CS151B/EE116C - Homework #1, Due 4/10/2017 (questionnaire by Wed, 4/5)

Reading Assignment:

Chapter 1: pp. 3-22, 24-26

Appendix A: pp. A-3 – A-10, A-14 – A-15, A-18 – A-19, A-47 – A-81

Appendix B: pp. B-3 – B-20, B-48 – B-56, B-58 – B-61

Chapter 2: pp. 62-96, 111-120, 161-163

Chapter 3: pp. 178-181

Week 2 material: Appendix B: pp. B-26 – B-37 Chapter 4: pp. 244-272

Note: It is not nearly as bad as it looks. Most of the material should be known to you from CS M51A and CS 33. Scan through all the material. You do not need to really read in detail A-51 – A-80, just scan the material to know what is there.
Note that the material on the MIPS implementation in Appendix B (B-26 – B-37) and chapter 4 is preparation for next week's lecture.

Problems:

Note: When writing assembly code, use only assembly instructions and assembler directives. Do not use pseudoinstructions.
Unless specified otherwise, when writing MIPS assembly code, you **must** use **only** the $reg-number notation (e.g., $0, $22, $5) to specify a register. Any other notation will be considered incorrect.

(0) A) Please complete the online **Background Questionnaire** by Wednesday, April 5, **8:00am**. The questionnaire is on the UCLA CCLE CS M151B web site. To access the questionnaire, log in to https://ccle.ucla.edu/ using your UCLA logon ID. Select M151B, LEC 1 - Computer Systems Architecture. The link to the ''Background Questionnaire'' is on the first page.

B) Access the class web page at:
http://www.cs.ucla.edu/classes/spring17/csM151B
If you are enrolled in the class but cannot log in with your user name and password, as explained in class, contact the instructor.

C) Carefully go over the **errata** for the book and correct all the errors in your copy. There is a link to the errata on the class web page. This is **important** since the errors may confuse you when you least expect it...

D) Remember to always **staple together** the pages of your homework solutions. Failure to do so will result in a deduction of 15 points from the homework score.

(1) Figure B.8.8 on page B-55 illustrates the implementation of the read ports of the register file for the MIPS datapath. Your task is to design a new register file that has only two registers and each register has only three bits of data. This new register file has only one read port. Redraw Figure B.8.8 so that every wire in your diagram corresponds to only 1 bit of data (unlike the diagram in Figure B.8.8, in which some wires are 5 bits and some wires are 32 bits). Redraw the registers using D flip-flops. You do not need to show how to implement a D flip-flop or a multiplexer.

(2) Show the minimal sequence of MIPS instructions for the following C statement:

```
b[12-i] = b[i+j] - x ;
```

In the C program, `b` is declared as an array of four-byte integers, and `i, j, x` are declared as four-byte integer scalars. Variables `i, j, x` are stored in, respectively, registers 5, 8, and 13. The base

address of array b is 3,880,220 (note that this is decimal).

(3) Show the minimal sequence of MIPS instructions that extract bits 11 through 16 from register $11 and use that value to replace bits 26 through 31 of register $12, without changing register $11 and without changing the other 26 bits of register $12. Note that the bit numbering is little endian.

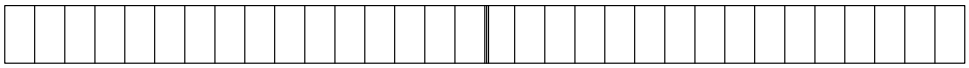
(4) Before the MIPS processor executes the following code segment, the value in register 5 ($5) is 29. What will be the values in register 1 and in register 2 after the entire code segment is executed? Show the values **in hex.** Explain your answer.

```
lui   $5, 414
sw    $5, 24($0)
lbu   $1, 25($0)
lbu   $2, 26($0)
```

(5) Translate the following MIPS assembly instruction into machine code.

```
sh  $4, 36($27)
```

Show the machine code in **binary**, one bit per square.



(6) Consider the following C code segment:

```
while  (a[i] != 33)  {
   if  (a[i] > y)   z = z + a[i] ;
   else             z = z - y ;
   i++ ;    }
```

Assume that all the variable are declared as four-byte integers. i,y,z are stored in registers $5,$7,$12, respectively. The base of array a is stored in register $11. The program was compiled into the assembly to the right. Convert the assembly program to machine code. Write the machine code **in the format shown in class**, with the address of each instruction to the left of the instruction. Note that **all** values should be presented in binary.

```
          .text  452
          addi   $15,$0,33
loop:     sll    $4,$5,2
          add    $4,$11,$4
          lw     $4,0($4)
          beq    $4,$15,next
          addi   $5,$5,1
          slt    $3,$7,$4
          beq    $3,$0,notlt
          add    $12,$12,$4
          j      loop
notlt:    sub    $12,$12,$7
          j      loop
next:
```

(7) Consider the seq pseudoinstruction (page A-59).
Produce **efficient** assembly code implementation of

```
sleu  $8, $13, $22
```

Use only real instructions.
Do not use any labels.
Your main goal is to minimize the number of assembly instructions. A secondary goal is to minimize the number of additional registers used.
State any necessary assumptions.

(8) Assume that the lhu instruction does not work. Thus, you **cannot** use the lhu instruction. In addition, you cannot use the lh instruction.
Assume that MIPS is a **big endian** processor.

Add a new pseudoinstruction to the MIPS assembly language. The new pseudoinstruction is `lhwrdu` (load a half-word from memory, unsigned). This pseudoinstruction requires the specification of a destination register number, a base register number, and an offset. The contents of the base register are added to the offset and the sum is the address of a (two byte) half word in memory which is loaded, unsigned, into the destination register. Thus, the semantics are thus exactly the same as for the original lhu instruction).

Show an **efficient** assembly code implementation of

```
lhwrdu  $7,240($14)
```

using **only** real MIPS instructions (but you cannot use the lh or lhu instructions). Minimize the use of registers. You **cannot** use any labels in your assembly code. State clearly **every** assumption you make.

(9) Write a MIPS assembly program that will produce different results depending on whether the processor is big endian or little endian. Specifically, your program must store the value 0 into the byte at address 149 if the processor is little endian but store the value 1 into the byte at address 149 if the processor is big endian. If necessary, your program may modify bytes 6-13 in main memory as well as registers $7, $8, and $9. Your program may not modify any other memory locations or registers.

Practice problems: You do not need to hand in a solution to the problems below.

(10) Draw a logic circuit of an adder of two 4-bit numbers. Your design should be simple and modular.

(11) Using the circuit from problem 10 as a building block, show the design of a 4-bit counter that counts down. The only external input to this circuit is the clock. The output four bits continuously follow the sequence:
. . . , 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 15, 14, 13, 12, 11, . . .
Your goal is to minimize the circuitry needed in addition to the module from problem 10.

(12) Instead of using the multiply instruction, it is possible to multiply using shift and add instructions. This may actually be faster when multiplying by a small constant. Suppose we want to store five times the value of register $16 into register $17, ignoring any overflow that may occur. Show a minimal sequence of MIPS instructions for doing this without using a multiply instruction.
Convert your program to machine code. Assume that your program is to be placed in memory starting address 128. Write the machine code **in the format shown in class**, with the address of each instruction to the left of the instruction. Note that **all** values should be presented in binary.

(13) Consider the `abs` pseudoinstruction (page A-51). Show an efficient assembly code implementation of

```
abs  $11,$4
```

Use only real instructions.
Do not use any labels.
State any necessary assumptions.

(14) Consider adding a new pseudoinstruction to the MIPS assembly language. The new instruction is `jgt` (jump-greater-than): the values of two registers are compared and if the first one is greater than the second, control is transferred to an arbitrary 32-bit destination address.

```
jgt   $7,$11,0x3A015432
```

will jump to location 0x3A015432 if the value in register $7 is greater than the value in register $11. Show an efficient assembly code implementation of the example above using only real MIPS instructions. State clearly any assumptions you make.

(15) Consider adding a new pseudoinstruction to the MIPS assembly language. The new pseudoinstruction is `mvmemb` (move-memory-byte): a single byte is copied from an arbitrary 32-bit source address to an arbitrary 32-bit destination address.

```
mvmemb    0x6ABD,0xDCBA9886
```

copies the single byte stored in address 0xDCBA9886 to address 0x6ABD.
Show an **efficient** assembly code implementation of the example above using **only** real MIPS instructions. State clearly **every** assumption you make.

(16) Add a new pseudoinstruction to the MIPS assembly language. The new pseudoinstruction is `swaph` (swap-half-words): the least significant 16 bits of the source register become the most significant 16 bits of the destination register and the most significant 16 bits of the source register become the least significant 16 bits of the destination register.

For example:

```
swaph   $8, $3
```

The source register is $3 and the destination register is $8.

Show an **efficient** assembly code implementation of the example above using **only** real MIPS instructions. Minimize the use of registers. You **cannot** use any labels in your assembly code. State clearly **every** assumption you make.

(17) Consider the following MIPS machine instruction, specified in binary:

1010 0010 1100 0101 0000 0000 0000 1100

Prior to the execution of this instruction, some of the registers contain the following values:

```
 $1 = 0x00002345  $2 = 0x000A0080
 $5 = 0x07AC6182  $7 = 0x000A0120
$11 = 0x54320010 $22 = 0x00030022
$26 = 0x00300060 $30 = 0x000A0820
```

Specify **all** the state changes that will be caused by the execution of this instruction. Be as specific as possible when indicating what changes and what are the new values. All values **must** be specified in **hex**.
Work through this carefully — it is easy to mess up...

(18) Convert the assembly program in the example on pages 92-93 of the book to machine code. Assume that the program is preceded by the directive

```
.text 136
```

Write the machine code **in the format shown in class**, with the address of each instruction to the left of the instruction. Note that **all** values should be presented in binary.