How to write an ASM program
Write some functions.

- A program is a bunch of functions.
- Execution starts at “main”.
How to write an ASM function
General Steps

- Signature
- Pseudocode
- Variable mappings
- Skeleton
- Write the body
Signature & Pseudocode

- Individual lines of assembly code are too fiddly to be easy to think in.
- It’s much easier to write code with a plan.
- The design recipe, in any language, suggests signature first.
  - What are our arguments?
  - What are we going to return?
- Pseudocode is useful here to figure out *how* we’re going to compute our function. Then we can translate / expand that into ASM.
  - C makes good pseudocode here, but whatever else is fine too.
Variable Mappings

- Pseudocode should allow us to predetermine which registers we need.
  - Registers for arguments to be copied into.
  - Registers for local variables.
  - Registers for intermediate values.
- This is worth figuring out and writing down.
Function skeleton.

label:

- Prologue - Set stuff up
- Body - # TODO
- Epilogue - Clean up, mostly reversing the prologue, and return.

We can use this same basic pattern for every function.
Prologue

● Allocate a stack frame.
  ○ 1 word for $ra, 1 word per $tX or $sX register, any extra you need
  ○ Decrease $sp by at least 4 for each word.

● Save $ra
  ○ At 0($sp)

● Save any registers you plan to use in this function.
  ○ At 4($sp), 8($sp), 12($sp), etc.

● Copy arguments in $aX regs to $tX regs.

Stack frame?

● An array of words, starting at $sp, that we happen to index in 4’s.
Epilogue

- Copy return value from $tX register to $v0.
- Reverse the work done in the prologue.
- Restore (load) $ra and any saved registers.
  - They’re right where you left them.
- Move the stack pointer back. If you decreased it by 100 in the prologue, increase it by 100 here.
- jr $ra
Function body.

- You should be able to simply translate the pseudocode.
- There are patterns for all C constructs.
  - E.g. Assignment is move, arithmetic is various ops, if is a cond branch, etc.
An Example Program

```c
int main() {
    int x = read_int();
    int y = read_int();
    int z = foo(x, y);
    print_int(z);
    return 0;
}

int foo(int a, int b) {
    return 2 * a + b + 3;
}

int read_int() {
    print("Enter a number:
    return read_int_syscall();
}

void print_int(int k) {
    printf("Your number is: %d\n", k);
}
```
TODO

- That’s 4 functions.
- So we need to build 4 functions.
main

- Signature: whatever -> int
- Pseudocode: check
- Variable mappings:
  - $x = t0$
  - $y = t1$
  - $z = t2$
- Skeleton

```c
int main() {
    int x = read_int();
    int y = read_int();
    int z = foo(x, y);
    print_int(z);
    return 0;
}
```

```asm
main:
    $sp -= 16;
    Push $ra, $t0, $t1, $t2
    ...
    Pop $t2, $t1, $t0, $ra
    $sp += 16
```
Push? pop?

Push $ra, $t0, $t1, $t2

Is shorthand for

sw $ra, 0($sp)
sw $t0, 4($sp)
sw $t1, 8($sp)
sw $t2, 12($sp)

Pop $t2, $t1, $t0, $ra

Is shorthand for

lw $t2, 12($sp)
lw $t1, 8($sp)
lw $t0, 4($sp)
lw $ra, 0($sp)
main

main:
   $sp -= 16;
   Push $ra, $t0, $t1, $t2

   jal read_int; move $t0, $v0
   jal read_int; move $t1, $v0

   move $a0, $t0; move $a1, $t1
   jal foo; move $t2, $v0

   Pop $t2, $t1, $t0, $ra
   $sp += 16

int main() {
   int x = read_int();
   int y = read_int();
   int z = foo(x, y);
   print_int(z);
   return 0;
}

● Variable mappings:
   ○ x = $t0
   ○ y = $t1
   ○ z = $t2
foo

- Signature: int, int -> int
- Pseudocode: check
- Skeleton

```assembly
foo:
    $sp -= 20;
    Push $ra, $t0, $t1, $t2, $t3, $t4
    move $t0, $a0; move $t1, $a1
    ...
    move $v0, $t4
    Pop $t4, $t3, $t2, $t1, $t0, $ra
    $sp += 20
```

```c
int foo(int a, int b) {
    return 2 * a + b + 3;
}
```

- Variable mappings:
  - a = $t0
  - b = $t1
  - 2*a = $t2
  - 2*a+b = $t3
  - 2*a+b+3 = $t4
int foo(int a, int b) {
    return 2 * a + b + 3;
}

Variable mappings:
- $a = $t0
- $b = $t1
- $2*a = $t2
- $2*a+b = $t3
- $2*a+b+3 = $t4
read_int

- Signature: nothing -> int
- Pseudocode: check
- Skeleton

```
read_int:
    $sp -= 8
    Push $ra, $t0
    ...
    move $v0, $t1
    Pop $t0, $ra
    $sp += 8
```

```
int read_int() {
    print(“Enter a number:\n”);
    return read_int_syscall();
}
```

Variable mappings:

- Integer read = $t0
- Address of the string: ??
read_int

.data
ri_msg: .asciiz "Enter a number:\n")
.text
read_int:
    $sp -= 8;
    Push $ra, $t0

    li $v0, 4; la $a0, ri_msg
    syscall

    li $v0, 5; syscall
    move $t0, $v0

    move $v0, $t1
    Pop $t0, $ra
    $sp += 8

int read_int() {
    print("Enter a number:\n");
    return read_int_syscall();
}

Variable mappings:
- Integer read = $t0
- Address of the string = straight to $a0
print_int

- Signature: int -> nothing
- Pseudocode: check
- Skeleton

```c
void print_int(int k) {
    printf("Your number is: %d\n", k);
}

Variable mappings:
- k = $t0
```
void print_int(int k) {
    printf("Your number is: %d\n", k);
}

Variable mappings:

- k = $t0