Summary of β Instruction Formats

Operate Class:

\[
\begin{array}{ccccccc}
31 & 25 & 20 & 15 & 11 & 10 & 0 \\
\hline
10xxxx & Rc & Ra & Rb & unused \\
\end{array}
\]

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

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

\[
\begin{array}{ccccccc}
31 & 25 & 20 & 15 & 11 & 10 & 0 \\
\hline
11xxxx & Rc & Ra & literal (two’s complement) \\
\end{array}
\]

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

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

Other:

\[
\begin{array}{ccccccc}
31 & 25 & 20 & 15 & 11 & 10 & 0 \\
\hline
01xxxx & Rc & Ra & literal (two’s complement) \\
\end{array}
\]

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

Opcode Table: (*optional opcodes)

<table>
<thead>
<tr>
<th>2:0</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td></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>LDR</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>
</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>
</tr>
<tr>
<td>110</td>
<td>ADDC</td>
<td>SUBC</td>
<td>MULC*</td>
<td>DIVE</td>
<td>CMPEQC</td>
<td>CMPLTC</td>
<td>CMPLEC</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>
</tr>
</tbody>
</table>

6.004 Worksheet - 1 of 5 - L09 – Instruction Set Architecture
**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>0xC05F008</td>
<td>110000</td>
<td>00010</td>
<td>11111</td>
<td></td>
<td><strong>ADD</strong> (R3), OX8, R2</td>
</tr>
<tr>
<td>0x104</td>
<td>0xC03F000</td>
<td>110000</td>
<td>00011</td>
<td>11111</td>
<td></td>
<td><strong>ADD</strong> (R3), OX0, R1</td>
</tr>
<tr>
<td>0x108</td>
<td>0xE06000F</td>
<td>111000</td>
<td>00011</td>
<td>00000</td>
<td></td>
<td><strong>AND</strong> (R0, OXF, R3)</td>
</tr>
<tr>
<td>0x110</td>
<td>0xF021004</td>
<td>111100</td>
<td>00011</td>
<td>00001</td>
<td></td>
<td><strong>SHLC</strong> (R1, OX9, R1)</td>
</tr>
<tr>
<td>0x110</td>
<td>0xA4230800</td>
<td>010101</td>
<td>00011</td>
<td>00011</td>
<td></td>
<td><strong>OR</strong> (R3, R1, R1)</td>
</tr>
<tr>
<td>0x114</td>
<td>0xF400004</td>
<td>111101</td>
<td>00000</td>
<td>00000</td>
<td></td>
<td><strong>SHRC</strong> (R0, OX4, R0)</td>
</tr>
<tr>
<td>0x118</td>
<td>0xC442001</td>
<td>110001</td>
<td>00010</td>
<td>00010</td>
<td></td>
<td><strong>SUBC</strong> (R2, OX1, R2)</td>
</tr>
<tr>
<td>0x11C</td>
<td>0x73E2002</td>
<td>011100</td>
<td>11111</td>
<td>00010</td>
<td></td>
<td><strong>BEQ</strong> (R2, L1, R3)</td>
</tr>
<tr>
<td>0x120</td>
<td>0x73FFFFF9</td>
<td>011100</td>
<td>11111</td>
<td>11111</td>
<td></td>
<td><strong>BEQ</strong> (R3, L2, R3)</td>
</tr>
<tr>
<td>0x124</td>
<td>0xA4230800</td>
<td>010101</td>
<td>00001</td>
<td>00011</td>
<td></td>
<td><strong>LD</strong> (R3, OX12A, R2)</td>
</tr>
<tr>
<td>0x128</td>
<td>0x89040124</td>
<td>011000</td>
<td>00010</td>
<td>11111</td>
<td></td>
<td><strong>CMPEQ</strong> (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 0x0032400A.
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: \( \text{SHLC}(R2, 3, R1) \)

(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 \( R5 \): \( 0x34000000 \)
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: \( 0x1234 < 0x1238 \)

(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 \( \bigcirc \) no \( \bigcirc \)
(circle one) FASTER? yes \( \bigcirc \) no \( \bigcirc \)

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

Branch target for relocated BR (in hex): \( 0x2400 \)

(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 \( \bigcirc \) NO \( \bigcirc \) CAN’T TELL

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

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,R1)
SHLC(R1,2,R1)
LD(R1,X,R2)
HALT()
X: LONG(4)
  +4
  +12

R1 ← 3
R1 ← R2
R2 ← [Mem[X+12]]

Value left in R1: 0x C
Value left in R2: 0x 1

(B) . = 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)

Value assembler assigns to symbol X: 0x 20

Value left in R0: 0x 83063520
Value left in R1: 0x 4
Value left in R2: 0x 1

4 counts # of left shifts needed until MSB of R2 is 1.

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

R1 ← binary for CMPLTC inst.
R1 ← opcode field

Value left in R1: 0x 35 (CMPLTC opcode)
Value left in R2: 0x 1

(D) . = 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)

Value assembler assigns to symbol X: 0x 100

Value left in R0: 0x 0
Value left in R1: 0x 3
Value left in R2: 0x 14

# of right shifts until R0 is 0.

6.004 Worksheet - 4 of 5 -
(E) . = 0
LD(r31, X, r0) R0 = 0x87654321 (negative!)
CMPLE(r0, r31, r1) R1 = 1
BNE(r1, L1, r1)
ADDC(r31, 17, r2) R1 = 0x3c, branch taken
BEQ(r31, L2, r31)
L1: SRAC(r0, 4, r2) R2 = 0x8765432
L2: HALT()

. = 0x1ce8
X: LONG(0x87654321)

Value assembler assigns to L1: 0x14

(F) . = 0
LD(R31, i, R0) R0 = 3
SHLC(R0, 2, R0)
LD(R0, a-4, R1) R1 = Mem[12 + a - 4]
HALT()

a: LONG(0xbadbabe)
+4 LONG(0xdeadbeef)
+B LONG(0xc0ffee)
LONG(0x8badfood)

i: LONG(3)

(G) . = 0
LD(R31, Z, R1) R1 = binary for SUBC
SHRC(R1, 16, R2)
Z: SUBC(R2, 0x3c, R3)
HALT()

Value assembler assigns to symbol Z: 0x83

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

X: LONG(0xDeCAF)

Value left in R0: 0x0
Value left in R1: 0x14
Value left in R2: 0x14

Value left in R3: 0xc426
Value left in R1: 0xc03b