6.004 Computation Structures
Spring 2009

M A S S A C H U S E T T S I N S T I T U T E O F T E C H N O L O G Y
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.004 Computation Structures
Spring 2009

Quiz #5: May 8, 2009

Name	*Athena login name*	*Score*
Solutions		Avg: 20.1

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

Choose best answer (1 thru 6): 2

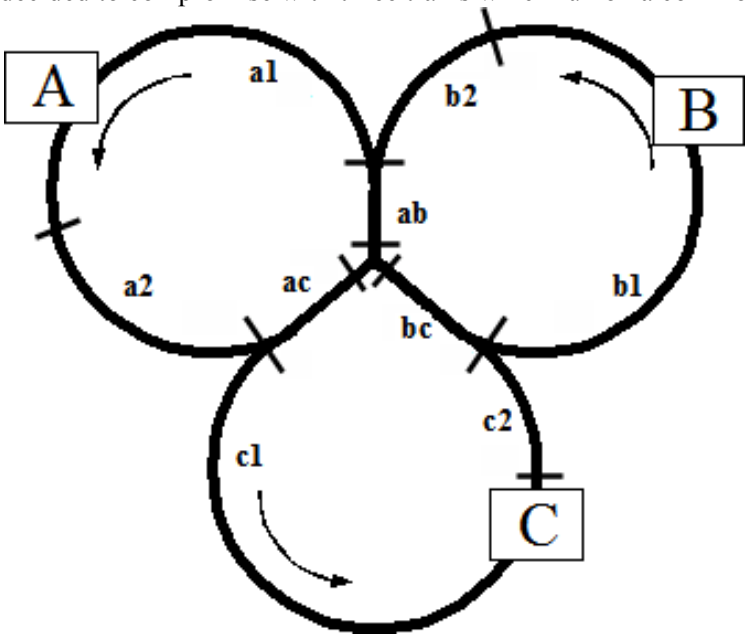
(B) (1 point) If we take into consideration physical realities like speed-of-light delays and minimum node size, the asymptotic worst-case latency between nodes of an N-node binary tree network is:

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

Choose best answer (1 thru 7): 2

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: R2

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
        …
```

(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.

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

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: 2

NOPs at AA: 2

NOPs at BB: 0

NOPs at CC: 1

(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?

Clocks per loop iteration: 6

END OF QUIZ!
(phew!)