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: \( \text{ADD} \) (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), \( \text{CMPLT} \) (less than), \( \text{CMPLE} \) (less than or equal) \[ \text{result} = 1 \text{ if true, } 0 \text{ if false} \]
\( \text{SHL} \) (left shift), \( \text{SHR} \) (right shift w/o sign extension), \( \text{SRA} \) (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: \( \text{ADDC} \) (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) \[ \text{result} = 1 \text{ if true, } 0 \text{ if false} \]
\( \text{SHLC} \) (left shift), \( \text{SHRC} \) (right shift w/o sign extension), \( \text{SRAC} \) (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): \( \text{Reg}[Rc] \leftarrow \text{Mem}[\text{Reg}[Ra] + \text{SEXT(literal)}] \)
ST(Rc,literal,Ra): \( \text{Mem}[\text{Reg}[Ra] + \text{SEXT(literal)}] \leftarrow \text{Reg}[Rc] \)
JMP(Ra,Rc): \( \text{Reg}[Rc] \leftarrow \text{PC} + 4; \text{PC} \leftarrow \text{Reg}[Ra] \)
BEQ/ BF(Ra,label,Rc): \( \text{Reg}[Rc] \leftarrow \text{PC} + 4; \text{if } \text{Reg}[Ra] = 0 \text{ then } \text{PC} \leftarrow \text{PC} + 4 + 4\times \text{SEXT(literal)} \)
BNE/ BT(Ra,label,Rc): \( \text{Reg}[Rc] \leftarrow \text{PC} + 4; \text{if } \text{Reg}[Ra] \neq 0 \text{ then } \text{PC} \leftarrow \text{PC} + 4 + 4\times \text{SEXT(literal)} \)
LDR(label,Rc): \( \text{Reg}[Rc] \leftarrow \text{Mem}[\text{PC} + 4 + 4\times \text{SEXT(literal)}] \)

Opcode Table: (*optional opcodes)

<table>
<thead>
<tr>
<th>5:3</th>
<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>
<td></td>
</tr>
<tr>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></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>
<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>CMPEQ</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>CMPEQC</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>0010</td>
<td>1111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x104</td>
<td>0xC03F0000</td>
<td>110000</td>
<td>0001</td>
<td>1111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x108</td>
<td>0xE060000F</td>
<td>111000</td>
<td>0011</td>
<td>0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x10C</td>
<td>0xF0210004</td>
<td>111100</td>
<td>0001</td>
<td>0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x110</td>
<td>0xA4230800</td>
<td>101001</td>
<td>0001</td>
<td>0011</td>
<td>0001</td>
<td>or (R3, R1, R1)</td>
</tr>
<tr>
<td>0x114</td>
<td>0xF4000004</td>
<td>111101</td>
<td>0000</td>
<td>0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x118</td>
<td>0xC4420001</td>
<td>110001</td>
<td>0010</td>
<td>0010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x11C</td>
<td>0x73E80002</td>
<td>011100</td>
<td>1111</td>
<td>0010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x120</td>
<td>0x73FFFFF9</td>
<td>011100</td>
<td>1111</td>
<td>1111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x124</td>
<td>0xA4230800</td>
<td>101001</td>
<td>0001</td>
<td>0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x128</td>
<td>0x605F0124</td>
<td>011000</td>
<td>0010</td>
<td>1111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12C</td>
<td>0x90211000</td>
<td>100100</td>
<td>0001</td>
<td>0001</td>
<td></td>
<td></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 Rd, e.g., 0x12345678 becomes 0x8754321. So the "passnumber" is the nibble reverse of 0xA4230800 which is 0x0C0324A.
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 \( R5: \) \( \text{0x3400 0000} \)
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: \( \boxed{10} \) at address: \( 9 + 0x1234 = 0x1238 \)

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

(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 \( 0x1000 \) 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?

Interpret 2nd operand as a constant expression. Value of symbol \( R2 \) is 2. Circle best answer: YES \( \boxed{\text{NO}} \) CAN’T TELL
Problem 1

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) R1 ← 3
SHLC(R1,2,R1) R1 ← R2
LD(R1,X,R2) R2 ← Mem[X+4]
HALT()

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

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

(B) . = 0
LD(R31,X,R0) Value left in R0: 0x 83063520
CMOVE(0,R1) Value left in R1: 0x 4
L: CMPLTC(R0,0,R2) Value left in R2: 0x 1
BNE(R2,DONE)
ADD(R1,1,R1)
SHLC(R0,1,R0)
BR(L)
DONE: HALT()

X: LONG(0x08306352)

Value assembler assigns to symbol X: 0x 20
4 counts # of left shifts needed until MSB of R0 is 1.

(C) . = 0
LD(R31,Z,R1) R1 ← binary for CMPLTC inst.
SHRC(R1,26,R1) R1 ← opcode field
Z: CMPLTC(R1,0x3c,R2) Value left in R1: 0x 35 (CMPLTC opcode)
HALT()

Value left in R2: 0x 1

(D) . = 0
LD(R31,X,R0) R8 ← 5
CMOVE(0,R1)
L: ADDC(R1,1,R1) Value left in R1: 0x 3
SHRC(R0,1,R0) Value left in R2: 0x 14
BNE(R0,L,R2)
HALT()

X: LONG(5)

Value assembler assigns to symbol X: 0x 100

6.004 Worksheet - 4 of 5 - LOG - Instruction Set Architecture
(E)  
\[ R0 \leftarrow 0 \times 87654321 \text{ (negative!)} \]
\[ \text{Value left in R0? } 0 \times \underline{87654321} \]
\[ \text{Value left in R1? } 0 \times \underline{C} \]
\[ \text{Value left in R2? } 0 \times \underline{E8765432} \]
\[ \text{Value assembler assigns to L1: } 0 \times \underline{14} \]

X: \text{ LONG}(0 \times 87654321)

(F)  
\[ R0 \leftarrow 3 \]
\[ \text{Contents of R0 (in hex): } 0 \times \underline{C} \]
\[ R0 \leftarrow 12 \]
\[ \text{Contents of R1 (in hex): } 0 \times \underline{C0FFEE} \]

LD(R31, i, RO)
SHLC(R0, 2, RO)
LD(R0, a-4, R1)
HALT()

a: \text{ LONG}(0 \times BADBABA)

+4 \text{ LONG}(0 \times DEADBEF)

+8 \text{ LONG}(0 \times COFFEE)

L: \text{ LONG}(0 \times 8BADFOOD)

i: \text{ LONG}(3)

(G)  
\[ R1 \leftarrow \text{ binary for SUBC} \]
\[ \text{Value left in R1: } 0 \times \underline{C46203BC} \]
\[ \text{Value left in R3: } 0 \times \underline{C426} \]
\[ \text{Value assembler assigns to symbol Z: } 0 \times \underline{3} \]

(H)  
\[ R0 \leftarrow \text{ DECAF} \]
\[ \text{Value left in R0: } 0 \times \underline{0} \]
\[ \text{Value left in R1: } 0 \times \underline{14} \]
\[ \text{Value left in R2: } 0 \times \underline{14} \]

X: \text{ LONG}(0 \times DECAF)

\[ \text{count # of right shifts to make } R0 \text{ equal to } 2 \times R0. \]