Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Prof. Archana Chidanandan

Homework 3 - Solutions
(Machine Language Representation, Exceptions, Register Transfer Language and Datapaths)
Maximum points : 55 points

Directions

This assignment is due Tuesday, Jan. 13th for Sections 1 and 2. Submit your solutions on a separate sheet of paper.

Comment: by 5:00 PM on

Learning Objectives

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

- Interpret machine language instruction formats and fields.
- 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.
- Design datapaths to support machine language instruction sets by specifying the interconnections between components and the behavior of the associated control units.
- Determine how to handle exceptions, including interrupts.

Problems

1. A computer has a 24-bit word length, and all the instructions are one word in length. Every instruction has a two bit field which specifies the instruction type. The number of registers (the register file) in the architecture of the computer is 26.

a. [2 pts] For a format that has an opcode field to determine the function of the instruction and three register fields, what is the maximum number of opcodes possible?

2 bits – instruction type
3 register fields:
26 registers => 5 bits to address each register as $2^4 < 26 < 2^5$
Therefore, 3 x 5 = 15 bits for the 3 register fields
Bits remaining = 24 – 2 – 15 = 7
Therefore, a total of 2^7 opcodes are possible.

b. [3 pts] For a format with two register fields, one immediate/address field, and a maximum of 64 opcodes, what is the maximum number of bits available for the immediate/address field? How many instructions, of this format, will be required, at a minimum, to copy a 24-bit immediate value into a 24-bit register?

2 bits – instruction type
10 bits – 2 register fields

64 opcodes => 6 bits
Therefore, bits remaining for the immediate field = 24 – 2 – 10 – 6 = 6

As the immediate field is 6 bits wide, a minimum of 4 instructions will be required to copy a 24-bit immediate value into a 24-bit register.

2. Figure 5.33 of Patterson & Hennessy, shows a complete datapath for a multicycle implementation of most R-Type MIPS assembly language instructions, as well as `lw`, `sw`, `beq`, and `j`. For this question, you are required to modify the datapath to include the multicycle implementation of the `addi` instruction. Specifically, you must:

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

Multi-cycle RTL description for the MIPS `addi` instruction.

```
1. IR = Mem[PC]; PC = PC + 4
2. A = Reg[IR[25:21]] ; B = Reg[IR[20:16]]
   ALUOut = PC + [SE[IR[15:0]] << 2]
   If (IR[31:26] == 8) then
3. Sum = A + SE[15:0]
4. Reg[IR[20:16]] = Sum
```

Note: Remember that any operation that involves memory, the ALU or the register file takes a significant amount of time. Therefore, if you do not want to extend the clock cycle, more than one of these operations should not be executed in the same clock cycle, unless they are independent of each other and can be executed in parallel.

b. [5 points] List all new and modified components required for the implementation of the `addi` instruction. Also, list the input, output, and control signals for each of those components. Indicate the number of bits in each signal.

No additional components are required or need to be modified. *Note:* Look at the operations in each clock cycle of the above RTL description. Check to see if the operations in each of the clock cycles can be done using the components, inputs and connections between components that are currently present in the datapath. If not, then make the necessary changes.

c. [5 points] Add any necessary datapaths and control signals to the multicycle datapath. You can photocopy existing figures or download figures from www.mkp.com/cod2e.htm to make it easier to show your modifications. A pdf version of the figure is also available on the class website(Homework).

No changes required here.

3. [5 points] For the previous question, you wrote the RTL description for the `addi` instruction. For this question, you must modify the RTL description for `addi`, to handle an ~~interrupt~~ exception i.e. an unexpected event that ~~occurs outside the processor, and~~ causes a change in the flow of execution of the program. *Note:* You do not need to list the new and modified components, nor do you need to show the modified datapath.

Comment: Changed from interrupt to exception

Assuming that the "exception check" cycle is executed before the execution of every new instruction, the RTL for the MIPS `addi` instruction with exception handling is:

```
1. If (SR[0]==1 AND ((CAUSE[15:11] & SR[15:11])!=0)) then
   # If Interrupts are enabled and if there is a pending interrupt and it is of the right priority, then
   SR[0] = 0; //Disable interrupts
   EPC = PC //Save the PC to return to, after the Interrupt Service Routine(ISR) is
            //done.
   PC = 0x8000 0080 //Change the PC to the address of the ISR. MIPS uses the
                    //Status Register Method. Therefore, for all exceptions
                    // the PC is modified to the same address.
2. IR = Mem[PC]; PC = PC + 4
3. A = Reg[IR[25:21]] ; B = Reg[IR[20:16]]
   ALUOut = PC + [SE[IR[15:0]] << 2]
   If (IR[31:26] == 8) then
4. Sum = A + SE[15:0]
5. Reg[IR[20:16]] = Sum
```

4. [15 points] Repeat Steps ~~1a, 1b and 1c~~ 2a, 2b and 2c for the `mfc0` and `mtc0` MIPS instructions (combine the component lists and the datapath modifications). *Hint:* Page A69 of Hennessey and Patterson, has a description for these two instructions.

   a. RTL description for `mfc0` and `mtc0`

      1. `mfc0 rt rd` - Move the contents of co-processor 0's register rd to register rt.

```
1. PC = PC + 4; IR = Mem[PC]
2. A = Reg[IR[25:21]];   B = Reg[IR[20:16]];
   ALUOut = PC + (SE[IR[15:0]] << 2);
   If((IR[31:26]==16)and(IR[25:21]==0)) then
3. D = CoP0Reg[IR[15:11]]; //Could do this in clock
                           //cycle 2 and save a clock cycle
4. Reg[IR[20:16]] = D;
```

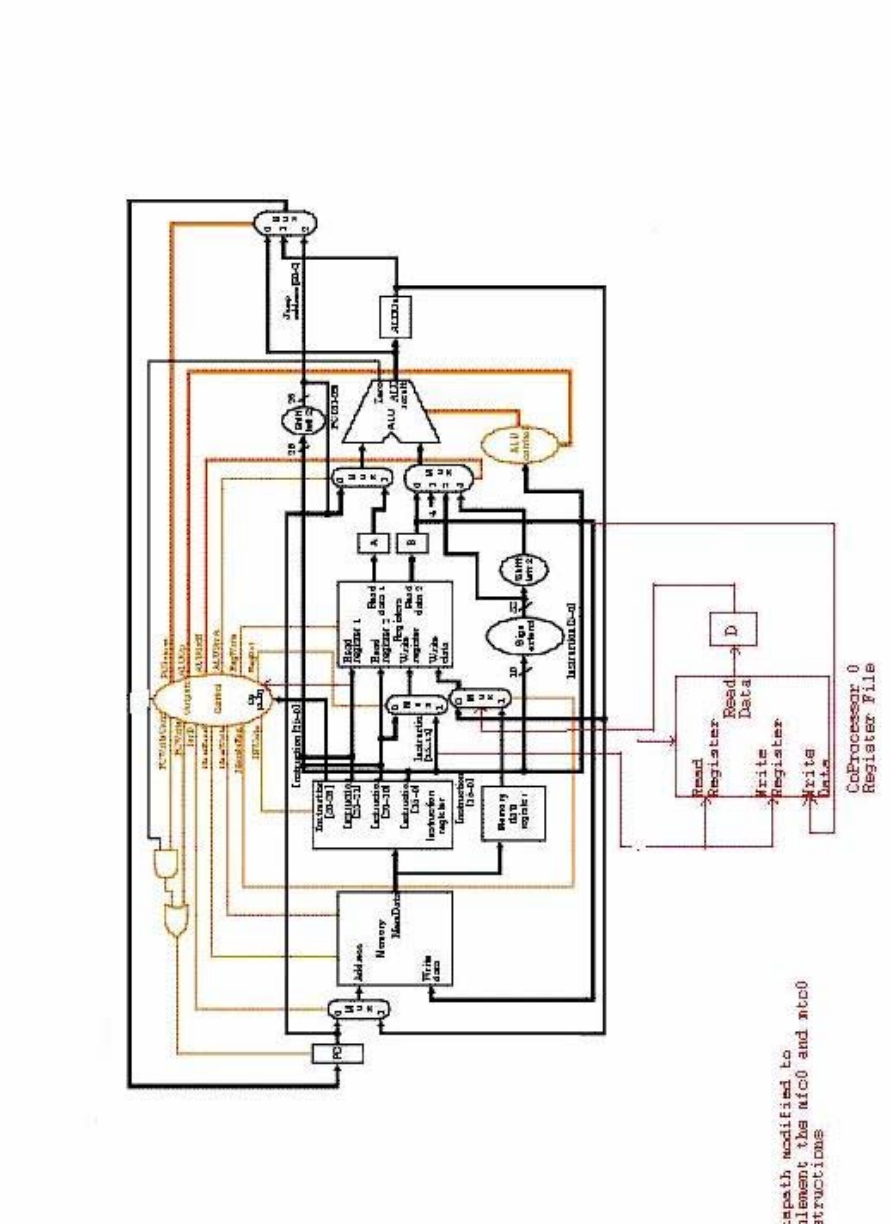
2. `mtc0 rt rd` – Move the contents of register rd to co-processor 0's register rt.

```
1. PC = PC + 4; IR = Mem[PC];
2. A = Reg[IR[25:12]];   B = Reg[IR[20:16]];
   ALUOut = PC + (SE[IR[15:0]] << 2);
   If((IR[31:26]==16)and(IR[25:21]==4))then
3. CoP0Reg[IR[15:11]] = B;
```

b. New and modified components list

Component	Inputs	Outputs	Control signals
Co-processor 0 Register file	$CP0WriteData_{31:0}$, $CP0ReadAddr_{4:0}$, $CP0WriteAddr_{4:0}$	$CP0DataOut_{31:0}$	CP0Write
D	$DIn_{31:0}$	$DOut_{31:0}$	-

c. The following datapath shows the changes required to implement the MIPS `mfc0` and `mtc0` instructions.



5. [15 points] Repeats Steps 1a, 1b and 1c for an "undefined" instruction in MIPS. *Hint:* Save PC-4 in EPC, modify the PC, and update the Status Register. Read through pages 410-413 and pages A32-A35 of Patterson and Hennessy for more information.

   a. RTL description for an "undefined" instruction

```
1. PC = PC + 4; IR = Mem[PC];
2. A = Reg[IR[25:21]];   B = Reg[IR[20:16]];
   Sum = PC + (SE[IR[15:0]] << 2);
    If (IR[31:26] == invalid opcode) then
    PC = 0x8000 0080; # Address of ISR
    Status[0] = 0;    # Disable interrupts
     EPC = PC         # Save PC not PC – 4; as we
                      # do not want to process
                      # the same instruction.
    Cause[5:2] = xxxx # Put the Cause for the
                      # exception in the Cause
                      # register
```

   b. New and modified components list

Component	Inputs	Outputs	Control signals
Cause Register	$CauseIn_{31:0}$	$CauseOut_{31:0}$	CauseWrite
EPC	$EPCIn_{31:0}$	$EPCOut_{31:0}$	EPCWrite
Status Register	$SRIn_{31:0}$	$SROut_{31:0}$	SRWrite

c. Datapath with changes to handle an “undefined” instruction.