MIPS-Lite Multicycle Control

COE608: Computer Organization and Architecture

Dr. Gul N. Khan
http://www.ee.ryerson.ca/~gnkhan
Electrical and Computer Engineering
Ryerson University

Overview

- Introduction
- MIPS Multicycle Datapath
- Multicycle Control
- Microprogramming Concepts
- Microprogrammed Control

Sections 4.4 and 4.6

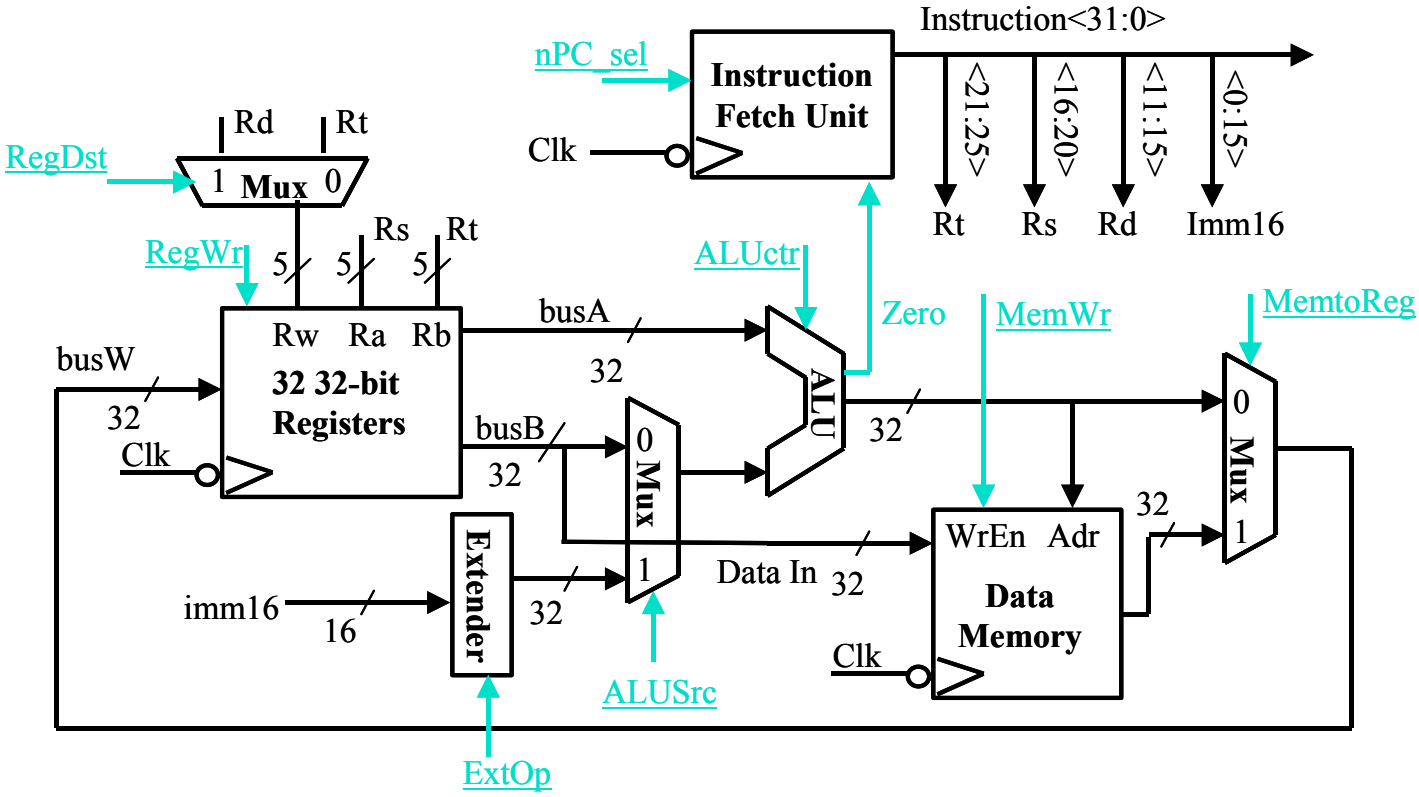
Multicycle Approach

- We will be reusing the functional units.
  - ALU used to compute address & increment PC.
  - Memory used for instruction and data.
- Control signals will not be determined solely by the instructions
- Control unit design by using classical FSM design is impractical due to large number of inputs and states it may have.
- An extension to the classical approach is used by experienced designer in designing control logic circuits:
  1. Sequence register and decoder method.
  2. One flip-flop per state method.
  3. Microprogram controller.
  4. PLA controller.

The PLA and micro-program control uses PLD and ROM/PROM.

Modification of PROM or replacing the ROM modifies the micro-program control.

Single Cycle Datapath



Long Cycle Time

- All instructions take as much time as the slowest.
- Real memory is not like our idealized memory

Cannot always get the job done in one (short) cycle

Arithmetic & Logical

PC	Inst Memory	Reg File	mux	ALU	mux	setup

Load

PC	Inst Memory	Reg File	mux	ALU	Data Mem	mux	setup

Store

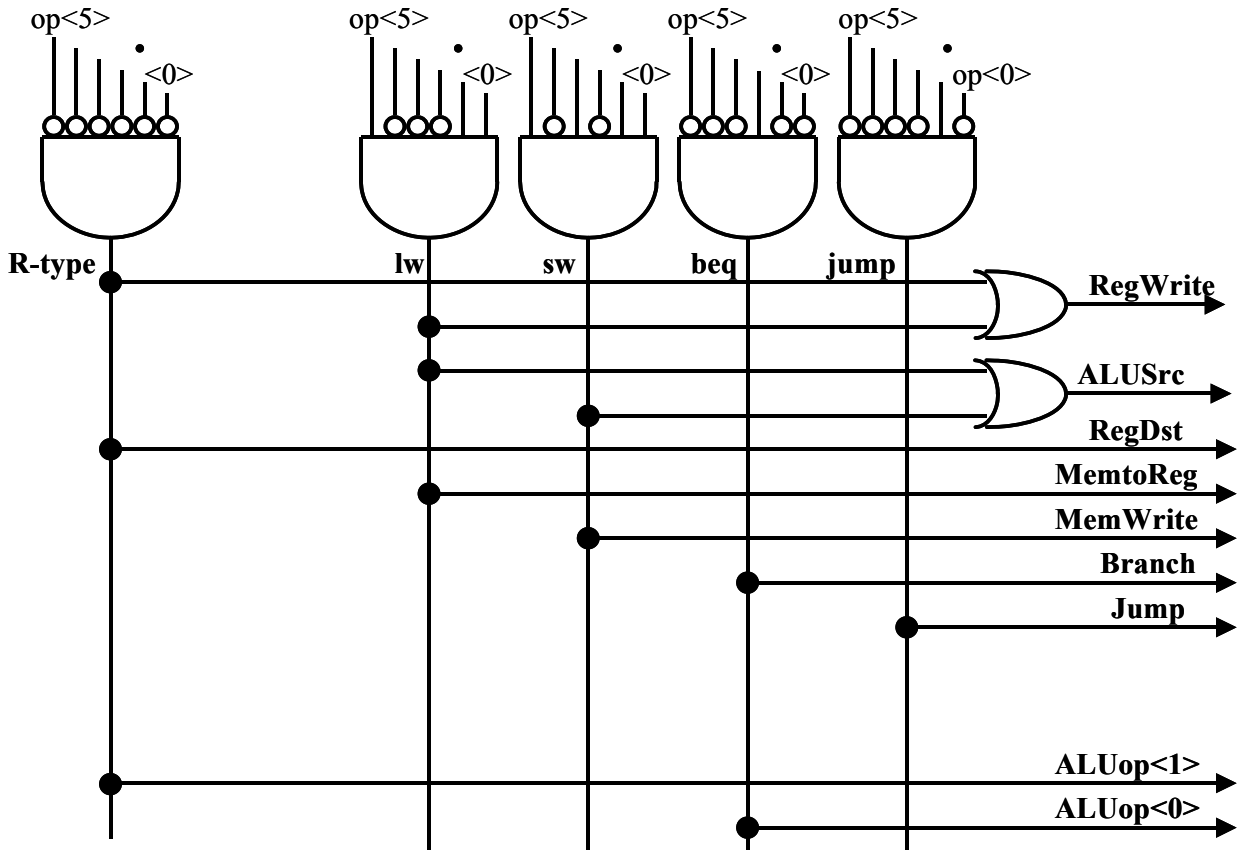
PC	Inst Memory	Reg File	mux	ALU	Data Mem

Branch

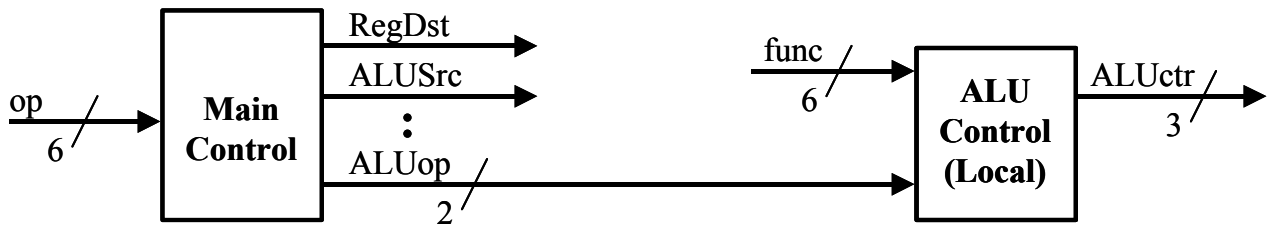
PC	Inst Memory	Reg File	cmp	mux

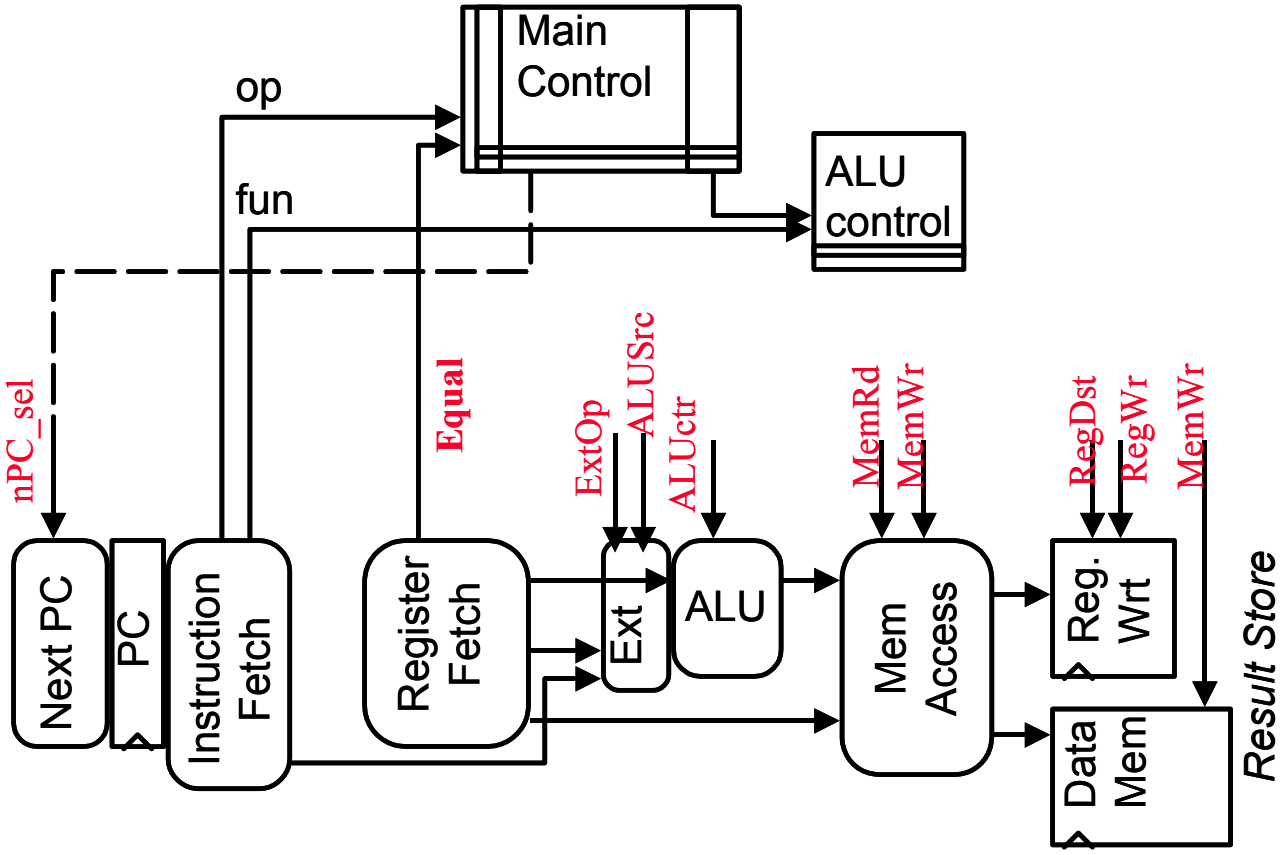
Single Cycle Control

op	00 0000		10 0011	10 1011	00 0100	00 0010
	R-type		**lw**	**sw**	**beq**	**jump**
RegDst	1		0	x	x	x
ALUSrc	0		1	1	0	x
MemtoReg	0		1	x	x	x
RegWrite	1		1	0	0	0
MemWrite	0		0	1	0	0
Branch	0		0	0	1	0
Jump	0		0	0	0	1
ALUop (Symbolic)	"R-type"		Add	Add	Subtract	xxx
ALUop <1>	1		0	0	0	x
ALUop <0>	0		0	0	1	x



Abstract View



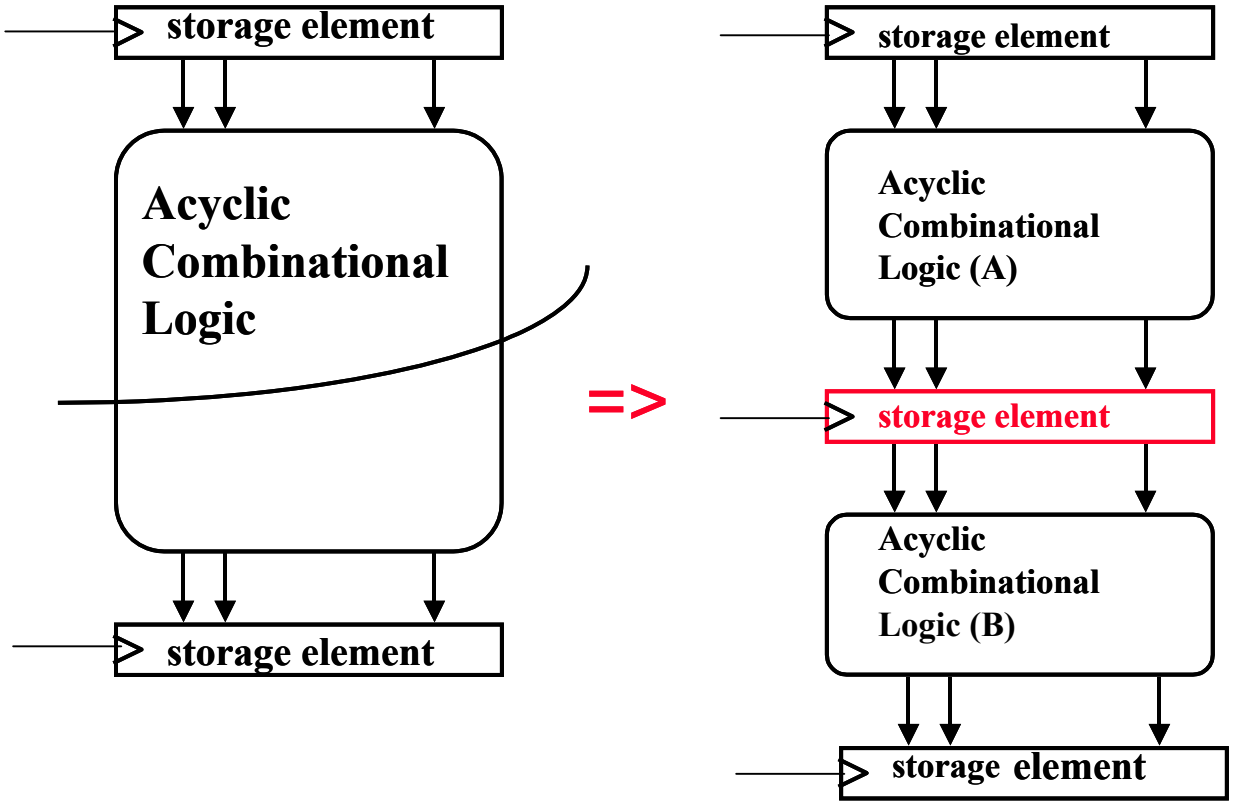


Similar to an FSM where state changes with a change in the PC value.

Reducing the Cycle Time

Cut combinational dependency graph and insert registers/latches

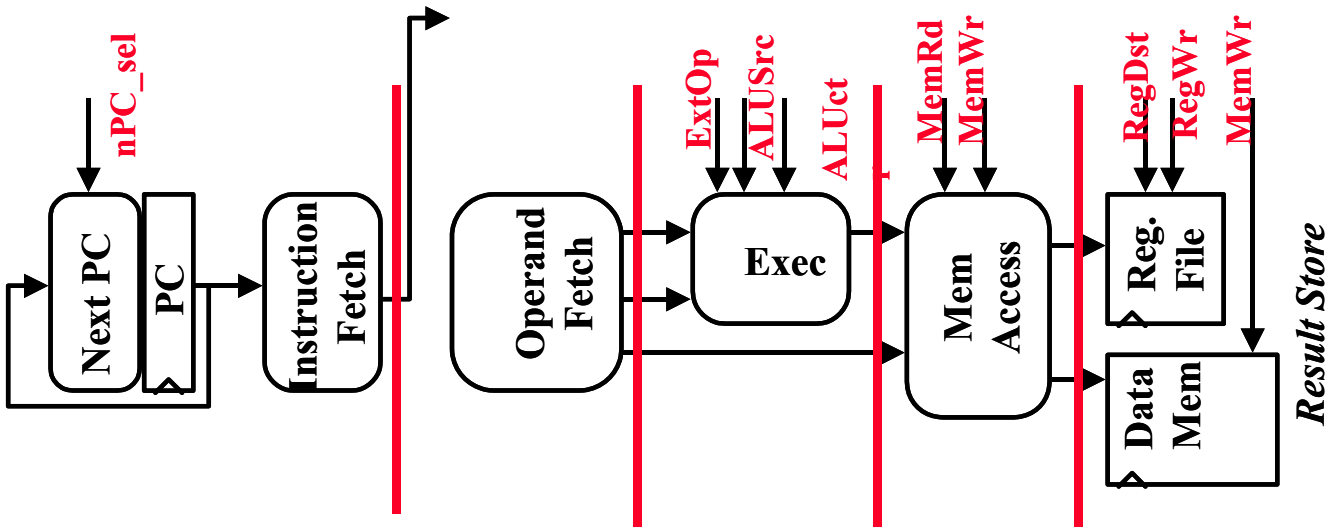
Do the same work in two fast cycles rather than one slower cycle.



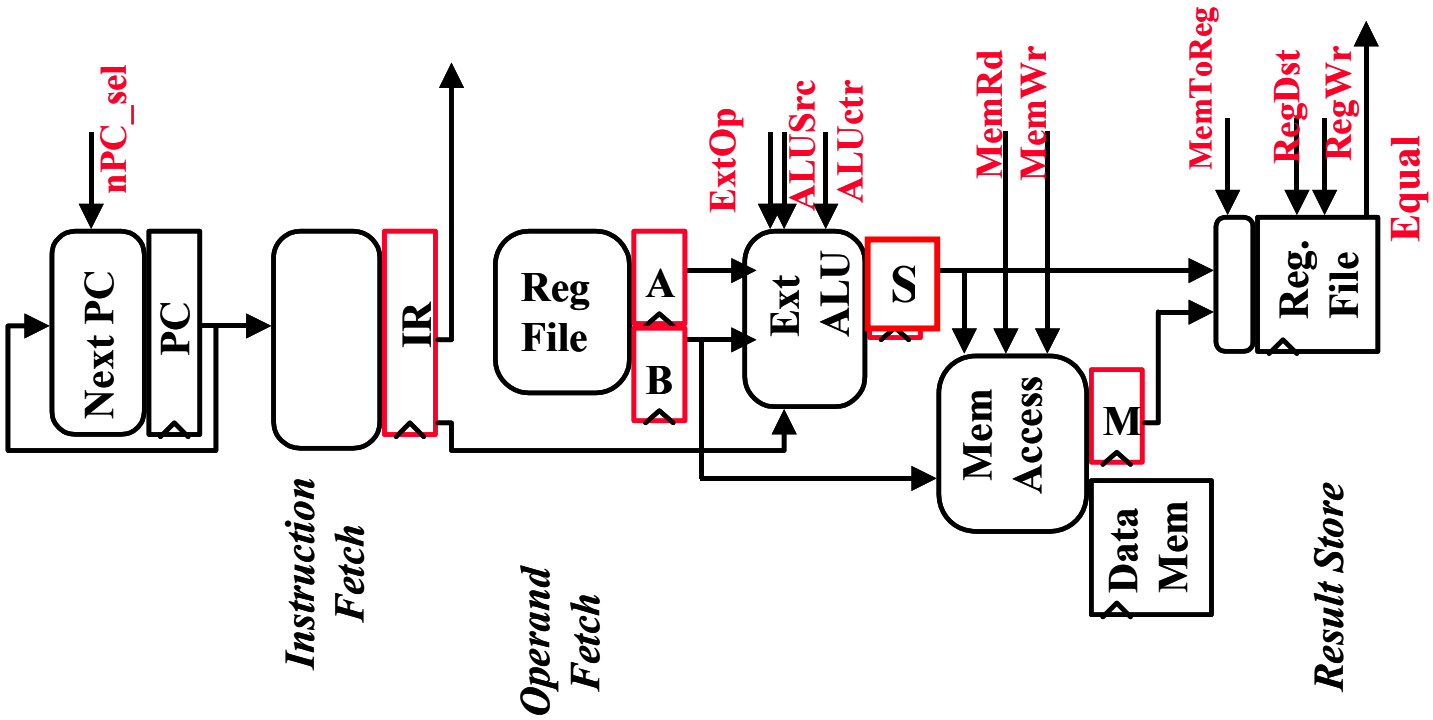
Multicycle Approach

Break instructions into steps

Partitioning Single-Cycle Datapath



Multicycle Datapath



R-type Instructions

(Add, Sub, etc. .)

Logical Register Transfer

ADDU R[rd] <= R[rs] + R[rt]; PC <= PC + 4

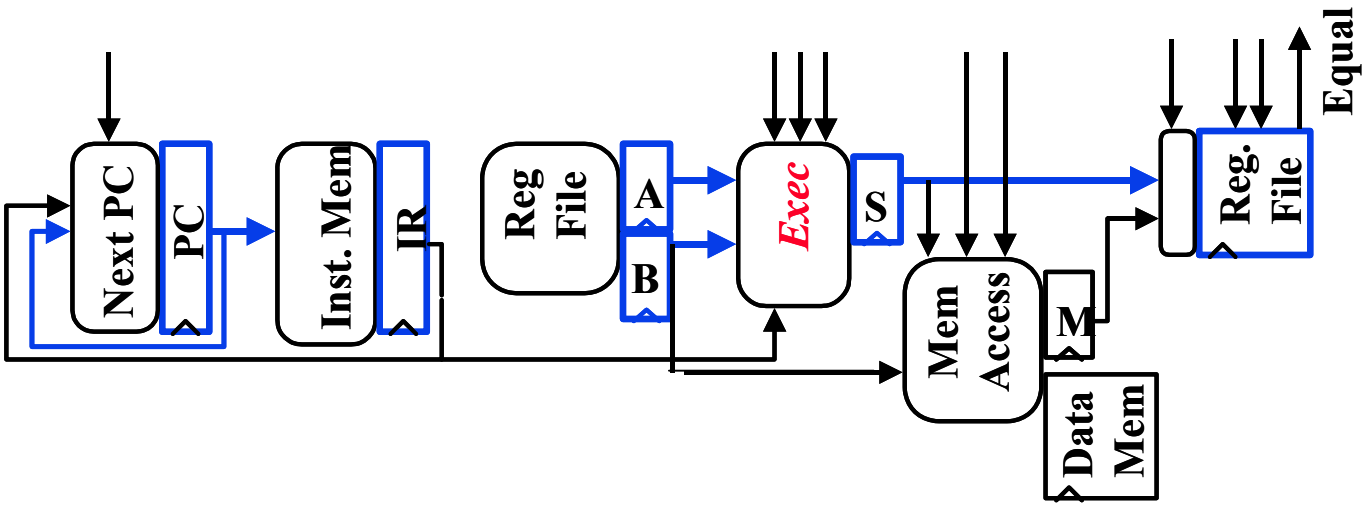
Physical Register Transfers

IR <= Mem [PC]

ADDU A<= R[rs]; B <= R[rt]

S <= A +/- B

R[rd] <= S; PC <= PC + 4



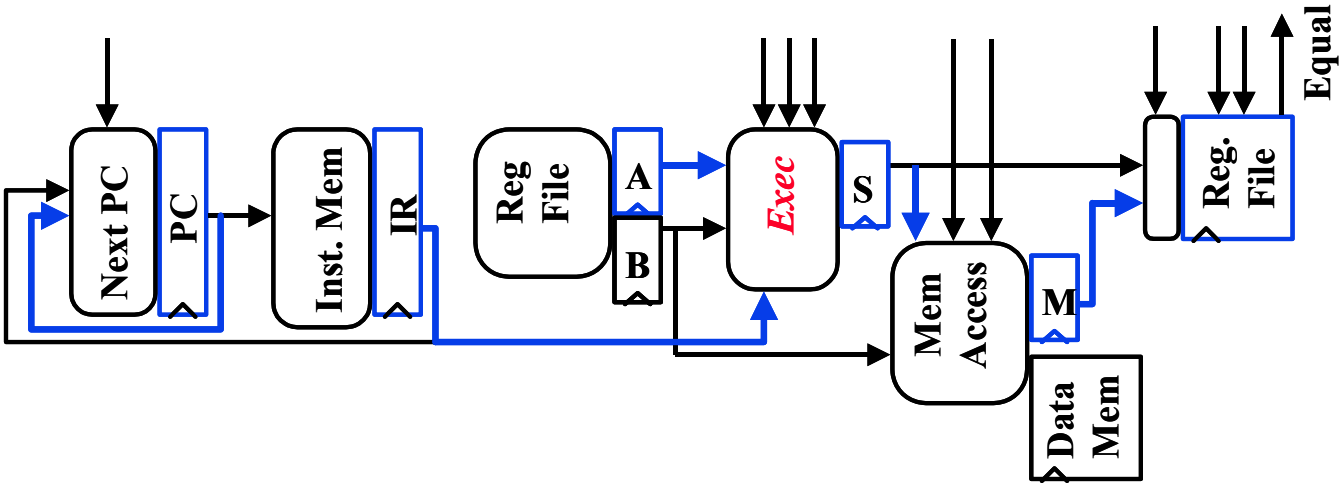
Load Instructions

Logical Register Transfer

LW R[rt] <= Mem (R[rs] + sx(Im16));
PC <= PC + 4

Physical Register Transfers

IR <= Mem [PC]
LW A<= R[rs] ;
S <= A + SignEx(Im16) ;
M <= Mem [S] ;
R[rd] <= M ; PC <= PC + 4



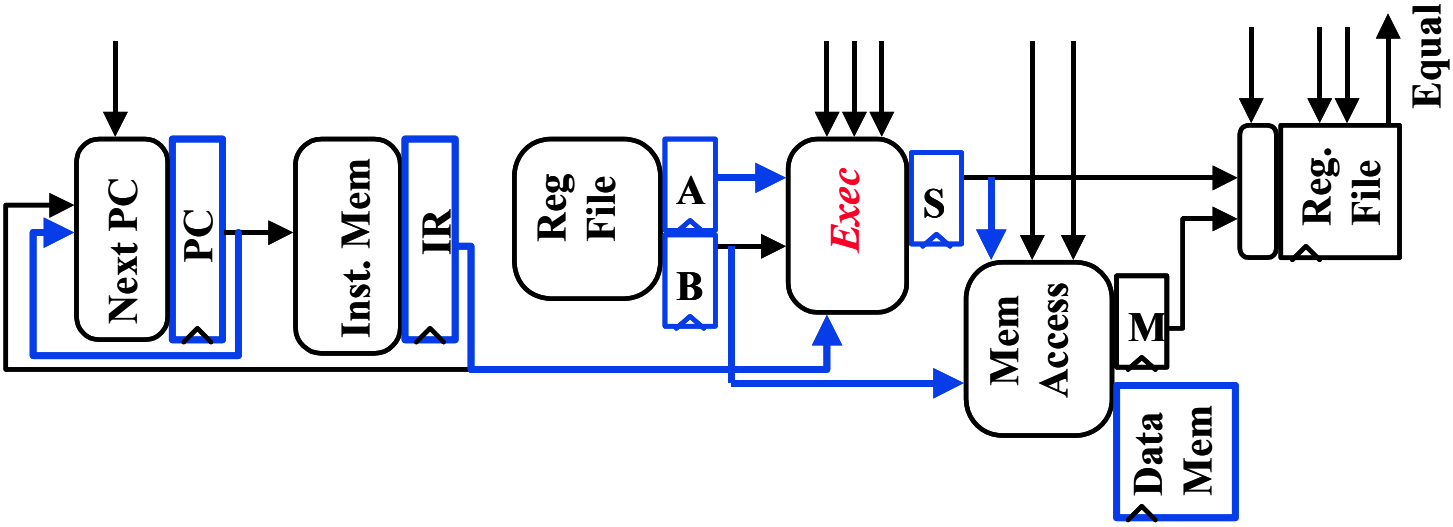
Store Instructions

Logical Register Transfer

SW MEM(R[rs] + sx(Imm16)) <= R[rt];
PC <= PC + 4

Physical Register Transfers

IR <= Mem[PC]
SW A<= R[rs] ; B <= R[rt];
S <= A + SignEx(Imm16) ;
Mem[S] <= B;
PC <= PC + 4



Branch Instructions

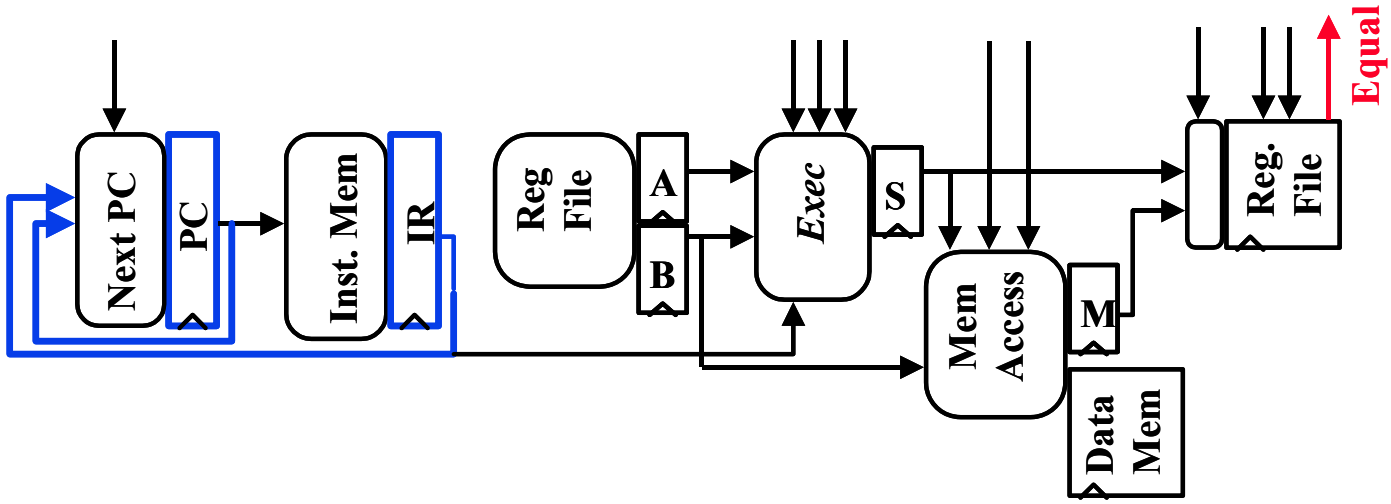
Logical Register Transfer

BEQ if R[rs] = = R[rt]
then PC <= PC + sx(Imm16) || 00
else PC <= PC + 4

Physical Register Transfers

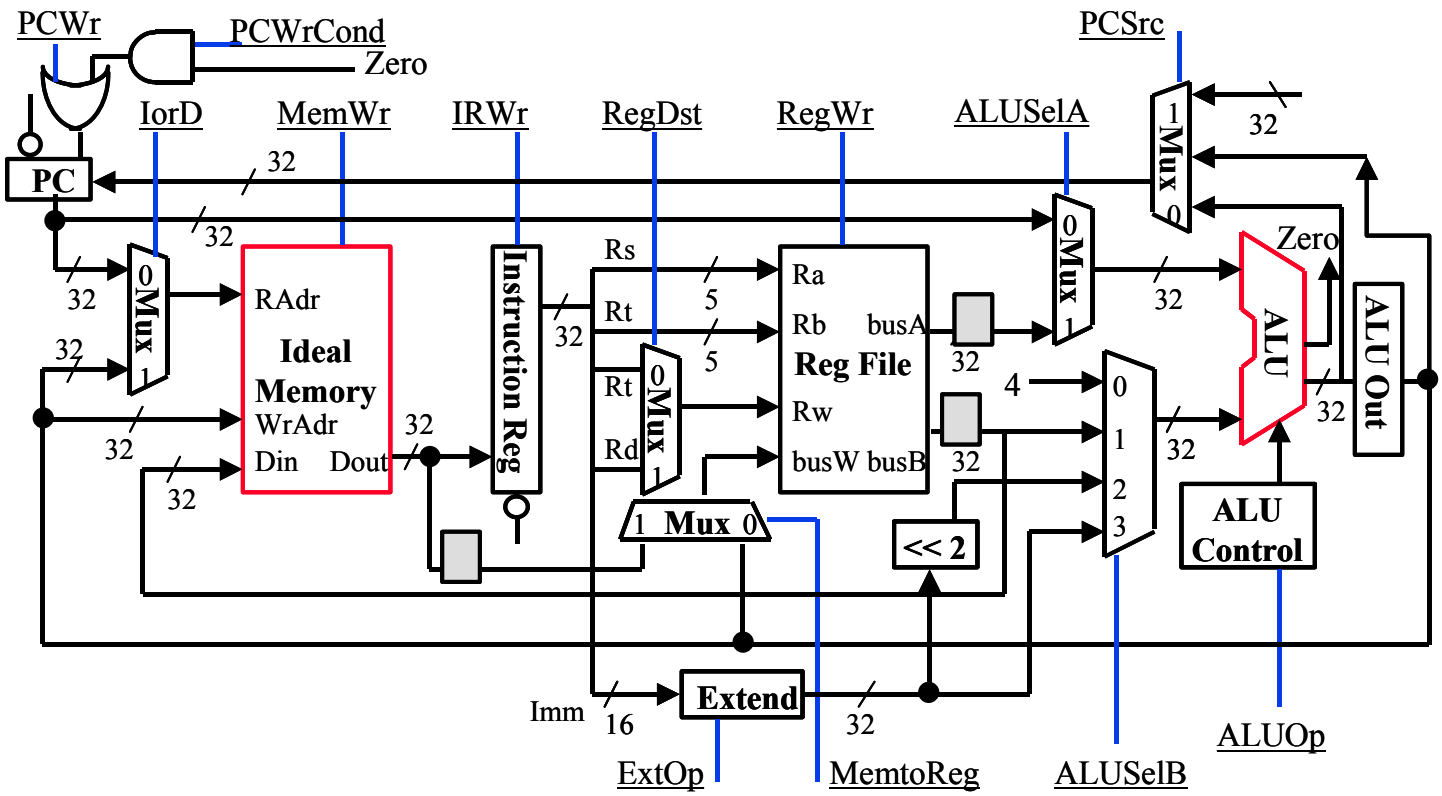
IR <= Mem [PC]

BEQ & not EQUAL PC <= PC + 4
BEQ & EQUAL PC <= PC + sx(Imm16) || 00 ;



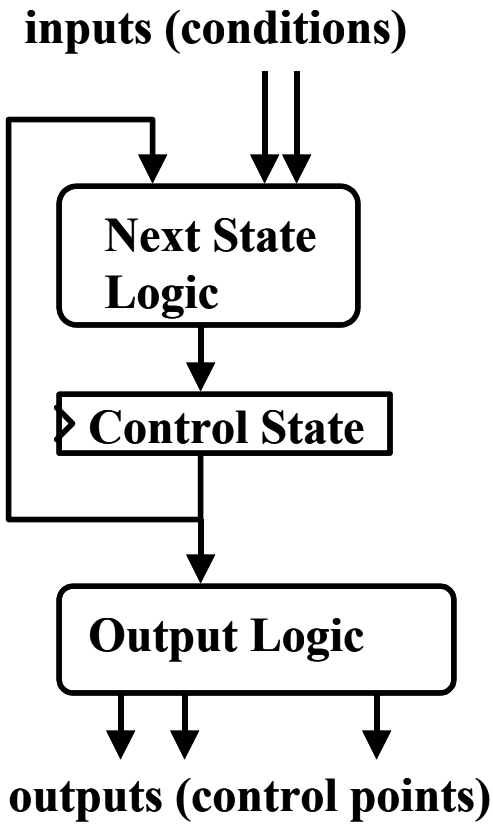
Multiple Cycle Datapath

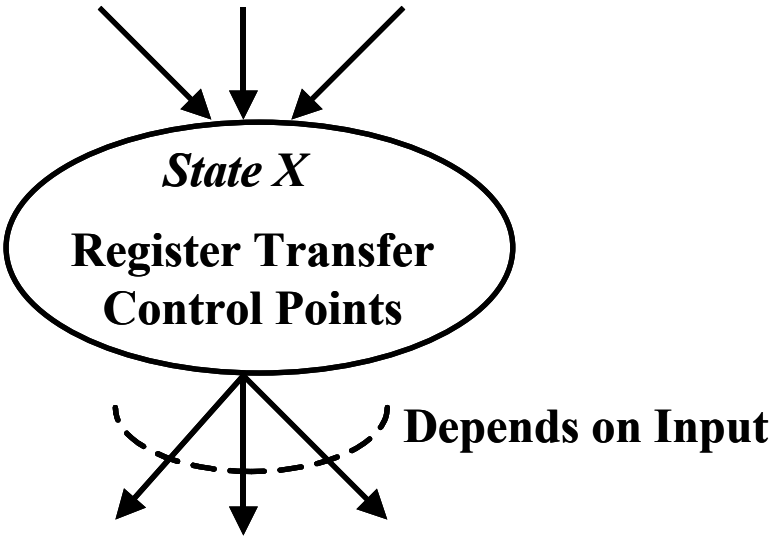
Miminizes Hardware:
One memory and one adder



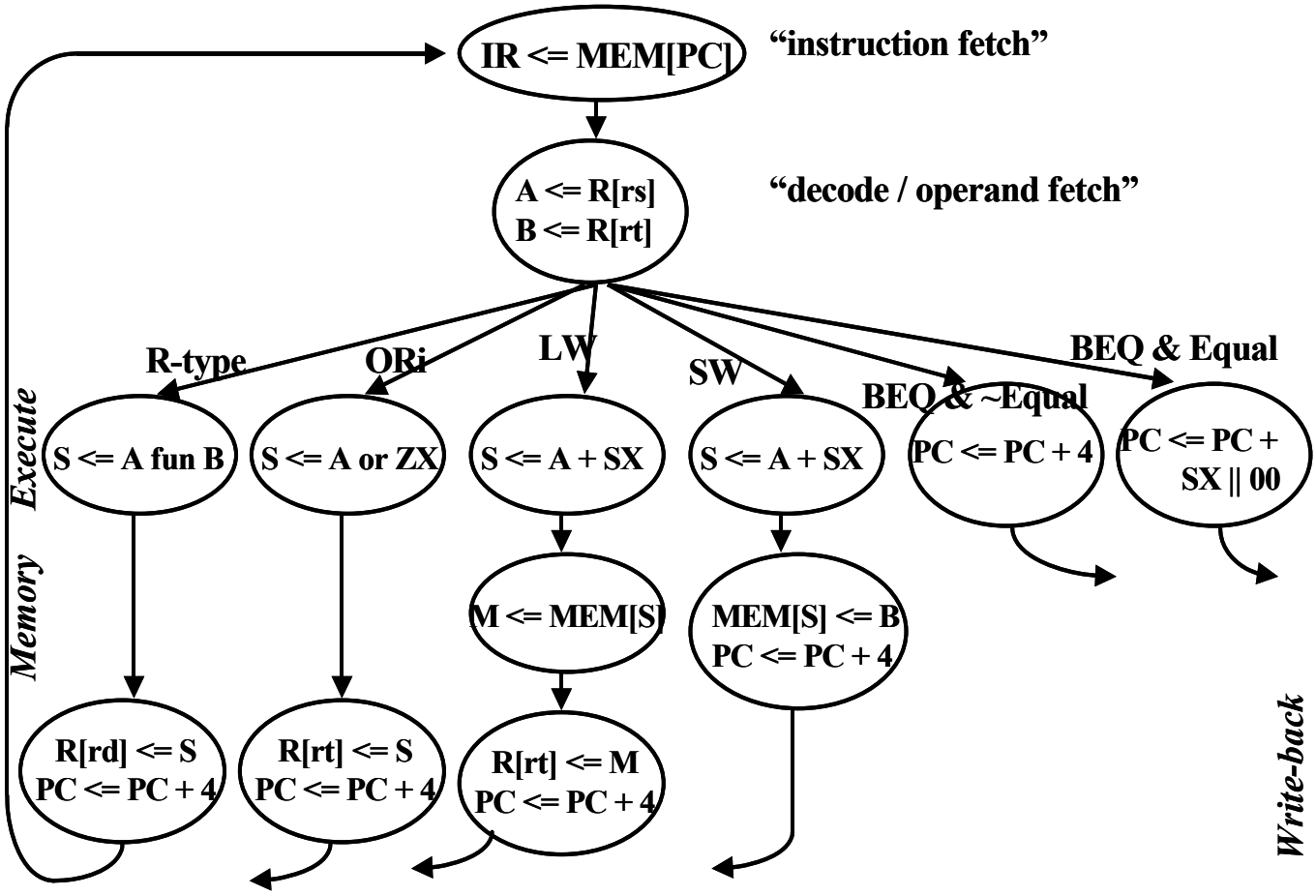
Control Model

State specifies control points for Register Transfer
Transfer occurs upon exiting state (falling edge) i.e. control outputs are of Mealy type.

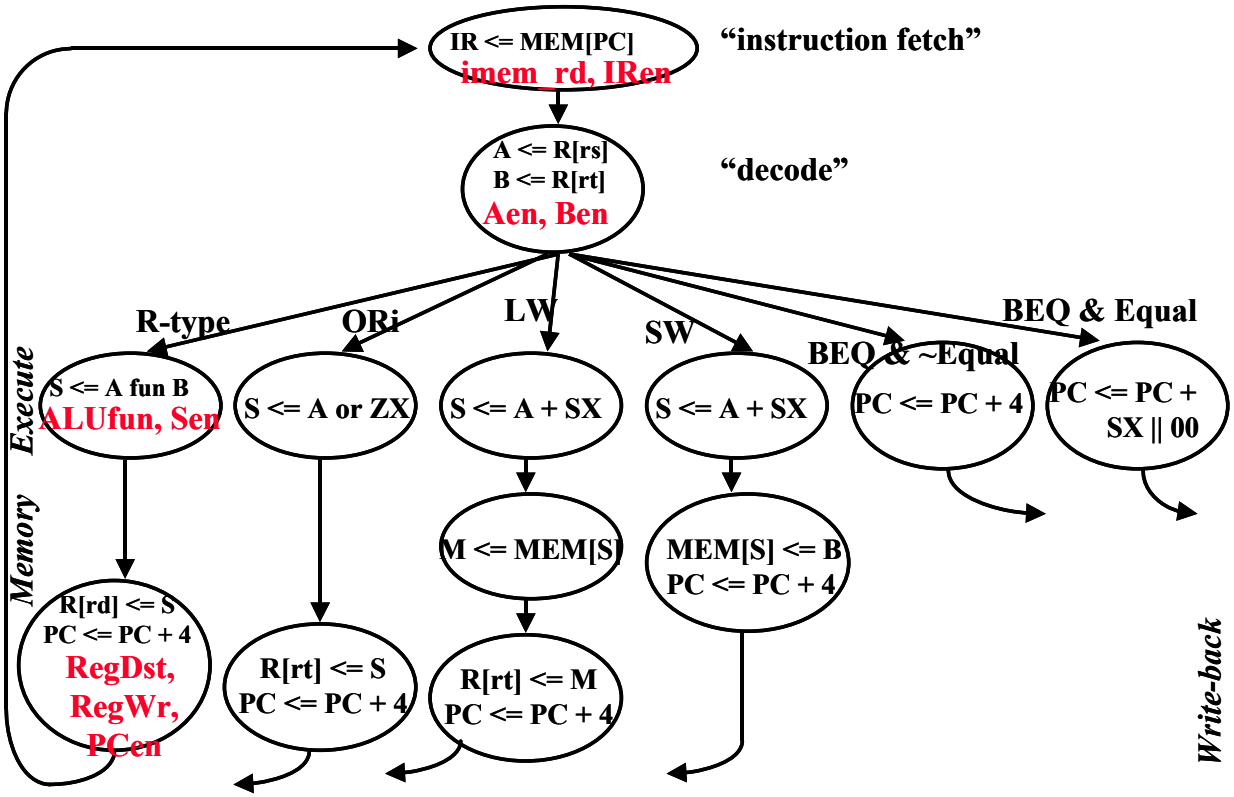




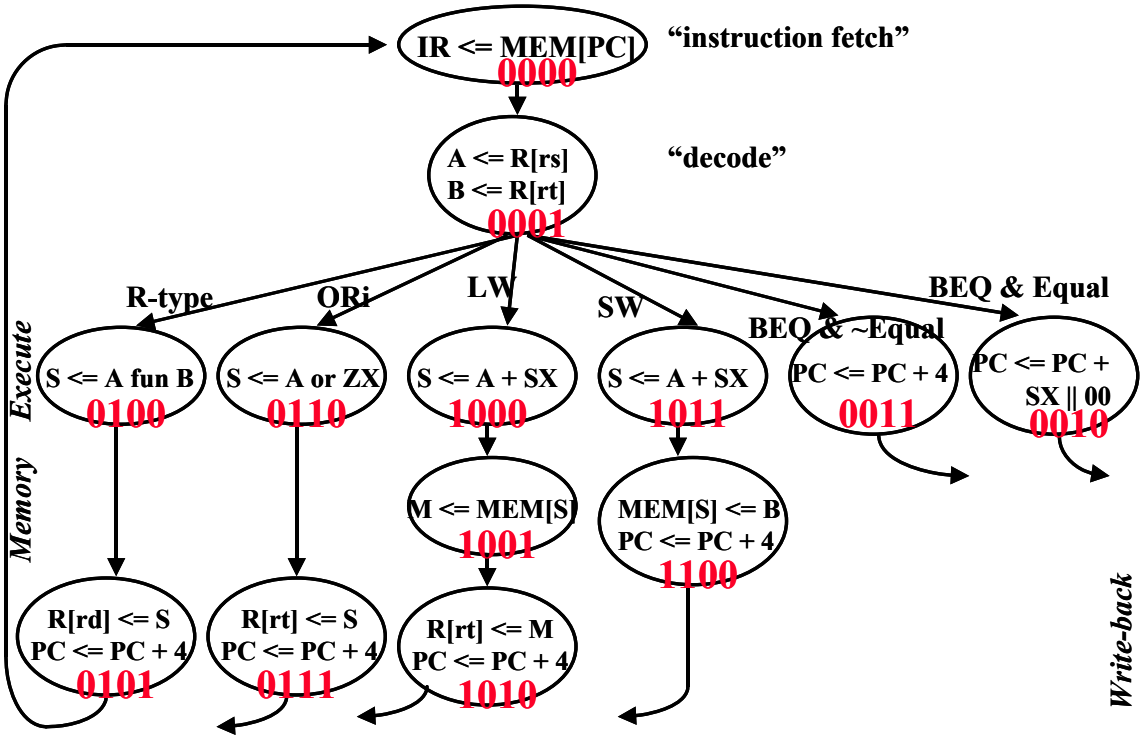
Control Specification



Mapping RTs to Control Points



State Assignments



Five Execution Steps

- Instruction Fetch
- Instruction Decode and Register Fetch
- Execution, Memory Address Computation, or Branch Completion
- Memory Access or R-type instruction completion
- Write-back step

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Performance Evaluation

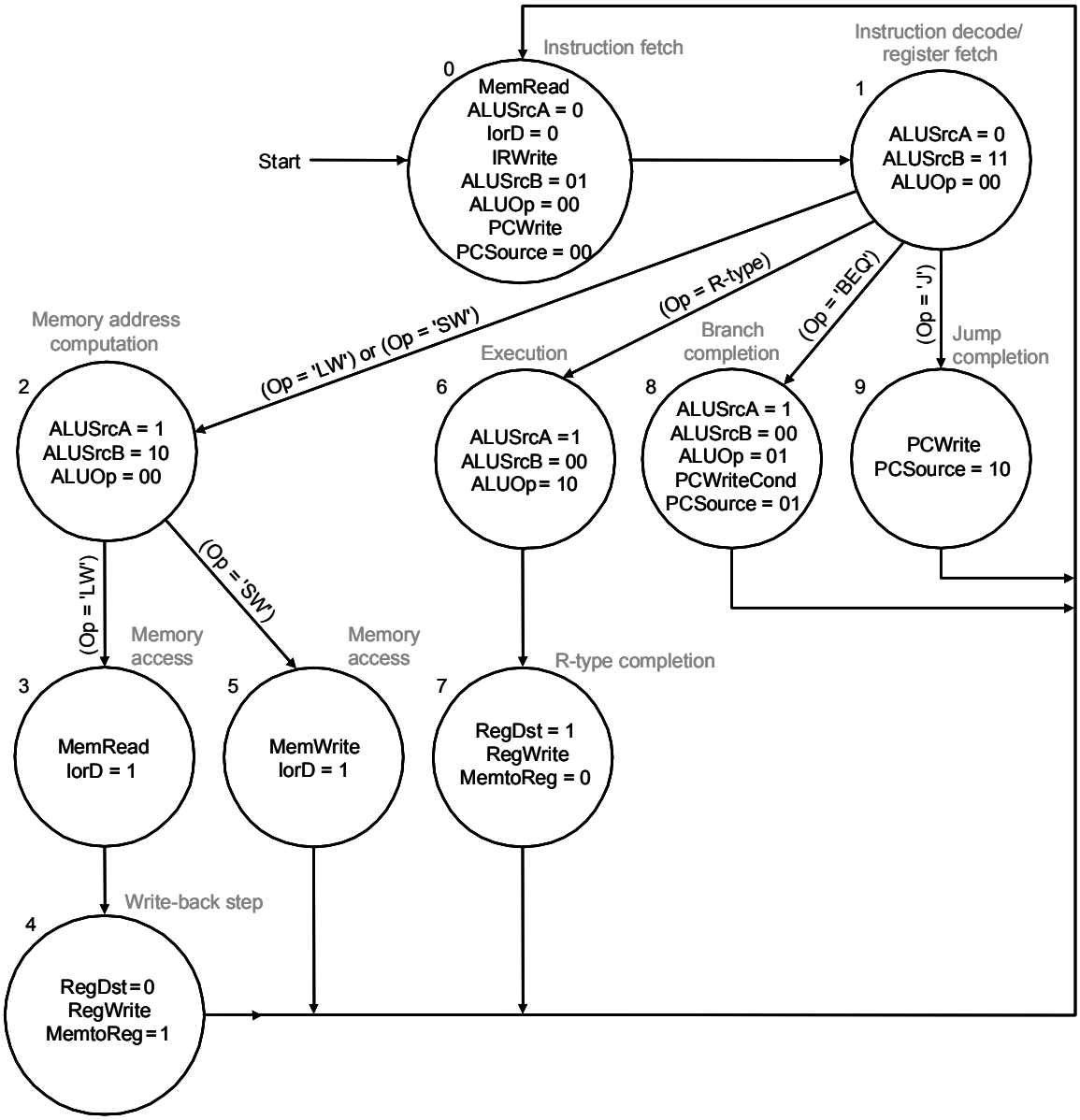
What is the average CPI?

State diagram provides CPI for each type of instruction.
The workload gives the frequency of each instruction type

Type	CPI_i for type	Frequency	CPI_i x $freqI_i$
Arith/Logic	4	40%	1.6
Load	5	20%	1.0
Store	4	20%	0.8
Branch	3	20%	0.6

Average CPI: 4.0

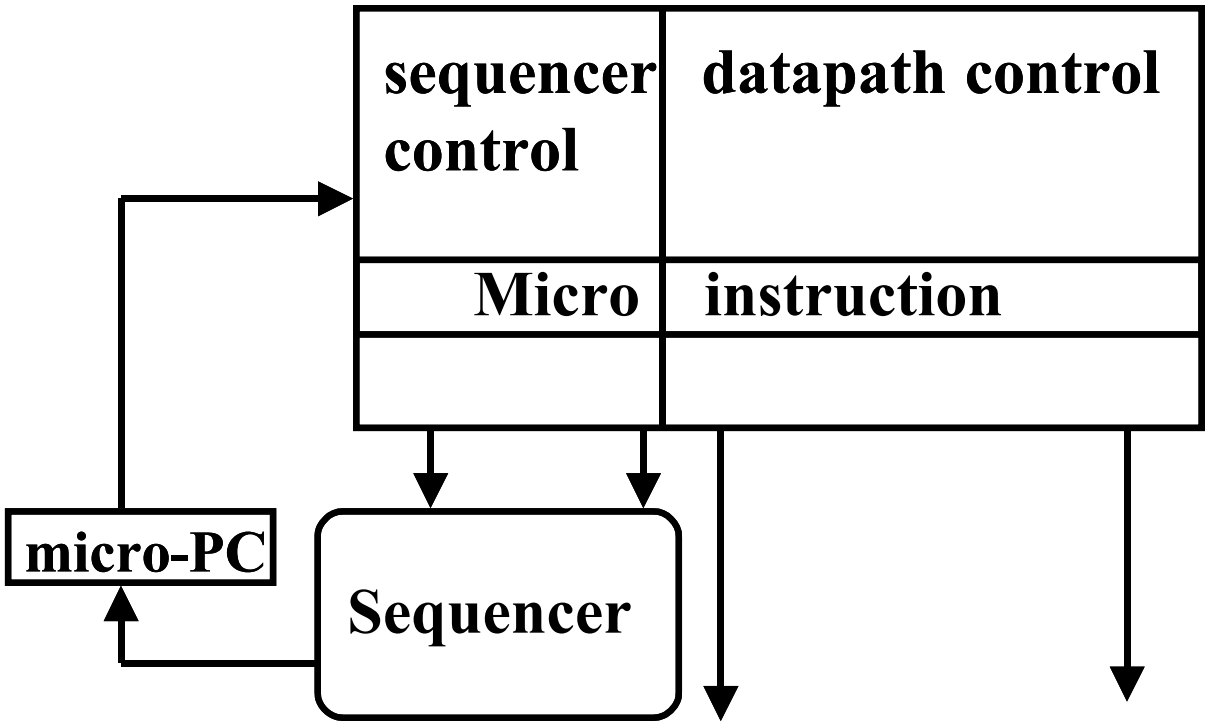
FSM for Datapath Control



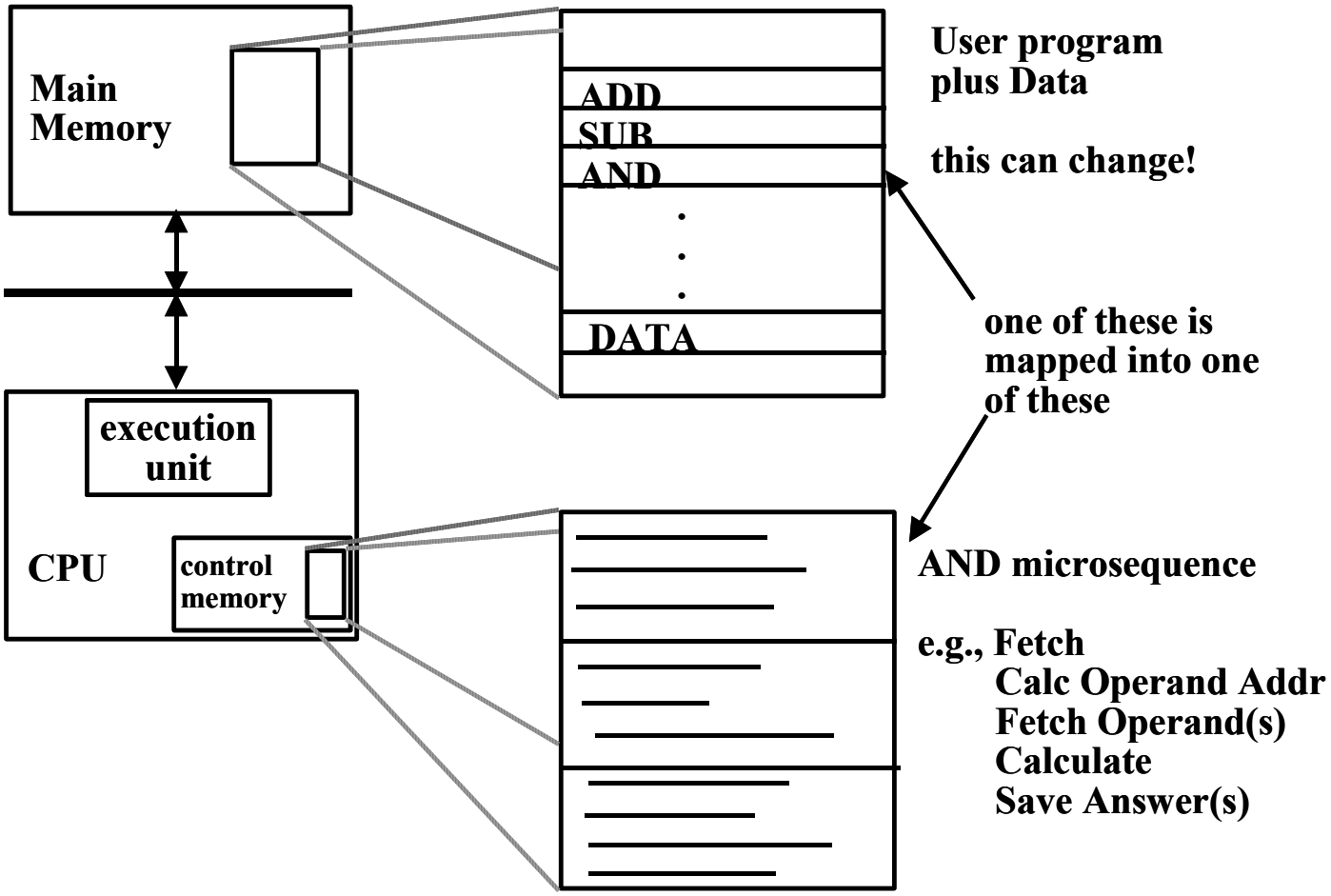
Microprogram Control Design

The state diagrams define the controller for an instruction set processor are highly structured.

- Use this structure to construct a simple "micro-sequencer"
- Control reduces to programming this very simple device using the concept of:
  - microprogramming



Processor Instruction Interpretation



Microprogramming

Horizontal vs. Vertical

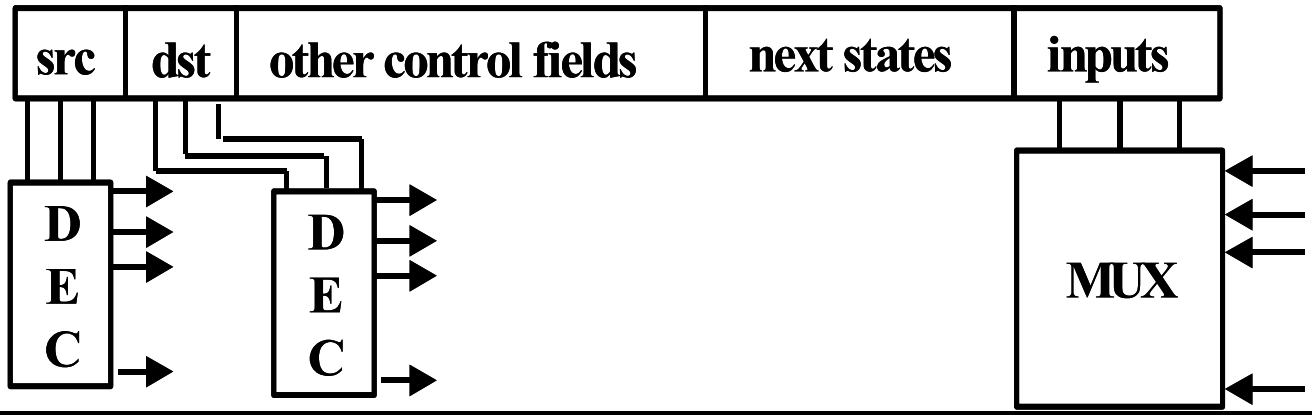
"Horizontal" Microcode

µseq	µaddr	A-mux	B-mux	bus enables	register enables	

- Control field for each control point in the machine.
- Lots of control states but provide more control over the parallelism in the datapath.

"Vertical" Microcode

- Compact microinstruction format for each class of µ-operation.
- Local decode to generate all control points.
- Extra level of decoding can slow down the machine.

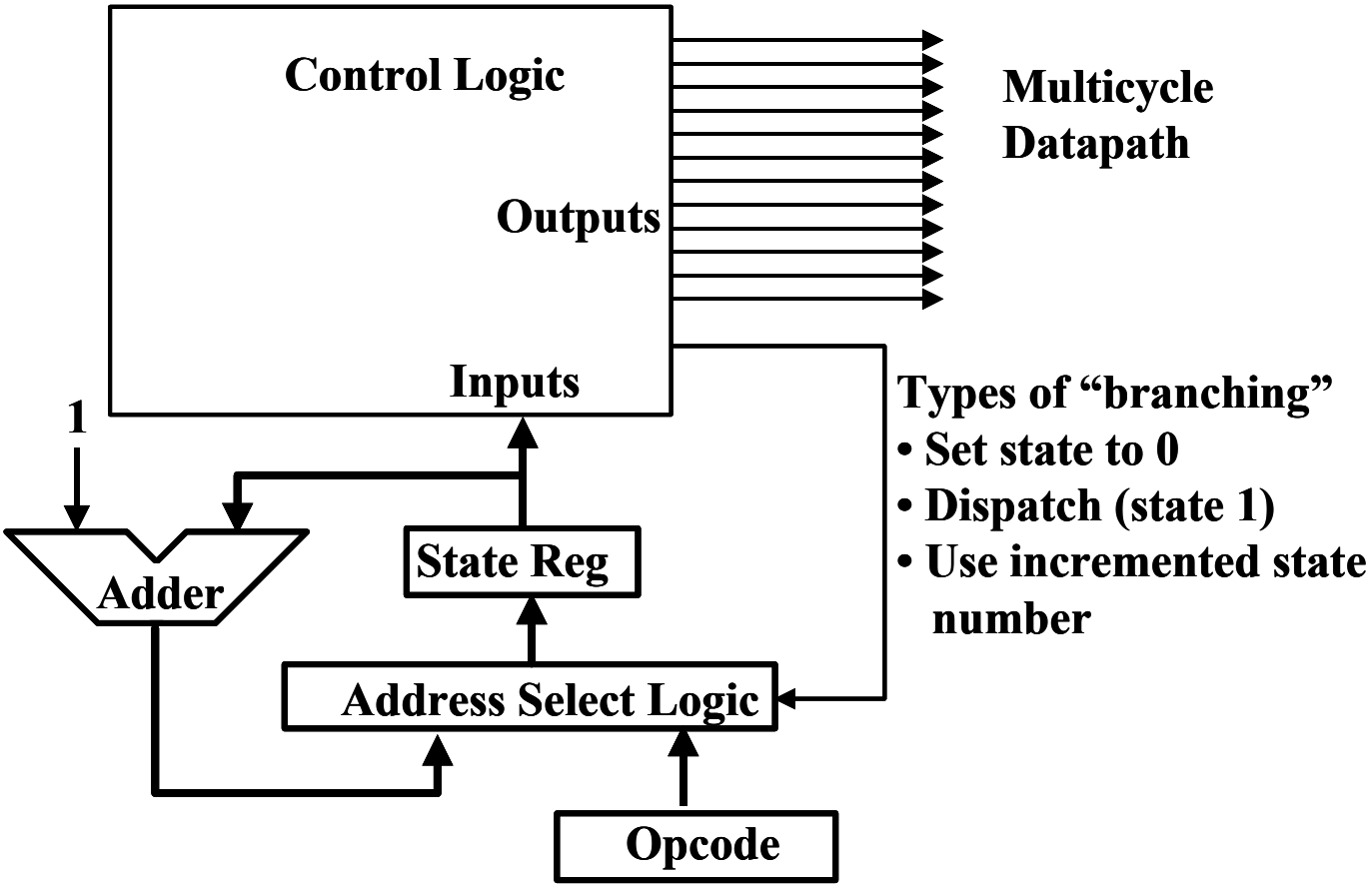


Microprogram-based Control

Control is the hard part of processor design

- Datapath is fairly regular and well-organized
- Memory is highly regular
- Control is irregular and global

Sequencer-based Control Unit



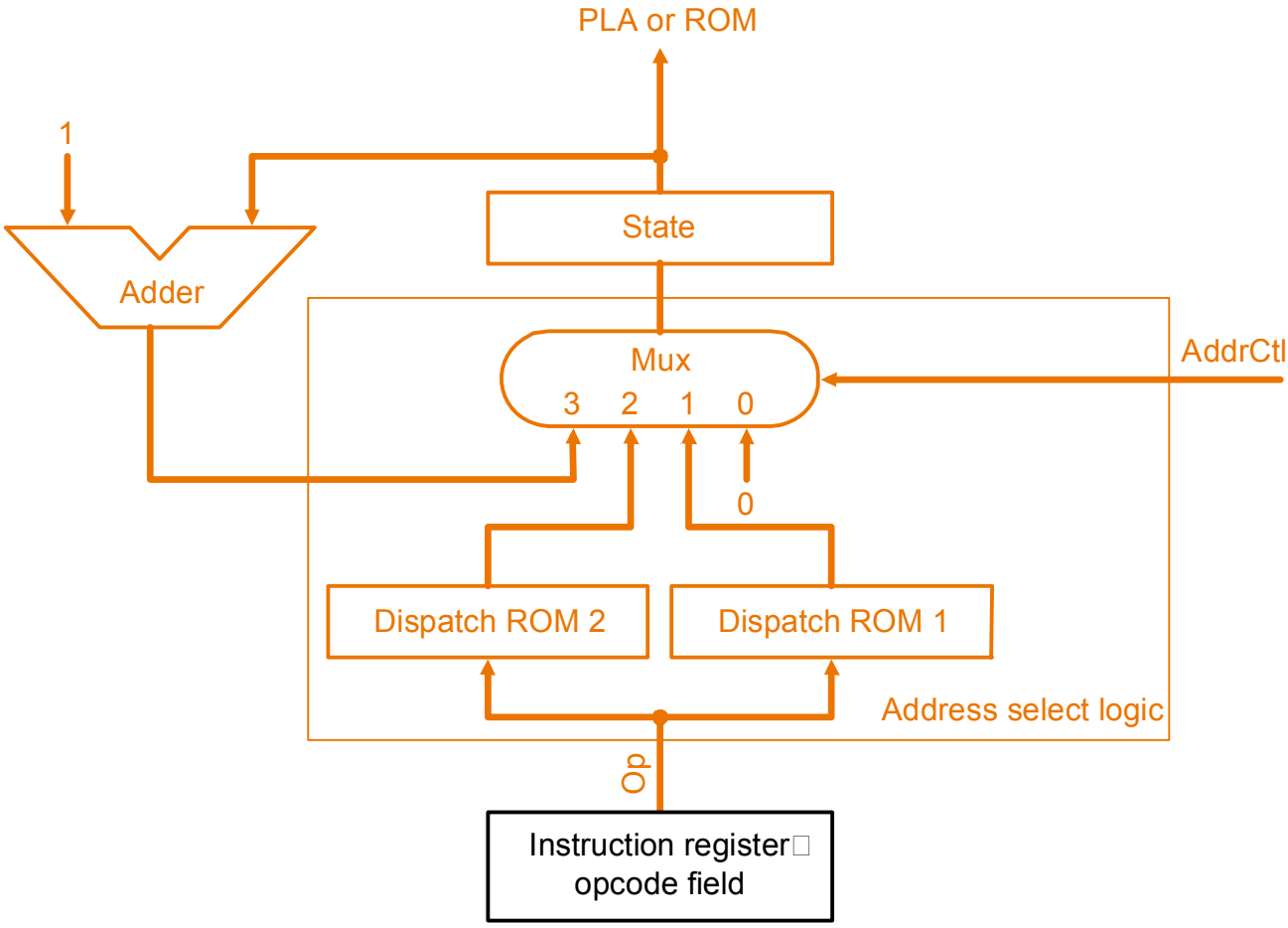
Designing a Microinstruction Set

- Start with list of control signals.
- Group signals together that make sense (vs. random): called "fields"
- Places fields in some logical order (e.g., ALU operation & ALU operands first and microinstruction sequencing last)
- Create a symbolic legend for μ-instruction format, showing name of field values and how they set the control signals
- Use computers to design computers. μ-assembler to get binary code.
- To minimize the width, encode operations that will never be used at the same time.

Microinstruction Fields and Control Signals

Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
SRC2	B	ALUSrcB = 00	Register B is the second ALU input.
	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshft	ALUSrcB = 11	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read		Read two registers using the rs and rt fields of the IR as the register numbers and putting the data into registers A and B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	MemRead, lorD = 0	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	MemRead, lorD = 1	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	MemWrite, lorD = 1	Write memory using the ALUOut as address, contents of B as the data.
PC write control	ALU	PCSource = 00 PCWrite	Write the output of the ALU into the PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	jump address	PCSource = 10, PCWrite	Write the PC with the jump address from the instruction.
Sequencing	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

Sequencer-based Control Unit



Signal	*Mux-select*	.
Sequencing	00	Next μaddress = 0
	01	Next μ-address = dispatch ROM-1
	10	Next μ-address = dispatch ROM-2
	11	Next μ-address = μ-address + 1

Dispatch ROM 1		
Op	**Opcode name**	**Value**
000000	R-format	0110
000010	`jmp`	1001
000100	`beq`	1000
100011	`lw`	0010
101011	`sw`	0010

Dispatch ROM 2		
Op	**Opcode name**	**Value**
100011	`lw`	0011
101011	`sw`	0101

Microprogram Details

Microprogram: Signals specified symbolically using microinstructions.

State number	Address-control action	Value of AddrCtl
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Microprogram Details

Micro Instructions for Fetch and Decode

Label	ALU Control	SRC1	SRC2	Reg. Control	Mem	PCWrite Control	Sequence
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Ext shift	Read			Dispatch-1

Mem: Fetch Instruction into IR
ALU, SRC1, SRC2: Compute PC + 4
PCWrite Control: Write the output of ALU to PC
Sequencing: go to the next microinstruction.

Next Microinstruction is Decode
ALU, SRC1, SRC2: PC + signext(IR[0:15]) << 2
Register Control: Read registers into A and B.
Sequencing: Use dispatch-1 to select one of the following four labels

- `Mem1`
- `Rformat1`
- `BEQ1`
- `JUMP1`