Summary of β Instruction Formats

Operate Class:

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>0</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>
<td></td>
</tr>
</tbody>
</table>

OP(Ra,Rb,Rc): \( \text{Reg[Rc]} \leftarrow \text{Reg[Ra]} \text{ op } \text{Reg[Rb]} \)

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), CMPEQ (less than or equal) \([\text{result} = 1 \text{ if true}, 0 \text{ if false}]\)
SHL (left shift), SHR (right shift w/o sign extension), SRA (right shift w/ sign extension)

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>11xxxx</td>
<td>Rc</td>
<td>Ra</td>
<td>literal (two’s complement)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPC(Ra,literal,Rc): \( \text{Reg[Rc]} \leftarrow \text{Reg[Ra]} \text{ op } \text{SEXT(literal)} \)

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), CMPEQC (less than or equal) \([\text{result} = 1 \text{ if true}, 0 \text{ if false}]\)
SHLC (left shift), SHRC (right shift w/o sign extension), SRAC (right shift w/ sign extension)

Other:

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>01xxxx</td>
<td>Rc</td>
<td>Ra</td>
<td>literal (two’s complement)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LD(Ra,literal,Rc): \( \text{Reg[Rc]} \leftarrow \text{Mem[Reg[Ra] + SEXT(literal)]} \)
ST(Rc,literal,Ra): \( \text{Mem[Reg[Ra] + SEXT(literal)]} \leftarrow \text{Reg[Rc]} \)
JMP(Ra,Rc): \( \text{Reg[Rc]} \leftarrow \text{PC + 4}; \text{PC} \leftarrow \text{Reg[Ra]} \)
BEQ/BF(Ra,lab, Rc): \( \text{Reg[Rc]} \leftarrow \text{PC + 4}; \text{if Reg[Ra] = 0} \text{ then PC} \leftarrow \text{PC + 4 + 4*SEXT(literal)} \)
BNE/BT(Ra,lab, Rc): \( \text{Reg[Rc]} \leftarrow \text{PC + 4}; \text{if Reg[Ra] = 0} \text{ then PC} \leftarrow \text{PC + 4 + 4*SEXT(literal)} \)
LDR(label,Rc): \( \text{Reg[Rc]} \leftarrow \text{Mem[PC + 4 + 4*SEXT(literal)]} \)

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>000</td>
<td>LD</td>
<td>ST</td>
<td>JMP</td>
<td>BEQ</td>
<td>BNE</td>
<td>LDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>ADD</td>
<td>SUB</td>
<td>MUL*</td>
<td>DIV*</td>
<td>CMPEQ</td>
<td>CMPLT</td>
<td>CMPEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</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>011</td>
<td>ADDC</td>
<td>SUBC</td>
<td>MULC*</td>
<td>DIVC*</td>
<td>CMPEQC</td>
<td>CMPLT</td>
<td>CMPEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</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>101</td>
<td>ADDC</td>
<td>SUBC</td>
<td>MULC*</td>
<td>DIVC*</td>
<td>CMPEQC</td>
<td>CMPLT</td>
<td>CMPEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</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>111</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>
</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>\text{ADDX} (R3), 0x8, R2</td>
</tr>
<tr>
<td>0x104</td>
<td>0xC03F0000</td>
<td>110000</td>
<td>00001</td>
<td>11111</td>
<td></td>
<td>\text{ADDX} (R3), 0x0, R1</td>
</tr>
<tr>
<td>0x108</td>
<td>0xE060000F</td>
<td>111000</td>
<td>00011</td>
<td>00000</td>
<td></td>
<td>\text{ANDC} (R0, 0xF, R3)</td>
</tr>
<tr>
<td>0x10C</td>
<td>0xF0210004</td>
<td>111100</td>
<td>00001</td>
<td>00001</td>
<td></td>
<td>\text{SHLC} (R1, 0x4, R1)</td>
</tr>
<tr>
<td>0x110</td>
<td>0xA4230800</td>
<td>101001</td>
<td>00001</td>
<td>00011</td>
<td>\text{add}</td>
<td>\text{OR} (R3, R1, R1)</td>
</tr>
<tr>
<td>0x114</td>
<td>0xF400004</td>
<td>111101</td>
<td>00000</td>
<td>00000</td>
<td></td>
<td>\text{SHLC} (R0, 0x4, R0)</td>
</tr>
<tr>
<td>0x118</td>
<td>0xC4420001</td>
<td>110001</td>
<td>00010</td>
<td>00010</td>
<td></td>
<td>\text{SUBC} (R2, 0x1, R2)</td>
</tr>
<tr>
<td>0x11C</td>
<td>0x73E20002</td>
<td>011100</td>
<td>11111</td>
<td>00010</td>
<td></td>
<td>\text{BEQ} (R2, L1, R3)</td>
</tr>
<tr>
<td>0x120</td>
<td>0x73FFFFF9</td>
<td>011100</td>
<td>11111</td>
<td>11111</td>
<td></td>
<td>\text{BEQ} (R3, L2, R3)</td>
</tr>
<tr>
<td>0x124</td>
<td>0xA4230800</td>
<td>101001</td>
<td>00001</td>
<td>00011</td>
<td></td>
<td>\text{not executed!}</td>
</tr>
<tr>
<td>0x128</td>
<td>0x605F0124</td>
<td>011000</td>
<td>00010</td>
<td>11111</td>
<td></td>
<td>\text{LD} (R3), 0x124, R2</td>
</tr>
<tr>
<td>0x12C</td>
<td>0x90210000</td>
<td>100100</td>
<td>00001</td>
<td>00011</td>
<td></td>
<td>\text{CMPB} (R1, R2, R1)</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 0xA4230820 which is 0x00B0324A.
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: \( \text{SHL}(R0, 3, R1) \)

(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 \( R5 \): \( 0x3400 \ 0000 \)

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

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

4. \( \text{ST}(R1, 4, R3) \) Value stored: \( 10 \) at address: \( 0x1238 \)

(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? yes \( \text{no} \)

(circle one) FASTER? yes \( \text{no} \)

(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 target (\( 0x1000 \)) now relative to \( 0x1400 \)

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

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

Interpret 2nd operand as a constant expression. Value of symbol \( R2 \) is 2.

Circle best answer: YES \( \text{NO} \) 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)

\[
\begin{align*}
. & = 0 \\
LD(R31,X+4,R1) & \quad R1 \leftarrow 3 \\
SHLC(R1,2,R1) & \quad R1 \leftarrow R2 \\
LD(R1,X,R2) & \quad R2 \leftarrow \text{Mem}[X+12] \\
HALT() & \\
X: & \text{LONG(4)} \\
14 & \text{LONG(3)} \\
18 & \text{LONG(2)} \\
12 & \text{LONG(1)} \\
0 & \text{LONG(0)}
\end{align*}
\]

Value left in R1: 0x\underline{C

Value left in R2: 0x\underline{1

(B)

\[
\begin{align*}
. & = 0 \\
LD(R31,X,R0) & \quad R0 \leftarrow 35 \text{ (CMPRTC opcode)} \\
CMOVE(0,R1) & \\
CMPLTC(R0,0,R2) & \\
BNE(R2,DONE) & \\
ADDC(R1,1,R1) & \quad \text{Value left in R0: } 0x\underline{83063520} \\
SHLC(R0,1,R0) & \quad \text{Value left in R1: } 0x\underline{4} \\
BR(L) & \quad \text{Value left in R2: } 0x\underline{1} \\
DONE: & \text{HALT()} \\
X: & \text{LONG(0x08306352)}
\end{align*}
\]

Value assembler assigns to symbol X: 0x\underline{20

(C)

\[
\begin{align*}
. & = 0 \\
LD(R31,Z,R1) & \quad \text{R1 < binnmtr CMPRTC inst.} \\
SHRC(R1,26,R1) & \quad \text{R1 = opcode field} \\
Z: & \text{CMPLTC(R1,0x3C,R2)} \\
HALT() & \quad \text{Value left in R1: } 0x\underline{35} \\
& \quad \text{Value left in R2: } 0x\underline{1}
\end{align*}
\]

(D)

\[
\begin{align*}
. & = 0 \\
LD(R31,X,R0) & \quad \text{RO<5} \\
CMOVE(0,R1) & \\
L: & \text{ADDC(R1,1,R1)} \\
SHRC(R0,1,R0) & \quad \text{Value left in R0: } 0x\underline{0} \\
BNE(R0,L,R2) & \quad \text{Value left in R1: } 0x\underline{3} \\
HALT() & \quad \text{Value left in R2: } 0x\underline{14} \\
. & = 0x100 \\
X: & \text{LONG(5)}
\end{align*}
\]

Value assembler assigns to symbol X: 0x\underline{100

\[\text{6.004 Worksheet - 4 of 5 - LO8 - Instruction Set Architecture}\]
(E) 
. = 0
LD(r31, X, r0) R0 = 0x87654321 (negative!)
CMPE(r0, r31, r1) R1 = 0x87654321  
Value left in R0? 0x87654321
BNE(r1, L1, r1)  
Value left in R1? 0x  
ADDC(r31, 17, r2) R1 = 0x  
BEQ(r31, L2, r31)  
Value left in R2? 0x  
L1: SRAC(r0, 4, r2) R2 = 0xF8765432  
Value assembler assigns to L1: 0x  
L2: HALT()

. = 0x1CE8
X: LONG(0x87654321)

(F) 
. = 0
LD(R31, i, R0) R0 = 0x3  
SHLC(R0, 2, R0) R0 = 0x12  
LD(R0, a-4, R1) R1 = Mem[12 + a - 4]  
HALT()

a: LONG(0xBADBABE)  0
+4 LONG(0xDEADBEEF)  0
+8 LONG(0xCOFFEE)  0
+12 LONG(0x8BADFOOD)  0
i: LONG(3)

(G) 
. = 0
LD(R31, Z, R1) R1 = binary for SUBC  
SHRC(R1, 16, R2) R2 = top half R1  
Z: SUBC(R2, 0x3C, R3)  
HALT()

Value assembler assigns to symbol Z: 0x  

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

X: LONG(0xDECAF)  

Value left in R1: 0x  
Value left in R3: 0x  
Value left in R0: 0x  
Value left in R1: 0x  
Value left in R2: 0x  

6.004 Worksheet