What does it do?

```
0   0100000000000101
1   0001000000000100
2   0101000000000101
3   1111000000000000
4   0000000000001010
```

1111	STP	Stop the computer
0001	ADD	Add accum. to operand
0010	SUB	Subtract operand from accum.
0011	LOD	Load memory cell into accum.
0100	LDI	Load immediate into accum.
0101	STO	Store accum. into memory cell
0110	INP	Input value and store into accum.
0111	OUT	Output the value from accum.
1000	JMP	Jump to instruction
1001	JNG	Jump to instruction if accum<0
1010	JZR	Jump to instruction if accum=0

SSC Programs

```
0     0100000000000101
      LDI 5       load the number 5
1     0001000000000100
      ADD 4       add contents of memory location 4
2     0101000000000101
      STO 5       store result in memory location 5
3     1111000000000000
      STOP
4     0000000000001010
      <data>      number to add
5     <data>      place to put result
```

SSC Programs

```
        LDI 5       load the number 5

        ADD 4       add contents of memory location 4

        STO 5       store result in memory location 5

        STOP

4       <data>      number to add
5       <data>      place to put result
```

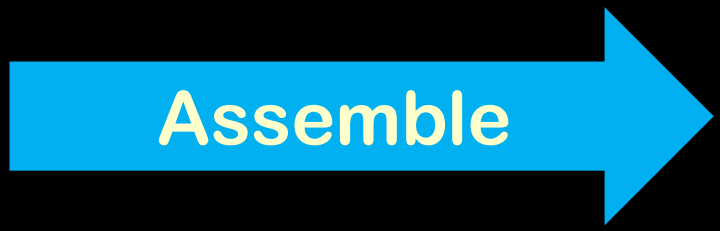
SSC Programs

```
        LDI 5       load the number 5

        ADD X       add contents of memory location 4

        STO Y       store result in memory location 5

        STOP

X       <data>      number to add
Y       <data>      place to put result
```

Assembly Language

- Mnemonic codes represent machine language instructions
  - Write these alphanumeric codes instead of binary (Yea!)

```
    LDI 5
    ADD X
    STO Y
    STP
X   DAT 10
Y   DAT 0
```




```
0  0100000000000101
1  0001000000000100
2  0101000000000101
3  1111000000000000
4  0000000000001010
5  0000000000000000
```

Assembly Language

- Mnemonic codes represent machine language instructions
  - Decode binary into codes

```
0   LDI 5
1   ADD 4
2   STO 5
3   STP
4   DAT 10
5   DAT 0
```


```
0   0100000000000101
1   0001000000000100
2   0101000000000101
3   1111000000000000
4   0000000000001010
5   0000000000000000
```

Assembly Language

- Mnemonic codes represent machine language instructions
  - Decode binary into codes
  - Humanize

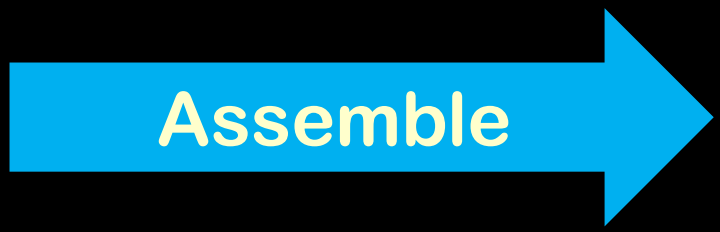
```
    LDI 5
    ADD X
    STO Y
    STP
X   DAT 10
Y   DAT 0
```


```
0  0100000000000101
1  0001000000000100
2  0101000000000101
3  1111000000000000
4  0000000000001010
5  0000000000000000
```

Assembly Language

- Mnemonic codes represent machine language instructions
  - Decode binary into codes
  - Humanize

```
    LDI 5
    ADD X
    STO Y
    OUT
    STP
X   DAT 10
Y   DAT 0
```

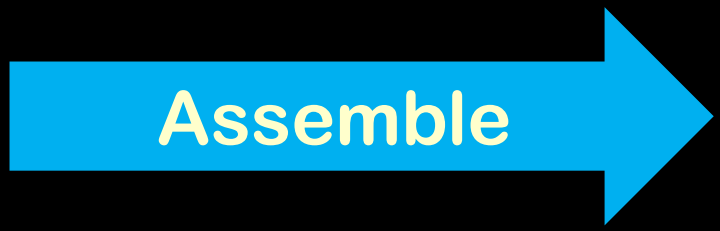



```
0   0100000000000101
1   0001000000000100
2   0101000000000101
3   1111000000000000
4   0000000000001010
5   0000000000000000
```

Assembly Language

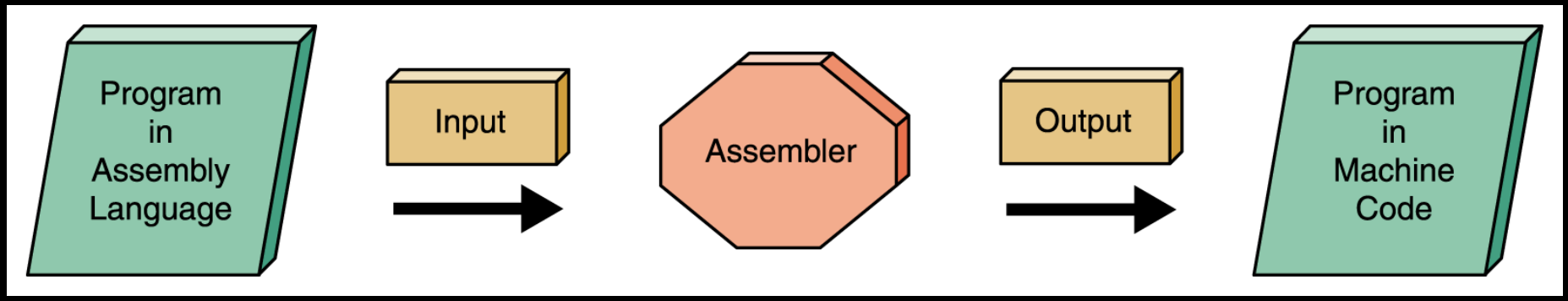
- Mnemonic codes represent machine language instructions
  - Decode binary into codes
  - Humanize

```
    LDI 5
    ADD X
    STO Y
    OUT
    STP
X   DAT 10
Y   DAT 0
```




```
0  0100000000000101
1  0001000000000101
2  0101000000000110
3  0111000000000000
4  1111000000000000
5  0000000000001010
6  0000000000000000
```

Assembly Process



LOAD Program into memory

EXECUTE

Set Program Counter to 0 and Fetch

SSC Programs

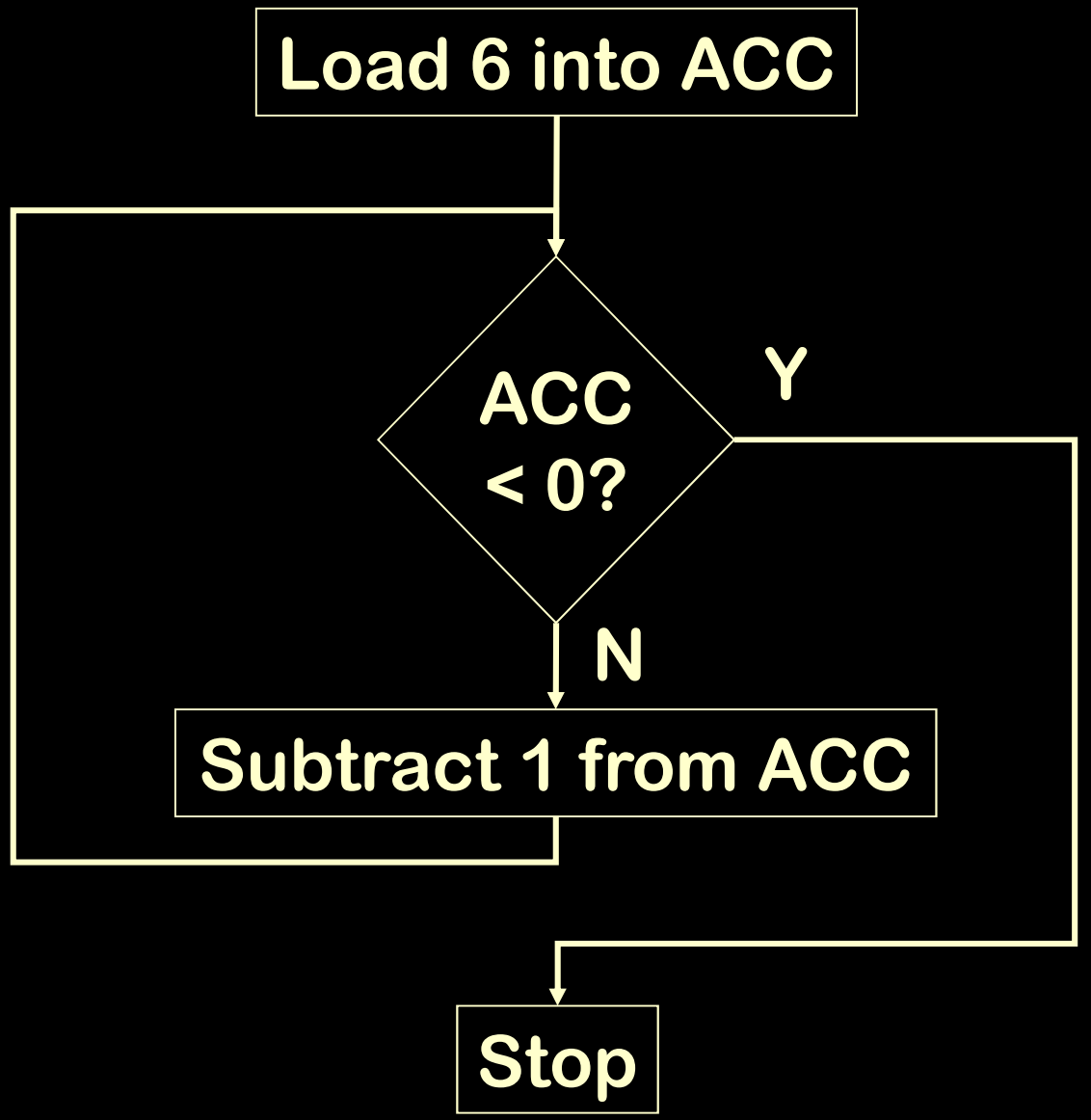
```
0     INP       get a number from the input field
1     OUT       write the acc. to the output field
2     STOP      stop
```

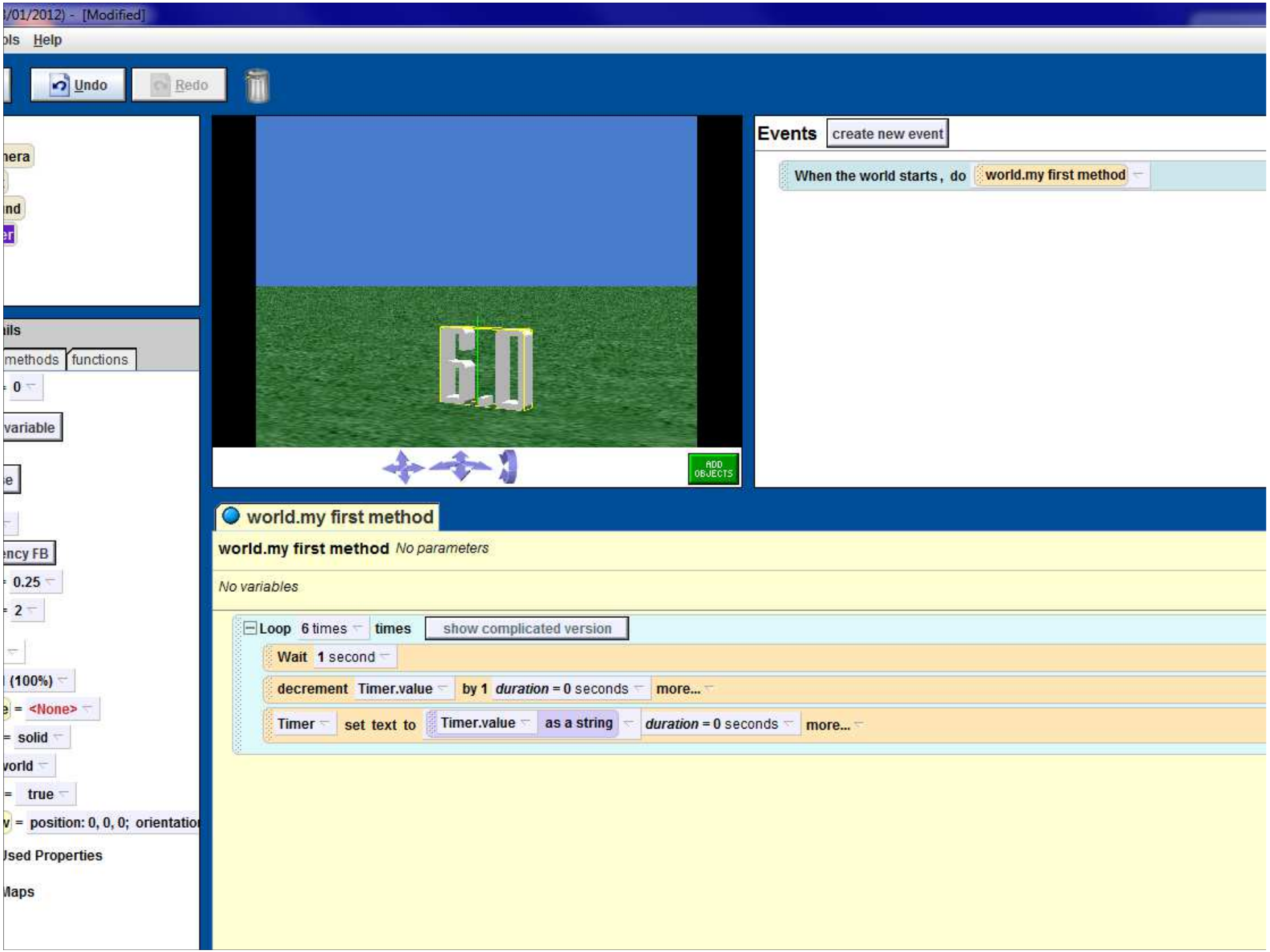
SSC Assembly Language...

Counting Loop

```
0   LDI 6        put "6" into accumlator
1   JNG 4        if acc < 0, jump to step 4
2   SUB 5        subtract value in location 5
3   JMP 1        jump to step 1
4   STP          stop
5   DAT 1        data, value is 1
```

Accumulator:


Load 6 into ACC
ACC
< 0?
Y
N
Subtract 1 from ACC
Stop


/01/2012) - [Modified]
ols Help
Undo
Redo
nera
nd
er
ils
methods
functions
0
variable
e
ency FB
0.25
2
(100%)
= <None>
= solid
vorld
= true
v = position: 0, 0, 0; orientatio
Used Properties
Maps
6.0
ADD OBJECTS
Events
create new event
When the world starts, do world.my first method
world.my first method
world.my first method No parameters
No variables
Loop 6 times times
show complicated version
Wait 1 second
decrement Timer.value by 1 duration = 0 seconds more...
Timer set text to Timer.value as a string duration = 0 seconds more...