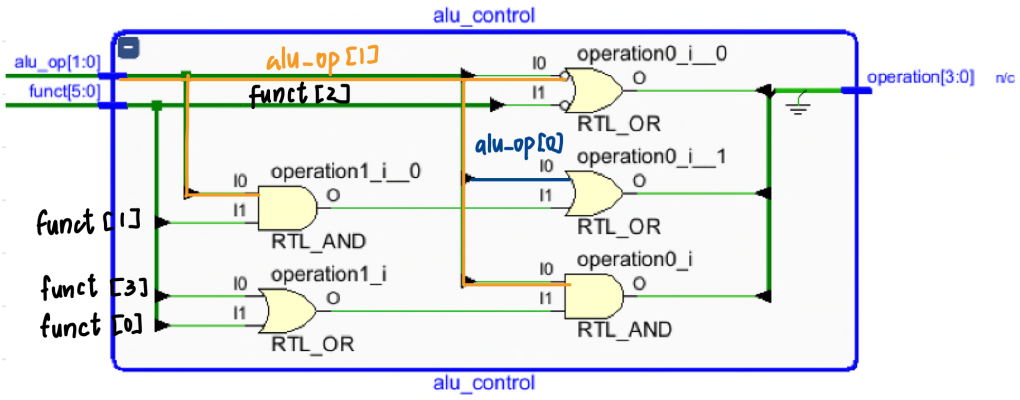
Computer Organization

Lab 2 single cycle processor

111550168 陳奕

1. Architecture Diagrams

(1) ALU control

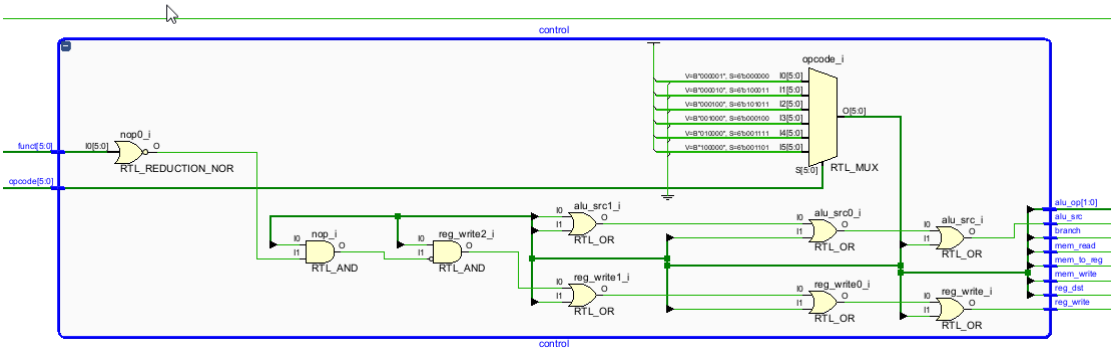


Design process :

Observe the bit relationship between alu_op, funct and alu_control_operation, use combinational logic to assign the output.

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	$C_3\ C_2\ C_1\ C_0$
0	0	X	X	X	X	X	X	0 0 1 0 (lw/sw)
X (0)	1	X	X	X	X	X	X	0 1 1 0 (beq)
1	X (0)	X	X	0	0	0	0	0 0 1 0 (add)
1	X (0)	X	X	0	0	1	0	0 1 1 0 (sub)
1	X (0)	X	X	0	1	0	0	0 0 0 0 (AND)
1	X (0)	X	X	0	1	0	1	0 0 0 1 (OR)
1	X (0)	X	X	1	0	1	0	0 1 1 1 (slt)

(2) Main control

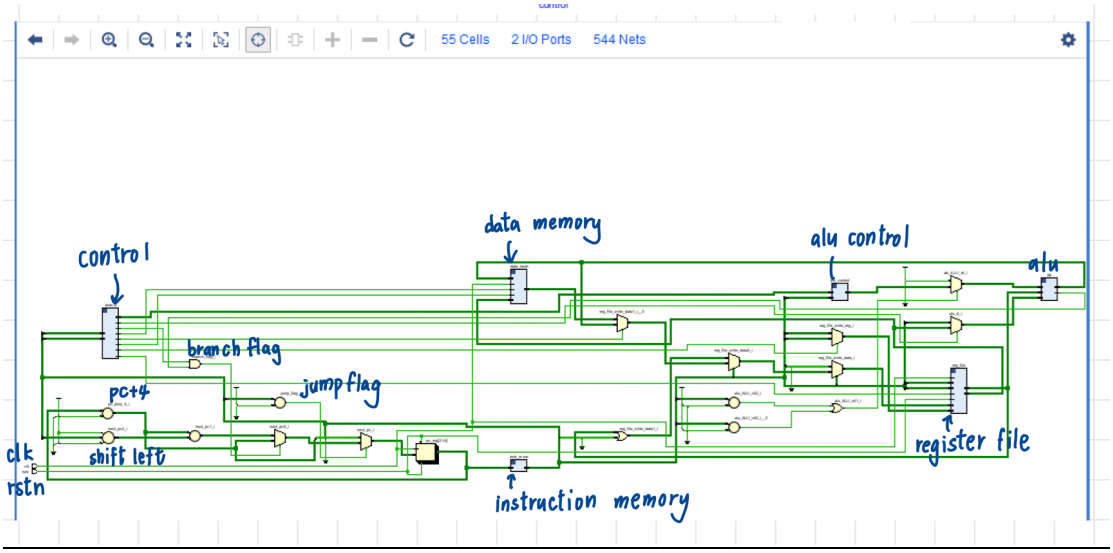


Input/output	signal name	R-format	lw	sw	beq	lui	ori
input	op5	0	1	1	0	0	0
	op4	0	0	0	0	0	0
	op3	0	0	1	0	1	1
	op2	0	0	0	1	1	1
	op1	0	1	1	0	1	0
	op0	0	1	1	0	1	1
output	RegDst	1	0	X	X	X	X
	ALUsrc	0	1	1	0	1	1
	MemtoReg	0	1	X	X	X	X
	Regwrite	1	1	0	0	1	1
	Memread	0	1	0	0	X	X
	Memwrite	0	0	1	0	X	X
	Branch	0	0	0	1	X	X
	ALUop1	1	0	0	0	X	X
	ALUop2	0	0	0	1	X	X

Design process :

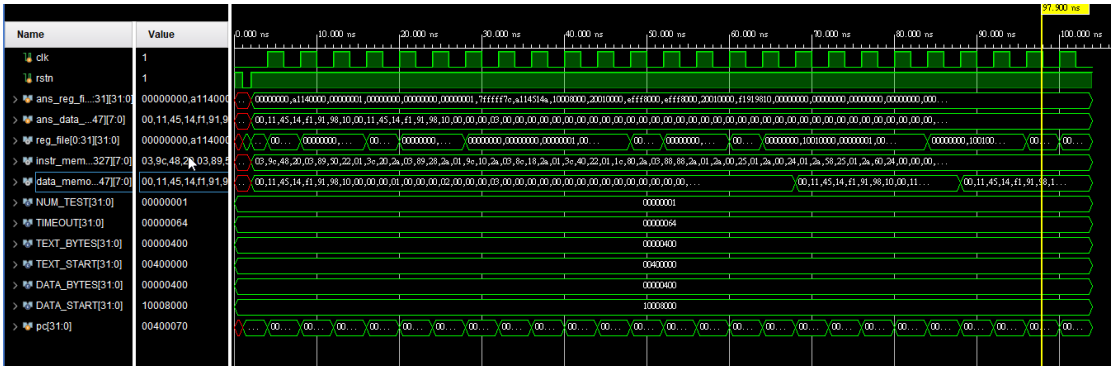
1. Observe the bit relationship between opcode and control signal, use combinational logic and mux to assign the output.
2. To handle “nop” instruction, I port function code into the control module to ensure that nop instruction don’t do any execution.
3. As for “ori” and “lui” instruction, they only use ALUsrc and Regwrite signal.

(3) Single-cycle processor



2. Experimental result

(1) Waveform



(2) other testcase(explain)

a. positive and negative offset : store negative offset of an array will find bad data address

code (added after 0.text.mem):

```
3c 01 00 98 //[00400074]  lui $1, 152                  ; 16: addi $t0, $zero, 10008000# Load base address of array
34 28 b5 c0 //[00400078]  ori $8, $1, -19008
8d 09 00 04 //[0040007c]  lw $9, 4($8)                 ; 17: lw $t1, 4($t0)# Load array[1] with positive offset
ad 09 ff fc //[00400080]  sw $9, -4($8)                 ; 18: sw $t1, -4($t0)# Store into array[-1] with negative offset (undefined behavior)
```

Result : pay attention on found value since I didn't modify the answer of this testcase

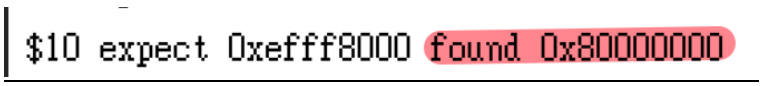
```
$9 expect 0x20010000 found 0xxxxxxxxx
```

b. add overflow : overflow occur when I add 1 to max positive int

code:

```
3c 01 7f ff //[00400008]  lui $1, 0x7FFF                ;Load max positive int
34 28 ff ff //[0040000c]  ori $8, $1, 0xFFFF
3c 01 00 00 //[00400010]  lui $2, $zero                 ; 36: li      $a3, 0xa114514a # test li (lui, ori)
34 49 00 01 //[00400014]  ori $9, $2, 0x1
01 28 50 20 //[00400018]  add $10, $9, $8              ; Overflow should occur here if detected
```

Result: adding positive int shouldn't output a negative number.(pay attention on found value since I didn't modify the answer of this testcase)



3. Questions

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

Write to a register or memory happens on the rising edge of the clock cycle; read operations occur before the write operations during the clock cycle, usually at the negative edge to ensure that the data is available and stable by the time the execution stage needs it.

(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 there is 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 using another j.

```
j med          # First jump to intermediate location within the current block
nop            # Delay slot

med:
lui $at, upper_bits          # Load upper bits of the target address into $at
ori $at, $at, lower_bits     # Combine with lower bits
jr $at                       # Jump to the target address
nop                          # Delay slot
```

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

This is due to several limitations:

1. Limited Clock Speed: The cycle must be long enough to accommodate the slowest operation, significantly reducing the overall processor speed.
2. Poor Resource Utilization: Processor components like the ALU and memory are not fully utilized as they remain idle for a significant part of the cycle.

Instead, modern processors use multi-cycle and pipelined architectures which allow for more efficient operation, better scalability, and higher clock speeds, addressing the shortcomings of single-cycle implementations.

4. Feedback

After finishing this lab, I understand the complete cycle of fetching, decoding, and executing instructions within a single clock cycle.
In addition, using theoretical principles from textbook in a practical hardware project makes me adept with the MIPS architecture, I understand the encoding and execution of its instructions. Finally, I can identify and resolve hardware and synchronization issues during the building and testing phases.
Although converting higher level language to assembly and binary instruction is quite a complicate job for me, understanding this lab is interesting.