Exercises in this document are to be done at the lab. We will not grade these solutions, so you do not need to send them to us.

### 1 Exam questions

1. Assume that a program’s memory is as follows:

   - Variable A stored at address 20 contains value 40.
   - Variable B stored at address 30 contains value 50.
   - Variable C stored at address 40 contains value 60.
   - Variable D stored at address 50 contains value 70.

   What will be the value of R2 after the following code?

   ```
   MOV R1,#B
   MOV R2,(R1)
   ADD R2,#20
   ```

   **Answer:** The first instruction will use immediate addressing and will load to R1 the address of variable B, that is, 30. The second instruction will use register indirect addressing and will load to R2 the contents of address location 30, which is the value 50. The third instruction will add the number #20 (immediate addressing) to R2, so R2 will end up with the value 70.

2. Assume that a program’s memory is as follows:

   - Variable A stored at address 10 contains value 20.
   - Variable B stored at address 20 contains value 30.
   - Variable C stored at address 30 contains value 40.
   - Variable D stored at address 40 contains value 50.

   What will be the value of R2 after the following code?

   ```
   MOV R1,B
   MOV R2,A(R1)
   ADD R2,#B
   ```

   **Answer:** The first instruction will use direct addressing (the assembler will replace it by “MOV R1,(20)”) and will load to R1 the contents of address 20, that is, the value 30. The second instruction will use indexed addressing and will load to R2 the contents of address location 10+30 (i.e., address 40), which is the value 50. The third instruction will use immediate addressing and will add the address of variable B (i.e., the value 20) to R2, so R2 will end up with the value 70.
3. Give two very different examples of situations that lead the CPU to receive a hardware interrupt.

Answer: There are many events which can generate hardware interrupts. The most notable ones are: (i) a device wants to send some data to the CPU; (ii) a DMA transfer has finished and the DMA chip wants to notify the CPU; (iii) a timer previously set by the OS has expired so it is time for the OS to regain control and schedule another process.

4. Explain what the TSL instruction does, and why it is useful for implementing process synchronization.

Answer: TSL means “Test and Set Lock”. It realizes two operations in an atomic fashion: (i) it checks if a variable is equal to 0; (ii) it sets the variable to 1. This instruction allows to implement lock variables: since the instruction is atomic it cannot be interrupted in the middle so it is impossible for two processes to execute it simultaneously and both believe they have acquired the lock.

5. We want to design a process scheduling discipline for a new operating system.

In this system, processes do not have priority levels. On the other hand, each process has an explicit attribute which states if it is CPU-bound or I/O bound.

In essence, a scheduling discipline must choose which process to execute first in the following situations:

- Two CPU-bound processes are in WAITING state
- Two I/O-bound processes are in WAITING state
- One CPU-bound and one I/O bound process are in WAITING state

Propose a simple scheduling discipline for each of these three cases and explain why it makes sense.

Answer: A simple policy is as follows: CPU-bound and I/O-bound processes make intensive use of different resources so we can schedule them independently of each other. We use a round-robin policy across all CPU-bound processes, and a round-robin policy across all I/O-bound processes. If a CPU-bound and an I/O-bound processes are in WAITING state, then we always choose the I/O bound one: we expect that it will use the CPU for a short period of time before being blocked, while the CPU-bound process may keep the CPU busy for a much longer duration.

There are of course many other possible policies. Acceptable policies are those which (i) make use of the distinction between CPU-bound and I/O-bound processes, and (ii) come with a plausible explanation.

6. In most operating systems a process can only be created by another process using the fork() system call. Explain how the first process is created.

Answer: The first process, called init, is created from scratch by the operating system’s kernel during the boot process. The init process starts up other processes in the system, which in turn can create more processes.

2 Assembly programming

The exercises described in this section and the home assignment will be based on the Mic1MMV microarchitecture simulator.
**Question 1:** Download files `echo.jas` and `ascii.jas` from the course’s web page and open them in a text editor. Read the programs line by line. Make yourself familiar with the IJVM assembly syntax and instruction set. Do you understand how the two programs work? Run them in Mic1MMV and see the results.

**Question 2:** We want to implement multiplication in assembly (.jas) instead of micro-code (.mal) like last week. How can this be done without updating the original microprogram? Write an IJVM program that pops two numbers from the stack, multiplies them, and pushes the result back onto the stack. Do not use the IMUL instruction (if you still have it). Compare the execution cost (in number of CPU cycles) of your program with the cost of the IMUL instruction you wrote last week. Which one is faster, the microprogram or the ISA-level program?

— the end —