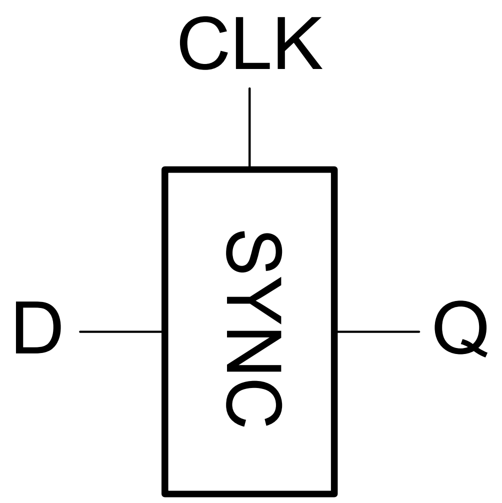
EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 12_3: Timing of Digital Systems

Xiaolin Xu
Department of ECE
Northeastern University

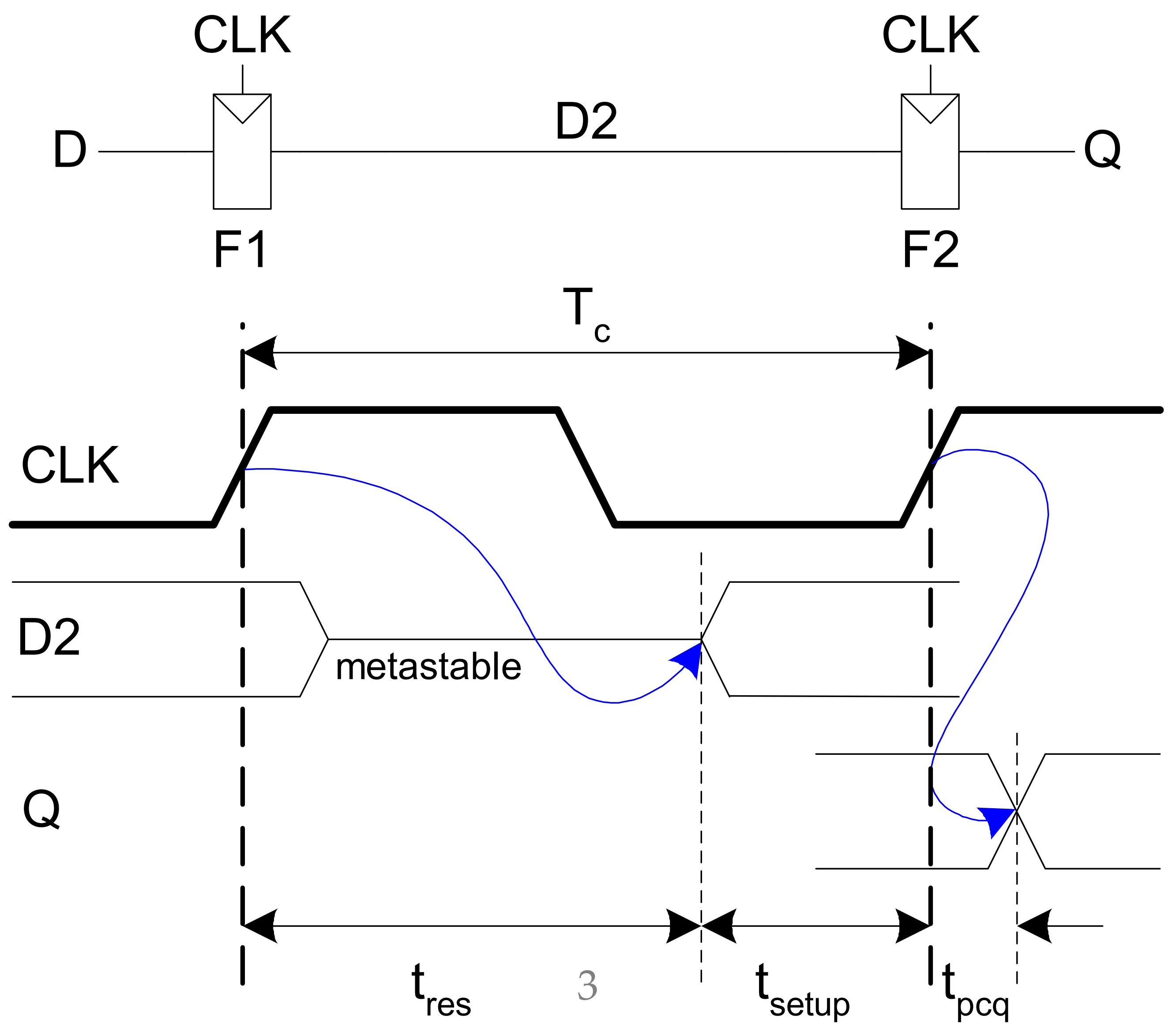
Synchronizers

- **Asynchronous inputs are inevitable** (user interfaces, systems with different clocks interacting, etc.)
- **Synchronizer goal:** make the probability of failure (the output *Q* still being metastable) low
- Synchronizer cannot make the probability of failure 0

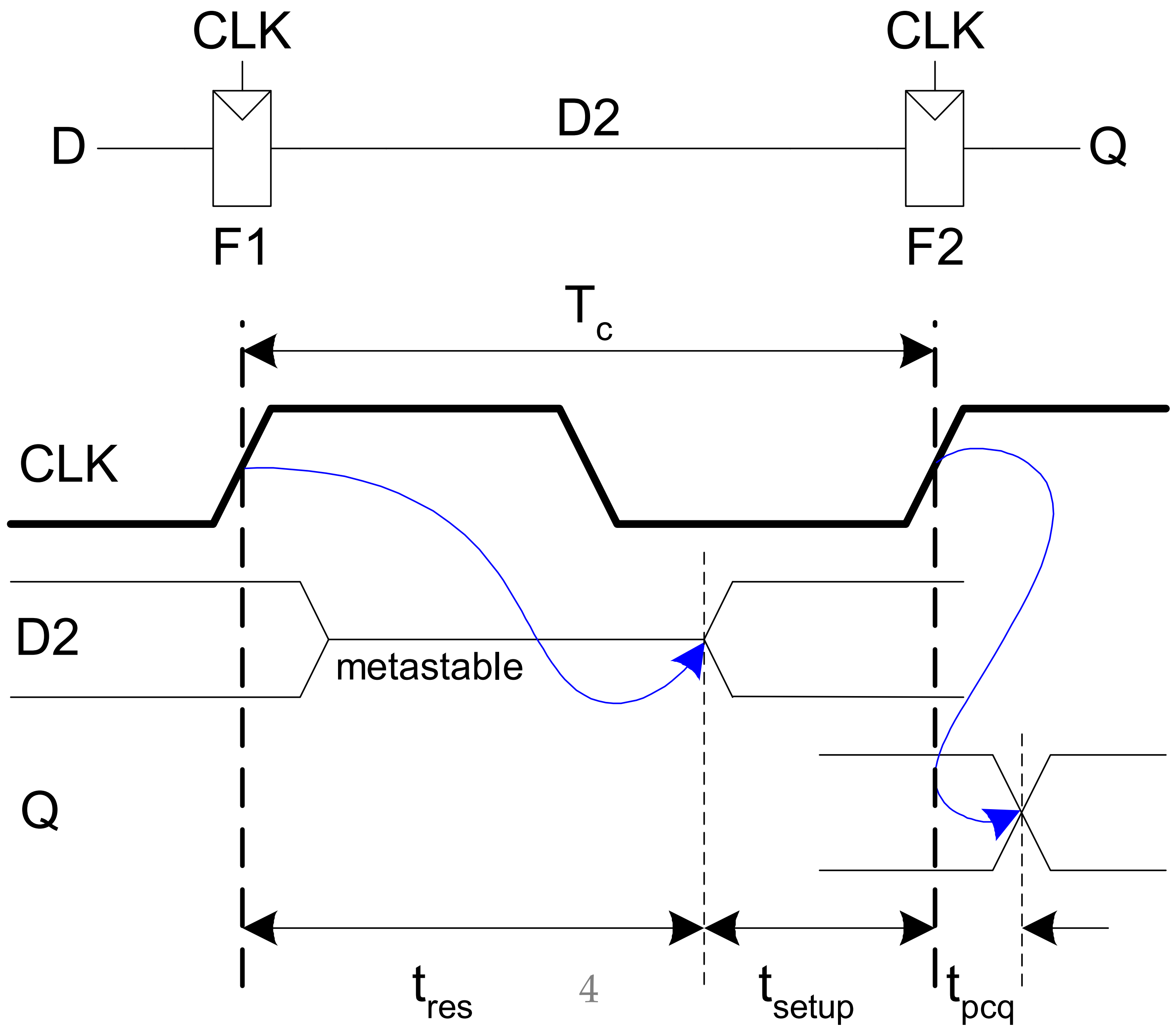


Synchronizer Internals

- Synchronizer: built with two back-to-back flip-flops
- Suppose D is transitioning when sampled by F1
- Internal signal D2 has (T_c - t_{setup}) time to resolve to 1 or 0

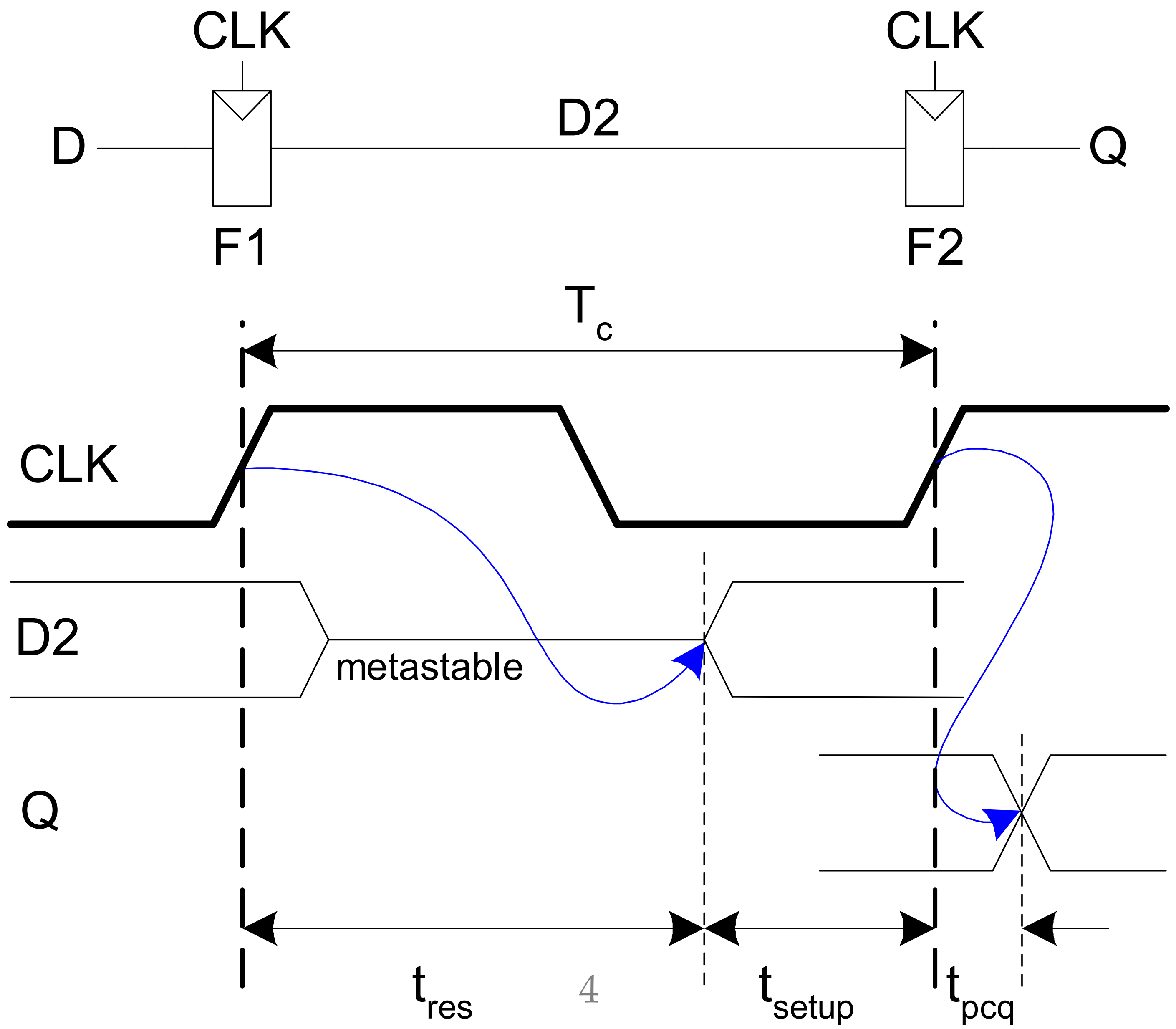


Synchronizer Probability of Failure



Synchronizer Probability of Failure

$$P(t_{res} > t) = (T_0 / T_c)\, e^{-t/\tau}$$

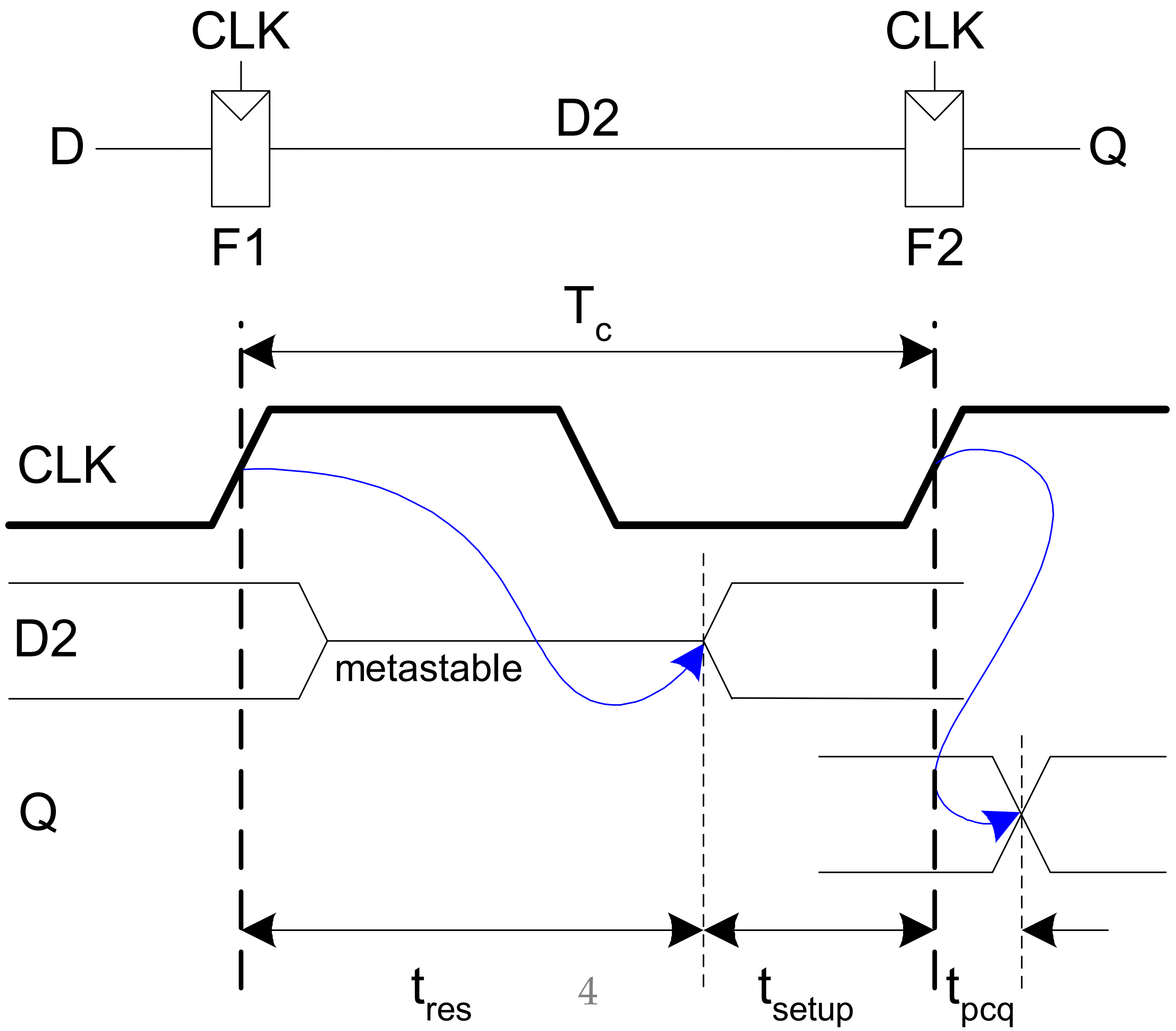


Synchronizer Probability of Failure

$P(t_{res} > t) = (T_0 / T_c)\, e^{-t/\tau}$

For each sample, probability of failure is:

$$\mathbf{P(failure)} = (T_0/T_c)\, e^{-(T_c - t_{setup})/\tau}$$



Synchronizer Mean Time Between Failures

- If asynchronous input changes once per second, probability of failure per second is P(failure).
- If input changes N times per second, probability of failure per second is:

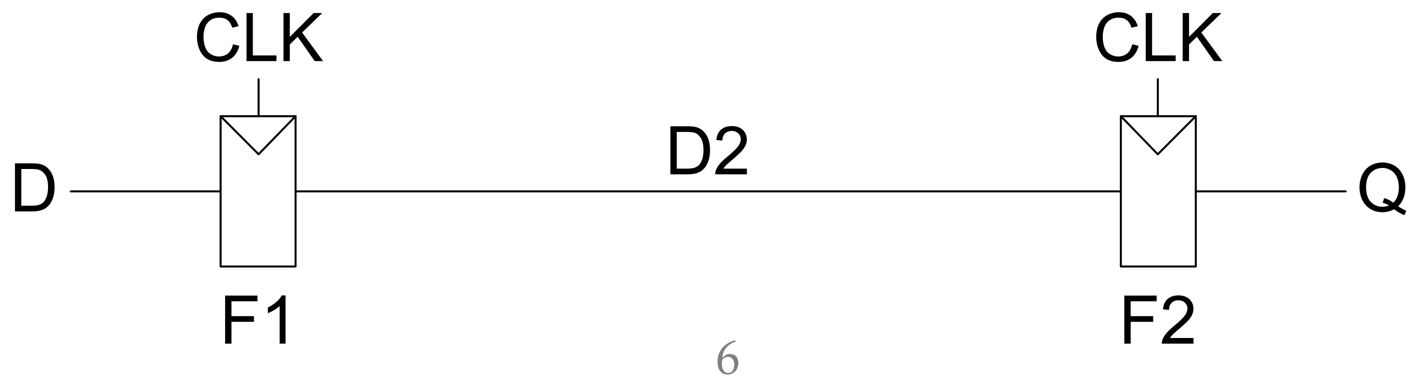
$$\mathbf{P(failure)/second = (NT_0/T_c)\ e^{-(T_c - t_{setup})/\tau}}$$

- Synchronizer fails, on average, 1/[P(failure)/second]
- Called ***mean time between failures***, MTBF:

$$\mathbf{MTBF = 1/[P(failure)/second] = (T_c/NT_0)\ e^{(T_c - t_{setup})/\tau}}$$

Example Synchronizer

- Suppose: $T_c = 1/500\text{ MHz} = 2\text{ ns}$ $\tau = 200\text{ ps}$
  $T_0 = 150\text{ ps}$ $t_{setup} = 100\text{ ps}$
  $N = 1$ events per second
- What is the probability of failure? MTBF?

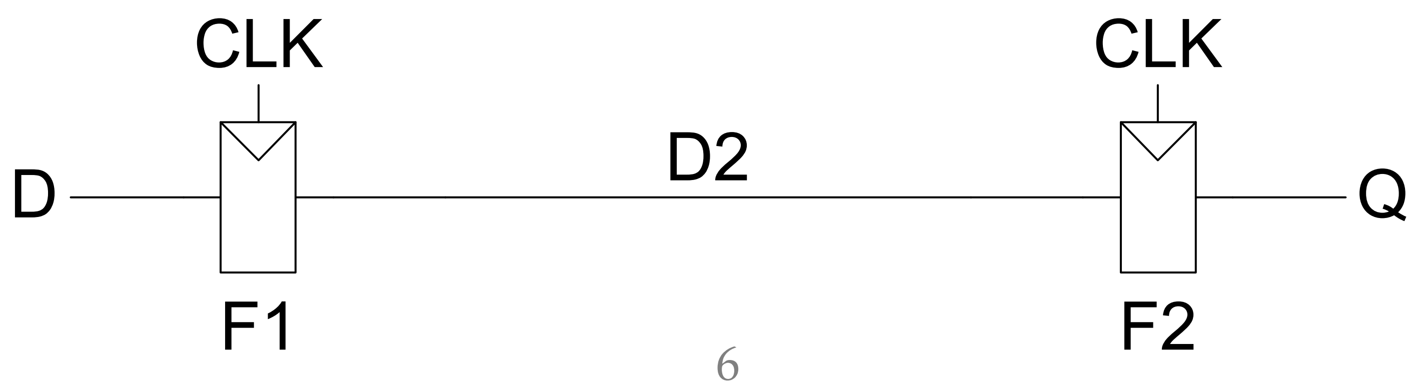


Example Synchronizer

- Suppose: $T_c = 1/500\text{ MHz} = 2\text{ ns}$ $\tau = 200\text{ ps}$
  $T_0 = 150\text{ ps}$ $t_{\text{setup}} = 100\text{ ps}$
  $N = 1$ events per second
- What is the probability of failure? MTBF?

Synchronizer fails, on average, 1/[P(failure)/second]
Called ***mean time between failures***, MTBF:

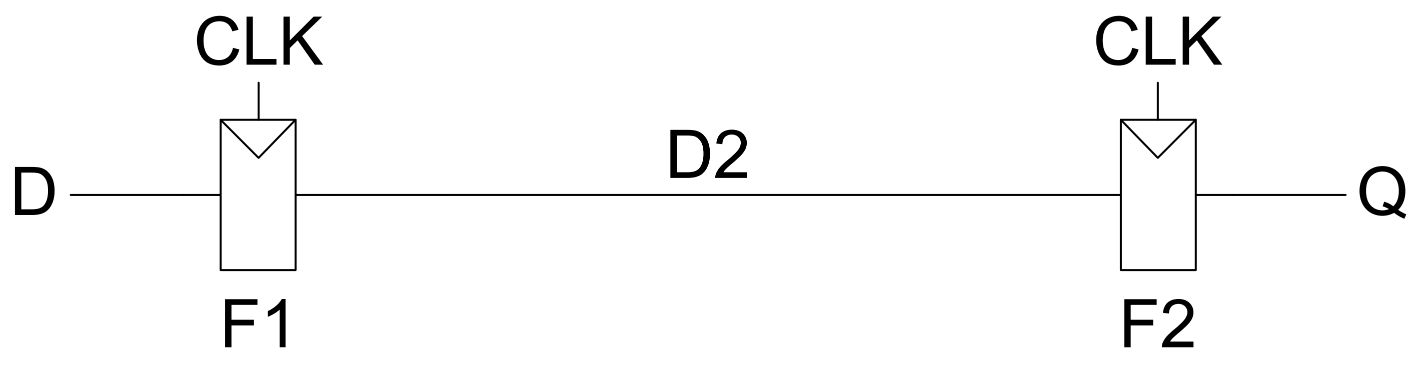
$$\text{MTBF} = 1/[P(\text{failure})/\text{second}] = (T_c/NT_0)\, e^{(T_c - t_{\text{setup}})/\tau}$$



Example Synchronizer

- Suppose: T_c = 1/500 MHz = 2 ns τ = 200 ps
  T_0 = 150 ps t_{setup} = 100 ps
  N = 1 events per second
- What is the probability of failure? MTBF?

$$P(\text{failure}) = (150\text{ ps}/2\text{ ns})\, e^{-(1.9\text{ ns})/200\text{ ps}}$$
$$= \mathbf{5.6 \times 10^{-6}}$$
$$P(\text{failure})/\text{second} = 10 \times (5.6 \times 10^{-6})$$
$$= 5.6 \times 10^{-5} / \text{second}$$
$$\text{MTBF} = 1/[P(\text{failure})/\text{second}] \approx \mathbf{5\ hours}$$



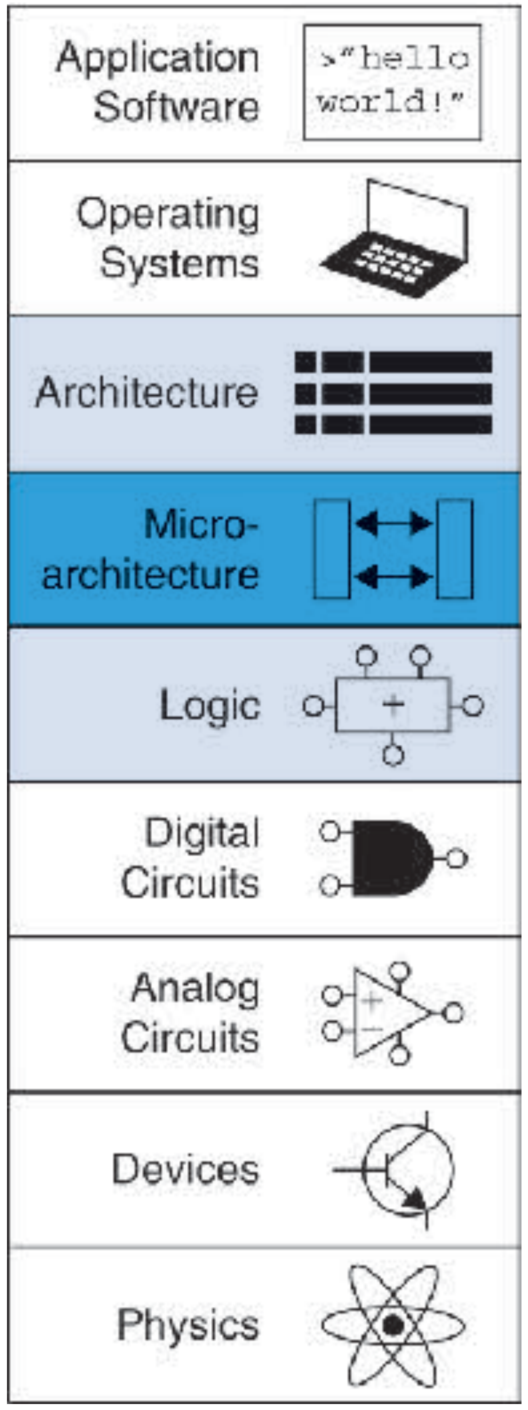
EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 12_3: Microarchitecture

Xiaolin Xu
Department of ECE
Northeastern University

Microarchitecture Topics

- Introduction
- Performance Analysis
- Single-Cycle Processor
- Multicycle Processor
- Pipelined Processor
- Exceptions
- Advanced Microarchitecture



Microarchitecture

- **Microarchitecture:** how to implement an Instruction-Set Architecture (ISA) in hardware
- Processor:
  - **Datapath:** functional blocks
  - **Control:** control signals

Application Software	programs
Operating Systems	device drivers
Architecture	instructions registers
Micro-architecture	datapaths controllers
Logic	adders memories
Digital Circuits	AND gates NOT gates
Analog Circuits	amplifiers filters
Devices	transistors diodes
Physics	electrons

Microarchitecture

- Multiple implementations for a single architecture:
  - **Single-cycle:** Each instruction executes in a single cycle
  - **Multicycle:** Each instruction is broken into series of shorter steps
  - **Pipelined:** Each instruction broken up into series of steps & multiple instructions execute at once

Processor Performance

- Program execution time

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

- Definitions:
  - CPI: Cycles/instruction
  - clock period: seconds/cycle
  - IPC: instructions/cycle = IPC
- Challenge is to satisfy constraints of:
  - Cost
  - Power
  - Performance

MIPS Processor

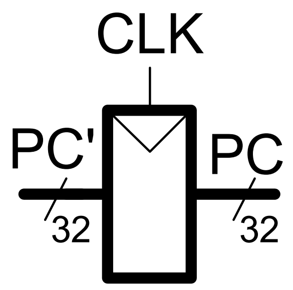
- Consider subset of MIPS instructions:
  - R-type instructions: `and`, `or`, `add`, `sub`, `slt`
  - Memory instructions: `lw`, `sw`
  - Branch instructions: `beq`

Architectural State

- What determine everything about a processor:
    - PC
    - 32 registers
    - Memory

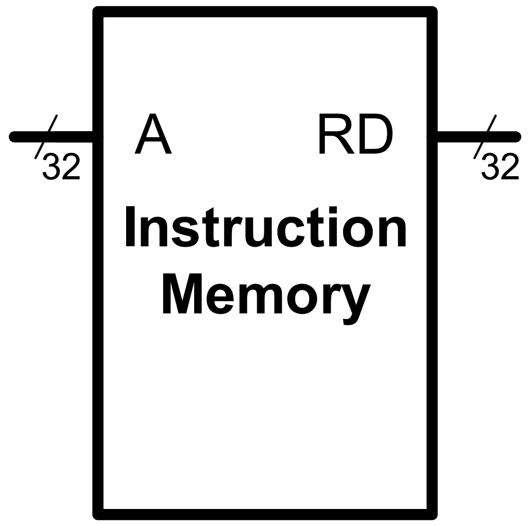
MIPS State Elements

- PC: program counter —> the next instruction



MIPS State Elements

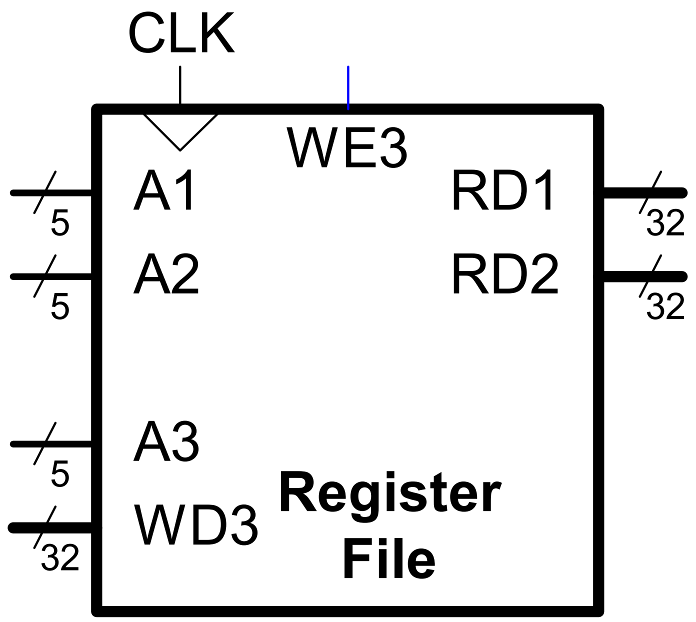
- Instruction memory:



  - Read (RD) port
    - 32-bit addr as input
    - 32-bit instruction as output

MIPS State Elements

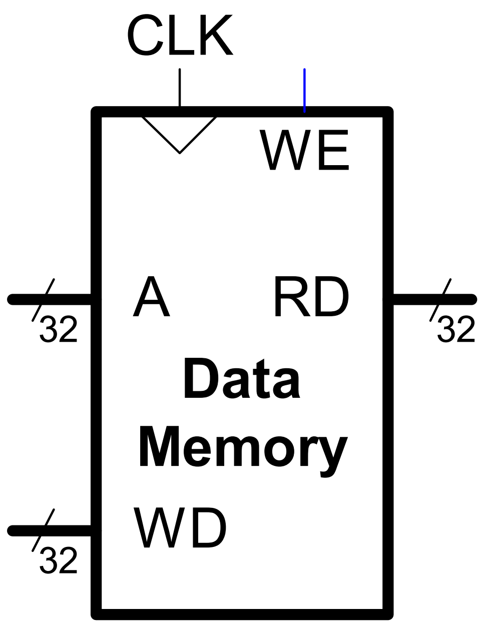
- Register file



- Two read ports — RD1 / RD2
  - 5-bit address inputs A1 / A2 — selecting registers as source operands
- One write port — WD3, with enable signal WE3 (1 active)
  - 5-bit address input A3 —

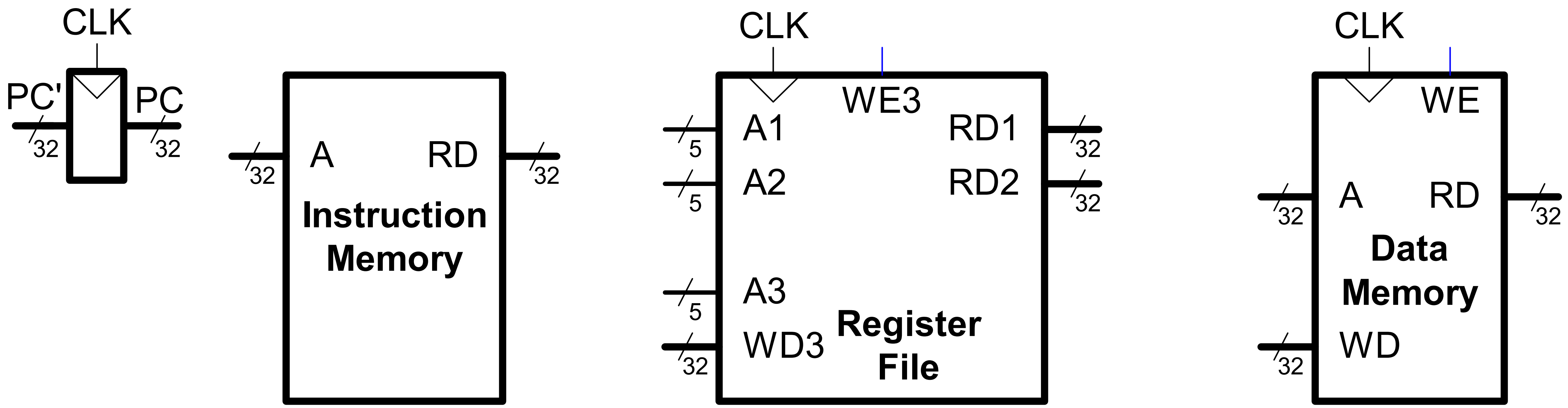
MIPS State Elements

- Data memory



- 32-bit address (A) as input, 32-bit write-in data (WD) input, with enable signal WE (1 active)
- WE = 0 — reading the data predicted by address A

MIPS State Elements



- Instruction memory, register file, and data memory, are being **read** as **combinational circuit, i.e., data change follows address change, while the writing follows clock-edge**
- **Benefits: all elements are synchronized, while MIPS processor is a FSM!**

Single-Cycle MIPS Processor

- Datapath
- Control

Overview: Single-Cycle Control

R-Type

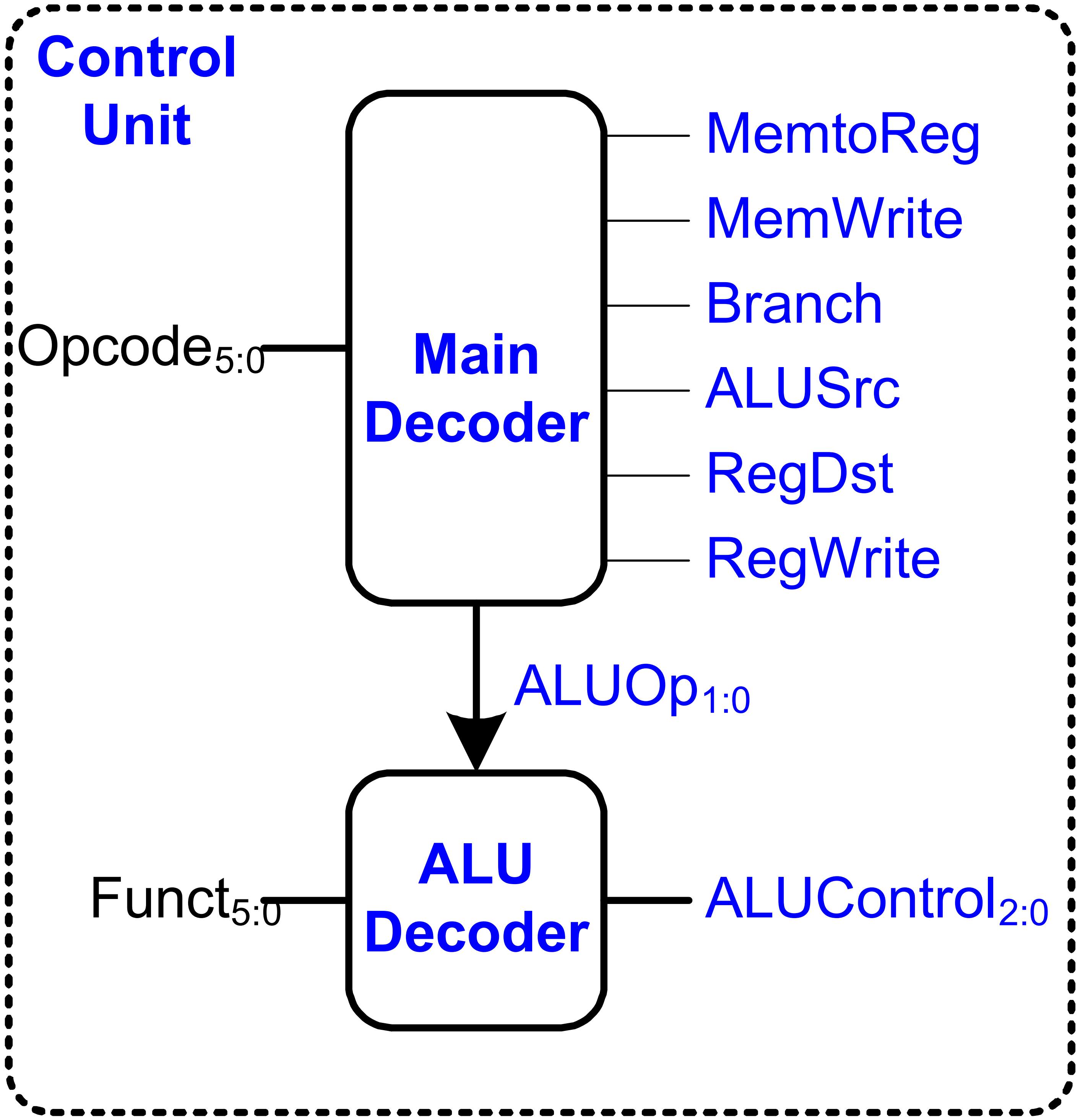
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

J-Type

op	addr
6 bits	26 bits



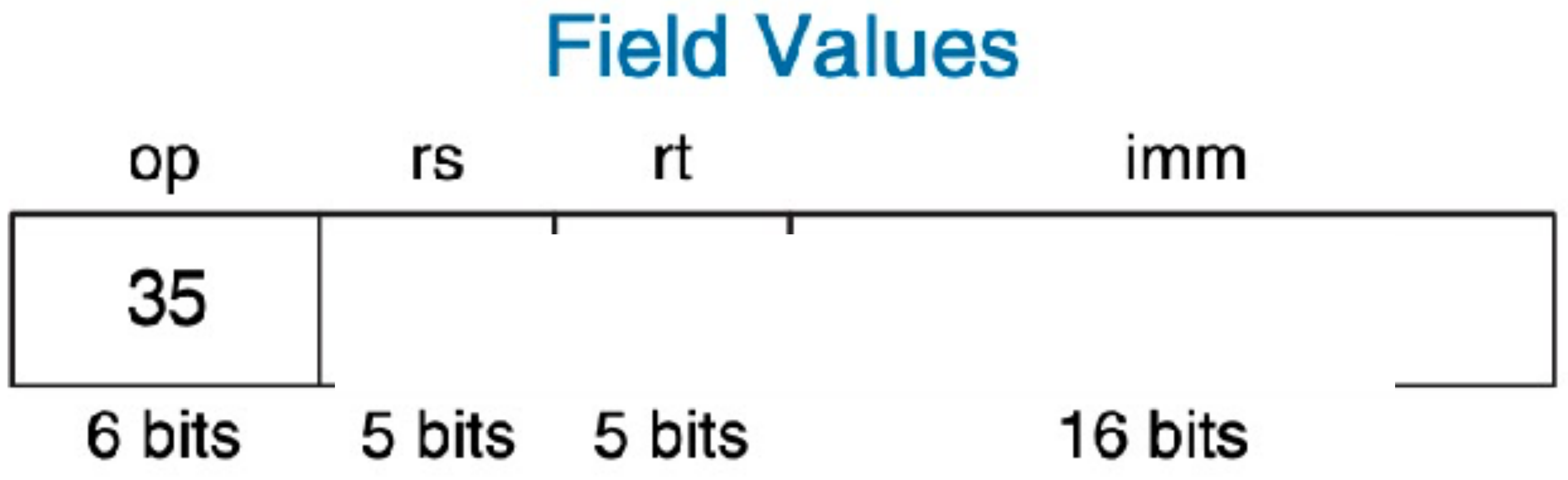
Single-Cycle Datapath Example: lw fetch

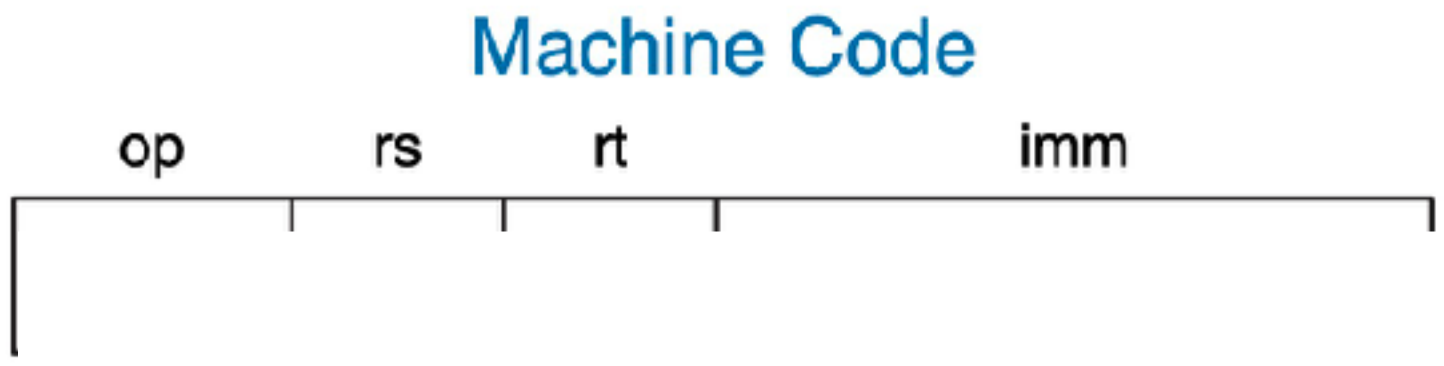
Revisit: Practice of I-type

`lw $s3, -24($s4)`

lw has opcode of 35

Name	Number
`$0`	0
`$at`	1
`$v0–$v1`	2–3
`$a0–$a3`	4–7
`$t0–$t7`	8–15
`$s0–$s7`	16–23
`$t8–$t9`	24–25
`$k0–$k1`	26–27
`$gp`	28
`$sp`	29
`$fp`	30
`$ra`	31



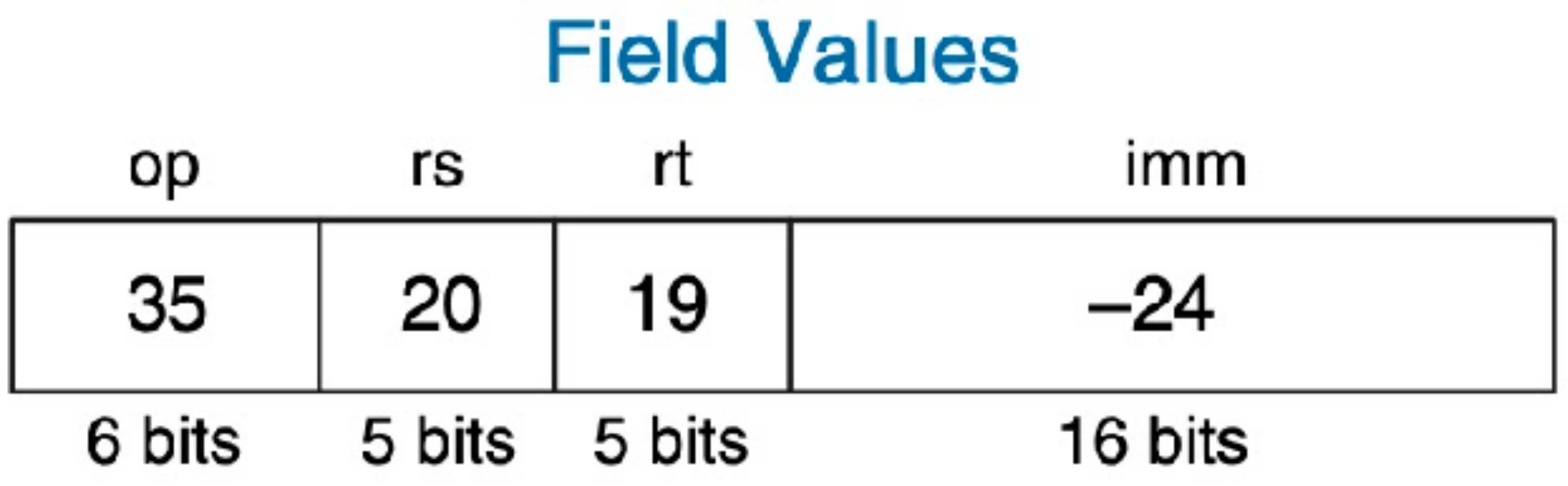


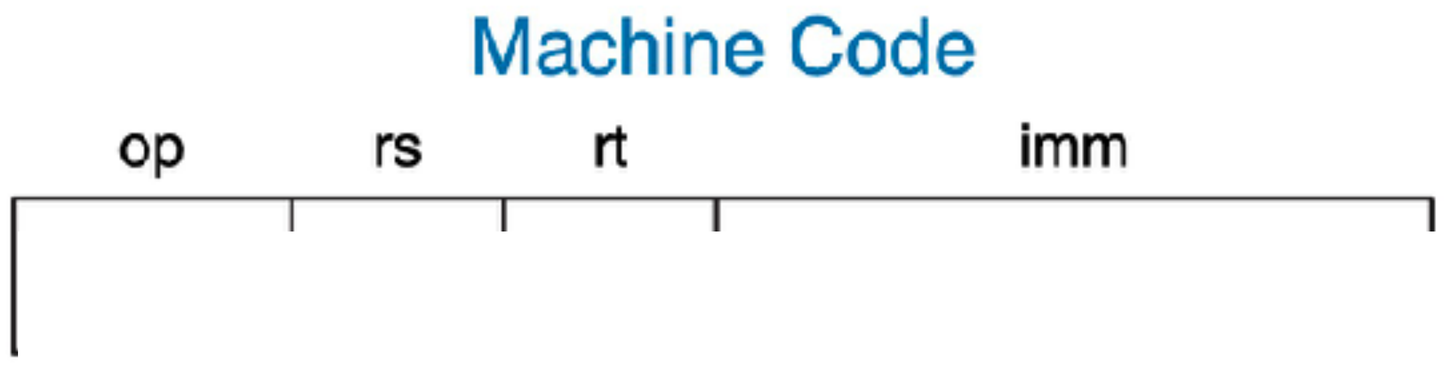
Revisit: Practice of I-type

`lw $s3, -24($s4)`

lw has opcode of 35

Name	Number
$0	0
$at	1
$v0–$v1	2–3
$a0–$a3	4–7
$t0–$t7	8–15
$s0–$s7	16–23
$t8–$t9	24–25
$k0–$k1	26–27
$gp	28
$sp	29
$fp	30
$ra	31



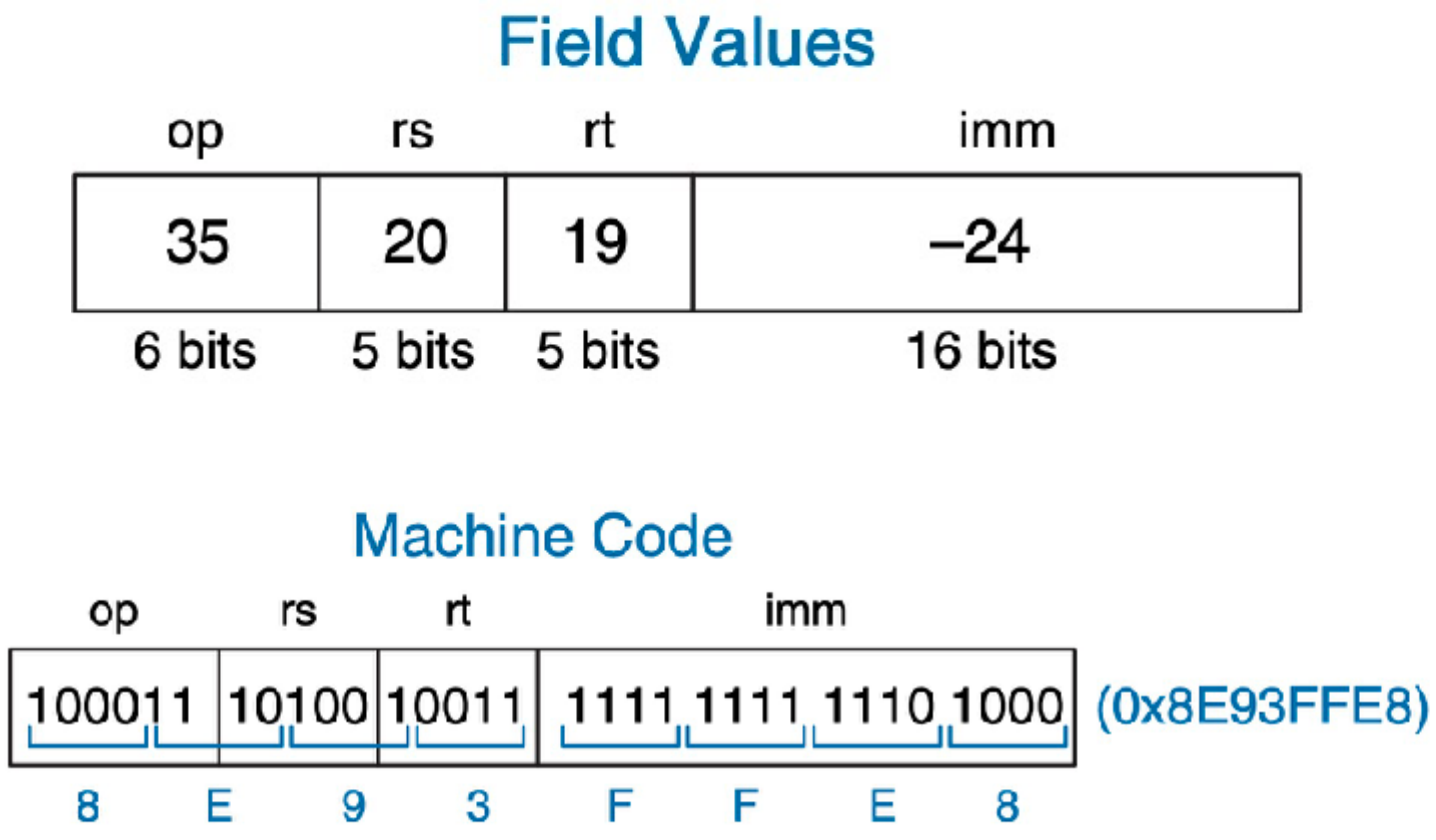


Revisit: Practice of I-type

```
lw $s3, -24($s4)
```

lw has opcode of 35

Name	Number
`$0`	0
`$at`	1
`$v0–$v1`	2–3
`$a0–$a3`	4–7
`$t0–$t7`	8–15
`$s0–$s7`	16–23
`$t8–$t9`	24–25
`$k0–$k1`	26–27
`$gp`	28
`$sp`	29
`$fp`	30
`$ra`	31

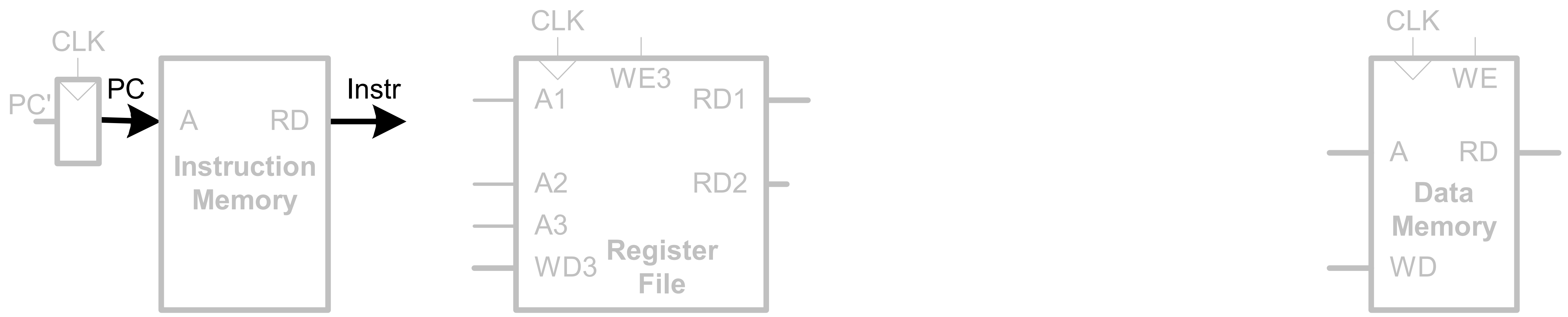


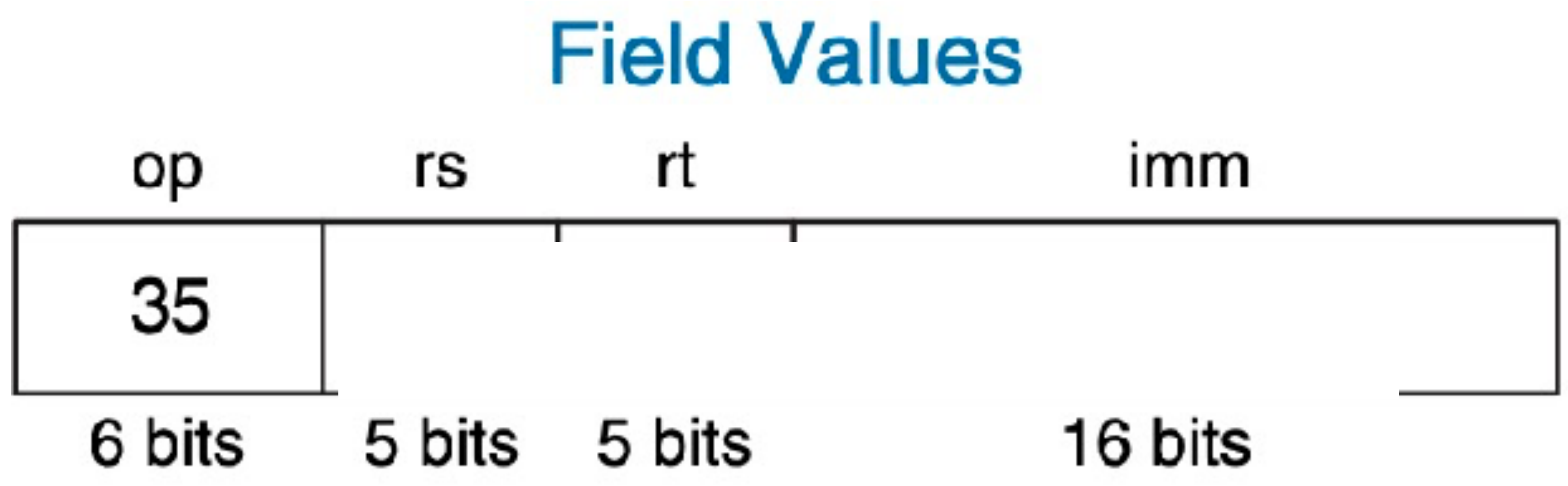
Single-Cycle Datapath Example: lw fetch

STEP 1: Fetch instruction

`lw $s3, -24($s4)`

lw has opcode of 35



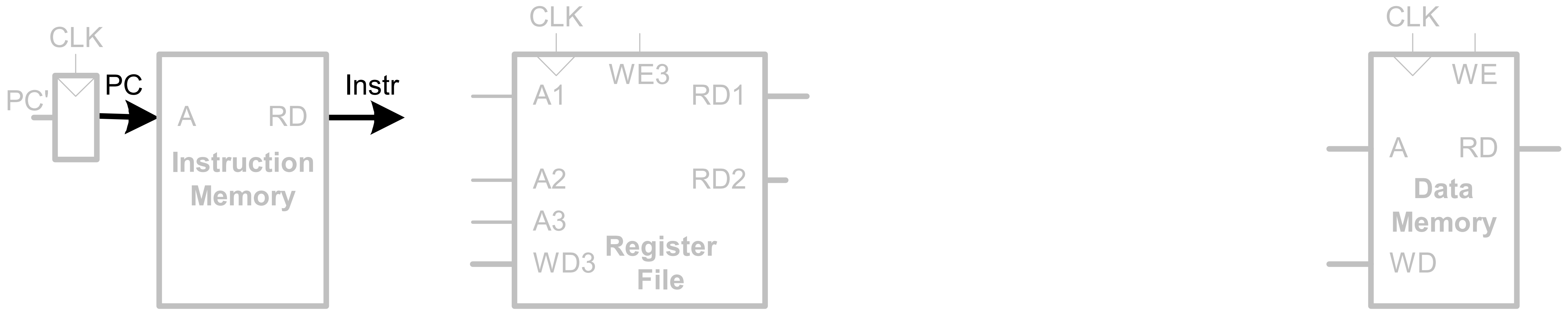


Single-Cycle Datapath Example: lw fetch

STEP 1: Fetch instruction

```
lw $s3, -24($s4)
```

lw has opcode of 35



Field Values

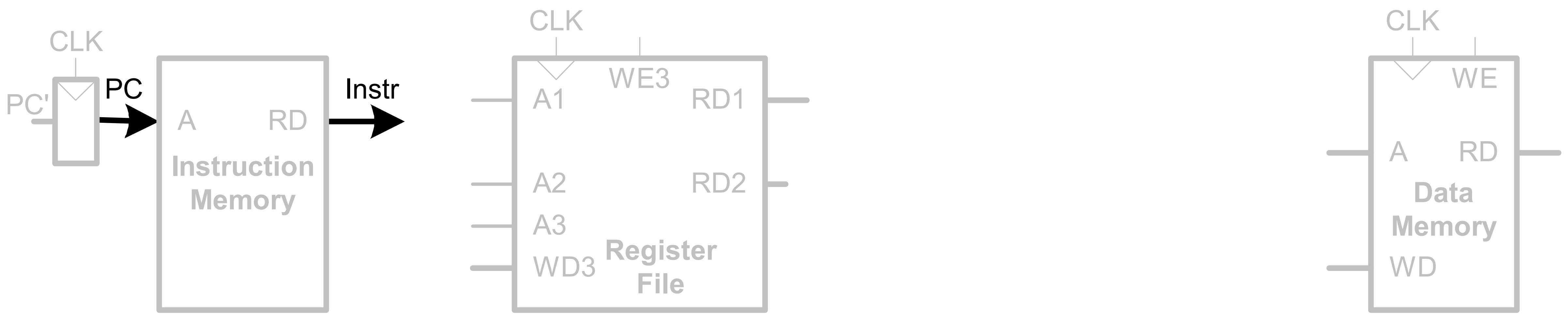
op	rs	rt	imm
35	20	19	–24
6 bits	5 bits	5 bits	16 bits

Single-Cycle Datapath Example: lw fetch

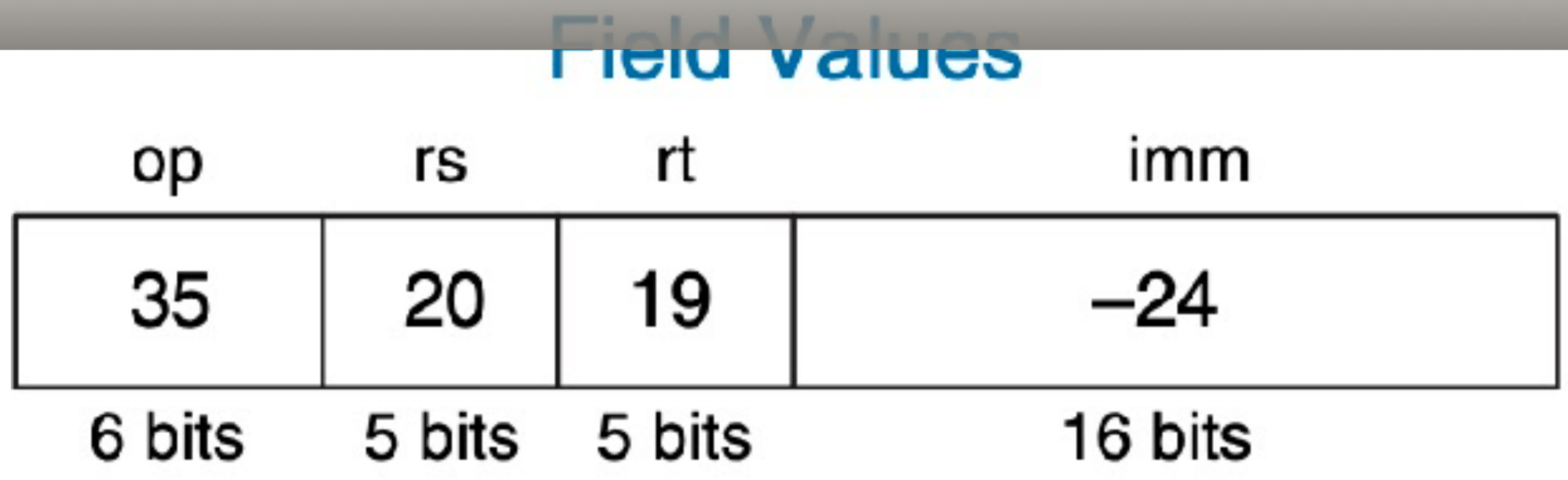
STEP 1: Fetch instruction

```
lw $s3, -24($s4)
```

lw has opcode of 35



The offset is stored in the immediate field of the instruction, $Instr_{15:0}$

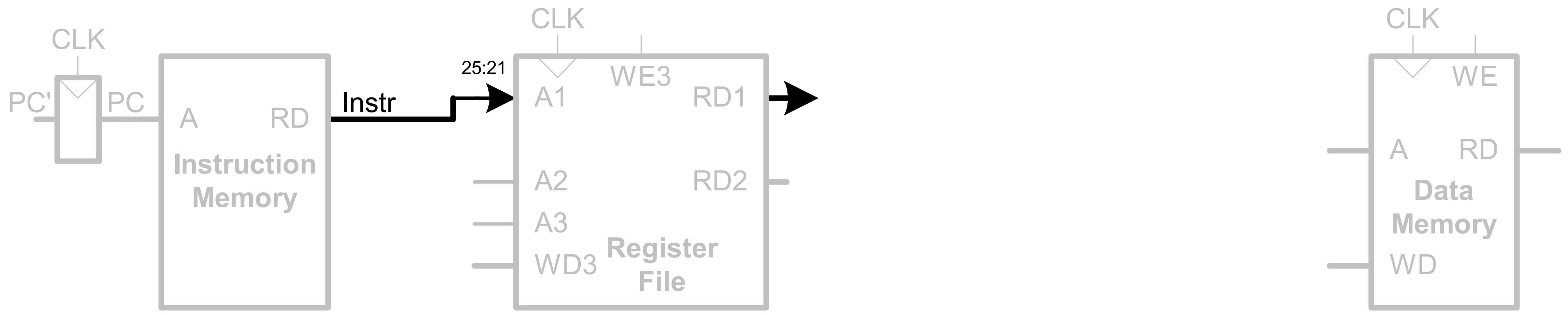

Field Values

op	rs	rt	imm
35	20	19	–24
6 bits	5 bits	5 bits	16 bits

Single-Cycle Datapath: lw Register Read

STEP 2: Read source operands from RF

```
lw $s3, -24($s4)
```



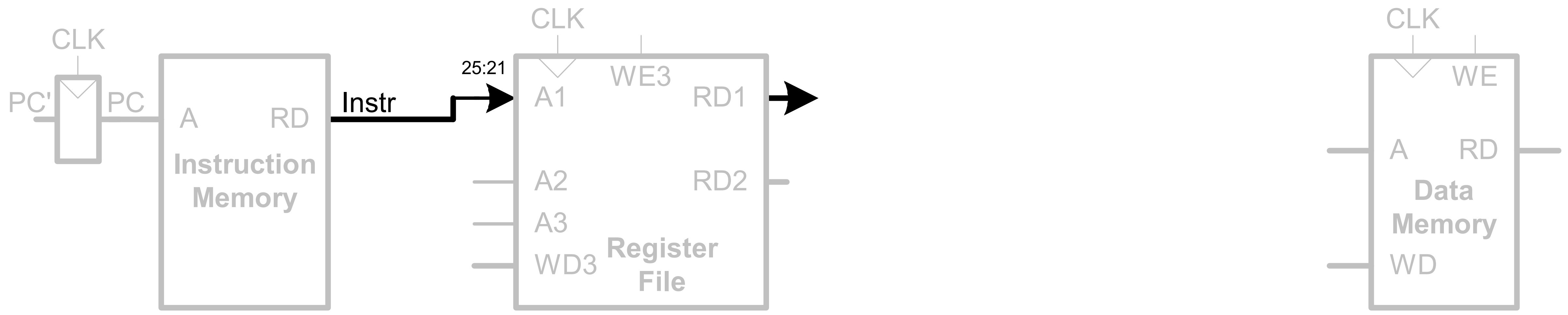
Field Values

op	rs	rt	imm
35			
6 bits	5 bits	5 bits	16 bits

Single-Cycle Datapath: lw Register Read

STEP 2: Read source operands from RF

```
lw $s3, -24($s4)
```



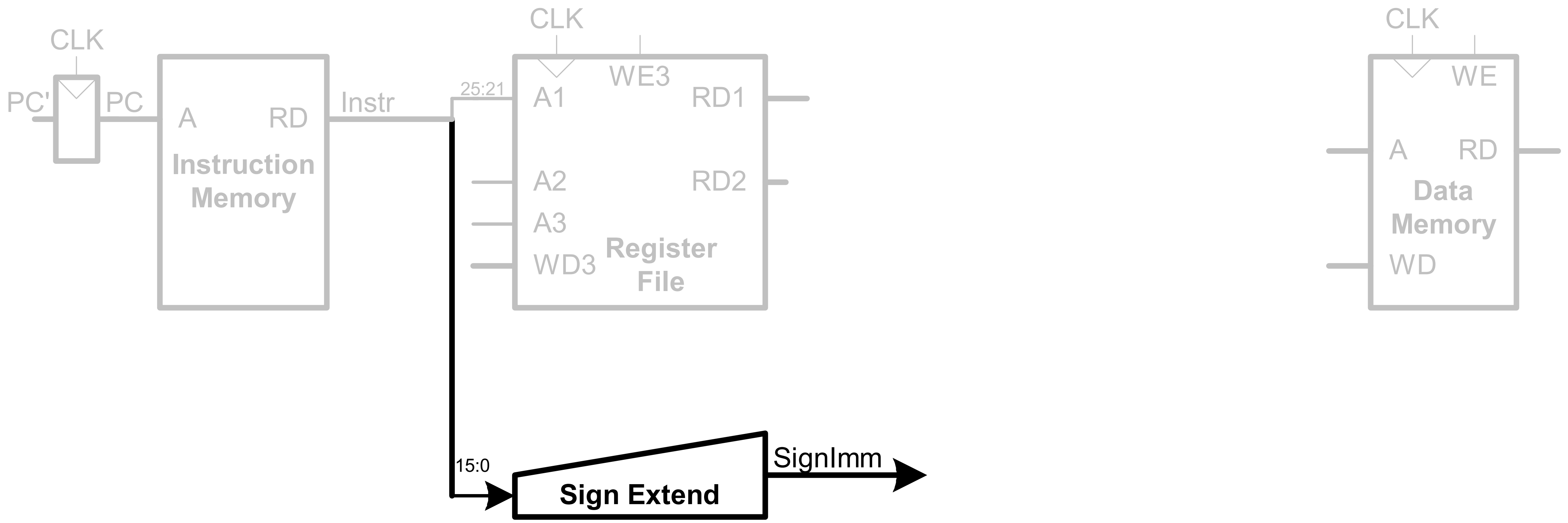
Field Values

op	rs	rt	imm
35	20	19	–24
6 bits	5 bits	5 bits	16 bits

Single-Cycle Datapath: lw Immediate

STEP 3: Sign-extend the immediate

```
lw $s3, -24($s4)
```



Field Values

op	rs	rt	imm
35			
6 bits	5 bits	5 bits	16 bits

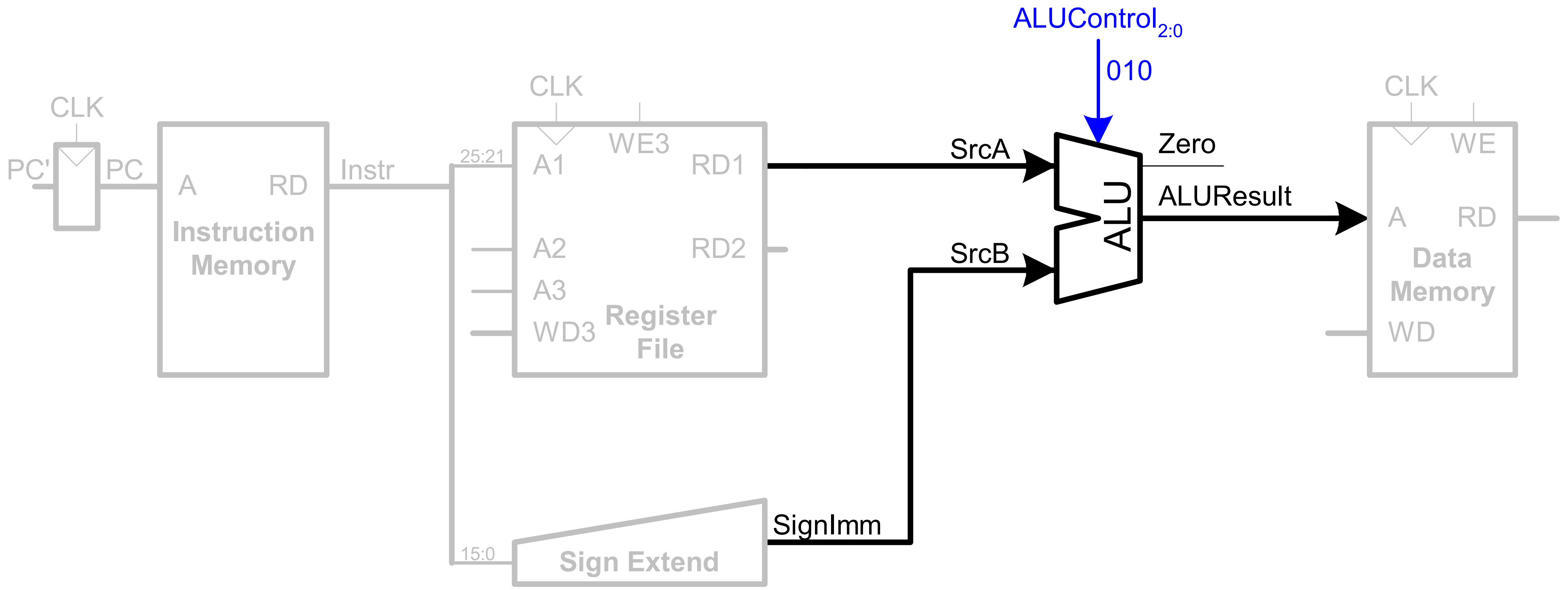
$SignImm_{15:0} = Instr_{15:0}$

$SignImm_{31:16} = Instr_{15}$

Single-Cycle Datapath: lw address

STEP 4: Compute the memory address

```
lw $s3, -24($s4)
```



Field Values

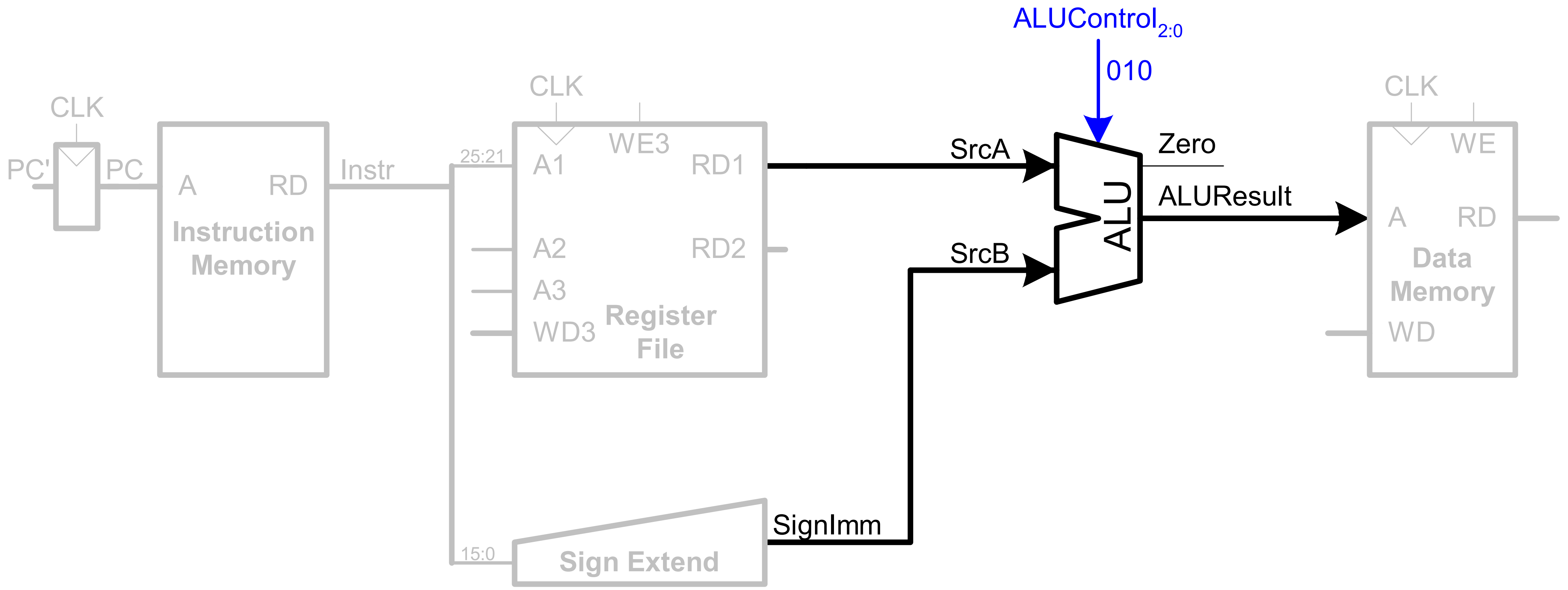
op	rs	rt	imm
35			
6 bits	5 bits	5 bits	16 bits

Single-Cycle Datapath: lw address

add the base address to the offset to find the address to read from memory

STEP 4: Compute the memory address

```
lw $s3, -24($s4)
```



Field Values

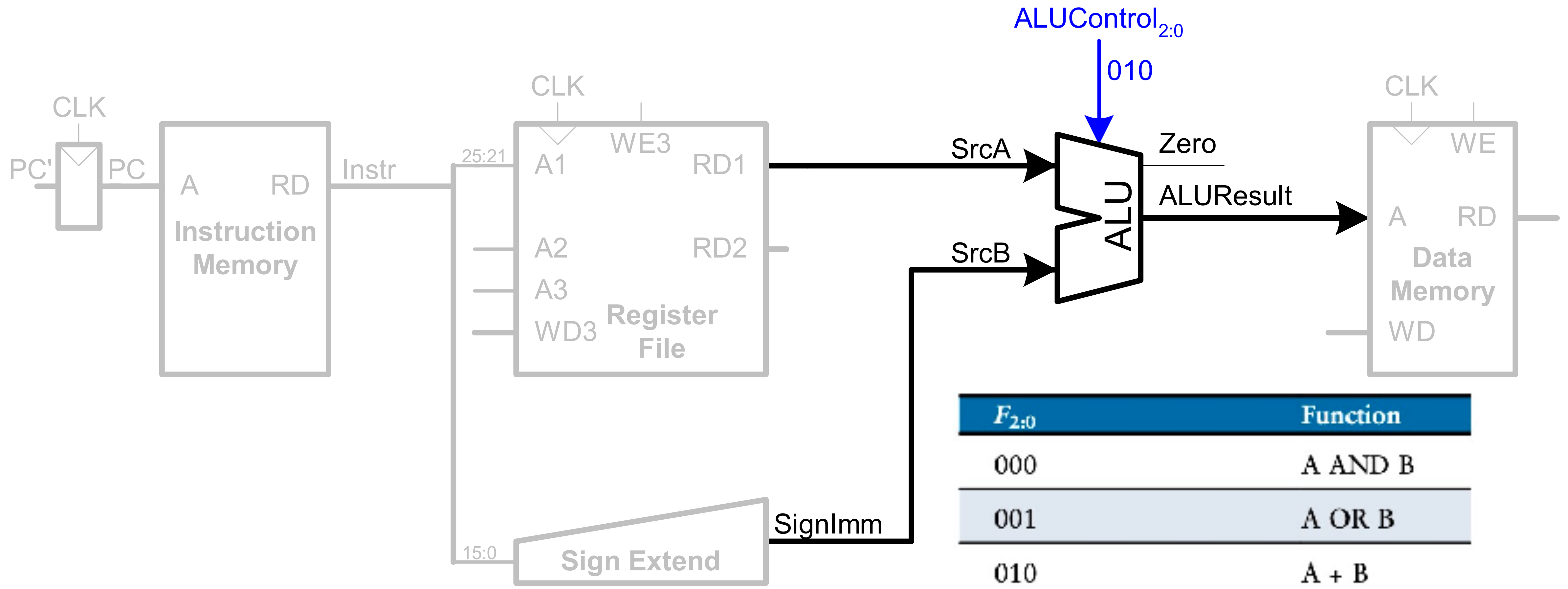
op	rs	rt	imm
35			
6 bits	5 bits	5 bits	16 bits

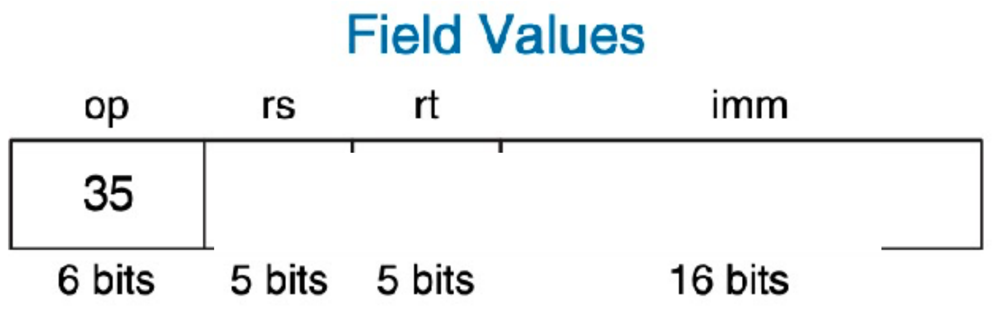
Single-Cycle Datapath: lw address

add the base address to the offset to find the address to read from memory

STEP 4: Compute the memory address

```
lw $s3, -24($s4)
```

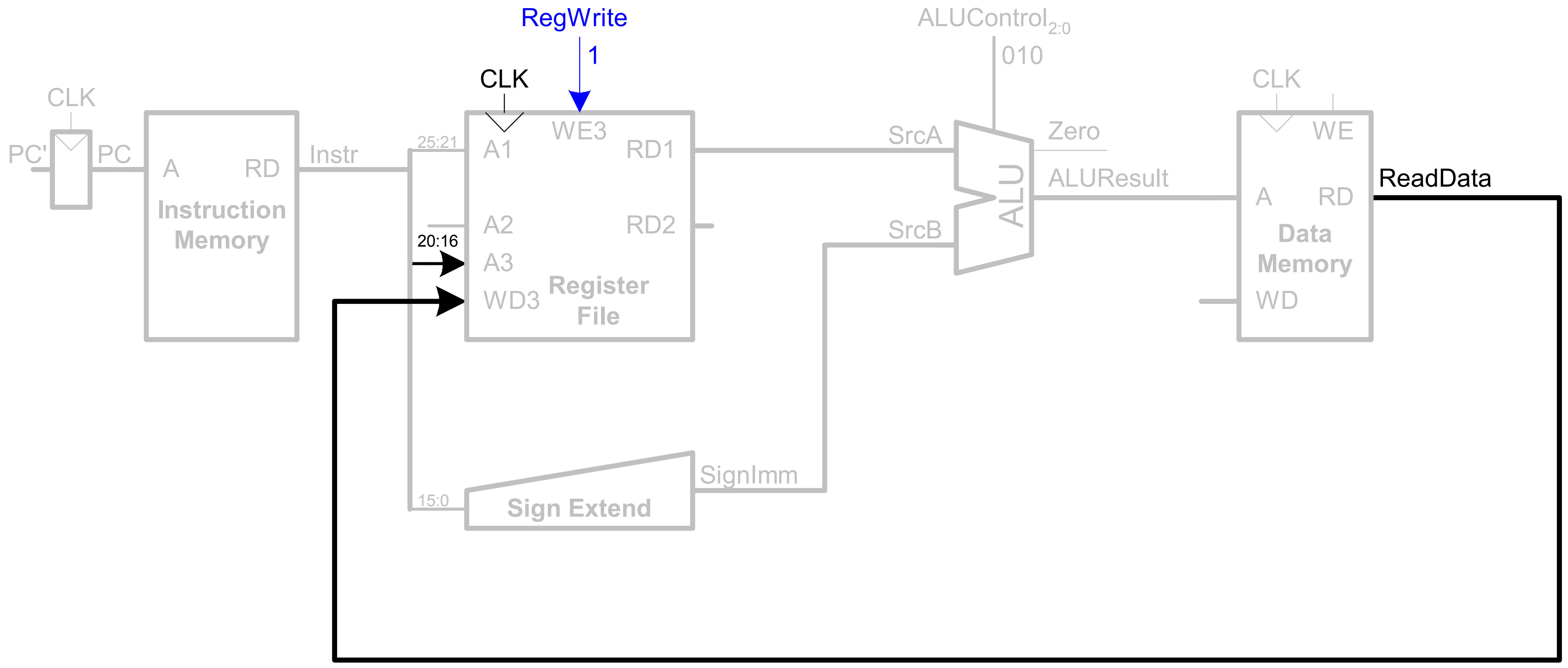




$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\bar{B}$
101	A OR $\bar{B}$
110	A − B
111	SLT

Single-Cycle Datapath: lw Memory Read

- **STEP 5:** Read data from memory and write it back to register file


RegWrite
1
CLK
ALUControl$_{2:0}$
010
CLK
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
A1
WE3
RD1
A2
RD2
20:16
A3
WD3
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WE
A
RD
Data Memory
WD
ReadData
15:0
Sign Extend
SignImm