CS 3: Introduction to Software Design

Low Level Assembly Made Accessible (LLAMA) VM

LLAMA Instruction Set

- `halt`: halt program
- `read outr`: Stop for user input, which will then be stored in register `outr`
- `write inr`: Print the (decimal) contents of register `inr` on standard output
- `print inr`: Print the (ascii) contents of register `inr` on standard output
- `const outr, CONST`: Sets `outr = CONST`
- `load outr, inr`: Sets `outr = memory[inr]`
- `store inr1, inr2`: Sets `memory[inr2] = inr1`
- `mov outr, inr`: Sets `outr = inr`
- `add outr, inr1, inr2`: Sets `outr = inr1 + inr2`
- `sub outr, inr1, inr2`: Sets `outr = inr1 - inr2`
- `mul outr, inr1, inr2`: Sets `outr = inr1 * inr2`
- `div outr, inr1, inr2`: Sets `outr = inr1 / inr2`
- `mod outr, inr1, inr2`: Sets `outr = inr1 % inr2`
- `jeq outr, LABEL`: Jumps to the line indicated `LABEL` if `outr == 0`
- `jgt outr, LABEL`: Jumps to the line indicated `LABEL` if `outr > 0`

div.llama

```
read r1              # read dividend from the user
write r1             # echo the input
read r2              # read divisor from the user
jeq r2, div_by_zero # jump to div_by_zero if trying to divide by 0

div r3, r1, r2       # divide user's parameters
write r3             # write the result
halt

div_by_zero:
const r3, 0          # 0 is the result for division by 0
write r3             # write the result
halt
```

guessing-game.llama

```
# Generate the answer using an LCG into r0
....


while:
  const r1 0
  print "Guess a Number >>" # This is shorthand for a bunch of prints...
  read r1
  sub r2 r1 r0
L1:
  [_________________________________]
  const r3 72 # 72 = 'H'
  const r4 73 # 73 = 'I'
L2:
  [_________________________________]
  const r3 76 # 76 = 'L'
  const r4 79 # 79 = 'O'

goto_while:
  print r3
  print r4
  const r2 10 # 10 = '\n'
  print r2
  const r5, 0
L3:
  [_________________________________]

win:
  const r0 89 # 89 = 'Y'
  print r0
  const r0 65 # 65 = 'A'
  print r0
  const r0 89 # 89 = 'Y'
  print r0
  const r0 10 # 10 = '\n'
  print r0
  halt
```

```
prime-sieve.llama
# Define true/false constants for usage throughout the program
const r0 0
const r1 1
const r2 2

# Read n, upper limit of prime sieve
read r3

mov r4 r3    # i = n

# Set all elements to true
loop:
L4: [_________________________________] # i -= 2
L5: [_________________________________] # break if we hit i == 2
L6: [_________________________________] # i--
L7: [_________________________________] # i += 2
L8: [_________________________________] # mem[i] = 1
L9: [_________________________________] # else goto loop

init_bc:
# Initialize 0, 1
store r0 r0
store r0 r1

# Initialize sieve variables
sieve:
sub r4 r0 r1

# Loop from -2 -> -n
loop2:
    const r2 2
    sub r4 r4 r1
    add r4 r4 r3
    jeq r4 done
    sub r4 r4 r3
    sub r4 r0 r4
    load r5 r4
    sub r4 r0 r4
    jeq r5 loop2
    const r2 0
    add r2 r0 r4
    add r2 r2 r4
div_loop:
    # This loop sets all the multiples of r4 memory cells to 0
    # (indicating they are not prime)  ...
done:
   mov r4 r3    # i = n
   mov r3 r0
# Finish up by counting the number of primes and printing those out ...
```