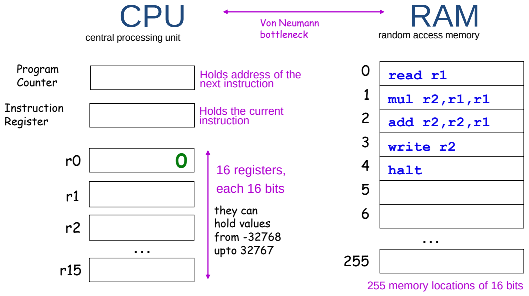
Hmmm Assembly Language

A QUICK OVERVIEW OF CPU

Hmmm
Harvey Mudd Miniature Machine



```
0   read r1
1   mul r2,r1,r1
2   add r2,r2,r1
3   write r2
4   halt
```

Fetch-Execution Cycle

CPU execution repeats the *Fetch-Execution Cycle*,

- **Fetch** a instruction from memory
- **Decode** to determine what the instruction intends to do
- **Execution** the instruction as specified

during which the **PC** is incremented properly.

Assembly Language

register-level programming

```
add r2 r2 r2        reg2 = reg2 + reg2
```
crazy, perhaps, but used ALL the time

```
sub r2 r1 r4        reg2 = reg1 - reg4
```
which is why it is written this way in python!

```
mul r7 r6 r2        reg7 = reg6 * reg2

div r1 r1 r2        reg1 = reg1 / reg2
```
INTEGER division - no remainders

```
setn r1 42          reg1 = 42
addn r1 -1          reg1 = reg1 - 1
```
you can replace 42 with anything from -128 to 127

a shortcut

assembly code

actual meaning

```
read r1
write r1
```
read from keyboard and write to screen

Each of these instructions (and many more) get implemented for a particular processor and particular machine… .

the assembler

a program that translates from human-readable assembly language into machine language (binary)

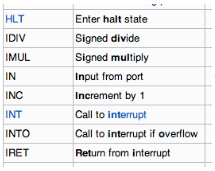
```
setn r1,6
setn r2,7
mul r3,r1,r2
write r3
```
assembly code

```
0000 0001 0000 0001
1000 0010 0001 0001
0110 0010 0010 0001
0000 0010 0000 0010
0000 0000 0000 0000
```
executable machine code

We use
hmmmAssembler.py
to assemble

We use
hmmmSimulator.py
to execute the machine code

Real Assembly Language

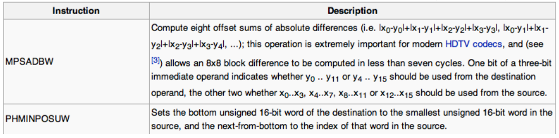


HLT	Enter halt state
IDIV	Signed divide
IMUL	Signed multiply
IN	Input from port
INC	Increment by 1
INT	Call to interrupt
INTO	Call to interrupt if overflow
IRET	Return from interrupt

Hmmm has a subset common to ***all*** real assembly languages.

A few of the many basic processor instructions (Intel)

two of the Intel instructions (SSE4, 2008)



Instruction	Description
MPSADBW	Compute eight offset sums of absolute differences (i.e. $\lvert x_0-y_0\rvert+\lvert x_1-y_1\rvert+\lvert x_2-y_2\rvert+\lvert x_3-y_3\rvert$, $\lvert x_0-y_1\rvert+\lvert x_1-y_2\rvert+\lvert x_2-y_3\rvert+\lvert x_3-y_4\rvert$, ...); this operation is extremely important for modern HDTV codecs, and (see [3]) allows an 8x8 block difference to be computed in less than seven cycles. One bit of a three-bit immediate operand indicates whether y_0 .. y_{11} or y_4 .. y_{15} should be used from the destination operand, the other two whether $x_0..x_3$, $x_4..x_7$, $x_8..x_{11}$ or $x_{12}..x_{15}$ should be used from the source.
PHMINPOSUW	Sets the bottom unsigned 16-bit word of the destination to the smallest unsigned 16-bit word in the source, and the next-from-bottom to the index of that word in the source.

What will this program output? (1)

Suppose your input is **42**
when line 0 is executed

r1 42
General-purpose register r1

r2
General-purpose register r2

r3
General-purpose register r3

```
0 read r1
1 setn r2 9
2 sub r3 r1 r2
3 div r3 r3 r2
4 addn r3 -1
5 write r3
6 halt
```

What will this program output? (2)

Suppose your input is **42**
when line 1 is executed

r1 42
General-purpose register r1

r2 9
General-purpose register r2

r3
General-purpose register r3

```
0 read r1
1 setn r2 9
2 sub r3 r1 r2
3 div r3 r3 r2
4 addn r3 -1
5 write r3
6 halt
```

What will this program output? (3)

Suppose your input is **42**
when line 2 is executed

r1 42
General-purpose register r1

r2 9
General-purpose register r2

r3 33
General-purpose register r3

```
      0 read r1
      1 setn r2 9
(x-9) 2 sub r3 r1 r2
      3 div r3 r3 r2
      4 addn r3 -1
      5 write r3
      6 halt
```

What will this program output? (4)

Suppose your input is **42**
when line 3 is executed

r1 42
General-purpose register r1

r2 9
General-purpose register r2

r3 3
General-purpose register r3

```
         0 read r1
         1 setn r2 9
   (x-9) 2 sub r3 r1 r2
(x-9) // 9 3 div r3 r3 r2
         4 addn r3 -1
         5 write r3
         6 halt
```

What will this program output? (5)

Suppose your input is **42**
when line 4 is executed

r1 42
General-purpose register r1

r2 9
General-purpose register r2

r3 2
General-purpose register r3

```
0 read r1
1 setn r2 9
2 sub r3 r1 r2
3 div r3 r3 r2
4 addn r3 -1
5 write r3
6 halt
```

What will this program output? (6)

Suppose your input is **42** when line 5 is executed

r1 **42**
General-purpose register r1

r2 **9**
General-purpose register r2

r3 **2**
General-purpose register r3

What is the equivalent of Python?

```
0 read r1
1 setn r2 9
2 sub r3 r1 r2
3 div r3 r3 r2
4 addn r3 -1
5 write r3
6 halt
```

Write an Hmmm program to compute

$x^2 + 3x - 4$

HINT: Use the previous program as a model

For your reference:
((x – 9) // 9) - 1

```
0  read  r1
1  setn r2, 9
2  sub   r3, r1, r2
3  div   r3, r3, r2
4  addn  r3, -1
5  write r3
6  halt
```

$x^2 + 3x - 4$

```
0    read    r1                 # r1 = read x
1    mul     r2, r1, r1         # r2 = x**2
2    setn    r3, 3              # Need 3 in reg
3    mul     r3, r3, r1         # r3 = 3*x
4    add     r2, r2, r3         # r2 = r2 + r3
5    addn    r2,-4              # x**2 + 3*x - 4
6    write   r2                 # Output result
7    halt
```

Is this enough?

Why ***couldn't*** we implement Python using our Hmmm Assembly so far?

What's missing?

```
0 read r1
1 mul r2 r1 r1
2 add r2 r2 r1
3 write r2
4 halt
```

Loops and `ifs`

We ***couldn't*** implement Python using our Hmmm Assembly Language so far... !

It's too linear!

"straight-line code"

```
0 read r1
1 mul r2 r1 r1
2 add r2 r2 r1
3 write r2
4 jumpn 1
```

Hmmm, Let's `jump` !

CPU
central processing unit

r1
General-purpose register r1

r2
General-purpose register r2

RAM
random access memory

```
0 setn r1 42
1 write r1
2 addn r1 1
3 jumpn 1
4 halt
```

What does this program do?

What if we replace 1 with 2?

jumps

Unconditional jump

jumpn 42 "jump to program line number **42**"

Conditional jumps

jeqzn r1 42	IF **r1 == 0** THEN jump to line number 42
jgtzn r1 42	IF **r1 > 0** THEN jump to line number 42
jltzn r1 42	IF **r1 < 0** THEN jump to line number 42
jnezn r1 42	IF **r1 != 0** THEN jump to line number 42

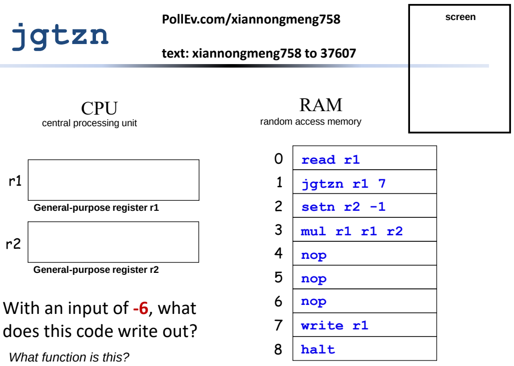
Indirect jump

jumpr r1 Jump to the line number ***stored*** in **reg1**!

jgtzn

PollEv.com/xiannongmeng758

text: xiannongmeng758 to 37607



```
0  read r1
1  jgtzn r1 7
2  setn r2 -1
3  mul r1 r1 r2
4  nop
5  nop
6  nop
7  write r1
8  halt
```

With an input of **-6**, what does this code write out?

What function is this?

(A) -42 (B) -6 (C) -1 (D) 6 (E) 42