Imperial College London

Lecture 7

Microarchitecture

# of a simplified RISC-V

Peter Cheung
Imperial College London

URL: www.ee.imperial.ac.uk/pcheung/teaching/EE2_CAS/
E-mail: p.cheung@imperial.ac.uk

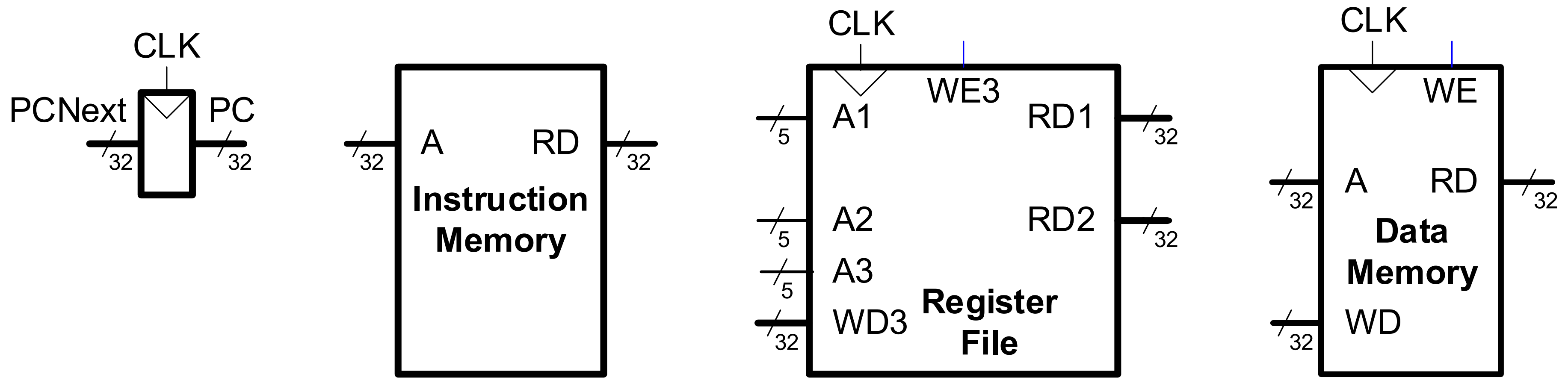
What is microarchitecture?

- **Microarchitecture:** how to implement an architecture in hardware
- Processor:
  - **Datapath:** functional blocks
  - **Control:** control signals

Based on: “*Digital Design and Computer Architecture (RISC-V Edition)*”
by Sarah Harris and David Harris (H&H),

RISC-V State Elements

- **State elements**: determines everything about a processor:
  - **Architectural state:**
    - 32 registers
    - Program Counter (PC)
    - Memory



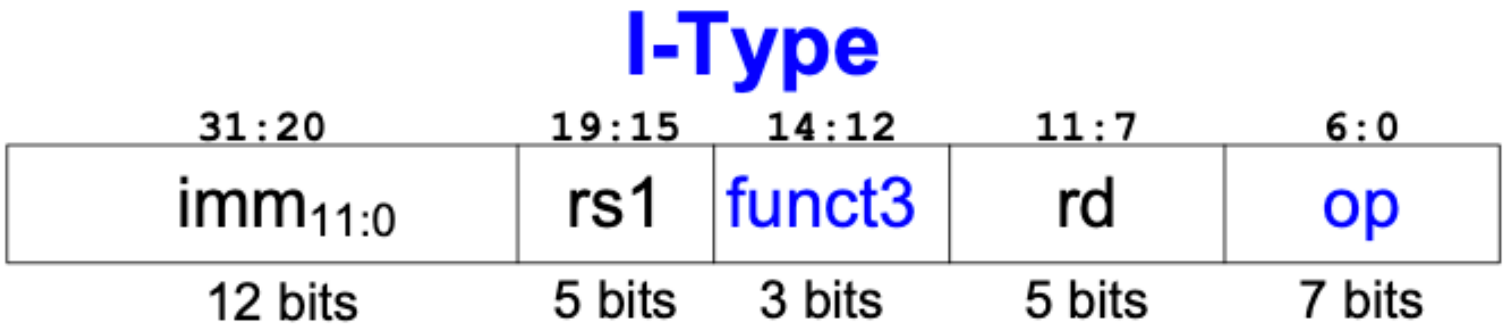
Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

Example Program

- Design datapath
- View example program executing

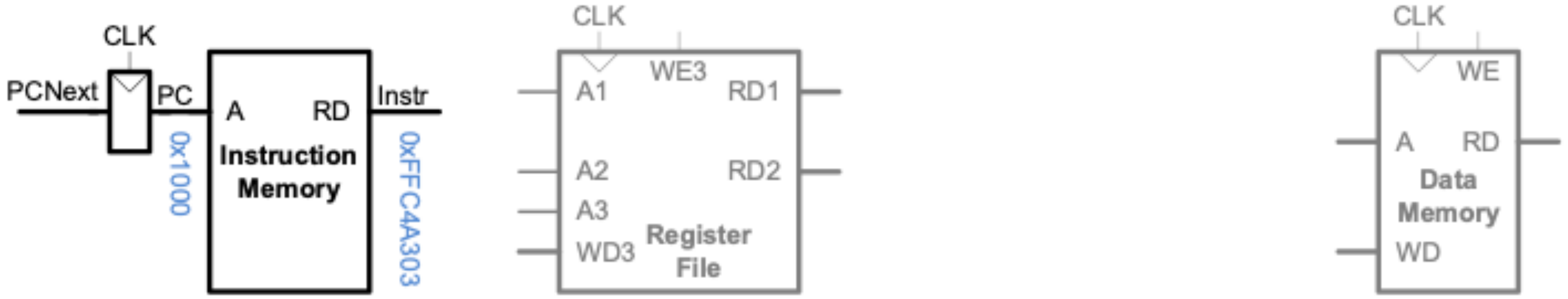
Address	Instruction	Type	Fields						Machine Language
`0x1000`	`L7: lw x6, -4(x9)`	I	**imm$_{11:0}$** `111111111100`		**rs1** `01001`	**f3** `010`	**rd** `00110`	**op** `0000011`	`FFC4A303`
`0x1004`	`sw x6, 8(x9)`	S	**imm$_{11:5}$** `0000000`	**rs2** `00110`	**rs1** `01001`	**f3** `010`	**imm$_{4:0}$** `01000`	**op** `0100011`	`0064A423`
`0x1008`	`or x4, x5, x6`	R	**funct7** `0000000`	**rs2** `00110`	**rs1** `00101`	**f3** `110`	**rd** `00100`	**op** `0110011`	`0062E233`
`0x100C`	`beq x4, x4, L7`	B	**imm$_{12,10:5}$** `1111111`	**rs2** `00100`	**rs1** `00100`	**f3** `000`	**imm$_{4:1,11}$** `10101`	**op** `1100011`	`FE420AE3`

```
lw x6, -4(x9)
lw rd, imm(rs1)
```



Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

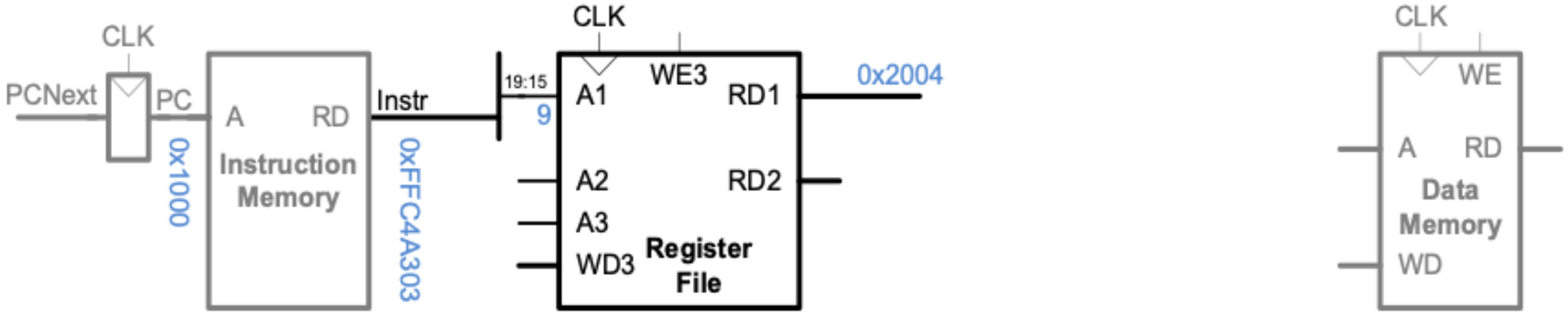
Step 1: Instruction Fetch



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

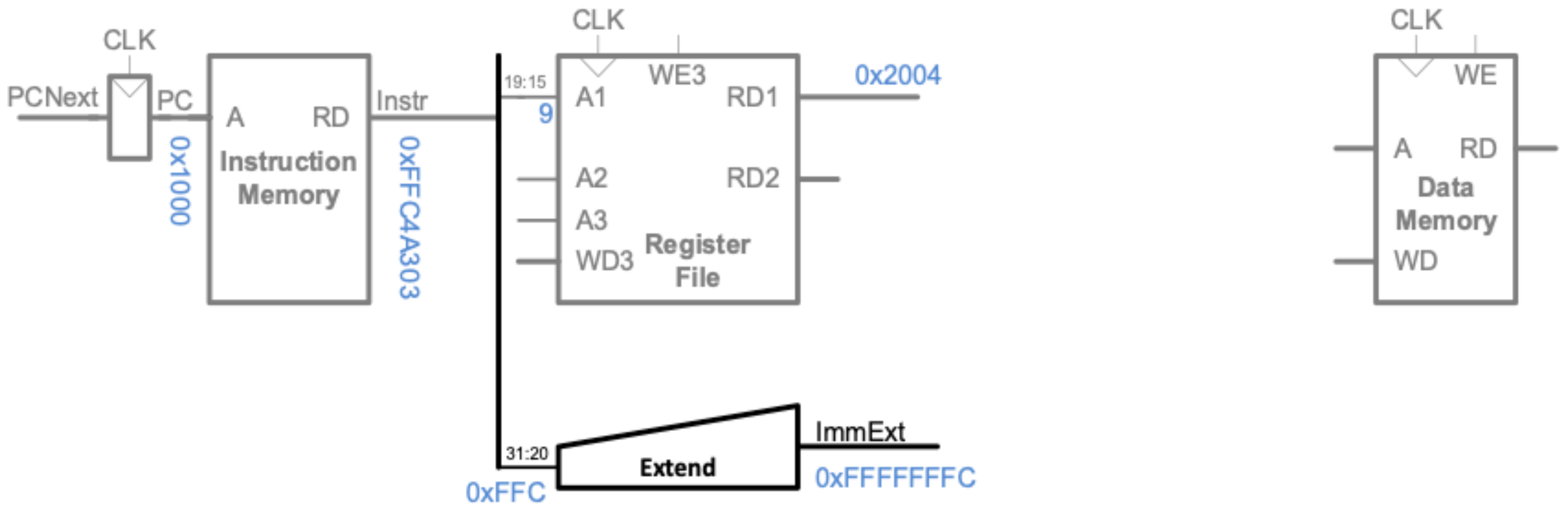
Step 2: Read Source Operand (rs1)



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

Step 3: Extend the immediate constant

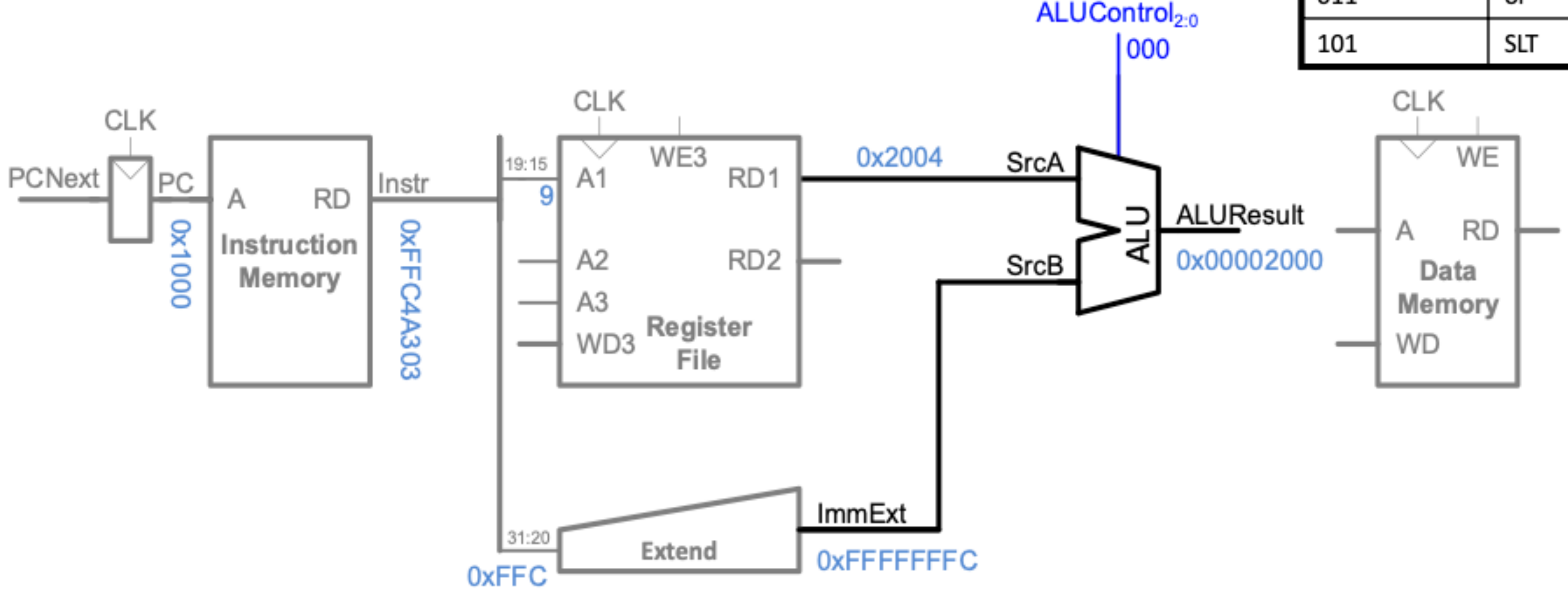


Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

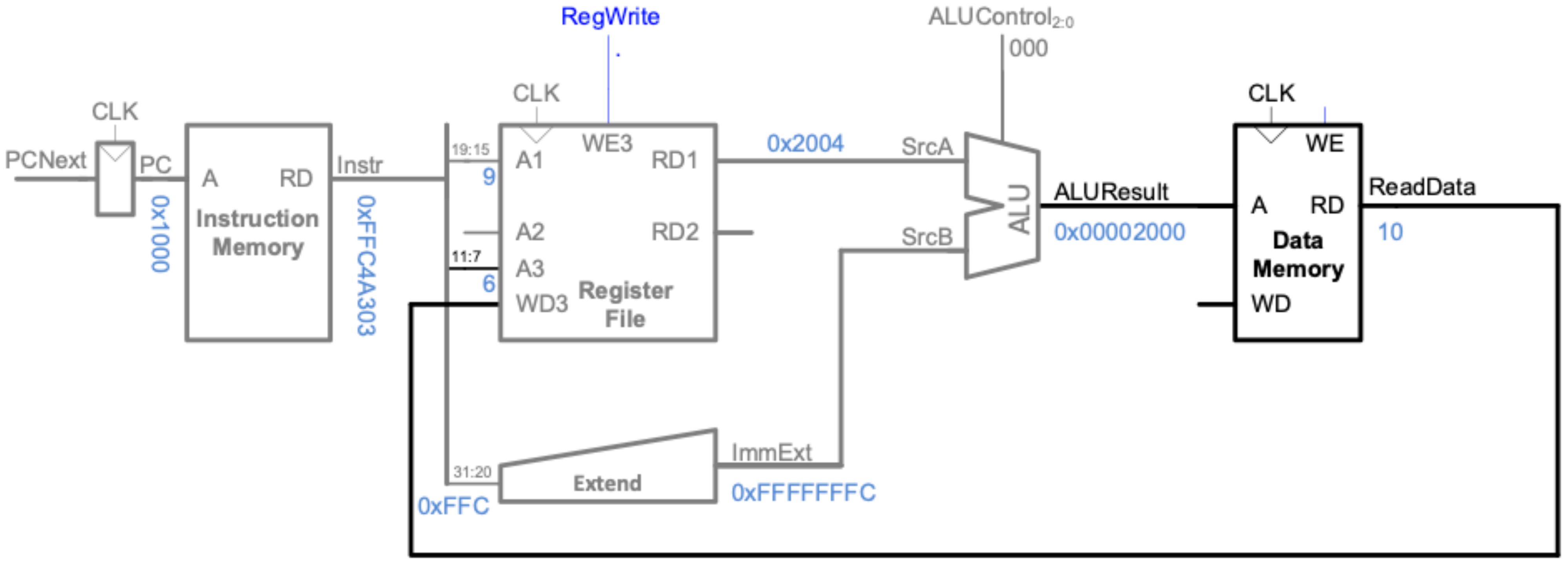
Step 4: Calculate memory address

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
`0x1000`	`L7: lw x6, -4(x9)`	I	`111111111100`	`01001`	`010`	`00110`	`0000011`	`FFC4A303`

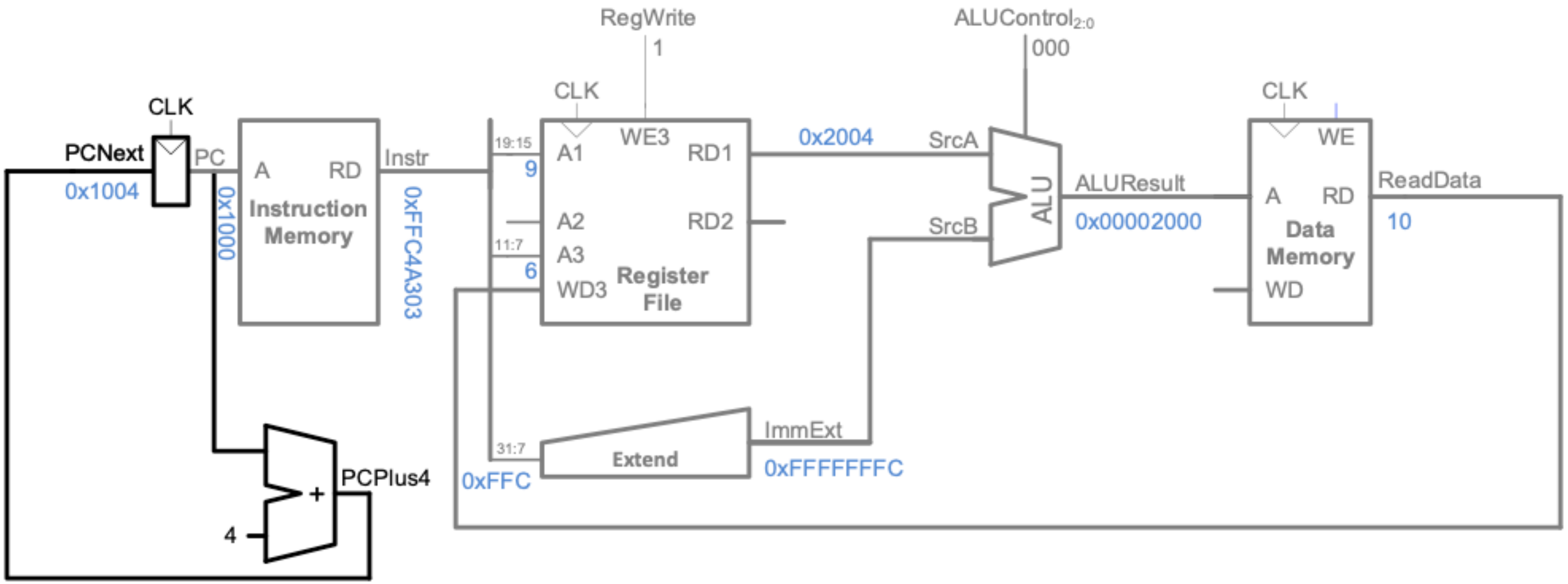
Step 5: Read data from memory & write to Reg



Address	Instruction	Type	Fields				Machine Language	
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

Step 6: Determine address of next instruction



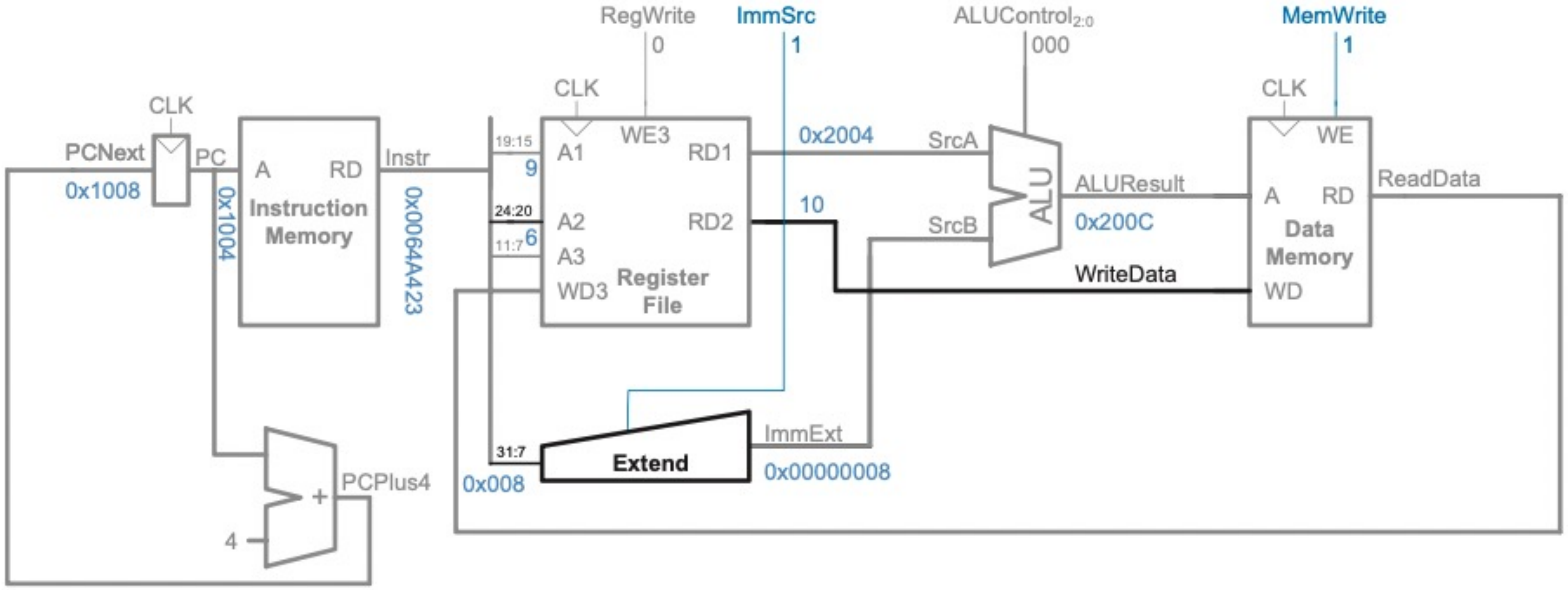
Address	Instruction	Type	Fields				Machine Language
			imm$_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Based on: “*Digital Design and Computer Architecture (RISC-V Edition)*”
by Sarah Harris and David Harris (H&H),

Implementation of the ”sw” instruction

- **Immediate:** now in {instr[31:25], instr[11:7]}
- **Add control signals:** ImmSrc, MemWrite

Address	Instruction	Type	Fields						Machine Language
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
0x1004	sw x6, 8(x9)	S	0000000	00110	01001	010	01000	0100011	0064A423

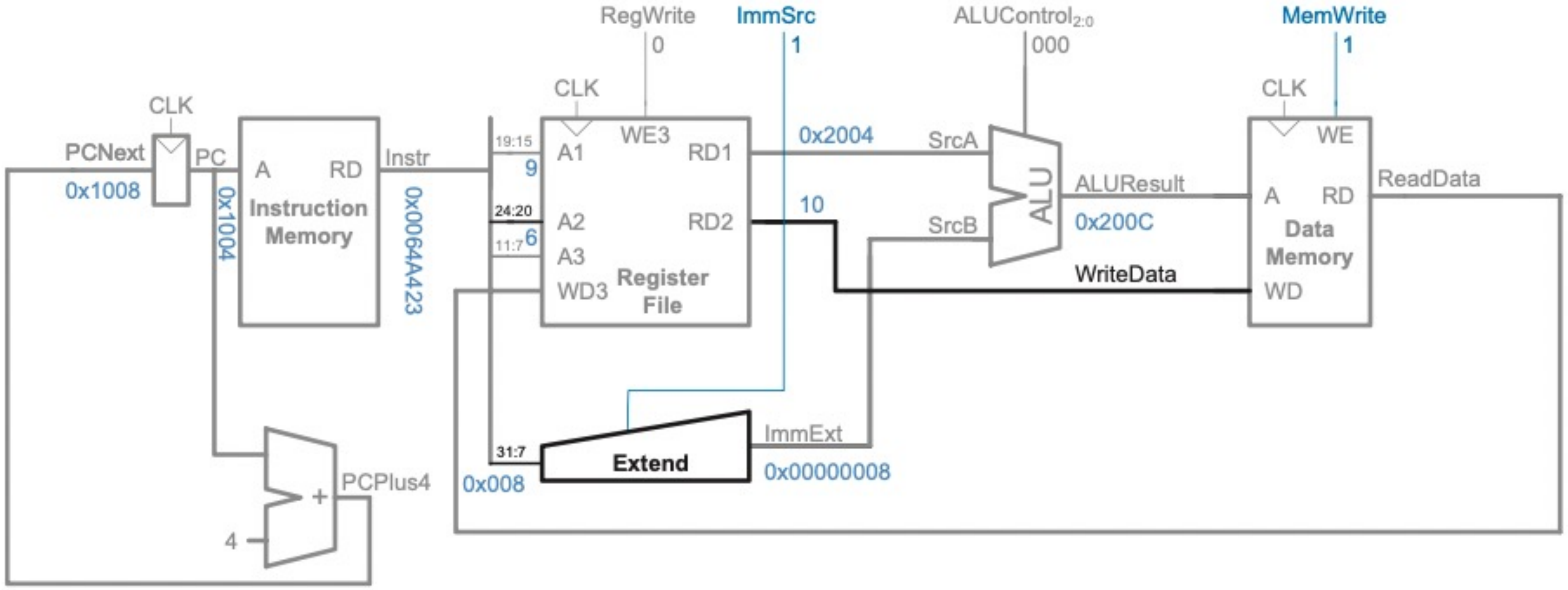


Based on: “*Digital Design and Computer Architecture (RISC-V Edition)*”
by Sarah Harris and David Harris (H&H),

Implementation of the ”sw” instruction

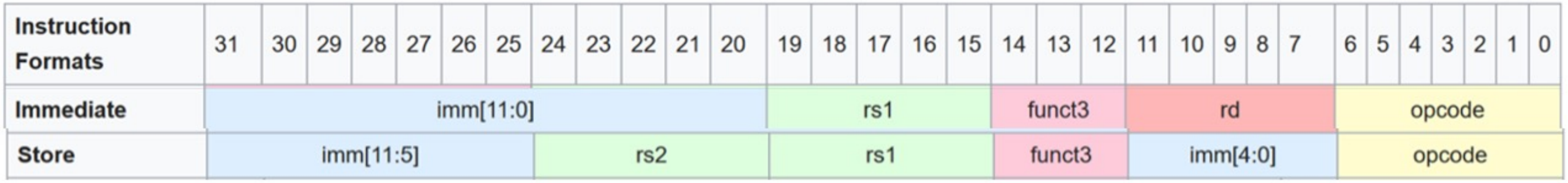
- **Immediate:** now in {instr[31:25], instr[11:7]}
- **Add control signals:** ImmSrc, MemWrite

Address	Instruction	Type	Fields						Machine Language
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
`0x1004`	`sw x6, 8(x9)`	S	`0000000`	`00110`	`01001`	`010`	`01000`	`0100011`	`0064A423`

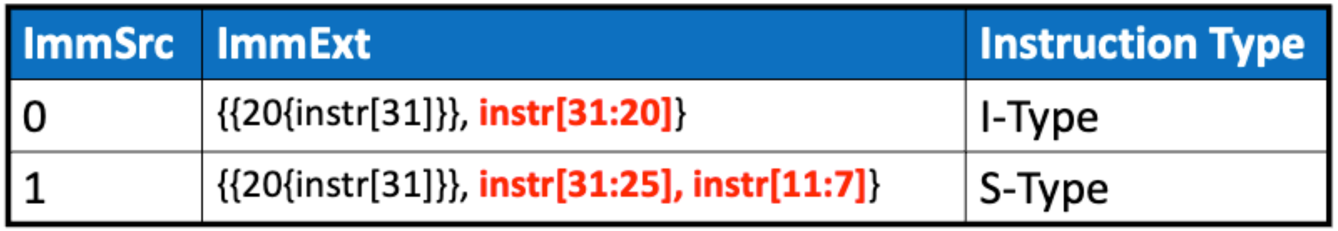


Based on: “*Digital Design and Computer Architecture (RISC-V Edition)*”
by Sarah Harris and David Harris (H&H),

Immediate offset for I-type and S-type are different



Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						



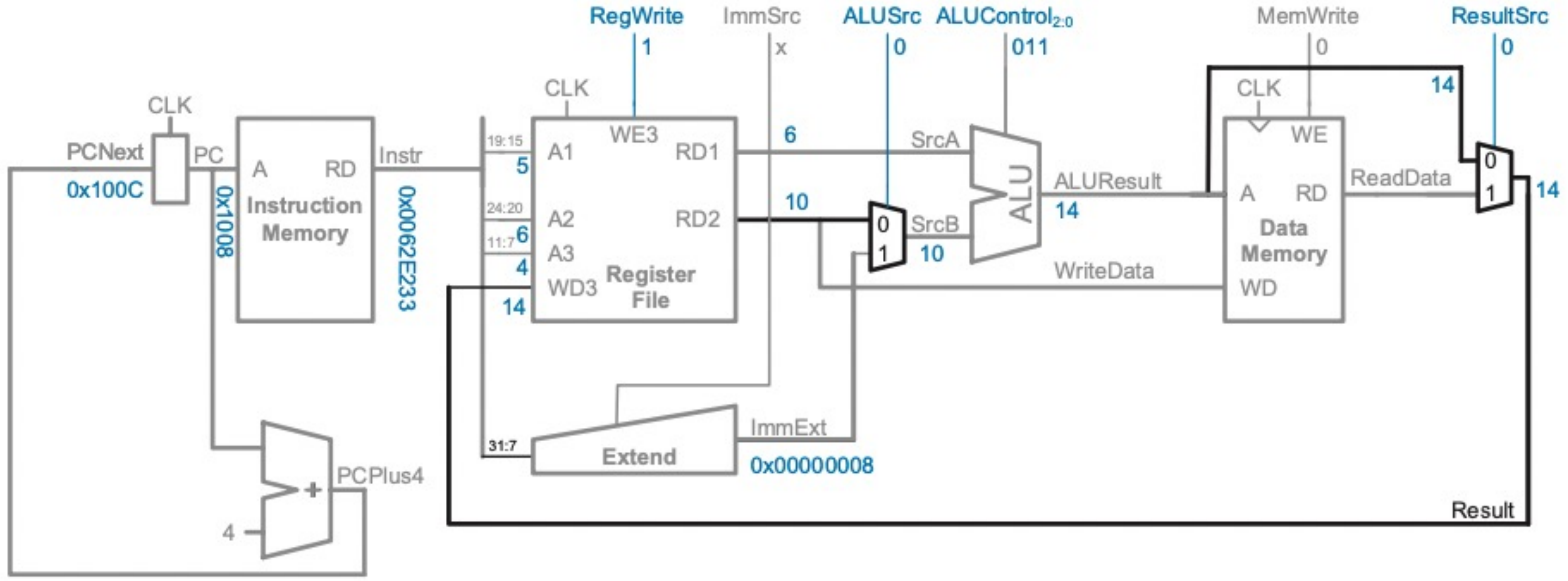
ImmSrc	ImmExt	Instruction Type
0	{{20{instr[31]}}, **instr[31:20]**}	I-Type
1	{{20{instr[31]}}, **instr[31:25], instr[11:7]**}	S-Type

Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

Implementation of the ”or” instruction

- Read from **rs1** and **rs2** (instead of **imm**)
- Write *ALUResult* to **rd**

Address	Instruction	Type	Fields						Machine Language
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

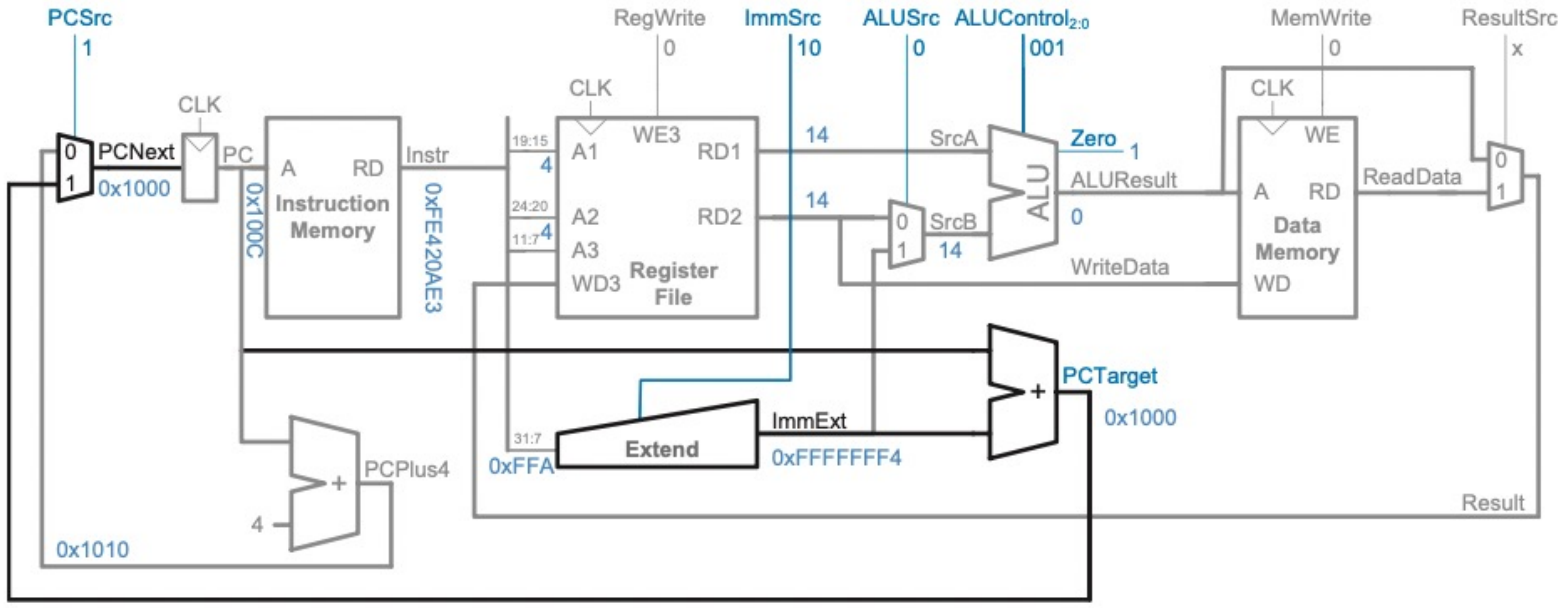


Based on: “*Digital Design and Computer Architecture (RISC-V Edition)*”
by Sarah Harris and David Harris (H&H),

Implementation of the ”beq” instruction

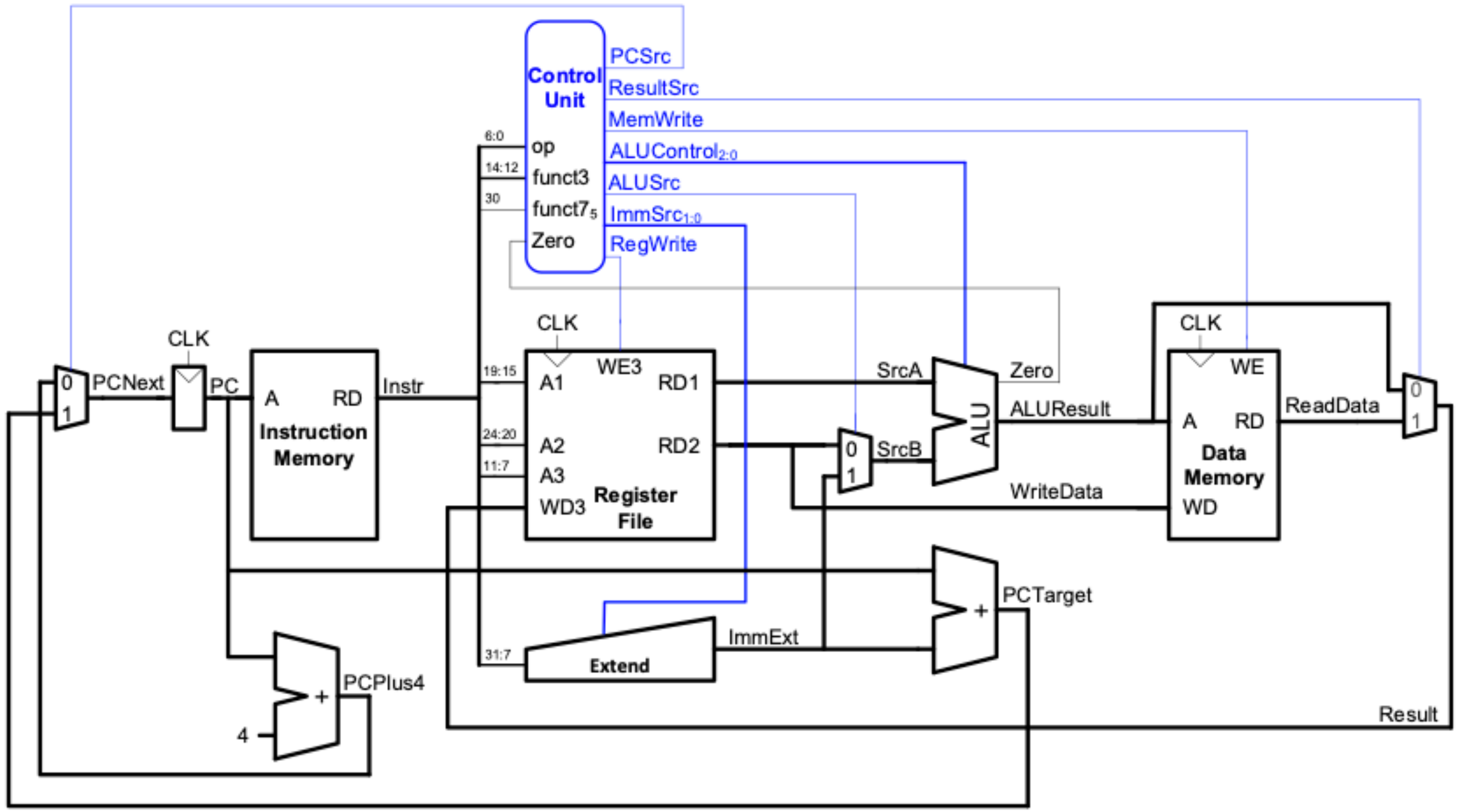
Calculate **target address:** PCTarget = PC + imm

Address	Instruction	Type	Fields						Machine Language
			$imm_{12,10:5}$	rs2	rs1	f3	$imm_{4:1,11}$	op	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3



Based on: “*Digital Design and Computer Architecture (RISC-V Edition)*”
by Sarah Harris and David Harris (H&H),

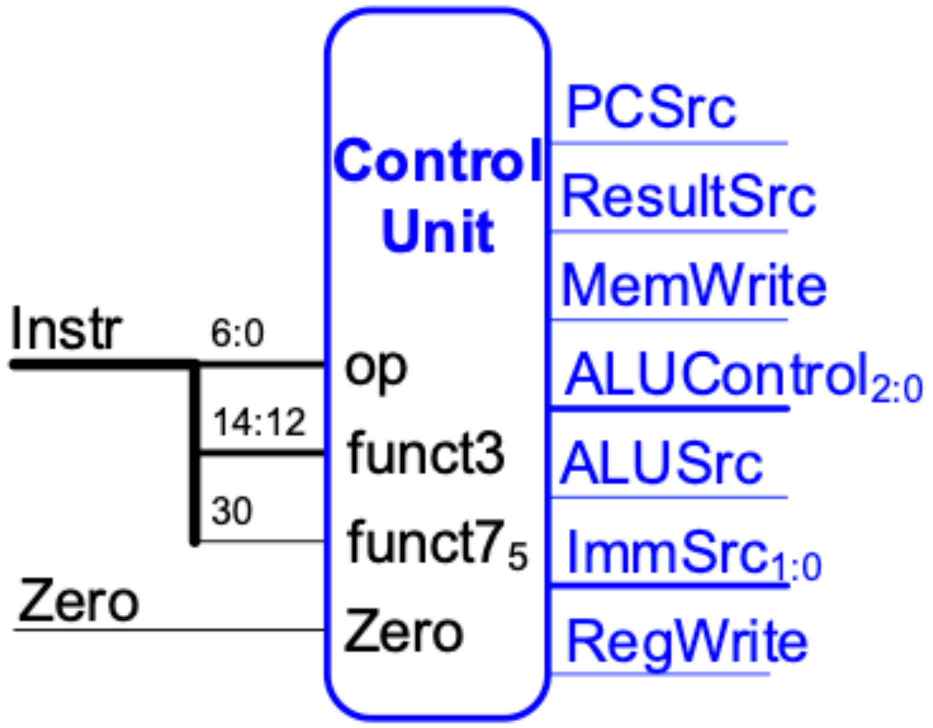
Adding the Control Unit



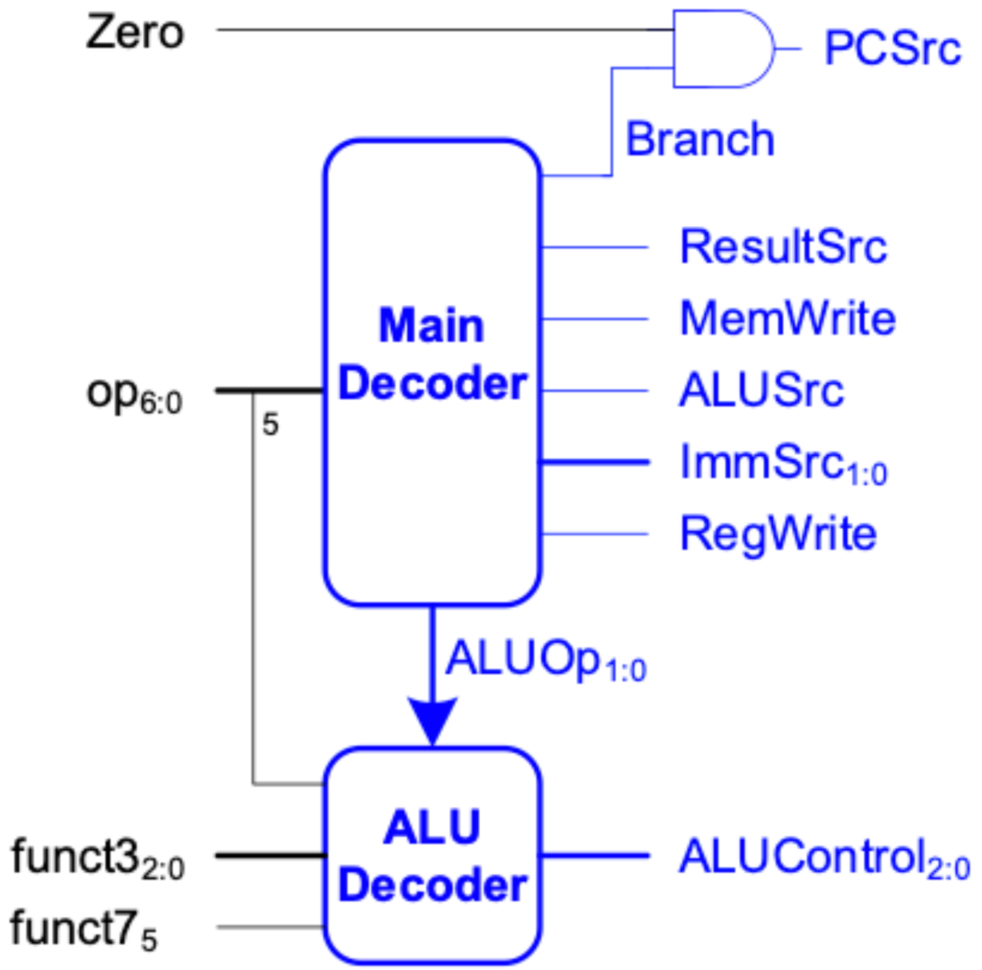
Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

Two different views of the Control Unit

High-Level View

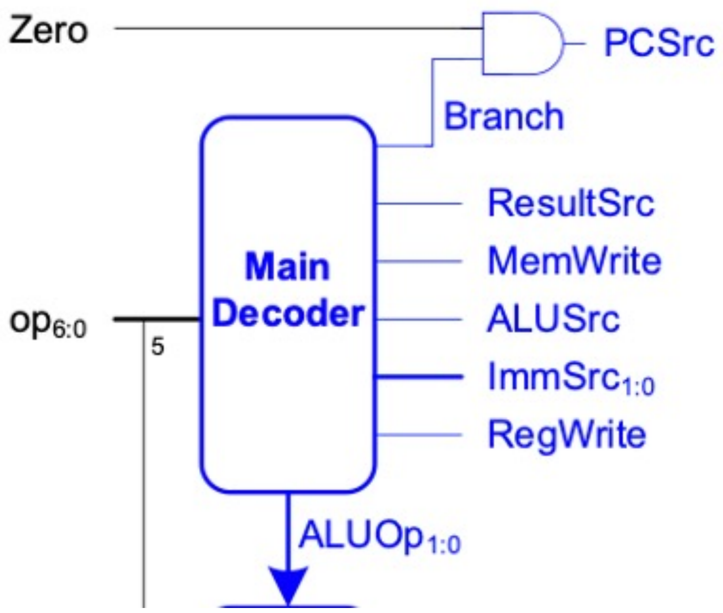


Low-Level View



Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*" by Sarah Harris and David Harris (H&H),

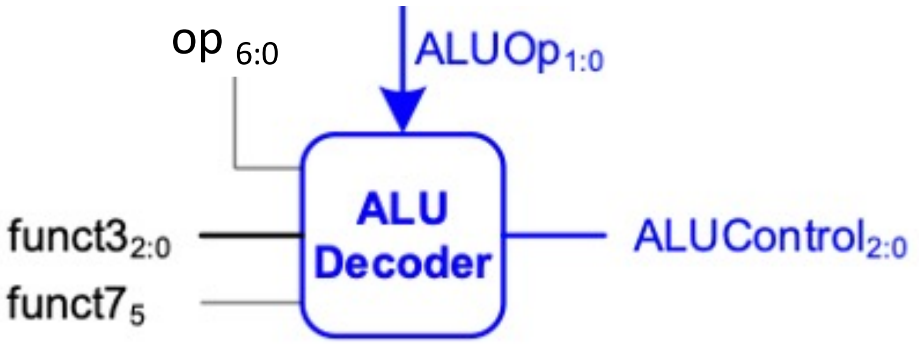
Main decoder

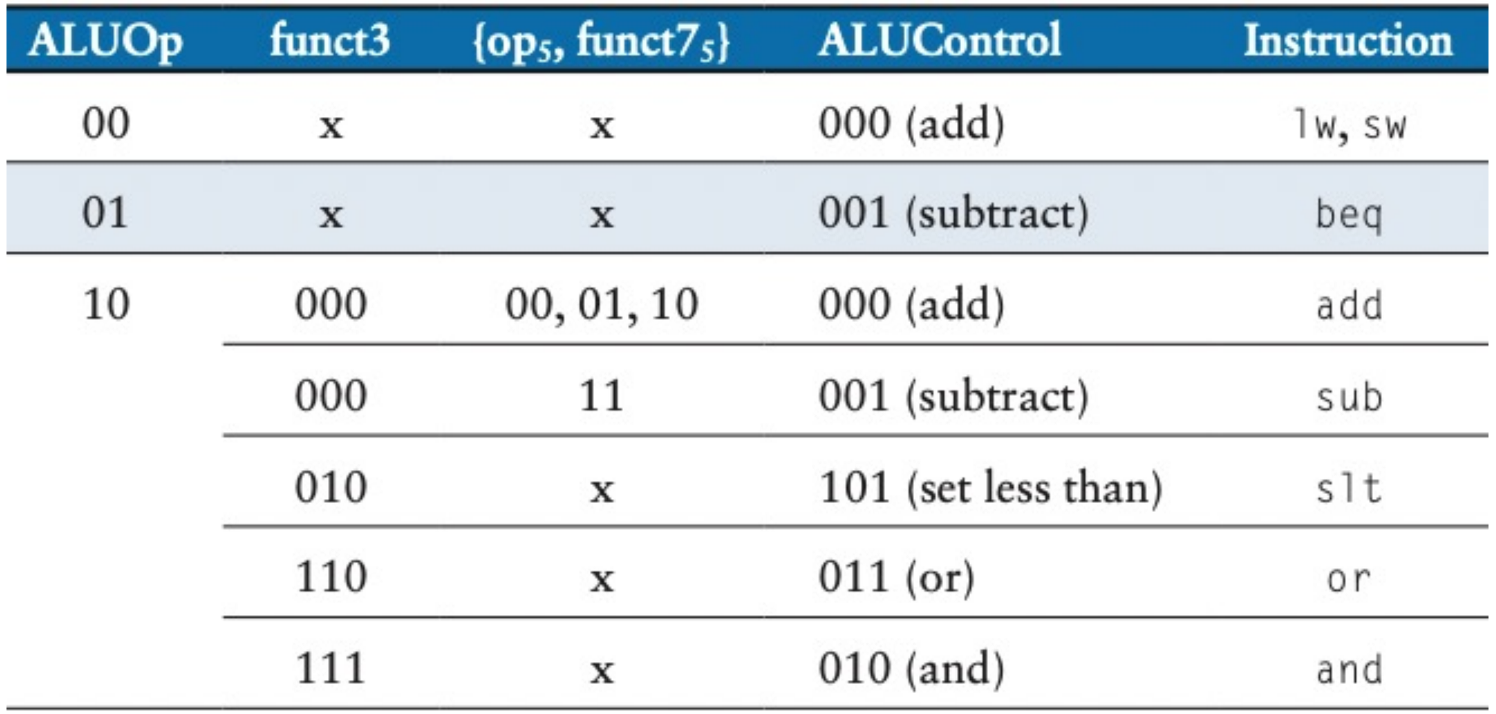


Instruction	Op	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
`lw`	0000011	1	00	1	0	1	0	00
`sw`	0100011	0	01	1	1	x	0	00
R-type	0110011	1	xx	0	0	0	0	10
`beq`	1100011	0	10	0	0	x	1	01

Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

ALU Decoder



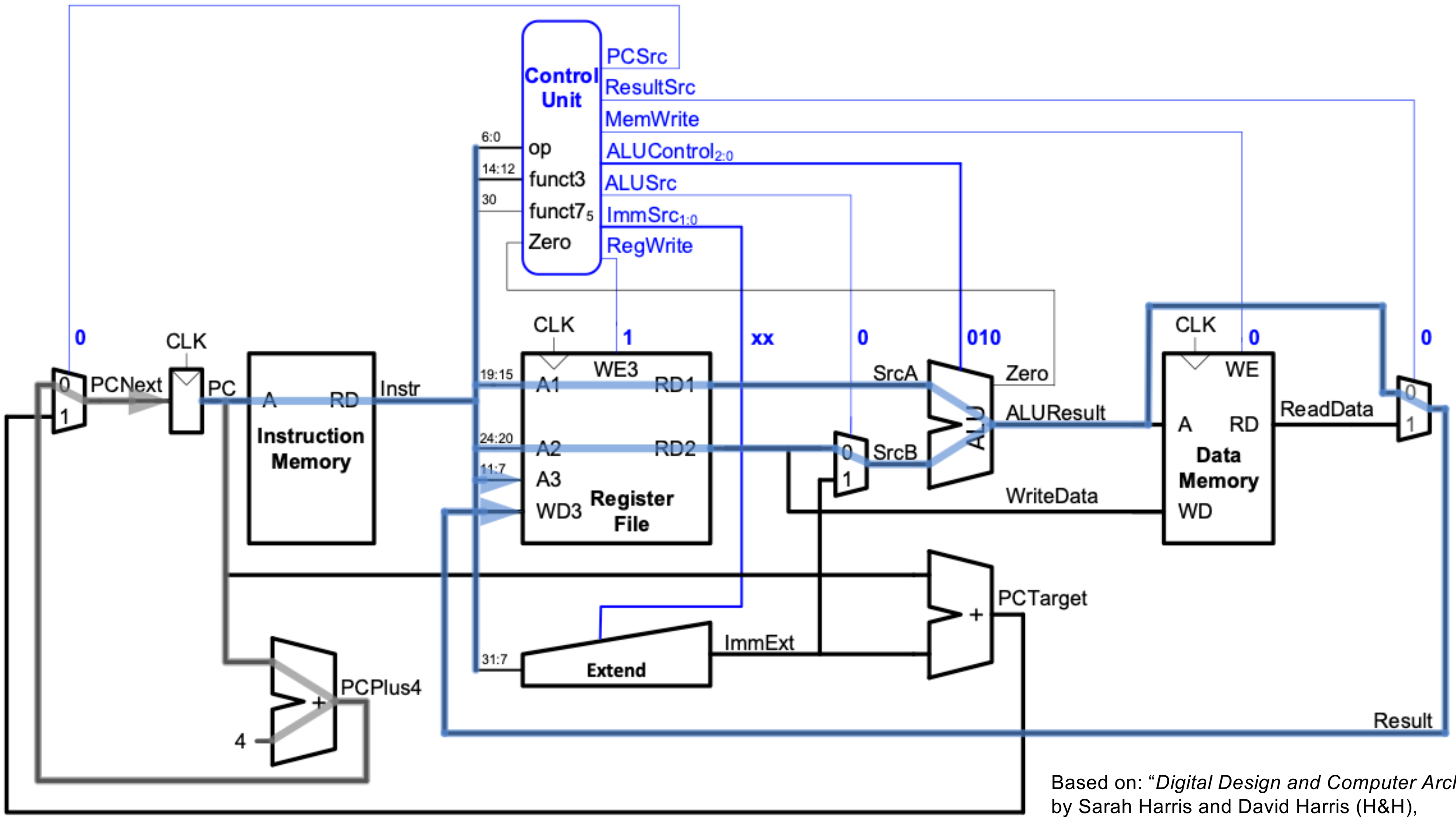


ALUOp	funct3	$\{op_5, funct7_5\}$	ALUControl	Instruction
00	x	x	000 (add)	`lw`, `sw`
01	x	x	001 (subtract)	`beq`
10	000	00, 01, 10	000 (add)	`add`
	000	11	001 (subtract)	`sub`
	010	x	101 (set less than)	`slt`
	110	x	011 (or)	`or`
	111	x	010 (and)	`and`

Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*"
by Sarah Harris and David Harris (H&H),

Example – Control for `and x5, x6, x7`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
51	**R-type**	1	XX	0	0	0	0	010

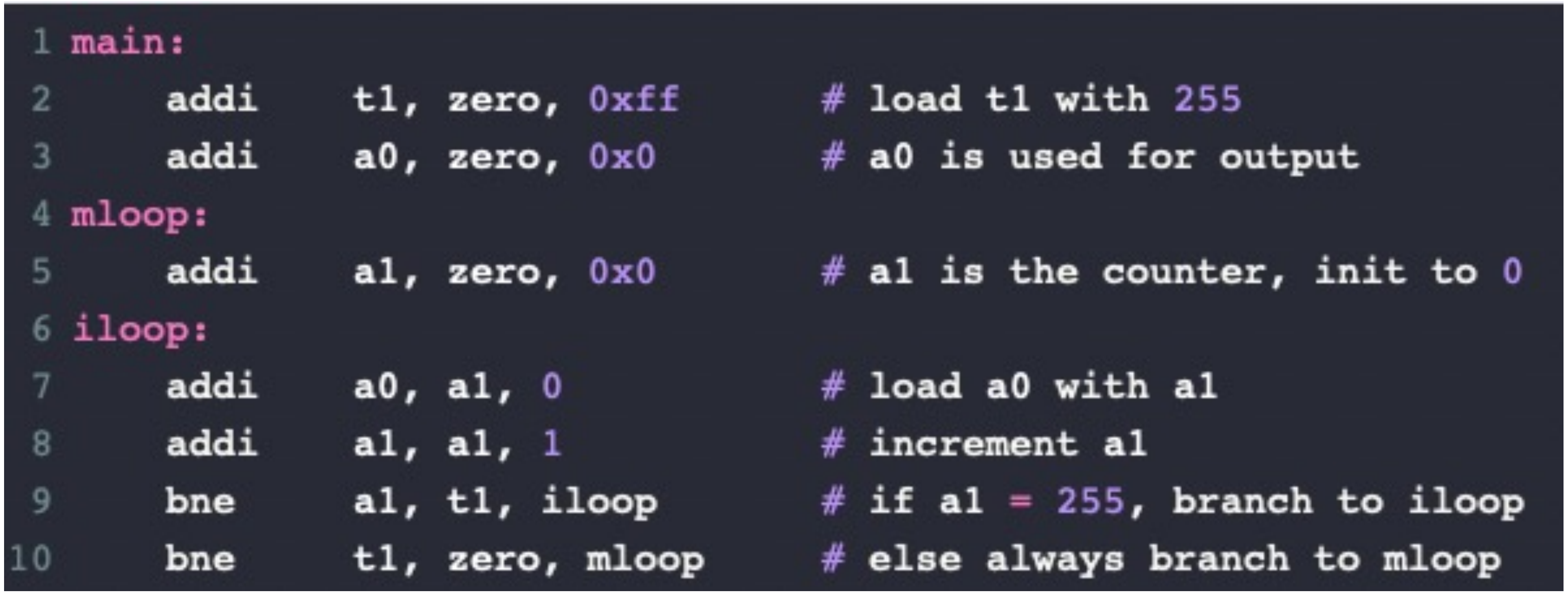


Based on: "*Digital Design and Computer Architecture (RISC-V Edition)*" by Sarah Harris and David Harris (H&H),

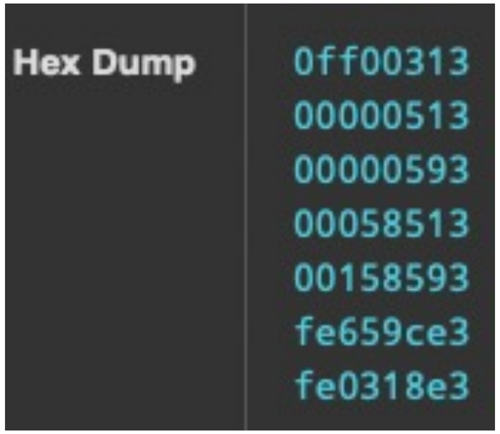
Lab 4 – A Very Basic RISC-V CPU

- Start working as a Team – 2 pairs allocated by me
- **Lab objectives**:
  1. To get to know your teammates.
  2. To establish a Github Repo for your team where everyone's contribute towards.
  3. To learn about TWO RISC-V instructions in great details.
  4. To design a simple CPU that executes these two instructions.
  5. To use execute a short program using only these two instructions. The program implements the binary counter in Lab 1, but in software.
  6. Stretched goal – to implement a third instruction accessing data memory. With this, implement the sinewave generator in software.

Lab 4 – Program to execute



```
main:
     addi    t1, zero, 0xff       # load t1 with 255
     addi    a0, zero, 0x0        # a0 is used for output
mloop:
     addi    a1, zero, 0x0        # a1 is the counter, init to 0
iloop:
     addi    a0, a1, 0            # load a0 with a1
     addi    a1, a1, 1            # increment a1
     bne     a1, t1, iloop        # if a1 = 255, branch to iloop
     bne     t1, zero, mloop      # else always branch to mloop
```

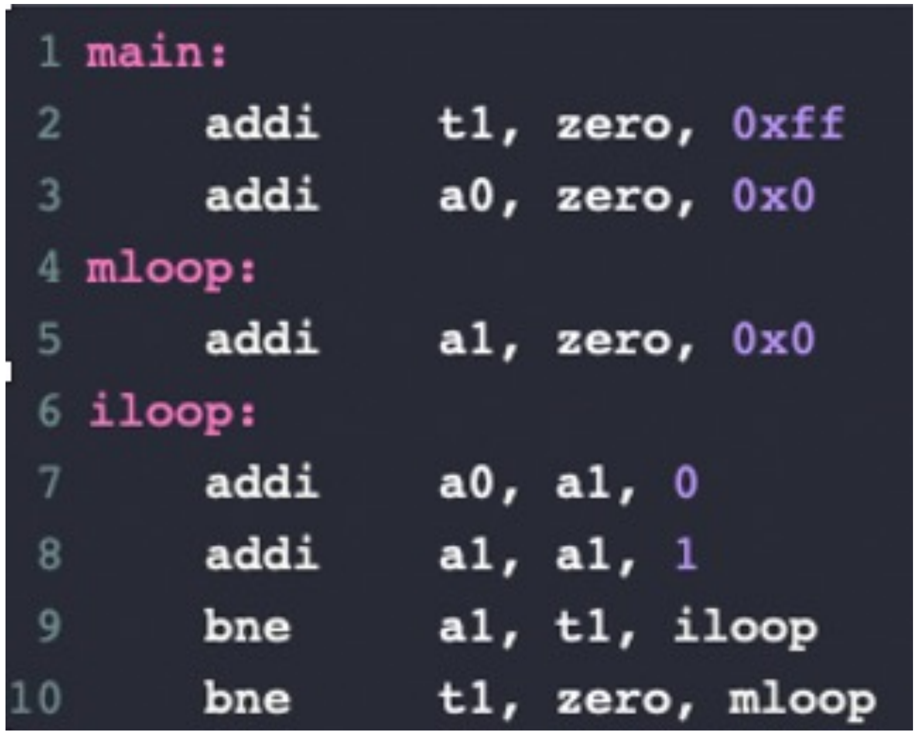


```
Hex Dump     0ff00313
             00000513
             00000593
             00058513
             00158593
             fe659ce3
             fe0318e3
```

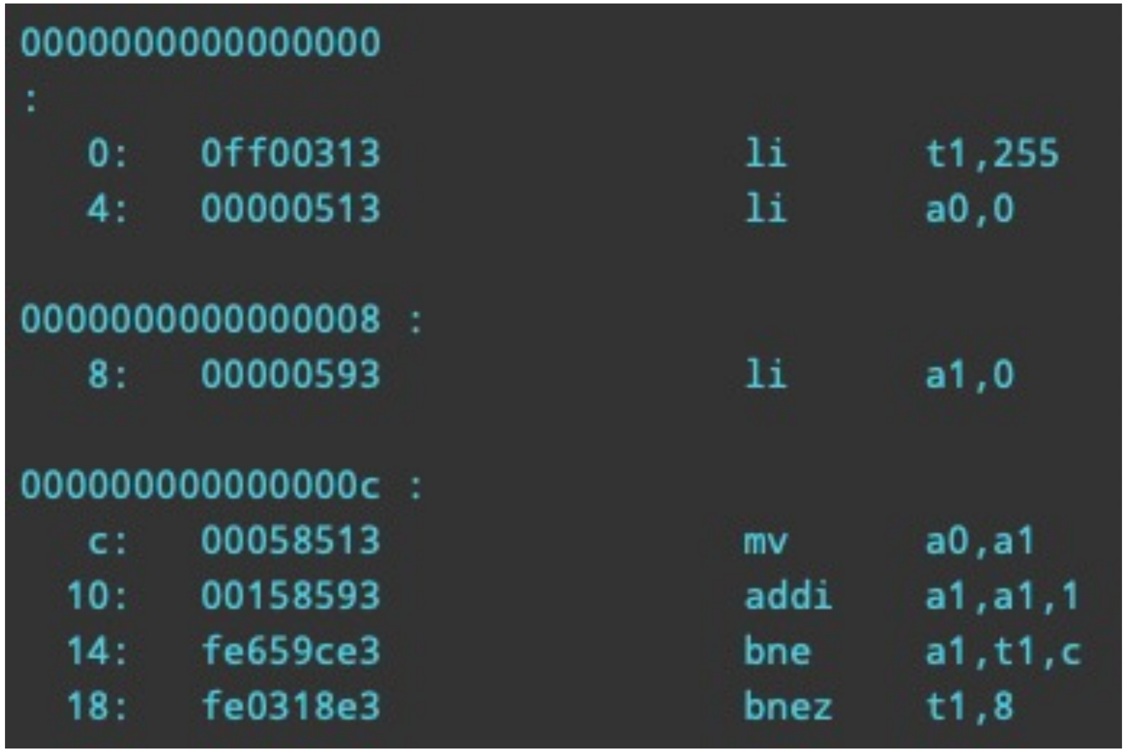
Online RISC-V Assembler:

https://riscvasm.lucasteske.dev

Lab 4 – Pseudoinstruction is easier to read



```
main:
        addi    t1, zero, 0xff
        addi    a0, zero, 0x0
mloop:
        addi    a1, zero, 0x0
iloop:
        addi    a0, a1, 0
        addi    a1, a1, 1
        bne     a1, t1, iloop
        bne     t1, zero, mloop
```

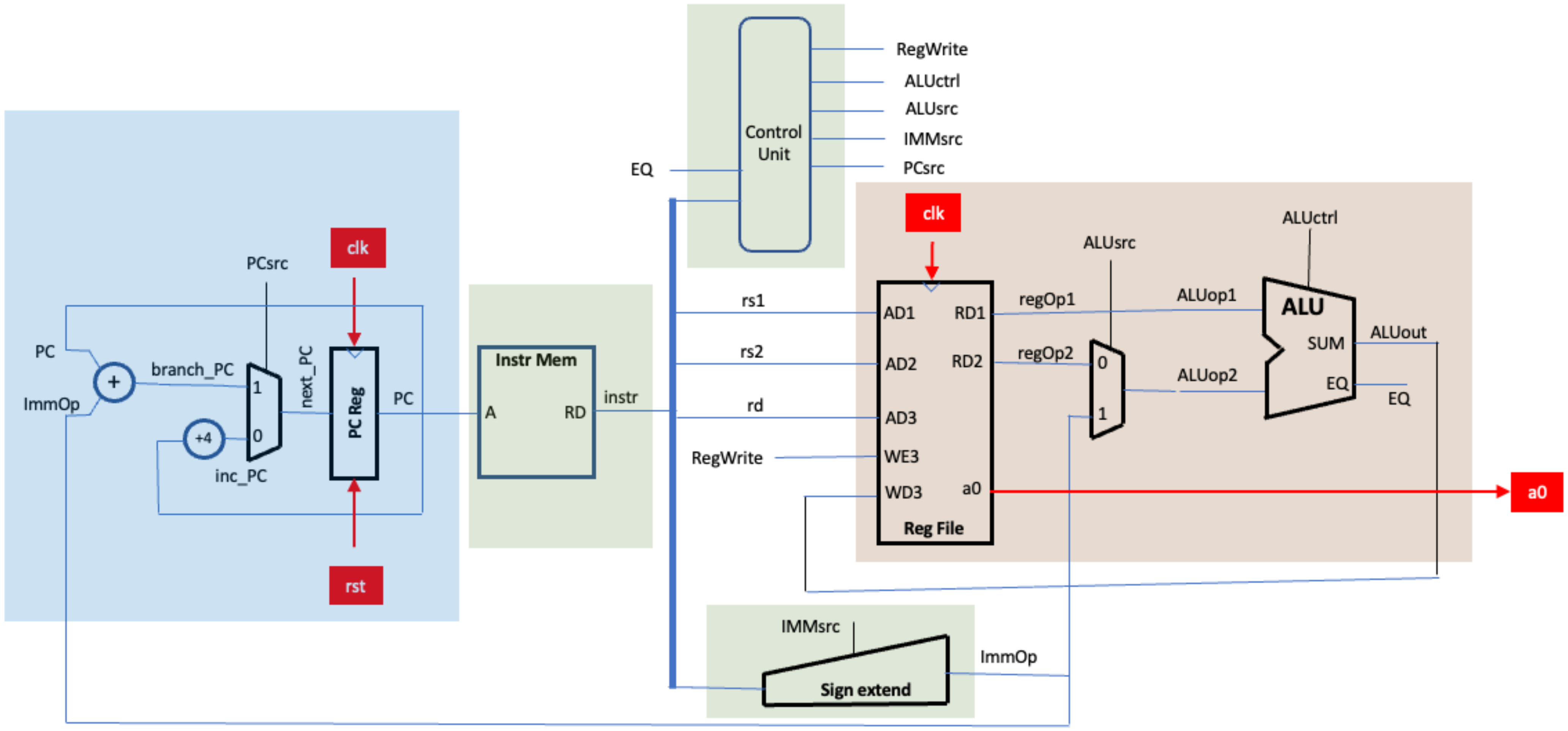


```
0000000000000000
:
   0:   0ff00313                li      t1,255
   4:   00000513                li      a0,0

0000000000000008 :
   8:   00000593                li      a1,0

000000000000000c :
   c:   00058513                mv      a0,a1
  10:   00158593                addi    a1,a1,1
  14:   fe659ce3                bne     a1,t1,c
  18:   fe0318e3                bnez    t1,8
```

Lab 4 – Overall block diagram



Control Unit
RegWrite
ALUctrl
ALUsrc
IMMsrc
PCsrc
EQ
PCsrc
clk
PC
branch_PC
1
0
ImmOp
+4
inc_PC
next_PC
PC Reg
PC
rst
Instr Mem
A
RD
instr
rs1
rs2
rd
RegWrite
clk
AD1
AD2
AD3
WE3
WD3
RD1
RD2
a0
Reg File
regOp1
regOp2
ALUsrc
0
1
ALUctrl
ALUop1
ALUop2
ALU
SUM
EQ
ALUout
EQ
a0
IMMsrc
Sign extend
ImmOp