Loops

Loops

As we have seen, assembly does not have all the nice features of high-level code, but we can learn to build those features ourselves

In addition to conditional statements, loops are one of the fundamental building blocks we learn to implement

First examine `while` loops

`for` loops can be made from `while` loops

Loops

We know what a loop is, but it can help to step back and think about how we would describe a loop. Here's one description:

Repeatedly execute a block of code until a condition is not met

This tells us we need to

- Have some way of going back to start of code block
- Have some way of skipping code block once we are done

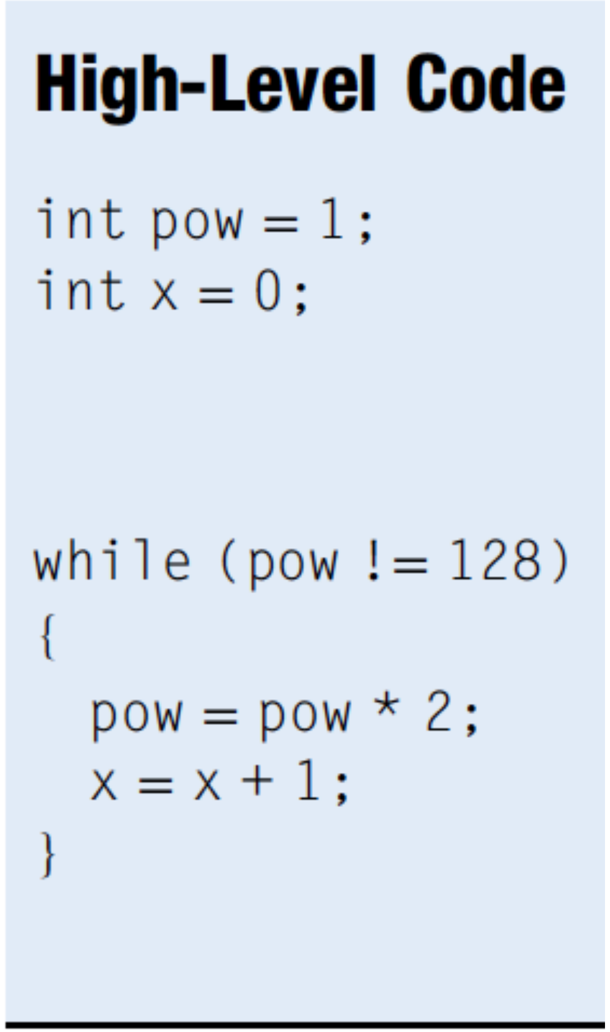
Loops

This tells us we need to

- Have some way of going back to start of code block
- Have some way of skipping code block once we are done

We already know how to move around in code: branches and jumps

Branches also let us test conditions, which will be necessary to decide when to leave the loop



High-Level Code

```
int pow = 1;
int x = 0;


while (pow != 128)
{
  pow = pow * 2;
  x = x + 1;
}
```

Don’t let the multiplication throw you off: remember that multiplication by powers of two can be implemented with shifts

High-Level Code

```
int pow = 1;
int x = 0;



while (pow != 128)
{
  pow = pow * 2;
  x = x + 1;
}
```

MIPS Assembly Code

```
# $s0 = pow, $s1 = x
  addi $s0, $0, 1        # pow = 1
  addi $s1, $0, 0        # x = 0

  addi $t0, $0, 128      # t0 = 128 for comparison
while:
  beq  $s0, $t0, done    # if pow == 128, exit while loop
  sll  $s0, $s0, 1       # pow = pow * 2
  addi $s1, $s1, 1       # x = x + 1
  j    while
done:
```

`while` loop

You can think of a loop as a conditional block that we execute many times

Just as with `if` statements, always test *opposite* condition for loop

Only difference between `while` and `if` is jump at the bottom that allows us to repeat ourselves

Once again – memorize that structure. Once it is second nature to you, you can focus on more important concerns in your code

`for` loop

Consider example `for` loop:

```
for (i = 0; i < 10; i++)
     ...
```

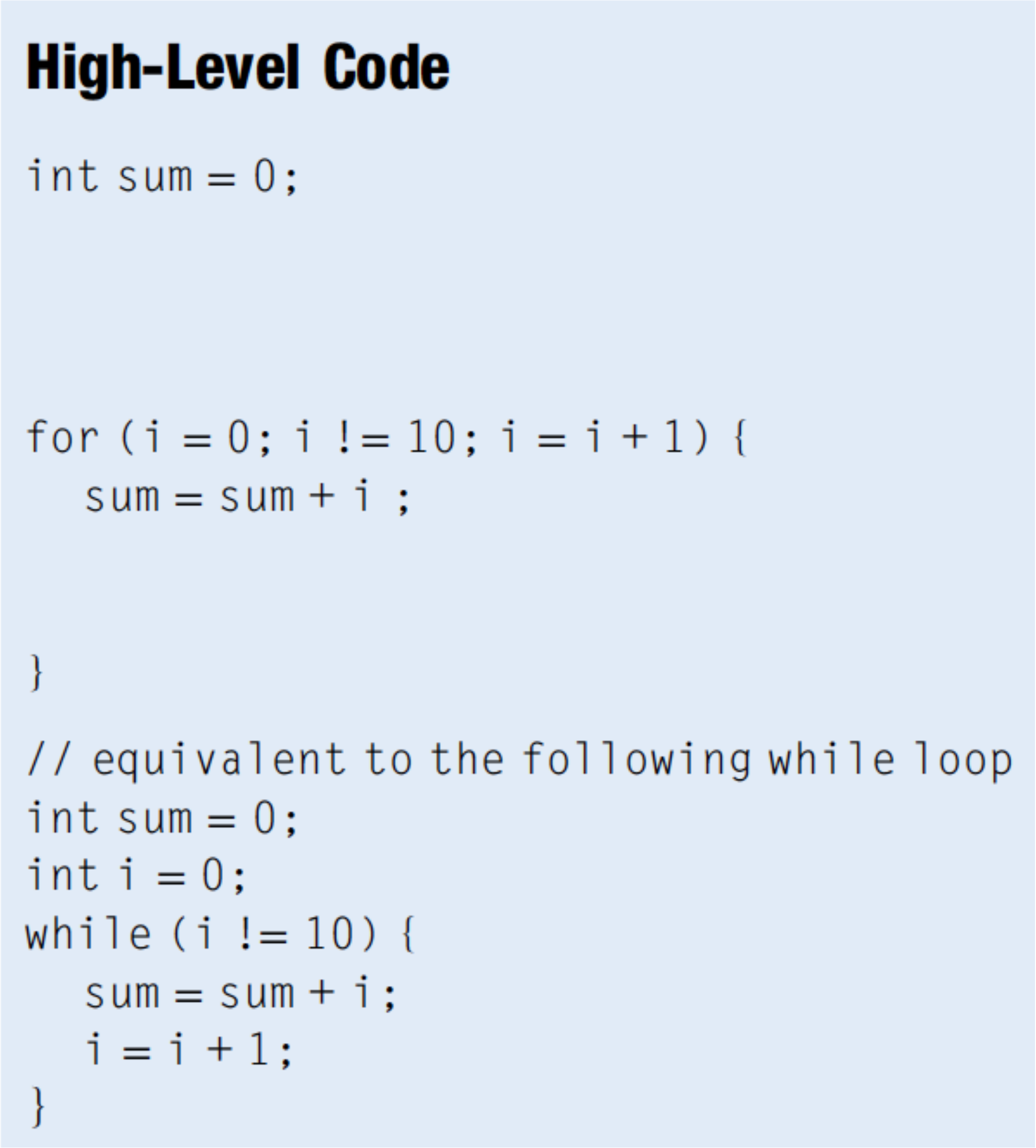
Actually just a `while` loop with some syntactic sugar

`for` loop

```
for (i = 0; i < 10; i++)
     ...
```

To turn `for` loop into `while` loop:

- Set `i` outside loop (this is done just once)
- Increment `i` at end of loop (we do this ourselves in while loops anyway)
- Condition works same way as always



High-Level Code

```
int sum = 0;




for (i = 0; i != 10; i = i + 1) {
  sum = sum + i ;



}

// equivalent to the following while loop
int sum = 0;
int i = 0;
while (i != 10) {
  sum = sum + i;
  i = i + 1;
}
```

High-Level Code

```
int sum = 0;




for (i = 0; i != 10; i = i + 1) {
   sum = sum + i ;



}

// equivalent to the following while loop
int sum = 0;
int i = 0;
while (i != 10) {
   sum = sum + i;
   i = i + 1;
}
```

MIPS Assembly Code

```
# $s0 = i, $s1 = sum
  add   $s1, $0, $0       # sum = 0
  addi  $s0, $0, 0        # i = 0
  addi  $t0, $0, 10       # $t0 = 10

for:
  beq   $s0, $t0, done    # if i == 10, branch to done
  add   $s1, $s1, $s0     # sum = sum + i
  addi  $s0, $s0, 1       # increment i
  j     for

done:
```

Loop Boilerplate

- Generally have
  - Setup of loop variable (register) above loop
  - Loop label
  - Branch statement to test loop condition
  - Loop body
  - Updating of loop variable (i++)
  - Jump to label

Still testing the opposite case for the loop condition!

Magnitude Comparison

Often use loop conditions based on inequality

- Use slt, just as we did with conditionals

It goes without saying that we still test opposite case

C Code

```c
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
  sum = sum + i;
}
```

C Code

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
       addi $s1, $0, 0
       addi $s0, $0, 1
       addi $t0, $0, 101
loop:  slt  $t1, $s0, $t0
       beq  $t1, $0, done
       add  $s1, $s1, $s0
       sll  $s0, $s0, 1
       j    loop
done:
```