COSC 2P13
Assignment 1

Due date: February 4th, 2022 at 23:59 (11:59 pm).
Team: The assignment can be completed in groups of up to two students.
Delivery method: the student needs to delivery the assignment only through Sakai.
Delivery contents: document with answers and [Java, C, C++] codes if applicable (see Submission instructions).
Attention: check the Late Assignment Policy.

Introduction [30]

1.1. Consider the diagram illustrated in Figure 1. Assume the following microcode corresponds to a instruction code already loaded in memory. Assume R1 is a general purpose register, and the Program Counter (PC) is at 2000. Also, assume that addresses 5000, 5001, and 5003 in main memory contain the values 23, 97, and 256, respectively. **[6]**

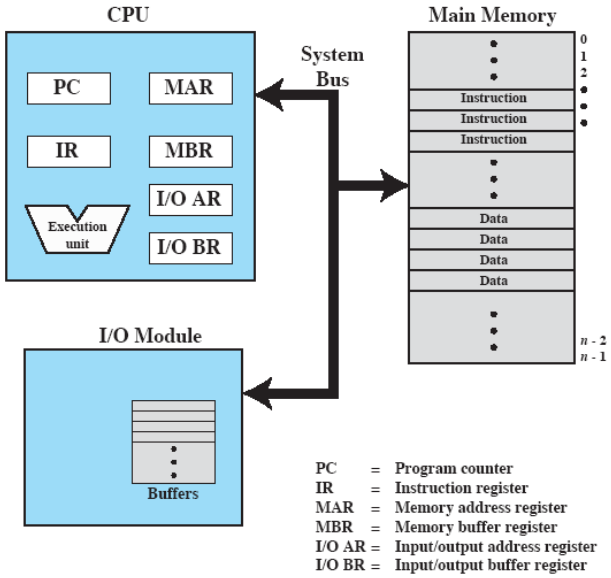

Figure 1: Top-level view of computer components.

Following the microcode, each certain set of lines correspond to an instruction.
MAR = PC, PC+1, assert MAR
IR = MBR, decode IR
put addr 2003 into MAR, assert MAR
move MAR to PC
MAR = PC, PC+1, assert MAR
IR = MBR, decode IR
put addr 5000 into MAR, assert MAR
move MBR to AC
MAR = PC, PC+1, assert MAR
IR = MBR, decode IR
put addr 5001 into MAR, assert MAR
sum AC with MBR, result in AC

MAR = PC, PC+1, assert MAR
IR = MBR, decode IR
put AC to 5002
MAR = PC, PC+1, assert MAR
IR = MBR, decode IR
put addr 5003 into MAR, assert MAR
move MBR to R1
MAR = PC, PC+1, assert MAR
IR = MBR, decode IR
put addr 21000 into I/O AR, assert I/O AR
put R1 in I/O BR

Table 1: Use the following table as your reference.

Address	**Instruction / data**	**Comments**
...		
2000		
2001		
2002		
2003		
2004		
2005		
2006		
2007		
2008		
2009		
...		
5000	23	
5001	97	
5002		
5003	256	
...		
21000		

Example of an instruction (expected answer):
- SUB AC, (1000)

Example of an explanation of an instruction and microcode (expected answer):
- It subtracts the value of AC with the value of address 1000; the result is placed in AC.
    - Loads the current PC in MAR and points PC to the next instruction
    - Loads the instruction information in MBR or MBR contains the instruction information
    - Loads the operation in IR
    - Loads MAR with address 1000
    - Subtracts AC with MBR, result in AC

(i) Identify which instructions are represented by this microcode. In other words, complete the table with the information presented in the microcode. [2]
(ii) Explain/comment each instruction, referring to the microcode. [2]
(iii) Write the final output and the form that this output is given. [2]

1.2. Interrupts **[8]**
(i) What is the role of interrupts in Operating Systems? [2]
(ii) How does an interrupt work? [2]
(iii) Why are they important towards processing and CPU utilization? [2]
(iv) How does the Operating System resolve the issue of receiving an Interrupt while it is still handling an earlier Interrupt? Discuss the possible solution(s). [2]

1.3. Assume the context of an Operating System that provides access to its service/library through System Calls. **[6]**
(i) Why are these calls needed in such Operating Systems? [3]
(ii) Illustrate your answer with an example and explain the functioning of such calls in low level. [3]

1.4. The PCB consists of the PID, Process State Information and Process Control Information. **[4]**
(i) For each element of the above, expand and exemplify. What is the use of each of them? What is contained within those elements/sections and why? [4]

1.5. What is the difference among 2-state Process Model (Figure 2), 5-state Process Model (Figure 3), and 7-state Process Model (Figure 4)? What advantages one model adds on the other? Explain in detail. **[6]**

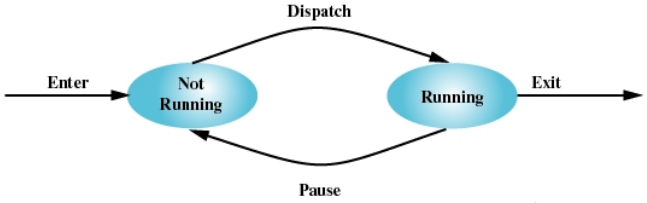

Figure 2: Two-state process transition diagram.

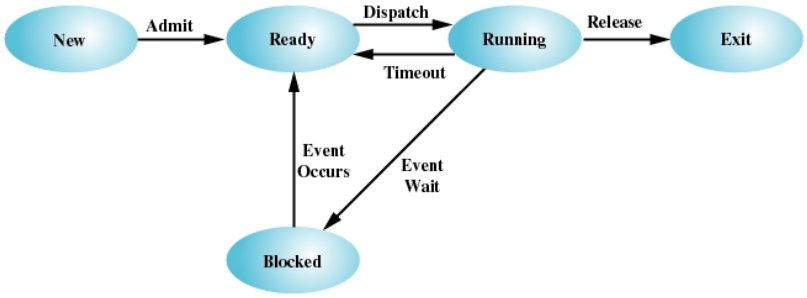

Figure 3: Five-state process transition diagram.

Process and Threads [35]

2.1. What are the basic differences between user-level threads and kernel-level threads? **[4]**

2.2. What are the benefits and drawbacks of having pure user-level threads? **[4]**

2.3. A binary semaphore is a semaphore that takes only the value 0 and 1. Would a binary semaphore be a mutex? **[4]**

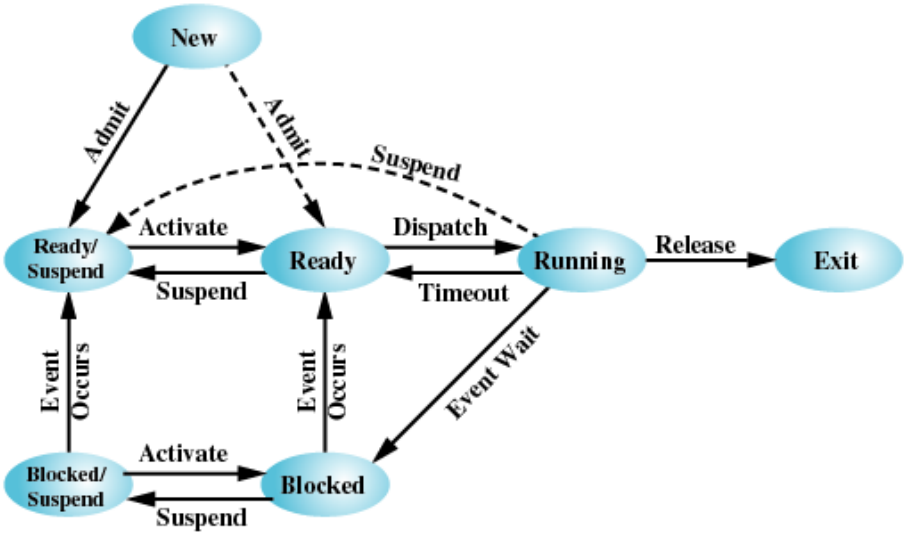

Figure 4: Seven-state process transition diagram.

2.4. Write a Java code spawns three threads that run independently and make shared-memory communication. The code can perform any simple task the student chooses. However, the code must have, and clearly demonstrate, (i) an issue with race condition among the three threads. There should also be a (ii) second version of this code where the race condition issue is solved, using locks, mutexes, or semaphores. The student must present/attach two pieces of code, one for each (i) and (ii). Also, the student must describe in detail the code, including the race condition and respective solution, as answer to the question here. **[9]**

2.5. Assume a multithreaded Producer/Consumer code. In the code, one thread produces values and places them into a shared variable (Linked List); the other thread consumes values by taking/removing them from the shared variable. One implementation of this supposed scenario is the attached Java code. In the code, the access to the share variable is controlled through synchronization: locks. One thread enters the critical region by taking/reserving the lock and then releases (notify). In case the lock has already been taken by the other thread, the current thread gets blocked (wait). With this duality, visited in class, we guarantee the mutual exclusion when accessing the critical region. Based on the provided code, (i) is possible to replace the "**wait** and **notify**" with just a "**timed wait**". Explain your answer and show it in the code (create a new code where you just use timed waits, if possible). (ii) By just using a "**timed wait**" and not using "**notify**", your code necessarily will suffer from a major concurrency issue when running in long term. Identify the issue and explain it. **[9]**

Memory Management [35]

3.1. In a scenario of Fixed Partitioning, our 128MB memory is split into 8MB-size partitions. We have 10 processes that need to be placed into the main memory; these program present the following sizes in MB: 7, 6, 1, 8, 5, 8, 8, 2, 4, 3. (i) Will there be waste of memory in placing such processes? If yes, what is the issue that is causing the memory management waste? (ii) If we have a 11th program with size of 9MB to be placed in memory, how would this program be placed? **[9]**

3.2. In a scenario of Dynamic Partitioning, we have a main memory with size of 128MB. We have 6 processes to be placed in memory, and they have the following sizes: P1(20MB), P2(14MB), P3(32MB), P4(10MB), P5(16MB), and P6(24MB). These processes might come in (IN) the memory and may leave the memory (OUT) from time to time, showing the following ordering of IN and OUT into memory (assuming an

empty memory at start): P1(IN), P2(IN), P4(IN), P1(OUT), P3(IN), P5(IN), P3(OUT), P1(IN), P2(OUT), P4(OUT), P3(IN). Depending on the placement policy used by the system, we may have different memory usage layouts at the end of the given placement sequence. For any policy, the dynamic partitioning will waste memory space cause by an inherent issue on its design. (i) Explain this inherent issue and how it affects memory management efficiency. (ii) What is Memory Compaction? Does it solve the problem? (iii) Following the each of the Dynamic Partitioning Placement Algorithms (Best-fit, First-fit, and Next-fit), give the memory layout upon applying them to fulfill the provided IN/OUT memory sequence. **[9]**

3.3. (i) What is memory address translation? Why is there a need for this translation in modern Computer Systems? (ii) How is Logical-to-Physical Address Translation done in Paging and in Segmentation? **[4]**

3.4. (i) What is Page Table? (ii) What is the Translation Lookaside Buffer (TLB)? (iii) Explain the whole process of Logical-to-Physical Address Translation using TLB for Virtual Memory Management. **[4]**

3.5. In memory management, (i) what is Cleaning Policy? (ii) What is Load Control? (iii) What does Virtual Memory Management have to do with Multiprogramming? Explain. **[4]**

Submission Material

The submission for this assignment will consist of two parts:

4.1. The ***Answer document (PDF)***. The answers to the assignment questions should be all included in this document. Also, include the description of your source Java code, especially the compilation and execution instructions. There is no need to worry about the formatting of this document; you must follow the provided Latex template.

- ***Latex template - a must for writing your assignment***. For writing your description file, use the Latex template enclosed in this assignment (update it accordingly!). You do not need to install Latex software in your computer. You can write it through Overleaf on your browser (it is a free tool). Just upload the latex template to your Overleaf project; it should compile/render the tex file gracefully.

4.2. The **Java code respective to the assignment questions**. The Java code be documented (commented) properly. The code should compile and run properly. The compilation and execution of your code should not rely on any IDE.

- ***Compilation***. Provide the command for compiling your source code from the command line.
- ***Running***. Provide the command for running your compiled code from the command line.

Submission

Submission is to be a PDF with answers and source Java files. All the submission should be performed electronically through Sakai.

You must guarantee that your code is legible, clear, and succinct. Keep in mind that any questionable implementation decision or copy from any source might have a negative effect on your mark. If you still have any questions regarding which file types are acceptable, please inquire prior to submission. Note that it is not the fault of the marker if they are unable to mark your assignment due to submitting an unreasonable or uncommon file format.

All content files should be organized and put together in a ZIP for the submission through Sakai.

* **Do not forget to include the names and student IDs of group members.**

Marking Scheme

Marks will be awarded for completeness and demonstration of understanding of the material. It is important that you fully show your knowledge when providing solutions in a concise manner. Quality and conciseness of solutions are considered when awarding marks. Every code added to the originals should be well commented and explicitly indicated in the Java files; lack of clarity may lead you to loose marks, so keep it simple and clear.

Late Assignment Policy

A penalty of 25% will be applied on late assignments. Late assignments are accepted until the Late Assignment Date, three days after the Assignment Due Date. No excuses are accepted for missing deadlines. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion.

Plagiarism

Students are expected respect academic integrity and deliver evaluation materials that are only produced by themselves. Any copy of content, text or code, from other students, books, web, or any other source is not tolerated. If there is any indication that an activity contains any part copied from any source, a case will be open and brought to a plagiarism committee's attention. In case plagiarism is determined, the activity will be cancelled, and the author(s) will be subject to the university regulations.
For further information on this sensitive subject, please refer to the document below:

`https://brocku.ca/academic-integrity/`