MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Spring 2004

Quiz #1: February 20, 2004

Name	*Athena Username*	*Score*
Scott Young		24/25

☐ WF 11, 34-303 Ward	☐ WF 1, 34-302 Chong	☐ WF 1, 34-304 Suh
☐ WF 12, 34-303 Ward	☐ WF 1, 26-310 Arvind	☐ WF 2, 34-303 Suh
☐ WF 12, 26-210 Berger	☐ WF 2, 26-310 Arvind	

Problem 1. (5 points): **Warmup** 4 4/5

(A) The propagation delay specified for an inverter is less than its contamination delay

X Circle one: **NEVER** **SOMETIMES** **ALWAYS**

(B) A two-input CMOS NOR gate, constructed as shown in lecture, is *lenient*

✓ Circle one: **NEVER** **SOMETIMES** **ALWAYS**

(C) With appropriate choices of V_{OL}, V_{OH}, V_{IL}, and V_{IH} one can construct a combinational buffer that satisfies the static discipline (with nonzero noise margins) using only resistors and wires.

✓ Circle one: **NEVER** **SOMETIMES** **ALWAYS**

(D) You listen to three phone lines carrying data with varying degrees of compression. On one line you hear a series of melodic tones; on another you hear mostly a steady buzz; on the last you hear what sounds like random noise. Which line is likely to carry the most compressed (highest information content) bit stream?

✓ Circle one: **TONES** **BUZZ** **NOISE**

(E) I roll two 6-sided dice, and tell you that the rolls total 12. How many bits of information have I given you about the outcome?

Give an expression for the amount of information in the message: $\lg(36)$ **bits** ✓

Problem 2 (7 points) 2-of-3 Detector Circuit

Consider the Boolean function F(A, B, C) whose value is 0 if two or more of the inputs are 1, otherwise the value is 1.

A	B	C	F(A,B,C)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

(A) (2 points) Fill in the truth table for F to the left.

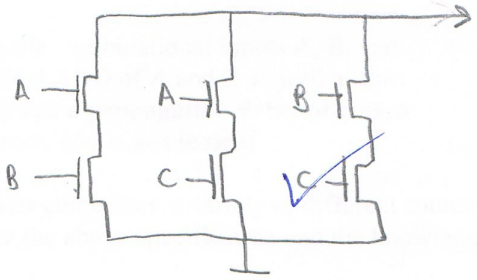
2/2 **(fill in table to left)**

(B) (2 points) Write a sum-of-products expression for F(A, B, C):

$\bar{A}\cdot\bar{B}\cdot\bar{C} + \bar{A}\cdot\bar{B}\cdot C + \bar{A}\cdot B\cdot\bar{C} \cdots$
$+ A\cdot\bar{B}\cdot\bar{C}$

2/2 **(give expression)**

(C) (3 points) Can F(A, B, C) be implemented as a single CMOS gate – that is, as a single output node connected to V_{DD} by a PFET pullup circuit and to GND by an NFET pulldown circuit? If so, diagram the **pulldown** circuit below; if not, explain why no such circuit is possible.



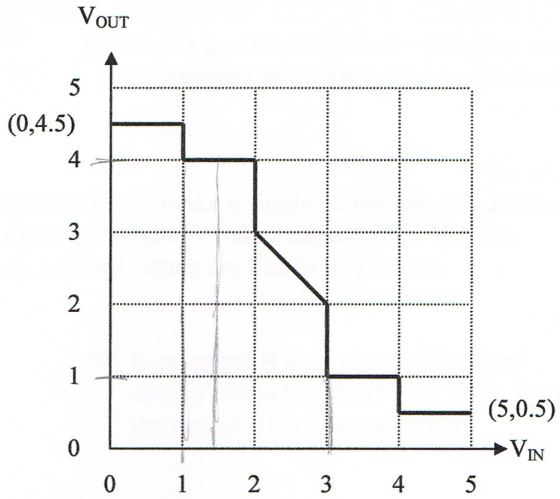
3/3

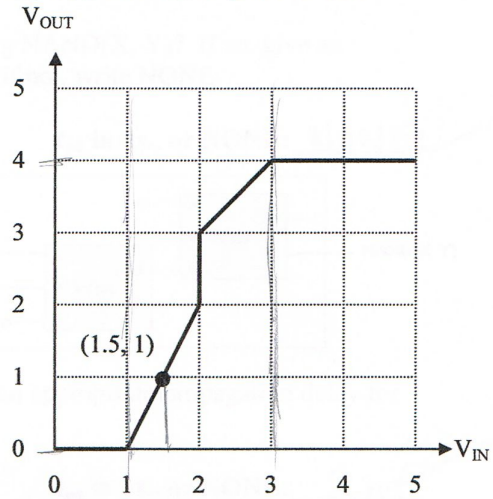
(pulldown circuit or explanation)

Problem 3 (6 points) Static Discipline

Following are voltage transfer curves of devices to be used in a new logic family as an inverter and buffer, respectively:

Your job is to choose logic representation parameters, V_{OL}, V_{IL}, V_{OH}, and V_{IH} to give the best noise margins you can using these devices. Fill in your answers below, together with the resulting noise margins. You'll get partial credit for anything that works with nonzero noise margins; for all six points, maximize each of the noise margins. [Extra copies of these diagrams are attached to this test for scratch purposes].





6/6

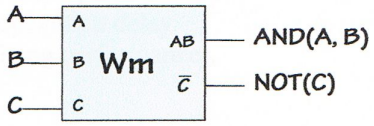
V_{OL} = 1 ; V_{IL} = 1.5 ; V_{IH} = 3 ; V_{OH} = 4

Low Noise Margin = 0.5 ; **High Noise Margin** = 1

Problem 4 (7 points) William Gates

Combinational Salvage Co has acquired a large number of *William* gates, which are 3-input combinational devices that work as follows:

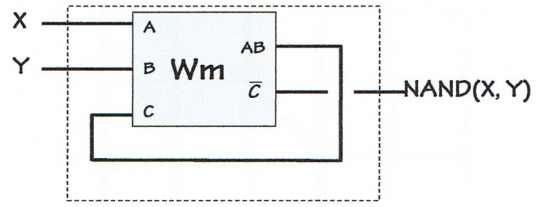
Each William gate takes the combinational inputs A, B, and C, and computes the logical AND of A and B as well as the inverse of C. The device has **a propagation delay of 1 ns**, a contamination delay of zero, and is **not lenient**.



Note: These William gates come from a variety of different sources, and their internals vary. Your only guarantees are the above specification and the knowledge that they are Combinational Devices.

Unfortunately, CSC has discovered a huge market for 2-input NAND gates, but none for inverters or ANDs. They ask their engineers to explore making devices that compute NAND from their supply of William gates. They get back three proposals, and have asked you to evaluate each.

Problem 4 (continued)

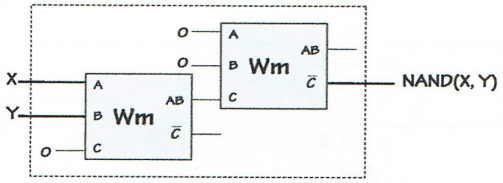


Proposal A is simply to make each William gate into a 2-input NAND by feeding its AB output back into its C input.

(A) Is this a valid combinational device computing NAND(X, Y)? If so, give an appropriate propagation delay for the device; if not, write NONE.

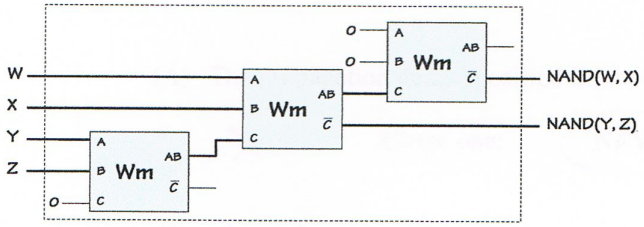
t_{pd} in ns., or NONE: NONE

Proposal B is to make a single combinational device computing 2-input NAND from a pair of William gates. Unused inputs are tied to 0 (ground).



(B) Is proposal B a valid combinational device computing NAND(X, Y)? If so, give an appropriate propagation delay for the device; if not, write NONE.

t_{pd} in ns., or NONE: 2 ns



Proposal C is to make a single combinational device that computes NANDS of *two pairs* of input signals, using a total of 3 William gates. Again, unused inputs are grounded.

(C) Is proposal C a valid combinational device computing NANDs of the four inputs? If so, give an appropriate propagation delay for the device; if not, write NONE.

t_{pd} in ns., or NONE: 3 ns

CSC decides to develop two compatible products, code-named Speedo and Cheapo. Each is a 10-input, 5-output combinational device that computes NANDs of **five** independent **pairs** of inputs. Each is built entirely from William gates, but Speedo has the minimum propagation delay possible using these devices, while Cheapo uses the fewest devices (sacrificing performance). Neither is claimed to be lenient.

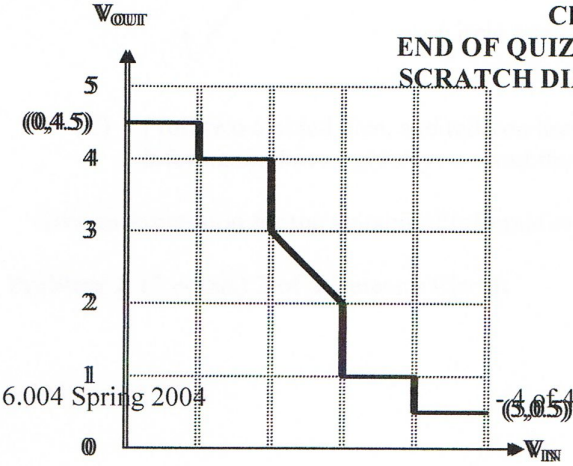
(D) Give an appropriate gate count and t_{pd} for each device:

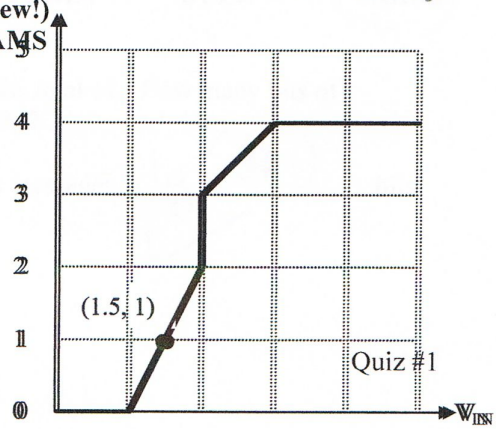
Speedo: 10 **William gates; tpd =** 2 ns

Cheapo: 6 **William gates; tpd =** 6 ns

END OF QUIZ 1 (Phew!)

SCRATCH DIAGRAMS





MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Spring 2004

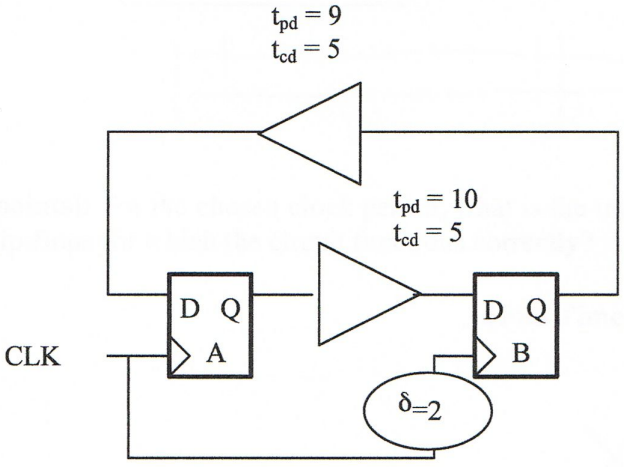
Quiz #2: March 12, 2004

Name	*Athena Username*	*Score*
SCOTT YOUNG		12/25

☐ WF 11, 34-303 Ward	☐ WF 1, 34-302 Chong	☐ WF 1, 34-304 Suh
☐ WF 12, 34-303 Ward	☐ WF 1, 26-310 Arvind	☐ WF 2, 34-303 Suh
☐ WF 12, 26-210 Berger	☐ WF 2, 26-310 Arvind	

Problem 1. [6 points] Dynamic Discipline

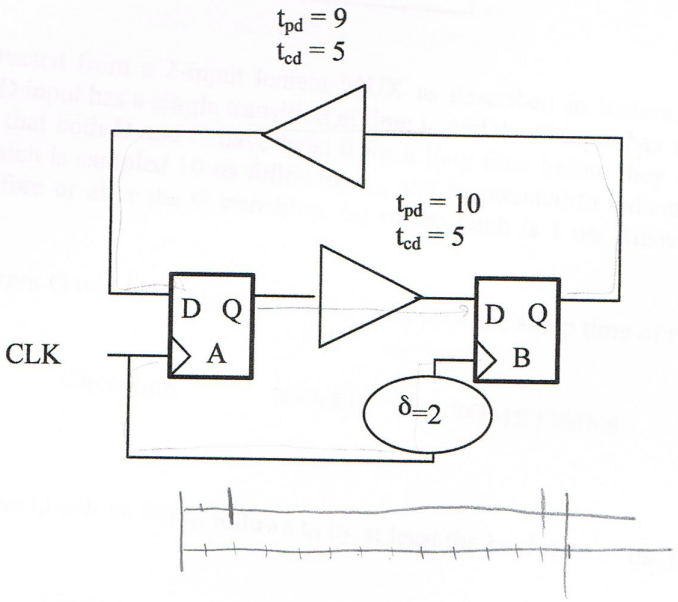
Consider the sequential logic circuit shown below. The memory elements are all rising edge-triggered flip-flops. The propagation and contamination delays of the gates are marked on the figure. The two flip-flops are identical. **We wish to clock this circuit at 13 ns.** You can assume that the clock has zero rise and fall time, and that the flip-flops have zero propagation and contamination delays. The delay on the clock line results in a skew of 2 ns, i.e., the clock arrives at flip-flop B 2 ns after it arrives at flip-flop A. **Show your work if you wish to receive partial credit.**



1(a). [3 points]: For the chosen clock period, what is the maximum setup time for the flip-flops for which the circuit functions correctly? As stated above, you can assume that the two flip-flops have identical setup times.

Setup Time t_s: 1ns

X



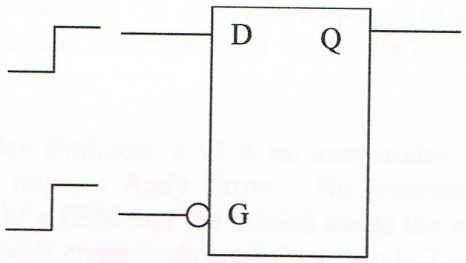
1(b). [3 points]: For the chosen clock period, what is the maximum hold time for the flip-flops for which the circuit functions correctly?

Hold Time t_h: 5ns

X

Problem 2. [4 points] Metastability

The following circuit is proposed for sampling an asynchronous input (e.g., from a pressed button) by a synchronous clocked device:



The latch is constructed from a 2-input lenient MUX as described in lecture; it is transparent when G is low. Its D input has a **single** transition at time t_D and the G input has a **single** transition at time t_G. Assume that both D and G have been 0 for a long time before they make transitions. The output of the latch is sampled 10 ns following t_G, and is intended to indicate whether the D transition comes before or after the G transition. t_{PD} of the latch is 1 ns. Answer the following questions:

2(a). The output Q will be 1 if t_D precedes t_G by at least the setup time of the latch.

Circle one: **NEVER** **SOMETIMES** **ALWAYS**

2(b). The output Q will be 0 if t_D follows t_G by at least the hold time of the latch.

Circle one: **NEVER** **SOMETIMES** **ALWAYS**

2(c). The output will be either a logical 0 or a logical 1 if t_D and t_G are sufficiently close in time.

Circle one: **NEVER** **SOMETIMES** **ALWAYS**

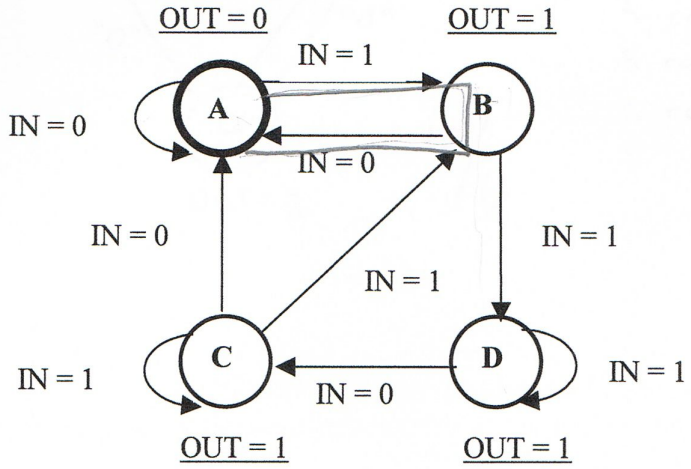
2(d). There exists a constant d such that regardless of the relative timing of t_D and t_G the output Q will be a valid 0 or a valid 1 at time $t_G + d$.

Circle one: **TRUE** **FALSE**

Problem 3. [8 points] Finite State Machines

Tintel CEO Andy "Treasure" Trove decides that in order to make more money Tintel must market a bit-serial sequential logic design. The specifications for a finite state machine threshold detector are as follows. The FSM receives a bit on its single input every clock cycle, and produces a 1 at its single output, if **at least 1 of the last 2 bits** it received are 1's. Assume that when the FSM begins its operation, it thinks that it has received only 0's so far.

3(a). [2 points] Ben Bitdiddle, a VI-A summer student at Tintel, figures that this is his chance to impress Andy Trove. He immediately cooks up the State Transition Graph of a FSM that (he thinks) meets the specifications, and sends it to Andy Trove. Ben's graph is shown below. Andy Trove takes a quick look and says, "That doesn't work! Didn't you take 6.004 at MIT?" Give an input sequence that causes Ben's FSM to fail specifications.

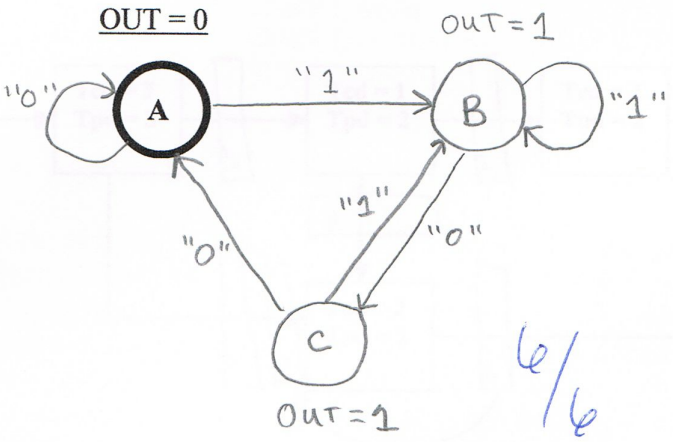


Input sequence that causes Ben's FSM to Fail:

0010 ✓

3(b). [6 points] Alyssa Hacker, who is sitting right next to Ben, jumps up and shows Andy her FSM. Andy gives it a quick look and says, "All right, at least we have one summer intern who knows what she's doing!".

Fill in Alyssa's machine below. You will get 4 points for any machine that works, and 6 points for a minimum-state machine.

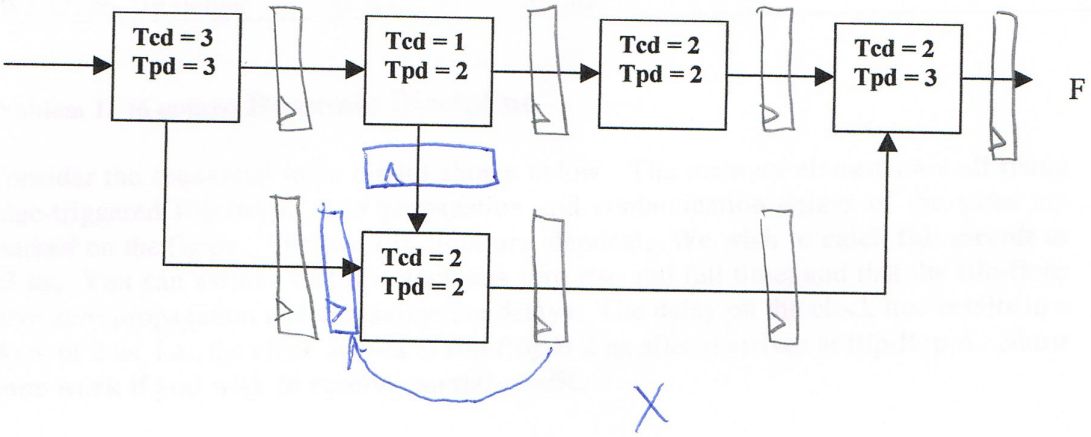


A represents last two = 0
B represents last = 1
C represents last two = 1

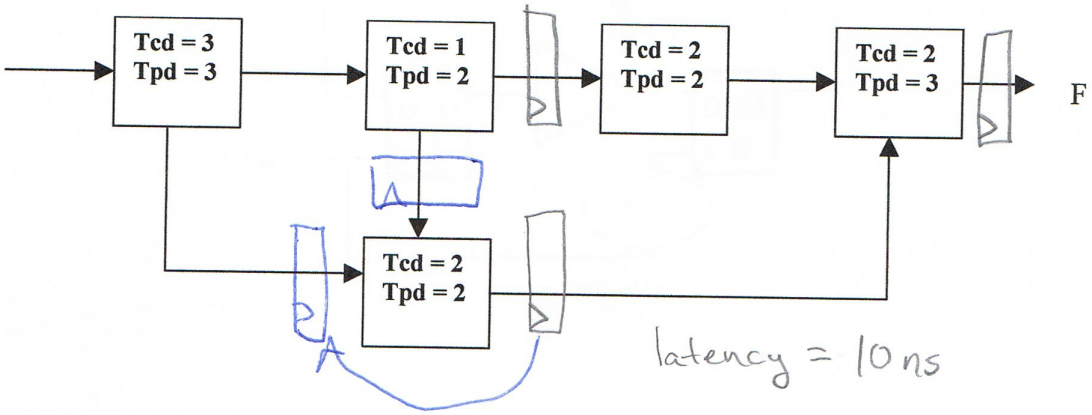
6/6

Problem 4. [7 points] Pipelining

4(a) [4 points] Assuming ideal registers, with setup and hold times equal to zero, and propagation delay equal to zero, pipeline the circuit below for maximum throughput. Remember our convention dictates a register at the output!



4(b) [3 points] Assume the same circuit (duplicated below) is pipelined for a throughput of 1/5 using the minimum number of registers. What is the latency?



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Spring 2004

Quiz #3: April 9, 2004

Name	*Athena Username*	*Score*
Scott Young		16/25

☐ WF 11, 34-303 Ward	☐ WF 1, 34-302 Chong	☐ WF 2, 34-304 Suh
☐ WF 12, 34-303 Ward	☐ WF 1, 26-310 Arvind	☐ WF 3, 34-303 Suh
☐ WF 12, 26-210 Berger	☐ WF 2, 26-310 Arvind	

Problem 1. (5 points): **CS Thesis Topics**

4/5

You are considering a number of suggestions for your MEng thesis topic, and are anxious to avoid embarrassing yourself by proposing to implement a device whose implementation is, in general, impossible. For each of the following, circle **OK** if the device can in theory be built (independently of whether you know how to do it right now) or **DUD** if it cannot.

A. An FSM that determines whether a given binary number, entered in bit-serial form, is divisible by 3.

Circle one: **OK** **DUD** ✓

B. A "Universal FSM": an FSM which takes (in bit-serial form) the description of another arbitrary FSM and emulates the behavior of the FSM described by that input.

Circle one: **OK** **DUD** ✓

C. A Turing Machine that halts if and only if there are fewer than one thousand "1"s written on its input tape.

Circle one: **OK** **DUD** ✓

D. A Turing Machine whose input tape contains integers i, j, and k, and which reliably determines whether the i^{th} TM operating on the j^{th} tape will halt within k steps.

Circle one: **OK** **DUD** X

E. A Turing Machine whose input tape contains the integer i and reliably determines whether the i^{th} TM operating on a tape containing the binary representation of "6004" halts.

Circle one: **OK** **DUD** ✓

Problem 2. [5 points] β Coding

Gill Bates, CEO of BetaSoft, has called you in to put the finishing touches on his new "Shorthorn" release of BS Windows. The code is complete except for the translation of one scrap of C code, which has to be hand translated because of a recently discovered bug in the compiler's translation of array indexing. The relevant C code is:

```
int A[1000];
    …
    for (i=0; i<1000; i = i+1)
      {
        A[i] = A[i] * 0x12345;
      }
    …
```

Following is an unfinished translation of the above code to Beta assembly language. You are to complete this code, by adding lines to implement the assignment statement in the body of the for loop:

```
| Translation to Beta assembly:
. = 0x1000                          | Start at hex 1000
A:                                  | Here's the array
. = .+4000                          | reserve 1000 ints
    …                               | Other stuff here
    ADD(R31, R31, R2)               | Start of for loop
L1: CMPLTC(R2, 1000, R3)
    BEQ(R3, L2)

    | Body of for loop (finish this!)
    MULC(R2, 4, R3)
    LD(R3, 0x1000, R4)
    MULC(R4, 0x12345, R4)   <- can't use 32 bits for MUL
    ST(R3, 0x1000, R4)

                                                    3/5

    ADDC(R2, 1, R2)
    BR(L1)

| Static stuff can be added here

L2: …                               | Other stuff here
```

(Complete above program)

Problem 3. (15 points): **Digging into Beta code**

You are given the following incomplete listing of a C procedure and its translation to Beta assembly code on the left:

```
int f(int a, int b)
 {
   if (a < b)
      return f(a+a, b+1);
   else return ????;
 }
```

```
f:   PUSH(LP)
     PUSH(BP)
     MOVE(SP, BP)

     PUSH(R1)
     PUSH(R2)

     LD(BP, -12, R0)   R0 = a
     LD(BP, -16, R1)   R1 = b
     CMPLT(R0, R1, R2)
xx:  BEQ(R2, L1)

     ADDC(R1, 1, R1)
     PUSH(R1)
     ADD(R0, R0, R0)
     PUSH(R0)

     BR(f, LP)
     SUBC(SP, 8, SP)

L2:  POP(R2)
     POP(R1)
     MOVE(BP, SP)
     POP(BP)
     POP(LP)
     JMP(LP)

L1:  SUB(R0, R1, R0)   a-b = R0
     BR(L2)
```

Note: while working this problem, you may wish to refer to the reference information (instruction set summary) attached to this quiz.

(A) (2 points) Give the HEX value of the instruction labeled '**L1:**' in the above program

Hex encoding of **SUB(R0, R1, R0)**: 0x C4010000 (1/2) ← out of order

(B) (2 points) Give the HEX value of the BR instruction at the end of the above program. Recall that the BR macro translates this unconditional branch to a BEQ instruction.

Hex encoding of **BR(L2)**: 0x 77FFFFE4 (1/2) FF ; -28 in 2's comp

(C) (1 point) What is the missing C expression corresponding to the **????** in the above C program?

else return (a - b);

(give C source fragment)

(D) (1 point) Suppose the instruction MOVE(BP, SP) were deleted from the above program. What would be the result? Mark exactly one answer:

The procedure would still work just fine: ✓

The procedure would compute the proper value, but not restore registers properly: _____

The procedure would no longer compute f(a,b) properly: _____

4/6

The call f(2, 5) is made via the instruction **`BR(f,LP)`** from an external main program and its execution is interrupted just prior to an execution (not necessarily the first) of the **`BEQ`** instruction labeled **xx:**. The contents of a region of memory are shown to the right.

NB: All addresses and data values are shown in hex. The contents of **BP** are **0x128.**

Address (HEX)	Contents (HEX)
100	5
104	2
108	AB
10C	0
110	0
114	6004
118	6
11C	4
120	54
124	110
BP → 128	6
12C	1

(E) (2 points) What are the arguments to the ***current*** (most recent) call to f?

Current arguments, a=4c; b=6

(F) (1 point) What value is in **SP**?

Contents of **SP** (HEX): 0x12F x
130

(G) (2 points) What is the address of the **`BR(f, LP)`** instruction that made the original call to `f(2,5)`?

Address of BR making original call:0xAB 1/2 (forgot to subt 4)

(H) (2 points) What value was in R2 at the time of the original call?

Original contents R2 (HEX): 0x6004 ✓

(I) (2 points) What is the hex address of the instruction tagged '**`L2:`**'?

Address of '**`L2:`**' (HEX): 0x118
58 x
5/9

End of Quiz 3

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Spring 2004

Quiz #4: April 23, 2004

Name	Athena Username	Score
		13/25

☐ WF 11, 34-303 Ward	☐ WF 1, 34-302 Chong	☐ WF 2, 34-304 Suh
☐ WF 12, 34-303 Ward	☐ WF 1, 26-310 Arvind	☐ WF 3, 34-303 Suh
☐ WF 12, 26-210 Berger	☐ WF 2, 26-310 Arvind	

Problem 1 [7 points]: BGE Instruction

3/7

Adrian Labstar implements a "compare and branch" instruction in the unpipelined Beta. This instruction **BGE(ra, label, rc)** branches to the instruction corresponding to **label** if **Reg[ra] >= Reg[rc]**, and does not branch if **Reg[ra] < Reg[rc]**. He uses the ALU to compute **Reg[ra] – Reg[rc]**, and adds logic which checks if the output of the ALU is >= 0. Armed with this new instruction, he starts to simplify the assembly code for the new Beta.

(A) [2 points]: Simplify the code below using the BGE instruction. Note that the value of r3 is not used except in the branch decision.

2/2

```
        CMPLE(r1, r2, r3)
        BNE(r3, done, r31)
        SUBC(r1, 1, r1)
done:   ADD(r1, r2, r3)
```

r1 <= R2 ? → R3
R3 ≠ 0 → done
SUBC(R1, 1, R1)
done: ADD(R1, R2) → R3

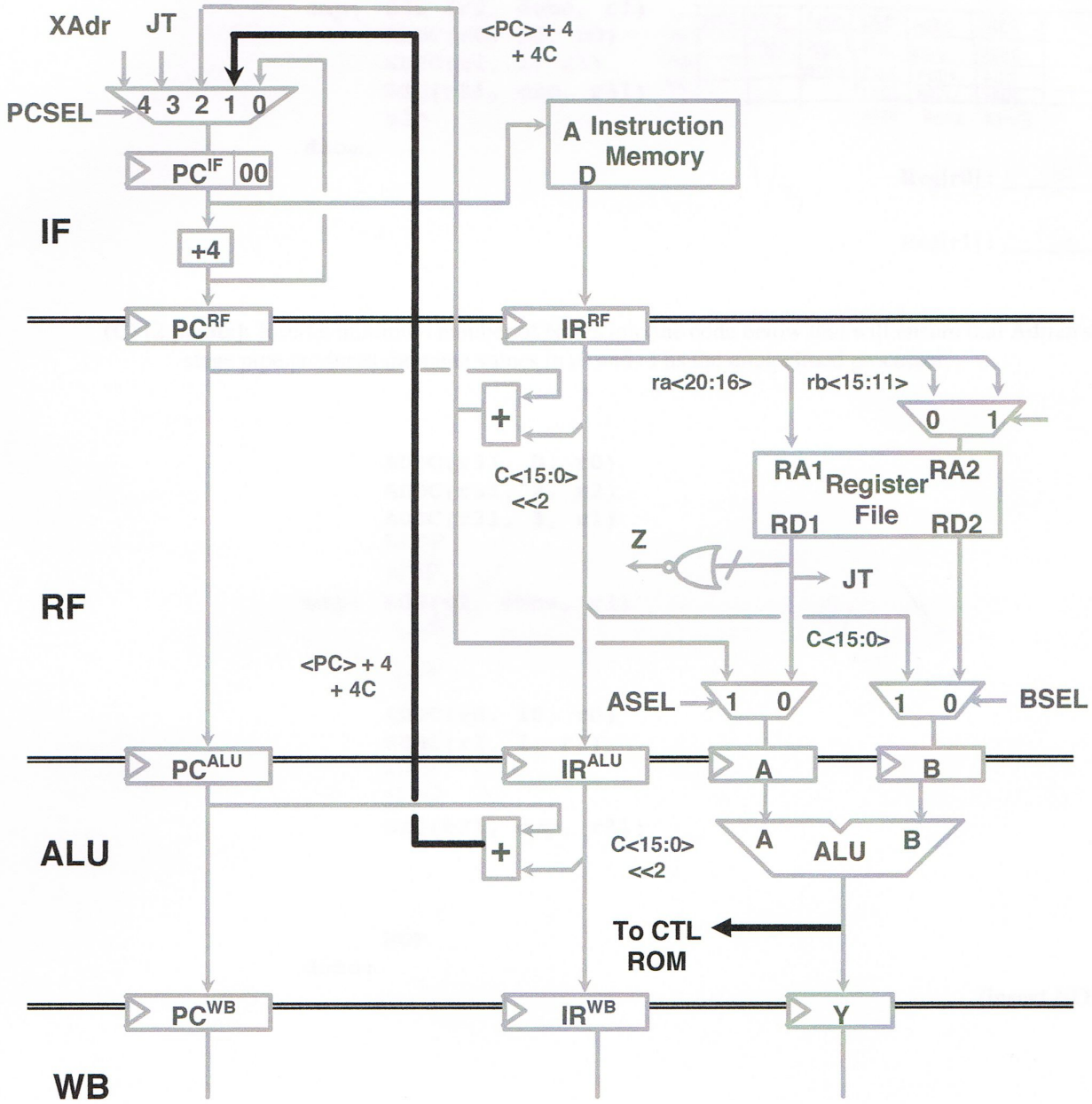
Branch if r1 > R2, otherwise don't

Write your simplified code below:

```
BGE(r1, done, r2)
SUBC(r1, 1, r1)
done: ADD(r1, r2, r3)
```

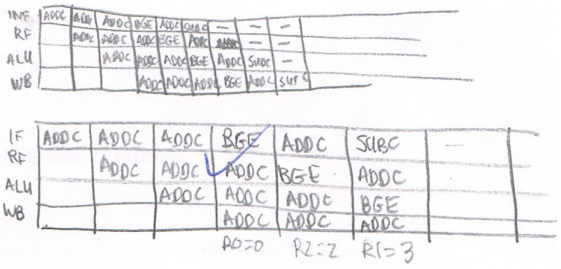
(Simplified Code)

Adrian then creates a 4-stage pipeline shown below.. This pipelined processor includes the BGE instruction, and adds an additional 1-bit input to the control ROM from the output of the ALU to support the branch decision in the BGE instruction. He decides to implement full bypassing as on the Beta pipeline (not shown), but chooses to deal with all types of branch/jump hazards in software.



(B) [3 points]: What are the results stored in r0 and r1 when the assembly code below is run on Adrian's 4-stage pipeline with full bypassing but no branch annulment?

```
        ADDC(r31, 0, r0)
        ADDC(r31, 2, r2)
        ADDC(r31, 3, r1)
cmp:    BGE (r2, done, r1)
        ADDC(r0, 10, r0)
        SUBC(r1, 1, r1)
        BEQ(r31, cmp, r31)
        NOP
done:
```



1/3

Reg[r0] : 10 X

Reg[r1] : 2 X

(C) [2 points]: Insert a minimum number of NOPs into the code below that will ensure that Adrian's 4-stage pipe produces the same values in r0 and r1 as the unpipelined processor.

```
        ADDC(r31, 0, r0)
        ADDC(r31, 2, r2)
        ADDC(r31, 3, r1)
        NOP
        NOP
cmp:    BGE(r2, done, r1)
        NOP
        NOP
        ADDC(r0, 10, r0)
        SUBC(r1, 1, r1)
        NOP
        NOP
        BEQ(r31, cmp, r31)


        NOP
done:
```

0/2

(insert NOPs)

Problem 2 [6 points]: Beta control signals

Marketing has asked for the following instructions to be added to an Extended Beta instruction set, for implementation on an *unpipelined* Beta.

```
SWAPR(Rx, Ry)                    // Swap register contents
  TMP ← Reg[Rx]
  Reg[Rx] ← Reg[Ry]
  Reg[Ry] ← TMP
  PC ← PC + 4

STR( Rx, C )                     // Store indexed
  EA ← PC+4+4*SEXT(C)
  Mem[EA] ← Reg[Rx]
  PC ← PC + 4
```

The Marketing people don't care about details of instruction coding (e.g., which fields are used to encode Rx and Ry in the above descriptions), but want to know which if any of the above can be implemented as a **single instruction** in the **existing unpipelined Beta simply by changing the control ROM**.

Your job is to decide which of the above instructions can be implemented on the existing Beta, making appropriate choices for Rx and Ry, and to specify control signals that implement those instructions.

You may wish to refer to the Beta diagram included among the reference material at the end of this quiz.

For each instruction either fill in the appropriate values for the control signals in the table below or put a line through the whole row if the instruction cannot be implemented using the existing unpipelined Beta datapath. Use "—" to indicate a "don't care" value for a control signal.

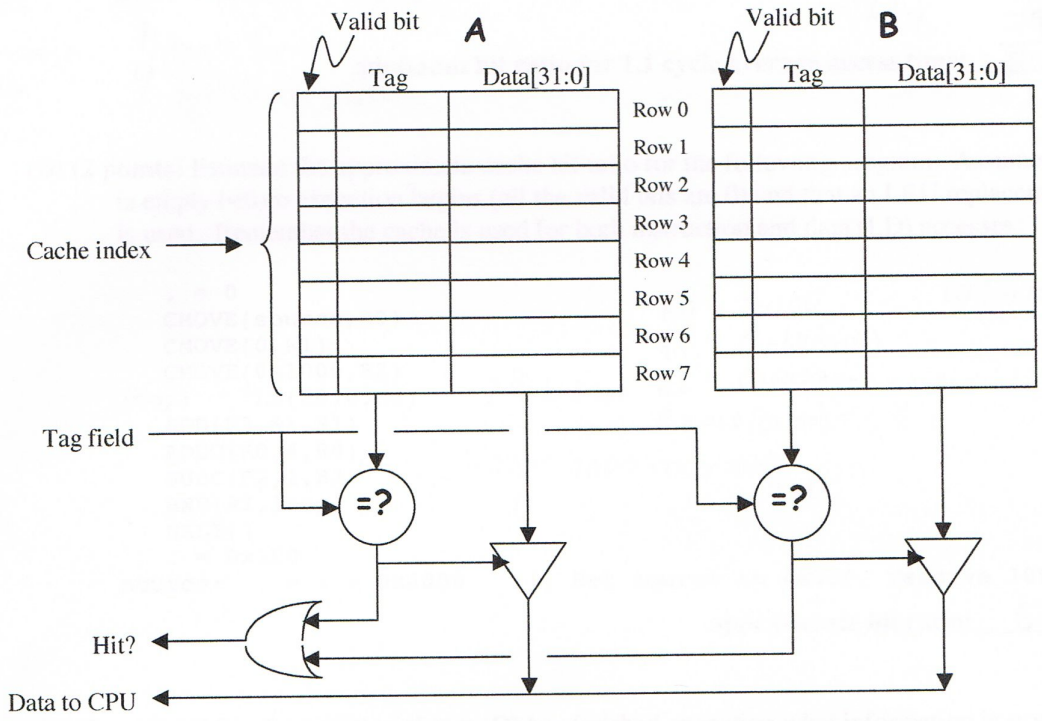
Instr	ALUFN	WERF	BSEL	WDSEL	WR	RA2SEL	PCSEL	ASEL	WASEL
SWAPR	✓	✓	✓	✓	✓	✓	✓	✓	✓
STR	ADD ✓	0 ✓	0 ✓	0 ✗	1 ✓	0 ✗	000	1 ✓	0

6/6 (grader says to round up 5⅔ to 6)

3/10

Problem 3. Caches [10 points]

Consider the following diagram for a 2-way set associative cache to be used with our Beta design:



Each cache line holds a single 32-bit word of data along with its associated tag and valid bit (0 when the cache line is invalid, 1 when the cache line is valid).

(A) (2 points) The Beta produces 32-bit byte addresses, A[31:0]. To ensure the best cache performance, which address bits should be used for the cache index? For the tag field?

address bits used for cache index: A[4 **:** 2 **]**

address bits used for tag field: A[31 **:** 5 **]**

0/2 X

(B) (2 points) Suppose the Beta does a read of location 0x5678. Identify which cache location(s) would be checked to see if that location is in the cache – for each location specify the cache section (A or B) and row number (0 through 7) –e.g., **3A** for row 3, section A. If there is a cache hit on this access what would be the contents of the tag data for the cache line that holds the data for this location?

8
1000

1/2 CARRY FWD A6 B6

cache location(s) checked on access to 0x5678: A Row 0 , B Row 0

cache tag data on hit for location 0x5678 (hex): 0x 5678 X

8 = 7.9/8 = HR

(C) (2 points) Assume that checking the cache on each read takes 1 cycle and that refilling the cache on a miss takes an *additional* 8 cycles. If we wanted the *average* access time over many reads to be 1.1 cycles, what is the minimum hit ratio the cache must achieve during that period of time? You needn't simplify your answer.

1.1 = 9 - 9HR + 1HR
= 9 - 8HR

1.1 = 9(1-HR) + 1HR

1 cycle for hit
9 cycles for miss

minimum hit ratio for 1.1 cycle average access time: 2/2 79/80 ✓

(D) (2 points) Estimate the approximate cache hit ratio for the following program. Assume the cache is empty before execution begins (all the valid bits are 0) and that an LRU replacement strategy is used. Remember the cache is used for both instruction and data (LD) accesses.

```
    . = 0
    CMOVE(source,R0)
    CMOVE(0,R1)
    CMOVE(0x1000,R2)
loop: →LD(R0,0,R3)
    ADD(R3,R1,R1)
    ADDC(R0,4,R0)
    SUBC(R2,1,R2)
    BNE(R2,loop)
    HALT()
    . = 0x100
source:  . = . + 0x4000    || Set source to 0x100, reserve 1000 words
```

R0 = 0x100 LD(0x100)
R1 = 0 + LD(0x100)
$2^{12} \doteq 5000$ times R2 = 0x1000 --
R3 = LD(0x100)
0001 0000 0000 0000 loops
(4 16 16

approximate hit ratio: 50% X 5/6

(E) (2 points) After the program of part (D) has finished execution what information is stored in row 4 of the cache? Give the **addresses** for the two locations that are cached (one in each of the sections) or briefly explain why that information can't be determined.

Addresses whose data is cached in "Row 4": 0x_____ and 0x_____

Can't be determined because data can be stored in either A, or B.

Problem 4 (2 points): Diabolical pipelining

This is a tricky, classic pipelining problem worth a measly two points. We recommend you not spend time on it unless you are confident of your answers to other questions.

The following instruction sequence is executed again on the pipelined processor of Problem 1 (a 4-stage pipelined Beta with bypass paths but no branch delay slot annulment, leaving branch delay slots to be handled in software):

```
        CMOVE(3, R0)              | Load 3 into R0
        CMOVE(4, R1)              | Load 4 into R1
        NOP()                     | wait, to stabilize registers
        NOP()
        BR(X)
        BR(Y)
Y:      ADDC(R0, 1, R0)           | Increment R0 contents by 1
        NOP()                     | wait, to stabilize registers
        NOP()

X:      SUBC(R1, 1, R1)           | Decrement R1 contents by 1
        NOP()                     | wait, to stabilize registers
        NOP()
        HALT ()
```

Your challenge is to figure out the final values in R0 and R1. You may want to use the scratch pipeline timing diagrams attached at the end of this exam.

1/2

Final R0 Value: 4 ✓

Final R1 Value: 4 2 ×

End of Quiz 4

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Spring 2009

Quiz #5: May 8, 2009

Name SCOTT YOUNG	*Athena login name*	*Score* 13/25

NOTE: Reference material and scratch diagrams appear on the backs of quiz pages.

Problem 1 (2 points): **I should have stayed awake during that lecture…**

(A) (1 point) A trend in modern computers is to replace parallel, shared backplane buses with

1) Hypercube networks
2) Serial, point-to-point switched connections
3) Wireless networks
4) Ribbon cables
5) Coaxial cables
6) None of the above

X

Choose best answer (1 thru 6): 4

(B) (1 point) If we take into consideration physical realities like speed-of-light delays and minimum node size, the asymptotic worst-case latency between nodes of an N-node binary tree network is:

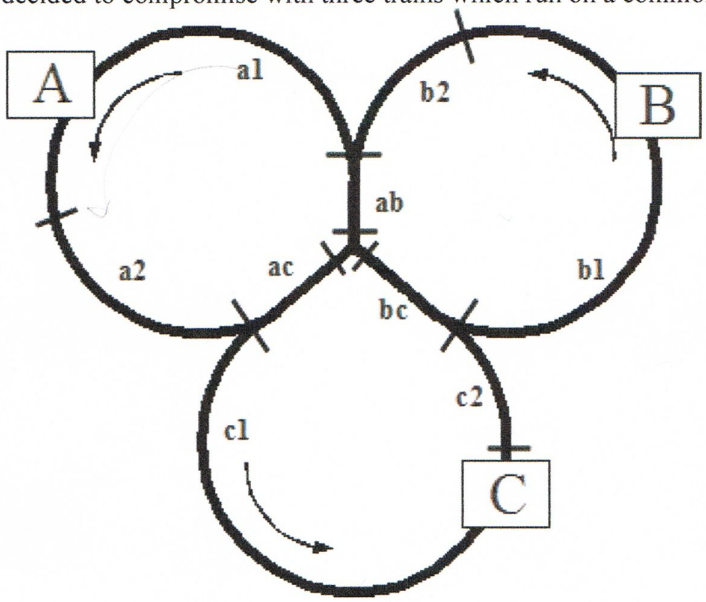
1) $O(\log(N))$
2) $O(\sqrt[3]{N})$
3) $O(\sqrt[2]{N})$
4) $O(N)$
5) $O(N^2)$
6) $O(N^3)$
7) None of the above

X

Choose best answer (1 thru 7): 1

Problem 2 (10 points): Process Synchronization

Gill Bates, who dropped out of MIT to found the fledgling Megahard Corporation, has decided to buy electric trains for his three kids: Alice, Bobby, and Chip. Gill is strapped for cash, due to the startup; he can't afford to buy a complete setup for each child. He vows that after Megahard makes him a billionaire, he will buy each kid a complete set of ***real*** trains. But for now, he has decided to compromise with three trains which run on a common track layout shaped as follows:



With this setup, each of Alice, Bobby, and Chip have their own trains, and each train travels in a circular path in the indicated (counterclockwise) direction. Note that there is a section of track shared between each pair of trains. In order to avoid sibling battles due to train wrecks, Gill has devised a system for segmenting the track, and has programmed each train to use a semaphore to avoid simultaneous use of the shared track sections **ac**, **ab**, and **bc**.

Gill's code is as follows:

Shared Memory:

```
semaphore S=???;
```

Process A:

```
while (ARun)
 { TravelTo(a2);
   wait(S);
   TravelTo(a1);
   signal(S);
 }
```

Process B:

```
while (BRun)
 { TravelTo(b2);
   wait(S);
   TravelTo(b1);
   signal(S);
 }
```

Process C:

```
while (CRun)
 { TravelTo(c2);
   wait(S);
   TravelTo(c1);
   signal(S);
 }
```

Note that **`TravelTo(x)`** moves the train forward until the train is entirely enclosed in the track segment labled **`x`**. Trains A, B, and C start out in segments **a1**, **b1**, and **c1** respectively.

(A) (1 point) What should the initial value of the semaphore S be?

Initial value for S: 1

Alice, a precocious 4-year-old, keeps asking her daddy why her train (marked "A") waits while B is traveling from segment **bc** to **b1** and C is using segment **c1**, none of which are used by A.

(B) (1 point) Choose the best explanation of the problem cited by Alice.

E1: There's a deadlock.

E2: Some unenforced essential precedence constraint.

E3: Some nonessential precedence constraint enforced.

E4: Some **`wait`** without a corresponding **`signal`**.

E5: There's no problem, tell Alice to shut up and go to bed.

Give number of best explanation: E3

After some deliberation, Gill modifies the code in the trains as follows:

Shared Memory:

```
semaphore Sab=1, Sbc=1, Sac=1;
```

Process A:

```
while (ARun)
 { TravelTo(a2);
   wait(Sac);
   TravelTo(ac);
   wait(Sab);
   TravelTo(ab);
   signal(Sac);
   TravelTo(a1);
   signal(Sab);
 }
```

Process B:

```
while (BRun)
 { TravelTo(b2);
   wait(Sab);
   TravelTo(ab);
   wait(Sbc);
   TravelTo(bc);
   signal(Sab);
   TravelTo(b1);
   signal(Sbc);
 }
```

Process C:

```
while (CRun)
 { TravelTo(c2);
   wait(Sbc);
   TravelTo(bc);
   wait(Sac);
   TravelTo(ac);
   signal(Sbc);
   TravelTo(c1);
   signal(Sac);
 }
```

Alice and Chip try the new setup; Bobby is busy painting his train with peanut butter and doesn't run it during this test. The A and C trains run flawlessly for hours. Gill wonders which combinations of TravelTo operations within processes A and C are prohibited by his semaphores.

(C) (2 points) Which of the following operations might process C execute while process A is executing **`TravelTo(ac)`**? Circle YES or NO for each case.

Can C be executing `TravelTo(bc)`? YES ... NO

Can C be executing `TravelTo(ac)`? YES ... NO

Can C be executing `TravelTo(c1)`? YES ... NO

Bobby returns and adds his train to the setup; after a few minutes all three trains stop permanently. Gill investigates the system, examining the state of each process.

(D) (1 point) Which line of code does he find Process A executing?

wait(Sab) **(give line of code)**

(E) (1 point) On which track segment has train C stopped?

Indicate segement at which C has stopped: bc

(F) (2 points) What values are in each of the semaphores?

Values in Sab: 0 ; **Sac:** 0 ; **Sbc:** 0

Meanwhile Chip, an 11-month old prodigy, insists on putting his train on the track so that it travels in a clockwise direction (while the other trains travel counterclockwise). After unsuccessfully arguing with Chip, Gill finally changes the code in Process C to reflect the change in the direction of Chip's train. To Gill's amazement, the trains now run perfectly. Chip smiles with great satisfaction at his fix. Unfortunately, he can't explain it to Gill since he hasn't learned to talk yet.

(G) (1 point) Gill's modified process C code still contains two **`wait`** operations followed by two **`signal`** operations. What are the arguments to the wait calls in the new code?

First call: wait(Sac**)**

Second call: wait(Sbc**)**

(H) (1 point) Choose the best generalization of Chip's fix as a rule for allocating multiple resources in a multiprocess system.

R1: Never use more than one semaphore in a multiprocess system.

R2: Make each process allocate required resources in a global, prescribed order.

R3: Always release resources in the opposite order from that with which they were allocated.

R4: Never make more than two trains take counterclockwise paths.

Give number of best generalization: R3

Problem 3 (8 points): Pipelined Beta

This problem concerns the 5-stage Beta pipeline described in Lecture (a diagram of which can be found on back of page 1). This Beta has full bypass and annulment logic.

Consider the execution of the following sequence in kernel mode on the 5-stage pipelined Beta. The loop sums the first 100 elements of the integer array X and stores the result in Y.

```
        ADDC(R31,400,R1)    | index = 100 * 4
        XORC(R31,0,R2)      | clear sum
A:      LD(R1,X-4,R3)       | load next array element
        ADD(R3,R2,R2)       | add it to sum
        SUBC(R1,4,R1)       | decrement index
        BNE(R1,A,R31)       | loop unless index is zero
        ST(R2,Y,R31)        | store result
        ...
```

400

(A) (5 Points) Fill the blank boxes in the pipeline diagram below by writing the appropriate instruction opcode (ADDC, XORC, LD, …) in each box, showing the first 12 clock cycles of execution. Use the opcode NOP to indicate what happens during pipeline stalls or branch delay slot annulments. There are scratch copies of the diagram on the back of the previous page.

2/5

Cycle	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
IF	LD	NOP	ADD	SUBC	NOP	BNE	ST	NOP	LD	NOP
RF	XORC	LD	NOP	ADD	SUBC	NOP	BNE	NOP	NOP	LD
ALU	ADDC	XORC	LD	NOP	ADD	SUBC	NOP	BNE	NOP	NOP
MEM		ADDC	XORC	LD	NOP	ADD	SUBC	NOP	BNE	NOP
WB			ADDC	XORC	LD	NOP	ADD	SUBC	NOP	BNE

(B) (3 Points) Add arrows to your pipeline diagram above to indicate any active bypass paths for each clock cycle. The arrow should point *from* the stage where the bypassed value comes from, *to* the stage where the bypassed value is used. For example, in Cycle 4 there's an upward arrow pointing from the ADDC opcode in the MEM stage to the LD opcode in the RF stage .

1/3

Problem 4 (5 points): Broken pipeline

You've been given a 5-stage pipelined Beta processor as shown in lecture, whose diagram can be found on the back of page 1. Unfortunately, the Beta you've been given is defective: it has no bypass paths, annulment of instructions in branch delay slots, or pipeline stalls.

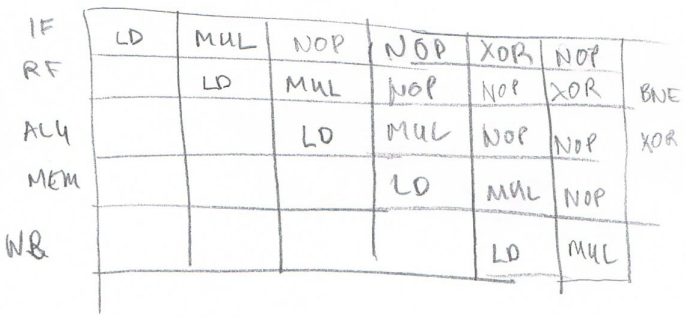
```
        ...
        NOP() NOP() NOP() NOP()
Loop:
        LD(R0, 0, R1)
        MUL(R2, R3, R4)
AA:
        XOR(R1, R0, R0)
BB:
        BNE(R4, LOOP, R0)
CC:
        XOR(R1, R2, R6)
        NOP() NOP() NOP() NOP()
        ...
```

You undertake to convert some existing code, designed to run on an unpipelined Beta, to run on your defective pipelined processor. The scrap of code on the left is a sample of the program to be converted. It doesn't make much sense to you – it doesn't to us either – but you are to add the **minimum** number of **NOP** instructions at the various tagged points in this code to make it give the same results on your defective pipelined Beta as it gives on a normal, unpipelined Beta.

Note that the code scrap begins and ends with sequences of **NOPs**; thus you don't need to worry about pipeline hazards involving interactions with instructions outside of the region shown.

Scratch instruction pipeline grids are provided for your convenience on the backs of this page and the previous page.

(A) **(4 points)** Specify the minimal number of **NOP** instructions (defined as **`ADD(R31,R31,R31)`**) to be added at each of the labeled points in the above program.



NOPs at Loop: 0 X

NOPs at AA: 2 ✓

1/4 **NOPs at BB:** 2 X

NOPs at CC: 3 X

(B) **(1 point)** On a **fully functional** 5-stage Beta pipeline (with working bypass, annul, and stall logic), the above code will run fine with no added **NOP**s. How many clock cycles of execution time are required by the fully functional 5-stage pipelined Beta **for each iteration** through the loop?

X **Clocks per loop iteration:** 86

END OF QUIZ!
(phew!)

Problem 4 (5 points): Broken pipeline

You've been given a 5-stage pipelined Beta processor as shown in lecture, whose diagram can be found on the back of page 1. Unfortunately, the Beta you've been given is defective: it has no bypass paths, annulment of instructions in branch delay slots, or pipeline stalls.

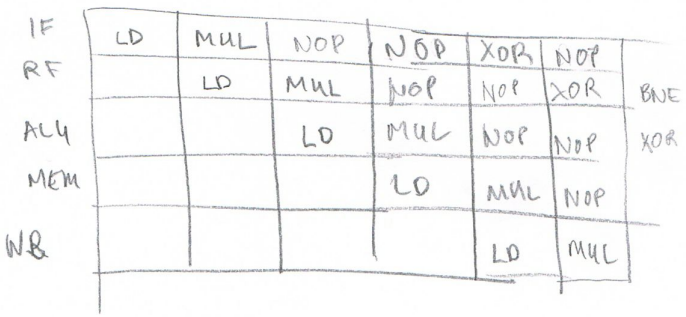
```
        ...
        NOP() NOP() NOP() NOP()
Loop:
        LD(R0, 0, R1)
        MUL(R2, R3, R4)
AA:
        XOR(R1, R0, R0)
BB:
        BNE(R4, LOOP, R0)
CC:
        XOR(R1, R2, R6)
        NOP() NOP() NOP() NOP()
        ...
```

You undertake to convert some existing code, designed to run on an unpipelined Beta, to run on your defective pipelined processor. The scrap of code on the left is a sample of the program to be converted. It doesn't make much sense to you – it doesn't to us either – but you are to add the **minimum** number of **NOP** instructions at the various tagged points in this code to make it give the same results on your defective pipelined Beta as it gives on a normal, unpipelined Beta.

Note that the code scrap begins and ends with sequences of **NOPs**; thus you don't need to worry about pipeline hazards involving interactions with instructions outside of the region shown.

Scratch instruction pipeline grids are provided for your convenience on the backs of this page and the previous page.

(A) **(4 points)** Specify the minimal number of **NOP** instructions (defined as **`ADD(R31,R31,R31)`**) to be added at each of the labeled points in the above program.



NOPs at Loop: 0 X

NOPs at AA: 2 ✓

NOPs at BB: 2 X

1/4

NOPs at CC: 3 X

(B) **(1 point)** On a **fully functional** 5-stage Beta pipeline (with working bypass, annul, and stall logic), the above code will run fine with no added **NOP**s. How many clock cycles of execution time are required by the fully functional 5-stage pipelined Beta **for each iteration** through the loop?

X

Clocks per loop iteration: 86

END OF QUIZ!
(phew!)