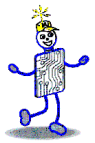


CHAPTER 7: The CPU and Memory

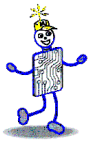
The Architecture of Computer Hardware, Systems Software & Networking:
An Information Technology Approach

4th Edition, Irv Englander

John Wiley and Sons ©2010

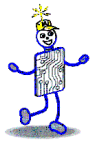
PowerPoint slides authored by Wilson Wong, Bentley University

PowerPoint slides for the 3rd edition were co-authored with Lynne Senne, Bentley College

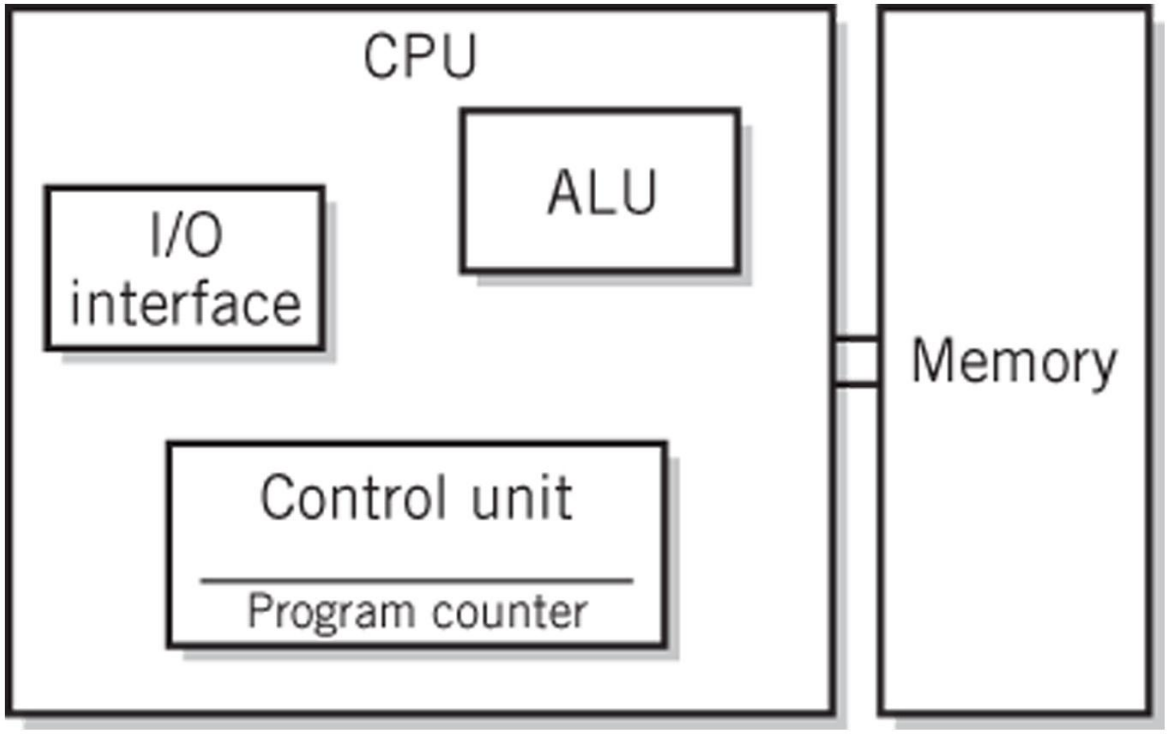


CPU: Major Components

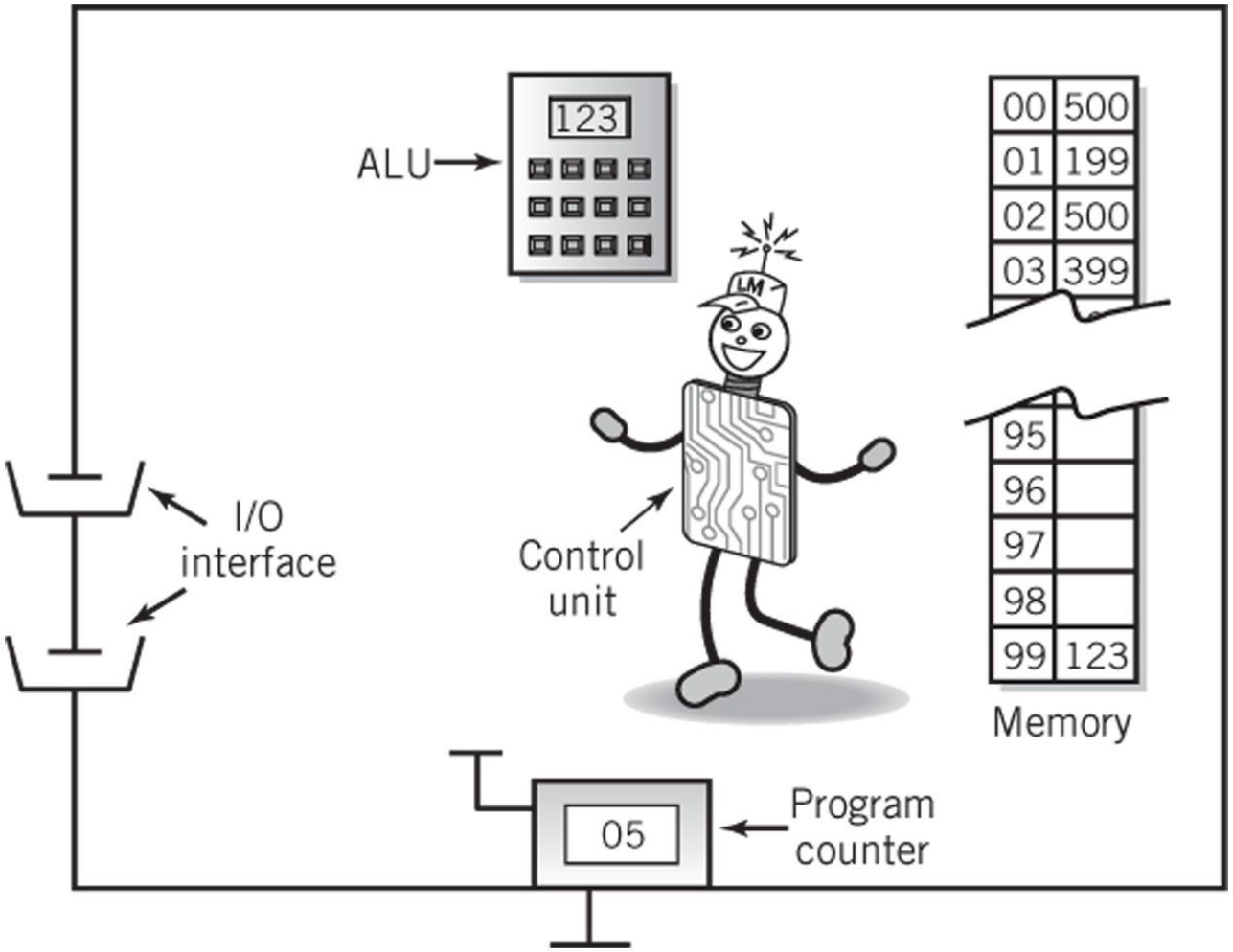
- *ALU* (arithmetic logic unit)
  - Performs calculations and comparisons
- *CU* (control unit)
  - Performs fetch/execute cycle
    - Accesses program instructions and issues commands to the ALU
    - Moves data to and from CPU registers and other hardware components
  - Subcomponents:
    - *Memory management unit:* supervises fetching instructions and data from memory
    - *I/O Interface:* sometimes combined with memory management unit as *Bus Interface Unit*

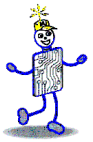


System Block Diagram



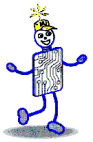
The Little Man Computer





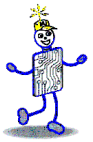
Concept of Registers

- Small, *permanent* storage locations within the CPU used for a particular purpose
- Manipulated directly by the Control Unit
- Wired for *specific function*
- Size in bits or bytes (not in MB like memory)
- Can hold data, an address or an instruction
- How many registers does the LMC have?
- What are the registers in the LMC?



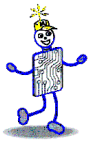
Registers

- Use of Registers
  - Scratchpad for currently executing program
    - Holds data needed quickly or frequently
  - Stores information about status of CPU and currently executing program
    - Address of next program instruction
    - Signals from external devices
- General Purpose Registers
  - *User-visible registers*
  - Hold intermediate results or data values, e.g., loop counters
  - Equivalent to LMC's calculator
  - Typically several dozen in current CPUs



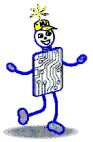
Special-Purpose Registers

- *Program Count Register* (PC)
  - Also called instruction pointer
- *Instruction Register (IR)*
  - Stores instruction fetched from memory
- *Memory Address Register (MAR)*
- *Memory Data Register (MDR)*
- *Status Registers*
  - Status of CPU and currently executing program
  - *Flags* (one bit Boolean variable) to track condition like arithmetic carry and overflow, power failure, internal computer error
- I/O Interface Registers: Address & Data



Register Operations

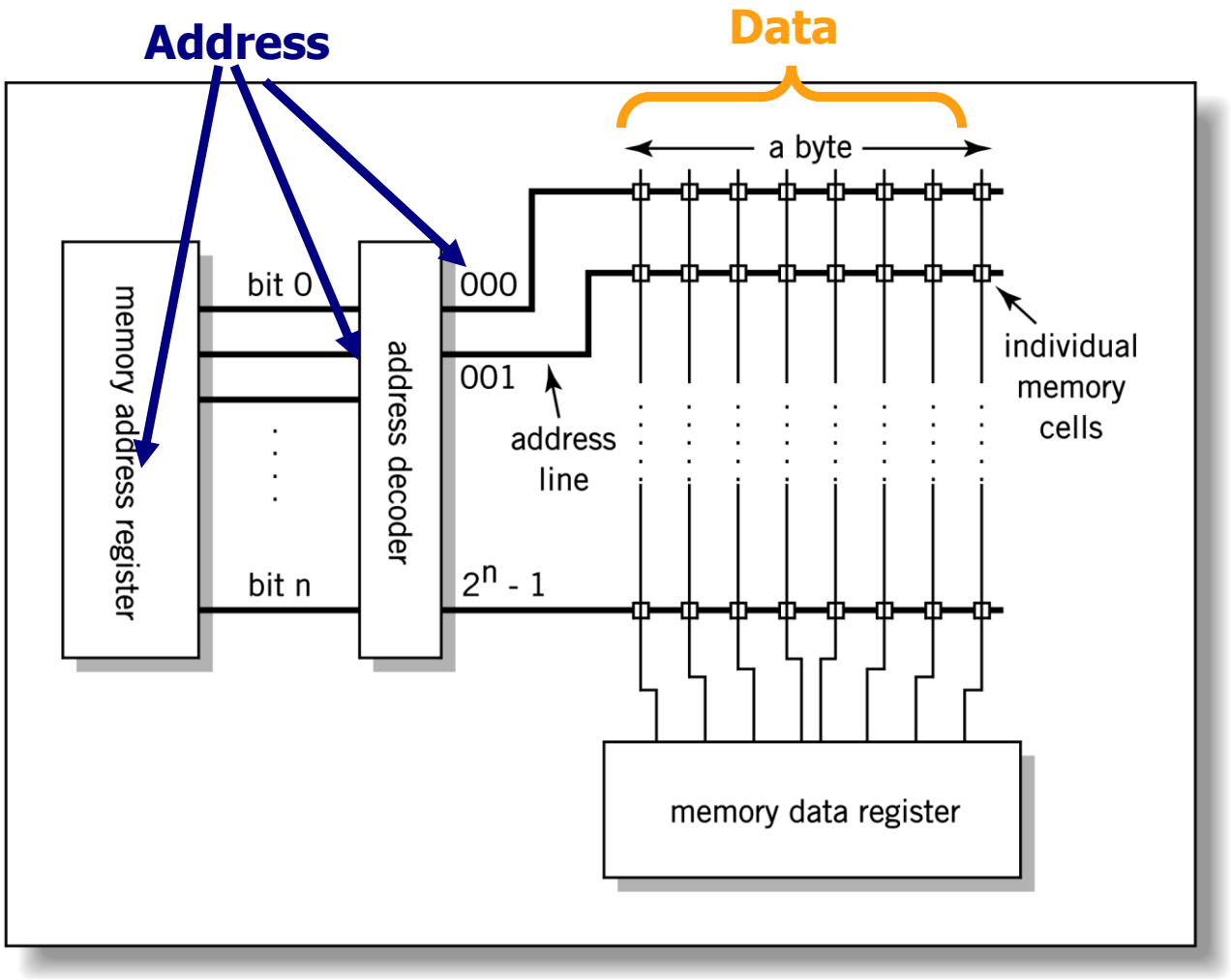
- Stores values from other locations (registers and memory)(Destroys previous value stored within)
- Addition and subtraction: Leaves the sum/difference
- Shift or rotate data: Multiply/Division
- Test contents for conditions such as zero or positive, negative, too large

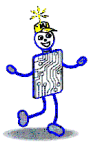


Operation of Memory

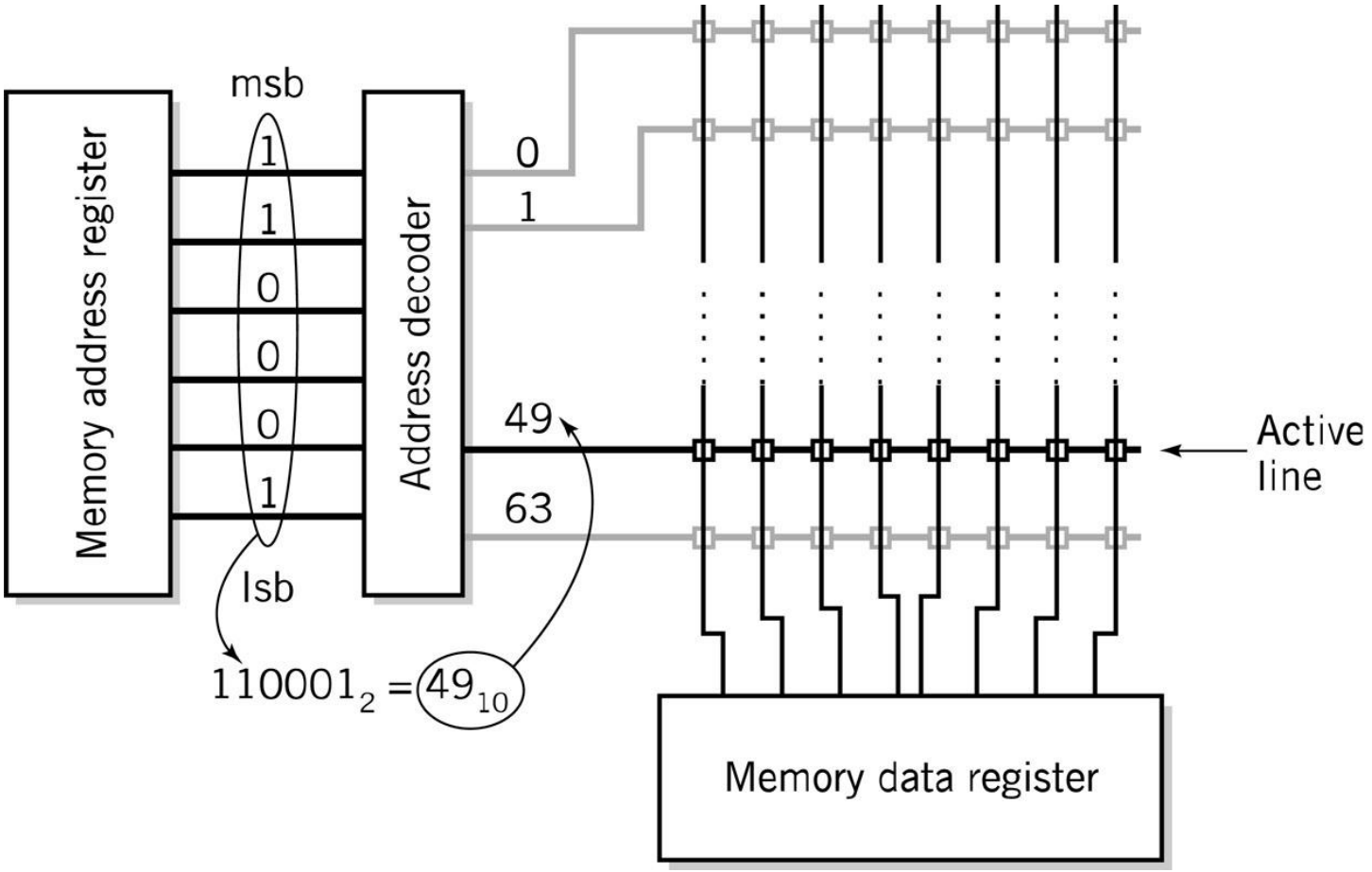
- Each memory location has a unique address
- Address from an instruction is copied to the MAR which finds the location in memory
- CPU determines if it is a store or retrieval
- Transfer takes place between the MDR and memory
- MDR is a two way register: What does this mean?

Relationship between MAR, MDR and Memory

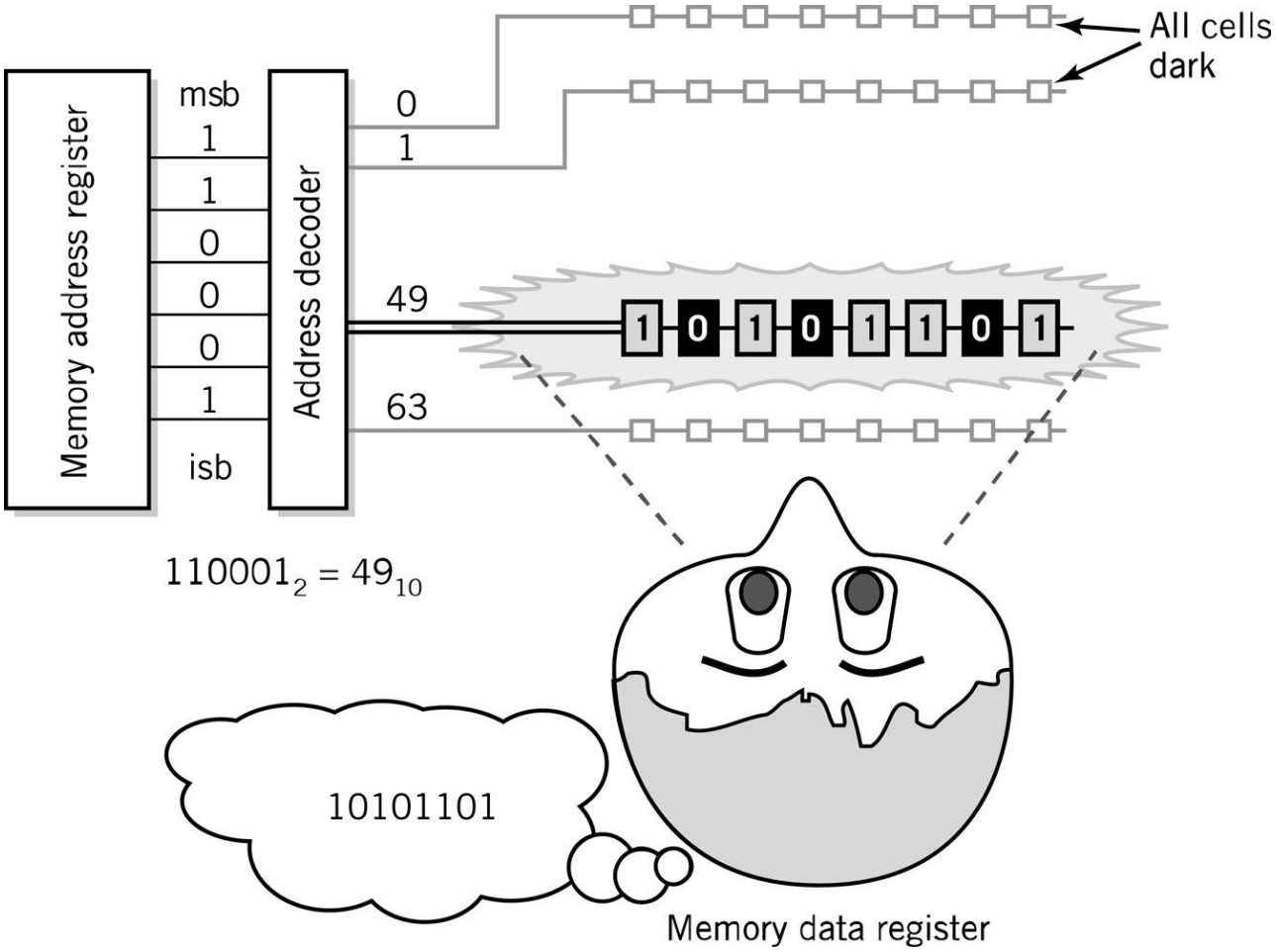




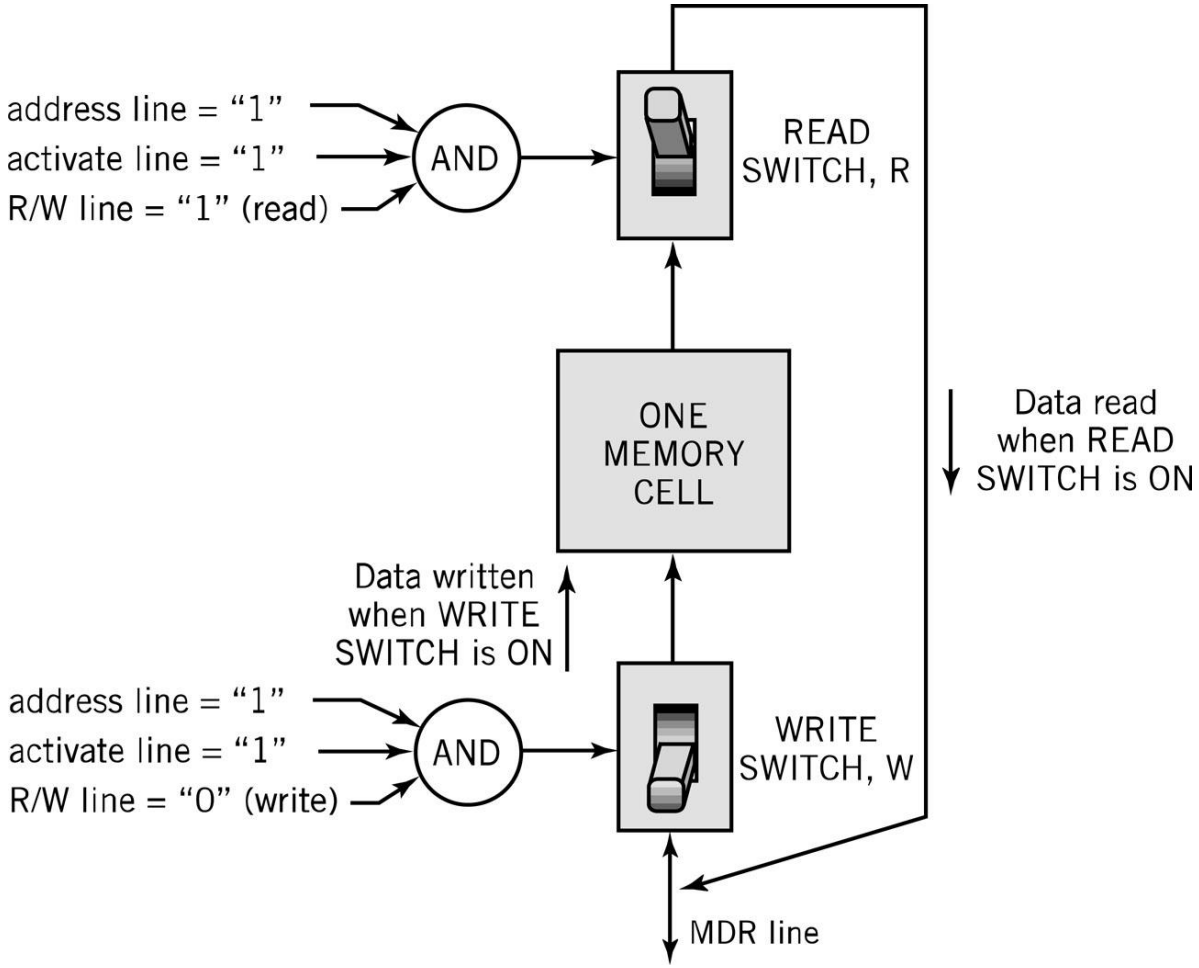
MAR-MDR Example

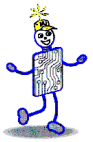


Visual Analogy of Memory



Individual Memory Cell



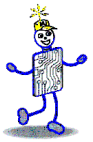


Memory Capacity

Determined by two factors

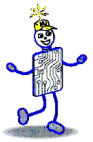
1. Number of bits in the MAR
   - LMC = 100 (00 to 99)
   - 2^K where K = width of the register in bits
2. Size of the address portion of the instruction
   - 4 bits allows 16 locations
   - 8 bits allows 256 locations
   - 32 bits allows 4,294,967,296 or 4 GB

- Size of data word retrieved in single operation depends on:
  - 1) Size of MDR
  - 2) Connection between Memory & CPU



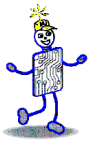
RAM: Random Access Memory

- *DRAM (Dynamic RAM)*
  - Most common, cheap, less electrical power, less heat, smaller space
  - Volatile: must be refreshed (recharged with power) 1000's of times each second
- *SRAM (static RAM)*
  - Faster and more expensive than DRAM
  - Volatile
  - Small amounts are often used in *cache memory* for high-speed memory access



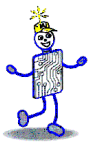
Nonvolatile Memory

- *ROM*
  - Read-only Memory
  - Holds software that is not expected to change over the life of the system
- EEPROM
  - Electrically Erasable Programmable ROM
- *Flash Memory*
  - Faster than disks but more expensive
  - Uses hot carrier injection to store bits of data
  - Slow rewrite time compared to RAM
  - Useful for nonvolatile portable computer storage

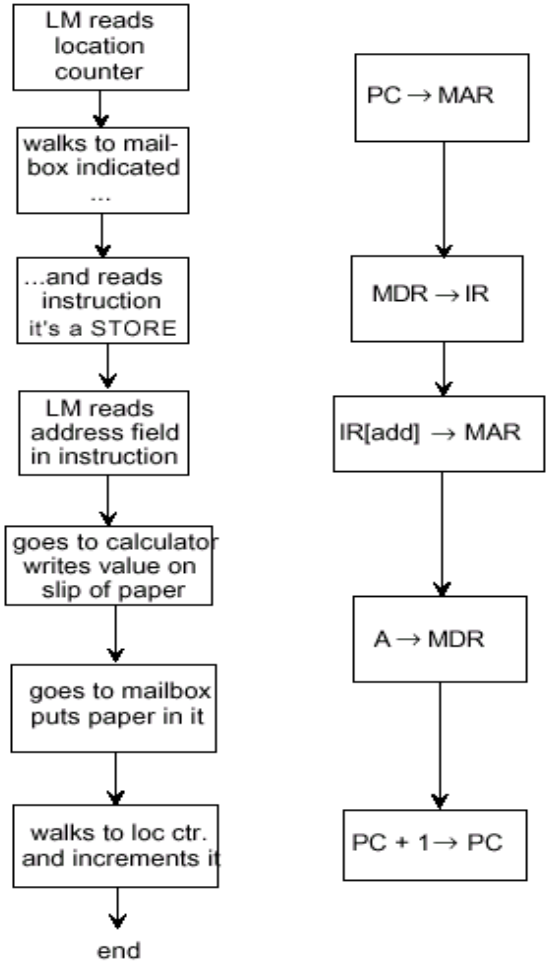


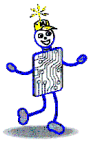
Fetch-Execute Cycle

- Two-cycle process because both instructions and data are in memory
- *Fetch*
  - Decode or find instruction, load from memory into register and signal ALU
- *Execute*
  - Performs operation that instruction requires
  - Move/transform data



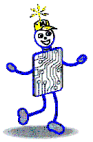
LMC vs. CPU
Fetch and Execute Cycle





Load Fetch/Execute Cycle

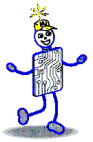
1. PC → MAR	Transfer the address from the PC to the MAR
2. MDR → IR	Transfer the instruction to the IR
3. IR[address] → MAR	Address portion of the instruction loaded in MAR
4. MDR → A	Actual data copied into the accumulator
5. PC + 1 → PC	Program Counter incremented



Store Fetch/Execute Cycle

1. PC → MAR	Transfer the address from the PC to the MAR
2. MDR → IR	Transfer the instruction to the IR
3. IR[address] → MAR	Address portion of the instruction loaded in MAR
4. A → MDR*	Accumulator copies data into MDR
5. PC + 1 → PC	Program Counter incremented

*Notice how Step #4 differs for LOAD and STORE



ADD Fetch/Execute Cycle

1. PC → MAR	Transfer the address from the PC to the MAR
2. MDR → IR	Transfer the instruction to the IR
3. IR[address] → MAR	Address portion of the instruction loaded in MAR
4. A + MDR → A	Contents of MDR added to contents of accumulator
5. PC + 1 → PC	Program Counter incremented

LMC Fetch/Execute

SUBTRACT

PC → MAR

MDR → IR

IR[addr] → MAR

A – MDR → A

PC + 1 → PC

IN

PC → MAR

MDR → IR

IOR → A

PC + 1 → PC

OUT

PC → MAR

MDR → IR

A → IOR

PC + 1 → PC

HALT

PC → MAR

MDR → IR

BRANCH

PC → MAR

MDR → IR

IR[addr] → PC

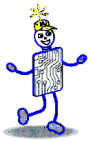
BRANCH on Condition

PC → MAR

MDR → IR

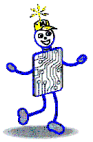
If condition false: PC + 1 → PC

If condition true: IR[addr] → PC



Bus

- The physical connection that makes it possible to transfer data from one location in the computer system to another
- Group of electrical or optical conductors for carrying signals from one location to another
  - Wires or conductors printed on a circuit board
  - *Line*: each conductor in the bus
- 4 kinds of signals
  1. Data
  2. Addressing
  3. Control signals
  4. Power (sometimes)

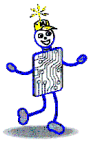


Bus Characteristics: Criteria

- Number of separate conductors
- Data width in bits carried simultaneously
- Addressing capacity
- Lines on the bus are for a single type of signal or shared
- Throughput - data transfer rate in bits per second
- Distance between two endpoints
- Number and type of attachments supported
- Type of control required
- Defined purpose
- Features and capabilities

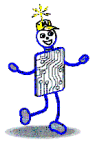
Bus Categorizations

- Parallel vs. serial buses
- Direction of transmission
  - Simplex – unidirectional
  - Half duplex – bidirectional, one direction at a time
  - Full duplex – bidirectional simultaneously
- Method of interconnection
  - Point-to-point – single source to single destination
    - Cables – point-to-point buses that connect to an external device
  - Multipoint bus – also broadcast bus or multidrop bus
    - Connect multiple points to one another



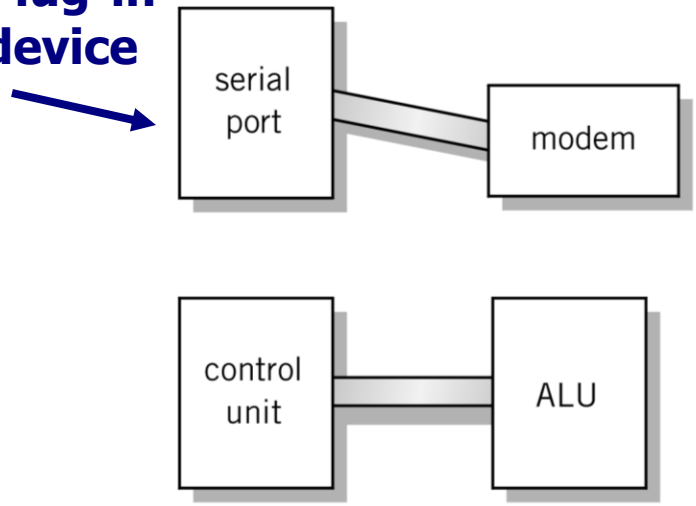
Parallel vs. Serial Buses

- Parallel
  - High throughput because all bits of a word are transmitted simultaneously
  - Expensive and require a lot of space
  - Subject to radio-generated electrical interference which limits their speed and length
  - Generally used for short distances such as CPU buses and on computer motherboards * SKEW *
- Serial
  - 1 bit transmitted at a timed
  - Single data line pair and a few control lines
  - For many applications, throughput is higher than for parallel because of the lack of electrical interference

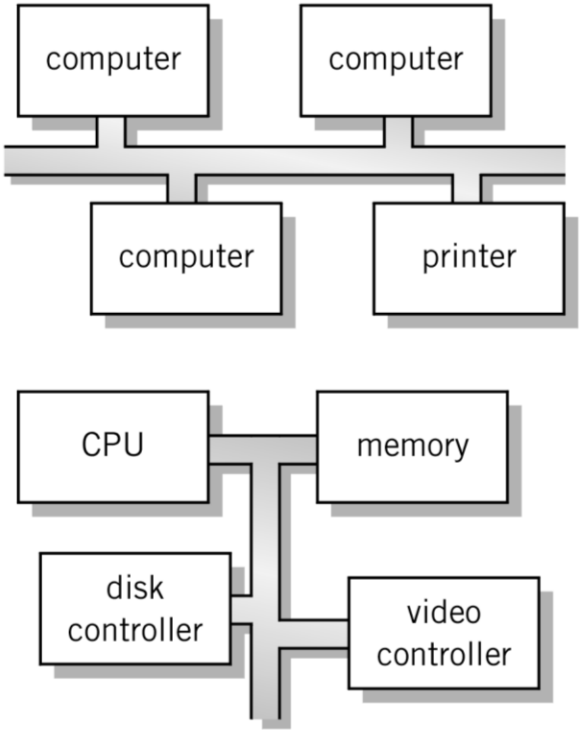


Point-to-point vs. Multipoint

Plug-in device



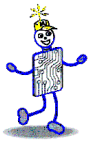
examples of point-to-point buses



examples of multipoint buses

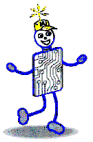
Broadcast bus Example: Ethernet

Shared among multiple devices



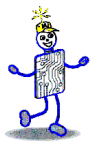
Classification of Instructions

- Data Movement (load, store)
  - Most common, greatest flexibility
  - Involve memory and registers
  - What's this size of a *word* ? 16? 32? 64 bits?
- Arithmetic
  - Operators **+ - / * ^**
  - Integers and floating point
- Boolean Logic
  - Often includes at least **AND, XOR,** and **NOT**
- Single operand manipulation instructions
  - Negating, decrementing, incrementing, set to 0

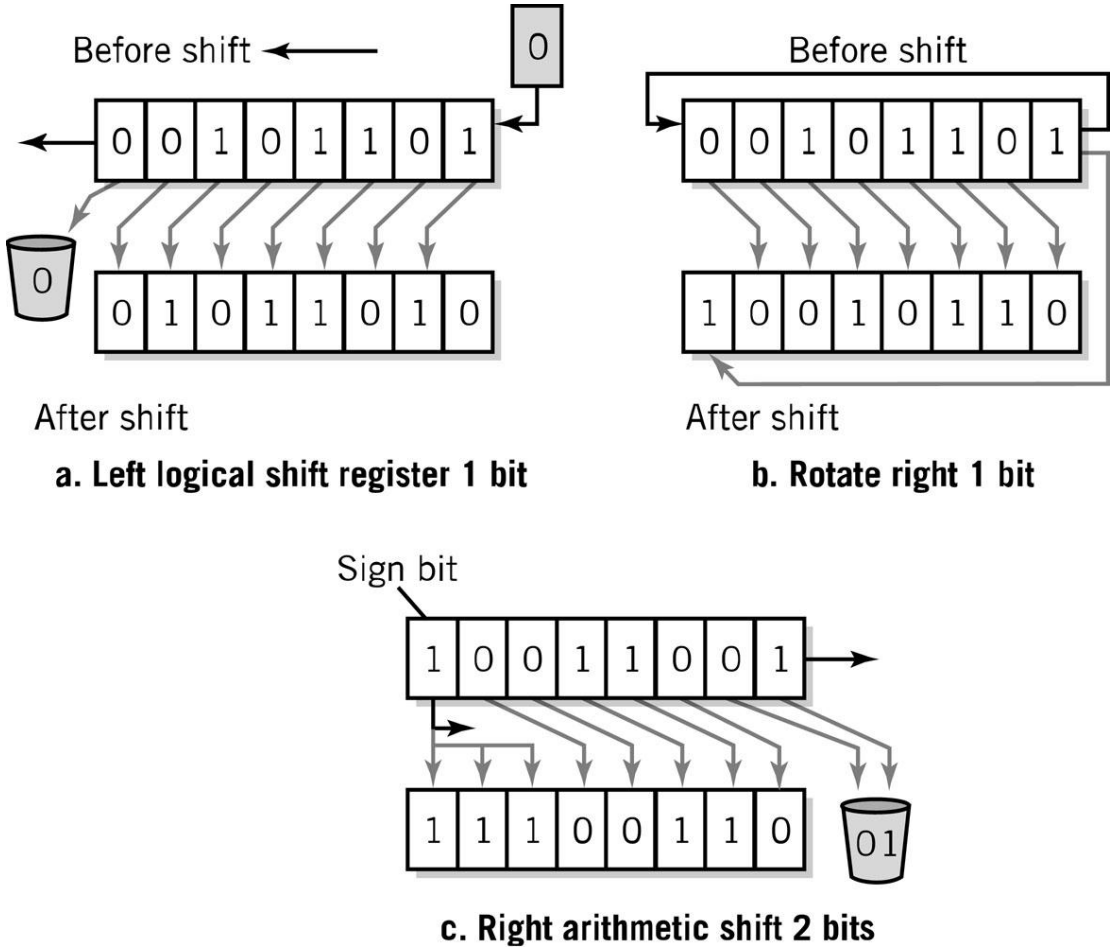


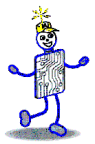
More Instruction Classifications

- Bit manipulation instructions
  - Flags to test for conditions
- Shift and rotate
- Program control
- Stack instructions
- Multiple data instructions
- I/O and machine control



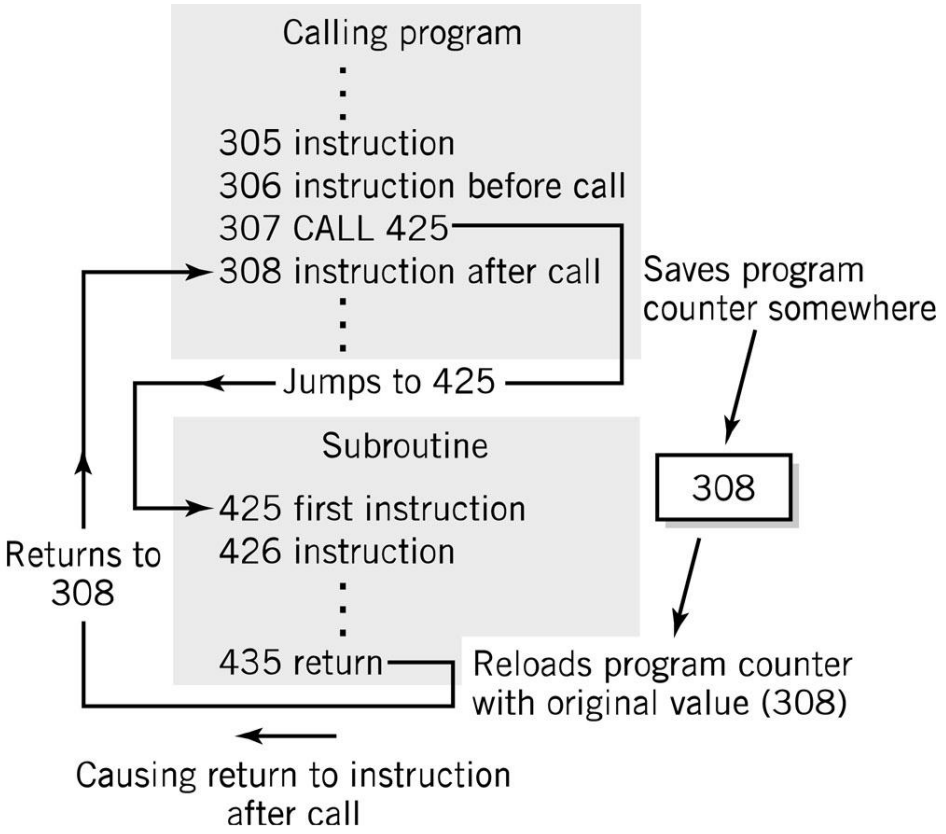
Register Shifts and Rotates

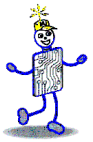




Program Control Instructions

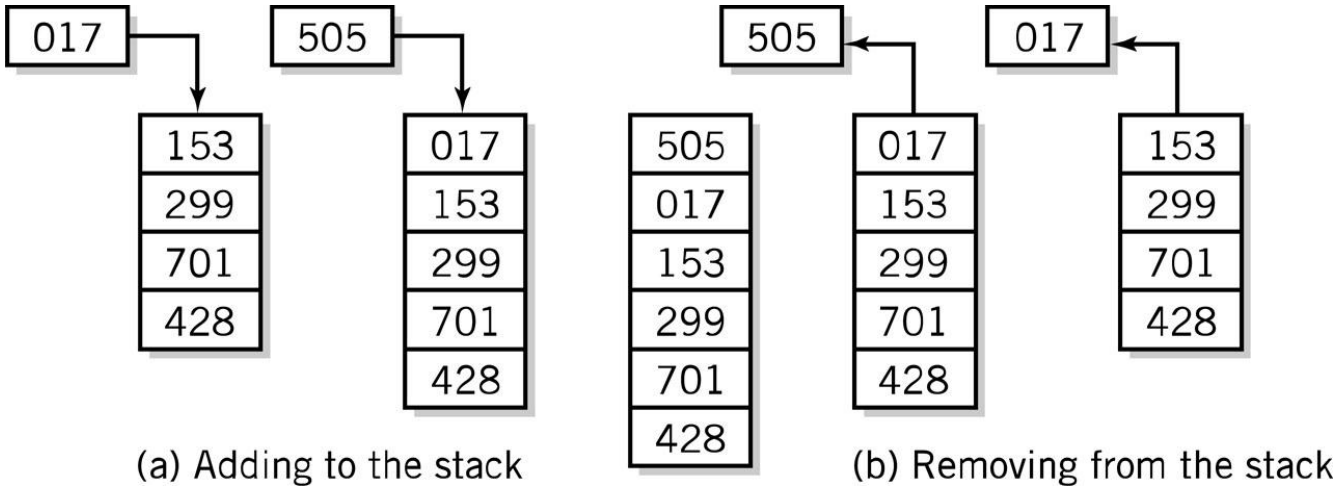
- Program control
  - Jump and branch
  - Subroutine call and return

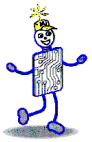




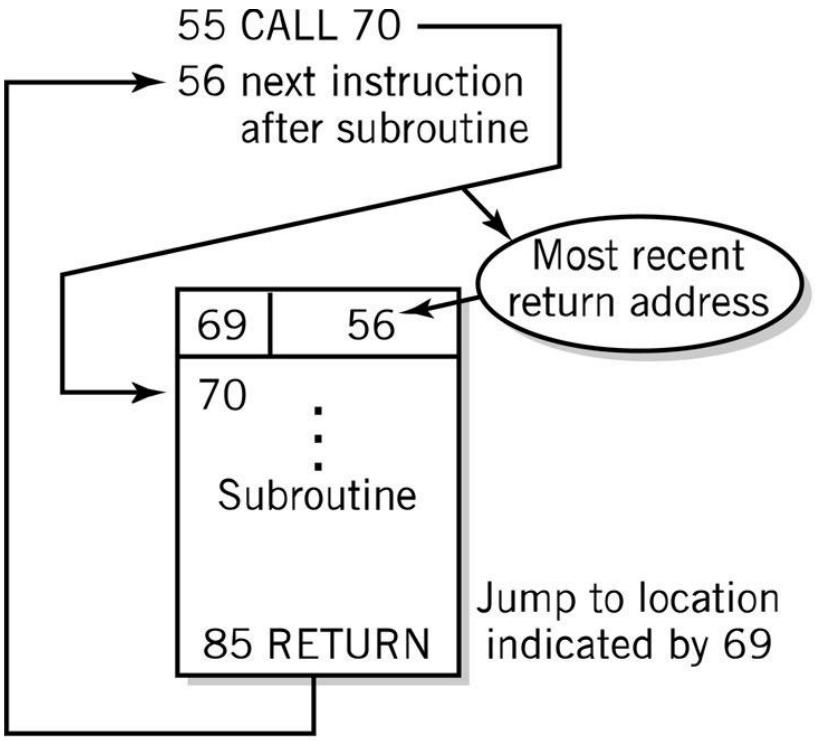
Stack Instructions

- Stack instructions
  - LIFO method for organizing information
  - Items removed in the reverse order from that in which they are added

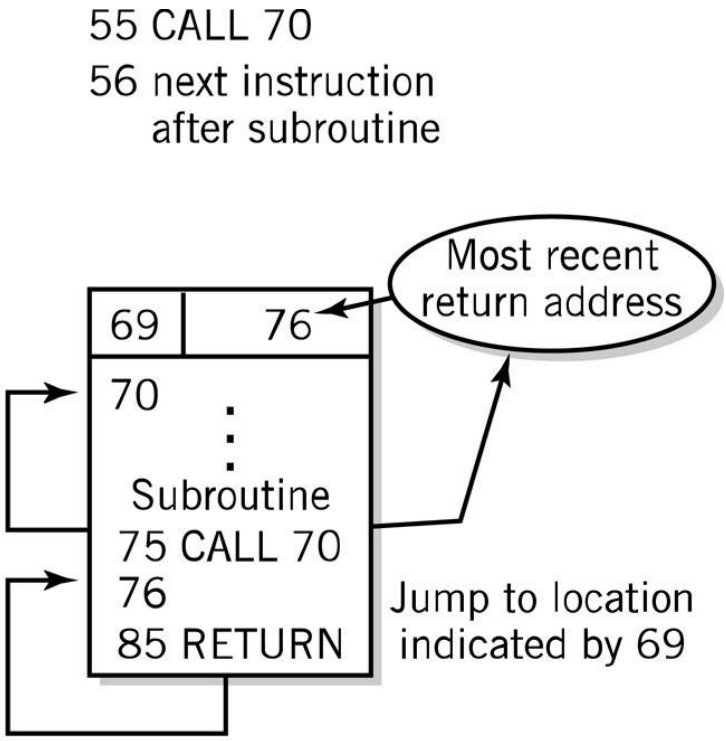




Fixed Location Subroutine Return Address Storage: *Oops!*

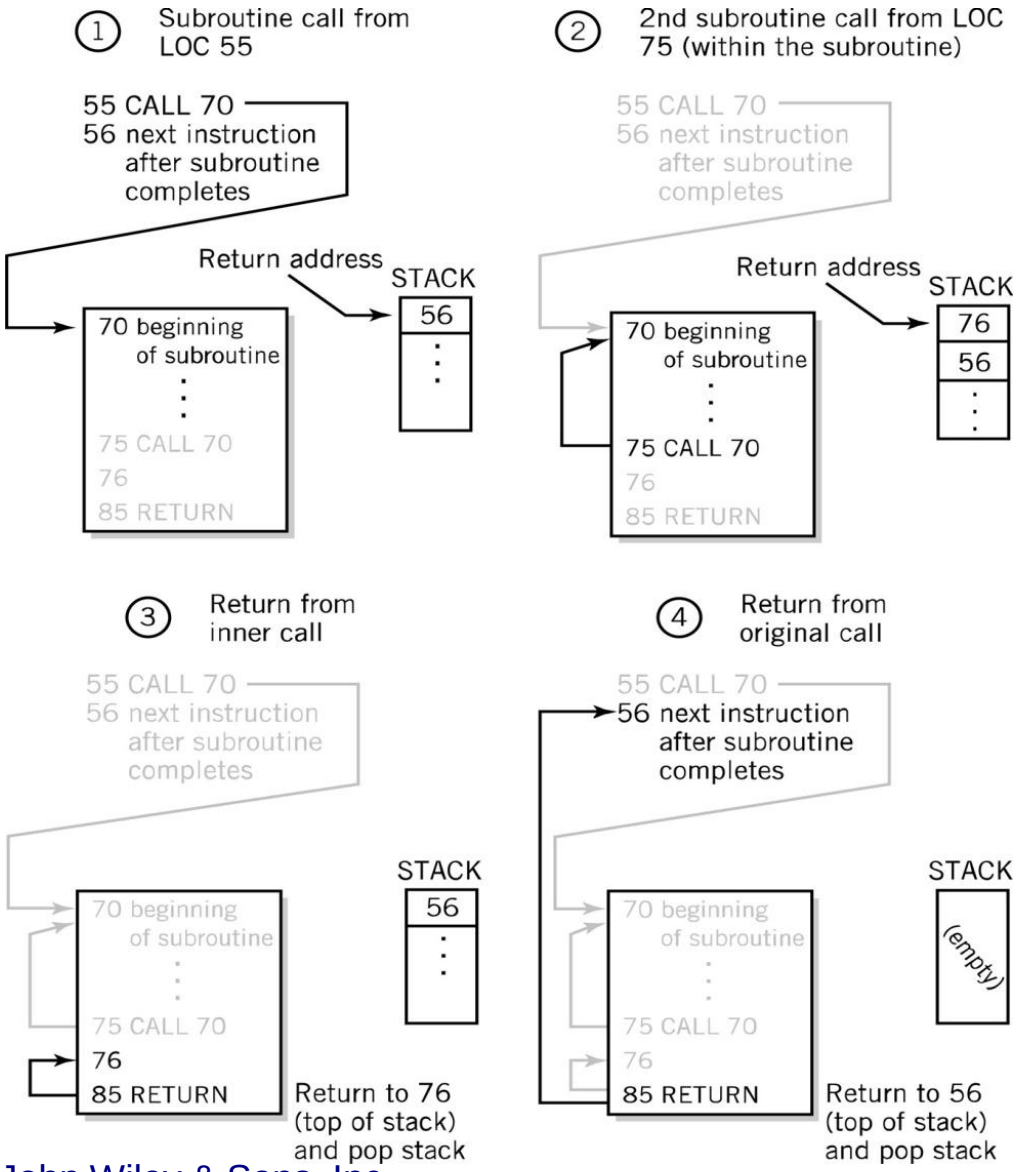


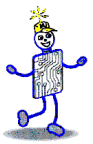
a. Subroutine called from loc.55



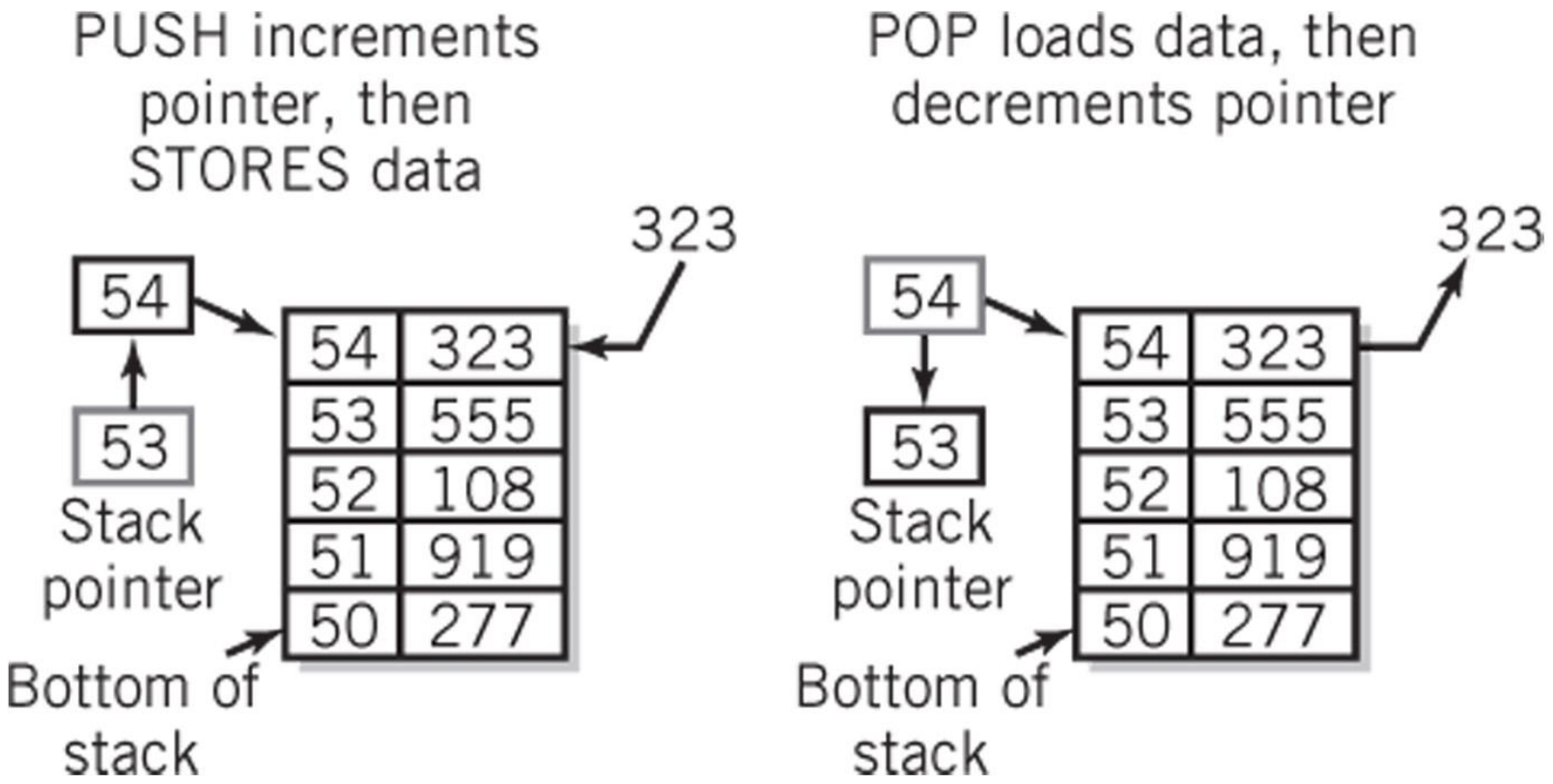
b. Subroutine re-called from 75, within the subroutine

Stack Subroutine Return Address Storage

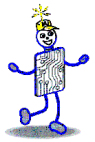




Block of Memory as a Stack: SIMD

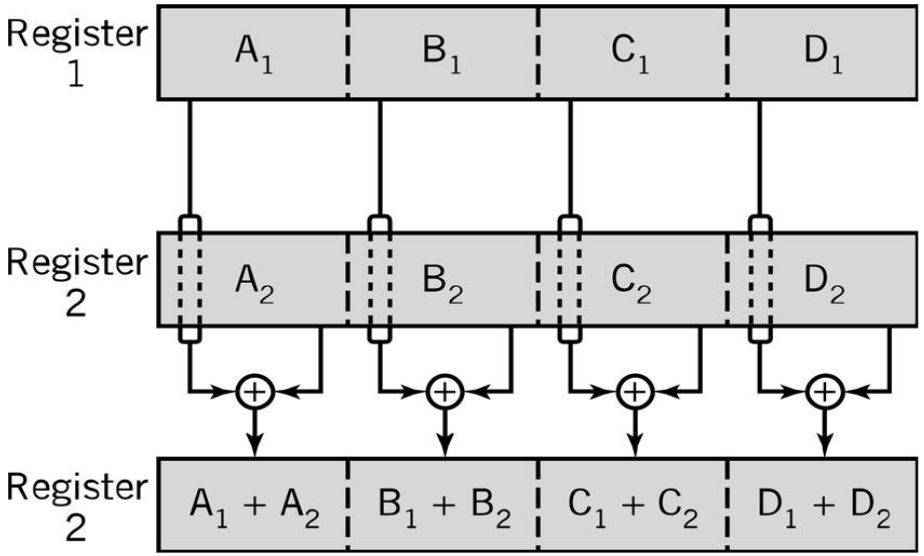


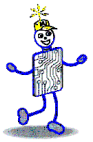
Usage: Picture Brightness, Compare Pixels, PS3 Graphics Strength etc.



Multiple Data Instructions

- Perform a single operation on multiple pieces of data simultaneously
  - SIMD: Single Instruction, Multiple Data
  - Commonly used in multimedia, *vector* and array processing applications





Instruction Elements

- OPCODE: task
- Source OPERAND(s)
- Result OPERAND

} **Addresses**

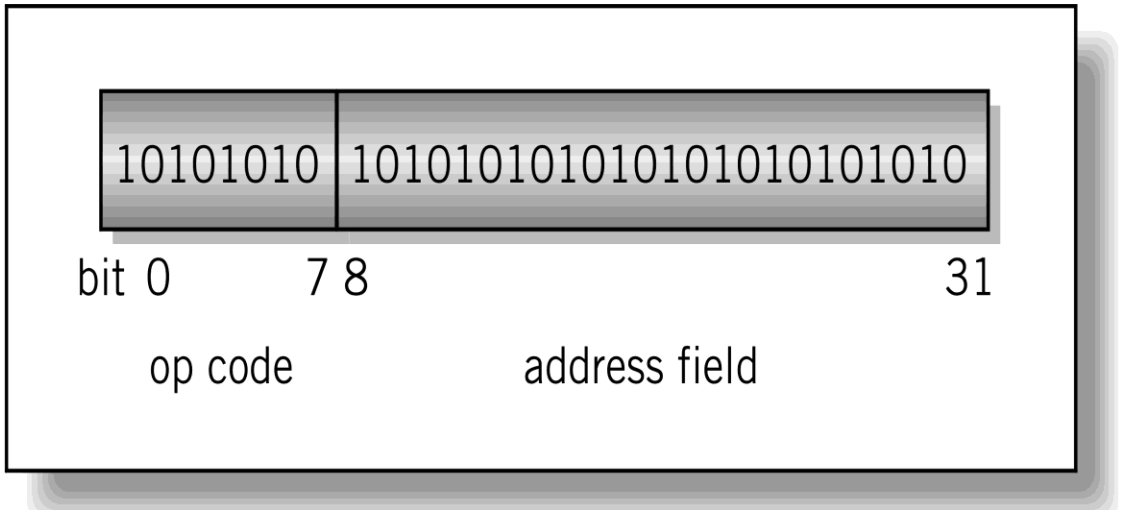
  - Location of data (register, memory)
    - Explicit: included in instruction
    - Implicit: default assumed (Accumulator)

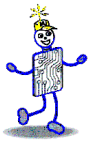
OPCODE	Source OPERAND	Result OPERAND

Instruction Format

- *Machine-specific* template that specifies
  - Length of the op code
  - Number of operands
  - Length of operands

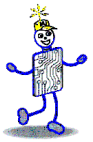
Simple 32-bit Instruction Format





Instructions

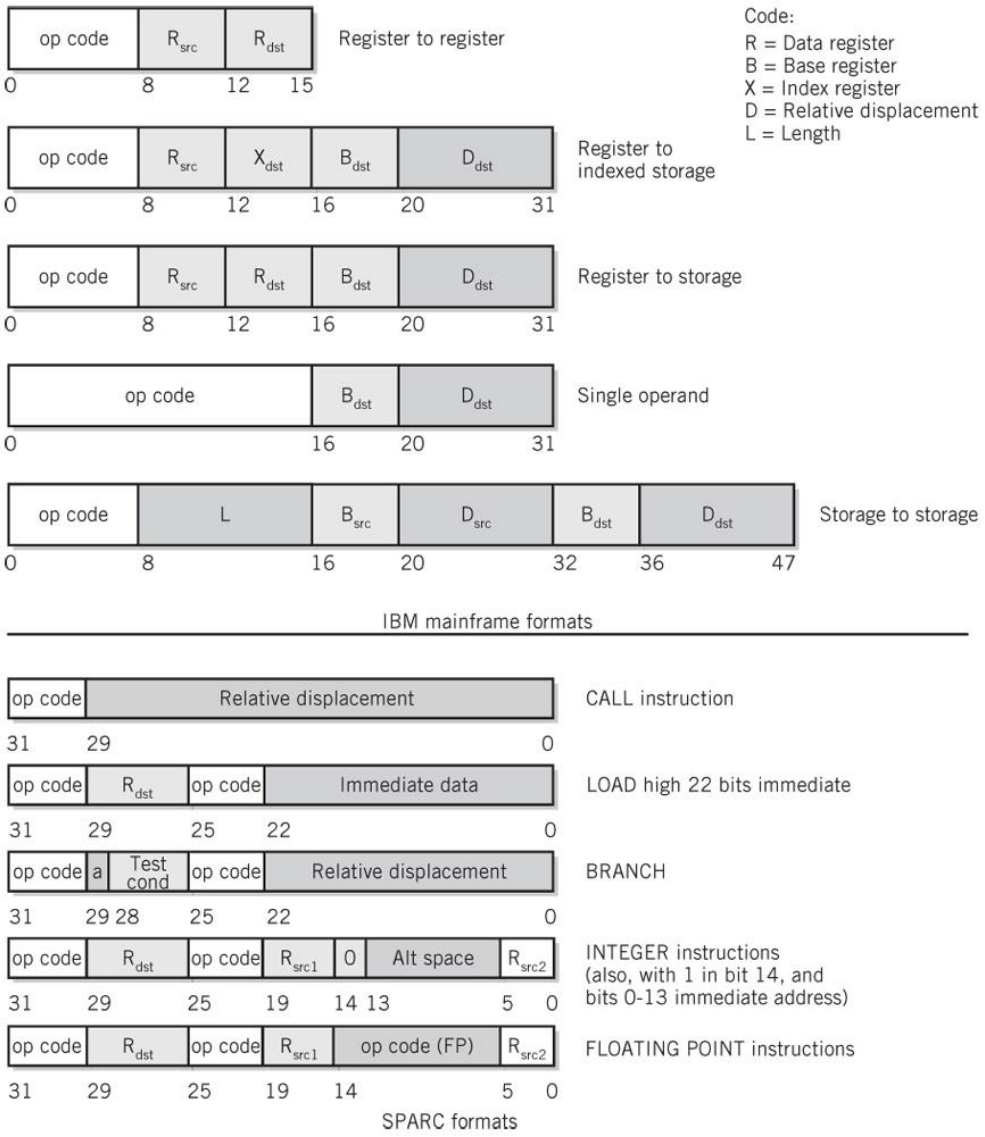
- Instruction
  - Direction given to a computer
  - Causes electrical or optical signals to be sent through specific circuits for processing
- Instruction set
  - Design defines functions performed by the processor
  - Differentiates computer architecture by the
    - Number of instructions
    - Complexity of operations performed by individual instructions
    - Data types supported
    - Format (layout, fixed vs. variable length)
    - Use of registers
    - Addressing (size, modes)

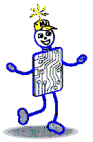


Instruction Word Size

- Fixed vs. variable size
  - Pipelining has mostly eliminated variable instruction size architectures
- Most current architectures use 32-bit or 64-bit words
- Addressing Modes
  - Direct
    - Mode used by the LMC
  - Register Deferred
  - Also immediate, indirect, indexed

Instruction Format Examples





Copyright 2010 John Wiley & Sons