Mac OS Assembly language

Google Chrome: 
(__TEXT,__text) section
0000000100000ef0        pushq   $0x0
0000000100000ef2        movq    %rsp, %rbp
0000000100000ef5        andq    $-0x10, %rsp
0000000100000ef9        movq    0x8(%rbp), %rdi
0000000100000efd        leaq    0x10(%rbp), %rsi
0000000100000f01        movl    %edi, %edx
0000000100000f03        addl    $0x1, %edx
0000000100000f06        shl    $0x3, %edx
0000000100000f09        addq    %rsi, %rdx
0000000100000f0c        movq    %rdx, %rcx
0000000100000f11        jmp     0x100000f15
0000000100000f15        addq    $0x8, %rcx
0000000100000f19        cmpq    $0x0, (%rcx)
0000000100000f1b        jne     0x100000f11
0000000100000f1f        callq   _main
0000000100000f24        movl    %eax, %edi
0000000100000f26        callq   0x100000f46      ## symbol stub for: _exit
0000000100000f2b        hlt
0000000100000f2c        nop
0000000100000f2d        nop
0000000100000f2e        nop
0000000100000f2f        nop
The Principles in CS 42

Theory of computation & Machines (~4 weeks)
What is a computer?

Functional programming (~4 weeks)
There is no difference between functions and variables.

Problem-solving techniques (~3 weeks)
Algorithms & Data structures
What is Computer Science?

Object-oriented programming (~3 weeks)
How do we design a program so that it can grow and change?
How’s CS 42 going?

1. The pace of this class is...
   1 = way too slow; 4 = just right; 7 = way too fast

2. I’m learning a lot in CS 42.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

3. I find the handouts helpful.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

4. I can get help / support from (e.g., Ben, grutors, Piazza), if and when I need it.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

5. When it comes to workload, so far, this is my heaviest course this semester.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

Full name: T. 10/2
What does it mean “to compute”?
An Engineer’s Viewpoint

Computation means **modifying** the bits in memory & registers, step-by-step until we’re done.

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>read r1</code></td>
<td># read dividend from the user</td>
</tr>
<tr>
<td>1</td>
<td><code>write r1</code></td>
<td># echo the input</td>
</tr>
<tr>
<td>2</td>
<td><code>read r2</code></td>
<td># read divisor from the user</td>
</tr>
<tr>
<td>3</td>
<td><code>jeqz r2 7</code></td>
<td># jump to 7 if trying to divide by 0</td>
</tr>
<tr>
<td>4</td>
<td><code>div r3 r1 r2</code></td>
<td># divide user's parameters</td>
</tr>
<tr>
<td>5</td>
<td><code>write r3</code></td>
<td># print the result</td>
</tr>
<tr>
<td>6</td>
<td><code>halt</code></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><code>setn r3 0</code></td>
<td># 0 is the return for division by 0</td>
</tr>
<tr>
<td>8</td>
<td><code>write r3</code></td>
<td># print the result</td>
</tr>
<tr>
<td>9</td>
<td><code>halt</code></td>
<td></td>
</tr>
</tbody>
</table>
Imperative programming
Step-by-step instructions for updating memory (data)

while n > 1:
    f = f*n
    n = n-1
A Mathematician’s viewpoint

Computation means **evaluating** an expression to get its value.

\[ 2 + 2 \]

\[ 8 \sin^3 x + y^2 \]

\[ \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) \ e^{-i\omega t} \ dt \]

\[ \{ \ x \in \mathbb{R} \ | \ x^3 > 5 \ \} \ \cap \ \{ \ y \in \mathbb{R} \ | \ y^2 < 5 \} \]
Functional programming
Calculating answers (by repeatedly evaluating sub-calculations)

\[
\text{fact}(n) := \begin{cases} 
1 & \text{if } n = 0 \\
n \times \text{fact}(n - 1) & \text{otherwise}
\end{cases}
\]

\[
\therefore \quad \text{fact}(4) = 4 \times \text{fact}(3) \\
= 4 \times (3 \times \text{fact}(2)) \\
= 4 \times (3 \times (2 \times \text{fact}(1)))) \\
= 4 \times (3 \times (2 \times (1 \times \text{fact}(0)))) \\
= 4 \times (3 \times (2 \times (1 \times 1)))) \\
= 4 \times (3 \times (2 \times 1)) \\
= 4 \times (3 \times 2) \\
= 4 \times 6 \\
= 24
\]
Features of functional programming

A functional program contains no assignment statements. A variable’s value, once initialized, never changes.

Functional programs use a somewhat limited set of language features. variables, primitive values, conditionals, function definitions & calls

A function’s only purpose is to compute its result; it has no side effects.

Functional programs have referential transparency. An expression always evaluates to the same result, given the same input.
Why are we learning functional programming?

It can teach us something about computation.

Most modern language are a hybrid of imperative & functional styles.

It helps us learn how to choose the right tool for the right job.
Prior experience: programming languages

- None
- Lots
- Soon
- Later

Languages:
- Assembly
- Racket
- Python
- Java
Math notation is not consistent

\sin x \quad x + y \quad x^2

\neg y \quad |\neg 3| \quad \sqrt{2}

(a + b) - c
Racket notation is consistent

\[
\sin x \\
\text{(sin } x) \\
\]
\[
x + y \\
\text{(+ } x \text{ y)} \\
\]
\[
x^2 \\
\text{(sqr } x) \\
\]
\[
y \quad \quad \quad \text{(- 3)} \\
\text{(abs } -3) \\
\]
\[
a + b - c \\
\text{(- (+ a b) c)} \\
\]
Racket: operations (s-expressions)

\[(\text{op } \text{arg}_1 \text{ arg}_2 \ldots \text{arg}_n)\]

- **Rules:**
  - the operation always comes first
  - its arguments (if there are any) follow the operation
  - no commas between arguments
  - everything goes between parentheses

- **Common mistakes:**
  - forgetting parentheses
  - rational vs. integer division (\(\div\) vs. quotient)
  - equality (= vs. equal?)
Dr. Racket
an Integrated Development Environment (IDE) for Racket

#lang racket

boilerplate: the version of Racket we're using

"definitions" (i.e., programs) go here

Run the program!

"interactions" go here
Racket: “variables”

They’re called variables, but we won’t vary them (i.e., their values are constant).

"bind" a value to a variable

(let* ([var₁ expr₁]
    ...
    [varₙ exprₙ])

body)

"scope" of variables
Welcome to DrRacket, version 6.6 [3m].
Language: racket, with debugging; memory limit: 128 MB.
> (let* ([x 30]
        [y 12])
    (+ x y))
42
> x
x: undefined;
cannot reference an identifier before its definition
> (let* ([x 30]
        [z 12])
    (+ x y))
y: undefined;
cannot reference an identifier before its definition
> |