University of Puerto Rico – Mayagüez
School of Engineering
Department of Electrical and Computer Engineering


INEL 4206 – Microprocessors – Spring 2002

Problem Set 4 – Due Monday April 15, 2002 (In class)


**Important: Start to work on this problem set early. Do not leave this to the last minute or you will not be able to complete the work.**

In this problem set you will implement a simulator for a modified version of the Easy I architecture discussed in class. The simulator will run on SPIM. It must be able to interpret a sequence of Easy I instructions stored in memory and should accomplish the same functionality as if the Easy I program was run natively on an Easy I processor.

You should structure your simulator using procedures. In general you should implement one procedure to simulate each Easy I instruction. Also you should add procedures to fetch and decode instructions as well as to fetch the operands. You may add more procedures as you see fit to achieve a well structured design.

You must remember that the Easy I is a byte addressable accumulator architecture with 16-bit instructions and 16-bit word size. Other important details about the Easy I architecture are provided in the following pages.

## Description of the main procedures

| Procedure name | Contract |
| --- | --- |
| reset | Initializes Easy I registers |
| fetch | Fills instruction register with next instruction |
| decode | Decides which execute procedure to call based on the opcode |
| fetchop | Fills data buffer register with operand<br><br>Only applies for indirect mode instructions |
| comp | One-Complements the accumulator |
| shr | Shift right accumulator one bit |
| brn | Branch if accumulator is negative. Target address refers to Easy I data segment |
| jump | Unconditional jump to target address within Easy I text segment |
| jal | Same as jump, but saves PC+2 in accumulator. NEW EASY I INSTRUCTION. |
| jac | Unconditional jump to address contained in accumulator. NEW EASY I INSTRUCTION. |
| store | Stores accumulator in memory data segment |
| load | Load accumulator from memory data segment |
| and | Bitwise AND accumulator with operand. Put result in accumulator. |
| add | Add accumulator with operand. Put result in accumulator. |
| loadsp | Move stack pointer to accumulator. NEW EASY I INSTRUCTION. |
| storesp | Move accumulator to stack pointer. NEW EASY I INSTRUCTION. |
| stop | Stops the simulation. |
| run | Performs Easy I simulation. Should call reset and then loop through each Easy I instruction. Should finally call stop and then return. |

**Easy I Memory Model**

To keep the project as simple as possible you may assume that the Easy I program, data segments and stack segment will be stored at fixed locations within the MIPS data segment as follows:

| Easy I segment | MIPS data segment address - All 64K long |
|---|---|
| Text segment | 0x10000000 - 0x1000FFFF |
| Data Segment | 0x10010000 - 0x1001FFFF |
| Stack segment | 0x10020000 - 0x1002FFFF |

You may also assume that the Easy I will run on a 16-bit address space. That is, it may access memory locations 0 through 65535.  Therefore, address 0 of the Easy I data segment would map to the first memory location inside the MIPS data segment where the Easy I data segment resides.

**Register Allocation**

You must use the following register allocation in order to keep all projects as uniform as possible.  This will facilitate discussion among students as well as grading.  Notice that the modified Easy I architecture has a stack pointer (SP). Also notice that the simulator will need to simulate both the programmer-visible registers, like the AC, as well as the hidden ones like the PC.  The instruction register holds the instruction currently being executed (i.e. simulated).

| Easy I Register | MIPS register |
|---|---|
| Instruction register | $s0 |
| Program counter | $s1 |
| Accumulator | $s3 |
| Address Buffer Register | $s4 |
| Data Buffer Register | $s5 |
| Stack pointer | $s6 |

## Easy I Instruction Format

The format of an easy one instruction will be identical to the one discussed in class.
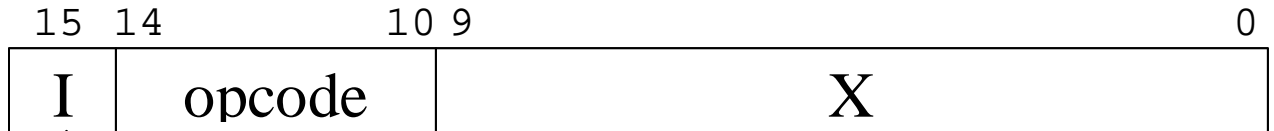


## Easy I Instruction Set

The following table describes the full set of instructions that your simulator should be able to execute.

| Name | Opcode | Action I= 0 | Action I = 1 |
|------|--------|-------------|--------------|
| Comp | 00 000 | AC ← not AC | Same as I = 0 |
| shR | 00 001 | AC ← AC / 2 | Same as I = 0 |
| BrN | 00 010 | If (AC < 0): PC ← X | If (AC < 0): PC ← MEM[X] |
| Jump | 00 011 | PC ← X | PC ← MEM[X] |
| Store | 00 100 | MEM[X] ← AC | MEM[MEM[X]] ← AC |
| Load | 00 101 | AC ← MEM[X] | AC ← MEM[MEM[X]] |
| And | 00 110 | AC ← AC and X | AC ← AC and MEM[X] |
| Add | 00 111 | AC ← AC + X | AC ← AC + MEM[X] |
| Jal | 01 000 | AC ← PC+2; PC ← X | AC ← PC+2; PC ← MEM[X] |
| Jac | 01 001 | PC ← AC | PC ← MEM[AC] |
| loadSp | 01 010 | AC ← SP | AC ← MEM[SP] |
| storeSp | 01 011 | SP ← AC | MEM[SP] ← AC |

**Testing your program and Grading**

A set of test programs will be provided to you within the next few days. The tests will consist of sample program segments in Easy I machine code that your simulator should be able to execute to completion. The state of the Easy I processor at the end of each segment should emulate that of a real Easy I processor running the same set of instructions.

**Your program must pass some minimal tests in order to qualify for grading.**

**Your program will be graded based on correctness, quality and efficiency as described in the "prontuario" of the course.**