ngGebze Technical University
Department Of Computer Engineering
CSE 312 / CSE 504

Operating Systems

Homework #01
Due Date: March 30th 2018
8080 CPU Emulation and Simple OS

In this homework, you will run processes on a simulated Intel8080 CPU emulator. The emulator is provided as a C++ class. Your job in this homework will be to write a few assembly programs to run on this emulator. However since you will need some OS systems calls and we do not have any OS available, you will also have to write some simple OS system calls mainly for screen input and output. Here are the files that we provide with this homework:

8080Book.pdf : Good book on 8080 Assembly programming
8080_Z80 Instruction Set.pdf : reference for 8080 assembly programming

8080emu.cpp : Emulator implementation for 8080. **You will not change this file**
8080emuCPP.h: Emulator interface for 8080. **You will not change this file**
gtuos.h : sample header for GTUOS. You will rewrite this file
gtuos.cpp : sample implementation for GTUOS. You will rewrite this file
main.cpp : sample main function for this HW. You will rewrite this file
memory.cpp and memoryBase.cpp: implementation of physical memory. You will not change these files for this homework.

Some sample programs
sum.asm: Adds numbers from 0 to 10 and prints the results on the screen.
sum2.asm : Adds numbers from 0 to 10 and prints the results on the screen using a different system call.
printstr.asm : 8080 Assembly file that prints "Hello world" on screen.

You need to use an assembler to produce the executable machine code for your emulator. Use the simple assembler at , http://asdasd.rpg.fi/~svo/i8080/ which helps a lot in writing the assembly code. You can directly download the executable .com file from this site with some browsers.

There are two main tasks for you to do in this homework.
First, you need to implement a number of GTUOS systems calls as described below

Call	**Params to pass**	**Function**	**How many cycles**
PRINT_B	Register A = 1, Register B holds the value to be printed	Prints the contents of register B on the screen as a decimal number.	10
PRINT_MEM	Register A = 2, Register BC = address	Prints the contents of the memory pointed by registers B and C as a decimal number.	10

READ_B	Register A = 3, Register B will hold the value	Reads an integer from the keyboard and puts it in register B.	10
READ_MEM	Register A = 4, Register BC = address	Reads an integer from the keyboard and puts it in the memory location pointed by registers B and C	10
PRINT_STR	Register A = 5, Register BC = address	Prints the null terminated string at the memory location pointed by B and C	100
READ_STR	Register A = 6, Register BC = address	Reads the null terminated string from the keyboard and puts the result at the memory location pointed by B and C	100
GET_RND	Register A = 7, Register B = random byte	Produces a random byte and puts in register B	5

You will rewrite the two files (gtuos.h, gtuos.cpp) to implement the above system calls and you will write another file (main.cpp) to use your OS with the 8080 CPU. Use the sample main file as a guide.

You will also write and test the following assembly files. Use the provided sample assembly files to learn about how to use the assembler and how we call the OS.

1. PrintNumbers.asm: file that prints integers from 0 to 50 by steps of 2 on the screen. Each number will be printed on a new line.
2. Sort.asm: file that produces 50 random bytes, sorts them in increasing order and prints them on screen.
3. Binary Search.asm: Produces 50 random bytes and sorts them. Your program then take a number from the keyboard, makes a binary search on these numbers. If found prints the memory address else prints "error"
4. test.asm: a file that tests each of the system calls one by one.

Your program should work with this command line
`sim8080 exe.com debugFlag`

There should be parameters for the program name and debug flag. Sim8080 is your program, exe.com will be produced by the assembler, and the debug will work as follows

- **`sim8080 exe.com 1`** : will read the program from exe.com. In debug mode 1, the status of the CPU will be printed to the screen after each instruction execution. At the end of the simulation, the whole memory will be saved to exe.mem as a text file of hexedecimal numbers. Each line of the file will start with the memory addres, then it will show 16 bytes of hexadecimal numbers separated with spaces.
- In Debug mode 0, the program will be run and the contents of the memory will be saved as in Debug mode 1.
- In Debug mode 2, after each instruction execution, the CPU state will be displayed. Your simulation will wait for an from the keyboard and it will continue for the next tick. The contents of the memory will be saved as in Debug mode 1.

All three modes will display the total number of cycles used by the program at the end of the executions.

We will provide the submission instructions in a separate document. You should strictly follow these instructions otherwise your submission may not get graded.