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
Fall 2006

Quiz #5: December 8, 2006

Name		*Athena login name*	*Score*
TA: Justin Mazzola Paluska ☐ WF 10, 36-372 ☐ WF 11, 36-372	TA: Katarzyna Puchala ☐ WF 11, 34-303 ☐ WF 12, 34-303	TA: Daniel Taub ☐ WF 12, 34-304 ☐ WF 1, 34-304	TA: Hubert Pham ☐ WF 1, 34-303 ☐ WF 2, 34-303

Problem 1. Pipelined execution (6 Points)

In this example, we examine the execution of a "busy wait" loop on a standard 5-stage pipelined Beta. The process repeatedly examines the memory location that is labelled **`flag`**, waiting for it to become non-zero (e.g. having been set by another process). You may assume, however, that no process actually writes to this location, and the contents of the memory location **`flag`** remain zero.

```
flag:  long(0)              | Always zero, in this example
        …
loop:  LD(R31, flag, R0)    | poll the status flag
       BEQ(R0, loop, R31)   | keep polling if it's zero
       ADD(R0, R1, R2)      | ahh, something useful to do
       …
```

Please fill in the 14 blanks in the pipeline diagram below, showing the execution of this instruction sequence on a standard 5-stage pipelined Beta with full bypassing and one branch delay slot with annulment, as well as 2-stage memory access. When filling in the blanks assume that the **`flag`** location reads as zero, i.e., the **`BEQ`** is taken each time through the loop. Enter a symbolic opcode (such as "**ADD**") in each blank, or **NOP** for an annulled instruction.

You may want to refer to the pipelined Beta diagram on the reverse side of this page.

IF	LD	BEQ				
RF		LD				
ALU			LD			
MEM				LD		
WB					LD	

Problem 2. 3-process pipeline (11 Points)

In lecture, you saw the use of semaphores to mediate a communication stream between a Producer and a Consumer process. In this problem, we assume the existence of three asynchronous processes: a Producer process, producing a stream of characters; a Consumer process, which consumes a stream of characters; and a Filter process spliced between the Consumer and Producer processes. The Filter process takes characters from the producer, processes them (via a **`translate`** function), and passes the result to the Consumer process. The Producer and Consumer processes each communicate directly only with the Filter process.

The following is in Shared Memory (shared among Producer, Filter, and Consumer processes):

```
Semaphore charsA=???, spaceA=???, charsB=???, spaceB=???;
char buf[100];
char indata;
int in=0, out=0;
```

and the following code runs in the Filter Process:

```
while (1) {                    /* loop forever… */
   char temp;                  /* local variable */

   wait(charsA);
   temp = indata;
   signal(spaceA);
   temp = translate(temp);     /* do the actual translation */
   wait(spaceB);
   buf[in] = temp;
   in = (in+1)%100;            /* increment 'in' modulo 100 */
   signal(charsB);
}
```

(A) (1 point) What is the maximum number of characters that can be produced by the Producer process but not yet processed by the Filter process?

Maximum unprocessed characters produced: _____

(B) (4 points) What are appropriate initial values for each of the semaphores?

initial value for charsA: _____

initial value for spaceA: _____

initial value for charsB: _____

initial value for spaceB: _____

(C) (5 points) For each of the following lines of code, indicate whether you would expect to find them in the **P**roducer process, the **C**onsumer process, or **Neither** process:

`out = (out+1)%100;` in process **(circle one): P C Neither**

`signal(charsA);` in process **(circle one): P C Neither**

`wait(spaceA);` in process **(circle one): P C Neither**

`signal(spaceB);` in process **(circle one): P C Neither**

`wait(charsB);` in process **(circle one): P C Neither**

(D) (1 point) Assuming that the only process synchronization appearing in the Producer and Consumer processes is the use of the four semaphores shown, will the above implementation work with multiple Producer processes? Multiple Filter processes? Multiple Consumer processes? "Work" means each character produced by a Producer is translated by exactly one Filter process and then consumed by exactly one Consumer process, i.e., no character is lost or processed twice.

Works with multiple Producers (circle one): YES NO

Works with multiple Filters (circle one): YES NO

Works with multiple Consumers (circle one): YES NO

Problem 3 (8 Points): **Real Virtuality**

Real Virtuality, Inc. markets three different computers, each with its own operating system. The systems are:

> **Model A**: A timeshared, multi-user Beta system whose OS kernel is uninterruptable.
> **Model B**: A timeshared Beta system which enables device interrupts during handling of SVC traps.
> **Model C**: A single-process (not timeshared) system which runs dedicated application code.

Each system runs an operating system that supports concurrent I/O on several devices, including an operator's console with a keyboard. Les N. Dowd, RVI's newly-hired OS expert, is in a jam: he has dropped the shoebox containing the master copies of OS source for all three systems. Unfortunately, three disks containing handlers for the ReadKey SVC trap, which reads and returns the ASCII code for the next key struck on the keyboard, have gotten confused. Of course, they are unlabeled, and Les isn't sure which handler goes into the OS for which machine. The handler sources are

```
ReadCh_h() {                              /* VERSION R1 */
  if (BufferEmpty(0))                     /* Has a key been typed? */
    User->Regs[XP] = User->Regs[XP]-4;    /* Nope, wait.      */
  else
    User->Regs[0] = ReadInputBuffer(0); /* Yup, return it. */
}

ReadCh_h() {                              /* VERSION R2 */
  int kbdnum=ProcTbl[Cur].KbdNum;
  while (BufferEmpty(kbdnum)) ;           /* Wait for a key to be hit*/
  User->Regs[0] = ReadInputBuffer(kbdnum);  /*...then return it. */
}

ReadCh_h() {                              /* VERSION R3 */
  int kbdnum=ProcTbl[Cur].KbdNum;
  if (BufferEmpty(kbdnum)) {              /* Has a key been typed? */
    User->Regs[XP] = User->Regs[XP]-4;    /* Nope, wait. */
    Scheduler();
  } else
    User->Regs[0] = ReadInputBuffer(kbdnum);   /* Yup, return it. */
}
```

(A) (2 points) Show that you're smarter than Les by figuring out which handler goes with each OS, i.e., for each operating system (A, B and C) indicate the proper handler (R1, R2 or R3).

R1 goes with Model _____

R2 goes with Model _____

R3 goes with Model _____

But Les isn't that clever. In order to figure out which handler code goes with each OS version, Les makes copies of each disk and distributes them as "updates" to several beta-test teams for each OS. Les figures that if each handler version is tried by some beta tester in each OS, the comments of the testers will allow him to determine the proper OS for each handler.

Les sends out the alleged source code updates, routing each handler source to testers for each OS. In response, he gets a barrage of complaints from many of the testers. Of course, he's forgotten which disk he sent to each tester. He asks your help to figure out which combination of system and hander causes each of the complaints.

For each complaint below, indicate which handler and which OS the complainer is trying to use.

(B) (2 points) Complaint: "I get compile-time errors; Scheduler and ProcTbl are undefined!"

User has handler _____ on system _____

(C) (2 points) Complaint: "Hey, now the system always reads everybody's input from keyboard 0. Besides that, it seems to waste a lot more CPU cycles than it used to."

User has handler _____ on system _____

(D) (2 points) Complaint: "Neat, the new system seems to work fine. It even seems to waste less CPU time than it used to!"

User has handler _____ on system _____

END OF QUIZ 5 (“phew!”)

END OF 6.004 (“aw…”)

Enjoy a stellar career at MIT
And beyond!