Ian Zhang
netid: icz
Partner: Jimmy Zhang
Partner net id: jz100
Computer Science 250
Hours spent: way too many. Around 21, 22 hours.

Lovely, lovely, grader. I beseech you. Be gentle and caress this circuit and my grade.

Okay here we go:

1. Only correctly processes simple.s until the branch instruction 008a, about 20 “clock ticks” into the program.
2. All individual instructions except branch, lw, and input/output work correctly, that is, if you connect the “connect to test” 16 bit manual input and test the load, store, add/sub, etc. instructions the correct values will appear in the registers.
3. Description of the circuits (from decoder -> ALU -> main)

Overall flow (intended lol):

16 bit instruction -> Decoder

Decoder:
1. Receives 16 bit input instruction from PC.
2. Outputs 2 sets of data
    -the instruction parsing (i.e., rs, rd, rt, shamt, immediate, jump address, input, output, and opcode)
    -the control outputs based on the opcode (for my processor, I have implemented write enable, write to register, use immediate, write to mem, branch, memory to register, use j address, Use PC + 1, and constant $r7)

The decoder then propagates values several values through to the ALU, to the PC, or to the Data Memory (ROM), increments the PC for j and JAL instructions, or sets PC to the value in $r7 (the return instruction).

ALU:
1. Receives op code from decoder, shamt from decoder, S register value from the register file, and either the T register value from the register file or the 16 bit sign extended immediate value from the decoder.
2. Outputs a 16 bit value based on the opcode instruction received from decoder.
    These will be the instructions: add, ldi, sub, and, xor, rotl, rotr, lw, sw, bnez, input, and output
3. 16 bit value is then propagated out to the data memory, the comparator gate (for branch), or the TTY output.

What follows is a line by line description of my data paths for each instruction:

add 0001 R add $rd, $rs, $rt $rd=$rs+$rt

The add instruction enters the decoder. Based on the opcode the control signal for write enable is turned on. We now look to the ALU, which has the opcode propagated through. This selects the 0001 option of the mux, which adds the values from Register T and Register S, then outputs it. At this point the 16 bit added value goes to three areas:

1. A mux which determines whether we need to write the data stored in memory location determined by output into the register determined by the instruction set. If not enabled, this output will be enter another mux, which determines whether or not it should receive an input from the keyboard. It then enters a final mux which is enabled only when the instruction for return is processed. Since none of these muxes will be enabled, the output from the ALU will be written into the register determined by rD in the instruction.

Working

2. The TTY output (only prints out something if output instruction is processed)

Not-working (guess: need to put this through another mux that occurs before address port of RAM that will help parse ALU output correctly—i.e. taking bottom 7 bits)

3. The address port of the RAM.

Working

ldi 0010 I ldi $rt, Imm $rt=Imm

The load immediate instruction enters the decoder. Based on the opcode, the control signals for write enable, write to register, and use immediate are turned on. We now look to the ALU, which has the opcode propagated through. This selects the 0010 option of the mux, which takes the 16 bit sign extended immediate value (instead of the value in the T register because of the immediate enable control) and propagates it through to the ALU output. At this point the 16 bit immediate value will go through the mux pathing as described above. All are unactivated therefore will write into register as determined by rT in instruction.

Working.

sub 0011 R sub $rd, $rs, $rt $rd=$rs-$rt

Same as add, but ALU op code will have S register value subtract T register value instead of added.

Working.

and 0100 R and $rd, $rs, $rt $rd = $rs AND $rt

Same as and, but ALU op code will AND the S and T registers instead of add.

Working.

xor 0101 R xor $rd, $rs, $rt $rd = $rs XOR $rt

Same as and, but ALU op code will XOR the S and T registers instead of add.

Working.

rotl 0110 R rotl $rd, $rs, <shamt> $rd = $rs rotated <shamt> to left

Working.

rotr 0111 R rotr $rd, $rs, <shamt> $rd = $rs rotated <shamt> to right

Working.

lw 1000 I lw $rt, D($rs) $rt = Mem[$rs+D]

Same as add, but ALU op code will add the sign extended 16 bit immediate value to the register value in S.

Not-working (guess: not implementing immediate add to $rs correctly in mux closest to the PC)

sw 1001 I sw $rt, D($rs) Mem[$rs+D] = $rt

Same as lw.

Working.

bnez 1010 I bnez $rs, B if ($rs!=0) then PC=PC+1+b

Same as add, but ALU op code will propagate the immediate value through, then add this output to the PC + 1 (using the adder up near the top left), then carries the value through to the PC.

Not-working. (guess: So this one is good up until about 20 clock ticks in, and is the source of my infinite loop (I think). So I feed the opcode into the ALU, and the immediate value paths through the muxes until it gets added to the PC+1. In the following mux, I'm enabling a PC+1 OR a PC+1 and Jump Location. So this is definitely where I'm messing up. I think the logic is more towards, mux for PC+1, mux for PC+1+immediate+$rs, then enable those values through separate op control).

j 1011 J j L PC = L (upper 4 bits same)

Sends jump address from decoder to PC.

Working.

ret 1100 R ret PC=$r7

Enables constant $r7, takes the value from register 7, then sends it to PC.

Working.

jal 1101 J jal L $r7=PC+1; PC = L

Not working. (guess: PC+1 mux pathing is faulty, see bnez for more on this)

input 1110 I input $rt $rt = keyboard input

Not working. Sweet baby jesus. Could be a million things.

output 1111 I output $rs print $rs on a TTY display

See above.