****

**Instruction Set Architecture Worksheet**

****

**Problem 1.**

An unnamed associate of yours has broken into the computer (a Beta of course!) that 6.004 uses for course administration. He has managed to grab the contents of the memory locations he believes holds the Beta code responsible for checking access passwords and would like you to help discover how the password code works. The memory contents are shown in the table below:

Addr Contents Opcode Rc Ra Rb Assembly

0x100 0xC05F0008 110000 00010 11111 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x104 0xC03F0000 110000 00001 11111 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x108 0xE060000F 111000 00011 00000 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x10C 0xF0210004 111100 00001 00001 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x110 0xA4230800 101001 00001 00011 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x114 0xF4000004 111101 00000 00000 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x118 0xC4420001 110001 00010 00010 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x11C 0x73E20002 011100 11111 00010 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x120 0x73FFFFF9 011100 11111 11111 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x124 0xA4230800 101001 00001 00011 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x128 0x605F0124 011000 00010 11111 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0x12C 0x90211000 100100 00001 00001 \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Further investigation reveals that the password is just a 32-bit integer which is in R0 when the code above is executed and that the system will grant access if R1 = 1 after the code has been executed. What "passnumber" will gain entry to the system?

**Problem 2.**

1. What assembly instruction could a compiler use to implement y = x \* 8 on the Beta assuming that MUL and MULC are not available?Assume x is in R0 and y is in R1.

**Equivalent assembly instruction: \_\_\_\_\_\_\_\_\_\_\_\_**

(B) Assume that the registers are initialized to: R0=8, R1=10, R2=12, R3=0x1234, R4=24 before execution of each of the following assembly instructions. For each instruction, provide the value of the specified register or memory location. **If your answers are in hexadecimal, make sure to prepend them with the prefix 0x.**

1. SHL(R3, R4, R5) **Value of R5: \_\_\_\_\_\_\_\_\_\_\_\_**
2. ADD(R2, R1, R6) **Value of R6: \_\_\_\_\_\_\_\_\_\_\_\_**
3. ADD(R0, 2, R7) **Value of R7: \_\_\_\_\_\_\_\_\_\_\_\_**
4. ST(R1, 4, R3) **Value stored: \_\_\_\_\_\_\_\_\_\_\_ at address: \_\_\_\_\_\_\_\_\_\_\_\_**

(C) A student tries to optimize his Beta assembly program by replacing a line  
containing  
 **ADDC(R0, 3\*4+5, R1)**  
by  
 **ADDC(R0, 17, R1)**  
Is the resulting binary program smaller? Does it run faster?  
  
 **(circle one) Binary program is SMALLER? yes … no  
  
 (circle one) FASTER? yes … no**

(D) A BR instruction at location 0x1000 branches to 0x2000. If the binary representation for that BR were moved to location 0x1400 and executed there, where will the relocated instruction branch to?  
  
  
 **Branch target for relocated BR (in hex): 0x \_\_\_\_\_\_\_\_\_\_\_\_**

(E) A line in an assembly-language program containing “ADDC(R1,2,R3)” is changed to “ADDC(R1,R2,R3)”. Will the modified program behave differently when executed?  
  
  
 **Circle best answer: YES … NO … CAN’T TELL**

**Problem 3.**

Each of the following programs is loaded into a Beta’s main memory starting at location 0 and execution is started with the Beta’s PC set to 0. Assume that all registers have been initialized to 0 before execution begins. Please determine the specified values after execution reaches the HALT() instruction and the Beta stops. Write “CAN’T TELL” if the value cannot be determined.

**Please write all values in hex.**

(A)   
 **Value left in R1: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
 Value left in R2: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

. = 0   
 LD(R31,X+4,R1)

SHLC(R1,2,R1)

LD(R1,X,R2)

HALT()

X: LONG(4)

LONG(3)

LONG(2)

LONG(1)

LONG(0)

(B)   
  
 **Value left in R0: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **Value left in R1: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
 Value left in R2: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
 Value assembler assigns to symbol X: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

. = 0   
 LD(R31,X,R0)  
 CMOVE(0,R1)

L: CMPLTC(R0,0,R2)

BNE(R2,DONE)  
 ADDC(R1,1,R1)

SHLC(R0,1,R0)

BR(L)

DONE: HALT()

X: LONG(0x08306352)

(C)   
  
 **Value left in R1: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
 Value left in R2: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

. = 0   
 LD(R31,Z,R1)  
 SHRC(R1,26,R1)  
Z: CMPLTC(R1,0x3C,R2)

HALT()

(D)   
  
 **Value left in R0: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **Value left in R1: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
 Value left in R2: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
 Value assembler assigns to symbol X: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

. = 0   
 LD(R31,X,R0)  
 CMOVE(0,R1)

L: ADDC(R1,1,R1)

SHRC(R0,1,R0)  
 BNE(R0,L,R2)

HALT()

. = 0x100

X: LONG(5)

(E)

. = 0

LD(r31, X, r0)

CMPLE(r0, r31, r1)

BNE(r1, L1, r1)

ADDC(r31, 17, r2)

BEQ(r31, L2, r31)

L1: SRAC(r0, 4, r2)

L2: HALT()

. = 0x1CE8

X: LONG(0x87654321)

**Value left in R0? 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
  
 Value left in R1? 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
  
 Value left in R2? 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
  
 Value assembler assigns to L1: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

. = 0  
 LD(R31, i, R0)  
 SHLC(R0, 2, R0)

LD(R0, a-4, R1)

HALT()

a: LONG(0xBADBABE)  
 LONG(0xDEADBEEF)  
 LONG(0xC0FFEE)

LONG(0x8BADF00D)

i: LONG(3)

(F) **Contents of R0 (in hex): 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
 Contents of R1 (in hex): 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(G)   
 . = 0 **Value left in R1: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**   
 LD(R31,Z,R1)  
 SHRC(R1,16,R2)  
Z: SUBC(R2,0x3C,R3) **Value left in R3: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  
 HALT()

**Value assembler assigns to symbol Z: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(H) . = 0   
 LD(R31,X,R0)  **Value left in R0: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**   
 CMOVE(0,R1)  
  
L: ADDC(R1,1,R1) **Value left in R1: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  
 SHRC(R0,1,R0)  
 BNE(R0,L,R2)  
 HALT() **Value left in R2: 0x\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  
  
X: LONG(0xDECAF)