

Homework 7

Due: November 18, 11:59pm

MIPS Simulator**1 Description**

Write a **Java** program that simulates the execution of a MIPS program. Your simulator will read a “binary” file that contains hexadecimal representations of the text and initialized data segments (see Section 2). The program will utilize only a subset of the MIPS R2000 instruction set (see below). Your simulator should provide two modes: single step and run-to-completion. During single-step you should be able to print out the value of a specified register or to print all registers (see Section 4 for more details on the interface). In general, your program will perform the fetch, decode execute cycle, reading instructions from memory, decoding them, and performing the appropriate operation on “simulated” registers and/or memory.

You will do two things to test your simulator. First, you will run a "torture test" program that is provided in <http://kedem.cs.duke.edu/cps104/Homework.html>. Running this program in single step mode will help isolate problems. You should also run the “Fibonacci” program from homework-2 and “Tower of Hanoi” through your simulator. Make sure to “correct” these programs so they terminate via a syscall. Section 5 provides more information on what to turn in.

Although we have reasoned about memory as a single linear array in class, you can not allocate memory in your simulator as a single array. Instead, you should allocate four separate arrays for the text, static data, dynamic data, and stack segments, respectively. You must read the instructions and static (initialized) data from the “binary” file into the corresponding simulator arrays.

*You can assume the text segment will not exceed 2 kilobytes (2*1024), the static data segment will not exceed 4 kilobytes.*

Be sure to consider that from the program’s perspective, the text segment begins at address **0x00400000** and the static data segment begins at address **0x10010000**.

The program expects a valid stack pointer at address **0x7fffffff**, (so the first valid aligned stack address is **0x7fffffff0**, which is the value the stack pointer should have when the program begins execution). When the program accesses one of these segments, you must compute the appropriate offset into your “simulated” segment.

2 MIPS Binary File**Generating the Binary File**

To generate the binary files from your assembly programs, Use the web interface: <http://www.cs.duke.edu/~kedem/asmr2000.html> to create an input file for your program.

The input file format is described below.

Binary File Format

The binary files will have a marker (DATA SEGMENT) that indicates where the code (text) segment ends and the static data segment begins—you need to jump to loading the data into address **0x10010000** at that point. Also, since I don’t want to include all 4K entries of the static data segment, I only list those which are not 0; I list their address followed by the data in word-aligned format. As an example, "sum.s" that we looked at early in the semester lists as follows (see the example in the Homework section of the course web page for the source):

0x27bdf8d8
0xafbf0024
0xafb30020
0xafb2001c
0xafb10018
0xafb00014
0x00001021
0x00008821
0x3c011001
0x34300000
0x3c011001
0x34320024
0x3c011001
0x34330024
0x8e0e0000
0x022e8821
0x34020004
0x00122021
0x0000000c
0x34020001
0x00112021
0x0000000c
0x34020004
0x3c011001
0x34240030
0x0000000c
0x26100004
0x1613fff3
0x00001021
0x8fb00014
0x8fb10018
0x8fb2001c
0x8fb30020
0x8fbf0024
0x27bd0028
0x03e00008

DATA SEGMENT

0x10010000 0x00000023
0x10010004 0x00000010
0x10010008 0x0000002a
0x1001000c 0x00000013
0x10010010 0x00000037
0x10010014 0x0000005b
0x10010018 0x00000018
0x1001001c 0x0000003d
0x10010020 0x00000035
0x10010024 0x54686520
0x10010028 0x73756d20
0x1001002c 0x69732000
0x10010030 0x0a000000

3 Instructions you must Simulate

Below is the complete list of instructions that you must correctly simulate. Notice there are no floating point or coprocessor instructions, but that syscall is a requirement. As you are simulating actual hardware, you will not simulate pseudoinstructions (like LI: load immediate, or MOVE) which are translated by the assembler into actual instructions (usually ORI: OR-immediate in the case of LI) - you will only simulate "real" instructions that have an opcode. In your programs, you can continue to use pseudoinstructions, as the assembler will continue to make all the translations SPIM would make. Use **Appendix B** to obtain the corresponding OPCODES, and instruction formats.

You do not need to simulate the actual hardware execution of each individual operation. For example, to simulate **add \$a0,\$a1,\$a2**, you can use the + operator in **Java**, to compute **\$a0 = \$a1 + \$a2**, where **\$a0**, **\$a1**, & **\$a3** are "simulated" registers. Similarly, *****, **/**, **-**, **^**, **<<**, **>>**, etc., can be used to "simulate" instruction execution.

You need to simulate system call values: 1, 4, 5, 8 and 10. That is: **Print_Int**, **Print_String**, **Read_Int**, **Read_String**, and **Exit**.

Below is a table with MIPS instruction that your program must be able to simulate. These are the same instructions you used for Homework-6

Instruction	Example
Store Word	sw \$fp,0x0008(\$sp)
Load word	lw \$t5,0x0000(\$t3)
Load upper immediate	lui \$t0,0x5555
Add	add \$t0,\$0,\$0
Subtract	sub \$t6,\$t0,\$t0
Subtract unsigned	subu \$sp,\$sp,\$a0
Add unsigned	addu \$fp,\$sp,\$a0
Add immediate	addi \$t0,\$0,0x0001
Add immediate unsigned	addiu \$t0,\$v0,0x0001
AND	and \$t0,\$t3,\$t6
OR	or \$t5,\$t2,\$t5
NOR (not OR)	nor \$t3,\$t1,\$t4
XOR	xor \$s2,\$t6,\$t0
OR immediate	ori \$t4,\$t0,0x5555
AND immediate	andi \$s6,\$s6,0x3333
XOR immediate	xori \$t2,\$t2,0x5678
Branch equal	beq \$s5,\$0,label
Branch not equal	bne \$s1,\$t0,label
Branch greater-equal 0	bgez \$s3,label
Branch less than 0	bltz \$s4,label
Branch less equal 0	blez \$t1,label
Branch greater than 0	bgtz \$t1,label
Jump	j label
Jump register	jr \$ra
Jump and link	jal label
Jump and link register	jalr \$t1
Shift left logical	sll \$v1,\$a0,1
Shift left logical variable	sllv \$t0,\$t0,\$t4
Shift right logical	srl \$t6,\$a2,1
Shift right logical variable	srlv \$s6,\$s6,\$t1
Shift right arithmetic	sra \$t7,\$a1,1
Shift right arithmetic variable	srav \$s6,\$s6,\$t1
Set less than	slt \$t2,\$t0,\$t1
Set less than unsigned	sltu \$t3,\$t0,\$t1
Set less than immediate	slti \$t4,\$t0,0xffff

Set less than immediate unsigned	sltui \$t5,\$t0,0xffffb
System call	syscall

4 Command Line Interface

Your simulator must provide two modes of operation: single step, and run to completion. Programs terminate by using the *exit syscall*. In single step mode, your simulator should print the assembly representation of the instruction that will be executed. Also, in single step mode you must support the following commands:

- p *reg* print a specific register (e.g., p 4, prints the contents in hex of register 4)
- p *all* print the contents of all registers, including the PC, in hex
- p *addr* print the contents of memory location *addr* in hex, assume *addr* is a word address in hex.
- s *n* execute the next n instructions and stop (should print each instruction executed)

5 What to Submit

You must turn in the output produced from the execution of the program "TestP.s" (to be provided later). You must also submit the Java code of your simulator and a concise (1-2) page description of the principles of operation of your simulator, any design decisions or "interpretations" of the SPIM specs you make, operation of instructions, etc.