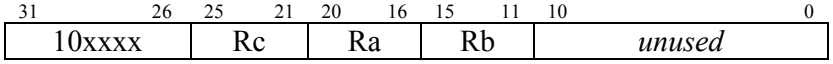
Computation Structures

Instruction Set Architecture Worksheet

Summary of β Instruction Formats

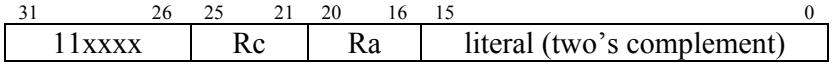
Operate Class:

Register	Symbol	Usage
R31	R31	Always zero
R30	XP	Exception pointer
R29	SP	Stack pointer
R28	LP	Linkage pointer
R27	BP	Base of frame pointer



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

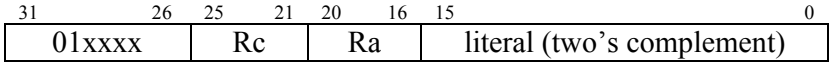
Opcodes: **ADD** (plus), **SUB** (minus), **MUL** (multiply), **DIV** (divided by)
AND (bitwise and), **OR** (bitwise or), **XOR** (bitwise exclusive or), **XNOR** (bitwise exclusive nor),
CMPEQ (equal), **CMPLT** (less than), **CMPLE** (less than or equal) [result = 1 if true, 0 if false]
SHL (left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



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:



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)

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

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	_____	ADDC(R31, 0x8, R2)
	0x104	0xC03F0000	110000	00001	11111	_____	ADDC(R31, 0x0, R1)
L2:	0x108	0xE060000F	111000	00011	00000	_____	ANDC(R0, 0xF, R3)
	0x10C	0xF0210004	111100	00001	00001	_____	SHLC(R1, 0x4, R1)
	0x110	0xA4230800	101001	00001	00011	00001	OR(R3, R1, R1)
	0x114	0xF4000004	111101	00000	00000	_____	SHRC(R0, 0x4, R0)
	0x118	0xC4420001	110001	00010	00010	_____	SUBC(R2, 0x1, R2)
	0x11C	0x73E20002	011100	11111	00010	_____	BEQ(R2, L1, R31)
		+2					
	0x120	0x73FFFFF9	011100	11111	11111	_____	BEQ(R31, L2, R31)
		-7					
	0x124	0xA4230800	101001	00001	00011	_____	not executed!
L1:	0x128	0x605F0124	011000	00010	11111	_____	LD(R31, 0x124, R2)
	0x12C	0x90211000	100100	00001	00001	_____	CMPEQ(R1, R2, R1)

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, eg., 0x12345678 becomes 0x87654321.

So the "passnumber" is the nibble reverse of 0xA4230800 which is 0x00830324A.

Problem 2.

(A) 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: 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. SHL(R3, R4, R5) **Value of R5:** 0x3400 0000
2. ADD(R2, R1, R6) **Value of R6:** 22
3. ADD(R0, 2, R7) **Value of R7:** 20
4. ST(R1, 4, R3) **Value stored:** 10 **at address:** 4 + 0x1234 = 0x1238

op = ADDC so interpreted as R2!

(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)

expression value computed at assembly time

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?

Original branch offset (0x1000) now relative to 0x1400

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

(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 ... 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)

```
        . = 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)
```

R1 ← 3
R1 ← 12
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 left in R0: 0x 83063520

Value left in R1: 0x 4

Value left in R2: 0x 1

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 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)
```

R0 ← 5

Value left in R0: 0x 0

Value left in R1: 0x 3

Value left in R2: 0x 14

Value assembler assigns to symbol X: 0x 100

↑ count # of right shifts until R0 == 0.

(E)

```
        . = 0
0       LD(r31, X, r0)
4       CMPLE(r0, r31, r1)
8       BNE(r1, L1, r1)
C       ADDC(r31, 17, r2)
10      BEQ(r31, L2, r31)
14  L1: SRAC(r0, 4, r2)
18  L2: HALT()

        . = 0x1CE8
    X:  LONG(0x87654321)
```

R0 ← 0x87654321 (negative!)

R1 ← 1

R1 ← 0xC, branch taken

R2 ← 0xF8765432

Value left in R0? 0x 87654321

Value left in R1? 0x C

Value left in R2? 0x F8765432

Value assembler assigns to L1: 0x 14

(F)

```
        . = 0
        LD(R31, i, R0)
        SHLC(R0, 2, R0)
        LD(R0, a-4, R1)
        HALT()
    a:  LONG(0xBADBABE)
+4      LONG(0xDEADBEEF)
+8      LONG(0xC0FFEE)
        LONG(0x8BADF00D)
    i:  LONG(3)
```

R0 ← 3

R0 ← 12

R1 ← Mem[12 + a - 4]

= i or 8

Contents of R0 (in hex): 0x C

Contents of R1 (in hex): 0x C0FFEE

(G)

```
        . = 0
0       LD(R31,Z,R1)
4       SHRC(R1,16,R2)
8   Z:  SUBC(R2,0x3C,R3)
C       HALT()
```

R1 ← binary for SUBC

R2 ← top half R1

SUBC 3 2

110001	00011	00010	0x3C
op	R2	R0	

Value left in R1: 0x C4620003C

Value left in R3: 0x C426

Value assembler assigns to symbol Z: 0x 8

(H)

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

8   L:  ADDC(R1,1,R1)
C       SHRC(R0,1,R0)
10      BNE(R0,L,R2)
14      HALT()

    X:  LONG(0xDECAF)
```

R0 ← DECAF

↑ count # of right shifts to make R0 equal to zero.

Value left in R0: 0x 0

Value left in R1: 0x 14

Value left in R2: 0x 14

6.004 Computation Structures
Spring 2017