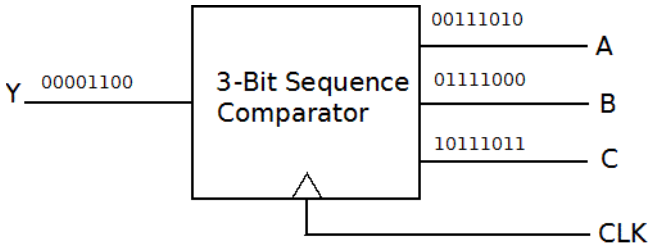
Class	CS147, Sec 01
Homework	II
Due Date	Nov 05, 2018 11:59 PM PST
Instructions	1. There are 5 questions with total 100 points. 2. Please create electronic document with your answer. 3. There is no need to include the question itself. However, you **MUST** include question number and sub-part index if any. Example: 5(b) 4. Please create a PDF document **hw2.pdf** and ***upload that in Canvas*** assignment page by the due date. 5. **Please re-check you submission for any logistic errors (empty file, corrupted PDF, and many more) and re-submit if needed. Once grading is started, any file with logistics errors will be given 0 point.** 6. **NO** handwritten (and scanned) document is accepted. You answer must be typed in (and computer drawn if diagram is asked) using word processing software. 7. **NO LATE SUBMISSION.** 8. **Please explain your answer clearly – just writing the final answer in a word or two is not sufficient in most of the cases.**

1. Design a 3-bit sequence comparator circuit which has 3 bits data inputs (A, B, C), 1 clock signal (CLK) and 1 bit output (Y). This circuit takes stream of bits per clock cycle through these 3 data input pins. Output turns 1 in a clock cycle if latest 3 bit sequence in 3 bit streams matches. Show all the necessary steps to implement this logic circuit and draw final schematic diagram. Use D-F/F as storage if needed. Sample input / output is given in following block diagram of this circuit. [**20pts**]



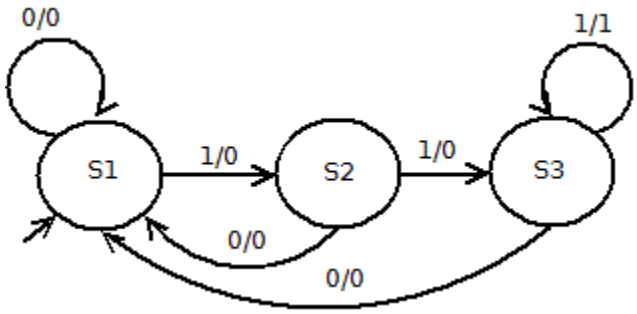
Ans:

Essentially this machine will have a state machine which should identify 3 consecutive matches between 3 bit streams. To keep the truth table under control (not more than 4 variables) for this sequential machine, let's have two part of the design. First part determines whether there is a match at this moment and second part determines if there are 3 consecutive matches. The first part is pure combinational circuit and the second part is sequential digital circuit.

Truth table for the first part is as following, where M is output indicating matching of input or not. Therefore logic equation for M is M = A'B'C' + ABC

A	B	C	M
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

For the second part primary input is this M. The state diagram will be as following.



- State S1 denotes that no match has been seen. As far as there is no match found from this stage it will remain at this stage. Once it seems a match at state S1, machine will switch to state S2.
- State S2 denotes that one match has been seen. If the next input is a match it will go to S3. However, if next input is 0 then it will go back to S1.
- State S3 denotes that two match has been seen. If the next input is a match it will output 1 as indication of three consecutive matches and it will stay at S3 to catch successive 3 consecutive matches. If no match is found at this point it will go back to S1.

State transition and output table will look as following.

Present State	Next State		Output	
	M=0	M=1	M=0	M=1
S1	S1	S2	0	0
S2	S1	S3	0	0
S3	S1	S3	0	1

3 states needs 2 bits for encoding the state. Let's have encoding as S1=00, S2=01, S3=11

With this above state assignments, state transition table and output table will look like following.

Present State (RS)	Next State (RS)		Output	
	M=0	M=1	M=0	M=1
00	00	01	0	0
01	00	11	0	0
11	00	11	0	1

Corresponding truth table will be as following for output Y, and next state value (R_next, S_next)

	M	R	S	Y	R_next	S_next
m0	0	0	0	0	0	0
m1	0	0	1	0	0	0
m3	0	1	1	0	0	0
m4	1	0	0	0	0	1
m5	1	0	1	0	1	1
m7	1	1	1	1	1	1

Therefore equation in SOP forms will be as following.

- $Y = \sum m(7) + d(2,6)$
- $R\_next = \sum m(5,7) + d(2,6)$
- $S\_next = \sum m(4,5,7) + d(2,6)$

K-map for $Y = \sum m(7) + d(2,6) = M.R$

M\RS	00	01	11	10
0				X
1			1	X

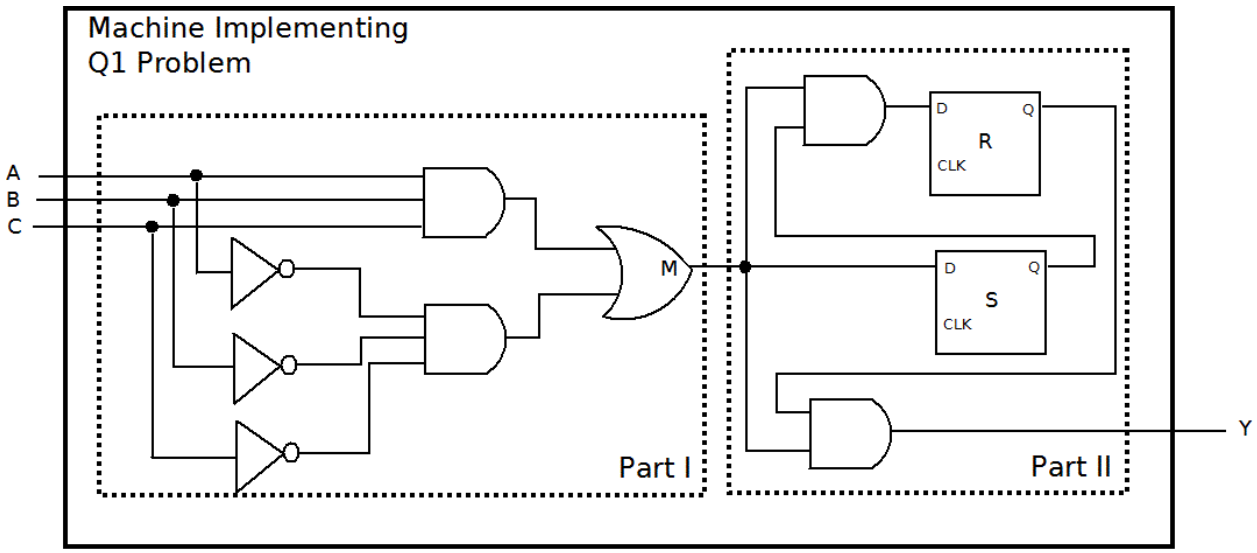
K-map for $R\_Next = \sum m(5,7) + d(2,6) = M.S$

M\RS	00	01	11	10
0				X
1		1	1	X

K-map for S_Next = ∑m(4,5,7) + d(2,6) = M

M\RS	00	01	11	10
0				X
1	1	1	1	X

Combining first part and second part of the circuit, logic circuit of entire machine will look like following.



2. How many basic logic gates a 32-bit ripple carry adder-subtractor circuit will have (as in lecture note 9, page 14). Assume this digital implementation has basic logic gate list as 2-input NAND, 2-input NOR, and NOT. [**10pts**]

Ans:

Single bit full adder sum equation is S = CI'.(A'.B + A.B') + CI.(A.B + A'.B') [Lecture 19, page 7]

Single bit full adder carry out equation is C = CI. (A + B) + A.B [Lecture 19, page 7]

Expressing these equation with 2-input NAND, 2-input NOR And NOT we get following.

S = CI'.(A'.B + A.B') + CI.(A.B + A'.B')

= (CI + (A'B + AB')')' + (CI' + (AB + A'B')')'

= (((CI + (A'B + AB')')' + (CI' + (AB + A'B')')')')'

= (**(** CI + ((A+B')' + (A'+B)')' **)'** + **(**CI' + ((A'+B')' + (A + B)')' **)'**)')'

Logic gate needed

- **4 NOT** :
  - A'
  - B'
  - CI'
  - (((CI + ((A+B')' + (A'+B)')')' + (CI' + ((A'+B')' + (A + B)')')')')'
- **9 NOR** :
  - (A+B')'
  - (A'+B)'
  - (A'+B')'
  - (A + B)'
  - **(**(A+B')'+(A'+B)'**)'**
  - **(**(A'+B')'+(A + B)'**)'**
  - **(**CI + ((A+B')' + (A'+B)')' **)'**
  - **(**CI' + ((A'+B')' + (A + B)')' **)'**
  - ((CI + ((A+B')' + (A'+B)')')' + (CI' + ((A'+B')' + (A + B)')')')'

Y = CI(A+B) + AB = (CI' + (A+B)')' + (A'+B')'

= (((CI' + (A+B)')' + (A'+B')')')'

We can reuse following terms from logic circuit of S. Then number of gates required for this circuit is as following.

- **1 NOT**
  - (((Cl’ + (A+B)’)’ + (A’+B’)’)’)’

- **2 NOR**
  - (Cl’ + (A+B)’)’
  - ((Cl’ + (A+B)’)’ + (A’+B’)’)’

Hence 1-bit full adder will have total **5 NOT and 11 NOR gates.**

Since 32-bit adder / subtractor uses 32 1-bit full adder, it will have total (5*32) = 160 NOT gates and (11*32) = 352 NOR gates.

This adder/subtractor circuit also has 16 2-input XOR gates whose logic equation is A’B + AB’ = ((A’B)’(AB’)’)’. Hence each XOR will have 3 2-input NAND gates (re-using A’ and B’ from previous circuit). Therefore there will be (3*32) = 96 2-input NAND gates.

Therefore total number of basic logic gates in a 32-bit adder/subtractor circuit will be (160+352+96) = 608.

[PS: Answer can be different depending on your transformation of the logic expression using only 2-input NAND, NOR and NOT gate]

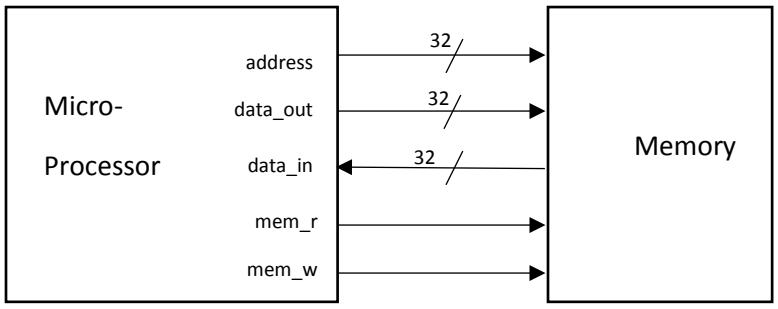
3. For a non-pipeline implementation of data and control path in following diagram for a processor implementing CS147DV show the control signal logic values (in compact Hex format) at different phase of the processor executing the following instructions. You need to construct 10 tables similar to Table shown (may use hexadecimal for multi-bus signal, put 0 if don't care). Assumptions on this design are as following. [**30pts**]
   a) Assume that the memory / register file with read=0, write=0 is hold configuration (hold the previous read data) and read=1, write=1 causes electrical isolation of the memory (HiZ). Both of memory and register file reads with read=1, write=1 and writes with read=0, write=1. If memory or register needs to hold previous value, keep it at hold configuration (not in 'read' configuration).
   b) ALU assumes operation code as in Lecture 11 notes. For 'alu_oprn' CTRL[25] is MSB and CTRL[22] is LSB.
   c) Instruction register is implemented with a transparent latch. This means, as soon as 'ir_load' is turned to 1, input is transferred to output without any waiting for successive clock cycle.

I.	add	r1, r2, r1
II.	srl	r15, r19, 0xa3
III.	jr	r8
IV.	addi	r15, r14, 0x1234
V.	andi	r3, r4, 0x8a5f
VI.	lui	r14, 0xabcd
VII.	beq	r21, r30, 0x123a; // r21 = 0x2; r30 = 0x1
VIII.	sw	r29, r10, 0xa5a5
IX.	jal	0x3A0B12A
X.	push	

Control Signal	CTRL	IF	ID/RF	EXE	MEM	WB
CTRL[0]	pc_load					
CTRL[1]	pc_sel_1					
CTRL[2]	pc_sel_2					
CTRL[3]	pc_sel_3					
CTRL[4]	ir_load					
CTRL[5]	mem_r					
CTRL[6]	mem_w					
CTRL[7]	r1_sel_1					
CTRL[8]	reg_r					
CTRL[9]	reg_w					

CTRL[10]	wa_sel_1					
CTRL[11]	wa_sel_2					
CTRL[12]	wa_sel_3					
CTRL[13]	wd_sel_1					
CTRL[14]	wd_sel_2					
CTRL[15]	wd_sel_3					
CTRL[16]	sp_load					
CTRL[17]	op1_sel_1					
CTRL[18]	op2_sel_1					
CTRL[19]	op2_sel_2					
CTRL[20]	op2_sel_3					
CTRL[21]	op2_sel_4					
CTRL[22:25]	alu_oprn					
CTRL[26]	ma_sel_1					
CTRL[27]	ma_sel_2					
CTRL[28]	md_sel_1					
CTRL Signal Value in Hex		32'hxxxxxxxx	32'hxxxxxxxx	32'hxxxxxxxx	32'hxxxxxxxx	32'hxxxxxxxx

The system looks like following with data path of the microprocessor as in https://sjsu.instructure.com/courses/1265088/modules/items/9704715



Ans:

Index	Instruction	Micro Instruction Codes				
		IF	ID/RF	EXE	MEM	WB
I	add r1, r2, r1	0x08000020	0x00000110	0x00600000	0x00600000	0x0060920B
II	srl r15, r19, 0xa3	0x08000020	0x00000110	0x02940000	0x02940000	0x294920A
III	jr r8	0x08000020	0x00000110	0x0000000	0x00000000	0x00000009
IV	addi r15, r14, 0x1234	0x08000020	0x00000110	0x00480000	0x00480000	0x0048960B
V	andi r3, r4, 0x8a5f	0x08000020	0x00000110	0x01400000	0x01400000	0x0140960B
VI	lui r14, 0xabcd	0x08000020	0x00000010	0x00000000	0x00000000	0x0000D60B
VII	beq r21, r30, 0x123a	0x08000020	0x00000110	0x00A00000	0x00A00000	0x00A0000B
VIII	sw r29, r10, 0xa5a5	0x08000020	0x00000110	0x00480000	0x00480040	0x0000000B
IX	jal 0x3A0B12A	0x08000020	0x00000010	0x00000000	0x00000000	0x00000A01
X	push	0x08000020	0x00000190	0x00000000	0x14000040	0x0093000B

Details at https://sjsu.instructure.com/files/51927328/download?download_frd=1

4. A computing system X is running on 1.6GHz clock. Another system Y running on 2.5GHz clock. Both of these systems support 4 types of instruction A, B, C, D. The following is the CPI table per instruction type in both the system. To compare performance between these two system one benchmark program has been used which has mix of 25% type A, 25% type B, 30% type C and 20% type D. Compare performance between these two systems (P_Y/P_X) with respect to this benchmark program? [**20pts**]

Instruction Type	CPI of X	CPI of Y
A	4	3
B	2	5
C	1	4
D	3	1

Ans:

Let's assume there is N number of instructions in the benchmark program. Therefore distribution of the instructions per type will be as following.

Instruction Type	# Instructions
A	0.25N
B	0.25N
C	0.3N
D	0.2N

Number of clock cycles needed by this benchmark program in processor X would be (4*0.25N) + (2*0.25N) + (1*0.3N) + (3*0.2N) = N + 0.5N + 0.3N + 0.6N = 2.4N. Hence total run time in processor X will be (2.4N/1.6GHz).

Similarly, number of clock cycles needed by this benchmark program in processor Y would be (3*0.25N) + (5*0.25N) + (4*0.3N) + (1*0.2N) = 0.75N + 1.25N + 1.2N + 0.2N = 3.4N. Hence total run time in processor Y will be (3.4N/2.5GHz).

Therefore performance comparison

$(P_Y/P_X) = (E_X/E_Y)$

= (2.4/1.6GHz) / (3.4N/2.5GHz)

= (2.4*2.5) / (1.6*3.4)

= 6/5.44 = 1.103

5. Consider the following piece of code in a 5-stage pipeline processor (as discussed in class).
    a. Fill out the data hazard and resolution table. Use <inst#>-<stage> to denote instruction-stage in pipe line. For example, 1-MEM means 'MEM stage for instruction 1'. Mark the resolution methods as FWD (data forward) and STALL (stall). Consider no reordering in this case. [**5pts**]
    b. Fill out data hazard and resolution table after the stall is inserted. [**5pts**]
    c. Write down minimally re-ordered code to avoid stall. Fill out data hazard and resolution after re-ordering. Do not alter the instruction ID numbers in left most columns [**10pts**]

ID	Instruction	Pipeline Stages											
1	lw r1, r20, 0x2056	IF	ID	EXE	MEM	WB							
2	lw r2, r21, 0xF5C4		IF	ID	EXE	MEM	WB						
3	add r3, r1, r2			IF	ID	EXE	MEM	WB					
4	lw r4, r22, 0x0014				IF	ID	EXE	MEM	WB				
5	addi r8, r4, 0x1A					IF	ID	EXE	MEM	WB			
6	add r5, r8, r4						IF	ID	EXE	MEM	WB		

Ans:

a) The data hazard table for original code as following.

Data Hazard (Original)			
STAGE	**DEPENDENCY**	**RESOLUTION**	**FWD-FROM**
3-EXE	1-WB	FWD	1-MEM
3-EXE	2-WB	STALL	
5-EXE	4-WB	STALL	
6-EXE	5-WB	FWD	5-EXE
6-EXE	4-WB	FWD	4-MEM

b) Data hazard table after the STALL.

ID	Instruction	Pipeline Stages											
1	lw r1, r20, 0x2056	IF	ID	EXE	MEM	WB							
2	lw r2, r21, 0xF5C4		IF	ID	EXE	MEM	WB						
3	add r3, r1, r2			IF	ID	**0**	EXE	MEM	WB				
4	lw r4, r22, 0x0014				IF	**0**	ID	EXE	MEM	WB			
5	addi r8, r4, 0x1A					**0**	IF	ID	**0**	EXE	MEM	WB	
6	add r5, r8, r4							IF	**0**	ID	EXE	MEM	WB

Data Hazard (After Stall)			
STAGE	**DEPENDENCY**	**RESOLUTION**	**FWD-FROM**
3-EXE	1-WB	FWD	1-MEM
3-EXE	2-WB	FWD	2-MEM
5-EXE	4-WB	FWD	4-MEM
6-EXE	5-WB	FWD	5-EXE
6-EXE	4-WB	FWD	4-MEM

c) Minimal re-ordered code to avoid STALL would be as following.

ID	Instruction	Pipeline Stages											
1	lw r1, r20, 0x2056	IF	ID	EXE	MEM	WB							
2	lw r2, r21, 0xF5C4		IF	ID	EXE	MEM	WB						
4	**lw r4, r22, 0x0014**			IF	ID	EXE	MEM	WB					
3	add r3, r1, r2				IF	ID	EXE	MEM	WB				
5	addi r8, r4, 0x1A					IF	ID	EXE	MEM	WB			
6	add r5, r8, r4						IF	ID	EXE	MEM	WB		

The data hazard table would be as following.

Data Hazard (After Reorder)			
STAGE	**DEPENDENCY**	**RESOLUTION**	**FWD-FROM**
3-EXE	1-WB	FWD	1-MEM
3-EXE	2-WB	FWD	2-MEM
5-EXE	4-WB	FWD	4-MEM
6-EXE	5-WB	FWD	5-EXE
6-EXE	4-WB	FWD	4-MEM