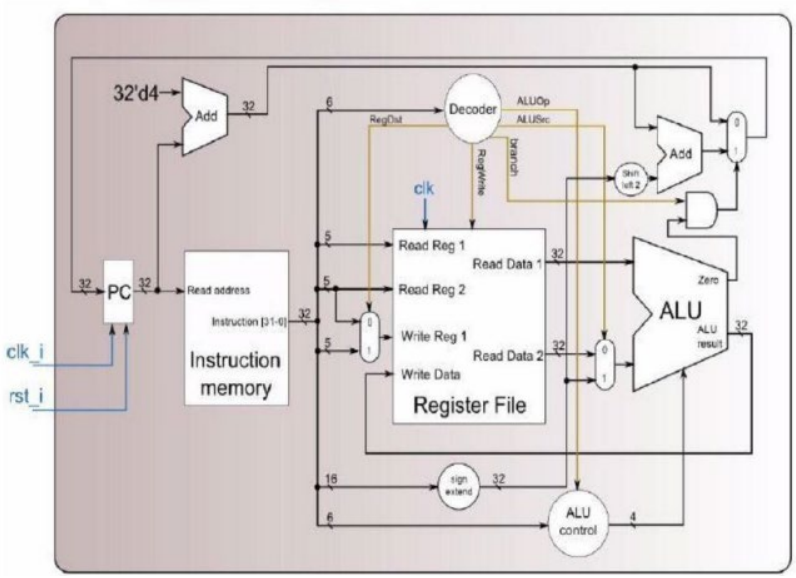
Computer Organization

Architecture diagrams:



Hardware module analysis:

- Adder

Composed of 32 full-adders, with “cin” of the first bit be 0.
full-adder:

Input			Output	
a	b	cin	result	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

➔ result = a ^ b ^ cin
cout = (a & b) | (a & cin) | (b & cin)

➢ ALU_Ctrl

	Input									Output			
	fc5	fc4	fc3	fc2	fc1	fc0	op2	op1	op0	ctr3	ctr2	ctr1	ctr0
add	1	0	0	0	0	0	0	0	0	0	0	1	0
sub	1	0	0	0	1	0	0	0	0	0	1	1	0
and	1	0	0	1	0	0	0	0	0	0	0	0	0
or	1	0	0	1	0	1	0	0	0	0	0	0	1
slt	1	0	1	0	1	0	0	0	0	0	1	1	1
addi	x	x	x	x	x	x	1	0	0	0	0	1	0
slti	x	x	x	x	x	x	1	0	1	0	1	1	1
beq	x	x	x	x	x	x	0	1	0	0	1	1	0

given function field | ALU opcode designed by me | ALU control

➔ ctr3 = 0
ctr2 = (fc1 & (~op2)) | op1 | op0
ctr1 = (op2 | op1 | op0) | (~fc2)
ctr0 = op0 | (~op2 & ~op1 & (fc0 | fc3))

➢ ALU

I use the behavioral 32-bit ALU from the textbook directly.

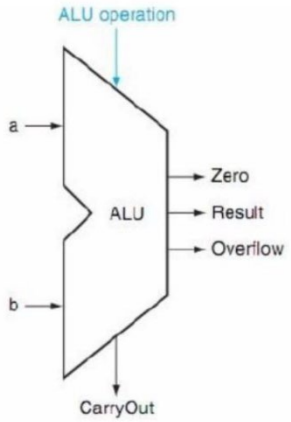


FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12. This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.

➢ Decoder

	Input						Output						
	op5	op4	op3	op2	op1	op0	RW	alu op2	alu op1	alu op0	Src	RD	Brh
R-f	0	0	0	0	0	0	1	0	0	0	0	1	0
addi	0	0	1	0	0	0	1	1	0	0	1	0	0
slti	0	0	1	0	1	0	1	1	0	1	1	0	0
beq	0	0	0	1	0	0	0	0	1	0	0	x	1

given op field — RegWrite — ALU opcode (designed by me) — ALUSrc — RegDst — Branch

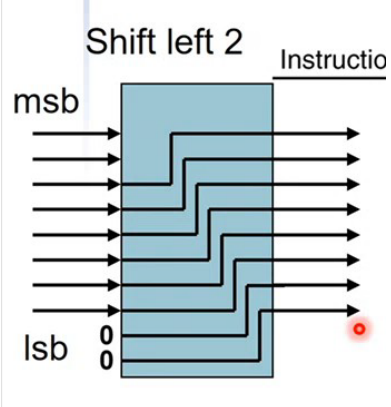
➔ RW = ~op2
alu_op2 = op3
alu_op1 = op2
alu_op0 = op1
Src = op3
RD = ~op3
Brh = op2

➢ MUX_2to1

The function of this module is to choose one data from the given two data. So the output depends on how “select_i” specifies and chooses the corresponding data.

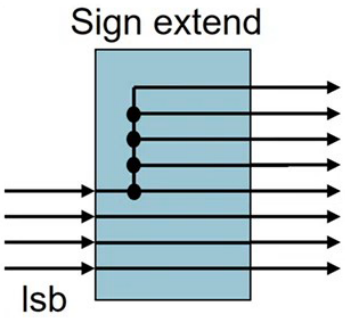
➢ Shift_Left_Two_32

Every bit shifts left by two bits, with the last two lsb given 0s.



- Sign_Extend

Bit 16~31 are the duplicate of msb of the input, while the remaining bits are the same as the input.



- Simple_Single_CPU

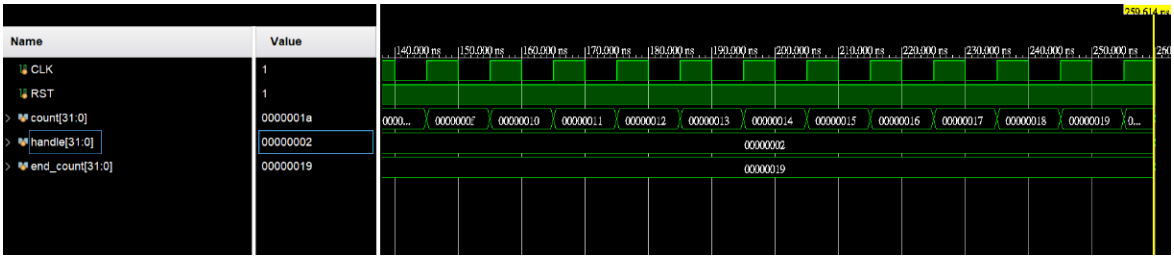
By reference to the architecture diagram, include all the required modules, and carefully connect the wires between them, done!

Finished part:

in Tcl Console:

data1.txt	data2.txt
# run 1000ns 2 r0= 0 r1= 10 r2= 4 r3= 0 r4= 0 r5= 6 r6= 0 r7= 0 r8= 0 r9= 0 r10= 0 r11= 0 r12= 0	# run 1000ns 2 r0= 0 r1= 1 r2= 0 r3= 0 r4= 0 r5= 0 r6= 0 r7= 14 r8= 0 r9= 15 r10= 0 r11= 0 r12= 0

From the result showing above, we see that the single cycle CPU can correctly handle with the given instructions. It can distinguish different instructions from the binary code and do the corresponding computation. The calculation result is correct!



According to the wave chart, we see that count increments by one at the positive edge of CLK. When the number of count is equal to end_count (25), the results will be written to Result.txt.

in Result.txt:

data1.txt	data2.txt
CO_P2_Result.txt - 記事本 檔案(F) 編輯(E) 格式(O) 檢視(V) r0= 0 r1= 10 r2= 4 r3= 0 r4= 0 r5= 6 r6= 0 r7= 0 r8= 0 r9= 0 r10= 0 r11= 0 r12= 0	CO_P2_Result.txt - 記事本 檔案(F) 編輯(E) 格式(O) 檢視(V) r0= 0 r1= 1 r2= 0 r3= 0 r4= 0 r5= 0 r6= 0 r7= 14 r8= 0 r9= 15 r10= 0 r11= 0 r12= 0

Problems you met and solutions:

1. When I first time ran my code, I got error messages like this:

I looked for solutions on the internet, and saw someone suggests that I could restart my project or reboot my computer. Following his suggestion, I successfully ran my code the next time.

To double check there wasn't any problem with my code, I even asked my friend to run my code on her computer, and she got no error as well.

2. I didn’t get the correct answer at the first try.

After carefully examined my code, I found that I misunderstood the meaning of RegDst. So the solution was to revise the truth table, then I got the correct answer.

Summary:

Such an exciting experiment!

At first sight of the requirement, I thought implementing a single cycle CPU was very complicated and challenging. However, after spend some time realizing the relationship among each module, it was not a hard task. All I need to do is understand how the CPU operates, then I can work on the design of each module. Carefully connect the wires and the modules, a single cycle CPU is done!