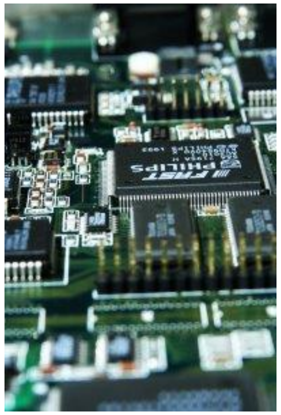
CSE 321a

Computer Organization (1)

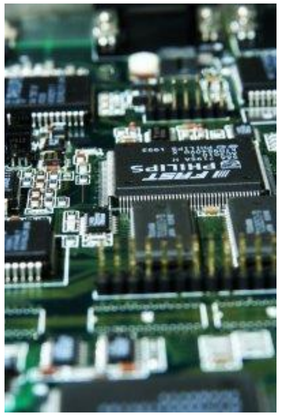
تنظيم الحاسبات (1)



3rd year, Computer Engineering

Fall 2016

Lecture #7



Dr. Hazem Ibrahim Shehata

Dept. of Computer & Systems Engineering

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

Administrivia

- Assignment #2:
  - Due: Thursday, Nov. 10, 2016.
  - Solution will posted on Friday.
- Midterm:
  - Date: Thursday, Nov. 17, 2016.
  - Time: 12:30pm – 2:00pm.
  - Location: قاعة 4د.
  - Coverage: lecture #1 ➔ lecture #6.
- Previous midterms were posted.

Website: http://hshehata.github.io/courses/zu/cse321a
Office hours: Sunday 12:00pm-1:00pm

Chapter 12. Instruction Sets: Characteristics and Functions

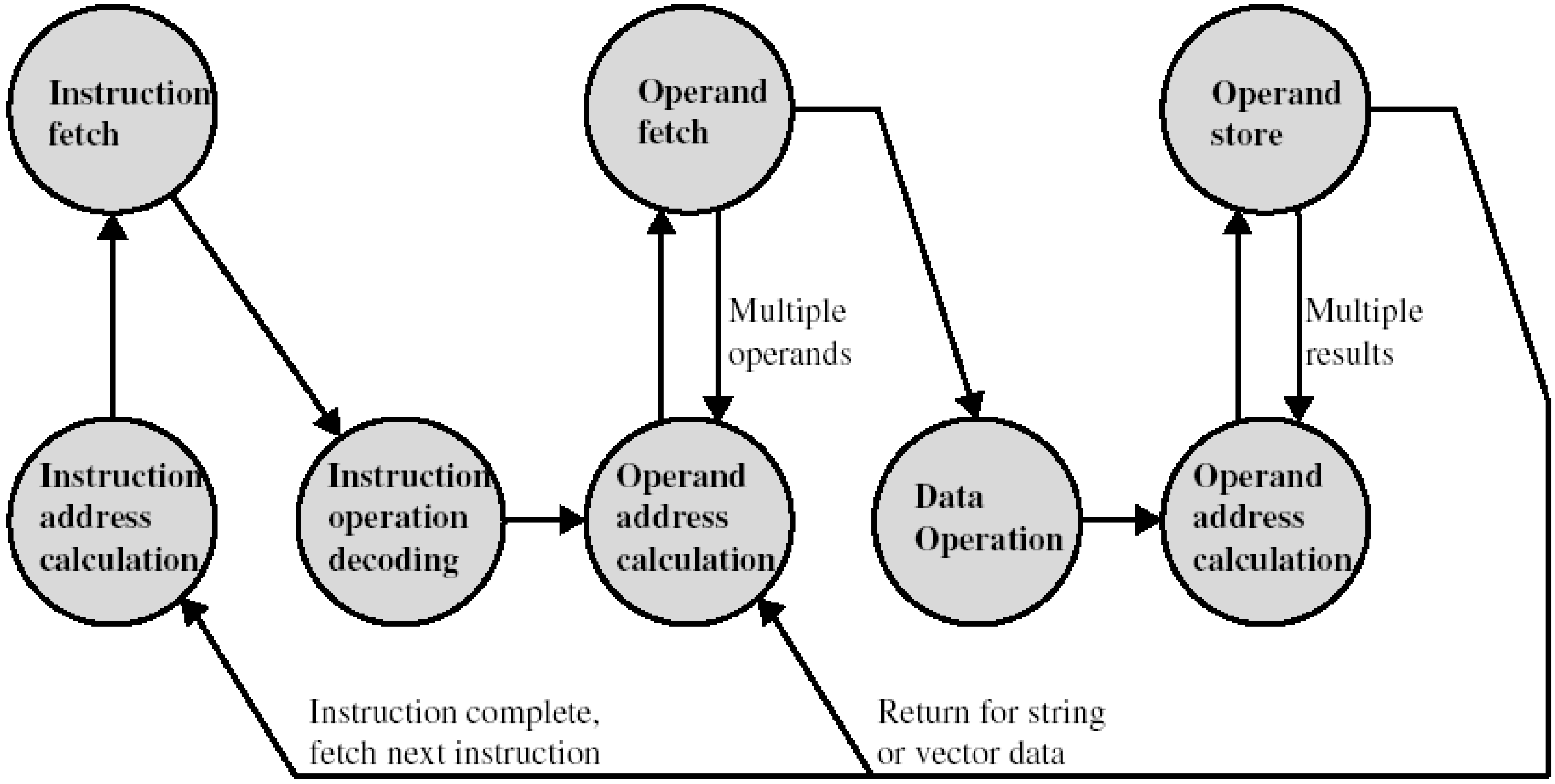
Outline

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

Outline

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

Instruction Cycle State Diagram


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

Instructions

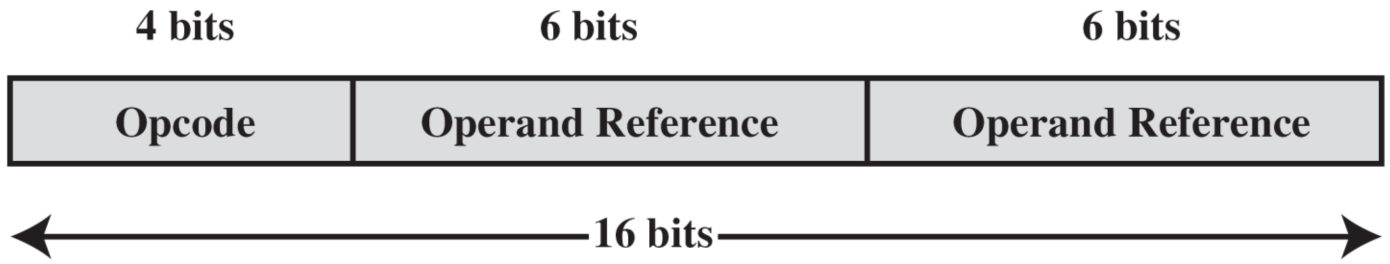
- Instruction set
  - Complete collection of instructions that are understood by a certain CPU, i.e., the set of all machine instructions.
  - Instructions are represented in a binary form (machine language), or a symbolic form (assembly language).
- Elements of an instruction
  - Operation code (opcode)
    - Do this … (e.g., ADD, I/O).
  - Source operand(s) reference
    - To this …
  - Result operand reference
    - Put the answer here …
  - Next instruction reference
    - When you have done that, do this next ...
    - Usually implicit (from PC).

Source and result operand references:

- Immediate: value.
- Main memory: address.
- CPU register: register number. Implicit if one register exists.
- I/O device: I/O module & device. MM address if memory-mapped I/O.

Instruction Representation

- Within computer, each instruction is represented by a sequence of bits, divided into fields.
- During the execution phase, an instruction is read from the IR register.
- CPU extracts the data from these fields to perform the required operation.
- Programmers represent instructions symbolically:
  - Opcodes ➔ mnemonics, e.g., ADD, SUB, LOAD.
  - Operands ➔ symbolic names, e.g., Y = 514.
  - Example: ADD R1, Y



Types of Instructions

- Data processing
  - Arithmetic and logic instructions.
  - Arithmetic: computations (processing) on numeric data.
  - Logic: operate on the bits of a word (any data type).
- Data storage
  - Memory instructions.
- Data movement
  - I/O instructions.
- Program flow control
  - Jump and branch instructions.

Number of Addresses (1)

- What is the maximum number of addresses that need to be represented in an instruction?
  - "addresses" here means "explicit operand references".
  - Instructions that perform binary arithmetic/logic operations (e.g., +, &) require most # of "addresses".
    - 2 addresses to specify source operands, 1 to specify destination operand (i.e., result), 1 to specify next instruction.
    - Four addresses must be specified in this case!
  - Problem: Specifying 4 addresses ➔ too long instruction!
  - Solution: Specify some operands implicitly!
    - Address of next instruction is implicitly known ➔ [PC]+1!
    - Destination: same as source, or always the accumulator!
  - In most architectures, most instructions are one-, two-, or three-address instructions.

Program to Execute: Y = (A–B)/(C+D×E)

3-address instructions

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	$Y \leftarrow Y \div T$

2-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	$Y \leftarrow Y \div T$

"MPY D, E" would alter D. So, it is replaced by these two instructions!

1-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	$Y \leftarrow AC$

Number of Addresses (2)

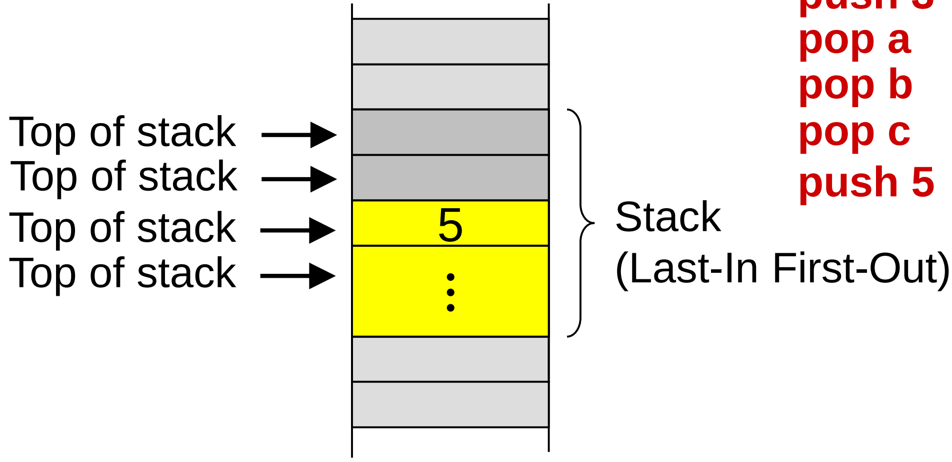
- 3 addresses
  - Operand 1, Operand 2, Result.
  - Template: a = b + c.
  - Needs very long words to hold everything.
- 2 addresses
  - One address doubles as operand and result.
  - Template: a = a + b.
  - Reduces length of instruction.
  - Requires some extra work.
    - Temporary storage to hold some results!

Number of Addresses (3)

- 1 address
  - Implicit second address.
  - Usually a register (accumulator: AC).
  - Common on early machines.
- 0 (zero) addresses
  - All addresses implicit.
  - Usually the top element(s) of the stack.
    - Stack: set of locations used as last-in-first-out buffer.
  - Ex.:

c = a + b

push a
push b
add
pop c



push 3
pop a
pop b
pop c
push 5

Instruction Addresses (nonbranching instructions)

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	$A \leftarrow B \text{ OP } C$
2	OP A, B	$A \leftarrow A \text{ OP } B$
1	OP A	$AC \leftarrow AC \text{ OP } A$
0	OP	$T \leftarrow (T - 1) \text{ OP } T$

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

Summary: How Many Addresses?

- More addresses
  - 👎 —More complex instructions ➔ more complex CPU.
  - 👎 —Longer-length instructions ➔ slower fetch/execution.
  - 👍 —Fewer instructions per program.
  - 👍 —More registers are available ➔ more operations performed solely on registers ➔ quicker processing
- Fewer addresses
  - 👍 —More primitive instructions ➔ less complex CPU.
  - 👍 —Shorter-length instructions ➔ faster fetch/execution.
  - 👎 —More instructions per program.
- Most modern machines employ a mixture of one-, two- and three-address instructions.

Instruction Set Design

- Instruction set defines how many functions performed by CPU ➔ significantly affects CPU implementation.
- Most important issues in designing instruction set:
  - —Operation repertoire: number of op's, their complexity.
  - —Data types: types of data dealt with by instructions.
  - —Instruction formats: instruction length, number of addresses, size of various fields.
  - —Registers: number of CPU registers available, which operations can be performed on which registers?
  - —Addressing modes: ways of referencing operands.
  - —Style: RISC vs CISC!

Outline

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

Types of Operands

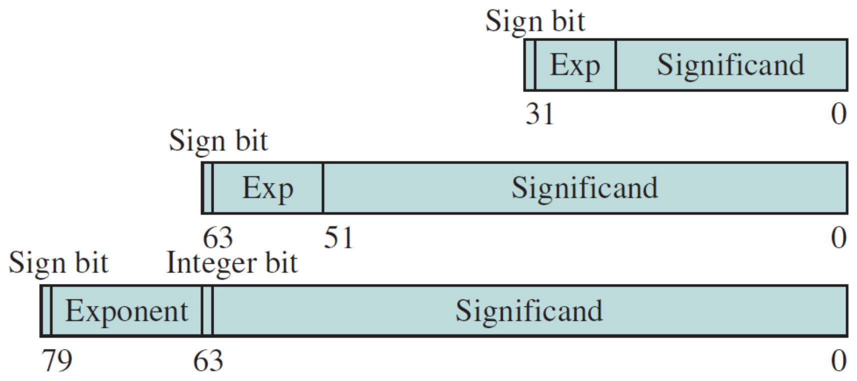
- Machine instructions operate on data (strings of 1's and 0's) interpreted as one of the following:
  - Addresses
    - Unsigned integers representing pointers to memory locations.
  - Numbers
    - Signed/unsigned integers, floating point, BCD.
    - Limited ➔ limited magnitude (& precision for real numbers).
  - Characters
    - IRA (ASCII): 7-bit code. An 8th bit could be used for parity.
    - EBCDIC: 8-bit code (used on IBM mainframes).
  - Logical Data
    - Bit-oriented view of data ➔ array of logical values (true/false).
    - Each bit in the array can be individually manipulated!

Ex.: x86 Data Types

- The x86 instructions can handle so many types of data. Most common types are:
  - General: arbitrary binary contents.
    - Size: byte, word (16 bit), double-word (32 bit), quad-word (64 bit), and double-quad-word (128 bit).
    - Note: x86 memory is byte-addressable.
  - Integer: singed binary value (two's complement)
    - Size: byte, word, double-word, quad-word.
    - Note: Multi-byte numeric data in x86 are saved in "little-endian" style, i.e., least significant bytes are stored first.
  - Ordinal: unsigned binary values
    - Size: byte, word, double-word, quad-word.

Ex.: x86 Data Types

- … (cont.):
  - Floating-point: conforming to IEEE-754 standard.



Single precision

Double precision

Double-extended precision

  - Binary-Coded-Decimal (BCD):
    - Unpacked BCD: One BCD digit per byte.
    - Packed BCD: Two BCD digits per byte.
  - Bit fields: sequence of independent bits (up to 32 bits).
  - Byte strings: sequence of bytes, words, or double-words.
  - Others: near/far pointers, packed byte/word/… SIMD, and many more!!

Outline

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

1. Data Transfer

- Instruction must specify:
  - —Location of source operand.
  - —Location of destination operand.
  - —Amount of data.
  - —Addressing mode for each operand.

  (Location of source/destination operand:)
  - Memory.
  - Register.
  - Top of stack.

- Location of operand and amount of data could be specified as part of opcode or operand fields:
  1. Specified in opcode ➔ different instructions for different kinds of movements (reg./mem.) and amounts of data.
     – e.g. IBM EAS/390 (next slide) ➔ Disadv.: hard to program!
  2. Specified in operand ➔ 1 instruction for same amount of data. Kind of movement is specified in operand.
     – e.g. VAX and x86 ➔ Disadv.: less compact!

1. Data Transfer – Ex.: IBM EAS/390

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

1. Data Transfer – Common Operations

Operation Name	Description
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

2. Arithmetic

- Add, Subtract, Multiply, Divide
  - Signed Integer (fixed-point) numbers
  - Floating point numbers.
- May include
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)
  - Absolute (if a<0 then -a else a)
- May involve data transfer
- Desired arithmetic operation performed by ALU.

3. Logical – Common Operations

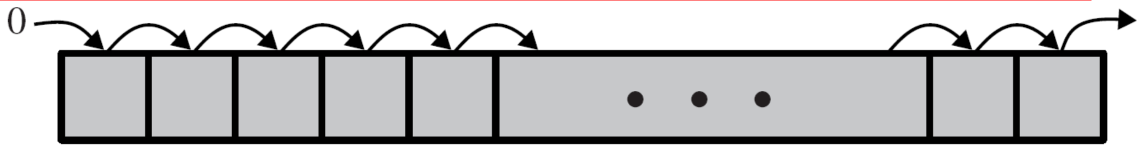
Operation Name	Description
AND	Perform logical AND
OR	Perform logical OR
NOT (complement)	Perform logical NOT
Exclusive-OR	Perform logical XOR
Test	Test specified condition; set flag(s) based on outcome
Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
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

3. Logical – AND, OR, NOT, XOR, ...

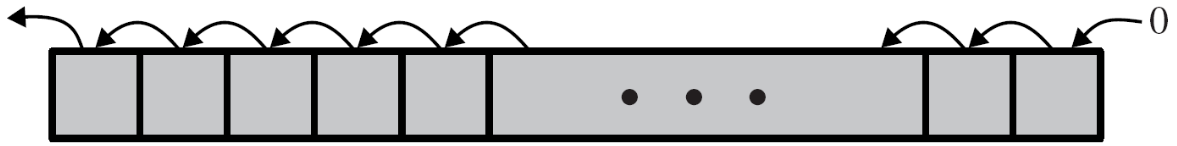
- To manipulate individual bits → bit twiddling.
- Could be used for **bit masking**:
  - Reset a specific group of bits:
    - Operation: AND
    - Mask: 0 for each bit to be reset and 1 otherwise.
  - Set a specific group of bits:
    - Operation: OR
    - Mask: 1 for each bit to be set and 0 otherwise.
  - Toggle a specific group of bits:
    - Operation: XOR
    - Mask: 1 for each bit to be inverted and 0 otherwise.

3. Logical – Shift and Rotate Operations

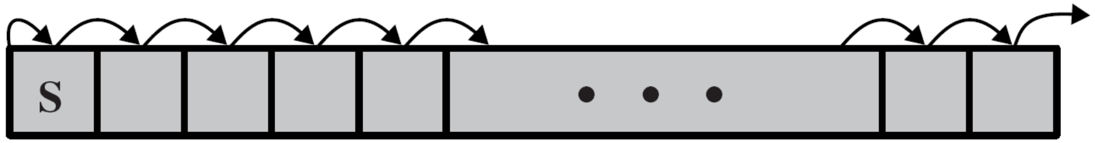
Logical right shift



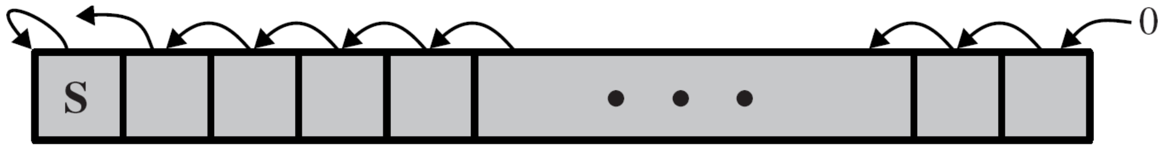
Logical left shift



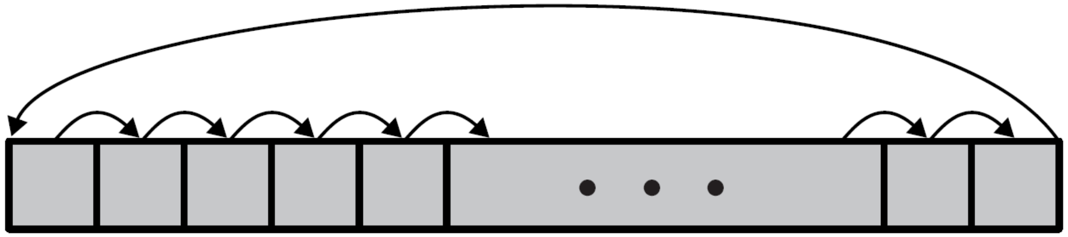
Arithmetic right shift



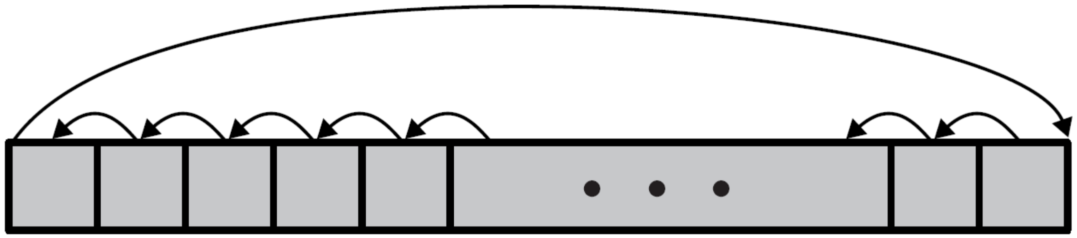
Arithmetic left shift



Right rotate



Left rotate



3. Logical – Shift and Rotate Operations

- Example: suppose we wish to **unpack** 2 characters from 16-bit location and send them to an I/O device.
  1. Process left-hand character:
     a) Load word into register.
     b) Shift 8-bit positions to right.
     c) Perform I/O.
  2. Process right-hand character:
     a) Load word again into register.
     b) AND with 0000000011111111.
     c) Perform I/O.

Reading Material

- Stallings, Chapter 12:
  - Pages 406 – 424