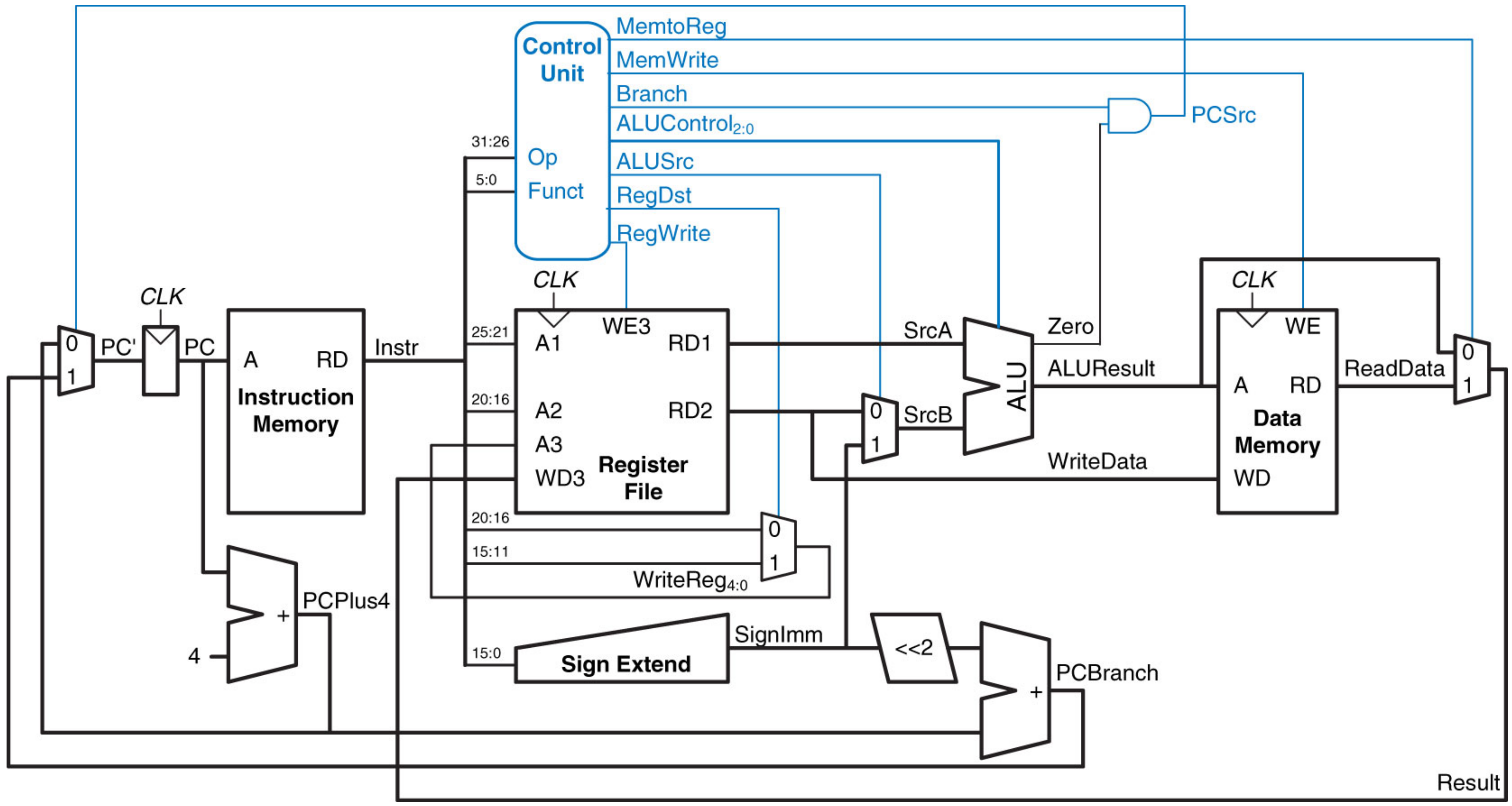
MIPS Control Logic


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
PCSrc
31:26
5:0
Op
Funct
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
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
A
RD
WD
Data Memory
ReadData
WriteReg4:0
PCPlus4
4
Sign Extend
SignImm
<<2
PCBranch
Result
0
1

Handling various instruction types

Our single circuit acts as one of several different circuits depending on instruction

When `add` instruction read, does one thing, but when `beq` instruction read, does a different thing

Muxes allow one circuit to handle all of the different MIPS instructions

Control logic tells the muxes how to behave

Implementing control logic

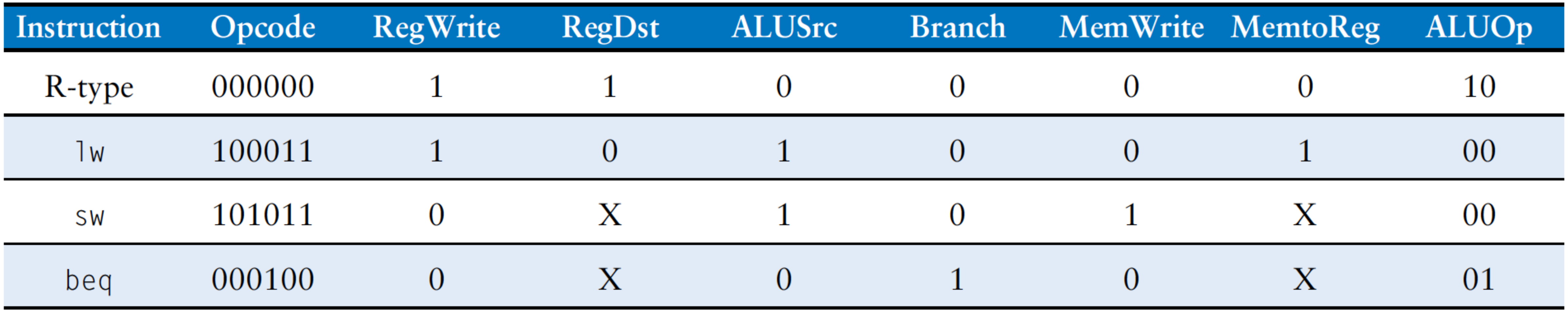
How can control logic know what to do for particular instruction?

Simple – because you know what each mux should do for particular instruction

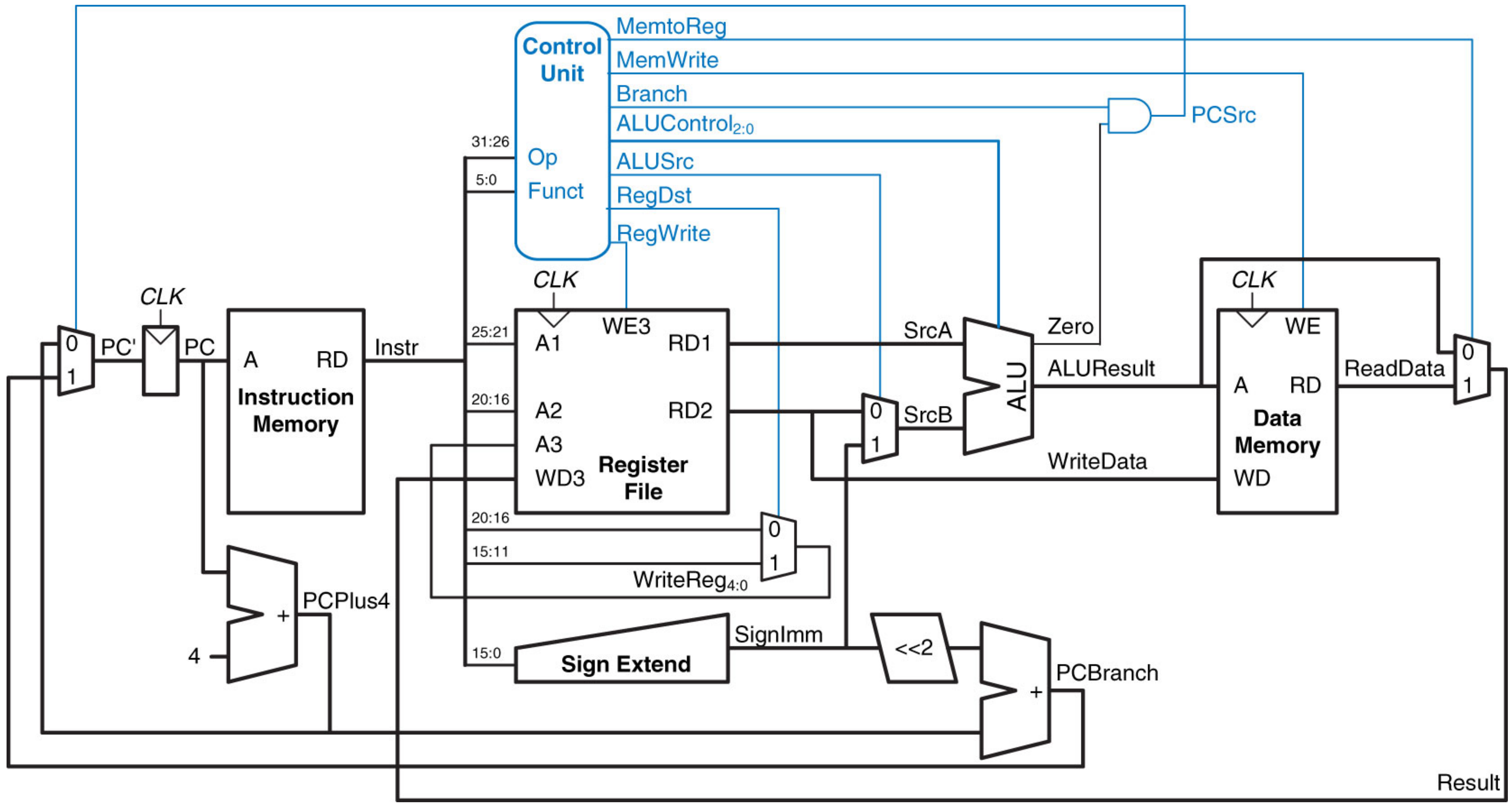
Designer essentially creates lookup table for each mux for each instruction

For example, when circuit sees opcode for `add`, sets `ALUSrc` to 0

Control Inputs



Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
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


Control Unit
Op
Funct
31:26
5:0
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
PCSrc
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
WE
Data Memory
WD
ReadData
Result
Sign Extend
SignImm
<<2
PCBranch
PCPlus4
4
+
0
1

Boolean algebra

Of course, by now we are too good at designing circuits to want to use a simple lookup table

How big would the table need to be to determine, e.g., `ALUOp` for every single opcode and function code combination?

Boolean algebra

Of course, by now we are too good at designing circuits to want to use a simple lookup table

How big would the table need to be to determine, e.g., `ALUOp` for every single opcode and function code combination?

6-bit opcode, 6-bit function code

2^{12} = 4096. We can do much better by looking for patterns

Control Inputs

First, recall something about our R-type circuits – every instruction used same datapath!

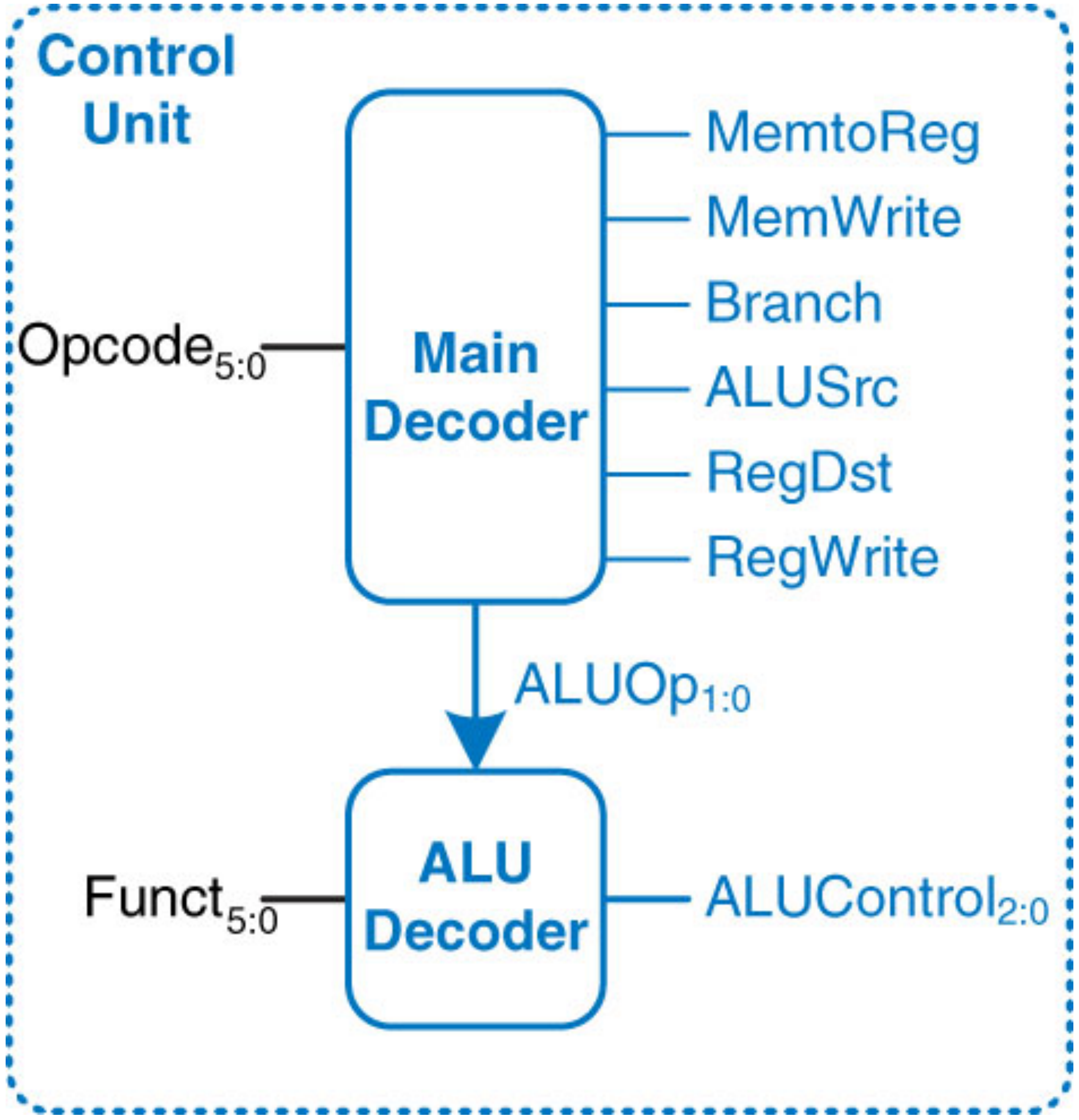
Only `ALUOp` needs to change among various R-type instructions

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
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

Control Inputs

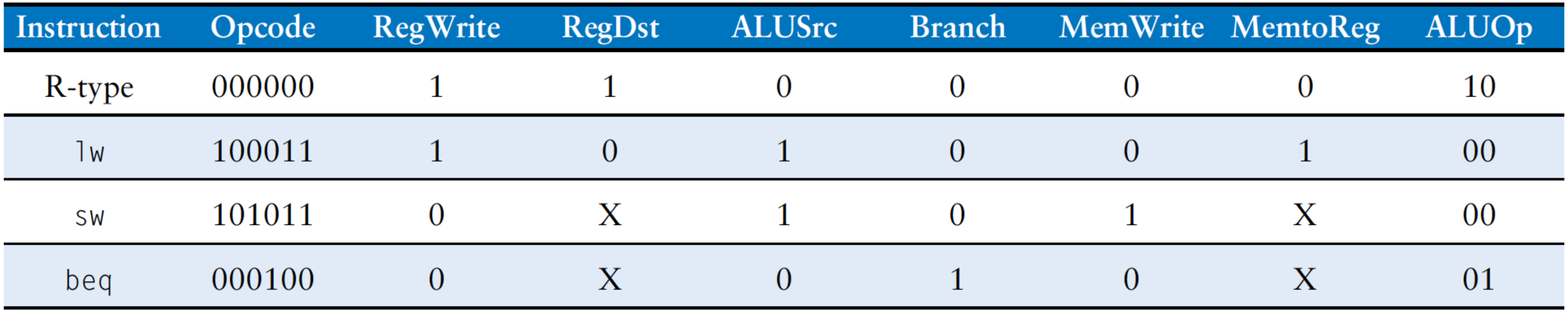
Perhaps using the same opcode for all R-type instructions was not such a strange idea after all!

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
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


Control Unit
Opcode$_{5:0}$
Main Decoder
MemtoReg
MemWrite
Branch
ALUSrc
RegDst
RegWrite
ALUOp$_{1:0}$
Funct$_{5:0}$
ALU Decoder
ALUControl$_{2:0}$

Control Inputs

We can create smaller circuits to determine value of each control wire



Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
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

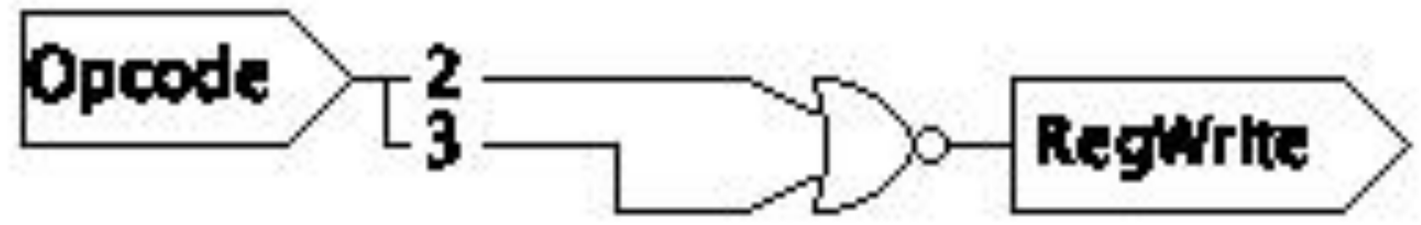
Control Inputs

We can create smaller circuits to determine value of each control wire

Assuming CPU supported only instructions listed below, how would you create circuit that takes opcode as input and outputs `RegWrite`?

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
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

RegWrite



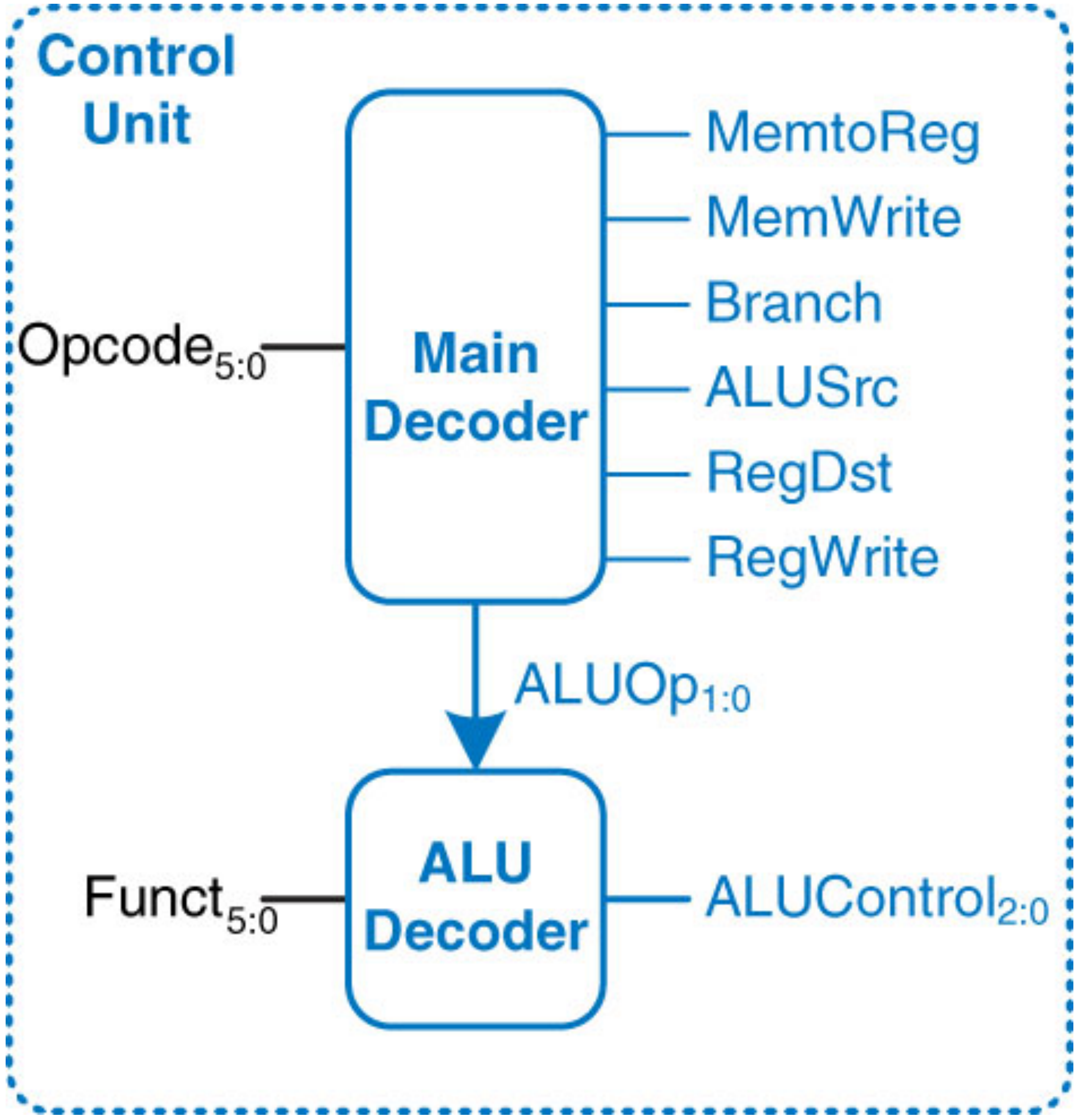
Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01

ALU decoder

ALU decoder is its own small circuit (as we saw previously)

Design in the same way – it’s all combinational circuits at this point

Has 3-bit output instead of 1-bit, but you’ve designed circuits with more than one bit of output before


Control Unit
Main Decoder
ALU Decoder
Opcode$_{5:0}$
Funct$_{5:0}$
MemtoReg
MemWrite
Branch
ALUSrc
RegDst
RegWrite
ALUOp$_{1:0}$
ALUControl$_{2:0}$

ALU Decoder

ALUOp	Meaning
00	add
01	subtract
10	look at funct field
11	n/a

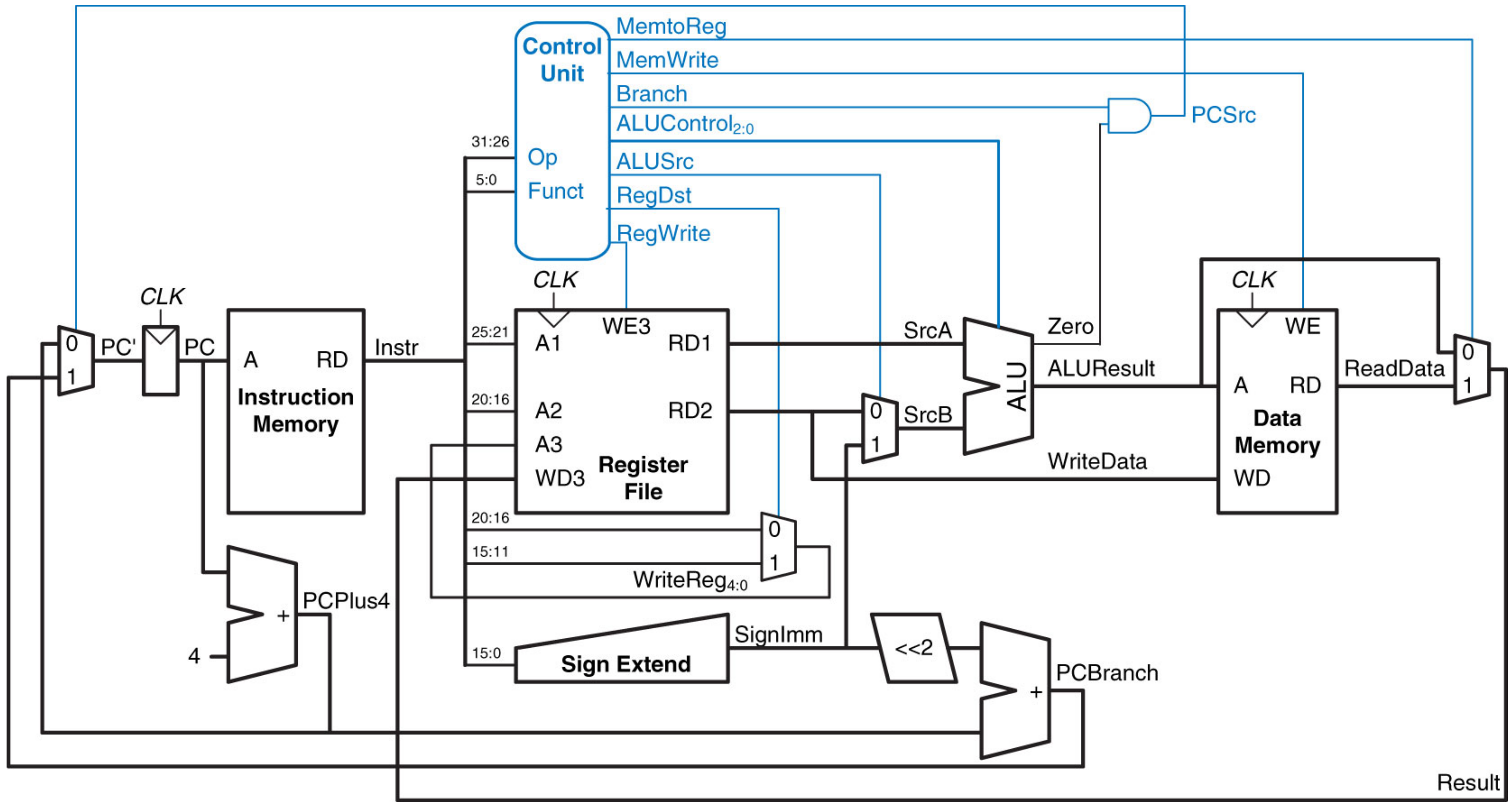
ALUOp	Funct	ALUControl
00	X	010 (add)
X1	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)

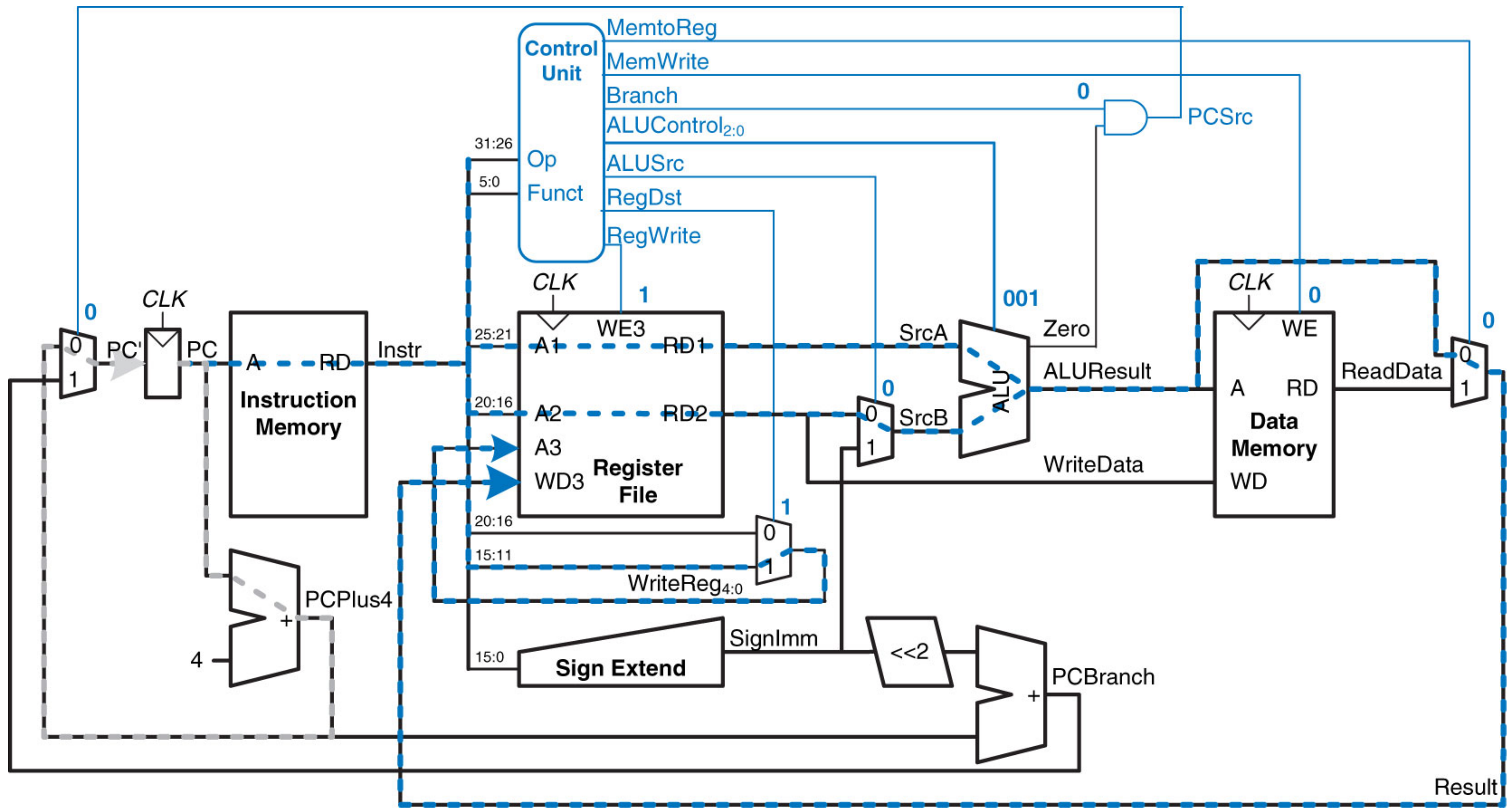
Control logic

Two keys to figuring out control logic:

1. You have to know what you want the circuit to do
2. Once you use that information to make a table, you can use all your usual tricks to turn that into a combinational circuit

OR instruction? (R-type)





Control Unit
Op
Funct
31:26
5:0
MemtoReg
MemWrite
Branch
ALUControl$_{2:0}$
ALUSrc
RegDst
RegWrite
0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
1
20:16
15:11
WriteReg$_{4:0}$
15:0
Sign Extend
SignImm
<<2
PCBranch
PCPlus4
4
SrcA
SrcB
ALU
001
Zero
ALUResult
WriteData
Data Memory
WE
WD
ReadData
Result