

CS 211 Homework 5

Assembly interpreter

Fall 2020

Introduction

In this assignment, you're asked to **implement an interpreter for a 16-bit CPU's assembly code**. The CPU has registers but no access to main memory.

You can assume:

- all input is correctly formatted
- all arguments are the correct type
 - we won't try to jump to a non-existent location
 - we won't use a constant when a register is expected
 - we won't use constants larger than registers can store
- no program will be more than 100 lines long
- programs will be written in all lowercase

Registers

The machine has the following registers, all 16-bits wide, for storing signed integers (using two's complement):

Register	Purpose
ax	General purpose
bx	General purpose
cx	General purpose
dx	General purpose

This CPU does not support referring to 8-bit subsets of 16-bit registers.

Instructions

Arguments to instructions are separated by a single space.

Moving data

Instruction	Args	Description
<code>mov</code>	<code>x y</code>	Copy x to y. This leaves x unchanged.

Note: y must be a register.

Arithmetic

Instruction	Args	Description
<code>add</code>	<code>x y</code>	Add x and y, place result in y
<code>sub</code>	<code>x y</code>	Subtract x from y, place result in y
<code>mul</code>	<code>x y</code>	Multiply x and y, place result in y
<code>div</code>	<code>x y</code>	Divide x by y, place result in y (discards remainder)

Note: y must be a register.

Jumps

Jumps don't use condition codes. Rather, they (may) take some arguments to compare. Also, instead of explicit labels, we use the line of the program (starting from 0) as a location to jump to.

Instruction	Args	Description
<code>jmp</code>	<code>L</code>	Jump to L
<code>je</code>	<code>L x y</code>	Jump to L if $x = y$
<code>jne</code>	<code>L x y</code>	Jump to L if $x \neq y$
<code>jg</code>	<code>L x y</code>	Jump to L if $x > y$
<code>jge</code>	<code>L x y</code>	Jump to L if $x \geq y$
<code>jl</code>	<code>L x y</code>	Jump to L if $x < y$
<code>jle</code>	<code>L x y</code>	Jump to L if $x \leq y$

Note: L must be an integer.

I/O

Instruction	Args	Description
<code>read</code>	<code>x</code>	Reads a 16-bit signed integer from stdin, stores it in <code>x</code>
<code>print</code>	<code>x</code>	Prints <code>x</code> on stdout

Note: for `read`, `x` must be a register.

Argument formats

Unless noted above, an argument to an instruction can either be a register (`ax`, `bx`, etc.) or a constant.

```
mov ax bx    ; copy value of ax to bx
mov 42 cx    ; put 42 in cx
add 1 ax     ; increment ax by 1
sub cx bx    ; decrement bx by cx
mul -1 cx    ; multiply cx by -1
```

Code format

A program is a number of lines of code, each of which is either blank or has one instruction.

A blank line is treated as a noop (“no op”, an instruction that does nothing).

Interpreter execution

The interpreter takes a single command-line argument, the assembly file to execute. Then any `read` instructions read from stdin, and `print` instructions write to stdout.

```
$ ./interpret foo.asm
```

Example programs

Read n , print $2 \times n$

```
read ax
mul 2 ax
print ax
```

Sample execution of interpreter:

```
$ ./interpret ex1.asm
10  <-- your input
20  <-- interpreter output
```

Read n , print “0” n times

```
read ax
jle 7 ax 0

print 0          <-- line 3
sub 1 ax
jle 7 ax 0
jmp 3
                  <-- line 7
```

Sample execution of interpreter:

```
$ ./interpret ex2.asm
3    <-- your input
000  <-- interpreter output
```

Submission

As usual, create a tar file with your Makefile and all `.c` and `.h` files. Your tar file should have these contents:

```
hw5
├─ Makefile
└─ interpret.c
```

You can create additional `.c` and `.h` files as needed, but the main file should be `interpret.c`. If you create a separate file, say `foo.c`, you should add `foo.o` to the `OBJECTS` variable in the `Makefile`.