Introduction to Programming in Python

Building a Computer: Von Neumann Architecture

Outline

Von Neumann Architecture

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices for doing arithmetic and a small number of (scratch) memory called registers

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices for doing arithmetic and a small number of (scratch) memory called registers

The computer's main memory (ie, its RAM) is separate from the CPU but connected to it

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices for doing arithmetic and a small number of (scratch) memory called registers

The computer's main memory (ie, its RAM) is separate from the CPU but connected to it

A program, which is a long list of instructions, is stored in the RAM and executed in the CPU, one instruction at a time

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices for doing arithmetic and a small number of (scratch) memory called registers

The computer's main memory (ie, its RAM) is separate from the CPU but connected to it

A program, which is a long list of instructions, is stored in the RAM and executed in the CPU, one instruction at a time

The CPU has two special registers

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices for doing arithmetic and a small number of (scratch) memory called registers

The computer's main memory (ie, its RAM) is separate from the CPU but connected to it

A program, which is a long list of instructions, is stored in the RAM and executed in the CPU, one instruction at a time

The CPU has two special registers

1. A program counter to track the next instruction to execute

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices for doing arithmetic and a small number of (scratch) memory called registers

The computer's main memory (ie, its RAM) is separate from the CPU but connected to it

A program, which is a long list of instructions, is stored in the RAM and executed in the CPU, one instruction at a time

The CPU has two special registers

1. A program counter to track the next instruction to execute
2. An instruction register to store the next instruction for execution

Marvin Machine

Marvin Machine

Marvin simulates a computer that has sixteen 16-bit registers and 65,536 32-bit words of main memory

Marvin Machine

Marvin simulates a computer that has sixteen 16-bit registers and 65,536 32-bit words of main memory

In addition to the sixteen registers, Marvin has a 16-bit program counter `pc` and a 32-bit instruction register `ir`

Marvin Machine

Marvin simulates a computer that has sixteen 16-bit registers and 65,536 32-bit words of main memory

In addition to the sixteen registers, Marvin has a 16-bit program counter `pc` and a 32-bit instruction register `ir`

A Marvin program (ie, a `.marv` file) is assembled and loaded into memory starting at location 0

Marvin Machine

Marvin simulates a computer that has sixteen 16-bit registers and 65,536 32-bit words of main memory

In addition to the sixteen registers, Marvin has a 16-bit program counter `pc` and a 32-bit instruction register `ir`

A Marvin program (ie, a `.marv` file) is assembled and loaded into memory starting at location 0

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

System instructions

`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

System instructions

`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

`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

`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

`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 setting register data

`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

`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`

SumOfSquares.marv

Standard input	a (int) and b (int)
Standard output	$a^2 + b^2$

SumOfSquares.marv

Standard input	a (int) and b (int)
Standard output	$a^2 + b^2$

~/workspace/ipp

```
$ _
```

SumOfSquares.marv

Standard input	a (int) and b (int)
Standard output	$a^2 + b^2$

~/workspace/ipp

```
$ python3 marvin.py -v data/SumOfSquares.marv
```

SumOfSquares.marv

Standard input	a (int) and b (int)
Standard output	$a^2 + b^2$

~/workspace/ipp

```
$ python3 marvin.py -v data/SumOfSquares.marv
    0: 00000001 00000000 00000000 00000000                  0: read   r0
    1: 00000001 00000000 00000000 00000001                  1: read   r1
    2: 00001100 00000000 00000010 00000000                  2: mul    r2 r0 r0
    3: 00001100 00000000 00000011 00010001                  3: mul    r3 r1 r1
    4: 00001010 00000000 00000100 00100011                  4: add    r4 r2 r3
    5: 00000010 00000000 00000000 00000100                  5: write  r4
    6: 00000000 00000000 00000000 00000000                  6: halt


-
```

SumOfSquares.marv

Standard input	a (int) and b (int)
Standard output	$a^2 + b^2$

~/workspace/ipp

```
$ python3 marvin.py -v data/SumOfSquares.marv
    0: 00000001 00000000 00000000 00000000                  0: read   r0
    1: 00000001 00000000 00000000 00000001                  1: read   r1
    2: 00001100 00000000 00000010 00000000                  2: mul    r2 r0 r0
    3: 00001100 00000000 00000011 00010001                  3: mul    r3 r1 r1
    4: 00001010 00000000 00000100 00100011                  4: add    r4 r2 r3
    5: 00000010 00000000 00000000 00000100                  5: write  r4
    6: 00000000 00000000 00000000 00000000                  6: halt

6
```

SumOfSquares.marv

Standard input	a (int) and b (int)
Standard output	$a^2 + b^2$

~/workspace/ipp

```
$ python3 marvin.py -v data/SumOfSquares.marv
    0: 00000001 00000000 00000000 00000000                  0: read   r0
    1: 00000001 00000000 00000000 00000001                  1: read   r1
    2: 00001100 00000000 00000010 00000000                  2: mul    r2 r0 r0
    3: 00001100 00000000 00000011 00010001                  3: mul    r3 r1 r1
    4: 00001010 00000000 00000100 00100011                  4: add    r4 r2 r3
    5: 00000010 00000000 00000000 00000100                  5: write  r4
    6: 00000000 00000000 00000000 00000000                  6: halt

6
-
```

SumOfSquares.marv

Standard input	a (int) and b (int)
Standard output	$a^2 + b^2$

~/workspace/ipp

```
$ python3 marvin.py -v data/SumOfSquares.marv
    0: 00000001 00000000 00000000 00000000                  0: read   r0
    1: 00000001 00000000 00000000 00000001                  1: read   r1
    2: 00001100 00000000 00000010 00000000                  2: mul    r2 r0 r0
    3: 00001100 00000000 00000011 00010001                  3: mul    r3 r1 r1
    4: 00001010 00000000 00000100 00100011                  4: add    r4 r2 r3
    5: 00000010 00000000 00000000 00000100                  5: write  r4
    6: 00000000 00000000 00000000 00000000                  6: halt

6
8
```

SumOfSquares.marv

Standard input	a (int) and b (int)
Standard output	$a^2 + b^2$

~/workspace/ipp

```
$ python3 marvin.py -v data/SumOfSquares.marv
    0: 00000001 00000000 00000000 00000000                  0: read   r0
    1: 00000001 00000000 00000000 00000001                  1: read   r1
    2: 00001100 00000000 00000010 00000000                  2: mul    r2 r0 r0
    3: 00001100 00000000 00000011 00010001                  3: mul    r3 r1 r1
    4: 00001010 00000000 00000100 00100011                  4: add    r4 r2 r3
    5: 00000010 00000000 00000000 00000100                  5: write  r4
    6: 00000000 00000000 00000000 00000000                  6: halt

6
8
100
$ _
```

</> SumOfSquares.marv

```
# Accepts a (int) and b (int) from standard input and writes to standard output the value of
# a^2 + b^2.

0    read     r0          # read a
1    read     r1          # read b
2    mul      r2 r0 r0    # c = a * a
3    mul      r3 r1 r1    # d = b * b
4    add      r4 r2 r3    # e = c + d
5    write    r4          # write e
6    halt                 # halt the machine
```

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
```

Registers

pc		
ir		
r0		
r1		
r2		
r3		
r4		

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
_
```

Registers

pc	0	
ir	00000001 00000000 00000000 00000000	read r0
r0		
r1		
r2		
r3		
r4		

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
```

Registers

pc	0	
ir	00000001 00000000 00000000 00000000	read r0
r0		
r1		
r2		
r3		
r4		

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
-
```

Registers

pc	1	
ir	00000001 00000000 00000000 00000001	read r1
r0	6	
r1		
r2		
r3		
r4		

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
8
```

Registers

pc	1	
ir	00000001 00000000 00000000 00000001	read r1
r0	6	
r1		
r2		
r3		
r4		

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
8
```

Registers

pc	2	
ir	00001100 00000000 00000010 00000000	mul r2 r0 r0
r0	6	
r1	8	
r2		
r3		
r4		

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
8
```

Registers

pc	3	
ir	00001100 00000000 00000011 00010001	mul r3 r1 r1
r0	6	
r1	8	
r2	36	
r3		
r4		

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
8
```

Registers

pc	4	
ir	00001010 00000000 00000100 00100011	add r4 r2 r3
r0	6	
r1	8	
r2	36	
r3	64	
r4		

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
8
```

Registers

pc	5	
ir	00000010 00000000 00000000 00000100	write r4
r0	6	
r1	8	
r2	36	
r3	64	
r4	100	

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
8
100
```

Registers

pc	6	
ir	00000000 00000000 00000000 00000000	halt
r0	6	
r1	8	
r2	36	
r3	64	
r4	100	

Memory

0	00000001 00000000 00000000 00000000	read r0
1	00000001 00000000 00000000 00000001	read r1
2	00001100 00000000 00000010 00000000	mul r2 r0 r0
3	00001100 00000000 00000011 00010001	mul r3 r1 r1
4	00001010 00000000 00000100 00100011	add r4 r2 r3
5	00000010 00000000 00000000 00000100	write r4
6	00000000 00000000 00000000 00000000	halt
7	...	
8	...	
...	...	
65535	...	

Terminal

```
>_
$ python3 marvin.py data/SumOfSquares.marv
6
8
100
$ _
```

Registers

pc		
ir		
r0		
r1		
r2		
r3		
r4		

Countdown.marv

Standard input	n (int)
Standard output	a countdown from n to 0

Countdown.marv

Standard input	*n* (int)
Standard output	a countdown from *n* to 0

>_ ~/workspace/ipp

```
$ _
```

Countdown.marv

Standard input	*n* (int)
Standard output	a countdown from *n* to 0

>_ ~/workspace/ipp

```
$ python3 marvin.py data/Countdown.marv
```

Countdown.marv

Standard input	*n* (int)
Standard output	a countdown from *n* to 0

>_ ~/workspace/ipp

```
$ python3 marvin.py data/Countdown.marv
-
```

Countdown.marv

Standard input	*n* (int)
Standard output	a countdown from *n* to 0

>_ ~/workspace/ipp

```
$ python3 marvin.py data/Countdown.marv
5
```

Countdown.marv

Standard input	*n* (int)
Standard output	a countdown from *n* to 0

~/workspace/ipp

```
$ python3 marvin.py data/Countdown.marv
5
5
4
3
2
1
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
```