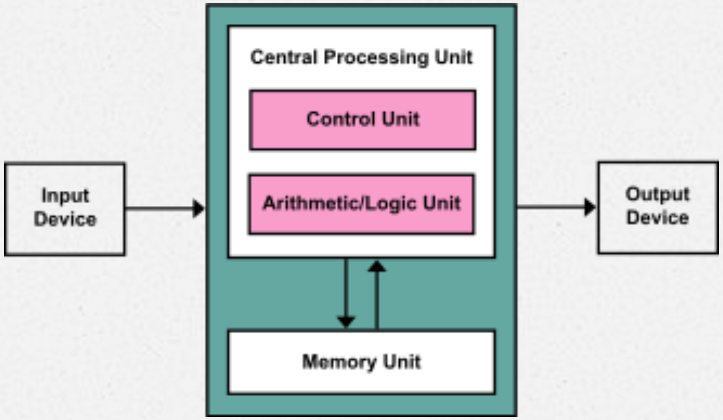
# COMP1003 Computer Organization

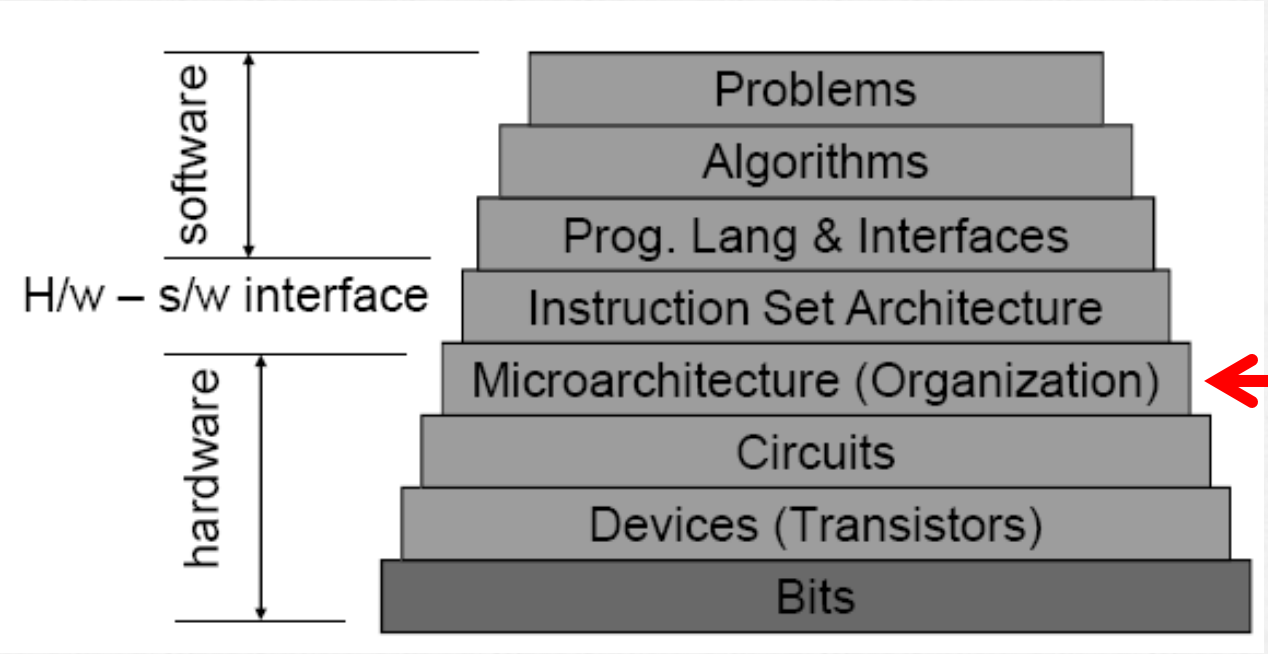
## Lecture 8 Microarchitecture




## United International College

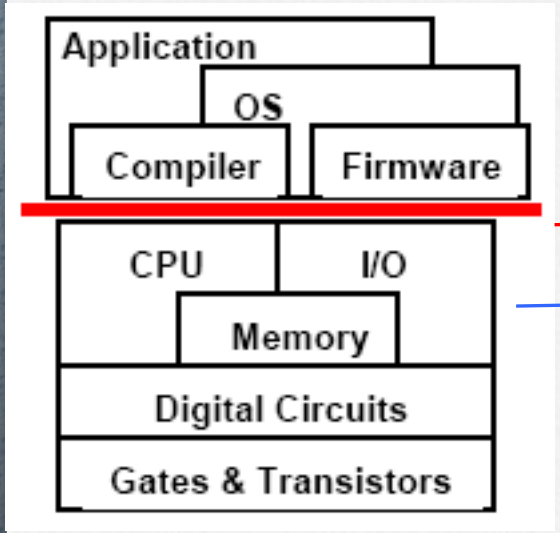
# Microarchitecture

- ISA = Instruction Set Architecture
- how to use circuits to implement the ISA?
- Microarchitecture connects circuits together.



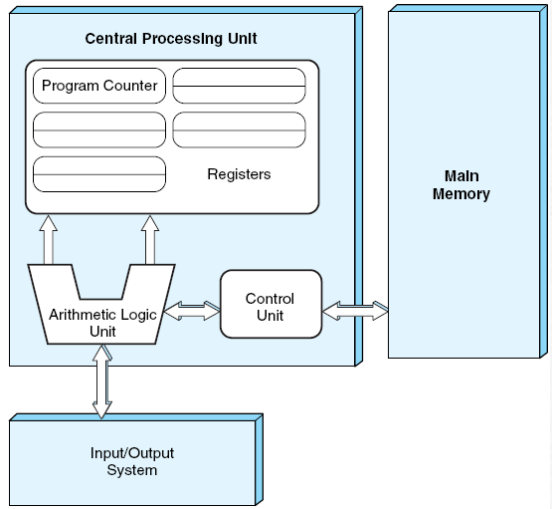
# ISA and Microarchitecture

- ISA specifies what hardware does, but not how
- Microarchitecture specifies how it does it




ISA

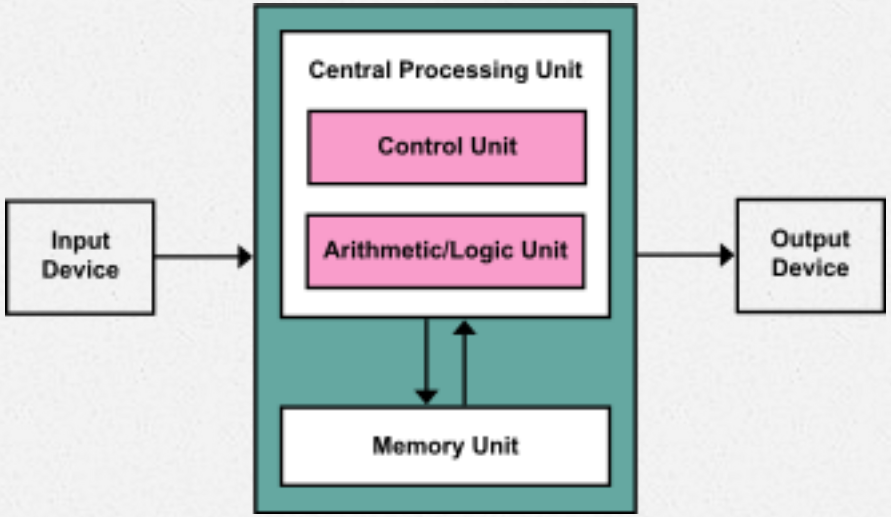
Microarchitecture



# ISA and Microarchitecture

- The ISA is the interface of a processor as seen by an assembly language programmer or compiler writer
- Micro-architecture transfers the ISA into an implementation
- For a given ISA, there might be many different microarchitectures
- An architecture is a collection of circuits connected together

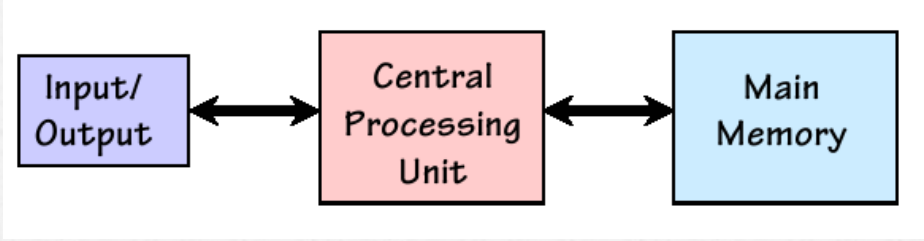
# The von Neumann Architecture




John von Neumann (1903 – 1957)

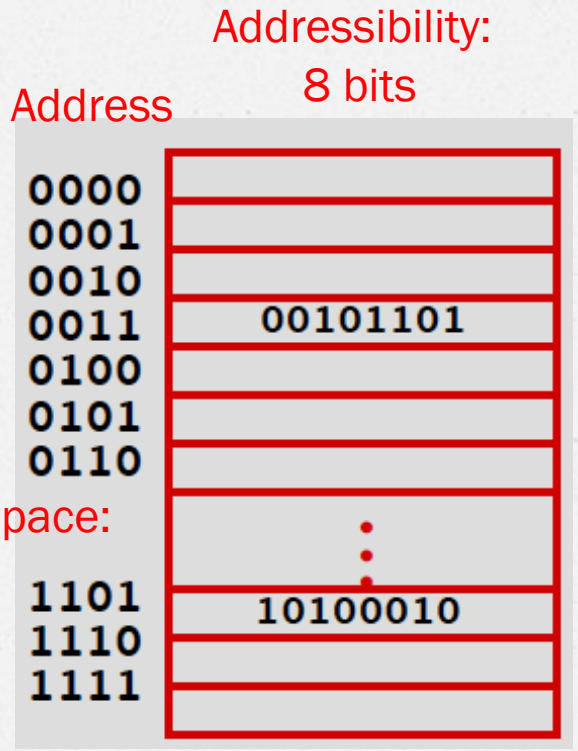
# The Stored Program Computer

- Memory stores not only data, but coded instructions that make up a computer program
- CPU fetches and executes – interprets - successive instructions of the program
- Program is simply data for the interpreter – as in a Universal Turing Machine!
- Single expandable resource pool – main memory – constrains both data and program size



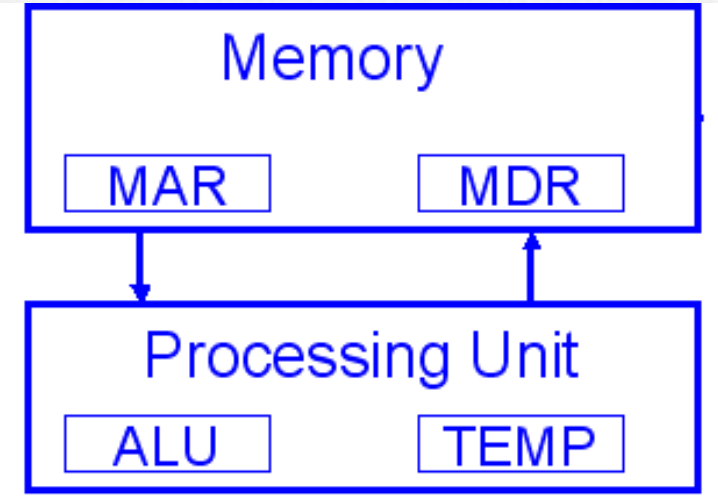
# Memory

- Array of stored bits
- Address
  - unique (n-bit) identifier of location
- Contents
  - m-bit value stored in location
- Basic Operations
  - LOAD
  - read a value from a memory location
  - STORE
  - write a value to a memory location



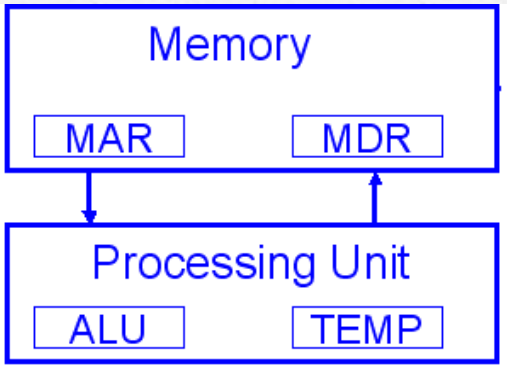
# Interface to Memory

- How does CPU get data to/from memory?
  1. MAR: Memory Address Register (D flip-flops)
  2. MDR: Memory Data Register (D flip-flops)



# Read to & Write from Memory

- To read a location (A):
  1. Write the address (A) into the MAR.
  2. Send a “read” signal to the memory.
  3. Read the data from MDR.
- To write a value (X) to a location (A):
  1. Write the data (X) to the MDR.
  2. Write the address (A) into the MAR.
  3. Send a “write” signal to the memory.

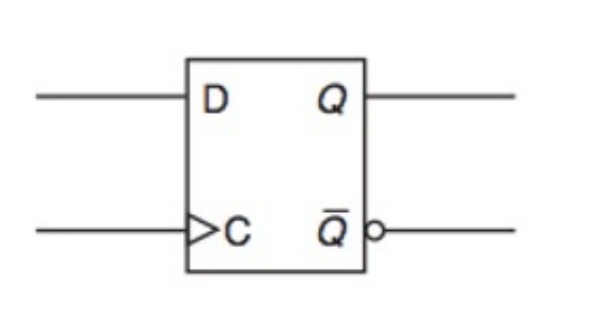


# CPU

- The brain of the computer
- It is the part that actually executes the machine instructions
- Inside the CPU
  - Data path
    - Registers
    - CPUs
  - Control Path
    - IR (instruction register), PC (program counter), FSM (finite state machine)

# Registers

- Small, temporary storage for operands and intermediate results
- Why registers?
  - Closer to processing unit, allow quicker access to intermediate results instead of going to memory
- Using D flip-flops to implement

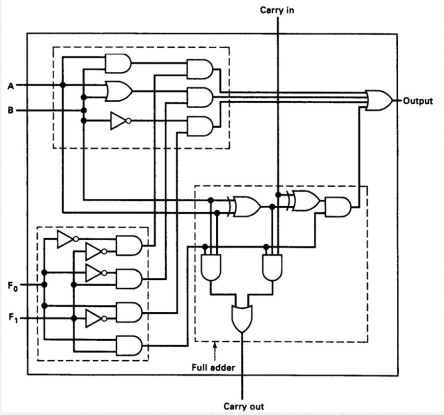



# ALU

- Arithmetic Logic Unit
  - Perform arithmetic and logic operations (AND, NOT, ADD) on values stored in registers

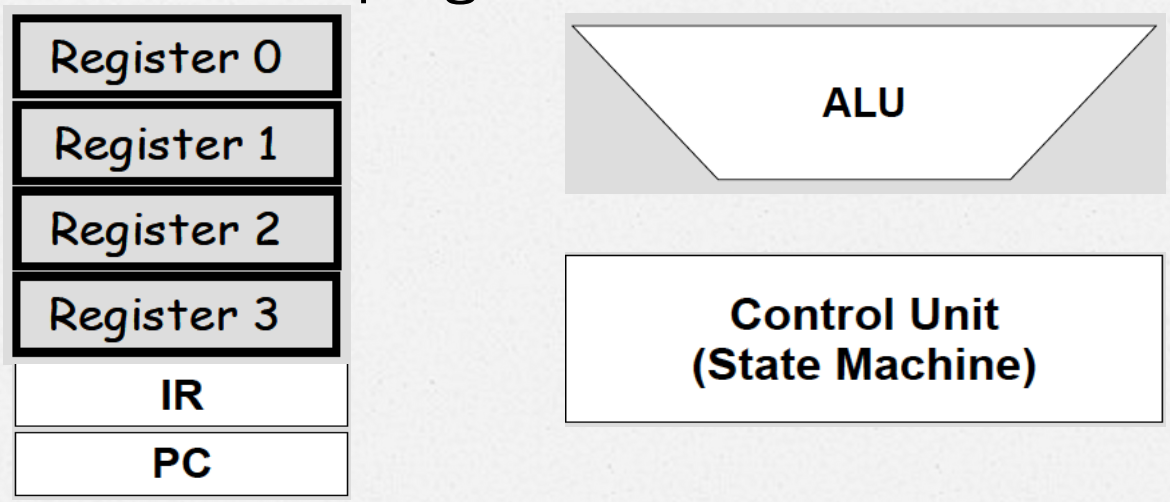


1-bit ALU



# Control Unit, IR and PC

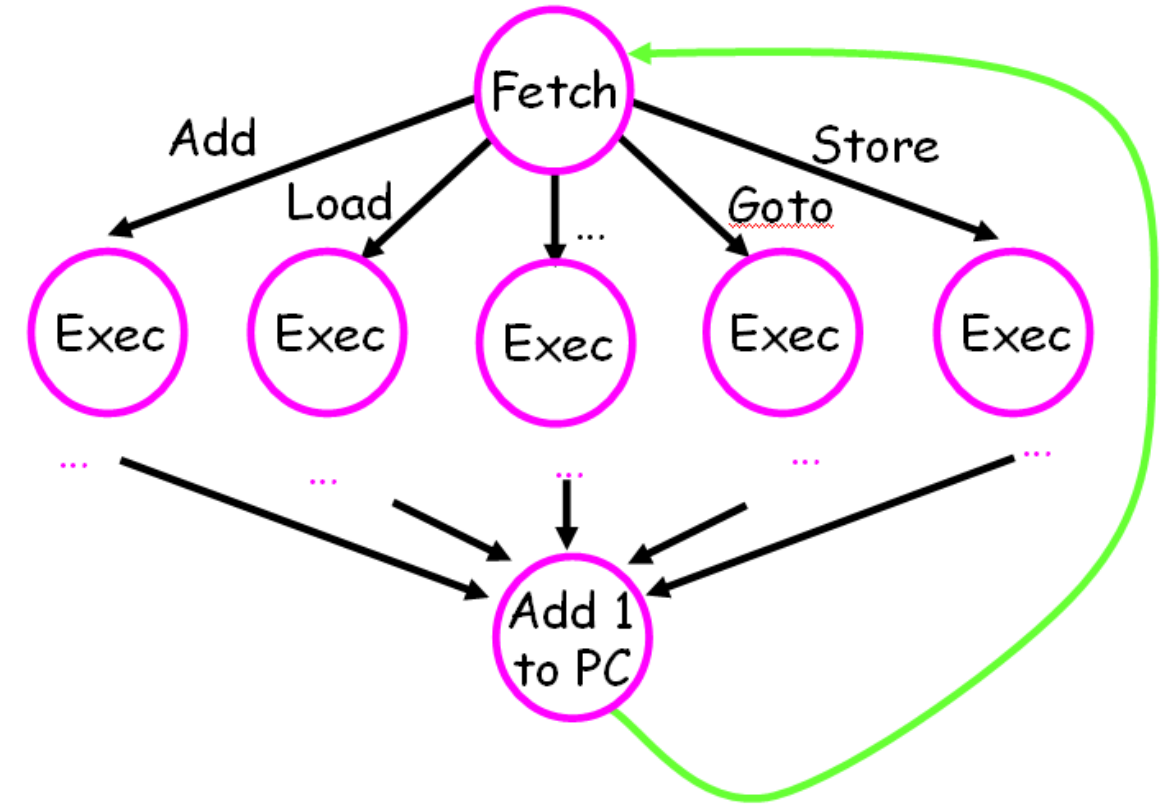
- IR: instruction register contains the current instruction
- PC: program counter contains the address of the next instruction (a better name may be instruction pointer)
- Control Unit: a finite state machine coordinates execution of the program



# Control Unit

- Coordinates execution of the program
- Reads an instruction from memory
- Interprets the instruction, generating signals that tell the other components what to do

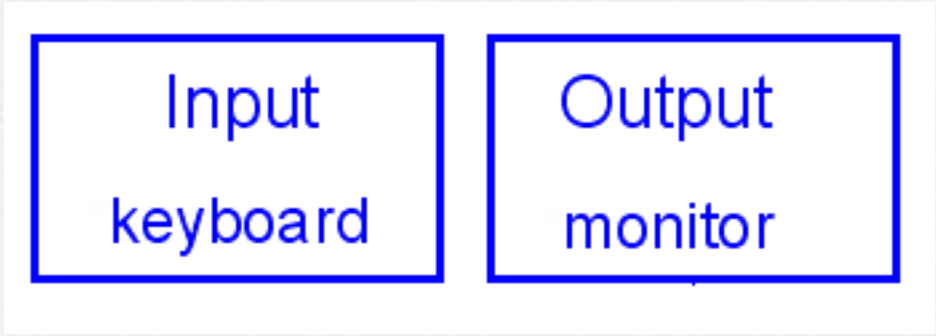
# Control Unit as a Finite State Machine



# LC3 FSM Diagram

# Input and Output

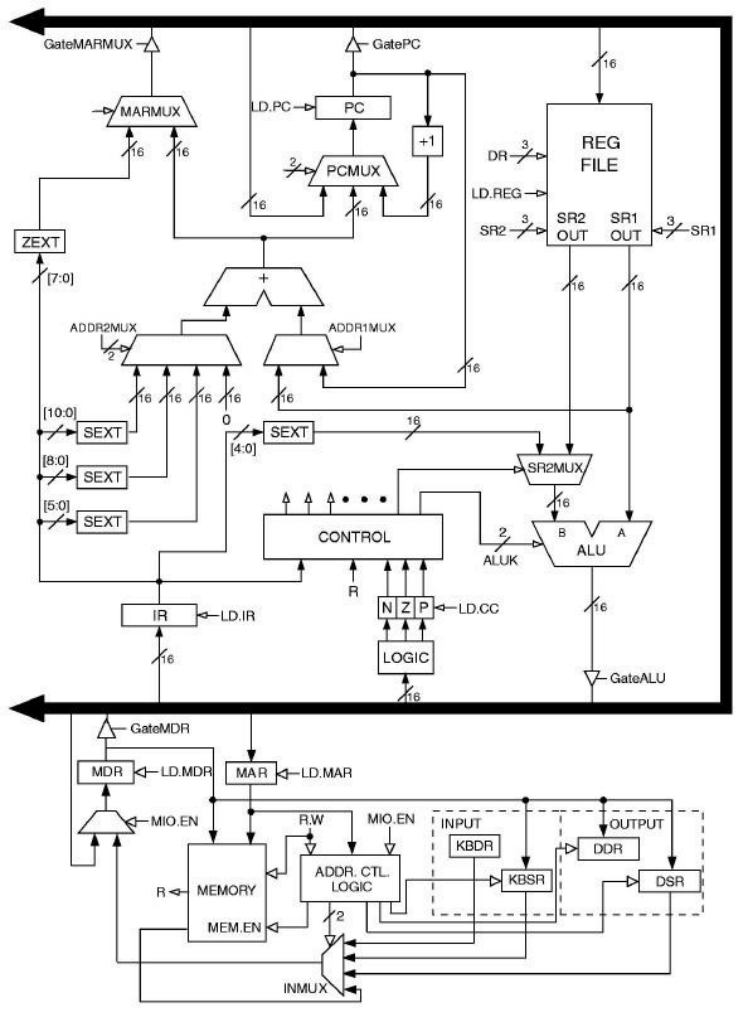
- Devices for getting data into and out of memory
- Each device has its own interface, usually a set of registers like the memory's MAR and MDR
  - keyboard: data register (KBDR) and status register (KBSR)
  - console: data register (DDR) and status register (DSR)



# The LC (Little Computer)-3

- Memory
  - Address space: $2^{16}$ memory locations
  - Addressability: 16-bit addressable, MAR and MDR
- Input and Output
  - Keyboard (KBDR & KBSR)
  - Monitor (DDR & DSR)
- The Processing Unit
  - ALU (ADD, AND, NOT)
  - 8 generał register (R0, R1, …, R7)
- The Control Unit
  - The FSM that directs all the activities
  - The instruction Register (IR) and Program
  - Program Counter (PC)

# Data Path of the LC-3

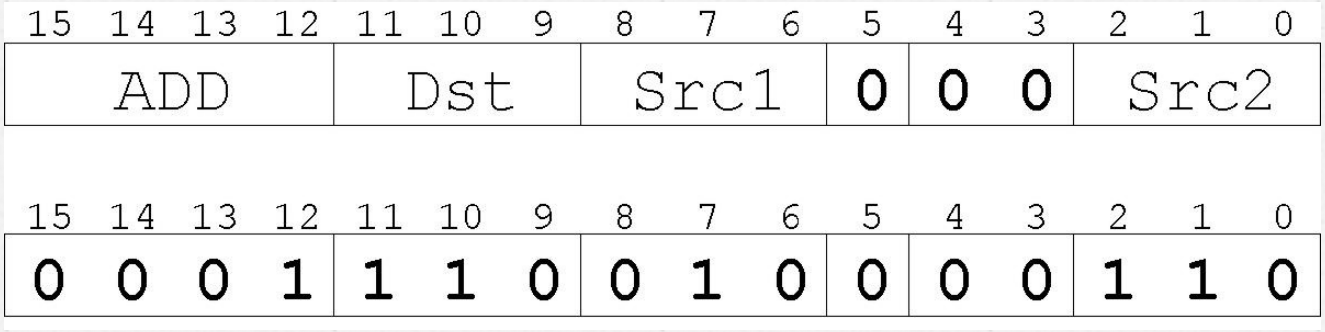


# Instruction

- The instruction is the most basic unit of computer processing.
- One instruction specifies two things:
  - opcode: operation to be performed
  - operands: data/locations to be used for operation
- An instruction is encoded as a sequence of bits (just like data!)
- Control unit interprets instruction
- A computer's instructions and their formats is known as its Instruction Set Architecture (ISA).

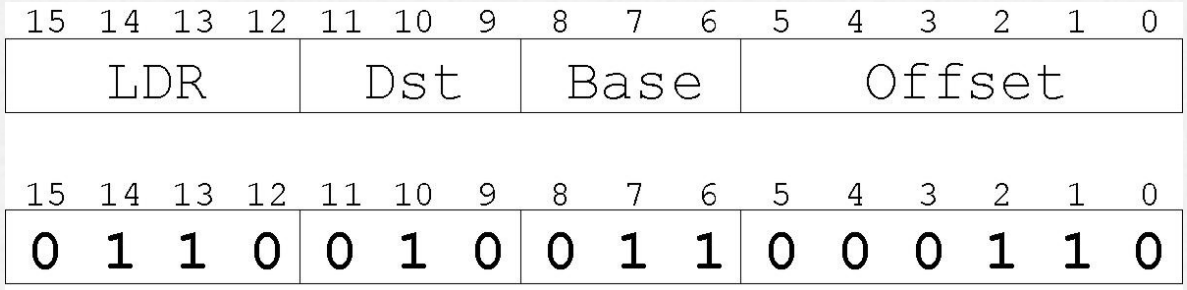
# LC-3's ADD Instruction

- Opcode: bits[15:12] 0001
- Operands: Src1, Src2, Dst
  - Src1 + Src2 →Dst
  - R6 = R6 + R2



# LC-3's LDR Instruction

- Opcode: bits [15:12] 0110
- Operands: Src, Dst
  - Load the value in memory location Src into register Dst
  - Move [Base + Offset] to Dst
  - Load memory content at address (R3 + 6) to R2



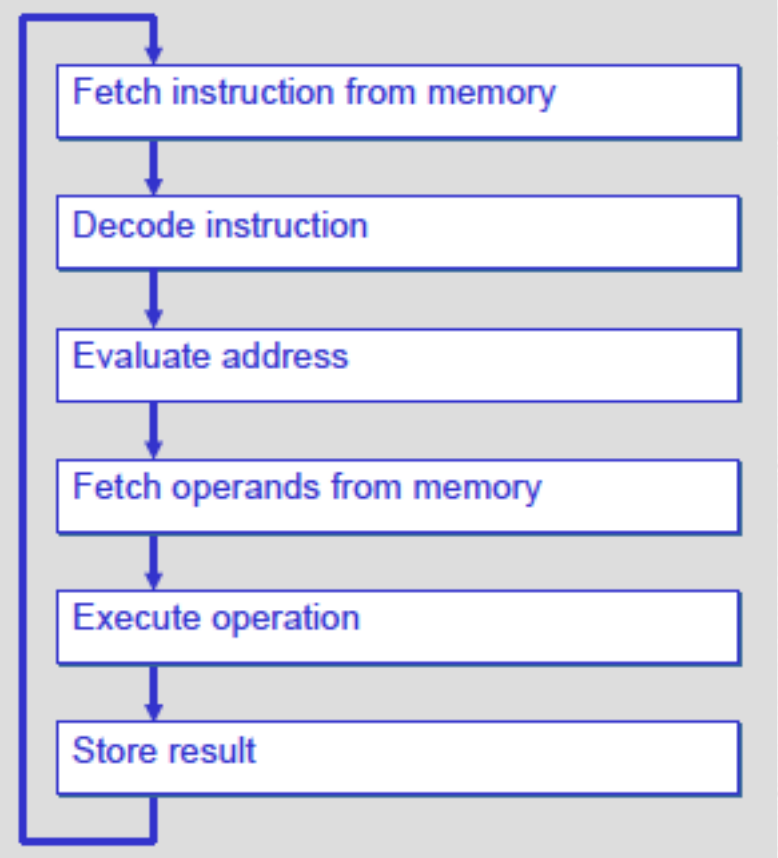
# Micro-Architecture Level

- Describes a large number of details that are hidden in the programming model
  - Constituent parts of the processor and
  - How these interconnect and interoperate to implement the architectural specification
- Computer = processing unit + memory system + I/O
- Processing unit = control + datapath
- Control = finite state machine
  - Inputs = machine instruction, datapath conditions
  - Outputs = register transfer control signals, ALU operation codes
  - Instruction interpretation = instruction fetch, decode, execute, write
- Datapath = functional units + registers
  - All the logic used to process information
    - Functional units = ALU, multipliers, dividers, etc.
    - Registers = program counter (PC), instruction register (IR), storage registers

# Control Unit

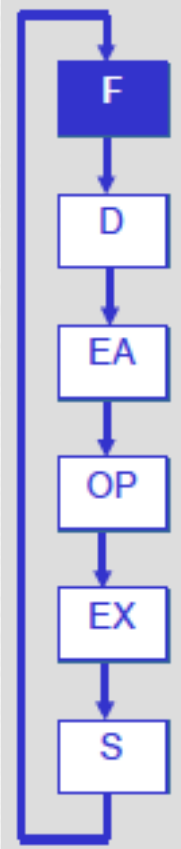
- Circuitry that
  - controls the flow of information through the processor, and
  - Coordinates activities of the other units within it.
- Is a FSM
  - States enumerate all possible configurations the machine can be in
  - Using the opcode information & some other inputs (e.g. Condition Code, Interrupt Signal) determines next state and output
    - Decides for each stage in instruction processing cycle
    - Which registers/memory location are enabled?
    - Which operation should ALU perform?
    - Choose ALU output or Memory Output?

# Instruction Processing



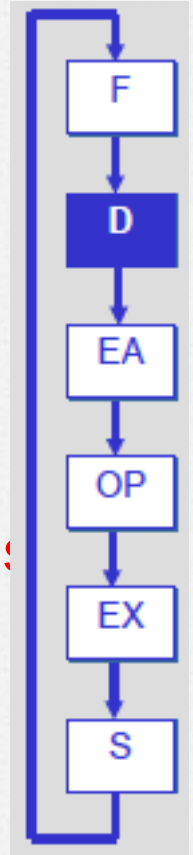
# Fetch

- Load next instruction (at address stored in PC) from memory into Instruction Register (IR).
  - Load contents of PC into MAR.
  - Send “read” signal to memory.
  - Read contents of MDR, store in IR.
- Then increment PC, so that it points to the next instruction in sequence.
  - PC becomes PC+1.



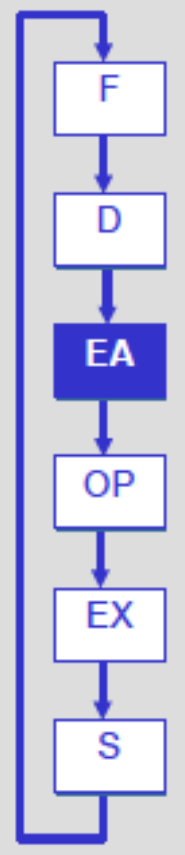
# Decode

- First identify the opcode.
  - In LC-3, this is always the first four bits of instruction.
  - A 4-to-16 decoder asserts a control line corresponding to the desired opcode.
- Depending on opcode, identify other operands from the remaining bits.
  - for LDR, last six bits is offset
  - for ADD, last three bits is source operand #2



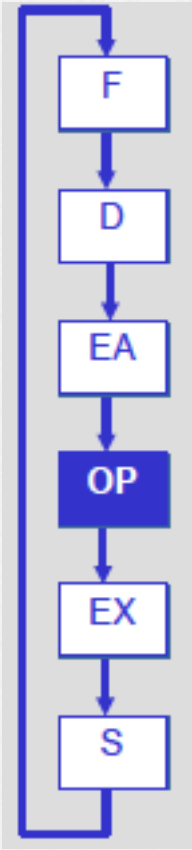
# Evaluate Address

- For instructions that require memory access, compute address used for access.
- Examples:
  - add offset to base register (as in LDR)
  - add offset to PC (or to part of PC)



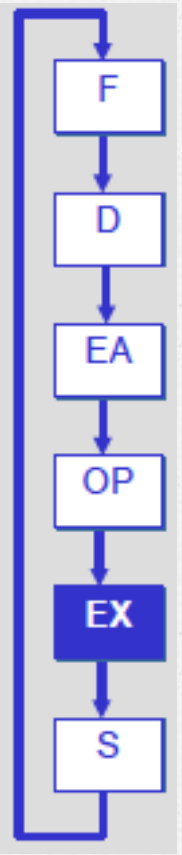
# Fetch Operands

- Obtain source operands needed to perform operation.
- Examples:
  - load data from memory (LDR)
  - read data from register file (ADD)



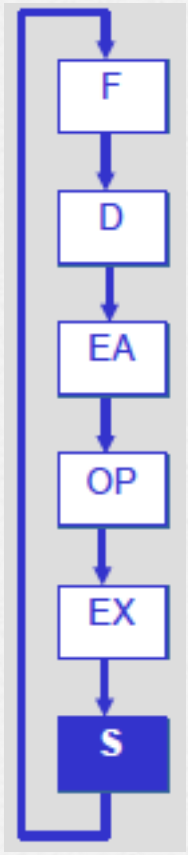
# Execution

- Perform the operation, using the source operands.
- Examples:
  - send operands to ALU and assert ADD signal
  - do nothing (e.g., for loads and stores)



# Store

- Write results to destination (register or memory)
- Examples:
  - result of ADD is placed in destination register
  - result of memory load is placed in destination register
  - for store instruction, data is stored to memory

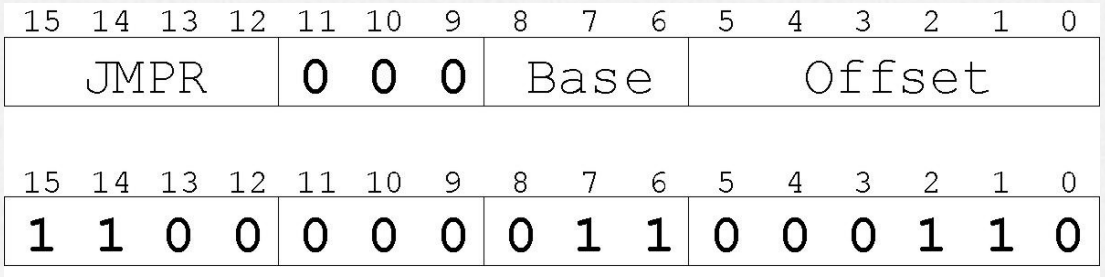


# Changing the Sequence of Execution

- In the FETCH phase, PC is incremented by 1 automatically (counter)
- What if we don't want to execute the instruction that follows this one?
  - Examples: if-then, loop, function call
- Need special instruction that changes the content of PC
  - Jumps (unconditionals)
  - Branches (conditional)

# LC-3's Jump Instruction

- Set the PC to the value obtained by adding an offset to a register.
- This becomes the address of the next instruction to fetch.




Add the value of 6 (offset) to the contents of R3 (Base), and load the result into the PC

# Instruction Processing Summary

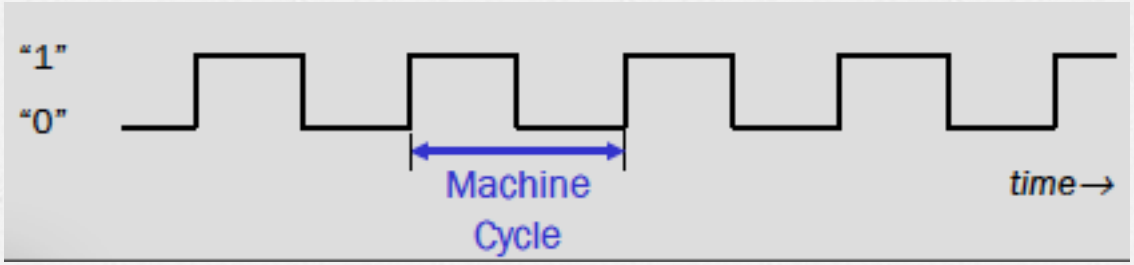
- Three basic kinds of instructions:
  - computational instructions (ADD, AND, …)
  - data movement instructions (LD, ST, …)
  - control instructions (JMP, BRnz, …)
- Six basic phases of instruction processing:

  F → D → EA → OP → EX → S

  - not all phases are needed by every instruction
  - phases may take variable number of machine cycles

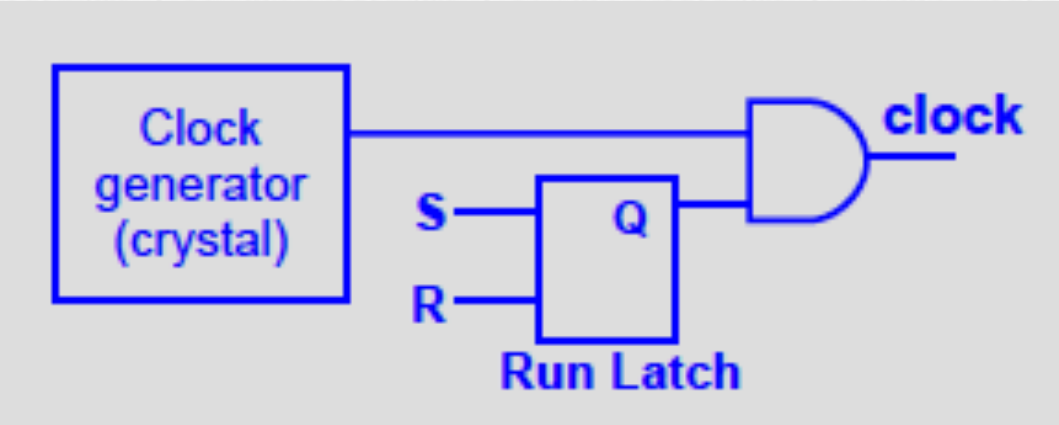
# Driving Force: the Clock

- The clock is a signal that keeps the control unit moving.
  - At each clock "tick," control unit moves to the next machine cycle -- may be next instruction or next phase of current instruction.



# Stopping the Computer

- Stopping the instruction cycle requires stopping the clock



# What's Next

- The ISA of the LC3 Computer