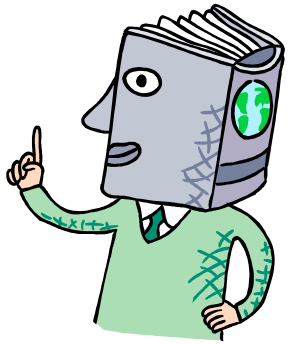
Lecture B - 04

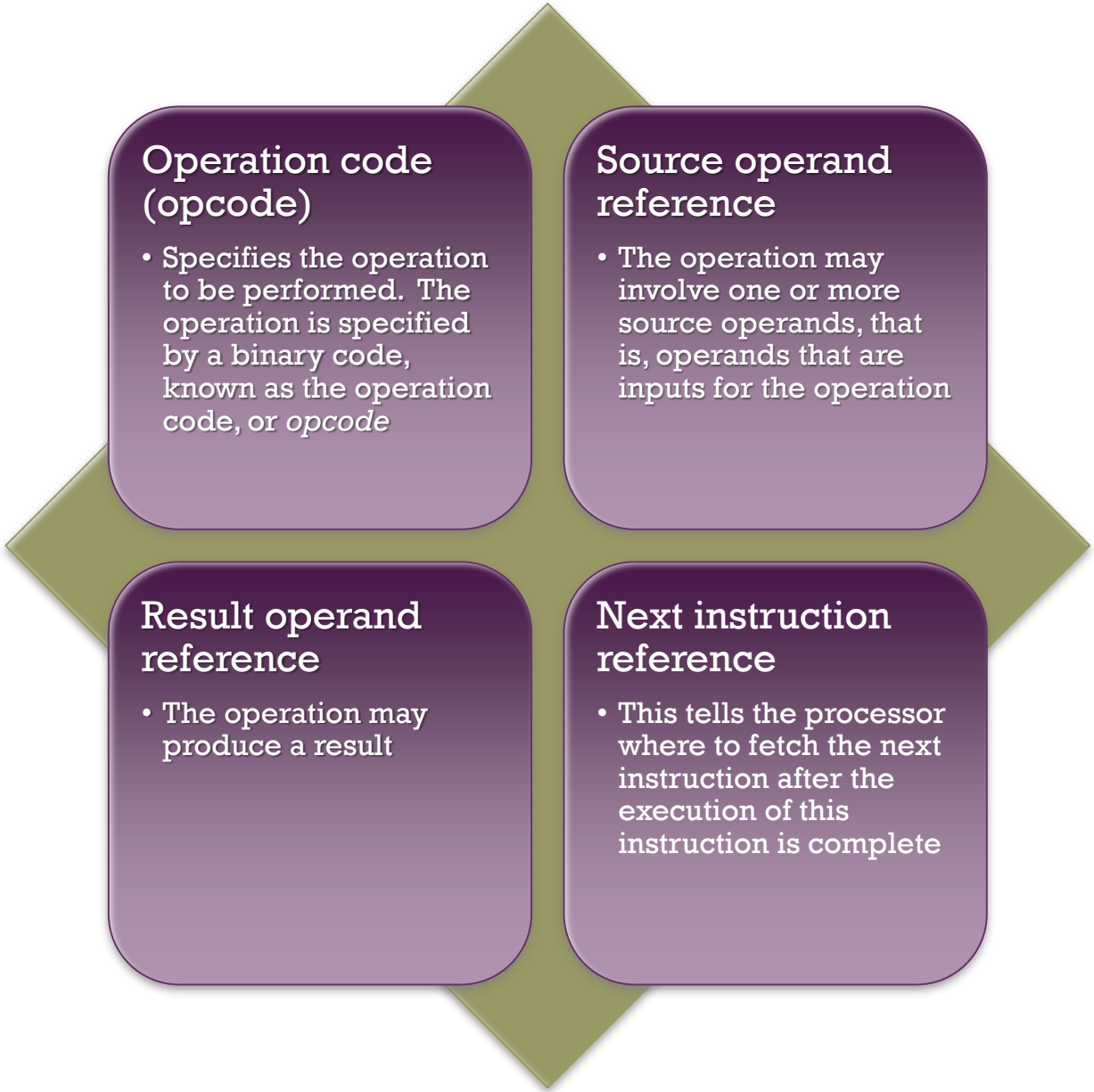
Instruction Sets Architecture and Design

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



Instruction Cycle State Diagram

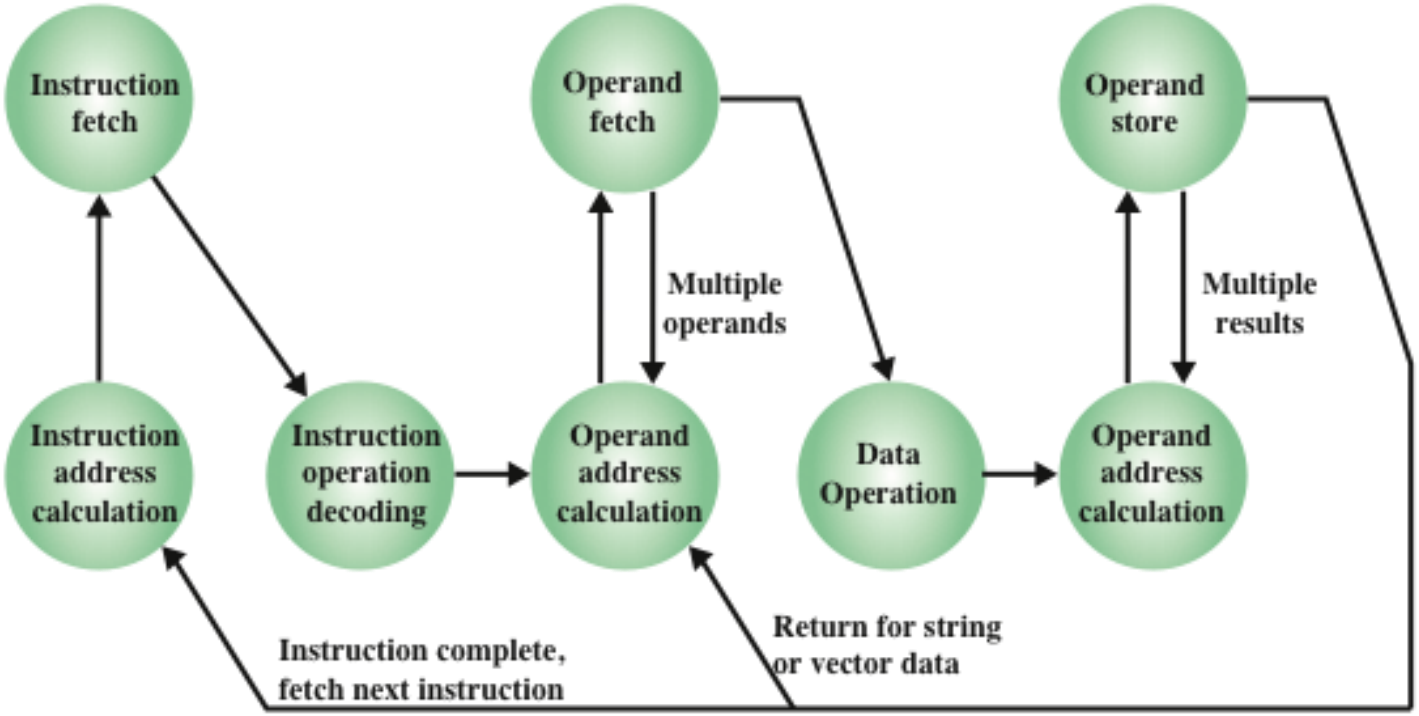


Figure 12.1 Instruction Cycle State Diagram

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

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

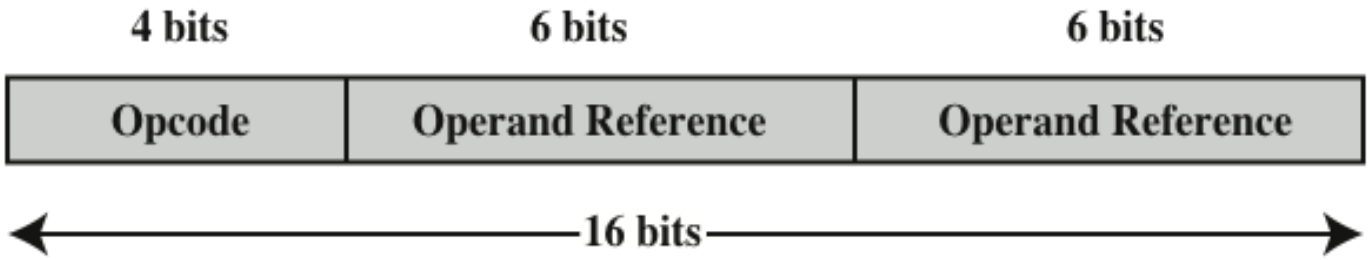


Figure 12.2 A Simple Instruction Format

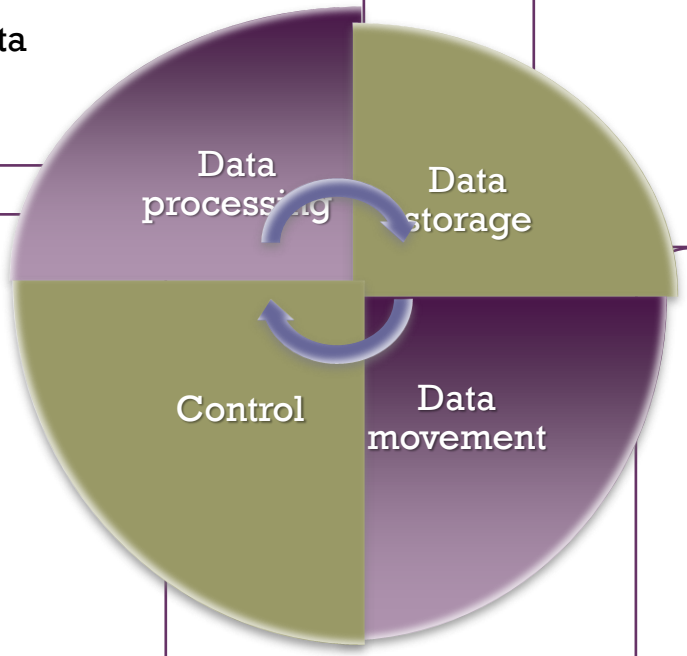
Instruction Representation

- 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

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

Number of Addresses

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

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

Example a

- C = [(T - (N + P * K)) - (A * B)]

One-Address	Two-Address	Three-Address
LOAD K **MUL P** **ADD N** ~~**SUB T**~~ **STORE R_1** **LOAD A** **MUL B** ~~**SUB R_1**~~ **STORE C**	**MOV R_1, K** **MUL R_1 ,P** **ADD R_1 , N** ~~**SUB R_1, T**~~ **MOV R_2, B** **MUL R_2, A** ~~**SUB R_1, R_2**~~ ~~**MOVE C, R_1**~~	**MUL R_1, P, K** **ADD R_1, N, R_1** **SUB C, T, R_1** **MUL R_2, A, B** **SUB C , C , R_2**

STORE R2
LOAD T
SUB R2

SUB T, R1

STORE R4
LOAD R1
SUB R4

SUB T, R2
MOVE C,T

Example b

- Y = (A-B) / (C+D*E)

Zero address Instruction	Comment
PUSH A	A → TOS
PUSH B	B → TOS
SUB	A-B → TOS
PUSH C	C → TOS
PUSH D	D →TOS
PUSH E	E → TOS
MUL	D*E → TOS
ADD	C+ D*E → TOS
DIV	(A-B)/(C+(D*E)) → TOS
POP Y	TOS → Y

Try to do one address, two address and three address

Instruction Set Design

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

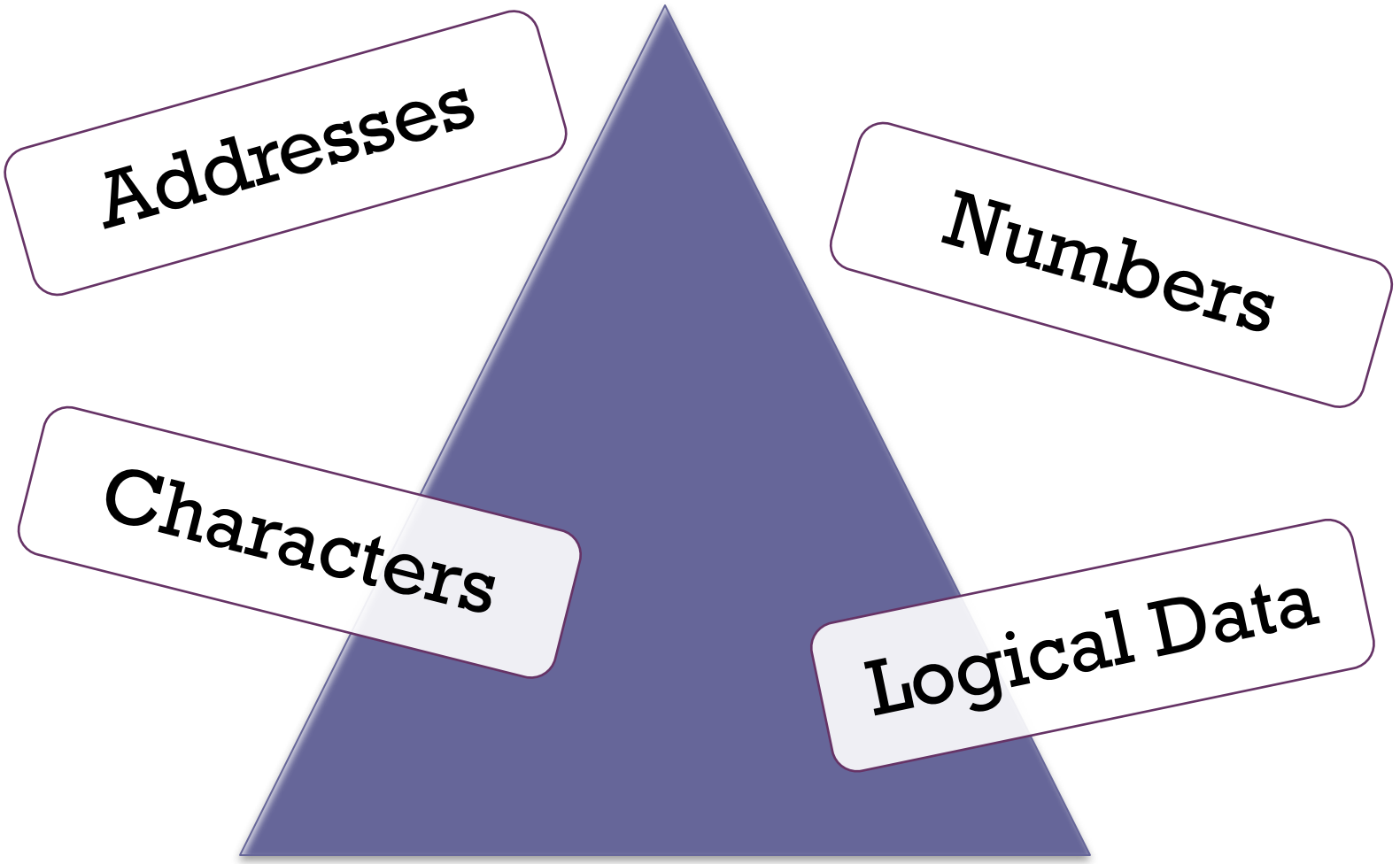
Defines many of the functions performed by the processor

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

Types of Operands



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 (e.g., the code for 246 is 0000 0010 0100 0110)

Characters

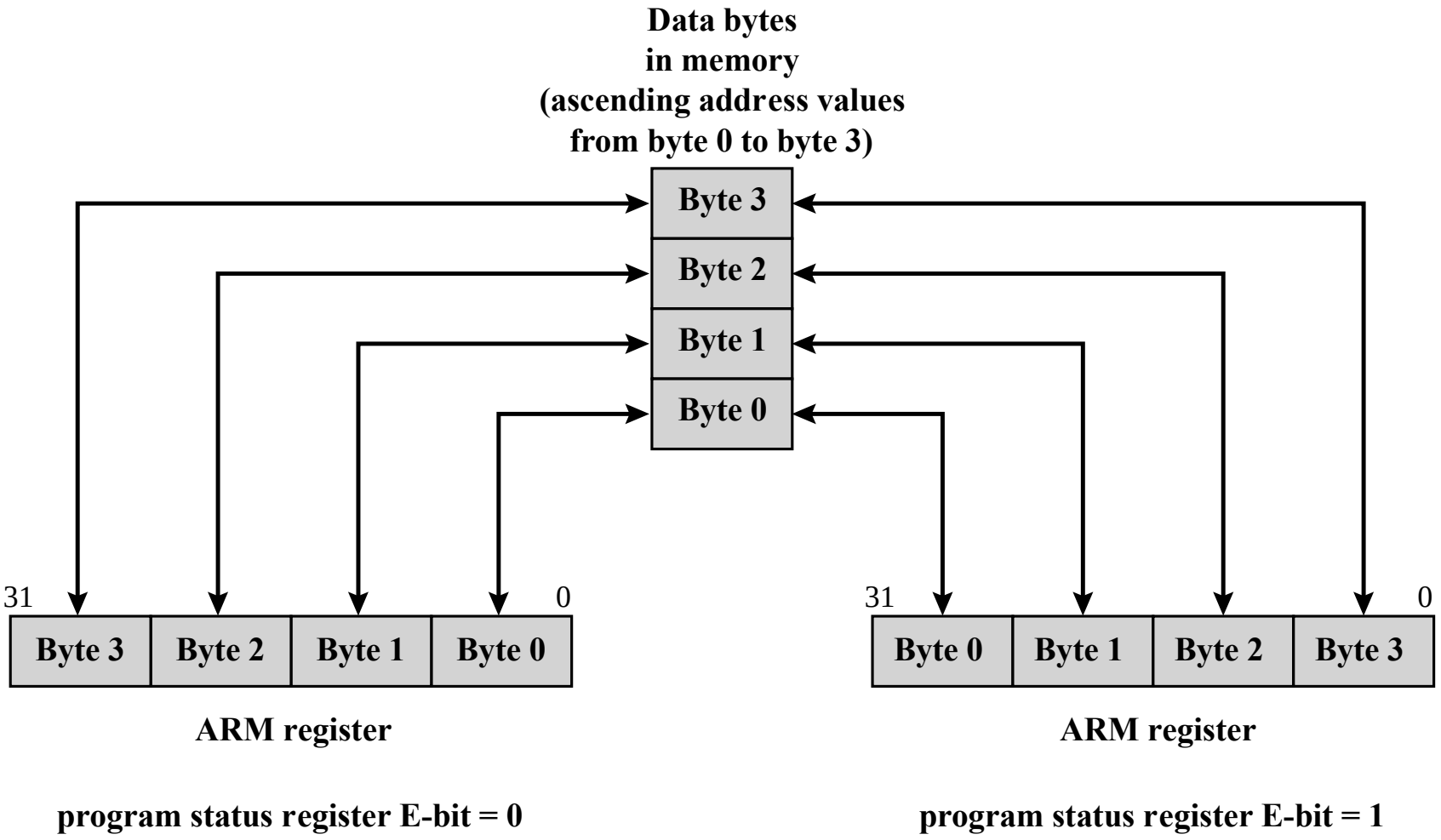
- A common form of data is text or character strings
- Textual data in character form cannot be easily stored or transmitted by data processing and communications systems because they are designed for binary data
- Most commonly used character code is the International Reference Alphabet (IRA)
  - Referred 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

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

The same data are treated sometimes as logical and other times as numerical or text. The "type" of a unit of data is determined by the operation being performed on it. While this is not normally the case in high-level languages, it is almost always the case with machine language.

ARM Endian Support



Common Instruction Set Operations (page 1 of 2)

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 (page 2 of 2)

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)

Processor Actions for Various Types of Operations

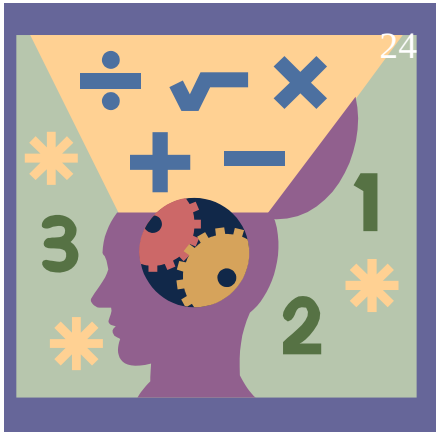
Data Transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write
Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes 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

Data Transfer

Most fundamental type of machine 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



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

Logical

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

Basic Logical Operations

Shift and Rotate Operations

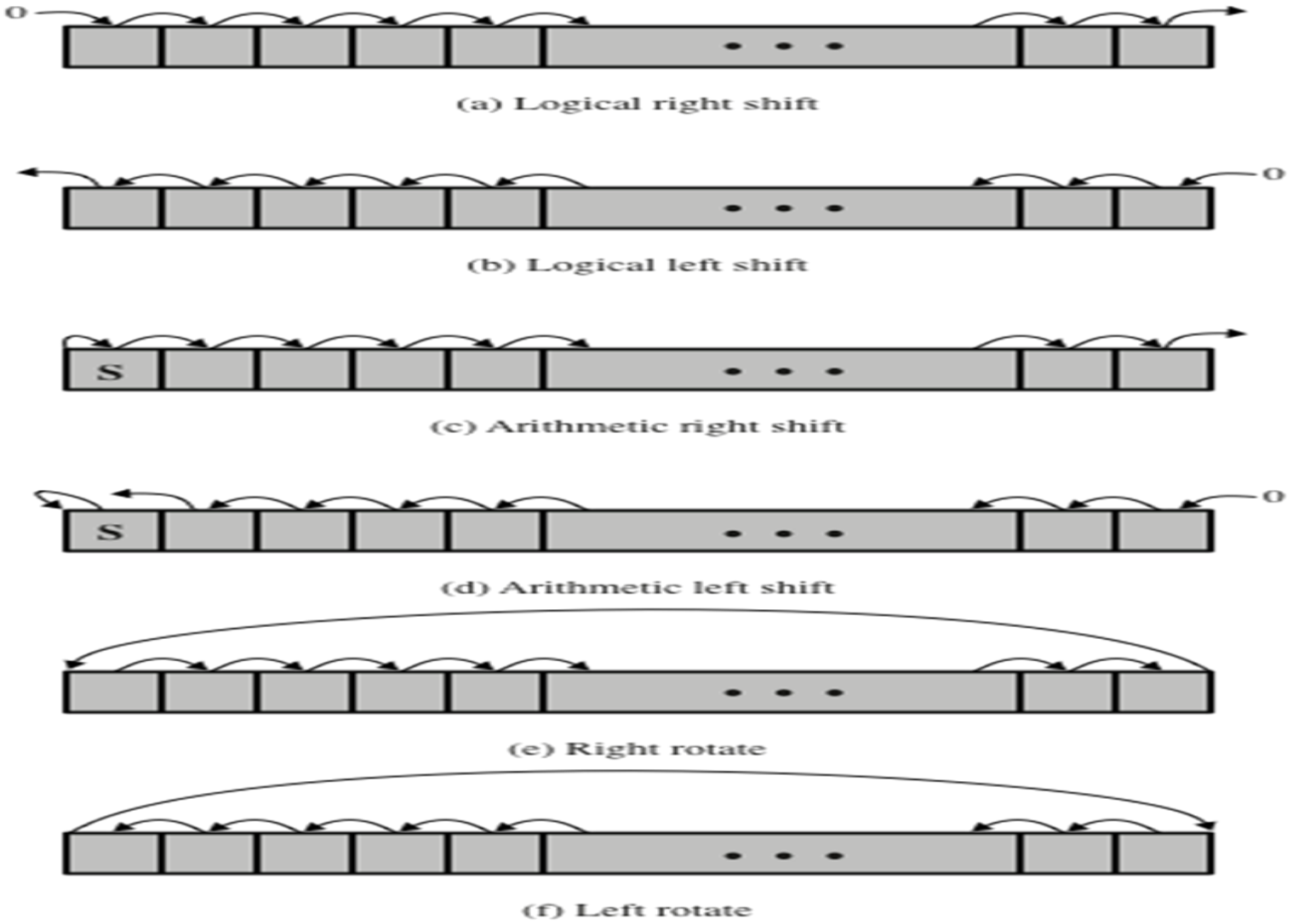


Figure 12.6 Shift and Rotate Operations

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

Examples of Shift and Rotate Operations

Conversion

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

An example is converting from decimal to binary

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

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

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

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

Branch Instruction

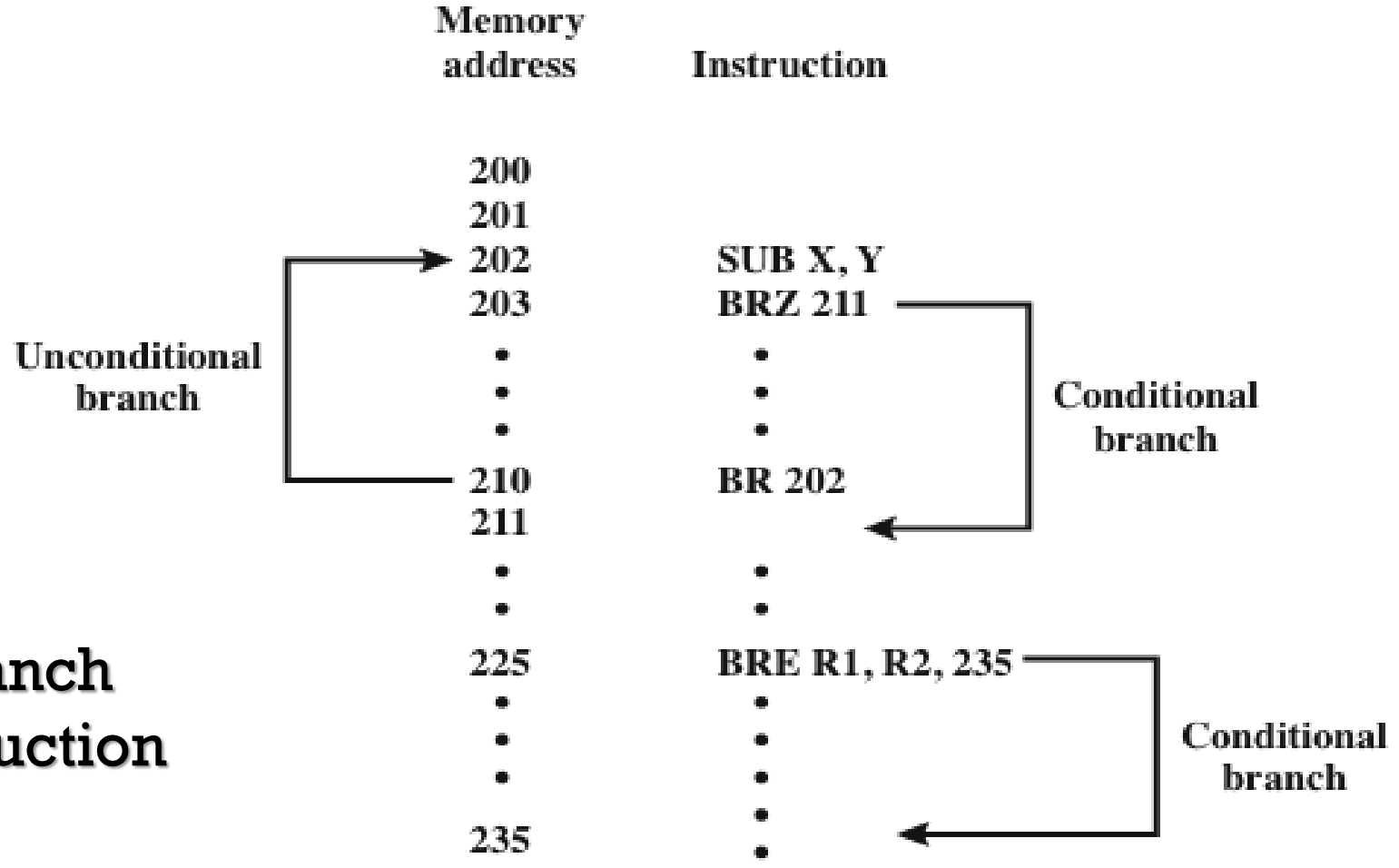


Figure 12.7 Branch Instructions

Skip Instructions

Includes an implied address

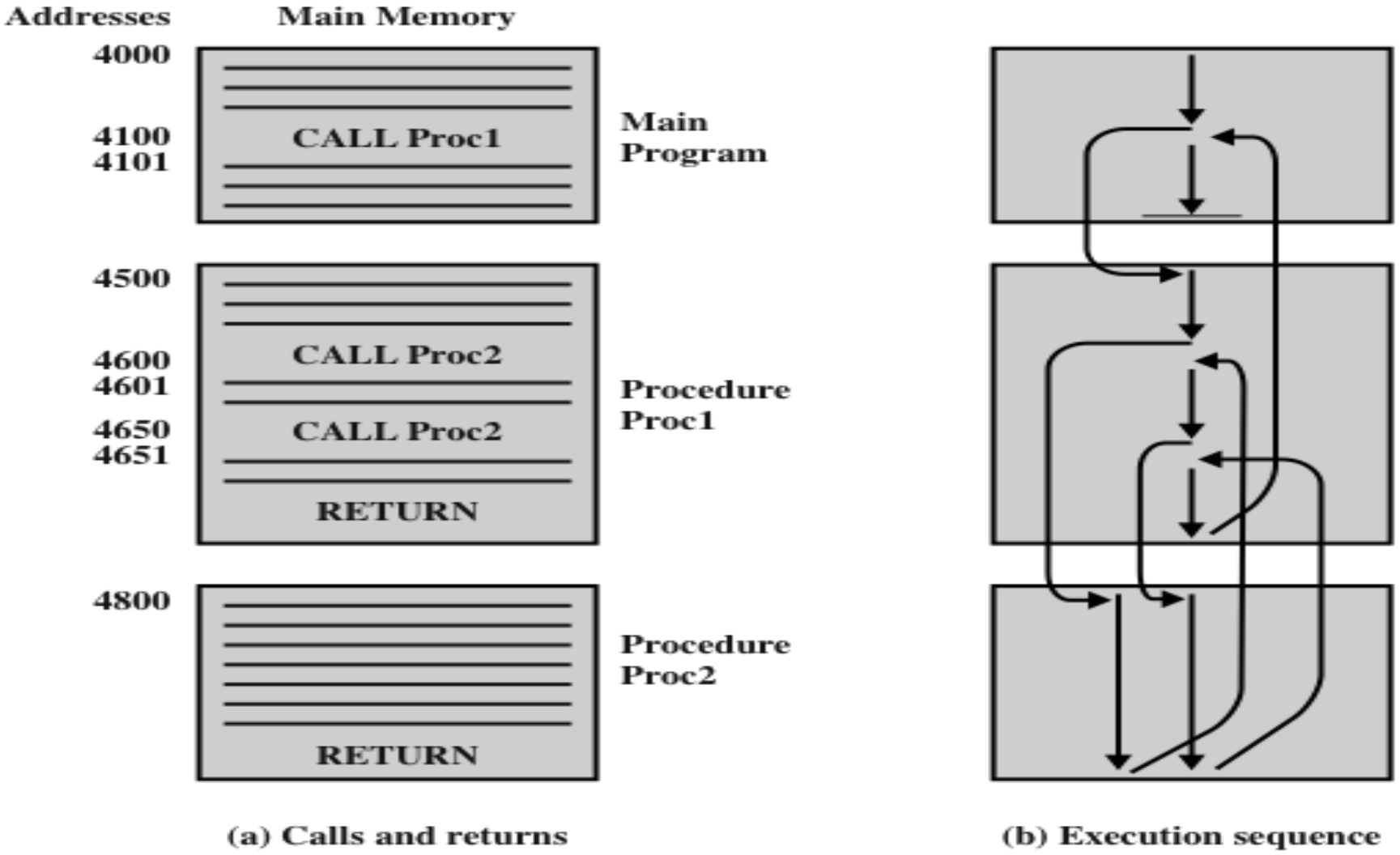
Typically implies that one instruction be skipped, thus the implied address equals the address of the next instruction plus one instruction length

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

Procedure Call Instructions

- 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 - modularity is the degree to which a system's components may be separated and recombined, often with the benefit of flexibility and variety in use
- 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



Nested Procedures

Figure 12.8 Nested Procedures

Use of Stack to Implement Nested Procedures

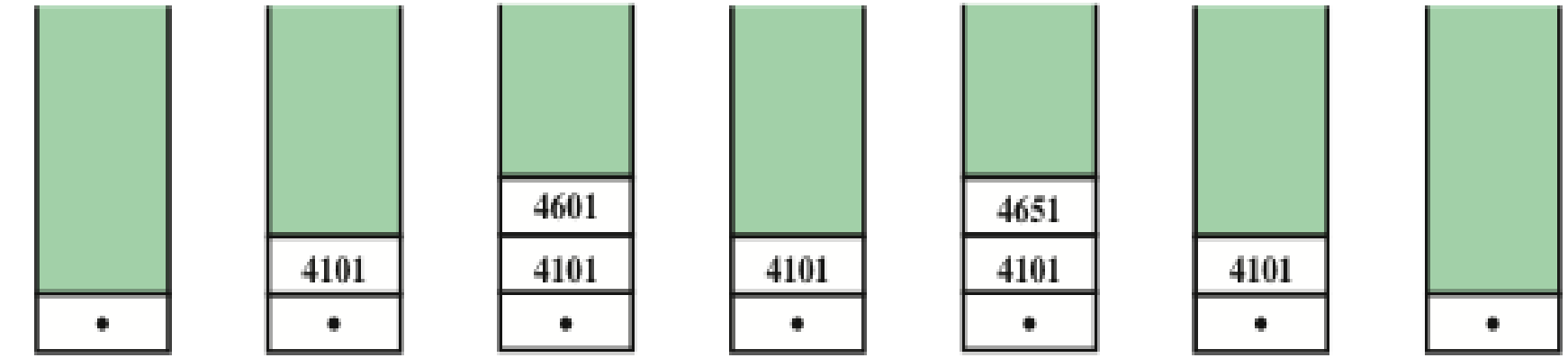


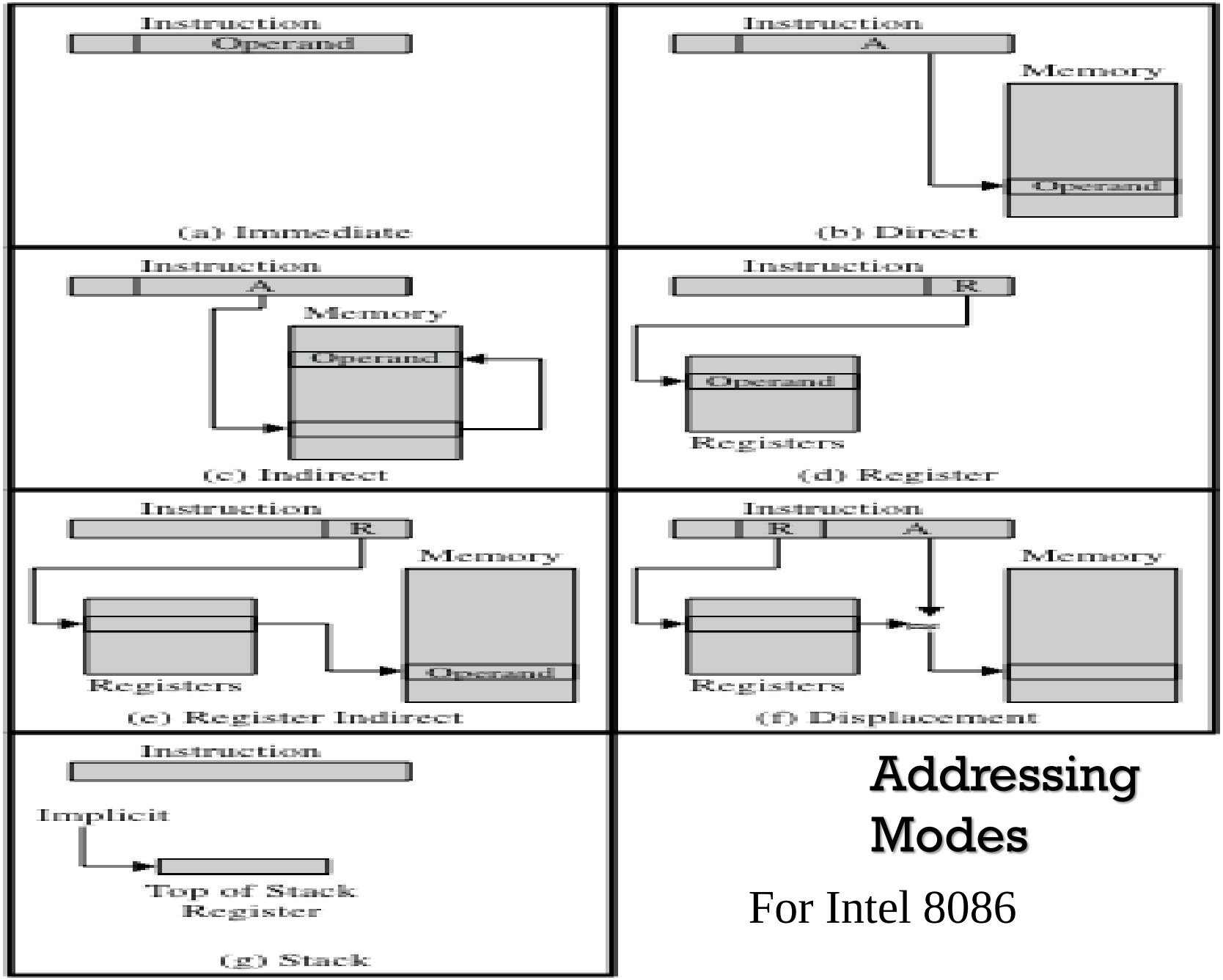
Figure 12.9 Use of Stack to Implement Nested Procedures of Figure 12.8

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

ARM Conditions for Conditional Instruction Execution

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



Addressing Modes

For Intel 8086

Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Immediate Addressing

- Simplest form of addressing
- Operand = A
- This mode can be used to define and use constants or set initial values of variables
  - Typically the number will be stored in twos complement form
  - The leftmost bit of the operand field is used as a sign bit
- Advantage:
  - no memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle
- Disadvantage:
  - The size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the word length

Direct Addressing

Address field contains the effective address of the operand

Effective address (EA) = address field (A)

Was common in earlier generations of computers

Requires only one memory reference and no special calculation

Limitation is that it provides only a limited address space

Indirect Addressing

- Reference to the address of a word in memory which contains a full-length address of the operand
- EA = (A)
  - Parentheses are to be interpreted as meaning *contents of*
- Advantage:
  - For a word length of *N* an address space of 2^N is now available
- Disadvantage:
  - Instruction execution requires two memory references to fetch the operand
    - One to get its address and a second to get its value
- A rarely used variant of indirect addressing is multilevel or cascaded indirect addressing
  - EA = (. . . (A) . . .)
  - Disadvantage is that three or more memory references could be required to fetch an operand

Register Addressing

- Address field refers to a register rather than a main memory address
- EA = R
- Advantages:
  - Only a small address field is needed in the instruction
  - No time-consuming memory references are required
- Disadvantage:
  - The address space is very limited

Register Indirect Addressing

- Analogous to indirect addressing
  - The only difference is whether the address field refers to a memory location or a register
- EA = (R)
- Address space limitation of the address field is overcome by having that field refer to a word-length location containing an address
- Uses one less memory reference than indirect addressing

Displacement Addressing

- Combines the capabilities of direct addressing and register indirect addressing
- EA = A + (R)
- Requires that the instruction have two address fields, at least one of which is explicit
  - The value contained in one address field (value = A) is used directly
  - The other address field refers to a register whose contents are added to A to produce the effective address (EA)
- Most common uses:
  - Relative addressing
  - Base-register addressing
  - Indexing

Relative Addressing

- The implicitly referenced register is the program counter (PC)
  - The next instruction address is added to the address field to produce the EA
  - Typically the address field is treated as a twos complement number for this operation
  - Thus the effective address is a displacement relative to the address of the instruction
- Exploits the concept of locality
- Saves address bits in the instruction if most memory references are relatively near to the instruction being executed

Base-Register Addressing

- The referenced register contains a main memory address and the address field contains a displacement from that address
- The register reference may be explicit or implicit
- Exploits the locality of memory references
- Convenient means of implementing segmentation
- In some implementations a single segment base register is employed and is used implicitly
- In others the programmer may choose a register to hold the base address of a segment and the instruction must reference it explicitly

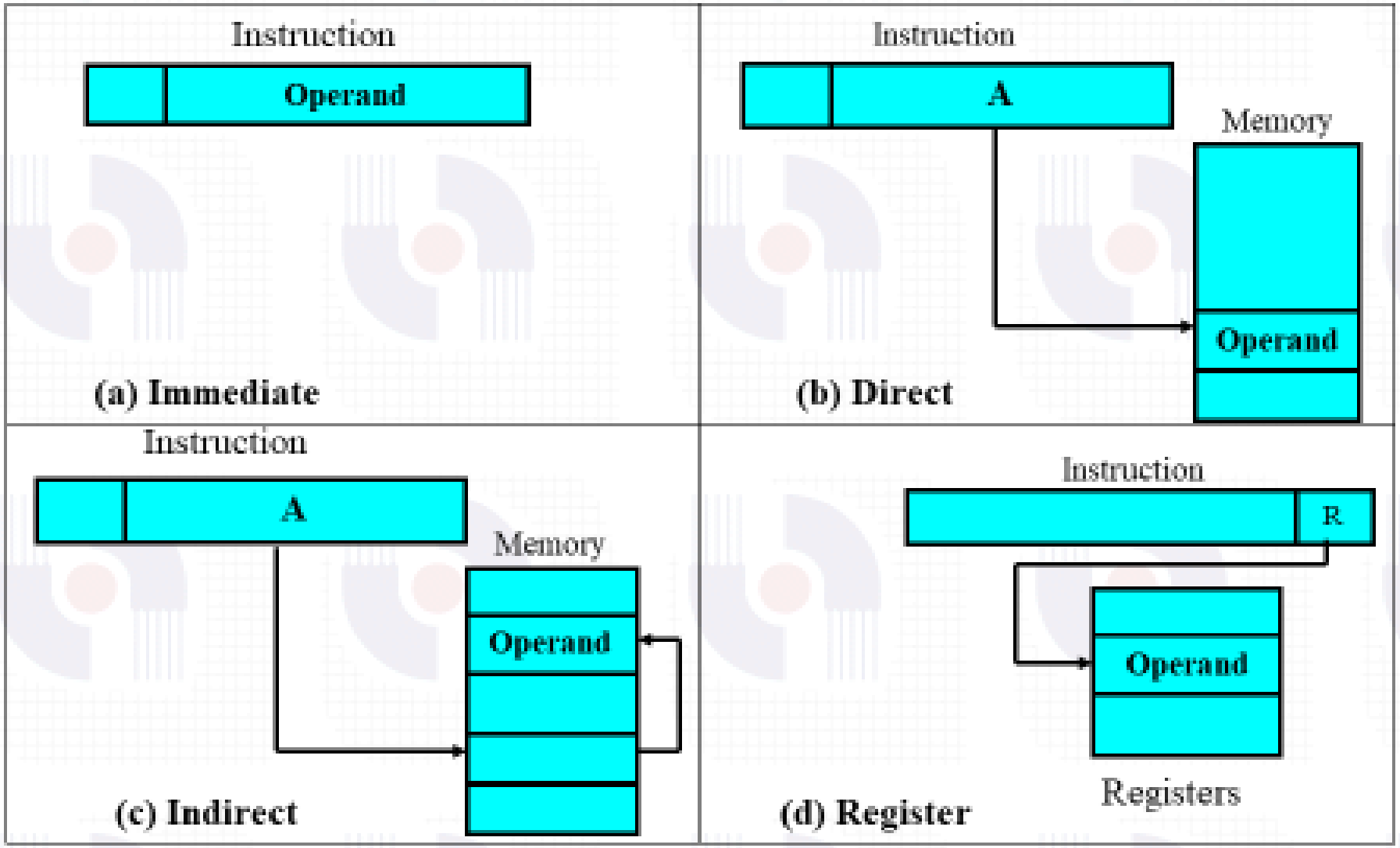
Indexed Addressing

- The address field references a main memory address and the referenced register contains a positive displacement from that address
- The method of calculating the EA is the same as for base-register addressing
- An important use is to provide an efficient mechanism for performing iterative operations
- Autoindexing
  - Automatically increment or decrement the index register after each reference to it
  - EA = A + (R)
  - (R) ← (R) + 1
- Postindexing
  - Indexing is performed after the indirection
  - EA = (A) + (R)
- Preindexing
  - Indexing is performed before the indirection
  - EA = ((A + (R))

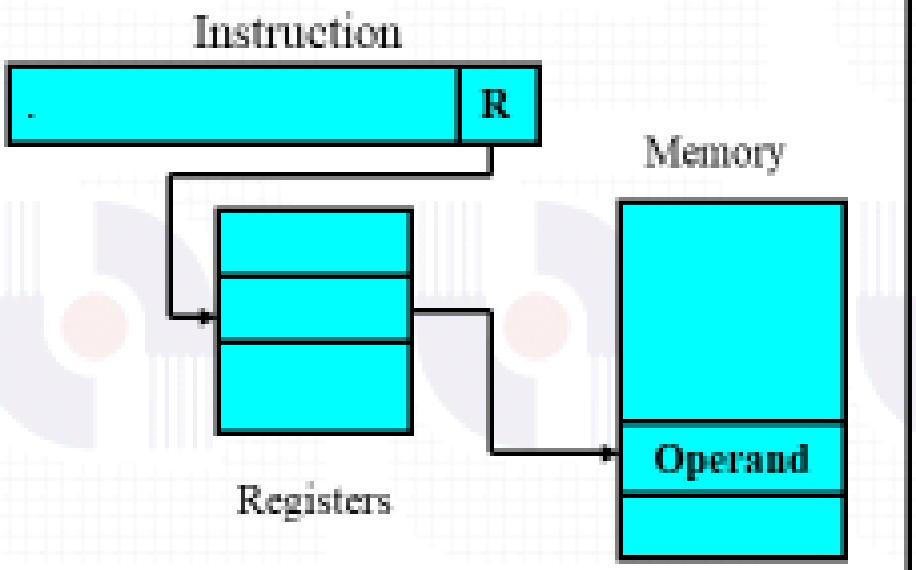
Stack Addressing

- A stack is a linear array of locations
  - Sometimes referred to as a *pushdown list* or *last-in-first-out queue*
- A stack is a reserved block of locations
  - Items are appended to the top of the stack so that the block is partially filled
- Associated with the stack is a pointer whose value is the address of the top of the stack
  - The stack pointer is maintained in a register
  - Thus references to stack locations in memory are in fact register indirect addresses
- Is a form of implied addressing
- The machine instructions need not include a memory reference but implicitly operate on the top of the stack

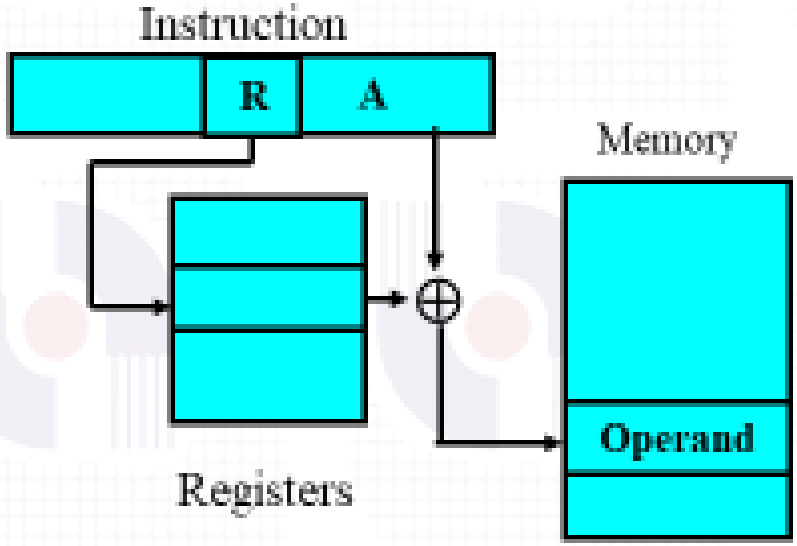
Addressing Modes



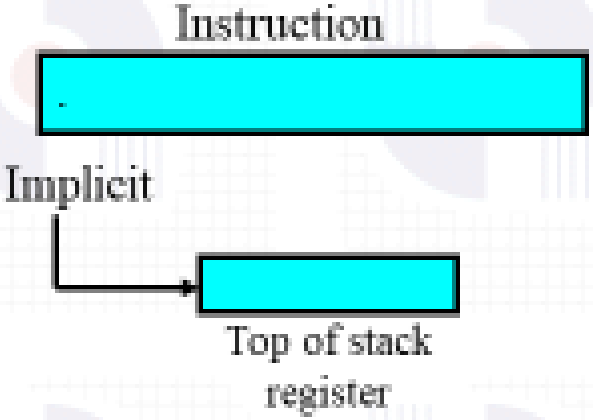
Addressing Modes



(e) Register Indirect



(f) Displacement



(g) Stack

A : Contents of an address field in the instruction

R: contents of an address field in the instruction that refers to a register

EA: Actual (Effective) address of the location containing the referenced operand

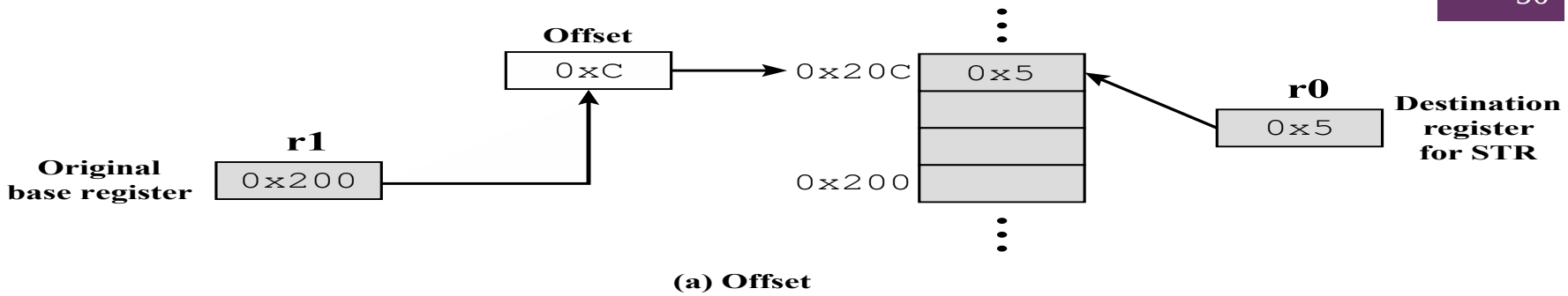
(X) : content of memory location X or register X

ARM Data Processing Instruction Addressing and Branch Instructions

- Data processing instructions
  - Use either register addressing or a mixture of register and immediate addressing
  - For register addressing the value in one of the register operands may be scaled using one of the five shift operators

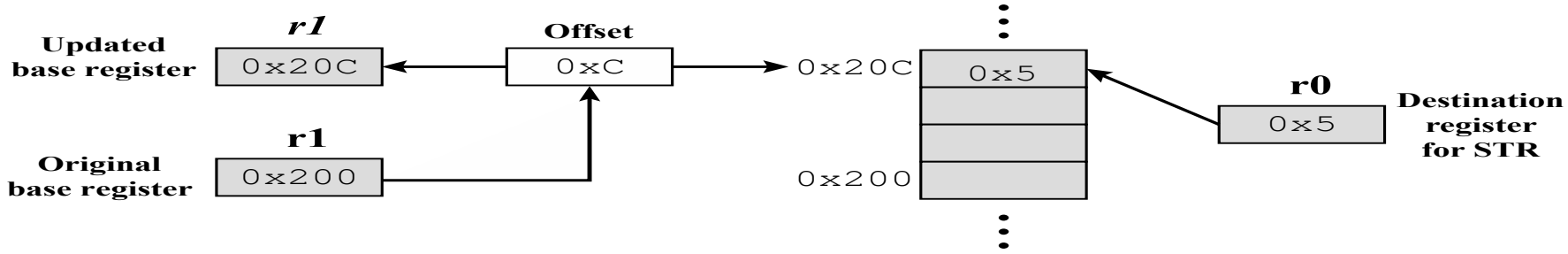
STR R0, [R1, #0X0C]

```
STRB r0, [r1, #12]
```



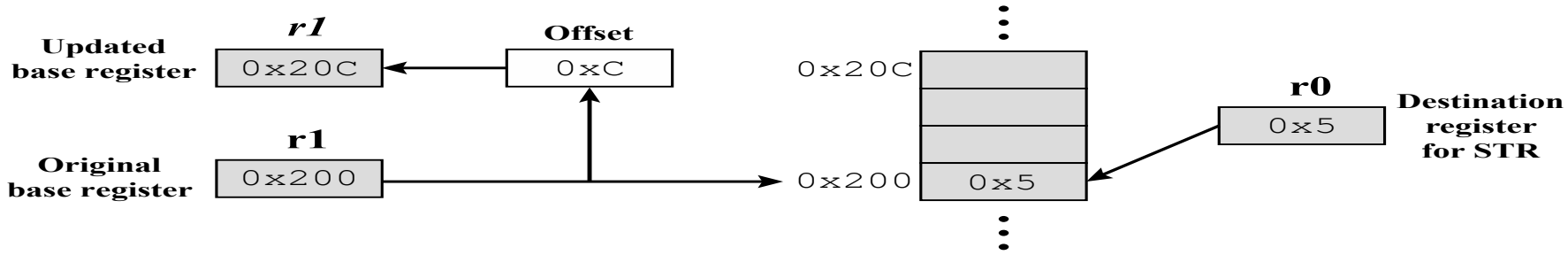
(a) Offset

```
STRB r0, [r1, #12]!
```



(b) Preindex

```
STRB r0, [r1], #12
```



(c) Postindex

Figure 13.3 ARM Indexing Methods

```
LDMxx r10, {r0, r1, r4}
STMxx r10, {r0, r1, r4}
```

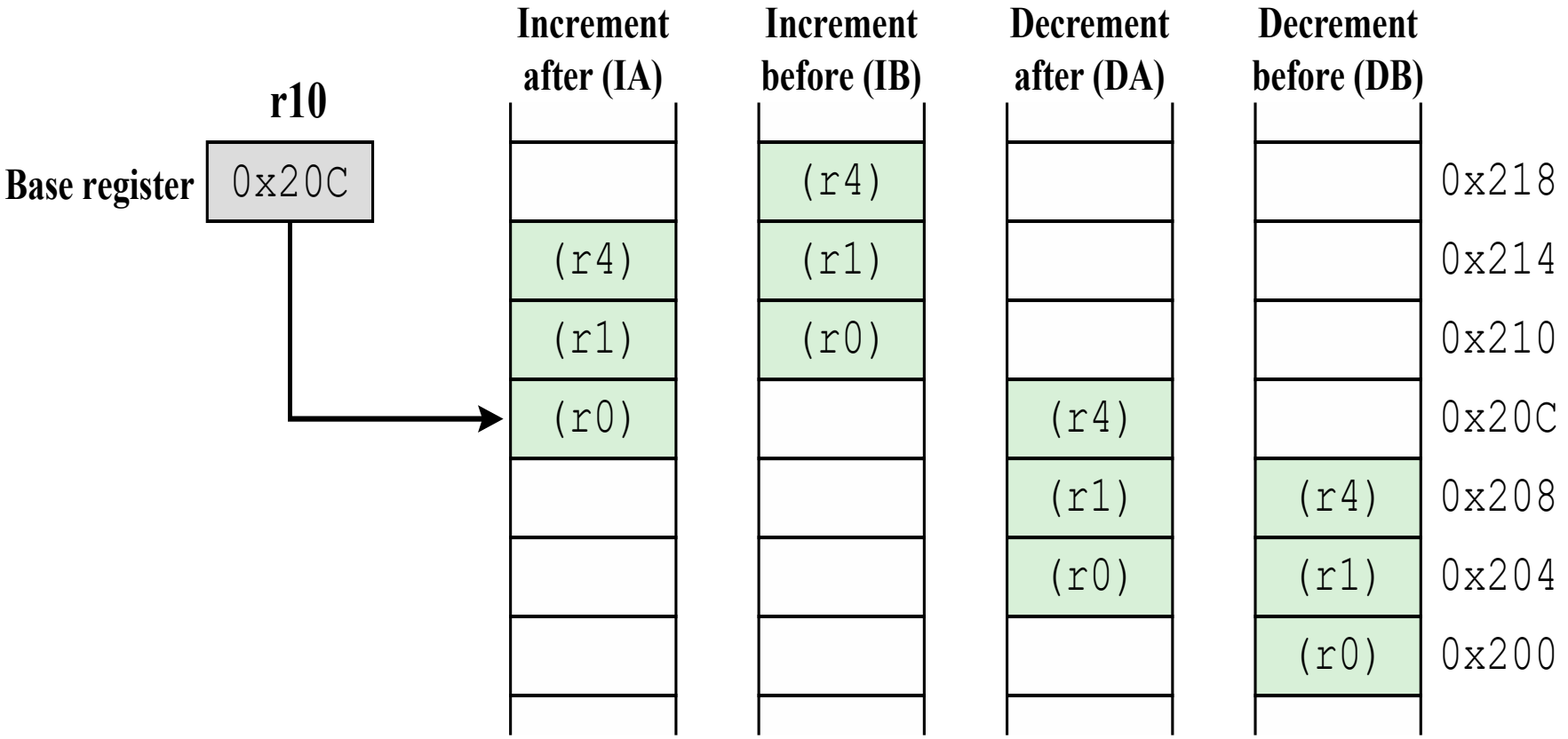


Figure 13.4 ARM Load/Store Multiple Addressing

Instruction Formats

Define the layout of the bits of an instruction, in terms of its constituent fields

Must include an opcode and, implicitly or explicitly, indicate the addressing mode for each operand

For most instruction sets more than one instruction format is used

Instruction Length

- Most basic design issue
- Affects, and is affected by:
  - Memory size
  - Memory organization
  - Bus structure
  - Processor complexity
  - Processor speed
- Should be equal to the memory-transfer length or one should be a multiple of the other
- Should be a multiple of the character length, which is usually 8 bits, and of the length of fixed-point numbers

Allocation of Bits

- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address granularity

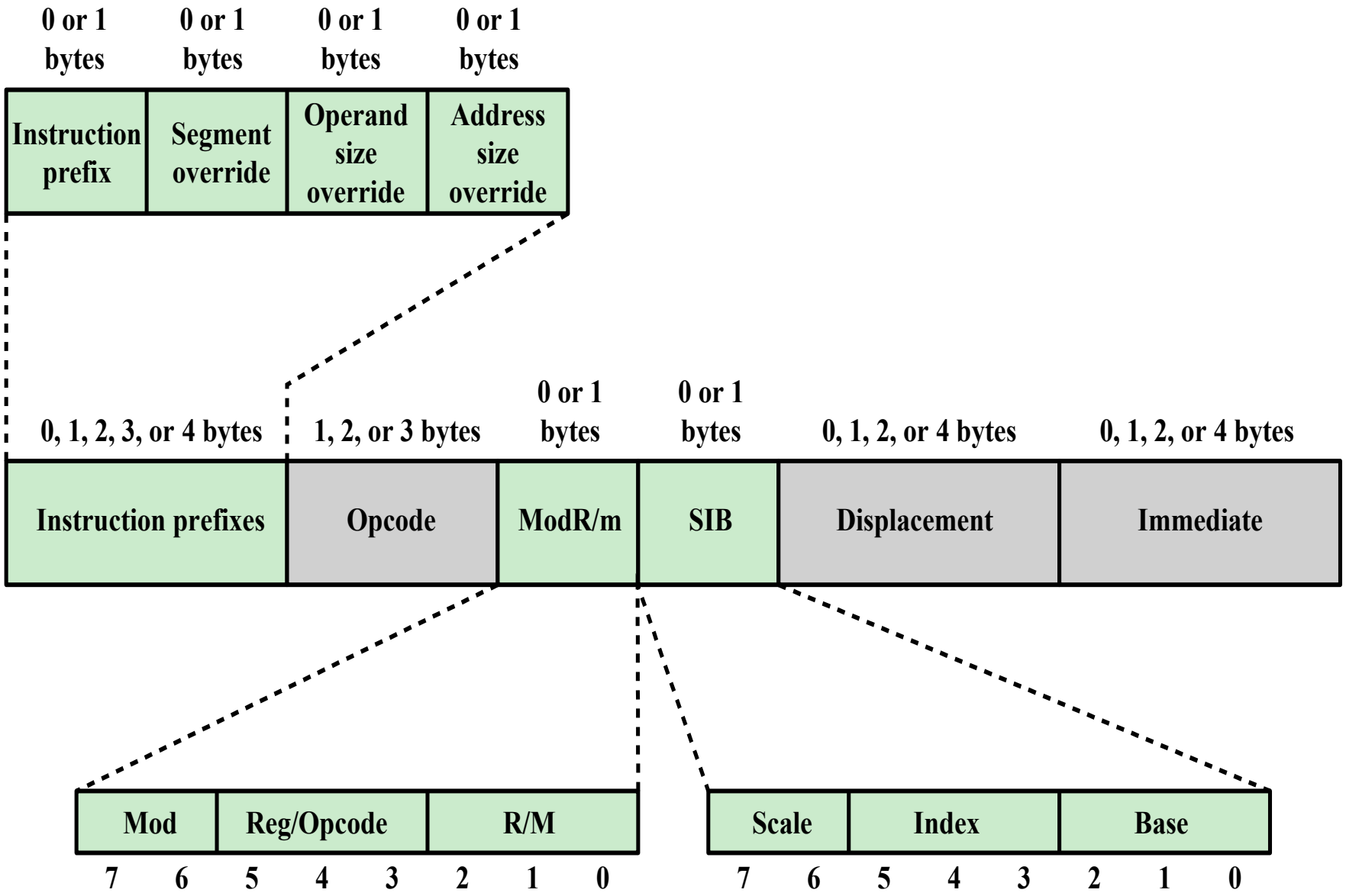
What is granularity in computing?

Granularity (parallel computing) Jump to navigation Jump to search. In parallel computing, granularity (or grain size) of a task is a measure of the amount of work (or computation) which is performed by that task. Another definition of granularity takes into account the communication overhead between multiple processors or processing elements.

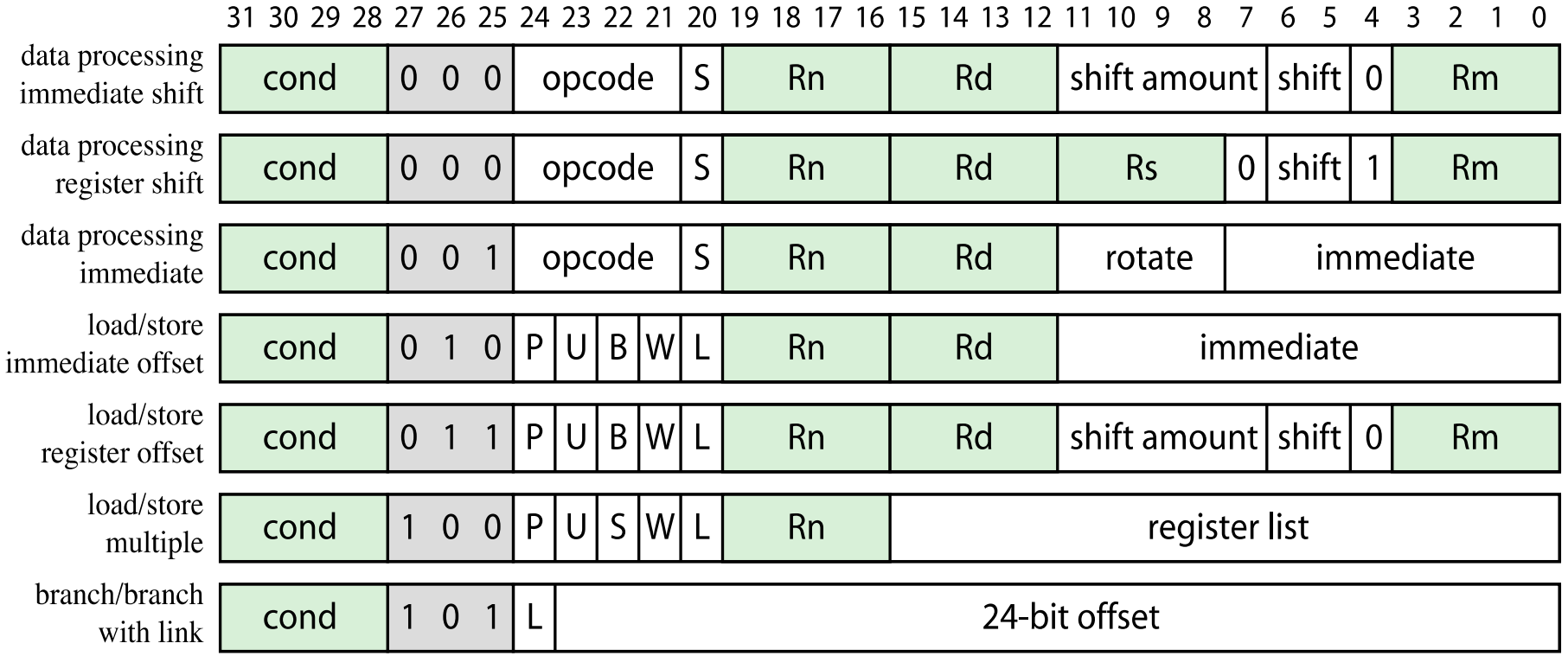
Variable-Length Instructions

- Variations can be provided efficiently and compactly
- Increases the complexity of the processor
- Does not remove the desirability of making all of the instruction lengths integrally related to word length
  - Because the processor does not know the length of the next instruction to be fetched a typical strategy is to fetch a number of bytes or words equal to at least the longest possible instruction
  - Sometimes multiple instructions are fetched

x86 Instruction Format



ARM Instruction Format



S = For data processing instructions, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

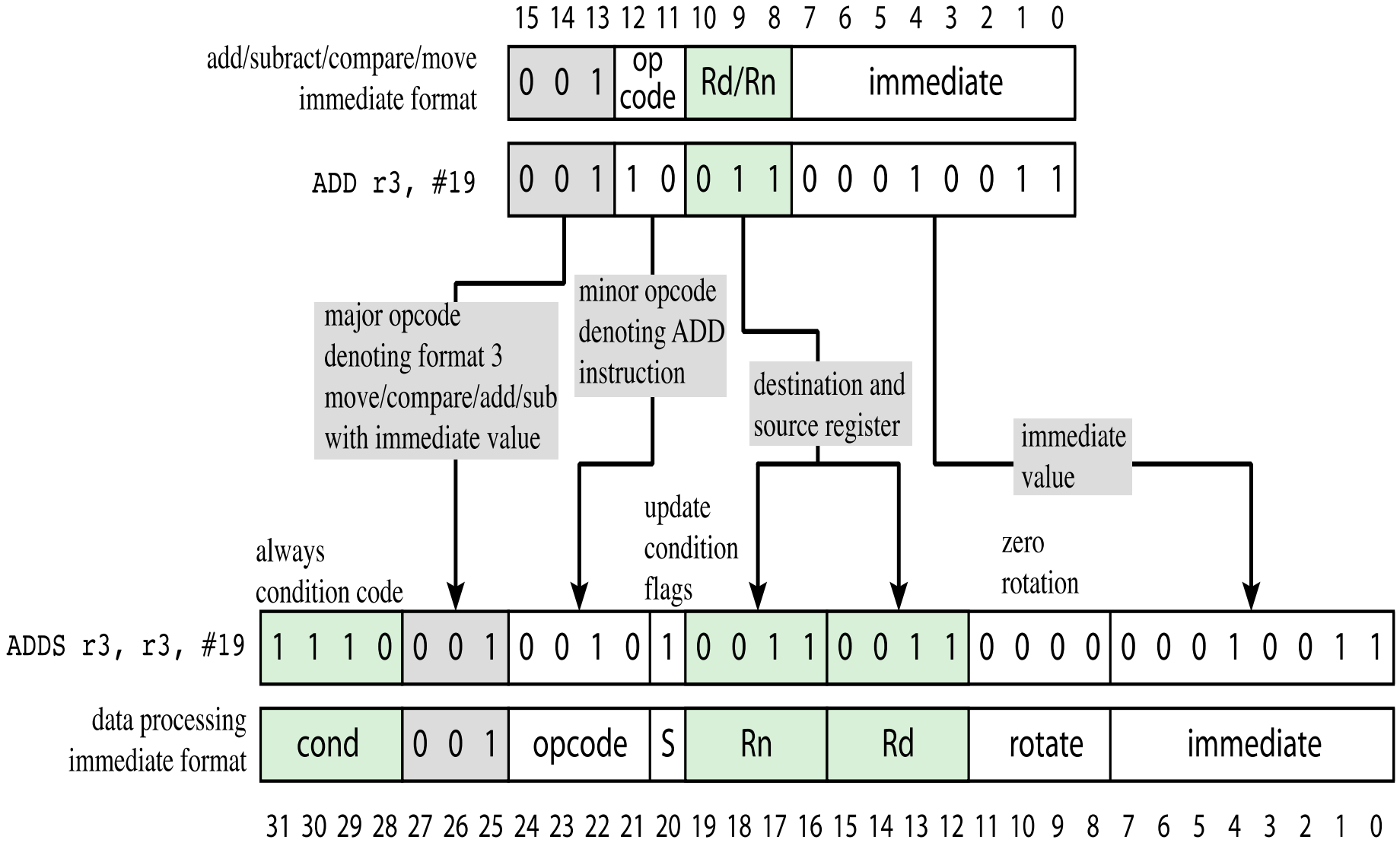
P, U, W = bits that distinguish among different types of addressing_mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

Expanding a THUMB ADD instruction into its ARM equivalent



Problem 1

❖Given the following memory values and a one-address machine with an accumulator,

- Word 20 contains 40
  - Word 30 contains 50
  - Word 40 contains 60
  - Word 50 contains 70

what values do the following instructions load into the accumulator?

A. LOAD IMMEDIATE 20

B. LOAD DIRECT 20

C. LOAD INDIRECT 20

D. LOAD IMMEDIATE 30

E. LOAD DIRECT 30

F. LOAD INDIRECT 30

Summary

Lecture B - 04

Instruction Set Architecture and Design

- 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
- Addressing modes
  - Immediate addressing
  - Direct addressing
  - Indirect addressing
  - Register addressing
  - Register indirect addressing
  - Displacement addressing
  - Stack addressing
- Instruction formats
  - Instruction length
  - Allocation of bits
  - Variable-length instructions
- X86 & ARM instruction formats

- Slides adopted from:
  - Computer Organization and Architecture, 9th Edition

    William Stallings

    ISBN-10: 013293633X | ISBN-13: 9780132936330