# LE11.1.1 Expressions

0 points possible (ungraded)

Hand-compile the following C fragments into Beta assembly language. You can also assume that all variables and arrays are C integers, i.e., 32-bit values, and that the necessary storage allocation for each variable or array has been done and that a UASM label has been defined that indicates the first storage location for that variable or array.

There's no automated checking for this problem. Just write your answer out on a piece of paper and then compare it with the solutions to see how you did!

(A) `x = 3;`

Explanation
Using templates:

```
CMOVE(3,r0)
ST(r0,x)
```

(B) `d = b + 3*c;` [Note: in C, multiplication has a higher precedence than addition, so C treats this expression as "b+(3*c)".]

Explanation
Using templates (optimizations possible):

```
LD(b,r0)
CMOVE(3,r1)
LD(c,r2)
MUL(r1,r2,r1)
ADD(r0,r1,r0)
ST(r0,d)
```

(C) `d = (b*3 + 1)/(c – b);`

Explanation
Using templates (optimizations possible):

```
LD(b,r0)        // b
CMOVE(3,r1)
MUL(r0,r1,r0)   // b*3
CMOVE(1,r1)
ADD(r0,r1,r0)   // b*3 + 1
LD(c,r1)        // c
LD(b,r2)        // b
SUB(r1,r2,r1)   // c - b
DIV(r0,r1,r0)   // (b*3 + 1)/(c - b)
ST(r0,d)
```

(D) `a[1] = a[0] + 1;` [Note: in C, the first element of an array has index 0. Remember that each element of the "a" array occupies 4 bytes (i.e., bsize = 4).]

Explanation
Using templates (optimizations possible):

```
CMOVE(0,r0)
MULC(r0,4,r0)
LD(r0,a,r0)    // load a[0]
CMOVE(1,r1)
ADD(r0,r1,r0)
CMOVE(1,r1)
MULC(r1,4,r1)
ST(r0,a,r1)    // store to a[1]
```

(E) `a[j-1] = a[j] + 1;`

Explanation
Using templates (optimizations possible):

```
LD(j,r0)
MULC(r0,4,r0)  // convert index to byte offset
LD(r0,a,r0)
CMOVE(1,r1)
ADD(r0,r1,r0)
LD(j,r1)
CMOVE(1,r2)
SUB(r1,r2,r1)
MULC(r1,4,r1)
ST(r0,a,r1)
```

Submit

Answers are displayed within the problem

# LE11.1.2 Array access

0/1 point (ungraded)

What C statement might have been compiled into the code fragment below?

```
I = 0x5678
B = 0x1234

LD(I,R0)
SHLC(R0,2,R0)
LD(R0,B,R1)
MULC(R1,17,R1)
ST(R1,B,R0)
```

- ✔ $B[I] = B[I] * 17$
- $B[I] = B[I * 17]$
- ✖ $B[I] = B[4 * I] * 17$
- $B[I] = B[4 * I * 17]$

Explanation

The LD(I,R0) loads the value of I into R0. I is the array index so it needs to be multiplied by 4 in order to produce the correct offset from the beginning of the array because each element is made up of 4 bytes. The SHLC(R0,2,R0) sets R0 = 4*I. The LD(R0,B,R1) takes the contents of MEM[R0 + B] = array element I and loads it into R1. This loaded value is then multiplied by 17 and tha result is stored back into R1. So R1 now equals B[I] * 17. This new value of R1 is the stored into the location whose address is B + R0, or in other words the memory location of array element I, or B[I].

Submit

Answers are displayed within the problem

# LE11.1.3 Array access

0.0/1.0 point (ungraded)

For each of the assembly language sequences below, click the associated box if it might have resulted from compiling the following C statement.

```
int x[20];
int y;
y = x[1] + 4;
```

☐

```
A: LD(R31,x+1,R0)
   ADDC(R0,4,R0)
   ST(R0,y,R31)
```

☐

```
B: CMOVE(4,R0)
   ADDC(R0,x+4,R0)
   ST(R0,y,R31)
```

☐

```
C: LD(R31,x+4,R0)
   ST(R0,y+4,R31)
```

☐

```
D: CMOVE(4,R0)
   LD(R0,x,R1)
   ST(R1,y,R0)
```

☐

```
E: LD(R31,x+4,R0)
   ADDC(R0,4,R0)
   ST(R0,y,R31)
```

```
F: ADDC(R31,x+1,R0)
   ADDC(R0,4,R0)
   ST(R0,y,R31)
```

Submit

## Discussion

**Topic:** 11. Compilers / LE11.1

**Hide Discussion**

**Add a Post**

Show all posts by recent activity

| | | |
|---|---|---|
| Another compilation rule | I think the following might be true and useful. Although I think it might spoil the fun of figuring it ou... | 2 |
| Misprint in question 11.1.3 ? | Umm...is it just me or is there a slight misprint in the correct solution for question 11.1.3 ? Shouldn't ... | 3 |
| [STAFF] LE11.1.2 ARRAY ACCESS | "C" is case-sensitive. | 2 |
| 11.1.2 provides incorrect answer? | The explanation says that > The SHLC(R0,2,R0) sets R0 = 4*I. then > The LD(R0,B,R1) takes the co... | 5 |