SIMP (with bonus)

Written by:

Natanel Levi 314915364

Yonatan Erez 208987073

Overview:
As we were instructed, we decided to split the project into a couple of stages:

The first stage is the Assembler in which it will receive the assembly file and generate "memin.txt" output file which contains the given SIMP assembly instructions converted into machine language.

The second stage is the actual Simulator which receives the "memin.txt", "diskin.txt" and "irq2in.txt" files and generates 9 output files as requested.

For clarity and for keeping things clean, we've decided to have a designated C program that holds all the character-integers manipulation we needed and to have each structure have its own C program. We included all these files inside a header.h file which also contains all the external libraries we've used aswell as commonly used pre-processed parameters.

The Assembler:

The assembler will go over the assembly file twice and parse its lines into SIMP machine language. In the first iteration it will setup the labels data base inside a linked list and match them up with their corresponding PC lines. In the second iteration it will load 1 line at a time from the assembly file, correspond it to either R-format, I-format,. Word and Label and handle it according to the given project instructions.

Assumptions:

1. Labels:
    a. Label lines cannot start with a dot (".").
    b. Each label must end with colons (":").
    c. Labels cannot start with ". word" and cannot contain neither "$imm" nor hashtags ("#") nor colons (":") inside of them.
    d. A line that has a label cannot contain further assembly instructions (excluding comments).

2. .ASM input
    a. The input file (the assembly program) must not contain blank lines ANYWHERE inside of it (including at the end).
    b. Each line must be a known format such as: R-Format, I-Format, .Word instruction or Label.
    c. Comments can only be written at the end of an assembly line and must start with hashtag "#".
    d. An R-format instruction must have a "0" on its immediate value.

The Simulator:

The simulator will iterate over the lines of the memin.txt file and each iteration it will update the trace.txt and hwregtrace.txt files, when it reaches a "halt" SIMP instruction it will terminate the simulation and generate the rest of the output files.

The simulator does not start with a stack and global pointer initiated to some value, rather its up to the user to define them and use them as one sees fit.

We assume the user is aware of the limitations of the simulator is aware and possibly prefer that the "trace registers" (such as $s1, $t2 etc..) can overflow when calculating a result which is represented by more than 32bits.

The simulator is written in a manner such that it first uploads the input files "memin.txt", "irq2in.txt" and "diskin.txt" into accessible databases, it than fetches, decodes and executes each instruction while maintaining the status of all the i/o registers up to date. On an interrupt it will store its current instruction and go over the ISR in order to handle it. At the end it will generate all the output files needed.

We've designated each "block" of registers its own "manager" function to simulate different hardware blocks connected to one another.

The Assembly programs are:

Fibo:

Computer_Structure/assembly_files/fibo.asm

We defined the Fibonacci series to start with 0,1,1,… so starting address 0x100 we will hold the value 0. We used 1 ".word" command to already write a "0x1" into the next address and starting from that the other numbers were calculated and written by the simulator. The code is written such that it will keep updating 2 registers with the sum of its previous 2 calculated numbers and halt whenever the sum is bigger than "0x7ffff", we've chosen this number because it's the "largest" signed number represented by 20b therefore bigger will cause overflow when reading back from the memory (we will not get back our written number but instead we will receive a negative one due to sign extension).
Requires: memin.txt

Square:

Computer_Structure/assembly_files/square.asm

First we are loading the start_address, len of the square.

Then we are validating the input so the square will not overflow out of the monitor bounds.

After that we are having both valid start_address, len.

In order to draw the square we decided to calculate the start_address % 256 so it will be easier to jump to the next line of the square.

For each iteration of drawing we are checking whether we reached the end of the square line or the end of the total square.

Then we are branching to Halt and disabling the monitor's drawing register.

Given start address, edge length we are building a circle when:

a. $0 \leq start\ adress \leq 256^2 - 1 = 65535$
b. $0 \leq edge\ length \leq 255$

When edge length=0, represents a single pixel.

In our test we used:

$$start\ adress = 257$$

$$len = 10$$

Requires: memin.txt

Disktest:

Computer_Structure/assembly_files/disktest.asm

The assembly program is written such that it will read "sector 0" and "sector 1" of the hard disk and upload them respectively to address range 639-512 and 767-640. The program will use the interrupt

given from the hard disk in order to notify the simulator that the DMA has finished the transaction, afterwards it will calculate the sum of its first 8 words and store it in addresses 0x100 and 0x101 respectively. Address 0x102 will hold the greater sum of them aswell.
Requires: memin.txt, diskin.txt