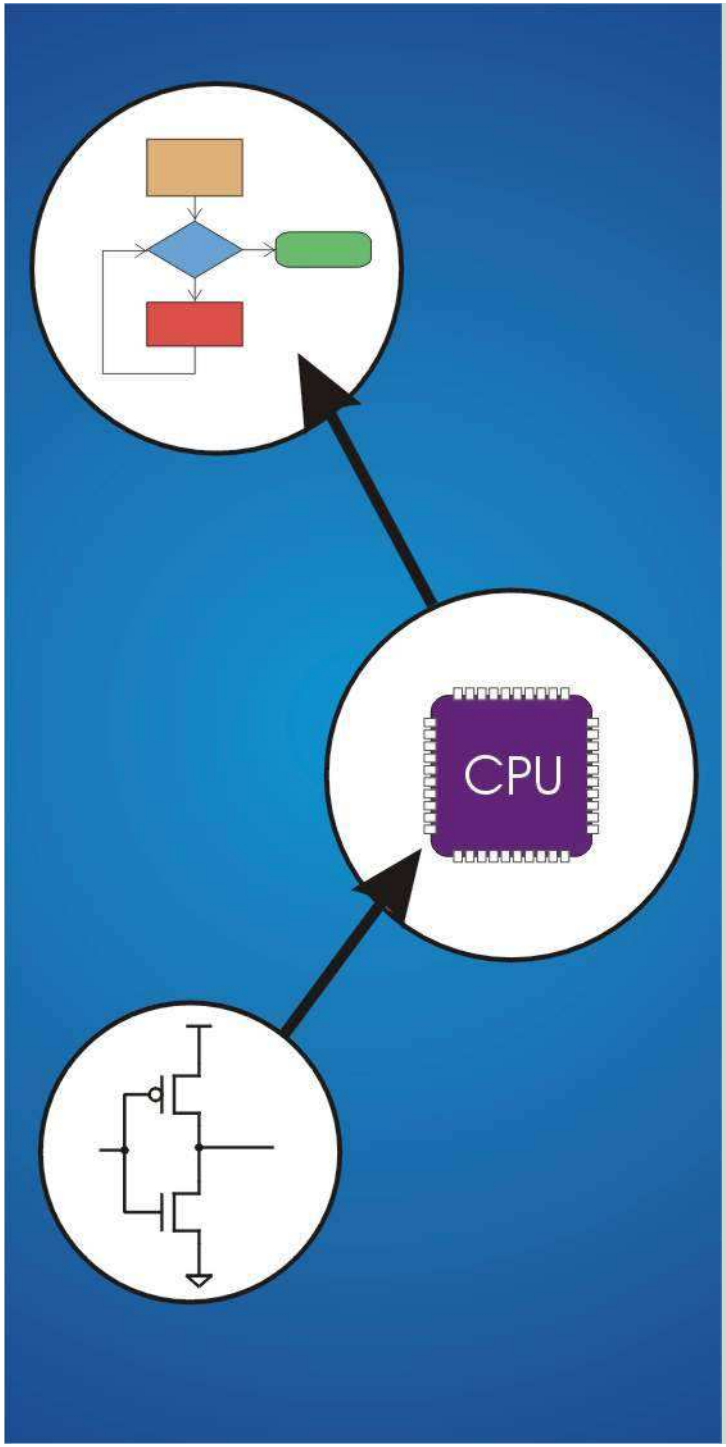
Digital Logic Recap

- Combinational logic
  - Gates
  - Functional blocks: MUX, Decoder, Adders etc
- Sequential logic
  - Storage elements
  - Registers: row of storage elements + stuff
  - Memory systems: arrays, need addressing
  - Clock for synchronization
  - 2 GHz frequency => 1/2G i.e. 0.50 nanosec clock period



Chapter 4
The Von Neumann Model

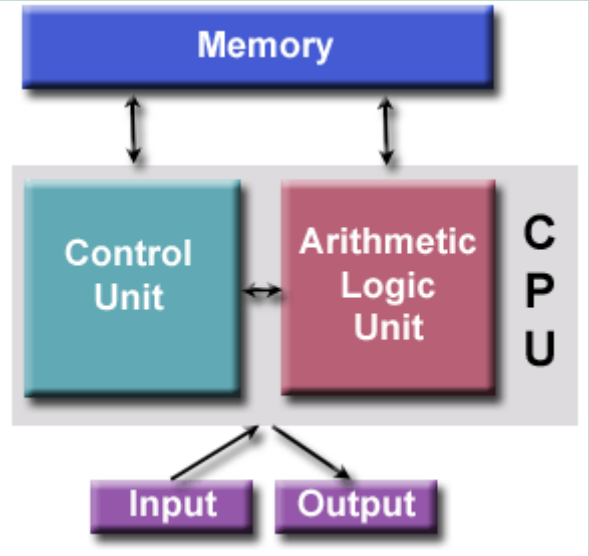
Original slides from Gregory Byrd, North Carolina State University

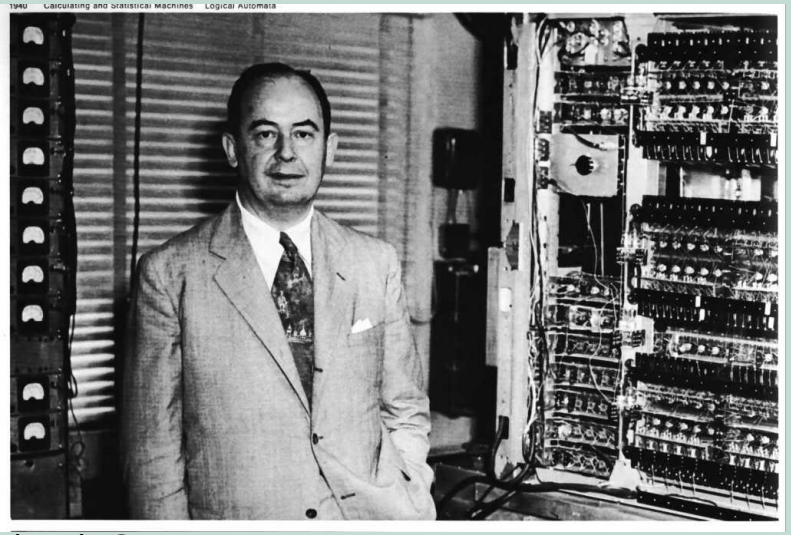
Modified slides by C. Wilcox, Y. Malaiya
Colorado State University

von Neumann Architecture

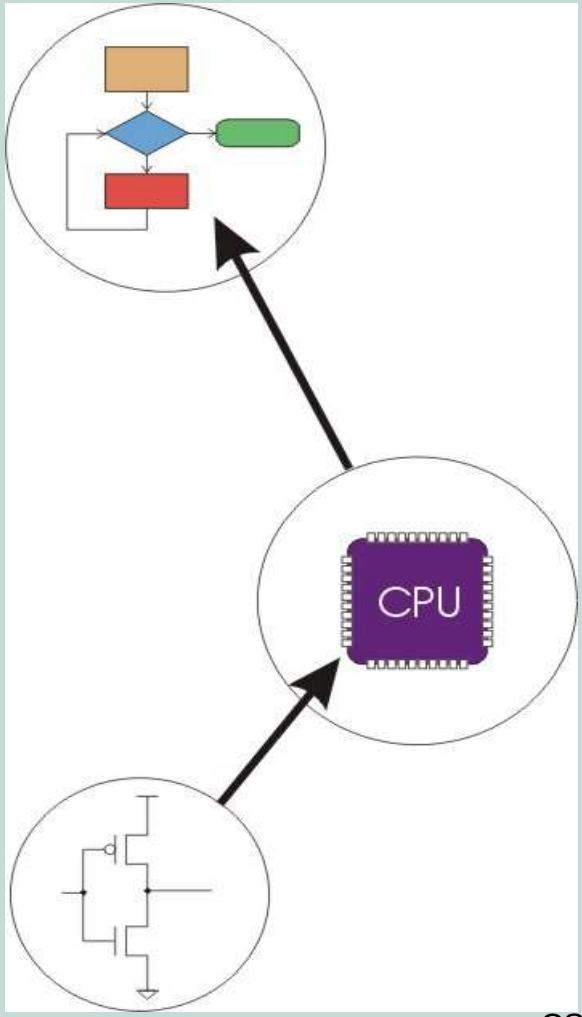
Four main components:

- Memory
- Control Unit
- Arithmetic Logic Unit
- Input/Output





Computing Layers



Problems

Algorithms

Language

Instruction Set Architecture

Microarchitecture

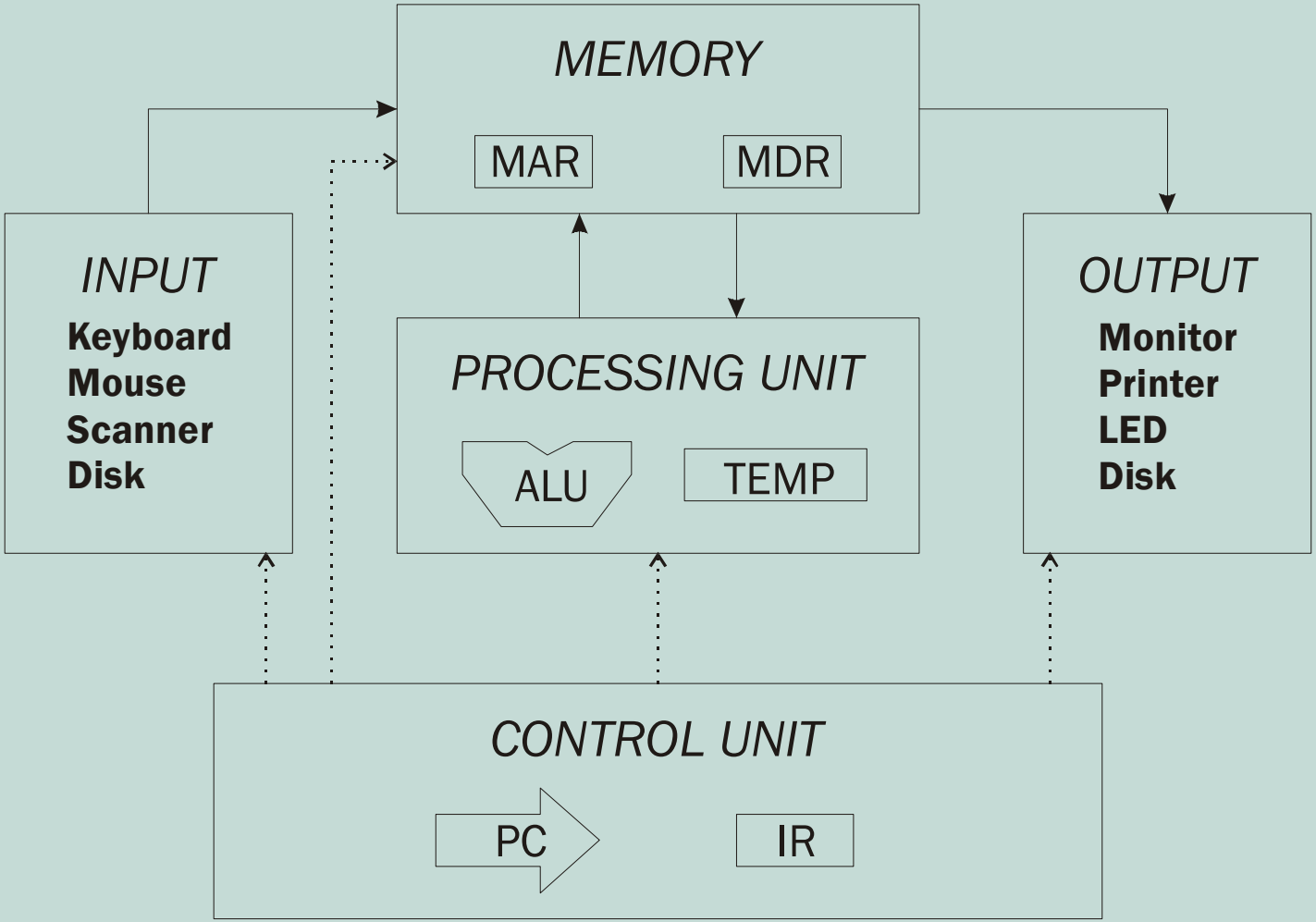
Circuits

Devices

The Stored Program Computer

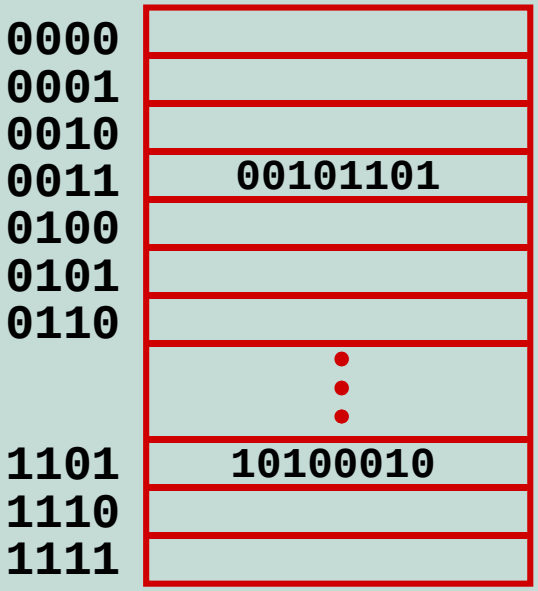
- 1939 John V. Atanasoff, Iowa State College
- 1941 Konrad Zuse, Berlin, program controlled computer
- 1943: ENIAC
  - Eckert and Mauchly, U Penn
  - Hard-wired program -- settings of dials and switches.
- 1945: John von Neumann, Princeton
  - wrote a report on the stored program concept
- 1964: Supercomputer, Cray
- 1971: Intel Microprocessor: processor on a single chip
- 1997: Smart Phone
- Questions on who was first, controversies, law suits …

Von Neumann Model



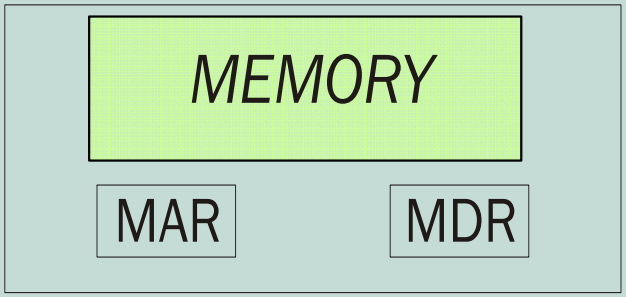
Memory

- *Organization*
  - 2^k x m array of stored bits
- Address
  - unique (k-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

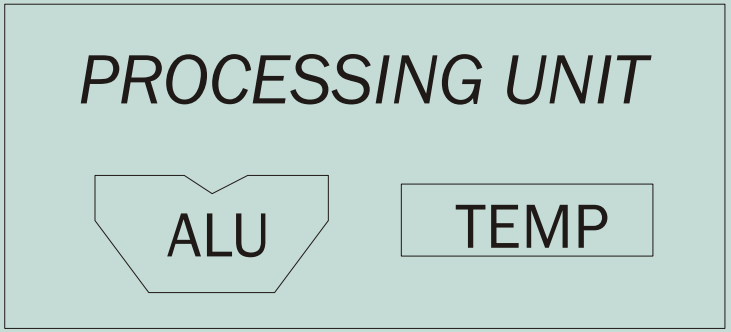
- How does CPU get data to/from memory?
  - MAR: Memory Address Register
  - MDR: Memory Data Register



- To LOAD 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 STORE 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.

Processing Unit

- Functional Units
  - ALU = Arithmetic and Logic Unit
  - could have many functional units. (multiply, square root, …)
  - LC-3 performs ADD, AND, NOT
- Registers
  - Small, temporary storage
  - Operands and results of functional units
  - LC-3 has eight registers (R0, …, R7), each 16 bits wide
- Word Size
  - number of bits processed by ALU in one instruction
  - also width of registers
  - LC-3 is 16 bits



Input and Output

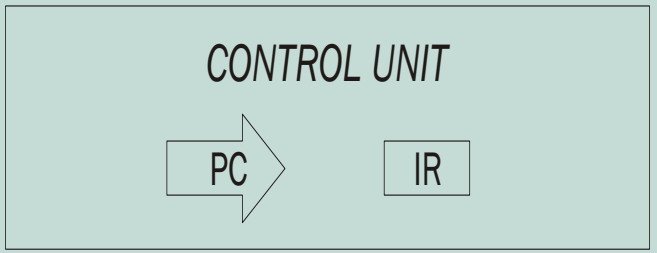
- Devices for getting data into and out of computer memory
- Each device has its own interface, usually a set of registers like the memory's MAR and MDR

INPUT	*OUTPUT*
Keyboard	**Monitor**
Mouse	**Printer**
Scanner	**LED**
Disk	**Disk**

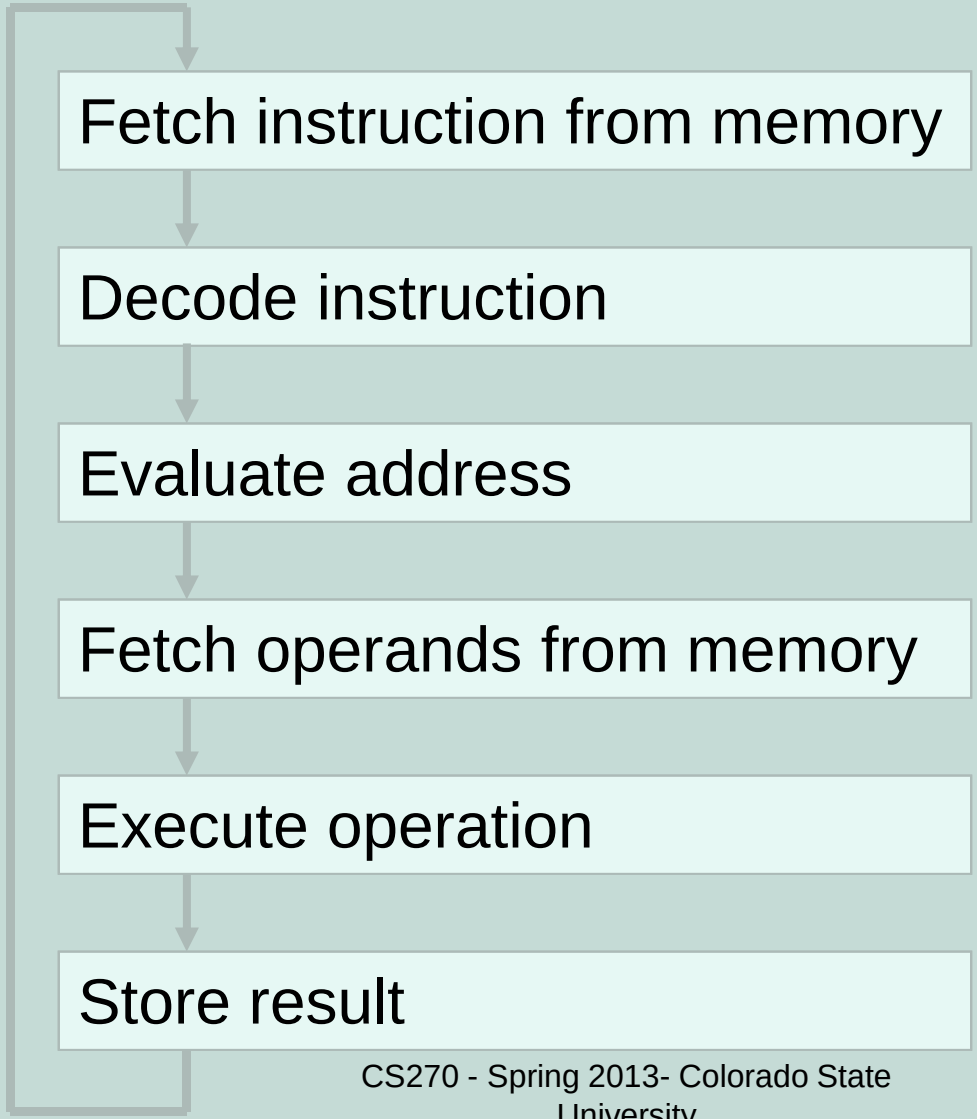
  - LC-3 supports keyboard (input) and monitor (output)
  - keyboard: data (KBDR) and status (KBSR) registers
  - monitor: data register (DDR) and status register (DSR)
- Some devices provide both input and output
  - disk, network
- Program that controls access to a device is usually called a *driver*.

Control Unit

- Orchestrates execution of the program
- Instruction Register (IR) contains the *current instruction*.
- Program Counter (PC) contains the *address* of the next instruction to be executed.
- Control unit:
  - reads an instruction from memory
    - the instruction's address is in the PC
  - interprets the instruction, generating signals that tell the other components what to do
    - an instruction may take many *machine cycles* to complete



Instruction Processing

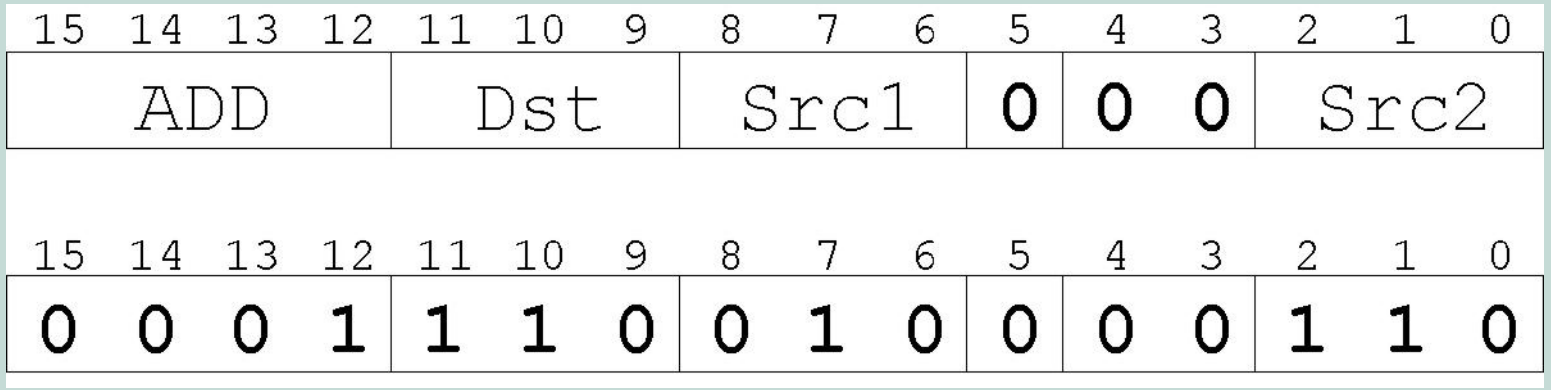


Instruction

- The instruction is the fundamental unit of work:
  - *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!)*
  - Often, but not always, instructions have a fixed length, such as 16 or 32 bits.
  - Control unit interprets instruction: generates sequence of control signals to carry out operation.
  - Operation is either executed completely, or not at all.
- A computer's instructions and their formats is known as its *Instruction Set Architecture (ISA)*.

Example: LC-3 ADD Instruction

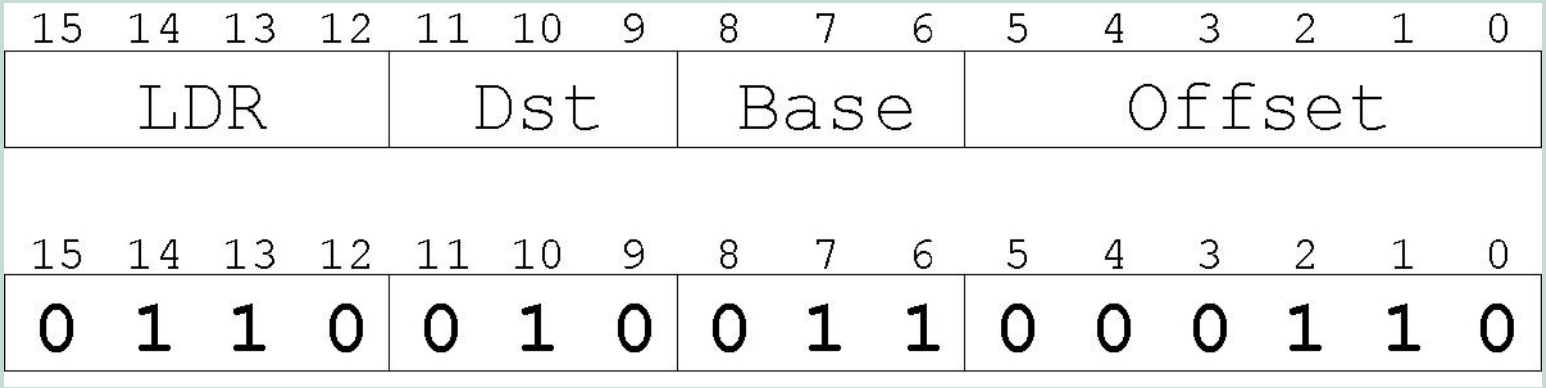
- LC-3 has 16-bit instructions.
  - Each instruction has a four-bit opcode, bits [15:12].
- LC-3 has eight *registers* (R0-R7) for temporary storage.
  - Sources and destination of ADD are registers.



"Add the contents of R2 to the contents of R6, and store the result in R6."

Example: LC-3 LDR Instruction

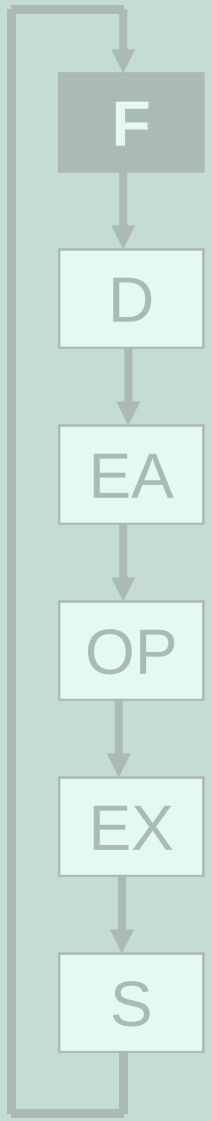
- Load instruction -- reads data from memory
- Base + offset mode:
  - add offset to base register -- result is memory address
  - load from memory address into destination register



"Add the value 6 to the contents of R3 to form a memory address. Load the contents of that memory location to R2."

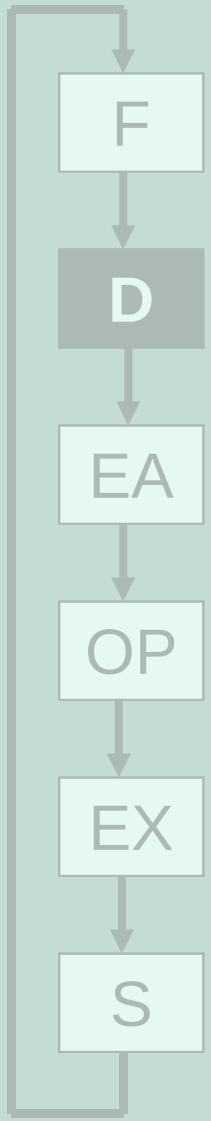
Instruction Processing: FETCH

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



Instruction Processing: DECODE

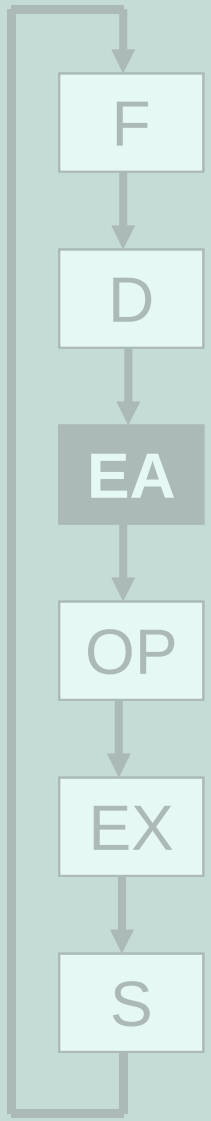
- First identify the opcode.
  - In LC-3, this is always the first four bits of instruction.
  - A 4-to-16 decoder asserts a control line corresponding to the desired opcode.
- Depending on opcode, identify other operands from the remaining bits.
  - Example:
    - for LDR, last 6 bits is offset
    - for ADD, last 3 bits is source operand #2



Instruction Processing: EVALUATE ADDRESS

- For instructions that require memory access, compute address used for access.

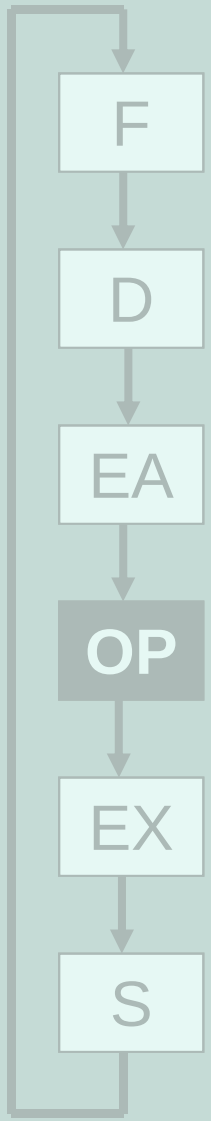
- Examples:
  - add offset to base register (as in LDR)
  - add offset to PC
  - add offset to zero



Instruction Processing: FETCH OPERANDS

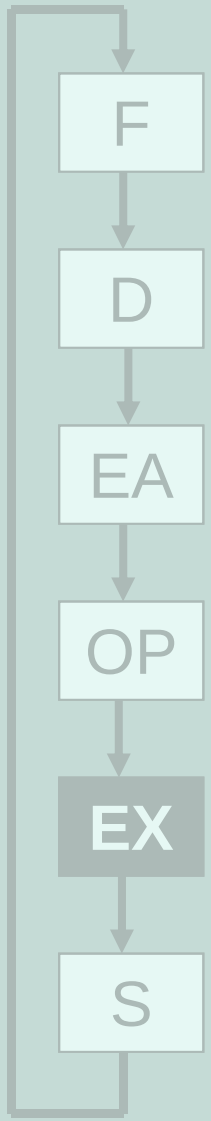
- Obtain source operands needed to perform operation.

- Examples:
  - load data from memory (LDR)
  - read data from register file (ADD)



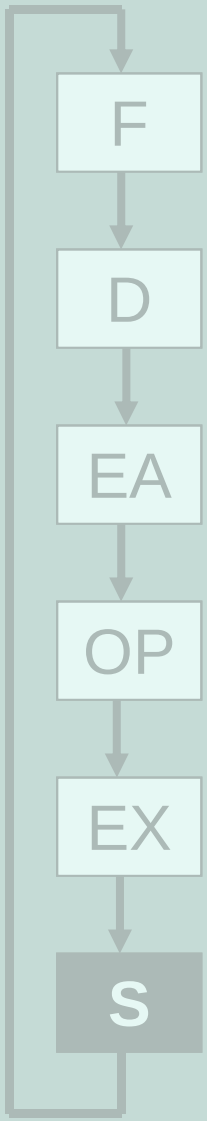
Instruction Processing: EXECUTE

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



Instruction Processing: STORE RESULT

- 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
    - write address to MAR, data to MDR
    - assert WRITE signal to memory

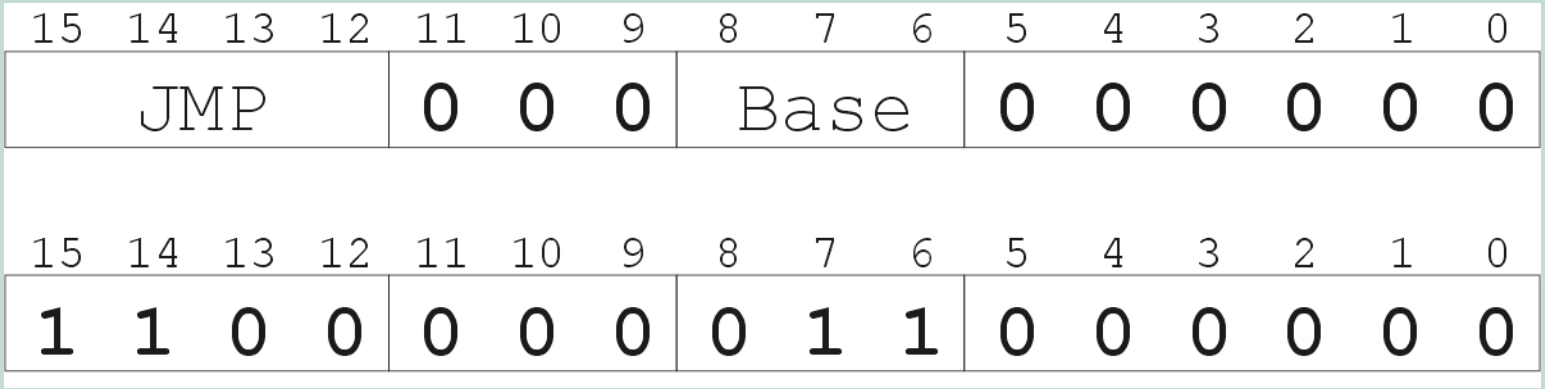


Changing the Sequence of Instructions

- In the FETCH phase,
  we increment the Program Counter by 1.
- What if we don't want to always execute the instruction
  that follows this one?
  - examples: loop, if-then, function call
- Need special instructions that change the contents of the PC.
- These are called *control instructions*.
  - jumps are unconditional -- they always change the PC
  - branches are conditional -- they change the PC only if some condition is true (e.g., the result of an ADD is zero)

Example: LC-3 JMP Instruction

- Set the PC to the value contained in a register. This becomes the address of the next instruction to fetch.



"Load the contents of R3 into the PC."

Instruction Processing Summary

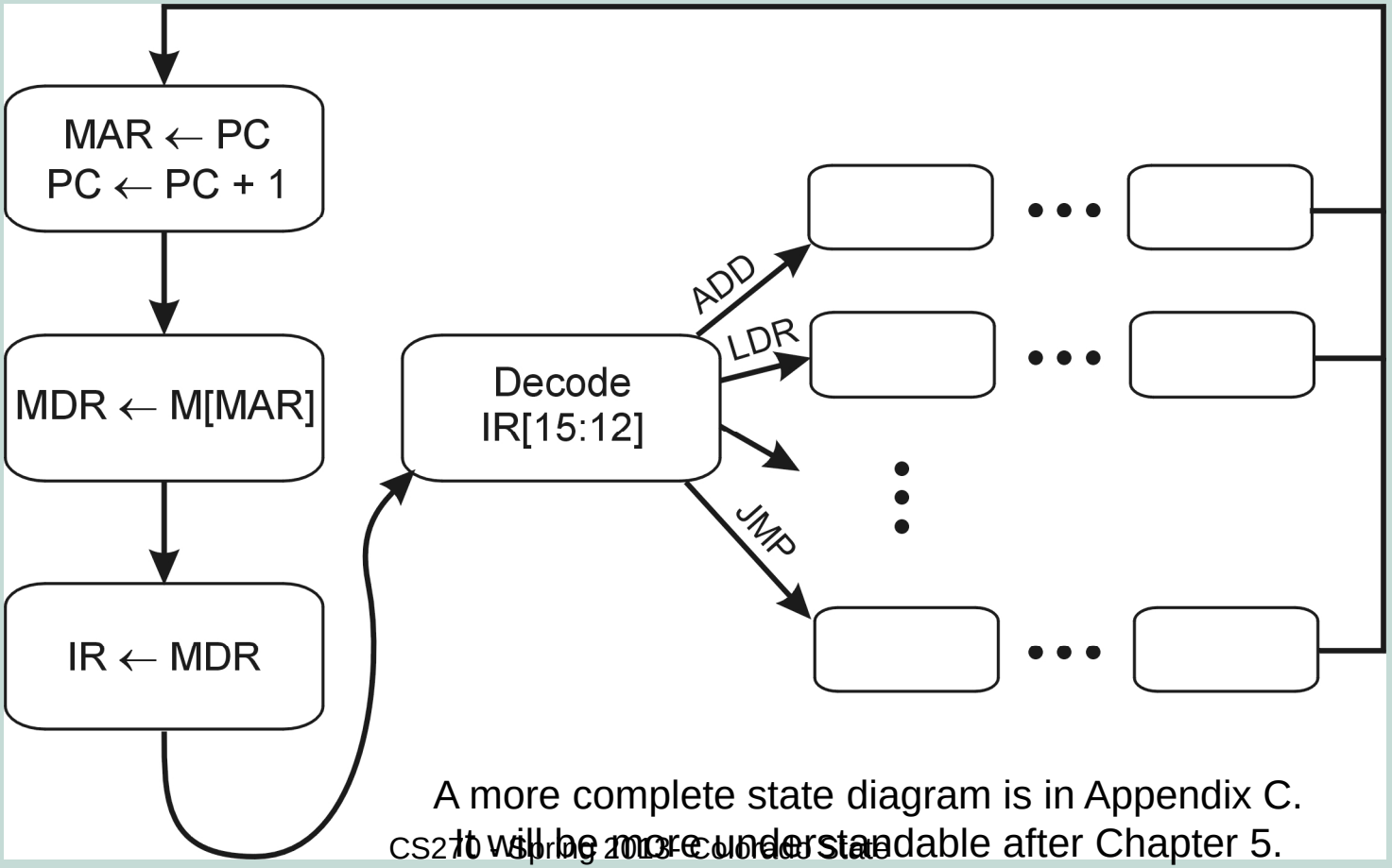
- Instructions look just like data -- it's all interpretation.
- Three basic kinds of instructions:
  - computational instructions (ADD, AND, …)
  - data movement instructions (LD, ST, …)
  - control instructions (JMP, BRnz, …)
- Six basic phases of instruction processing:

  $F \rightarrow D \rightarrow EA \rightarrow OP \rightarrow EX \rightarrow S$

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

Control Unit State Diagram

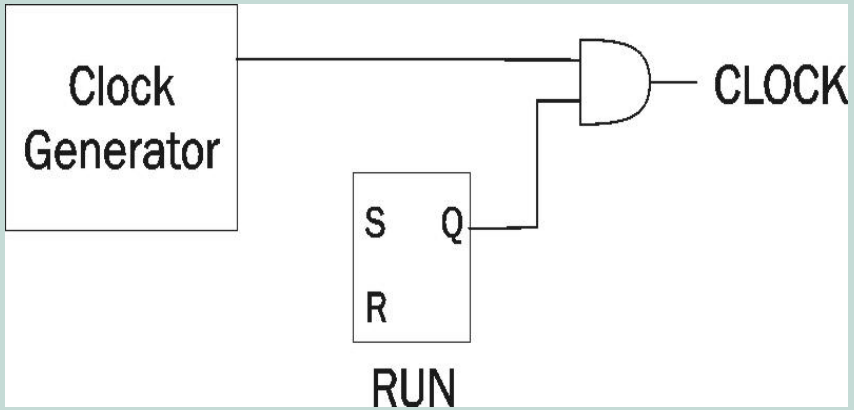
- The control unit is a state machine. Here is part of a simplified state diagram for the LC-3:



A more complete state diagram is in Appendix C.
It will be more understandable after Chapter 5.

Stopping the Clock

- Control unit will repeat instruction processing sequence as long as clock is running.
  - If not processing instructions from your application, then it is processing instructions from the Operating System (OS).
  - The OS is a special program that manages processor and other resources.



- To stop the computer:
  - AND the clock generator signal with ZERO
  - When control unit stops seeing the CLOCK signal, it stops processing.