**Instruction Set Architecture Worksheet**

### Summary of β Instruction Formats

#### Operate Class:

<table>
<thead>
<tr>
<th>31 26 25 21 20 16 15 11 10 0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10xxxx</td>
<td>Rc</td>
<td>Ra</td>
<td>Rb</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OP(Ra,Rb,Rc): Reg[Rc] ← Reg[Ra] op Reg[Rb]

Opcodes: ADD (plus), SUB (minus), MUL (multiply), DIV (divided by), AND (bitwise and), OR (bitwise or), XOR (bitwise exclusive or), XNOR (bitwise exclusive nor), CMPEQ (equal), CMPLT (less than), CMPLE (less than or equal) [result = 1 if true, 0 if false], SHL (left shift), SHR (right shift w/o sign extension), SRA (right shift w/ sign extension)

<table>
<thead>
<tr>
<th>31 26 25 21 20 16 15 0</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11xxxx</td>
<td>Rc</td>
<td>Ra</td>
<td>literal</td>
<td></td>
</tr>
</tbody>
</table>

OPC(Ra,literal,Rc): Reg[Rc] ← Reg[Ra] op SEXT(literal)

Opcodes: ADDC (plus), SUBC (minus), MULC (multiply), DIVC (divided by), ANDC (bitwise and), ORC (bitwise or), XORC (bitwise exclusive or), XNORC (bitwise exclusive nor), CMPEQC (equal), CMPLTC (less than), CMPLEC (less than or equal) [result = 1 if true, 0 if false], SHLC (left shift), SHRC (right shift w/o sign extension), SRAC (right shift w/ sign extension)

#### Other:

<table>
<thead>
<tr>
<th>31 26 25 21 20 16 15 0</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01xxxx</td>
<td>Rc</td>
<td>Ra</td>
<td>literal</td>
<td></td>
</tr>
</tbody>
</table>

LD(Ra,literal,Rc): Reg[Rc] ← Mem[Reg[Ra] + SEXT(literal)]

ST(Rc,literal,Ra): Mem[Reg[Ra] + SEXT(literal)] ← Reg[Rc]

JMP(Ra,Rc): Reg[Rc] ← PC + 4; PC ← Reg[Ra]

BEQ/BF(Ra,label,Rc): Reg[Rc] ← PC + 4; if Reg[Ra] = 0 then PC ← PC + 4 + 4*SEXT(literal)

BNE/BT(Ra,label,Rc): Reg[Rc] ← PC + 4; if Reg[Ra] ≠ 0 then PC ← PC + 4 + 4*SEXT(literal)

LDR(label,Rc): Reg[Rc] ← Mem[PC + 4 + 4*SEXT(literal)]

#### Opcode Table: (*optional opcodes)

<table>
<thead>
<tr>
<th>2:0</th>
<th>5:3</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>000</td>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>LD</td>
<td>ST</td>
<td>JMP</td>
<td>BEQ</td>
<td>BNE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>ADD</td>
<td>SUB</td>
<td>MUL</td>
<td>DIV</td>
<td>CMPEQ</td>
<td>CMPLT</td>
<td>CMPLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>AND</td>
<td>OR</td>
<td>XOR</td>
<td>XNOR</td>
<td>SHL</td>
<td>SHR</td>
<td>SRA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>ADDC</td>
<td>SUBC</td>
<td>MULC</td>
<td>DIVC</td>
<td>CMPEQC</td>
<td>CMPLTC</td>
<td>CMPLEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>ANDC</td>
<td>ORC</td>
<td>XORC</td>
<td>XNORC</td>
<td>SHLC</td>
<td>SHRC</td>
<td>SRAC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Addr</th>
<th>Contents</th>
<th>Opcode</th>
<th>Rc</th>
<th>Ra</th>
<th>Rb</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>0xC05F0008</td>
<td>110000</td>
<td>00010</td>
<td>11111</td>
<td></td>
<td><strong>ADDCl (R3), OX8, R2</strong></td>
</tr>
<tr>
<td>0x104</td>
<td>0xC03F0000</td>
<td>110000</td>
<td>00001</td>
<td>11111</td>
<td></td>
<td><strong>ADDCl (R3), OX0, R1</strong></td>
</tr>
<tr>
<td>0x108</td>
<td>0xE060000F</td>
<td>111000</td>
<td>00011</td>
<td>00000</td>
<td></td>
<td><strong>ANDCl (R0, OXF, R3)</strong></td>
</tr>
<tr>
<td>0x10C</td>
<td>0xF0210004</td>
<td>111100</td>
<td>00001</td>
<td>00001</td>
<td></td>
<td><strong>SHLC (R1, OX4, R1)</strong></td>
</tr>
<tr>
<td>0x110</td>
<td>0xA4230800</td>
<td>101001</td>
<td>00001</td>
<td>00001</td>
<td>0001</td>
<td><strong>OR (R3, R1, R1)</strong></td>
</tr>
<tr>
<td>0x114</td>
<td>0xF4000004</td>
<td>111101</td>
<td>00000</td>
<td>00000</td>
<td></td>
<td><strong>SHRC (R0, OX4, R0)</strong></td>
</tr>
<tr>
<td>0x118</td>
<td>0xC4420001</td>
<td>110001</td>
<td>00010</td>
<td>00010</td>
<td></td>
<td><strong>SUBC (R2, OX1, R2)</strong></td>
</tr>
<tr>
<td>0x11C</td>
<td>0x73E20002</td>
<td>011100</td>
<td>11111</td>
<td>00010</td>
<td></td>
<td><strong>BEQ (R2, L1, R3)</strong></td>
</tr>
<tr>
<td>0x120</td>
<td>0x73FFFFFFF</td>
<td>011100</td>
<td>11111</td>
<td>11111</td>
<td></td>
<td><strong>BEQ (R3, L2, R3)</strong></td>
</tr>
<tr>
<td>0x124</td>
<td>0xA4230800</td>
<td>101001</td>
<td>00001</td>
<td>00011</td>
<td></td>
<td><strong>LD (R3, OX12A, R2)</strong></td>
</tr>
<tr>
<td>0x128</td>
<td>0x605F0124</td>
<td>011000</td>
<td>00010</td>
<td>11111</td>
<td></td>
<td><strong>CMPEQ (R1, R2, R1)</strong></td>
</tr>
</tbody>
</table>

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?

The loop reverses the order of the nibbles (4-bit chunks) of the value in R0, e.g., 0x12345678 becomes 0x87654321.

So the "passnumber" is the nibble reverse of 0xA4230800 which is 0x0080324A.
Problem 2.

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

Equivalent assembly instruction: \( \text{SHLC(R0, 3, R1)} \)

(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. \( \text{SHL(R3, R4, R5)} \)
   Value of \( R5: \quad 0x3400 \quad 0000 \)

2. \( \text{ADD(R2, R1, R6)} \)
   Value of \( R6: \quad 22 \quad \) 

3. \( \text{ADD(R0, 2, R7)} \)
   Value of \( R7: \quad 20 \quad \) 

4. \( \text{ST(R1, 4, R3)} \)
   Value stored: \( 10 \quad \) at address: \( 0x138 \frac{A}{x+0x1234}{ } \quad 0x1238 \)

(C) A student tries to optimize his Beta assembly program by replacing a line containing \( \text{ADDC(R0, 3*4+5, R1)} \)
by \( \text{ADDC(R0, 17, R1)} \)
Is the resulting binary program smaller? Does it run faster?

(circle one) Binary program is SMALLER? \( \quad \text{yes} \quad \quad \text{no} \)
(circle one) FASTER? \( \quad \text{yes} \quad \quad \text{no} \)

(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?

Original branch offset \( \text{(0x1000) now relative to 0x1400} \)

Branch target for relocated BR (in hex): \( 0x \quad 2400 \)

(E) A line in an assembly-language program containing \( "\text{ADDC(R1,2,R3)}" \) is changed to \( "\text{ADDC(R1,R2,R3)}" \). Will the modified program behave differently when executed?

Circle best answer: \( \text{YES} \quad \text{NO} \quad \text{CAN'T TELL} \)

Interpret 2nd operand as a constant expression.
Value of symbol \( R2 \) is 2.
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) . = 0
LD(R31,X+4,R1)
SHLC(R1,2,R1)
LD(R1,X,R2)
HALT()

X: LONG(4)
+14 LONG(3)
+18 LONG(2)
+12 LONG(1)
LONG(0)

R1 ← R2
R1 ← 3

Value left in R1: 0x C

R2 ← Mem[X+A]

Value left in R2: 0x 1

(B) . = 0
LD(R31,X,R0)
CMOVE(0,R1)
CMPLTC(R0,0,R2)
BNE(R2,DONE)
ADDC(R1,1,R1)
SHLC(R0,1,R0)
BR(L)
DONE: HALT()

X: LONG(0x08306352)

Value assembler assigns to symbol X: 0x 20

Counts # of left shifts needed until MSB of R0 is 1.

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

R1 ← binary CMPCTC int.
R1 ← opcode field

Value left in R1: 0x 35 (CMPCTC opcode)

Value left in R2: 0x 1

(D) . = 0
LD(R31,X,R0)
CMOVE(0,R1)
ADDC(R1,1,R1)
SHRC(R0,1,R0)
BNE(R0,L,R2)
HALT()

. = 0x100
X: LONG(5)

Value assembler assigns to symbol X: 0x 100

R0 ← 5

Value left in R0: 0x 0

Value left in R1: 0x 3

Value left in R2: 0x 14

6.004 Worksheet - 4 of 5
(E)  
\[ . = 0 \]
\[ \text{LD}(r31, X, r0) \quad R0 \leftarrow 0x87654321 \] (negative!)
\[ \text{CMPE}(r0, r31, r1) \quad R1 \leftarrow 1 \] Value left in R0? 0x87654321
\[ \text{BNE}(r1, L1, r1) \quad R1 \leftarrow 0 \] branch taken
\[ \text{ADDC}(r31, 17, r2) \quad R2 \leftarrow 0x0, c \] Value left in R1? 0x
\[ \text{BEQ}(r31, L2, r31) \]
\[ \text{L1: SRAC}(-r0, 4, r2) \quad R2 \leftarrow 0xF, 8765432 \]
\[ \text{L2: HALT}() \]
\[ . = 0x1CE8 \]
\[ X: \text{LONG}(0x87654321) \]

Value assembler assigns to L1: 0x14

(F)  
\[ . = 0 \]
\[ \text{LD}(R31, i, R0) \quad R0 \leftarrow 3 \]
\[ \text{SHLC}(R0, 2, R0) \quad R0 \leftarrow 12 \]
\[ \text{LD}(R0, a-4, R1) \quad R1 \leftarrow \text{Mem}[12 + a - 4] \]
\[ \text{HALT}() \]
\[ a: \text{LONG}(0xBADABE) \]
\[ +4 \quad \text{LONG}(0xDEADBEEF) \]
\[ +8 \quad \text{LONG}(0xCOFFEE) \]
\[ \quad \text{LONG}(0x8BADFOOD) \]
\[ i: \text{LONG}(3) \]

Contents of R0 (in hex): 0xC

Contents of R1 (in hex): 0xCFEE

(G)  
\[ . = 0 \]
\[ \text{LD}(R31, Z, R1) \quad R1 \leftarrow \text{binary for SUBC} \]
\[ \text{SHRC}(R1, 16, R2) \quad R2 \leftarrow \text{top half} R1 \]
\[ \text{Z: SUBC}(R2, 0x3C, R3) \]
\[ \text{HALT}() \]
\[ \text{Value assembler assigns to symbol Z: 0x3} \]

(H)  
\[ . = 0 \]
\[ \text{LD}(R31, X, R0) \quad R0 \leftarrow \text{DECAF} \]
\[ \text{CMOVE}(0, R1) \]
\[ \text{L: ADDC}(R1, 1, R1) \]
\[ \text{SHRC}(R0, 1, R0) \]
\[ \text{BNE}(R0, L, R2) \]
\[ \text{HALT}() \]
\[ X: \text{LONG}(0x0DECAF) \]

Value left in R1: 0xC620C3C

Value left in R3: 0xC42C

Value left in R2: 0x14

Value left in R0: 0x0

Value left in R1: 0x14

X: LONG(0x0DECAF)

\[ \text{count # of right shifts to make R0 equal to R0.} \]