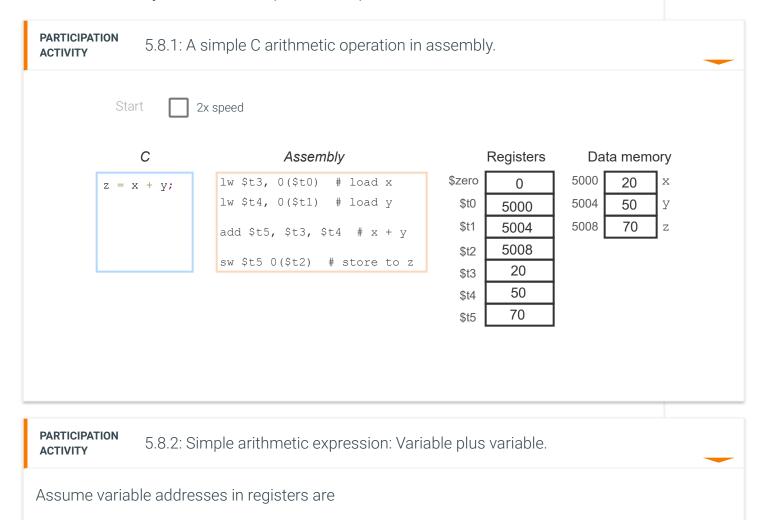
1/3/2019 5.8. Expressions

5.8 Expressions

Simple arithmetic expressions

In C, a statement may assign a variable with the result of a simple arithmetic expression, such as z = x + y. In assembly, x a into registers, an arithmetic assembly instruction computes the expression's result, and the result is then stored into z.



- x: \$t0
- y: \$t1
- z: \$t2

Indicate the assembly instructions to carry out: x = y + z. Each question below represents one instruction in a sequence.

- 1) Get y
 - O lw \$t3, 0(\$t0)
 - O lw \$t3, 0(\$t1)
 - O lw \$t3, 0(\$t2)
- 2) Get z
 - O lw \$t4, 0(\$t0)
 - O lw \$t4, 0(\$t1)
 - O lw \$t4, 0(\$t2)
- 3) Add y + z
 - O add \$t3, \$t3, \$t3
 - O add \$t3, \$t1, \$t2
 - O add \$t3, \$t3, \$t4
- 4) Assign x with y + z
 - O sw \$t0, 0(\$t3)
 - O sw \$t3, 0(\$t0)

PARTICIPATION ACTIVITY

5.8.3: Simple arithmetic expression: Variable plus literal.

Assume variable addresses in registers are

- x: \$t0
- y: \$t1
- 1) Which instructions implement x = y + 5?
 - O lw \$t2, 0(\$t1) lw \$t3, 0(\$t5) add \$t2, \$t2, \$t3

sw \$t2, 0(\$t0)

- O lw \$t2, 0(\$t1) addi \$t2, \$t2, 5 sw \$t2, 0(\$t0)
- O lw \$t2, 0(\$t1) add \$t2, \$t2, 5 sw \$t2, 0(\$t0)
- 2) Which instructions implement x = x * y?
 - O lw \$t2, 0(\$t0) lw \$t3, 0(\$t1) mul \$t2, \$t2, \$t3 sw \$t2, 0(\$t0)
 - O lw \$t2, 0(\$t0) lw \$t3, 0(\$t1) add \$t2, \$t2, \$t3 sw \$t2, 0(\$t0)
 - O Not possible.

Sequences of arithmetic operations

Sometimes C statements write a variable several times. Intermediate results need not be stored into memory and may inst written to a register.

1/3/2019 5.8. Expressions

Figure 5.8.1: Intermediate writes to a variable need not be stored into memory.

Assume x's value is in \$t3, y's value is in \$t4, and z's address is in \$t2.

C statements	Inefficient assembly		More efficient assembly	
z = x + y; z = z + 1;	add \$t5, \$t3, \$t4 sw \$t5, 0(\$t2) z lw \$t5, 0(\$t2) addi \$t5, \$t5, 1 sw \$t5, 0(\$t2) z		add \$t5, \$t3, \$t4 \$t5 = x + y addi \$t5, \$t5, 1 \$t5 = \$t5 + 1 sw \$t5, 0(\$t2) Store into z	# # #

In the assembly above, the intermediate result of x + y need not be stored into z, since that result in z would just be overwrift of the next instruction (addi) that adds 1 and stores the new result into z.

5.8.4: Intermediate results.

Given the following C that computes z = x + x + y + 1;

```
z = x + x;

z = z + y;

z = z + 1;
```

PARTICIPATION

ACTIVITY

Which assembly instructions should be deleted from the following for efficiency?

```
1: add $t5, $t3, $t3 # $t5 = x + x
2: sw $t5, 0($t2) # z = $t5
3: lw $t5, 0($t2) # Load z
4: add $t5, $t5, $t4 # $t5 = z + y
5: sw $t5, 0($t2) # z = $t5
6: lw $t5, 0($t2) # Load z
7: addi $t5, $t5, 1 # $t5 = z + 1
8: sw $t5, 0($t2) # z = $t5
```

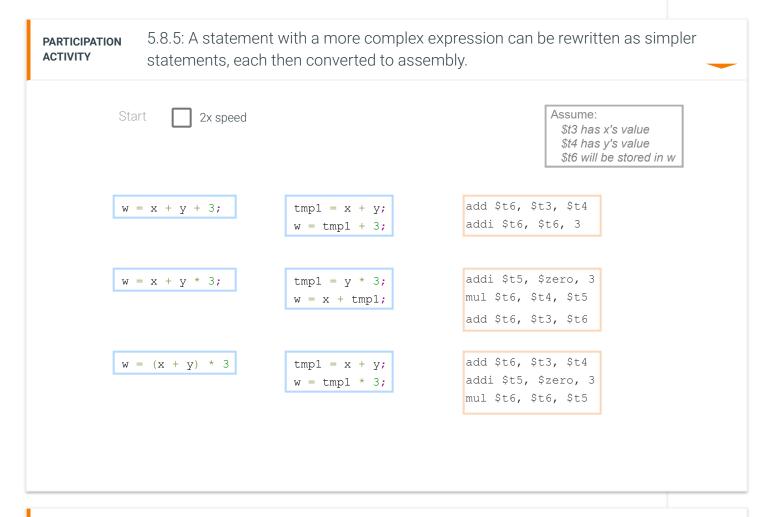
1/3/2019

1: add \$t5, \$t3, \$t3 # \$t5 = x + xO Keep O Delete 2) 2: sw \$t5, 0(\$t2) # z = \$t5 O Keep O Delete 3) 3: lw \$t5, 0(\$t2) # Load z O Keep O Delete 4) 4: add \$t5, \$t5, \$t4 # \$t5 = z + y O Keep O Delete 5) 5: sw \$t5, 0(\$t2) 6: lw \$t5, 0(\$t2) # z = \$t5# Load z O Keep O Delete 6) 7: addi \$t5, \$t5, 1 # \$t5 = z + 1 O Keep O Delete 7) 8: sw \$t5, 0(\$t2) # z = \$t5 O Keep O Delete

More complex expressions

A C statement with a simple expression having one operator, like w = x + y, can be converted to an assembly instruction, add \$t5, \$t3, \$t4. To convert a statement having a more complex expression, like w = x + y + 3, the statement may rewritten as several simpler statements, like tmp1 = x + y followed by w = tmp1 + 3. (tmp1 is a temporary variable use a rewrite). Each statement can then be converted to assembly.

Precedence rules should be obeyed, such as the * operator having higher precedence than +, and expressions within parentipher precedence. For equal precedence operators, C specifies left-to-right evaluation. Ex: For x + y + 3, expression x + y sh computed first.



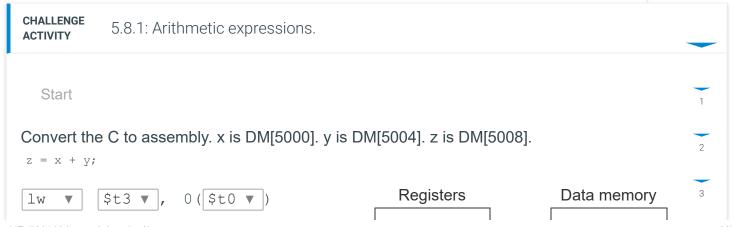
ACTIVITY

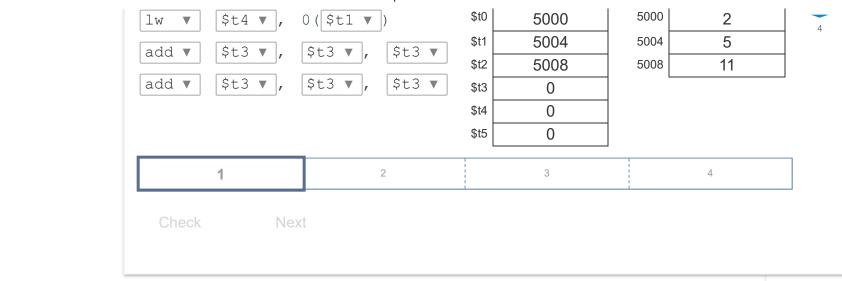
5.8.6: Rewriting a statement into statements with one-operator expressions.

Select the statements that are a correct rewrite involving one-operator expressions.

- 1) w = x + y + z;
 - tmp1 = x + y; w = x + z;
 - tmp1 = x + y; w = tmp1 + z;
- 2) w = x + y z;
 - tmp1 = y z; w = x + tmp1;
 - tmp1 = x + y; w = tmp1 - z;
- 3) w = x + y * z;
 - tmp1 = x + y; w = tmp1 * z;
 - tmp1 = y * z; w = x + tmp1;
- 4) w = w + x + 1;
 - tmp1 = w + x; w = tmp1 + 1;
 - O Not possible; w can't appear on the right.
- 5) u = w + x + y + z;
 - tmp1 = w + x; tmp2 = tmp1 + y; u = tmp2 + z;
 - 0

```
tmp1 = w + x + y;
           u = tmp1 + z;
6) w = x * (y + z);
          u = x * tmp1;
           tmp1 = y + z;
           tmp1 = y + z;
           w = x * tmp1;
7) u = (w + x) * (y + z);
           tmp1 = w + x;
           tmp2 = tmp1 * y;
           u = tmp2 + z;
          tmp1 = w + x;
           tmp2 = y + z;
           u = tmp1 * tmp2;
8) u = w + (x * (y + z));
           tmp1 = w + x;
           tmp2 = tmp1 * y;
           u = tmp2 + z;
           tmp1 = y + z;
           tmp2 = x * tmp1;
           u = w + tmp2;
```





Provide feedback on this section