Computer Structure Project – Simulator

The simulator was built in two files $sim.c$ and $sim.h$ that work together to simulate the actions of SIMP processor.

sim.h

- Imported libraries: “stdio.h”, “stdint.h”, “stdlib.h”.
- Defined data structure:
  - Linked list “struct log” that contains 3 “head-tail” pointers.
    1. struct status – documentation of executed instructions in each cycle.
       The number of executed instructions is unknown; hence a linked list is essential.
       Interesting fields within the struct:
       - unsigned long cycle – non-negative large number.
       - uint16_t pc – 12-bit address.
       - uint64_t inst – 48-bit number.
       - int32_t r[16] – 16 registers s.t. each contains 32-bit signed number.
    2. struct hw_access – documentation of every user hardware access through in/out commands. The number of hardware accesses is unknown; hence a linked list is essential. Interesting fields within the struct:
       - unsigned long cycle – non-negative large number.
       - uint8_t rw – type of hardware access {READ : 1, WRITE : 2}.
       - uint8_t IOReg – hardware port that have been accessed $IOReg \in [0,22]$.
       - uint32_t data – at most 32-bit value of the hardware register after the action.
    3. struct irq2in – documentation of all the coming irq2in interrupts. Here a linked list is not essential and can be done with an array and have better I/O performances. Interesting fields within the struct:
       - unsigned long cycle – cycle which raise irq2in.

sim.c

Data structure (global variables):

- uint16_t pc – 12-bit address of the current program counter.
- uint8_t irq_busy – 1 if currently handling interrupt, else 0
- unsigned long dist_last_cmd_cycle – used to simulate 1024 cycles s.t. the disk is unavailable.
- uint64_t i_mem[4096] – instruction at pc i located at index i. Each instruction is 48-bit.
- int32_t d_mem[4096] – data at address i located at index i. Each slot is 32-bit wide. Could be referred to as $memory[4096]$ In the following documentation.
- uint32_t r[16] – internal registers data. Each register is 32-bit wide.
- uint32_t IORegister[23] – IO registers data, each register is at most 32-bit wide.
- uint32_t disk[128][128] – memory i at sector s is allocated at $disk[s]\left[\left\lfloor \frac{i}{32} \right\rfloor + i\%32\right]$. meaning at sector s the memory located at row $\left\lfloor \frac{i}{32} \right\rfloor$ and bit $i\%32$ within the current sector. Since every row is 32-bit wide.
- uint8_t monitor[256][256] – pixel in coordinates (x, y) is allocated in $monitor[y][x]$.
- struct log data_log – used to track each needed step of the procedure for the output files, as elaborated at “sim.h”.
- unsigned long cycles – number of cycles the program made.
- Const char* const IOLabels[] – I/O labels for output file “hwregtrace”.

The main procedure:

1. Initialization
   a. Initialize “head” pointers to $NULL$ and global variables counters to 0.
   b. Read 4 input files.
2. Main loop
   a. Execute instruction.
   b. Handle monitor.
   c. Handle timer.
   d. Handle disk.
   e. Interrupt service routine.
3. Finalization
   a. Write 9 output files.
   b. Free dynamically allocated linked list: “status”, “hw_access”, “irq2in”.

Error handling – in case of dynamic allocation fault or file open/close fault, corresponding error message is raised and 1 is returned. Used with macro $err\_msg(msg)$ that prints into $stderr$ the following message:

$$Error: < msg >$$
$$pc: \ < pc >$$
$$line: < line >$$

Main Loop

$execute\_instruction()$ – Takes the current instruction $i\_mem[pc]$ and breaks it into 7 parts:

$$[opcode, rd, rs, rt, rm, imm1, imm2]$$

The function calls $update\_log\_status()$ to save the current machine state.
Additionally performs $\$0 \leftarrow 0$ (constant value) and $extend\_sign()$ to $\$imm1, \$imm2$.
Then decides which operation needed to be executed by the assignment documentation.
In case of invalid opcode the function returns suitable code and the program finishes with an error $err\_msg("Invalid\ opcode")$.

In case of I/O instruction, the function calls $update\_log\_hw\_access()$ to save the current hardware access.

$update\_log\_state(), update\_log\_hw\_access()$ – Both functions dynamically allocate new struct that plays as a new node in each linked list. Saves the corresponding values of it and insert the new node into the tail of its corresponding linked list.

$extend\_sign(uint32\_t\ reg, uint8\_t\ sign\_bit)$ – $sign\_bit \in [0,31]$, the function extend sign to the received reg's value s.t. the sign bit is at $sign\_bit$ index, starting from 0 LSB.

$handle\_monitor()$ – in case of $monitorcmd == 1$ (write command) the function assign $monitordata$ to $monitor[row][col]$ s.t. $col = monitoraddr_{[0:7]}, row = monitoraddr_{[8:15]}$.

$handle\_timer()$ – in case of $timerenable == 1$ checks if $timercurrent$ reached $timermax$, if so, resets $timercurrent \leftarrow 0$ and raises $irq0status \leftarrow 1$ Otherwise $timercurrent += 1$.

$handle\_disk()$ – check if 1024 cycles have passed since last disk command, if so perform $diskstatus \leftarrow 0$ and $irq1status \leftarrow 1$. After that, if $diskcmd \in \{1, 2\}$ and the disk is not busy, the function copies $disk[disksector]$ to $memory[diskbuffer]$.

$interrupt\_service\_routine()$ – in case $irq\_busy == 0$ (irq is available), the function first get the next irq2in into $irq2status$ with $check\_irq2in()$ function, then check if an interrupt occurs by:

$$irq \leftarrow (irq0enable \,\&\, irq0status) \mid (irq1enable \,\&irq1status) \mid (irq2enable \,\&irq2status)$$

If $irq == 1$ then moves $irqreturn \leftarrow pc;\ pc \leftarrow irqhandler$

$tick\_clk()$ – performs $clks += 1; cycles += 1$.

Initialization and Finalization

$read\_file\_dmem\_imem(dmemin\_path, imemin\_path)$ – reads the contents of $dmemin$ and $imemin$ into the data structures d_mem and i_mem respectively.

$read\_file\_diskin(diskin\_path)$ – reads the contents of $diskin$ into the data structure $disk.$

$read\_file\_irq2in(irq\_path)$ – reads the contents of $diskin$ into the data structure $disk.$

$write\_file\_dmemout(dmemout\_path)$ – write $memory$'s content into the file.

$write\_file\_diskout(diskout\_path)$ – write $disk$'s content into the file.

$write\_file\_trace(trace\_path)$ – write $data\_log.status\_head$ content into the file.

$write\_file\_hwregtrace\_leds\_display7seg(hwregtrace, leds, display7seg)$ –

- Write $data\_log.hw\_head$ into $hwregtrace$ file.
- Filter $data\_log.hw\_head$ data by $IOReg == leds$ and write it into $leds$ file.
- Filter $data\_log.hw\_head$ data by $IOReg == display7seg$ and write it into $display7seg$ file.

$write\_file\_cycles\_regout(cycles\_path, regout\_path)$ – write $cycles$ and r array into the paths respectively.

$write\_file\_monitor(monitor\_path, is\_binary)$ – if $is\_binary == 0$, writes $monitor$ into the file path as text. Otherwise writes it as binary file. Used to write $monitor.txt$ and $monitor.yuv.$

$free\_log\_status(), free\_log\_hw\_access(), free\_log\_irq2in()$ – Free dynamically allocated nodes in the linked lists.