Quiz 1

1. Match the technical term with the most appropriate definition:

CPU	→	**Central Processing Unit**
Register	→	**High-speed storage in the CPU**
RAM	→	**Random access memory, which can be written**
ROM	→	**Random access memory, which cannot be written**
Machine Language	→	**Binary coded instructions, to be executed by CPU**
Assembly Language	→	**Machine language with mnemonic operations and symbolic addresses**
Compiler	→	**Software which translates a program in a high-level programming language to an equivalent program in machine language**
Keyboard	→	**A peripheral input device**
Printer	→	**A peripheral output device**
Disk	→	**A peripheral device for permanent storage of data**

2a. The binary number 11010111101000112 can be written in hexadecimal as **d7a3**.

2b. The hexadecimal number f4c16 can be written in binary as **1111 0100 1100**.

3. Which, zero or more, of the following is/are true about binary numbers stored as two's complement representation in a 32-bit field?

a.	Every positive value has a complement	→ **true**
b.	Every negative value has a complement	→ **false**
c.	All numbers are either positive or negative	→ **false**
d.	All negative numbers have a 1 in the high-order bit position	→ **true**

4. Which is the smallest number that can be stored in a 5-bit field, using two's complement representation? **-16**

5. 4T / 32G * 8K = ? Show your response using KMGT notation given in the lecture and textbook. **1M**

Quiz 2

1. Given the following program segment, what hexadecimal is left in register $t0 after the instructions have been executed? **0000 0014**

```
li      $t0, 7                  # $t0 = 7
li      $t1, 17
li      $t2, 3
add     $t0, $t1, $t2
```

2. Given the following program segment, what hexadecimal value is left in register $t0 after the instructions have been executed? **0000 0001**

```
li      $t0, 3                  # $t0 = 3
li      $t1, 5
and     $t0, $t0, $t1
```

3. Given the following program segment, what hexadecimal is left in register $t0 after the instructions have been executed? **0000 000c**

```
li      $t0, 3                  # $t0 = 3
sll     $t0, $t0, 2
```

4. Which instruction will effectively clear the high order 28 bits of register $s1, leaving the other bits unchanged? **andi $s1, $s1, 0x000f**

5. Which will be the final value, in hexadecimal, left in register $v0 after the instructions shown below have been executed? **ffff ffec**

```
li      $v0, 5
sll     $v0, $v0, 2
sub    $v0, $0, $v0
```

Quiz 3

1. We wish to convert an array of integers to all non-negative values, using an absolute value function. If the given array is [4, -3, 0, -4, 8] the array would become [4, 3, 0, 4, 8]. Show the MIPS statements which would accomplish this (no need to define a complete function). Do not use floating point instructions.

   Assume:

   - The address of the start of the array is in register $a0
   - The length of the array (number of integer values) is in register $a1 [the length could be 0]
   - Assume there is a function named abs which will return the absolute value of a given integer (you need not define this function, but you can call it):
     - The abs function expects the given integer in register $a2
     - The abs function places the absolute value of that integer in register $v0

```
 .text (SUPPOSED TO STORE THE RESULT IN THE ORIGINAL ARRAY)
# Driver
main:
   la    $a0, array
   lw    $a1, length
   la    $a3, new       # gives place to where abs array is stored

lp_main:
   beq    $a1, 0, done_main    # is counter done?
   lw    $a2, 0($a0)            # grades[$a0], using explicit
   jal    abs                    # perform abs
   sw    $v0, 0($a3)            # store into next value of $a3
   addi    $a3, $a3, 4           # address of next word for abs array
   addi    $a0, $a0, 4           # address of next word in array
   addi    $a1, $a1, -1          # decrement loop counter
   j    lp_main

done_main:
   li    $v0, 10                 # terminate normal
   syscall
.data
array:    .word    4, -3, 0, -4, 8
length:    .word    5
new:    .word
```

2. You are writing a MIPS function named 'foo' which uses registers $s0 and $s1 for temporary storage. It may also call other functions. Upon completion, the function branches to **done_foo**, where it returns to the calling function. Show the first 4 instructions in the function preamble, which saves registers on the stack.

```
######### Begin function foo

foo:

         ###### Fill in the missing preamble  (4 instructions)


                                  ## Statements in the function foo
(not shown)

done_foo:
        lw      $s1, 8($sp)                 # load regs from stack
        lw      $s0, 4($sp)
        lw      $ra, 0($sp)
        addi    $sp, $sp, 12
        jr      $ra                         # return to calling fun
ction

############ End function foo
```

addi $sp, $sp, -12
sw $ra, 0($sp)
sw $s0, 4($sp)
sw $s1, 8($sp)

3. A function, shown below, is designed to determine whether a string of characters consists of numeric characters only: “34w16” false ($v0 = 0) and “30416” true ($v0 = 1). This function has one logic error in the body of the loop. The comments are intended to be helpful. The logic error is:

```
################ Begin function isNumeric
### Determine whether a given string consists entirely
### of numeric characters.
### Register $a0 points to the given string,
### which is null-terminated.
### Return 1 in $v0 if it is numeric, 0 if not numeric
### String of length 0 is not numeric.
    .text
isNumeric:
    addi   $sp, $sp, -8                 # save regs on stack
    sw     $ra, 0($sp)
    sw     $s0, 4($sp)
    lbu    $t0, zero_isNumeric           # "0"
    lbu    $t1, nine_isNumeric           # "9"
    lbu    $s0, 0($a0)
    beq    $s0, $0, not_isNumeric        # empty string not numeri
c
    li     $v0, 1                        # assume result is true
```

```
lp_isNumeric:
    lw      $s0, 0($a0)                     # Get one char       #1
    beq     $s0, $0, done_isNumeric         # end of string      #2
    blt     $s0, $t0, not_isNumeric         # char < '0'?        #3
    bgt     $s0, $t1, not_isNumeric         # char > '9'?        #4
    addi    $a0, $a0, 1                     # increment pointer  #5
    j       lp_isNumeric
not_isNumeric:
    li      $v0, 0

done_isNumeric:
    lw      $s0, 4($sp)
    lw      $ra, 0($sp)
    addi    $sp, $sp, 8
    jr      $ra

.data
zero_isNumeric: .ascii "0"
nine_isNumeric: .ascii "9"
################### End function isNumeric
```

Options:

Line #5 should be addi $a0, $a0, 4

Line #1 should be lbu $s0, 0($a0)

Line #4 should be beq $s0, $t0, done

Line #2 should be bne $s0, $0, done_isNumeric

Line $4 should be j lp_isNumeric

4. Show what value is left in register $t0 after the following instructions have been executed: **3**

```
li      $t0,23              # $t0 = 23
li      $t1,5               # $t1 = 5
div     $t0, $t1
mflo    $t1
mfhi    $t0
```

5. Write a function called mpg to calculate a car's gas mileage, in miles per gallon, leaving the result in floating point register $f4. Use the following API (shown by comments):

```
##########   Begin function mpg
## Calculate gas mileage in miles per gallon
## Pre:   Register $f0 contains the number of miles driven (cannot be negative)
##        Register $f2 contains the number of gallons consumed (must be positive)
## Post:  Register $f4 contains the miles per gallon
## Author:   Sarah Pham
mpg:
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)             # return address

        div.s   $f4, $f0, $f2           # divide driven miles/gallons consumed
                                        # store in $f4

        lw      $ra, 0($sp)
        addi    $sp, $sp, 4             # get return address from stack
        jr      $ra

########## End function
```

Quiz 4

1. The move instruction is actually a pseudo-op. The assembler replaces it with a true MIPS instruction which is equivalent, i.e. performs the same function. The instruction "move $t3, $a1" will copy the value in register $a1 into register $t3. Which, 0 or more, of the following represent equivalent MIPS instructions?

addi	$t3, $a1, 0	**equivalent**
add	$a1, $t3, $0	**not equivalent**
add	$t3, $a1, $0	**equivalent**
addi	$t3, $a1, 1	**not equivalent**
sub	$t3, $a1, $0	**equivalent**

2. Show the following assembly language instruction in machine language, in hexadecimal. Your solution should be 8 hex digits. Use the steps below and show your work. The instruction is "addi $v0, $a0, 15". First, show the opcode and register fields in binary and the immediate field in hex. Then, group the binary fields into groups of 4 and convert each group of 4 bits to a hex digit.

- The opcode is 08 in hex according to semantics
- $rt is $v0, which is register 2
- $rs is $a0, which is register 4
- Immediate is 15 in decimal

opcode	rs	rt	immediate
00 1000	0 0100	0 0010	000f

Grouping of binary fields: 0010 0000 1000 0010

Converting into hex: **$2082\ 000f_{16}$**

Quiz 5

1. Convert the bne instruction shown below to machine language. Your final result should be a hexadecimal word. Use the steps provided, and show your work. First, show the opcode and register fields in binary (the immediate field can be shown in hex), group binary fields into groups of 4 bits, and convert to hex.

```
        bne     $a0, $a1, done
        add     $t0, $0, $0
        add     $a0, $a0, $a1
done:
        sw      $a0, result
```

opcode	rs	rt	immediate
00 0101	0 0100	0 0101	0002

Grouping into binary fields: 0001 0100 1000 0101
Converting into hex: **$1485\ 0002_{16}$**

2. Show in hexadecimal the floating point representation of -12.5. Show your work by showing the sign field, writing the absolute value of the number as a fixed-point number in binary, normalize the result from the previous step (single one to left of binary point, multiplied by a power of 2), determine the exponent in excess -127 notation, show the 23-bit fraction, dropping the high order 1, assembling the three fields into a 32-bit number, grouping the bits into 4, and writing the result in hexadecimal.

Sign: 1
Absolute value: 12.5 = 12 ½ = 1100.1
Normalizing: $1100.1 = 1.1001 * 2^3$
Excess -127 notation: 1000 0010 (130 - 127 = 3)
23-bit fraction: 1001 0000 0000 0000 0000 000
32-bit number: 11000001010010000000000000000000
Groups of 4: 1100 0001 0100 1000 0000 0000 0000 0000
Hexadecimal result: **$c148\ 0000_{16}$**

3. Convert the following program segment to machine language, showing the result in hexadecimal. Assume don't cares are always 0 (basically, fill out tables for add.s, c.eq.s, and bc1t and show them in binary and hex; the immediates can be in hex).

```
      c.eq.s   $f2, $f4
      bc1t     done
      add.s    $f2, $f4, $f6
done:
```

add.s $f2, $f4, $f6

opcode	fmt	ft	fs	fd	funct
01 0001	1 0000	0 0110	0 0100	0 0010	00 0000

Binary instruction: 0100 0110 0000 0110 0010 0000 1000 0000
Hexadecimal instruction: **$4606\ 2080_{16}$**

c.eq.s $f2, $f4

opcode	fmt	ft	fs	fd	funct
01 0001	1 0000	0 0100	0 0010	0 0000	11 0010

Binary instruction: 0100 0110 0000 0100 0001 0000 0011 0010
Hexadecimal instruction: **$4604\ 1032_{16}$**

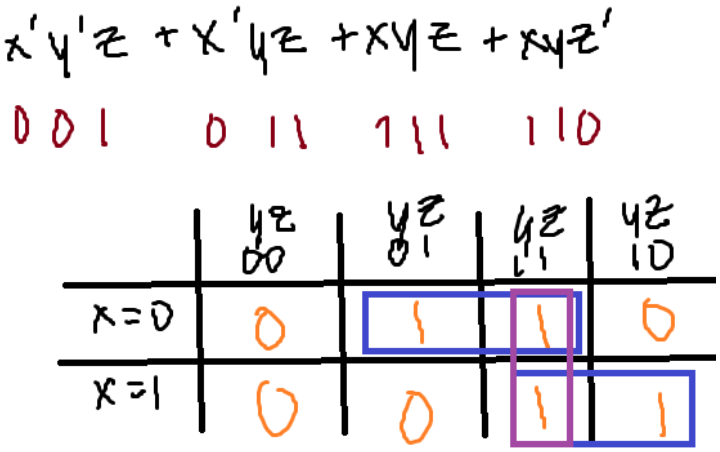
bc1t done

opcode	fmt	ft	immediate
01 0001	0 1000	0 0001	0001

Binary instruction: 0100 0101 0000 0001 0000 0000 0000 0001
Hexadecimal instruction: **$4501\ 0001_{16}$**

Quiz 6

1. The boolean expression x'y'z + x'yz + xyz + xyz' can be simplified to which of the following (choose the best answer)? Hint: Use a Karnaugh Map.



The answers from this K-map (that I got, and also the simplest one was confirmed by this calculator and https://getcalc.com/karnaugh-map/3variable-kmap-solver.htm) were x'z + xy and yz + x'y'z + xyz'.

However, we got the answer below deriving from the optimal solution. This was the best choice. Notice how the first two terms are nearly identical except for the *z*'s, which represent the term "xy" since *z* can be anything in these two situations.

Answer: **x'y'z + x'yz + xy**

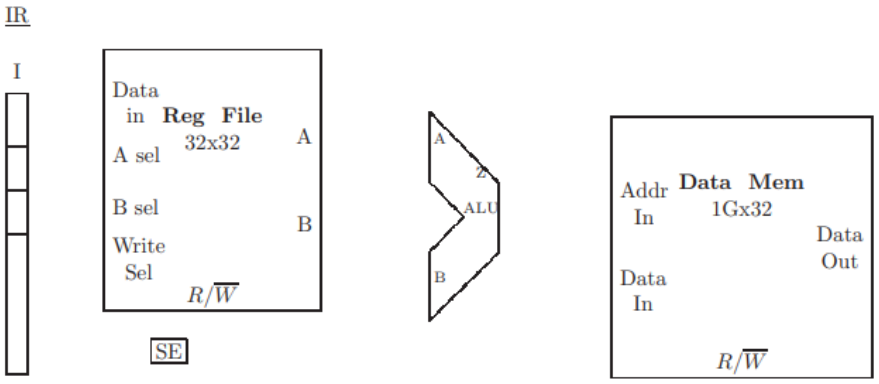
2. A 4x2 Multiplexer has binary control inputs 10. The data inputs to the MUX are I3 = 11, I2 = 10, I1 = 00, and I0 = 11. What is the output?

My reasoning: The input is 10 (binary), which is 2 in decimal, so select I2. The answer therefore is **10**.

3. Refer to the 32-bit ALU defined in section 6.5 of the textbook. If the A input is 0xf3ab0346 and the B input is 0xf3ab0346, and if the ALU operation select is 0110, what is the 32-bit output and Z-output? **0x00000000, 1**

Quiz 7

1. Refer to the below diagram to a portion of the MIPS datapath. It shows the Instruction Register (IR), the Register File (Reg File), the Arithmetic Logic Unit (ALU), Sign Extend (SE), and the Data Memory (Data Mem). In order to implement an sw instruction (I format), which of the following components must be connected with busses?



There must be a bus from the Data Mem output to the Reg File Data In. **true**
There must be a bus from the ALU output to the Data Mem Addr. **true**
There must be a bus from the IR imm field to the SE component input. **true**
There must be a bus from the Reg File B output to the Data Mem Addr In. **true**
There must be a bus from the IR rs field to the Data Mem Data In. **false**

2. Figure 7.22 in the textbook defines the outputs of the control unit, as a function of the instruction being executed. In our classwork we extended that table with two additional rows: One for the jr (jump register) instruction and one for the jal (jump and link) instruction. We will now fill in the column labeled DW for those two rows. The column labeled DW (Data Write) is the one-bit signal to the Data Memory to control writing to the Data Memory as shown in Figure 7.3 or Figure 7.10.

The DW output of the control unit for the jr instruction should be: **1**
The DW output of the control unit for the jal instruction should be: **1**

s