CSE 378 Spring 2003

Machine Organization and Assembly Language Programming

Problem Set #5

Due: Friday May 9th

In this assignment you'll use the Smok tool to put together a single cycle implementation of the MIPS datapath. This is a prelude to the pipeline implementation so you might want to keep this in mind when defining components/containers.

You can work in teams of two if you so desire.

The machine you build should be able to calculate the factorial of a given number, i.e., you need to implement the following instructions (maybe not all of the arithmetic ones are needed but it won't be extra work to do it!):

- **Arithmetic**: ADD, ADDU, SUB, SUBU, MUL (Note that the MUL given by the Smok tool is different from the one given by the MIPS implementation. Follow the easy route, i.e., the Smok one. Also, Smok's ALU has different opcodes than those given by the function bits of the MIPS opcode. It is quite OK to, again, follow the easy route i.e., follow the Smok ALU convention).
- **Immediate instructions**: ADDI and anything you need to implement LI, e.g., ORI.
- **Load Store instructions**: LW and SW.
- **Transfer of control**: JAL, JR, BEQ, BNE.

The code for factorial is given in MIPS assembly language on a following page (it computes 5! but we might want to test your program on different values). The binary, i.e., the .smokmem file is also on a following page. (If you look at the .html version of this assignment, you'll find the links to download the programs.) Note that while MIPS programs start with a PC of 0x04000000, the Smok tool expects a beginning PC of 0x00000000.

Your 32 registers and the PC should be like MIPS's, i.e., 32-bit long. However, we are going to restrict the amount of memory to 1024 Bytes, i.e., 256 words. This explains the beginning of the main program calling the factorial function, i.e., the top of the stack is now at address 1024 (minus 4) and will grow toward address 0.

If you need static and dynamic data, you should start storing them at the 512 byte (128 word) boundary.

In order for us to test your implementation more easily, you should include a "halt" component that will be triggered when the self-loop of the main program is reached.

A few Smok-MIPS quirks.

- Beware that SPIM (in the default configuration) doesn't calculate branch offsets from PC+4 (it calculates them from PC), and that jump addresses will be different if you assemble the program with SPIM. The .smokmem file uses a displacement from PC+4, compatible with the real MIPS implementation.

- In the single cycle implementation of MIPS, there are separate memories for instruction and data. In Smok you can use the same memory with different interfaces: an IF for instruction fetch and a regular Memory Interface for the data memory. Whether you use 1 or 2 memories is your choice.
- Although Smok provides limited I/O capabilities through direct mapped I/O controllers, you are not asked to implement this. It would require the use of library routines, syscalls, and more generally the Cebollita system (you are welcomed to try it for extra credit). At the end of the computation, your result should be in register $v0.
- When you reinitialize the Smok machine, all registers are reset to 0. Whether you want to hardwire register 0 to zero is your choice.
- Testing your machine. It might be useful to test each instruction type one at a time (e.g., build a one instruction .smokmem file).

What you have to **turnin**.

- a README file indicating briefly how to "operate" your machine in case it is not obvious, i.e., which files to load etc.. In case you are aware of bugs and did not have time to correct them, indicate it in the README file.

- a .smok file for your implementation (and any .smokcont for containers), a .smokpla file for the PLA, a .smokmem for the memory.

- More specifically, send e-mail to Tapan (tapan@cs) with subject 378ASS5, including files LASTNAME.README, LASTNAME.smok, LASTNAME.smokmem, and LASTNAME.smokpla. Also LASTNAME-ComponentType.smokcont for any required containers.

One turnin per set of partners is enough but be sure to indicate both names in the README file.

```
main:   # Will start at PC 0x00000000
#
# Main program entry
#
        addi $sp,$0,1024    #the stack will start at (byte) address
        addi    $sp,$sp,-4     #1024 and grow downwards
        li      $a0, 5          #will compute factorial of 5
        jal     fact
                addi    $sp,$sp,4       #restores the stack
                beq     $0,$0,-1        # a "halt" instruction
# PROCEDURE:  Fact
# PARAMETERS:  an integer in a0
# LOCALS:
# RETURN:  fact(a0) in v0
# DESCRIPTION:  Computes recursively factorial
#
fact:            sw      $ra,0($sp)      #save return address
         addi    $sp,$sp,-4      #move sp to new position in frame
#
# Procedure body
#
# test argument for 0 ;  if so return with value 1
         beq     $a0,$0,nore     #test a0 ;  if zero go to out
# otherwise save a0 on the stack
         sw      $a0,0($sp)      #push argument on stack
         addi    $sp,$sp,-4
         addi     $a0,$a0,-1     #decrement argument
#
#recursive entry
#
        jal     fact            #call
#
#back from recursive level v0 =v 0 * ao
#
        addu    $sp,$sp,4
        lw      $t0,0($sp)      #pop argument from stack
        mul     $v0,$v0,$t0     #get result at this level
        b       out             #exit that level
#
#end of recursion
#
nore:         li      $v0,1           #result in v0 and exit
#
# Procedure exit
#
out:         addi    $sp,$sp,4
        lw      $ra,0($sp)      #restore return address
        jr      $ra             #return
```

The .smokmem file

```
@0
201d0400
23bdfffc
20040005
0c000006
23bd0004
1000ffff
afbf0000
23bdfffc
10800008
afa40000
23bdfffc
2084ffff
0c000006
23bd0004
8fa80000
00481018
10000001
34020001
23bd0004
8fbf0000
03e00008
```