Processor Data Path and Control

CIT 595
Spring 2007

What Do We Know?

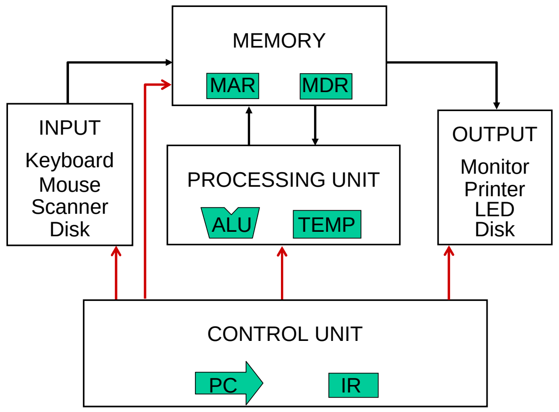
Already discovered:

- Gates (AND, OR..)
- Combinational logic circuits (decoders, mux)
- Memory (latches, flip-flops)
- Sequential logic circuits (state machines)
- Simple processors (programmable traffic sign)

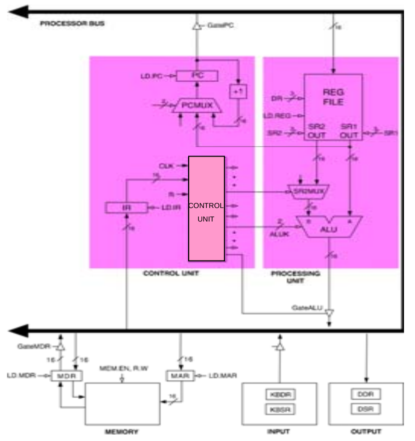
What's next?

- Apply all this to build a working processor

Von Neumann Model



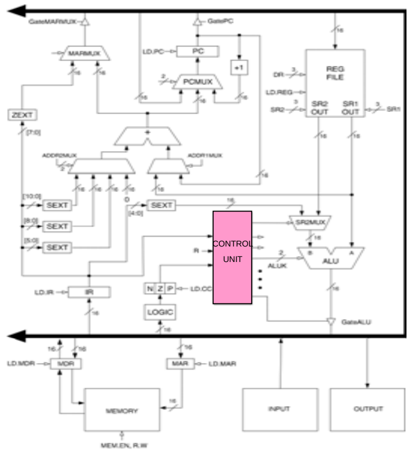
LC-3 Processor Von Nuemann Model



LC-3 Data Path

The data path of a computer is all the logic used to process information

Filled arrow
= info to be processed.
Unfilled arrow
= control signal.



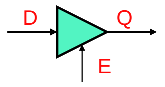
CIT 595

7 - 5

One More Device

Tri-state buffer
- NOT an inverter!
- Device with a special output that can take a *third* state (i.e. besides 0 and 1)

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



Z = “high impedance” state

Allows wires to be “shared”
- Alternative to mux
- Only one source may drive at a time!
- Usually used to control data over a bus

CIT 595

7 - 6

Data Path Components

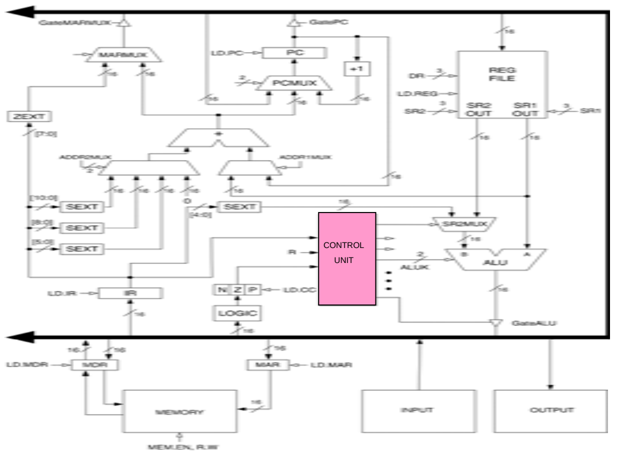
Global bus
- **Set of wires that carry 16-bit signals to many components**
- **Inputs to bus are controlled by triangle structure called tri-state devices**
  - ➢ **Place signal on bus when enabled**
  - ➢ **Only one (16-bit) signal should be enabled at a time**
  - ➢ **Control unit decides which signal “drives” the bus**
- **Any number of components can read bus**
  - ➢ **Register only captures bus data if write-enabled by the control unit**

Memory and I/O
- **Control signals and data registers for memory and I/O devices**
- **Memory: MAR, MDR (also control signal for read/write)**
- **Input (keyboard): KBSR, KBDR**
- **Output (text display): DSR, DDR**

CIT 595

7 - 7

LC-3 Data Path



CIT 595

Filled arrow = info to be processed. **Unfilled arrow** = control signal.

7 - 8

Data Path Components (cont.)

ALU

- Input: register file or sign-extended bits from IR (immediate field)
- Output: bus; used by…
  - ➢ Condition code registers
  - ➢ Register file
  - ➢ Memory and I/O registers

Register File

- Two read addresses, one write address (3 bits each)
- Input: 16 bits from bus
  - ➢ Result of ALU operation or memory (or I/O) read
- Outputs: two 16-bit
  - ➢ Used by ALU, PC, memory address
  - ➢ Data for store instructions passes through ALU

Data Path Components (contd..)

PC and PCMUX

- Three inputs to PC, controlled by PCMUX
  1. Current PC plus 1 (normal operation)
  2. Adder output (BR, JMP, …)
  3. Bus (TRAP)

MAR and MARMUX

- Some inputs to MAR, controlled by MARMUX
  1. Zero-extended IR[7:0] (used for TRAP; more later)
  2. Adder output (LD, ST, …)

Data Path Components (cont..)

Condition Code Logic

- Looks at value (from ALU) on bus and generates N, Z, P signals
- N,Z,P Registers are set only when control unit enables them

Control Unit

- For each stage in instruction processing decides:
  - ➢ Who drives the bus?
  - ➢ Which registers are write enabled?
  - ➢ Which operation should ALU perform?

Lets Look at Instruction Processing next..

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)
- Atomic: operation is either executed completely, or not at all

Instruction Processing

- FETCH instruction from mem.
- DECODE instruction
- EVALUATE ADDRESS
- FETCH OPERANDS
- EXECUTE operation
- STORE result

Instruction Processing: FETCH

Idea

- Put next instruction in IR & increment PC

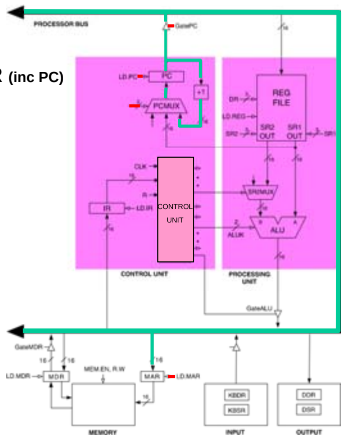
Steps

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

F
D
EA
OP
EX
S

FETCH in LC-3

Load PC into MAR (inc PC)

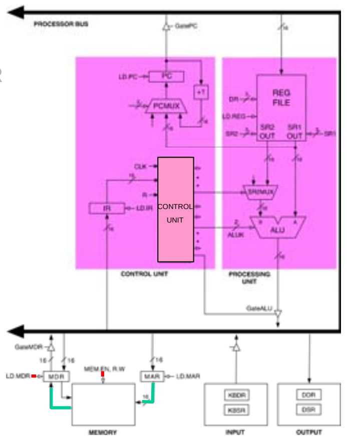


→ Control
— Data

FETCH in LC-3

Load PC into MAR

Read Memory



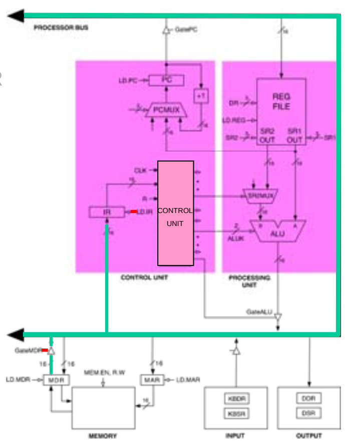
→ Control
— Data

FETCH in LC-3

Load PC into MAR

Read Memory

Copy MDR into IR



→ Control

— Data

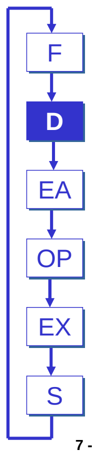
Instruction Processing: DECODE

Identify opcode

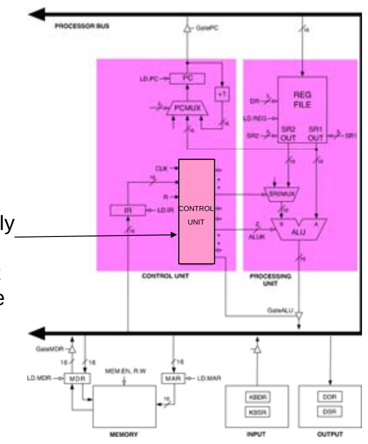
- In LC-3, always first four bits of instruction
- 4-to-16 decoder asserts control line corresponding to desired opcode

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



DECODE in LC-3



Decoding usually a part of the Control Unit but can be seperate

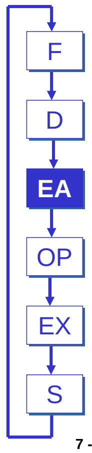
Instruction Processing: EVALUATE ADDRESS

Compute address

- For loads and stores
- For control-flow instructions

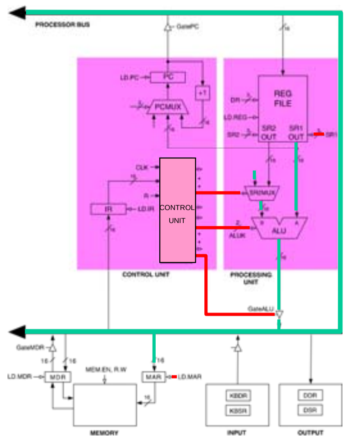
Examples

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



EVALUATE ADDRESS in LC-3

Load/Store



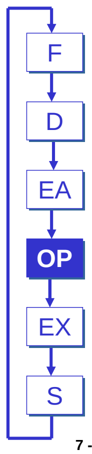
CIT 595

7 - 21

Instruction Processing: FETCH OPERANDS

Get source operands for operation

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

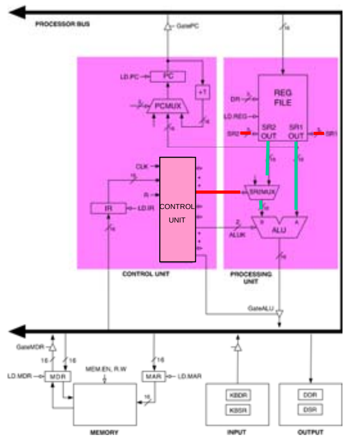


CIT 595

7 - 22

FETCH OPERANDS in LC-3

ADD

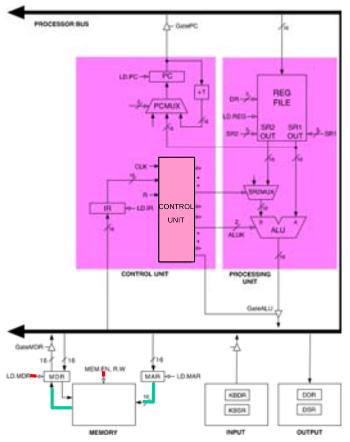


CIT 595

7 - 23

FETCH OPERANDS in LC-3

LDR



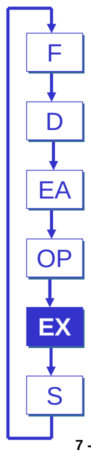
CIT 595

7 - 24

Instruction Processing: EXECUTE

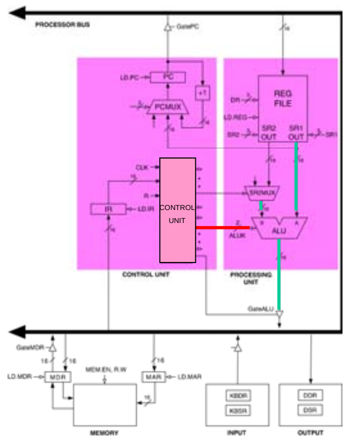
Actually perform operation

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



EXECUTE in LC-3

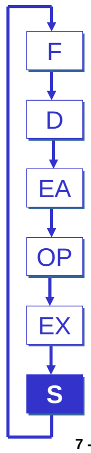
ADD



Instruction Processing: STORE

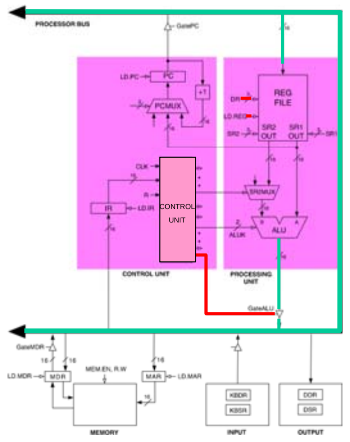
Write results to destination
- Register or memory

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



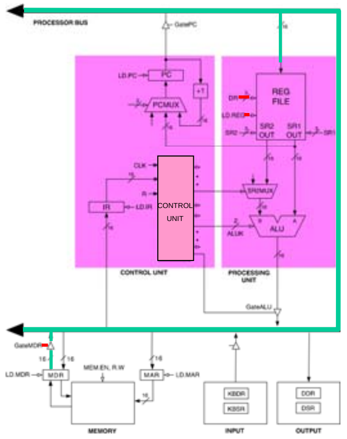
STORE in LC-3

ADD



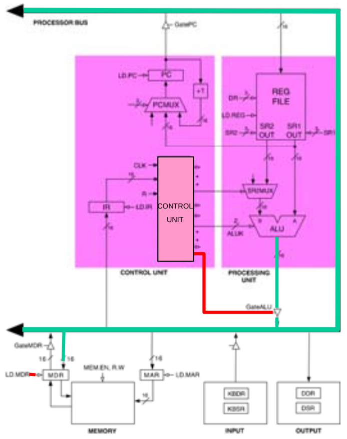
STORE in LC-3

LDR



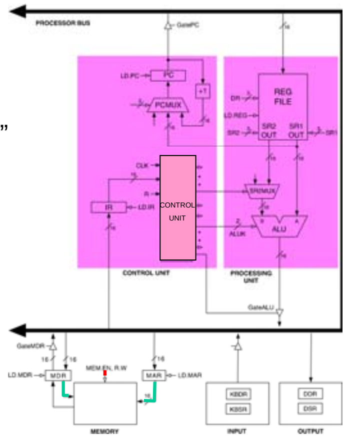
STORE in LC-3

STORE
Set MDR



STORE in LC-3

STORE
Set MDR
Assert “write”



Time to Complete One Instruction

- It takes fixed number of clock ticks (repetition of rising or falling edge) to execute each instruction
  - ➢ The time interval between ticks is known as ***clock cycle***
  - ➢ Thus instruction performance is measured in clock cycles
- Hence the clock sequences each phase of an instruction by raising the right signals as the right time
- So what determines the time between ticks i.e. the length of the clock cycle?

Clocking Methodology

• Defines when **signals** can be read and when they can be written

• It is important to specify the timing of reads and writes because, if a value is written at the same time it is read, the value of read could be old, new or mix of both

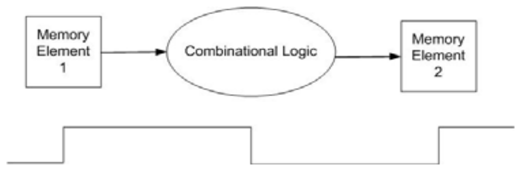
• All values are stored on clock edge (edge-triggered) i.e. within a defined interval of time (length of the clock cycle)

• In a processor, since only memory elements can store values this means that

- ➢ Any collection of combinational logic must have its inputs coming from a set of memory elements and its outputs written into a set of memory elements

Clocking Methodology (contd..)

• The length of the clock cycle is determined as follows:



• The time necessary for the signals to reach memory element 2 defines the length of the clock cycle

- ➢ i.e. minimum clock cycle time must be at least as great as the maximum propagation delay of the circuit

How does the control unit work ?

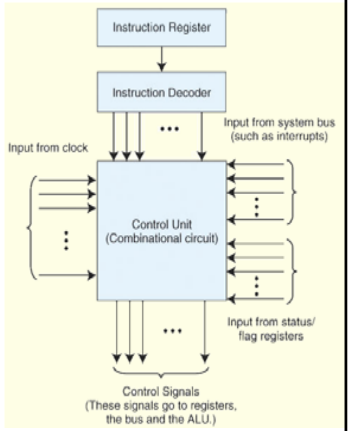
Two approaches:

- Hardwired Control
- Microprogrammed Control

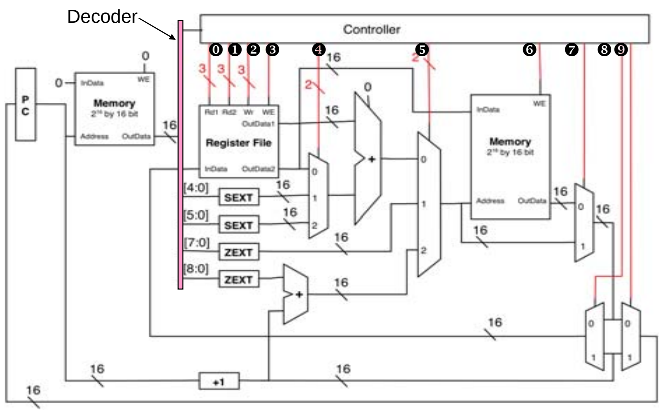
Approach I: Hardwired Control

Hardwired Control

- ➢ Directly connects the control lines to actual machine instructions
- ➢ The instructions are divided into fields, and bits in the fields are connected to input lines that drives the various digital logic components
- ➢ The control signals are some combination of the instruction bit plus other signals such as interrupts, or condition codes from previous instruction

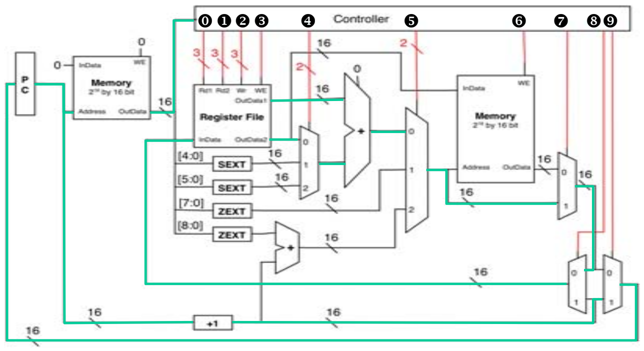

Instruction Register
Instruction Decoder
Input from clock
Input from system bus (such as interrupts)
Control Unit (Combinational circuit)
Input from status/ flag registers
Control Signals (These signals go to registers, the bus and the ALU.)

LC3 as Hardwired Control Implementation



The Control Signals (red colored lines) are outputs by some combination of inputs from the instruction bit fields

ADD Instruction



Instr	Opcode I[15:12]	Opcode I[5]	Control ⓿	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

Sequencing the Stages in Hardwired Implementation

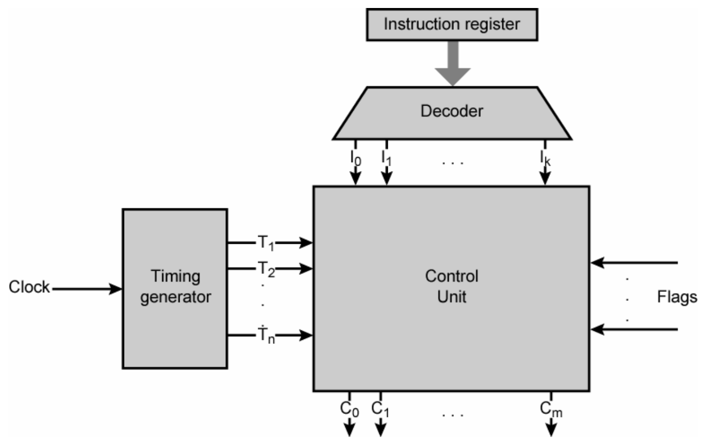
- The combination Control Unit set all the control lines needed by an instruction
- How do we ensure that we sequence through Instruction cycle i.e. F->D->EA->OP->EX->S?
  - We connect the clock to a synchronous counter and the counter to the decoder
  - The decoder output enabled is based on counter outputs (i.e. which cycle you are in)
  - The decoder output is then used as enable signal (gating) to enable only certain control signals during a particular cycle
  - Note: the diagram does not show sequencing logic to avoid cluttering the diagram

Sequencing Instruction Stages in Hardwired Implementation (contd..)

Example:

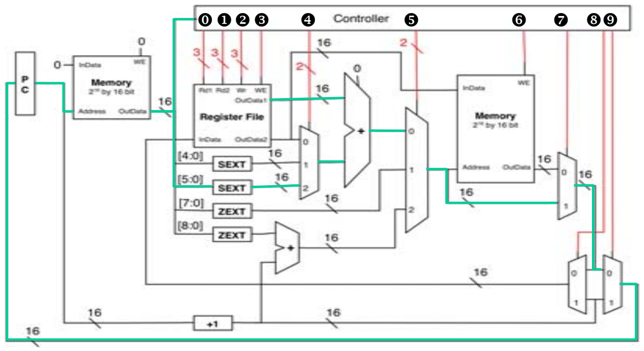
- Suppose the max. number of cycles an instruction takes is 8
- Then we would have 3-bit counter whose outputs are fed into 3 x 8 decoder
- The output of the decoder, T_0 to T_7 are enable based on count i.e.
  - ➢ T_0 = 1 when count = 000 (cycle 0), all others are disabled
    …
  - ➢ T_7 = 1 when count = 111(cycle 7), all others are disabled
- The decoder output that is enabled used to define the behavior in a particular cycle
- If < 8 clock cycles are required by another instr., then the counter is reset back to 0 (so the next instruction can properly function as well)

Hardwire Control with Timing (Sequencing) Control



CIT 595

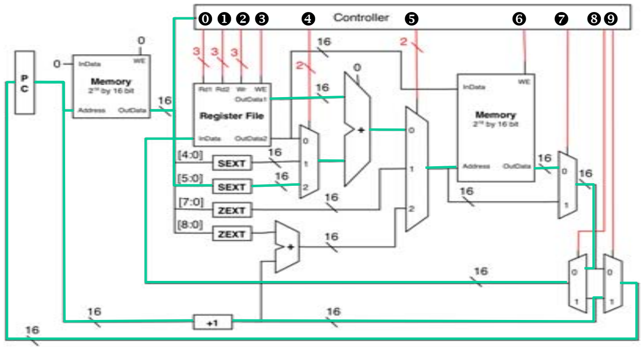
JMP Instruction



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

CIT 595

LDR Instruction

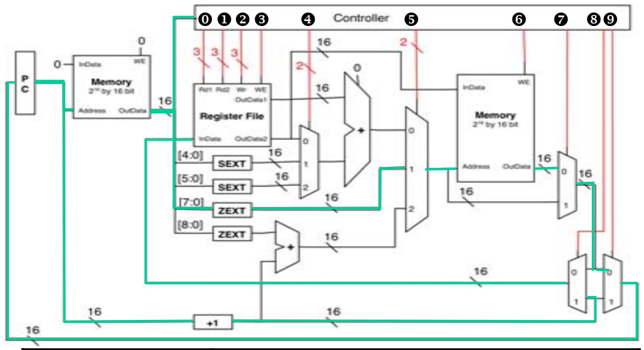


	Opcode		Control									
Instr	I[15:12]	I[5]	⓿	❶	❷	❸	❹	❺	❻	❼	❽	❾
LDR	0110	-	I[8:6]	-	I[11:9]	1	10	00	0	0	0	1

BaseR DR

CIT 595

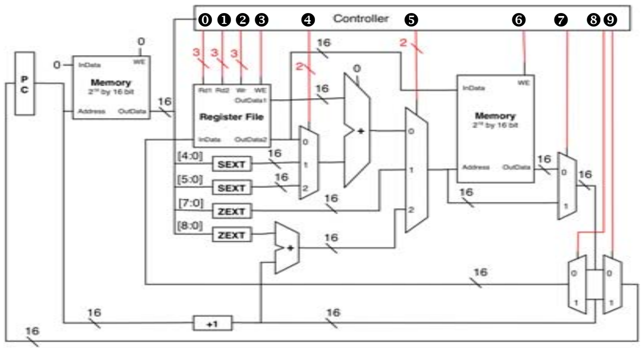
TRAP



	Opcode		Control									
Instr	I[15:12]	I[5]	⓿	❶	❷	❸	❹	❺	❻	❼	❽	❾
TRAP	1111	-	-	-	111	1	xx	01	0	0	1	0

CIT 595

Implementation Diagram is not complete

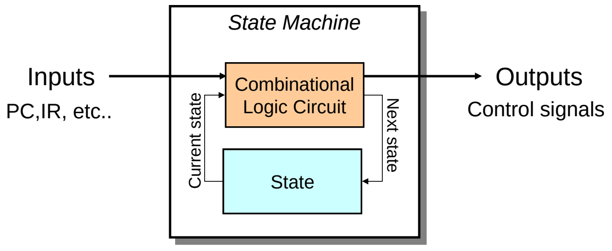


What about AND? NOT? BR? What changes would you make?

Approach II: Microprogrammed Control

- In microprogrammed control, each machine instruction is in turn implemented by a series of microinstructions
- Machine instructions are the input for a microprogram that converts the 1s and 0s of an instruction into control signals
- The microinstructions are often stored in firmware or read only memory, which is also called the control store
- Microprogram is also known as Microcode in some literature
- Microprogram Control is essentially a Finite State Machine

Microprogrammed Control is a FSM

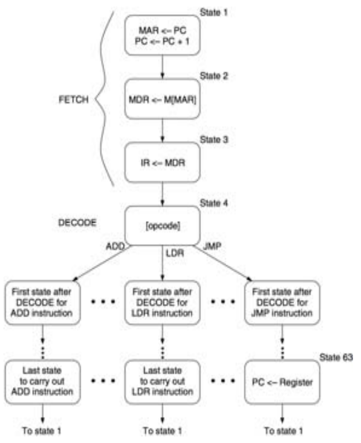


LC3 Microprogrammed Control: State Diagram

Finite state machine
- Input: PC, IR, etc..
- Output: *many* control signals

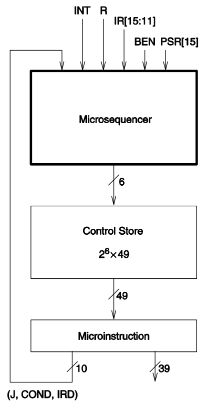
Need to map abstract ops to control signals
- *E.g.,* MAR <- PC
  ⇒ GatePC and LD.MAR
- *E.g.,* PC <- PC + 1
  ⇒ PCMUX=2 and LD.PC

If in state 1, then the micro-Instruction for state 1 will enocode information for GatePC, LD.MAR, PCMUX, LD.PC and next state as state 2


State 1
MAR <- PC
PC <- PC + 1
State 2
FETCH
MDR <- M[MAR]
State 3
IR <- MDR
State 4
DECODE
[opcode]
ADD
LDR
JMP
First state after DECODE for ADD instruction
First state after DECODE for LDR instruction
First state after DECODE for JMP instruction
State 63
Last state to carry out ADD instruction
Last state to carry out LDR instruction
PC <- Register
To state 1
To state 1
To state 1

LC3 Implemented using Microprogram Control

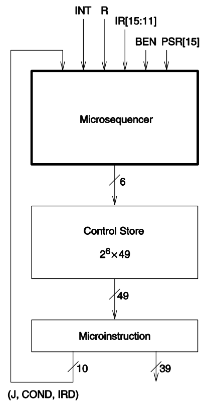
- The behavior of LC-3 during a given clock cycle is completely described by the 49 bit microinstruction
  - 39 control signals to assert datapath components
  - 10 signals + 9 other to determine the control signals for the next clock cycle
- Each phase of instruction cycle may require more than one microinstruction
- Hence a microinstruction is retrieved during each clock cycle



Microprogram Control

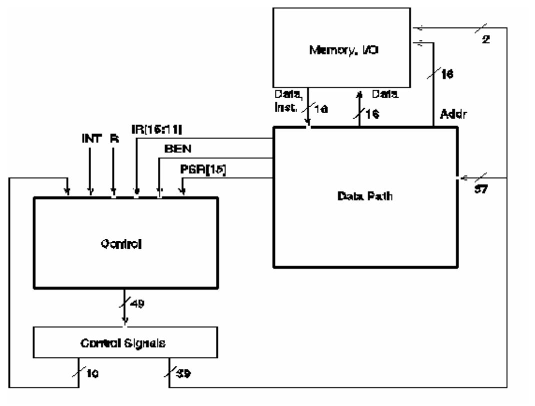
LC3 Implemented using Microprogram Control (contd..)

- All possible processor behavior (state) is stored into memory called Control Store
  - i.e. each location stores one microinstruction
  - There are 52 possible microinstructions (states) that can describe LC3's behavior
  - Hence need a 6-bit address to lookup the control store
- The microsequencer produces the 6 bit address from combination of 10 bits of Microinstruction + 9 bit additional info, which will correspond to the next behavior of the processor

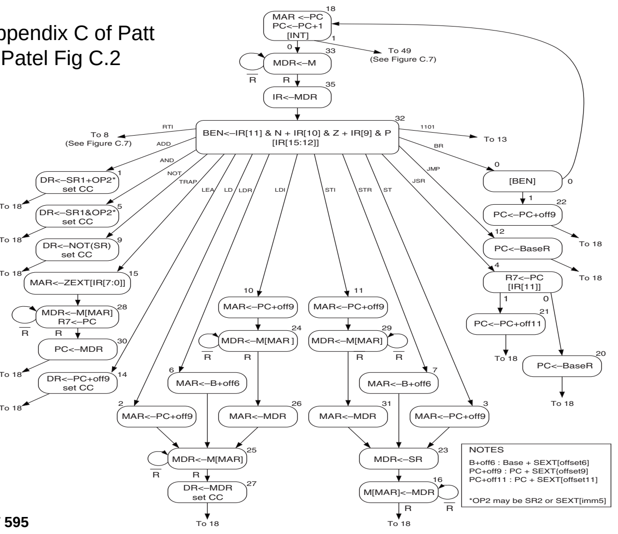


Microprogram Control

Big Picture: LC3 as Microprogrammed Control



Appendix C of Patt & Patel Fig C.2



LC3 Microprogram: Control Signals for Current State

CURR STATE	GATE			LD						MUX			MISC				
	MARMX	PC	ALU	MDR	PC	REG	CC	IR	MAR	MAR	PC	SR2	SR1	ALUK	DR	R.W	MEM EN
18	0	1	0	0	1	0	0	0	1	X	0 (PC + 1)	X	X	X	X	0	0
1	0	0	1	0	0	1	1	0	0	X	X	IR[5]	IR[8:6]	0 (ADD)	IR[11:9]	0	0

The table above provides the control signal values (some of the 39 signals) for two states (18, 1)

State 18: Is performing part of the FETCH stage

State 1: Is performing stages OP-EX-STORE for ADD inst

Note:

• Assume all Register and Memory Read/Write signal are set 0 unless in case of write (i.e. set to 1)

• Also there is no need of timing circuit like in hardwired control, as each signal behavior is defined for every clock cycle

LC3 Microprogram: Next State

Depends on:

- 10 bits of current microinstruction
  - J (6-bits): encodes the next state (mostly likely states of the possible next states)
  - COND (3-bits): field indicates special tests that must occur to compute the true next state
    - 0 – Unconditional (that next state is the encoded state)
    - 1 – Memory Read
    - 2 – Branch
    - 3 – Addressing Mode (for JSR and JSRR instructions)
    - 5 – Interrupt Test
  - IRD (1-bit) : If set to IRD = 1, ignore J and COND. This only happens in state 32 and as we want to use the bits from the IR to select the next state

LC3 Microprogram: Next State (contd..)

Depends on (contd..):

• INT: To indicate interrupt from another program or device, Only tested if in state 18 (because that is before the start of an instruction cycle)

• R: indicate the end of memory operation

• IR[15:11]: current opcode

• PSR[15]: processor executing in supervisor or user mode

• BEN: indicates whether or not a branch should be taken

LC3 Next State Example

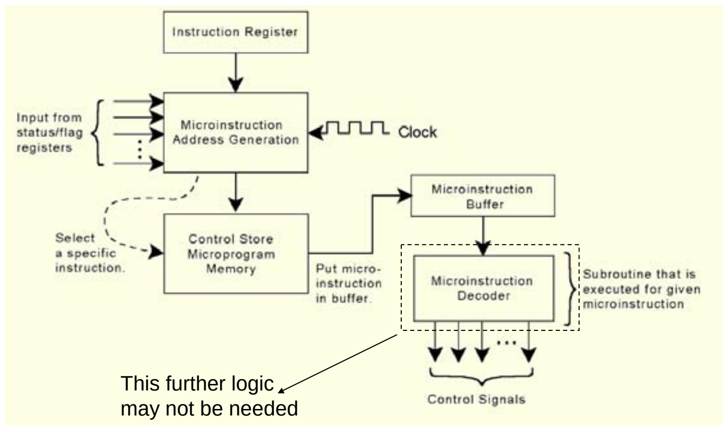
CURR STATE	J	COND	IRD	POSSIBLE NEXT STATE (depends also on 9 other Bits)
18	33	5	0	33,49
1	18	0	0	18
32	0	0	1	0-15

• State 18: Most likely next state is 33 but need to check for INT (COND = 5). If INT = 1, Next State = 49

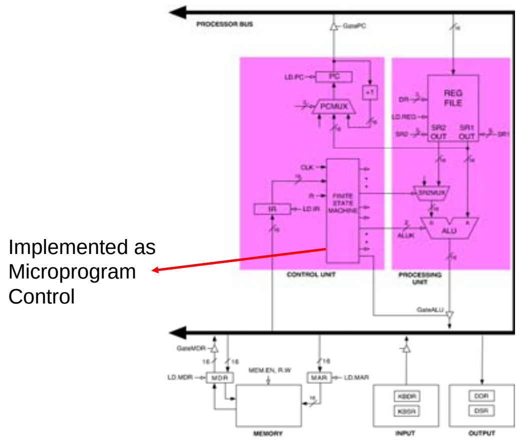
• State 1: Next State is just 18 (this because you are done finishing ADD instr and want to start a new instr. Cycle)

• State 32: J and COND ignored as IRD = 1, 0 – 15 are your next possible states

Generic Microprogram Control


Instruction Register
Input from status/flag registers
Microinstruction Address Generation
Clock
Select a specific instruction.
Control Store Microprogram Memory
Put micro-instruction in buffer.
Microinstruction Buffer
Microinstruction Decoder
Subroutine that is executed for given microinstruction
This further logic may not be needed
Control Signals

LC3 Processor



Hardwired vs. Microprogrammed

Complexity

- There is an extra level of instruction interpretation in microprogrammed control, which makes it slower than hardwired control

Flexibility

- Instruction and Control Logic are tied together in hardwired control, which makes it difficult to modify
- New instructions can be easily added by only making changes to the microprogram in microprogrammed control implementation