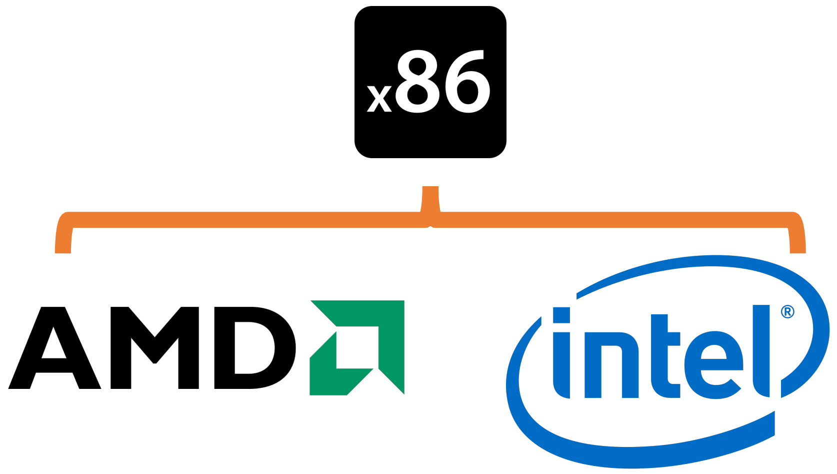
MIPS Microarchitecture

Recall

Microarchitecture: Specific arrangement of registers, memories, and ALUs.

Connection between logic and architecture.

Many different microarchitectures for a single architecture.



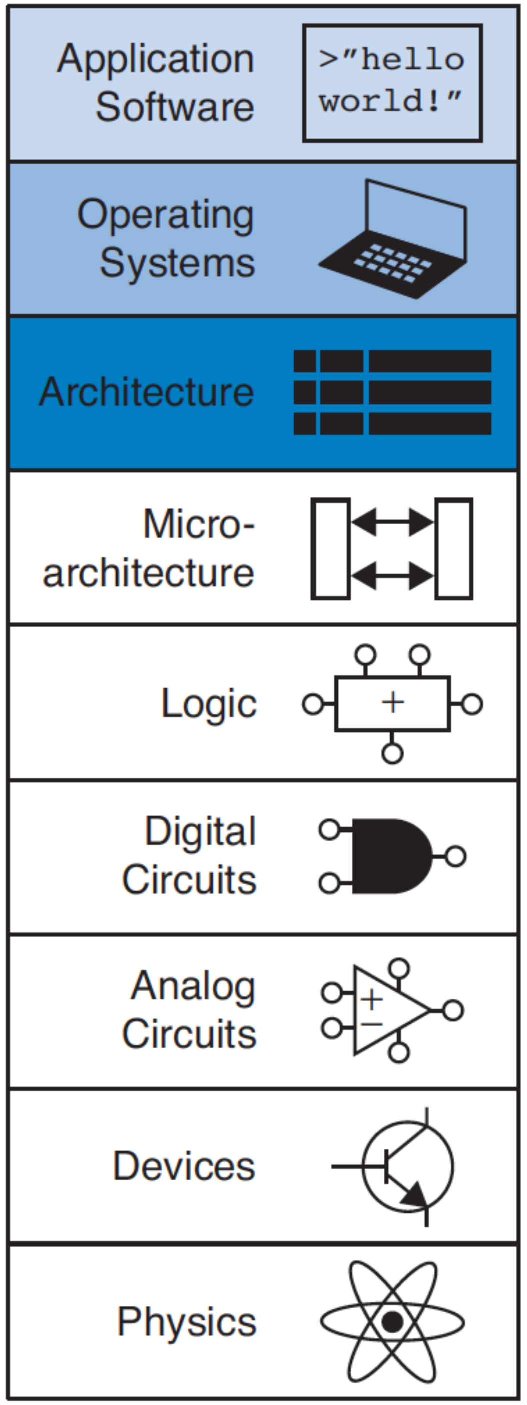
Recall

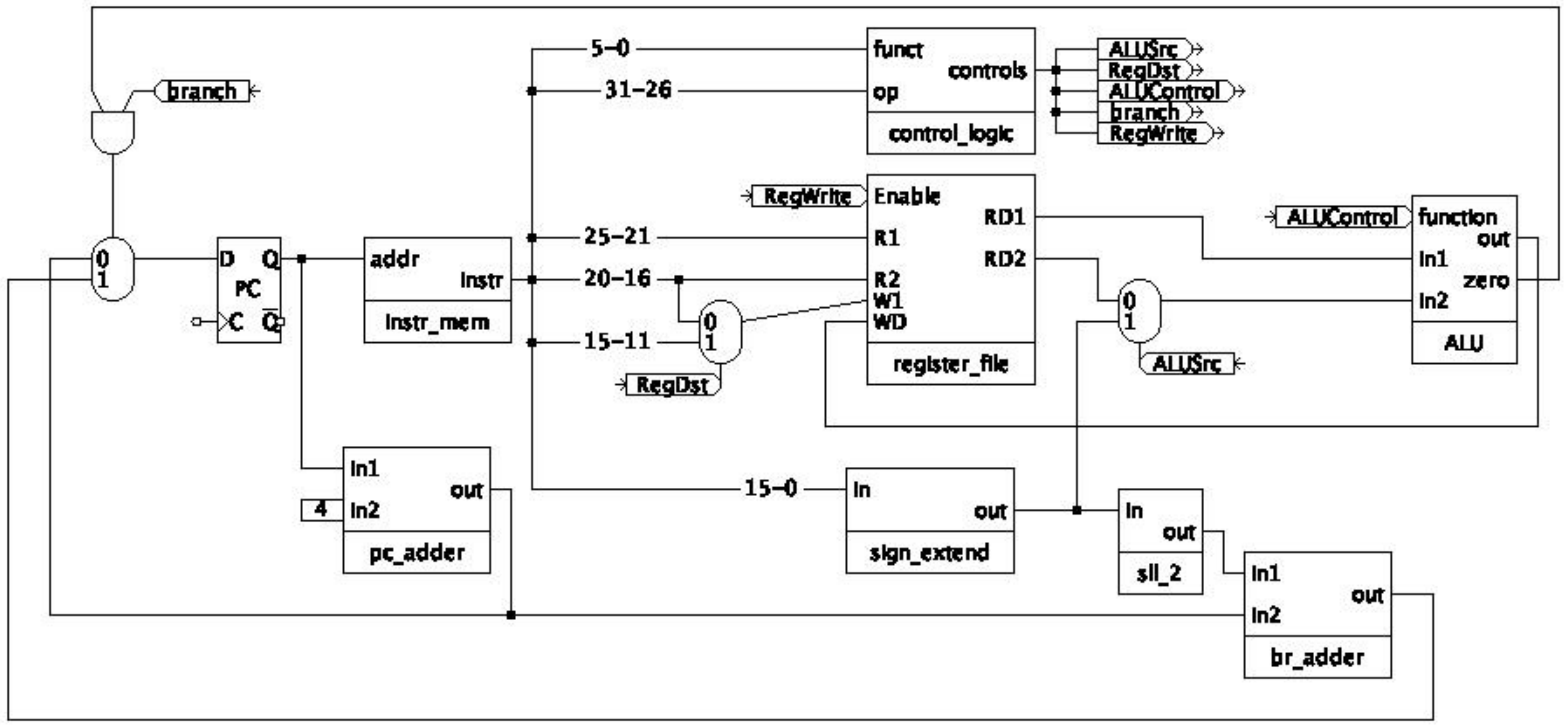
Architecture: Programmer's view of a computer

Abstracts away from low-level logic.

Defined by the instruction set (language) and architectural state (program counter and 32 registers)

x86, MIPS, SPARC, PowerPC




branch
0
1
D
Q
PC
C
Q
addr
Instr
instr_mem
5–0
31–26
25–21
20–16
15–11
15–0
funct
op
controls
control_logic
ALUSrc
RegDst
ALUControl
branch
RegWrite
RegWrite
Enable
R1
R2
W1
WD
RD1
RD2
register_file
0
1
RegDst
0
1
ALUSrc
ALUControl
function
out
In1
zero
In2
ALU
In1
4
In2
out
pc_adder
In
out
sign_extend
In
out
sll_2
In1
In2
out
br_adder

Load and Store

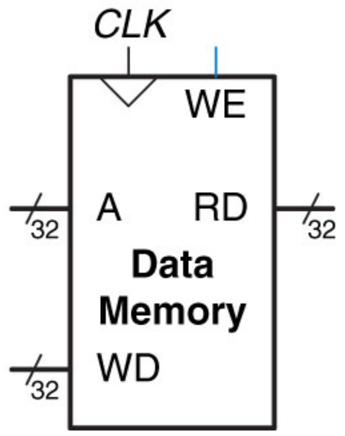
One last thing to implement in our microarchitecture: load and store of data memory

A few instructions (notably, jump) not included in this microarchitecture for simplicity

Should understand how you would add them if required (e.g., on homework or exam)

Load and Store

Can't have load and store without having memory



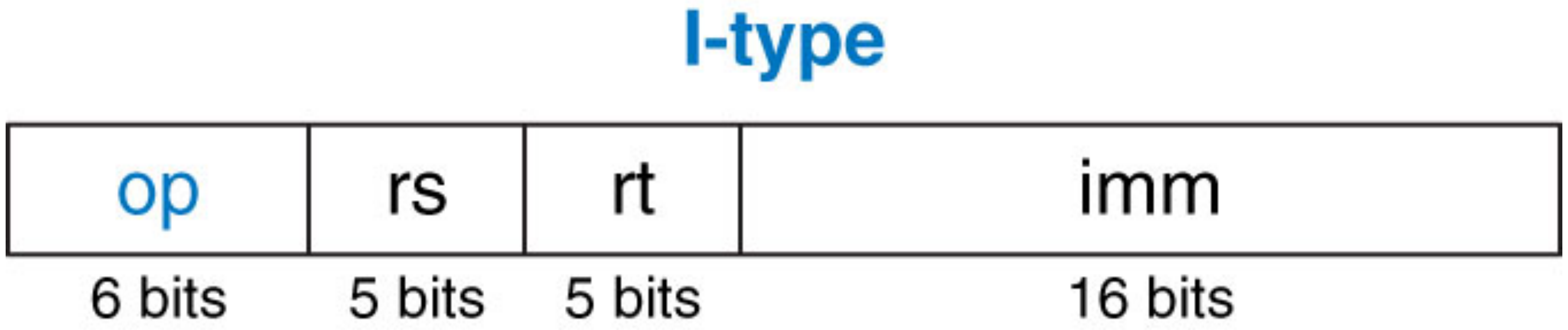
Load and Store

Load consists of three parts:

- Read base address from register
- Add offset to produce full address
- Read from memory and store into register

Load and Store

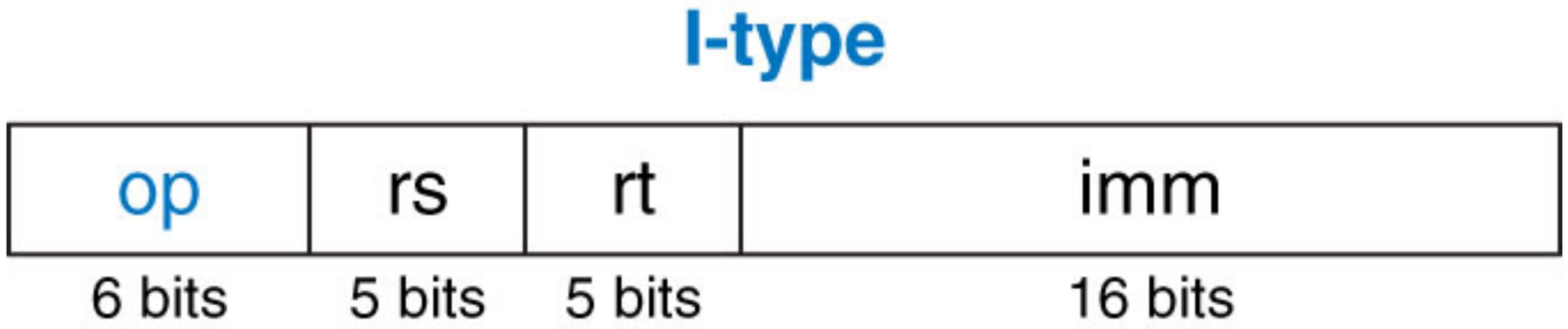
`lw` and `sw` each have two registers and an offset, so it may not surprise you to learn they are I-type instructions



Load and Store

`rs` always base register for address

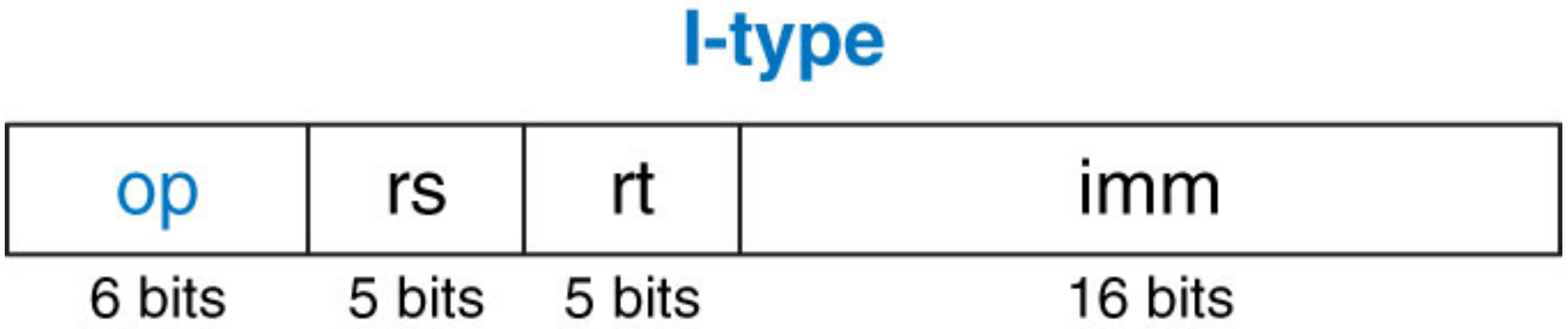
Meaning of `rt` changes depending whether we are loading or storing



Load and Store

`rs` always base register for address

Meaning of `rt` changes depending whether we are loading or storing



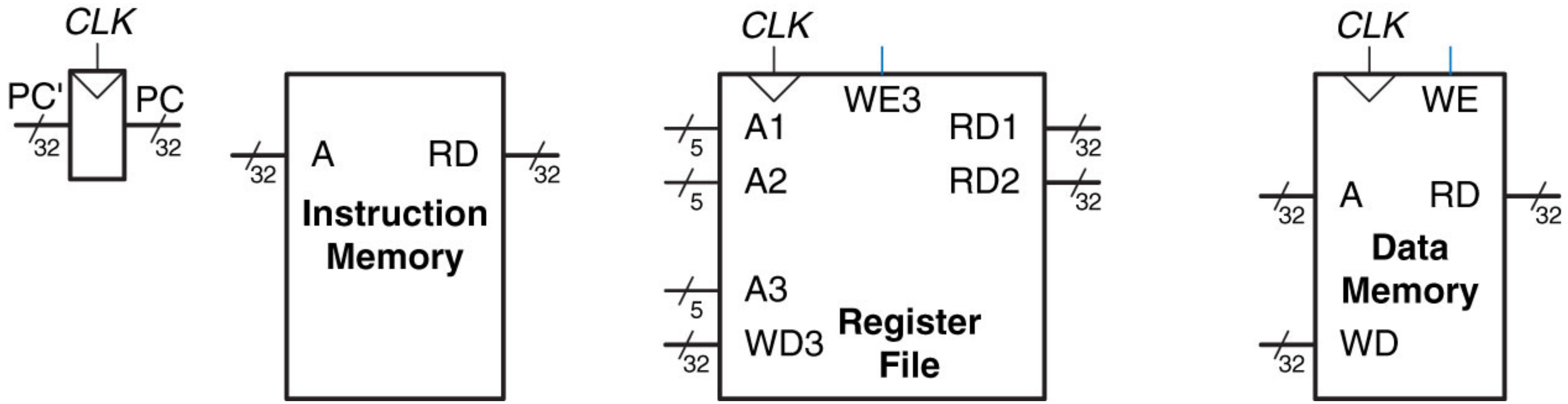
Implementation

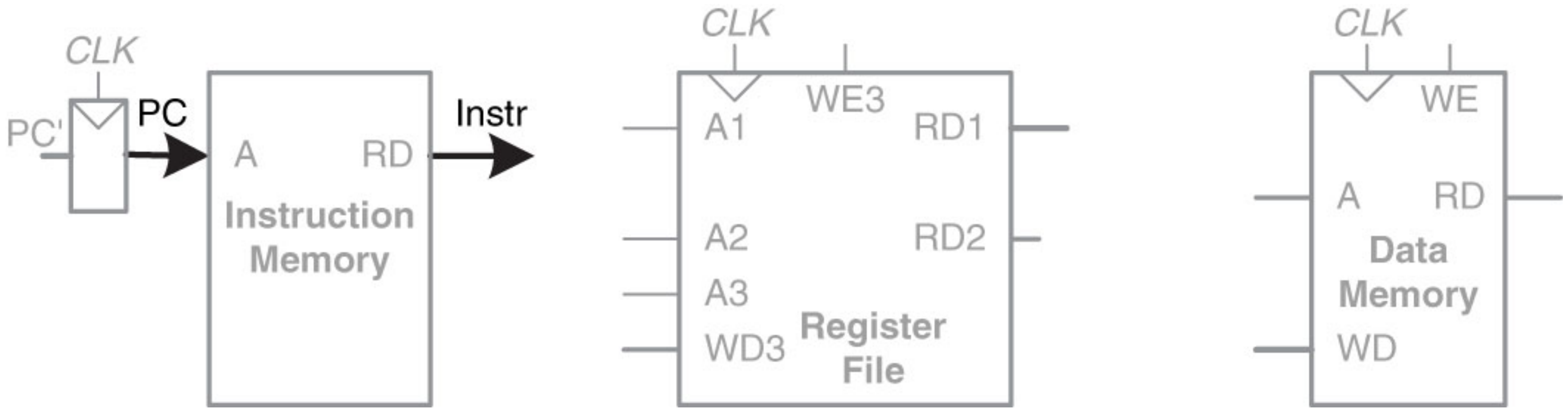
Start from scratch to implement load and store, then add other functionality back in

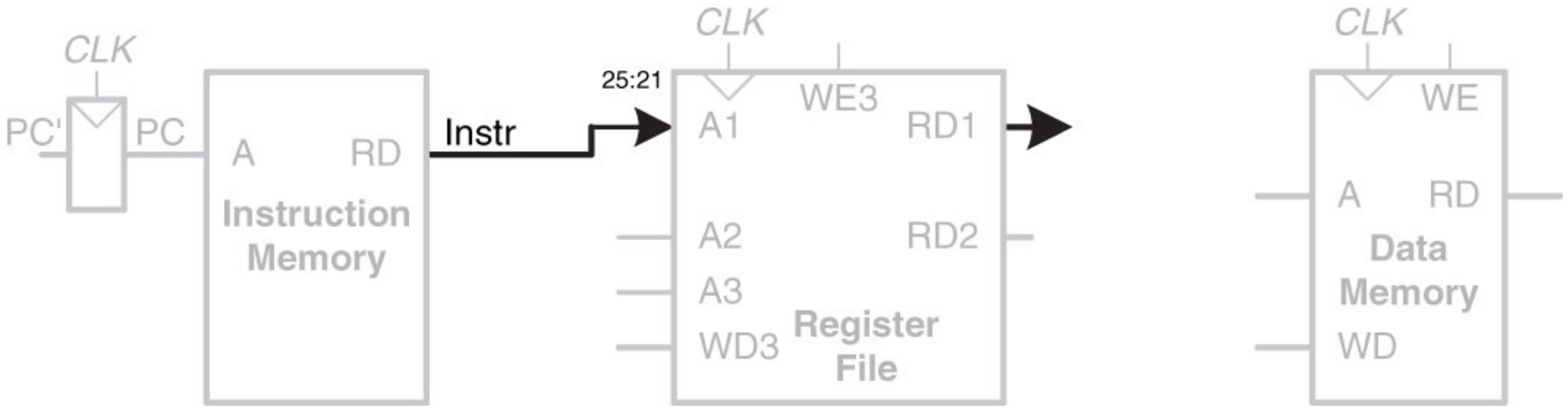
Could add new behavior to previous diagram, but, frankly, book already has nice diagrams for doing it this way

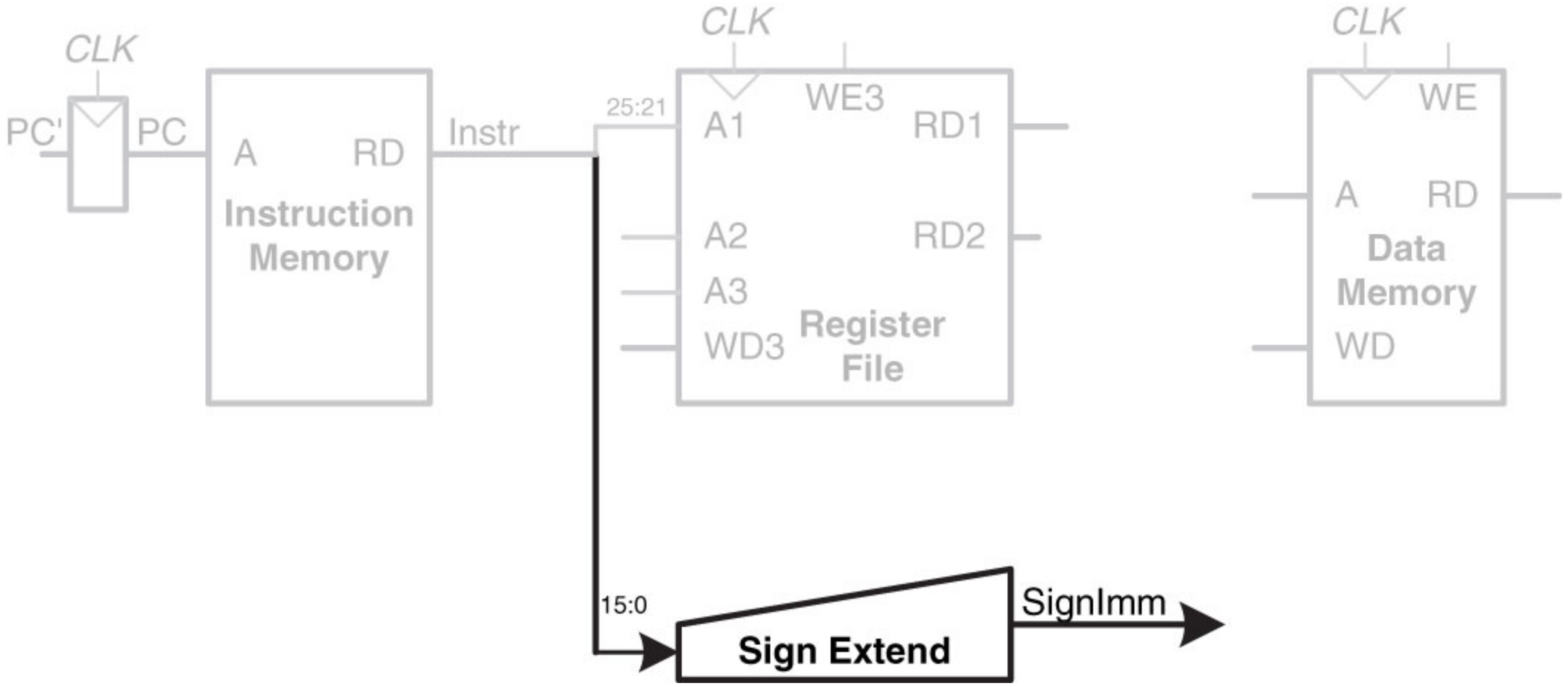
We will contrast with previous microarchitecture at the end and see that differences are minimal

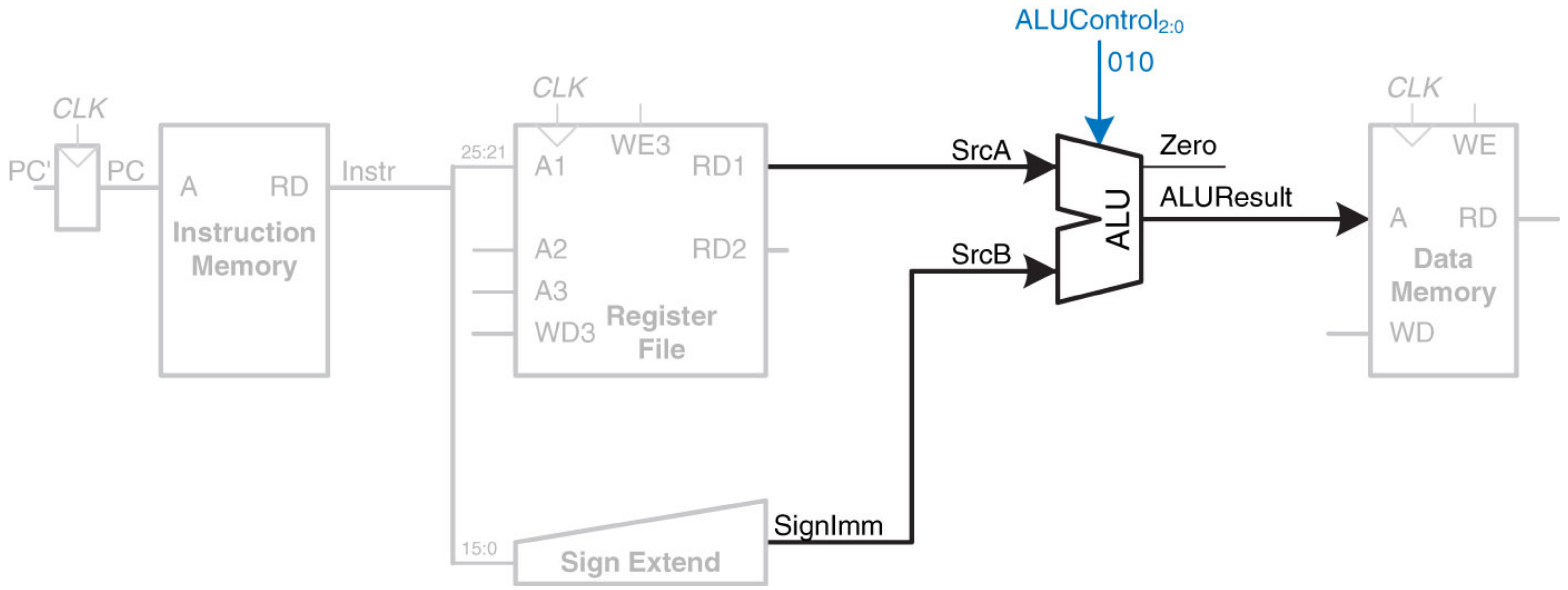
Single-Cycle CPU



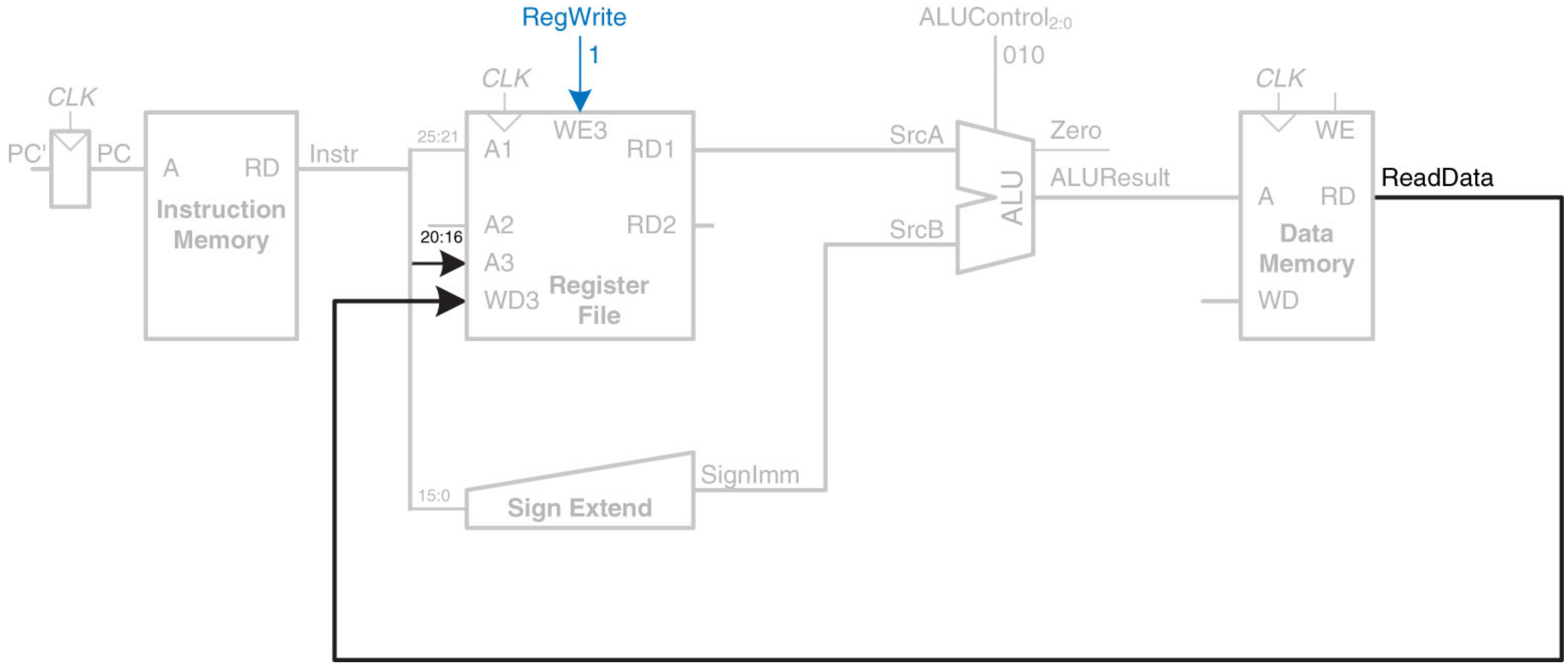

CLK
PC'
PC
A
RD
Instruction Memory
Instr
CLK
WE3
A1
RD1
A2
RD2
A3
WD3
Register File
CLK
WE
A
RD
Data Memory
WD

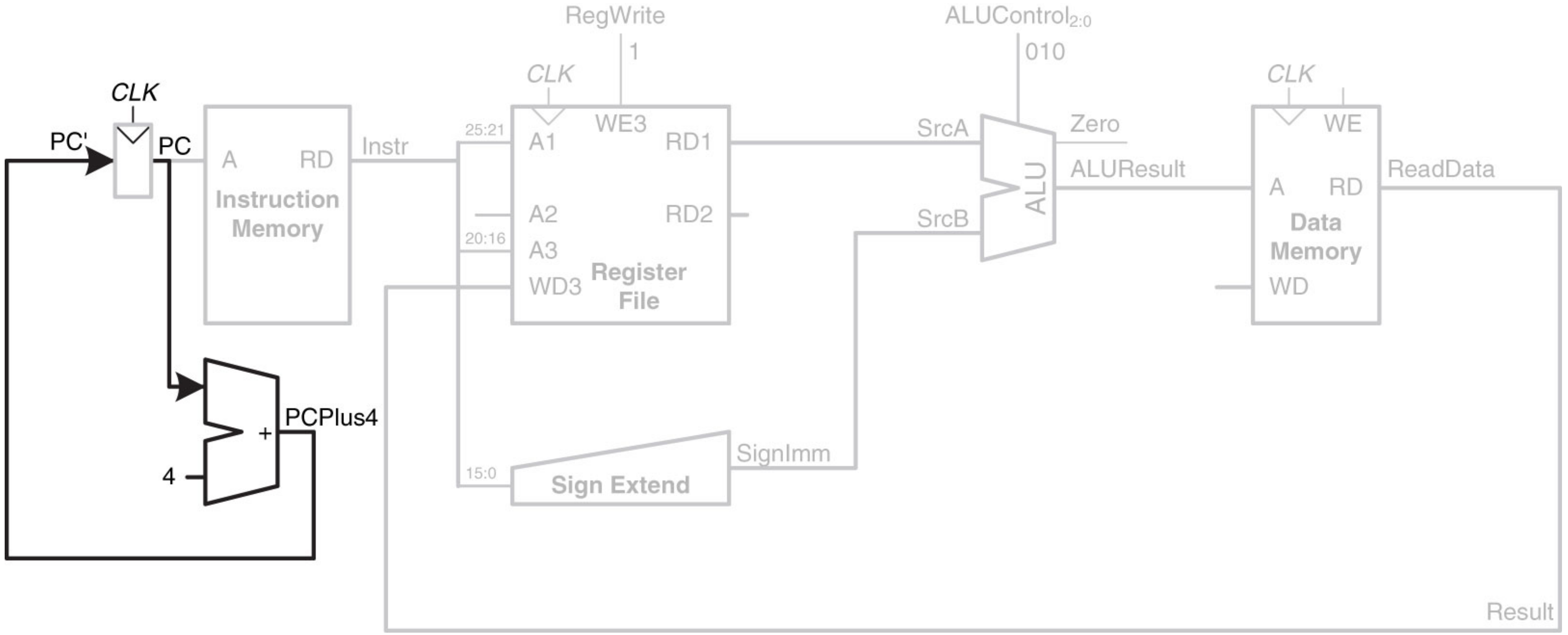

CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
CLK
WE3
A1
RD1
A2
RD2
A3
WD3
Register File
CLK
WE
A
RD
Data Memory
WD


CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
CLK
WE3
A1
RD1
A2
RD2
A3
WD3
Register File
CLK
WE
A
RD
Data Memory
WD
15:0
Sign Extend
SignImm



ALUControl2:0
010
CLK
CLK
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
A1
WE3
RD1
A2
RD2
A3
WD3
Register File
SrcA
SrcB
Zero
ALU
ALUResult
WE
A
RD
Data Memory
WD
15:0
Sign Extend
SignImm

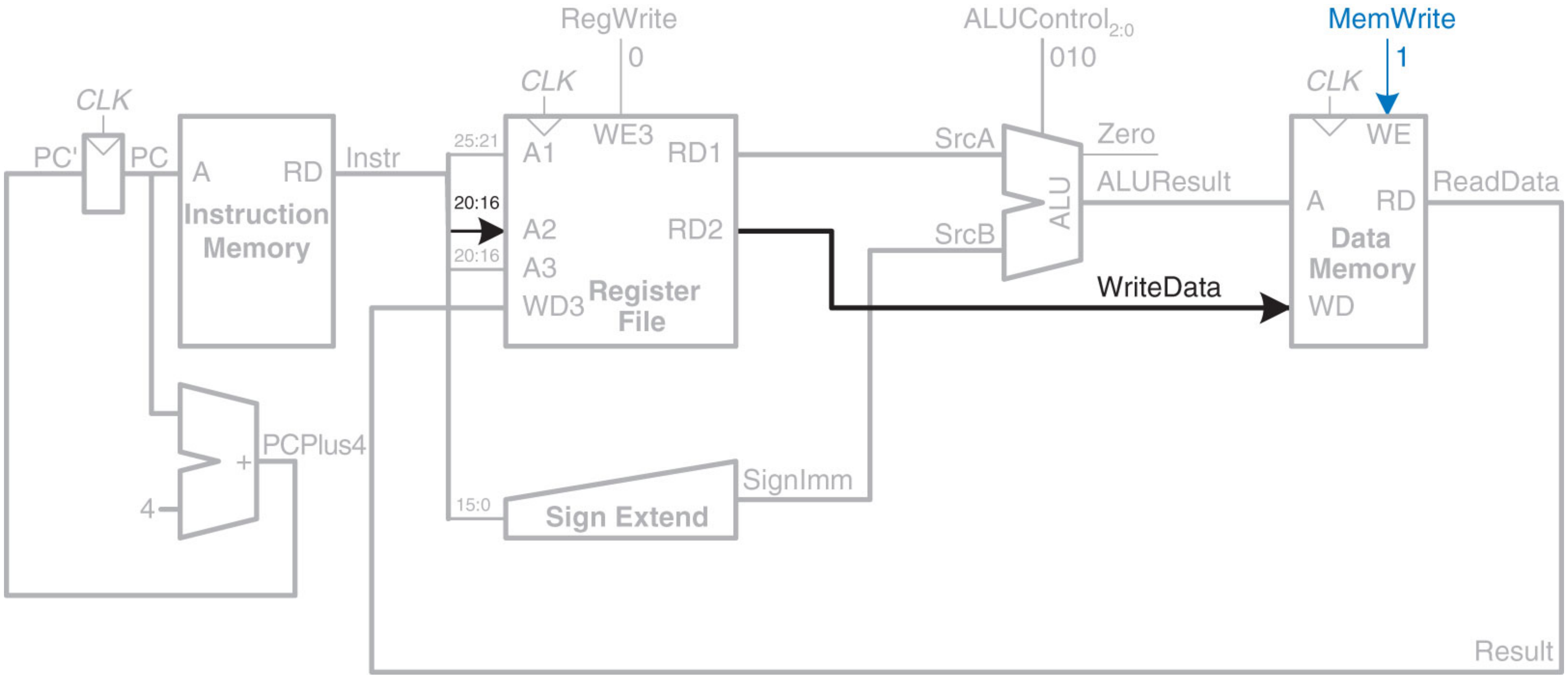

RegWrite
1
ALUControl$_{2:0}$
010
CLK
CLK
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
A1
WE3
RD1
SrcA
Zero
WE
ALUResult
ALU
ReadData
A2
RD2
SrcB
A
RD
Data Memory
20:16
A3
WD3
Register File
WD
SignImm
15:0
Sign Extend


CLK
PC'
PC
A
RD
Instruction Memory
Instr
PCPlus4
+
4
RegWrite
1
CLK
25:21
A1
WE3
RD1
A2
RD2
20:16
A3
WD3
Register File
15:0
Sign Extend
SignImm
ALUControl$_{2:0}$
010
SrcA
SrcB
ALU
Zero
ALUResult
CLK
WE
A
RD
Data Memory
WD
ReadData
Result

Adding store instruction

Previous diagram implemented `lw` instruction

As always, keep existing wires and use mux when implementing `sw` instruction



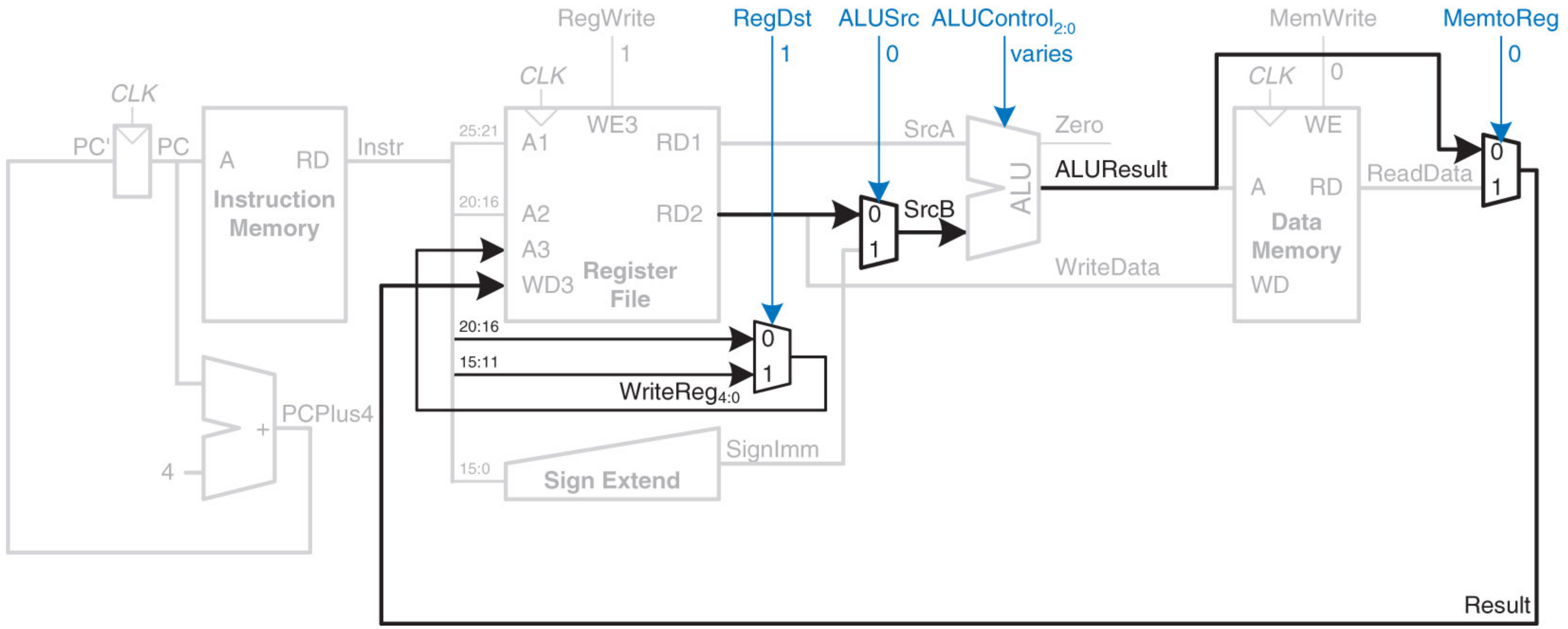
RegWrite
0
ALUControl$_{2:0}$
010
MemWrite
1
CLK
CLK
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
20:16
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WE
A
RD
Data Memory
WD
ReadData
WriteData
PCPlus4
4
+
Sign Extend
SignImm
Result

Changes

Note the new `MemWrite` control

Needed for same reason as `RegWrite` – not all instructions write to memory, so we need to stop memory from updating when not appropriate

What about R-type instructions?


RegWrite
1
RegDst
1
ALUSrc
0
ALUControl$_{2:0}$
varies
MemWrite
0
MemtoReg
0
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
A
RD
WE
WD
Data Memory
ReadData
20:16
15:11
WriteReg$_{4:0}$
PCPlus4
4
15:0
Sign Extend
SignImm
Result

R-type

Already saw `RegDst` and `ALUSrc` muxes before

New mux is `MemtoReg`, which allows us to choose whether we want ALU result or data read from memory

We cannot stop reading from memory, but we can ignore it in favor of ALU result (R-type) or prevent registers from changing at all (`sw`) using control signals

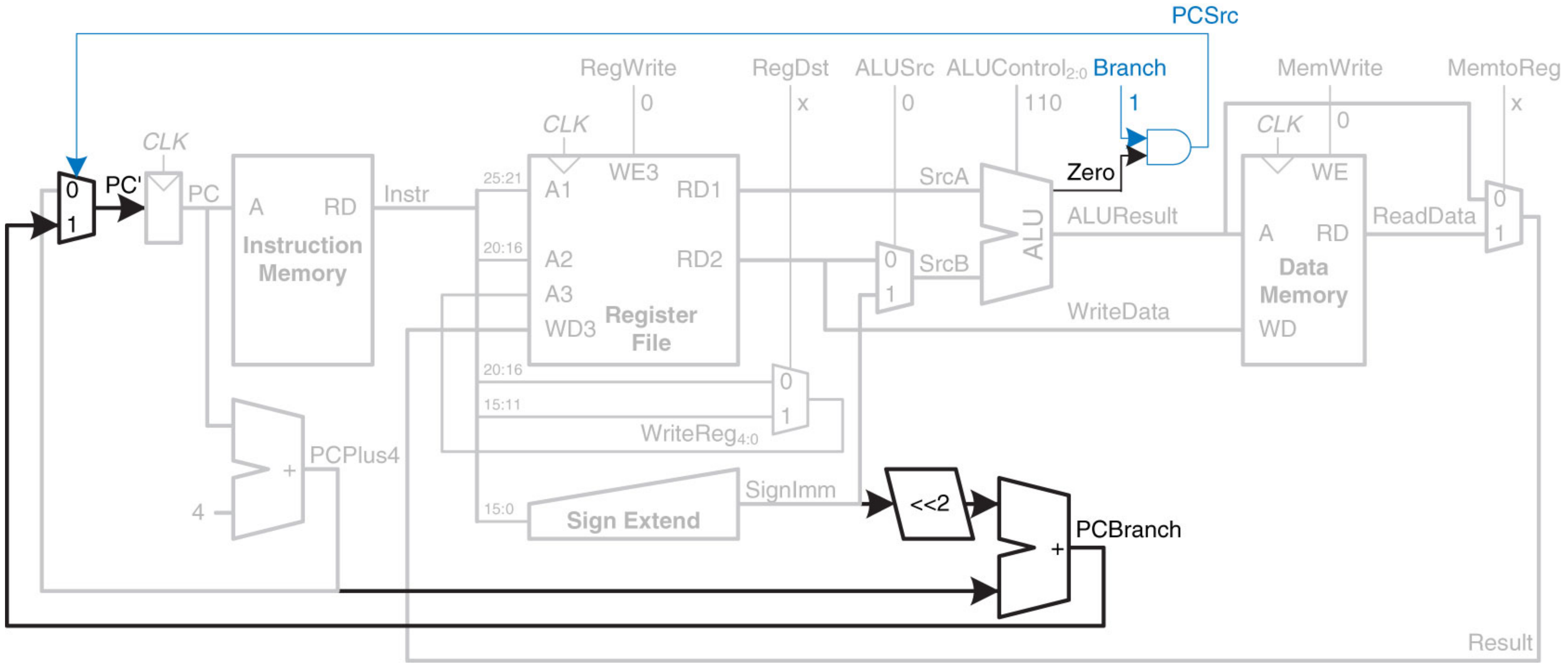
With `MemWrite = 0` and `MemtoReg = 0`, it is as though memory is not there at all

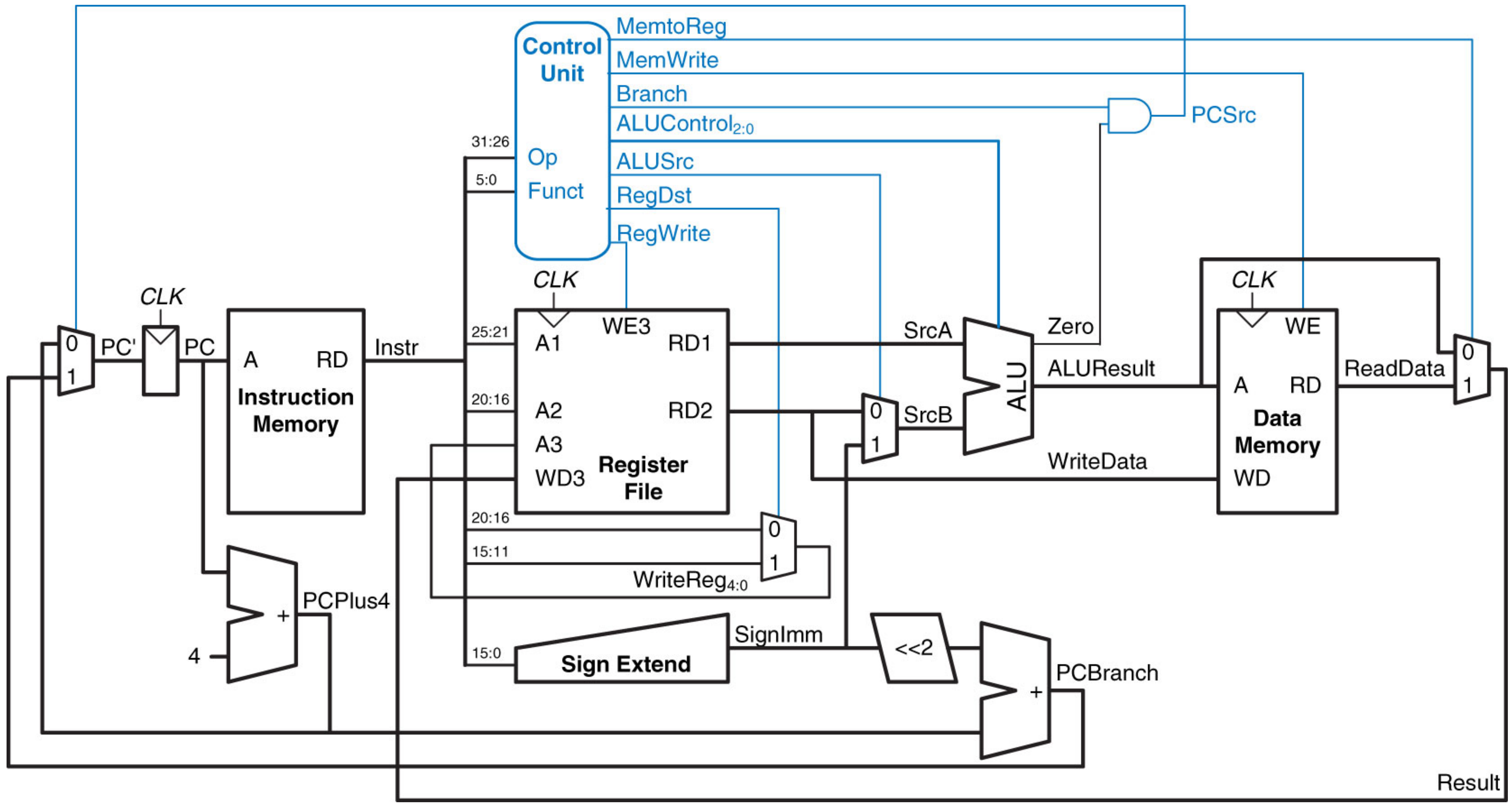
Previous diagram works for load, store, and R-type instructions

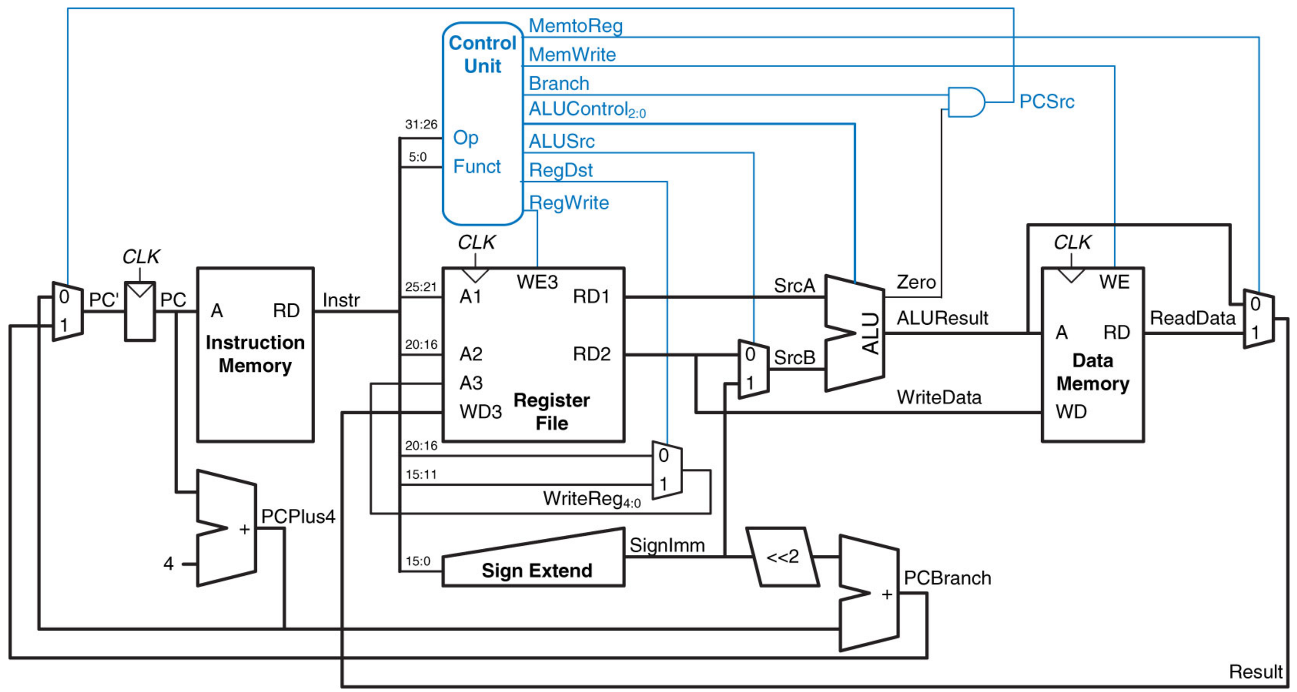
Note that we also picked up `addi` and friends for free along the way

- Load/store instructions gave us path for performing arithmetic on register and immediate
- R-type instructions allowed us to write ALU result to register
- Combination gives us various other I-type instructions

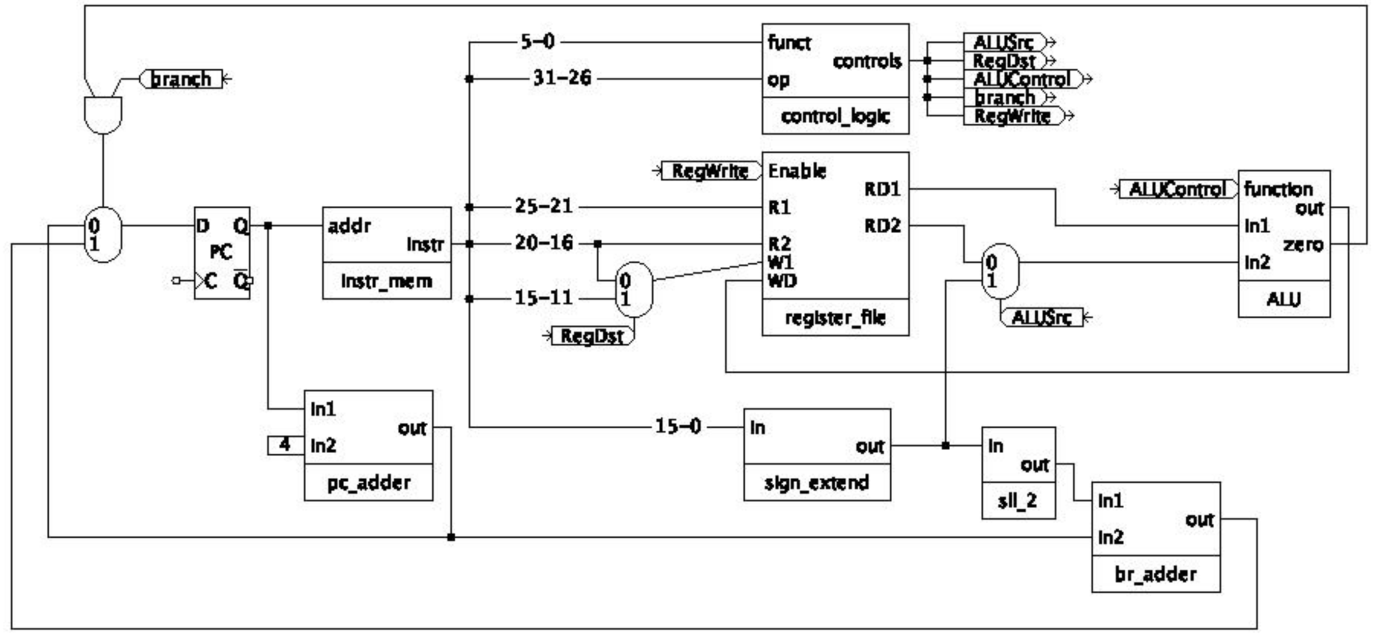
Finally, the branch instruction


PCSrc
RegWrite
RegDst
ALUSrc
ALUControl2:0
Branch
MemWrite
MemtoReg
0
x
0
110
1
0
x
CLK
CLK
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
Zero
ALU
ALUResult
WriteData
A
RD
WE
WD
Data Memory
ReadData
20:16
15:11
WriteReg4:0
PCPlus4
4
15:0
Sign Extend
SignImm
<<2
PCBranch
Result


Control Unit
Op
Funct
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
PCSrc
31:26
5:0
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
WriteReg4:0
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
Data Memory
WE
WD
ReadData
Result
PCPlus4
4
Sign Extend
SignImm
<<2
PCBranch
0
1
+



Control Unit
MemtoReg
MemWrite
Branch
ALUControl$_{2:0}$
ALUSrc
RegDst
RegWrite
PCSrc
Op
Funct
31:26
5:0
CLK
0
1
PC'
PC
A
RD
Instruction Memory
Instr
25:21
A1
WE3
RD1
20:16
A2
RD2
A3
WD3
Register File
20:16
15:11
WriteReg$_{4:0}$
15:0
Sign Extend
SignImm
<<2
PCBranch
PCPlus4
4
SrcA
SrcB
Zero
ALU
ALUResult
WriteData
WE
Data Memory
WD
ReadData
Result



branch
5-0
31-26
funct
controls
op
control_logic
ALUSrc
RegDst
ALUControl
branch
RegWrite
RegWrite
Enable
RD1
R1
RD2
R2
W1
WD
register_file
25-21
20-16
15-11
RegDst
0
1
ALUControl
function
out
In1
zero
In2
ALU
ALUSrc
D
Q
PC
C
addr
instr
instr_mem
In1
In2
out
4
pc_adder
15-0
In
out
sign_extend
In
out
sll_2
In1
out
In2
br_adder

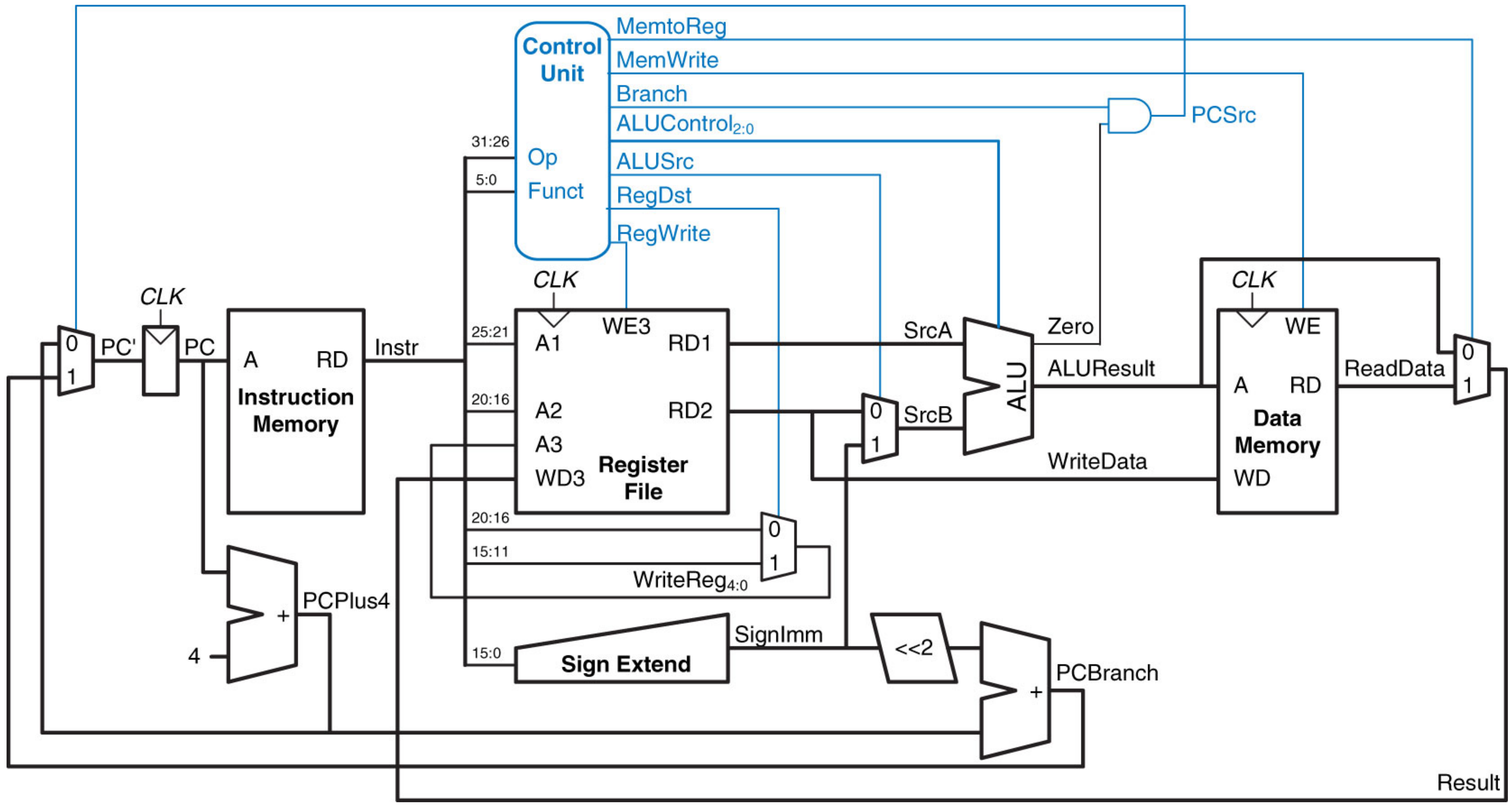
Last big question

We now have circuit that implements all* MIPS instructions

Relies heavily on muxes to do the right thing

Later lecture will examine control logic, which is just standard combinational logic

* close enough


Control Unit
Op
Funct
31:26
5:0
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
PCSrc
CLK
CLK
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
WriteReg4:0
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
A
RD
WE
WD
Data Memory
ReadData
Result
PCPlus4
4
Sign Extend
SignImm
<<2
PCBranch
0
1