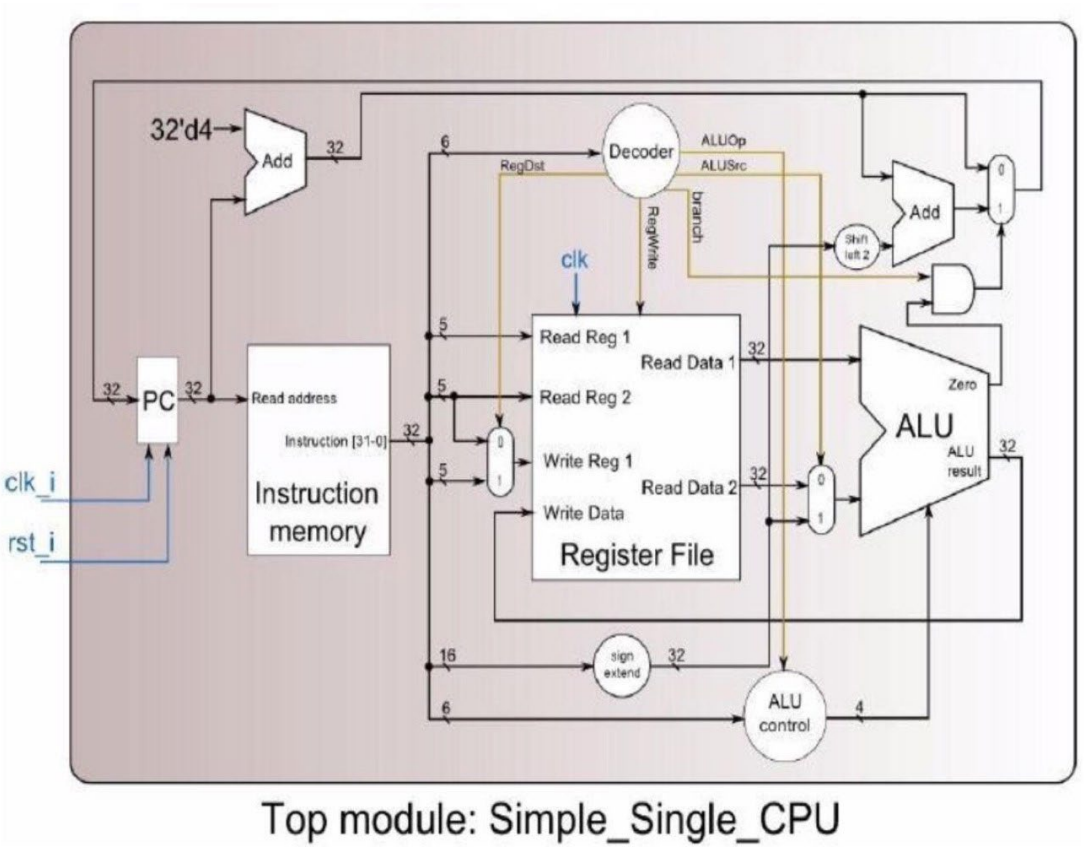
Computer Organization Lab2

Name: 陳子祈

ID: 0819823

Architecture diagrams:



Top module: Simple_Single_CPU

R type						
Instruction set	**Op code**	**rs**	**rt**	**rd**	**shamt**	**funct**
Instr location	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]
add $rd,$rs,$rt	000000 (0)				00000	100000 (32)
sub $rd,$rs,$rt	000000 (0)				00000	100010 (34)
and $rd,$rs,$rt	000000 (0)				00000	100100 (36)
or $rd,$rs,$rt	000000 (0)				00000	100101 (37)
slt $rd,$rs,$rt	000000 (0)				00000	101010 (42)
I type						
Instruction set	**Op code**	**rs**	**rt**	**immediate**		

Instr location	[31:26]	[25:21]	[20:16]	[15:0]
addi $rt,$rs,imm	001000(8)			
slti $rt,$rs,imm	001010(10)			
beq $rt,$rs,imm	000100(4)			

Hardware module analysis:

(explain how the design work and its pros and cons)

Single cycle MIPS CPU 每過一段固定的 cycle time 就做 PC 與暫存器的運算。PC 的運算只有分為 Sequential 的運算與 beq 跳行的運算。暫存器的運算流程如下:

1. IF: Instruction fetch from memory 從 memory 請求指令
2. ID: Instruction decode & register read 解碼指令、產生控制訊號，並把暫存器讀出來
3. EX: Execution operation ~~or calculate address~~ 執行指令，這次實驗沒有使用到 lw、sw 存取記憶體的指令，因此不需計算地址

~~(原 4) MEM: Access memory operand~~ 同上，不需存取記憶體的指令

4. WB: Write result back to register 將 ALU 計算出來的結果寫回去 Write register

Single cycle MIPS CPU 優點就是不會發生 hazards; 缺點就是以 Longest delay 的指令當作 clock period，導致執行大部分指令都有很多 CPU idle 的時間。

各module的description:

1) Decoder

功能：透過6bit的instruction operation code 決定各種控制訊號。

Port description：

instr_op_i ：6bit input instruction operation code

RegWrite_o ：1bit output RegFile Write or not

ALU_op_o ：2bit output for ALU_Ctrl to determine operation type

ALUSrc_o ：1bit output determine ALU source

RegDst_o ：1 bit output determine Read reg2 is rt or rd

Branch_o ：1bit output the instruction is branch type or not

Decoder	RegDst	RegWrite	ALU_op	ALUSrc	Branch
R type					
add	1	1	11 (3)	0	0
sub	1	1	11 (3)	0	0
and	1	1	11 (3)	0	0
or	1	1	11 (3)	0	0
slt	1	1	11 (3)	0	0

I type					
addi	0	1	00 (0)	1	0
slti	0	1	01 (1)	1	0
beq	0	0 (X)	10 (2)	0	1

2) ALU_Ctrl

功能：將ALU_op及function code轉成ALU所需的ALUCtrl，決定ALU的動作及控制其他MUX、Shifter。

Port description：

funct_i ：6bit input function code

ALUOp_i ：2bit input for ALU_Ctrl to determine operation type

ALUCtrl_o ：4bit output to ALU control

ALU_Ctrl	ALUCtrl
R type	
add	0010
sub	0110
and	0000
or	0001
slt	0111

ALU_Ctrl	ALUCtrl
I type	
addi	0010
slti	0111
beq	0110

3) ALU

功能：32bit運算邏輯單位，參考課本附錄程式，可做add、sub、or、and、slt。

Port description：

src1_i ：32bit input data

src2_i ：32bit input data

ctrl_i ：4bit ALU_Control

result_o ：32bit result for ALU

zero_o ：1 bit when the output is 0, zero must be set

4) Adder

功能：輸入兩個data輸出其相加結果。

Port description：

src1_i ：32bit input data

src2_i ：32bit input data

sum_o ：32bit output sum

5) Sign_Extend

功能：將輸入data做Sign Extend，data_i複製到data_o低16位，data_i最高位bit複製到data_o高16位。

Port description：

data_i ：16bit input data

data_o ：32bit output data

6) Shift_Left_Two_32

功能：將input data左移兩個bit。

Port description：

data_i ：32bit input data

data_o ：32bit output data

7) MUX_2to1

功能：如果 select_i = 0 則輸出 data0_i；select_i = 1 則輸出data1_i。

Port description：

data0_i ：32bit input data

data1_i ：32bit input data

select_i ：1bit select for MUX

data_o ：32bit output data

8) Simple_Single_CPU

功能：將上述所提到之 Module 依照 Architecture diagram 的附圖做連接，完成 Simplified Single-cycle CPU。

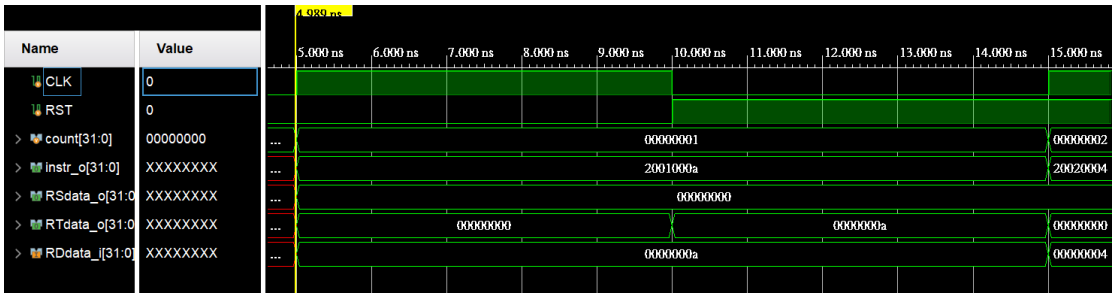
Finished part:

Case1

Instruction 1: `addi $r1,$r0,10`

Instruction set	**Op code**	**rs**	**rt**	**immediate**
Instr location	[31:26]	[25:21]	[20:16]	[15:0]
`addi $r1,$r0,10`	`001000(8)`	`00000`	`00001`	`0000000000001010`

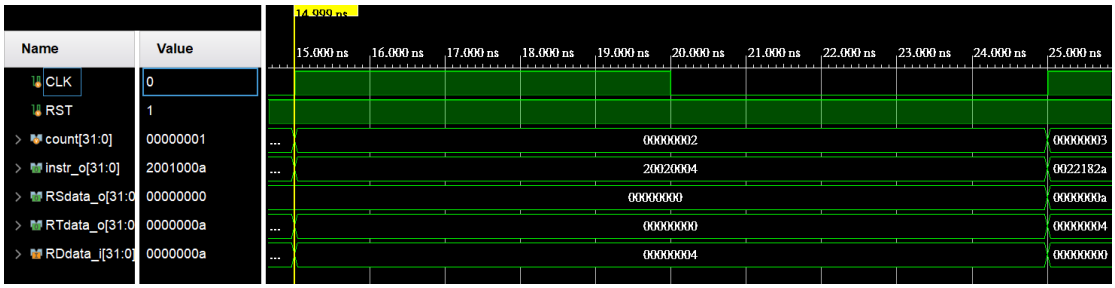
Description: r0 = 0, r1 = r0+10 = 10



Instruction 2: `addi $r2,$r0,4`

Instruction set	Op code	rs	rt	immediate
Instr location	[31:26]	[25:21]	[20:16]	[15:0]
`addi $r2,$r0,4`	`001000(8)`	`00000`	`00010`	`0000000000000100`

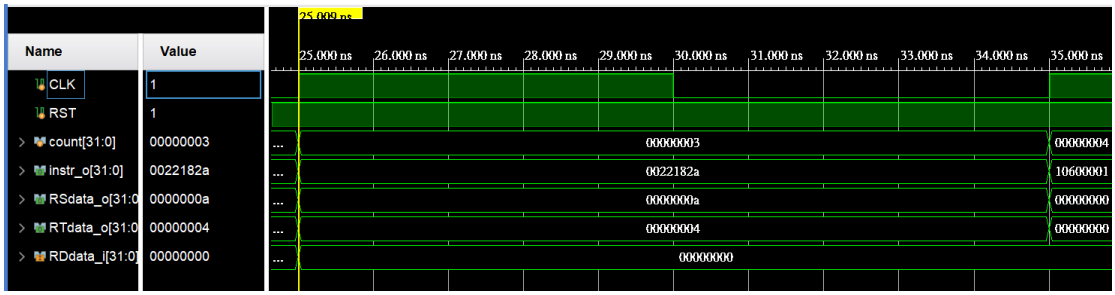
Description: r0 = 0, r2 = r0+4 = 4



Instruction 3: `slt $r3,$r1,$r2`

Instruction set	Op code	rs	rt	rd	shamt	funct
Instr location	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]
`slt $r3,$r1,$r2`	`000000(0)`	`00001`	`00010`	`00011`	`00000`	`101010(42)`

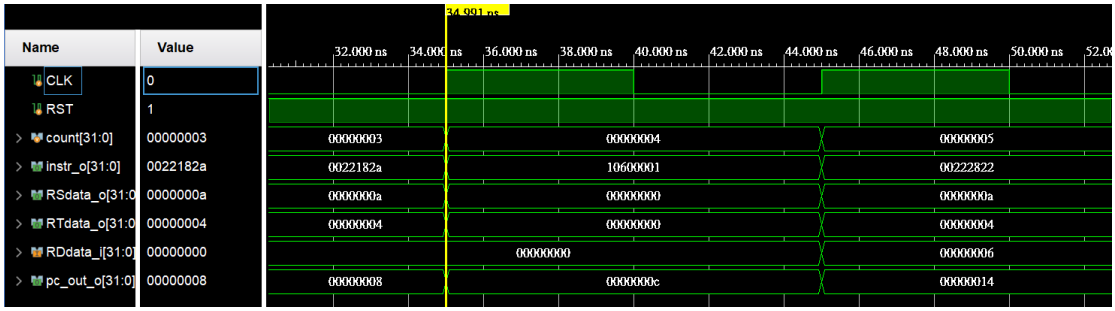
Description: r1 = 10, r2 = 4; Since r1-r2=6>0, r3 = 0



Instruction 4: `beq $r3,$r0,1`

Instruction set	Op code	rs	rt	immediate
Instr location	[31:26]	[25:21]	[20:16]	[15:0]
`beq $r3,$r0,1`	`000100(4)`	`00011`	`00000`	`0000000000000001`

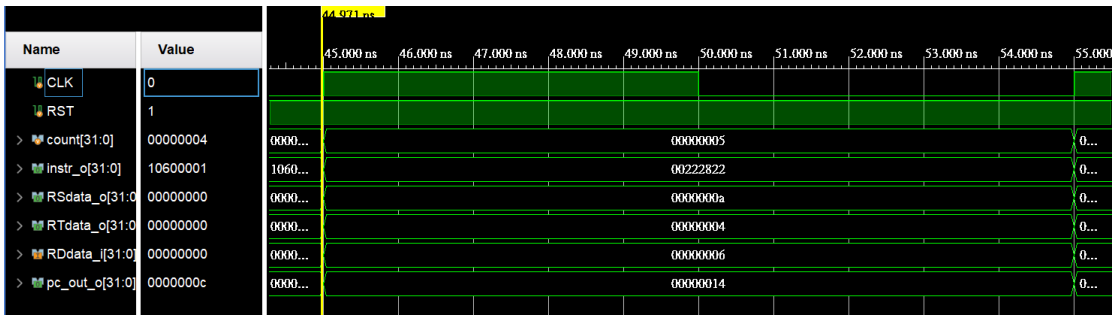
Description: r0 = r3, PC = PC + 4 + 4 = PC + 8 = $0000000c_{16} + 8_{16} = 00000014_{16}$



Instruction 5: `sub $r5,$r1,$r2` (跳過 `add $r4,$r1,$r2`)

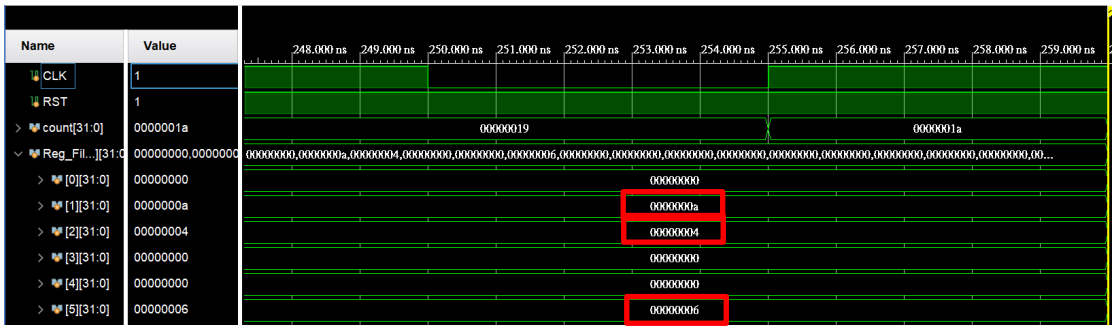
Instruction set	Op code	rs	rt	rd	shamt	funct
Instr location	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]
`sub $r5,$r1,$r2`	`000000(0)`	`00001`	`00010`	`00011`	`00000`	`101010(42)`

Description: r1 = 10, r2 = 4; r5 = r1 – r2 = 6



最後各暫存器結果為:

r0 = 0 ; r1 = 10 ; r2 = 4 ; r3 = 0 ; r4 = 0 ; r5 = 6

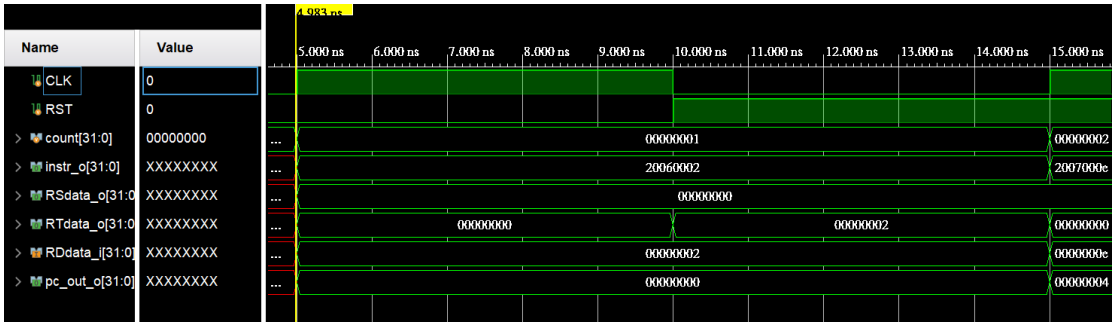


Case2

Instruction 1: `addi $r6,$r0,2`

Instruction set	Op code	rs	rt	immediate
Instr location	[31:26]	[25:21]	[20:16]	[15:0]
`addi $r6,$r0,2`	`001000(8)`	`00000`	`00110`	`0000000000000010`

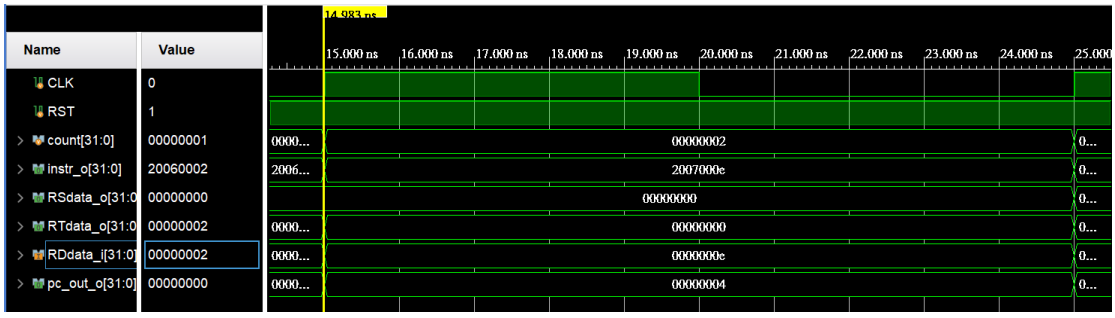
Description: r0 = 0, r6 = r0+2 = 2



Instruction 2: `addi $r7,$r0,14`

Instruction set	Op code	rs	rt	immediate
Instr location	[31:26]	[25:21]	[20:16]	[15:0]
`addi $r7,$r0,14`	`001000(8)`	`00000`	`00111`	`0000000000001110`

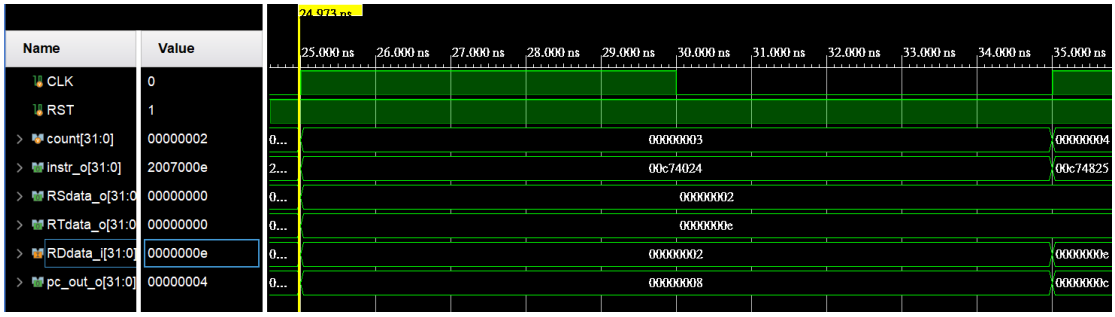
Description: r0 = 0, r7 = r0+14 = 14



Instruction 3: `and $r8,$r6,$r7`

Instruction set	Op code	rs	rt	rd	shamt	funct
Instr location	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]
`and $r8,$r6,$r7`	`000000(0)`	`00110`	`00111`	`01000`	`01000`	`100100(36)`

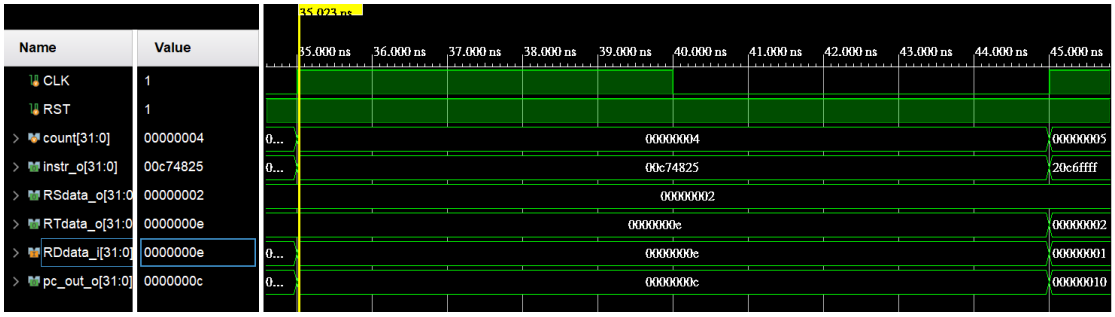
Description: r8 = r6 & r7 = 2 & 14 = 00010_2 & 01110_2 = 00010_2 = 2_{10}



Instruction 4: `or $r9,$r6,$r7`

Instruction set	Op code	rs	rt	rd	shamt	funct
Instr location	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]
`and $r9,$r6,$r7`	`000000(0)`	`00110`	`00111`	`01001`	`00000`	`100101(37)`

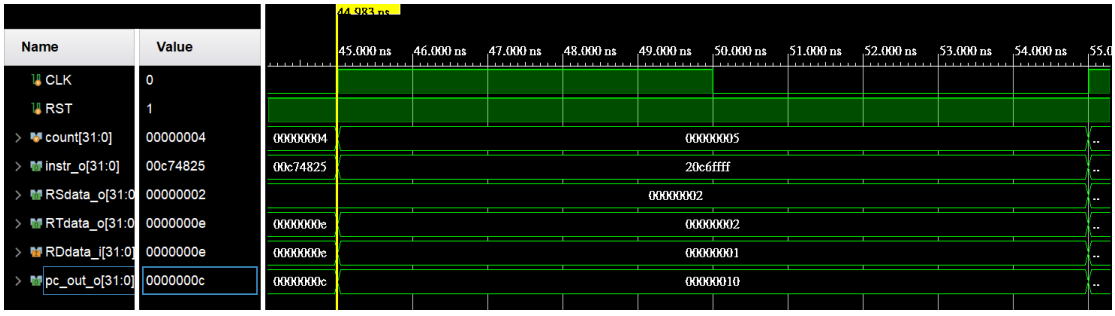
Description: r9 = r6 | r7 = 2 | 14 = 00010_2 | 01110_2 = 01110_2 = 14_{10}



Instruction 5: `addi $r6,$r6,-1`

Instruction set	Op code	rs	rt	immediate
Instr location	[31:26]	[25:21]	[20:16]	[15:0]
`addi $r6,$r6,-1`	`001000(8)`	`00110`	`00110`	`1111111111111111`

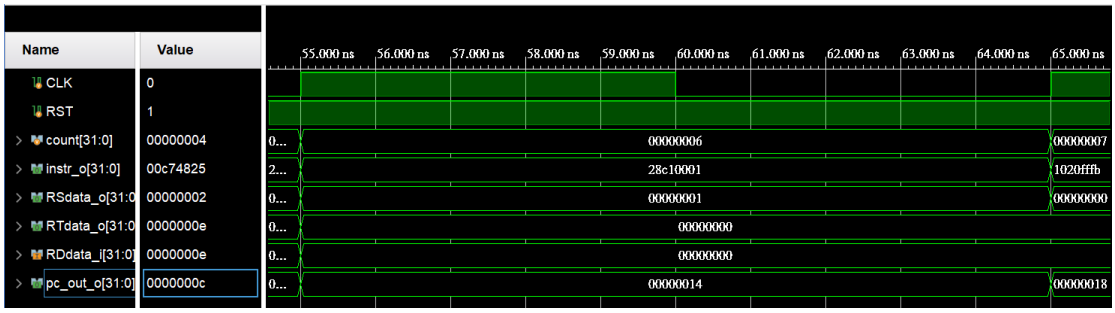
Description: r6 = r6-1 = 2-1 = 1



Instruction 6: `slti $r1,$r6,1`

Instruction set	Op code	rs	rt	immediate
Instr location	[31:26]	[25:21]	[20:16]	[15:0]
`slti $r1,$r6,1`	`001010(10)`	`00110`	`00001`	`0000000000000001`

Description: Since r6>=1, r1 = 0

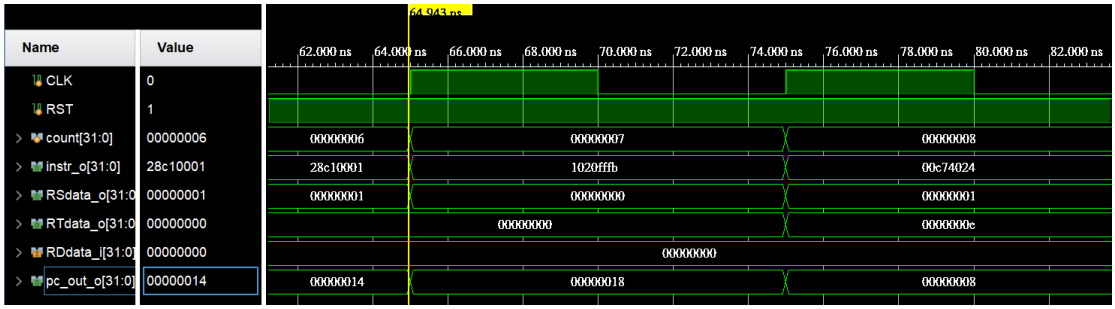


Instruction 7: `beq $r1,$r0,-5`

Instruction set	Op code	rs	rt	immediate
Instr location	[31:26]	[25:21]	[20:16]	[15:0]
`beq $r1,$r0,-5`	`000100(4)`	`00000`	`00001`	`1111111111111011`

Description: Since r1=r0,

PC = PC + 4 – 5*4 = PC – 16 = $00000018_{16} - 00000010_{16} = 00000008_{16}$

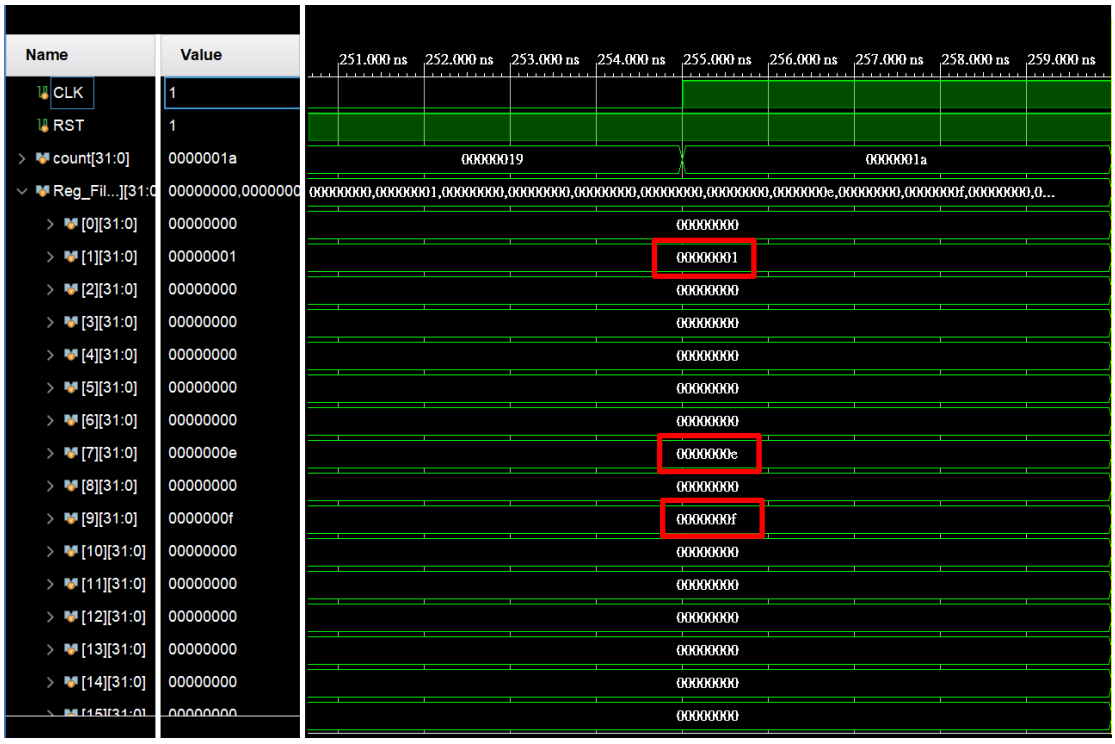


程式會回到 Instruction 3。為求精簡，不一一擷取示波圖，只看執行接下來所有的指令以及最後的結果。

Instruction	Desciption
`and $r8,$r6,$r7`	r8 = r6 & r7 = 1 & 14 = 00001_2 & 01110_2 = 00000_2 = 010
`or $r9,$r6,$r7`	r9 = r6 | r7 = 1 | 14 = 00001_2 & 01110_2 = 01111_2 = 15_{10}
`addi $r6,$r6,-1`	r6 = r6-1 = 1-1 = 0
`slti $r1,$r6,1`	Since r6<1, r1 = 1
`beq $r1,$r0,-5`	Since r1!=r0, the program terminates.

最後各暫存器結果為:

r0 = 0 ; r1 = 1 ; r6 = 0 ; r7 = 14 ; r8 = 0 ; r9 = 15



Problems you met and solutions:

P1 講義上沒有寫 addi、slti 的 ALU_op 要如何設定。

Q1 我自己重新設定 addi、slti、beq、R_format 的 ALU_op。

P2 已經照著講義寫好所有的模組，並接好線了，但是 register 最終都還是 0，不知道錯在哪裡。

Q2 後來發現 addi 及 slti 的 Regdest 寫錯，改成 0 就正常了。

Summary:

這次的 Lab 比第一次還要複雜很多，不過有了前幾次寫 code 的經驗之後，認真看完講義的內容，就比之前還知道要怎麼寫 code 建模組與接線。我覺得這次的 Lab 比較像是複習講義的內容，讓我更知道 single cycle MIPS CPU 是怎麼運作的。我這次還有學到怎麼利用示波圖 trace code 來除錯，學長有教我如果指令都有讀進來，接線也沒問題，那最有可能出錯的就是控制訊號，果然我的其中一個控制訊號不小心寫錯了，導致結果也跟著出錯，調整完控制訊號就成功了。希望以後還有類似這種 Lab 可以既複習上課內容又加強我寫 Verilog 的能力。