Linguaggi Macchina: caratteristiche e funzioni

Corso di Architettura degli Elaboratori (teoria)

Dott. Francesco De Angelis
francesco.deangelis@unicam.it

Scuola di Scienze e Tecnologie - Sezione di Informatica

Architettura degli Elaboratori e Laboratorio

William Stallings
Computer Organization and Architecture
8th Edition

Chapter 10

Instruction Sets: Characteristics and Functions

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes

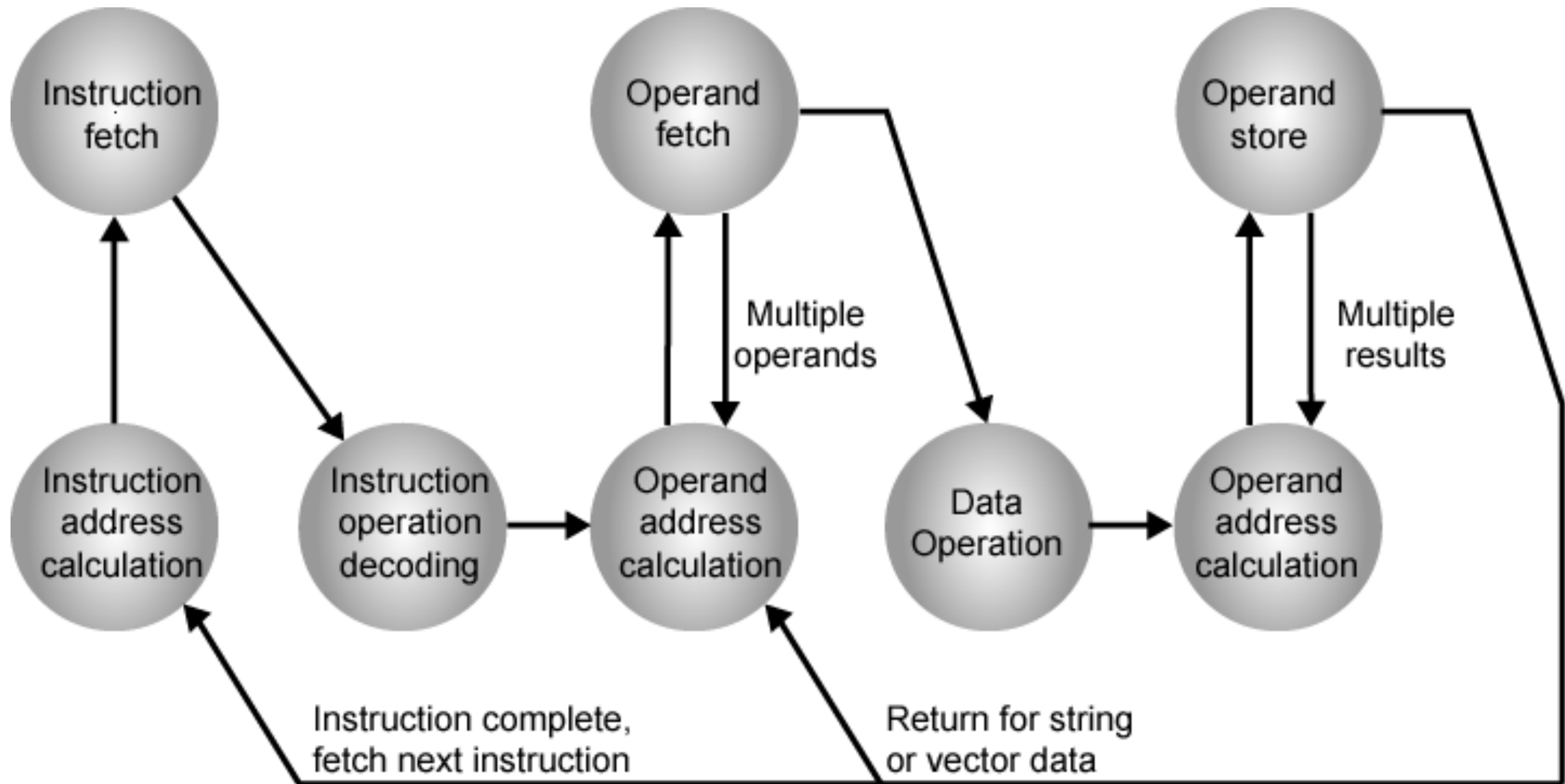
Elements of an Instruction

- Operation code (Op code)
  - Do this
- Source Operand reference
  - To this
- Result Operand reference
  - Put the answer here
- Next Instruction Reference
  - When you have done that, do this...

Where have all the Operands Gone?

- Long time passing….
- (If you don't understand, you're too young!)
- Main memory (or virtual memory or cache)
- CPU register
- I/O device

Instruction Cycle State Diagram

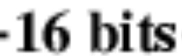


Instruction Representation

- In machine code each instruction has a unique bit pattern
- For human consumption (well, programmers anyway) a symbolic representation is used
  - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
  - ADD A,B

Simple Instruction Format

4 bits	6 bits	6 bits
Opcode	Operand Reference	Operand Reference



Instruction Types

- Data processing
- Data storage (main memory)
- Data movement (I/O)
- Program flow control

Number of Addresses (a)

- 3 addresses
  - Operand 1, Operand 2, Result
  - a = b + c;
  - May be a forth - next instruction (usually implicit by the PC register)
  - Not common
  - Needs very long words to hold everything

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Programs to Execute $\mathbf{Y = \frac{A - B}{C + (D \times E)}}$

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

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

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

Number of Addresses (c)

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

Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) One-address instructions

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

Number of Addresses

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) One-address instructions

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

Use of Addresses

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

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

How Many Addresses

- More addresses
  - More complex (powerful?) instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster fetch/execution of instructions

Design Decisions (1)

- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses

Design Decisions (2)

- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes (later…)

Types of Operand

- Addresses
- Numbers
  - Integer/floating point
- Characters
  - ASCII etc.
- Logical Data
  - Bits or flags
- (Aside: Is there any difference between numbers and characters? Ask a C programmer!)

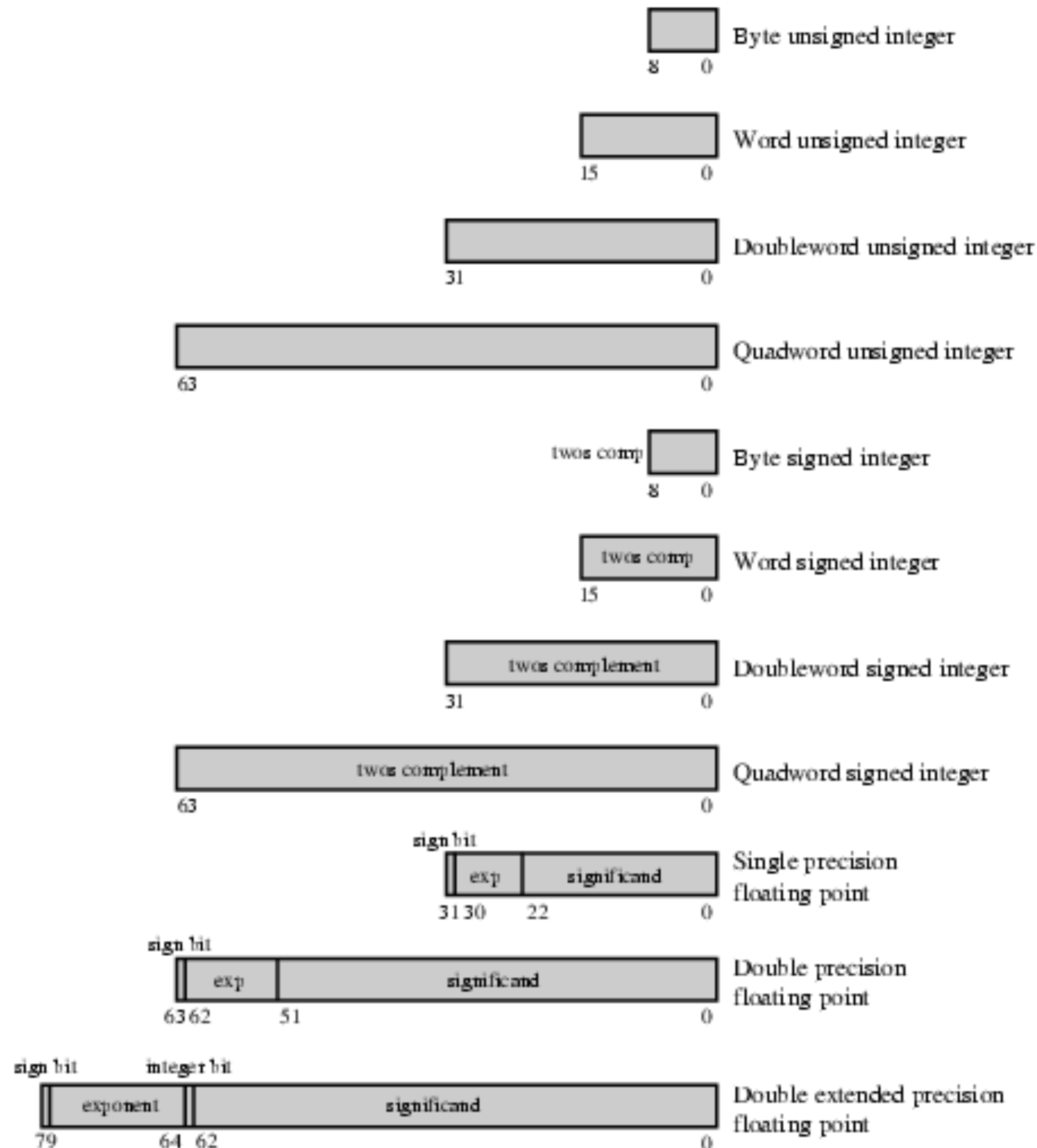
x86 Data Types

- 8 bit Byte
- 16 bit word
- 32 bit double word
- 64 bit quad word
- 128 bit double quadword
- Addressing is by 8 bit unit
- Words do not need to align at even-numbered address
- Data accessed across 32 bit bus in units of double word read at addresses divisible by 4
- Little endian

SIMD Data Types

- Integer types
  - Interpreted as bit field or integer
- Packed byte and packed byte integer
  - Bytes packed into 64-bit quadword or 128-bit double quadword
- Packed word and packed word integer
  - 16-bit words packed into 64-bit quadword or 128-bit double quadword
- Packed doubleword and packed doubleword integer
  - 32-bit doublewords packed into 64-bit quadword or 128-bit double quadword
- Packed quadword and packed qaudword integer
  - Two 64-bit quadwords packed into 128-bit double quadword
- Packed single-precision floating-point and packed double-precision floating-point
  - Four 32-bit floating-point or two 64-bit floating-point values packed into a 128-bit double quadword

x86 Numeric Data Formats

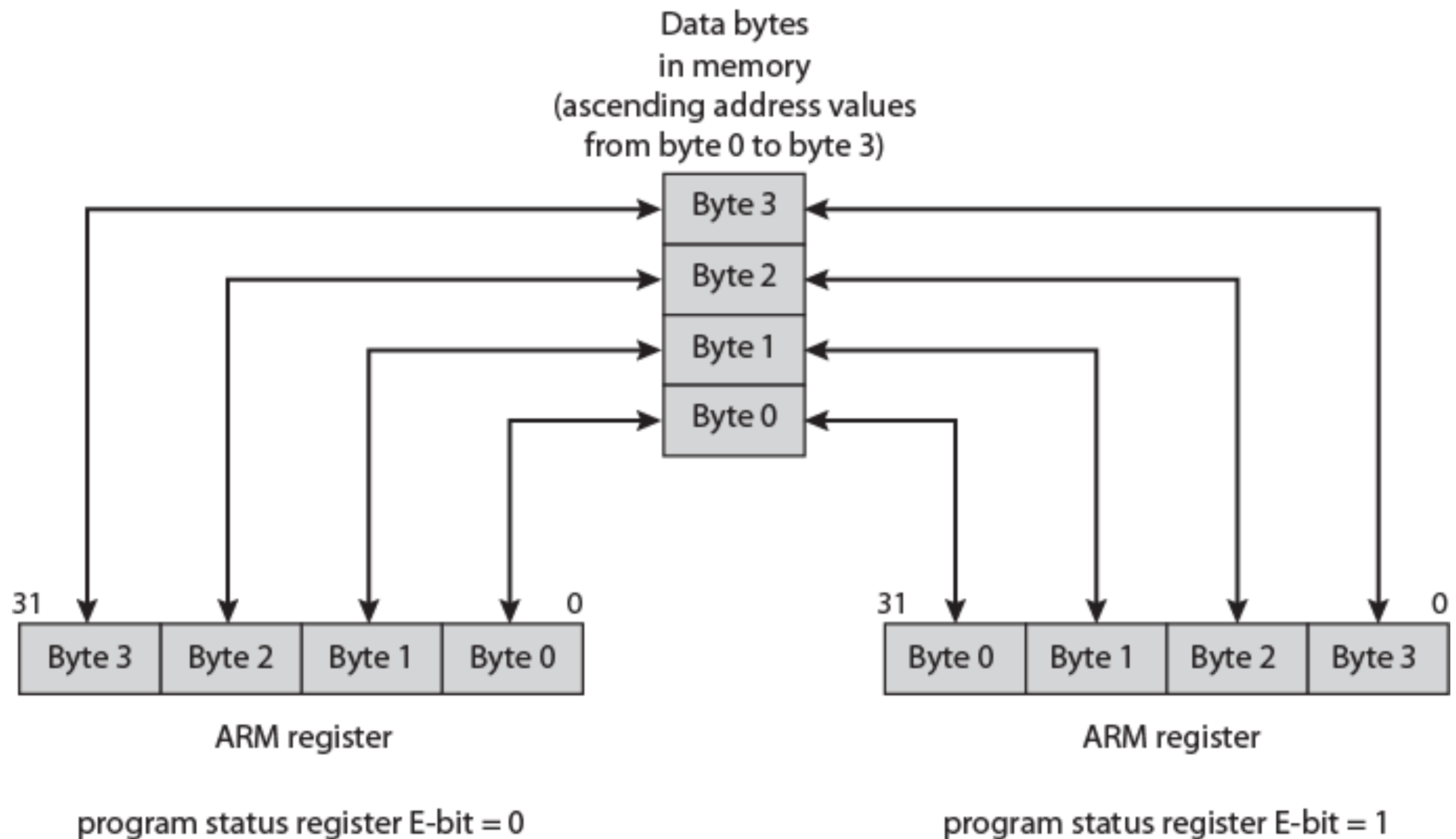


ARM Data Types

- 8 (byte), 16 (halfword), 32 (word) bits
- Halfword and word accesses should be word aligned
- Nonaligned access alternatives
  - Default
    - Treated as truncated
    - Bits[1:0] treated as zero for word
    - Bit[0] treated as zero for halfword
    - Load single word instructions rotate right word aligned data transferred by non word-aligned address one, two or three bytes
  - Alignment checking: Data abort signal indicates alignment fault for attempting unaligned access
  - Unaligned access: Processor uses one or more memory accesses to generate transfer of adjacent bytes transparently to the programmer
- Unsigned integer interpretation supported for all types
- Twos-complement signed integer interpretation supported for all types
- Majority of implementations do not provide floating-point hardware
  - Saves power and area
  - Floating-point arithmetic implemented in software
  - Optional floating-point coprocessor
  - Single- and double-precision IEEE 754 floating point data types

ARM Endian Support

- E-bit in system control register
- Under program control



Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

Types of Operation

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 OR NOT (Complement) Exclusive-OR	Perform the specified logical operation bitwise
	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
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)

Types of Operation

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

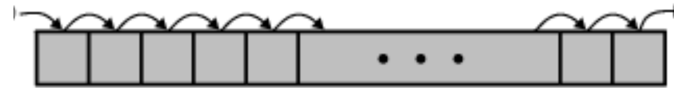
Data Transfer

- Specify
  - Source
  - Destination
  - Amount of data
- May be different instructions for different movements
  - e.g. IBM 370
- Or one instruction and different addresses
  - e.g. VAX

Arithmetic

- Add, Subtract, Multiply, Divide
- Signed Integer
- Floating point ?
- May include
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)

Shift and Rotate Operations



(a) Logical right shift

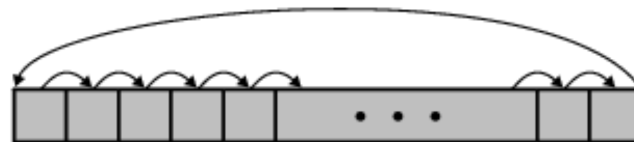
Logical shift

(b) Logical left shift

(c) Arithmetic right shift

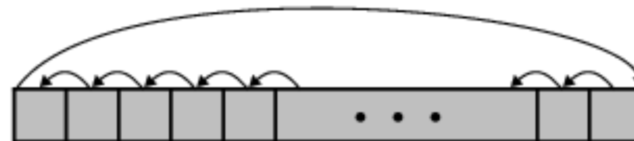
Arithmentic shift

(d) Arithmetic left shift



(e) Right rotate

Rotation



(f) Left rotate

Shift and Rotate Examples

Table 10.7 Examples of Shift and Rotate Operations

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
- Example:
- (R1) = 10100101
- (R2) = 11111111
- (R1) xor (R2) = 01011010

Conversion

- E.g. Binary to Decimal
- Example EBCDIC to IRA
- Conversion table stored starting from location 1000
- Location 2100 from 2103 are F1 F9 F8 F4
- R1 is address 2100
- R2 is address 1000
- TR R1 (4), R2 translate F1 to 31, F9 to 39 , F8, to 38 and F4 to 34 following the table.

Input/Output

- May be specific instructions
- May be done using data movement instructions (memory mapped)
- May be done by a separate controller (DMA)

Systems Control

- Privileged instructions
- CPU needs to be in specific state
  - Ring 0 on 80386+
  - Kernel mode
- For operating systems use

Transfer of Control

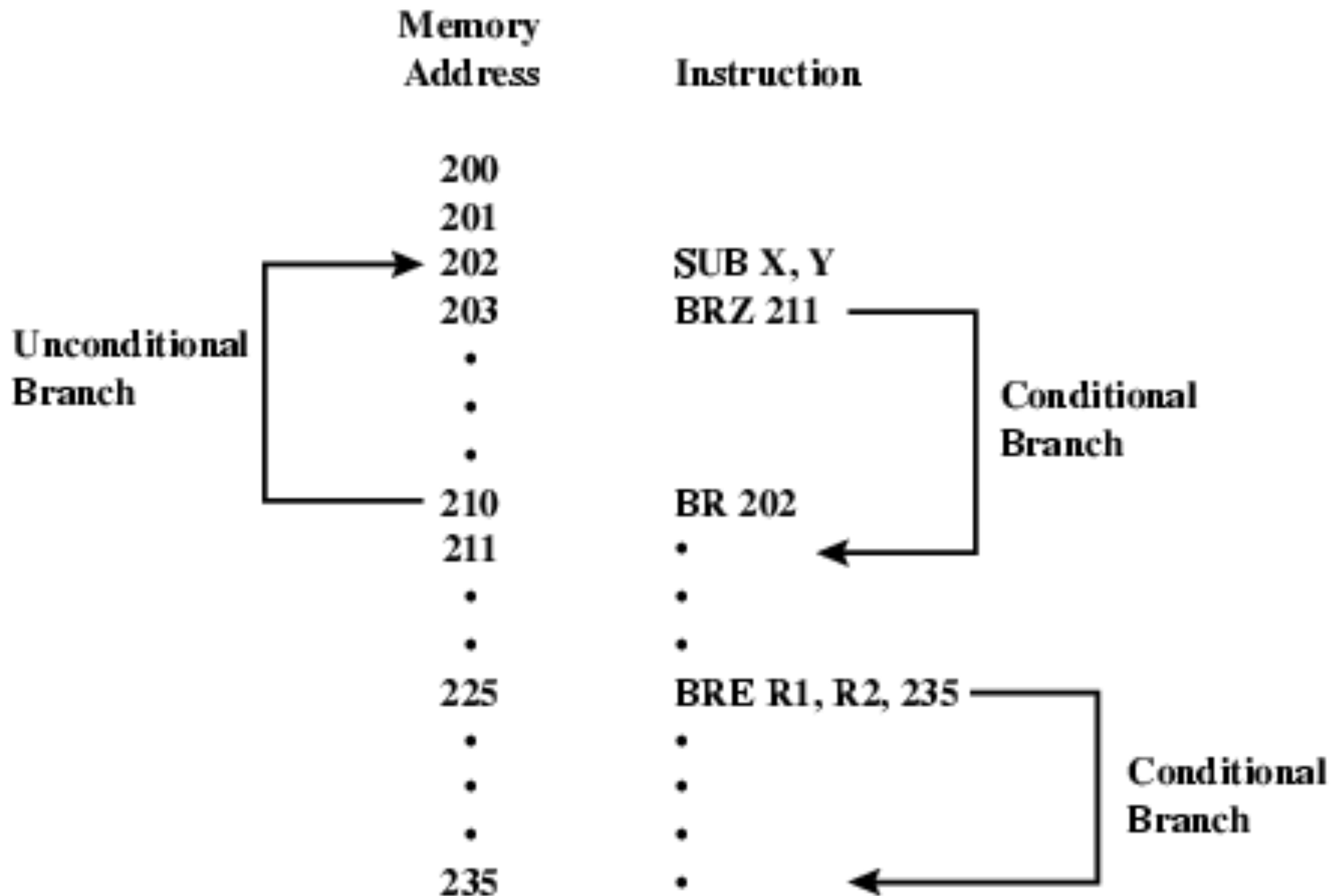
- Branch
  - e.g. branch to x if result is zero
- Skip
  - e.g. increment and skip if zero
  - ISZ Register1
  - Branch xxxx
  - ADD A
- Subroutine call
  - c.f. interrupt call

Transfer of Control

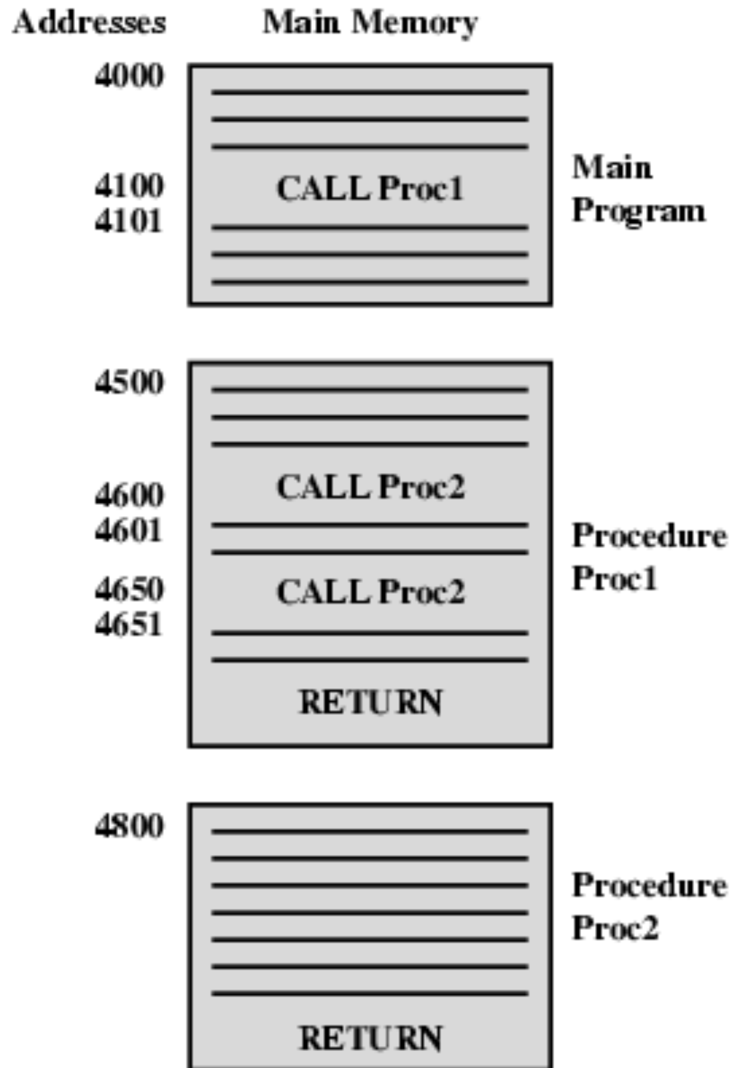
- BRP X branch if positive
- BRN X branch if negative
- BRZ X branch if zero
- BRO X branch if overflow

- BRE R1, R2, X branch if equal

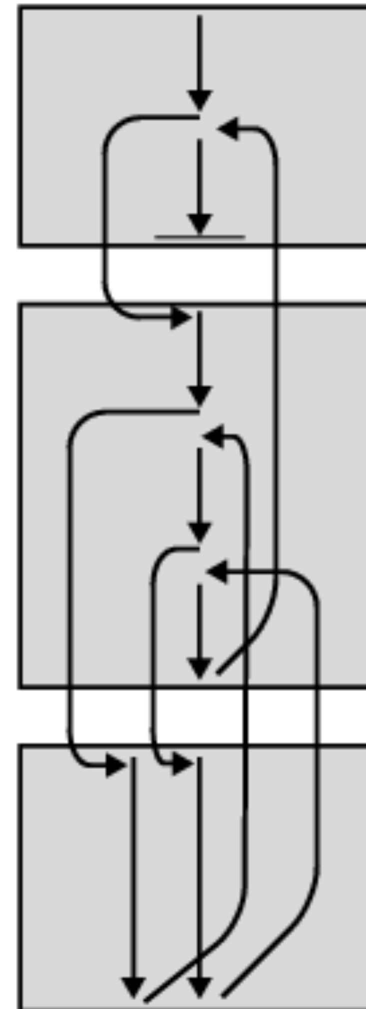
Branch Instruction



Nested Procedure Calls



(a) Calls and returns

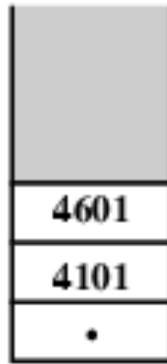


(b) Execution sequence

Use of Stack

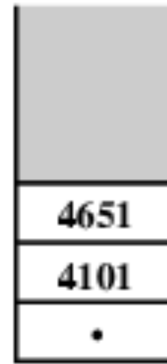
(a) Initial stack contents

(b) After CALL Proc1



(c) Initial CALL Proc2

(d) After RETURN

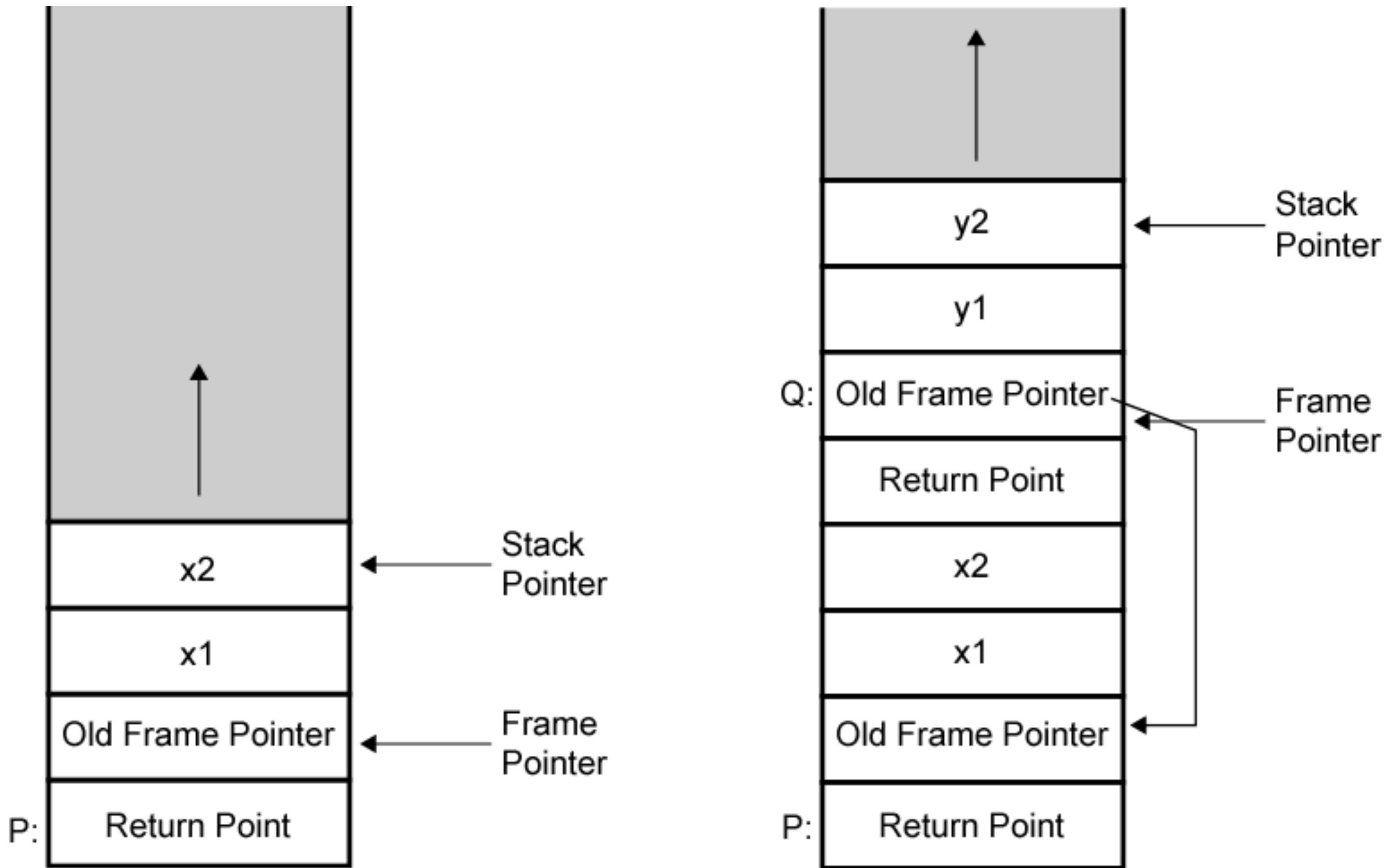


(e) After CALL Proc2

(f) After RETURN

(g) After RETURN

Stack Frame Growth Using Sample Procedures P and Q



Stack

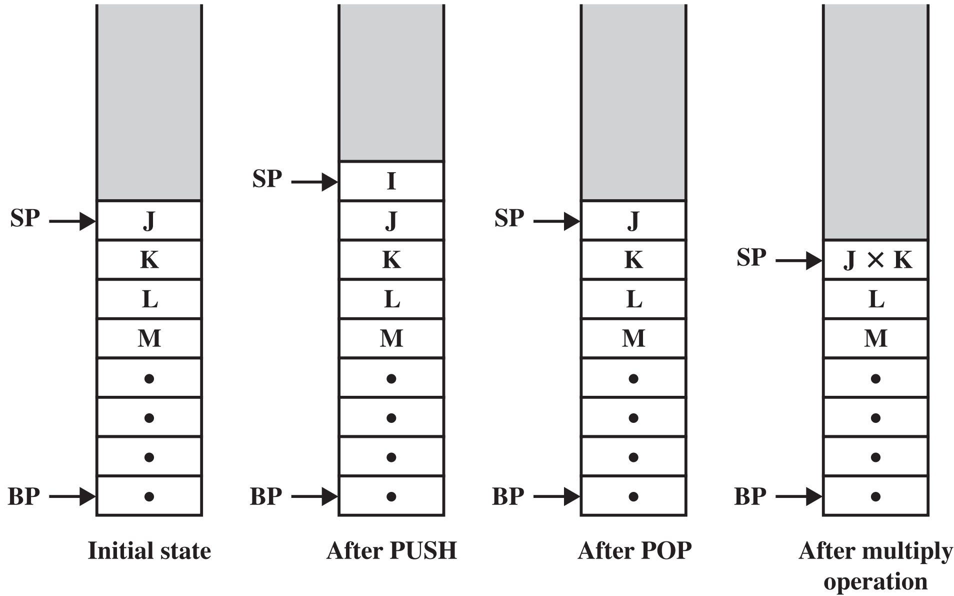


Figure 10.13 Basic Stack Operation (full/descending)

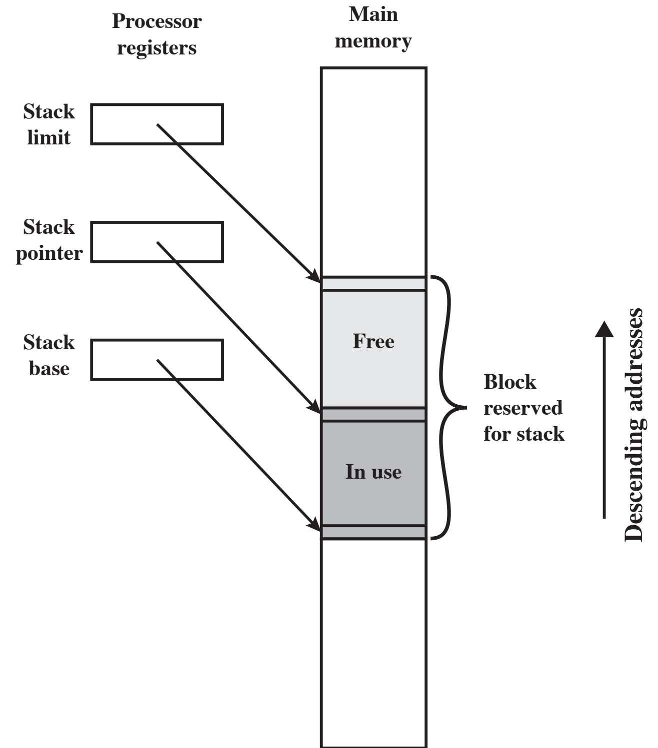


Figure 10.14 Typical Stack Organization (full/descending)

Expression evaluation

	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

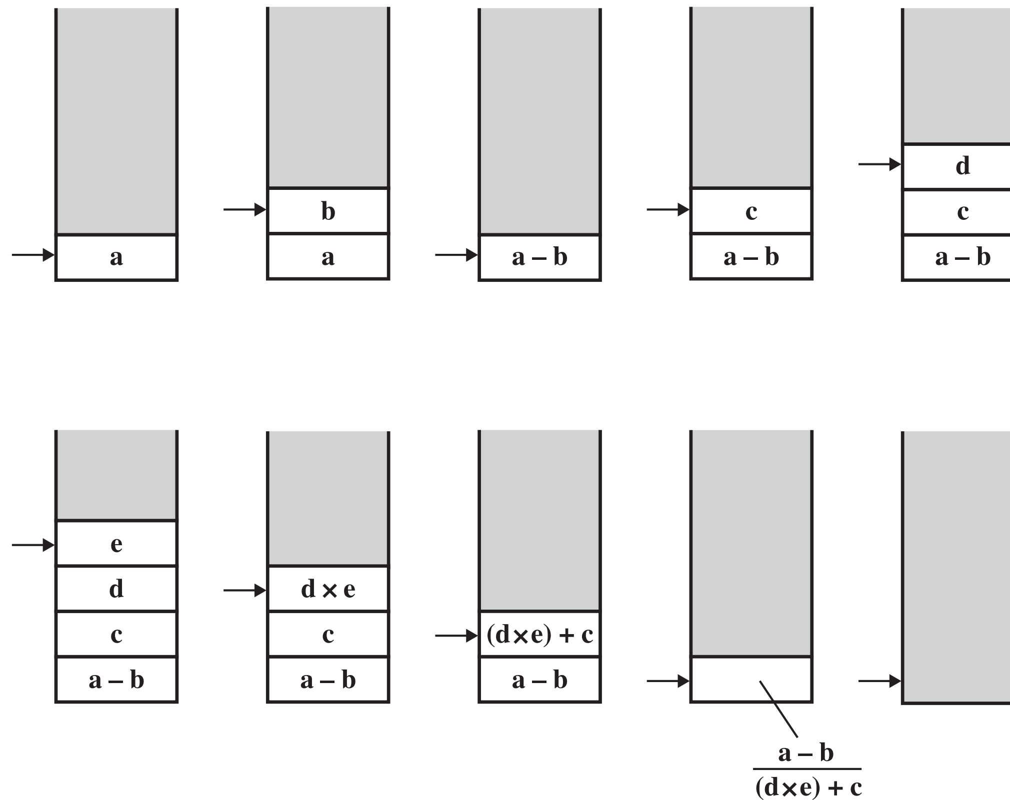


Figure 10.16 Use of Stack to Compute f = (a – b)/[(d × e) + c]

ab-cdex+/

Byte Order
(A portion of chips?)

- What order do we read numbers that occupy more than one byte
- e.g. (numbers in hex to make it easy to read)
- 12345678 can be stored in 4x8bit locations as follows

Byte Order (example)

- Address Value (1) Value(2)
- 184 12 78
- 185 34 56
- 186 56 34
- 187 78 12

- i.e. read top down or bottom up?

Byte Order Names

- The problem is called Endian
- The system on the left has the least significant byte in the lowest address
- This is called big-endian
- The system on the right has the least significant byte in the highest address
- This is called little-endian

Example of C Data Structure

```
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;     //0x6161_6364                   word
} s;
```

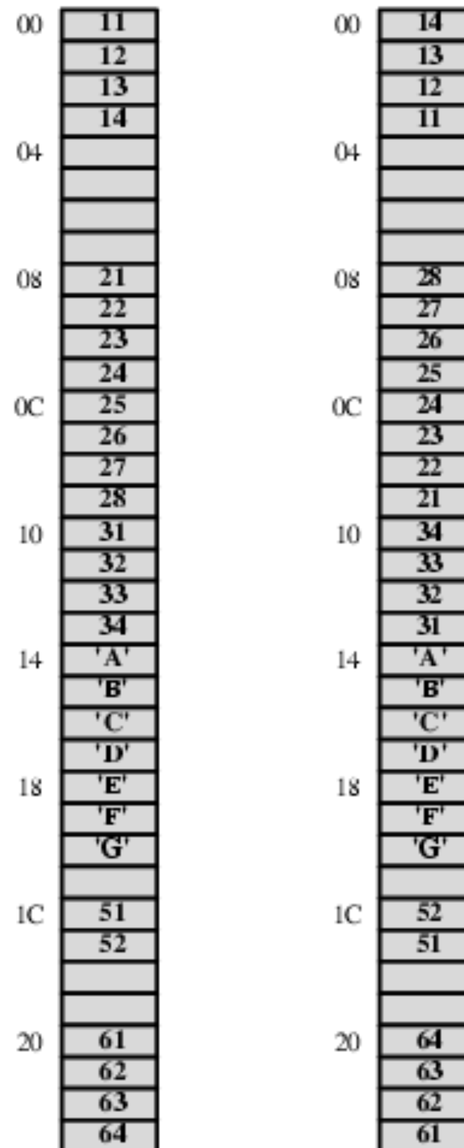
Big-endian address mapping

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

Little-endian address mapping

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

Alternative View of Memory Map



(a) Big-endian

(b) Little-endian

Standard…What Standard?

- Pentium (x86), VAX are little-endian
- IBM 370, Motorola 680x0 (Mac), and most RISC are big-endian
- Internet is big-endian
  - Makes writing Internet programs on PC more awkward!
  - WinSock provides htoi and itoh (Host to Internet & Internet to Host) functions to convert