RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

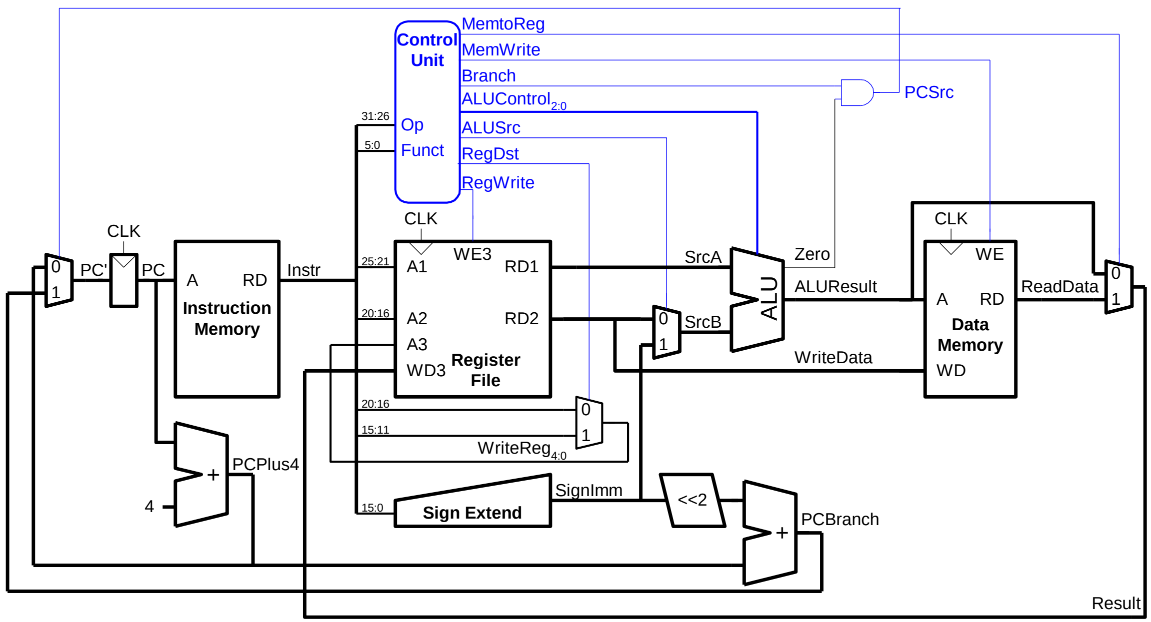
ECE-332:437
DIGITAL SYSTEMS DESIGN (DSD)

Fall 2016 – Lecture 9
Micro Architecture (MIPS) – Additional Material

Nagi Naganathan
November 3, 2016

Control Unit Main Decoder

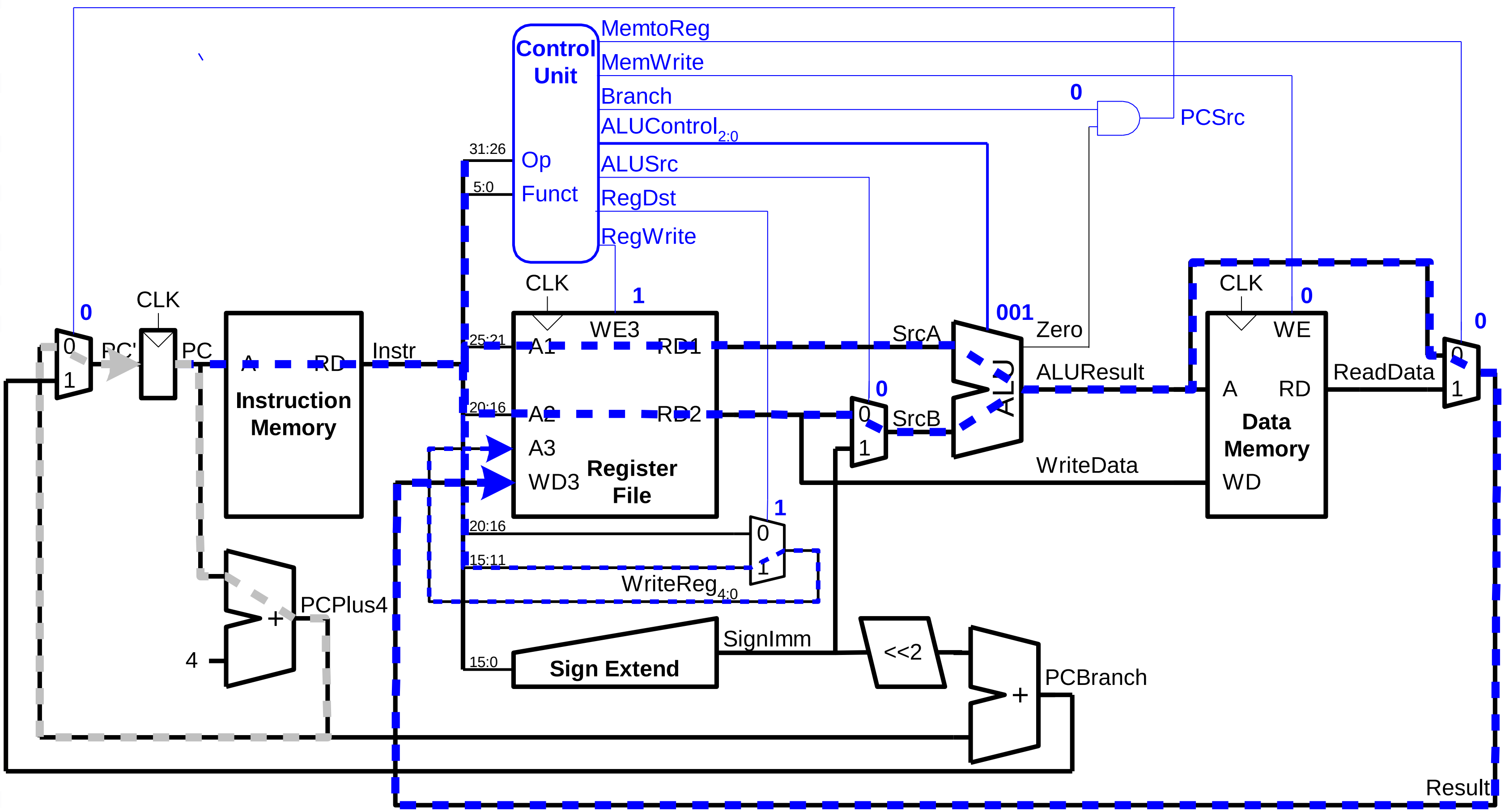
Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000							
`lw`	100011							
`sw`	101011							
`beq`	000100							



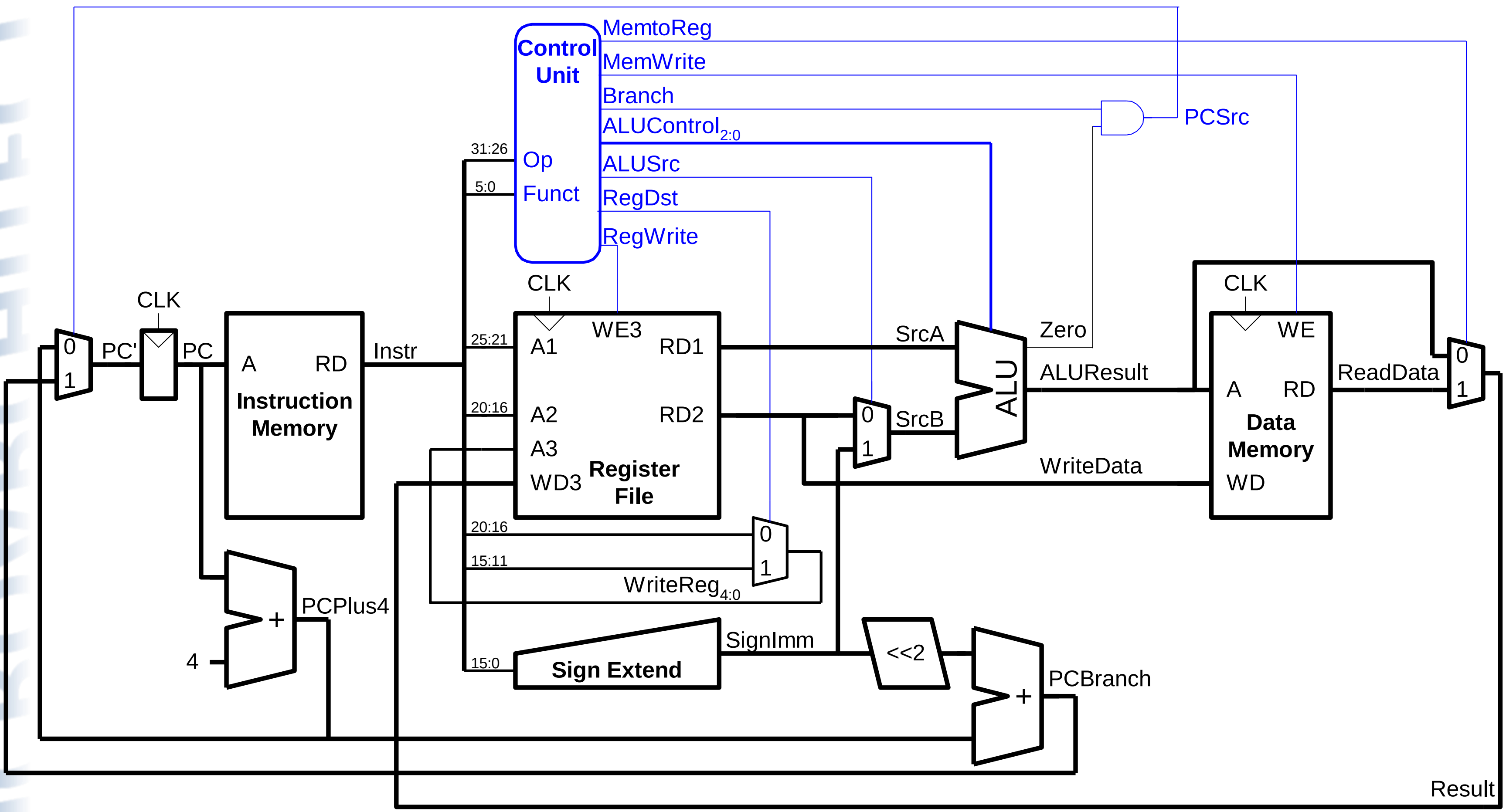
Control Unit: Main Decoder

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	0	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01

Single-Cycle Datapath: `or`



Extended Functionality: `addi`



No change to datapath

Control Unit: addi

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
addi	**001000**							

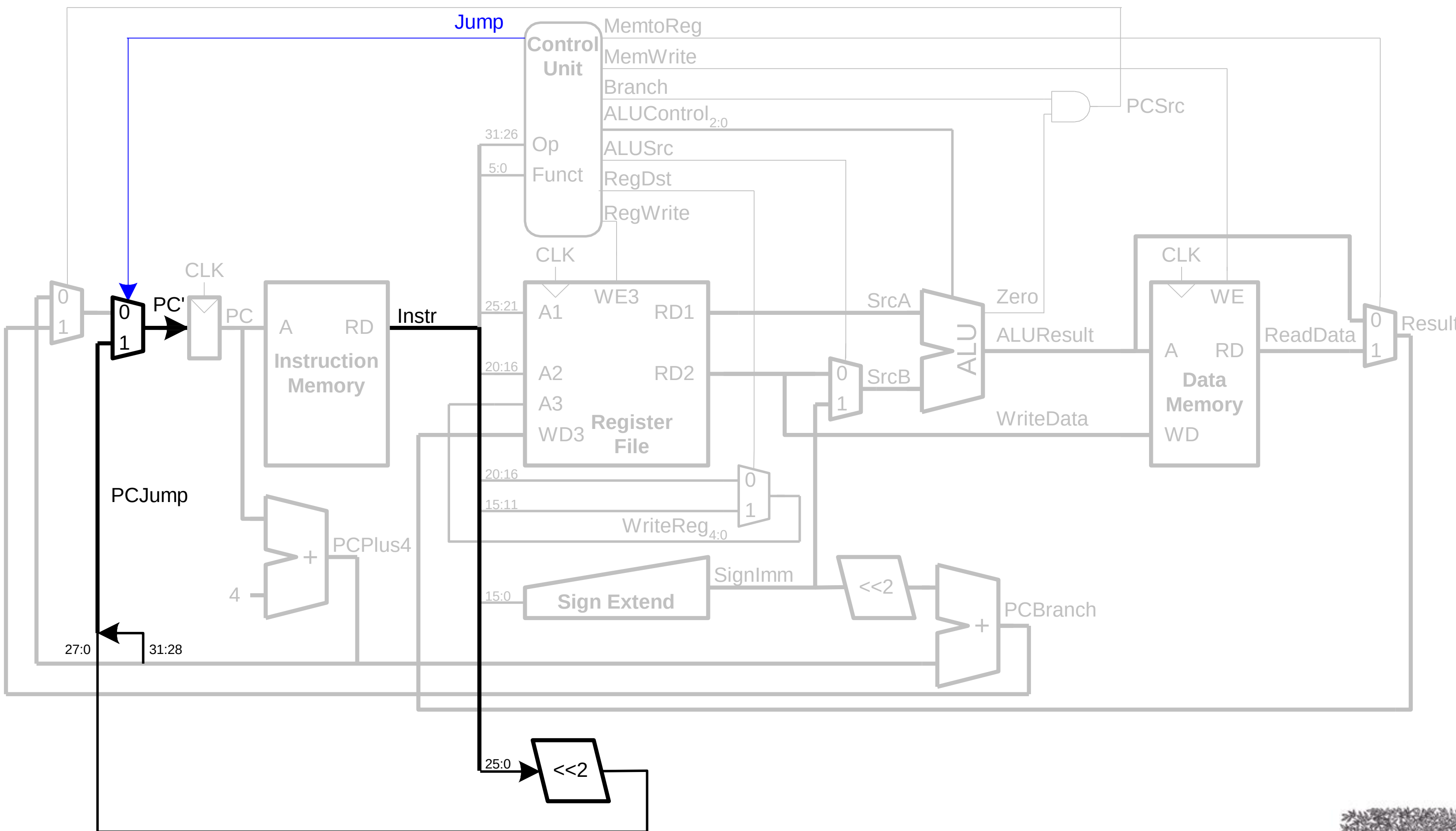
MICROARCHITECTURE

ELSEVIER

Control Unit: addi

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
addi	**001000**	**1**	**0**	**1**	**0**	**0**	**0**	**00**

Extended Functionality: j



ELSEVIER

Control Unit: Main Decoder

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	000000	1	1	0	0	0	0	10	0
`lw`	100011	1	0	1	0	0	1	00	0
`sw`	101011	0	X	1	0	1	X	00	0
`beq`	000100	0	X	0	1	0	X	01	0
`j`	000010								

Control Unit: Main Decoder

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	000000	1	1	0	0	0	0	10	0
`lw`	100011	1	0	1	0	0	1	00	0
`sw`	101011	0	X	1	0	1	X	00	0
`beq`	000100	0	X	0	1	0	X	01	0
`j`	000010	0	X	X	X	0	X	XX	1

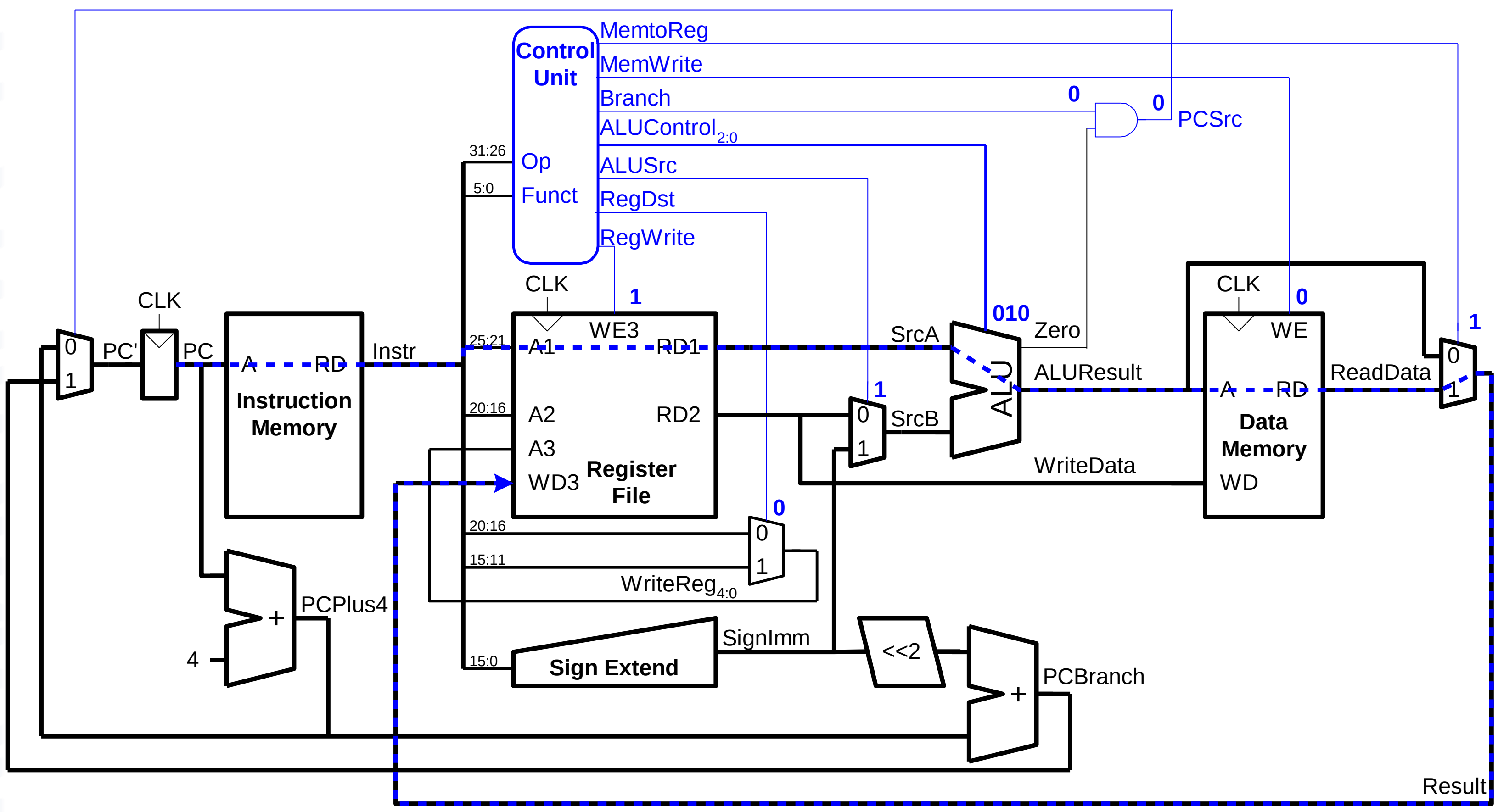
Review: Processor Performance

Program Execution Time

= (#instructions)(cycles/instruction)(seconds/cycle)

= # instructions x CPI x T_C

Single-Cycle Performance



T_C **limited by critical path (`lw`)**

Single-Cycle Performance

- Single-cycle critical path:

  $T_c = t_{pcq\_PC} + t_{\mathrm{mem}} + \max(t_{RF\mathrm{read}}, t_{sext} + t_{\mathrm{mux}}) + t_{\mathrm{ALU}} + t_{\mathrm{mem}} + t_{\mathrm{mux}} + t_{RF\mathrm{setup}}$

- Typically, limiting paths are:
  – memory, ALU, register file
  – $T_c = t_{pcq\_PC} + 2t_{\mathrm{mem}} + t_{RF\mathrm{read}} + t_{\mathrm{mux}} + t_{\mathrm{ALU}} + t_{RF\mathrm{setup}}$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

$T_c = ?$

MICROARCHITECTURE

ELSEVIER

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	$t_{RF\text{read}}$	150
Register file setup	$t_{RF\text{setup}}$	20

$$T_c = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{mux}} + t_{\text{ALU}} + t_{RF\text{setup}}$$
$$= [30 + 2(250) + 150 + 25 + 200 + 20] \text{ ps}$$
$$= 925 \text{ ps}$$

Single-Cycle Performance Example

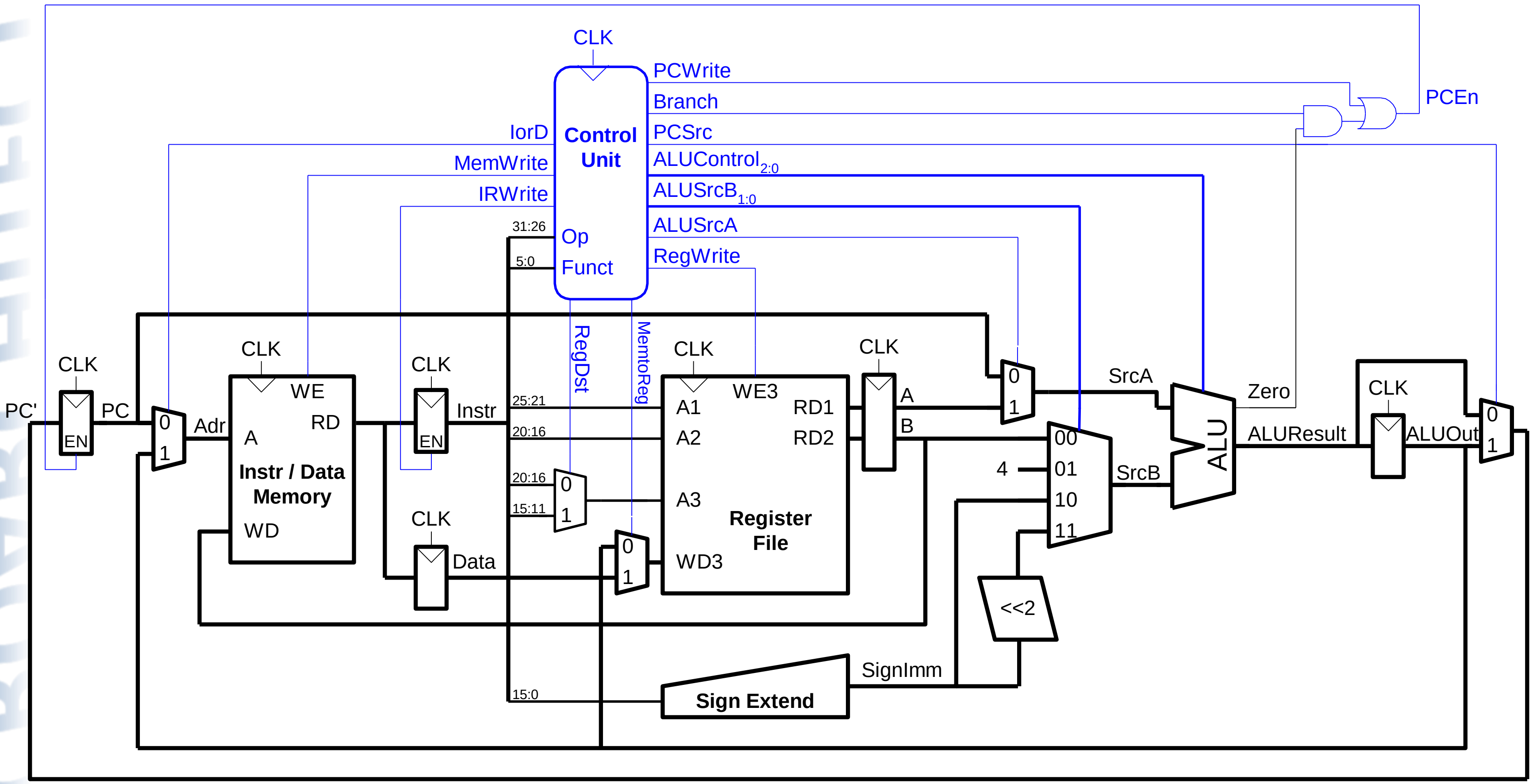
Program with 100 billion instructions:

Execution Time = # instructions x CPI x T_C

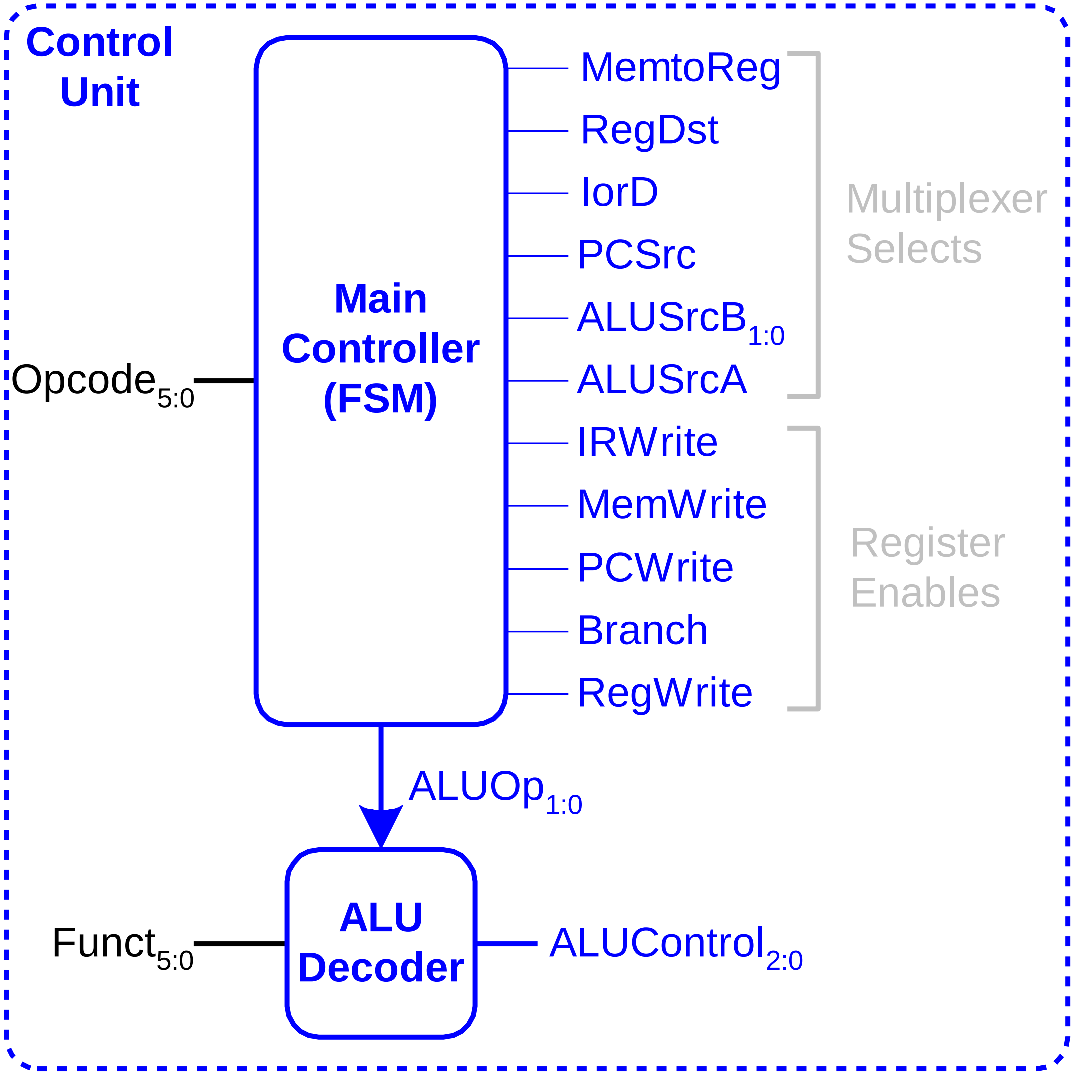
$= (100 \times 10^9)(1)(925 \times 10^{-12}\text{ s})$

= **92.5 seconds**

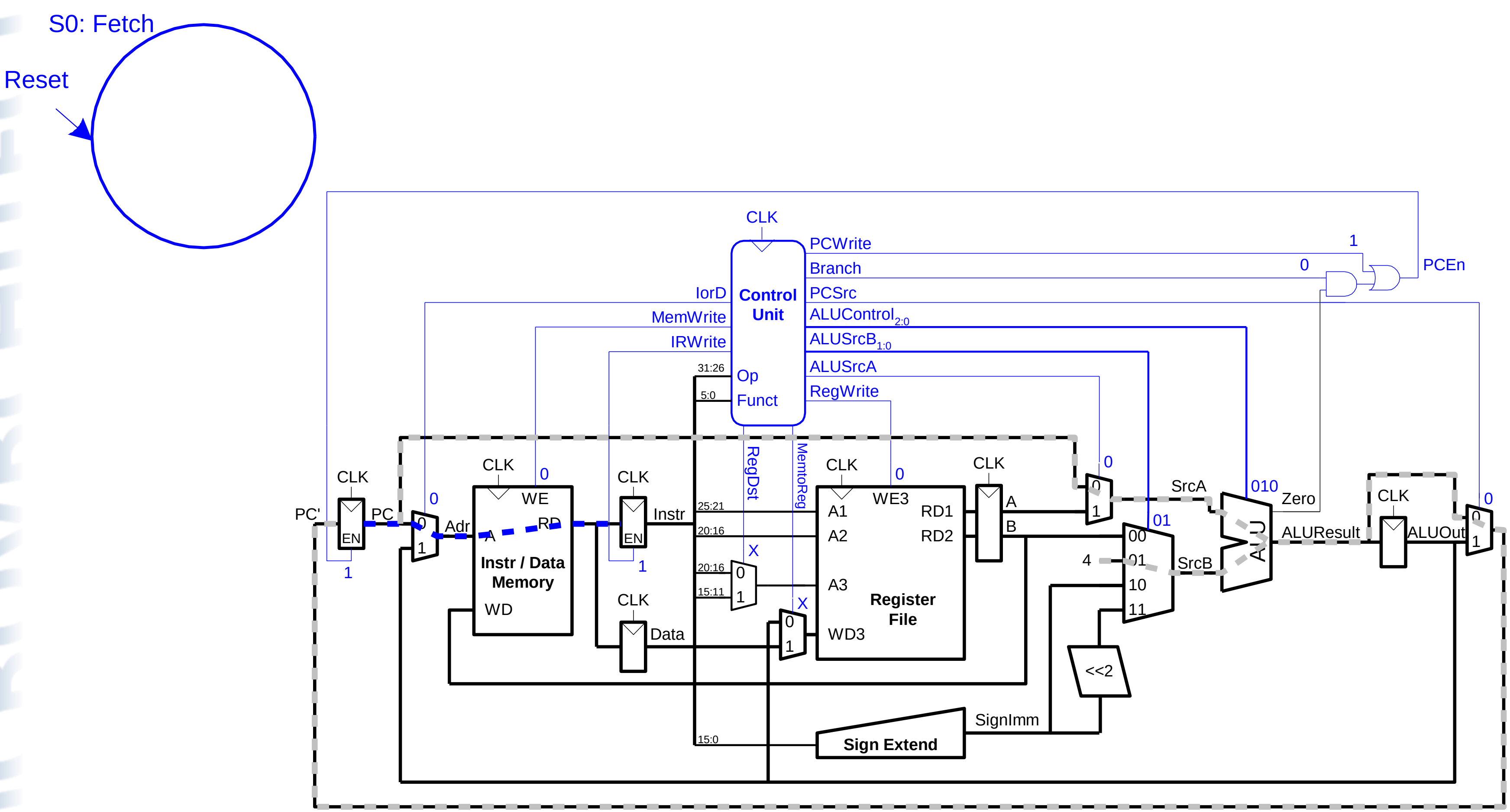
Multicycle Processor



Multicycle Control

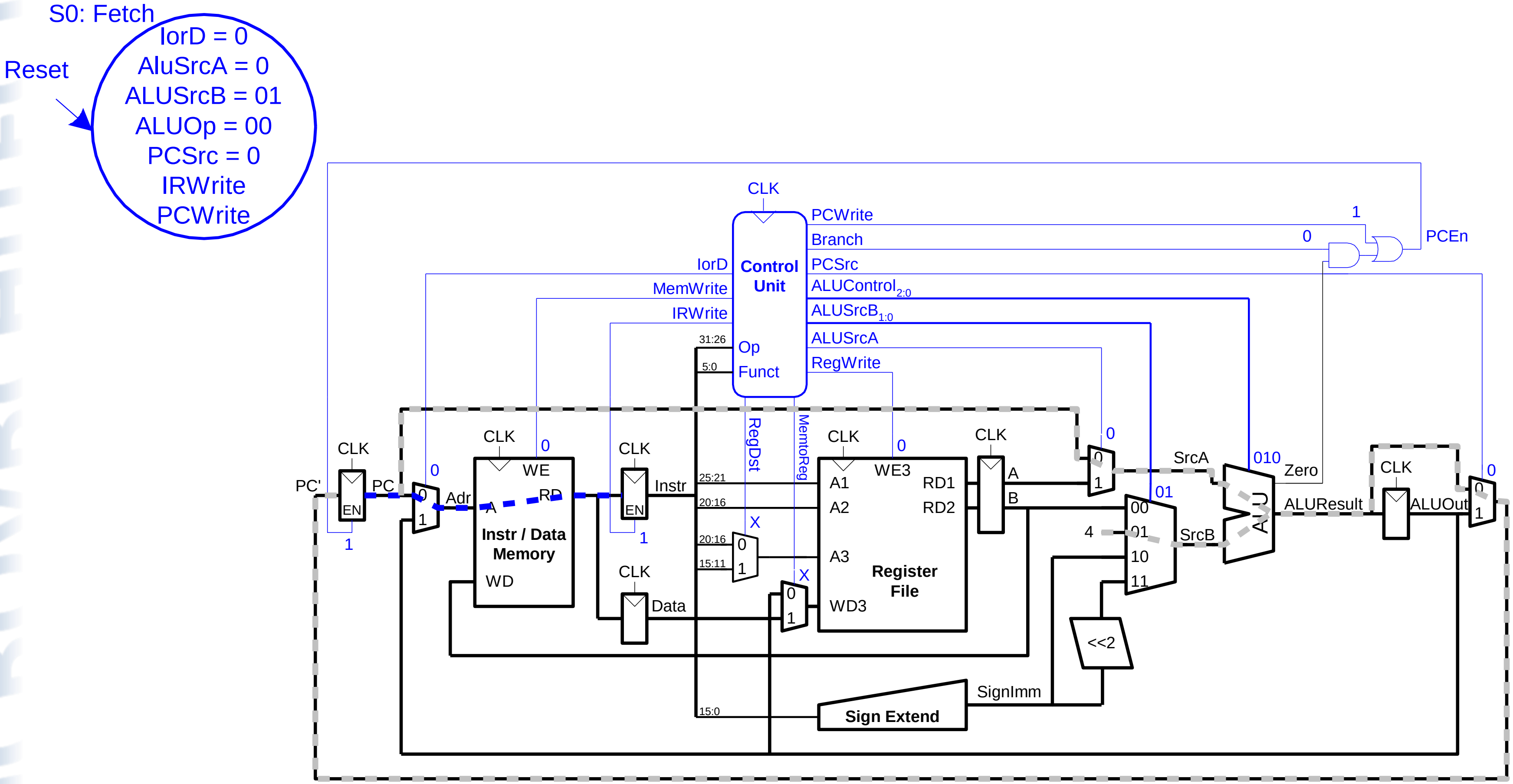


Main Controller FSM: Fetch



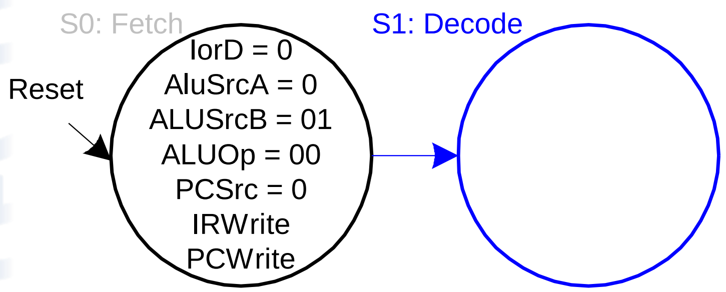
MICROARCHITECTURE

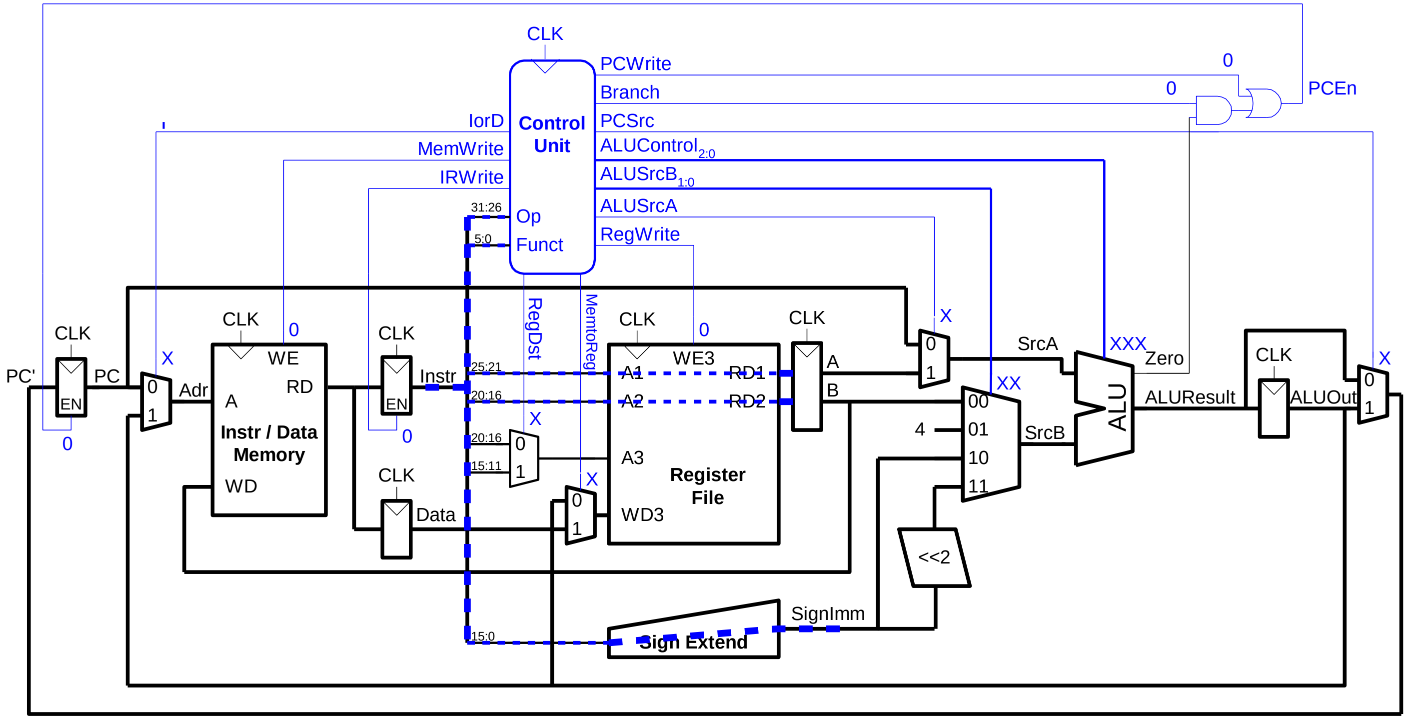
Main Controller FSM: Fetch



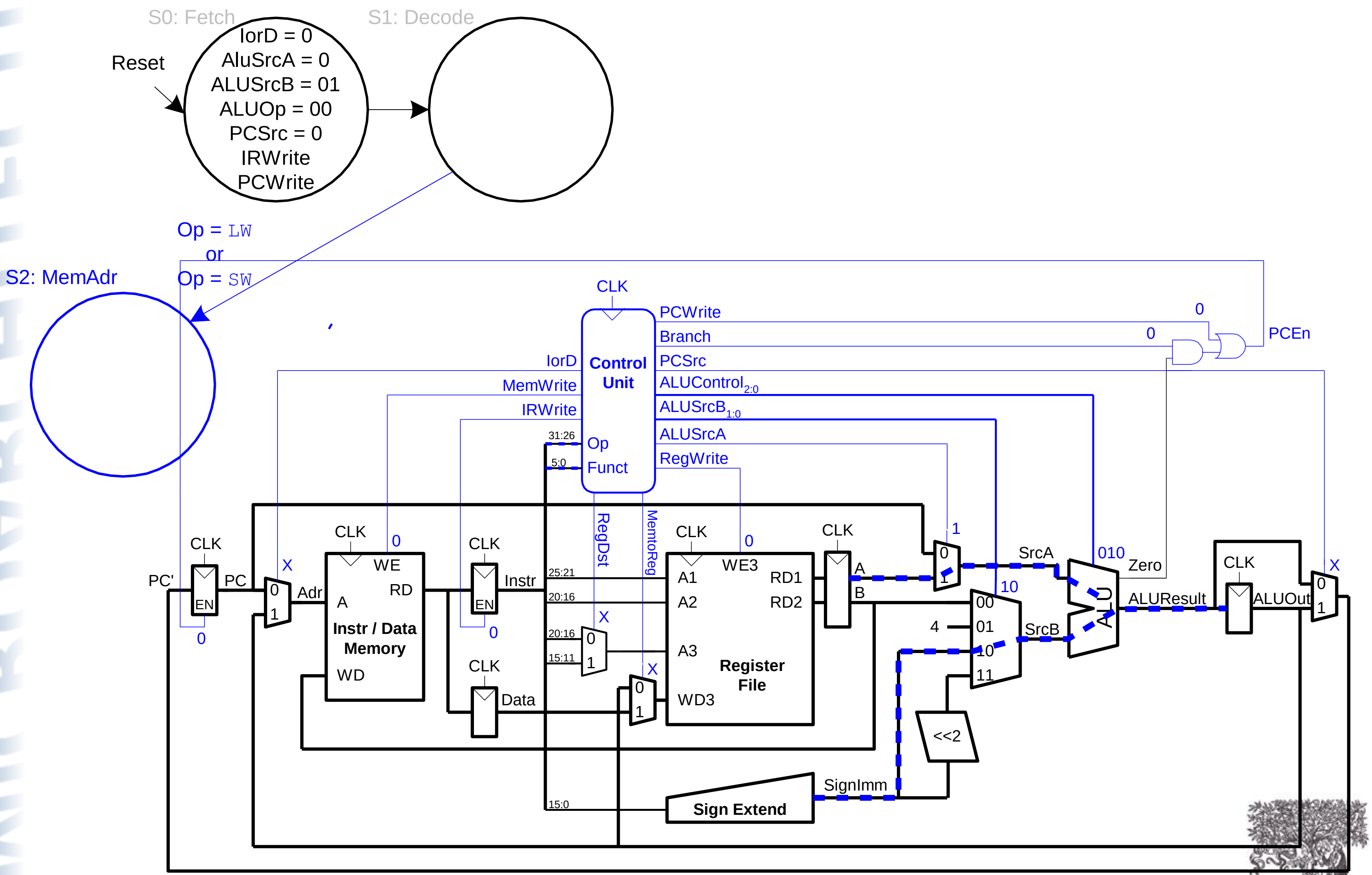
ELSEVIER

Main Controller FSM: Decode

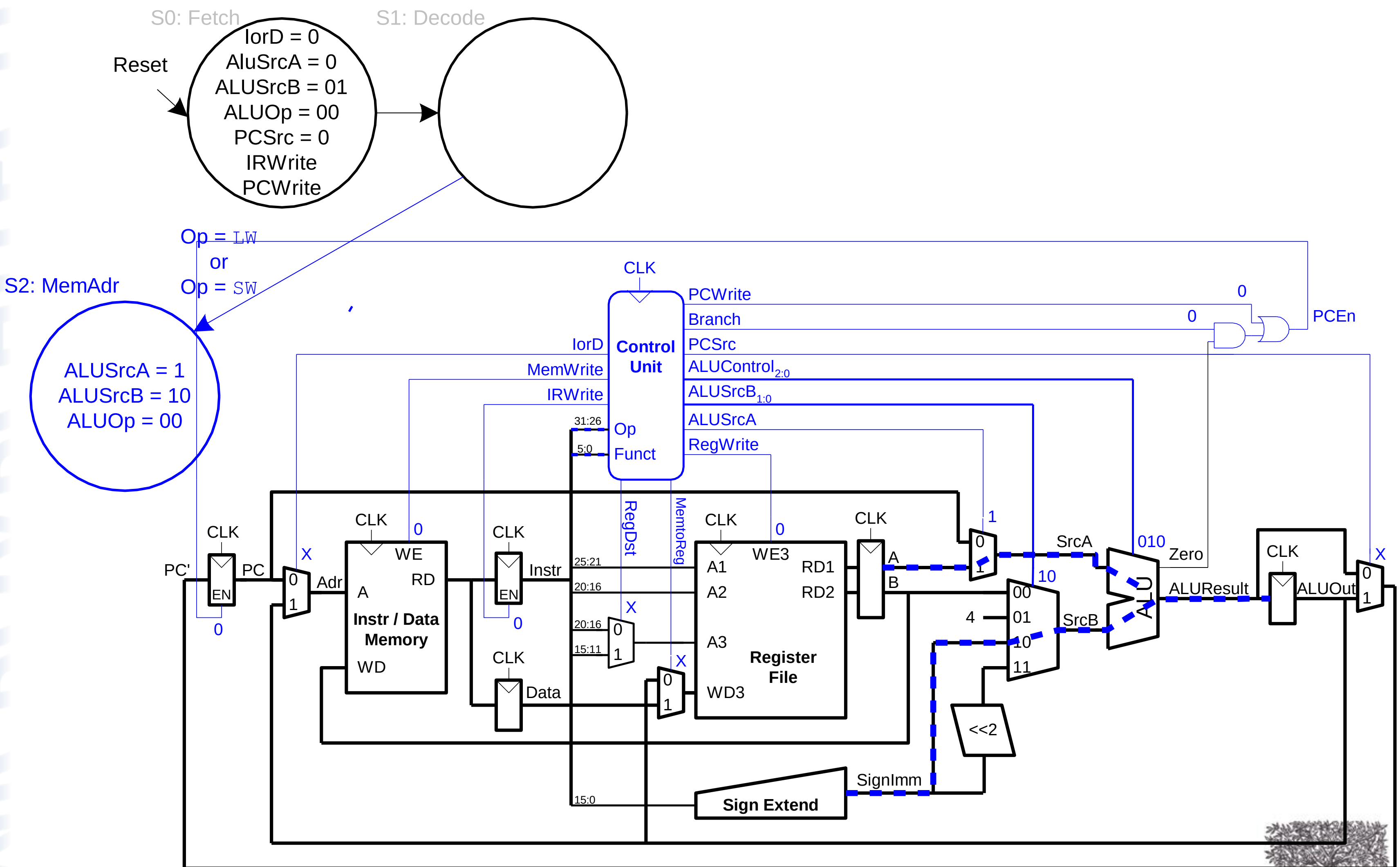




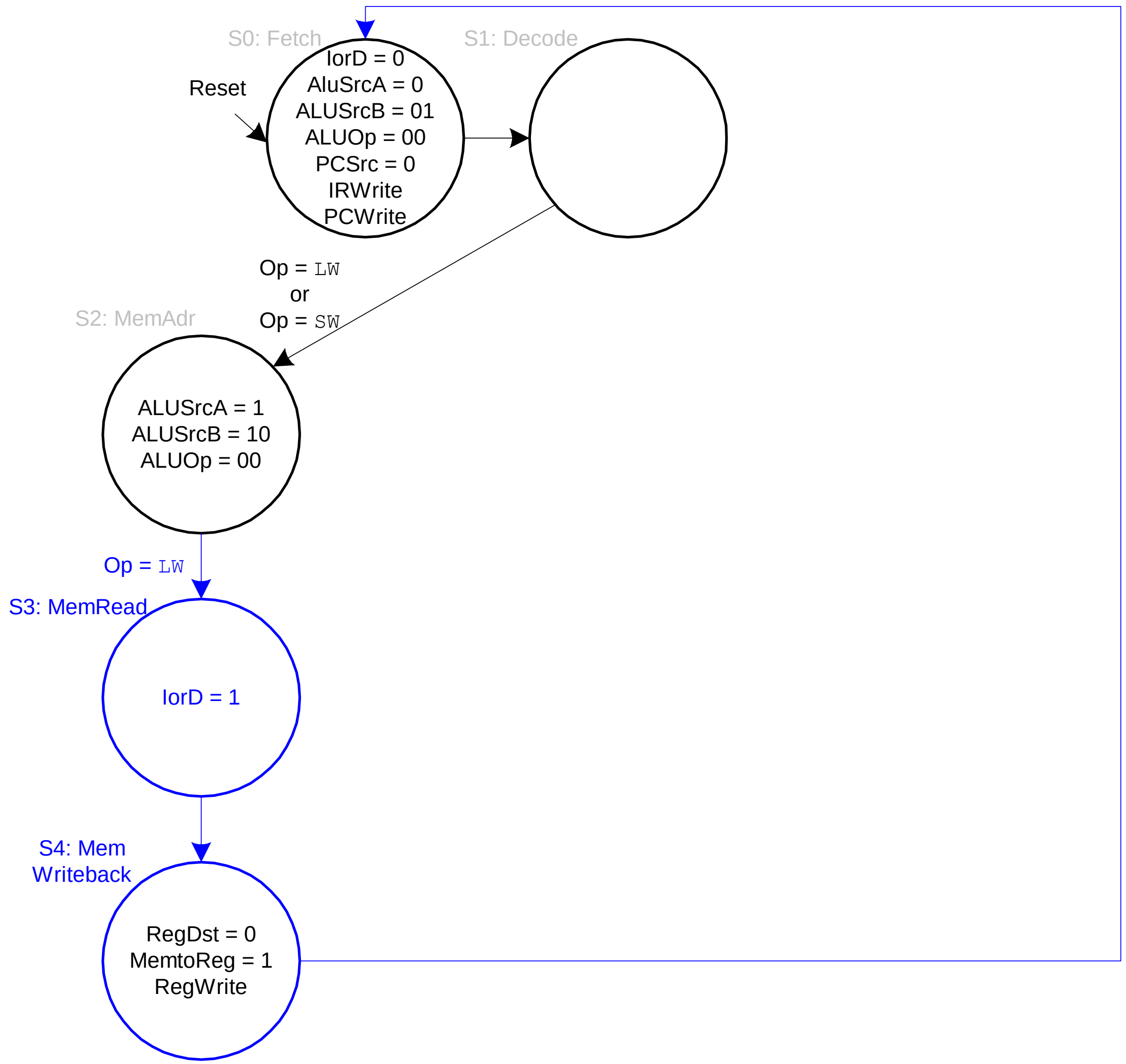
Main Controller FSM: Address



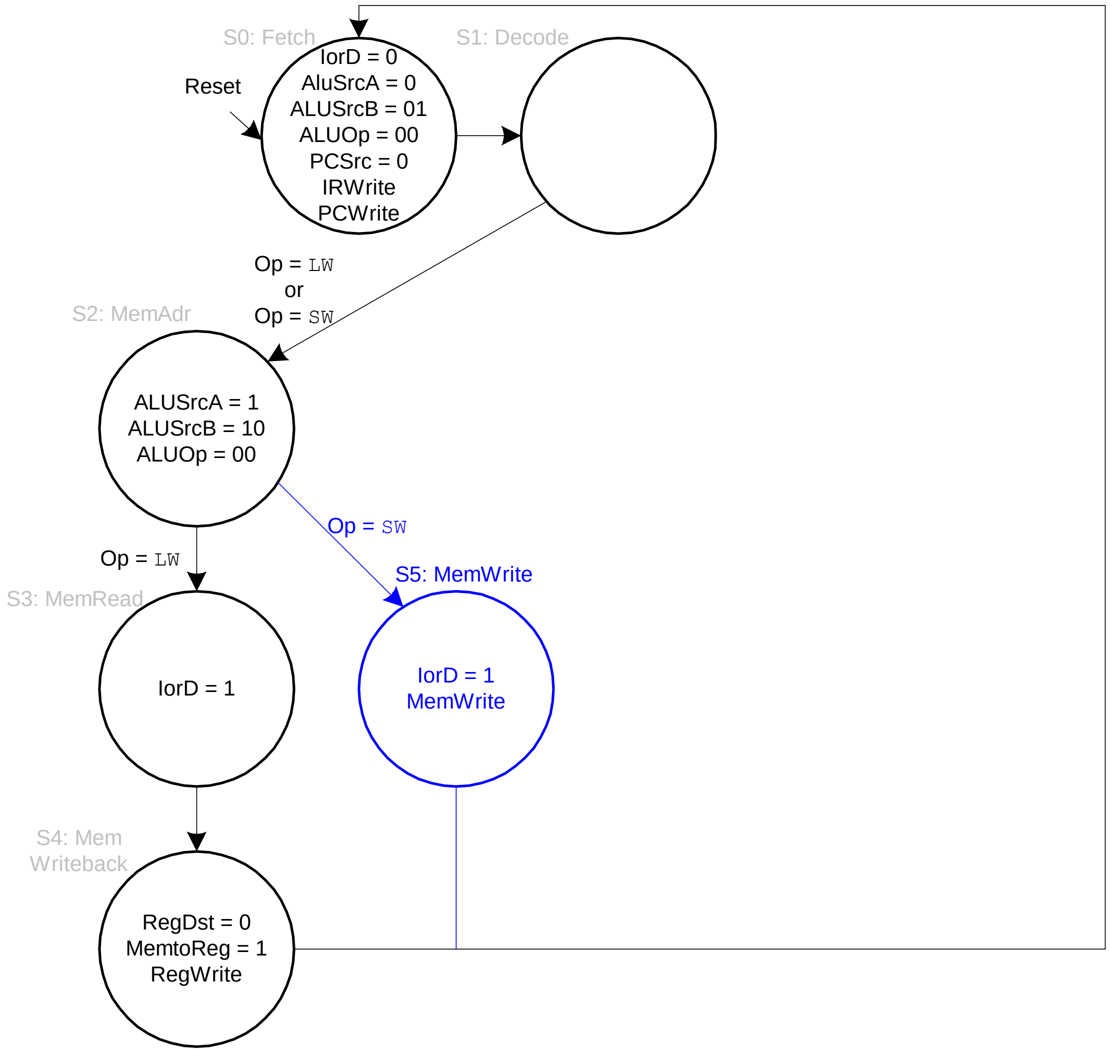
Main Controller FSM: Address



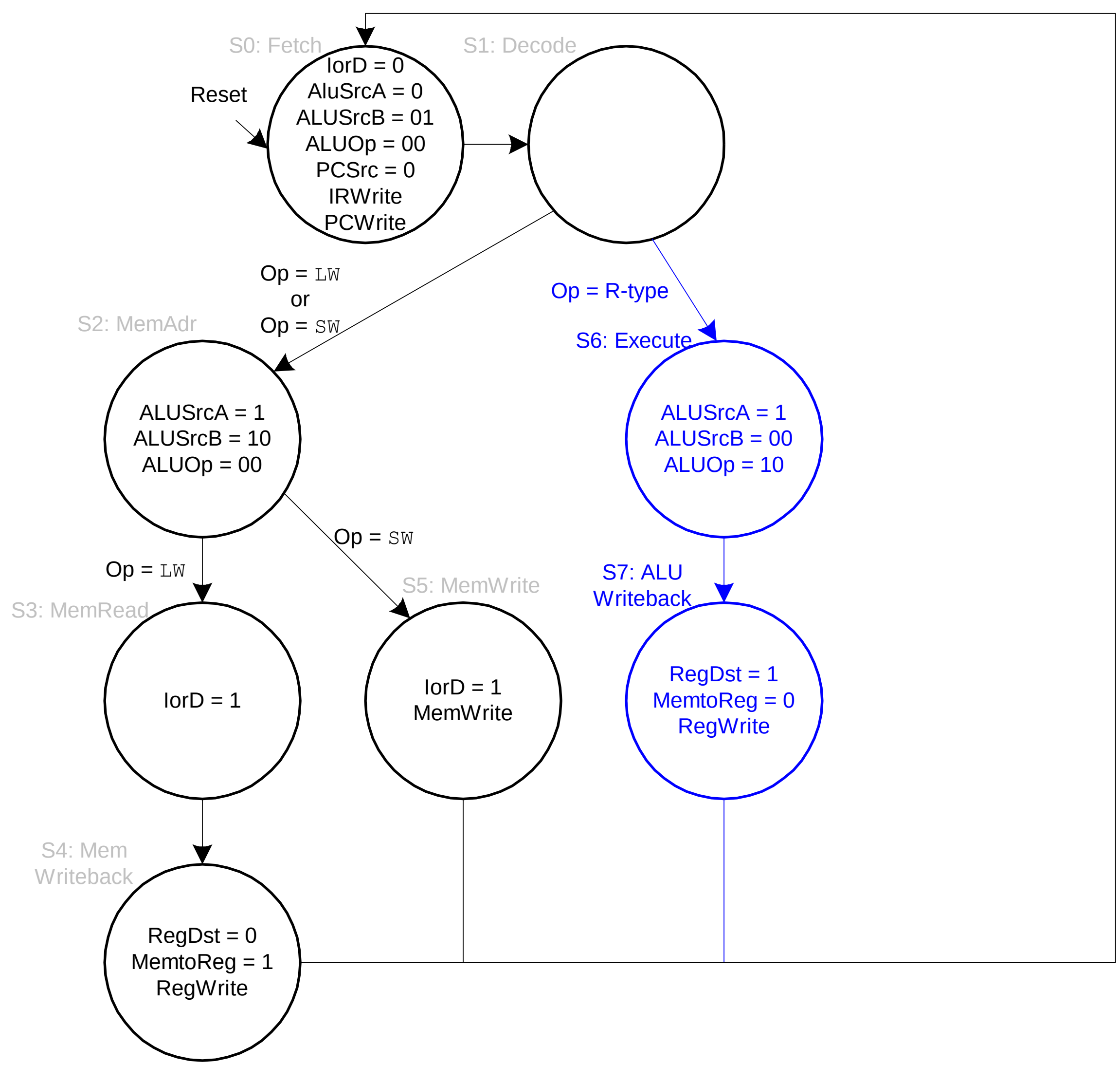
Main Controller FSM: `lw`



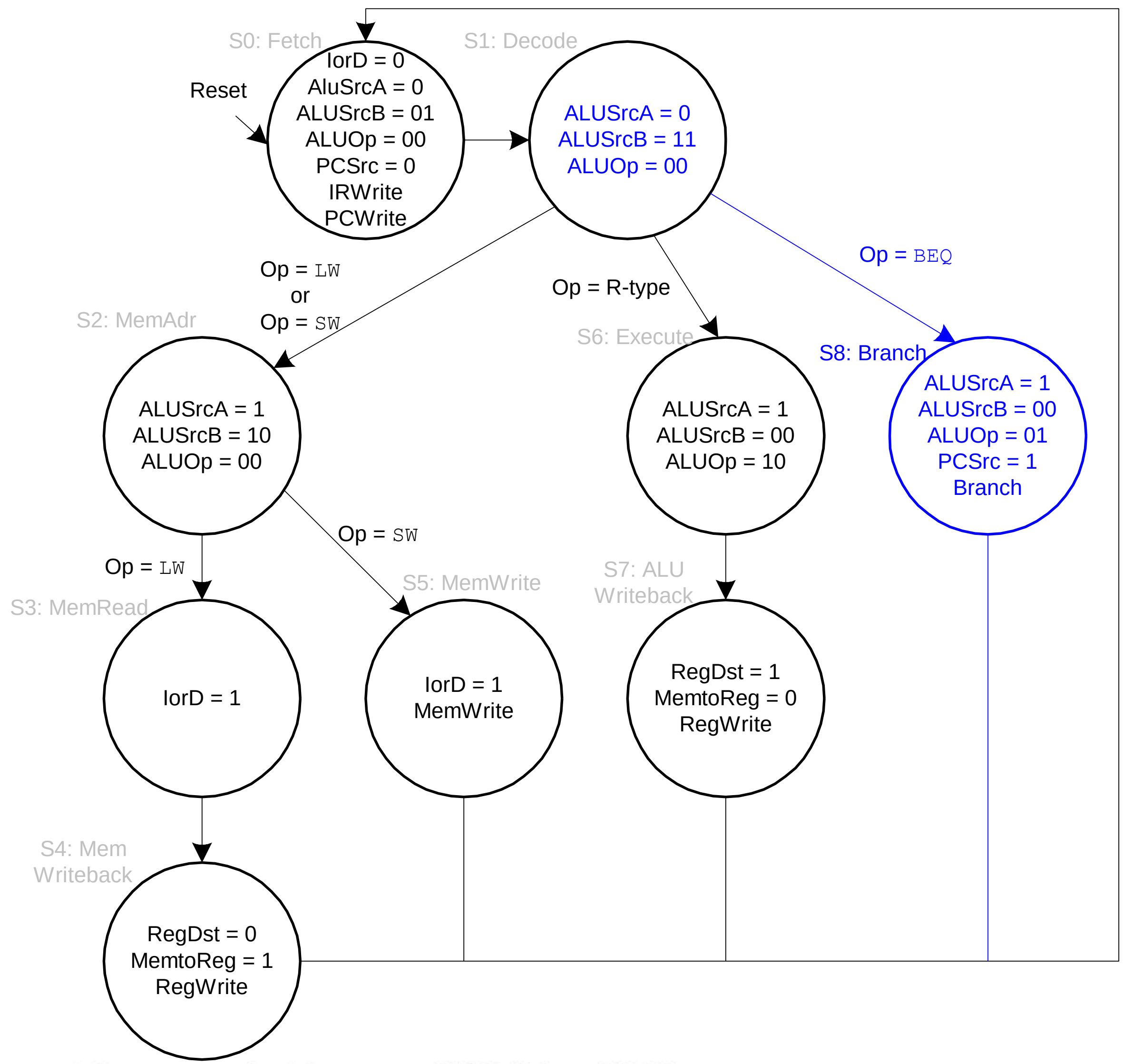
Main Controller FSM: `sw`



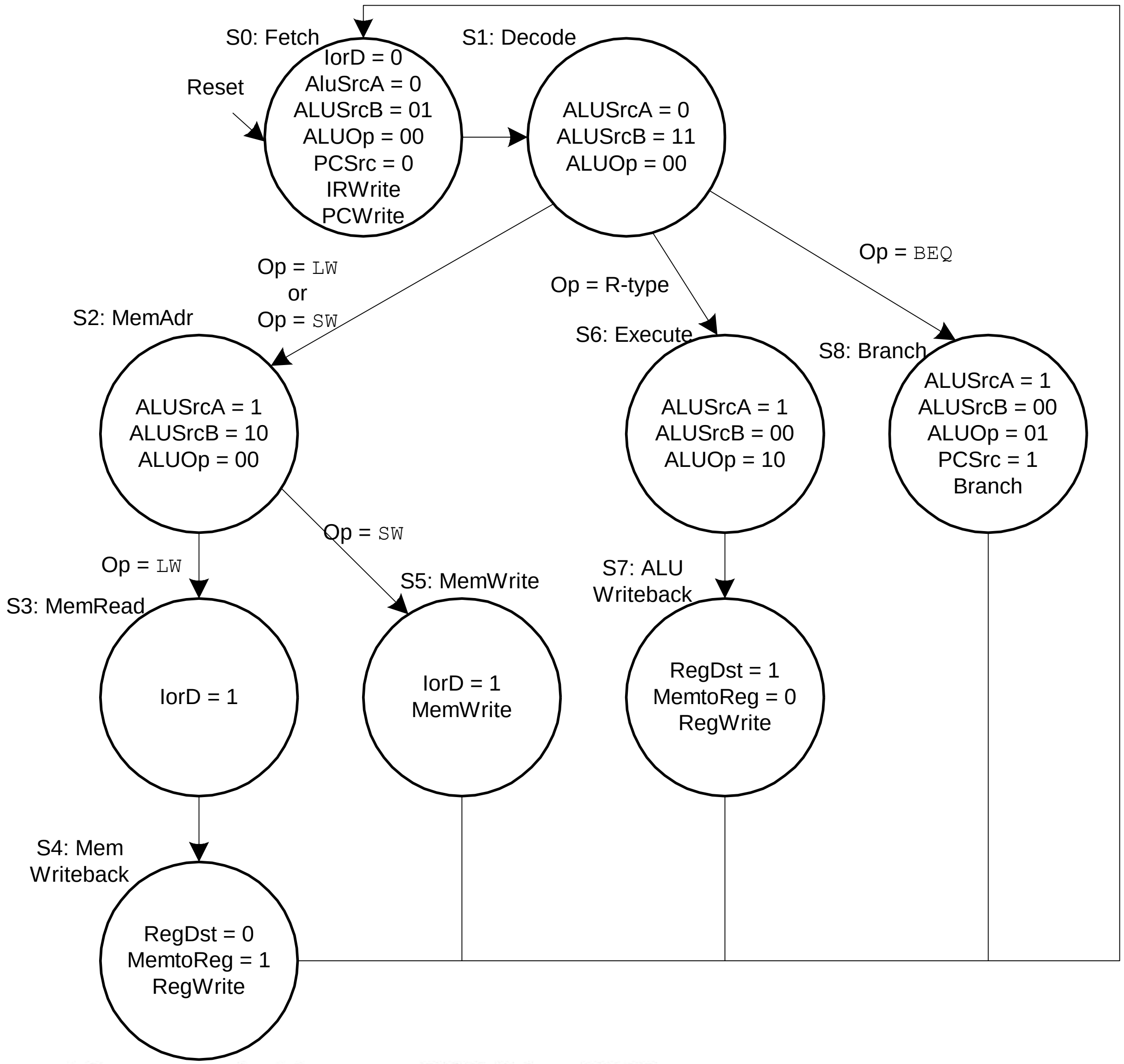
Main Controller FSM: R-Type



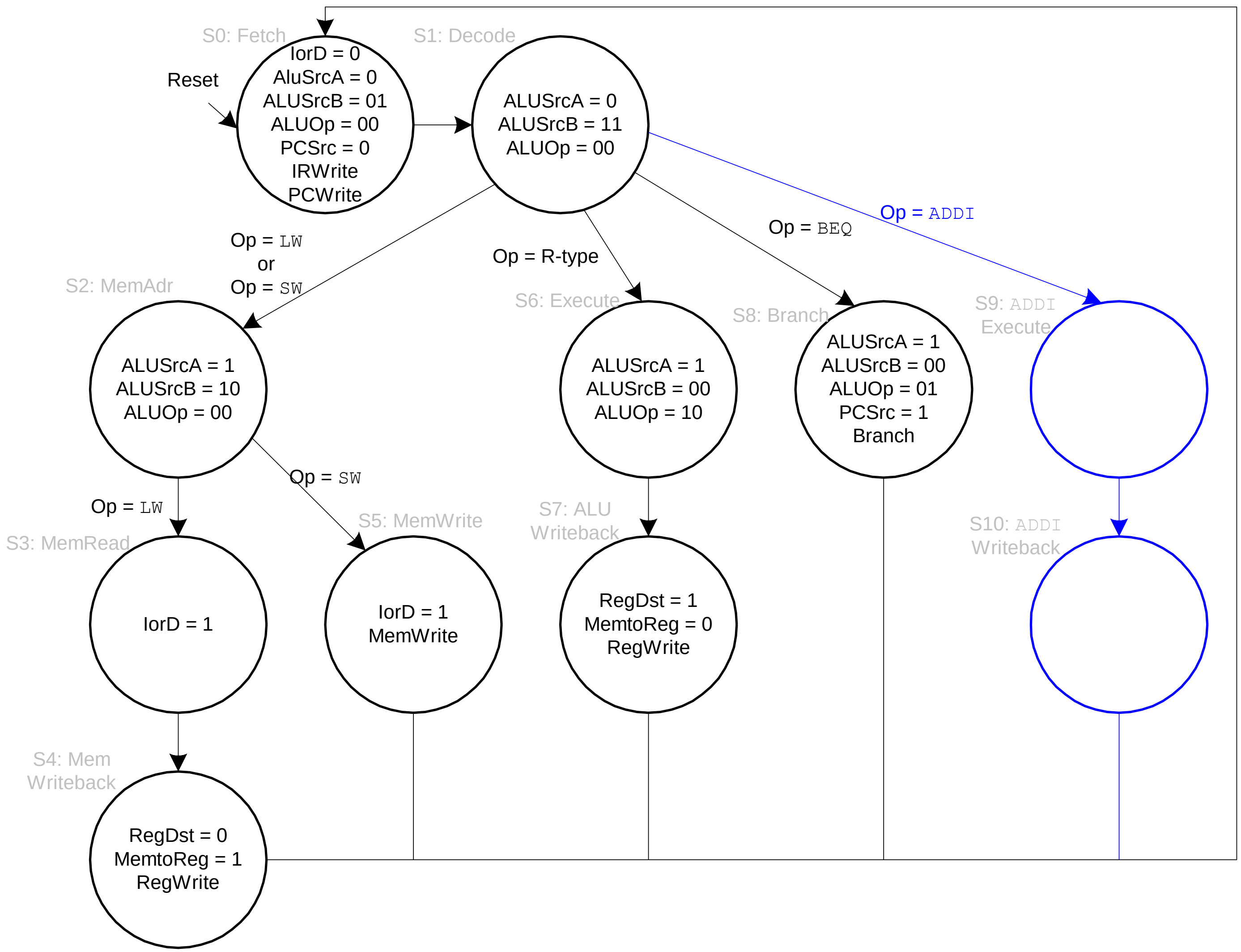
Main Controller FSM: beq



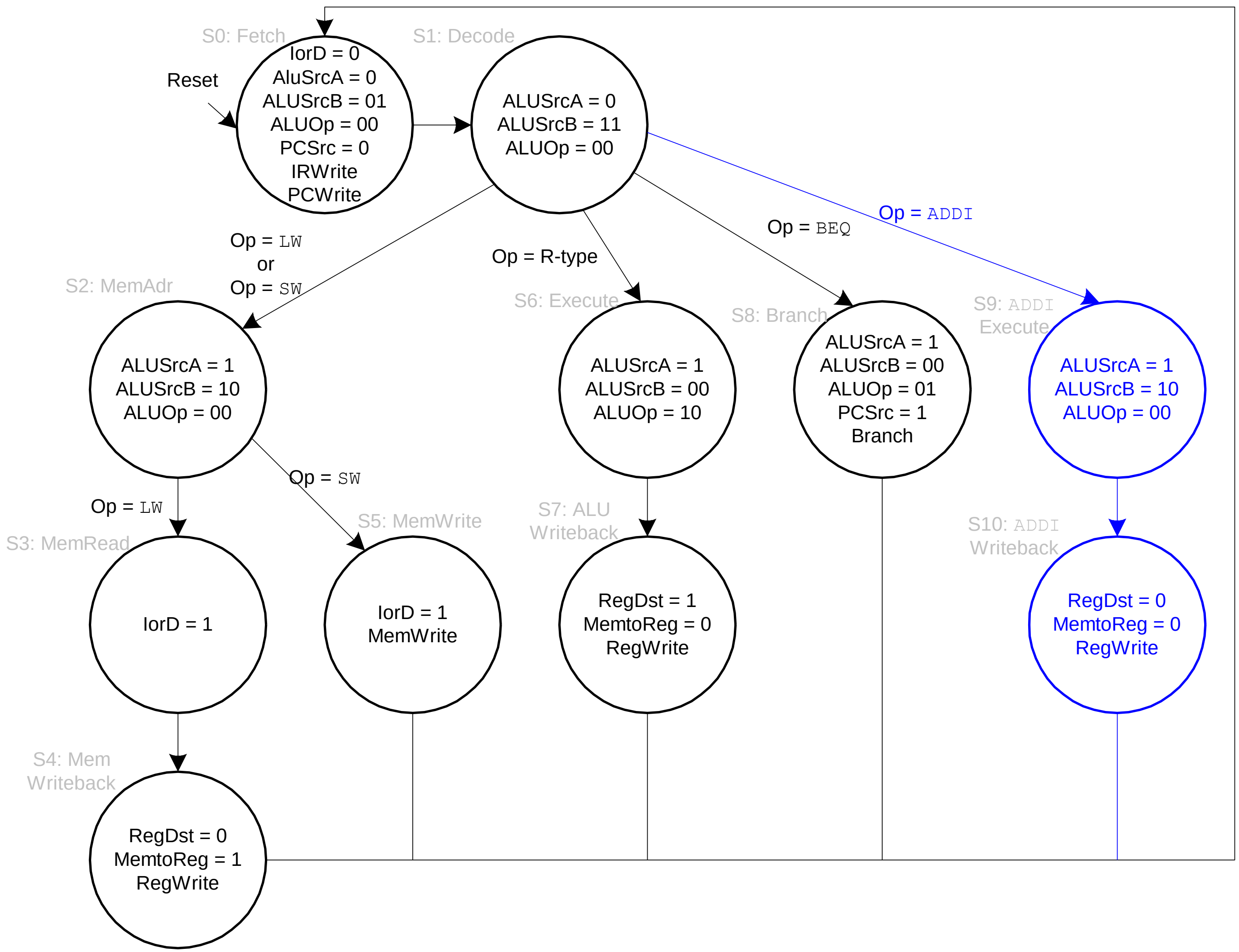
Multicycle Controller FSM



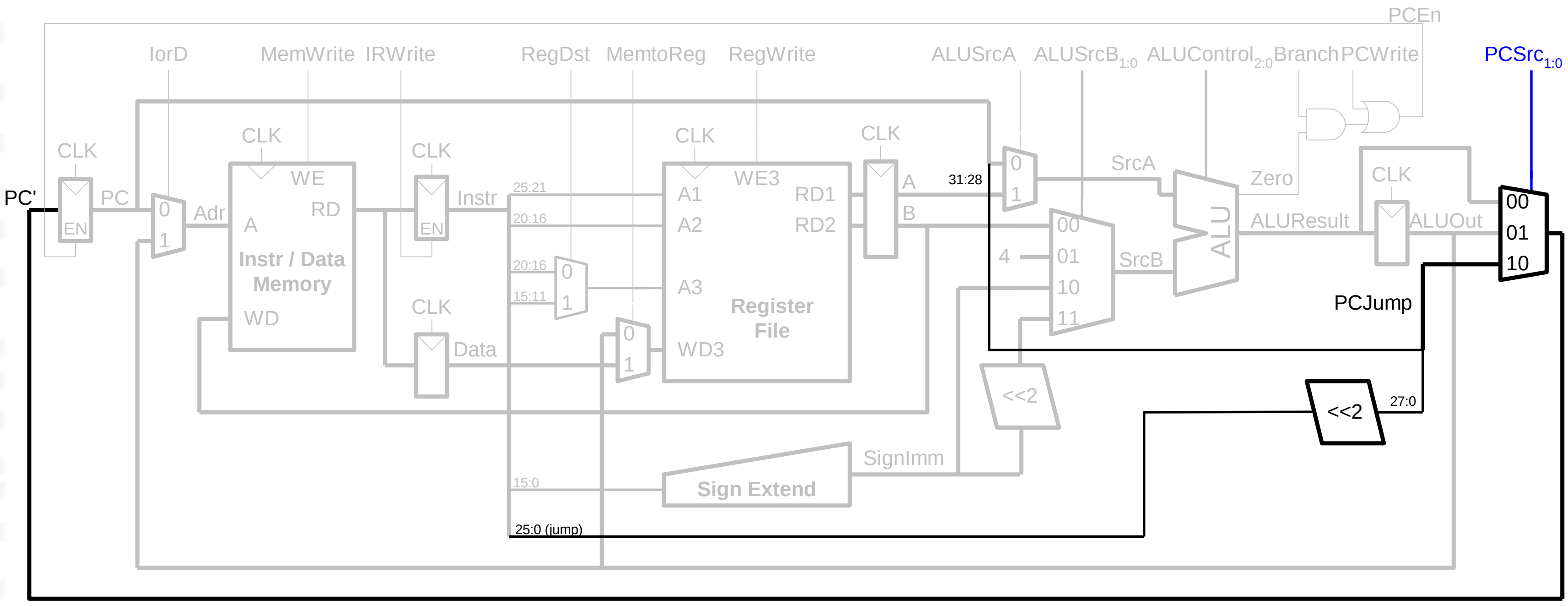
Extended Functionality: addi



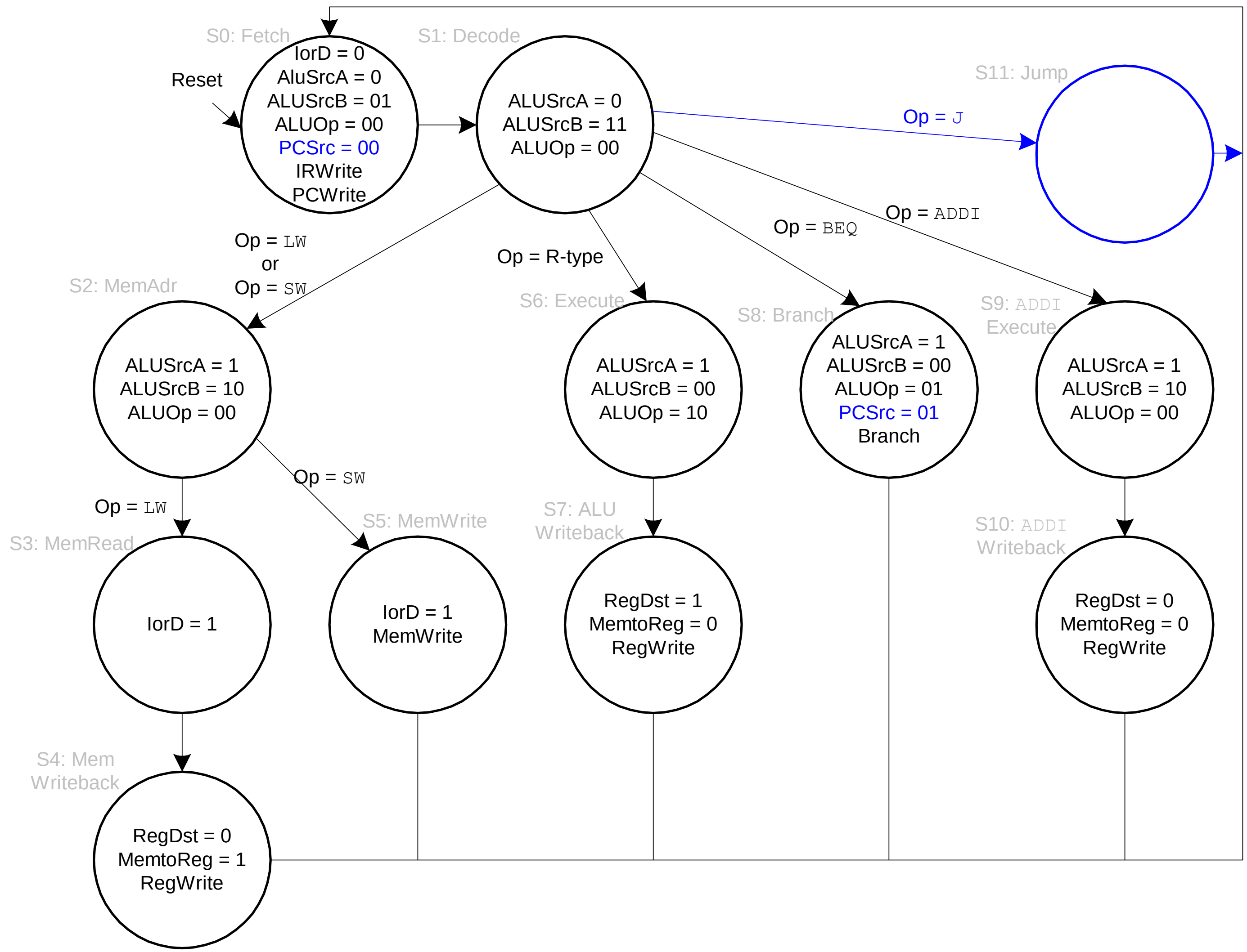
Main Controller FSM: addi



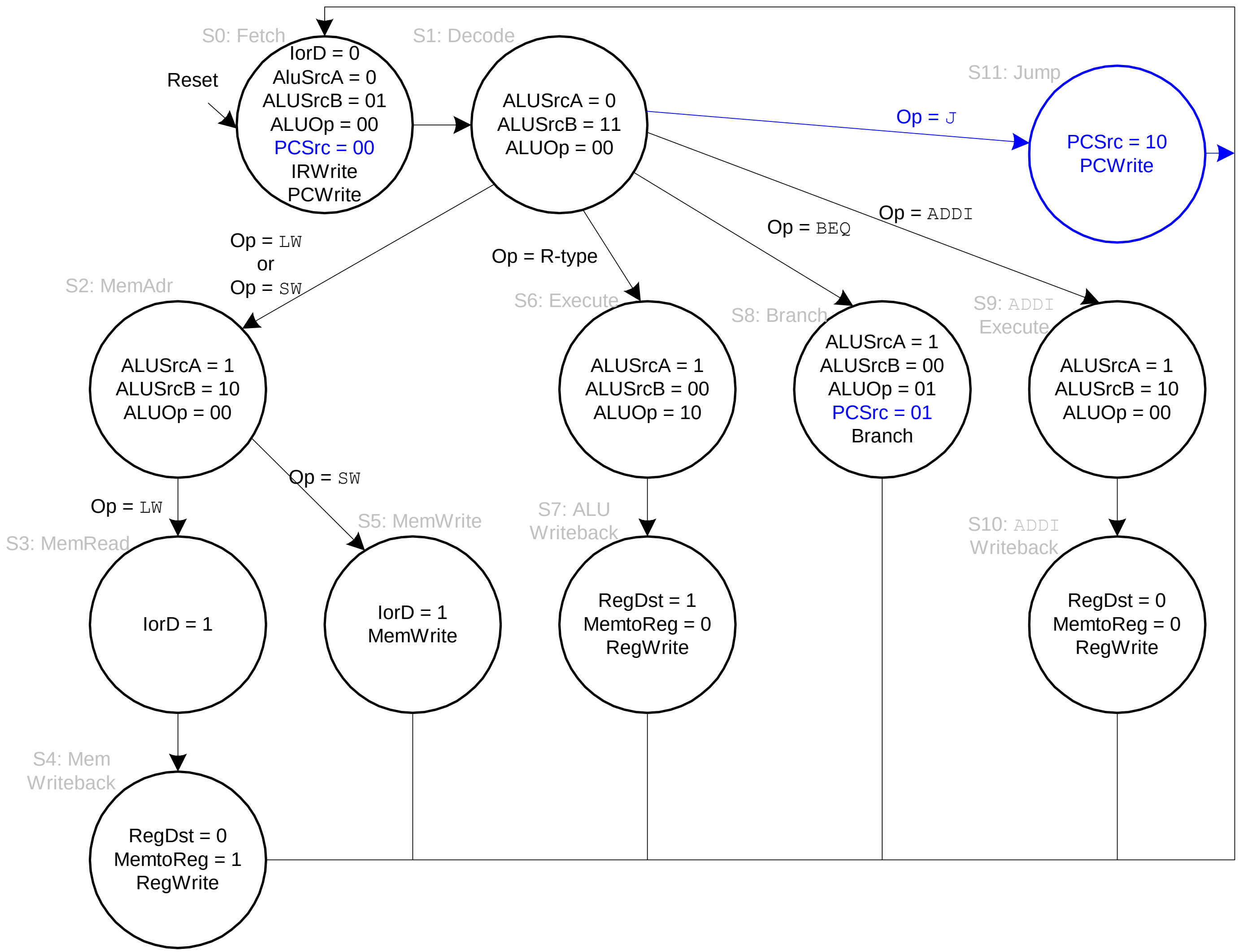
Extended Functionality: j



Main Controller FSM: j



Main Controller FSM: j



ELSEVIER

Multicycle Processor Performance

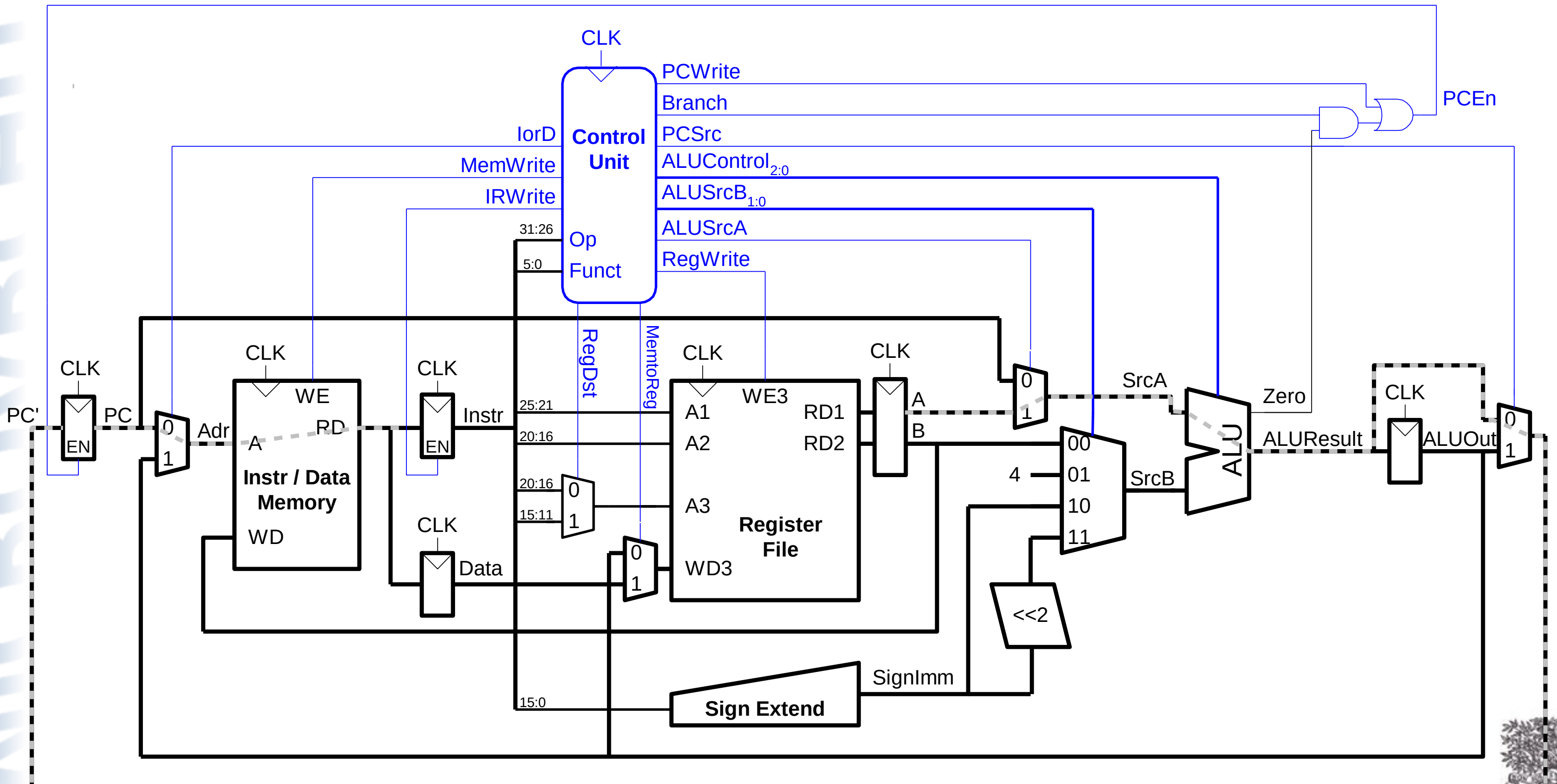
- Instructions take different number of cycles:
  - 3 cycles: `beq`, `j`
  - 4 cycles: R-Type, `sw`, `addi`
  - 5 cycles: `lw`
- CPI is weighted average
- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

Average CPI = (0.11 + 0.02)(3) + (0.52 + 0.10)(4) + (0.25)(5) **= 4.12**

Multicycle Processor Performance

Multicycle critical path:

$$T_c = t_{pcq} + t_{\mathrm{mux}} + \max(t_{\mathrm{ALU}} + t_{\mathrm{mux}}, t_{\mathrm{mem}}) + t_{\mathrm{setup}}$$



Multicycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	$t_{RF\text{read}}$	150
Register file setup	$t_{RF\text{setup}}$	20

$T_c = ?$

Multicycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	$t_{RF\mathrm{read}}$	150
Register file setup	$t_{RF\mathrm{setup}}$	20

$$\begin{aligned} \boldsymbol{T_c} &= t_{pcq\_PC} + t_{\mathrm{mux}} + \max(t_{\mathrm{ALU}} + t_{\mathrm{mux}}, t_{\mathrm{mem}}) + t_{\mathrm{setup}} \\ &= t_{pcq\_PC} + t_{\mathrm{mux}} + t_{\mathrm{mem}} + t_{\mathrm{setup}} \\ &= [30 + 25 + 250 + 20]\ \mathrm{ps} \\ &= \mathbf{325\ ps} \end{aligned}$$

MICROARCHITECTURE

Multicycle Performance Example

Program with 100 billion instructions

Execution Time = ?

Multicycle Performance Example

Program with 100 billion instructions

Execution Time = (# instructions) × CPI × T_c

$$= (100 \times 10^{9})(4.12)(325 \times 10^{-12})$$

$$= \mathbf{133.9\ seconds}$$

This is **slower** than the single-cycle processor (92.5 seconds). Why?

Multicycle Performance Example

Program with 100 billion instructions

Execution Time = (# instructions) × CPI × T_c

$= (100 \times 10^{9})(4.12)(325 \times 10^{-12})$

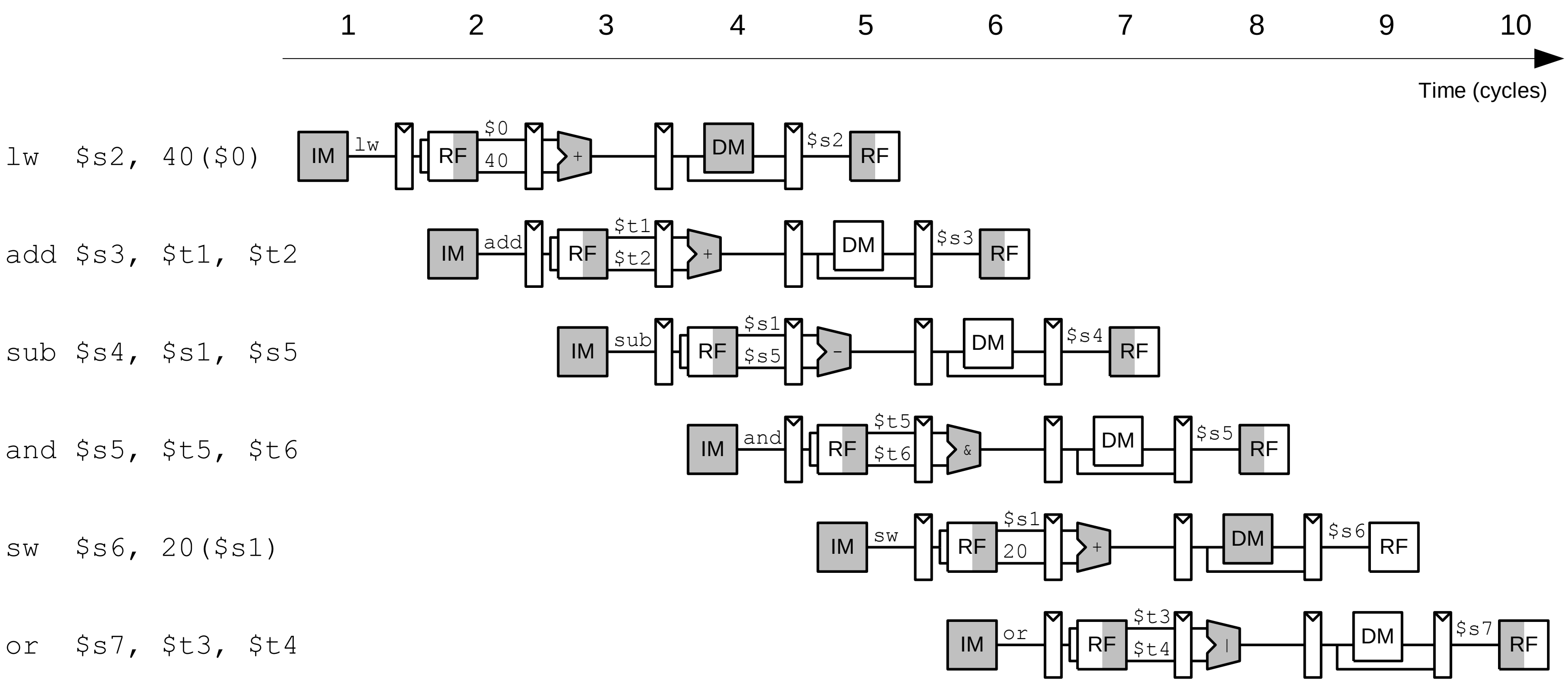
= **133.9 seconds**

This is **slower** than the single-cycle processor (92.5 seconds). Why?

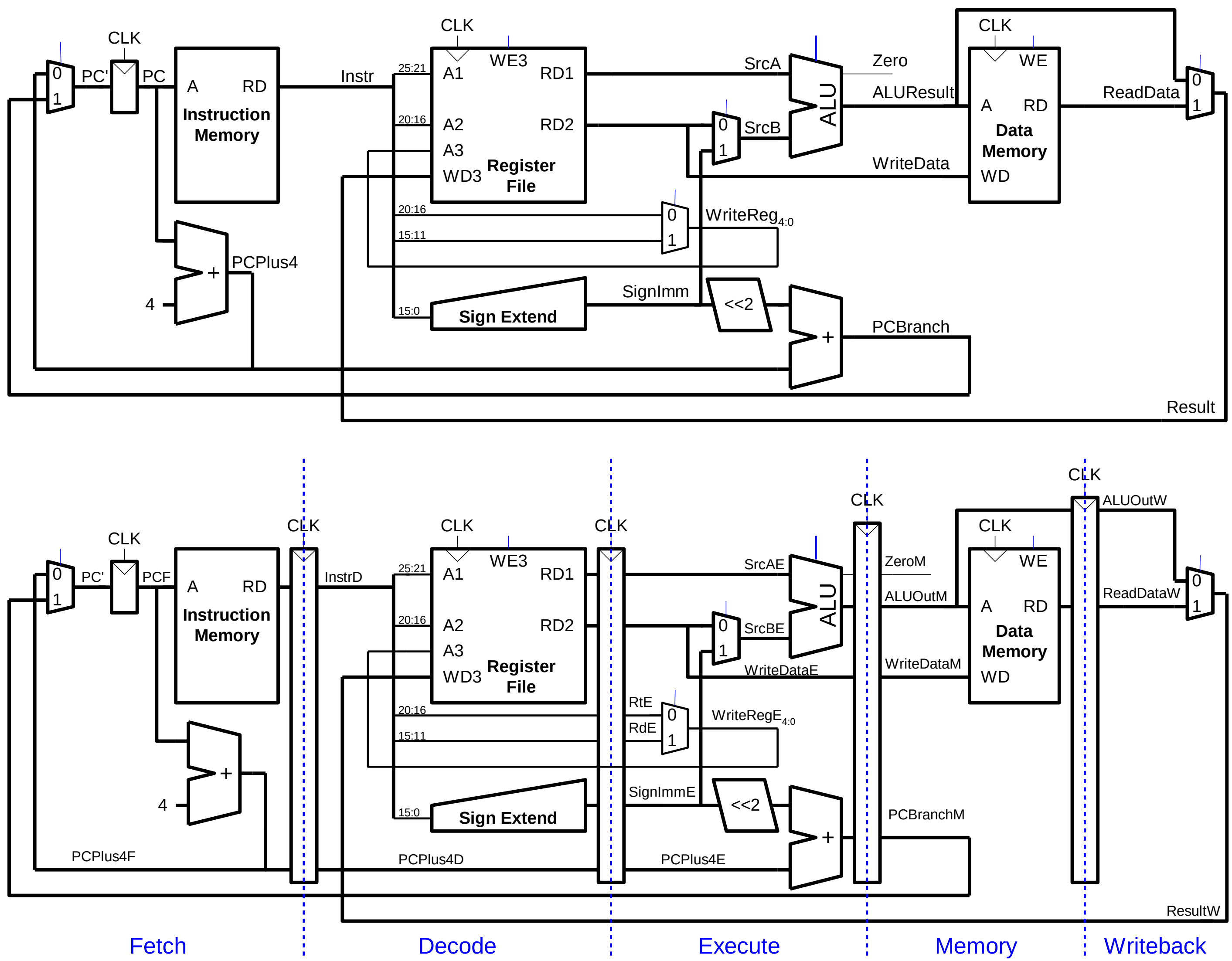
– Not all steps same length

– Sequencing overhead for each step ($t_{pcq} + t_{setup} = 50$ ps)

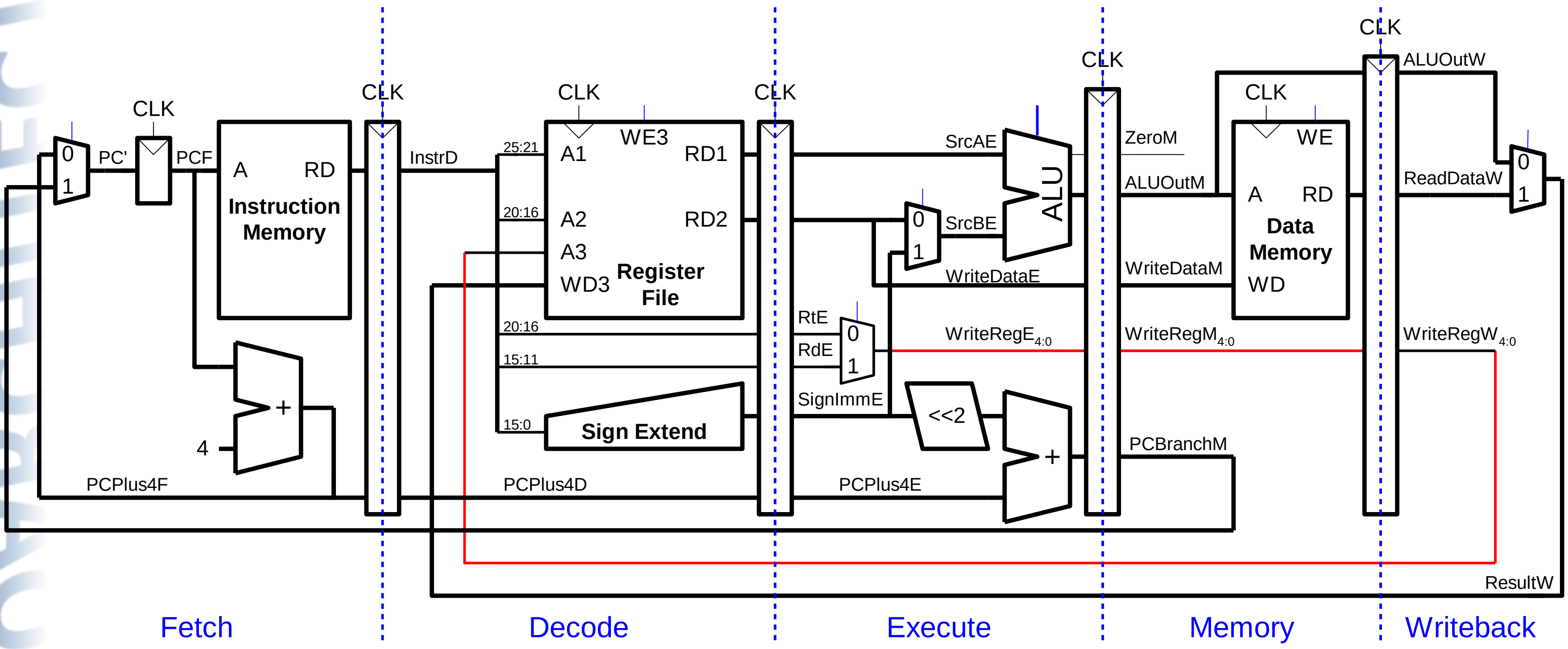
Pipelined Processor Abstraction



Single-Cycle & Pipelined Datapath

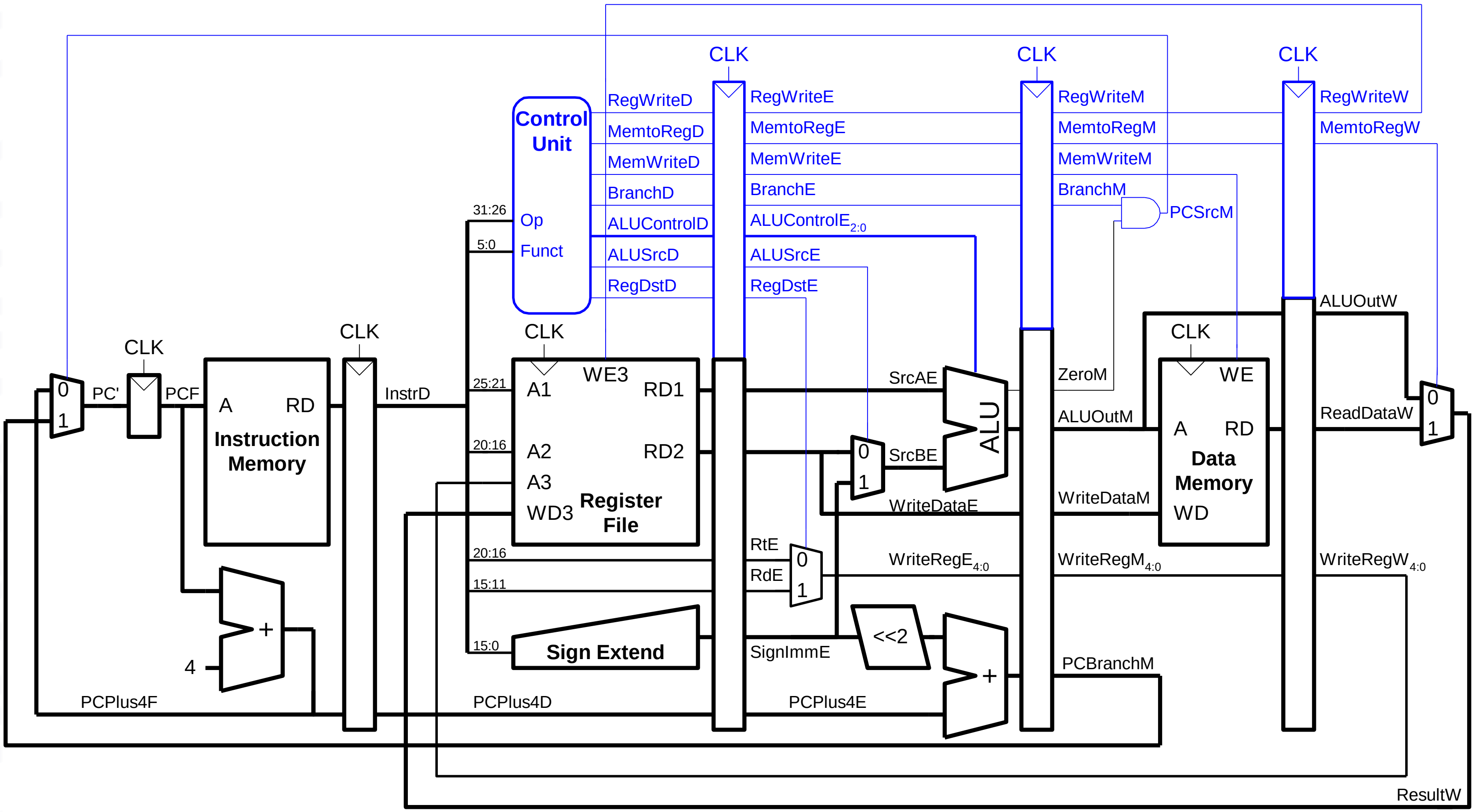


Corrected Pipelined Datapath



WriteReg **must arrive at same time as** ***Result***

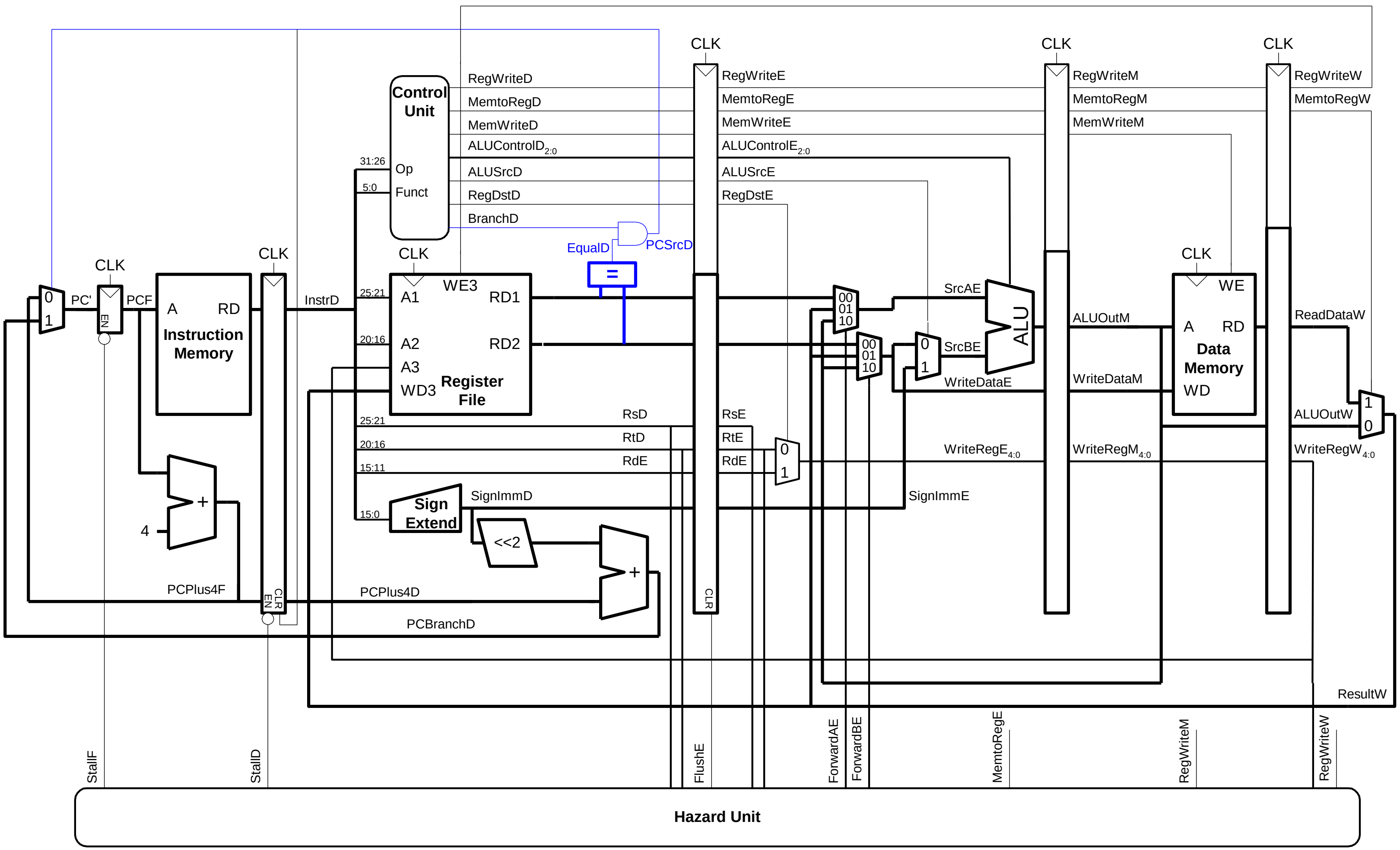
Pipelined Processor Control



- **Same control unit as single-cycle processor**
- **Control delayed to proper pipeline stage**

ELSEVIER

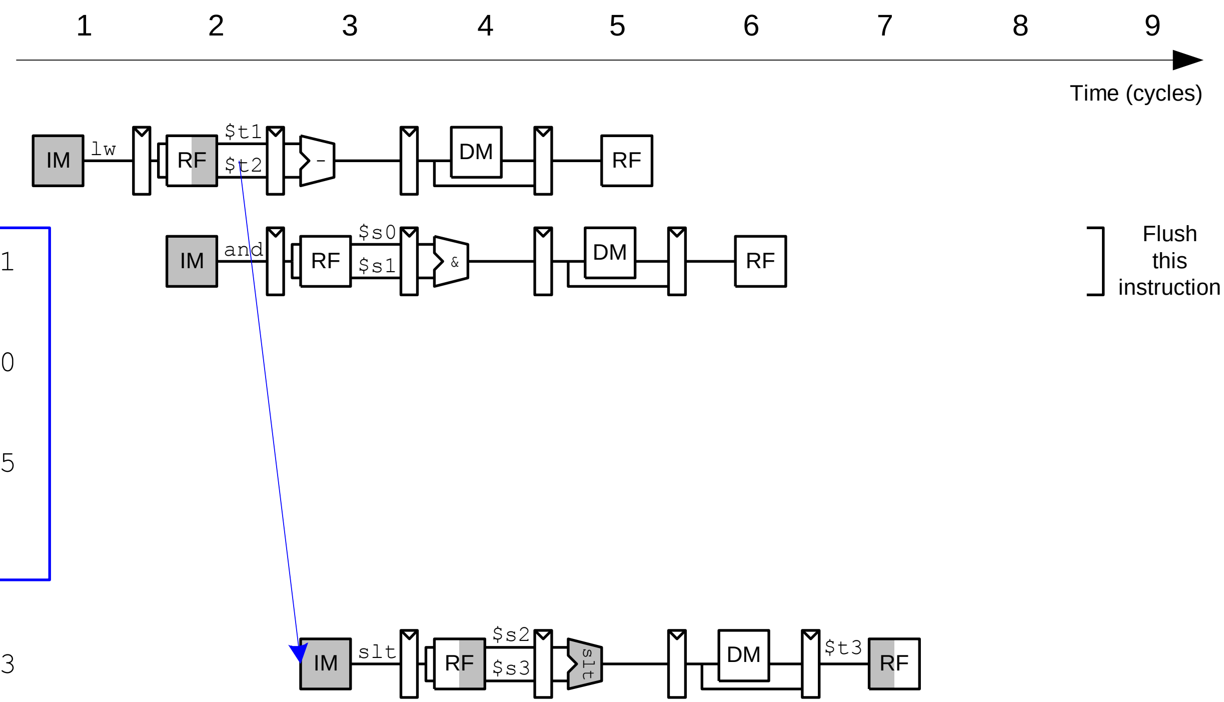
Early Branch Resolution



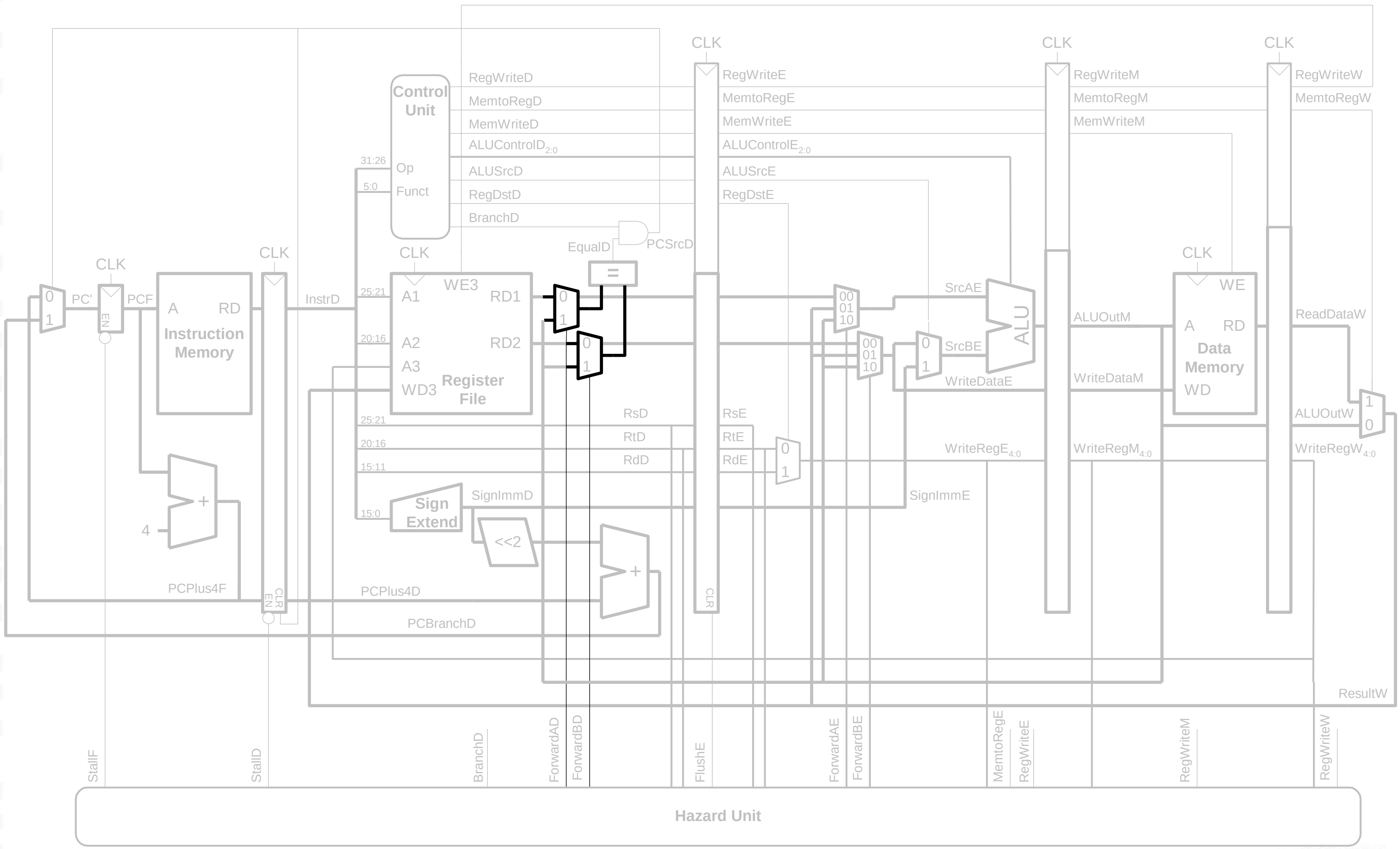
Introduced another data hazard in Decode stage

Early Branch Resolution

```
20   beq $t1, $t2, 40

24   and $t0, $s0, $s1

28   or  $t1, $s4, $s0

2C   sub $t2, $s0, $s5

30   ...
...
64   slt $t3, $s2, $s3
```



Handling Data & Control Hazards



Control Forwarding & Stalling Logic

- **Forwarding logic:**

```
ForwardAD = (rsD !=0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD !=0) AND (rtD == WriteRegM) AND RegWriteM
```

- **Stalling logic:**

```
branchstall = BranchD AND
  [RegWriteE AND ((WriteRegE == rsD) OR (WriteRegE == rtD))
                 OR
  [MemtoRegM AND ((WriteRegM == rsD) OR (WriteRegM == rtD))]

StallF = StallD = FlushE = lwstall OR branchstall
```

MICROARCHITECTURE

ELSEVIER

Branch Prediction

- Guess whether branch will be taken
  - Backward branches are usually taken (loops)
  - Consider history to improve guess
- Good prediction reduces fraction of branches requiring a flush

Pipelined Performance Example

- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction
  - 25% of branches mispredicted
  - All jumps flush next instruction
- **What is the average CPI?**

Pipelined Performance Example

- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction
  - 25% of branches mispredicted
  - All jumps flush next instruction
- **What is the average CPI?**
  - Load/Branch CPI = 1 when no stalling, 2 when stalling
  - $CPI_{lw} = 1(0.6) + 2(0.4) = 1.4$
  - $CPI_{beq} = 1(0.75) + 2(0.25) = 1.25$

Average CPI = (0.25)(1.4) + (0.1)(1) + (0.11)(1.25) + (0.02)(2) + (0.52)(1)

= 1.15

Pipelined Performance

- Pipelined processor critical path:

$$T_c = \max \{$$

$$t_{pcq} + t_{\text{mem}} + t_{\text{setup}}$$

$$2(t_{\text{RFread}} + t_{\text{mux}} + t_{\text{eq}} + t_{\text{AND}} + t_{\text{mux}} + t_{\text{setup}})$$

$$t_{pcq} + t_{\text{mux}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{setup}}$$

$$t_{pcq} + t_{\text{memwrite}} + t_{\text{setup}}$$

$$2(t_{pcq} + t_{\text{mux}} + t_{\text{RFwrite}}) \}$$

Pipelined Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	$t_{RF\text{read}}$	150
Register file setup	$t_{RF\text{setup}}$	20
Equality comparator	t_{eq}	40
AND gate	t_{AND}	15
Memory write	t_{memwrite}	220
Register file write	$t_{RF\text{write}}$	100

$$T_c = 2(t_{\text{RFread}} + t_{\text{mux}} + t_{\text{eq}} + t_{\text{AND}} + t_{\text{mux}} + t_{\text{setup}})$$
$$= 2[150 + 25 + 40 + 15 + 25 + 20] \text{ ps} = \mathbf{550\ ps}$$

Pipelined Performance Example

Program with 100 billion instructions

Execution Time = (# instructions) × CPI × T_c

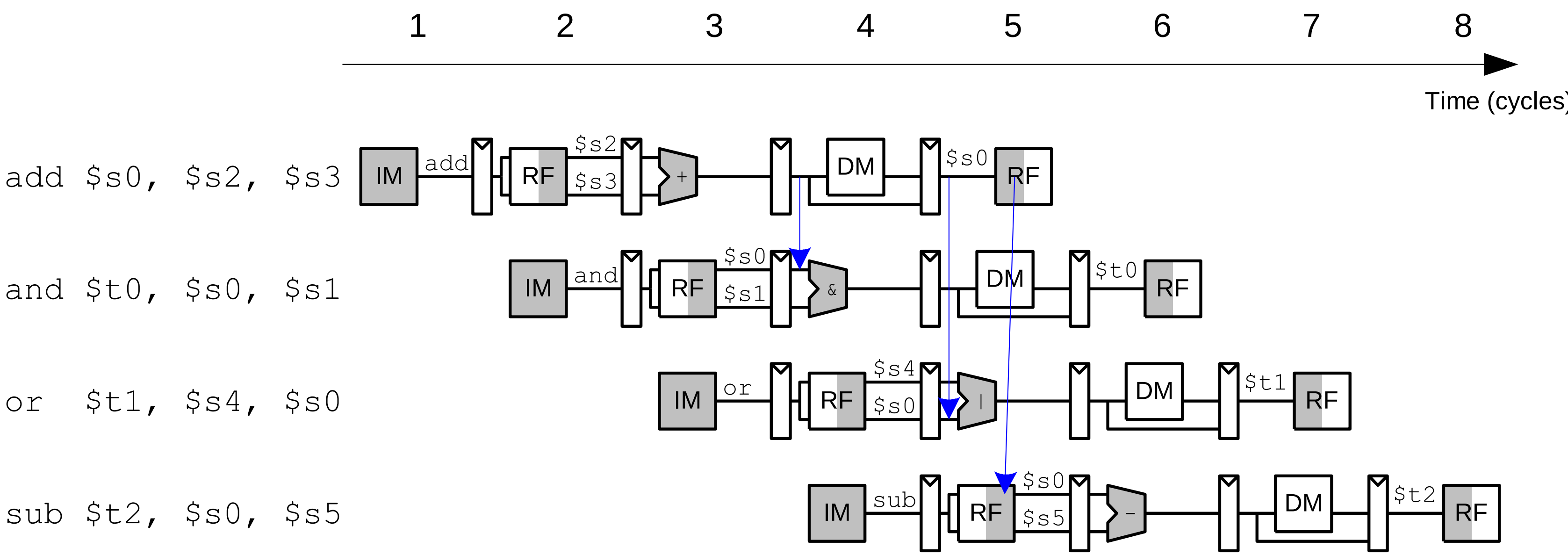
= $(100 \times 10^9)(1.15)(550 \times 10^{-12})$

= 63 seconds

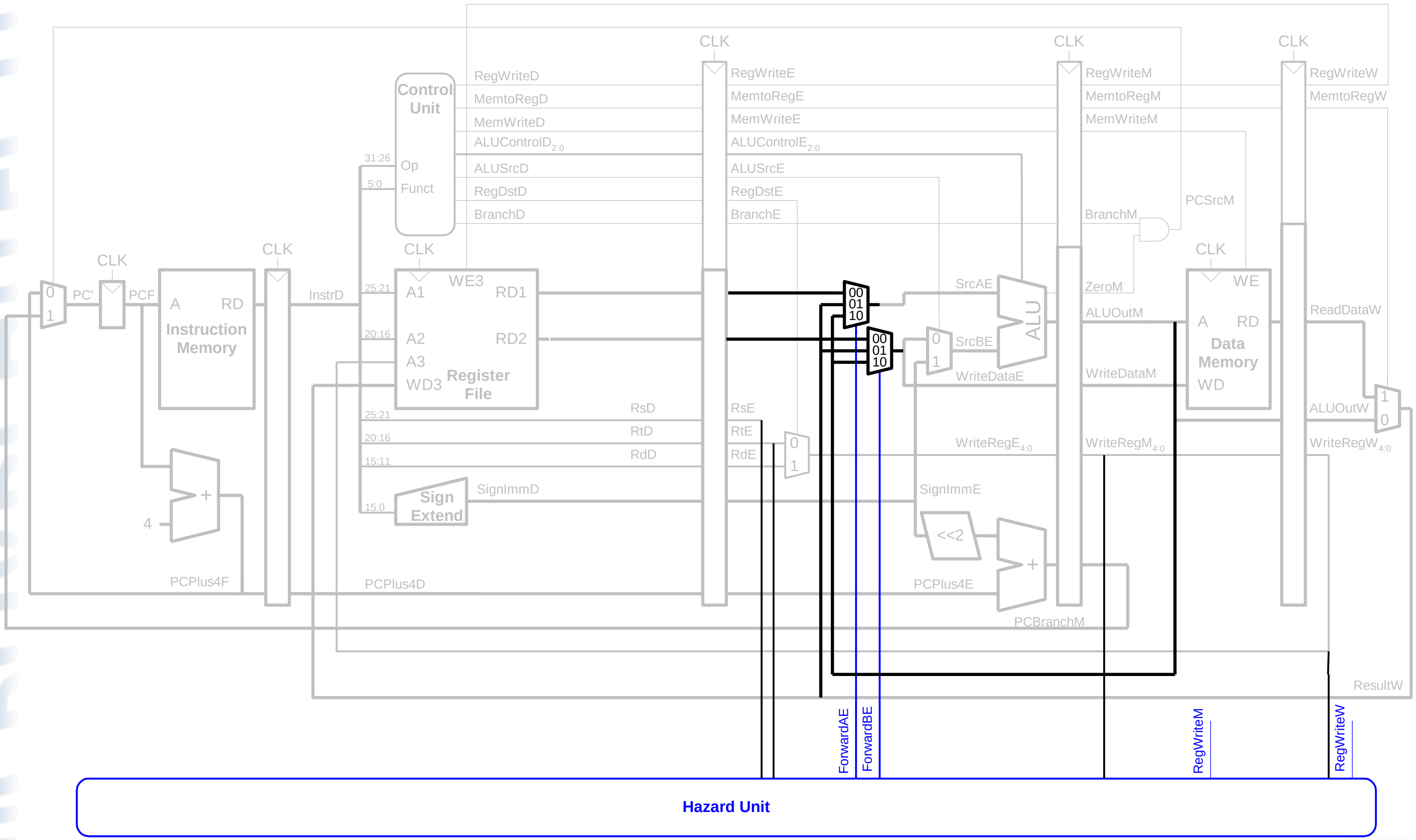
Processor Performance Comparison

Processor	Execution Time (seconds)	Speedup (single-cycle as baseline)
Single-cycle	92.5	1
Multicycle	133	0.70
Pipelined	63	1.47

Data Forwarding



Data Forwarding



Data Forwarding

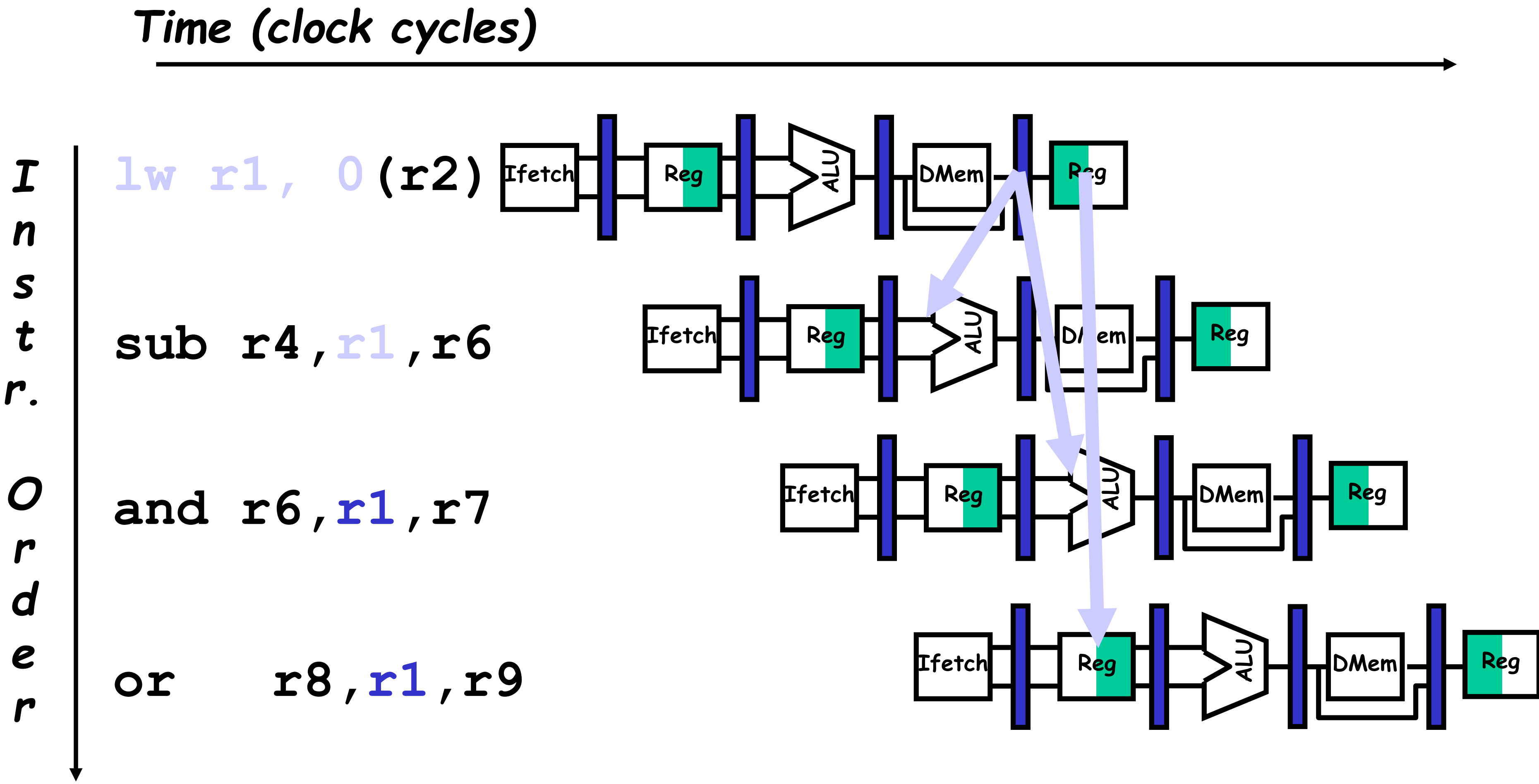
- Forward to Execute stage from either:
  - Memory stage or
  - Writeback stage
- Forwarding logic for *ForwardAE*:

```
if      ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
    then    ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
    then    ForwardAE = 01
else        ForwardAE = 00
```

Forwarding logic for *ForwardBE* same, but replace *rsE* with *rtE*

Data Hazards

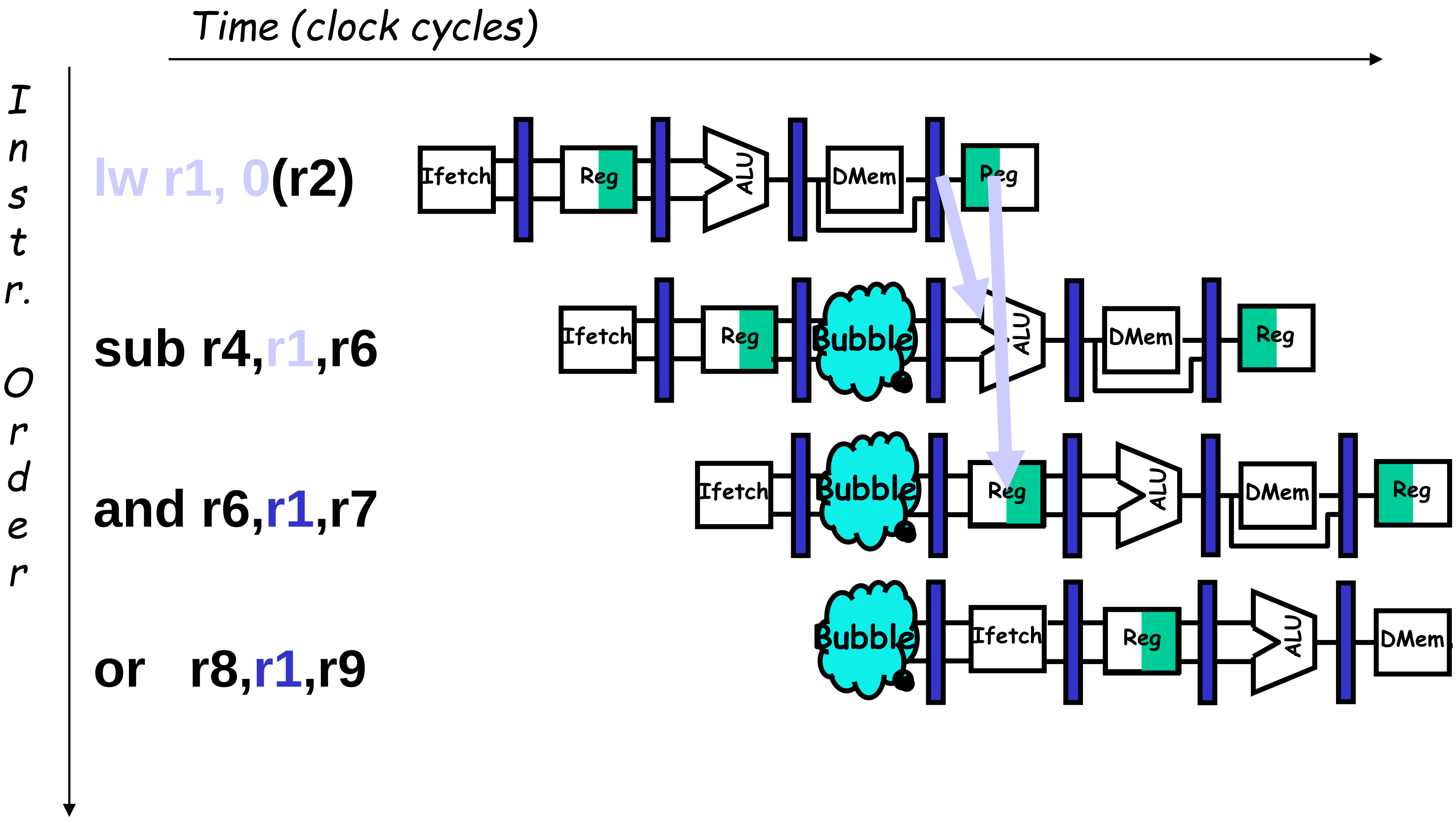
The data isn't loaded until after the MEM stage.



There are some instances where hazards occur, even with forwarding.

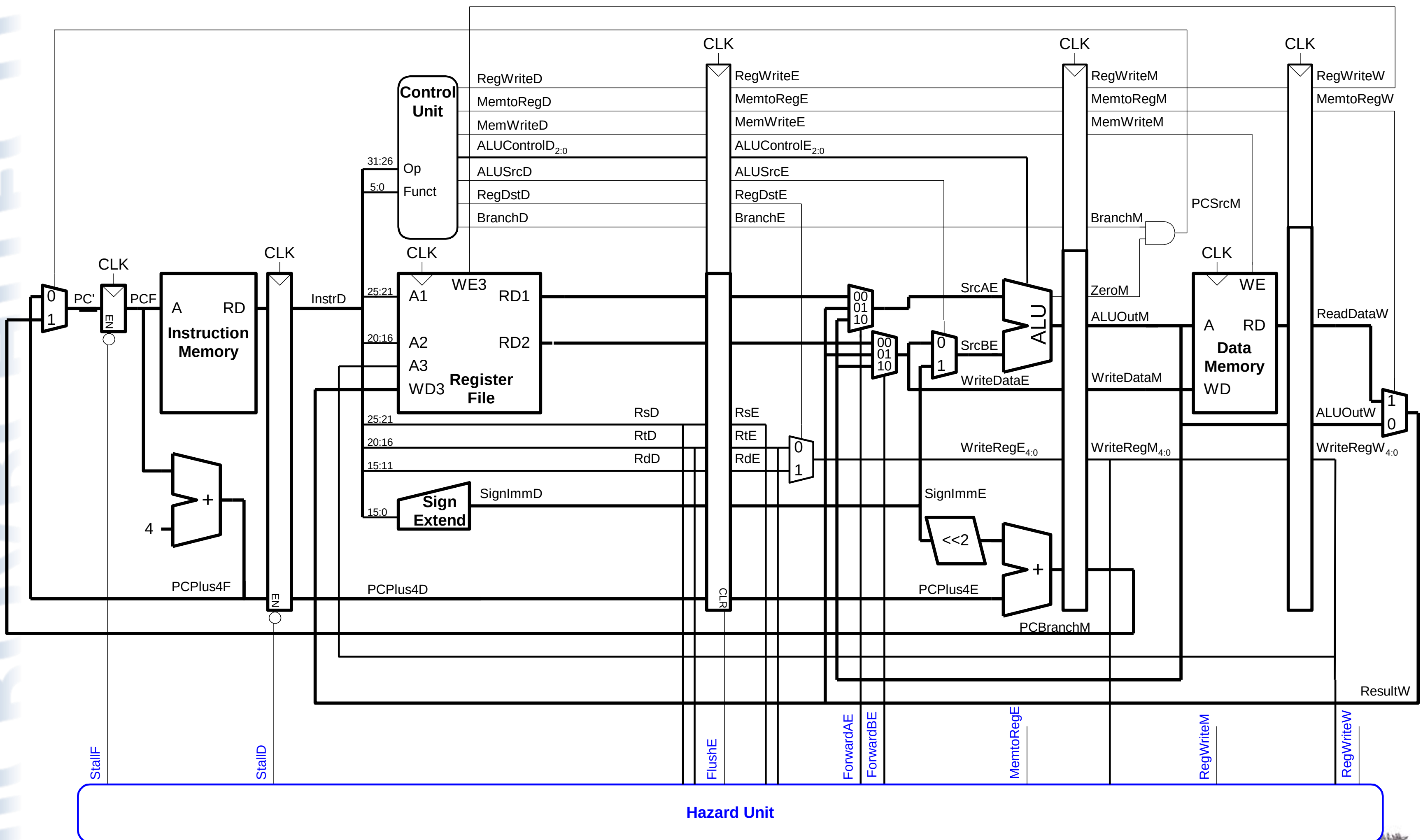
Data Hazards

The stall is necessary as shown here.



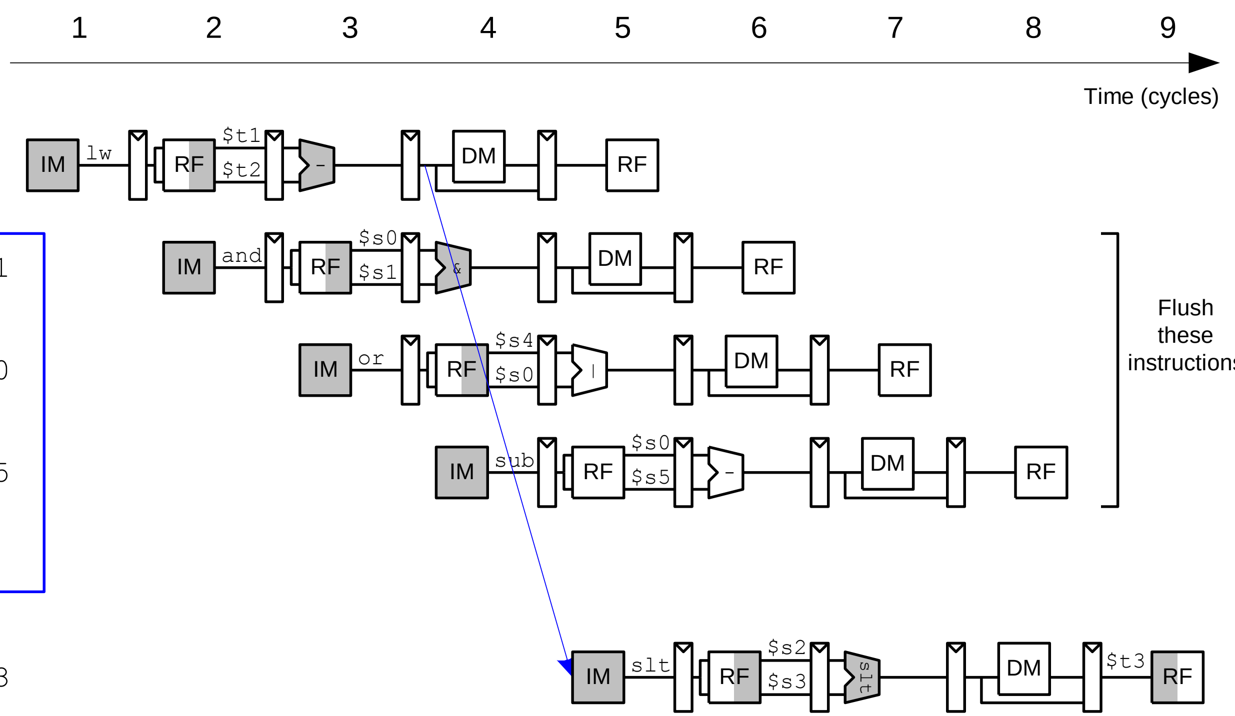
There are some instances where hazards occur, even with forwarding.

Control Hazards: Original Pipeline

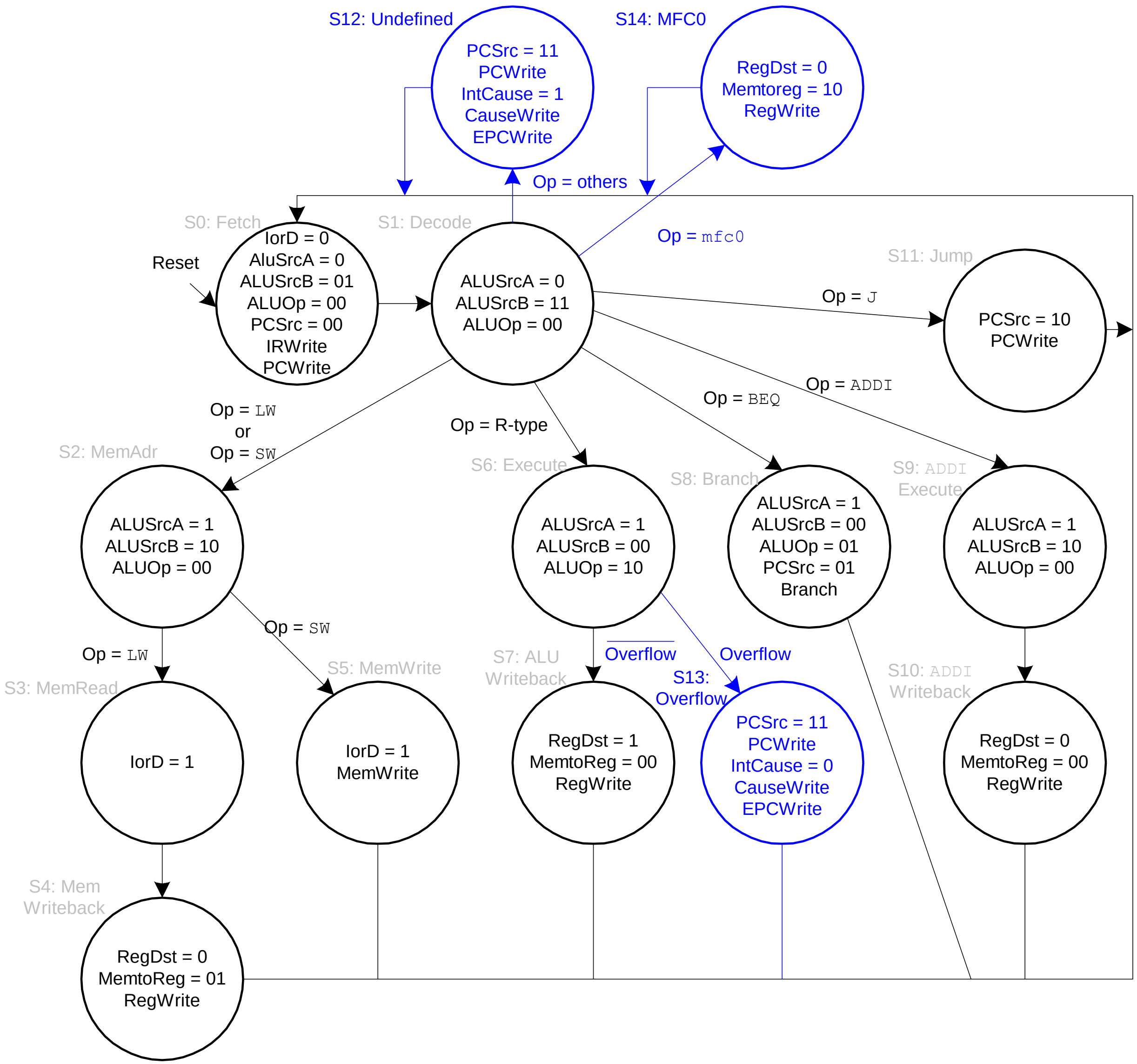


Control Hazards

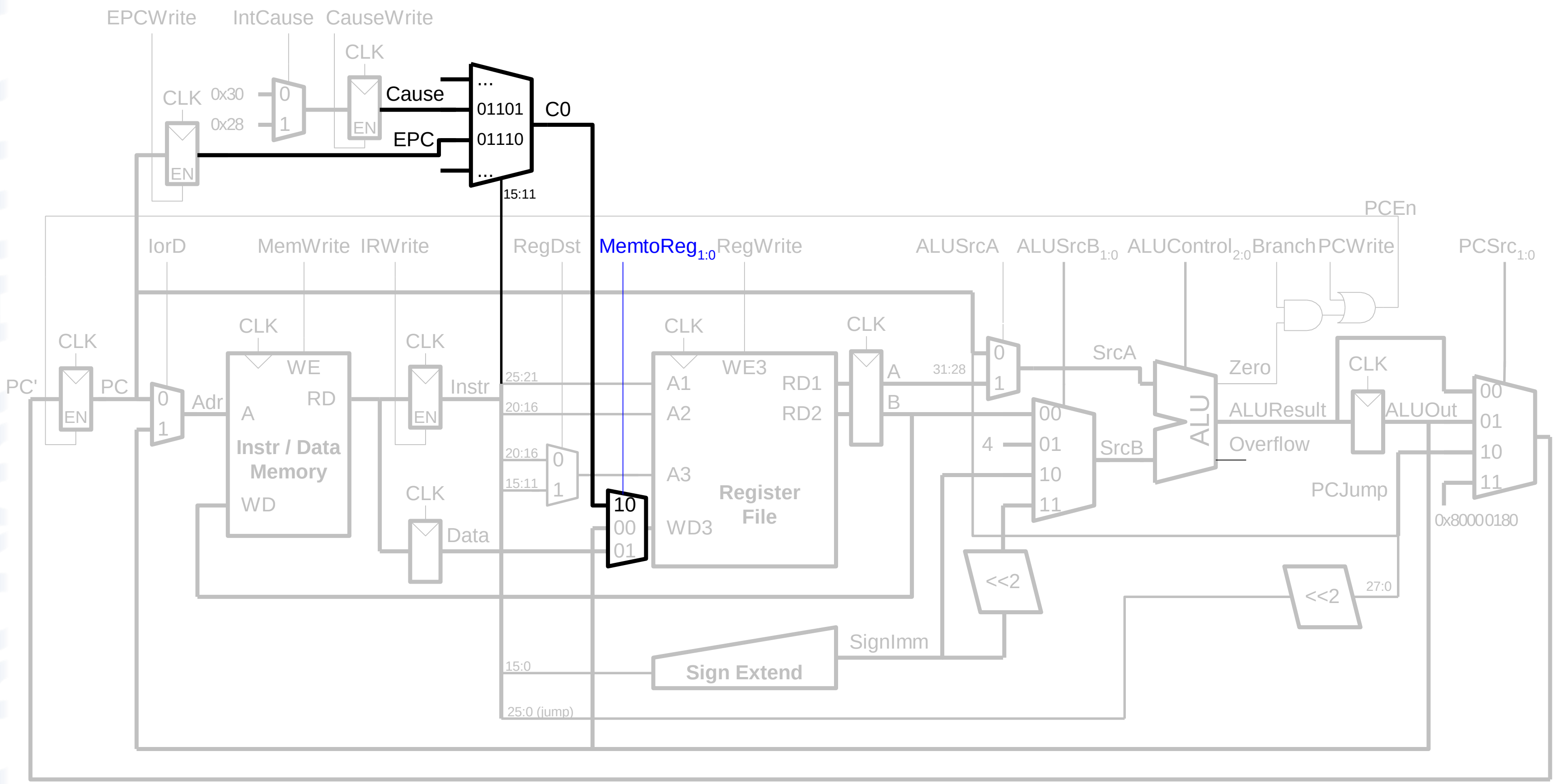
```
20    beq $t1, $t2, 40
24    and $t0, $s0, $s1
28    or  $t1, $s4, $s0
2C    sub $t2, $s0, $s5
30    ...
...
64    slt $t3, $s2, $s3
```



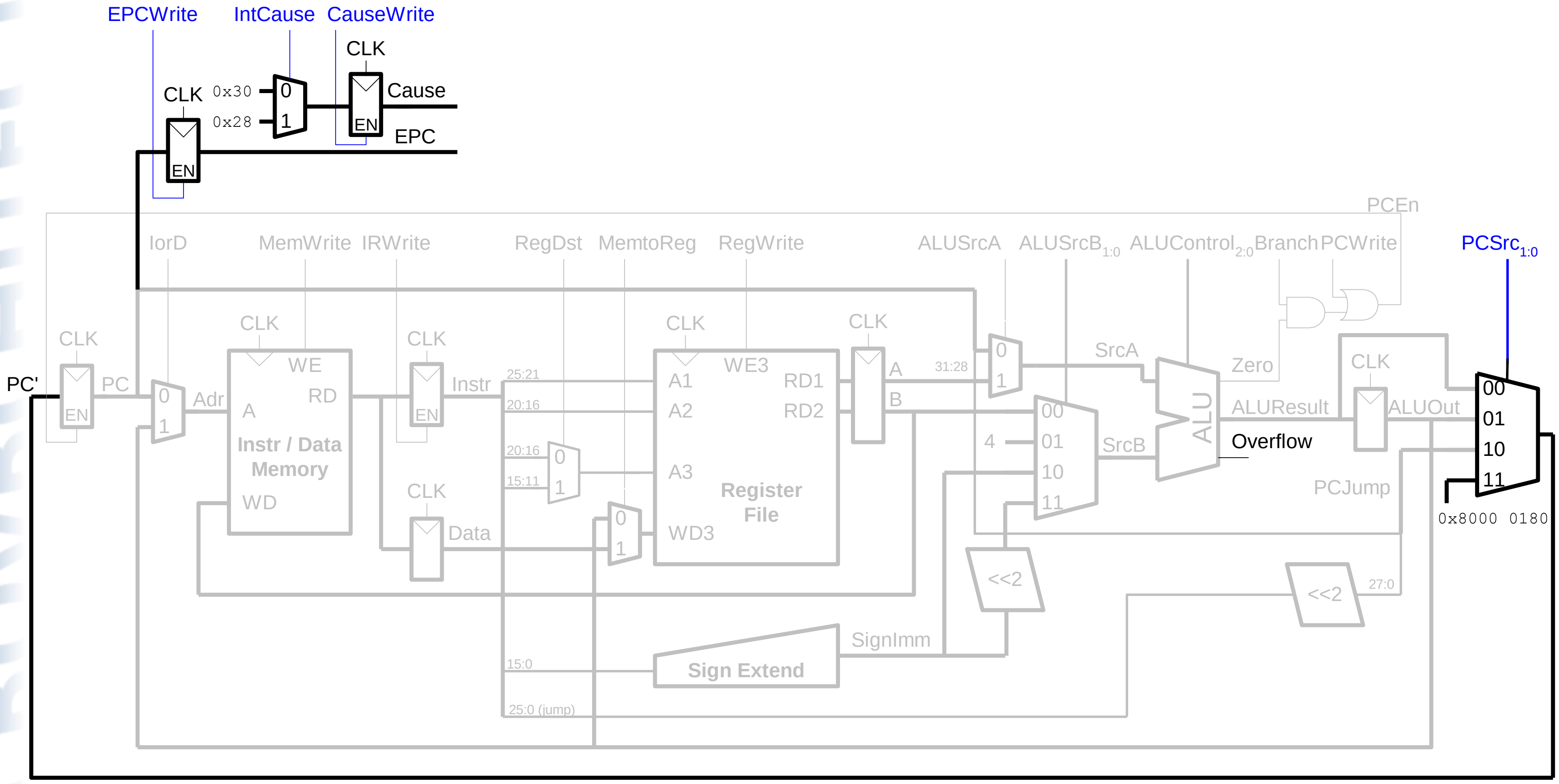
Control FSM with Exceptions



Exception Hardware: `mfc0`



Exception Hardware: EPC & Cause



Branch Prediction Example

```
  add  $s1, $0, $0        # sum = 0
  add  $s0, $0, $0        # i   = 0
  addi $t0, $0, 10        # $t0 = 10
for:
  beq  $s0, $t0, done     # if i == 10, branch
  add  $s1, $s1, $s0      # sum = sum + i
  addi $s0, $s0, 1        # increment i
  j    for
done:
```

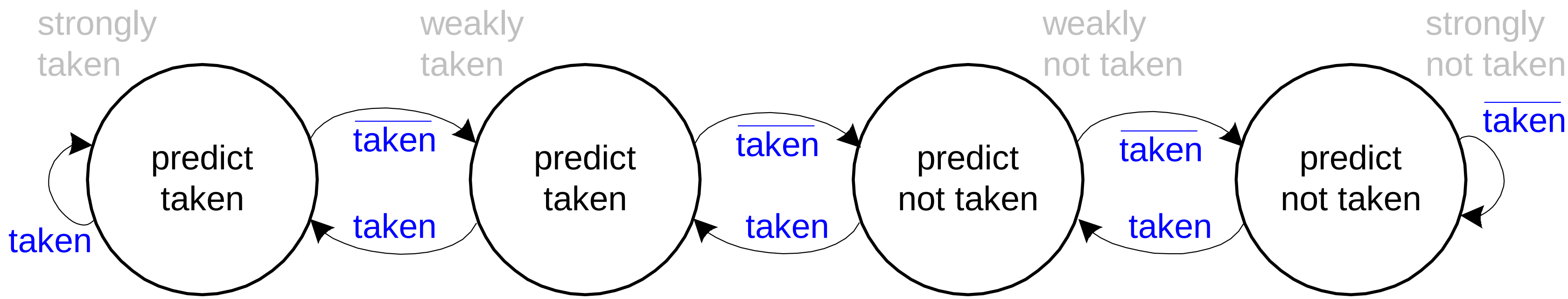
1-Bit Branch Predictor

- Remembers whether branch was taken the last time and does the same thing
- Mispredicts first and last branch of loop

MICROARCHITECTURE

ELSEVIER

2-Bit Branch Predictor



Only mispredicts last branch of loop

MICROARCHITECTURE

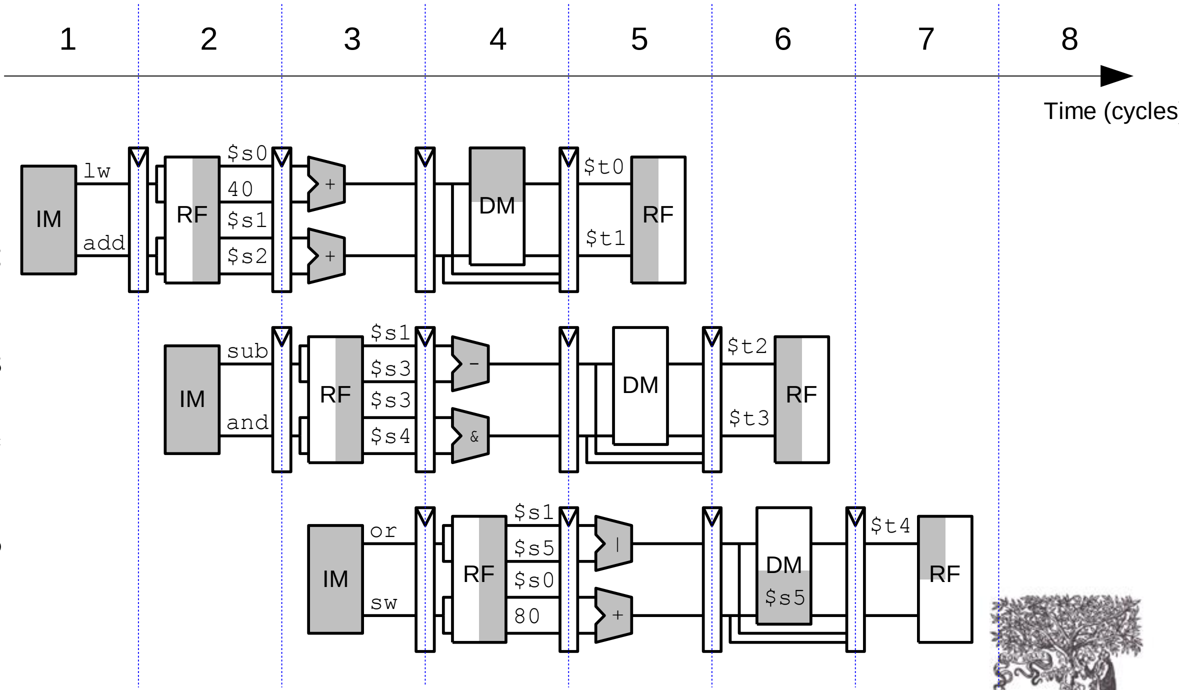
ELSEVIER

Superscalar Example

```
lw   $t0, 40($s0)
add  $t1, $t0, $s1
sub  $t0, $s2, $s3
and  $t2, $s4, $t0
or   $t3, $s5, $s6
sw   $s7, 80($t3)
```

Ideal IPC: 2

Actual IPC: 2



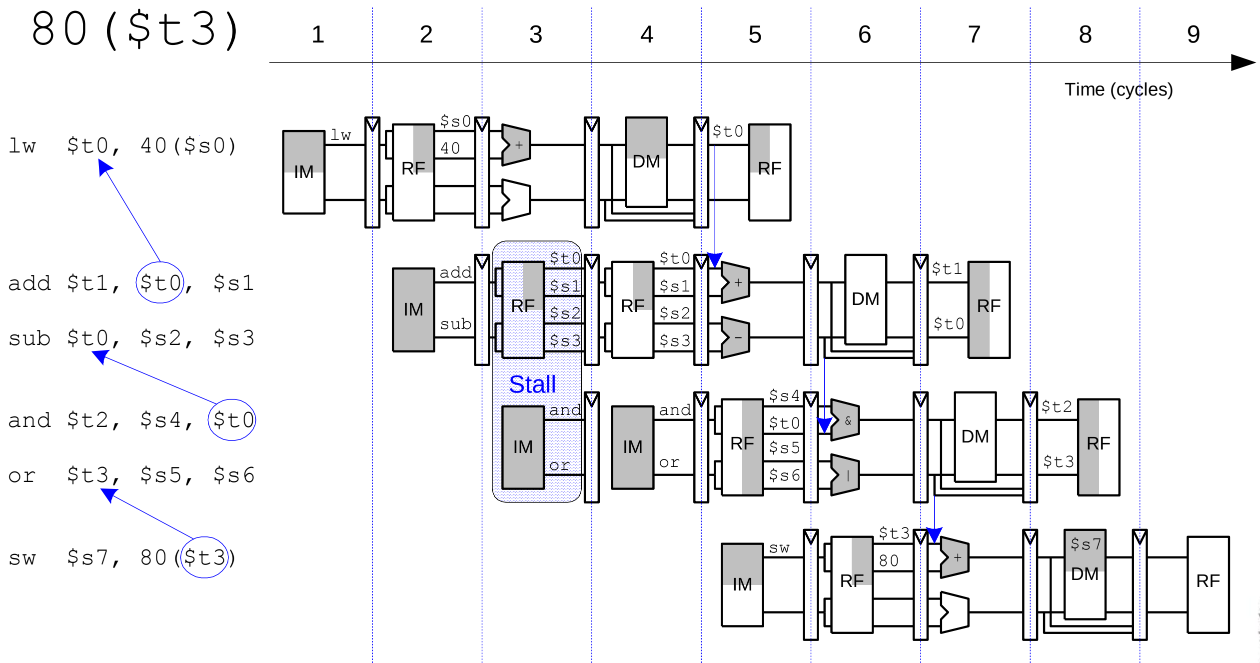
ELSEVIER

Superscalar with Dependencies

```
lw   $t0, 40($s0)
add  $t1, $t0, $s1
sub  $t0, $s2, $s3
and  $t2, $s4, $t0
or   $t3, $s5, $s6
sw   $s7, 80($t3)
```

Ideal IPC: **2**

Actual IPC: **6/5 = 1.2**

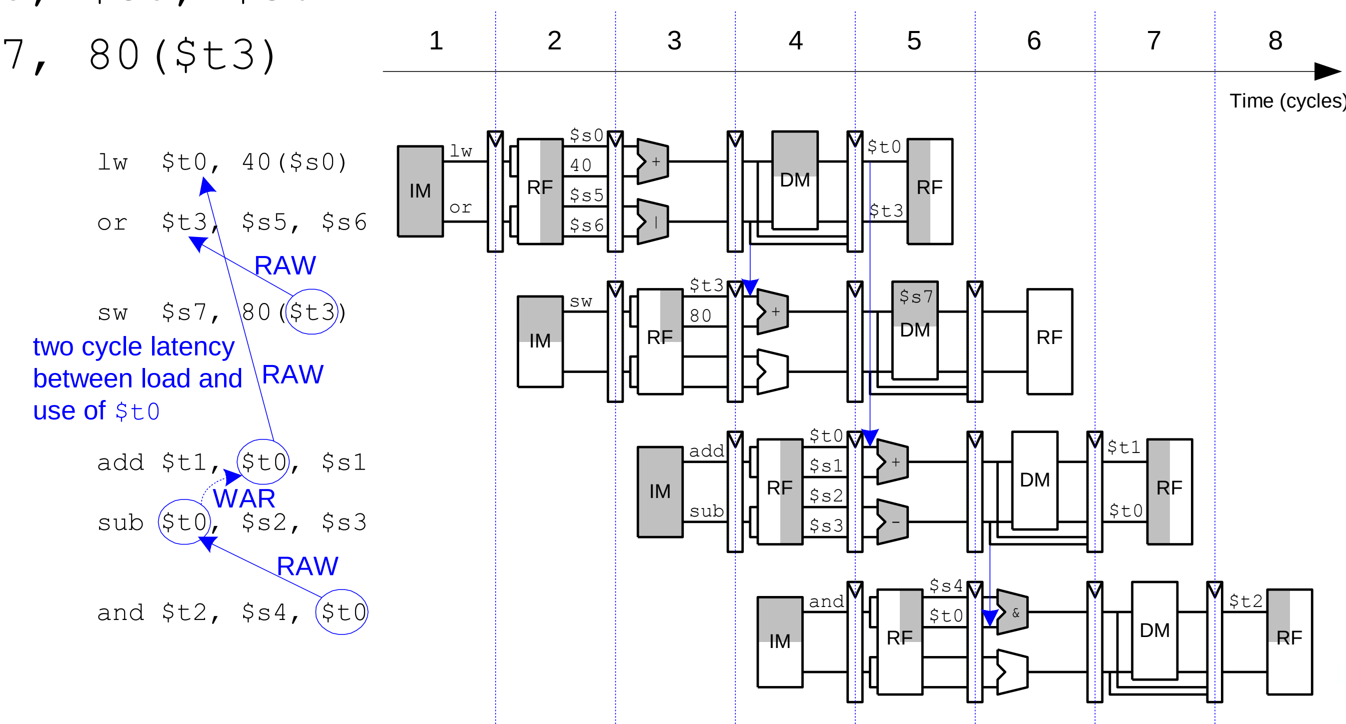


Out of Order Processor Example

```
lw   $t0, 40($s0)
add  $t1, $t0, $s1
sub  $t0, $s2, $s3
and  $t2, $s4, $t0
or   $t3, $s5, $s6
sw   $s7, 80($t3)
```

Ideal IPC: 2

Actual IPC: 6/4 = 1.5

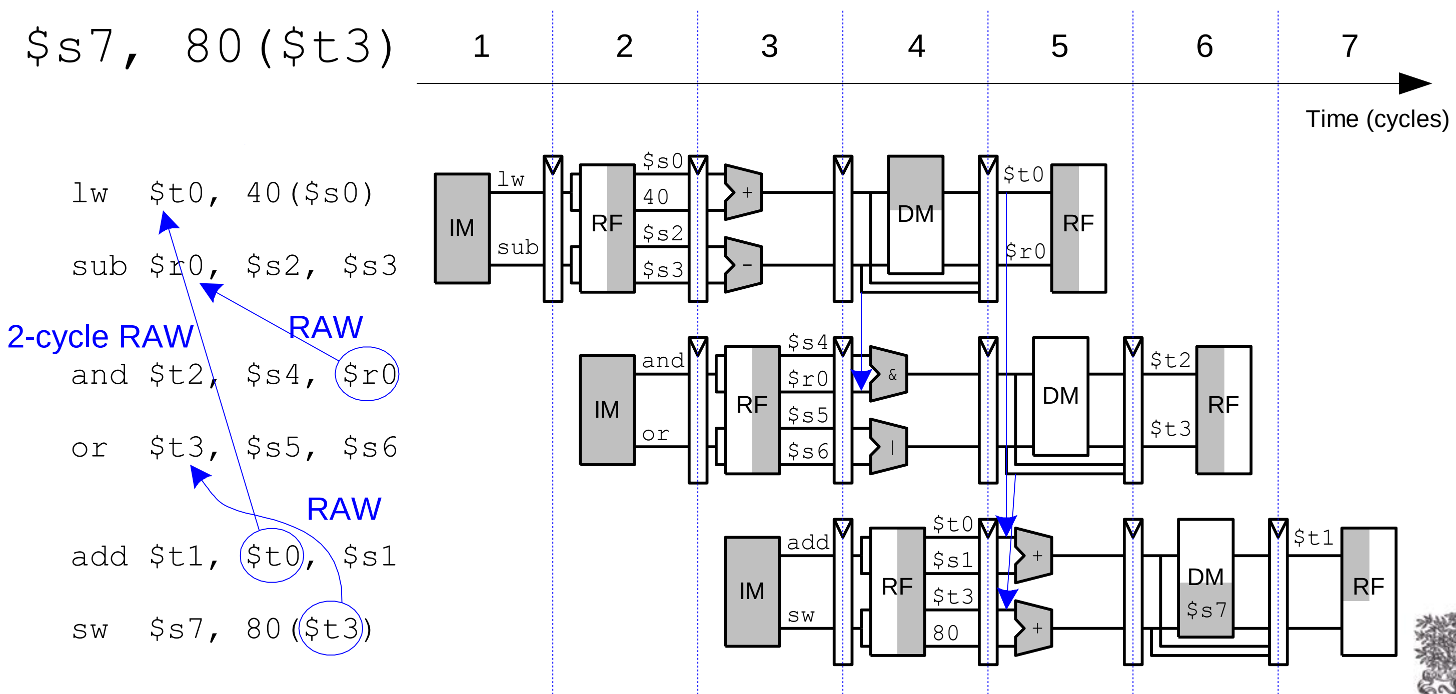


Register Renaming

```
lw   $t0, 40($s0)
add  $t1, $t0, $s1
sub  $t0, $s2, $s3
and  $t2, $s4, $t0
or   $t3, $s5, $s6
sw   $s7, 80($t3)
```

Ideal IPC: **2**

Actual IPC: **6/3 = 2**



Other Resources

- Patterson & Hennessy's: *Computer Architecture: A Quantitative Approach*
- Conferences:
  - www.cs.wisc.edu/~arch/www/
  - ISCA (International Symposium on Computer Architecture)
  - HPCA (International Symposium on High Performance Computer Architecture)

MICROARCHITECTURE