
EXAM 1
Sec 001

Instructions

1. This is an open books/notes exam.
2. It comprises 17 multiple-choice questions and 1 fill-in-the-blanks question.
3. There are 12 pages in this booklet.
4. Each multiple-choice question has only one correct answer.
5. For each multiple-choice question, indicate your answer on the computer scantron sheet provided.
6. For the fill-in-the-blanks question (#18), enter your answers on the exam booklet.
7. You have 75 minutes to complete the exam.
8. Please make sure you enter your name and UMID in the space below AND on the scantron sheet.
9. By signing below, you certify that your conduct throughout the exam has been in accordance with the College of Engineering Honor Code.
10. At the end of the exam, please turn in this exam booklet AND your scantron sheet.

KEY

Name

UMID

Signature

Instruction Set Architecture

1. [5 Points] Assume a 64-bit wide RISC machine with 64 registers and 200 distinct instructions. Its branch instruction compares two registers for equality, and uses the 2's complement offset field to compute the PC-relative target. What is the branch target range?
 - a. -2^{44} to $2^{44}-1$
 - b. $-2^{44} - 1$ to 2^{44}
 - c. -2^{43} to $2^{43}-1$ **
 - d. $-2^{43} - 1$ to 2^{43}
 - e. -2^{45} to $2^{45}-1$

2. [5 Points] Which of the following is NOT allocated on the stack?
 - a. parameters to a function
 - b. registers saved by callee
 - c. registers saved by caller
 - d. return address
 - e. static local variables **

3. [5 Points] Under the caller-save restoring convention
 - I. the caller saves registers
 - II. the caller restores registers
 - III. the callee restores registers
 - IV. the callee assumes that registers are unchanged by the call
 - V. the caller assumes that registers are unchanged by the call
 - a. I, II **
 - b. I, III
 - c. I, II, IV
 - d. I, III, IV
 - e. I, II, V

4. [5 Points] Which addressing mode is most useful in addressing local variables within functions?
- a. direct
 - b. indirect
 - c. immediate
 - d. base + displacement **
 - e. PC-relative
5. [5 Points] Which of the following attributes are characteristic of RISC ISAs?
- I. fixed-length instructions
 - II. immediate operands
 - III. register-to-register operations
 - IV. indirect addressing mode
 - V. program stack
- a. I, II
 - b. I, III **
 - c. I, III, IV
 - d. III, IV
 - e. III, V

Compilation of C Functions

6. [5 Points] Consider the following program

```
int a;

void main()
{
    int b;
    int c = 0xabcd;
    .....
    foo( b+c );
}

void foo( int x )
{
    static int y = 0xffff;
    char *z;
    z = (char *) malloc( x*y+1 );
    .....
    free(z);
    return;
}
```

Which of the following data mappings in memory is consistent with the above program?

- a. stack: b, c, y; heap: *z; static: a
- b. stack: b, c, x; heap: z; static: a, y
- c. stack: b, c, *z; heap: x; static: y
- d. stack: b, c, x; heap: *z; static: a, y ****
- e. stack: b, c, z; heap: *z; static: x, y

Execution of C Functions

A recursive function `foo()` is defined as follows:

```
int foo(int n)
{
    if (n == 1)
        return 1;
    else
        return foo(n-1);
}
```

Assume the function `foo(3)` was called by another function. The table below is filled with the activation record for each recursive call to `foo()` until `foo()` has reached the last leaf node. To simplify the task, we only store the input argument, `n`, on to the stack (no return address nor return value). We also ignore the stack of the caller of `foo(3)`.

Call	<code>foo(3)</code>	<code>foo(2)</code>	<code>foo(1)</code>
Stack	3	3	3
↓		2	2
			1

(Continued on next page)

Now, let's consider the Fibonacci function defined as follows:

```
int fib(int n)
{
    if (n <= 1)
        return 1;
    else
        return fib(n-2) + fib(n-1);
}
```

7. [5 Points] For fib(4), fill in the following table with the activation record for each recursive call until fib() has reached the last leaf node. To simplify the problem, you only need to store the input argument on to the stack, (no return address nor return value). You may also ignore the stack of the caller of fib(4). Please note that: (1) stack grows from top to bottom; (2) the space provided below may be larger than necessary.

Call	fib(4)	fib(2)	fib(0)	fib(1)	fib(3)	fib(1)	fib(2)	fib(0)	fib(1)
Stack	4	4	4	4	4	4	4	4	4
↓		2	2	2	3	3	3	3	3
			0	1		1	2	2	2
								0	1

8. [5 Points] If each activation frame requires 28 bytes of storage, and the maximum stack size possible for the machine is 512 bytes, what is the largest of the calls below which can execute correctly? (Ignore the stack of the caller of fib.)
- fib(17)
 - fib(18) **
 - fib(19)
 - fib(20)
 - None of the above

Floating Point Numbers

9. [5 Points] What is the IEEE 754 representation of **-0.21875**?
- a. 0 01111110 100000000000000000000000
 - b. 0 01111100 110000000000000000000000
 - c. 1 01111110 100000000000000000000000
 - d. **1 01111100 110000000000000000000000 ****
 - e. None of the above
10. [5 Points] The number of distinct floating point numbers that can be represented in the IEEE 754 single-precision format is about
- a. 2^{23}
 - b. about 1 million
 - c. **about 4 billion ****
 - d. about 2^{128}
 - e. about 2^{255}
11. [5 Points] A summer intern approaches you with the following great idea for your NextGen LC2Kn machine: Do not support operations on 32-bit integers. Programmers can simply use 32-bit IEEE 754 floats instead. Having taken EECS 370, you immediately realize this is a bad idea. What is your main reason?
- a. Floating point operations are more complex than integer operations.
 - b. The largest 32-bit float that can be represented is not as large as the largest 32-bit integer.
 - c. **Certain 32-bit integers cannot be represented as 32-bit floats. ****
 - d. Backward compatibility with older code requires the use of integer operations.
 - e. RISC ISAs are required to support integer operations.

Single-Cycle Datapath Design

Consider the single-cycle datapath we discussed in class for the LC2K5 computer which is reproduced as Figure 1 on page 11. We would like to make several alternative extensions to the LC2K5 Instruction Set Architecture and you are asked to make the minimal necessary modifications to the datapath in order to implement each of those extensions.

- The first alternative extension adds two new “memory” instructions defined as follows:

addm regA regB destR	$\text{destR} \leftarrow \text{regA} + \text{mem}[\text{regB}]$
nandm regA regB destR	$\text{destR} \leftarrow \sim(\text{regA} \& \text{mem}[\text{regB}])$

Make the minimal necessary modifications to the datapath to implement these two instructions and answer the following two questions:

12. [5 Points] The following additional components are needed:
 - a. One 2-to-1 MUX.
 - b. One adder.
 - c. Two 2-to-1 MUXes and one adder.
 - d. One 2-to-1 MUX and widening an existing 2-to-1 MUX to become a 3-to-1 MUX. **
 - e. No additional components are needed; the two new instructions can be accommodated with the existing datapath.
13. [5 Points] The control ROM needs to be expanded by adding:
 - a. Two rows and two columns. **
 - b. One row and three columns.
 - c. Two rows and four columns.
 - d. Four rows and two columns.
 - e. No extension necessary; just modify ROM content.

- The second alternative extension adds an “immediate” instruction defined as follows:

addi regA regB offset $\text{regB} \leftarrow \text{regA} + \text{signextend}(\text{offset})$

Make the minimal necessary modifications to the original datapath (i.e., disregard any modifications you made to implement the first alternative extension above) to implement this instruction and answer the following two questions:

14. [5 Points] The following additional components are needed:
 - a. One 2-to-1 MUX.
 - b. One adder.
 - c. Two 2-to-1 MUXes and one adder.
 - d. One 2-to-1 MUX and widening an existing 2-to-1 MUX to become a 3-to-1 MUX.
 - e. **No additional components are needed; the new instruction can be accommodated with the existing datapath. ****

15. [5 Points] The control ROM needs to be expanded by adding:
 - a. Two rows and two columns.
 - b. One row and three columns.
 - c. Two rows and four columns.
 - d. **One row and no additional columns. ****
 - e. No extension necessary; just modify ROM content.

- The final alternative extension adds an unconditional jump instruction defined as follows:

jump regA PC ← regA

Make the minimal necessary modifications to the original datapath (i.e., disregard any modifications you made to implement the two alternative extensions above) to implement this instruction and answer the following two questions:

16. [5 Points] The following additional components are needed:
- a. One 2-to-1 MUX.
 - b. One adder.
 - c. Two 2-to-1 MUXes and one adder.
 - d. Widening an existing 2-to-1 MUX to become a 3-to-1 MUX.
 - e. **No additional components are needed; the new instruction can be accommodated with the existing datapath. ****
17. [5 Points] The control ROM needs to be expanded by adding:
- a. Two rows and two columns.
 - b. **One row and no additional columns. ****
 - c. Two rows and four columns.
 - d. One row and one column.
 - e. No extension necessary; just modify ROM content.

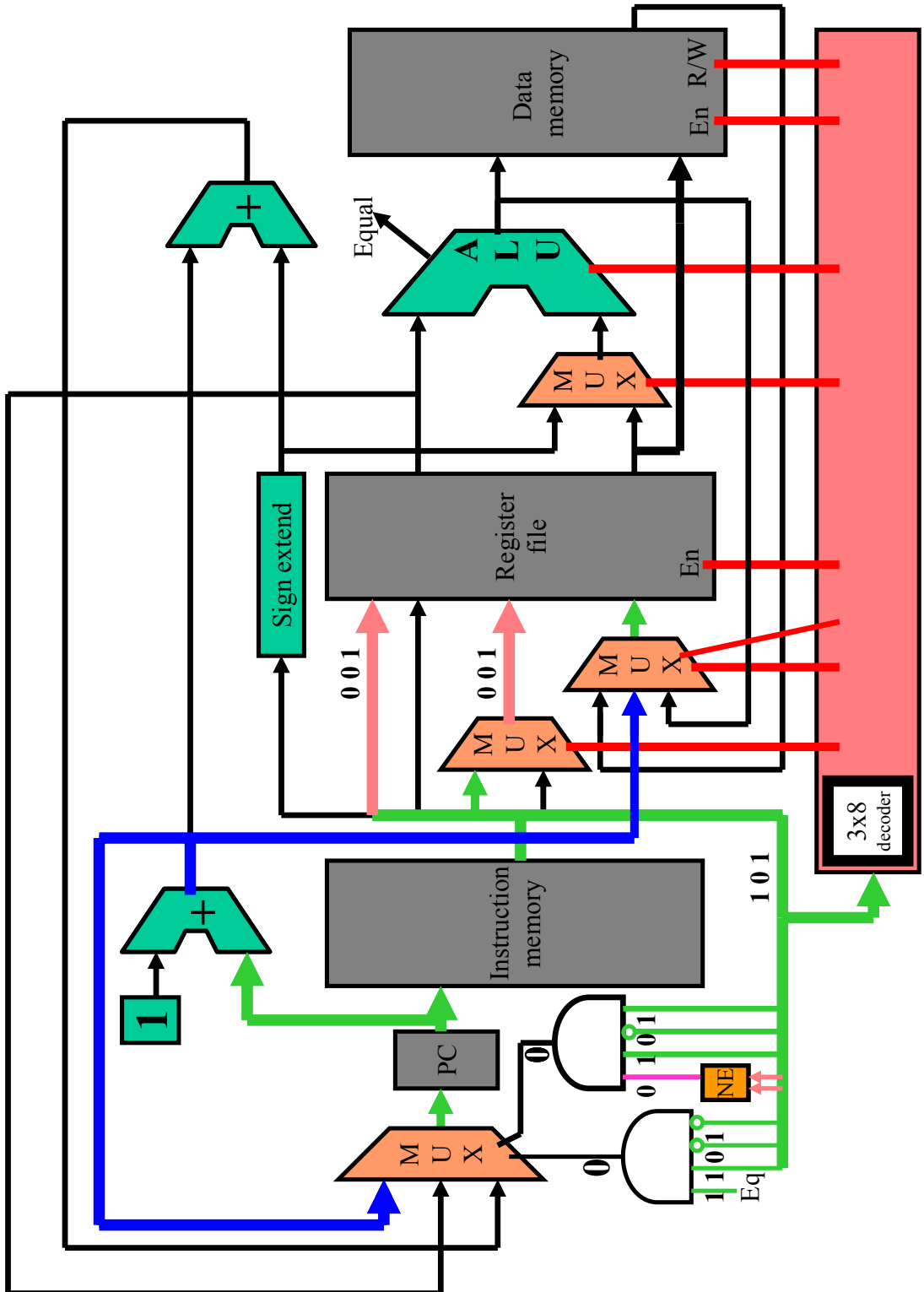


Figure 1: LC2K5 Datapath for questions 12 through 17

MIPS Assembly Language

18. [15 Points] The following recursive C function sums up the elements of an array.

```
int sum( int arr[], int size ) {
    if ( size == 0 )
        return 0 ;
    else
        return sum( arr, size - 1 ) + arr[ size - 1 ] ;
}
```

Given below is a MIPS translation of the above C code with some instructions fields and comments missing (indicated by underlines.) Fill in the missing instruction fields with the correct values for registers and immediates. In addition, fill in appropriate comments where indicated.

Assume that `arr` is in `$a0` and `size` is in `$a1`.

Also, note that the result of `mult` is stored in `$LO`, and that the instruction `mflo $rx` moves the value from `$LO` to `$rx`.

```
sum:    addi $sp, $sp, -8          # Adjust sp                ""
        addi $t0, $a1, -1       # Compute (size - 1)
        sw   $t0, 0($sp)        # Save (size - 1) to stack
        sw   $ra, 4($sp)       # Save return address
        bne $a1, $zero, ELSE    # branch ! ( size == 0 )    ""
        addi $v0, $zero, 0     # Set return value to 0
        addi $sp, $sp, 8        # Adjust sp
        jr  $ra                # Return

ELSE:   move  $a1, $t0           # update second arg
        jal  sum
        lw   $t0, 0($sp)        # Restore size - 1 from stack    ""
        addi $t7, $zero, 4     # t7 = 4
        mult $t0, $t7           # Multiply (size - 1) by 4
        mflo $t1                # Put result in t1
        add  $t1, $t1, $a0       # Compute & arr[ size - 1 ]    ""
        lw   $t2, 0($t1)       # t2 = arr[ size - 1 ]
        add  $v0, $v0, $t2       # retval = $v0 + arr[size - 1]
        lw   $ra, 4($sp)        # restore return address from stack ""
        addi $sp, $sp, 8        # Adjust sp
        jr  $ra                # Return
```