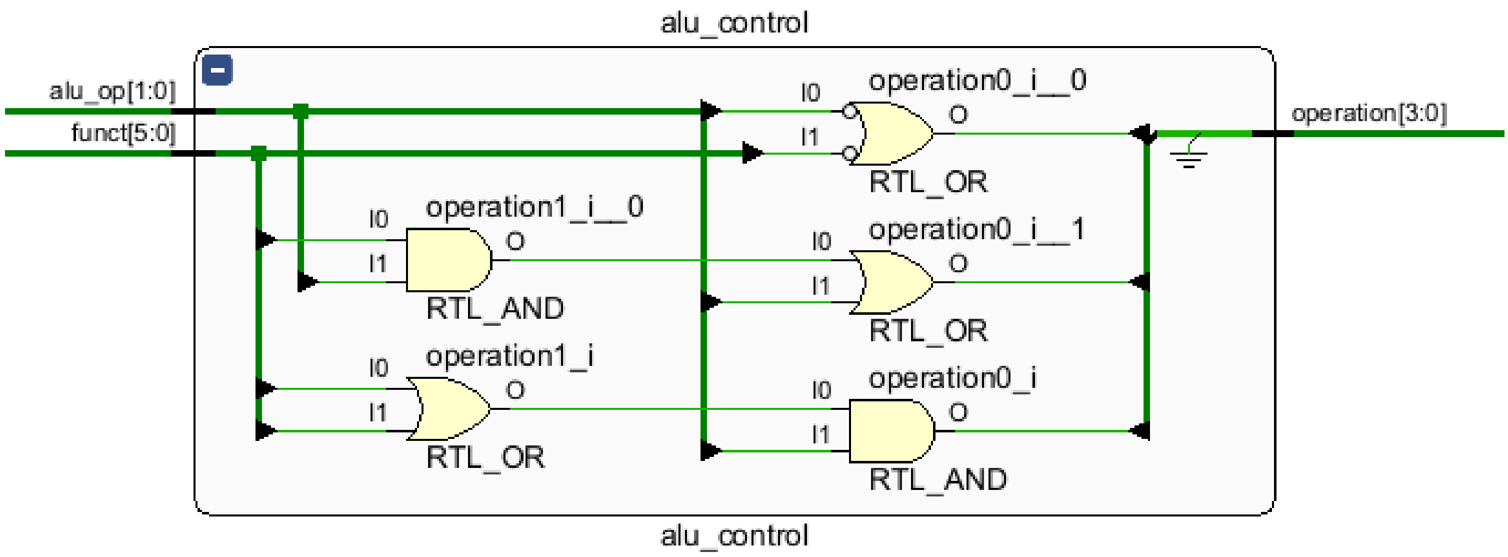
CO_Lab2_111550076

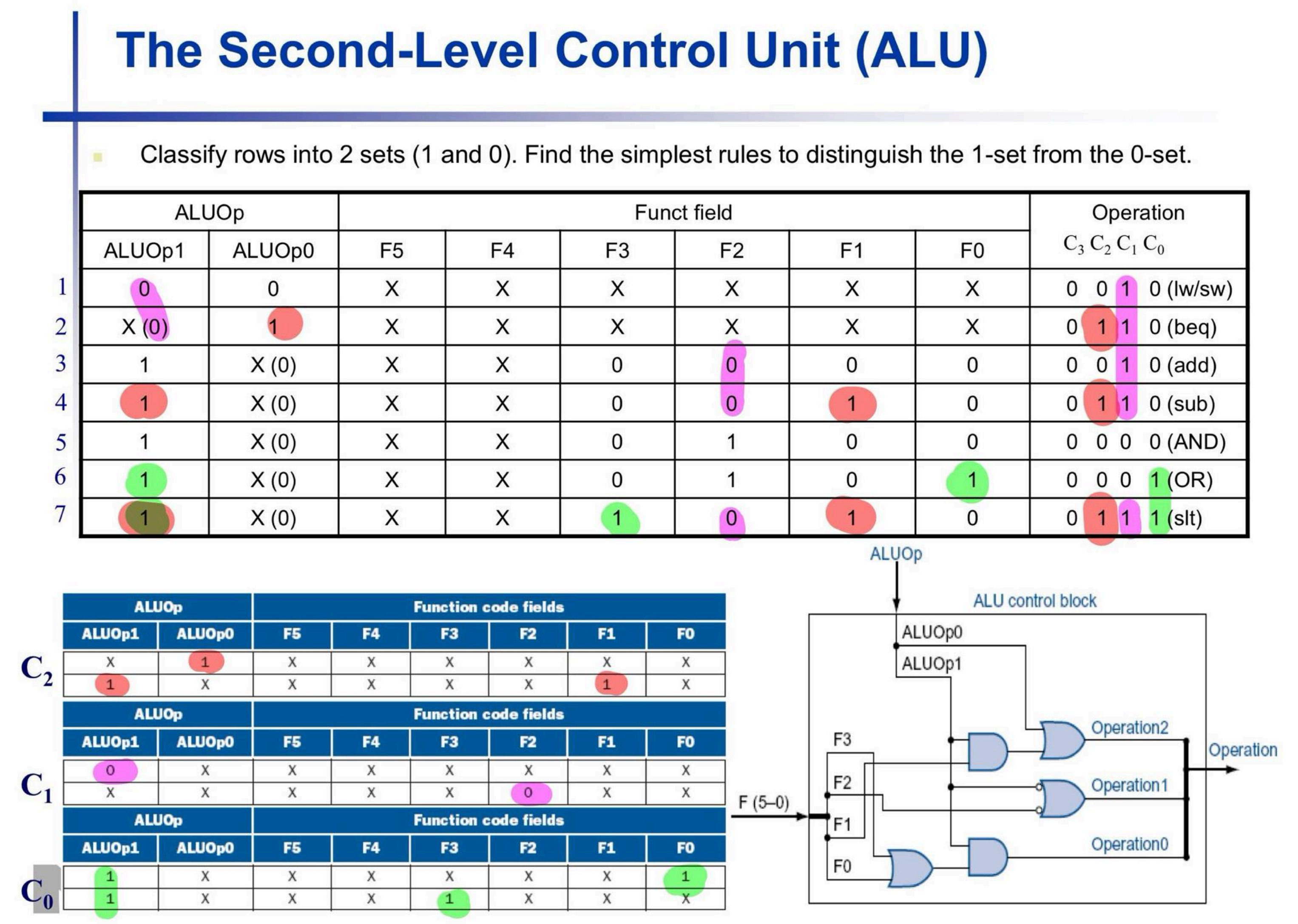
1. Architecture Diagrams

Show your ALU control (if you have one), main control and single-cycle processor design by "Schematic" tool in Vivado, or draw them by yourself. And briefly explain them.

1. alu_control



Design: follow by the textbook



The Second-Level Control Unit (ALU)

- Classify rows into 2 sets (1 and 0). Find the simplest rules to distinguish the 1-set from the 0-set.

	ALUOp		Funct field						Operation
	ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	C_3 C_2 C_1 C_0
1	0	0	X	X	X	X	X	X	0 0 1 0 (lw/sw)
2	X (0)	1	X	X	X	X	X	X	0 1 1 0 (beq)
3	1	X (0)	X	X	0	0	0	0	0 0 1 0 (add)
4	1	X (0)	X	X	0	0	1	0	0 1 1 0 (sub)
5	1	X (0)	X	X	0	1	0	0	0 0 0 0 (AND)
6	1	X (0)	X	X	0	1	0	1	0 0 0 1 (OR)
7	1	X (0)	X	X	1	0	1	0	0 1 1 1 (slt)

C_2

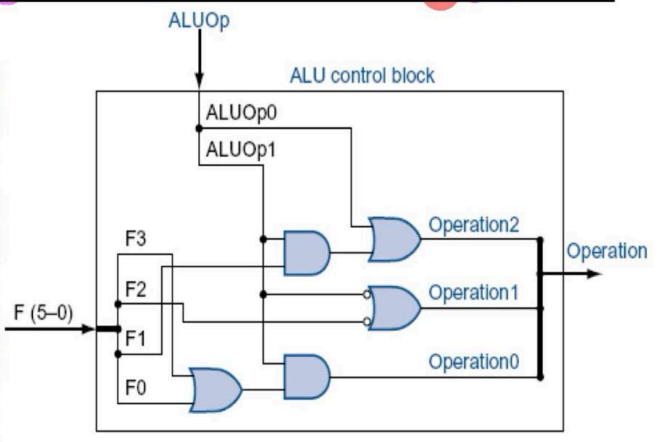
ALUOp		Function code fields					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
X	1	X	X	X	X	X	X
1	X	X	X	X	X	1	X

C_1

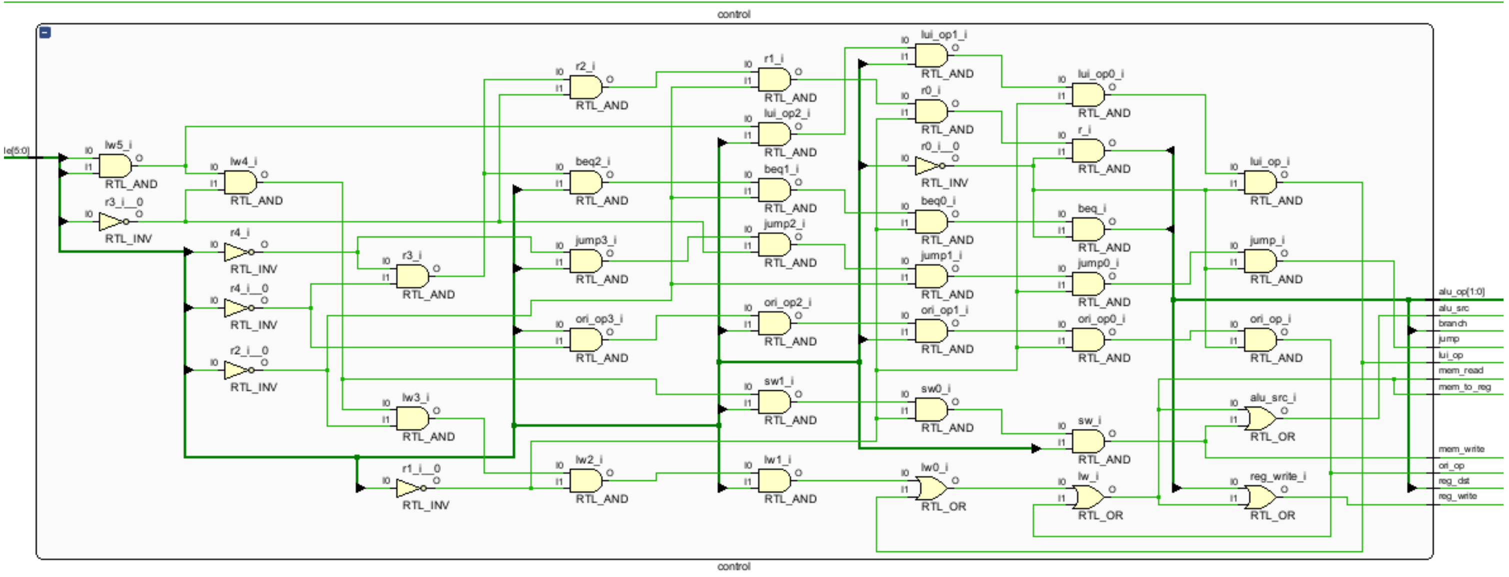
ALUOp		Function code fields					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
0	X	X	X	X	X	X	X
X	X	X	X	X	0	X	X

C_0

ALUOp		Function code fields					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
1	X	X	X	X	X	X	1
1	X	X	X	1	X	X	X

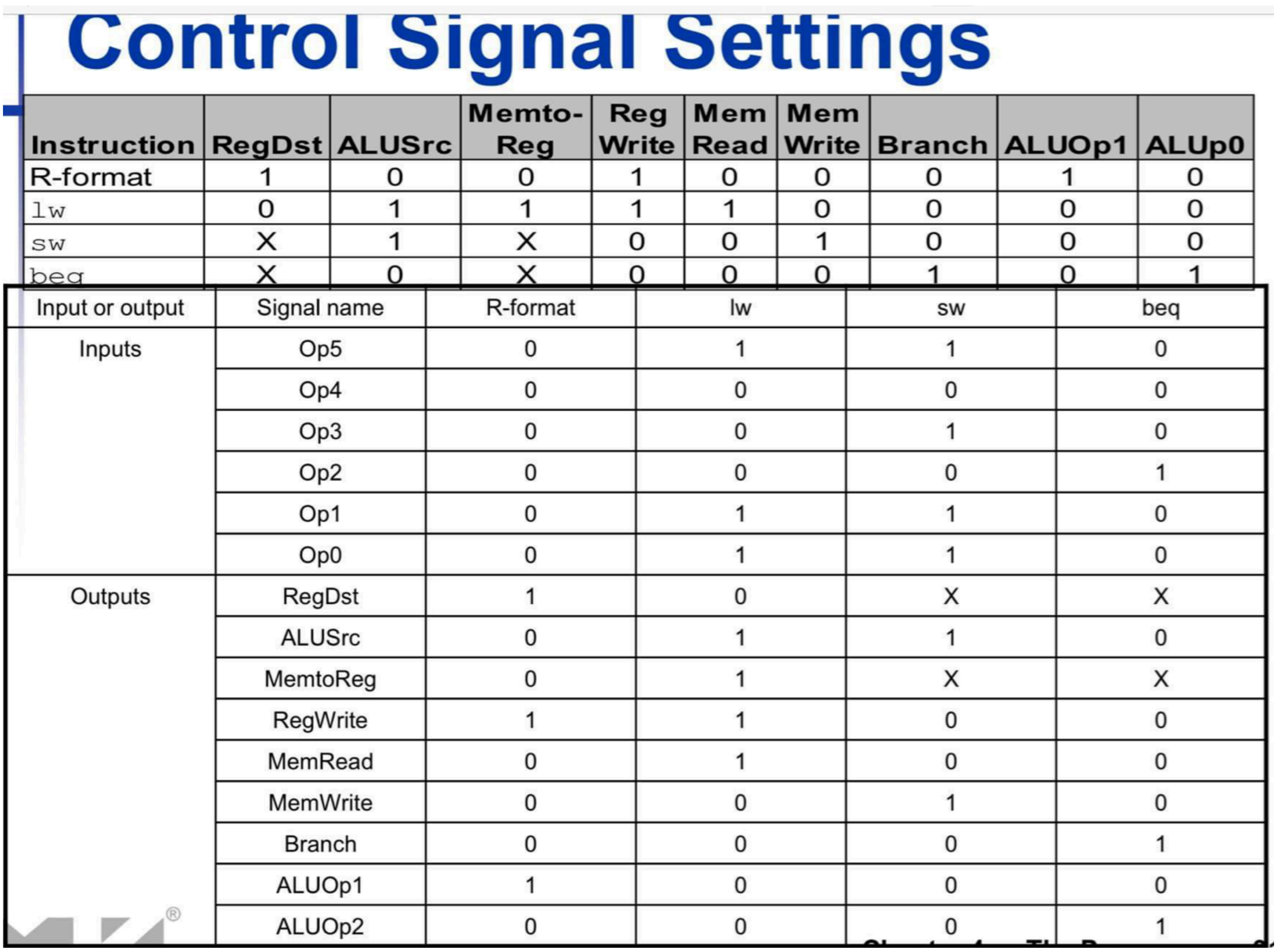


2. Main control



Design:

1. follow the textbook, use combinational logic and mux to assign the output.
2. put function code into control.v to ensure nop instruction don't do any execution;
3. The description mentions that the `ori` and `lui` instructions only use the regWrite operation. These instructions require `regDist` to be set to 0, which automatically occurs if the opcode is not in the R-format.

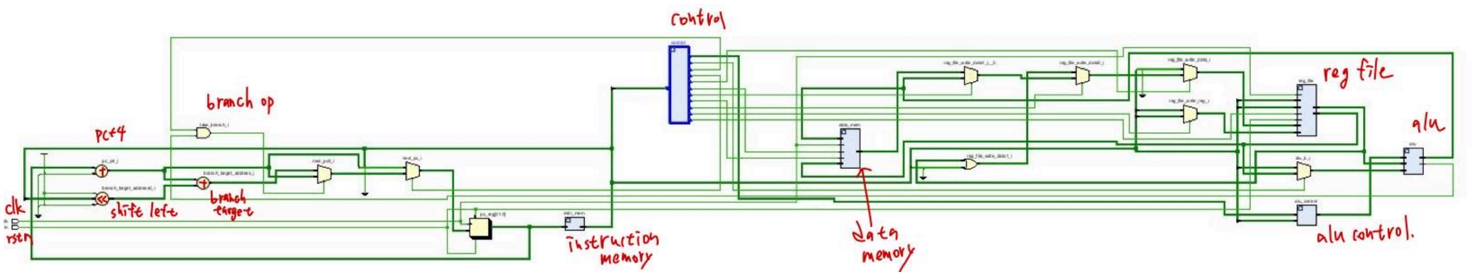


Control Signal Settings

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

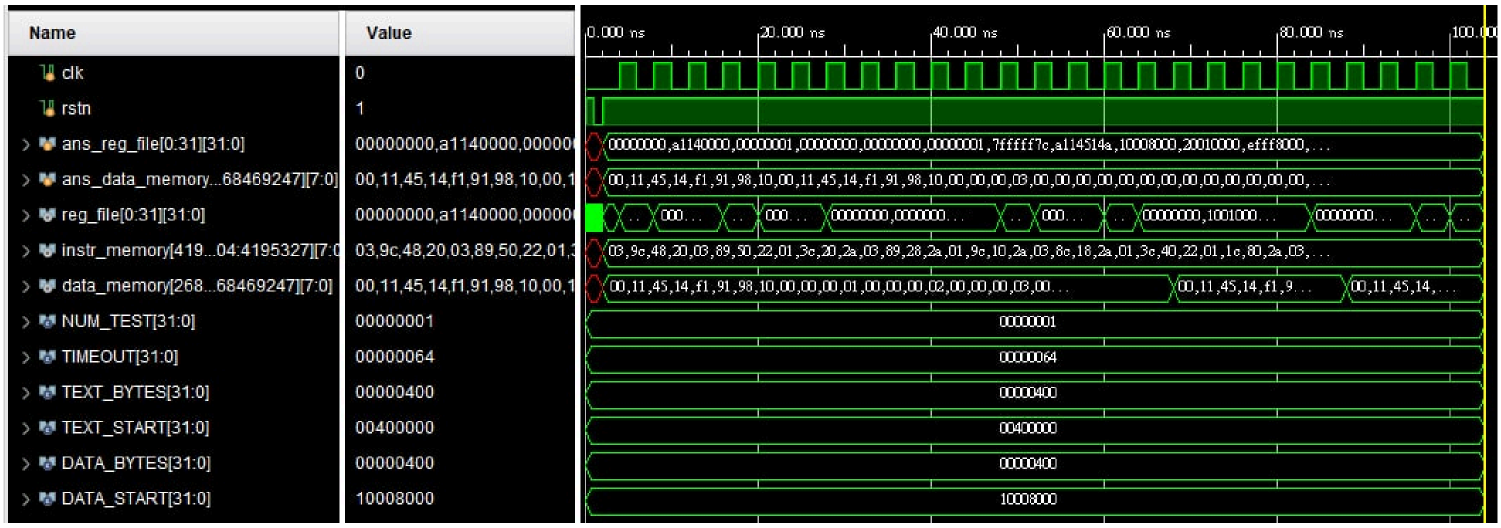
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp2	0	0	0	1

3. single cycle.



2. Experiment result

1. waveform



```
Name                               Value
clk                                0
rstn                               1
ans_reg_file[0:31][31:0]           00000000,a1140000,00000
ans_data_memory...68469247][7:0]   00,11,45,14,f1,91,98,10,00,1
reg_file[0:31][31:0]               00000000,a1140000,00000
instr_memory[419...04:4195327][7:0 03,9c,48,20,03,89,50,22,01,3
data_memory[268...68469247][7:0]   00,11,45,14,f1,91,98,10,00,1
NUM_TEST[31:0]                     00000001
TIMEOUT[31:0]                      00000064
TEXT_BYTES[31:0]                   00000400
TEXT_START[31:0]                   00400000
DATA_BYTES[31:0]                   00000400
DATA_START[31:0]                   10008000
```

```
0.000 ns   20.000 ns   40.000 ns   60.000 ns   80.000 ns   100.000
00000000,a1140000,00000001,00000000,00000000,00000001,7fffff7c,a114514a,10008000,20010000,efff8000,...
00,11,45,14,f1,91,98,10,00,11,45,14,f1,91,98,10,00,00,00,03,00,00,00,00,00,00,00,00,00,00,00,00,00,...
.. 000... .. 000... 00000000,0000000... .. 000... .. 00000000,1001000... 00000000... .. ..
03,9c,48,20,03,89,50,22,01,3c,20,2a,03,89,28,2a,01,9c,10,2a,03,8c,18,2a,01,3c,40,22,01,1c,80,2a,03,...
00,11,45,14,f1,91,98,10,00,00,00,01,00,00,00,02,00,00,00,03,00... 00,11,45,14,f1,9... 00,11,45,14,...
00000001
00000064
00000400
00400000
00000400
10008000
```

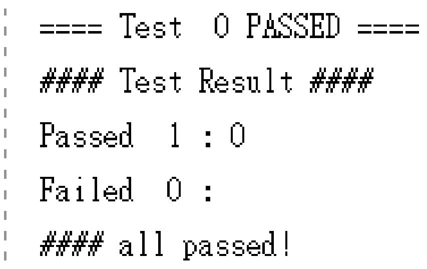
2. Other testcase

```
        .data   0x10008000      # start of Dynamic Data (pointed by $gp)
hun:    .word   0x00114514      # 0($gp)
hah:    .word   0xf1919810
        .word   0x1
        .word   0x2
        .word   0x3             # 16($gp)

        .text   0x00400000      # start of Text (pointed by PC),
                                # Be careful there might be some other ins
                                # Recommend at least 9 instructions to cov
main:   li      $t0, 0x12345678
        sub     $t1, $gp, $t0
        slt     $a0, $t1, $gp
        add     $t2, $t1, $gp   # $t0 = $gp
        nop                     # test NOP
        and     $t4, $t1, $t2   # test AND
        or      $t5, $t1, $t2   # test OR
        nop                     # test NOP
        lw      $t5, hun     # test LW
        sw      $t5, 8($gp)     # test SW
        nop                     # test NOP
        j       end             # [btg] should jump
end:    lw      $t5, hah        # [end]
        sw      $t5, 12($gp)
        nop                     # test NOP
        li      $a3, 0xa114514a # test li (lui, ori)
        slt     $t6, $t2, $t3
        beq     $zero, $t6, func
func:   li      $t6, 0x18476502
```

測試大部分的operation，順便多放幾個nop看看有沒有忽略。

result:



```
==== Test  0 PASSED ====
#### Test Result ####
Passed  1 : 0
Failed  0 :
#### all passed!
```

3. Answer the following questions

1. When does write to register/memory happen during the clock cycle? How about read?

In a clock cycle, writes to registers or memory typically occur at the rising or falling edge of the clock signal, depending on the system design. Reads, on the other hand, can be performed at any time during the clock cycle, as long as the data is stable and available when needed.

2. Translate the "branch" pseudo instructions (blt , bgt , ble , bge) in the Green Card into
real instructions. Only at register can be modified, and other common registers should not be modified.

blt:

slt $at, $t0, $t1

bne $at, $zero, label

bgt:

slt $at, $t1, $t0

bne $at, $zero, label

ble:

slt $at, $t1, $t0

beq $at, $zero, label

bge:

slt $at, $t0, $t1

beq $at, $zero, label

---

3. Give a single beq assembly instruction that causes infinite loop. (consider that there's no delay slot)

```
loop: beq $zero, $zero, loop
```

---

4. The j instruction can only jump to instructions within the "block" defined by "(PC+4)[31:28]". Design a method to allow j to jump to the next block (block number + 1) using another j.

```
j func
nop


func:
lui $at, upper_bits
ori $at, $at, lower_bits
jr $at
nop
```

---

5. Why a Single-Cycle implementation is not used today?5.

Single-cycle processor implementations are not favored in modern computing primarily due to performance and efficiency concerns:

1. **Clock Speed Limitation**: In a single-cycle design, all operations of an instruction (fetching, decoding, executing, and writing back) must complete within one clock cycle. To ensure the slowest operations can be completed, the clock speed must be reduced, which limits the overall performance of the processor.
2. **Inefficient Resource Utilization**: Since all stages of instruction processing are bound to a single cycle, faster stages must wait for the slower ones to complete. This mismatch leads to poor utilization of the processor's capabilities, as quicker operations are delayed by the inherent limitations of slower ones.

These constraints make single-cycle processors unsuitable for high-performance applications, leading to the adoption of more efficient architectures like pipelining, where different stages of multiple instructions are processed simultaneously to enhance throughput and maximize hardware utilization.

---

4. Problems Encountered & Solution

I discovered that the system is more complex and challenging to debug, so I've started taking notes and labeling diagrams in the textbook to ensure that all paths are connected and clear. This helps me ensure a better understanding and tracking of the entire process.

---

5. feedback

It’s exciting to really understand the CPU architecture from the textbook diagrams, rather than just trying to imagine it. Being able to see and annotate the figures directly helps clarify the handwriting issues and makes for excellent homework practice.