Practice Questions (and Answers) for Midterm Exam
CS 64, Fall 2019, Matni

IMPORTANT NOTE: These questions are NOT representative of EVERYTHING you need to study for the midterm exam! You should also review your lab assignments questions and also all the examples and demos done in class.

A. Data Representation

1. Convert the following decimal numbers into signed 8-bit binary and two-digit hexadecimal.
    a. 39
    b. 104
    c. -59
    d. -98

2. Convert the following hexadecimal numbers into decimal.
    a. 0x46
    b. 0x1D
    c. 0x3F2

3. Convert these unsigned binary numbers into decimal.
    a. 1010
    b. 10101010
    c. 11000001

4. Convert these signed binary numbers into decimal.
    a. 1010
    b. 00110100
    c. 11000001

5. In 5 bits, what is the most negative value and the most positive value representable in signed form, using two's complement? Express your answers in both binary and decimal.

6. Same question as #5, but representable in unsigned form?

7. Suppose you are given the following 4-bit binary number, shown in two's complement: **1001** You're not told whether or not the number is signed or unsigned. Is this information important in knowing what the value of the number is, in decimal? That is, do you need to know if it's signed or unsigned to say what the decimal value is? Why or why not?

B. Binary Arithmetic

8. Perform the following two's complement addition, noting whether or not the carry bit (C) and/or the overflow bit (V) get set: **01111111 + 11111111**

9. Perform the following two's complement addition, noting whether or not the carry bit (C) and/or the overflow bit (V) get set: **00100101 + 10110111**

10. Perform the following two's complement addition, noting whether or not the carry bit (C) and/or the overflow bit (V) get set: **11100101 + 10000000**

C. Bitwise Operations

11. Given the hexadecimals X and Y in each of the following below, find:
    X && Y, X | | Y, and X ^ Y. Give your answers in hexadecimal.
    a. X = 0x2D, Y = 0xFE
    b. X = 0x67, Y = 0x67
    c. X = 0x4A, Y = 0x91

12. Consider the following C code, which is intended to extract 7 bits from bits position 7 thru 13 of the given input i, leaving the bits in their original position:

```
int unsignedBits7through13(int i) {
  return __________;
}
```

    Fill in the blank with a **single** bitwise expression which will make the code do what it is intended to do.

13. Consider the following C code, which is intended to extract the next 7 bits of the given input i, treating the result as a signed value, putting the bits in the rightmost position:

```
int signedBits7through13(int i) {
  // Some lines of code here
  return u;
}
```

    Fill in the empty part with valid C/C++ code (can be multiple lines) which will make the code do what it is intended to do.

D. Assembly

14. Why isn't **`li`** an actual MIPS instruction?

15. Translate the following pseudo-C code into MIPS assembly. Where <<read integer from the user>> is used, you should use special functionality provided by SPIM to read in an integer from the console. Where <<print integer s1>> is used, you should use special functionality provided by SPIM to print the integer stored in s1 to the console. The variables used below should be placed in the register with the same name. For example, variable s0 should be placed in register $s0. If you need additional registers than what the code below uses, use registers $t0 - $t9. You do **not** need to exit the program properly.

```
int s0 = <<read integer from the user>>;
int s1 = 2;
if (s0 < 7) {
  s1 = 3;
} else {
  s1 = s0 + s0;
}
<<print integer s1>>
```

16. Write an entire assembly program that takes an integer from standard input (i.e. the user) and checks to see if it is a 0, or 1, or 2 (assume the user behaves and only enters one of these 3 numbers and you do not have to check if they do otherwise). The program then prints to the standard output the text “zero”, or “one”, or “two”, as the appropriate case may be. This has to be followed by a newline character. Bonus points if you can do this using only 2 branch statements.

17. What is the machine code (in hexadecimal) for these instructions?

```
a. add $a0, $s3, $a1
b. andi $t2, $t2, 119
```

18. Given the C++ function described below:

```
int thatone(int num1, int num2) {
      ans = 4*num1 + 2*num2;
      return (ans);
}
```

a. Which registers would you use for **num1**, **num2**, and **ans**, assuming MIPS Calling Convention rules?
b. What assembly instruction would you have at the end of the function?
c. What assembly instruction would you have in the code where the function is called?

19. Consider the C/C++ code below:

```
int sum( int n ) {
   if ( n == 0 ) return 0;
   else return (n + sum(n - 1));
}
```

   a. Knowing that you have to follow the MIPS Calling Convention, which variables should be preserved either directly (via the stack) or indirectly (in an S-register) in order to maintain the intended program behavior?
   b. Implement the previously shown C/C++ code using MIPS assembly, taking care to preserve the values you identified previously. Ignore the **.data** part and just focus on the **.text** part of the program. Assume, for this implementation, you have to call the function from your main program like this: `cout << sum(4)`

ANSWERS:

A. Data Representation

1.
   a. 00100111, 0x27
   b. 01101000, 0x68
   c. 11000101, 0xC5
   d. 10011110, 0x9E

2.
   a. 70
   b. 29
   c. 1010

3.
   a. 10
   b. 170
   c. 193.

4.
   a. -6
   b. 52
   c. -63

5. Most negative: 10000, -16. Most positive: 01111, 15.

6. You cannot represent negative numbers with unsigned binaries. Most positive: 11111, 31.

7. We know it's signed ***because it's in two's complement***. If we hadn't been told it was two's complement, then because of the left-most bit being a 1, we would be unable to determine the value of the number unless more information was given. This is because the left-most bit would make it a negative number if the number was signed, and a different positive number if the number was unsigned.

B. Binary Arithmetic

8.
```
 01111111
+11111111
---------
 01111110
```
The carry bit gets set (i.e. C = 1, V = 0)

9.
```
 00100101
+10110111
---------
 11011100
```
Neither gets set (i.e. C = 0, V = 0)

10.
```
 11100101
+10000000
---------
 01100101
```
Both get set (i.e. C = 1, V = 1)

C. Bitwise Operations

11.
   a. 0x2C, 0xFF, 0xD3
   b. 0x67, 0x67, 0x00
   c. 0x00, 0xDB, 0xDB

12. (i & 0x00003F80)

13.
```
int signedBits7through13(int i) {
  int u = (i & 0x00003F80) >> 7;
  if (u & 0x00000040) {
    u |= 0xFFFFFF80;
  }
  return u;
}
```

D. Assembly

14. All MIPS instructions are exactly 32 bits large. li can be used to load a 32 bit constant into a register. Therein lies a problem: the constant's entire value couldn't possibly be held in a single instruction, because there aren't enough bits (in addition to the 32 bits of the constant, there would need to be other bits to encode what the instruction is, which requires in total more than 32 bits). It is for this reason that li is a *psuedoinstruction* which can be automatically translated to multiple instructions, whenever we need to specify a constant using the full 32 bits.

15.

```
main:
# read in the integer from the user, and initialize s1
  li $v0, 5
  syscall
  move $s0, $v0
  li $s1, 2

  # check if $s0 < 7
  li $t0, 7
  slt $t1, $s0, $t0

  # jump to the else branch if this isn't true
  beq $t1, $zero, else_branch

  # fall through to the true branch
  li $s1, 3
  j print

else_branch:
  add $s1, $s0, $s0
  # fall through to the print

print:
  li $v0, 1
  move $a0, $s1
  syscall
```

16.

```
.data
zero: .asciiz "zero\n"
one: .asciiz "one\n"
two: .asciiz "two\n"

.text
main:
    # Get user standard input
    li $v0, 5
    syscall
    move $s0, $v0

    li $t0, 0
    li $t1, 1
    li $t2, 2

    # Assume that input is 0
    # Check to see if input is 1 or 2 and branch accordingly
    beq $s0, $t1, print1
    beq $s0, $t2, print2
```

```
        # Print "zero"
        li $v0, 4
        la $a0, zero
        syscall
        j end

    print1: # Print "one"
        li $v0, 4
        la $a0, one
        syscall
        j end

    print2: # Print "two"
        li $v0, 4
        la $a0, two
        syscall

    end:
        li $v0, 10
        syscall
```

17.
    a. 0x02652020
    b. 0x314A0077

18.
    a. $a0 for num1, $a1 for num2, $v0 for ans
    b. jr $ra
    c. jal thatone

19.

a. $a0 for n, $v0 for the returned value (n + sum(n – 1))

```
int sum( int n ) {
   if ( n == 0 ) return 0;
   else return (n + sum(n - 1));
}
```

b.

```
   .text
sum:
    addiu $sp, $sp, -8      # PUSH
    sw $ra, 4($sp)
    sw $s0, 0($sp)

    beq $a0, $zero, return  # is size !=0?

    addi $a0, $a0, -1       # n is now: n - 1
    move $s0, $a0           # preserve a0 (variable n)
    jal sum                 # recursive call

return:
    lw $ra, 4($sp)          # POP
    lw $s0, 0($sp)
    addiu $sp, $sp, 8
    jr $ra

main:
   li $v0, 0                # Initialize sum ($v0)
   li $a0, 4                # n = 4
   jal sum                  # Call sum(4); expect $v0 to be 10

   move $a0, $v0
   li $v0, 1
   syscall

   li $v0, 10
   syscall
```