Problem 1. Interrupts and Real-time (7 points)

A particular real-time system has three interrupt handlers. The following table shows the maximum rate at which each interrupt occurs (rate), the time taken to execute each handler (service time), and the maximum allowable interval between the interrupt and completion of the handler (deadline). In your analysis, assume that A, B, and C interrupts can arrive at any time.

<table>
<thead>
<tr>
<th>Task</th>
<th>Rate</th>
<th>Service time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$1/20\text{ms}$</td>
<td>10ms</td>
<td>20ms</td>
</tr>
<tr>
<td>B</td>
<td>$1/80\text{ms}$</td>
<td>10ms</td>
<td>80ms</td>
</tr>
<tr>
<td>C</td>
<td>$1/25\text{ms}$</td>
<td>5ms</td>
<td>25ms</td>
</tr>
</tbody>
</table>

(A) (2 point) What is the percentage idle time for this system?

Percentage Idle Time (%): ________

(B) (2 points) Assuming a weak priority system. Can the priority ordering C > A > B satisfy all the constraints of the system?

C > A > B (Yes/No): ________

(C) (3 points) Assuming a strong priority system with priority C > A > B, what is the worst-case delay between the task’s interrupt and completion of the task’s interrupt handler?

Worst-case delay for task A (ms): ________

Worst-case delay for task B (ms): ________

Worst-case delay for task C (ms): ________
Problem 2. Virtual Memory (10 points)

Consider a Beta processor that includes a 40-bit virtual address, an MMU that supports 4096 ($2^{12}$) bytes per page, $2^{32}$ bytes of physical memory, and a large Flash memory that serves as a disk. The MMU and the page fault handler implement an LRU replacement strategy.

(A) (2 points) What is the size of the page map for this processor? Assuming the page map includes the standard dirty and resident bits, specify the width of each page map entry in bits, and number of entries in the page map.

Size of page map entry in bits:_________

Number of entries in the page map:_________

(B) (6 points) The following test program is running on this Beta processor. The first 8 locations of the page table, just before executing this test program, are shown below; the least-recently-used page (“LRU”) and next least-recently-used page (“next LRU”) are as indicated. This Beta processor also has a 4 element, fully associative, Translation Lookaside Buffer (TLB) that caches page map translations from VPN to PPN.

. = 0x0
ADDC(R31,0x2800,R3)
LD(R3,0,R5)
ST(R5,0x4100,R31)

<table>
<thead>
<tr>
<th>TLB</th>
<th>Tag (VPN)</th>
<th>D</th>
<th>R</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU→</td>
<td>0x3</td>
<td>1</td>
<td>1</td>
<td>0x1</td>
</tr>
<tr>
<td></td>
<td>0x2</td>
<td>0</td>
<td>1</td>
<td>0x3</td>
</tr>
<tr>
<td></td>
<td>0x6</td>
<td>0</td>
<td>1</td>
<td>0x2</td>
</tr>
<tr>
<td></td>
<td>0x1</td>
<td>0</td>
<td>1</td>
<td>0x5</td>
</tr>
<tr>
<td>Next LRU→</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Map</th>
<th>VPN</th>
<th>D</th>
<th>R</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU→ 3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0x1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0x3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0x0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0x2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0x6</td>
</tr>
<tr>
<td>Next LRU→</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For each virtual page that is accessed by this program, specify the VPN, whether or not it results in a TLB hit on the first access to that page, whether or not it results in a page fault, and the PPN that the page ultimately maps to. *You may not need to use all rows of the table.*

<table>
<thead>
<tr>
<th>VPN</th>
<th>TLB Hit (Yes/No)</th>
<th>Page Fault (Yes/No)</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(C) (1 point) Which physical pages, if any, need to be written to disk during the execution of the test program in part B?

Physical page numbers written to disk or NONE: ________

(D) (1 point) What is the physical address of the LD instruction?

Physical address of LD instruction: 0x________

Problem 3. Semaphores (7 points)

The following three processes are run on a shared processor. They can coordinate their execution via shared semaphores that respond to the standard signal(S) and wait(S) procedures. Their intent is to print the word HELLO. Assume that execution may switch between any of the three processes at any point in time.

(A) (3 points) Assuming that no semaphores are being used, for each of the following sequences of characters, specify whether or not this system could produce that output.

LEHO (YES/NO): ______ HLOE (YES/NO): ______ LOL (YES/NO): ______

(B) (4 points) You would like to ensure that only the sequence HELLO can be printed and that it will be printed exactly once. Add any missing wait(S) and signal(S) calls to the code below (where S is one of a, b or c) to ensure that the three processes can only print HELLO exactly once. Remember to specify the initial value for each of your semaphores. Recall that semaphores cannot be initialized to negative numbers.

Semaphores: a = ___; b = ___; c = ___;

Process 1                           Process 2                           Process 3
Loop1:                           Loop2:                           Loop3:
  wait(a)                         wait(b)                         wait(c)
  print("H")                     print("L")                     print("O")
  print("E")                     goto Loop2                      goto Loop3
  signal(b)
  goto Loop1

6.004 Fall 2016  - 3 of 5 -  Quiz #4
Problem 4. Interrupt Handler (6 points)

In order to better understand the behavior of interrupt handlers, one of your fellow classmates has decided to experiment with minor changes to the operating system in Lab 7. He creates an interrupt handler, INT_H which makes use of the following macro definitions.

- **Savestate**() – saves all the hardware registers into UserMState data structure
- **Restorestate**() - restores hardware registers from UserMState data structure
- **Scheduler**() – copies UserMState of current process into ProcTbl array which stores state of all processes. It then updates the current process to point to the next process and copies its state into the UserMState data structure.
- **DecrementXP**() – decrements XP register in UserMState data structure by 4 to point to interrupted instruction which did not get executed.

INT_H also makes use of the preexisting I_WAIT and I_RTN code from lab 7. As a reminder, here is the pseudocode for I_WAIT and I_RTN together with the code for INT_H.

```plaintext
INT_H:    Savestate()
          Branch to I_WAIT

I_WAIT:   DecrementXP()   // Make XP of current process point to interrupted instruction
          Call Scheduler()  // Switch current process
          Branch to I_RTN

I_RTN:    Restorestate()  // Restore hardware registers with contents of new process
          Jump to XP       // Return to user mode of new process
```

To test his interrupt handler, he runs the operating system using the following two process stubs.

```plaintext
Proc0:
  ADDC(R31, 12, R0)
  ADDC(R0, 2, R0)  ← Interrupted instruction
  ADDC(R0, 3, R0)
  HALT()

Proc1:
  ADDC(R31, 10, R0)
  ADDC(R0, 20, R0)
  HALT()
```

Assume that for each trial an interrupt occurs while Proc0 is executing the **ADDC(R0,2,R0)** instruction. The interrupted instruction is not completed until after the interrupt handler has run.

For each of the following trials, please specify the value stored in R0 by each of the processes once they reach their HALT() statement. **Enter CAN’T TELL if you cannot tell the value of a particular R0 from the information given.**
Trial 1 (2 points): Run INT_H code as shown above without any further modifications.

Value left in R0 of Proc0: 

Value left in R0 of Proc1: 

Trial 2 (2 points): Remove the DecrementXP() call in I_WAIT.

Value left in R0 of Proc0: 

Value left in R0 of Proc1: 

Trial 3 (2 points): Modify I_WAIT so that call to Scheduler() precedes DecrementXP().

Value left in R0 of Proc0: 

Value left in R0 of Proc1: 

END OF QUIZ 4! Have a great break…