CMPE 140 Lab 7

Dr. Donald Hung

Computer Engineering Department, San Jose State University

Enhanced Single-cycle MIPS Processor

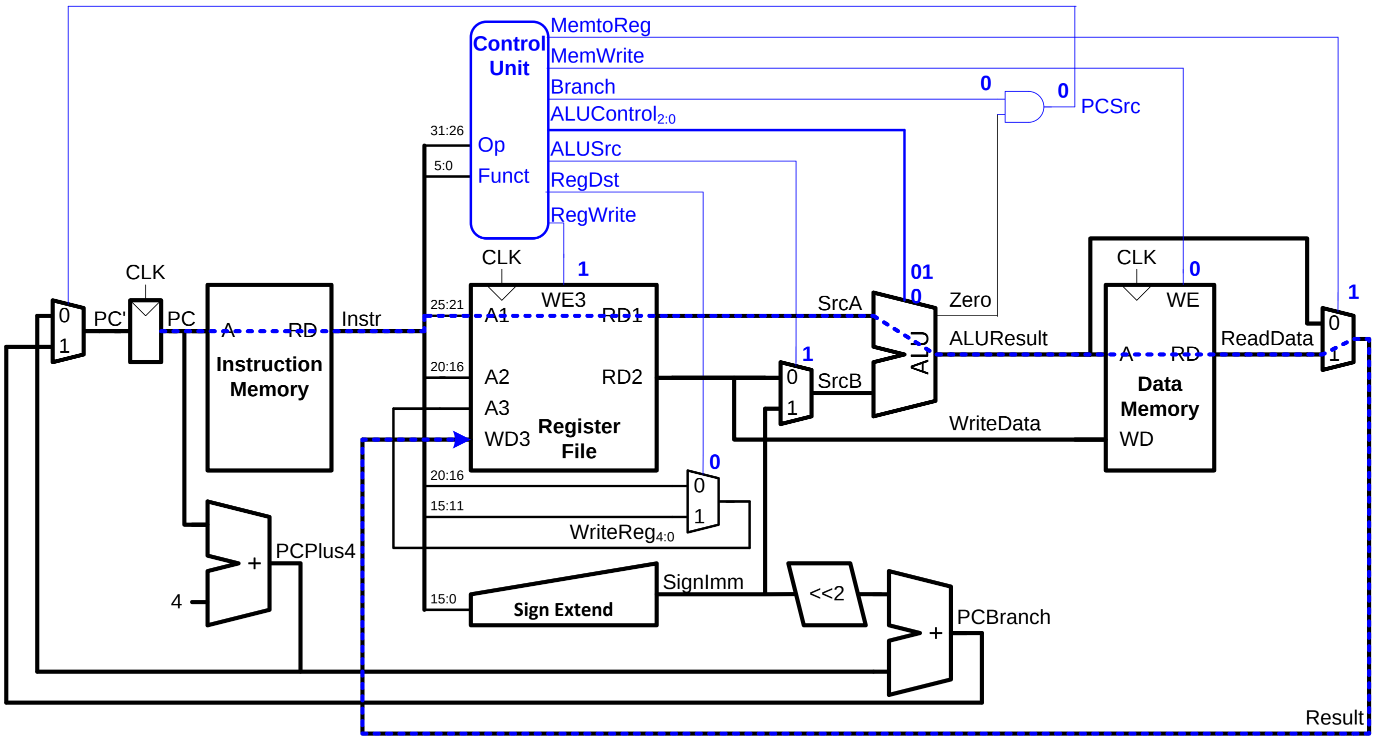
Description:

Based on the initial design of the single-cycle MIPS processor tested in Assignments 5 and 6, in this midterm exam you will extend the initial design to support more MIPS instructions. Specifically, you are required to enhance the single-cycle MIPS processor's functionality by extending its instruction support from the initial sub set {`ADD, SUB, AND, OR, SLT, LW, SW, BEQ, J, ADDI`} to cover the following additional instructions: `MULTU, MFHI, MFLO, JR, JAL.`

Your design must be tested via both functional verification and FPGA validation. Schematics of the initial design and the test program are attached as Attachment 1 and Attachment 2 respectively. As a reminder, machine code of a test programs should be stored in the memory file named *memfile.dat*, which should be placed in your project directory. This file should contain one 32-bit machine code (in hex) per line (unused memory space should be filled with zero's), and if you change any content of this file, you will have to re-synthesize and implement the entire design before programming the FPGA on the Nexys2 board.

You should document your work professionally. In addition to descriptions and discussions, your report should include schematics (use of Microsoft Visio recommended) for the enhanced processor microarchitecture, tables for control decoders, simulation log and/or necessary waveforms, and commented source code. **Any changes (modification and/or extension) to the initial design (figure and source code) must be in a different color.**

Attachment 1: Initial Design of the Single-cycle MIPS (w/o jump)



Attachment 2: Test Program I

Test code for extended design (Again, use the MIPS assembler/simulator to test it first!):

```
.data
.globl main
.text
main:
     # addi $sp, $0, 48        not for SPIM
       addi $a0, $0, 4      # set arg
       jal factorial        # compute the factorial
       add $s0, $v0, $0     # move result into $s0
       j end
factorial:
       addi $sp, $sp, -8    # make room on stack
       sw $a0, 4($sp)       # store $a0
       sw $ra, 0($sp)       # store $ra
       addi $t0, $0, 2      # $t0 = 2
       slt $t0, $a0, $t0    # a <= 1 ?
       beq $t0, $0, else    # no - goto else
       addi $v0, $0, 1      # yes - return 1
       addi $sp, $sp, 8     # restore $sp
       jr $ra               # return
else:
       addi $a0, $a0, -1    # n = n - 1
       jal factorial        # recursive call
       lw $ra, 0($sp)       # restore $ra
       lw $a0, 4($sp)       # restore $a0
       addi $sp, $sp, 8     # restore $sp
       mult $a0, $v0        # n * factorial(n-1)
       mflo $v0             # mv result into $v0
       jr $ra
end:
```