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

INEL 4206 – Microprocessors – Spring 2003

Problem Set 4– Simulator for Extended Easy I Architecture

Due Thursday May 8, 2003 (via electronic submit)

Important: Start working 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 will be provided a template assembly language file to start from.

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 simulator procedures

Procedure name	Description
reset	Initializes Easy I registers
run	Performs Easy I simulation. Should call reset and then loop through each Easy I instruction performing the full fetch-decode-execute cycle.
fetch	Fills instruction register with next instruction
decode	Decides which execute procedure to call based on the opcode
execute	Executes each instruction according to opcode
fetchop	Fills data buffer register with indirect operand. Only called for indirect mode instructions.
fcomp	One-Complements the accumulator
fshr	Shift right accumulator one bit
fbrn	Branch if accumulator is negative. Target address refers to Easy I data segment
fjump	Unconditional jump to target address within Easy I text segment
fjal	Same as jump, but saves PC+2 in accumulator. NEW EASY I INSTRUCTION.
fjac	Unconditional jump to address contained in accumulator. NEW EASY I INSTRUCTION.
fstore	Stores accumulator in memory data segment
fload	Load accumulator from memory data segment
fand	Bitwise AND accumulator with operand. Put result in accumulator.
fadd	Add accumulator with operand. Put result in accumulator.
floadsp	Move stack pointer to accumulator. NEW EASY I INSTRUCTION.
fstoresp	Move accumulator to stack pointer. NEW EASY I INSTRUCTION.

Easy I Memory Model

To keep the project as simple as possible you may assume that the Easy I text, data and stack segments 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 begins.

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

Extended Easy I Instruction Format

The format of an easy one instruction will be identical to the one discussed in class. Please refer to the Easy I Quick Reference Sheet for details.

http://www.ece.uprm.edu/~bvelez/courses/Spring2003/INEL4206/lecturas/easy1-sheet.pdf

Extended 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 correctly 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 and will be graded based on correctness, quality and efficiency as described in the "prontuario" of the course.