

Unit 4. Processor II: Design and control of the datapath. Single-cycle architecture

Escuela Politécnica Superior - UAM

Outline

- **Introduction**
- Single-cycle datapath
- Single-cycle control path
- Adding more instructions

Introduction

APPLICATION SOFTWARE	PROGRAMS
OPERATING SYSTEMS	DRIVERS
ARCHITECTURE	INSTRUCTIONS REGISTERS
MICRO-ARCHITECTURE	DATAPATH CONTROLLERS
LOGIC	ADDERS MEMORIES
DIGITAL CIRCUITS	LOGIC GATES
ANALOG CIRCUITS	AMPLIFIERS FILTERS
DEVICES	TRANSISTORS DIODES
PHYSICS	ELECTRONS

- **Architecture:**

- The programmer's view of the computer
- Defined by instructions (operations) and the operand locations (U3)

- **Microarchitecture:**

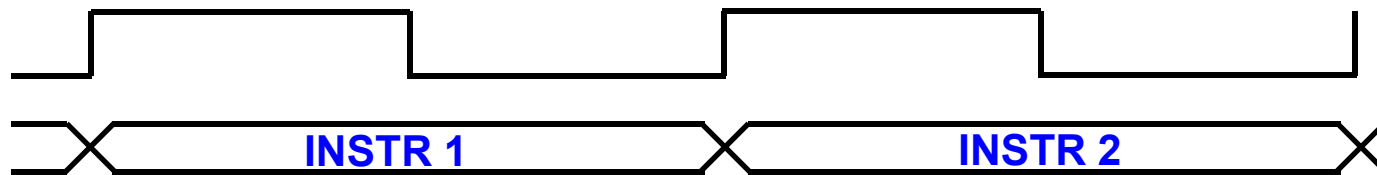
- The hardware implementation of the computer / microprocessor (U4 and U5)
- Datapath: functional blocks
- Control path: internal control signals

Microarchitecture

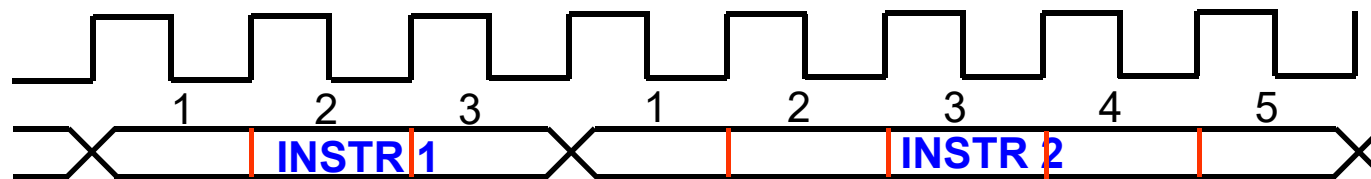
- Multiple implementations for the same architecture (instructions set):
 - Single-cycle (U4)
 - Each instruction executes in a single cycle
 - Muticycle (U5)
 - Each instruction is broken up into a series of shorter steps, getting a faster clock
 - Pipelined (3rd year)
 - Each instruction is broken up into a series of steps
 - Multiple instructions execute at once, each one in a different step

Microarchitecture

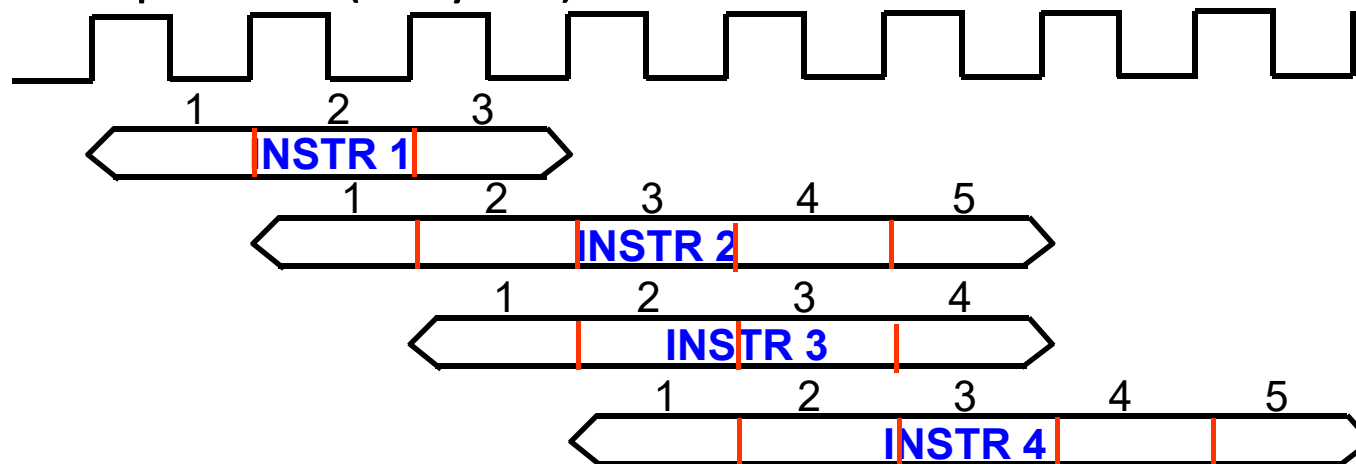
- Single-cycle (U4)



- Multicycle (U5)



- Pipelined (3rd year)



MIPS Processor

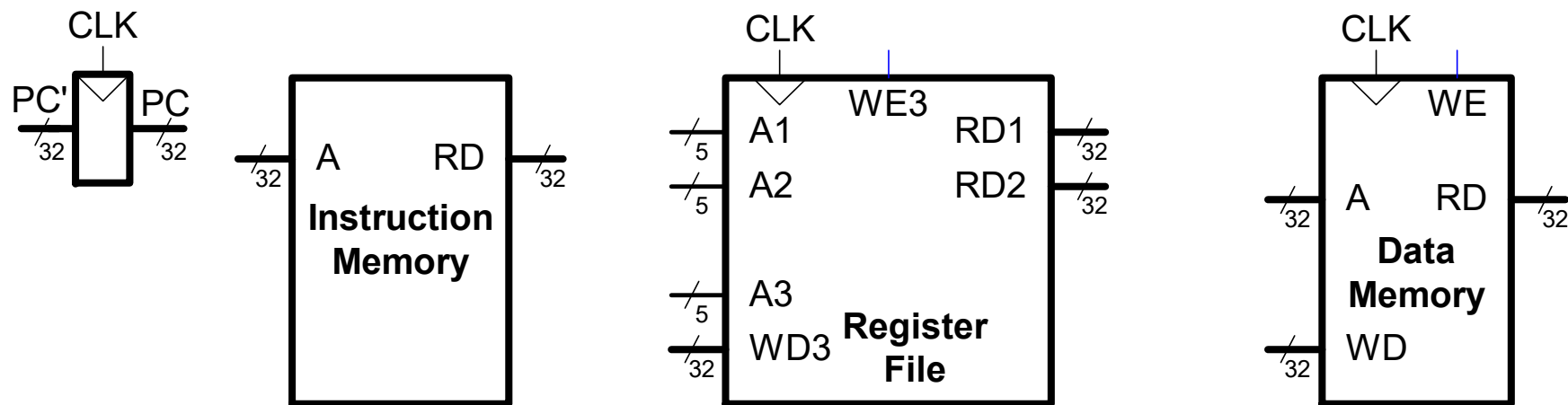
- For the microarchitecture, we consider a subset of MIPS instructions:
 - ✓ R-Type: `and`, `or`, `add`, `sub`, `slt`
 - ✓ I-Type, de memoria : `lw`, `sw`
 - ✓ I-Type, de saltos: `beq`
- Later we will add other instructions (`addi`, `j` (J-Type)).
- The subset of instructions for the lab sessions may be different.

Outline

- Introduction
- **Single-cycle datapath**
- Single-cycle control path
- Adding more instructions

Architectural State

- These elements are enough for determining the state of a processor:
 - PC
 - Register file (32 registers)
 - Memory (text and data)
- First elements to take into account:



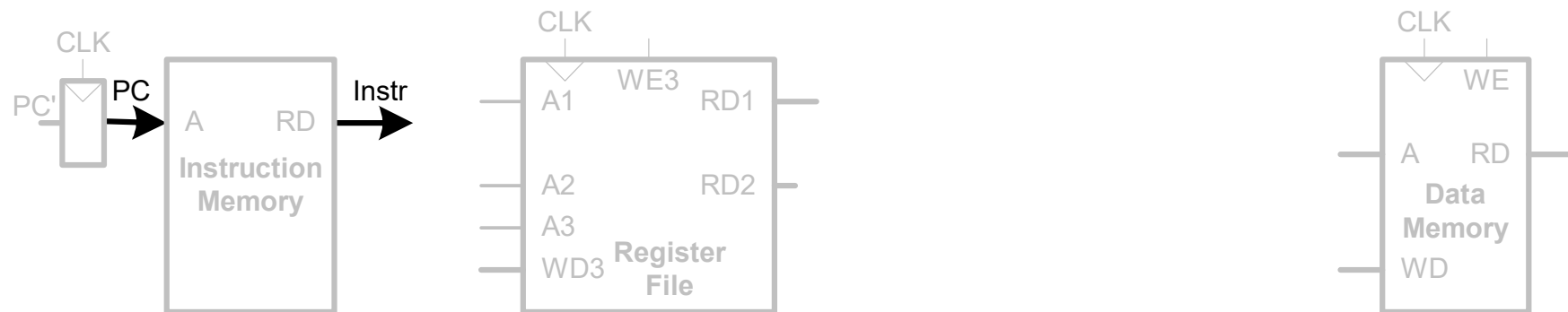
Single-Cycle Datapath: lw fetch

Let's consider the instruction:

(0x8C112000) lw \$s1, 0x2000(\$0)

and the steps to perform it:

➤ **STEP 1: fetch instruction (*fetch*)**

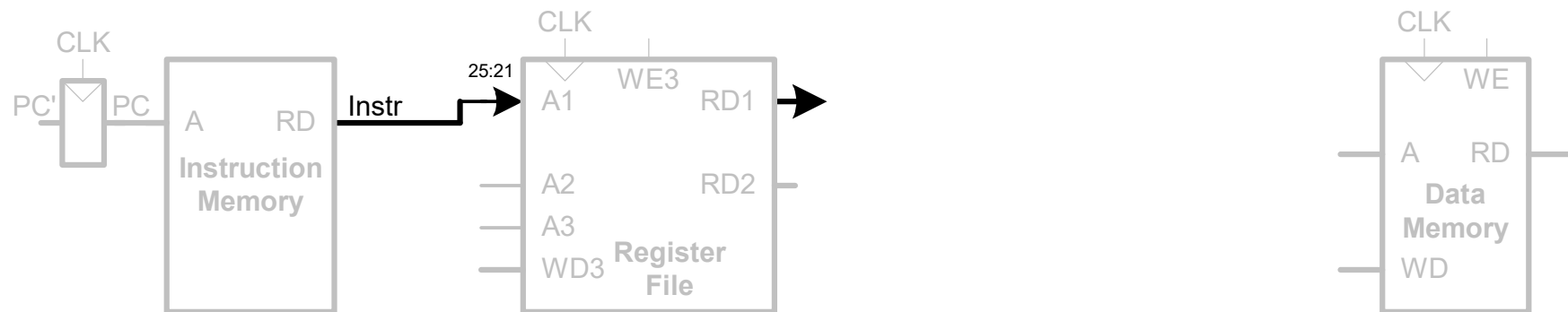


Instr <= "100011 00000 10001 0010000000000000"

op: Instr [31:26] = "100011"	=>	lw
rs: Instr [25:21] = "00000"	=>	\$0
rt: Instr [20:16] = "10001"	=>	\$s1
imm: Instr [15:0] = "0010000000000000"	=>	0x2000

Single-Cycle Datapath: `lw` register read

- **STEP 2:** read source operands from register file

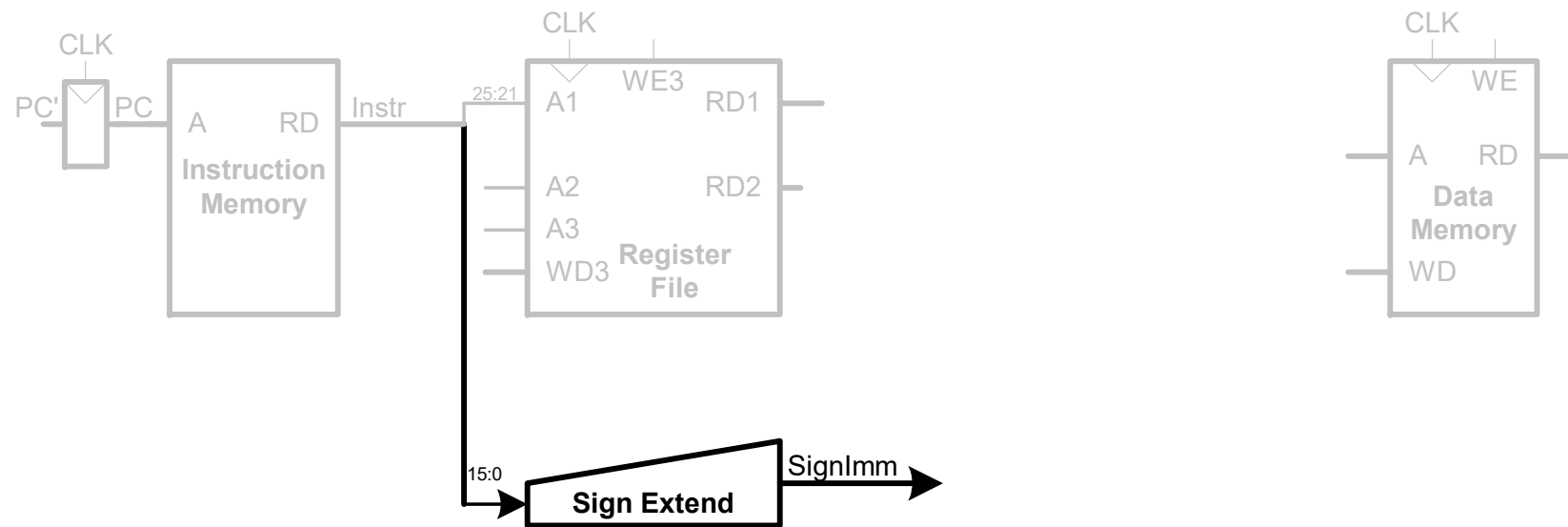


$A1: \text{Instr}[25:21] = \text{"00000"}; \quad RD1 \leq \text{"0x00000000"} = (\$0)$

`lw $s1, 0x2000($0)`

Single-Cycle Datapath: lw immediate

➤ STEP 3: sign-extend the immediate

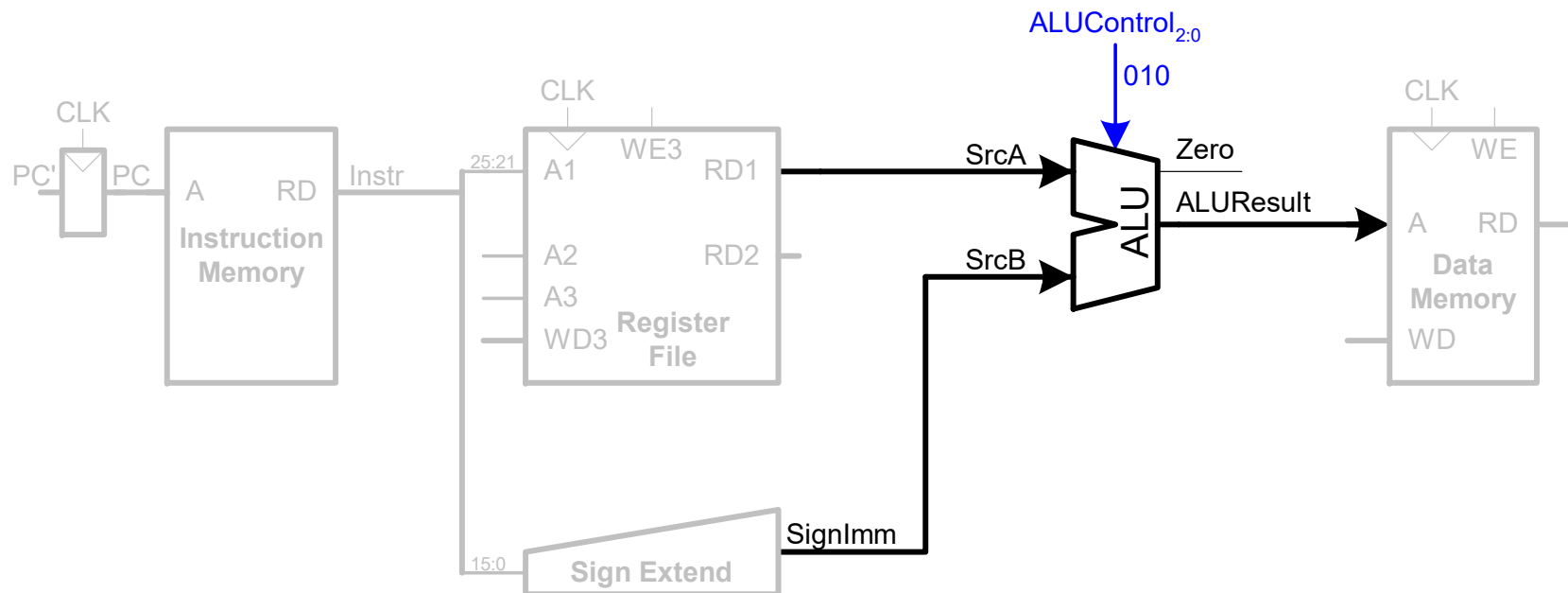


$\text{SignImm} \leq 0x00002000 = \text{Sign Extend } (0x2000)$

`lw $s1, 0x2000($0)`

Single-Cycle Datapath: lw address

- **STEP 4:** compute the memory address ($[rs] + \text{SignImm}$) in the ALU

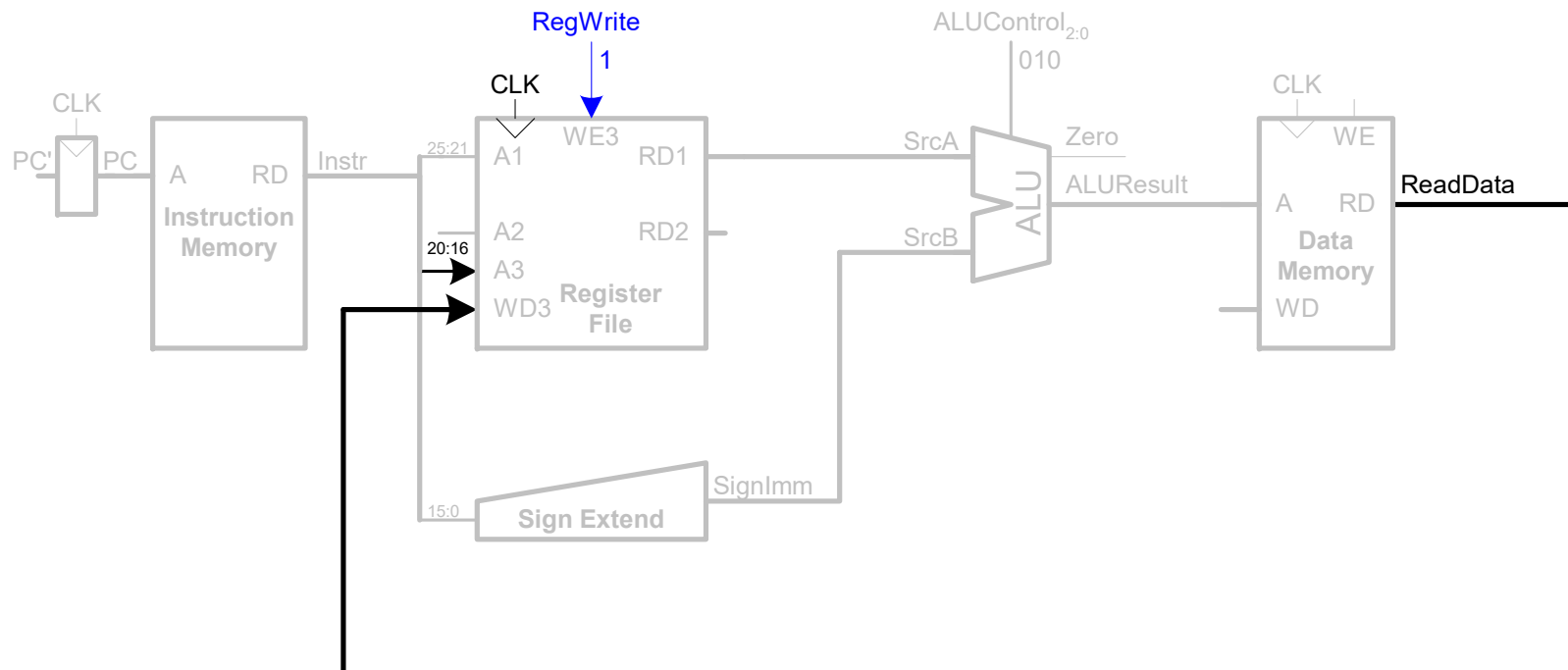


$\text{ALUResult} \leq 0x00002000 = 0x00000000 + 0x00002000$

`lw $s1, 0x2000($0)`

Single-Cycle Datapath: lw memory read

- **STEP 5:** read data from memory and write it back to register file, rt



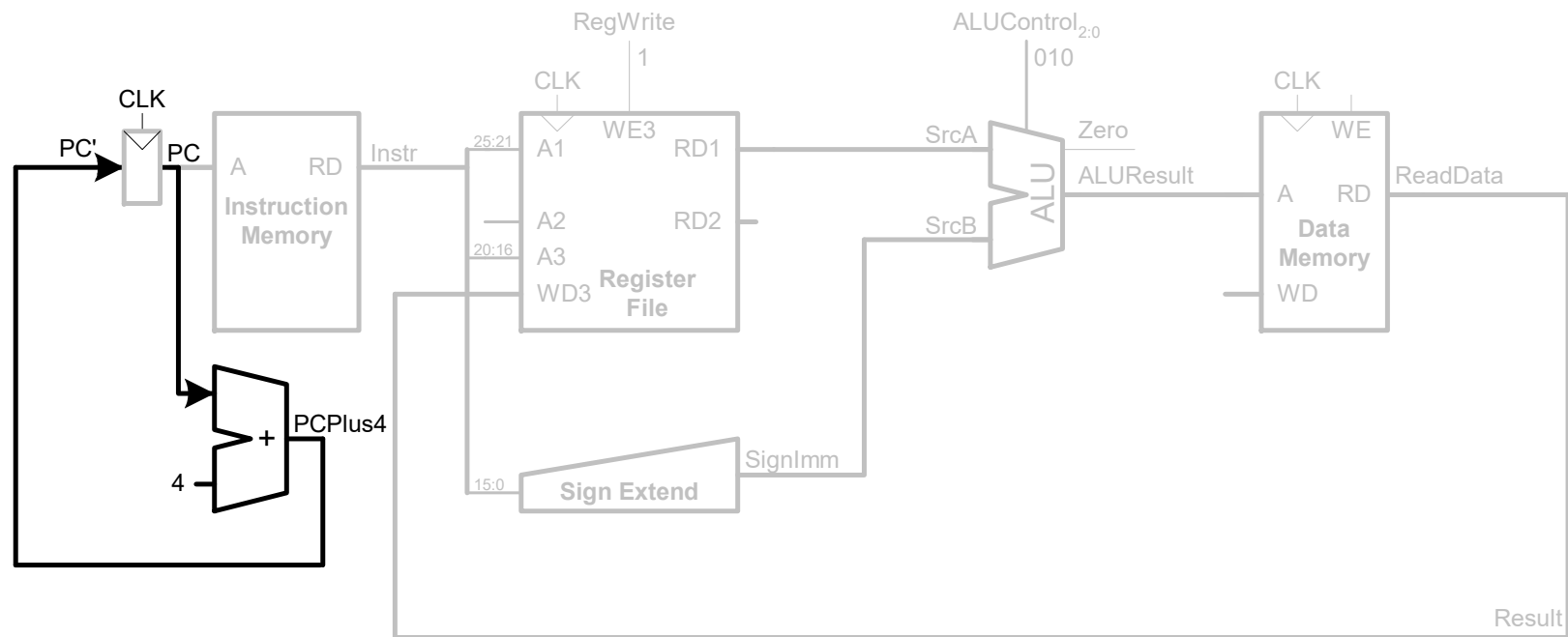
A3: Instr [20:16] = "10001" (\$s1)

\$s1 <= WD3 = MEM[0x00002000]

`lw $s1, 0x2000($0)`

Single-Cycle Datapath: lw PC increment

- **STEP 6:** determine the address of the next instruction (PC+4)



$\$PC \leq \$PC + 4$

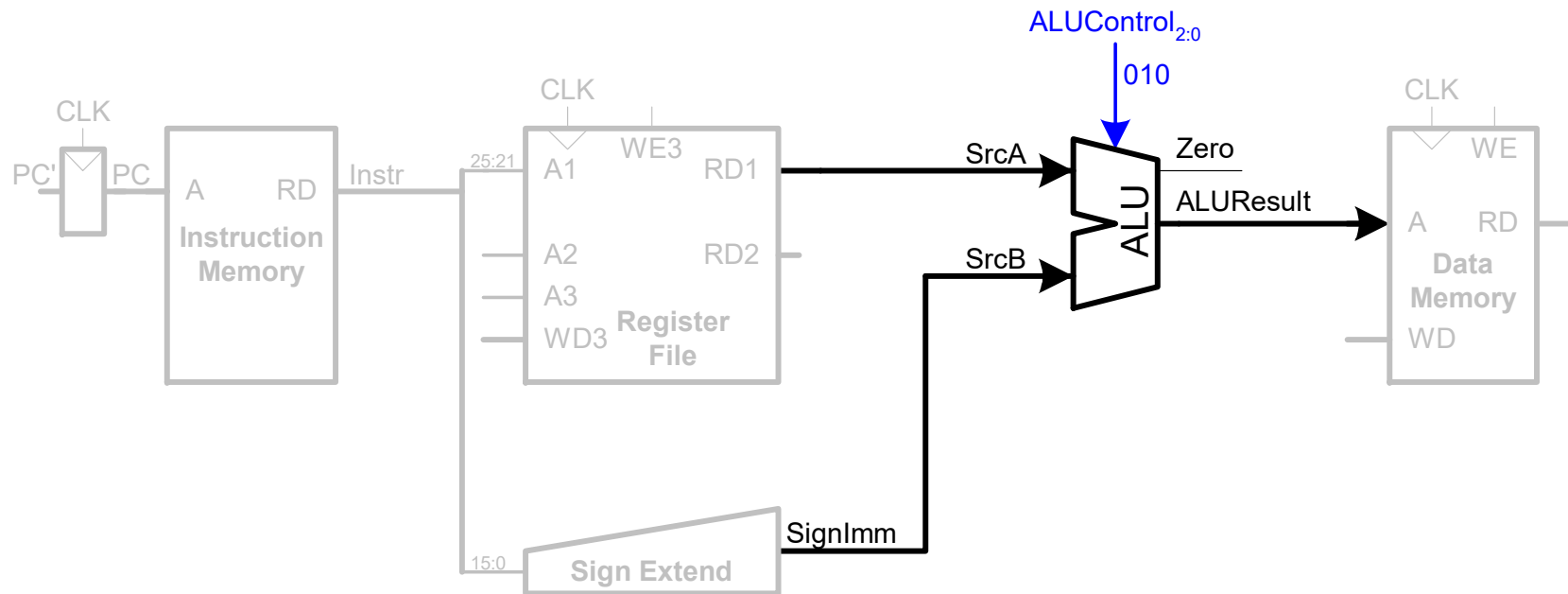
`lw $s1, 0x2000($0)`

Single-Cycle Datapath: SW

Check if other instructions can be executed with the same datapath:

Start checking `sw $s1, 0x2000($0)` (**0xAC112000**)

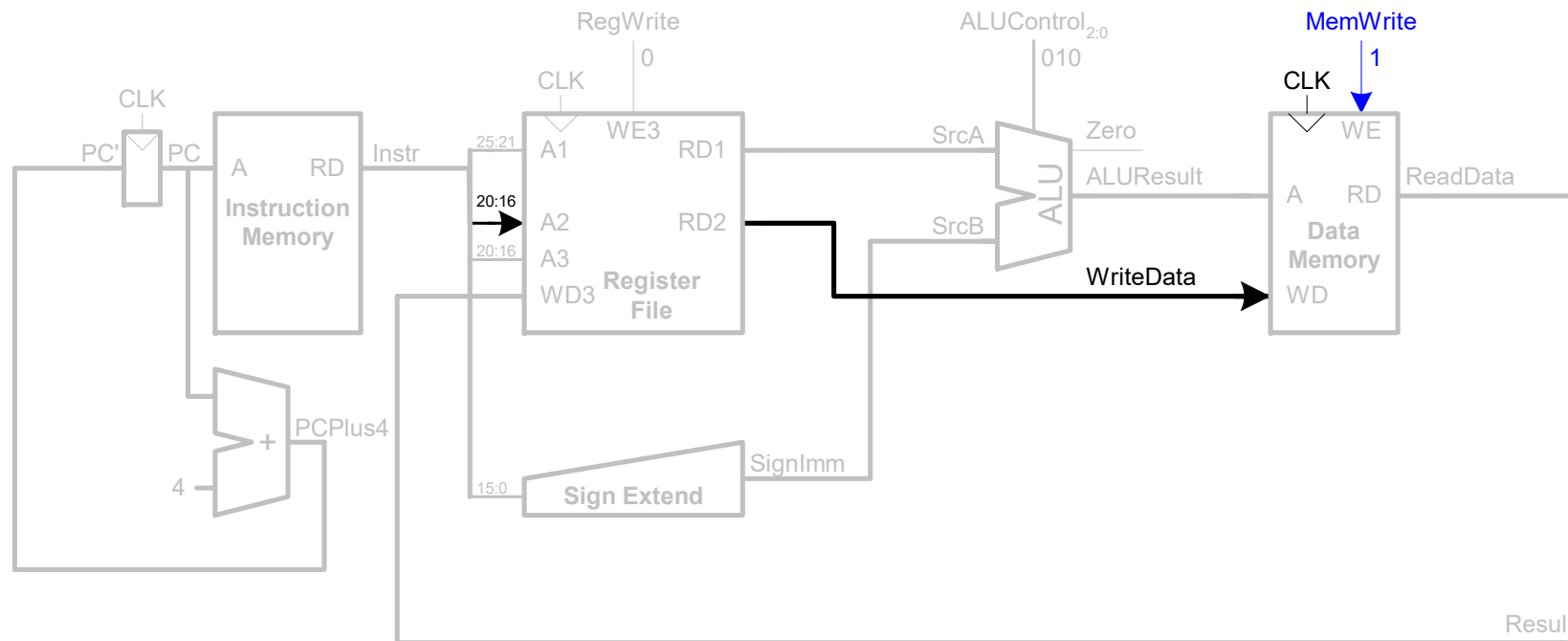
✓ Steps 1, 2, 3 and 4 are the same as in `lw`



$ALUResult \leq 0x00002000 = 0x00000000 + 0x00002000$

Single-Cycle Datapath: SW

- ✓ Missing part: read a second register, *rt*, and write it into memory



A2: Instr [20:16] = "10001" (\$s1)

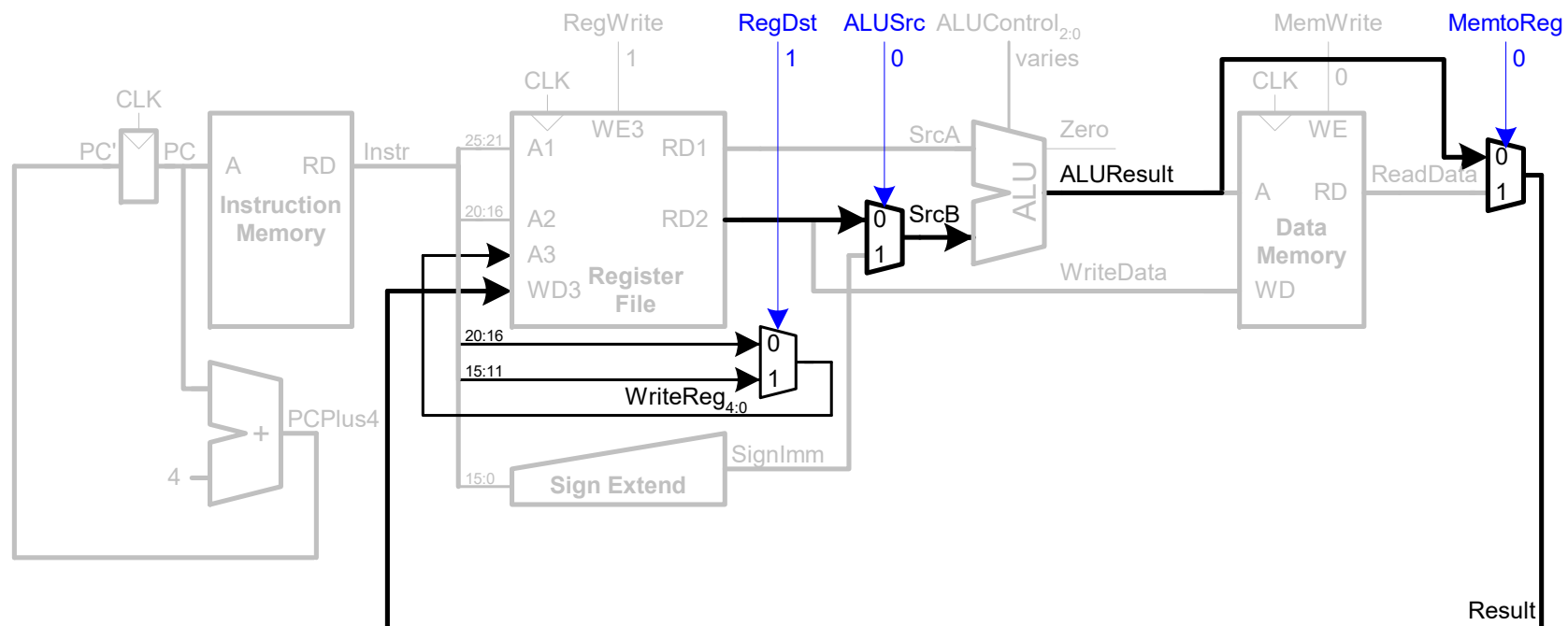
MEM[0x00002000] <= RD2 = (\$s1)

`sw $s1, 0x2000($0)`

Single-Cycle Datapath: R-type instructions

Example: `add $s1, $t1, $t2` (**0x012A8820**)

- ✓ Read from rs and rt, both to the ALU
- ✓ Write ALUResult instead of the data from memory
- ✓ Write to rd (instead of rt)



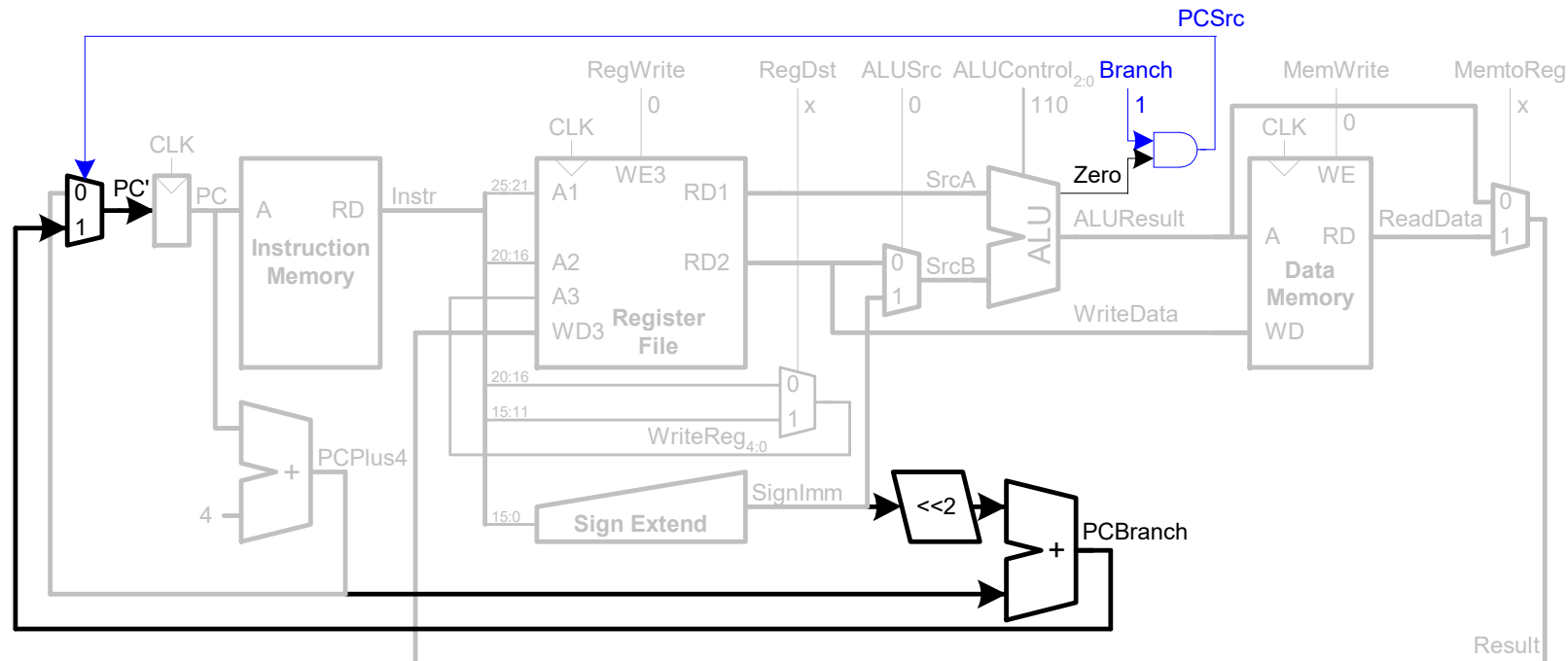
A3: Instr [15:11] = "10001" (\$s1) \$s1 <= WD3 = ALUResult (\$t1+\$t2)

Single-Cycle Datapath: beq

Determine whether values in rs and rt are equal (Z flag)

✓ Calculate *Branch Target Address*:

$$\text{BTA} = (\text{PC}+4) + (\text{Sign Extend } \{\text{imm} \ll 2\})$$

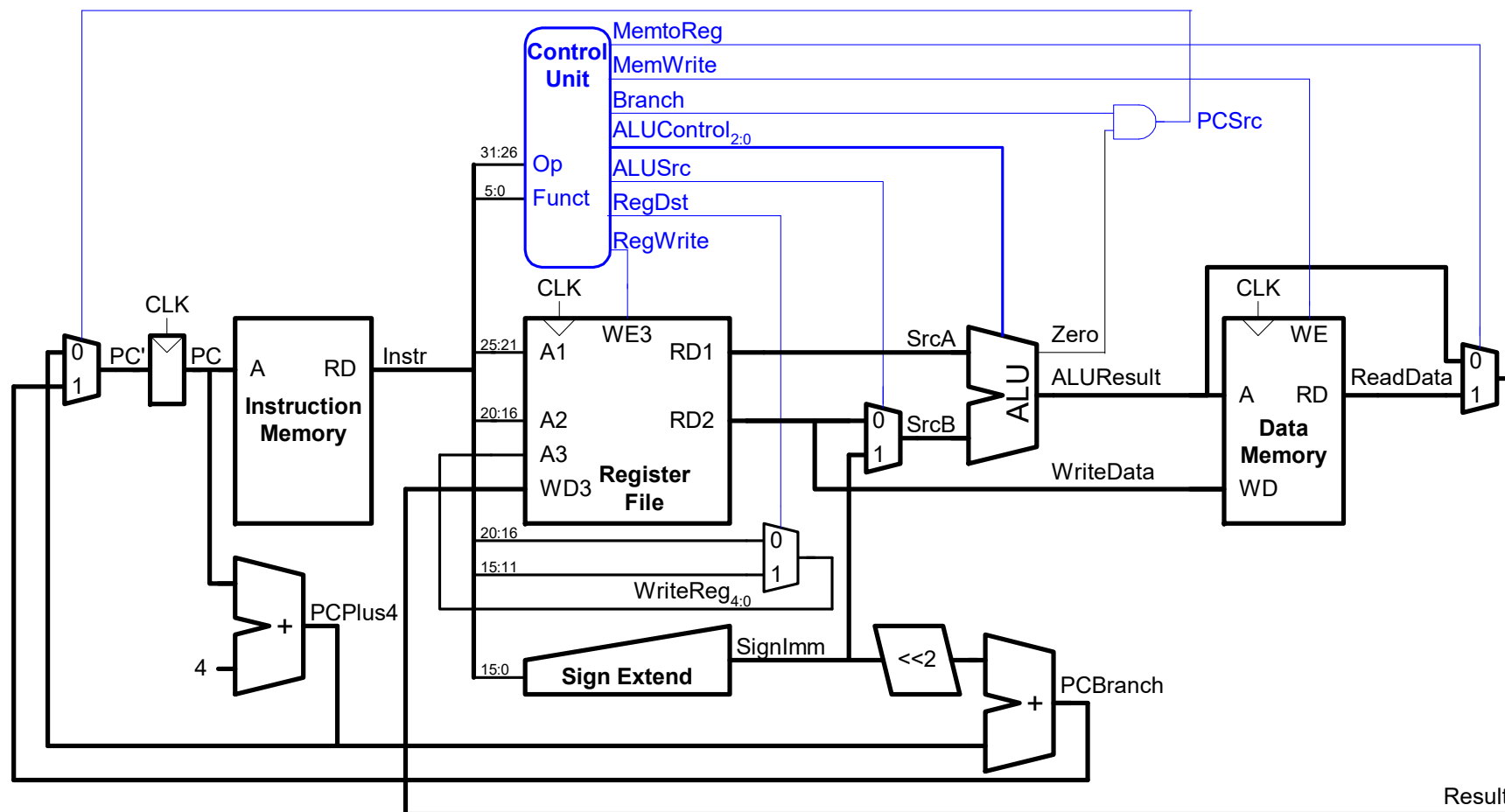


Branch is set to 1 in all `beq` instructions

PCSrc is set to 1 only if Branch=1 and the branch condition is met (Z=1)

Datapath and control path

The control signals are generated depending on the instruction
(Decode opcode and funct)

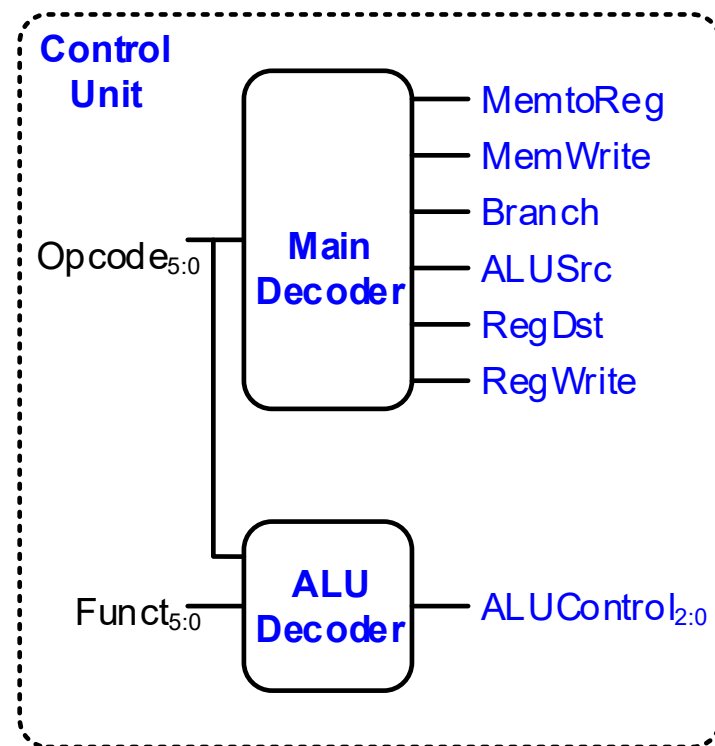


Outline

- Introduction
- Single-cycle datapath
- **Single-cycle control path**
- Adding more instructions

Control Unit

- The control unit generates two control signals groups:
 - ALUControl (3 bits): depends on **opcode** and **funct**
 - Others (6 bits): only depends on **opcode**, not on **funct**

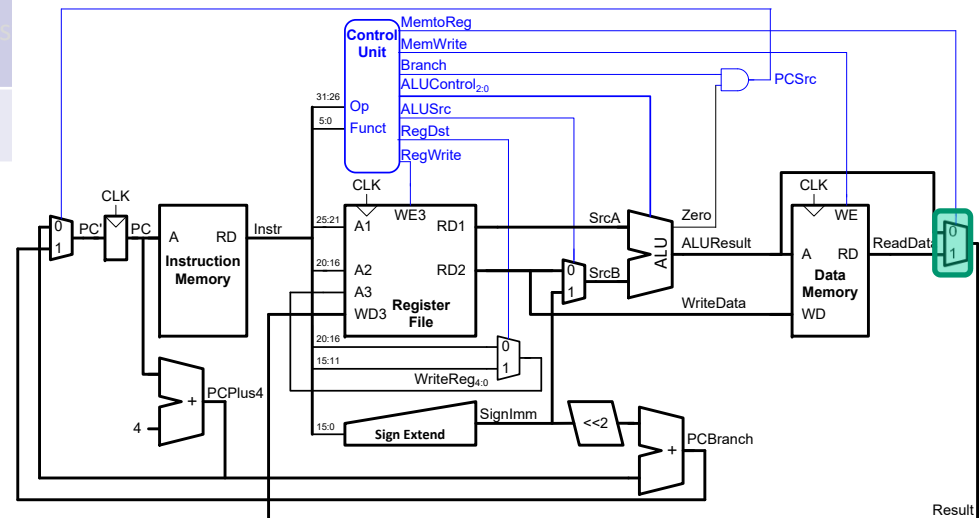


ALU Decoder

OPCODE	Funct	ALUControl _{2:0}
100011 (lw) 101011 (sw)	X	010 (Add)
000100 (beq)	X	110 (Subtract)
000000	100000 (add)	010 (Add)
000000	100010 (sub)	110 (Subtract)
000000	100100 (and)	000 (And)
000000	100101 (or)	001 (Or)
000000	101010 (slt)	111 (SetLessThan)

Control path

Signal	Purpose	Value meaning	
		0	1
MemToReg	Decides whether the data sent to the register file comes from memory or from the ALU	From ALU	From data memory
MemWrite	Decides if the data memory is written (changed)	No	Yes
Branch	Indicates if a beq is being executed	No	Yes
PcSrc	Indicates if the branch must be taken. Only active when Branch = 1 and Z = 1	No	Yes
ALUSrc	Decides which is the second ALU operand	Register RT [20:16]	Immediate
RegDst	Indicates the address of the destination register		
RegWrite	Indicates if the register file is written		



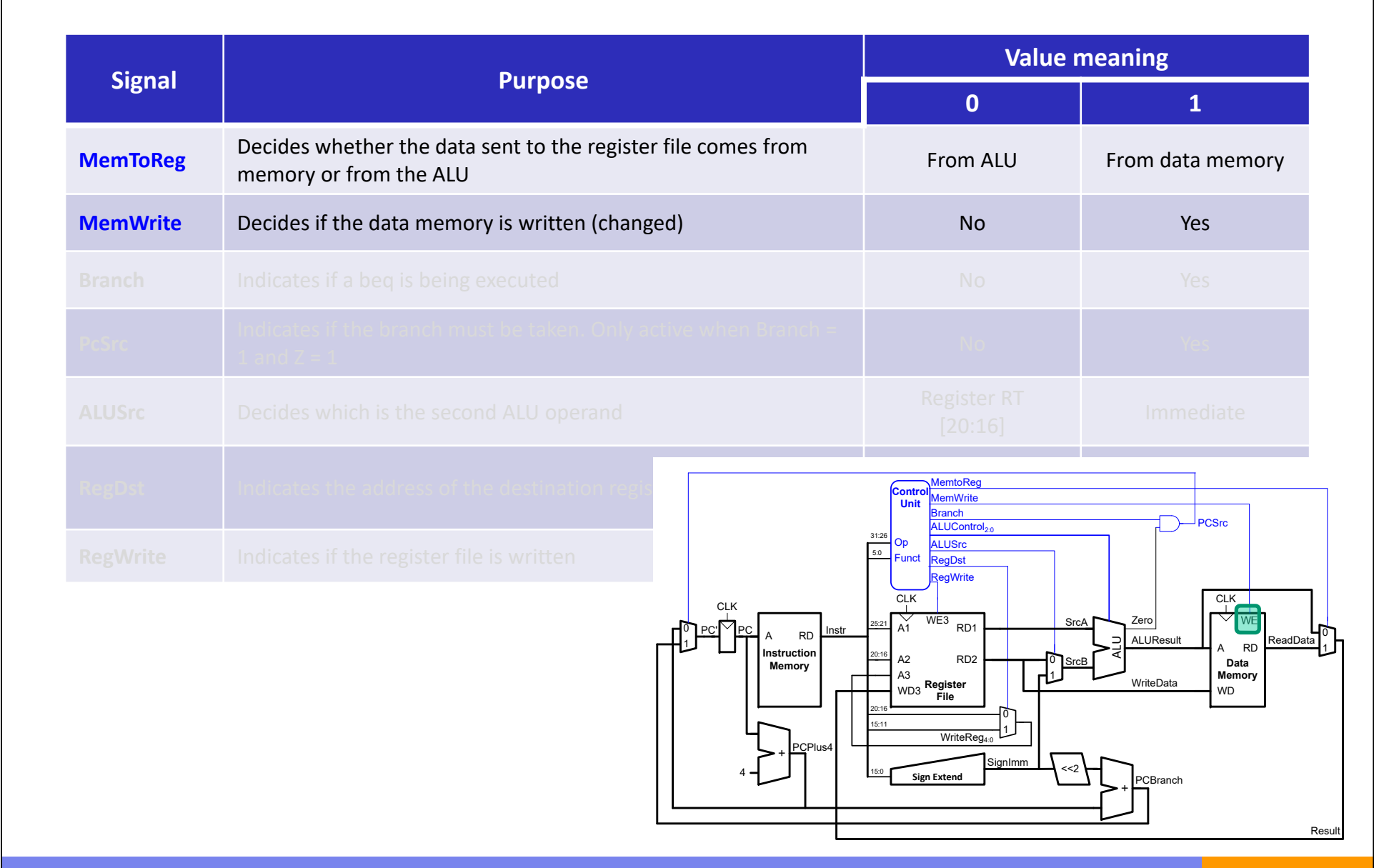
Control path

The diagram shows a horizontal bar divided into two sections. The left section is blue and labeled 'Control path'. The right section is orange and labeled 'Data path'.

Signal	Purpose	Value meaning	
		0	1
MemToReg	Decides whether the data sent to the register file comes from memory or from the ALU	From ALU	From data memory
MemWrite	Decides if the data memory is written (changed)	No	Yes
Branch	Indicates if a beq is being executed	No	Yes
PcSrc	Indicates if the branch must be taken. Only active when Branch = 1 and Z = 1	No	Yes
ALUSrc	Decides which is the second ALU operand	Register RT [20:16]	Immediate
RegDst	Indicates the address of the destination register		
RegWrite	Indicates if the register file is written		

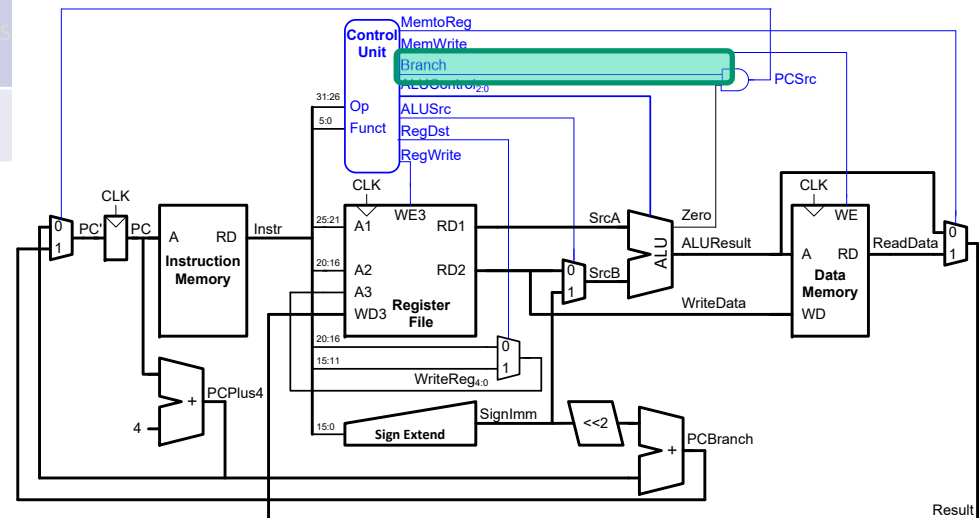
The diagram illustrates a simplified MIPS processor architecture. It includes the following components and connections:

- Control Unit:** Receives control signals (Op, Funct, ALUSrc, RegDst, RegWrite, MemToReg, MemWrite, Branch, PcSrc) and outputs control signals to the other units.
- Instruction Memory:** Receives the PC (Program Counter) and outputs the instruction (Instr) to the Register File.
- Register File:** Receives the instruction and outputs register values (A1, A2, A3, WD3) to the ALU and Data Memory. It also receives the WriteReg4:0 signal.
- ALU:** Receives the register values (A1, A2, A3, WD3) and the ALUSrc signal. It outputs the ALUResult to the Data Memory and the Zero signal to the Branch and PcSrc signals.
- Data Memory:** Receives the ALUResult and the WriteData signal. It outputs the ReadData to the Register File.
- PC (Program Counter):** Receives the PCPlus4 signal and the PcSrc signal. It outputs the PC to the Instruction Memory.
- Sign Extend:** Receives the SignImm signal and outputs the SignImm to the ALU.
- PCBranch:** Receives the SignImm signal and outputs the PCBranch signal to the PC.



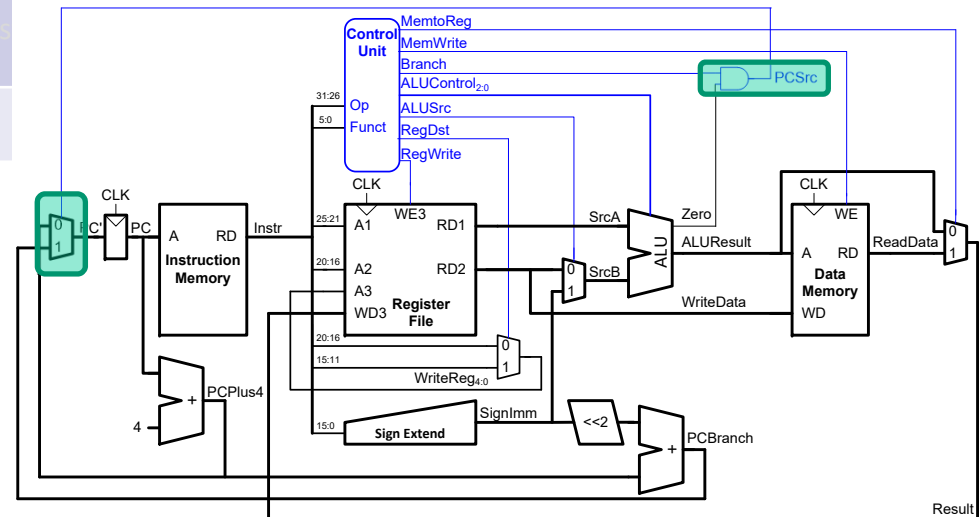
Control path

Signal	Purpose	Value meaning	
		0	1
MemToReg	Decides whether the data sent to the register file comes from memory or from the ALU	From ALU	From data memory
MemWrite	Decides if the data memory is written (changed)	No	Yes
Branch	Indicates if a beq is being executed	No	Yes
PcSrc	Indicates if the branch must be taken. Only active when Branch = 1 and Z = 1	No	Yes
ALUSrc	Decides which is the second ALU operand	Register RT [20:16]	Immediate
RegDst	Indicates the address of the destination register		
RegWrite	Indicates if the register file is written		



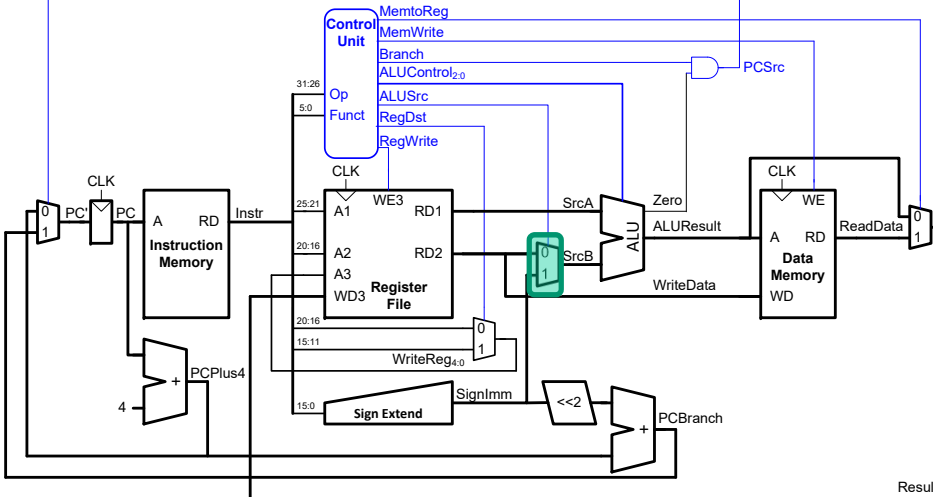
Control path

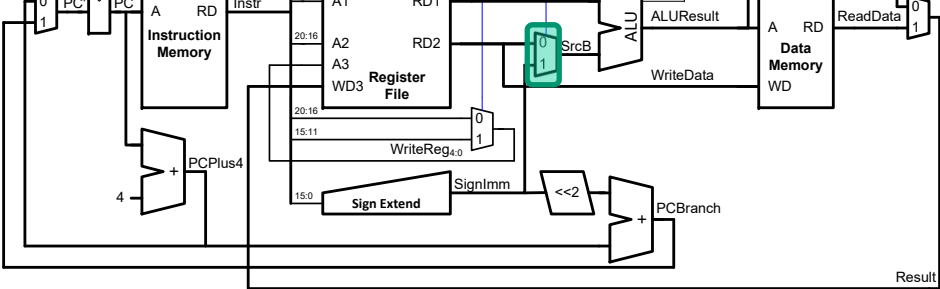
Signal	Purpose	Value meaning	
		0	1
MemToReg	Decides whether the data sent to the register file comes from memory or from the ALU	From ALU	From data memory
MemWrite	Decides if the data memory is written (changed)	No	Yes
Branch	Indicates if a beq is being executed	No	Yes
PcSrc	Indicates if the branch must be taken. Only active when Branch = 1 and Z = 1	No	Yes
ALUSrc	Decides which is the second ALU operand	Register RT [20:16]	Immediate
RegDst	Indicates the address of the destination register		
RegWrite	Indicates if the register file is written		



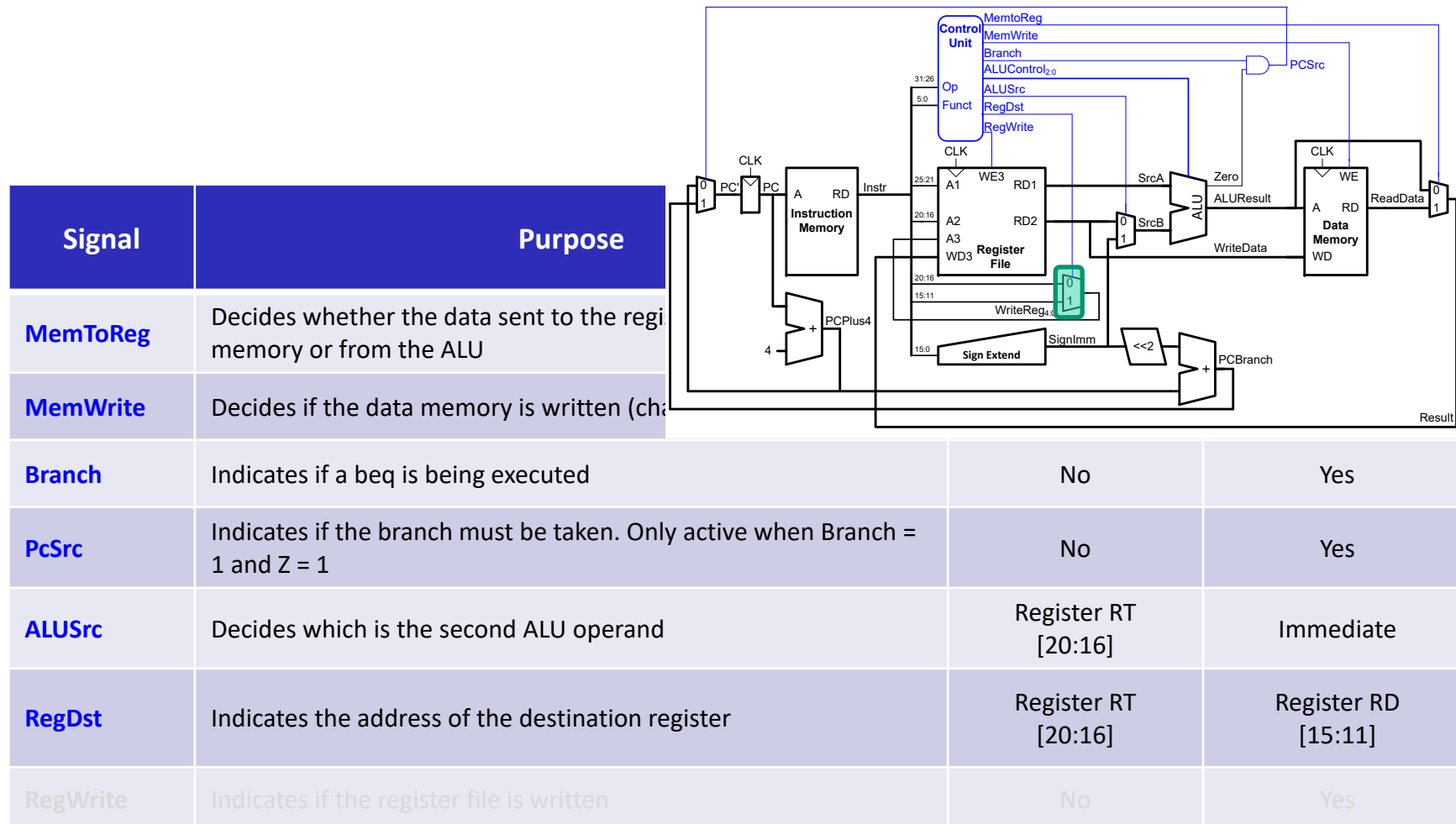
Control path

The diagram shows a horizontal bar divided into two sections. The left section is blue and labeled 'Control path'. The right section is orange and labeled 'Data path'.

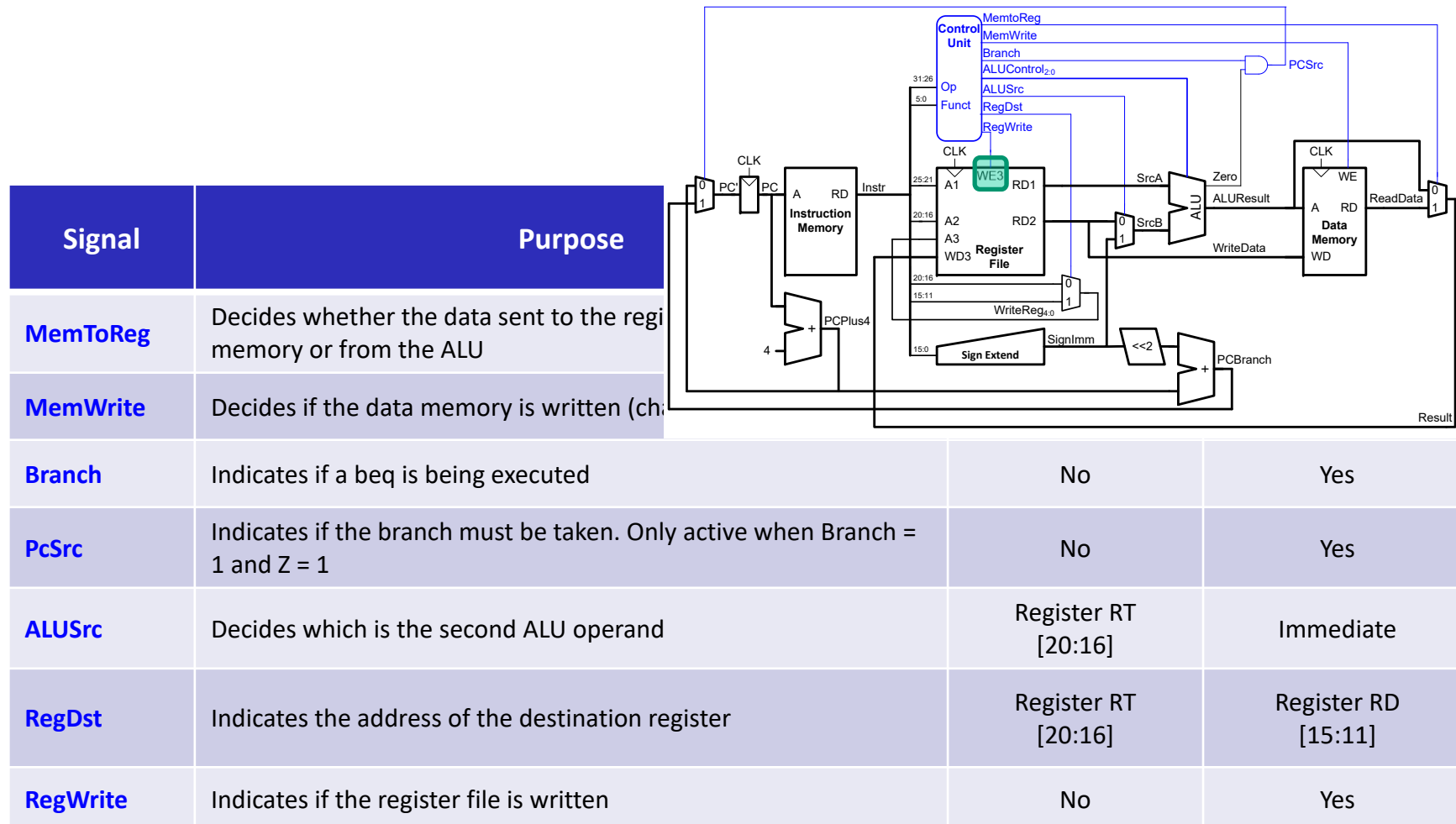


Signal	Purpose		
MemToReg	Decides whether the data sent to the register memory or from the ALU		
MemWrite	Decides if the data memory is written (checked)		
Branch	Indicates if a beq is being executed	No	Yes
PcSrc	Indicates if the branch must be taken. Only active when Branch = 1 and Z = 1	No	Yes
ALUSrc	Decides which is the second ALU operand	Register RT [20:16]	Immediate
RegDst	Indicates the address of the destination register	Register RT [20:16]	Register RD [15:11]
RegWrite	Indicates if the register file is written	No	Yes

Control path

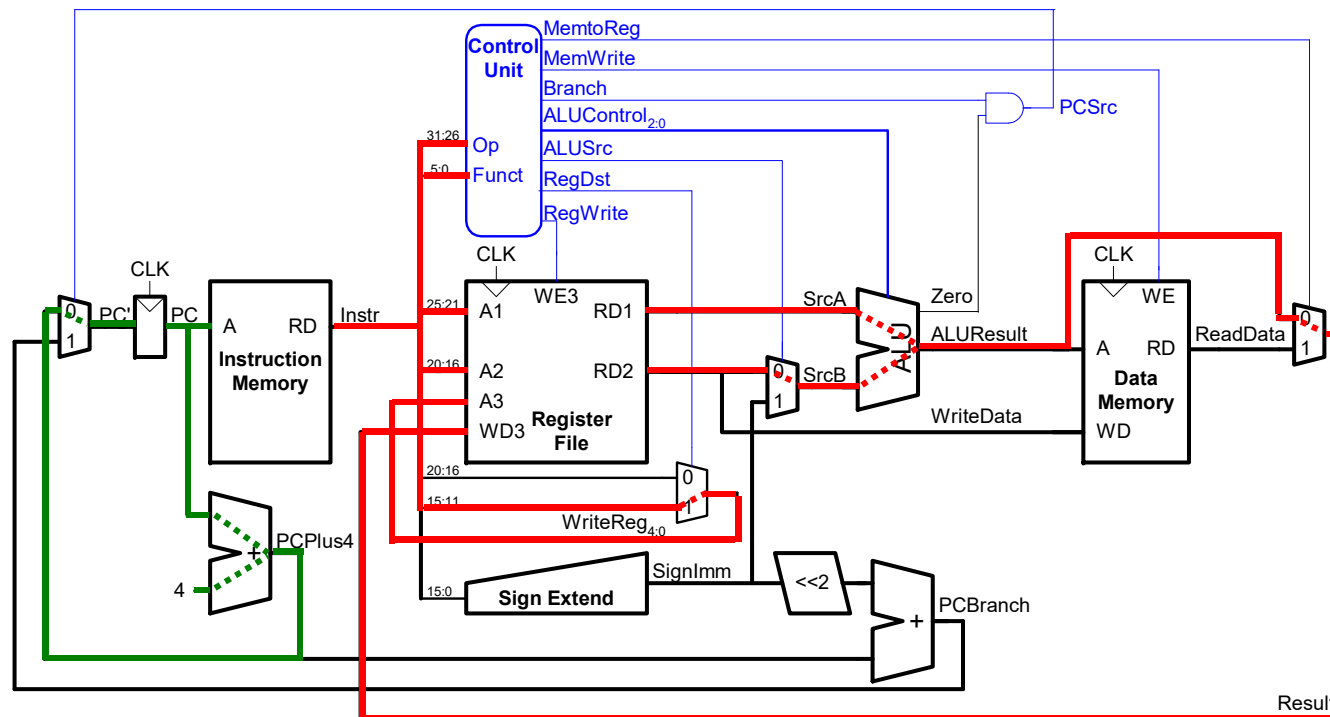


Control path



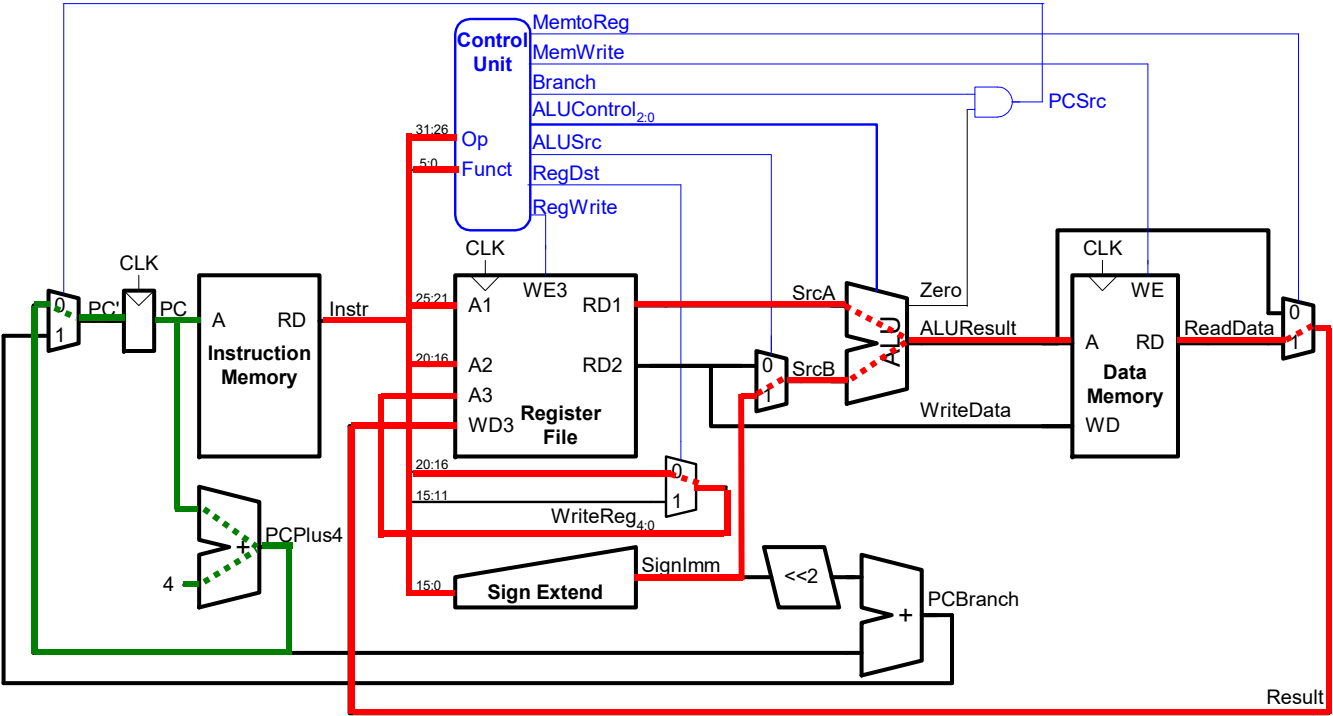
Control Unit: R-type instructions

Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	ALUCtrl
R-type	000000	1	1	0	0	0	0	
lw	100011							
sw	101011							
beq	000100							



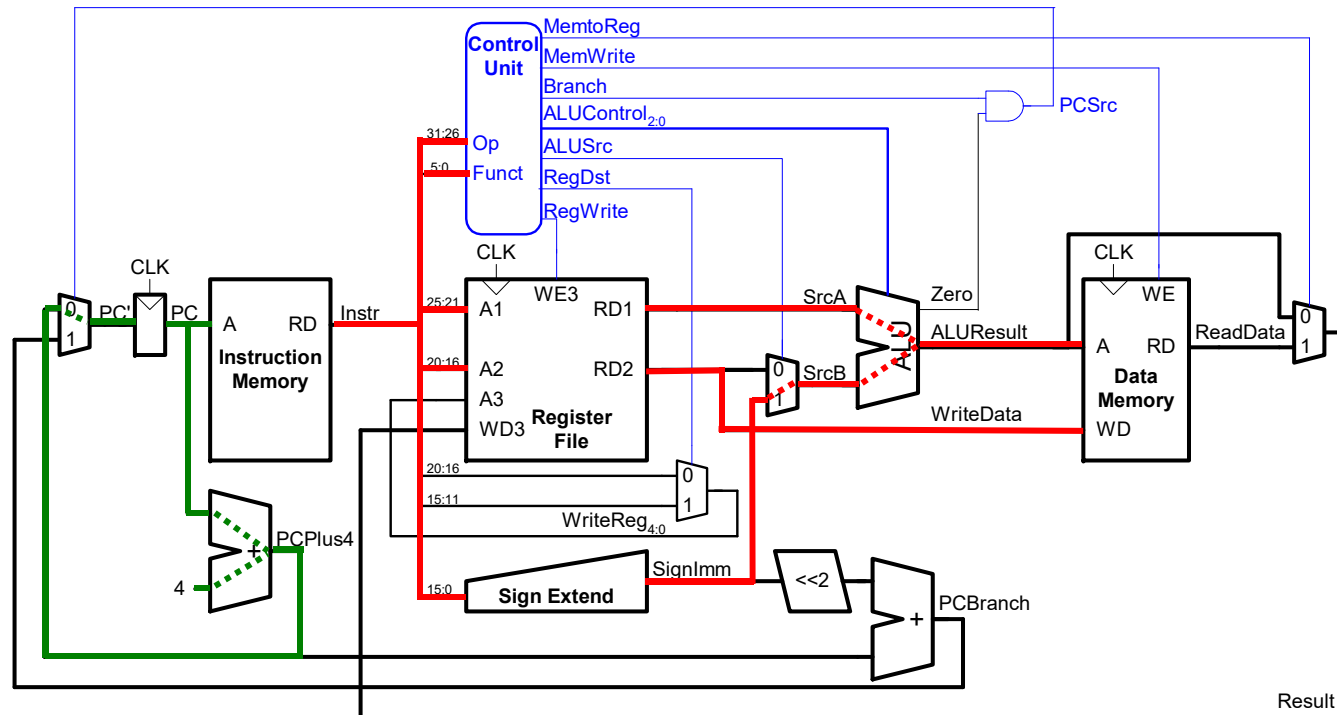
Control Unit: lw

Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	ALUCtrl
R-type	000000	1	1	0	0	0	0	
lw	100011	1	0	1	0	0	1	010
sw	101011							
beq	000100							



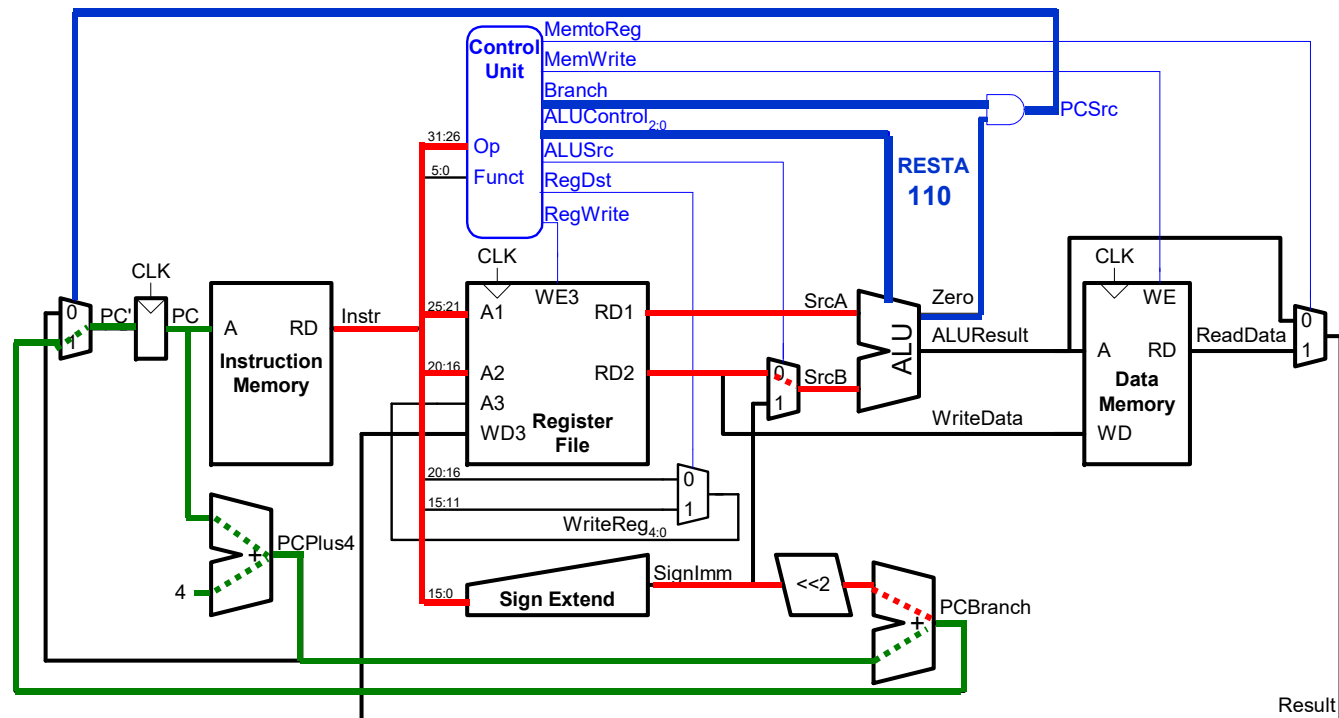
Control Unit: sw

Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	ALUCtrl
R-type	000000	1	1	0	0	0	0	
lw	100011	1	0	1	0	0	1	010
sw	101011	0	X	1	0	1	X	010
beq	000100							



Control Unit: beq

Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	ALUCtrl
R-type	000000	1	1	0	0	0	0	
lw	100011	1	0	1	0	0	1	010
sw	101011	0	X	1	0	1	X	010
beq	000100	0	X	0	1	0	X	110

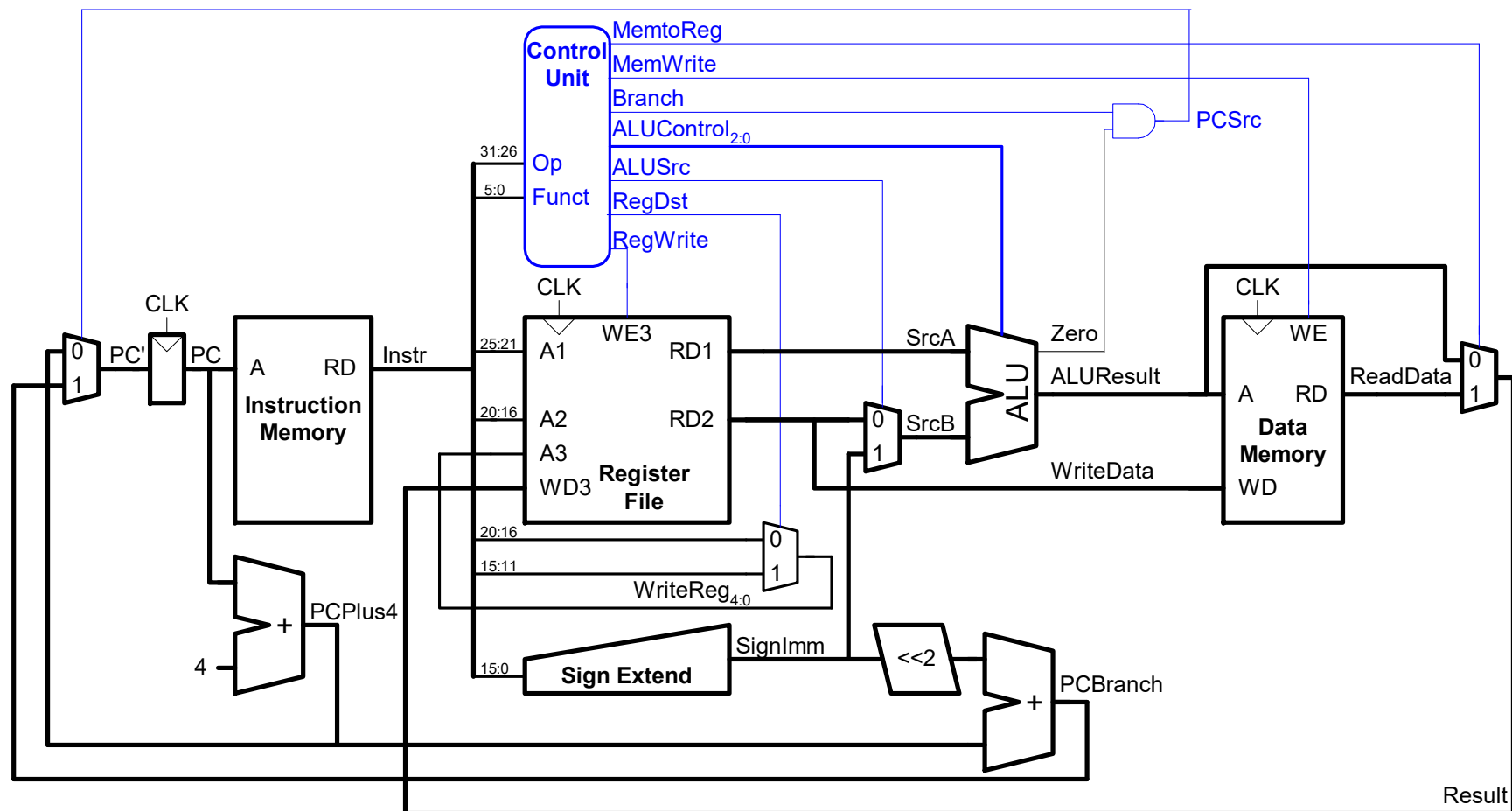


Outline

- Introduction
- Single-cycle datapath
- Single-cycle control path
- **Adding more instructions**

Adding more instructions: addi

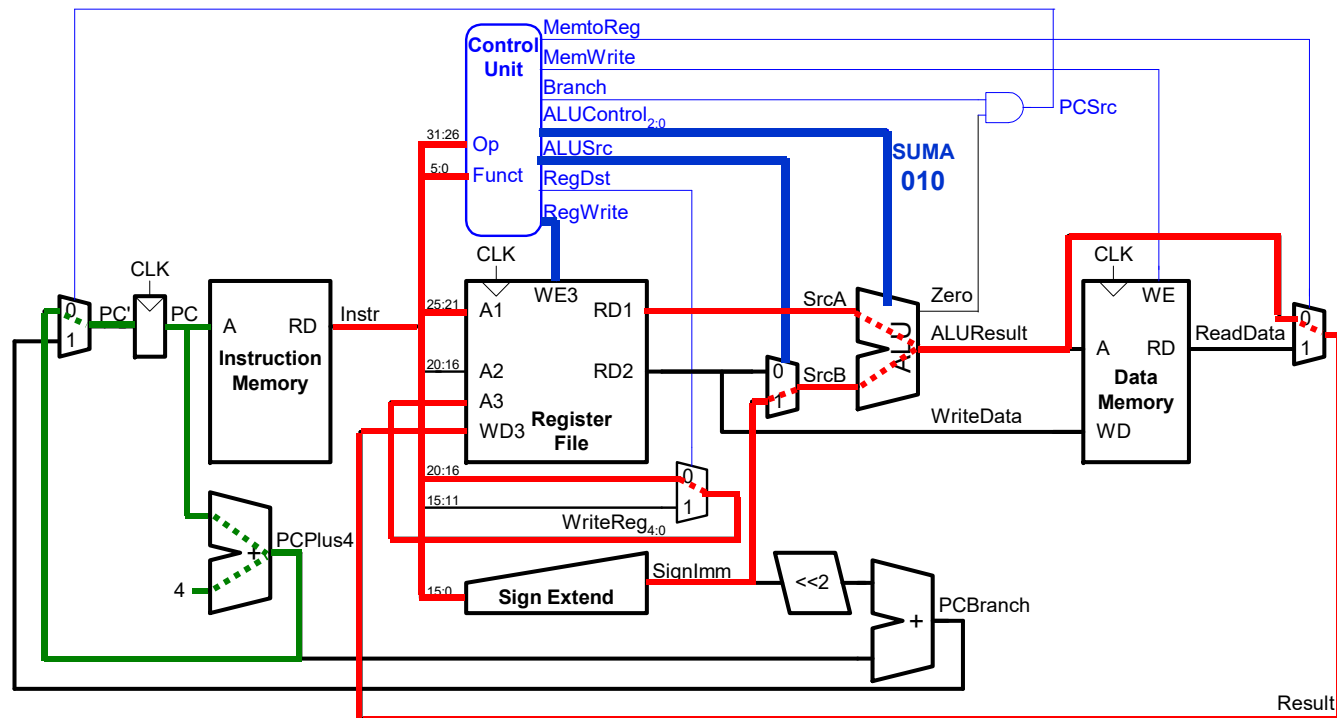
➤ No change to datapath



Control Unit: addi

➤ No change to datapath

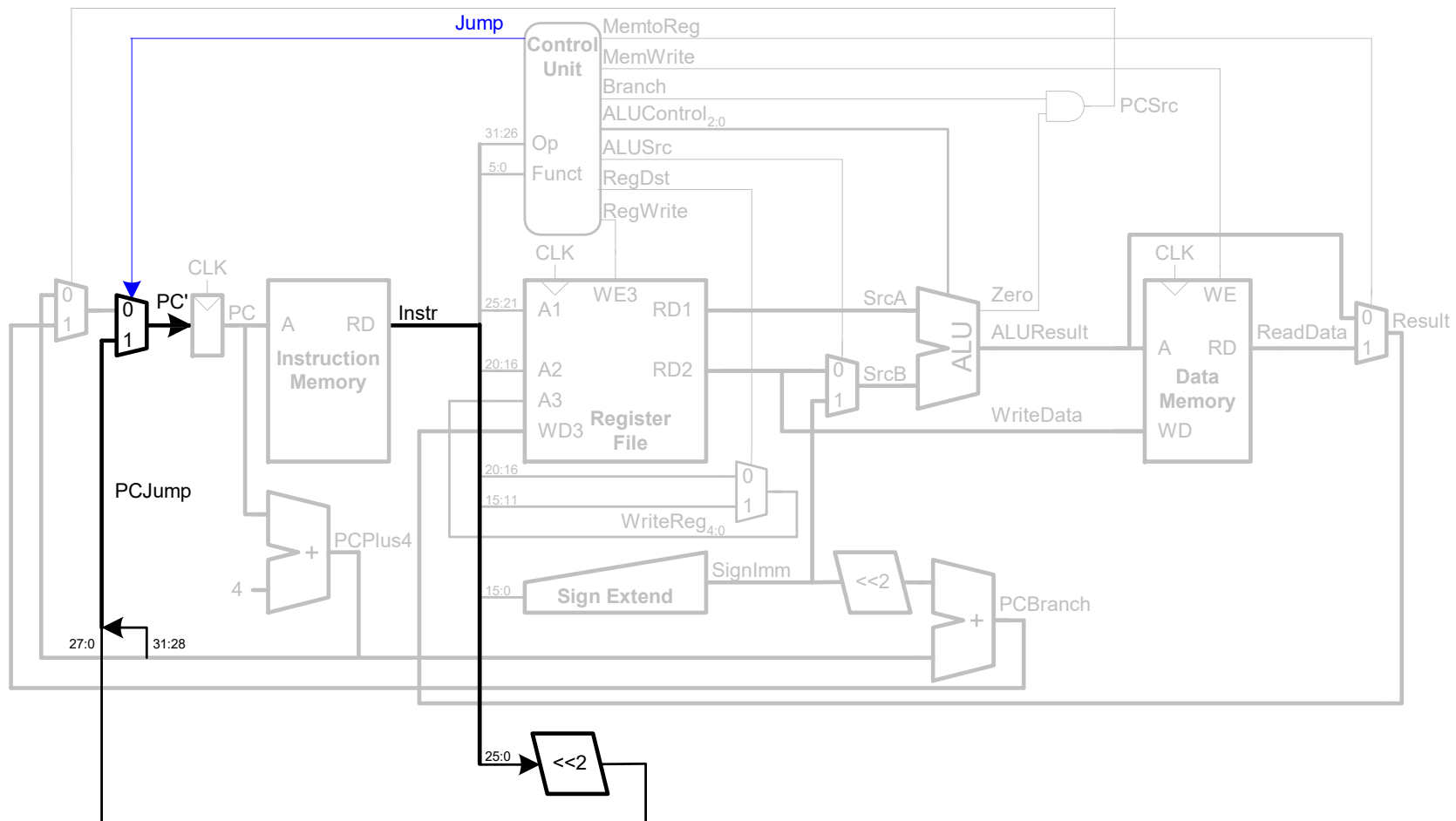
Instruction	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	ALUctrl
addi	001000	1	0	1	0	0	0	010



Adding more instructions: j

- Calculate the *Jump Target Address*:

$$\text{JTA} = (\text{PC}+4)[31:28] \& \text{addr} \& \text{"00"}$$



Control Unit: j

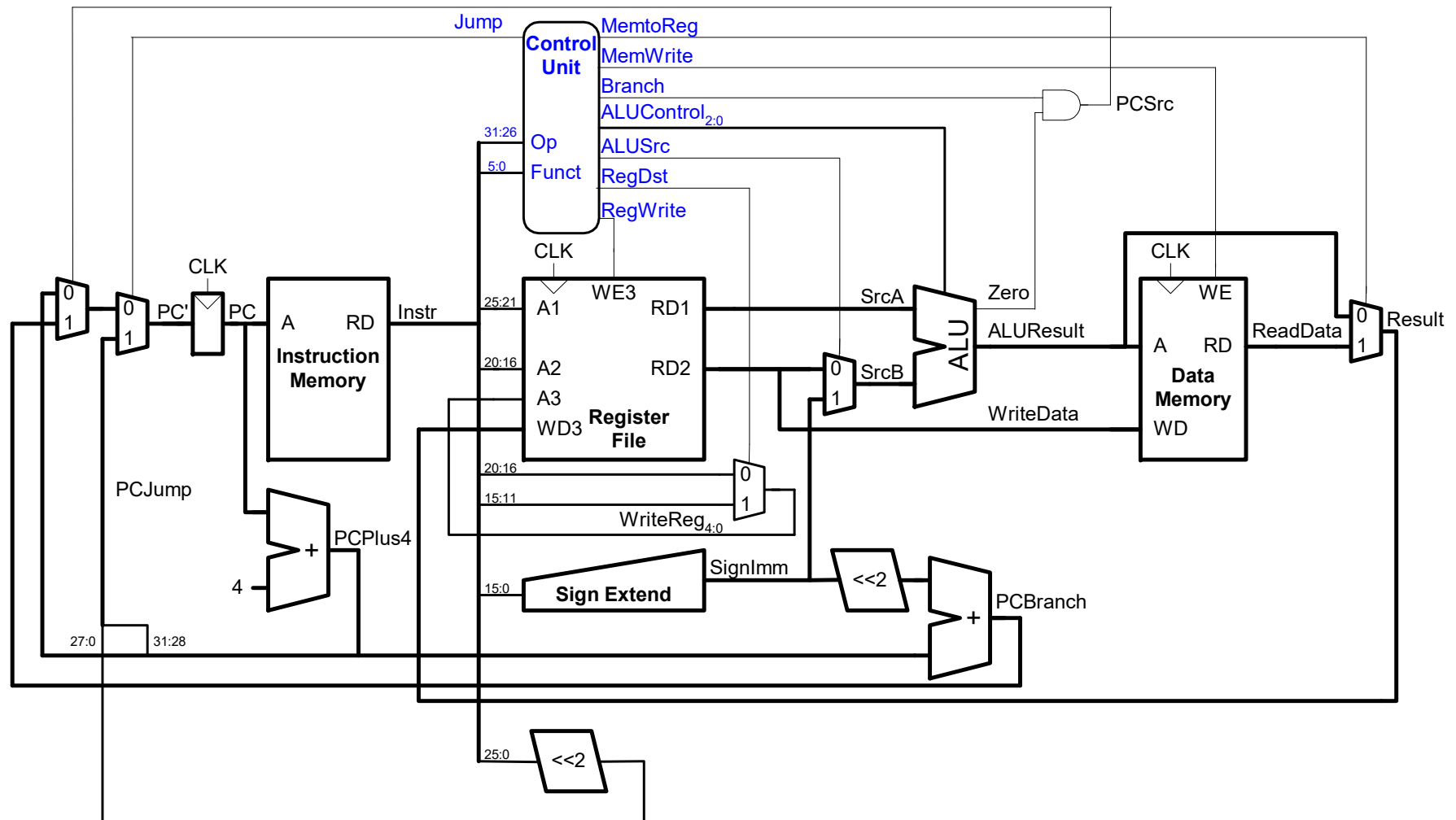
Instrucción	Op _{5:0}	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	ALUCtrl	Jump
R-type	000000	1	1	0	0	0	0		0
lw	100011	1	0	1	0	0	1	010	0
sw	101011	0	X	1	0	1	X	010	0
beq	000100	0	X	0	1	0	X	110	0
addi	001000	1	0	1	0	0	0	010	0
j	000010	0	X	X	X	0	X	XX	1

Control Signals

Signal	Purpose	Value meaning		Use
		0	1	
MemToReg	Decides whether the data sent to the register file comes from memory or from the ALU	From ALU	From data memory	1: lw X: sw, beq, jump 0: Others
MemWrite	Decides if the data memory is written (changed)	No	Yes	1: sw 0: Others
Branch	Indicates if a beq is being executed	No	Yes	1: beq X: jump 0: Others
PcSrc	Indicates if the branch must be taken. Only active when Branch = 1 and Z = 1	No	Yes	1: beq and condition met 0: Others
ALUControl	Indicates the operation of the ALU	Multiple bits	Multiple bits	Multiple values
ALUSrc	Decides which is the second ALU operand	Register RT [20:16]	Immediate	1: I-type, except beq 0: R-type, beq X: jump
RegDst	Indicates the address of the destination register	Register RT [20:16]	Register RD [15:11]	1: R-type 0: I-type, except sw, beq X: sw, beq, jump
RegWrite	Indicates if the register file is written	No	Yes	0: sw, beq, jump 1: Others
Jump	Indicates a jump is being executed	No	Yes	1: jump 0: Others

Summary: Single-Cycle MIPS

P



Unit 4. Processor II: Design and control of the datapath. Single-cycle architecture

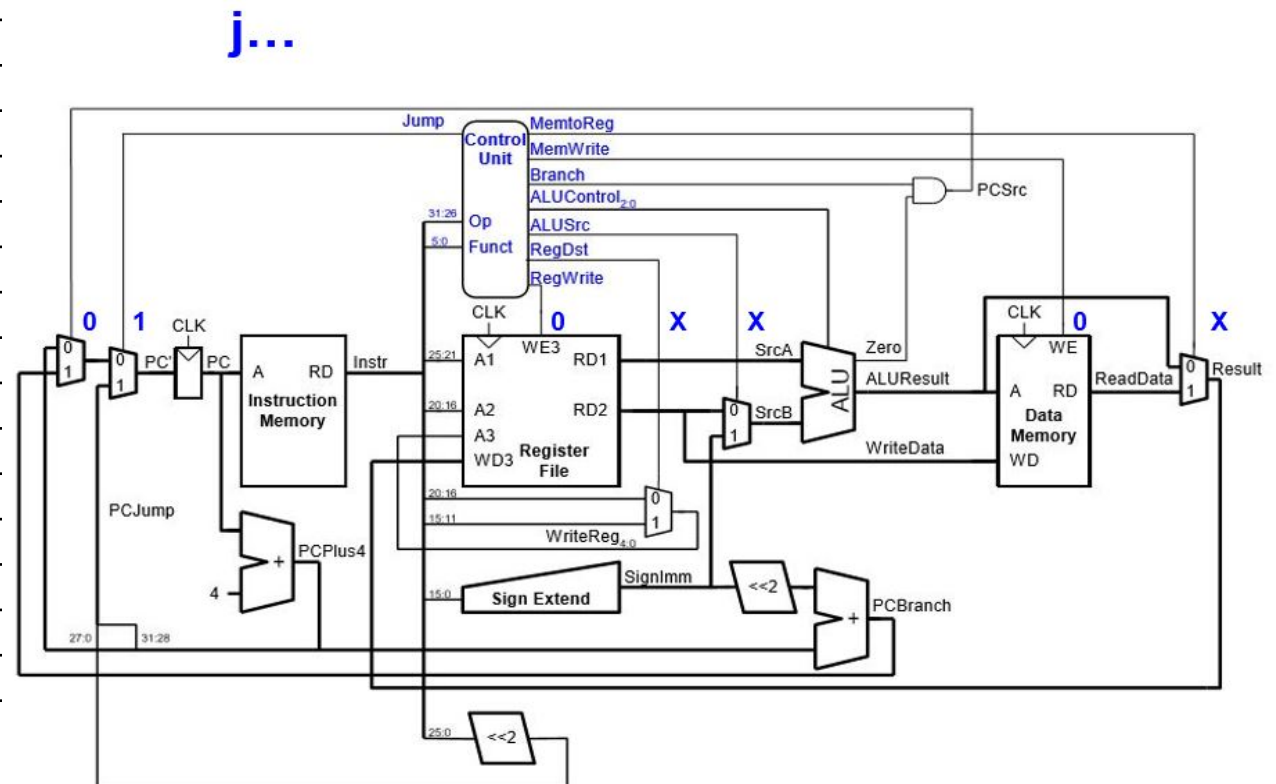
Escuela Politécnica Superior - UAM

4.7. The following program is executed in a single-cycle MIPS processor. Please, fill in the signals and registers values of the table when the code is executed.

Note: The table includes the initial content of four data memory elements.

Código	Señal/registro	T	T+1
<pre> .text 0x0800 lw \$s1, B(\$0) lw \$s2, C(\$0) and \$s3, \$s1, \$s2 sw \$s3, D(\$0) add \$s4, \$s1, \$s2 lw \$s5, D(\$0) fin: j fin </pre>	pc	0x0800	
	Jump	0	
	MemtoReg	1	
	MemWrite	0	
	Branch	0	
	ALUSrc	1	
	RegDst	0	
	RegWrite	1	
<pre> .data 0x2000 A: 0x00000005 B: 0x0000000C C: 0x00000007 D: 0x0000002F </pre>	\$s1	0x0000	
	\$s2	0x0000	
	\$s3	0x0000	
	\$s4	0x0000	
	\$s5	0x0000	

Code	Signal
<pre> .text 0x0800 lw \$s1, B(\$0) lw \$s2, C(\$0) and \$s3, \$s1, \$s2 sw \$s3, D(\$0) add \$s4, \$s1, \$s2 lw \$s5, D(\$0) fin: j fin </pre>	pc
	Jump
	MemtoReg
	MemWrite
	Branch*
	ALUSrc
	RegDst
	RegWrite
	Registro
<pre> .data 0x2000 A: 0x00000005 B: 0x0000000C C: 0x00000007 D: 0x00000004 </pre>	\$S1
	\$S2
	\$S3
	\$S4
	\$S5

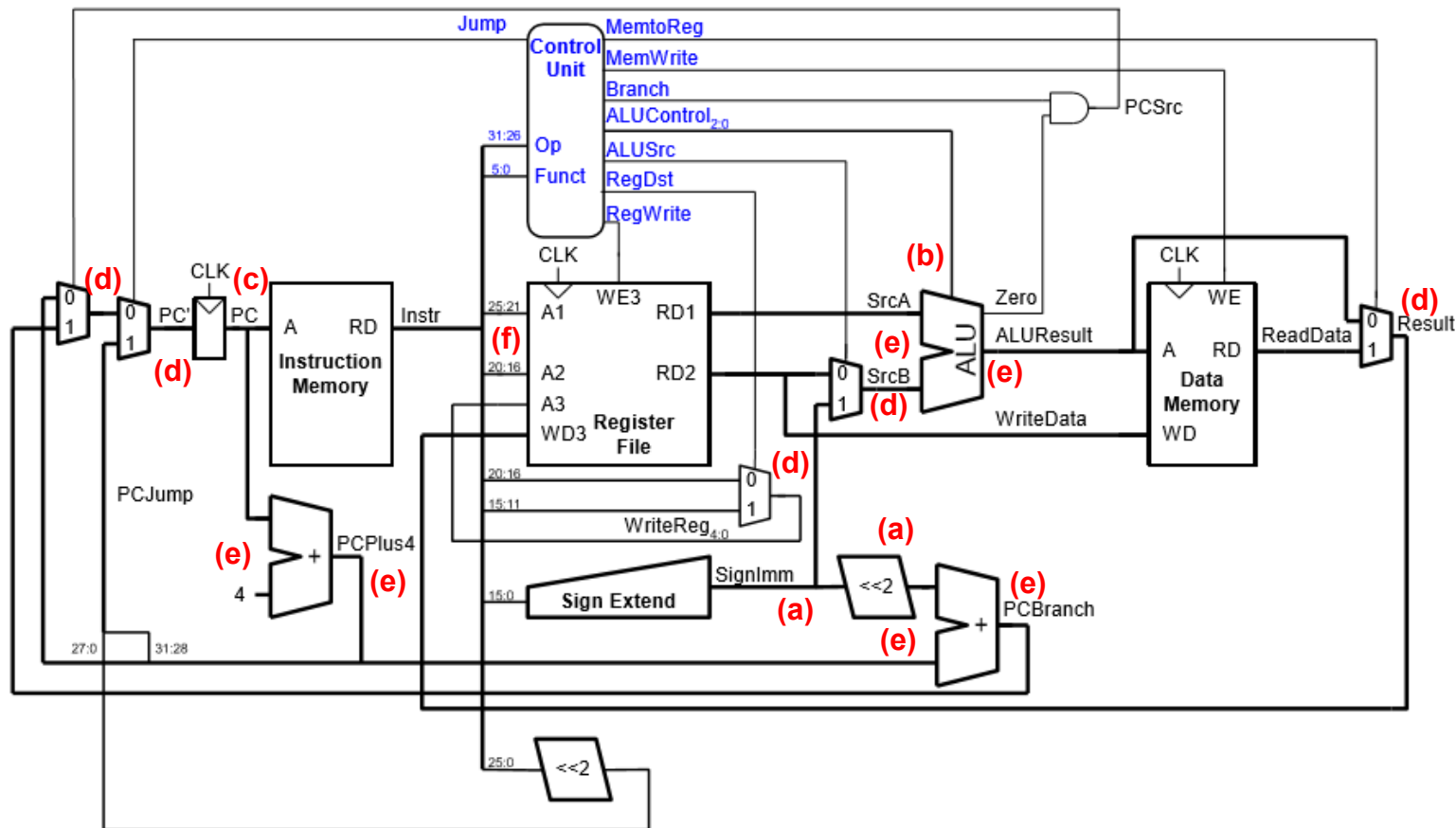


4.5.- The machine code instructions 0x8C430010 and 0x1023000C are going to be executed in a single-cycle MIPS. All memory addresses have the value 0x0FF and the initial content of the registers is shown in the following table:

\$0	\$at	\$v0	\$v1	\$a0	\$a1	\$a2	\$t0	\$t4	\$ra
0	-16	-2	-3	4	-10	-6	-1	8	-4

For each of the previous instructions answer the following questions:

- Which is the output of the sign-extend block and the "Shift.Left.2" block in the unconditional jump datapath?
- Which is the binary value of the ALU control signal ALUControl_{2:0}?
- Which is the new value of PC after the instruction?
- Write the hexadecimal value of the output of each multiplexer during the execution. If the value can not be known, write X.
- Write the hexadecimal values of the ALU inputs and the inputs of both adding blocks?
- Write the values of all the inputs to the register file. Write it in binary for the 5-bit inputs and in hexadecimal for the 32-bit inputs?



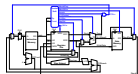
4.15.- During the execution of the following MIPS code, in the cycle T the values of some control signals, registers and memory addresses are as indicated in the tables.

a. Analysing the control signals at cycle T, identify the instruction being executed and fill in the rest of the control signals table after executing the rest of the code.

b. Fill in (in hexadecimal) the values of the registers and memory addresses of the table after executing the code (cycle T+5).

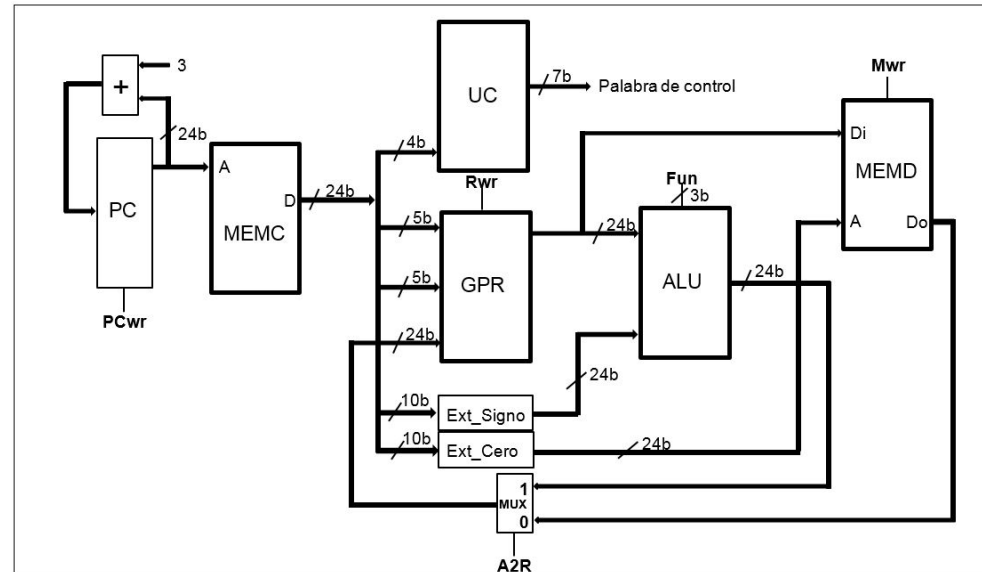
Code	Data memory (current, T)	Data memory (final, T+5)
<pre> .text 0x0000 addi \$s1, \$0, 0x2000 lw \$s2, 8(\$s1) add \$s1, \$s2, \$s3 .text 0x0020 and \$s2, \$s1, \$s3 sw \$s2, -4(\$s3) beq \$s1, \$s2, etiq lw \$s4, 4(\$s2) addi \$s3, \$s1, -1 or \$s2, \$s1, \$s2 fin: j fin etiq: lw \$s3, C(\$0) slt \$s3, \$s2, \$s1 add \$s2, 4(\$s1) </pre>	.data 0x2000	.data 0x2000
	A: 0x0001	A: ¿?
	B: 0xFF00	B: ¿?
	C: 0x0400	C: ¿?
	Registers (current, T)	Registers (final, T+5)
	\$pc = ¿?	\$pc = ¿?
	\$s1 = 0x00FF	\$s1 = ¿?
	\$s2 = 0x2000	\$s2 = ¿?
	\$s3 = 0x200C	\$s3 = ¿?
	\$s4 = 0x0000	\$s4 = ¿?

<pre> .text 0x0000 addi \$s1, \$0, 0x2000 lw \$s2, 8(\$s1) add \$s1, \$s2, \$s3 .text 0x0020 and \$s2, \$s1, \$s3 → sw \$s2, -4(\$s3) beq \$s1, \$s2, etiq lw \$s4, 4(\$s2) addi \$s3, \$s1, -1 or \$s2, \$s1, \$s2 fin: j fin etiq: lw \$s3, C(\$0) slt \$s3, \$s2, \$s1 add \$s2, 4(\$s1) </pre>	Instruc.	sw \$s2...	beq \$s1...	lw \$s4...	addi \$s3..	or \$s2...	j fin
	Ciclo	T	T+1	T+2	T+3	T+4	T+5
	ALUSrc	1	0	1	1	0	X
	Jump	0	0	0	0	0	1
	MemtoReg	X	X	1	0	0	X
	MemWrite	1	0	0	0	0	0
	PCSrc	0	0	0	0	0	0
	RegDst	X	X	0	0	1	X
	RegWrite	0	0	1	1	1	0
<pre> .data 0x2000 A: 0x0001 B: 0xFF00 C: 0x2000 </pre>	\$pc	0x0024	0x0028	0x002C	0x0030	0x0034	0x0038
	\$s1	0x00FF	=	=	=	=	=
	\$s2	0x2000	=	=	=	=	0x20FF
	\$s3	0x200C	=	=	=	0x00FE	=
	\$s4	0x0000	=	=	0xFF00	=	=



4.17.- The figure shows the architecture of a single-cycle processor. There are five main blocks: both memories (MEMC, MEMD), register file (GPR), arithmetic-logic unit (ALU) and control unit (UC). There are other known digital blocks that, along with the control signals, are used for configuring the datapath. According to the shown architecture, answer the following questions briefly justifying each answer.

- Indicate the word size of this processor
- Maximum number of different instructions in this processor
- Maximum number of registers in the GPR



- Write the control signals (PCwr, Mwr, Rwr, A2R, Fun) for the instruction `add r1, r2, 8` ($r1 = r2 + 8$)
- Write the control signals (PCwr, Mwr, Rwr, A2R, Fun) for the instruction `sw r1, 8(r1)` ($r1 \Rightarrow \text{MEMD}(8)$)
- Supposing the ALU can perform the necessary operations, indicate other reasons why the following instructions can not be performed.

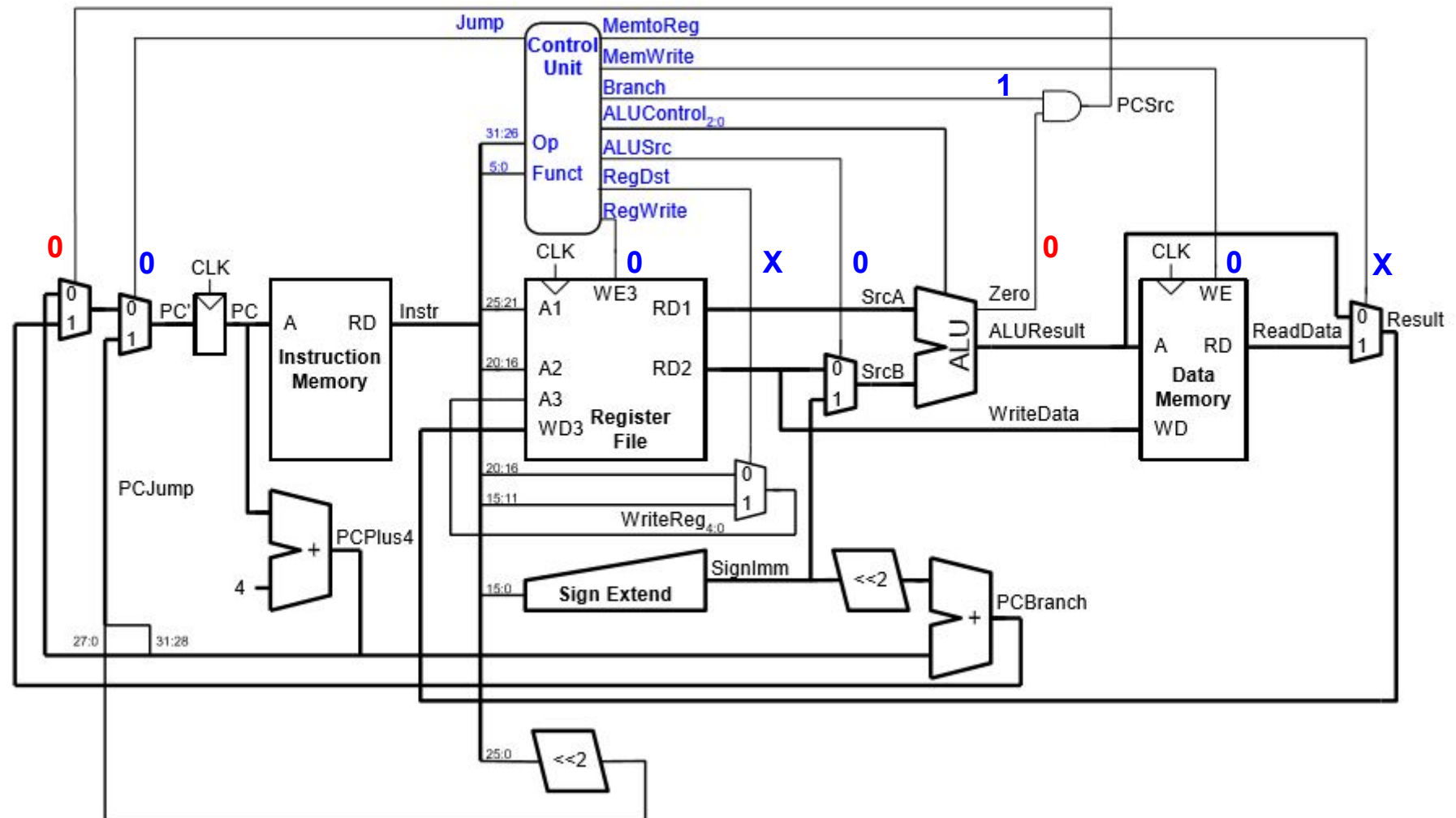
<code>add r1, r2, r3</code>	<code> #(r1 = r2 + r3)</code>
<code>lw r1, 5(r2)</code>	<code> # r1 = MEMD[(5+r2)]</code>

4.2.- The different elements of the processor have their own delay (delay: time from a change in the input to a change in the output). The critical path for an instruction is given by adding the delays in the longest path. For the single-cycle MIPS architecture, consider the following delays:

I-MEM	Add	Mux	ALU	RegFile	D-MEM	Control	SignExt	ShiftLeft2
500 ps	150 ps	30 ps	180 ps	220 ps	1000 ps	65 ps	90 ps	20 ps

- Which is the critical path for an “and” instruction? Which would be the clock cycle if all the instructions were ALU instructions (“and”, “add”, etc)?
- Which is the critical path for a “lw” instruction? Which would be the clock cycle if all the instructions were “lw” instructions?
- Which is the critical path for a “beq” instruction?
- Which would be the clock cycle if the instructions possible instructions are “and”, “beq” and “lw”?

beq...



4.9.- The following figure shows a single-cycle architecture with two independent ALUs whose operands are read/written from a 4 registers register file. The table shows the control signals of the ALUs and the operations that can be executed.

a) Write the code for a program that does the following:

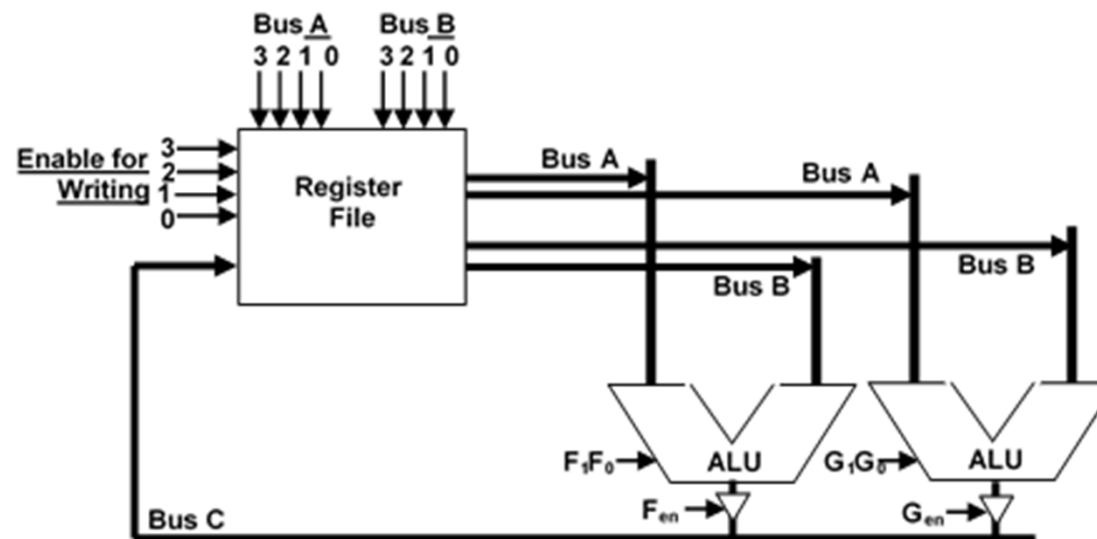
a1. XOR of the contents of the registers R0 y R1 storing the result in register R0. ($R_0 \leftarrow R_0 \oplus R_1$)

a2. The difference of registers R1 and R2 storing the result in register R3. ($R_3 \leftarrow R_1 - R_2$)

a3. The product of registers R0 and R1 storing the result in register R2. Suppose the initial values R2 = 0 and R3 = 1. Also suppose that the same control flow instructions of MIPS (branch and jump) are available. ($R_2 \leftarrow R_0 \times R_1$)

b. Write the control signals for the instructions of each of the previous codes. For case 3, ignore the control signals of the control flow instructions.

ALU F		ALU G		F _{en}	G _{en}	Bus A				Bus B				Bus C			
F ₁	F ₀	G ₁	G ₀			r ₃	r ₂	r ₁	r ₀	r ₃	r ₂	r ₁	r ₀	r ₃	r ₂	r ₁	r ₀



Code F ₁ F ₀	Function	Instruction Syntax	Code G ₁ G ₀	Function	Instruction Syntax
00	C = A + B	ADD (A,B,C)	00	C = A OR B	OR (A,B,C)
01	C = A + 1	INC (A,1,C)	01	C = A AND B	AND (A,B,C)
10	C = A * 2	MUL2 (A,2,C)	10	C = A XOR B	XOR (A,B,C)
11	C = A * 8	MUL8 (A,8,C)	11	C = A NAND B	NAND (A,B,C)

4.3.- Starting from the single-cycle MIPS architecture studied in class, we want to include three new instructions (independently from each other):

1. **sll \$rd, \$rt, despl** # rd <= rt << despl
2. **jal address** # PC <= (PC+4)[31:28] & address & "00"
3. **add3 \$rd, \$rs, \$rt, \$rx** # rd <= rt + rs + rx

- a. Which are the blocks (if any) that can be used for the new instructions?
- b. Which are the new blocks (if any) that are needed for each instruction?
- c. Which are the new control signals (if any) that are needed for each instruction?

