Exam 2 Information

The exam will be in class on Thursday, 12 November. It will cover all material from 29 September through 29 October. The main topics we have covered in that time are:

- MIPS loops
- MIPS function calls (jal, jr)
- MIPS register conventions
- using the MIPS stack; preserving registers; nested function calls
- integer multiplication and division in binary
- fractions in binary
- floating-point representation
- C programming—arrays; pointers (and their relation to arrays); bit operations (left and right shift; bitwise ‘and’ and ‘or’);

Question types will include:

- Short answer (“define . . .”, “what is the value of . . .”, etc.)
- Write portions of C code
- Explain portions of C code
- Write portions of MIPS code
- Explain portions of MIPS code
- Calculation problems (binary/decimal/hexadecimal computations, etc.)

You will be given the first page of the MIPS reference card and a list of fractional values in binary. Here are examples of the kinds of questions that might be asked. This is not intended to be a sample exam; the topics covered below are not intended to be an exhaustive review. In particular, knowing the answers to all the questions below will not guarantee a good grade on the exam!
Some Simple Fractions and Their Binary Representations

<table>
<thead>
<tr>
<th>Fraction</th>
<th>Decimal</th>
<th>Binary</th>
<th>Fraction</th>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{16}$</td>
<td>0.0625</td>
<td>0.0010</td>
<td>$\frac{9}{16}$</td>
<td>0.5625</td>
<td>0.1001</td>
</tr>
<tr>
<td>$\frac{3}{8}$</td>
<td>0.1250</td>
<td>0.0100</td>
<td>$\frac{5}{8}$</td>
<td>0.6250</td>
<td>0.1010</td>
</tr>
<tr>
<td>$\frac{3}{16}$</td>
<td>0.1875</td>
<td>0.0011</td>
<td>$\frac{11}{16}$</td>
<td>0.6875</td>
<td>0.1011</td>
</tr>
<tr>
<td>$\frac{1}{4}$</td>
<td>0.2500</td>
<td>0.0100</td>
<td>$\frac{3}{4}$</td>
<td>0.7500</td>
<td>0.1100</td>
</tr>
<tr>
<td>$\frac{5}{16}$</td>
<td>0.3125</td>
<td>0.0101</td>
<td>$\frac{13}{16}$</td>
<td>0.5625</td>
<td>0.1101</td>
</tr>
<tr>
<td>$\frac{3}{8}$</td>
<td>0.3750</td>
<td>0.0110</td>
<td>$\frac{7}{8}$</td>
<td>0.8750</td>
<td>0.1110</td>
</tr>
<tr>
<td>$\frac{7}{16}$</td>
<td>0.4375</td>
<td>0.0111</td>
<td>$\frac{15}{16}$</td>
<td>1.0000</td>
<td>0.1111</td>
</tr>
<tr>
<td>$\frac{1}{2}$</td>
<td>0.5000</td>
<td>0.1000</td>
<td>$\frac{1}{2}$</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

1. What are the final values of the integer variables $m$ and $n$ in the following C statements?

```c
int i = 20, j = 12;
int m = i | j;
int n = i & j;
```

2. What is the binary representation of the decimal value 6.3? (If the fractional part is infinite, show only the first 5 bits of the fractional representation.)

3. In the following MIPS program, what are the final values of registers $a0$ and $a1$? (NOTE: this is a complete MIPS program; nothing has been left out.)

```mips
.text
li $a0,10
li $a1,20
jal f
li $v0,10 # exit program
syscall

f:    add  $a0,$a0,$a1
      sub  $a1,$a0,$a1
      sub  $a0,$a0,$a1
      jr   $ra
```

4. Fill in the body of function “$g$” in the following C program. The function receives two parameters—a double array and the size of the array. Function $g$ returns the average of the first and last elements of array $a$. For instance, if $x$ contains the values 1.2, 2.2, 3.4, and 2.1, then $g(x,4)$ will return the value 1.65 (the average of 1.2 and 2.1).

```c
#include <stdio.h>

Handout 20  Handed out on 6 November 2015
/* Function prototype for g: */
double g(double a[], int n);

int main() {
    ...
double a[], int n;
    ... declarations of arrays x and y not shown ...

    printf("g(x,4)=%f\n",g(x,4));
    printf("g(y,8)=%f\n",g(y,8));
}

... Fill in function g here ...

5. What floating-point number is represented by this 32-bit binary value?

```
1 01111111 01100000000000000000000
```

6. A programmer determines that, at the beginning of a certain function she is writing, it is necessary to preserve the contents of registers $s0$, $s1$, and $ra$ on the stack. Write the MIPS statements needed to do this.

7. The same programmer as in the previous problem needs to restore registers $s0$, $s1$, and $ra$ to their original values before returning from the function. Write the MIPS statements to restore the register values (undo the actions taken in the previous question), restore the stack to its original state, and return from the function.

8. In C, what is the output of the following code:

```c
int i; /* loop counter */
int a[5] = {10,20,30,40,50};
printf("%d %d %d\n", *a, *(a+4), *(a+2));
```

9. In MIPS, suppose register $t0$ contains a positive integer $n$. The following loop is supposed to print out the values $n$, $n-1$, ..., 1, then halt. Several commands are missing from the loop—supply them!

```
... assume $t0$ contains n ...
loop:
    # check for loop termination
    move $a0,$t0
    # getting ready to print
    li $v0,1
    syscall
    # update loop counter
    # do it again
done: ...
```
10. Here is a C program to perform multiplication of two integers $a$ and $b$ by repeated shifting and adding. We are assuming that $a$ is the multiplier and $b$ is the multiplicand. Several statements have been left out—fill in the missing statements.

```c
prod = 0;
for (i = 0; i < 32; i++) {
    if (/* see if we should add multiplicand */) {
        prod = prod + b;
    }
    a = /* shift multiplier one place to the right */
    b = /* shift multiplicand one place to the left */
}
```

11. Here is the same computation as the previous question, but in MIPS rather than C. Fill in the missing instructions. (NOTE: we are assuming non-negative integers here, so no need to worry about the sign bit during shifting.)

```mips
... assume $s1$ contains multiplier, $s2$ contains multiplicand ...

li $t0,32  # initialize counter ($t0$ counts down to zero)
li $s0,0  # s0 = product
loop: beq $t0,$zero,done  # all finished?
    # check multiplier bit to see if it’s 1
    beq $zero,$t1,skip  # if not, don’t add
    add $s0,$s0,$s2  # add multiplicand to product
skip:
    # shift multiplicand left 1 bit
    # shift multiplier right 1 bit
addi $t0,$t0,-1  # subtract 1 from loop counter
    j loop
done: .... not shown ...
```

12. The above is not a full review!