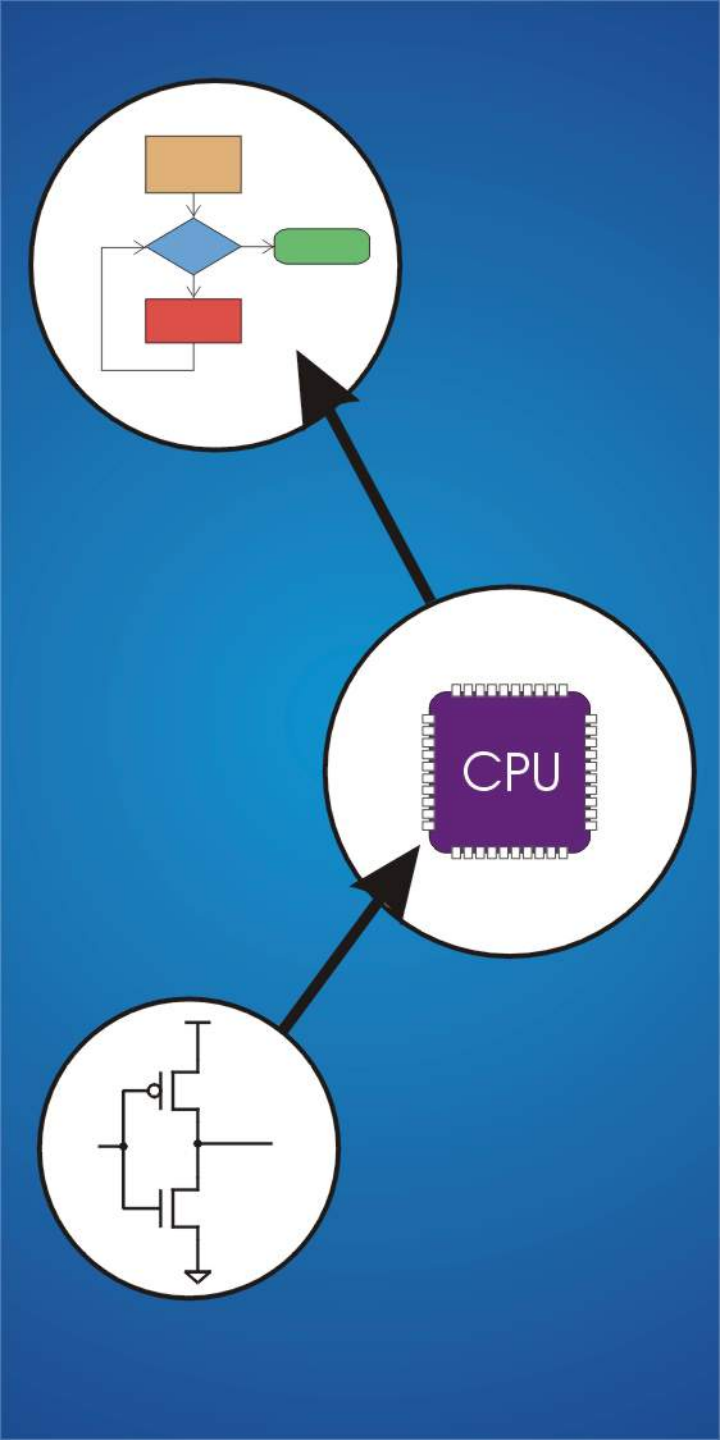
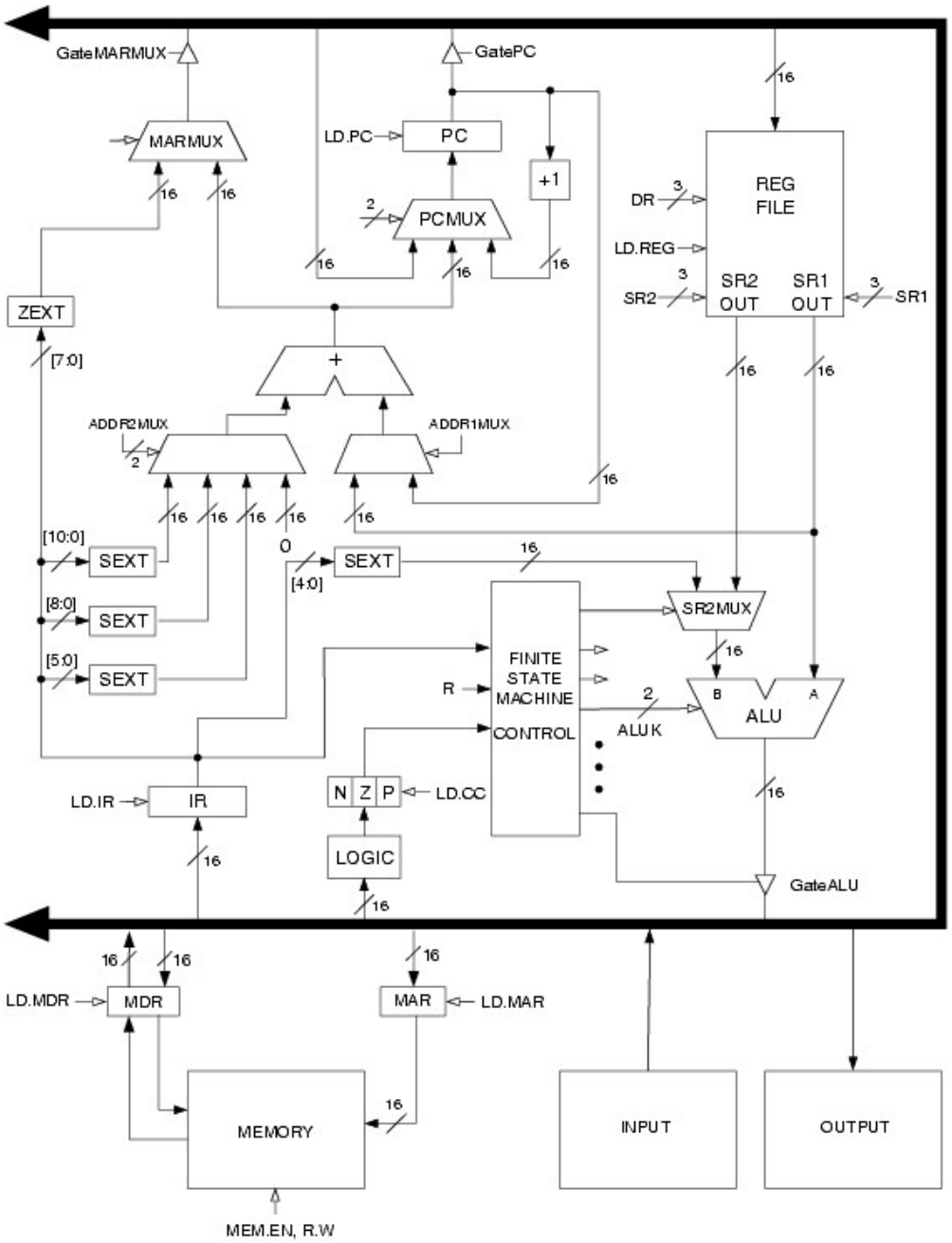



# The Microarchitecture of the LC-3

# LC-3 Data Path Revisited

**Now Registers and Memory**



# From Logic to Data Path

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

- **See the data path of the LC-3 on next slide.**

## Combinational Logic

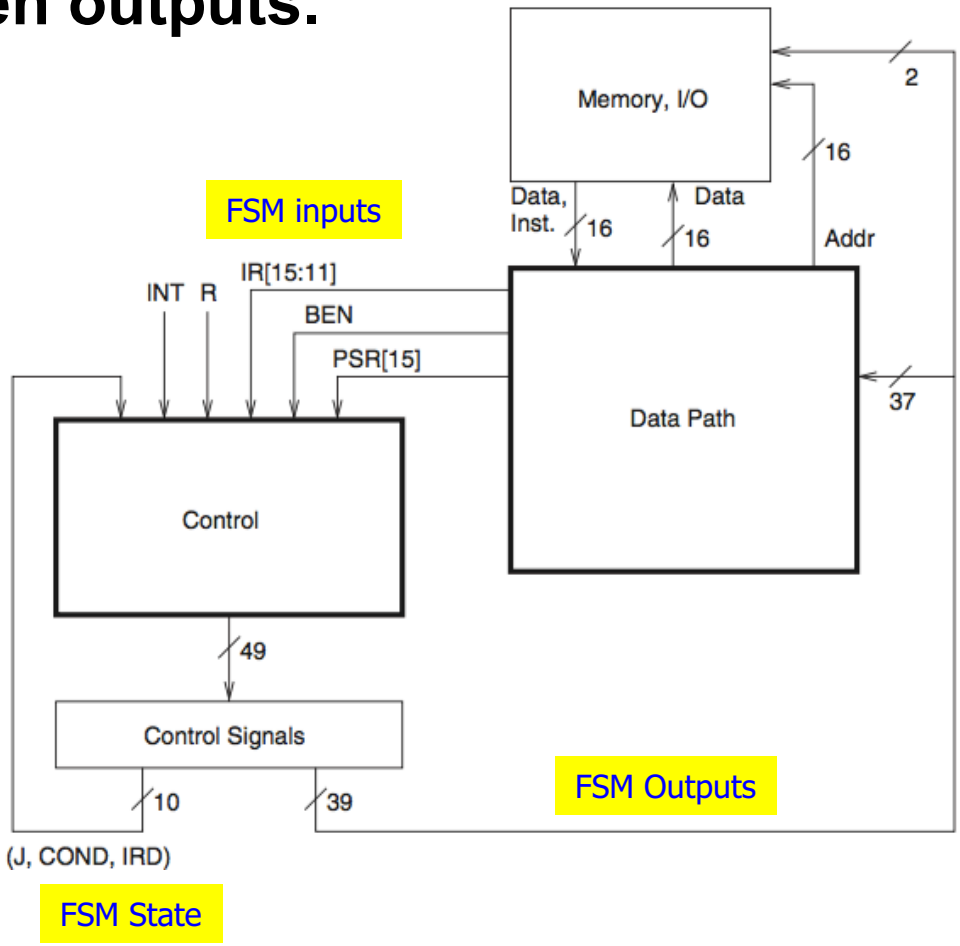
- **Decoders -- convert instructions into control signals**
- **Multiplexers -- select inputs and outputs**
- **ALU (Arithmetic and Logic Unit) -- operations on data**

## Sequential Logic

- **State machine -- coordinate control signals and data movement**
- **Registers and latches -- storage elements**

# LC-3 Control Architecture

## Control FSM: Design a FSM with given outputs.




| Signal Name | Signal Values | |
|---|---|---|
| LD.MAR/1: | NO, LOAD | |
| LD.MDR/1: | NO, LOAD | |
| LD.IR/1: | NO, LOAD | |
| LD.BEN/1: | NO, LOAD | |
| LD.REG/1: | NO, LOAD | |
| LD.CC/1: | NO, LOAD | |
| LD.PC/1: | NO, LOAD | |
| LD.Priv/1: | NO, LOAD | |
| LD.SavedSSP/1: | NO, LOAD | |
| LD.SavedUSP/1: | NO, LOAD | |
| LD.Vector/1: | NO, LOAD | |
| GatePC/1: | NO, YES | |
| GateMDR/1: | NO, YES | |
| GateALU/1: | NO, YES | |
| GateMARMUX/1: | NO, YES | |
| GateVector/1: | NO, YES | |
| GatePC-1/1: | NO, YES | |
| GatePSR/1: | NO, YES | |
| GateSP/1: | NO, YES | |
| PCMUX/2: | PC+1 | ;select pc+1 |
| | BUS | ;select value from bus |
| | ADDER | ;select output of address adder |
| DRMUX/2: | 11.9 | ;destination IR[11:9] |
| | R7 | ;destination R7 |
| | SP | ;destination R6 |
| SR1MUX/2: | 11.9 | ;source IR[11:9] |
| | 8.6 | ;source IR[8:6] |
| | SP | ;source R6 |
| ADDR1MUX/1: | PC, BaseR | |
| ADDR2MUX/2: | ZERO | ;select the value zero |
| | offset6 | ;select SEXT[IR[5:0]] |
| | PCoffset9 | ;select SEXT[IR[8:0]] |
| | PCoffset11 | ;select SEXT[IR[10:0]] |
| SPMUX/2: | SP+1 | ;select stack pointer+1 |
| | SP−1 | ;select stack pointer−1 |
| | Saved SSP | ;select saved Supervisor Stack Pointer |
| | Saved USP | ;select saved User Stack Pointer |
| MARMUX/1: | 7.0 | ;select ZEXT[IR[7:0]] |
| | ADDER | ;select output of address adder |
| VectorMUX/2: | INTV | |
| | Priv.exception | |
| | Opc.exception | |
| PSRMUX/1: | individual settings, BUS | |
| ALUK/2: | ADD, AND, NOT, PASSA | |
| MIO.EN/1: | NO, YES | |
| R.W/1: | RD, WR | |
| Set.Priv/1: | 0 | ;Supervisor mode |
| | 1 | ;User mode |

# LC-3 Control Architecture

**Control FSM: Design a FSM with given state diagram**

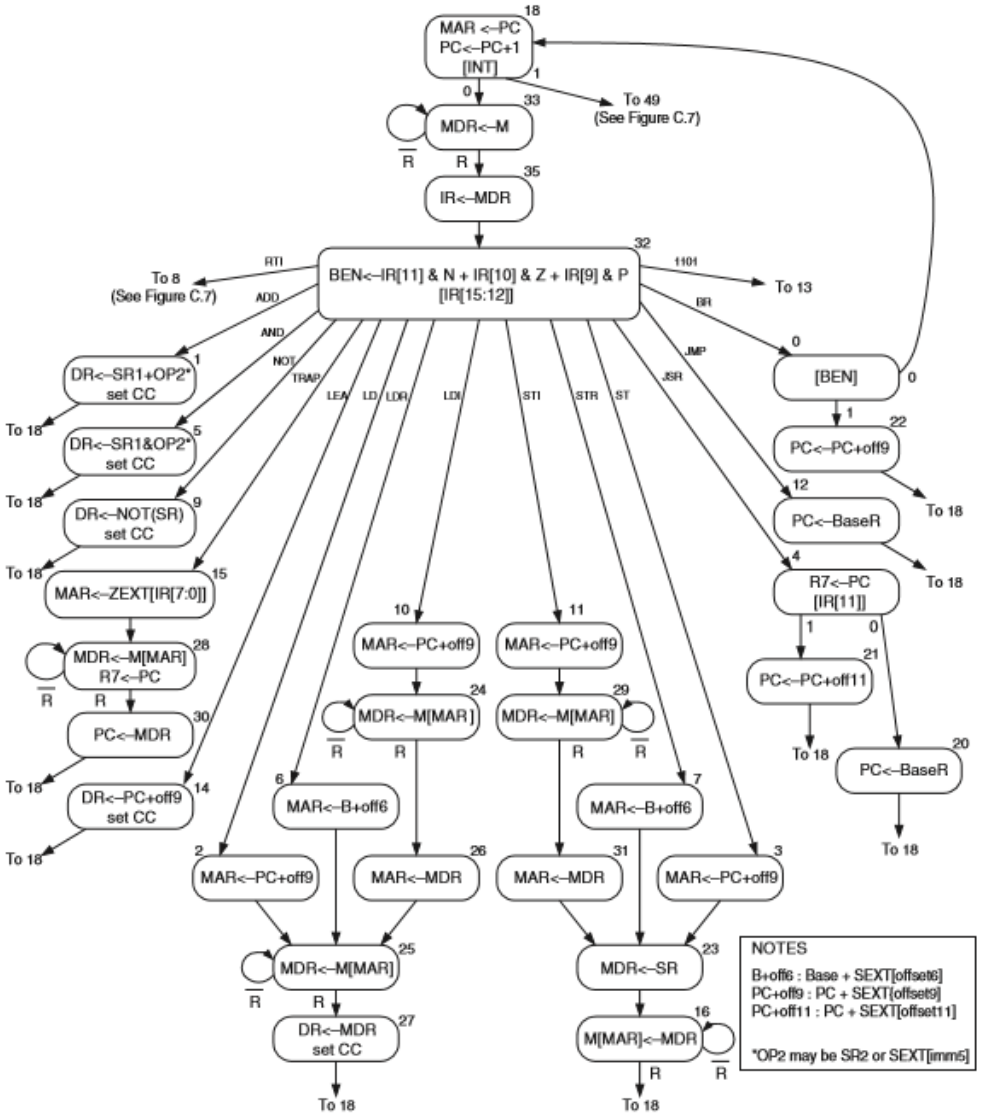



Figure C.2 A state machine for the LC-3

# FSM Description

**Instruction Fetch:**

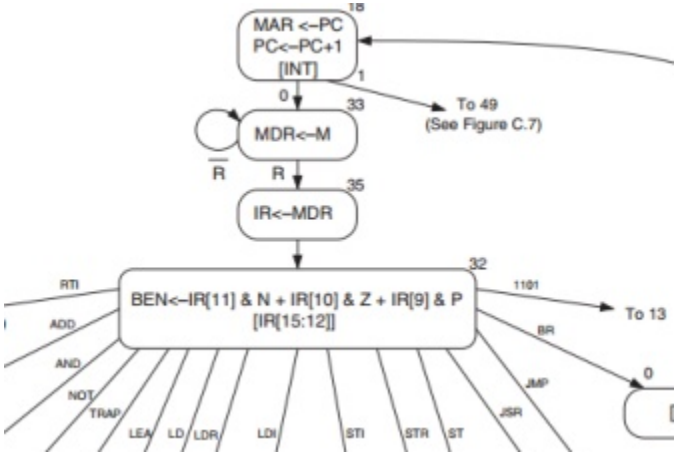
**S18: MAR<-PC, PC<-PC+1,**

**If no INT, go to S33**

**To implement**

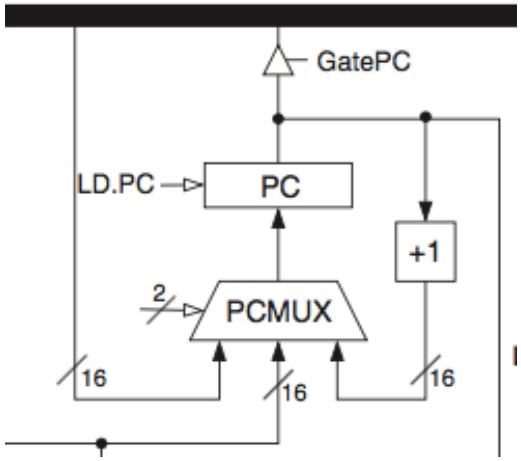
**MAR<-PC needs: GatePC =1, LD.MAR = 1,**

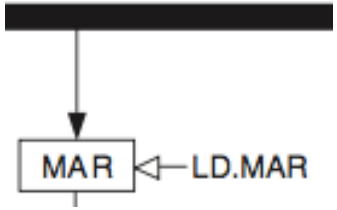
**PC = PC+1 needs: PCMUX select PC+1, LD.PC =1**



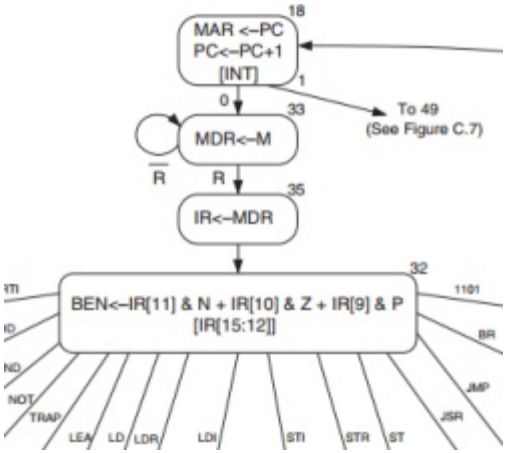

**For each state**

- **activate or select appropriate Control signals**
- **Go to appropriate next state**



# Example: ST instruction




**Instruction Fetch:**

**S18: MAR<-PC, PC<-PC+1, If no INT, goto S33**

**S33: MDR<-M, if R goto 35**

**S35: IR<- MDR, goto 32**

**Decode:**

**S32: BEN = [ IR[11] & N+IR[10] & Z+IR[9] &P[IR[15:12]] ] go to specific state**

**Evaluate address:**

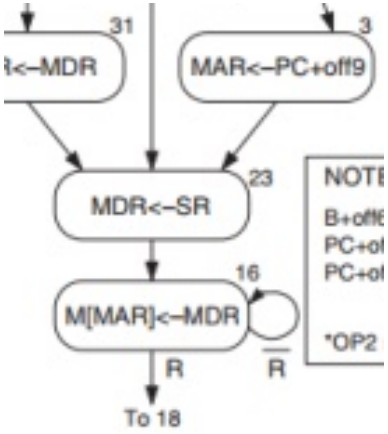
**S3: MAR<- PC + off9 , goto 23**

**Fetch Operand:**

**S23: MDR<-SR, goto 16**

**Store result:**

**S16: M[MAR] <- MDR, If R goto S18**




**As you can see some instructions need fewer cycles.**

# Processor Design Homework

**You need to design a tiny processor with a given ISA.**

- **Design the Data Flow part**
  - **We will do that for you.**
- **Identify the control signals needed**
  - **We will do that for you.**
- **Describe the sequential behavior using RTL/state diagram**
  - **We will do that too.**
- **Design the controller as a finite state machine**
  - **Flip-flops to hold the state**
  - **Logic to generate control signals and to go to the next state**
- **Construct, simulate and validate your design**

# Control Design options

**Hardwired control:**

- **Design the FSM using any hardware and optimize it.**
- **Large combinational blocks are usually designed using a PLA.**
- **Approach came back into vogue with RISC philosophy.**

**Microprogrammed control**

- **Fixed structure with a microsequencer**
- **Control signals stored in a ROM (or PROM)**
- **Control design then becomes "writing microinstructions"**
- **Changing/adding design easier, was popular at a time.**

**Modern processors use Verilog/VHDL specification automating part of the design.**

- **CSU was a major research center on Microprogramming for several years.**
- **I was personally responsible for dropping the term "Microprogramming" for the major international meeting in Dublin as chair of IEEE CS TC-Micro.**

# Control Design for Parallel processors

## Pipelining

- Multiple instructions active at the same time
- An instructions is finishing, some partly done and a new one being fetched.
- Control signals for several instructions active at the same time!

**Superscalar Processors:**

- Multiple instructions are being fetched at a single time.

**Simultaneous Multithreading Processors:**

- Hardware supports multiple threads running at the same time in a single processor.

**Multicore Processors:**

- Each core has its own control unit.