COMP30080 – Processor Design
Assignment 5 – Complete Processor

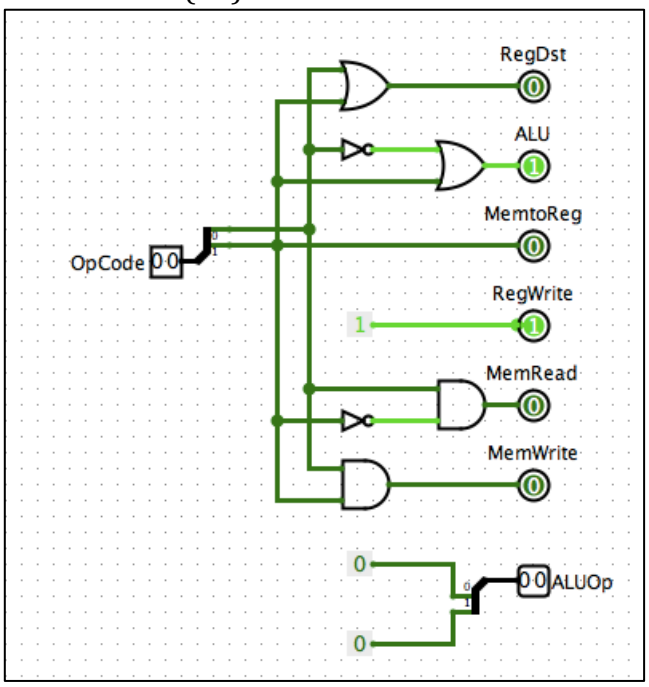
Team Name: Team Love N Stuff 2.0
Team Members:
4A 12363006 Sean Lawlor
4B 12386376 Ryan Kane
4C 11363776 Diarmuid Ryan

Question 1.
Write down the true table for the control circuit of the processor. The control circuit should take machine instructions as input and should output the necessary control signals to the Register File, ALU and Data RAM. Design a circuit which will implement the control circuit for the processor. Implement and test the circuit. Integrate the functional units developed in the previous assignments and Q1 into a single processor.

As a team we compiled the following truth table for the control circuit of our processor:

OpBit1	OPBit0	RegDst	ALU	MemtoReg	RegWrite	MemRead	MemWrite	ALUBit0	ALUBit1
0	0	0	1	0	1	0	0	0	0
0	1	1	0	0	1	1	0	0	0
1	0	1	1	1	1	0	0	0	0
1	1	1	1	1	1	0	1	0	0

resulting in the following control circuit, for ALUOp we have set to a constant because although ALU is capable of AND & subtract functionality all that is needed in our case is addition (00).



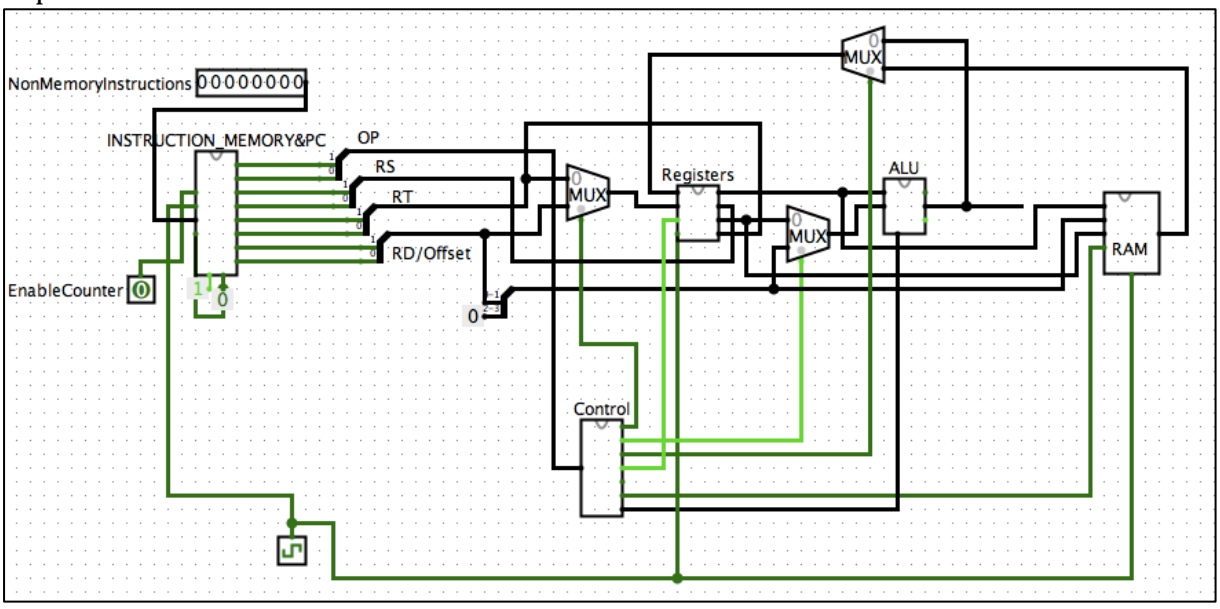
Our mini-MIPS processor would be made according to the following specifications:

- 4-bit processor
- 4 x 4-bit registers
- 4-bit Data RAM
- 8-bit Program RAM
- 4-bit ALU

The processor would follow the following Instruction Set Architecture:
lw rt, offset(rs) # load word from M[rs+offset] to R[rt]
sw rt, offset(rs) # store word from R[rt] to M[rs+offset]
addi rt, rs, const # add immediate R[rt] = R[rs] + const
add rd, rs, rt # add R[rd]=R[rs]+R[rt]

Whereby instructions would be in the format: {opcode; rs; rt; rd/offset/const} – with opcodes as follows:
addi : opcode = 00
add : opcode = 01
lw : opcode = 10
sw : opcode = 11

We began by building the following circuit for our processor which would implement the control circuit created from our truth table above:



We then tested the circuit by pre-loading the Instruction Memory with the following instructions (in hexadecimal format), the opcode bits of which would in turn be passed to the control unit to pass the necessary control signals to other components within our processor.

Sample Instructions pre-loaded		
Instruction (hexadecimal)	**Instruction (binary)**	**Intended function**
0x12	0001 0010	addi
0x9	0000 1001	addi
0x4b	0100 1011	add
0x14	0001 0100	addi
0xdc	1101 1100	sw
0xa4	10100100	lw

When tested our processor successfully read/write values into registers and memory as intended.

Question 2

Write an assembly program for the miniMIPS processor that will calculate and store the first 8 Fibonacci numbers in the data memory. Manually assemble the program. Load the machine code into Program RAM. Load a string and key into Data RAM. Execute the program by toggling the clock. Check the final results in Data RAM.

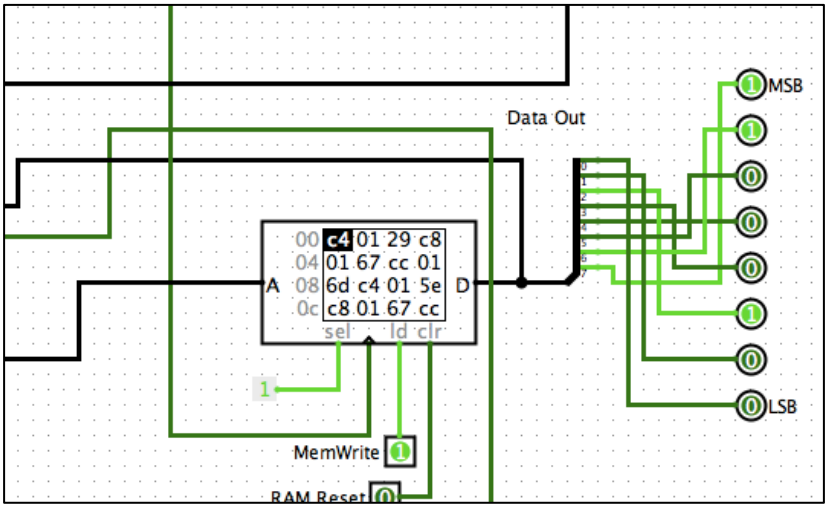
The following MIPS code was used as a guide in testing our processor to produce the Fibonacci numbers which could fit within 4 bits.

```
.data
# all registers start at 0

#$t0 - memory address
#$t1-3 used to calculate fibs
datain: .byte 0
#initialise first 2

.text
la $t0, datain
sb $t1, 0($t0) #save fib 0
addi $t0 $t0, 1 #next address

addi $t2, $t2, 1
sb $t2, 0($t0) #save fib 1
addi $t0 $t0, 1 #next address

######################

#calculate next

add $t3, $t2, $t1
sb $t3, 0($t0) #save fib 2
addi $t0 $t0, 1 #next address


add $t1, $t2, $t3
sb $t1, 0($t0) #save fib 3
addi $t0 $t0, 1 #next address


add $t2, $t1, $t3
sb $t2, 0($t0) #save fib 4
addi $t0 $t0, 1 #next address

#######
```

We loaded the following assembly language instructions into the Instruction Memory which would allow our processor to calculate and store the first 8 Fibonacci numbers into data memory.

Instruction (binary)	Instruction (hexadecimal)	Instruction (binary)	Instruction (hexadecimal)
1100 0100	c4	1100 1000	c8
00000001	01	00000001	01
0010 1001	29	0110 0111	67
1100 1000	c8	1100 1100	cc
00000001	01	00000001	01
0110 0111	67	0110 1101	6d
1100 1100	cc	1100 0100	c4
00000001	01	00000001	01
0110 1101	6d	0101 1110	5e
1100 0100	c4	1100 1000	c8
00000001	01	00000001	01
0101 1110	5e	1100 1000	c8

The below diagram shows the Instruction Memory loaded with the hexadecimal codes for the Fibonacci code as illustrated above:



The following diagram shows the contents of the RAM having reached the end of processing the above input. We see the RAM contains the list of all Fibonacci numbers which can be stored within 4 bits.