ECE 429 / 629 Programming Assignment #2

In this assignment you will write a program that simulates the single-cycle machine of COD Fig. 5.29. As with the previous assignment, the program will run from the command line, with a single argument indicating the name of the input file. Thus, if your executable is named *msimulate*, the syntax will be:

msimulate file.txt

where *file.txt* specifies the input file. The input file will contain a MIPS assembly language program, with the same format as before.

The program should parse the input file and create the following output files:

- *out_code.txt.* This file contains the 32-bit machine code, one line per instruction. In the file, each bit should be an ASCII 0 or an ASCII 1.
- *out_control.txt.* This file contains the 11 bits specifying the control lines, one line per clock cycle. The lines should be listed in the order in which they are shown in the figure (the ten bits output from the Control block, followed by the Zero bit output from the ALU): RegDst, Jump, Branch, …, RegWrite, Zero. Note that ALUOp itself contains 2 bits. In the file, each bit should be an ASCII '0' or an ASCII '1' or an ASCII 'X' (for "don't care").
- *out_registers.txt.* This file contains the value of the program counter, followed by the values of the first five registers, R0-R4. In the file, each byte should be written as a pair of hexadecimal digits (using capital letters).

Just as in the previous assignment,

- If any output file already exists, the program should overwrite it.
- Each line should terminate with a line feed + carriage return (Windows-style "\r\n").
- There should be no spaces in the output files.
- Only the following instructions need to be supported: add, sub, addi, beq, bne. If the program ever encounters an invalid input line, it should append "!!! invalid input !!!\r\n" (without the quotes) to the output file. The program should then close all output files and terminate.
- If, during the execution of the program, the simulated program counter ever goes out of bounds (i.e., before the first instruction or more than one word after the last instruction), then the line "!!! seg fault !!!\r\n" (without the quotes) should be appended to both the control and register output files. The program should then close all output files and terminate. *(Note: The out-of-bounds test should be performed at the beginning of each clock cycle, at the time that the new instruction would have been fetched. In other words, the instruction that causes the PC to go out of bounds should be allowed to complete.)* On the other hand, if the PC points to just after the last instruction, then the program should exit silently.

The output of the program should be exactly as specified above, byte for byte.

Notes

The initial value of all registers at the start of any program should be zero (0). The initial value of the program counter should be AAAA0000 (hexadecimal).

Do not worry about arithmetic overflow.

Be sure to enforce that R0 = 0 always.

Your simulated MIPS machine should contain all 32 MIPS registers. All registers should function properly, even though only a few of them are written to the output file.

MemRead should never be "Don't care", because reading a DRAM is a destructive operation. Therefore, asserting MemRead when it is not needed causes unnecessary refreshing.

Assume that additional hardware (not shown in the diagram) exists to support the branch-if-not-equal (bne) instruction. The meaning of the various control lines is unaffected.

Your program should execute for a maximum of 100 cycles. If, after 100 cycles, the program is still running, then "…\r\n" should be appended to both the control and register output files, and the program should terminate.

Assume an edge-triggered clock. Therefore, your program should, in a single clock cycle, write the values of the control lines computed during that clock cycle, along with the values of the registers available to be read during the same clock cycle. If a new value of a register is computed during a cycle, that new value will not be reflected until the next clock cycle.

It is recommended that you throw and catch an exception whenever an error occurs, as C++ allows. The C-style approach of returning and checking error values is error-prone and tedious.

Additional notes

Under no circumstances whatsoever should the program crash. The program should allow for wrong parameters being passed on the command line, as well as a corrupt input file. Buffer overflow should be guarded against.

Never use strcpy. Use strncpy instead – it is safer. Similarly, never use sccanf with a "%s" parameter. Instead, use "%ns", where n is the length of the allocated string, e.g., "%100s". For this assignment, you should need only fgets and strtok.

Whenever possible, it is suggested that you use std::string or CString rather than C-style char arrays to minimize the chance of low-level memory mistakes.

Avoid magic numbers. All constants should be defined at the top of your file in an obvious place.

By default, fopen opens a file in text mode on Windows. In text mode, “\n” will automatically be converted to “\r\n”. If you would like to avoid such under-the-hood translations, you should open the file in binary mode. This is done by inserting a ‘b’ into the second string, e.g., fopen(“file.txt”, “wb”).