Professor: Hong Zhang

CMPUT 229 Computer Organization and Architecture I A3

2000-2001

Midterm Examination

(November 15, 2000)

Name: ________________ SID: ________________

Do all problems. Closed book and notes. No calculator. Instruction set sheet is provided.

Problem 1 (8 marks)

The following program contains a `main` program and a recursive `fact` program that was discussed in class. `fact` is first called from `main` and then recursively. Show the contents of the three stack frames associated with the invocation of `main` and the first two invocations of `fact` (one non-recursive and one recursive call). Make sure to include the addresses of the items on the stack in your answer. Note all the relevant register contents and addresses given with the code.

```
        # initially, $sp = 0x7fffed10, $ra = 0x00400018    <-------
main:
        sub     $sp, $sp, 4      # make room on stack
        sw      $ra, ($sp)       # save the return address on stack
        li      $a0, 3           # pass 3 to fact()
        jal     fact             # make the call
        # address = [0x00400030] here                      <-------
        lw      $ra, ($sp)       # recover return address
        addi    $sp, $sp, 4      # clean the stack
        jr      $ra              # go back to caller

# subroutine fact(int n), which computes n!

        # address = [0x0040003c] here                      <-------
fact:
        addi    $sp, $sp, -8     # make space for saved registers
        sw      $ra, 4($sp)      # save return address
        sw      $a0, ($sp)       # save parameter n

        li      $t0, 1           # if (n > 1)
        ble     $a0, $t0, else
        sub     $a0, $a0, 1      # a0 = n-1
        jal     fact             # recursive call
        # address = [0x0040005c] here                      <-------
        lw      $t0, ($sp)       # load n to t0
        mul     $v0, $v0, $t0    # multiply it with fact(n-1)
        b       done             # v0 contains the result
else:
        li      $v0, 1           # return(1)
done:
        lw      $ra, 4($sp)      # restore return address
        lw      $a0, ($sp)       # and a0 (not necessary)
        addi    $sp, $sp, 8      # clean the stack
        jr      $ra              # go back to caller
```

Problem 2 (12 marks)

Implement the following C subroutine in MIPS subroutine. Assume that the three parameters are passed through $a0, $a1, $a2, respectively, and that addai returns its value via $v0. (Note that you do **not** need to show how addai is called.)

```
int addai(int a[], int b, int i) {
        return(a[i]+b);
}
```

Problem 3 (8 marks)

Answer the following questions with regard to exceptions.

1. Explain in your own words what is the use of the BadVAddr register in handling an exception.

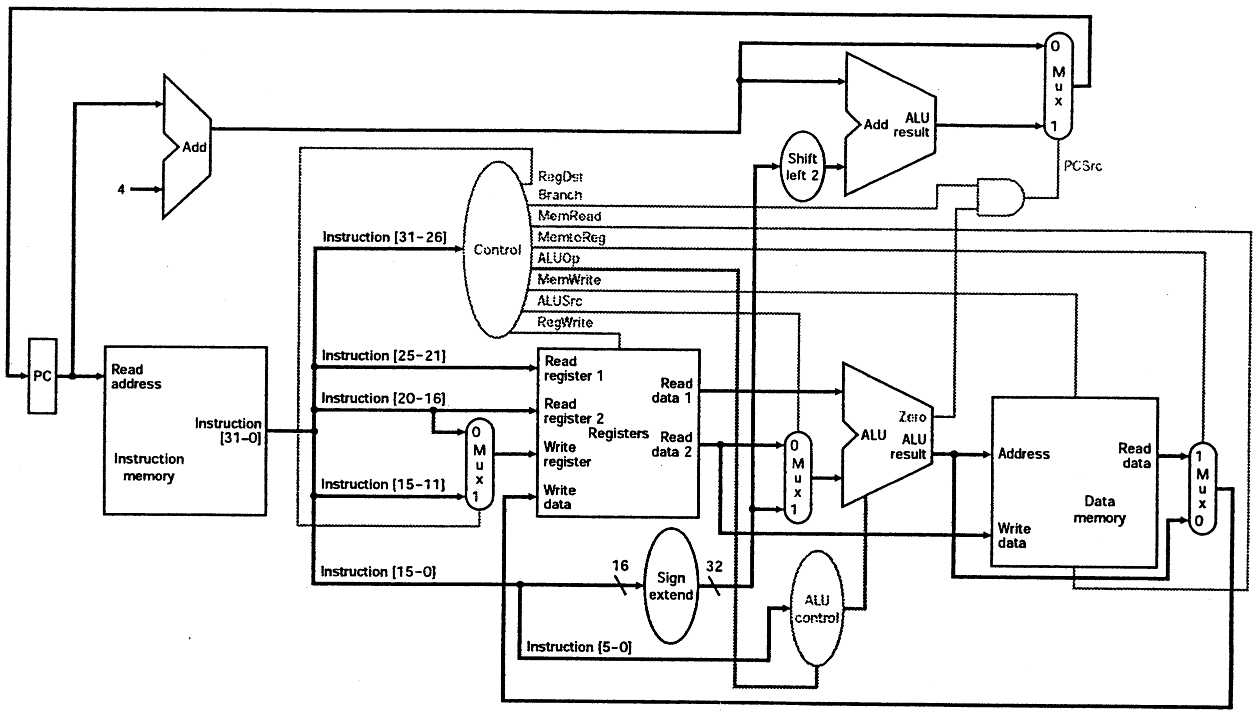
2. The address error exception ADDRL can occur either in a load instruction (e.g. lw) or during instruction fetch. Define a scenario in which the CPU would generate ADDRL exception **during instruction fetch**. Use either words or an actual instruction sequence.

3. In the following exception handler, what is the purpose of the first three instructions in terms of what information will be stored in $v0? (Note the format of $13 in the comments.)

4. In the last four instructions, why do we not add 2 to EPC directly to calculate the return address but use a kernel register like $k0 instead?

```
        .ktext 0x80000080
EH:     mfc0    $k0, $13            # $13 is Cause register whose bits[5-2] = exception code
                                    # bits[15-10] = pending interrupts
        andi    $k0, $k0, 0x3c
        slr     $v0, $k0, 2
        mfc0    $k0, $14            # $14 is EPC
        addi    $k0, $k0, 2
        rfe
        jr      $k0
```

Problem 4 (12 marks)



Refer to the above single cycle datapath. Assume it is executing the instruction `add $2, $3, $4`, and the three registers contain 5, 6, and 7 (in 32 bits), respectively. Answer the following questions.

1. In order for the add operation to be performed correctly, the select bits of the four multiplexers generated by `Control` should be:

   `PCSrc` = ___, `MemtoReg` = ___, `ALUSrc` = ___, `RegDst` = ___.

2. For the same `add $2, $3, $4` instruction, the `Read register 1`, `Read register 2`, and `Write register` inputs of the register file **in binary** are:

   `Read register 1` = _________, `Read register 2` = _________, `Write register` = _________.

3. Also for this add instruction, the outputs of the register file are:

   `Read data 1` = _________, `Read data 2` = _________.

If now the datapath is executing the instruction `addi $2, $3, -4` instead,

4. What are the inputs and outputs of the oval labelled *Sign extended*? Express your answers in **hexadecimal**.

5. Which output of the register file is no longer useful, *Read data 1* or *Read data 2*, and why?

Finally,

6. In general, what type of instructions uses the oval labeled *Shift left 2*?