CSC 258 FINAL REVIEW

CHEN SUN (JOJO), EZ 4

1 BASIC KNOWLEDGE FRAMEWORK BEFORE MIDTERM

In this part, we list the basic knowledge elements in CSC258 before the midterm. We will focus on those with relatively higher frequency appear on the final exam. I would suggest those do not familiar with course knowledge first read through this part and then review course slides as a complement.

1.1 *General Electricity Knowledge*

a. Semi-conductors:

[i] N (Negative) -Type Semiconductor: Dopant atoms with 5 valence electrons, e.g. (Phosphorous, Arsenic); Charge carrier: Electron (-)

[ii] P (Positive) -Type Semiconductor: Dopant atoms with 3 valence electrons, e.g. (Boron); Charge carrier: A hole missing electron (+)

b. P-N junctions and Transistors:

[i] Combination of holes and electrons at P-N junction will result diffusion current. (Current direction: P $\rightarrow$ N)

[ii] Combination of holes and electrons cancelling each other will create a section with no free carriers called depletion layer. The remaining doping atoms will create an electric field.

[iii] The electric field will cause a drift current. (Current direction: N $\rightarrow$ P). Forward Bias (Positive voltage to P): depletion layer narrower. Reverse Bias (Positive voltage to N): depletion layer wider.

[iv] N-MOS, gate high, connected. P-MOS: Gate low, connected

[v] Know how to use Transistors to create Logic Gates; Know how to identify gates from transistor made circuits.

Sample Question

- **True or False?**

[1] MOSFETs act as a switch by creating a conductive channel between the two n-type sections in a p-type substrate, or vice versa. (True)

[2] A pn junction won't have a depletion layer unless a forward or reverse bias in applied to the junction. (false)

[3] In p-type semiconductors, what are holes? (positive charge carriers)

[4] What is the voltage value in a circuit that is commonly associated with "High voltage"V_{cc} or 5V in TTL gates.

[5] What is MOSFET short for?

```
metal oxide semiconductor field effect transistor
```

[6] Implement a 3 port NAND gate using MOSFETs.

```
Input A,B,C and output Y;
only if A=B=C=1, output Y = 0;
Thus, Y should connect to GND in series;
Similarly, as long as one of the input is 0, output is 1;
Thus, Y should connect to Vcc connect in parallel
```

1.2 *Combinational Logic Design*

a. Truth table Maxterm/Minterm SOP/POS Kmap

Remember: Kmap is based on Gray Code (00,01,11,10)

N input will have 2^N rows of truth table

Understand simplification (1 AND 0 = 0; 1 OR 0 = 1; $A + \bar{A} = 1$; $A \oplus B = A\bar{B} + \bar{A}B$)

*Understand simple bit manipulation:

reverse bits - use XOR with all 1s;

making some part of the bits 1 and remain the other part of the bits unchanged - use OR with all 1s;

b. Combinational building blocks: MUX, Adder/Substractor, DEMUX, Decoder, Comparator, ALU

c. Number System, all the following are N bits

Unsigned Number: [0, 2^N - 1]

Signed-number, e.g. Twos Complement: [-2^{N-1} , 2^{N-1} - 1];

Most negative number : 100...000 = -2^{N-1}; 总长度为2^N

Most positive number : 011...11 = $2^{N-1}1$;

-1 = 11...111 (N bits);

0 = 00...000 (N bits);

To get 2's complement: inverting all the bits and adding 1

Understand Substractor (using XOR gates with Adder)

0x6C or F0 ---> 前半部分变1，后半部分不变

0x6C and 0F ---> 前半部分变0， 后半部分不变

1.3 *Sequential Logic Elements and Circuits*

- Latch: level triggered
- Flip-Flop: Edge Triggered

Flip Flop timing:

- Setup time: Input should be stable for some time before active clock edge
- Hold time: Input should be stable for some time immediately after the active clock edge

• SR (Set-Reset) :
- $S = 0, R = 0 \rightarrow Q' \Leftarrow Q$
- $S = 1, R = 0 \rightarrow Q' \Leftarrow 1$
- $S = 0, R = 1 \rightarrow Q' \Leftarrow 0$
- $S = 1, R = 1 \rightarrow$ unstable behavior, Q′ depends on which input change first

• JK (Similar to SR) : $J \leftrightarrow S$, $K \leftrightarrow R$
- $S = K = 1 \rightarrow$ Toggle, $Q' \Leftarrow \bar{Q}$

• T (Toggle) : Connect J and K ports together
- $T = 1 \rightarrow$ Toggle

• D :
- $Q' \Leftarrow D$

Synchronous and Asynchronous:

- Synchronous Reset: The output is reset to 0 only on the active edge of the clock
- Asynchronous Reset: The output is reset to 0 immediately (as soon as the asynchronous reset signal becomes active), independent of the clock signal

```
Synchronous: always@(posedge CLK);
             if (~reset)
```

```
Asynchronous : always@(posedge CLK or posedge reset)
```

- Register

  Shift Register:

  Linear-feedback shift register (LFSR): a shift register whose input bit is a linear function of its previous state

  Load Register:

- Counter

  Asynchronous Counter - ripple counter

  Synchronous Counter: FSM

Sample Question

Implement half adder using only 2-to-1 mux and not gates

A digital circuit have 5 inputs and 3 Flip-Flops, if write out the full truth table for this circuits, how many rows does this truth table have?

2^8

Design a Ripple Counter that counts down.

Design a Synchronous Counter that counts down.

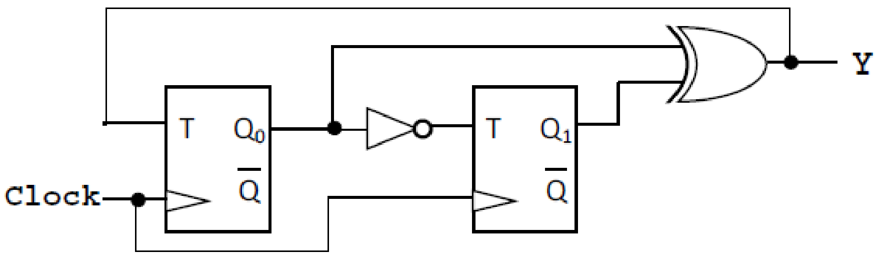
How many min-terms could you have in a circuit with three inputs and three flip-flops?

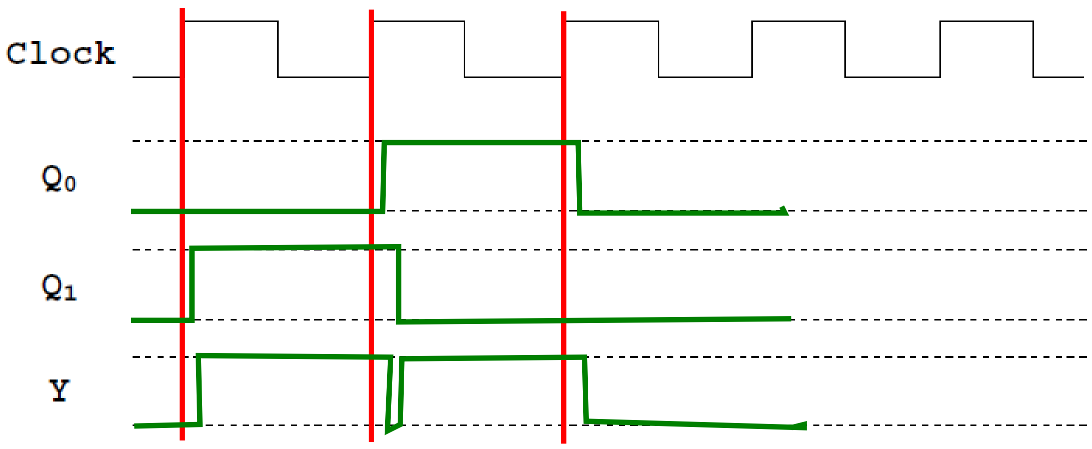
2^6

If you need to design a counter that counts up to 100, how many Flip-Flops are required?

log2(100) = 7

2. Given the following circuit, show what the output value of Q_0, Q_1 and Y will be in the waveform diagram. Assume that Q_0 and Q_1 start with initial values of zero. **(6 marks)**





1.4 *Finite State Machine*

Concept:

- A circuit with k registers can be in one of a finite number (2^k) of unique states

- An FSM has M inputs, N outputs, and k bits of state.

- An FSM consists of two blocks of combinational logic, namely, next state logic and output logic, and register that stores the state.

- Moore Machine: Output decides by states only, independent from input (Safe)

- Mealy Machine: Output decides by states and input together (Fast)

- Design steps:

1 Draw State Diagram

2 Derive state table from state diagram

3 Assign flip-flop configuration to each state

4 Redraw state table with flip-flop values (*)

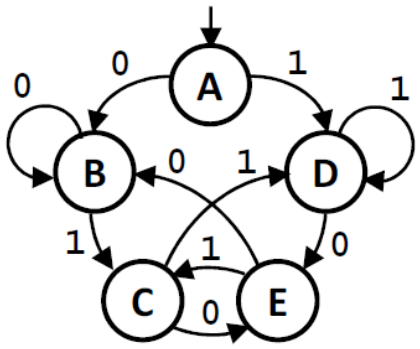
5 Derive combinational circuit for output and for each flip-flop input.

Reason for Step 4: When assigning states, you need to consider the issue of timing with the states.

Example:

Sample Question

7. Consider the finite state machine shown below. The output of this circuit goes high whenever it is in State C or State E. In the spaces below, indicate the operation that this finite state machine performs, and how many flip-flops this will need. **(5 marks)**



Operation: recognize when input alternate between 0 and 1

Number of flip-flops: 3

Design a Moore type FSM that given single bit input X is able to recognice over time 3-bit input sequence 010 and 000. This FSM should have a 2-bit output Y. The most significant bit of Y should be 1 if we have just recognized a 3-bit sequence starting and ending with 0. The least significant bit should match the middle bit of the recognized sequency. Overlapping sequences should also be recognized.

(1) Draw the state Diagram

(2) How many FFs would you need if you were using fully encoded states? How many FFs would you need if you were using one-hot encoding?

(3) If you were using fully encoded states, what is your output logic? write your state table as well.

2 BASIC KNOWLEDGE FRAMEWORK AFTER MIDTERM

2.1 *Memory Concept*

• **Memory**: An array with N-bit addresses and M-bit data has 2^N rows and M columns. Each row of data is called a *word**. Thus, the array contains $2^N \times M$-bit *word**.

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	Wang	Si	Tu	Shuo	Wo	Cong	Wei	Jian
01	Guo	You	Ru	Ci	Hou	Yan	Wu	Chi
10	Zhi	Tu	.	This	is	Horse	Fart	,
11	I	am	Comfortable	said	by	Zhuge	Cun	Fu

Table 1: A pseudo 4×8 bit Memory

Memory Unit Basics:

- 1 Byte = 2 nibble = 8 bit
- 1 Word = 4 Byte = 32 bit
- 1 Half Word = 2 Byte = 16 bit

One-Hot Decoder: The decoder takes in the n-bit binary address, and activates a single row in the memory array.

Tri-State Buffer

Reading and Writing in Memory:
Read From Memory:

- t_{AA}, Address Access Time, Time needed for address to be stable before reading data values.
- t_{OHA}, Output Hold Time, Time output data is held after change of address.

Write to Memory:
- t_{SA}, Address Setup Time, Time for address to be stable before enabling write signal.
- t_{SD}, Data Setup to Write End, Time for data-in value to be set-up at destination.
- t_{HD}, Data Hold from Write End, Time data-in value should stay unchanged after write signal changes.
- t_{AW}, Address Width Time, Time that write signal is high.

```
module ram_example (CLK, CE, Write_or_Read, Addr, Data_in, Data_out);
# (parameter DATA_WIDTH = 8, ADDR_WIDTH = 4)
   intput CLK, CE, WorR;
   input [DATAWIDTH - 1 : 0] Data_in;
   input [ADDR_WIDTH - 1 : 0] Addr;
   output reg [DATAWIDTH - 1 : 0] Data_out;

   // Declare ram variable
   reg [DATAWIDTH - 1 : 0] ram [2**ADDR_WIDTH - 1 : 0];

   always @ (posedge CLK)
   begin
      if (CE)
         begin
            if (Write_or_Read)
               ram[Addr] <= Data_in;
            else
               Data_out <= ram[Addr];
         end
   end
 endmodule
```

Sample Question

How many address bits are needed to specify each byte in a 512 byte memory unit?

Answer : 9

1. How many address bits are needed to specify each instruction in a 512 byte memory unit in a 32 bit machine?

Answer : 7

2. How many bytes can be stored in a memory unit that used 4 address bits and a 16-bit architecture?

2^4*2 = 2^5 bytes = 32 bytes

3. What will happen if the address access time - t_{AA} is too short during reading from Memory?

might read from the wrong address

4. What will happen if the Data Setup time - t_{SD} is too short during Writing to Memory?

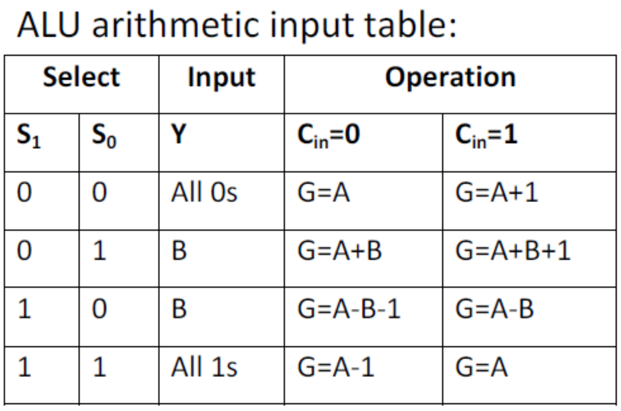
might write the wrong data

5. Write the tri-state buffer's Truth Table (input port: A, WE; output port Y)

2.2 *ALU Concept*

ALU : Arithmetic Logic Unit

- **ALU table** : When $S_2 = 0$:



ALU arithmetic input table:

Select		Input	Operation	
S_1	S_0	Y	C_{in}=0	C_{in}=1
0	0	All 0s	G=A	G=A+1
0	1	B	G=A+B	G=A+B+1
1	0	B	G=A-B-1	G=A-B
1	1	All 1s	G=A-1	G=A

ALU is controlled by S_2, S_1, S_0, when $S_2 = 1$, the output of the logic circuit block appears at the ALU output.

Special condition output:
V: Overflow condition (one of the interrupt)
C: Carry-out bit
N: Negative indicator
Z: Zero-condition indicator

The special condition output is very useful in
- Interrupt
- Branch Instructions (bge, beq etc.)

2.3 *Multiplier and Booth Algorithm*

Multiplication:

- Direct implementation: Layered rows of Adders (O(1) time, O(N^2) size)
- Accumulator Circuits: Single row of adders and several shifters

```
module accumulator_multi (A, B, P, CLK, reset)

input [7:0] A, B;
input CLK, reset;
output reg [15:0] P;
reg [15:0] A_prime;
integer i;

always @ (posedge CLK)
   begin
      if (~reset) begin
         P <= 16'b0;
         R <= 16'b0;
      end
      else begin
         for (i = 0; i < 8; i = i + 1)
            begin
               if (B[i] == 1'b 0)
                  A_prime <= 0;
               if (B[i] == 1'b 1)
                  A_prime <= A <<< i;
               P <= P + A_prime;
            end
      end
   end
endmodule
```

• **Booth Algorithm** : Solve $A \times B$, and store the results in P, A have n bits, B have n bits, P will have 2n bits.

- step 0: add one bit 0 to the right-most side of A, P = 2n bits of zeros, write out $-B$

- loop n times:

a) if last two digits of A == '11' or '00', do Arithmetic Right Shift 1 bit on A and P;

b) if last two digits of A == ′01′, add B to the highest bit of P, do Arithmetic Right Shift 1 bit on A and P;

c) if last two digits of A == ′10′, add $-B$ to the highest bit of P, do Arithmetic Right Shift 1 bit on A and P;

- After n times: multiplication result is in P

- Booth Algorithm in Verilog:

```
module booth_multi (A, B, P, CLK, reset)

input [7:0] A, B;
input CLK, reset;
output reg [15:0] P;
reg [7:0] N;
reg [15:0] N_prime;
reg [8:0] A_prime;
integer i;

always @ (posedge CLK)
begin
   if (~reset) begin
      P <= 16'b0;
      A_prime <= {A, 1'b0};
   end
   else begin
      for (i = 0; i < 8; i = i + 1)
         begin
            if (A_prime[1:0] == 2'b 01)
               N <= B;
            if (A_prime[1:0] == 2'b 10)
               N <= -B;
            N_prime <= {N, 8'b 0};
            P  <= P + N_prime;
            A_prime <=  A_prime >>> 1;
            P <= P >>> 1;
         end
   end
end
endmodule
```

Sample Question

1. Perform booth algorithm on binary values A = 10100, B = 01101

2. Write -5 times 2 using booth algorithm

2.4 *MIPS Concept*

How MIPS instruction works: Fetch $\rightarrow$ Decode $\rightarrow$ Execute $\rightarrow$ Move to the next instruction

Program Counter (PC) stores the memory address of the current instruction.
- PC start from 0 and update by $+4$

• **MIPS Instructions** :

R (Register) type: opcode (6), rs(5), rt(5), rd(5), shamt(5), funct(6)
– opcode will always be 000000 – rd stands for destination

I (Immediate) type: opcode (6), rs(5), rt(5), imm(16)
imm field could be used for:

- immediate operand: addi $ t1, $t1, 100
- a displacement for a memory operand: lh $t, 0($s)
- a branch target offset: beq $s0, $zero, LABEL For branch target offset operations, need shift two bits (since PC+4)

J (Jump) type: opcode (6), addr(26)
- Destination address(32 bit in total) = PC[31:28], the 26 bits addr(26), 00

Helpful Hints: SorTeD

Control Unit Signals:

Read/Write (ENABLE) Signals: PCWrite, PCWriteCond, MemRead, MemWrite, IRWrite, RegWrite

MUX signals: IorD, MemToReg, ALUOp, ALUSrcA, ALUSrcB, RegDst, PCSource
Remark:
– IorD: Instruction (0, from PC) or Data (1, from ALU operation) for memory access

– MemToReg: Register file receiving data from (0, memory) or (1, ALU output)

– RegDst: Which part of the instruction is providing the destination address for register write (1, rd) or (0, rt) or (X, no reg write)

Load and Save (I type) in MIPS:

How to assemble multiple bytes into a larger data-type (like int):

– Big Endian: The most significant byte of the word is stored first (i.e., at address X). The 2nd most significant byte at address X+1 and so on.

– Little Endian: The least significant byte of the word is stored first (i.e., at address X). The 2nd least significant byte at address X+1 and so on.

– MIPS processors are bi-endian, i.e., they can operate with either big-endian or little-endian byte order

Stack:

Stack is part of the Memory

The stack grows towards smaller (lower) addresses

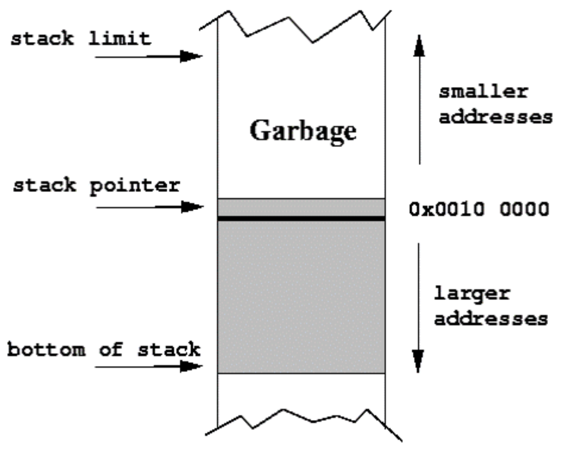


Figure 1: Stack structure

Pop value off the stack:

```
lw $t0, 0($sp) # pop that word off the stack
addi $sp, $sp, 4 # move stack pointer one word
```

Push value onto the stack:

```
addi $sp, $sp, -4 # move stack pointer one word
sw $t0, 0($sp) # push a word onto the stack
```

Function Call:
Call a function: jal LABEL;

jal (J type): It updates register $31 ($ra, return address register) and also the Program Counter.

After its executed, $ra contains the address of the instruction after the line that called jal.

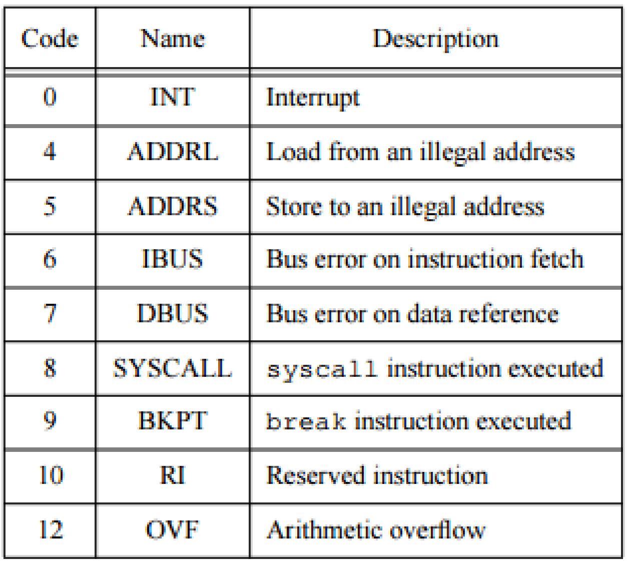
Return a Value: jr $ra;
The PC is set to the address in $ra.

Interrupt : takes place when an external event requires a change in execution.

Example: Overflow, System calls...

MIPS architecture use **polled handling** (for interrupt handling)

In polled handling, processor jumps to exception handler code based on the cause register



Code	Name	Description
0	INT	Interrupt
4	ADDRL	Load from an illegal address
5	ADDRS	Store to an illegal address
6	IBUS	Bus error on instruction fetch
7	DBUS	Bus error on data reference
8	SYSCALL	`syscall` instruction executed
9	BKPT	`break` instruction executed
10	RI	Reserved instruction
12	OVF	Arithmetic overflow

Figure 2: Cause register for polled interrupt handling

rfe: Return From Exception. If the original program can resume afterwards, by calling rfe intruction, interrupt handler can returns to program.

Additional:

MIPS DIV and MULT: $HI stores the Remaider, $LO stores Quotient in division.

```
div $s, $t  #  $LO = $s / $t; $HI = $s % $t
##############
mult $s, $t   #  $LO = $s * $t

mflo $d   # $d = $LO
mfhi $d   # $d = $HI
```

Bitwise Inversion: XORI with ~~FFFF~~

111...11111

Sample Question

I. Write the assembly language instruction(s) that corresponds to each of the tasks provided.

(1) Perform a right arithmetic shift on the value in $to. The number of bits to shift $to by is stored in $so. The result will be stored back into $to.
srav $to, $to, $so

(2) Load half a word from a memory address stored at $ao, and store that value in register $to. The upper bits of $to should be filled with zeroes.
lhu $to, o($ao)

(3) Jump back to the instruction at location top if the value in $t4 is anything other than zero.
bne $t4, $zero, top

(4) Divide the value in $to by 16 and store back into $to
sra $to, $to, 4

(5) Set the value of register $t2 to be the negative of its current value
xori $t2, $t2, ~~FFFF~~
-1

(6) Set the PC value to o
j o

II. For the following assembly language instructions, write the equivalent machine code and vice versa.

(1) addu \$t2, \$t0, \$t1

000000, 01000,01001,01010,xxxxx,100001

(2) lw \$t0, 20(\$s0)

100011,10000,01000,000...10100

(3) 001110 01000 00010 0000000011111111

xori \$v0, \$t0, 255

(4) 000000 10000 00100 01000 00000 100011

subu \$t0, \$s0, \$a0

III. Draw the datapath and write the control signal values

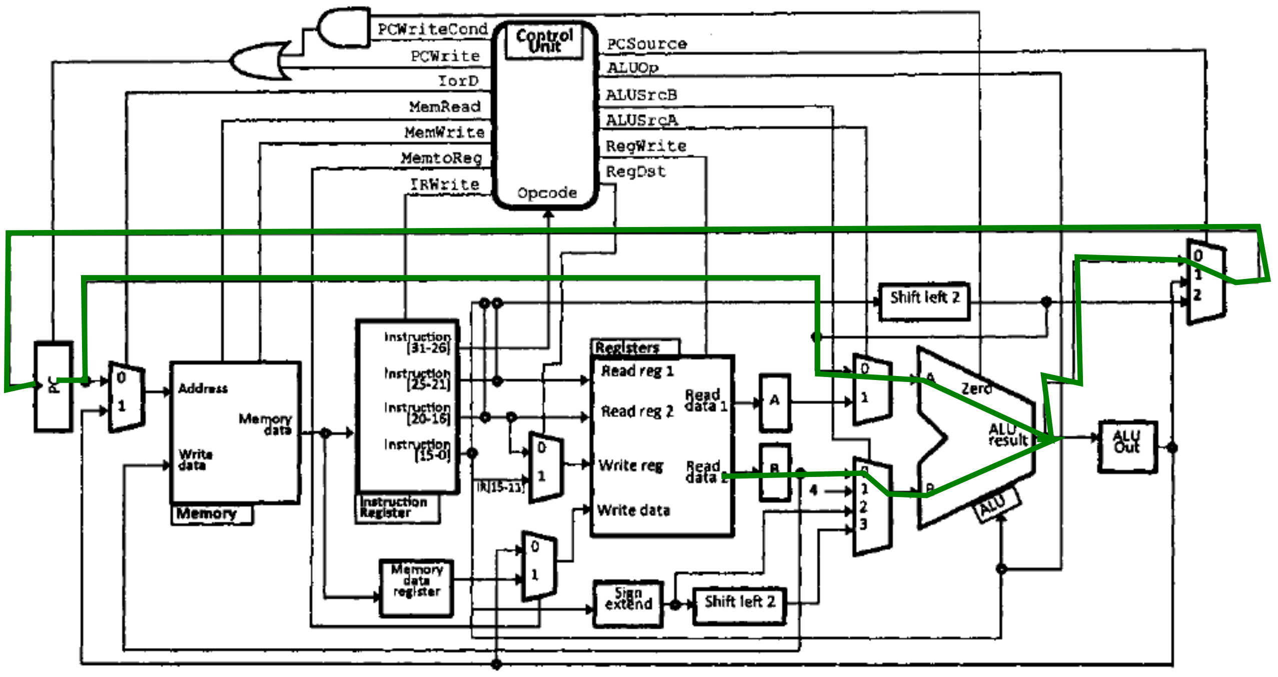
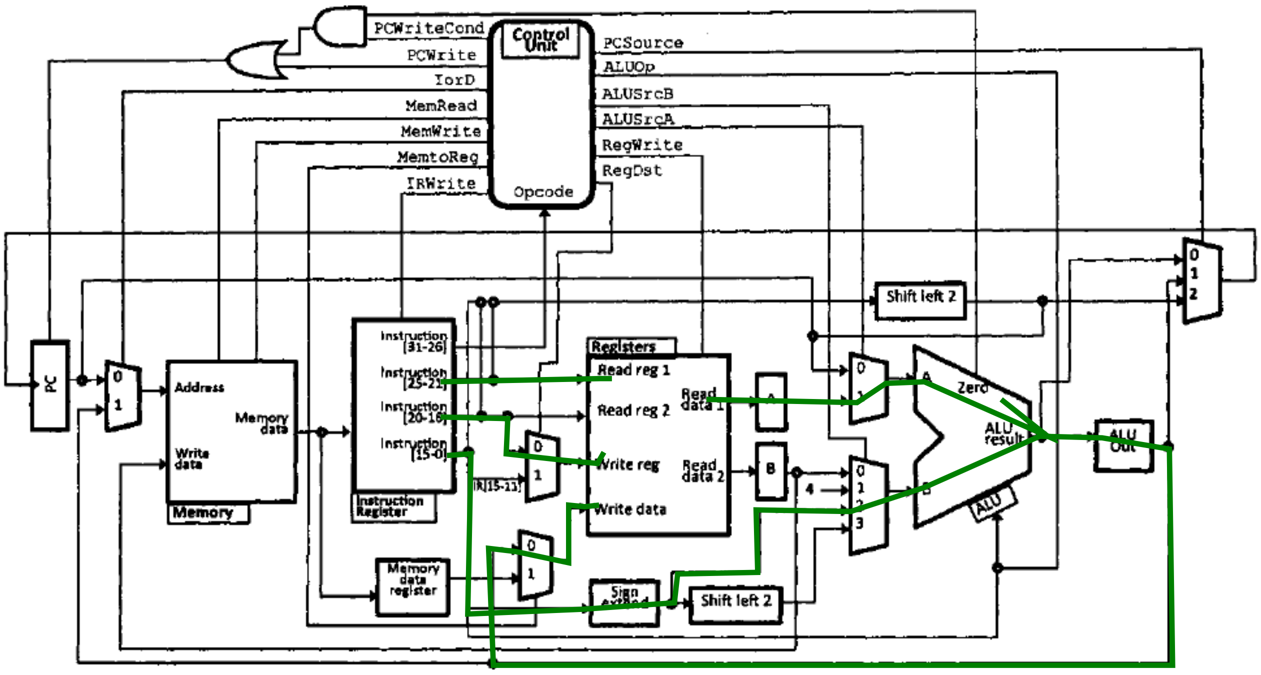


Figure 3: Reduce the program counter by the value stored in \$t0.

PCWrite = 1 , PCWriteCond= X , MemRead= 0 , MemWrite=0 , IRWrite= 0 , RegWrite= 0

IorD= X , MemToReg= X , ALUOp= 010 , ALUSrcA= 0 , ALUSrcB= 00 , RegDst= X , PCSource= 00



addi $s0, $s0, 64

Figure 4: Add 64 to $so and store the result back in $so.

PCWrite = 0 , PCWriteCond= 0 , MemRead=0 , MemWrite= 0 , IRWrite= 0 , RegWrite= 1

IorD= X , MemToReg= 0 , ALUOp= 001 , ALUSrcA=1 , ALUSrcB= 10 , RegDst= 0 , PCSource= XX

3 PRACTICE EXAM

Part A

1. How many bytes can be stored in a memory unit that uses 8 address bits and a 64-bit architecture?

`2^8*8 = 2^11`

2. (True or False) Multiplication with Booth's Algorithm uses the same number of clock cycles as an accumulator circuit.

`true`

3. Which of the following assembly instructions use the processor's sign extend unit?

a) jal [✓ b) bne] [✓ c) lb] d) add

4. Which of the following assembly instructions use the processor's shift unit?

[✓ a) jal] b) addi [✓ c) beq] d) xori

5. (True or False) Interupts are polled on a MIPS architecture.

`true`

6. The maximum distance that a program counter can traverse in memory as a result of a jump instruction is (`2^28`) bytes.
The maximum distance that a program counter can traverse in memory as a result of a branch instruction is (`2^18`) bytes.

7. (True or False) Counters are asynchronous circuits.

`false`

8. (True or False) A 4 bit shift register and load register will took the same amount of clock cycles to store a 4 bit number in the register.

`false`

9. What register can be used as a source operand in assembly instructions, but not a destination?

`$zero`

10. What register can always be divisible by 4?

`$zero`

11. If a FSM has 17 states, what is the minimum number of flip-flops that you would need to implement it in a circuit? How many flip-flop you would need if you use one-hot encoding?

log2(17) = 5

12.(True or False) A pn junction won't have a depletion layer unless a forward reverse biad is applied to the junction.

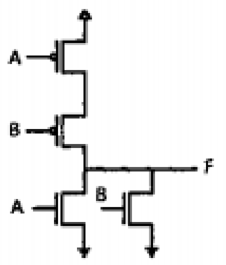
false

13. Which of the following Hex addresses are valid PC values when translated into 32-bit binary addresses?
a) E45CBAC1 b) 1234ABCD [✓ c) DCBA1234] [✓ d) FFFF0000]

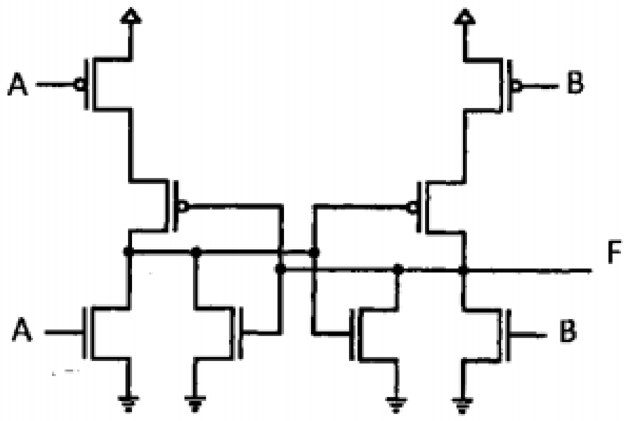
14. A complete truth table with 32 rows has (5) columns of 1-bit input.

15.

Recall the design of a NOR-gate:



What does the following circuit implement?



SR latch

if use NAND gate - then it will be S'R' latch

16. (True or False) When we push a value onto the stack, the value of the stack pointer would decrease.

true

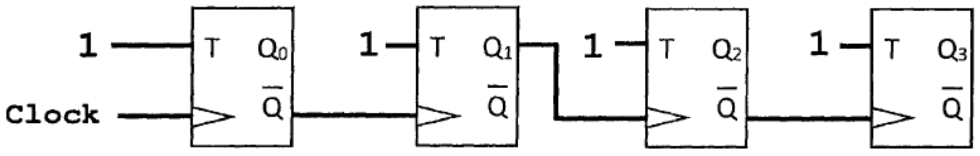
17. (True or False) The following two parts of code are equivalent

```
addi $sp, $sp, -8
sw $t0, 0($sp)
sw $t1, 4($sp)
```

```
addi $sp, $sp, -4
sw $t0, 0($sp)
addi $sp, $sp, -4
sw $t1, 0($sp)
```

Part B

1. Consider the flip-flop diagram below. What values will be on Q_3, Q_2, Q_1, and Q_0, after each consecutive clock cycle? Write your answer in the spaces provided, one space per clock cycle. **(6 marks)**

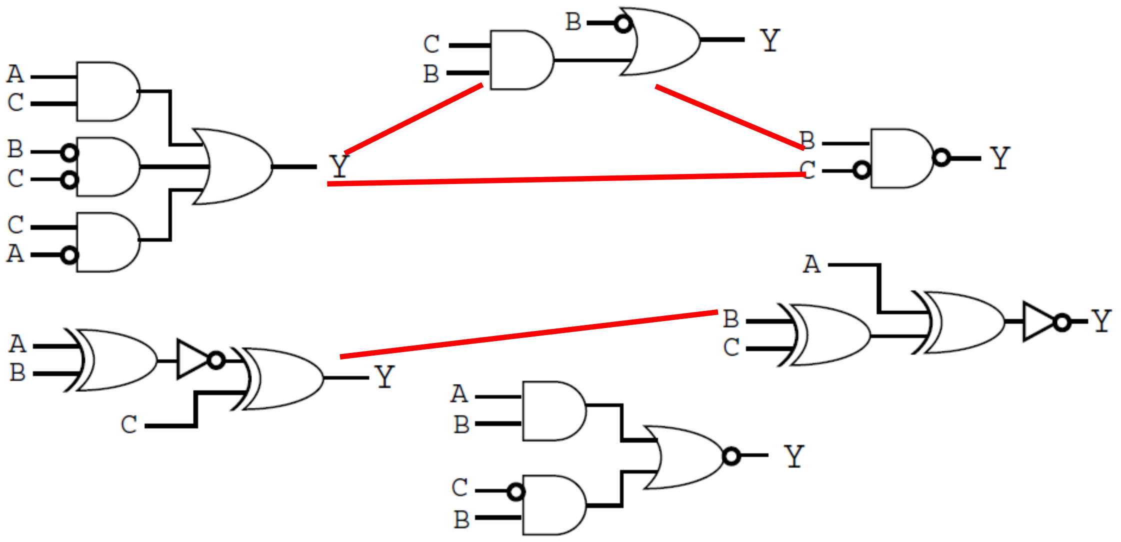


Q_3 Q_2 Q_1 Q_0

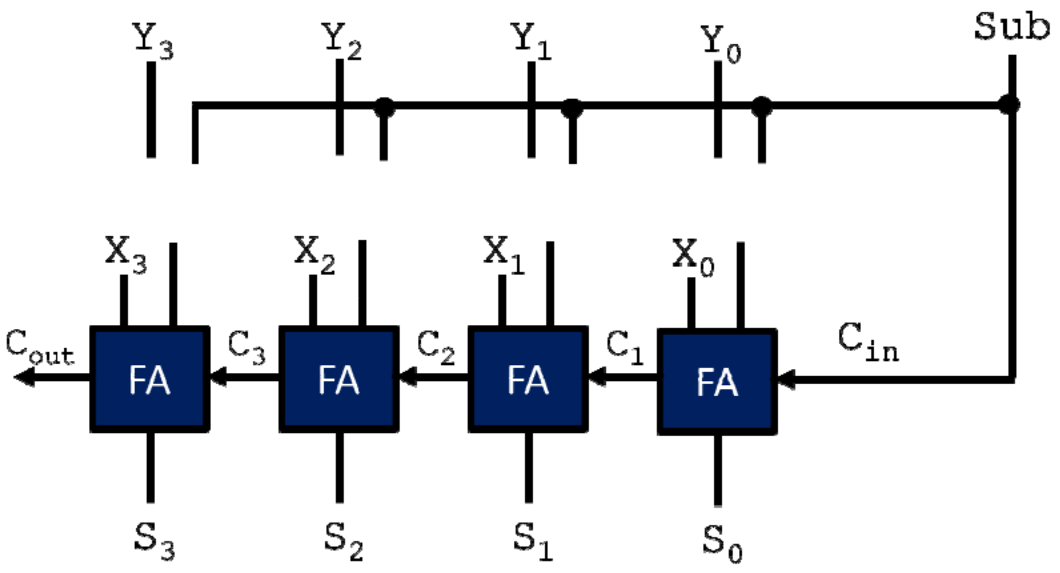
Initial value (before first clock pulse): 0000

0001

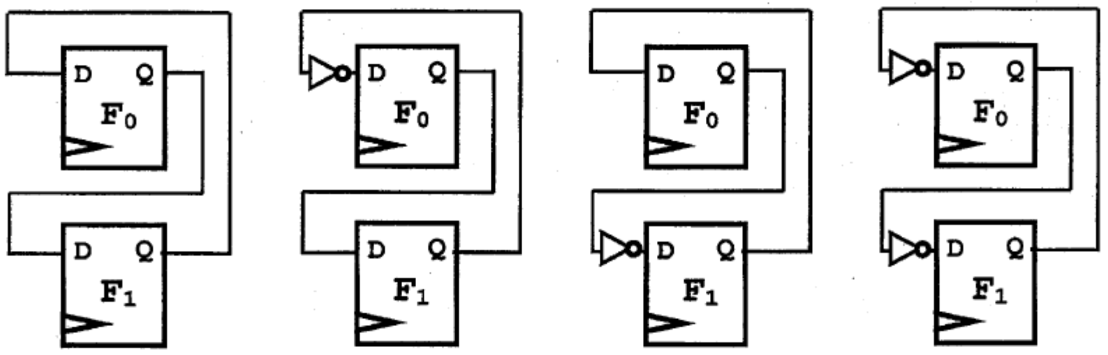
2. Which of the circuits below have equivalent behaviour? Draw lines that connect any circuits that match. **(6 marks)**



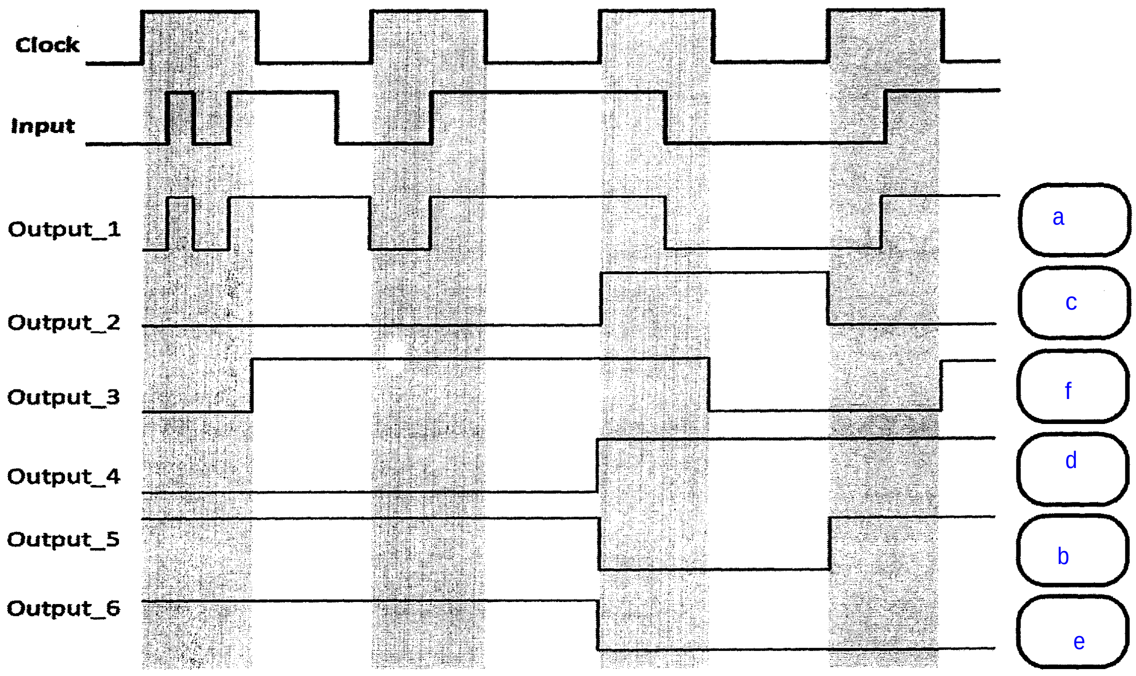
3. Draw gates into the following diagram to implement a subtractor circuit. **(2 marks)**

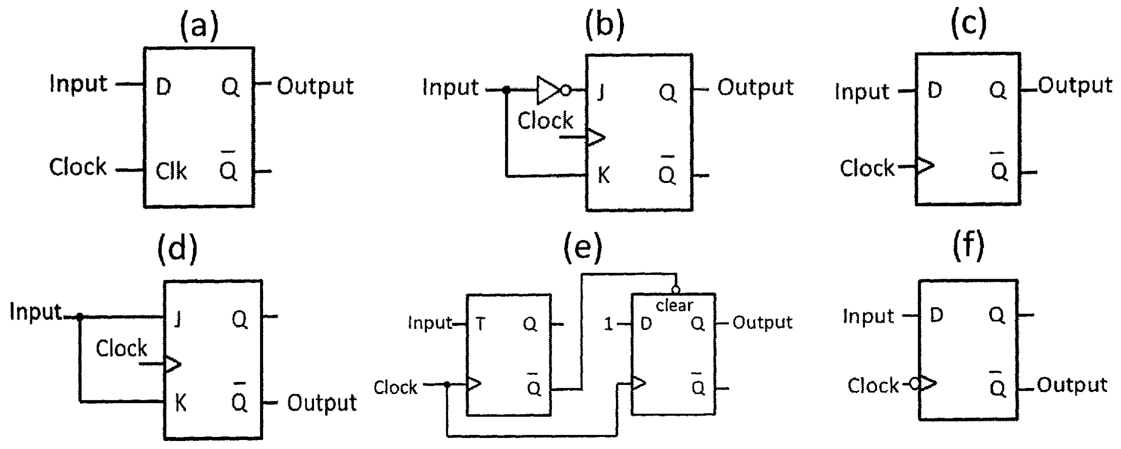


4. Draw the state diagram start from $F_1F_0 = 00$



3. For every output waveform below, specify in the spaces on the right which sequential circuits at the bottom would generate this output behaviour. If you think there is no match, write "None". Assume functional simulation and that the initial Q state, when unknown, was zero. **(6 marks)**





Part C

Write the assembly instruction, system signals, and draw the datapath on the following graphs

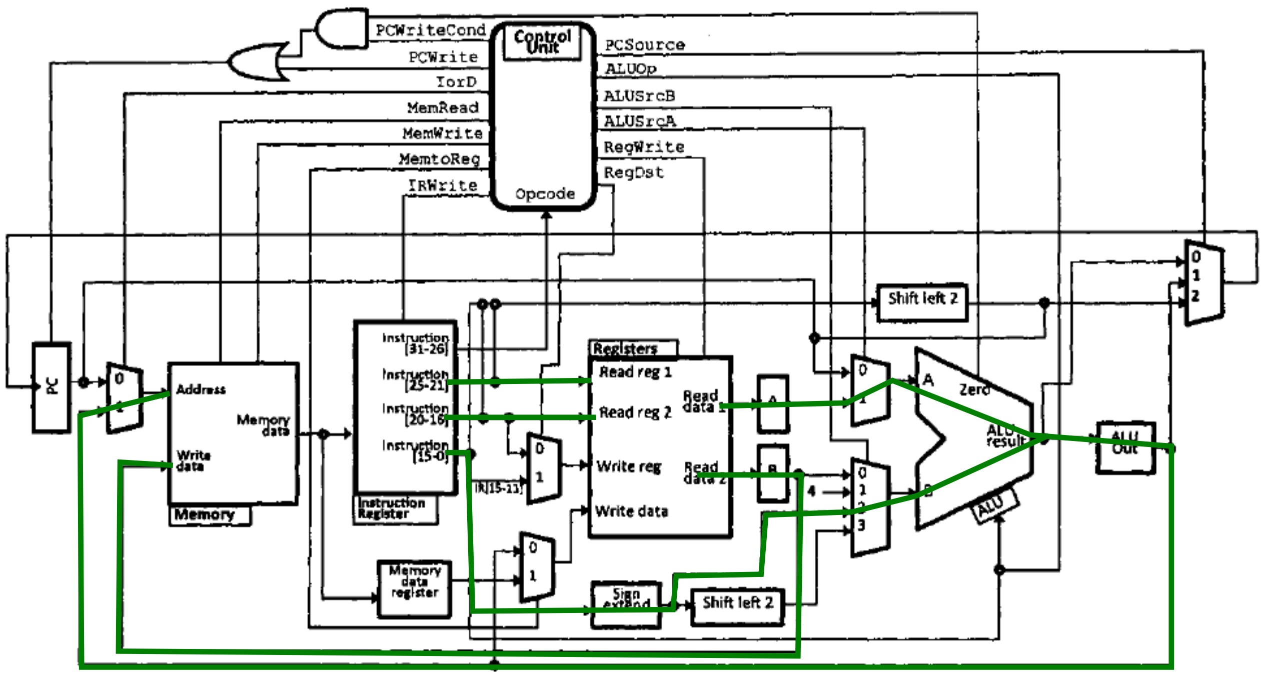


Figure 5: Push the value stored in $ao onto the stack with no offset

MIPS Instruction: sw $a0, 0($sp)

MIPS Instruction binary code: 101011,11101,00100,0000...0000

PCWrite = 0 , PCWriteCond= 0 , MemRead= 0 , MemWrite= 1 , IRWrite= 0 , RegWrite= 0

IorD= 1 , MemToReg= X , ALUOp= 001, ALUSrcA= 1 , ALUSrcB= 10 , RegDst= X , PCSource= XX

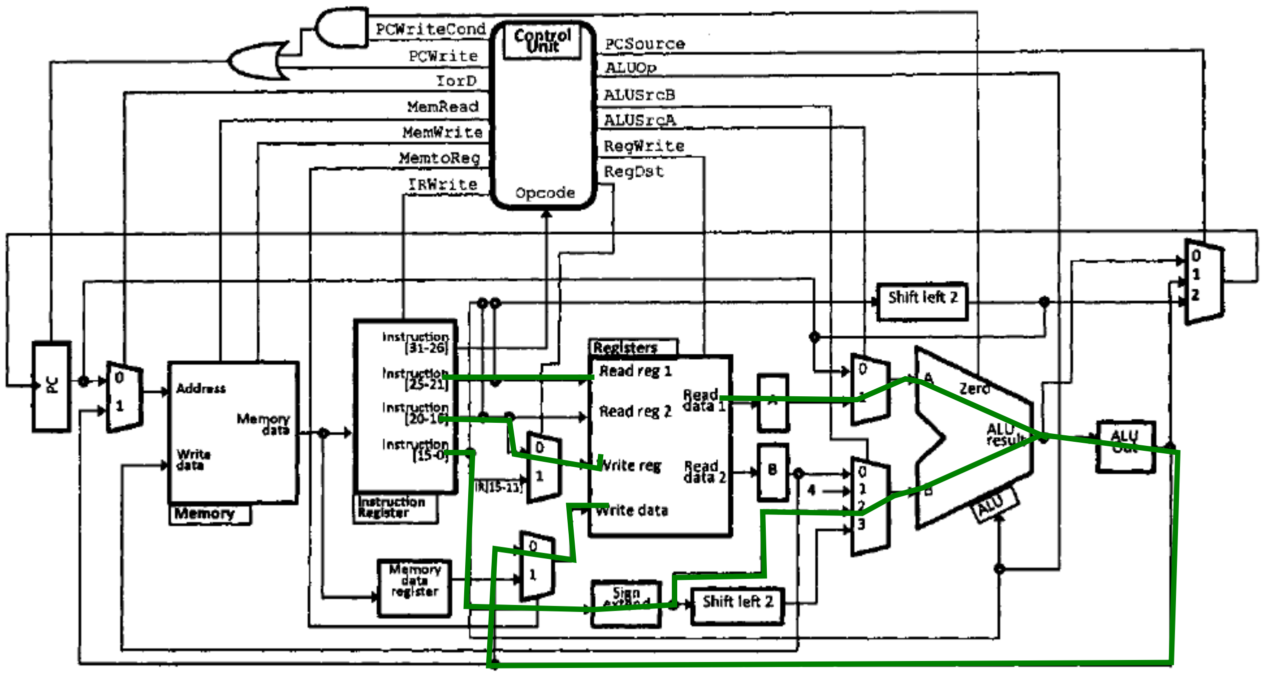


Figure 6: Add 4 to $to and store the result in $t1

MIPS Instruction: addi $t1, $t0, 4

MIPS Instruction binary code: 001000, 01000, 01001, 0000....000100

PCWrite = 0 , PCWriteCond= 0 , MemRead= 0 , MemWrite= 0 , IRWrite= 0 , RegWrite= 1

IorD= X , MemToReg= 0 , ALUOp= 001, ALUSrcA= 1 , ALUSrcB= 10 , RegDst= 0 , PCSource= XX

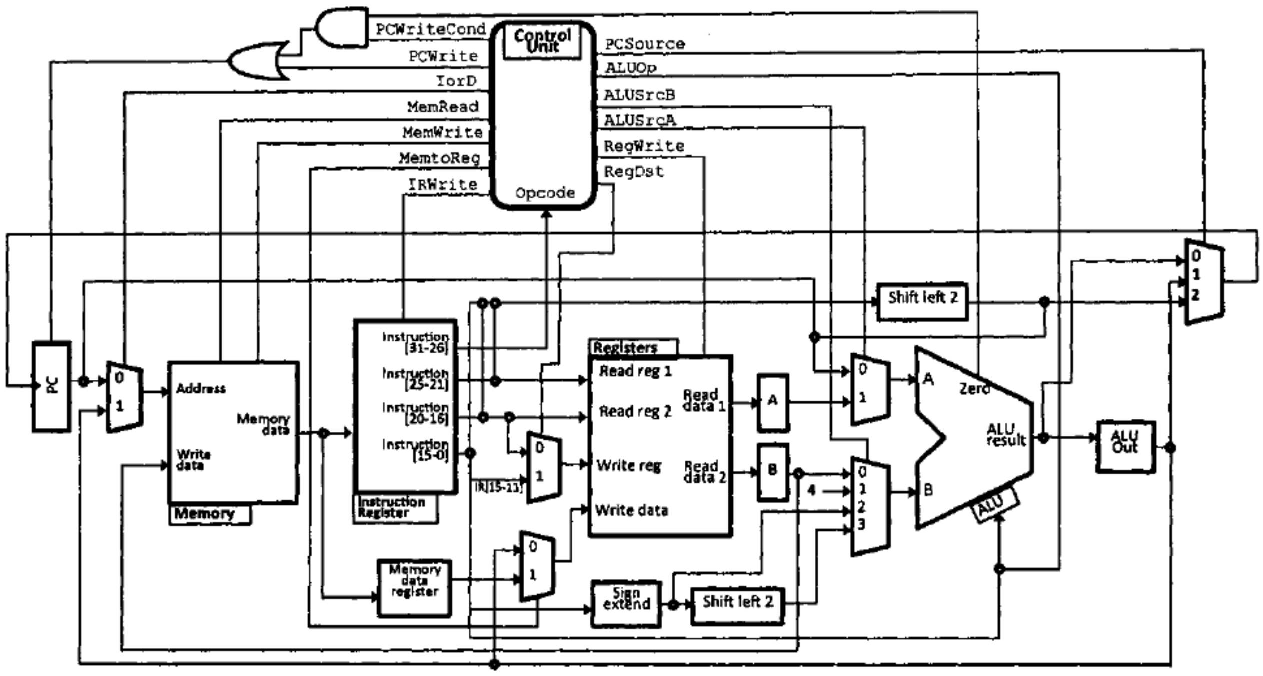


Figure 7: Branch to the new PC value LABEL if $t0 is equal to $t1, LABEL has a offset of -16 between the current PC

MIPS Instruction: `beq $t0, $t1, LABEL`

MIPS Instruction binary code: `000100,01000,01001,111...110000`

PCWrite = , PCWriteCond= , MemRead= , MemWrite= , IRWrite= , RegWrite=

IorD= , MemToReg= , ALUOp= , ALUSrcA= , ALUSrcB= , RegDst= , PCSource=

2. In the space provided, write the resulting `list` values for each assembly program. **(12 marks)**

```
        .data
len:    .word     5
list:   .word     -4, -1, 0, 1, 4
        .text
main:   addi $s0, $zero, list
        addi $s1, $zero, len
        lw $t1, 0($s1)
top:    lw $t0, 0($s0)
        add $t0, $t0, $t0
        sw $t0, 0($s0)
        addi $t1, $t1, -1
        addi $s0, $s0, 4
        bne $t1, $zero, top
end:    jr $ra
```

```
        .data
len:    .word     5
list:   .word     -4, -1, 0, 1, 4
        .text
main:   addi $s0, $zero, list
        addi $s1, $zero, len
        lw $t1, 0($s1)
top:    lw $t0, 0($s0)
        sub $t0, $t0, $t0
        sw $t0, 0($s0)
        addi $t1, $t1, -1
        addi $s0, $s0, 4
        bne $t0, $zero, top
end:    jr $ra
```

```
        .data
len:    .word     5
list:   .word     -4, -1, 0, 1, 4
        .text
main:   addi $s0, $zero, list
        addi $s1, $zero, len
        lw $t1, 0($s1)
top:    lw $t0, 0($s0)
        addi $t1, $t1, -1
        addi $s0, $s0, 4
        add $t0, $t0, $t0
        sw $t0, 0($s0)
        bne $t1, $zero, top
end:    jr $ra
```

```
        .data
len:    .word     5
list:   .word     -4, -1, 0, 1, 4
        .text
main:   addi $s0, $zero, list
        addi $s1, $zero, len
        lw $t1, 0($s1)
        lw $t0, 0($s0)
top:    addi $t1, $t1, -1
        add $t0, $t0, $t1
        bne $t1, $zero, top
        addi $s0, $s0, 4
        sw $t0, 0($s0)
end:    jr $zero
```

4 APPENDIX

MIPS program

"Gain new knowledge by reviewing old" —— Confucius

1. Convert the following code to MIPS assembly

```
a = 0
b = 1
while a != 100
   a = a + 1
   b = b - a
a = b*4
```

```
        addi $t0, $zero, 0;   # \$t0 hold value of a
        addi $t1, $zero, 1;    # \$t1 hold value of b
        addi $t3, $zero, 100;

LOOP:   beq $t0, $t3, OVER
        addi $t0, $t0, 1
        sub $t1, $t1, $t0
        j LOOP
OVER:   mult $t1, 4
        mflo $t0
```

2. Converting Strcpy()

```
strcpy(char x[], char y[])
{
   int i;
   i = 0;
   while ( (x[i] = y[i]) != '\0' )
      i = i + 1;
   return i
}
```

```
strcpy:  lw $a0, 0($sp)       # pop x address off the stack
        addi $sp, $sp, 4
        lw $a1, 0($sp)        # pop y address off the stack
        addi $sp, $sp, 4
        add $s0, $zero, $zero  # $s0 = offset i
L1:     add $t1, $s0, $a0     # $t1 = x + i
        lb $t2, 0($t1)        # $t2 = x[i]
        add $t3, $s0, $a1     # $t3 = y + i
        sb $t2, 0($t3)        # y[i] = $t2
        beq $t2, $zero, L2    # y[i] = '\0'?
        addi $s0, $s0, 1   # i++
        j L1 # loop
L2:     addi $sp, $sp, -4     # push i onto top of stack
        sw $s0, 0($sp)
        jr $ra             # return
```

3. Write a Assembly computes the sum of the first N positive integers. For example, for N = 3, the code should produce $1 + 2 + 3 = 6$. The value N is always positive and is provided in memory location with Label N. Once the program computes the sum, it should store the result to memory at label RESULT. You can assume that all operations are 32 bit and ignore the possibility of overflow.

```
        .data
N:         .word   3
RESULT:    .space  4

        .text
main:   addi $s0, $zero, N
        lw $t0, 0($s0)   # $t0 = 3
        addi $t1, $zero, 0
        beq $t0, $zero, OUT
        add $t1, $t1, $t0
        addi $t0, $t0, -1
        j main
OUT:    addi $s1, $zero, RESULT
        sw $t1, 0($s1)
```

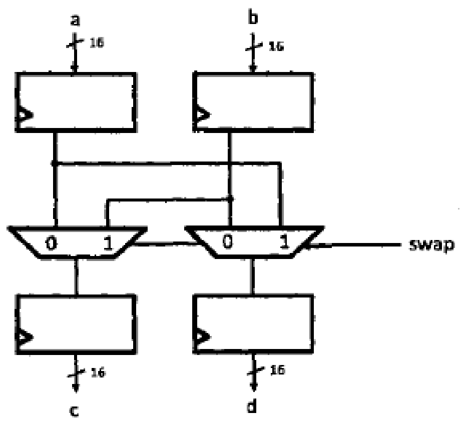
Verilog

1. FSM Verilog

From Lab 5

2.

The following schematic implements a circuit that optionally swaps two input values *a* and *b* to the outputs *c* and *d* depending on input *swap*. Both the inputs and the outputs are registered and carry 16−*bit* unsigned data. The connections to *clk* are not shown. There is no reset. The *swap* input controls both multiplexers. **Write Verilog that implements this circuit in a module named *OptionalSwap16*. Behavioural Verilog may be a good option...**



```
module OptionalSwap16 (CLK, a, b, c, d, swap);
   input [15:0] a, b;
   input swap, clk;
   output reg [15:0] c, d;
   reg [15:0] a_out, b_out;
   wire [15:0] c_in, d_in;

always @ (posedge clk)
   begin
      a_out <= a;
      b_out <= b;
      c <= c_in;
      d <= d_in;
   end

always @ (*)
   begin
      if (swap == 1)
         c_in = b_out;
         d_in = a_out;
      else
         c_in = a_out;
         d_in = b_out;
      end
endmodule
```

3. Verilog common notes:

- The difference between "⇐" and "="

- Assign in a always block will raise error

- reg type can not be directly connected to a structural net expression e.g.
  reg W;

  xor(W, B, C)

  will raise error

Level Hentai
Your instructors' Name: Steve Engels, Office Hours: MW 2-3, BA4266

Your instructors' Name: Sajad Shirali-Shahreza, Office Hours: T 9:30-10:20, W 11-12, BA3219

Your Lab is at ()?

Your TA's name:

The DE1-SOC board used in the labs has (10) switches, (4) keys.

The DE1-SOC board used (Cyclone V) (5CSEMA5F31C6) chip, and 24-bit VGA DAC