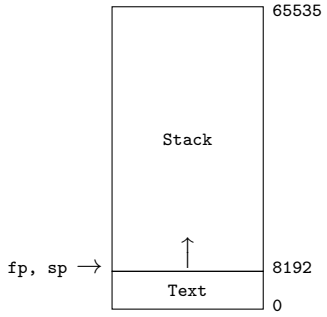
Marvin is a hypothetical computer with sixteen 16-bit registers and 65,536 32-bit words of main memory (aka RAM). In addition to the sixteen registers, Marvin has a 16-bit program counter `pc` and a 32-bit instruction register `ir`.

Registers: Marvin specifies the following conventions for its sixteen registers:

- `r0` – `r11` are general purpose registers.
- `r12` is reserved to store the return address (`ra`) of the calling subroutine (aka function).
- `r13` is reserved to store the return value of a subroutine.
- `r14`, called the frame pointer (`fp`), is reserved to store the base address of the most recent frame on the stack. It is initialized to 8192 when the computer boots.
- `r15`, called the stack pointer (`sp`), is reserved to store the address of the top of the stack. It is initialized to 8192 when the computer boots.

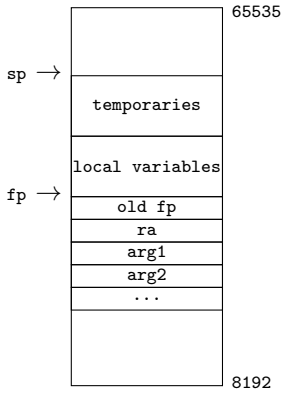
Main Memory: Marvin's main memory is divided into a text segment and a stack segment, as shown below:



The text segment extends from address 0 to address 8191. A Marvin program, which is a `.marv` file, is assembled and loaded into this segment starting at address 0.

The stack segment starts at address 8192 and grows upwards[1] up to address 65535. This is where a subroutine's arguments, its local variables, and temporaries are stored.

When a subroutine is called, a stack frame must be created for it on the stack, and it must be oraganized as shown below:



[1] Conventionally, the stack segment grows downwards from a higher address towards a lower address.

Instructions: Marvin supports 32 instructions, each of which accepts between 0 and 3 arguments (aka inputs).

System instructions:

Opcode	32-bit Machine Code	Description
`halt`	`00000000 00000000 00000000 00000000`	stops the machine
`read rX`	`00000001 00000000 00000000 0000XXXX`	sets `rX = N`, where `N` $\in [-2^{15}, 2^{15}-1]$ read from standard input
`write rX`	`00000010 00000000 00000000 0000XXXX`	writes `rX` to standard output
`nop`	`00000011 00000000 00000000 00000000`	does nothing

Arithmetic instructions:

Opcode	32-bit Machine Code	Description
`neg rX rY`	`00001001 00000000 00000000 XXXXYYYY`	sets `rX = -rY`
`add rX rY rZ`	`00001010 00000000 0000XXXX YYYYZZZZ`	sets `rX = rY + rZ`
`sub rX rY rZ`	`00001011 00000000 0000XXXX YYYYZZZZ`	sets `rX = rY - rZ`
`mul rX rY rZ`	`00001100 00000000 0000XXXX YYYYZZZZ`	sets `rX = rY * rZ`
`div rX rY rZ`	`00001101 00000000 0000XXXX YYYYZZZZ`	sets `rX = rY // rZ`
`mod rX rY rZ`	`00001110 00000000 0000XXXX YYYYZZZZ`	sets `rX = rY % rZ`

Jump instructions:

Opcode	32-bit Machine Code	Description
`jumpn N`	`00001111 00000000 NNNNNNNN NNNNNNNN`	jumps to instruction `N`
`jumpr rX`	`00010000 00000000 00000000 0000XXXX`	jumps to `rX`
`jeqzn rX N`	`00010001 0000XXXX NNNNNNNN NNNNNNNN`	jumps to instruction `N` if `rX == 0`
`jnezn rX N`	`00010010 0000XXXX NNNNNNNN NNNNNNNN`	jumps to instruction `N` if `rX != 0`
`jgen rX rY N`	`00010011 XXXXYYYY NNNNNNNN NNNNNNNN`	jumps to instruction `N` if `rX >= rY`
`jlen rX rY N`	`00010110 XXXXYYYY NNNNNNNN NNNNNNNN`	jumps to instruction `N` if `rX <= rY`
`jeqn rX rY N`	`00010100 XXXXYYYY NNNNNNNN NNNNNNNN`	jumps to instruction `N` if `rX == rY`
`jnen rX rY N`	`00010101 XXXXYYYY NNNNNNNN NNNNNNNN`	jumps to instruction `N` if `rX != rY`
`jgtn rX rY N`	`00010111 XXXXYYYY NNNNNNNN NNNNNNNN`	jumps to instruction `N` if `rX > rY`
`jltn rX rY N`	`00011000 XXXXYYYY NNNNNNNN NNNNNNNN`	jumps to instruction `N` if `rX < rY`
`calln rX N`	`00011001 0000XXXX NNNNNNNN NNNNNNNN`	sets `rX = pc + 1` and jumps to instruction `N`

Instructions for setting register data:

Opcode	32-bit Machine Code	Description
`set0 rX`	`00000100 00000000 00000000 0000XXXX`	sets `rX = 0`
`set1 rX`	`00000101 00000000 00000000 0000XXXX`	sets `rX = 1`
`setn rX N`	`00000110 0000XXXX NNNNNNNN NNNNNNNN`	sets `rX = N`, where `N` $\in [-2^{15}, 2^{15}-1]$
`addn rX N`	`00000111 0000XXXX NNNNNNNN NNNNNNNN`	sets `rX = rX + N`, where `N` $\in [-2^{15}, 2^{15}-1]$
`copy rX rY`	`00001000 00000000 00000000 XXXXYYYY`	sets `rX = rY`

Instructions for interacting with memory:

Opcode	32-bit Machine Code	Description
`pushr rX rY`	`00011010 00000000 00000000 XXXXYYYY`	sets `mem[rY++] = rX`
`popr rX rY`	`00011011 00000000 00000000 XXXXYYYY`	sets `rX = mem[--rY]`
`loadn rX rY N`	`00011100 XXXXYYYY NNNNNNNN NNNNNNNN`	sets `rX = mem[rY + N]`, where `N` $\in [-2^{15}, 2^{15}-1]$
`storen rX rY N`	`00011101 XXXXYYYY NNNNNNNN NNNNNNNN`	sets `mem[rY + N] = rX`, where `N` $\in [-2^{15}, 2^{15}-1]$
`loadr rX rY`	`00011110 00000000 00000000 XXXXYYYY`	sets `rX = mem[rY]`
`storer rX rY`	`00011111 00000000 00000000 XXXXYYYY`	sets `mem[rY] = rX`

Marvin Emulator: The Python program `marvin.py` serves as an emulator for the Marvin machine. Here is the usage syntax for the program:

Countdown.marv

```
$ python3 marvin.py

Usage: marvin.py [-v] <.marv file>

This program serves as an emulator for a register-based machine called Marvin (named after
the paranoid android character, Marvin, from The Hitchhiker's Guide to the Galaxy by
Douglas Adams). The design of the machine was inspired by that of the Harvey Mudd
Miniature Machine (HMMM) developed at Harvey Mudd College. The program accepts a .marv file
as input, assembles and simulates the instructions within, and prints any output to stdout.
Any input to the .marv program is via stdin. If the optional -v argument is specified,
the emulator prints the assembled instructions to stdout before simulating them.
$ _
```

Here is a sample Marvin program called `Coundown.marv` that accepts n (int) from standard input and writes to standard output a countdown from n to 0.

Countdown.marv

```
# Accepts n (int) from standard input and writes a countdown from n to 0 to standard output.

0    read     r0          # read n
1    set0     r1          # zero = 0
2    jltn     r0 r1 6     # if n < zero jump to 6
3    write    r0          # write n
4    addn     r0 -1       # n = n - 1
5    jumpn    2           # jump to 2
6    halt                 # halt the machine
```

Here is the output from running `Countdown.marv` using `marvin.py`, which is an emulator for the Marvin machine.

>_ ~/workspace/marvin

```
$ python3 marvin.py -v Countdown.marv
    0: 00000001 00000000 00000000 00000000                    0: read   r0
    1: 00000100 00000000 00000000 00000001                    1: set0   r1
    2: 00011000 00000001 00000000 00000110                    2: jltn   r0 r1 6
    3: 00000010 00000000 00000000 00000000                    3: write  r0
    4: 00000111 00000000 10000000 00000001                    4: addn   r0 -1
    5: 00001111 00000000 00000000 00000010                    5: jumpn  2
    6: 00000000 00000000 00000000 00000000                    6: halt

5
5
4
3
2
1
0
$ _
```