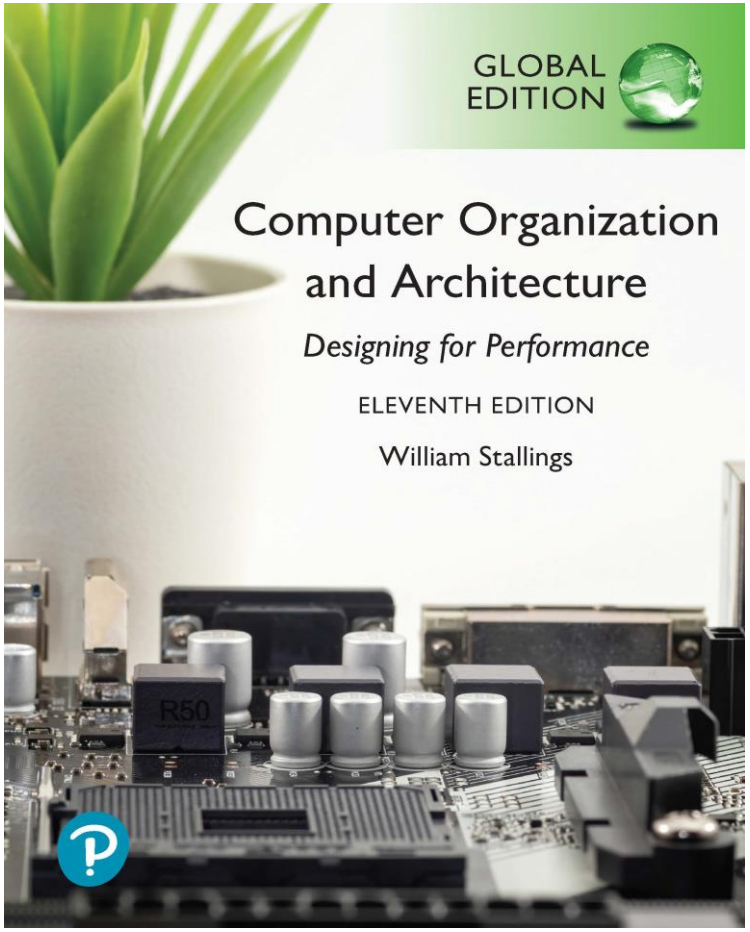
Computer Organization and Architecture

Designing for Performance

11th Edition, Global Edition

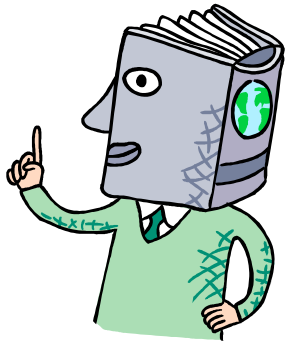


Chapter 13

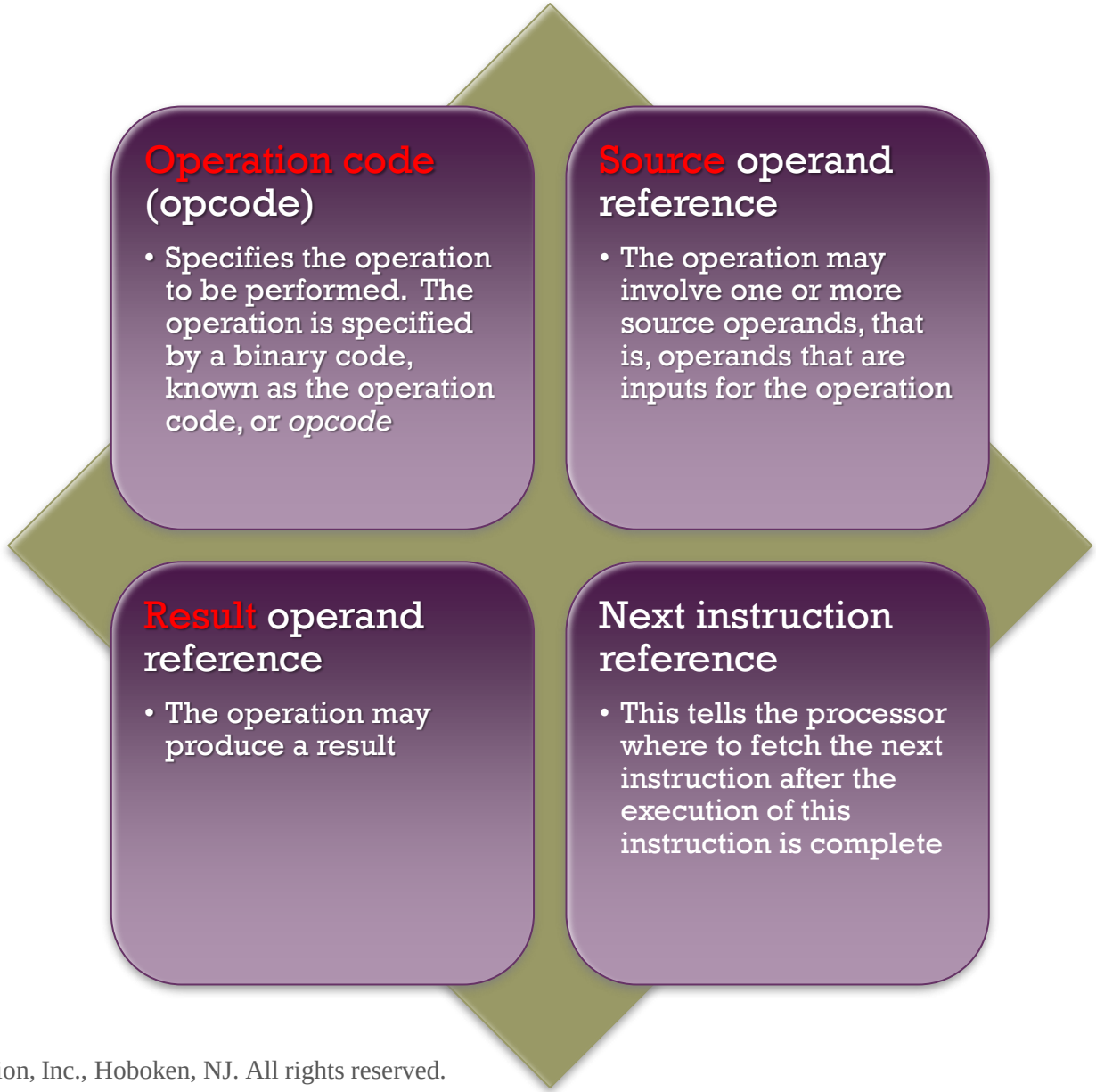
Instruction Sets: Characteristics and Functions

Machine Instruction Characteristics

- The operation of the processor is determined by the instructions it executes, referred to as *machine instructions* or *computer instructions*
- The collection of different instructions that the processor can execute is referred to as the processor's *instruction set*
- Each instruction must contain the information required by the processor for execution



Elements of a Machine Instruction



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

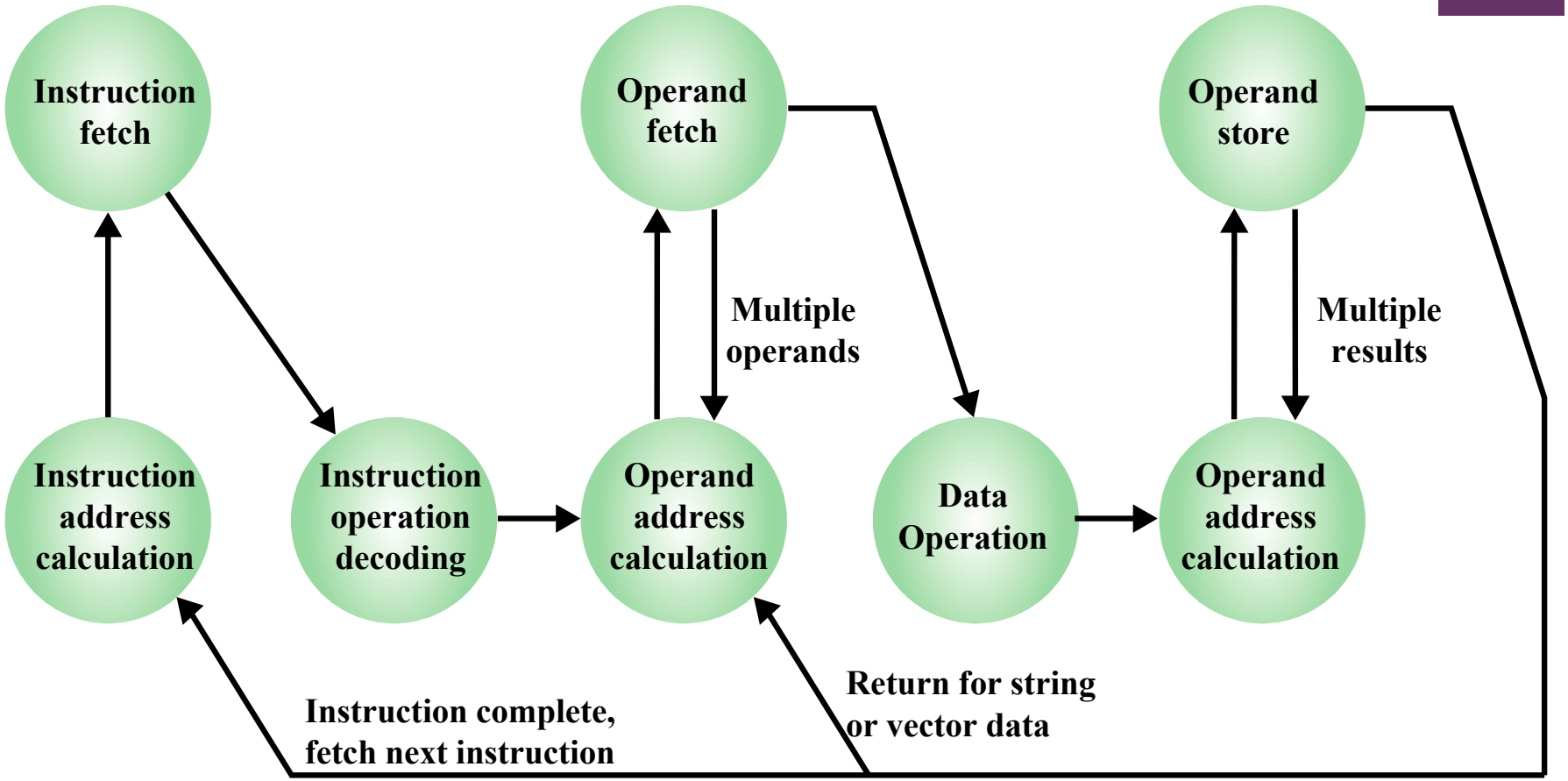


Figure 12.1 Instruction Cycle State Diagram

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

1) Main or virtual memory
   - As with next instruction references, the main or virtual memory address must be supplied

2) I/O device
   - The instruction must specify the I/O module and device for the operation. If memory-mapped I/O is used, this is just another main or virtual memory address

3) Processor register
   - A processor contains one or more registers that may be referenced by machine instructions.
   - If more than one register exists each register is assigned a unique name or number and the instruction must contain the number of the desired register

4) Immediate
   - The value of the operand is contained in a field in the instruction being executed

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Instruction Representation

- Within the computer each instruction is represented by a sequence of bits
- The instruction is divided into fields, corresponding to the constituent elements of the instruction

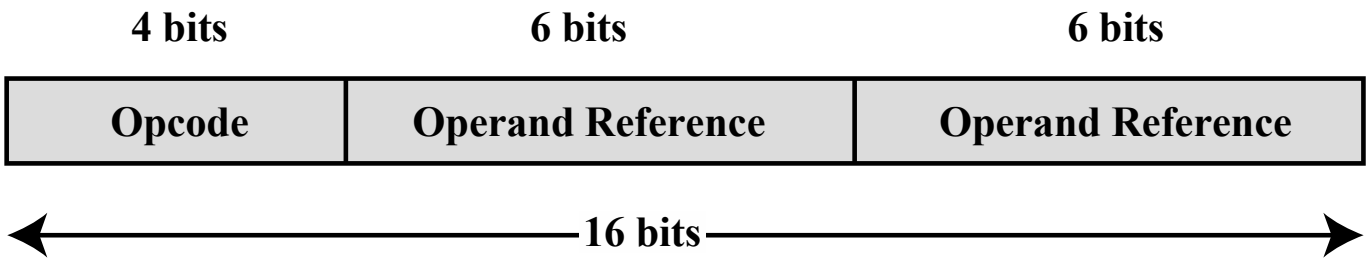


Figure 12.2 A Simple Instruction Format

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Instruction Representation

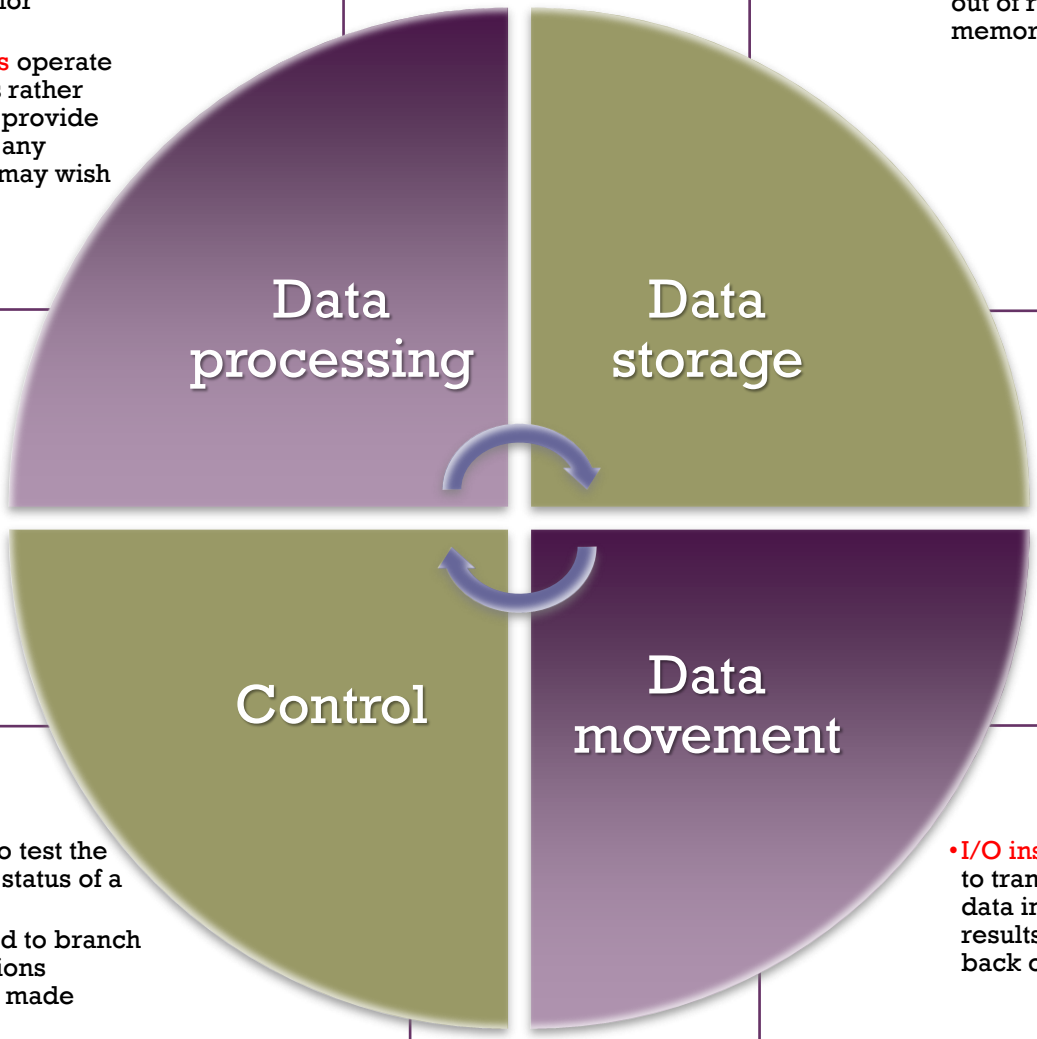
- Opcodes are represented by abbreviations called *mnemonics*
- Examples include:
  - ADD Add
  - SUB Subtract
  - MUL Multiply
  - DIV Divide
  - LOAD Load data from memory
  - STOR Store data to memory
- Operands are also represented symbolically
- Each symbolic opcode has a fixed binary representation
  - The programmer specifies the location of each symbolic operand

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Instruction Types

- Arithmetic instructions provide computational capabilities for processing numeric data
- Logic (Boolean) instructions operate on the bits of a word as bits rather than as numbers, thus they provide capabilities for processing any other type of data the user may wish to employ

- Movement of data into or out of register and or memory locations



- Test instructions are used to test the value of a data word or the status of a computation
- Branch instructions are used to branch to a different set of instructions depending on the decision made

- I/O instructions are needed to transfer programs and data into memory and the results of computations back out to the user

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Table 12.1

Utilization of Instruction Addresses (Nonbranching Instructions)

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

Simpler yet is the one-address instruction. For this to work, a second address must be implicit. This was common in earlier machines, with the implied address being a processor register known as the **accumulator** (AC). The accumulator contains one of the operands and is used to store the result. In our example, eight instructions are needed to accomplish the task.

It is, in fact, possible to make do with zero addresses for some instructions. Zero-address instructions are applicable to a special memory organization called a stack. A stack is a last-in-first-Out set of locations. The stack is in a known location and, often, at least the top two elements are in processor registers. Thus, zero-address instructions would reference the top two stack elements.

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

For the one-address instruction, a second address must be implicit. This was common in earlier machines, with the implied address being a processor register known as the **accumulator** (AC). The accumulator contains one of the operands and is used to store the result.

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Instruction Set Design

Very complex because it affects so many aspects of the computer system

Defines many of the functions performed by the processor

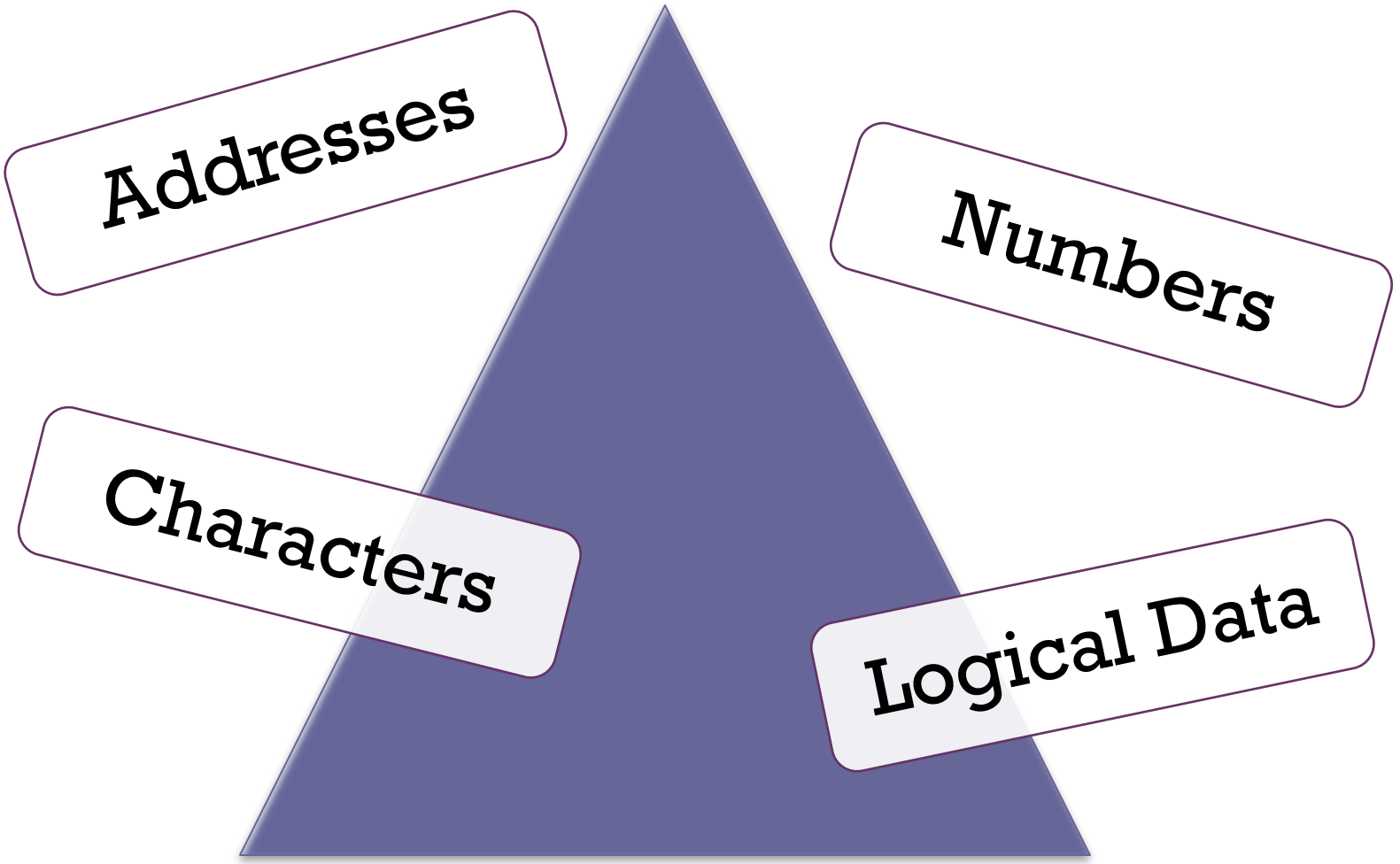
Programmer's means of controlling the processor

Fundamental design issues:

Operation repertoire	Data types	Instruction format	Registers	Addressing
• How many and which operations to provide and how complex operations should be	• The various types of data upon which operations are performed	• Instruction length in bits, number of addresses, size of various fields, etc.	• Number of processor registers that can be referenced by instructions and their use	• The mode or modes by which the address of an operand is specified

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Types of Operands



© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Numbers

- All machine languages include numeric data types
- Numbers stored in a computer are limited:
  - Limit to the magnitude of numbers representable on a machine
  - In the case of floating-point numbers, a limit to their precision
- Three types of numerical data are common in computers:
  - Binary integer or binary fixed point
  - Binary floating point
  - Decimal
- Packed decimal
  - Each decimal digit is represented by a 4-bit code with two digits stored per byte
  - To form numbers 4-bit codes are strung together, usually in multiples of 8 bits

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Characters

- A common form of data is text or character strings
- Textual data in character form cannot be easily stored or transmitted by data processing and communications systems because they are designed for binary data
- Most commonly used character code is the International Reference Alphabet (IRA)
  - Referred to in the United States as the American Standard Code for Information Interchange (ASCII)
- Another code used to encode characters is the Extended Binary Coded Decimal Interchange Code (EBCDIC)
  - EBCDIC is used on IBM mainframes

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Logical Data

- Normally, each word or other addressable unit (byte, halfword, and so on) is treated as a single unit of data. It is sometimes useful, however, to consider an n-bit unit as consisting of n 1-bit items of data, each item having the value 0 or 1. When data are viewed this way, they are considered to be logical data.
- An *n*-bit unit consisting of *n* 1-bit items of data, each item having the value 0 or 1
- Two advantages to bit-oriented view:
  - Memory can be used most efficiently for storing an array of Boolean or binary data items in which each item can take on only the values 1 (true) and 0 (false)
  - To manipulate the bits of a data item
    - If floating-point operations are implemented in software, we need to be able to shift significant bits in some operations
    - To convert from IRA to packed decimal, we need to extract the rightmost 4 bits of each byte

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Intel x86 Data Types

- The x86 can deal with data types of 8 (byte), 16 (word), 32 (doubleword), 64 (quad- word), and 128 (double quadword) bits in length.
- To allow maximum flexibility in data structures and efficient memory utilization, words need not be aligned at even- numbered addresses; doublewords need not be aligned at addresses evenly divisible by 4; and quadwords need not be aligned at addresses evenly divisible by 8; and so on. However, when data are accessed across a 32-bit bus, data transfers take place in units of doublewords, beginning at addresses divisible by 4. The processor converts the request for misaligned values into a sequence of requests for the bus transfer. As with all of the Intel 80x86 machines, the x86 uses the little-endian style; that is, the least significant byte is stored in the lowest address (see Appendix 12A for a discussion of endianness).
- The byte, word, doubleword, quadword, and double quadword are referred to as general data types. In addition, the x86 supports an impressive array of specific data types that are recognized and operated on by particular instructions. Table 12.2 summarizes these types.

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Data Type	Description
General	Byte, word (16 bits), doubleword (32 bits), quadword (64 bits), and double quadword (128 bits) locations with arbitrary binary contents.
Integer	A signed binary value co4ntained in a byte, word, or doubleword, using twos complement representation.
Ordinal	An unsigned integer contained in a byte, word, or doubleword.
Unpacked binary coded decimal (BCD)	A representation of a BCD digit in the range 0 through 9, with one digit in each byte.
Packed BCD	Packed byte representation of two BCD digits; value in the range 0 to 99.
Near pointer	A 16-bit, 32-bit, or 64-bit effective address that represents the offset within a segment. Used for all pointers in a nonsegmented memory and for references within a segment in a segmented memory.
Far pointer	A logical address consisting of a 16-bit segment selector and an offset of 16, 32, or 64 bits. Far pointers are used for memory references in a segmented memory model where the identity of a segment being accessed must be specified explicitly.
Bit field	A contiguous sequence of bits in which the position of each bit is considered as an independent unit. A bit string can begin at any bit position of any byte and can contain up to 32 bits.
Bit string	A contiguous sequence of bits, containing from zero to $2_{32} - 1$ bits.
Byte string	A contiguous sequence of bytes, words, or doublewords, containing from zero to $2_{32} - 1$ bytes.
Floating point	See Figure 12.4.
Packed SIMD (single instruction, multiple data)	Packed 64-bit and 128-bit data types

Table 12.2

x86 Data Types

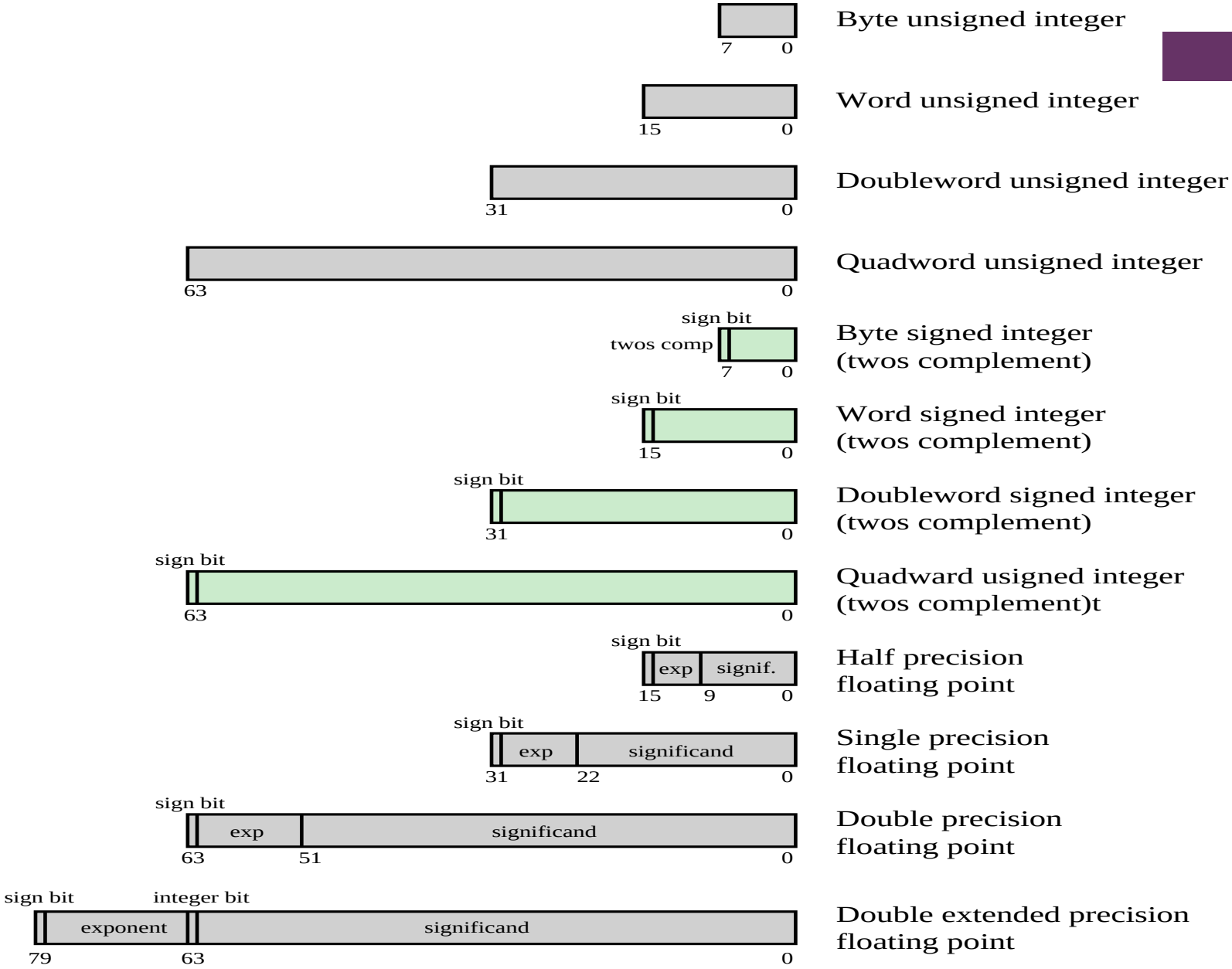


Figure 12.4 x86 Numeric Data Formats

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Single-Instruction-Multiple-Data (SIMD) Data Types

- Introduced to the x86 architecture as part of the extensions of the instruction set to optimize performance of multimedia applications
- These extensions include MMX (multimedia extensions) and SSE (streaming SIMD extensions)

Data types:

- **Packed byte and packed byte integer:** Bytes packed into a 64-bit quadword or 128-bit double quadword, interpreted as a bit field or as an integer
- **Packed word and packed word integer:** 16-bit words packed into a 64-bit quadword or 128-bit double quadword, interpreted as a bit field or as an integer
- **Packed doubleword and packed doubleword integer:** 32-bit doublewords packed into a 64-bit quadword or 128-bit double quadword, interpreted as a bit field or as an integer
- **Packed quadword and packed qaudword integer:** Two 64-bit quadwords packed into a 128-bit double quadword, interpreted as a bit field or as an integer
- **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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

ARM Processors

- ARMv7 is an older instruction set also from ARM Holdings plc, but with 32-bit addresses instead of ARMv8's 64 bits.
- More than 14 billion chips with ARM processors were manufactured in 2015, making them the most popular instruction sets in the world.

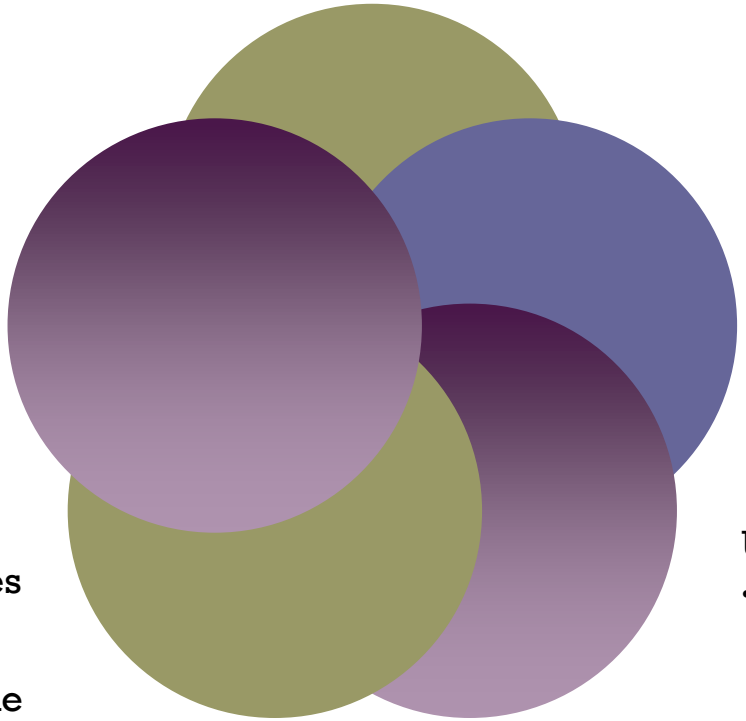
© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

ARMv7 Data Types

ARM processors support data types of:

- 8 (byte)
- 16 (halfword)
- 32 (word) bits in length

All three data types can also be used for twos complement signed integers



Alignment checking

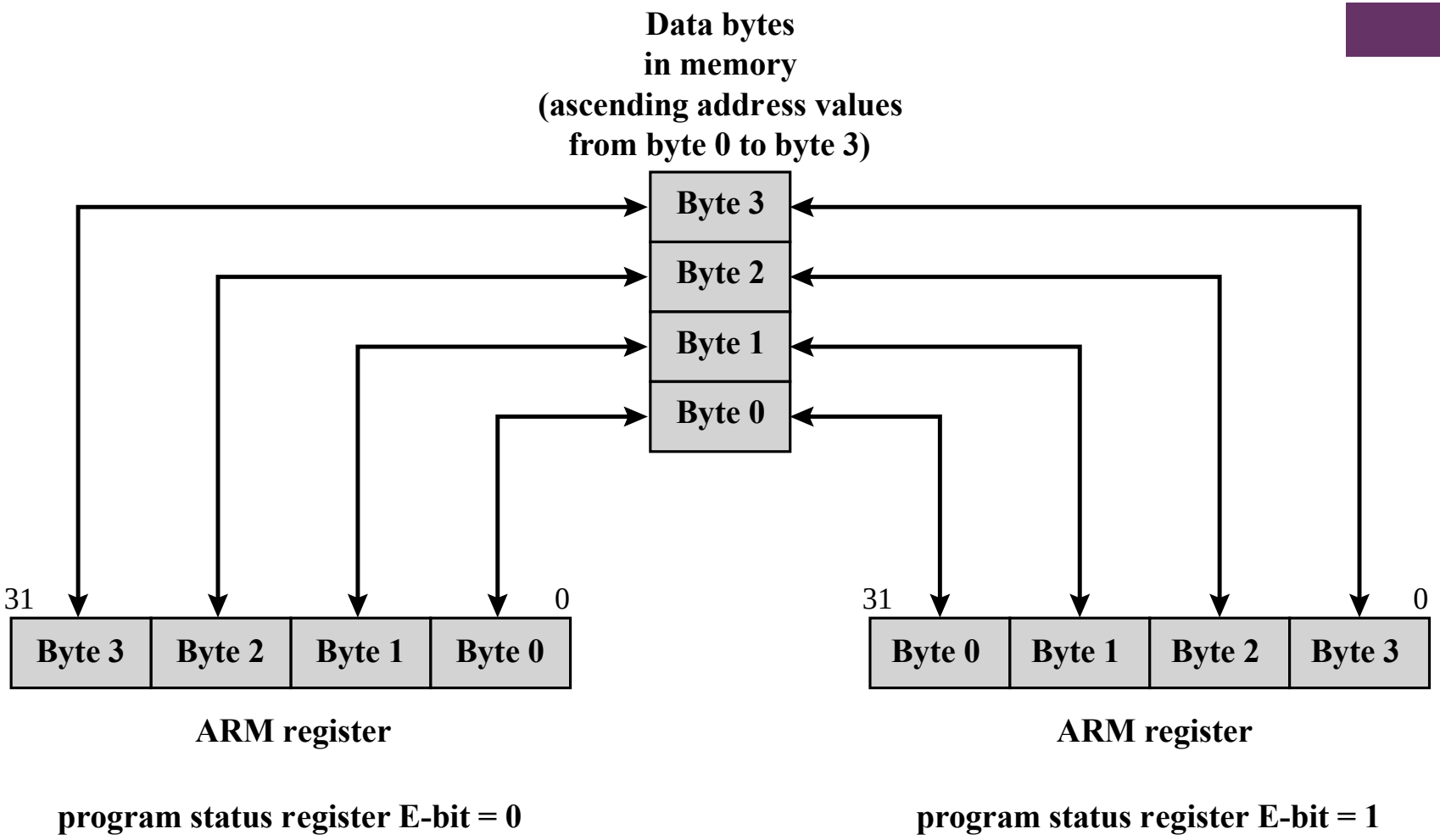
- When the appropriate control bit is set, a data abort signal indicates an alignment fault for attempting unaligned access

For all three data types an unsigned interpretation is supported in which the value represents an unsigned, nonnegative integer

Unaligned access

- When this option is enabled, the processor uses one or more memory accesses to generate the required transfer of adjacent bytes transparently to the programmer

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.



A state bit (E-bit) in the system control register is set and cleared under program control using the SETEND instruction. The E-bit defines which endian to load and store data

Figure 12.5 ARM Endian Support - Word Load/Store with E-bit

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

TYPES OF OPERATIONS

- The number of different opcodes varies widely from machine to machine. However, the same general types of operations are found on all machines. A useful and typical categorization is the following:
  - Data transfer
  - Arithmetic
  - Logical
  - Conversion
  - I/O
  - System control
  - Transfer of control

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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

Table 12.3

Common Instruction Set Operations (page 1 of 2)

(Table can be found on page 426 in textbook.)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

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)

Table 12.3

Common Instruction Set Operations (page 2 of 2)

(Table can be found on page 426 in textbook.)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Table 12.4
Processor Actions for Various Types of 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

(Table can be found on page 427 in textbook.)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Data Transfer

Most fundamental type of machine instruction is the data transfer instruction.

Must specify:

- Location of the source and destination operands
- The length of data to be transferred must be indicated
- The mode of addressing for each operand must be specified

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

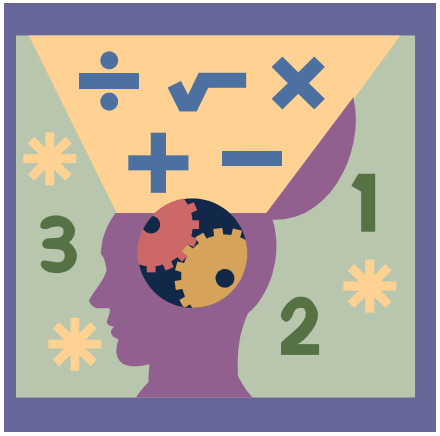
Table 12.5
Examples of IBM EAS/390 Data Transfer Operations

Operation Mnemonic	Name	Number of Bits Transferred	Description
L	Load	32	Transfer from memory to register
LH	Load Halfword	16	Transfer from memory to register
LR	Load	32	Transfer from register to register
LER	Load (Short)	32	Transfer from floating-point register to floating-point register
LE	Load (Short)	32	Transfer from memory to floating-point register
LDR	Load (Long)	64	Transfer from floating-point register to floating-point register
LD	Load (Long)	64	Transfer from memory to floating-point register
ST	Store	32	Transfer from register to memory
STH	Store Halfword	16	Transfer from register to memory
STC	Store Character	8	Transfer from register to memory
STE	Store (Short)	32	Transfer from floating-point register to memory
STD	Store (Long)	64	Transfer from floating-point register to memory

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

(Table can be found on page 428 in textbook.)

Arithmetic



- Most machines provide the basic arithmetic operations of add, subtract, multiply, and divide
- These are provided for signed integer (fixed-point) numbers
- Often they are also provided for floating-point and packed decimal numbers
- Other possible operations include a variety of single-operand instructions:
  - Absolute
    - Take the absolute value of the operand
  - Negate
    - Negate the operand
  - Increment
    - Add 1 to the operand

  Decrement
    - Subtract 1 from the operand

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Table 12.6
Basic Logical Operations

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

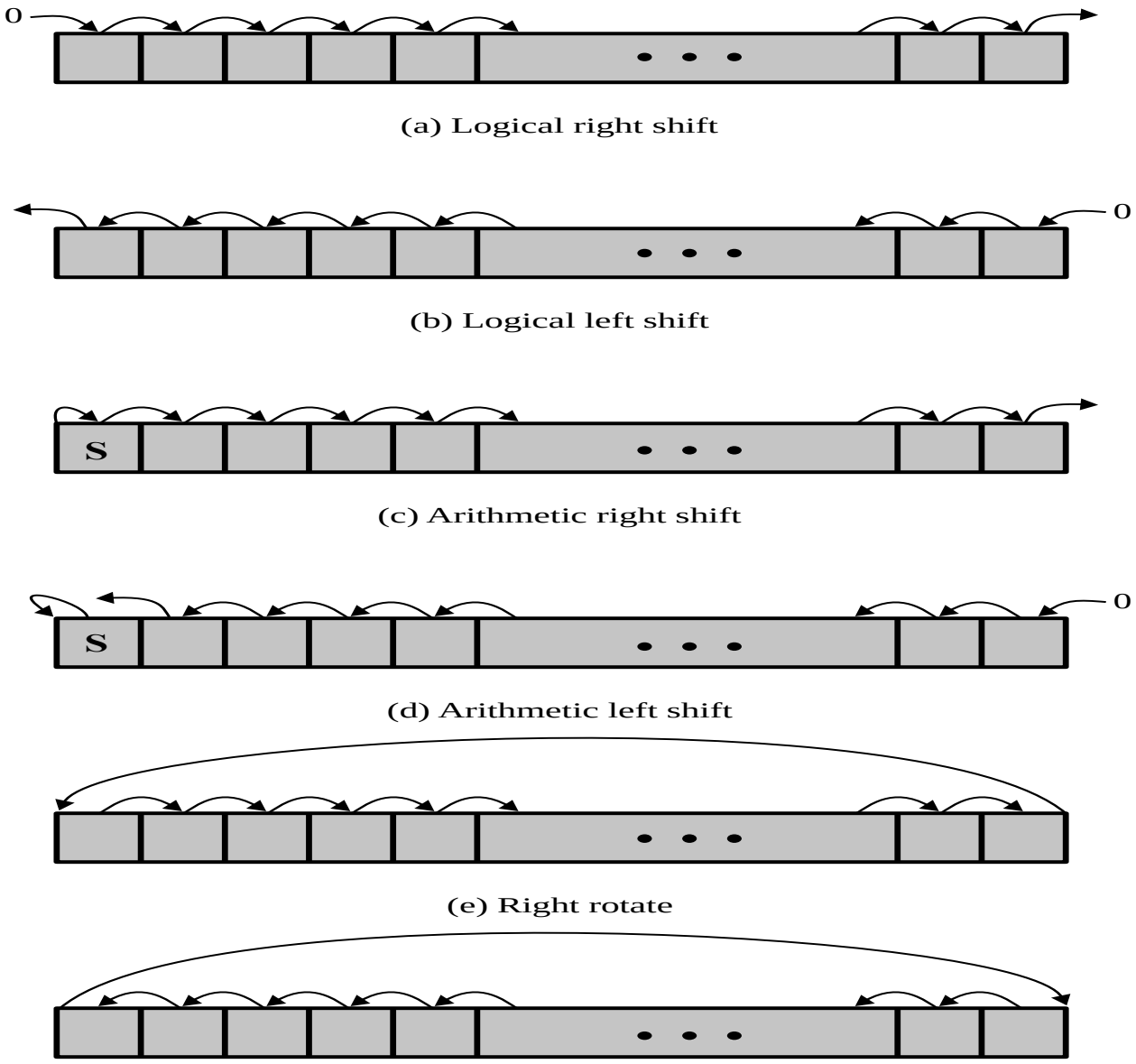


Figure 12.6 Shift and Rotate Operations

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Table 12.7

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

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Conversion

Instructions that change the format or operate on the format of data

An example is converting from decimal to binary

Or

translating from EBCDIC to IRA

An example of a more complex editing instruction is the EAS/390 Translate (TR) instruction

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Input/Output

- Variety of approaches taken:
  - Isolated programmed I/O
  - Memory-mapped programmed I/O
  - DMA
  - Use of an I/O processor
- Many implementations provide only a few I/O instructions, with the specific actions specified by parameters, codes, or command words

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

System Control

Instructions that can be executed only while the processor is in a certain privileged state or is executing a program in a special privileged area of memory

Typically these instructions are reserved for the use of the operating system

Examples of system control operations:

- A system control instruction may read or alter a control register
- An instruction to read or modify a storage protection key
- Access to process control blocks in a multiprogramming system

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Transfer of Control

- Reasons why transfer-of-control operations are required:
  - It is essential to be able to execute each instruction more than once
  - Virtually all programs involve some decision making
  - It helps if there are mechanisms for breaking the task up into smaller pieces that can be worked on one at a time
- Most common transfer-of-control operations found in instruction sets:
  - Branch
  - Skip
  - Procedure call

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

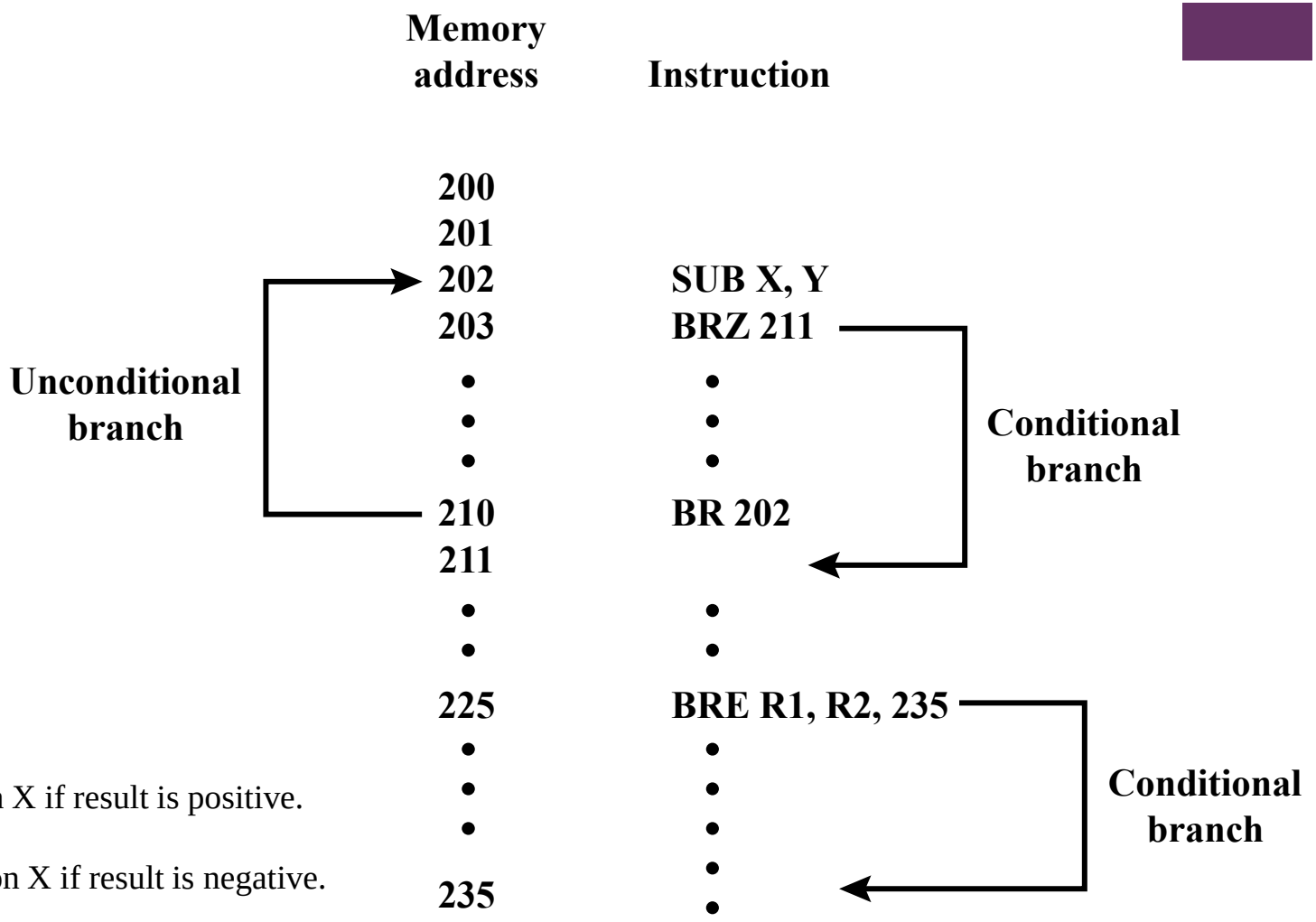


Figure 12.7 Branch Instructions

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Skip Instructions

Includes an implied address

Typically implies that one instruction be skipped, thus the implied address equals the address of the next instruction plus one instruction length

Because the skip instruction does not require a destination address field it is free to do other things

Example is the increment-and-skip-if-zero (ISZ) instruction

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

The increment-and-skip-if-zero (ISZ) instruction

In this fragment, the two transfer-of-control instructions are used to implement an iterative loop. R1 is set with the negative of the number of iterations to be performed. At the end of the loop, R1 is incremented. If it is not 0, the program branches back to the beginning of the loop. Otherwise, the branch is skipped, and the program continues with the next instruction after the end of the loop.

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Procedure Call Instructions

- A procedure (function) is self-contained computer program that is incorporated into a larger program
  - At any point in the program the procedure may be invoked, or *called*
  - Processor is instructed to go and execute the entire procedure and then return to the point from which the call took place
- Two principal reasons for use of procedures:
  - Economy
    - A procedure allows the same piece of code to be used many times
  - Modularity
- Involves two basic instructions:
  - A call instruction that branches from the present location to the procedure
  - Return instruction that returns from the procedure to the place from which it was called

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

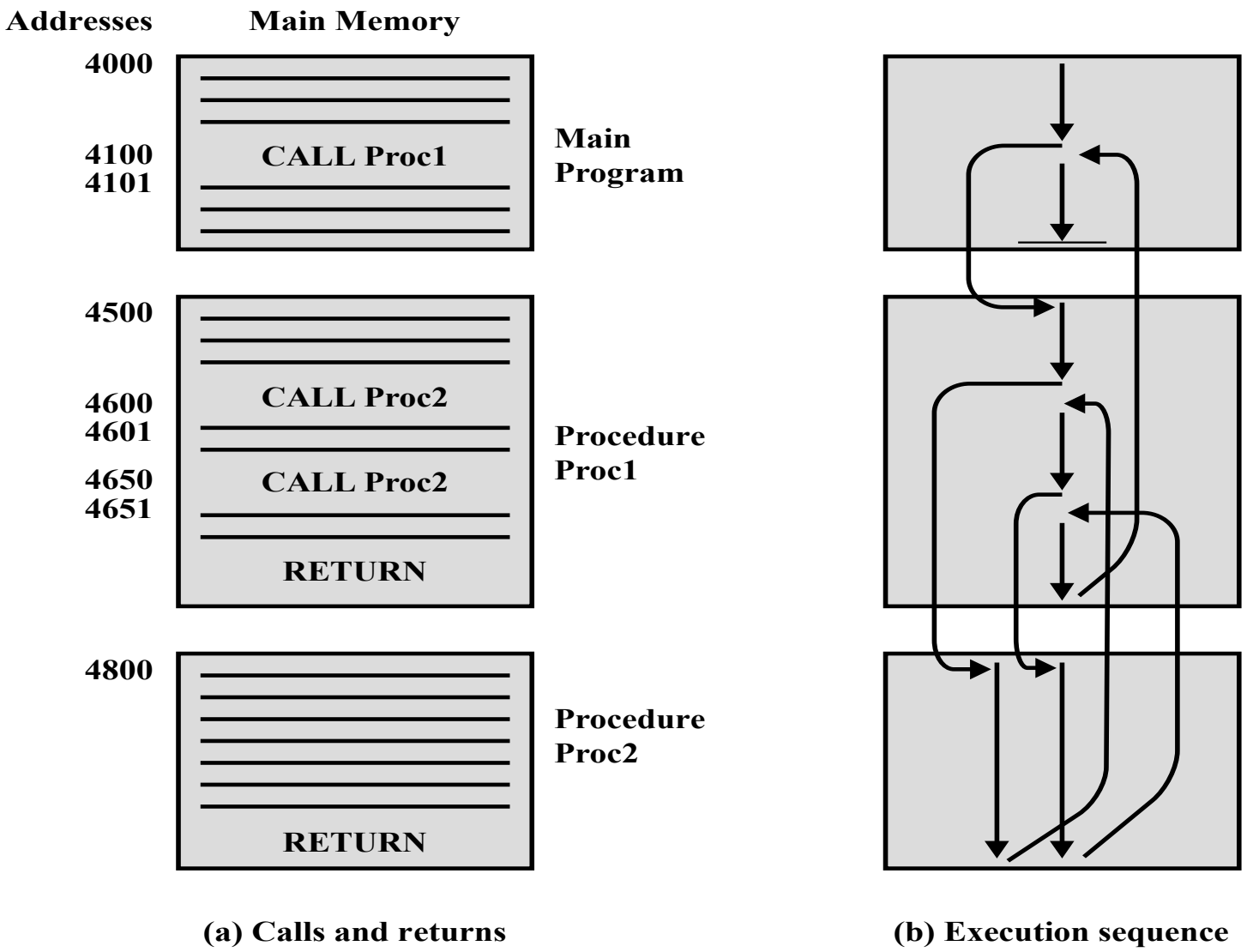


Figure 12.8 Nested Procedures

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

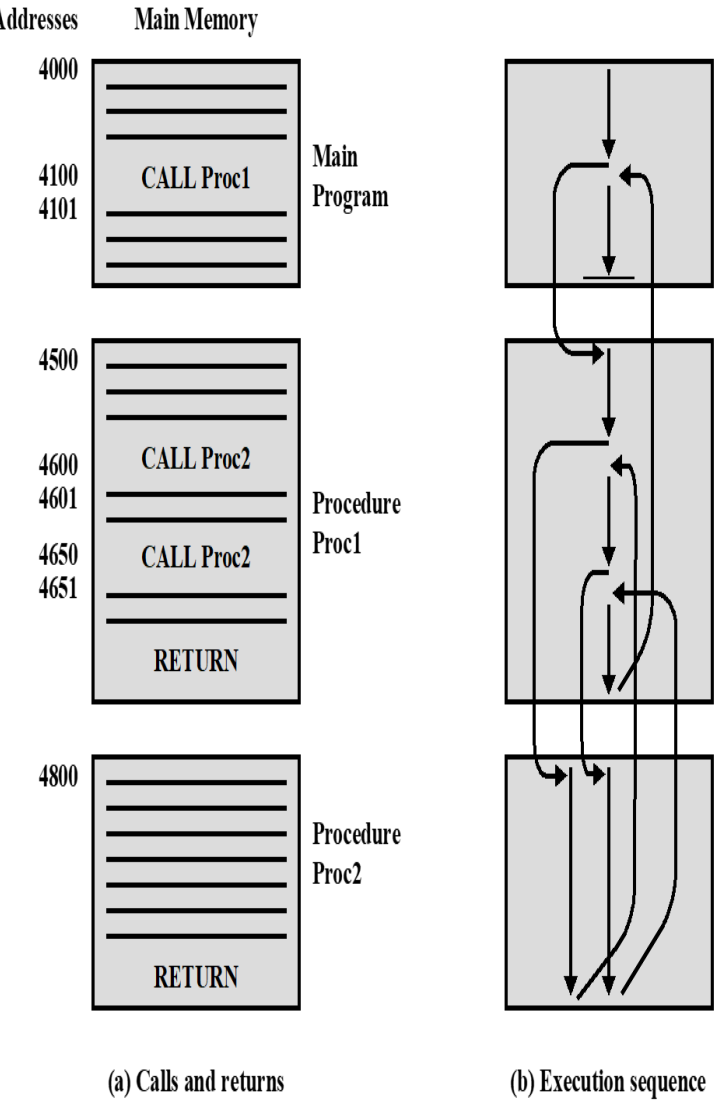


Figure 12.8 Nested Procedures

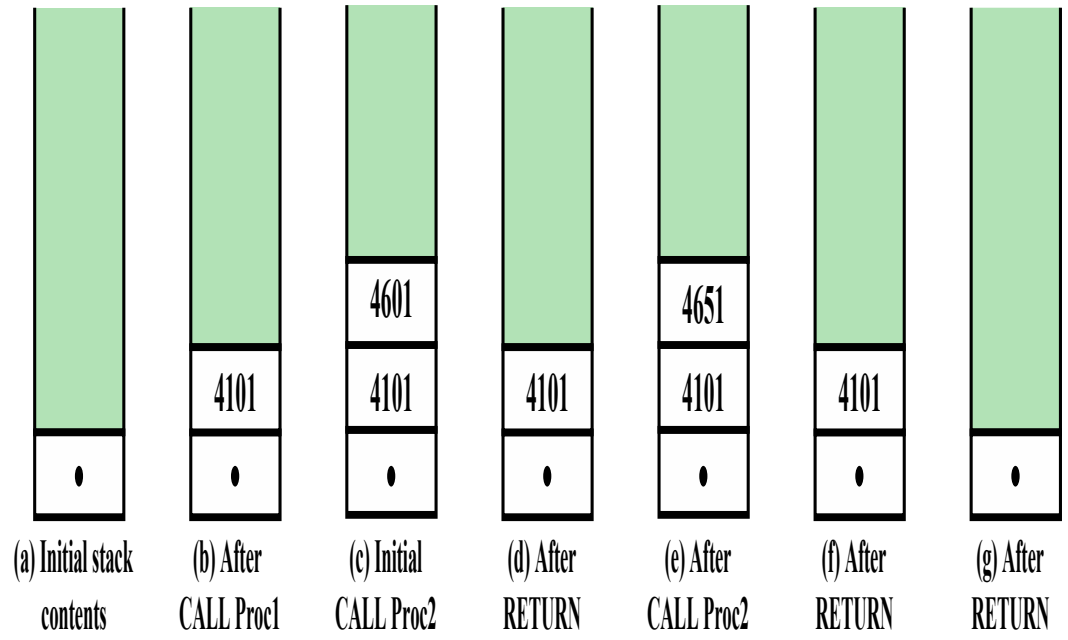
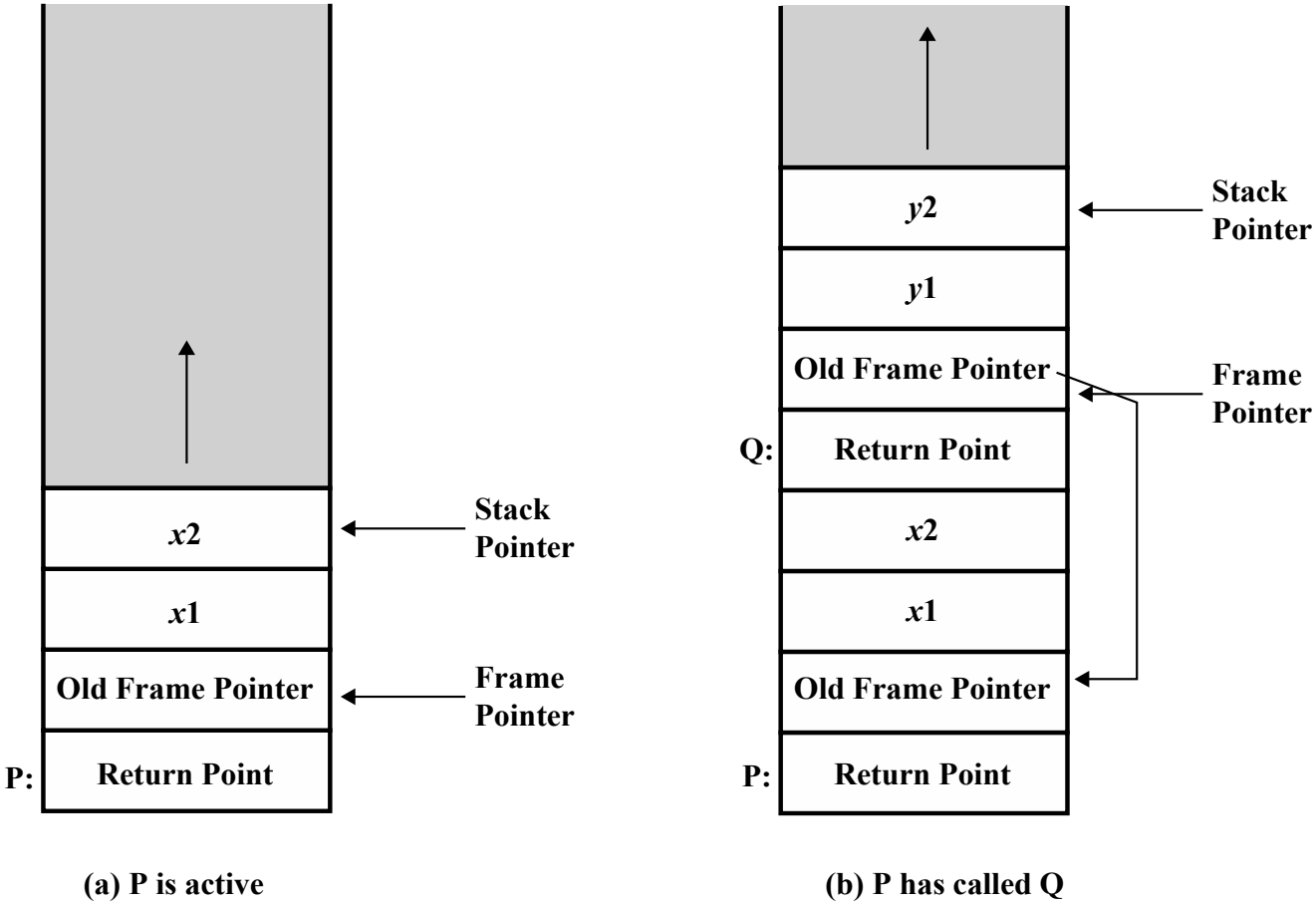


Figure 12.9 Use of Stack to Implement Nested Procedures of Figure 12.8

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

A more flexible approach to parameter passing is the stack. When the processor executes a call, it not only stacks the return address, it stacks parameters to be passed to the called procedure. The called procedure can access the parameters from the stack.



The example refers to procedure P in which the local variables *x1* and *x2* are declared, and procedure Q, which P can call and in which the local variables *y1* and *y2* are declared

Figure 12.10 Stack Frame Growth Using Sample Procedures P and Q

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

x86 Operation Types

- The x86 provides a complex array of operation types including a number of specialized instructions
- The intent was to provide tools for the compiler writer to produce optimized machine language translation of high-level language programs
- Provides four instructions to support procedure call/return:
  - CALL
  - ENTER
  - LEAVE
  - RETURN
- When a new procedure is called the following must be performed upon entry to the new procedure:
  - Push the return point on the stack
  - Push the current frame pointer on the stack
  - Copy the stack pointer as the new value of the frame pointer
  - Adjust the stack pointer to allocate a frame

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Table 12.8
x86 Status Flags

Status Bit	Name	Description
CF	Carry	Indicates carrying or borrowing out of the left-most bit position following an arithmetic operation. Also modified by some of the shift and rotate operations.
PF	Parity	Parity of the least-significant byte of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.
AF	Auxiliary Carry	Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation. Used in binary-coded decimal arithmetic.
ZF	Zero	Indicates that the result of an arithmetic or logic operation is 0.
SF	Sign	Indicates the sign of the result of an arithmetic or logic operation.
OF	Overflow	Indicates an arithmetic overflow after an addition or subtraction for twos complement arithmetic.

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Symbol	Condition Tested	Comment
A, NBE	CF=0 AND ZF=0	Above; Not below or equal (greater than, unsigned)
AE, NB, NC	CF=0	Above or equal; Not below (greater than or equal, unsigned); Not carry
B, NAE, C	CF=1	Below; Not above or equal (less than, unsigned); Carry set
BE, NA	CF=1 OR ZF=1	Below or equal; Not above (less than or equal, unsigned)
E, Z	ZF=1	Equal; Zero (signed or unsigned)
G, NLE	[(SF=1 AND OF=1) OR (SF=0 and OF=0)] AND [ZF=0]	Greater than; Not less than or equal (signed)
GE, NL	(SF=1 AND OF=1) OR (SF=0 AND OF=0)	Greater than or equal; Not less than (signed)
L, NGE	(SF=1 AND OF=0) OR (SF=0 AND OF=1)	Less than; Not greater than or equal (signed)
LE, NG	(SF=1 AND OF=0) OR (SF=0 AND OF=1) OR (ZF=1)	Less than or equal; Not greater than (signed)
NE, NZ	ZF=0	Not equal; Not zero (signed or unsigned)
NO	OF=0	No overflow
NS	SF=0	Not sign (not negative)
NP, PO	PF=0	Not parity; Parity odd
O	OF=1	Overflow
P	PF=1	Parity; Parity even
S	SF=1	Sign (negative)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Table 12.9

x86 Condition Codes for Conditional Jump and SETcc Instructions

(Table can be found on page 440 in the textbook.)

x86 Single-Instruction, Multiple-Data (SIMD) Instructions

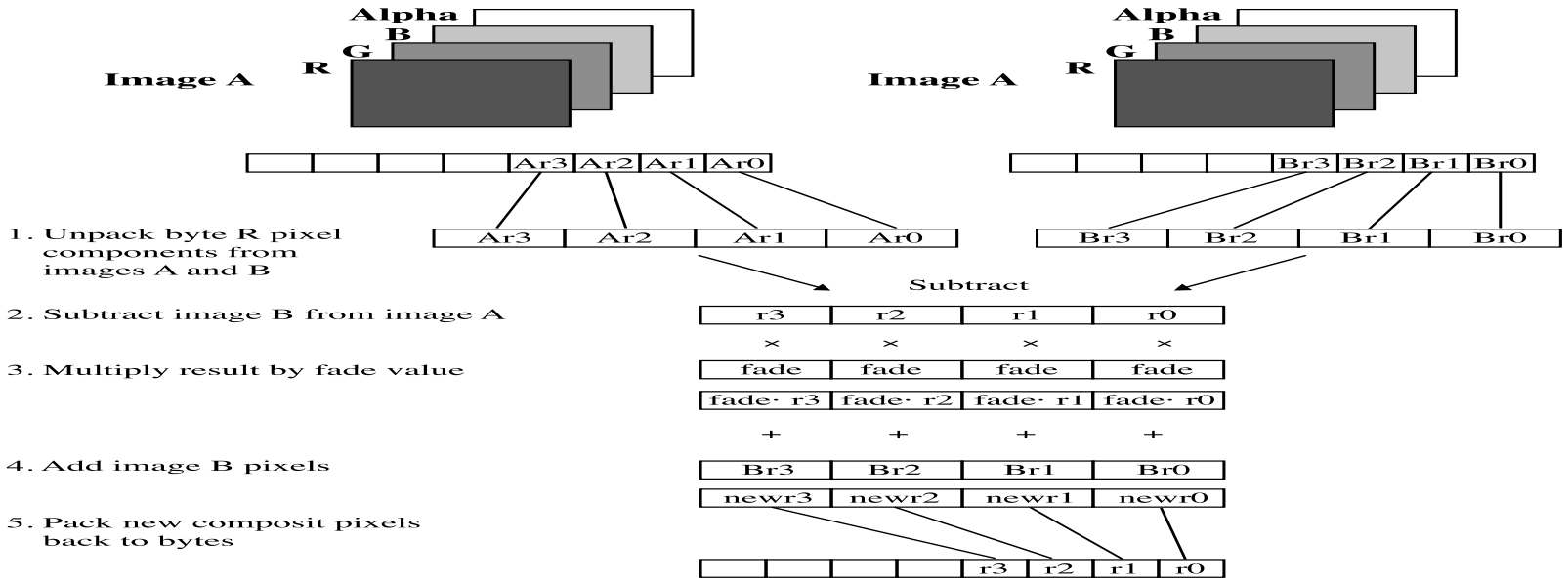
- 1996 Intel introduced MMX technology into its Pentium product line
  - MMX is a set of highly optimized instructions for multimedia tasks
- Video and audio data are typically composed of large arrays of small data types
- Three new data types are defined in MMX
  - Packed byte
  - Packed word
  - Packed doubleword
- Each data type is 64 bits in length and consists of multiple smaller data fields, each of which holds a fixed-point integer

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

To provide a feel for the use of MMX instructions, we look at an example, taken from [PELE97]. A common video application is the fade-out,fade-in effect, in which one scene gradually dissolves into another. Two images are combined with a weighted average:

Result_pixel = A_pixel * fade + B_pixel * (1 - fade)

This calculation is performed on each pixel position in A and B. If a series of video frames is produced while gradually changing the fade value from 1 to 0 (scaled appropriately for an 8-bit integer), the result is to fade from image A to image B. Figure 12.11 shows the sequence of steps required for one set of pixels.



MMX code sequence performing this operation:

```
pxor        mm7, mm7        ;zero out mm7
movq        mm3, fad_val    ;load fade value replicated 4 times
movd        mm0, imageA     ;load 4 red pixel components from image A
movd        mm1, imageB     ;load 4 red pixel components from image B
punpckblw   mm0, mm7        ;unpack 4 pixels to 16 bits
punpckblw   mm1, mm7        ;unpack 4 pixels to 16 bits
psubw       mm0, mm1        ;subtract image B from image A
pmulhw      mm0, mm3        ;multiply the subtract result by fade values
padddw      mm0, mm1        ;add result to image B
packuswb    mm0, mm7        ;pack 16-bit results back to bytes
```

Figure 12.11 Image Compositing on Color Plane Representation

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Table 12.10

MMX Instruction Set

Category	Instruction	Description
Arithmetic	PADD [B, W, D]	Parallel add of packed eight bytes, four 16-bit words, or two 32-bit doublewords, with wraparound.
	PADDS [B, W]	Add with saturation.
	PADDUS [B, W]	Add unsigned with saturation
	PSUB [B, W, D]	Subtract with wraparound.
	PSUBS [B, W]	Subtract with saturation.
	PSUBUS [B, W]	Subtract unsigned with saturation
	PMULHW	Parallel multiply of four signed 16-bit words, with high-order 16 bits of 32-bit result chosen.
	PMULLW	Parallel multiply of four signed 16-bit words, with low-order 16 bits of 32-bit result chosen.
	PMADDWD	Parallel multiply of four signed 16-bit words; add together adjacent pairs of 32-bit results.
Comparison	PCMPEQ [B, W, D]	Parallel compare for equality; result is mask of 1s if true or 0s if false.
	PCMPGT [B, W, D]	Parallel compare for greater than; result is mask of 1s if true or 0s if false.
Conversion	PACKUSWB	Pack words into bytes with unsigned saturation.
	PACKSS [WB, DW]	Pack words into bytes, or doublewords into words, with signed saturation.
	PUNPCKH [BW, WD, DQ]	Parallel unpack (interleaved merge) high-order bytes, words, or doublewords from MMX register.
	PUNPCKL [BW, WD, DQ]	Parallel unpack (interleaved merge) low-order bytes, words, or doublewords from MMX register.
Logical	PAND	64-bit bitwise logical AND
	PNDN	64-bit bitwise logical AND NOT
	POR	64-bit bitwise logical OR
	PXOR	64-bit bitwise logical XOR
Shift	PSLL [W, D, Q]	Parallel logical left shift of packed words, doublewords, or quadword by amount specified in MMX register or immediate value.
	PSRL [W, D, Q]	Parallel logical right shift of packed words, doublewords, or quadword.
	PSRA [W, D]	Parallel arithmetic right shift of packed words, doublewords, or quadword.
Data Transfer	MOV [D, Q]	Move doubleword or quadword to/from MMX register.
State Mgt	EMMS	Empty MMX state (empty FP registers tag bits).

Note: If an instruction supports multiple data types [byte (B), word (W), doubleword (D), quadword (Q)], the data types are indicated in brackets.

Most of the instructions involve parallel operation on bytes, words, or doublewords. For example, the PSLLW instruction performs a left logical shift separately on each of the four words in the packed word operand; the PADDB instruction takes packed byte operands as input and performs parallel additions on each byte position independently to produce a packed byte output.

(Table can be found on page 442 in the textbook.)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

ARM Operation Types

- Load and store instructions
- Branch instructions
- Data-processing instructions
- Multiply instructions
- Parallel addition and subtraction instructions
- Extend instructions
- Status register access instructions

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Code	Symbol	Condition Tested	Comment
0000	EQ	Z = 1	Equal
0001	NE	Z = 0	Not equal
0010	CS/HS	C = 1	Carry set/unsigned higher or same
0011	CC/LO	C = 0	Carry clear/unsigned lower
0100	MI	N = 1	Minus/negative
0101	PL	N = 0	Plus/positive or zero
0110	VS	V = 1	Overflow
0111	VC	V = 0	No overflow
1000	HI	C = 1 AND Z = 0	Unsigned higher
1001	LS	C = 0 OR Z = 1	Unsigned lower or same
1010	GE	N = V [(N = 1 AND V = 1) OR (N = 0 AND V = 0)	Signed greater than or equal
1011	LT	N ≠ V [(N = 1 AND V = 0) OR (N = 0 AND V = 1)]	Signed less than
1100	GT	(Z = 0) AND (N = V)	Signed greater than
1101	LE	(Z = 1) OR (N ≠ V)	Signed less than or equal
1110	AL	—	Always (unconditional)
1111	—	—	This instruction can only be executed unconditionally

Table 12.11

ARM Conditions for Conditional Instruction Execution

The ARM architecture defines four condition flags that are stored in the program status register: N, Z, C, and V (Negative, Zero, Carry and Overflow

There are two unusual aspects to the use of condition codes in ARM:

- All instructions, not just branch instructions, include a condition code field, which means that virtually all instructions may be conditionally executed. Any combination of flag settings except 1110 or 1111 in an instruction's condition code field signifies that the instruction will be executed only if the condition is met.

- All data processing instructions (arithmetic, logical) includes an S bit that signifies whether the instruction updates the condition flags.

(Table can be found on Page 445 in the textbook.)

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.

Summary

Chapter 12

Instruction Sets: Characteristics and Functions

- Machine instruction characteristics
  - Elements of a machine instruction
  - Instruction representation
  - Instruction types
  - Number of addresses
  - Instruction set design
- Types of operands
  - Numbers
  - Characters
  - Logical data
- Intel x86 and ARM data types
- Types of operations
  - Data transfer
  - Arithmetic
  - Logical
  - Conversion
  - Input/output
  - System control
  - Transfer of control
- Intel x86 and ARM operation types

© 2016 Pearson Education, Inc., Hoboken, NJ. All rights reserved.