D + E) × F	A B	+
C + (D + E) × F	A B	+ ×
+ (D + E) × F	A B C	+ ×
(D + E) × F	A B C × +	+
D + E) × F	A B C × +	+ (
+ E) × F	A B C × + D	+ (
E) × F	A B C × + D	+ (+
) × F	A B C × + D E	+ (+
× F	A B C × + D E +	+
F	A B C × + D E +	+ ×
empty	A B C × + D E + F	+ ×
empty	A B C × + D E + F × +	empty

Figure 10.17 Conversion of an Expression from Infix to Postfix Notation