ECE 120 Midterm 3 Review Session

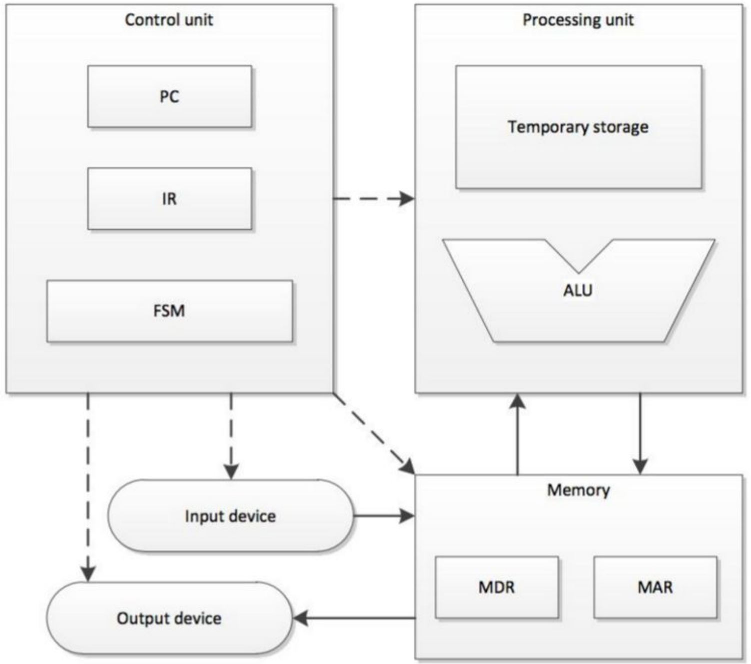
November 12th, 2016

ECE ILLINOIS

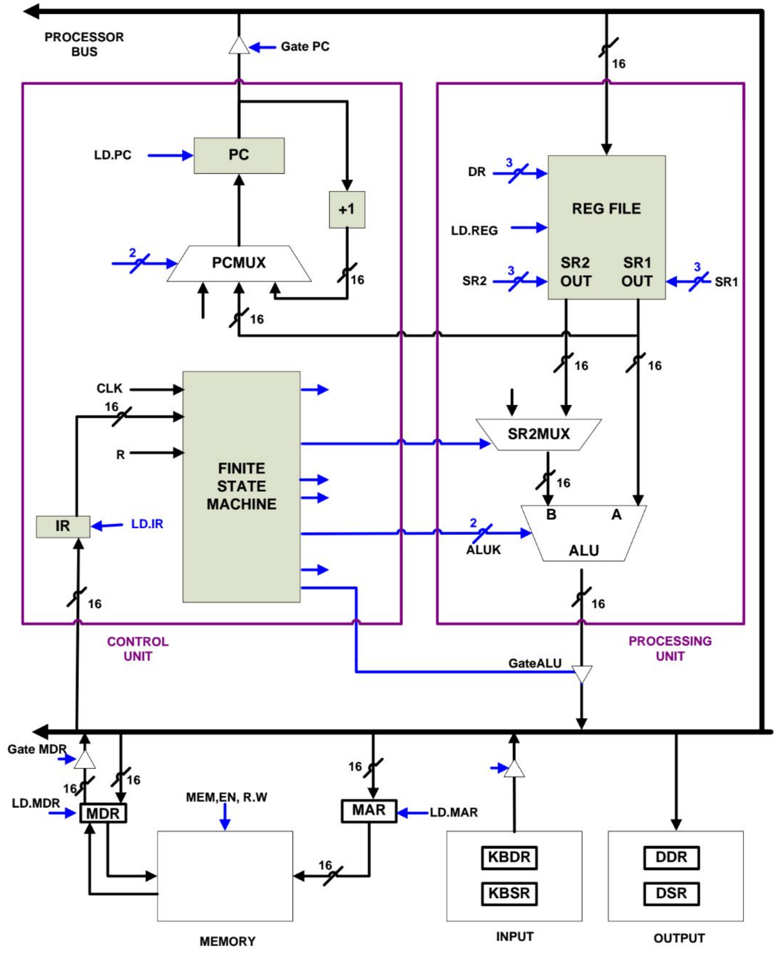
ILLINOIS

Von Neumann Model (Lecture 27)

- Components:
  - Memory
  - Processing Unit
  - Input
  - Output
  - Control Unit
  - (Page 101)
- Instruction Cycle:
  - Fetch
  - Decode
  - Evaluate Addr
  - Fetch Operands
  - Execute
  - Store Results
  - (Page 106)



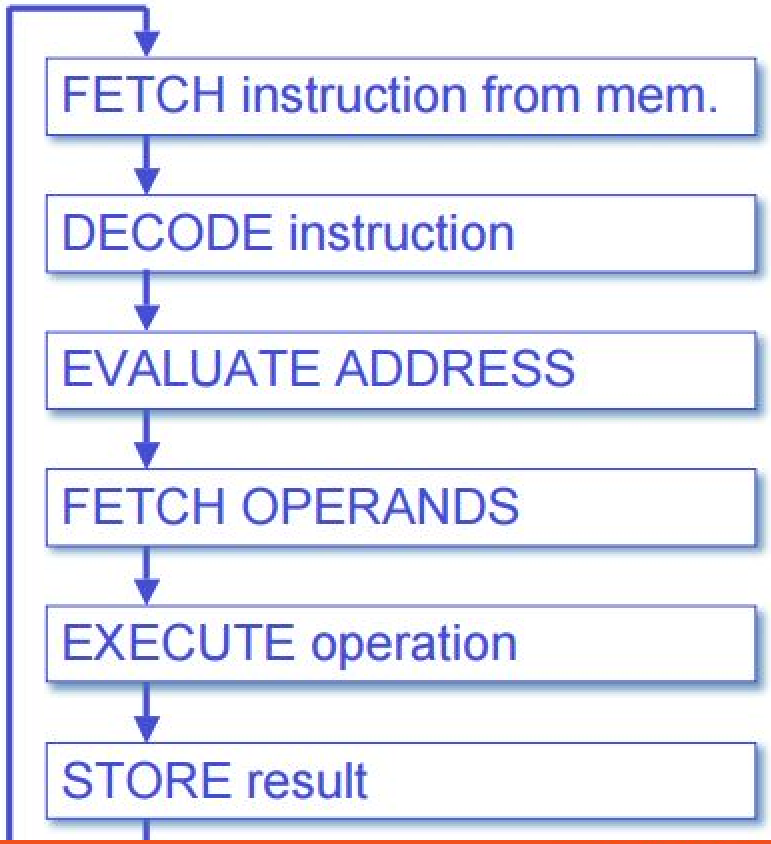
LC3 Datapath

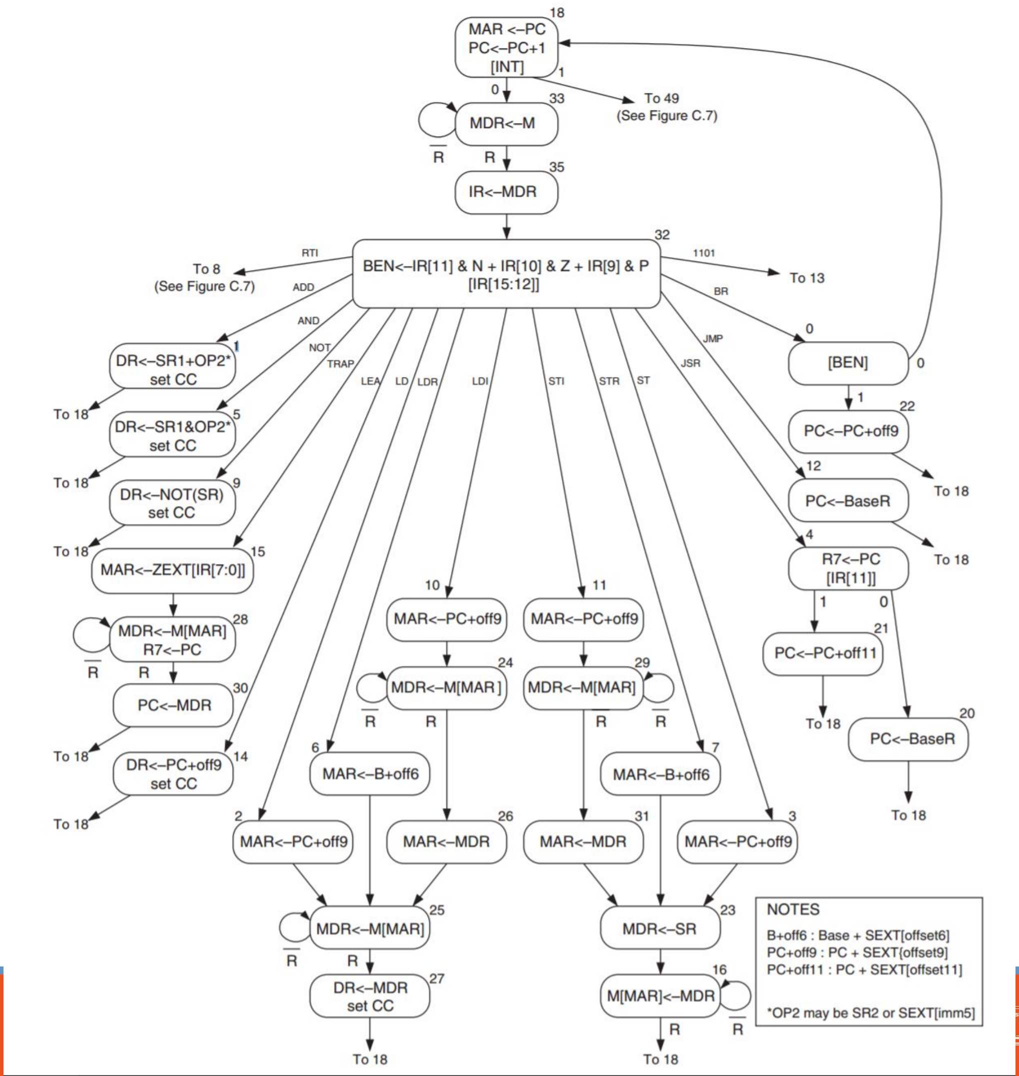


Instruction Processing

Question

- **How are instructions executed?**




18 MAR<–PC PC<–PC+1 [INT]
1 To 49 (See Figure C.7)
0
33 MDR<–M
$\overline{R}$ R
35 IR<–MDR
32 BEN<–IR[11] & N + IR[10] & Z + IR[9] & P [IR[15:12]]
RTI To 8 (See Figure C.7)
1101 To 13
ADD AND NOT TRAP LEA LD LDR LDI STI STR ST JSR JMP BR
1 DR<–SR1+OP2* set CC
5 DR<–SR1&OP2* set CC
9 DR<–NOT(SR) set CC
15 MAR<–ZEXT[IR[7:0]]
28 MDR<–M[MAR] R7<–PC
30 PC<–MDR
14 DR<–PC+off9 set CC
2 MAR<–PC+off9
6 MAR<–B+off6
10 MAR<–PC+off9
24 MDR<–M[MAR]
26 MAR<–MDR
25 MDR<–M[MAR]
27 DR<–MDR set CC
11 MAR<–PC+off9
29 MDR<–M[MAR]
31 MAR<–MDR
7 MAR<–B+off6
3 MAR<–PC+off9
23 MDR<–SR
16 M[MAR]<–MDR
0 [BEN] 0
1
22 PC<–PC+off9
12 PC<–BaseR
4 R7<–PC [IR[11]]
1 0
21 PC<–PC+off11
20 PC<–BaseR
To 18
NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
PC+off11 : PC + SEXT[offset11]
*OP2 may be SR2 or SEXT[imm5]

Instruction Processing: FETCH

Idea

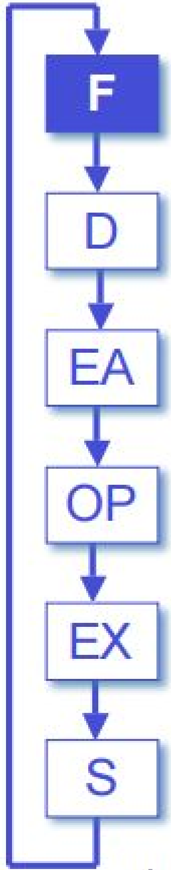
- Put next instruction in IR & increment PC

Steps

- Load contents of PC into MAR
- Increment PC
- Send "read" signal to memory
- Read contents of MDR, store in IR

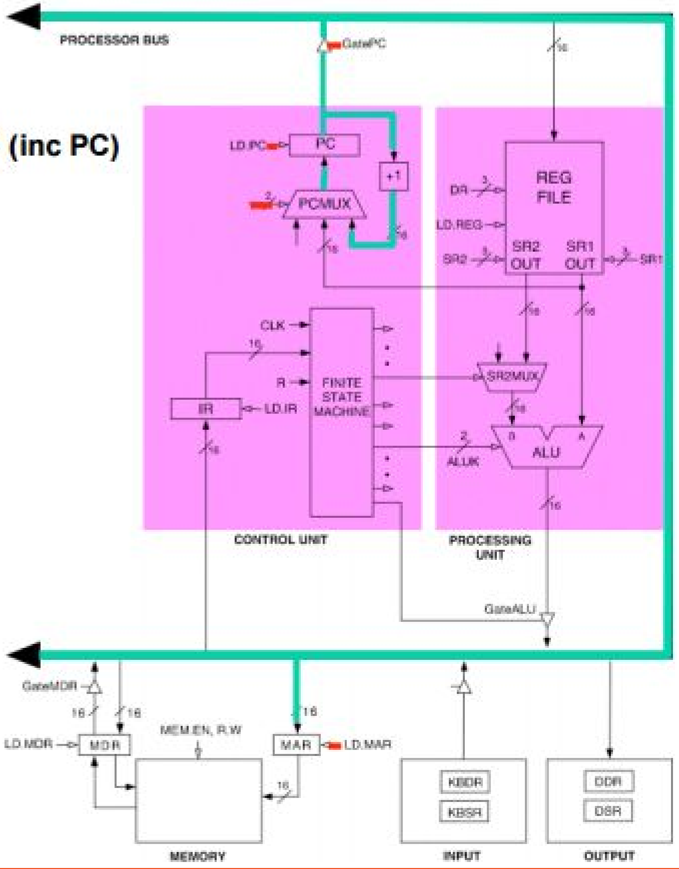
Who makes all this happen?

- Control unit



FETCH in LC-3

Load PC into MDR **(inc PC)**



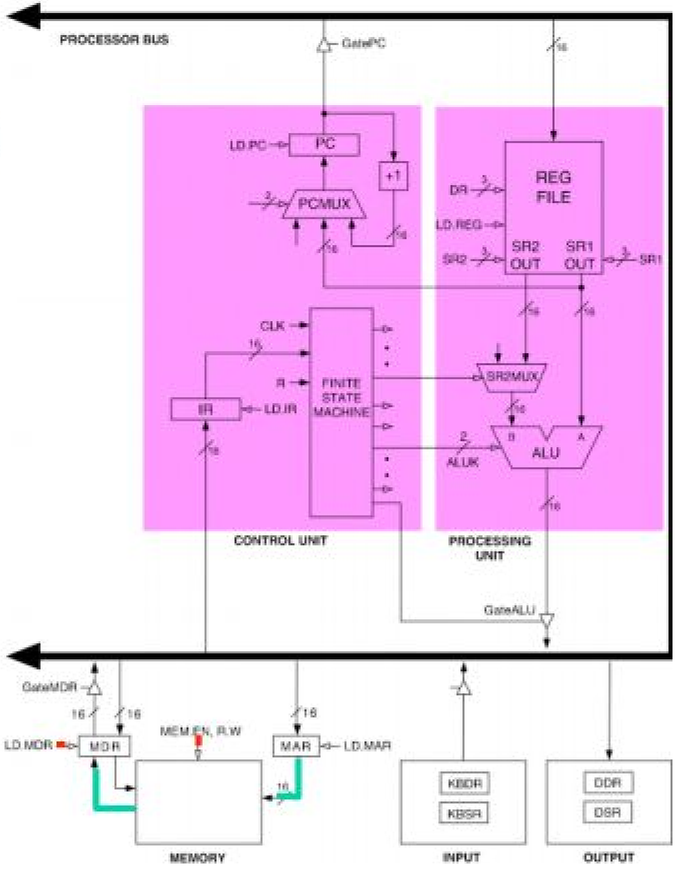
🡒 Control

🟩 Data

FETCH in LC-3

Load PC into MDR

Read Memory



→ Control

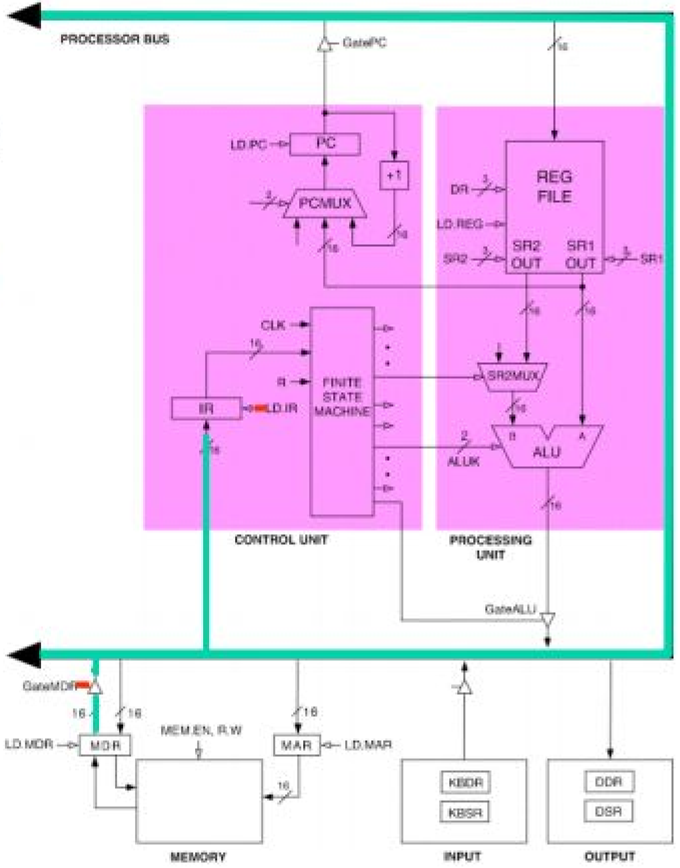
— Data

FETCH in LC-3

Load PC into MDR

Read Memory

Copy MDR into IR



→ Control

— Data

Instruction Processing: DECODE

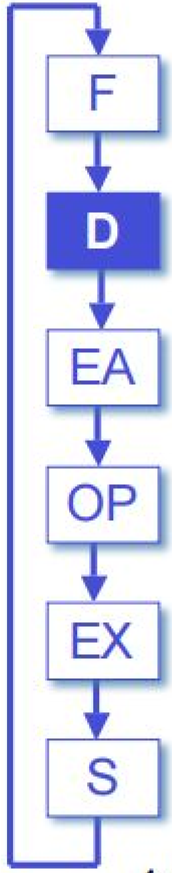
Identify opcode

- In LC-3, always first four bits of instruction
- 4-to-16 decoder asserts control line corresponding to desired opcode

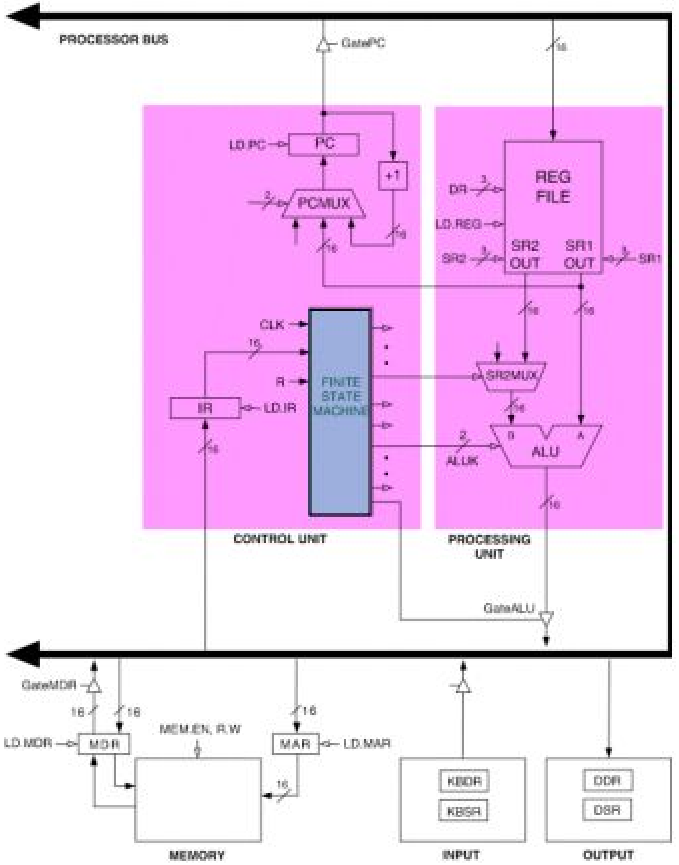
Identify operands from the remaining bits

- Depends on opcode
  e.g., for LDR, last six bits give offset
  e.g., for ADD, last three bits name source operand #2

Control unit implements DECODE



DECODE in LC-3



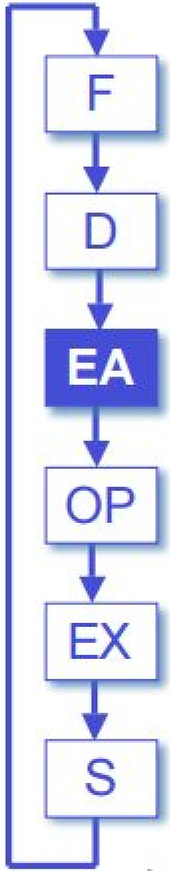
Instruction Processing: EVALUATE ADDRESS

Compute address

- For loads and stores
- For control-flow instructions (more later)

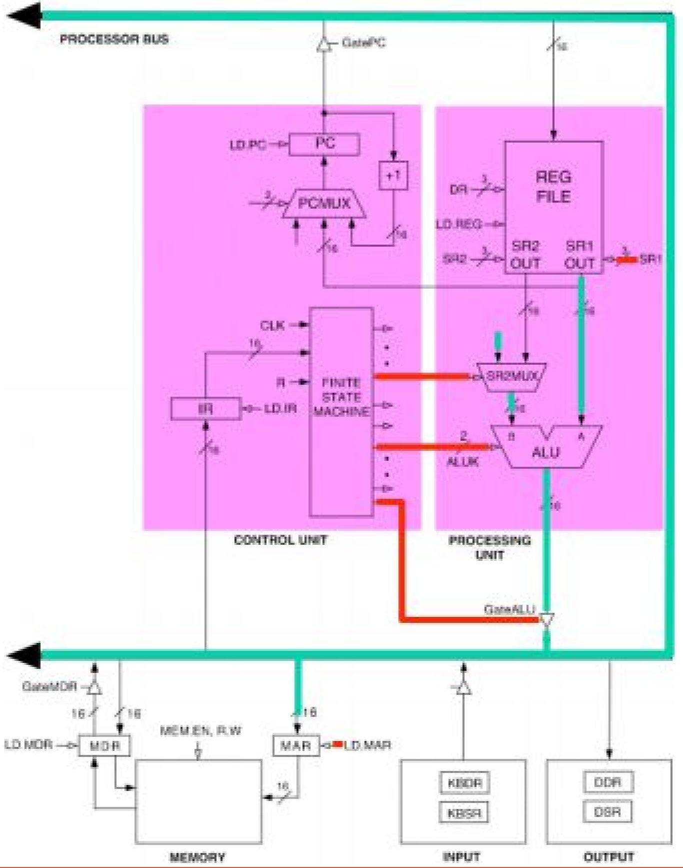
Examples

- Add offset to base register (as in LDR)
- Add offset to PC (as in LD and BR)



EVALUATE ADDRESS in LC-3

Load/Store

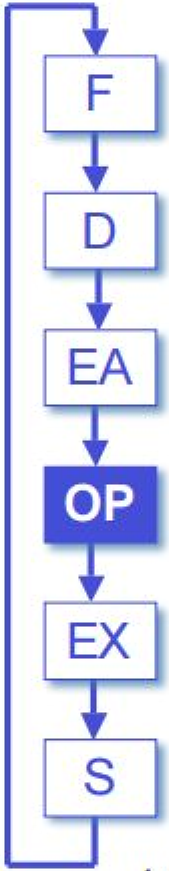


Instruction Processing: FETCH OPERANDS

Get source operands for operation

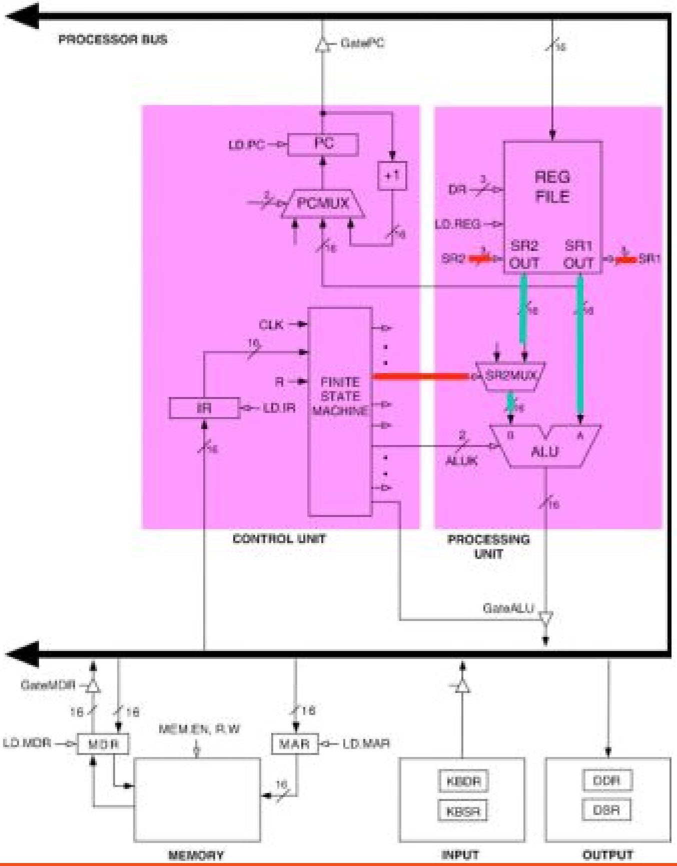
Examples

- **Read data from register file (ADD)**
- **Load data from memory (LDR)**



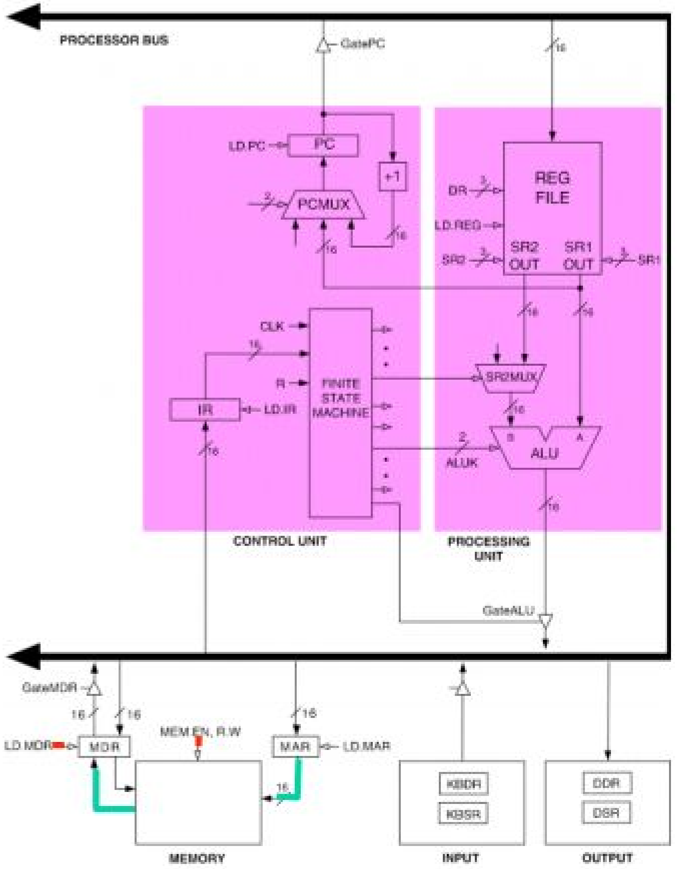
FETCH OPERANDS in LC-3

ADD



FETCH OPERANDS in LC-3

LDR

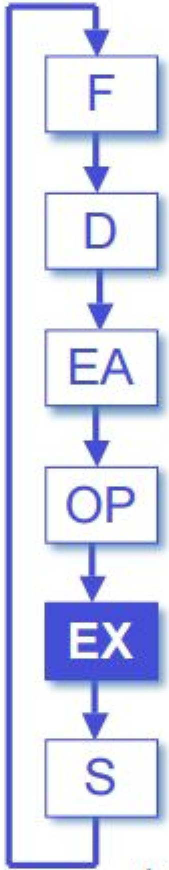


Instruction Processing: EXECUTE

Actually perform operation

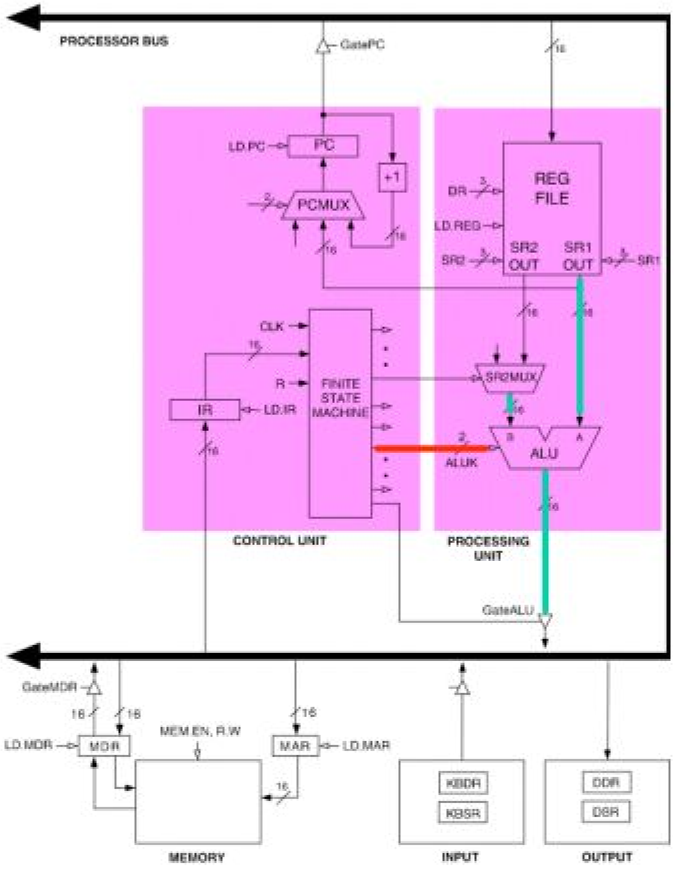
Examples

- **Send operands to ALU and assert ADD signal**
- **Do nothing (*e.g.*, for loads and stores)**



EXECUTE in LC-3

ADD



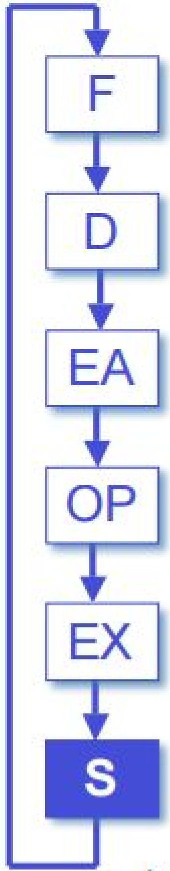
Instruction Processing: STORE

Write results to destination

- Register or memory

Examples

- Result of ADD is placed in destination reg.
- Result of load instruction placed in destination reg.
- For store instruction, place data in memory
  - ➢ Set MDR
  - ➢ Assert WRITE signal to memory



Memory

Memory is an important concept in computing. Memory is the ability to store a value.

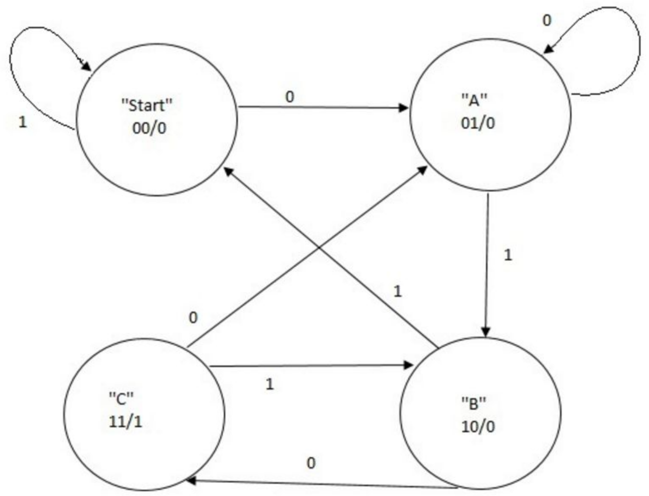
Memory has three basic parts to describe its structure:
Address: A unique identifier associated with each memory location.
Addressability: Number of bits in each location.
Address Space: The total number of uniquely identifiable memory locations.
Example:
2^n x m = 2^n address locations (address space), and m data bits (addressability)
You need n address bits [(n-1):0]

Memory (ctd.)

RAM
Addr: Address to save data to
Data: Information to be saved
CS: “Enable bit”, turns on RAM Chip
R/W: Specifies whether or not you can write (R =1, W = 0)
Dout: Output data
Notes: 1 MB = 2^20 and 1 KB = 2^10

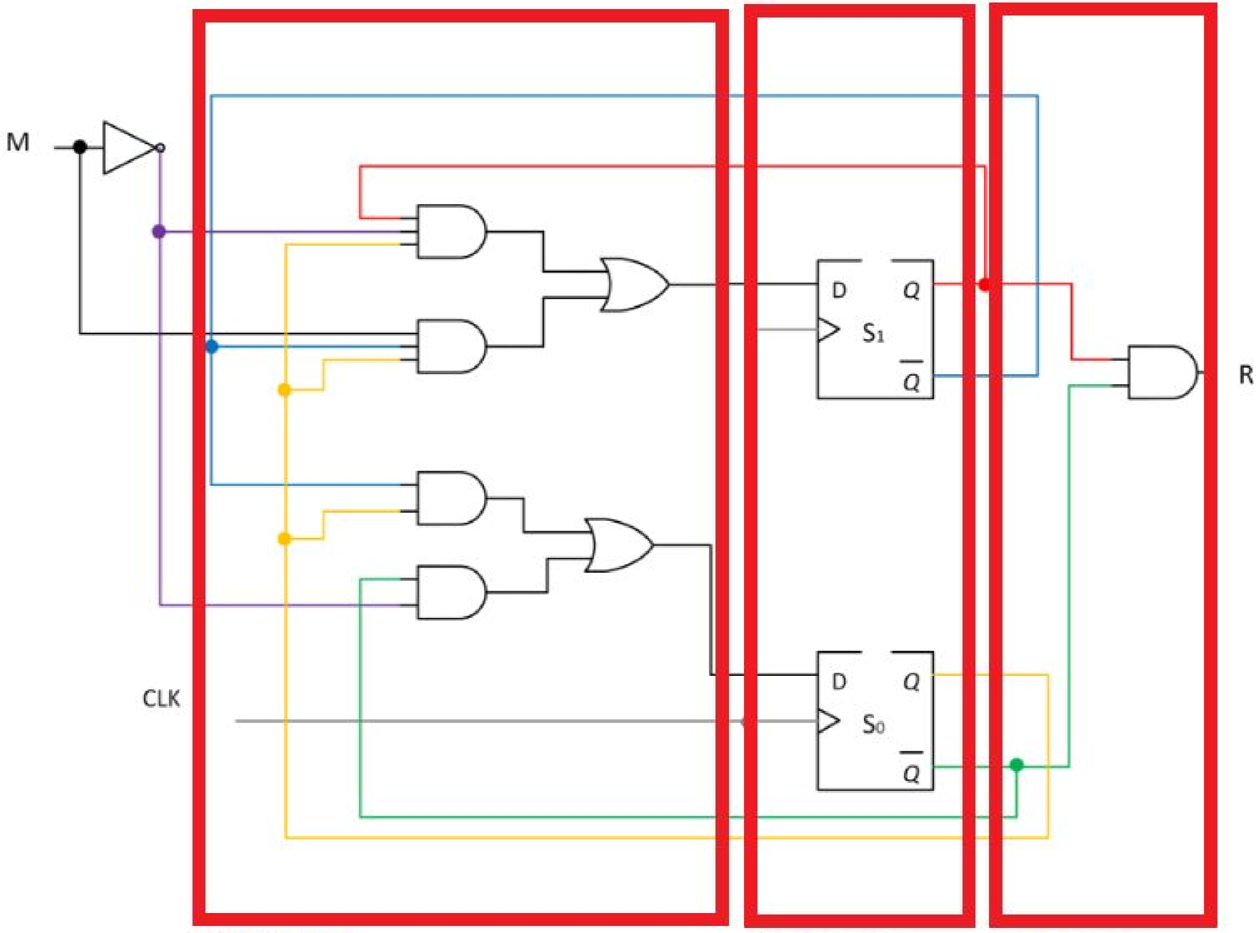
FSM Design

- Finite State Machines can be represented using a State Diagram:



FSM Design

- Finite state machines can be broken up into three parts:
  - Next State Logic
  - Current State
  - Output Logic



Next State Logic

Flip Flops store Current State

Output Logic

FSM Design

Current State:

- N states need $\lceil \log_2(N) \rceil$ bits to represent each state
- Current state is stored using memory (usually a series of flip flops)

FSM Design

Next State Logic:

- Create a Next State Table
  - Each given State should have a Next State based on both Inputs and the Current State
- Make a K-map for each Next State bit to create a minimal Boolean expression.
- Draw the gate logic for each Next State

FSM Design

Output Logic

- Output logic is only dependent on the current states in a Moore FSM
- Draw a K-map for each output bit based using the current state bits
- Create a minimal Boolean expression and then draw the gate logic