

# ← Lab 2: Variables, arrays, and conditionals

Finish by midnight on Wednesday, 9/19

In this lab, you'll be practicing the things we learned about this week: variables, arrays, and simple conditional flow control.

Make a new file in MARS and name it with your username like `abc123_lab2.asm`. Then, use this tiny code skeleton to get started:

```
.data
# put variables here.

.text # don't forget this!
.globl main
main:
    # put code here.
```

Make sure "Settings > Initialize Program Counter to global 'main'" is checked!

also it's .globl main, not .global main

## 1. Making variables

In the data segment, make three variables:

- a *byte* variable named `small`, initialized to 200
- a *half* variable named `medium`, initialized to 400
- a *word* variable named `large`, initialized to 0

Follow the pattern I showed in the slides.

**Never write a program all at once and *then* compile/assemble it.** Write a little at a time and let the assembler/compiler help you along the way. Then test it! Make sure

## 2. Accessing those variables

Now, in `main`, write some code to do the following:

1. load the value of `small` into a register
  - you don't need to keep this value around long, so a `t` register is the right choice here.
2. load the value of `medium` into another register
3. multiply those values together
4. **store** the product into `large`
  - you should be able to see its value in the "Data Segment" window after you run your program.
5. **print** the product out as well
  - it *should* be 80000, but see below if it's not.

### Hints

- **Use the right kind of load/store for each step.**
  - The assembler will happily let you use `lw` on a byte, `sh` on a word etc.
  - It's *your* responsibility to use the right loads and stores.
- **The `move a, b` instruction lets you *copy* the value from `b` into `a`.**
  - Sometimes you have a value in the "wrong" register and need to get it somewhere else.
  - Like when you have to use the print syscall and it wants the value to be in `a0` !

### How does $200 * 400 = -22400$ ???

Did this happen to you? Hmmm. Why could that be happening...

Try unchecking the 'hexadecimal values' option in the Data Segment window:

Data Segment				
	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0		0	0	0
0		0	0	0
0		0	0	0
0		0	0	0
0		0	0	0
0		0	0	0
0		0	0	0
0		0	0	0
0		0	0	0
0		0	0	0

☒ Hexadecimal Addresses
 ☐ Hexadecimal Values
 ☐ ASCII

**Then, have a look at the registers on the right.**

Do you notice anything weird about the values you loaded from `small` and `medium`?

Remember *sign versus zero extension*? And the *two versions of the loads*? Well, you gotta use the right one ;)

### 3. Arrays and loops

**Go back to the data segment** and add the following:

```
.eqv NUM_ITEMS 5
values: .word 0:NUM_ITEMS
```

`.eqv` makes a **named constant**. Naming constants is good practice.

The `values` array is an array of words, and it is `NUM_ITEMS` items long, and filled with 0s.

## Making a `for` loop

A **for** loop is really a **while** loop. If you write:

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

this is the same as:

```
int i = 0;
while(i < 10) {
    ...
    i++;
}
```

Okay? **Keep this pattern in mind.**

In `main` after the code you already wrote, start with this code:

```
ask_loop_top: # while(...)

ask_loop_body: # {


    j ask_loop_top
ask_loop_exit: # }
```

This is the basic form of a loop: a label for the top (where the condition is checked), a label for the body (the code inside), a `j` to go back to the top, and a label for the code *after* the loop.

Now, you need to turn this into the asm equivalent of `for(i = 0; i < NUM_ITEMS; i++)`.

## Hints

- Use `s0` as the register to represent `i`.
  - We'll learn about the `s` registers soon. Just trust me ;)
- Look at that pattern above. **Where should you set `i` to 0?**
  - Is that code gonna be *inside* the loop?
    - (no)
- **How do you increment `i`?**
  - *where* should you increment it?
- **Look at the slides where I talk about a simple conditional loop.**
  - We didn't get to it in class on Wednesday, but we will on Monday.
- **You can use `NUM_ITEMS` as the number to compare `s0` to.**
  - NAMED CONSTANTS GOOD.

Done correctly, this loop should run five times and then stop. You can use the step button  to step through it one instruction at a time to make sure this happens.

## Having problems?

If it never loops, or if it never *stops* looping, have a look at the contents of `s0` as you step through it and see what is happening.

**Being able to debug a program is a super important skill.** Try to figure it out instead of asking for help as soon as it doesn't work correctly. (But also, it's okay if you have no idea and ask for help instead of turning in a broken program. :D)

## 4. Making the loop do something

**Make sure the loop loops five times properly before trying this part. Seriously.**

Inside the loop (after `ask_loop_body:`), write some code to:

1. input a number from the user
  - syscall 5 is the one you want for this.
2. **store that number (which is in `v0` after the syscall)** into the correct item in the array.
  - you will have to calculate the array address like we learned in class.
  - go to the examples page and have a look at `arrays.asm` for an example array address calculation.

### Hints

- The way syscall 5 works is like this:

```
li      v0, 5
syscall # asks the user for a number
# now, whatever the user typed in is in v0 instead of
```

- **When syscall 5 runs, you'll be typing in the window at the bottom.**
  - Hit enter after you type the number.
  - You can try "Settings > Popup dialog for input syscalls" if you don't like that.
- **There are three parts of the array address calculation:**
  - `1a`
  - which register represents `i`?
  - it's an array of *words*.

If you did it correctly, after you run your program and type in 5 numbers, you should see those 5 numbers in the “Data Segment” window where your array is. I typed in 1, 2, 3, 4, 5 and got this:

Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0x00000001	0x00000002	0x00000003	0x00000004	0x00000005

# Submitting

Make sure your file is named `username_lab2.asm`, like `jfb42_lab2.asm`.

[Submit here.](#)

Drag your asm file into your browser to upload. **If you can see your file, you uploaded it correctly!**

You can also re-upload if you made a mistake and need to fix it.

© 2016-2018 Jarrett Billingsley