Computer Systems
Lab #5 — Assembly

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

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,#20

3. Give two very different examples of situations that lead the CPU to receive a hardware interrupt.

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

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.

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

## 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 \texttt{echo.jas} and \texttt{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 micro-program or the ISA-level program?

— the end —