Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Profs. Archana Chidanandan and Larry Merkle

Homework 3 Solutions
Register Transfer Language
Max Points: 45 points

Directions

This assignment is due Tuesday, October 10 for all three sections. Submit your solutions on a separate sheet of paper. You may use SPIM and a calculator as required.

Learning Objectives

In the process of completing this homework assignment, students will develop their abilities to

- Describe the implementation of machine language instructions using Register Transfer Language.
- Identify the components required to implement machine language instructions, including their input, output, and control signals, as well as their high-level behavior.
- Determine how to handle exceptions, including interrupts.

Problems

1. [15 pts] Consider the MIPS instructions `jr`, `sll`, and `lui`.

   a. Write a multicycle RTL description of an implementation of each instruction that uses as few cycles as possible without extending the clock cycle of your design.

   The first two cycles are the same for all instructions:

   1. IR = Mem[PC]
      PC = PC + 4
   2. A = Reg[IR[25 : 21]]
      B = Reg[IR[20 : 16]]
      ALUout = PC + SE(IR[15 : 0] || 00)

   `jr`:

   3. if((IR[31 : 26] == 0) && (IR[5 : 0] == 8)) then
      PC = A

   `sll`:

   3. if((IR[31 : 26] == 0) && (IR[5 : 0] == 0)) then
      ALUout = A[31 – IR[10 .. 6] : 0] || 00…00
   4. Reg[IR[15 : 11]] = ALUout

   `lui`:

   3. if(IR[31 : 26] == 15) then
      ALUout = IR[15 : 0] || 0x0000
   4. Reg[IR[15 : 11]] = ALUout

b. Identify any "new components" required to implement each of the instructions. Describe their input, output, and control signals, as well as their high-level behavior. In this context, "new components" means components that are not shown in Figure 5.28 of Patterson & Hennessy.

`jr`: no new components are required.

`sll`: a shifter is required (or the ALU must be modified to include a shifter). The shifter has a 32-bit input, a 5-bit control signal, and a 32-bit output. The control signal is interpreted as an unsigned integer shamt. The output is determined by concatenating bits 31 – shamt through 0 of the input with shamt zeroes.

`lui`: a concatenator is required. It has a 16-bit input and a 32-bit output. The output is determined by concatenating bits 15 through 0 of the input with 16 zeroes. Note that the implementation of this component is so trivial that it hardly deserves to be called a component.

2. [10 pts] Consider the MIPS instructions `mfc0`, and `mtc0`. Write a multicycle RTL description of an implementation of each instruction that uses as few cycles as possible without extending the clock cycle of your design. You are NOT required to identify the new components. Hint: Page A71 (on the CD) of Hennessey and Patterson, has descriptions for these two instructions.

   `mfc0`:
   2. (in addition to RTL statements shown above for cycle 2)
      Co0C = Co0Reg[IR[15 : 11]]
   3. if((IR[31 : 26] == 16) && (IR[25 : 21] == 0))then
      C = Co0C
   4. Reg[IR[20 : 16] = C

   `mtc0`:
   3. if((IR[31 : 26] == 16) && (IR[25 : 21] == 4)) then
      C = B
   4. Co0Reg[IR[15 : 11] = C

   In both of the above, C represents a 32-bit register that is not part of a register file. Several registers that have been used by previously considered instructions could be used rather than adding a new register.

3. [10 pts] Write a multicycle RTL description of an "undefined" instruction that uses as few cycles as possible without extending the clock cycle of your design. Hint: Save PC-4 in EPC, modify the PC, and update the Status Register. Read through pages 340-343 and pages A33-A35 (on the CD) of Patterson and Hennessy for more information (keeping in mind that for this exercise you are NOT concerned with the datapath or control system).

3. else
   Co0Reg[12] [4] = 0
   Co0Reg[12] [1] = 1
   Co0Reg[13] [8] = 1
   Co0Reg[13] [6 : 2] = code for undefined instruction
   Co0Reg[14] = PC – 4
   PC = 0x8000180

Notes:

- Co0Reg[12] is the same as Status.
  - Assume interrupts at this level are enabled (mask bit set to 1)
  - Set user mode bit to 0
  - Set exception level bit to 1
- Co0Reg[13] is the same as Cause.
  - Set interrupt pending bit to 1
  - Set exception code
- Co0Reg[14] is the same as EPC.
- Exception handling routing begins at 0x80000180 (SPIM uses 0x80000080).

4. Consider the (fictitious) MIPS instruction `addmem` described below.

   `addmem rd, rs, rt`

   The instruction reads the values from the memory locations indexed by the contents of registers `rs` and `rt`, adds the values, and stores the result in the memory location indexed by the contents of register `rd`.

   a. [5 pts] Write a multicycle RTL description of an implementation of the addmem instruction that uses as few cycles as possible without extending the clock cycle of your design.

3. if(IR[31 : 26] == addmem opcode) then
   A = Mem[A]
   B = B
4. A = A
5. B = Mem[B]
6. ALUout = A + B
   C = Reg[IR[15 : 11]]
7. Mem[C] = ALUout

Note: Instead of the assignments B = B in cycle 3 and A = A in cycle 4, we could change the descriptions of A and B to include write control signals.

   b. [5 pts] Write a single cycle RTL description of an implementation of the addmem instruction.

Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Profs. Archana Chidanandan and Larry Merkle

1. PC = PC + 4
   if(Mem[PC] [31 : 26] == addmem opcode) then
     Mem[Reg[Mem[PC] [15 : 11]]]
       =
       Mem[Reg[Mem[PC] [25 : 21]]]
       +
       Mem[Reg[Mem[PC] [20 : 16]]]