COMP 430
Spring 2020

Compiling in a Stack-Oriented Fashion

For this worksheet, we'll compile expressions and statements to a MIPS-like assembly language. This assembly language has the following registers:

- $gp0, $gp1, $gp2: general purpose registers
- $sp: stack pointer: holds memory address of the top of the stack

In addition, this assembly language has the following instructions:

- li register$_{\text{dest}}$, value: load immediate: puts the given value into the given register (e.g., li $gp0, 5 puts 5 in $gp0)
- push register$_{\text{input}}$: puts the 4-byte value in the given input register on top of the stack. Also adds 4 to the value in $sp
- pop register$_{\text{dest}}$: puts the 4-byte value on top of the stack into the given destination register. Also subtracts 4 from the value in $sp.
- add register$_{\text{dest}}$, register$_{\text{input1}}$, register$_{\text{input2}}$: adds the values in the two input registers, putting the result in the destination register
- mult register$_{\text{dest}}$, register$_{\text{input1}}$, register$_{\text{input2}}$: multiplies the values in the two input registers, putting the result in the destination register
- load register$_{\text{dest}}$, regsiter$_{\text{input}}$, offset: loads a value from memory into register$_{\text{dest}}$. The address from which to load is specified in register$_{\text{input}}$. offset is an offset from this address. For example, load $gp0, $sp, -4 will load the value on top of the stack into $gp0, without changing the value in $sp.
- store register$_{\text{input1}}$, register$_{\text{input2}}$, offset: stores the value in register$_{\text{input1}}$ into memory. The address to store at is specified in register$_{\text{input2}}$. offset is an offset from this address. For example, store $gp0, $sp, -4 will overwrite the value on top of the stack with the value in $gp0, without changing the value in $sp.

With the above instructions in mind, translate the following expressions into assembly. The result of any expression, **including subexpressions**, should end up on top of the stack. The first one has been done for you.

1.) `1 + 2`

```
li $gp0, 1    ; 1
push $gp0     ; 1
li $gp0, 2    ; 2
push $gp0     ; 2
pop $gp0      ; 1 + 2
pop $gp1      ; 1 + 2
add $gp2, $gp1, $gp0 ; 1 + 2
push $gp2     ; 1 + 2
```

2.) `123`

3.) `(1 + 2) * 3`

Now translate each of the following statements to this language. Variables should get pushed on the stack, but never popped off. You can assume that int is 4 bytes large. The first one has been done for you.

4.)

```
int x = 5;
x = 6;

li $gp0, 5   ; 5
push $gp0    ; int x = 5;
li $gp0, 6   ; 6
```

```
push $gp0   ; 6
pop $gp0    ; x = 6
store $gp0, $sp, -4 ; x = 6
```

5.)

```
int x = 0;
int y = x + 1;
```

6.)

```
int x = 2;
int y = 4;
int z = x + y;
```