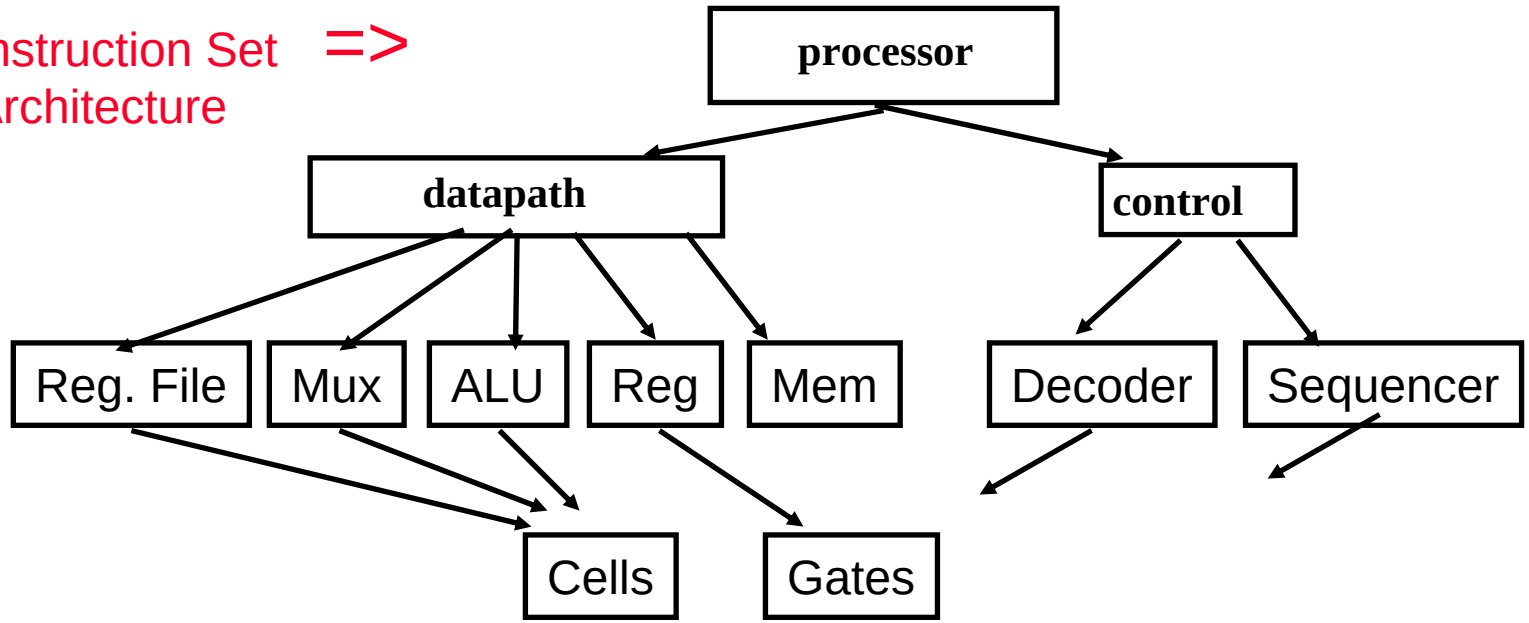
Computer Architecture
Designing a Multicycle Processor

Recap: Processor Design is a Process

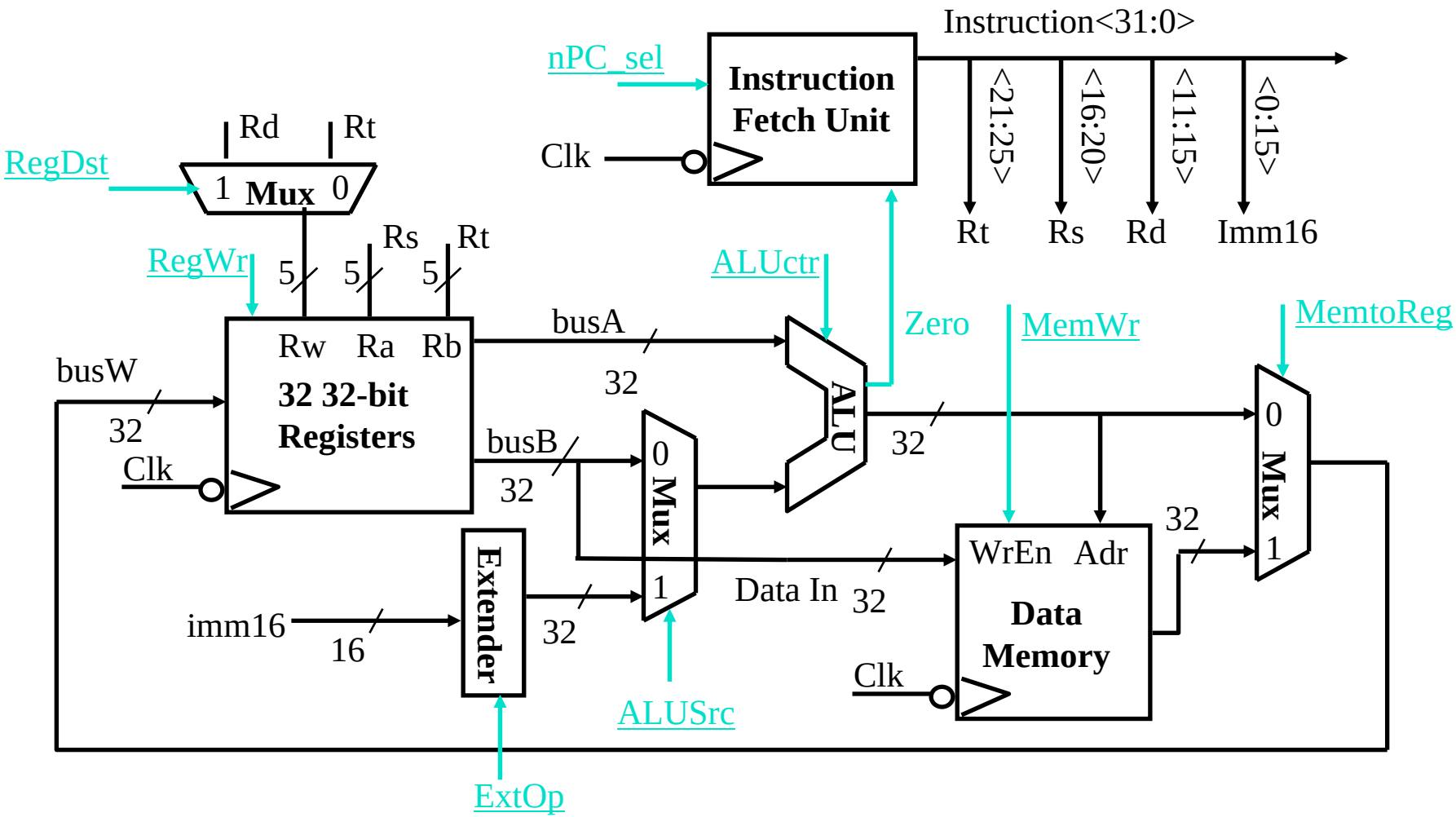
- **Bottom-up**
  - **assemble components in target technology to establish critical timing**
- **Top-down**
  - **specify component behavior from high-level requirements**
- **Iterative refinement**
  - **establish partial solution, expand and improve**

Instruction Set Architecture =>

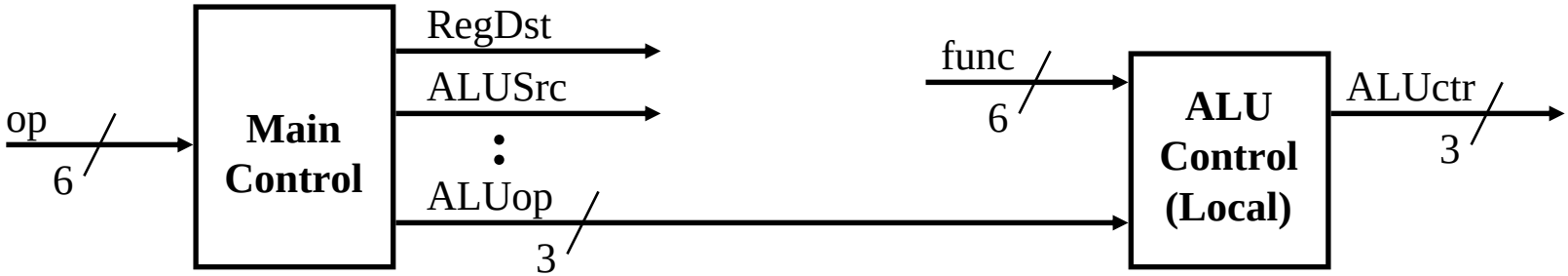


Recap: A Single Cycle Datapath

- We have everything except control signals (underline)
  - Today's lecture will show you how to generate the control signals

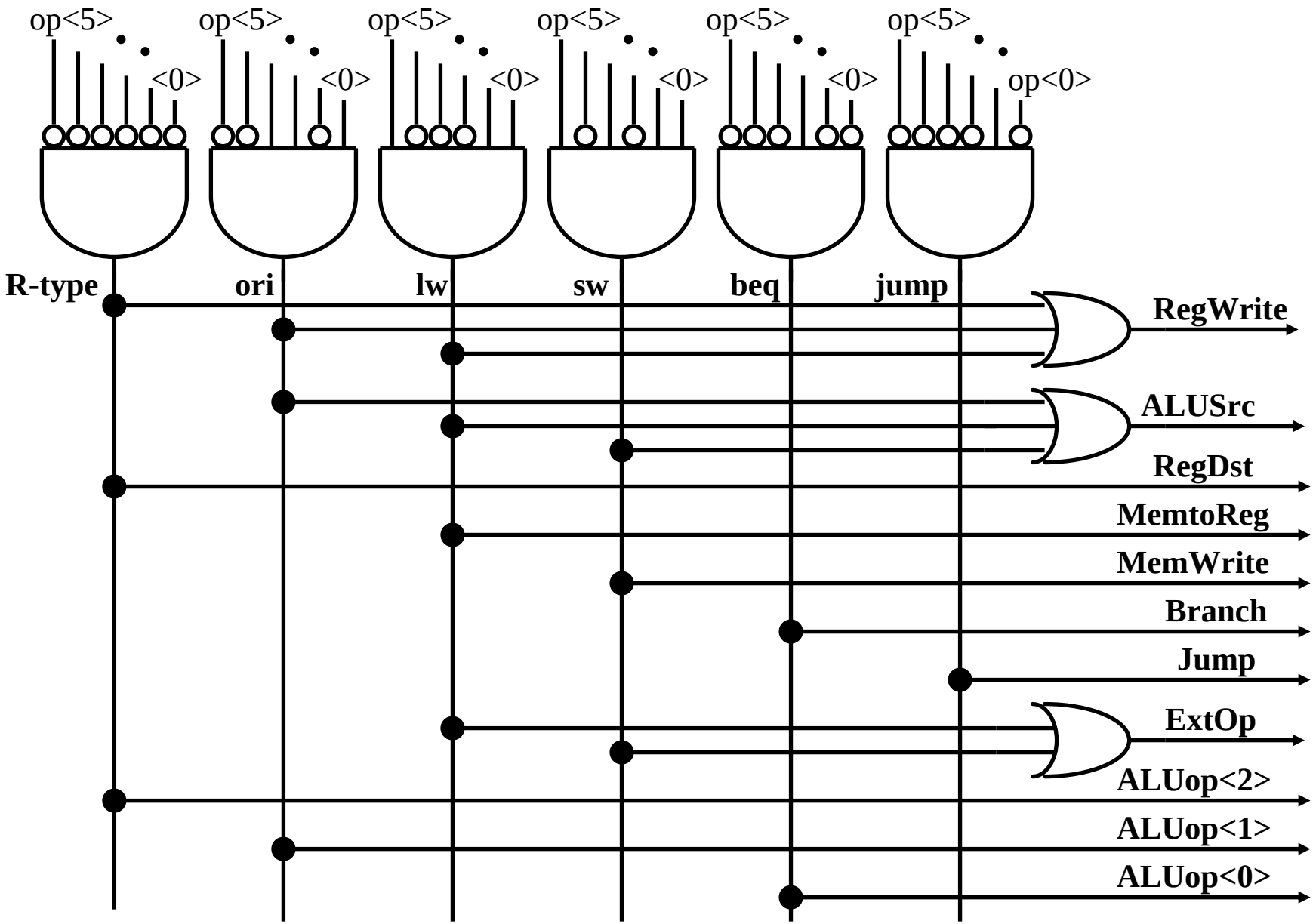


Recap: The "Truth Table" for the Main Control

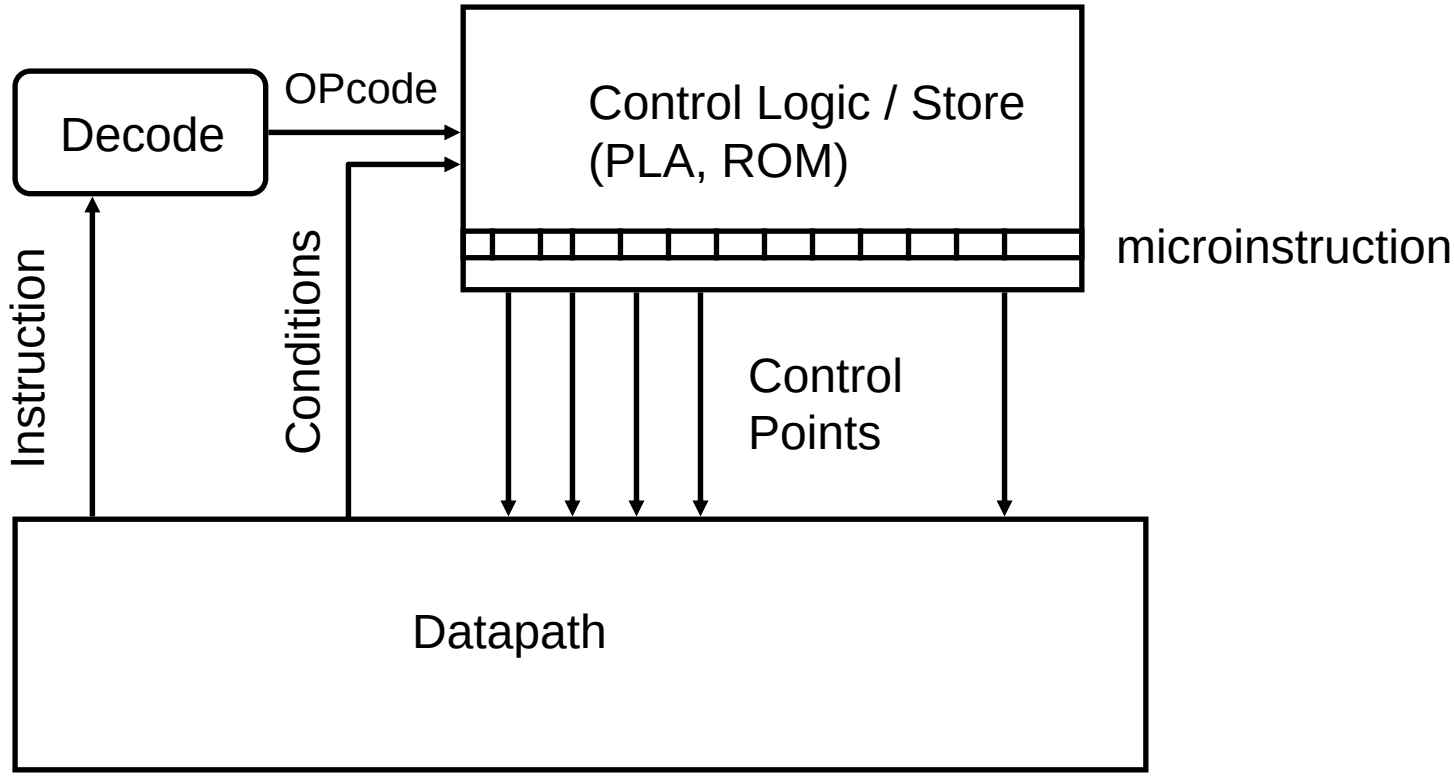


op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop (Symbolic)	"R-type"	Or	Add	Add	Subtract	xxx
ALUop <2>	1	0	0	0	0	x
ALUop <1>	0	1	0	0	0	x
ALUop <0>	0	0	0	0	1	x

Recap: PLA Implementation of the Main Control



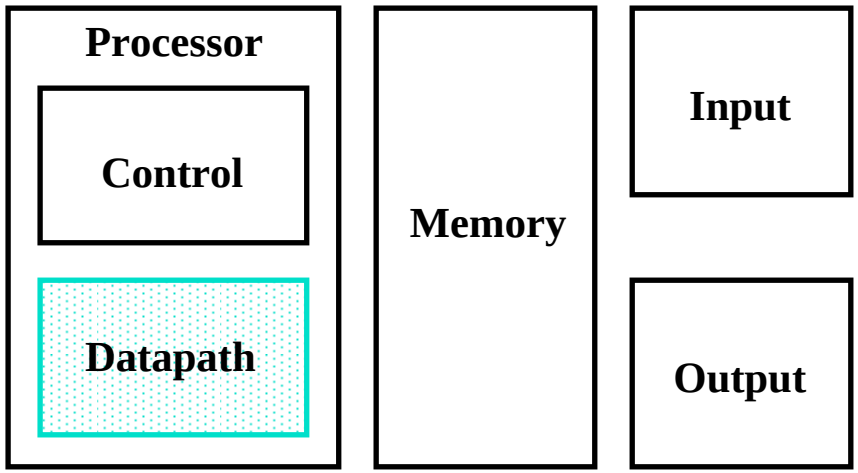
Recap: Systematic Generation of Control



- **In our single-cycle processor, each instruction is realized by exactly one control command or "*microinstruction*"**
  - **in general, the controller is a finite state machine**
  - **microinstruction can also control sequencing (see later)**

The Big Picture: Where are We Now?

- The Five Classic Components of a Computer



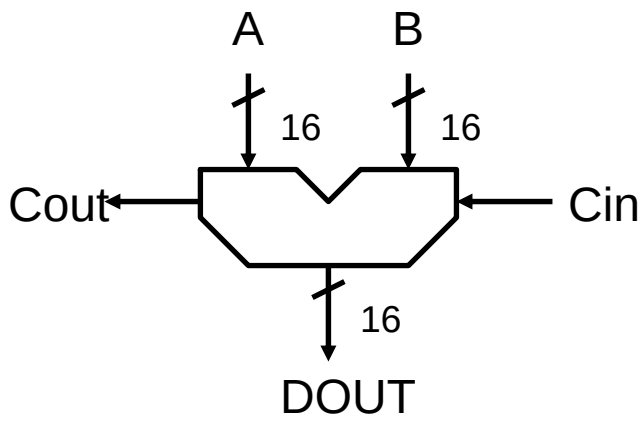
- Today's Topic: Designing the Datapath for the Multiple Clock Cycle Datapath

Outline of Today's Lecture

- Recap: single cycle processor
- VHDL versions
- Faster designs
- Multicycle Datapath
- Performance Analysis
- Multicycle Control

Behavioral models of Datapath Components

```
entity adder16 is
generic (ccOut_delay : TIME := 12 ns;
        adderOut_delay:  TIME := 12 ns);
port(A, B:      in  vlbit_1d(15  downto 0);
    DOUT:      out vlbit_1d(15 downto 0);
    CIN:    in vlbit;
    COUT:      out vlbit);
end adder16;
```




```
architecture behavior of adder32 is
begin
        adder16_process: process(A, B, CIN)

        variable tmp    : vlbit_1d(18 downto 0);
        variable adder_out : vlbit_1d(31 downto 0);
        variable carry: vlbit;

        begin
            tmp := addum (addum (A, B), CIN);
            adder_out := tmp(15 downto 0);
            carry :=tmp(16);

            COUT <= carry after ccOut_delay;
            DOUT <= adder_out after adderOut_delay;
        end process;
end behavior;
```

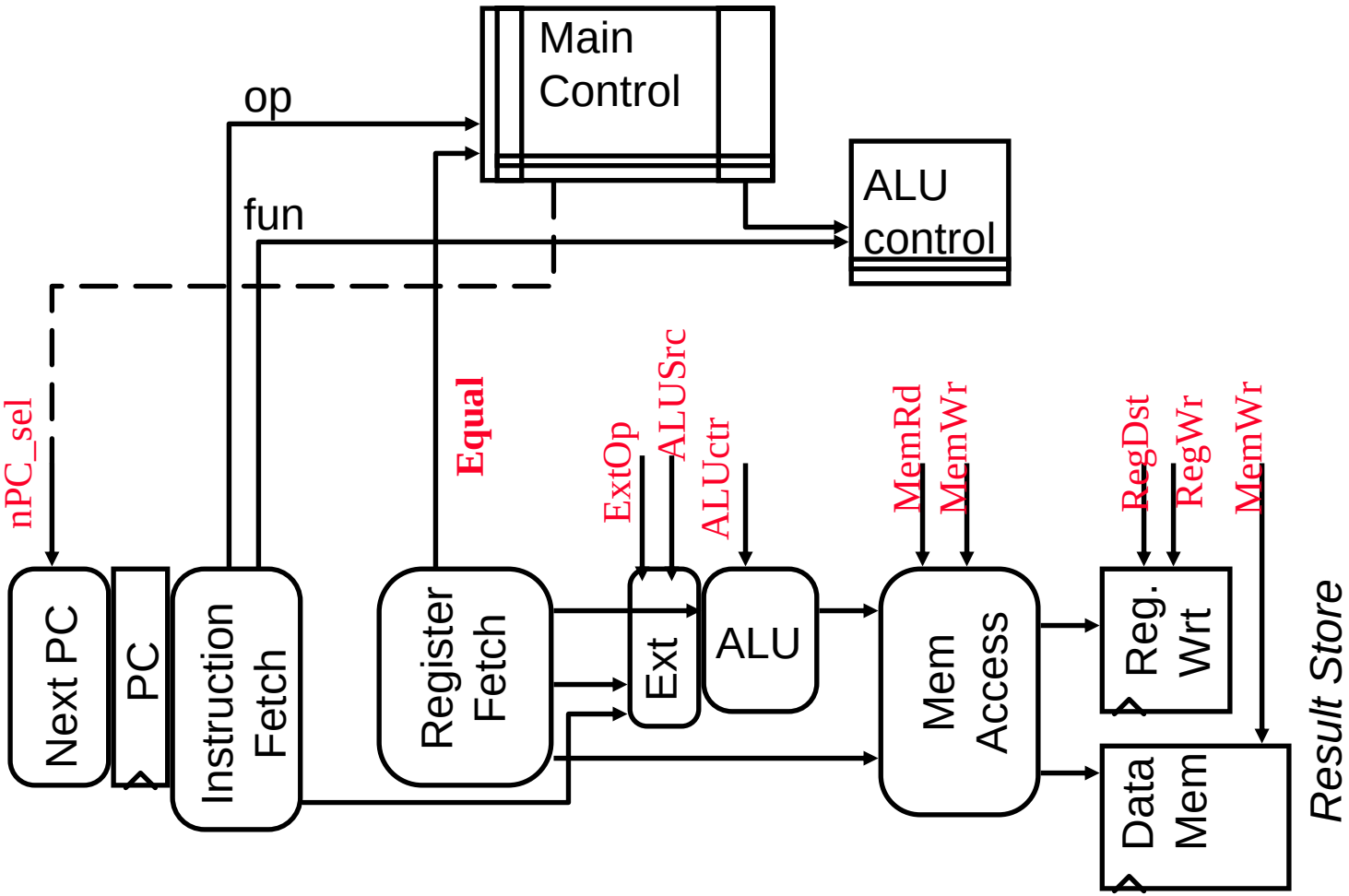
Behavioral Specification of Control Logic

```
entity maincontrol  is
port(opcode:        in  vlbit_1d(5 downto 0);
     equal_cond:    in vlbit;

      extop         out vlbit;
      ALUsrc        out vlbit;
      ALUop         out vlbit_1d(1 downto 0);
      MEMwr         out vlbit;
      MemtoReg      out vlbit;
      RegWr         out vlbit;
      RegDst        out vlbit;
      nPC           out vlbit;
end maincontrol;
```

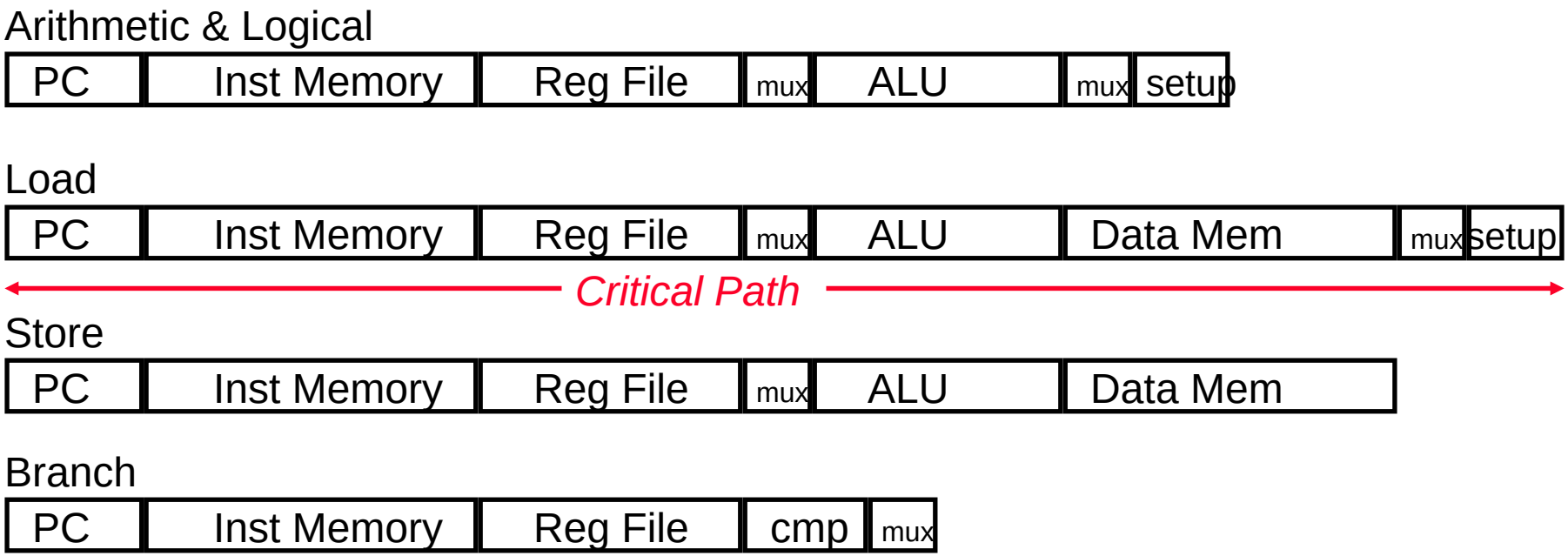
- Decode / Control-store address modeled by Case statement
- Each arm drives control signals for that operation
  - just like the microinstruction
  - either can be symbolic

Abstract View of our single cycle processor



- **looks like a FSM with PC as state**

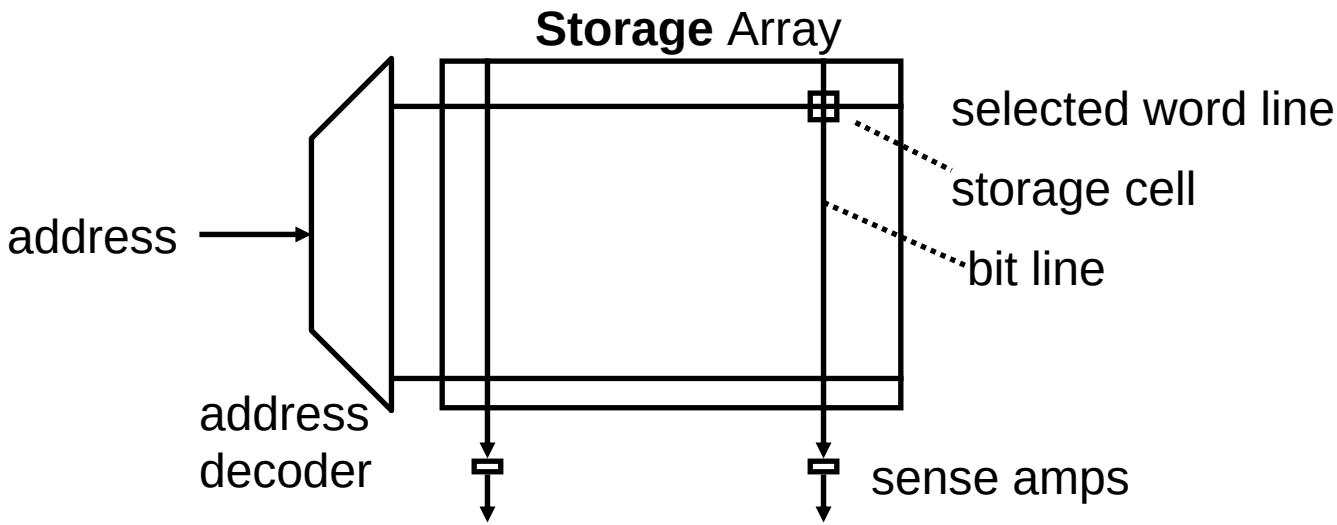
What's wrong with our CPI=1 processor?



- **Long Cycle Time**
- **All instructions take as much time as the slowest**
- **Real memory is not so nice as our idealized memory**
  - **cannot always get the job done in one (short) cycle**

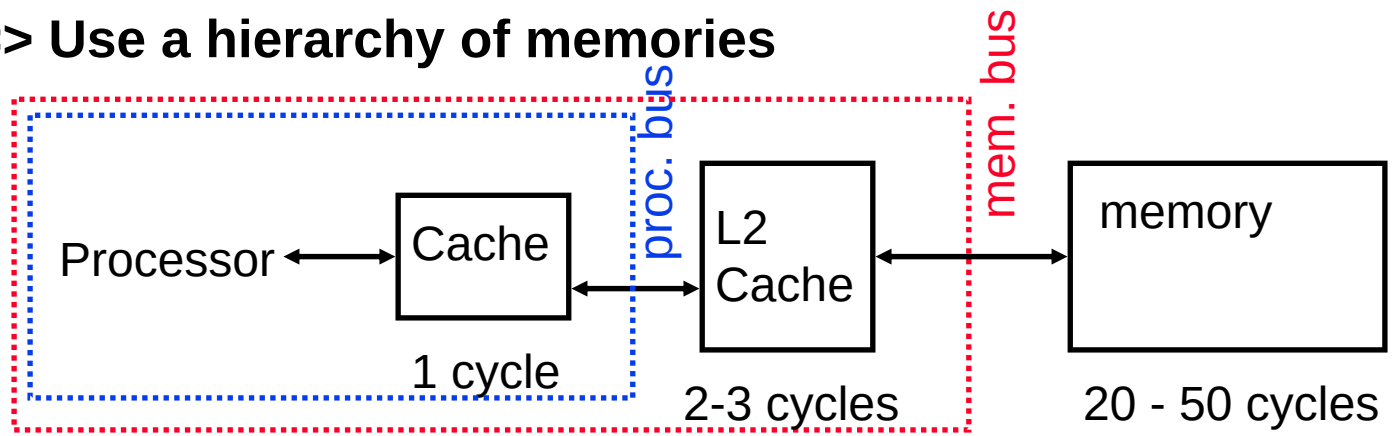
Memory Access Time

- **Physics => fast memories are small (large memories are slow)**



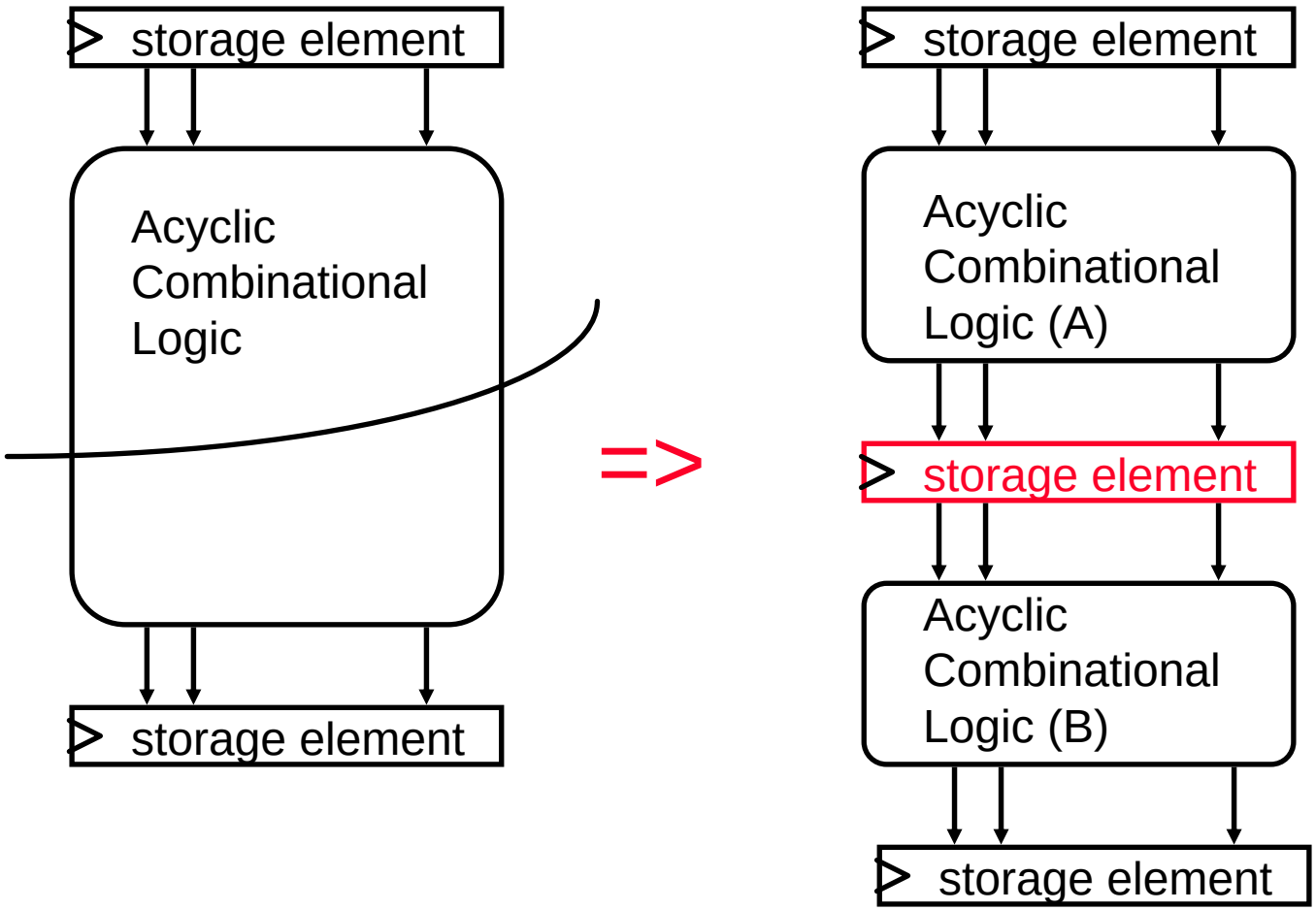
  - **question: register file vs. memory**

- **=> Use a hierarchy of memories**



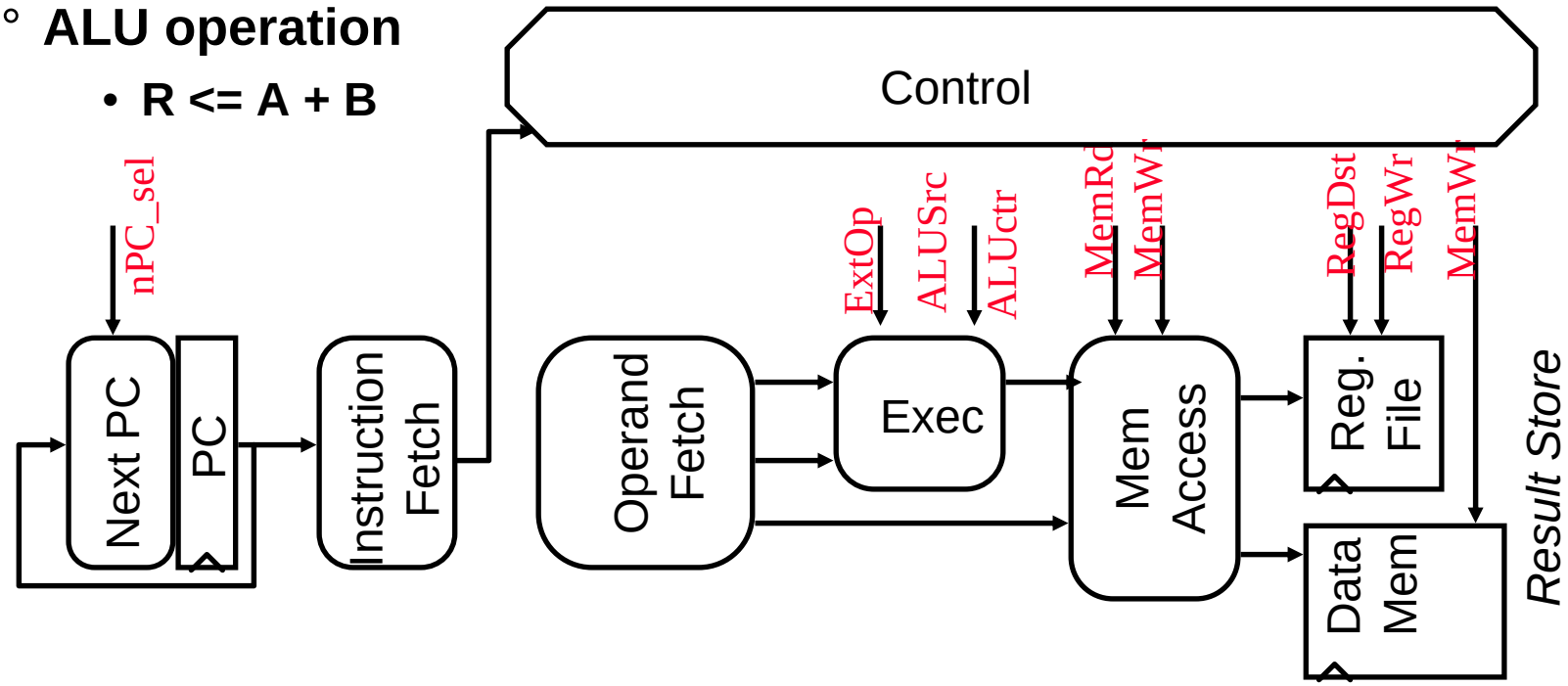
Reducing Cycle Time

- **Cut combinational dependency graph and insert register / latch**
- **Do same work in two fast cycles, rather than one slow one**



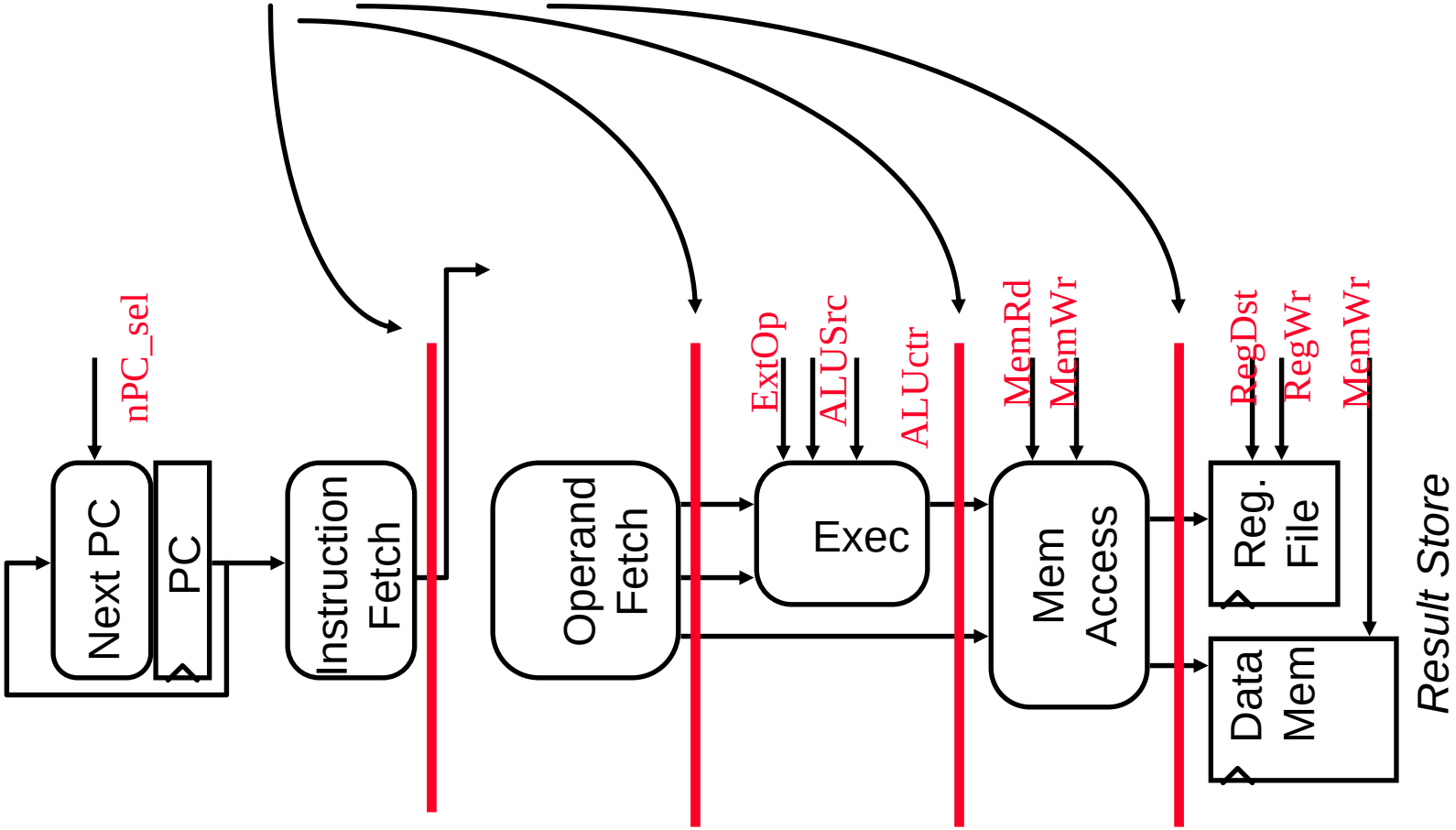
Basic Limits on Cycle Time

- **Next address logic**
  - **PC <= branch ? PC + offset : PC + 4**
- **Instruction Fetch**
  - **InstructionReg <= Mem[PC]**
- **Register Access**
  - **A <= R[rs]**
- **ALU operation**
  - **R <= A + B**

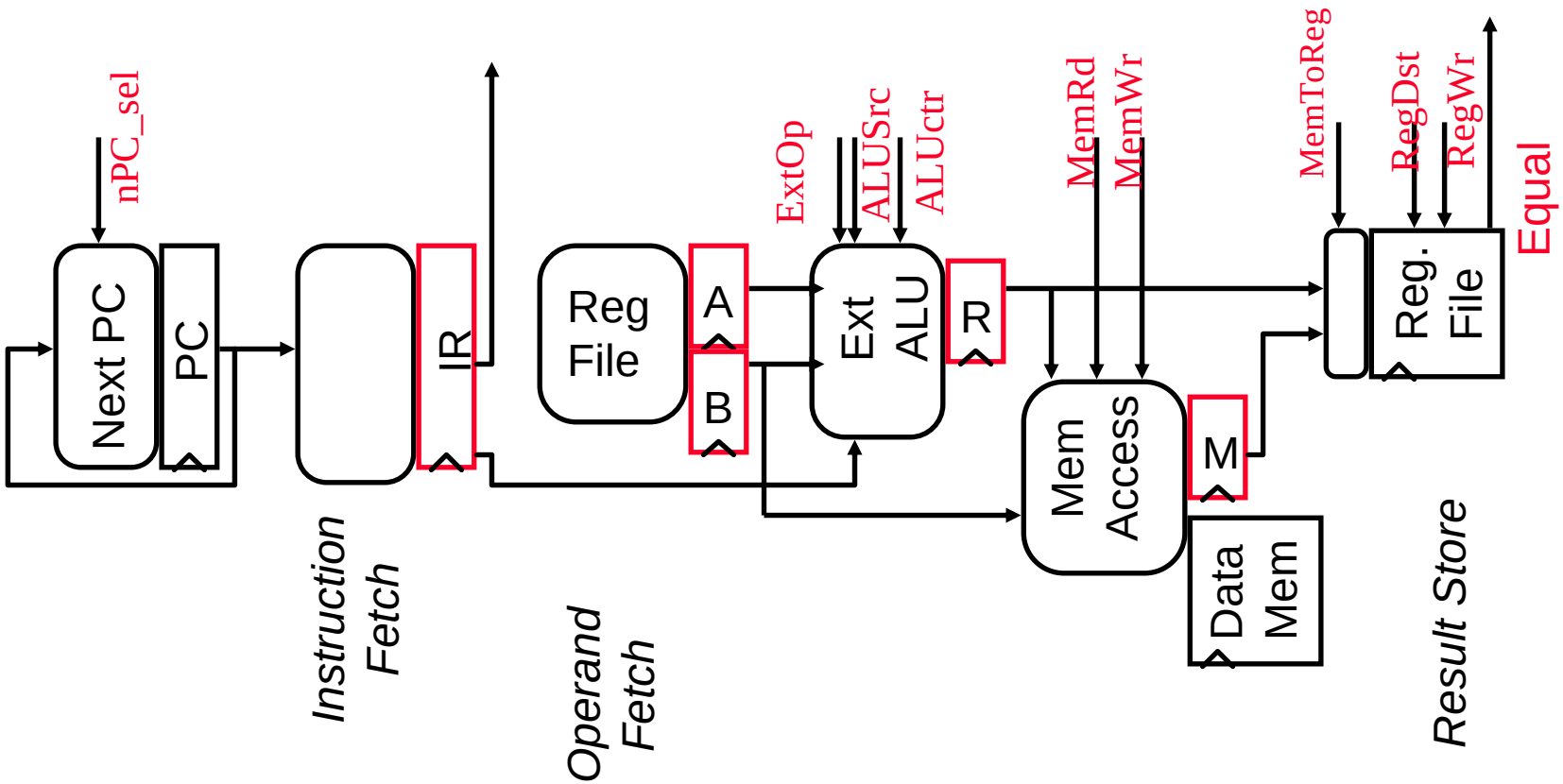


Partitioning the CPI=1 Datapath

- Add registers between smallest steps



Example Multicycle Datapath



° **Critical Path ?**

Recall: Step-by-step Processor Design

Step 1: ISA => Logical Register Transfers

Step 2: Components of the Datapath

Step 3: RTL + Components => Datapath

Step 4: Datapath + Logical RTs => Physical RTs

Step 5: Physical RTs => Control

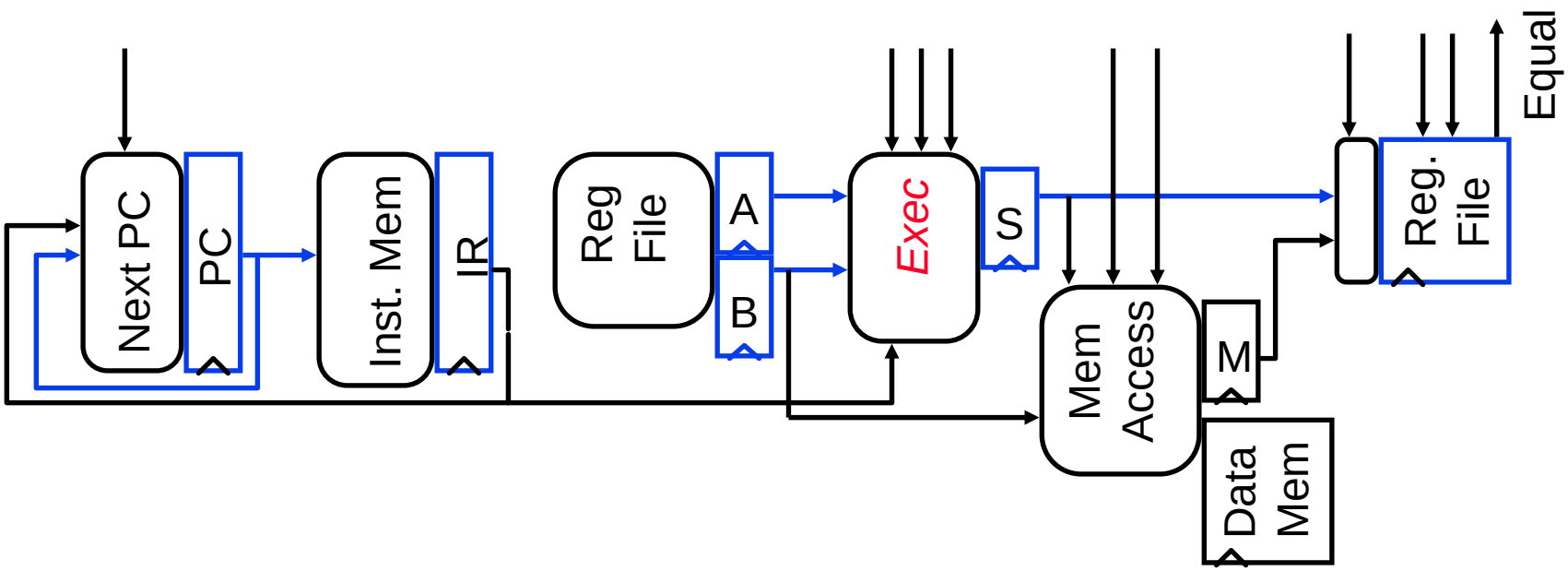
Step 4: R-rtype (add, sub, . . .)

- **Logical Register Transfer**

inst	Logical Register Transfers
ADDU	R[rd] <– R[rs] + R[rt]; PC <– PC + 4

- **Physical Register Transfers**

inst	Physical Register Transfers
	IR <– MEM[pc]
ADDU	A<– R[rs]; B <– R[rt]
	S <– A + B
	R[rd] <– S; PC <– PC + 4



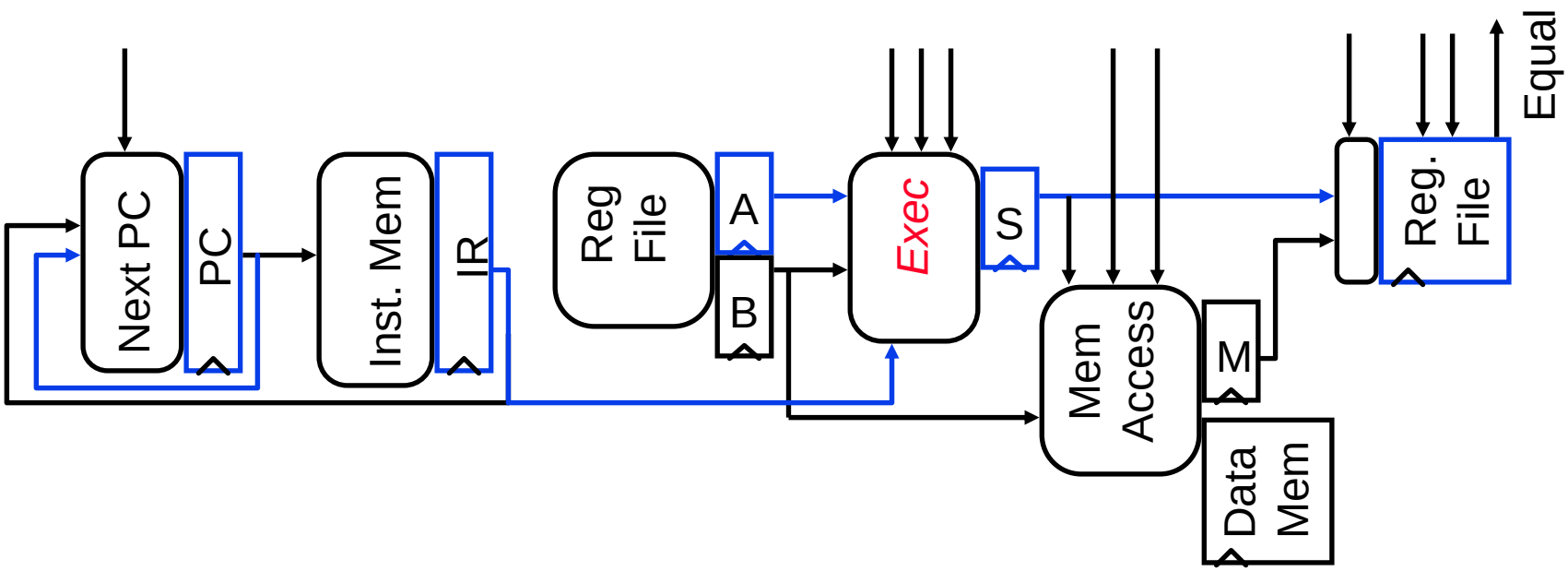
Step 4:Logical immed

- **Logical Register Transfer**

inst	Logical Register Transfers
ADDU	R[rt] <– R[rs] OR zx(Im16); PC <– PC + 4

- **Physical Register Transfers**

inst	Physical Register Transfers
	IR <– MEM[pc]
ADDU	A<– R[rs]; B <– R[rt]
	S <– A *or* ZeroExt(Im16)
	R[rt] <– S; PC <– PC + 4



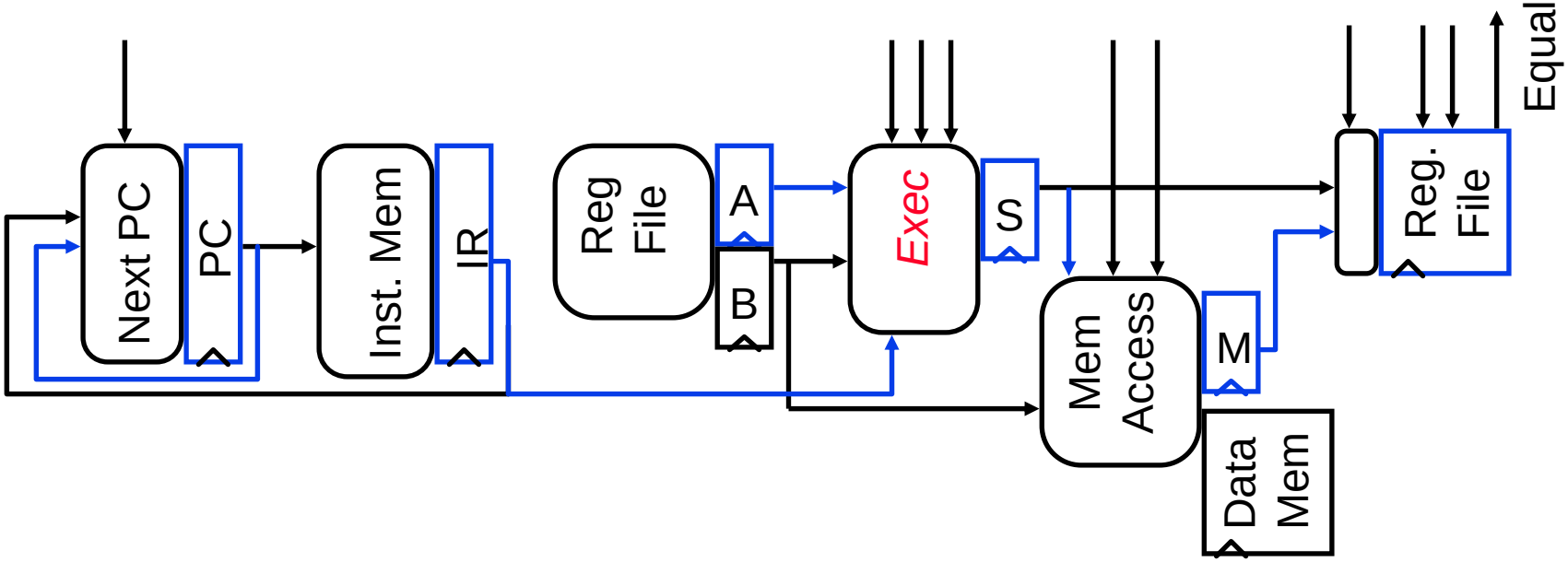
Step 4 : Load

- Logical Register Transfer

inst	Logical Register Transfers
LW	R[rt] <– MEM(R[rs] + sx(Im16);
	PC <– PC + 4

- Physical Register Transfers

inst	Physical Register Transfers
	IR <– MEM[pc]
LW	A<– R[rs]; B <– R[rt]
	S <– A + SignEx(Im16)
	M <– MEM[S]
	R[rd] <– M; PC <– PC + 4



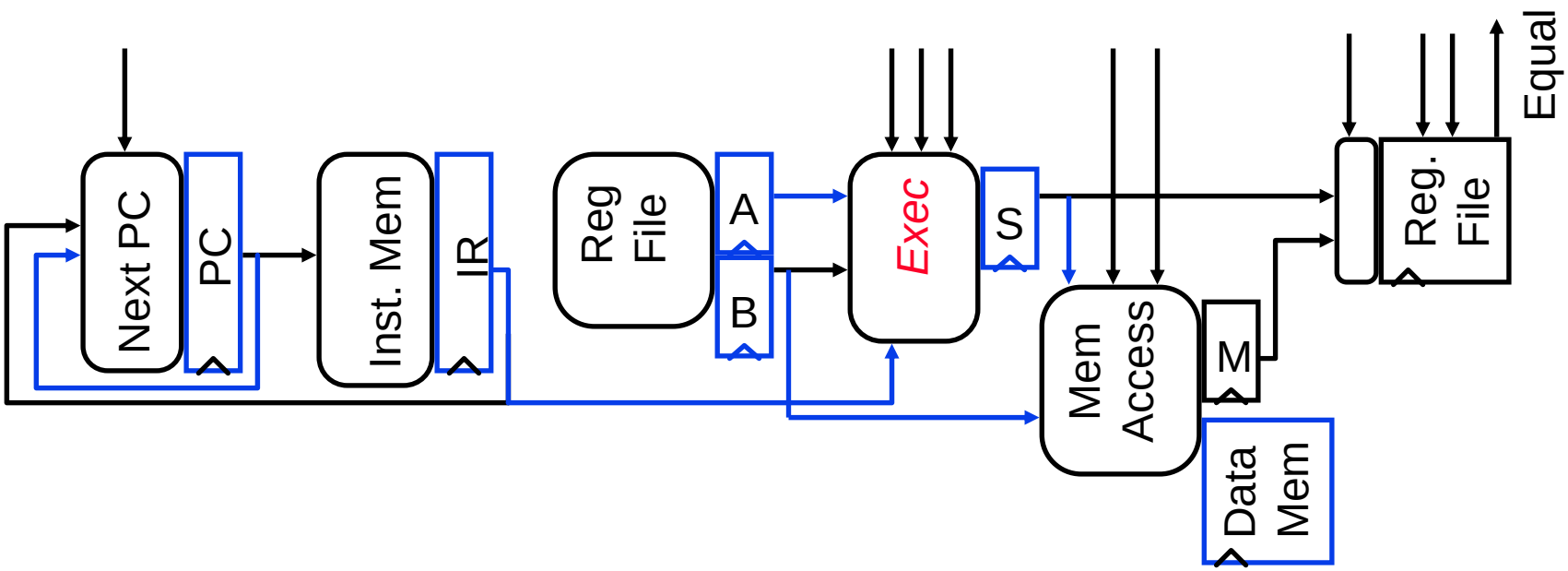
Step 4 : Store

- **Logical Register Transfer**

inst	Logical Register Transfers
SW	MEM(R[rs] + sx(Im16) <– R[rt];
	PC <– PC + 4

- **Physical Register Transfers**

inst	Physical Register Transfers	
	IR <– MEM[pc]	
SW	A<– R[rs]; B <– R[rt]	
	S <– A + SignEx(Im16);	
	MEM[S] <– B	PC <– PC + 4



Step 4 : Branch

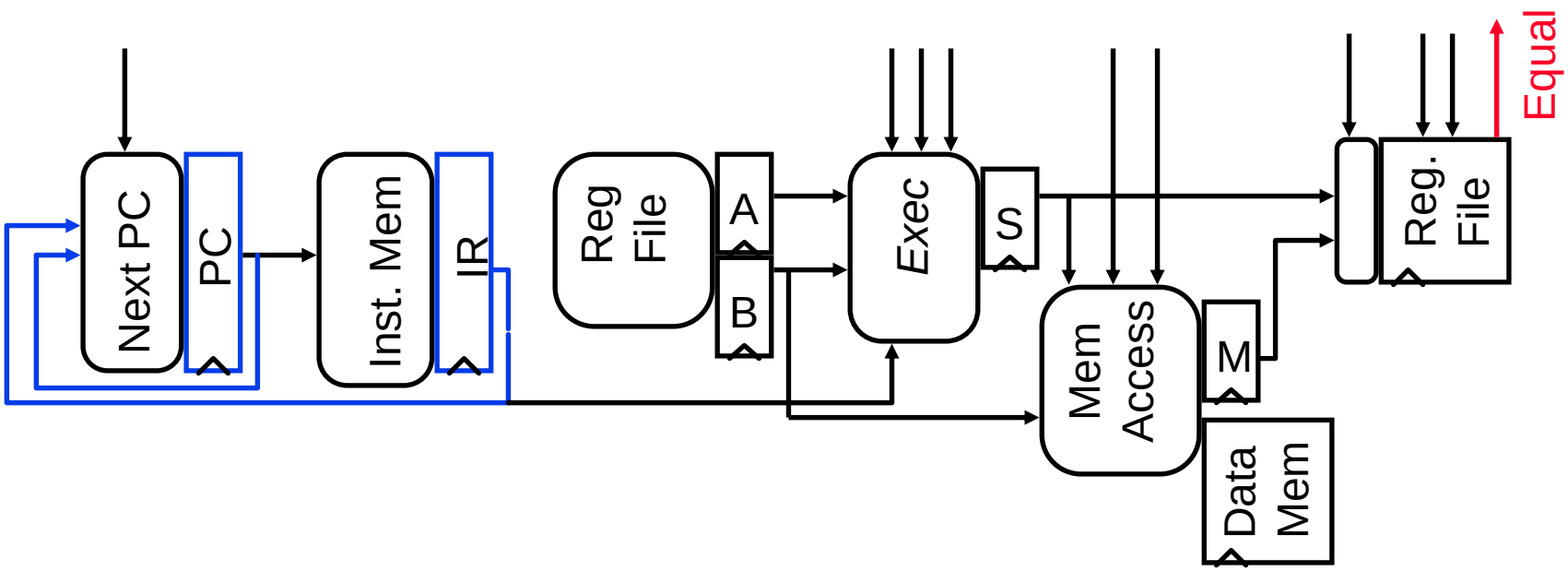
- **Logical Register Transfer**

inst	Logical Register Transfers
BEQ	if R[rs] == R[rt]
	then PC <= PC + sx(Im16) || 00
	else PC <= PC + 4

- **Physical Register Transfers**

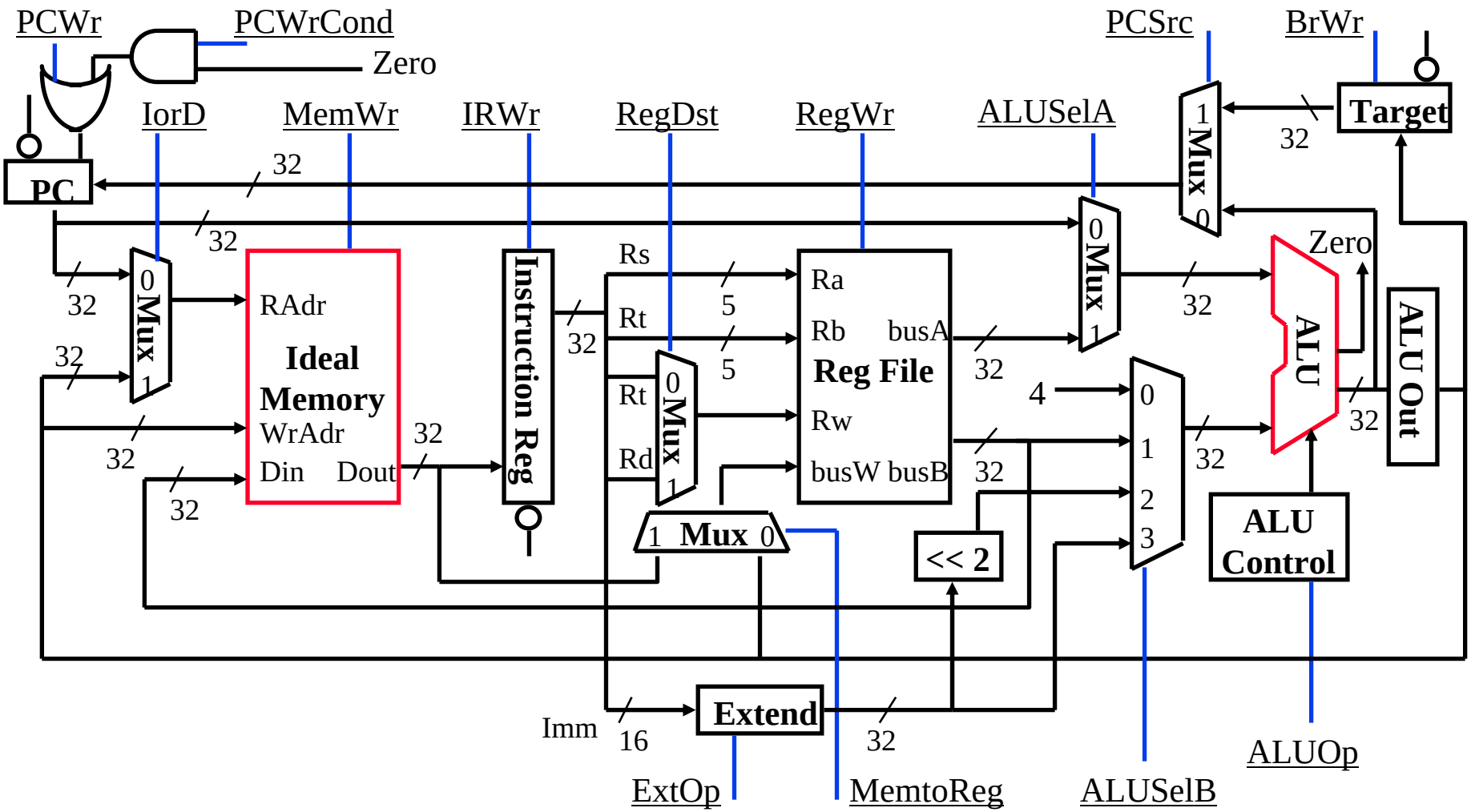
inst	Physical Register Transfers
	IR <– MEM[pc]
BEQ|$\bar{E}$q	PC <– PC + 4

inst	Physical Register Transfers
	IR <– MEM[pc]
BEQ|Eq	PC <– PC + sx(Im16) || 00



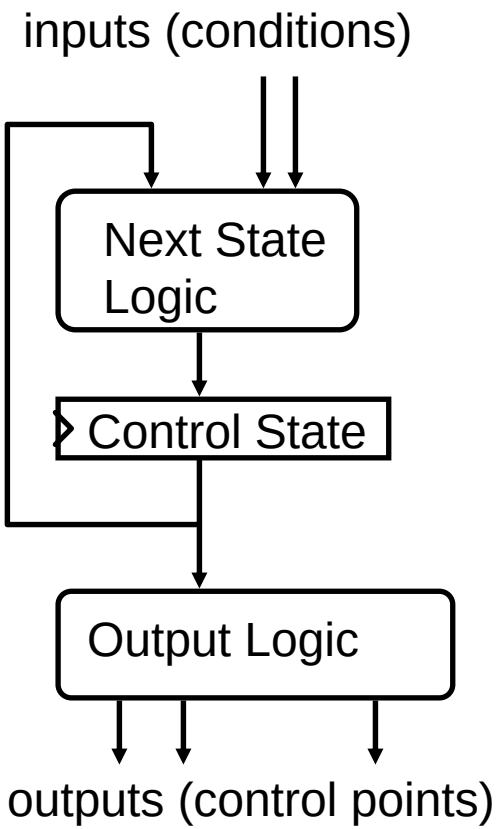
Alternative datapath (book): Multiple Cycle Datapath

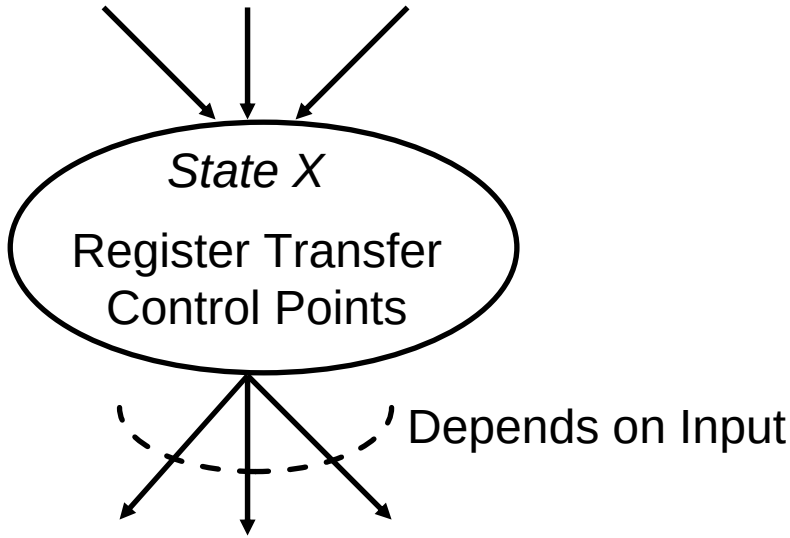
- **Miminizes Hardware: 1 memory, 1 adder**



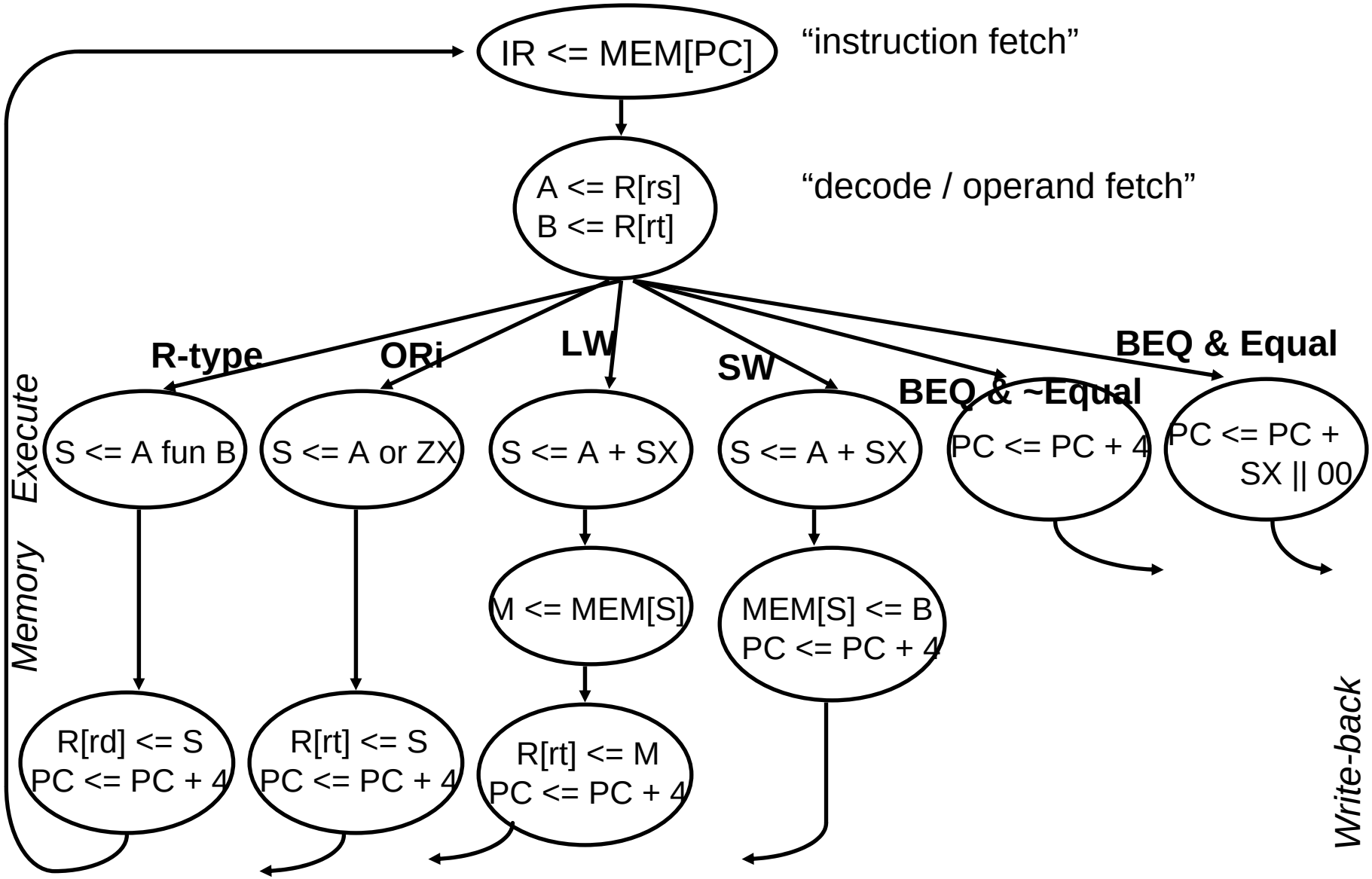
Our Control Model

- **State specifies control points for Register Transfer**
- **Transfer occurs upon exiting state (same falling edge)**

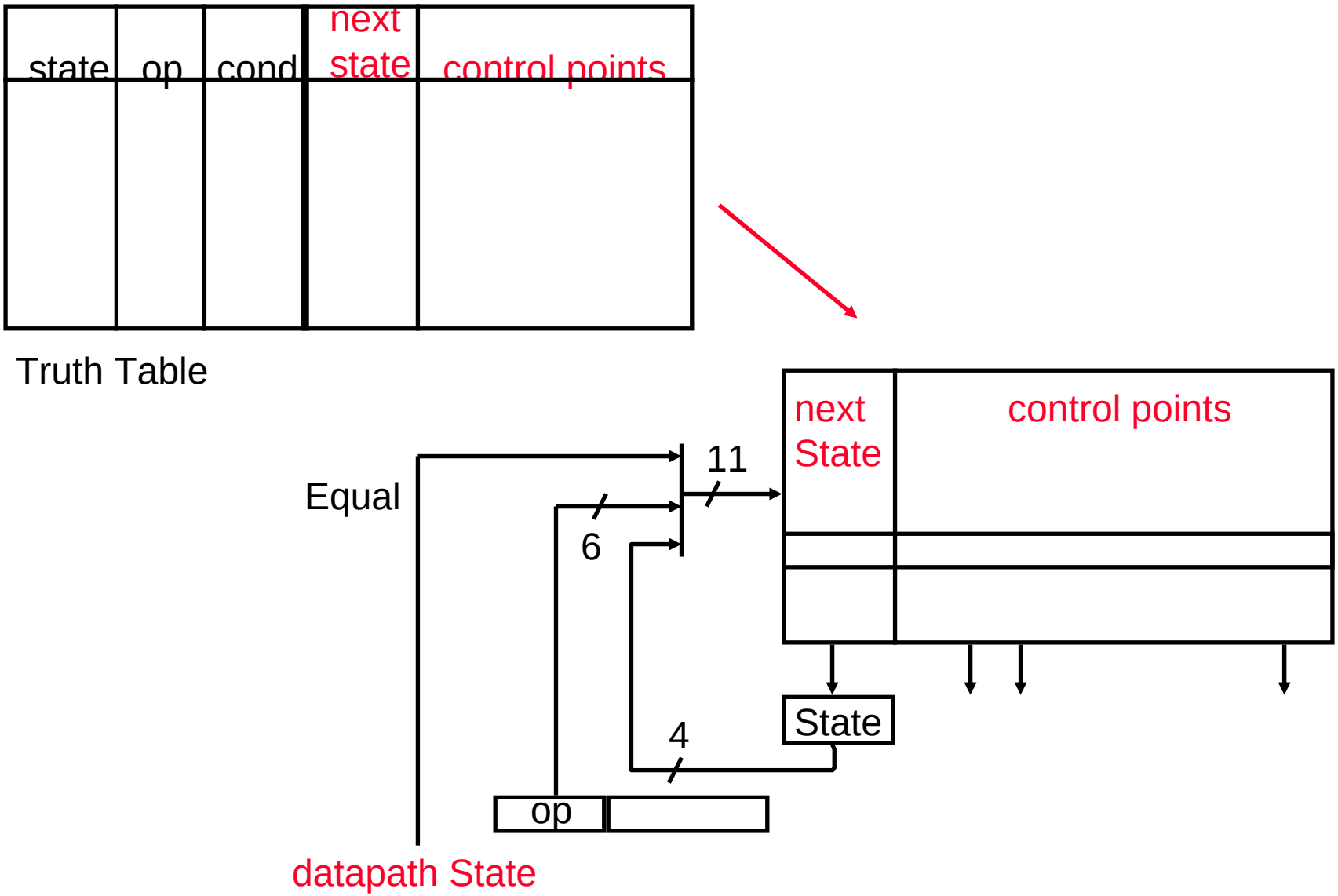




Step 4 => Control Specification for multicycle proc



Traditional FSM Controller

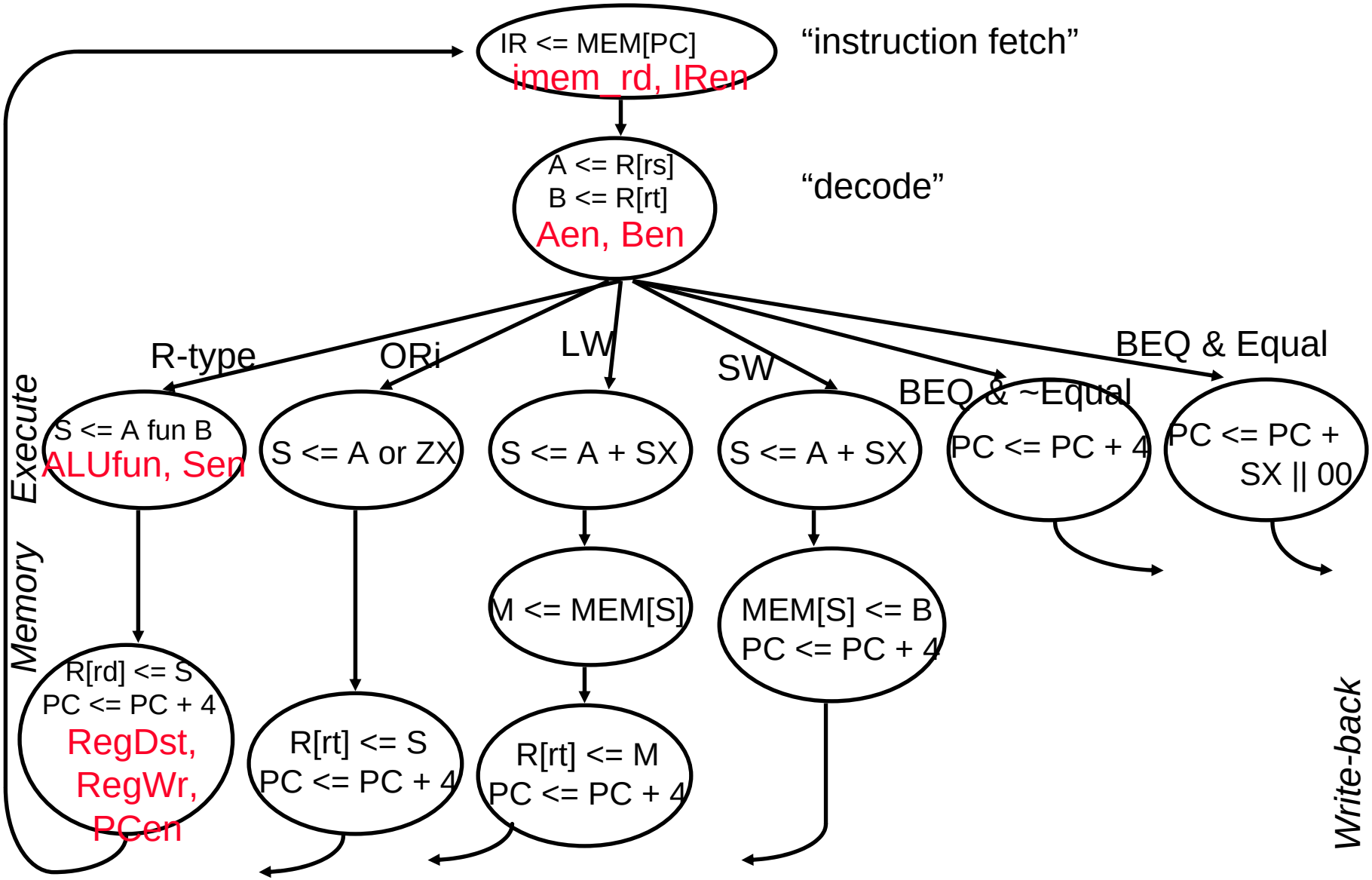


Step 5: datapath + state diagram => control

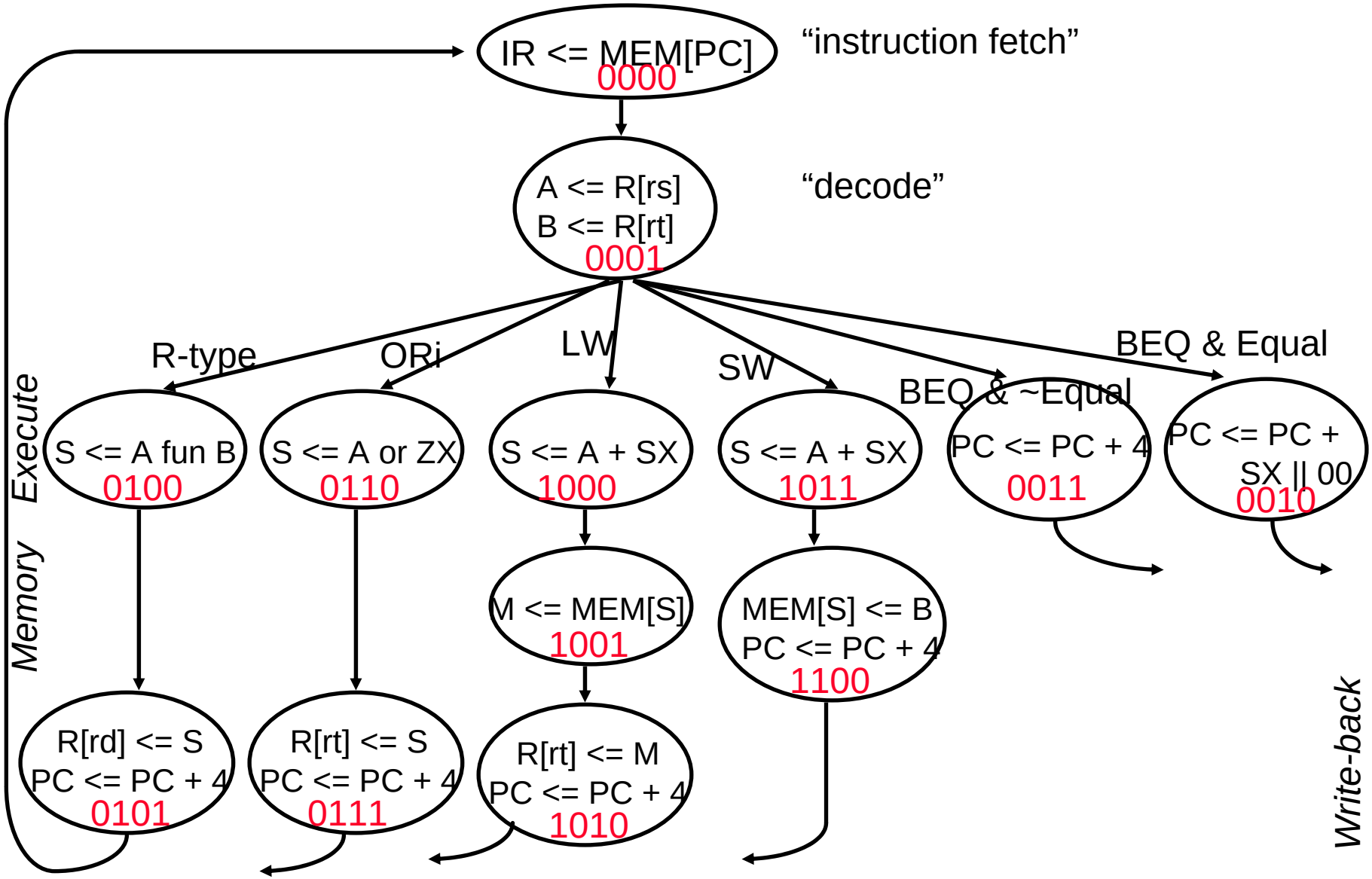
- Translate RTs into control points
- Assign states

- Then go build the controller

Mapping RTs to Control Points



Assigning States



Detailed Control Specification

	State	Op field	Eq	Next	IR	PC en	PC sel	Ops A	Ops B	Exec Ex	Exec Sr	Exec ALU	Exec S	Mem R	Mem W	Mem M	Write-Back M-R	Write-Back Wr	Write-Back Dst
	0000	??????	?	0001	1														
	0001	BEQ	0	0011				1	1	*-all same in Moore machine*									
	0001	BEQ	1	0010				1	1										
	0001	R-type	x	0100				1	1										
	0001	orl	x	0110				1	1										
	0001	LW	x	1000				1	1										
	0001	SW	x	1011				1	1										
	0010	xxxxxx	x	0000		1	1												
	0011	xxxxxx	x	0000		1	0												
R:	0100	xxxxxx	x	0101						0	1	fun	1						
	0101	xxxxxx	x	0000		1	0										0	1	1
ORi:	0110	xxxxxx	x	0111						0	0	or	1						
	0111	xxxxxx	x	0000		1	0										0	1	0
LW:	1000	xxxxxx	x	1001						1	0	add	1						
	1001	xxxxxx	x	1010										1	0	0			
	1010	xxxxxx	x	0000		1	0										1	1	0
SW:	1011	xxxxxx	x	1100						1	0	add	1						
	1100	xxxxxx	x	0000		1	0							0	1				

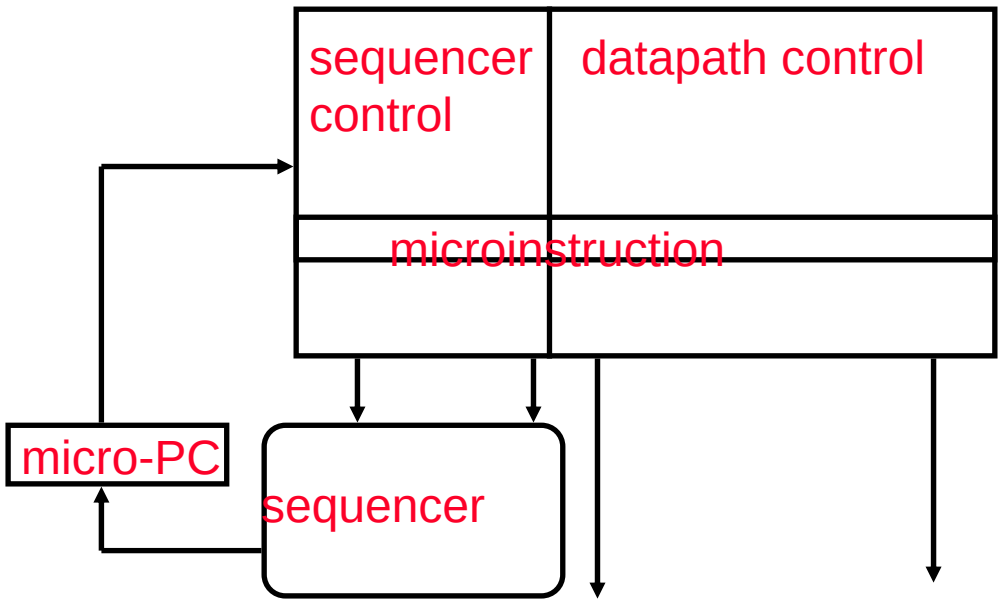
Performance Evaluation

- **What is the average CPI?**
  - **state diagram gives CPI for each instruction type**
  - **workload gives frequency of each type**

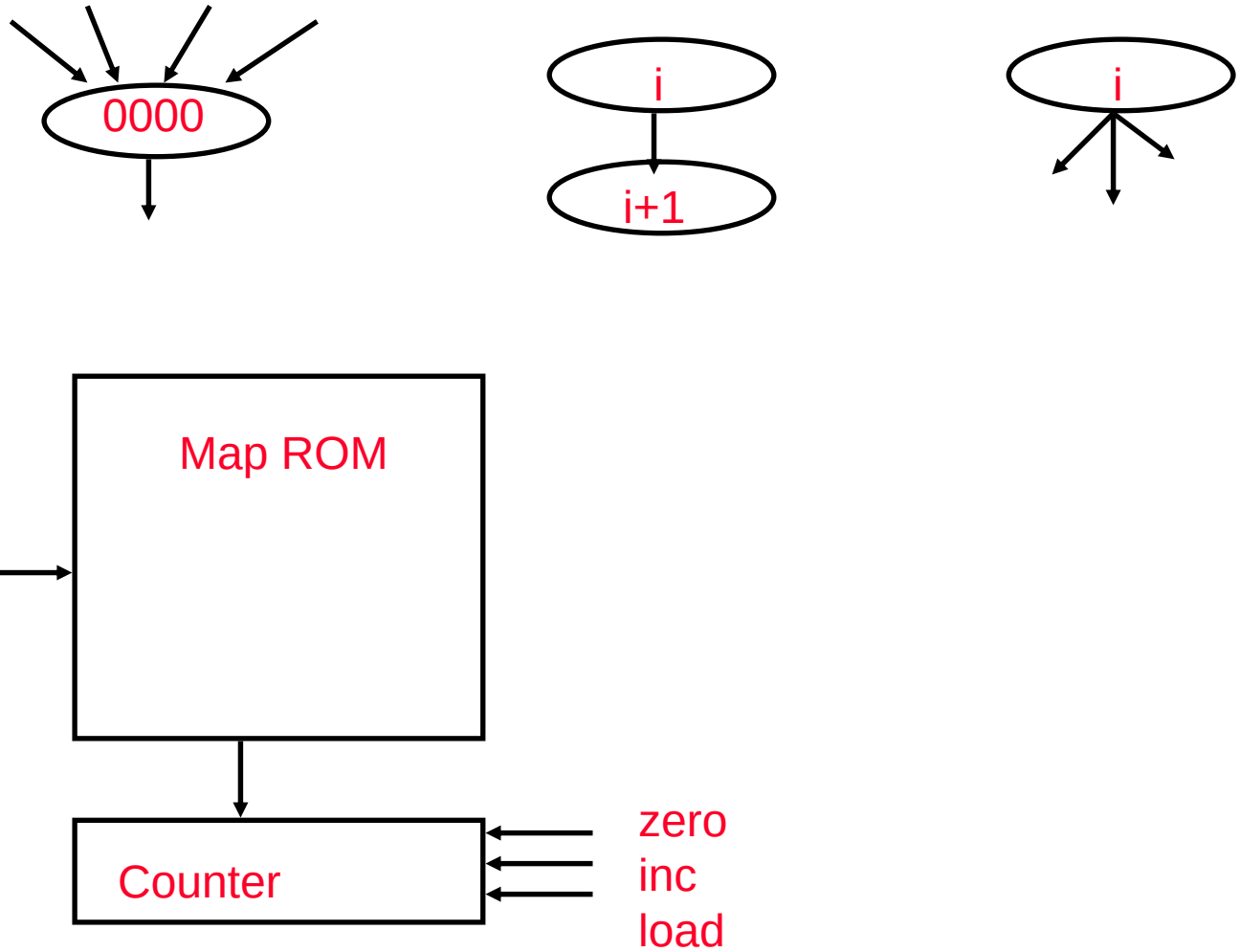
Type	CPI_i for type	Frequency	CPI_i x $freqI_i$
Arith/Logic	4	40%	1.6
Load	5	30%	1.5
Store	4	10%	0.4
branch	3	20%	0.6
		Average CPI:	4.1

Controller Design

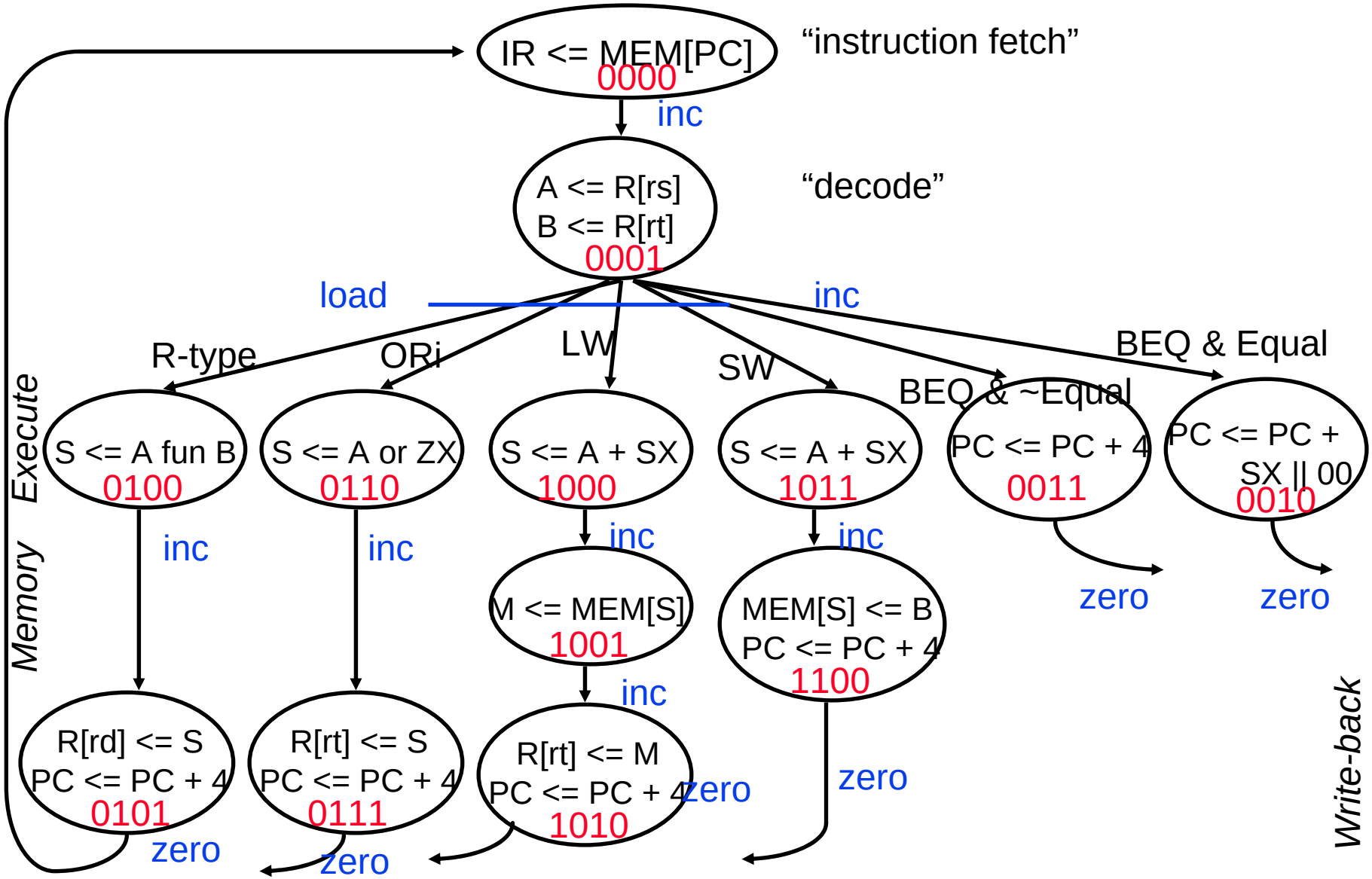
- The state digrams that arise define the controller for an instruction set processor are highly structured
- Use this structure to construct a simple “microsequencer”
- Control reduces to programming this very simple device
  - microprogramming



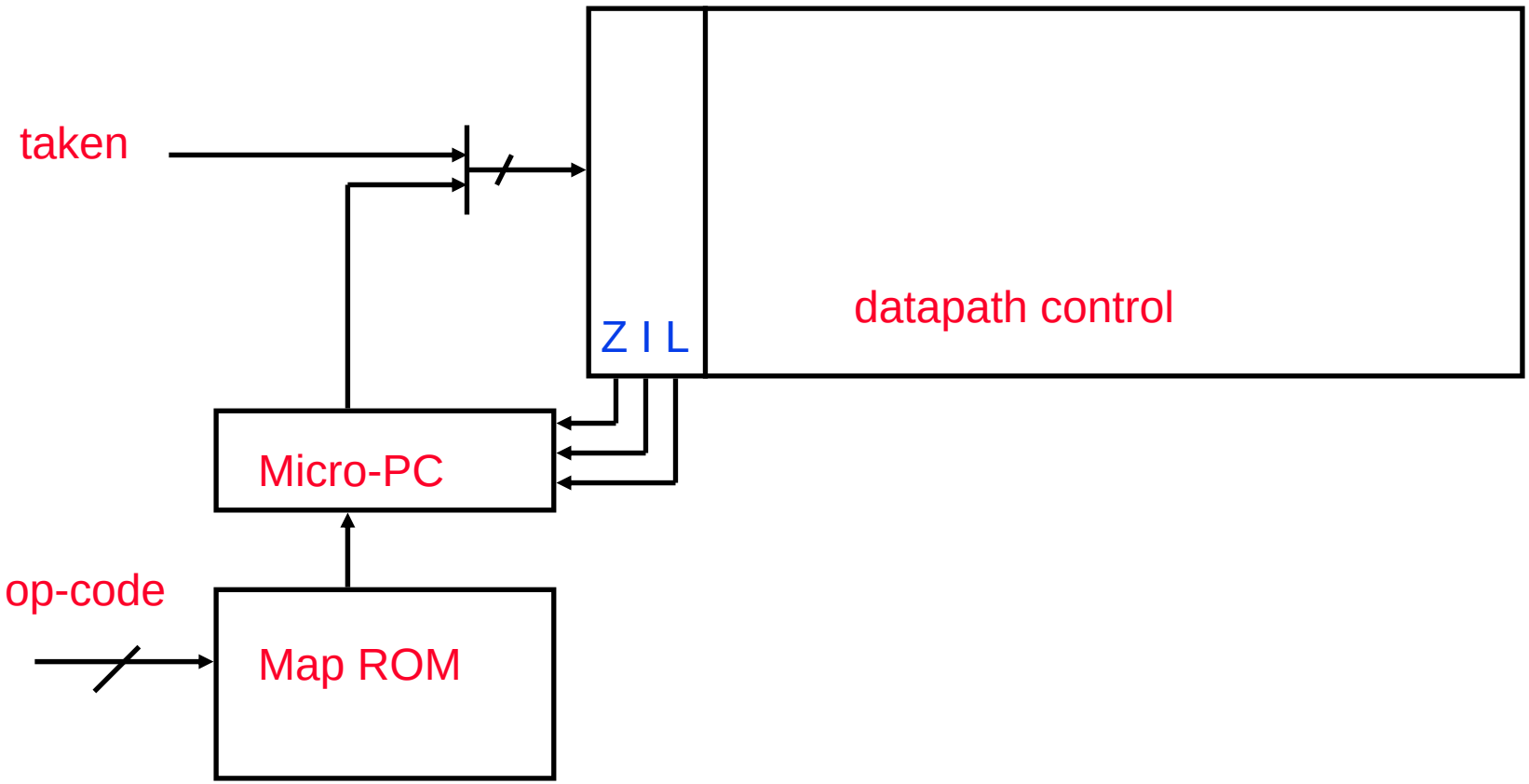
Example: Jump-Counter



Using a Jump Counter



Our Microsequencer



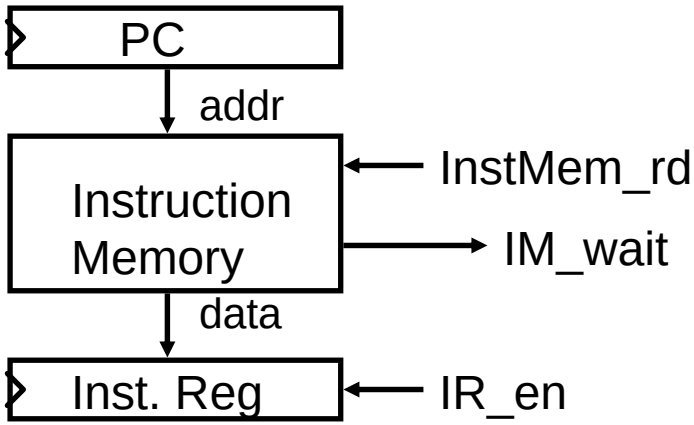
Microprogram Control Specification

	µPC	Taken	Next	IR	PC en	PC sel	Ops A	Ops B	Exec Ex	Exec Sr	Exec ALU	Exec S	Mem R	Mem W	Mem M	Write-Back M-R	Write-Back Wr	Write-Back Dst
	0000	?	inc	1														
	0001	0	load															
	0001	1	inc															
	0010	x	zero		1	1												
BEQ	0011	x	zero		1	0												
R:	0100	x	inc						0	1	fun	1						
	0101	x	zero		1	0										0	1	1
ORi:	0110	x	inc						0	0	or	1						
	0111	x	zero		1	0										0	1	0
LW:	1000	x	inc						1	0	add	1						
	1001	x	inc										1	0	0			
	1010	x	zero		1	0										1	1	0
SW:	1011	x	inc						1	0	add	1						
	1100	x	zero		1	0							0	1				

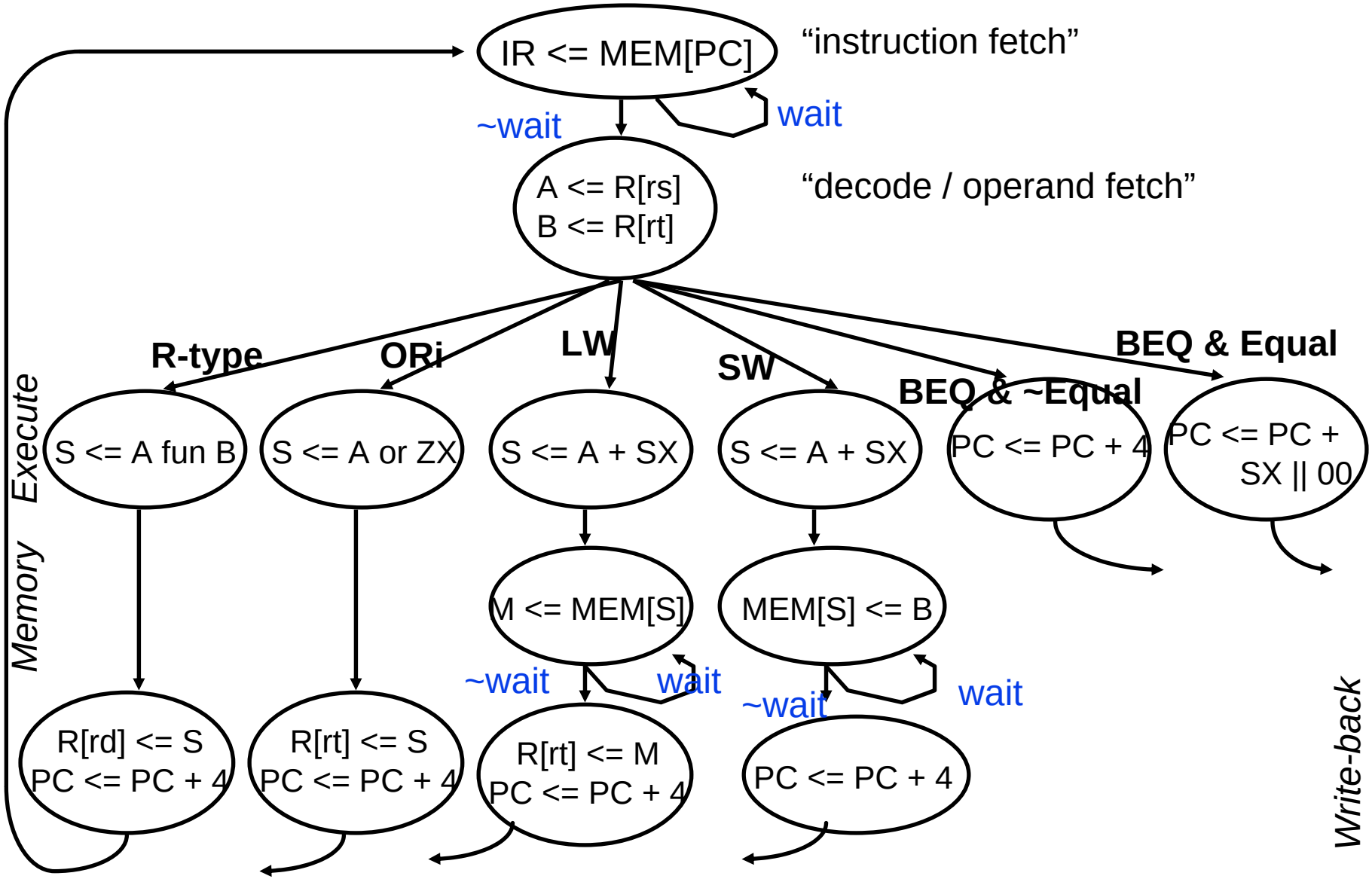
Mapping ROM

R-type	000000	0100
BEQ	000100	0011
ori	001101	0110
LW	100011	1000
SW	101011	1011

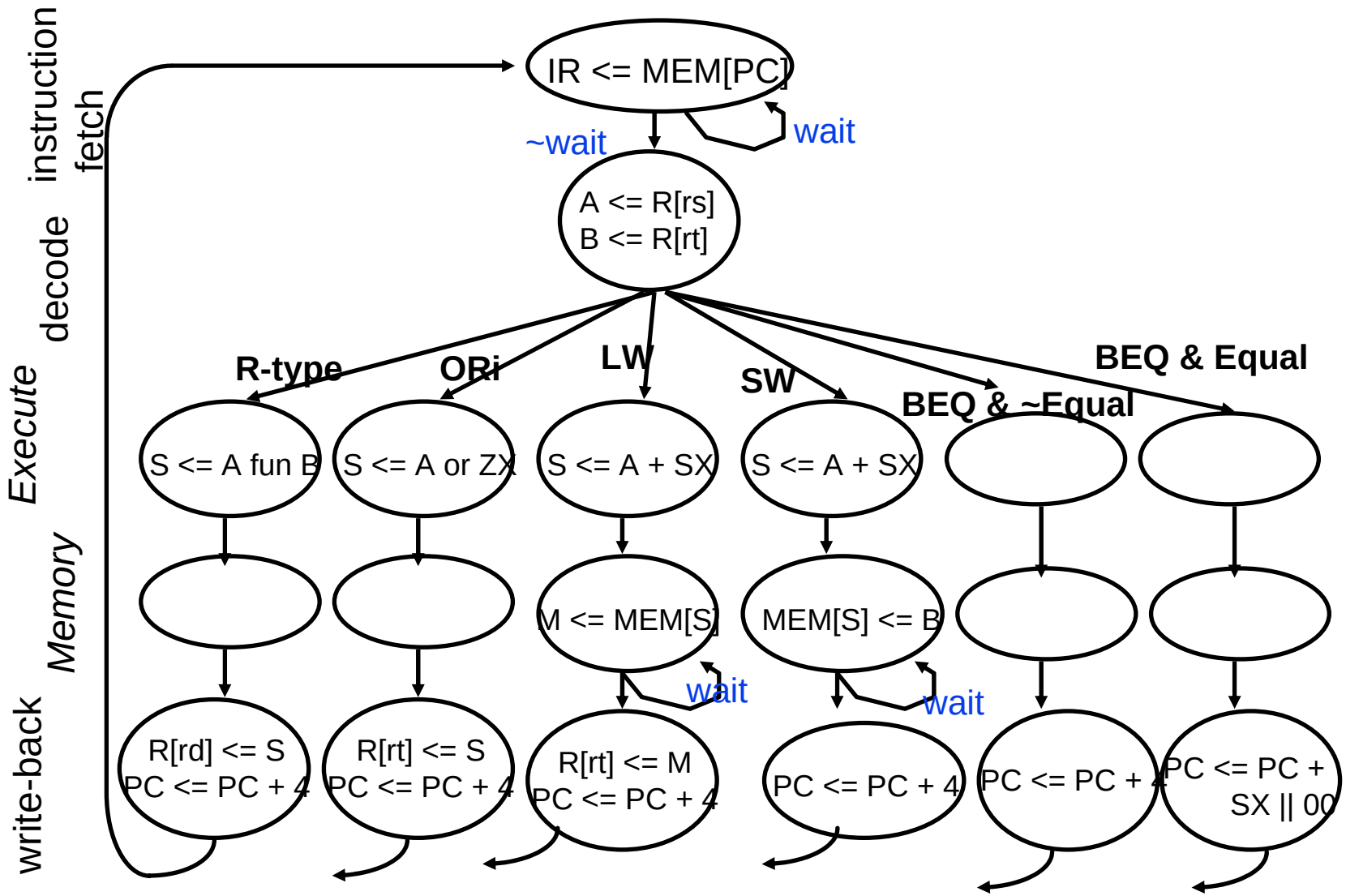
Example: Controlling Memory



Controller handles non-ideal memory

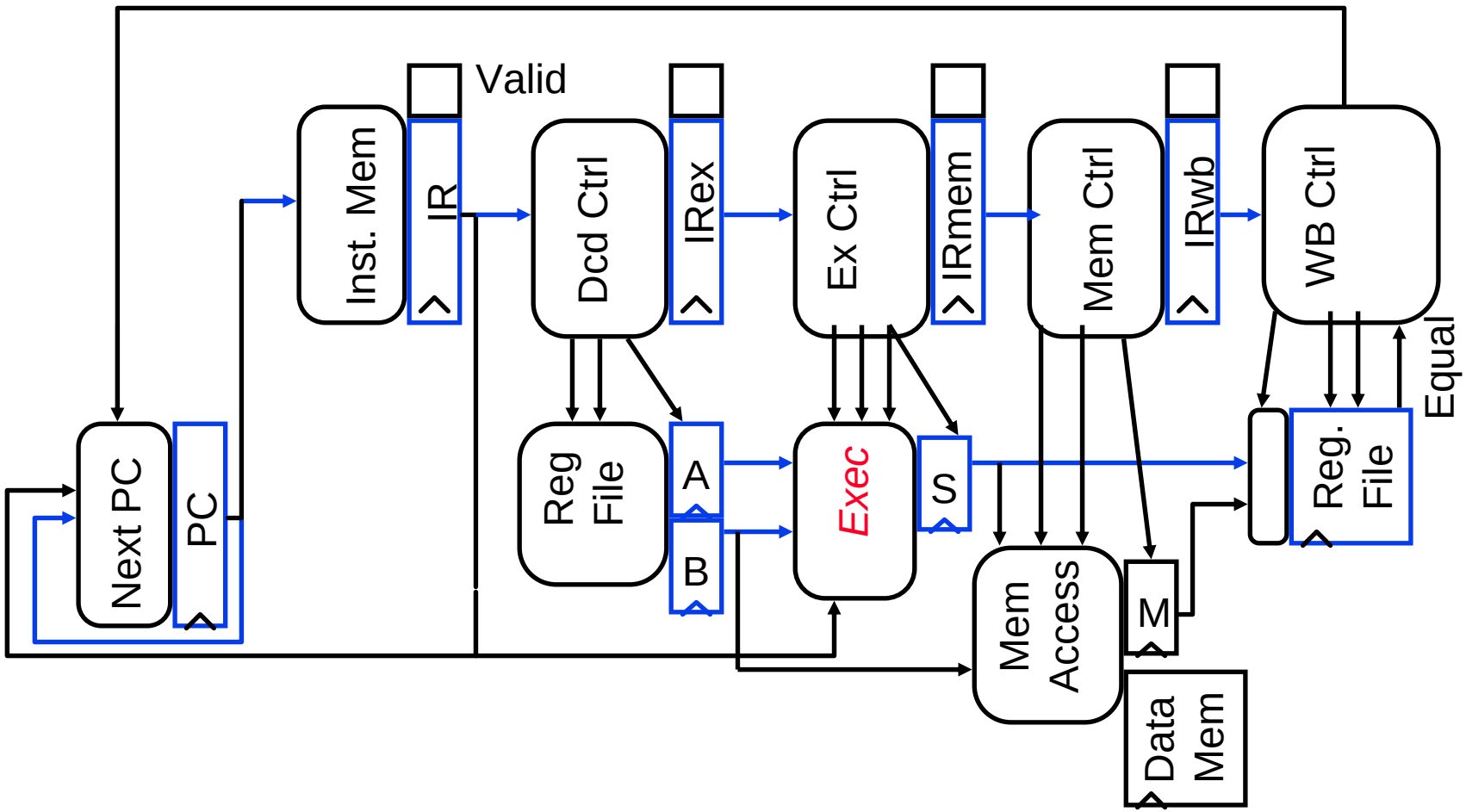


Really Simple Time-State Control



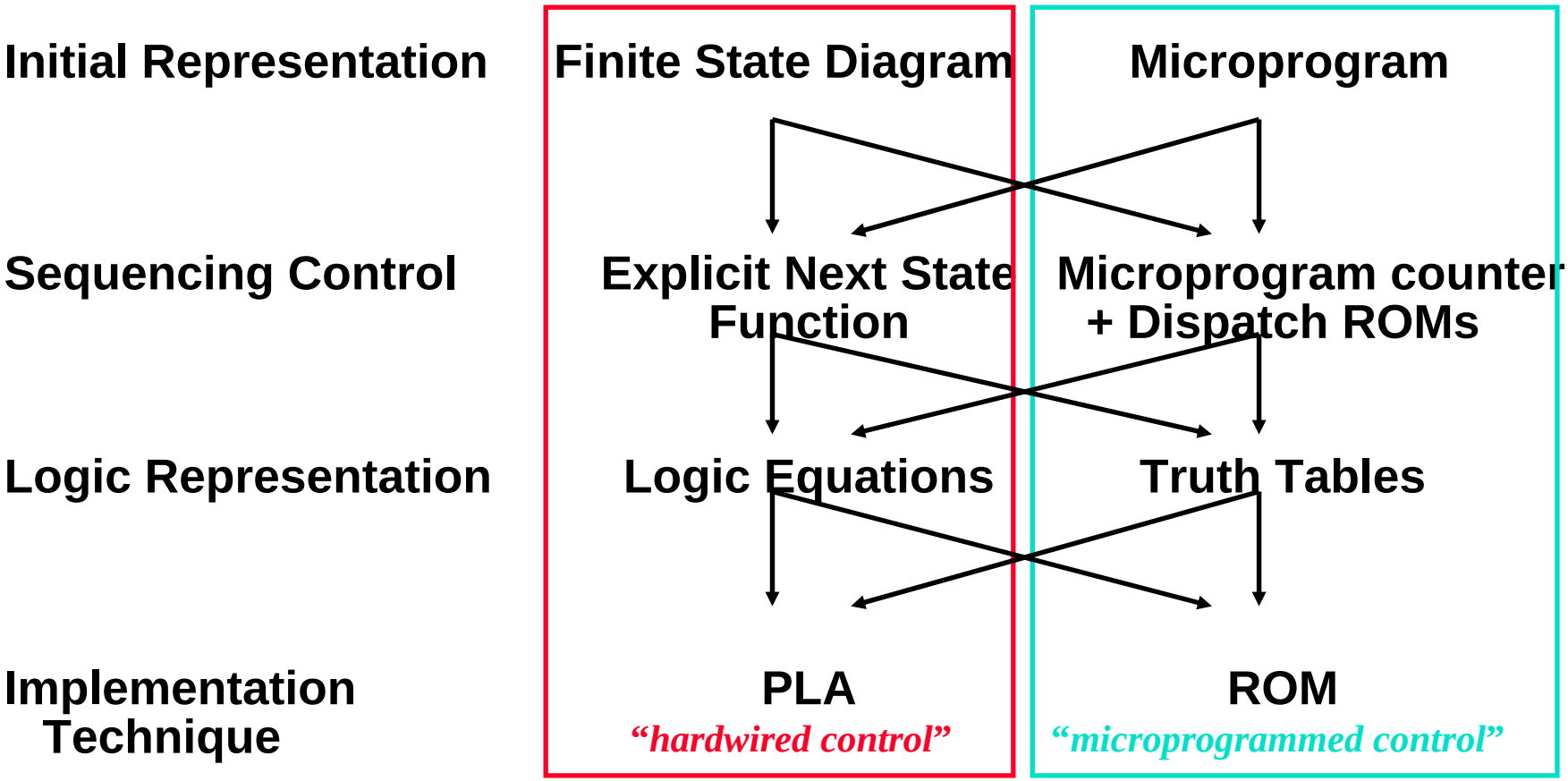
Time-state Control Path

- **Local decode and control at each stage**



Overview of Control

- Control may be designed using one of several initial representations. The choice of sequence control, and how logic is represented, can then be determined independently; the control can then be implemented with one of several methods using a structured logic technique.



Summary

- Disadvantages of the Single Cycle Processor
  - Long cycle time
  - Cycle time is too long for all instructions except the Load
- Multiple Cycle Processor:
  - Divide the instructions into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
- Partition datapath into equal size chunks to minimize cycle time
  - ~10 levels of logic between latches
- Follow same 5-step method for designing “real” processor

Summary (cont'd)

- Control is specified by finite state digram
- Specialize state-diagrams easily captured by microsequencer
  - simple increment & "branch" fields
  - datapath control fields
- Control design reduces to Microprogramming
- Control is more complicated with:
  - complex instruction sets
  - restricted datapaths (see the book)
- Simple Instruction set and powerful datapath => simple control
  - could try to reduce hardware (see the book)
  - rather go for speed => many instructions at once!

Where to get more information?

- **Next two lectures:**
  - **Multiple Cycle Controller: Appendix C of your text book.**
  - **Microprogramming: Section 5.5 of your text book.**
- **D. Patterson, “Microprograming,” Scientific America, March 1983.**
- **D. Patterson and D. Ditzel, “The Case for the Reduced Instruction Set Computer,” Computer Architecture News 8, 6 (October 15, 1980)**