Chapter 4
The Von Neumann Model

Warning!

This is a bottom-up course

- **No secrets, no magic**
  _e.g.,_ gates build on transistors, logic circuits from gates, _etc_.

But… This is a top-down lecture

- **You'll have to trust me for a couple slides**
- **Start with _very_ abstract discussion of computer architecture**
- **Meet with Chapter 3 material soon**

What Do We Know?

A LOT!!

- **Data representation** (binary, 2's complement, floating point, …)
- **Transistors** (p-type, n-type, CMOS)
- **Gates** (complementary logic)
- **Combinational logic circuits** (PLAs), **memory** (latches, flip-flops, …)
- **Sequential logic circuits** (state machines)
- **Simple "processors"** (programmable traffic sign)

What's next?

- **Apply all this to traditional computing**
- **Software interface: instructions**
- **Hardware implementation: data path**

A Little Context

1943: ENIAC

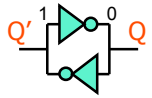
- **First general electronic computer (Presper Eckert and John Mauchly)**
  (Or was it Atananasoff in 1939? Or Konrad Zuse in 1941?)
- **18,000 tubes (had to replace 50 a day!)**
- **Memory: 20 10-digit numbers (decimal)**
- **Hard-wired program (via dials, switches, and cables)**
- **Completed in 1946**

See *Eniac* by Scott McCartney

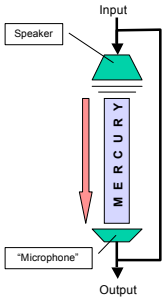
1944: Beginnings of EDVAC

- **Among other improvements, includes program stored in memory**
- **Gave birth to UNIVAC-I (1951)**
- **Completed in 1952**

Aside: Early Memories



Mercury delay lines!



Context Continued: Stored Program Computer

1945: John von Neumann

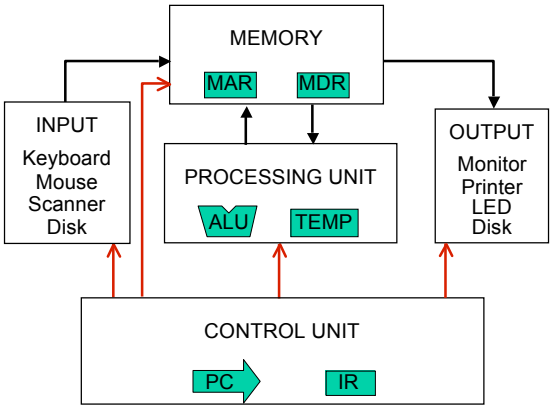
- ***First Draft of a Report on EDVAC***

See *John von Neumann and the Origins of Modern Computing* by William Aspray

Von Neumann Machine (or Model)

- ***Memory***, **containing instructions and data**
- ***Control unit***, **for interpreting instructions**
- ***Processing unit***, **for performing arithmetic and logical operations**
- ***Input/Output units***, **for interacting with *real* world**

Von Neumann Model



Memory

***k* x *m* array of stored bits (*k* is usually 2^n)**

Address

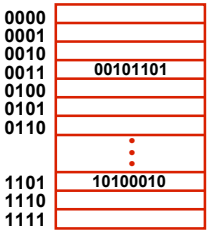
- **Unique (*n*-bit) identifier of location**

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**

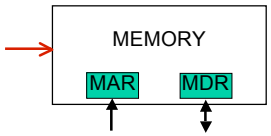


Interface to Memory

How does processing unit get data to/from memory?

MAR: Memory Address Register

MDR: Memory Data Register



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

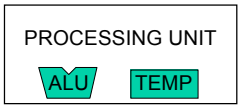
To write a value X to a location A

1. Write the data X to the MDR
2. Write the address A into the MAR
3. Send a "write" signal to the memory

Processing Unit

Functional Units

- ALU = Arithmetic and Logic Unit
- Could have many functional units (some special-purpose, *e.g.,* multiply, square root, …)
- LC-3: ADD, AND, NOT



Registers

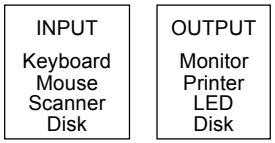
- Small, temporary storage
- Operands and results of functional units
- LC-3: eight register (R0, …, R7)

Word Size

- Number of bits normally processed by ALU in one instruction
- Also width of registers
- LC-3: 16 bits

Input and Output

Devices get data into and out of computer



Each device has own interface

- Often a set of registers like the memory's MAR and MDR
- LC-3 supports keyboard (input) and display (output)
- Keyboard: data register (KBDR) and status register (KBSR)
- Text display: data register (DDR) and status register (DSR)
- Graphical display: later…

Some devices provide both input and output

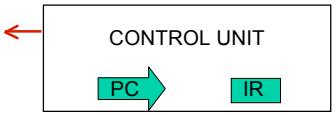
- Disk, network

Software that controls device access

- *Driver*

Control Unit

Orchestrates execution of the program



Instruction Register (IR)

- Contains the ***current instruction***

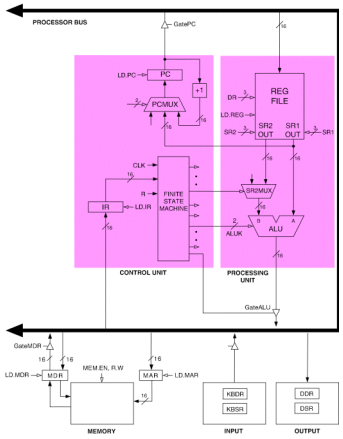
Program Counter (PC)

- Contains the ***address*** of the next instruction to execute

Control Unit

- Reads an instruction from memory (at PC)
- Interprets the instruction
- Generates signals that tell the other components what to do
- Instruction may take many *machine cycles* to complete

LC-3

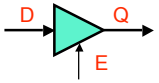


One More Gate

Tri-state buffer

- NOT an inverter!

E	D	Q
1	0	0
1	1	1
0	0	Z
0	1	Z



Z = "high impedance" state
(no current, *i.e.*, no "pressure")

Allows wires to be "shared"

- Alternative to mux
- Only one source may drive at a time!

Instructions

Fundamental unit of work

Constituents

- *Opcode*: operation to be performed
- *Operands*: data/locations to be used for operation

Encoded as a sequence of bits *(just like data!)*

- Sometimes have a fixed length (*e.g.*, 16 or 32 bits)
- Control unit interprets instruction
  - ➢ Generates control signals to carry out operation
- Atomic: operation is either executed completely, or not at all

Instruction Set Architecture (ISA)

- Computer's instructions, their formats, their behaviors

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD				Dst			Src1			0	0	0	Src2		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0

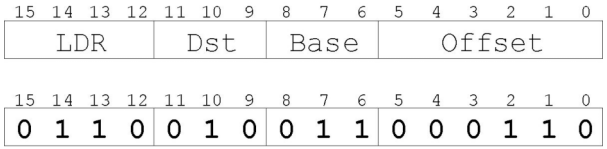
"Add the contents of R2 to the contents of R6, and store the result in R6."

Example: LC-3 LDR Instruction

Reads data from memory

Base + offset addressing mode

- **Add offset to base register to produce 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 memory at that address and place the resulting data in R2."

Instruction Processing

Question

- **How are instructions executed?**

Instruction Processing: FETCH

Idea

- **Put next instruction in IR & increment PC**

Steps

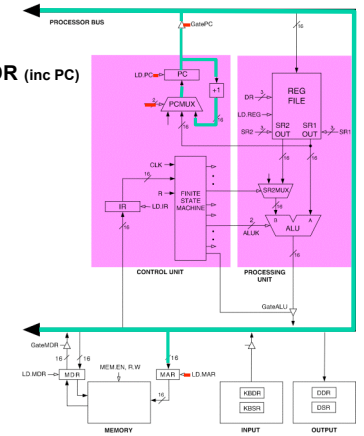
- **Load contents of PC into MAR**
- **Increment PC**
- **Send "read" signal to memory**
- **Read contents of MDR, store in IR**

Who makes all this happen?

- **Control unit**

FETCH in LC-3

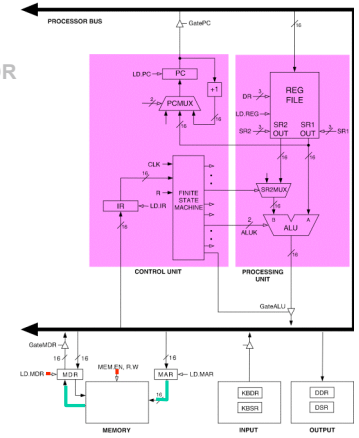
Load PC into MDR (inc PC)



FETCH in LC-3

Load PC into MDR

Read Memory



Control

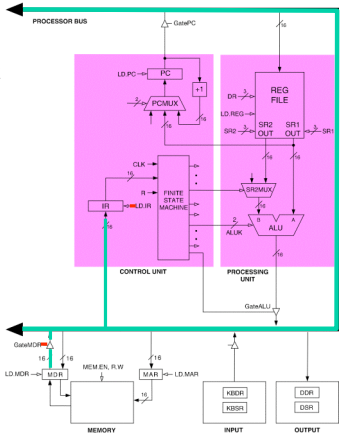
Data

FETCH in LC-3

Load PC into MDR

Read Memory

Copy MDR into IR



Control

Data

Instruction Processing: DECODE

Identify opcode

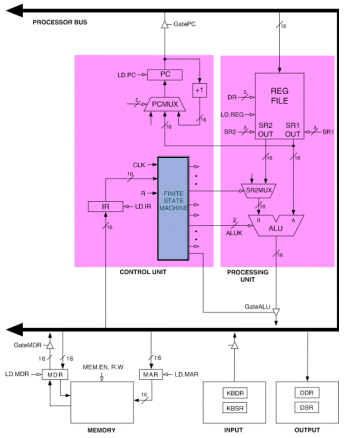
- **In LC-3, always first four bits of instruction**
- **4-to-16 decoder asserts control line corresponding to desired opcode**

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

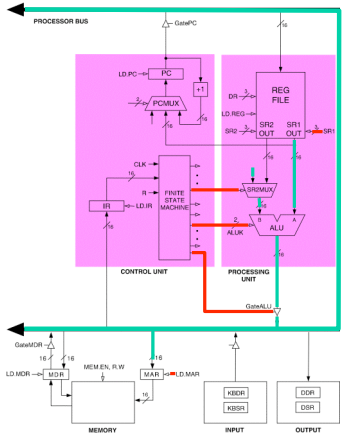
- For loads and stores
- For control-flow instructions (more later)

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

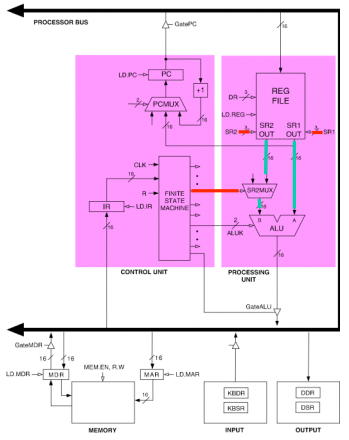
Get source operands for operation

Examples

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

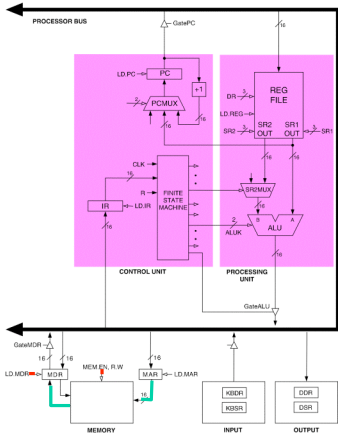
FETCH OPERANDS in LC-3

ADD



FETCH OPERANDS in LC-3

LDR



Instruction Processing: EXECUTE

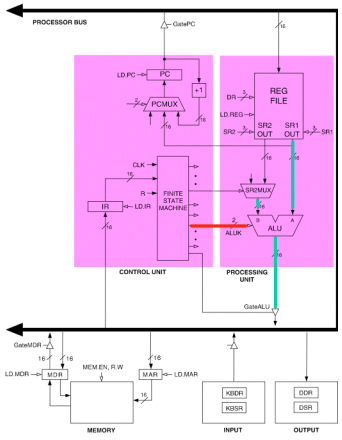
Actually perform operation

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

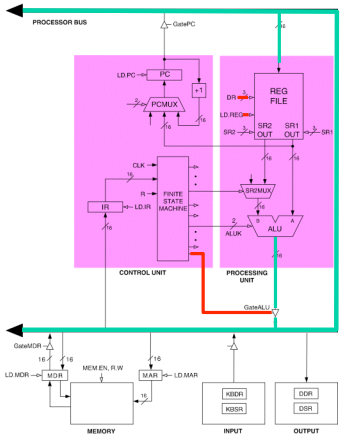
- **Register or memory**

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**

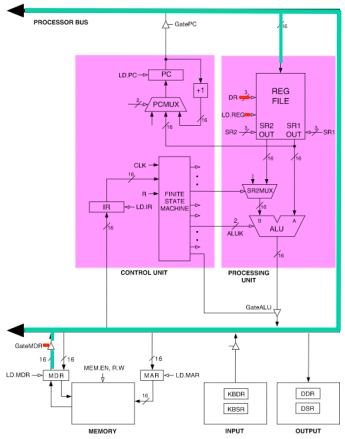
STORE in LC-3

ADD



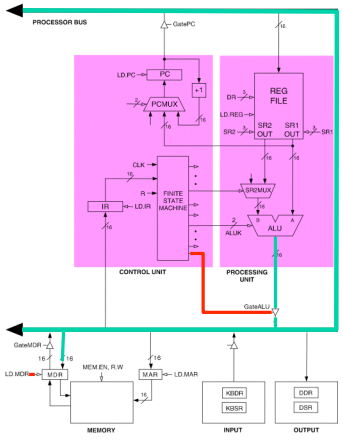
STORE in LC-3

LDR



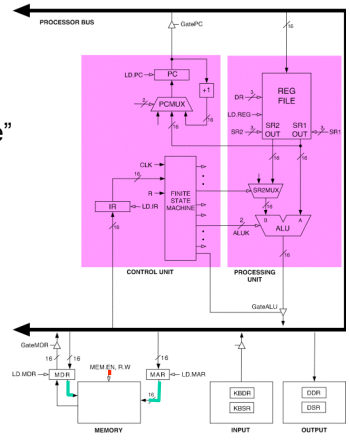
STORE in LC-3

STORE
Set MDR



STORE in LC-3

STORE
Set MDR
Assert “write”



Changing the Sequence of Instructions

Recall FETCH

- Increment PC by 1

What if we don't want linear execution?

- *E.g.,* loop, if-then, function call

Need instructions that change PC

- Jumps are unconditional
  - ➢ Always change the PC
- Branches are conditional
  - ➢ Change the PC only if some condition is true
    e.g., the contents of a register is zero

Example: LC-3 JMP Instruction

Set the PC to the value of a register

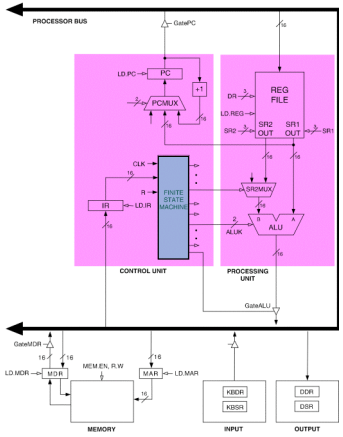
- Fetch next instruction from this address

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP				0	0	0	Base			0	0	0	0	0	0

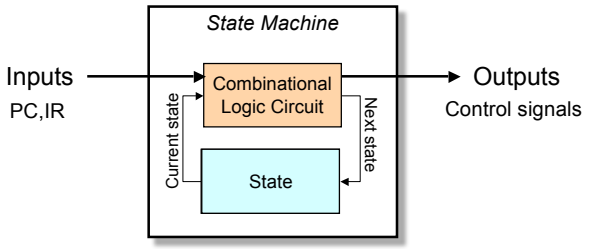
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0

"Load the contents of register R3 into the PC."

How Does Control Unit Work?



Remember Finite State Machines?



Control Unit Details

Finite state machine

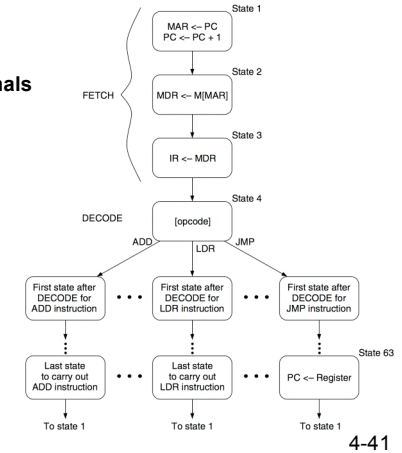
- Input: PC, IR
- Output: *many* control signals

Need to map abstract ops to control signals

- *E.g.,* MAR <- PC
  ⇒ GatePC and LD.MAR
- *E.g.,* PC <- PC + 1
  ⇒ PCMUX=2 and LD.PC

LC-3

- 35 states (Fig. C.2)



Instruction Processing Summary

Instructions look just like data

- Interpreted by machine (or software)

Three basic kinds of instructions

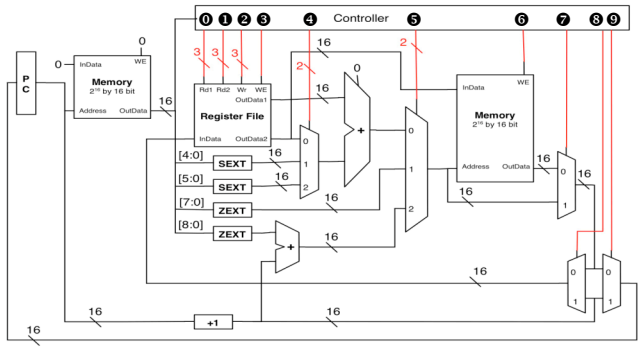
- Computational instructions (ADD, AND, …)
- Data movement instructions (LD, ST, …)
- Control instructions (JMP, BRnz, …)

Six basic phases of instruction processing

F → D → EA → OP → EX → S

- Not all phases are needed by every instruction
- Multiple phases per cycle possible
- Phases may take variable number of machine cycles

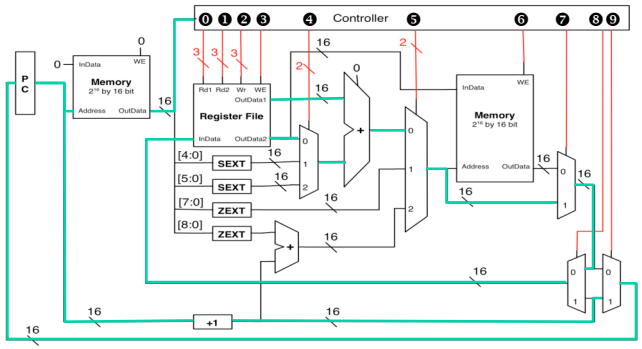
Alternate Implementation



Execute Each Instruction in Single Cycle

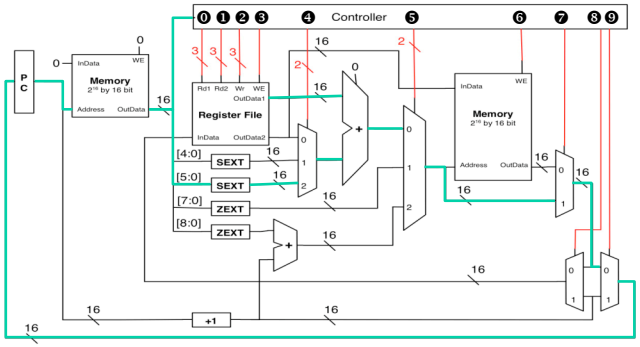
- Much simpler
- All phases happen in one cycle

Single-Cycle Implementation (ADD)



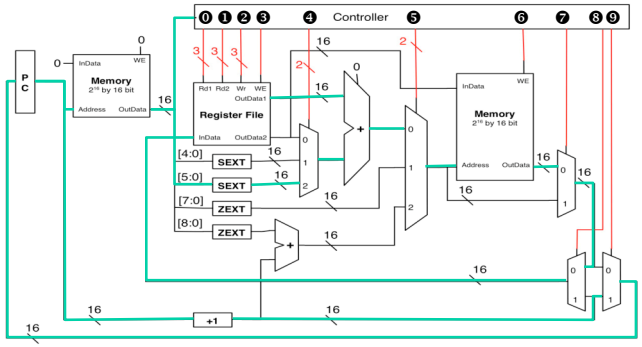
	Opcode		Control								
Instr	I[15:12]	I[5]	❶	❷	❸	❹	❺	❻	❼	❽	❾
ADD	0001	0	I[8:6]	I[2:0]	I[11:9]	1	00	00	0	1	0	1

Single-Cycle Implementation (JMP)



	Opcode		Control									
Instr	I[15:12]	I[5]	❶	❶	❷	❸	❹	❺	❻	❼	❽	❾
JMP	1100	-	I[8:6]	-	-	0	10	00	0	1	x	0

Single-Cycle Implementation (LDR)



	Opcode		Control									
Instr	I[15:12]	I[5]	❶	❶	❷	❸	❹	❺	❻	❼	❽	❾
LDR	0110	-	I[8:6]	-	I[11:9]	1	10	00	0	0	0	1

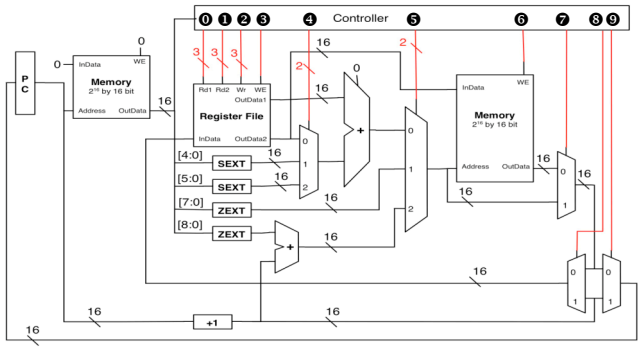
Single-Cycle Implementation Limitations

All instructions given same time to execute

- **Cycle time must be time required for longest instruction**
- **JMP versus ADD versus LDR**

Requires multi-ported memory

Single-Cycle Implementation Limitations (cont.)



AND? NOT? What changes would you make?
(LDI/STI? JSR? RTI? More on these later)

Next Time

Lecture

- LC-3

Reading

- Chapter 5 - 5.2

Quiz

- Online!

Upcoming

- Homework due Monday 11 October