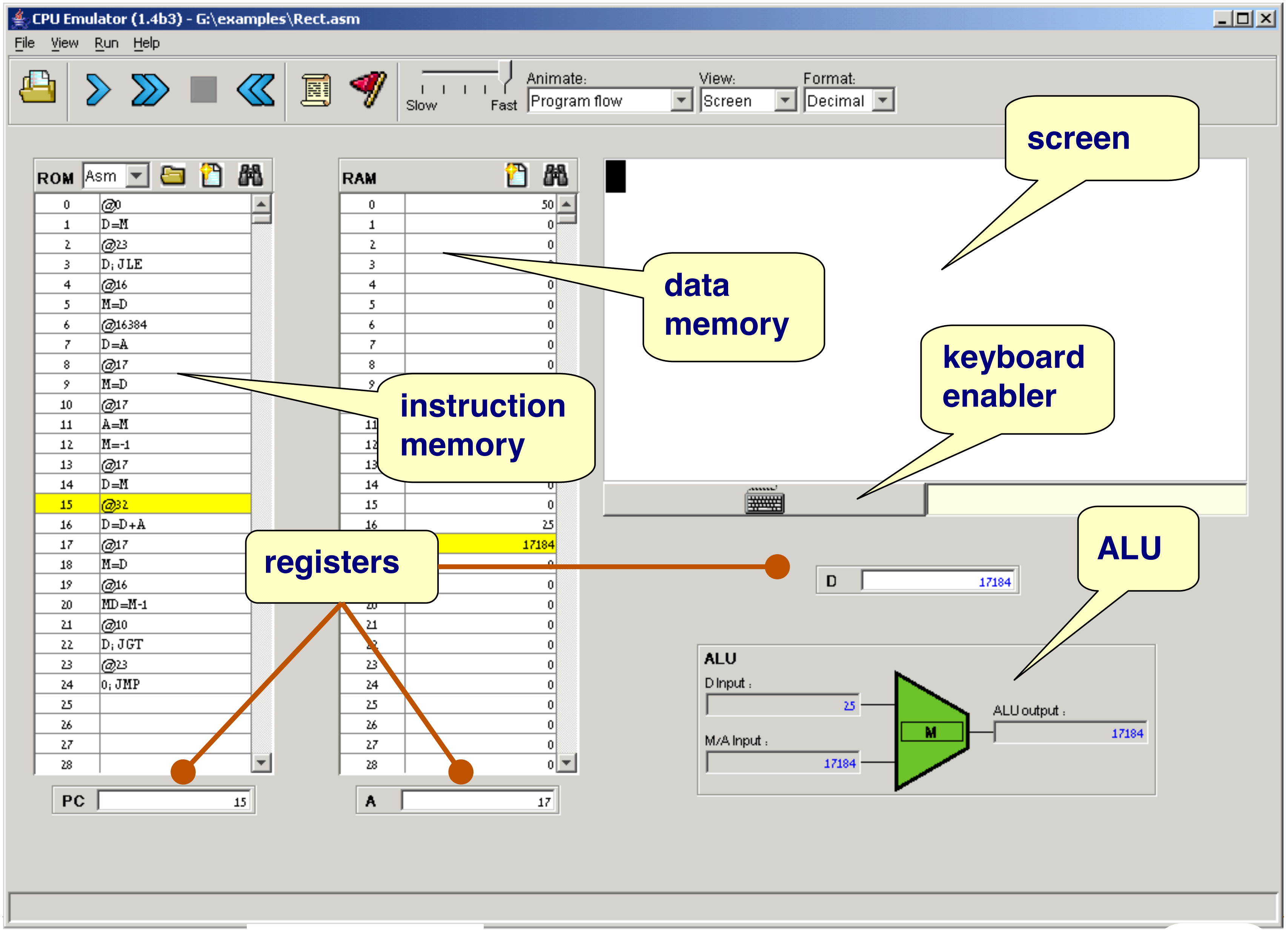
# The Hack Computer Platform



Animate: Program flow
View: Screen
Format: Decimal

ROM Asm

```
0   @0
1   D=M
2   @23
3   D;JLE
4   @16
5   M=D
6   @16384
7   D=A
8   @17
9   M=D
10  @17
11  A=M
12  M=-1
13  @17
14  D=M
15  @32
16  D=D+A
17  @17
18  M=D
19  @16
20  MD=M-1
21  @10
22  D;JGT
23  @23
24  0;JMP
25
26
27
28
```

RAM

| | |
|---|---|
| 0 | 50 |
| 1 | 0 |
| 2 | 0 |
| 3 | |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | |
| 11 | |
| 12 | |
| 13 | |
| 14 | |
| 15 | 0 |
| 16 | 25 |
| | 17184 |
| | |
| | 0 |
| | 0 |
| 21 | 0 |
| | 0 |
| 23 | 0 |
| 24 | 0 |
| 25 | 0 |
| 26 | 0 |
| 27 | 0 |
| 28 | 0 |

PC 15

A 17

D 17184

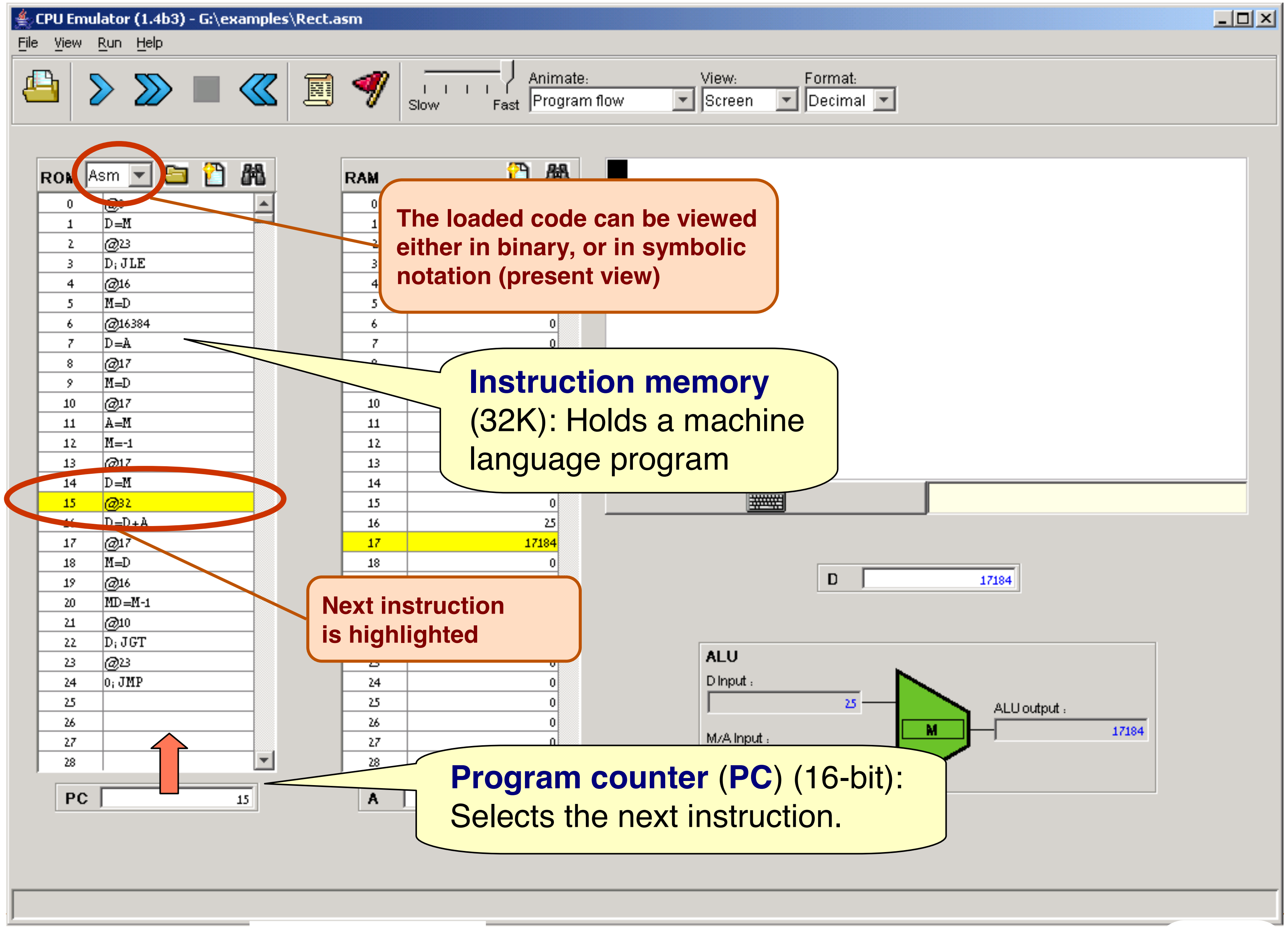
ALU

D Input : 25

M/A Input : 17184

ALU output : 17184

## Instruction memory



Animate: Program flow

View: Screen

Format: Decimal

ROM (Asm)

| | |
|---|---|
| 0 | @0 |
| 1 | D=M |
| 2 | @23 |
| 3 | D;JLE |
| 4 | @16 |
| 5 | M=D |
| 6 | @16384 |
| 7 | D=A |
| 8 | @17 |
| 9 | M=D |
| 10 | @17 |
| 11 | A=M |
| 12 | M=-1 |
| 13 | @17 |
| 14 | D=M |
| 15 | @32 |
| 16 | D=D+A |
| 17 | @17 |
| 18 | M=D |
| 19 | @16 |
| 20 | MD=M-1 |
| 21 | @10 |
| 22 | D;JGT |
| 23 | @23 |
| 24 | 0;JMP |
| 25 | |
| 26 | |
| 27 | |
| 28 | |

PC: 15

RAM

| | |
|---|---|
| 6 | 0 |
| 7 | 0 |
| 15 | 0 |
| 16 | 25 |
| 17 | 17184 |
| 18 | 0 |
| 24 | 0 |
| 25 | 0 |
| 26 | 0 |
| 27 | 0 |

D: 17184

ALU D Input: 25

ALU output: 17184

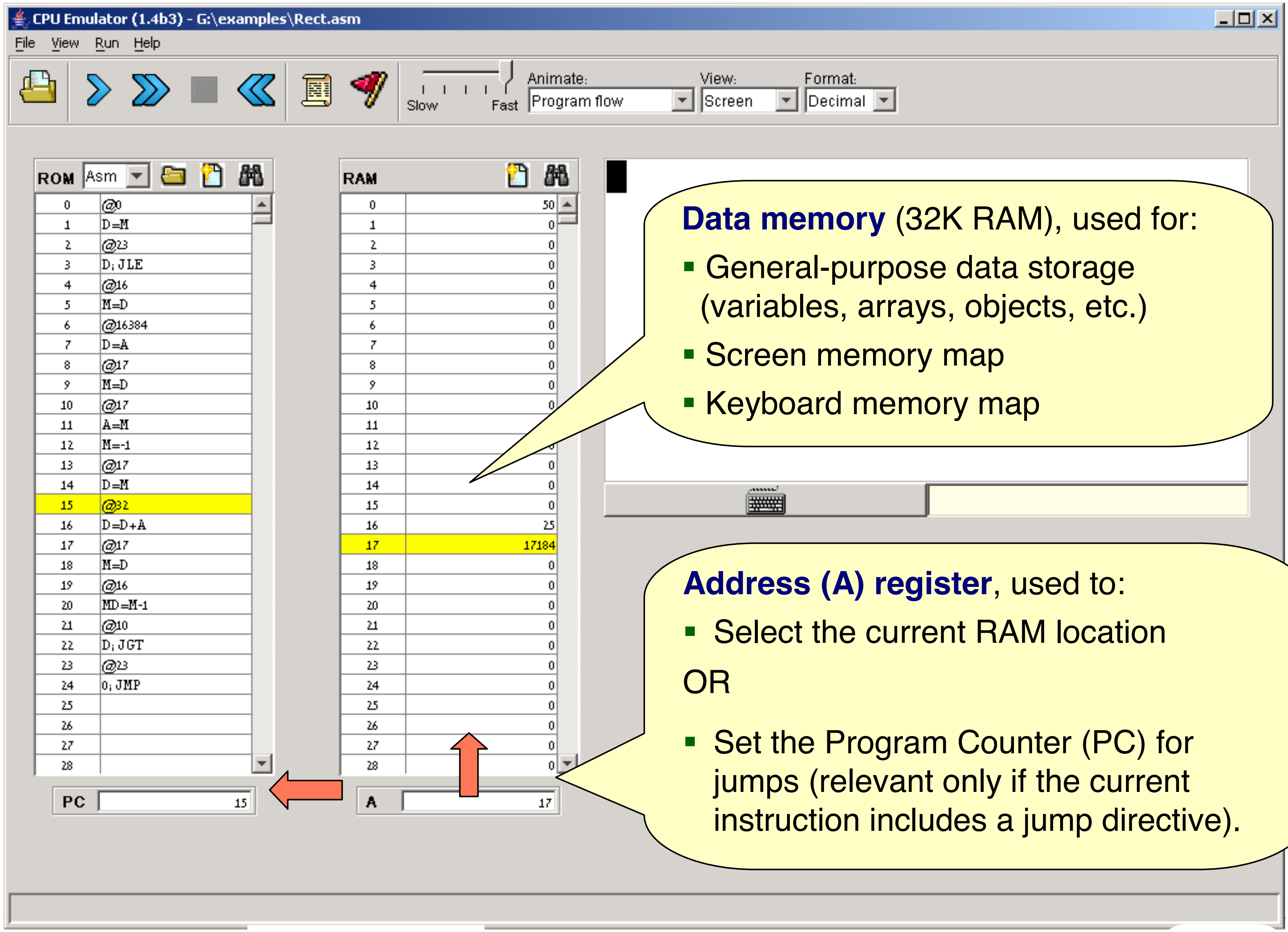
**The loaded code can be viewed either in binary, or in symbolic notation (present view)**

**Instruction memory** (32K): Holds a machine language program

**Next instruction is highlighted**

**Program counter** (**PC**) (16-bit): Selects the next instruction.

# Data memory (RAM)



Animate: Program flow  View: Screen  Format: Decimal

ROM (Asm)

| | |
|---|---|
| 0 | @0 |
| 1 | D=M |
| 2 | @23 |
| 3 | D;JLE |
| 4 | @16 |
| 5 | M=D |
| 6 | @16384 |
| 7 | D=A |
| 8 | @17 |
| 9 | M=D |
| 10 | @17 |
| 11 | A=M |
| 12 | M=-1 |
| 13 | @17 |
| 14 | D=M |
| 15 | @32 |
| 16 | D=D+A |
| 17 | @17 |
| 18 | M=D |
| 19 | @16 |
| 20 | MD=M-1 |
| 21 | @10 |
| 22 | D;JGT |
| 23 | @23 |
| 24 | 0;JMP |
| 25 | |
| 26 | |
| 27 | |
| 28 | |

RAM

| | |
|---|---|
| 0 | 50 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| 11 | |
| 12 | 0 |
| 13 | 0 |
| 14 | 0 |
| 15 | 0 |
| 16 | 25 |
| 17 | 17184 |
| 18 | 0 |
| 19 | 0 |
| 20 | 0 |
| 21 | 0 |
| 22 | 0 |
| 23 | 0 |
| 24 | 0 |
| 25 | 0 |
| 26 | 0 |
| 27 | 0 |
| 28 | 0 |

PC 15

A 17

**Data memory** (32K RAM), used for:

- General-purpose data storage (variables, arrays, objects, etc.)
- Screen memory map
- Keyboard memory map

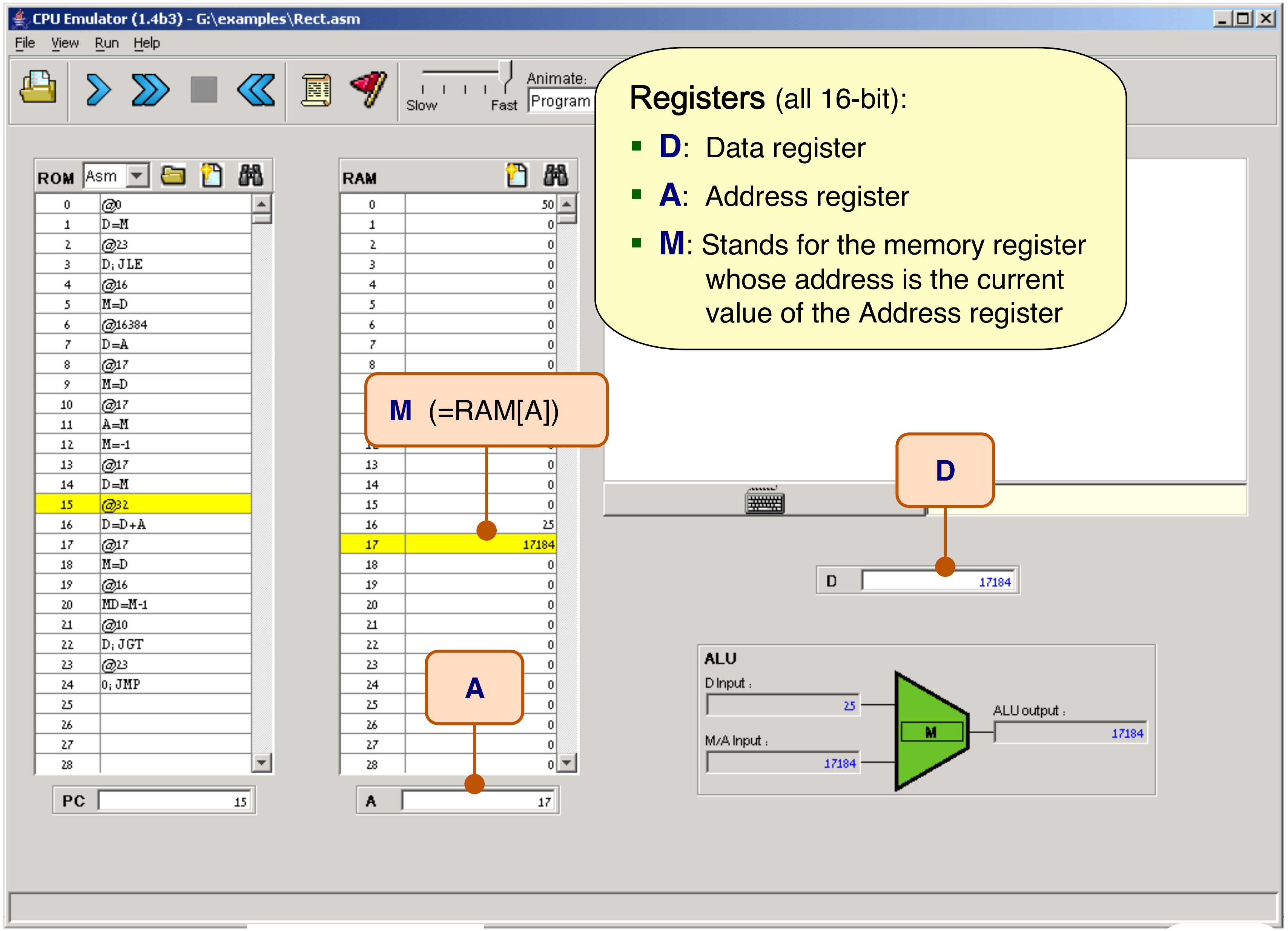
**Address (A) register**, used to:

- Select the current RAM location

OR

- Set the Program Counter (PC) for jumps (relevant only if the current instruction includes a jump directive).

# Registers



**Registers** (all 16-bit):

- **D**: Data register
- **A**: Address register
- **M**: Stands for the memory register whose address is the current value of the Address register

**M** (=RAM[A])

**D**

**A**

ROM

| | |
|---|---|
| 0 | @0 |
| 1 | D=M |
| 2 | @23 |
| 3 | D;JLE |
| 4 | @16 |
| 5 | M=D |
| 6 | @16384 |
| 7 | D=A |
| 8 | @17 |
| 9 | M=D |
| 10 | @17 |
| 11 | A=M |
| 12 | M=-1 |
| 13 | @17 |
| 14 | D=M |
| 15 | @32 |
| 16 | D=D+A |
| 17 | @17 |
| 18 | M=D |
| 19 | @16 |
| 20 | MD=M-1 |
| 21 | @10 |
| 22 | D;JGT |
| 23 | @23 |
| 24 | 0;JMP |
| 25 | |
| 26 | |
| 27 | |
| 28 | |

PC: 15

RAM

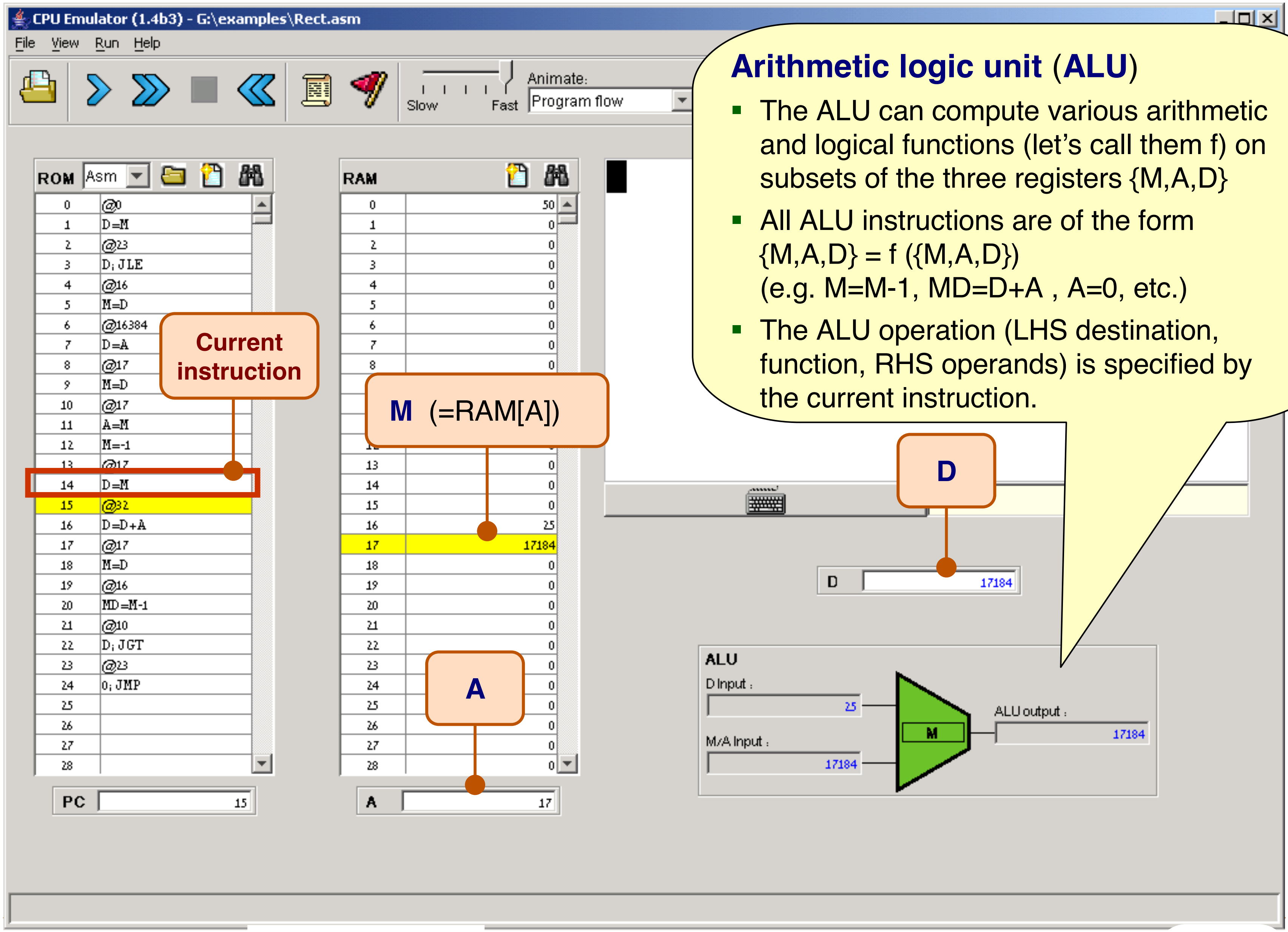
| | |
|---|---|
| 0 | 50 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 13 | 0 |
| 14 | 0 |
| 15 | 0 |
| 16 | 25 |
| 17 | 17184 |
| 18 | 0 |
| 19 | 0 |
| 20 | 0 |
| 21 | 0 |
| 22 | 0 |
| 23 | 0 |
| 24 | 0 |
| 25 | 0 |
| 26 | 0 |
| 27 | 0 |
| 28 | 0 |

A: 17

D: 17184

ALU
D Input : 25
M/A Input : 17184
ALU output : 17184

# Arithmetic/Logic Unit

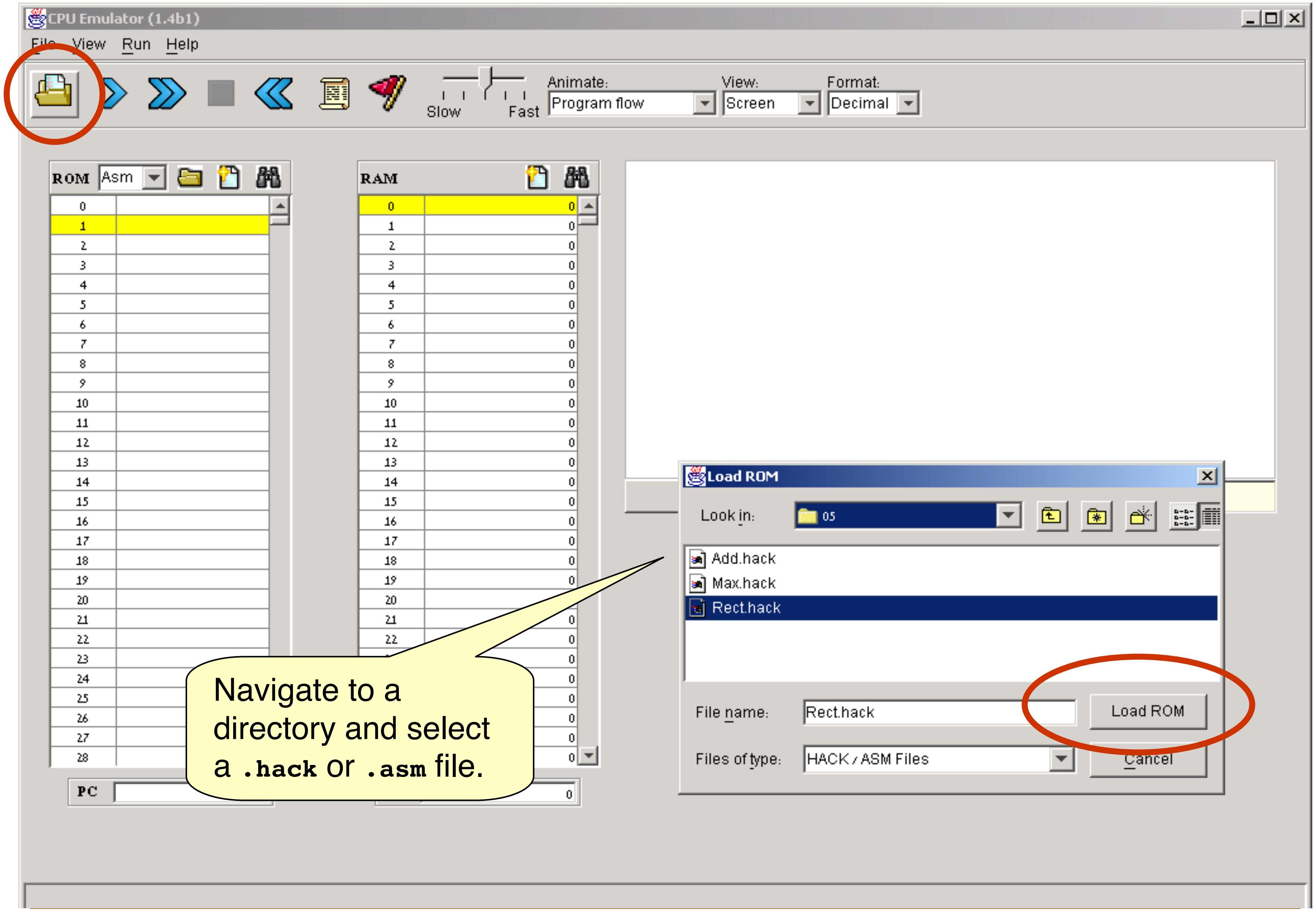


```
ROM                     RAM
0   @0                  0    50
1   D=M                 1     0
2   @23                 2     0
3   D;JLE               3     0
4   @16                 4     0
5   M=D                 5     0
6   @16384              6     0
7   D=A                 7     0
8   @17                 8     0
9   M=D
10  @17
11  A=M
12  M=-1
13  @17                 13    0
14  D=M                 14    0
15  @32                 15    0
16  D=D+A               16   25
17  @17                 17 17184
18  M=D                 18    0
19  @16                 19    0
20  MD=M-1              20    0
21  @10                 21    0
22  D;JGT               22    0
23  @23                 23    0
24  0;JMP               24    0
25                      25    0
26                      26    0
27                      27    0
28                      28    0

PC 15                   A 17

D 17184

ALU
D Input : 25
M/A Input : 17184
ALU output : 17184
```

**Arithmetic logic unit (ALU)**

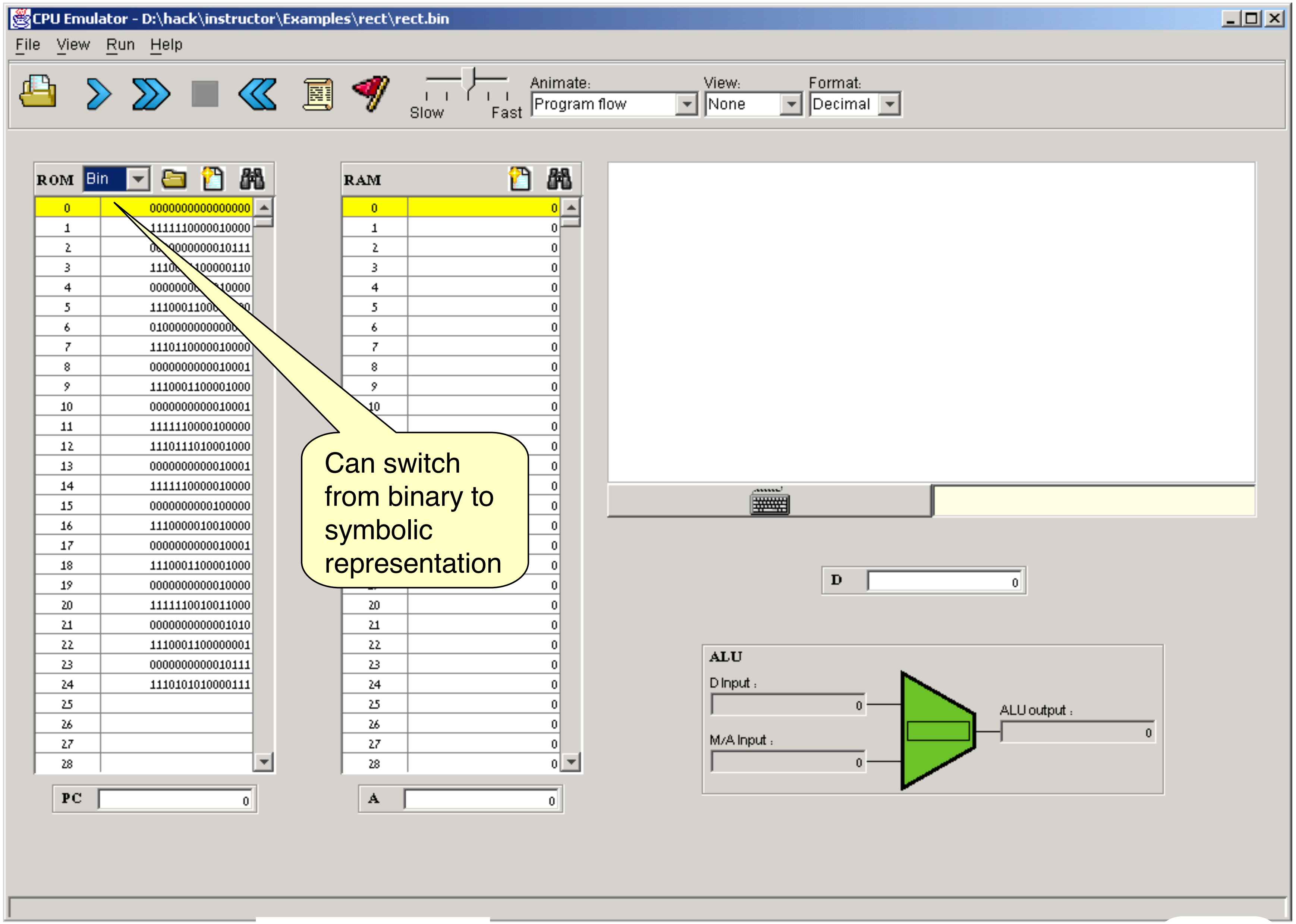
- The ALU can compute various arithmetic and logical functions (let's call them f) on subsets of the three registers {M,A,D}
- All ALU instructions are of the form {M,A,D} = f ({M,A,D}) (e.g. M=M-1, MD=D+A , A=0, etc.)
- The ALU operation (LHS destination, function, RHS operands) is specified by the current instruction.

## Loading a program



Navigate to a directory and select a **.hack** or **.asm** file.

# Loading a program



CPU Emulator - D:\hack\instructor\Examples\rect\rect.bin

Animate: Program flow
View: None
Format: Decimal

ROM Bin

| | |
|---|---|
| 0 | 0000000000000000 |
| 1 | 1111110000010000 |
| 2 | 0[illegible]000000010111 |
| 3 | 1110[illegible]100000110 |
| 4 | 00000000[illegible]10000 |
| 5 | 11100011000[illegible]0 |
| 6 | 01000000000000[illegible] |
| 7 | 1110110000010000 |
| 8 | 0000000000010001 |
| 9 | 1110001100001000 |
| 10 | 0000000000010001 |
| 11 | 1111110000100000 |
| 12 | 1110111010001000 |
| 13 | 0000000000010001 |
| 14 | 1111110000010000 |
| 15 | 0000000000100000 |
| 16 | 1110000010010000 |
| 17 | 0000000000010001 |
| 18 | 1110001100001000 |
| 19 | 0000000000010000 |
| 20 | 1111110010011000 |
| 21 | 0000000000001010 |
| 22 | 1110001100000001 |
| 23 | 0000000000010111 |
| 24 | 1110101010000111 |
| 25 | |
| 26 | |
| 27 | |
| 28 | |

PC 0

RAM

| | |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| [illegible] | 0 |
| [illegible] | 0 |
| [illegible] | 0 |
| [illegible] | 0 |
| [illegible] | 0 |
| [illegible] | 0 |
| [illegible] | 0 |
| [illegible] | 0 |
| [illegible] | 0 |
| 20 | 0 |
| 21 | 0 |
| 22 | 0 |
| 23 | 0 |
| 24 | 0 |
| 25 | 0 |
| 26 | 0 |
| 27 | 0 |
| 28 | 0 |

A 0

D 0

ALU

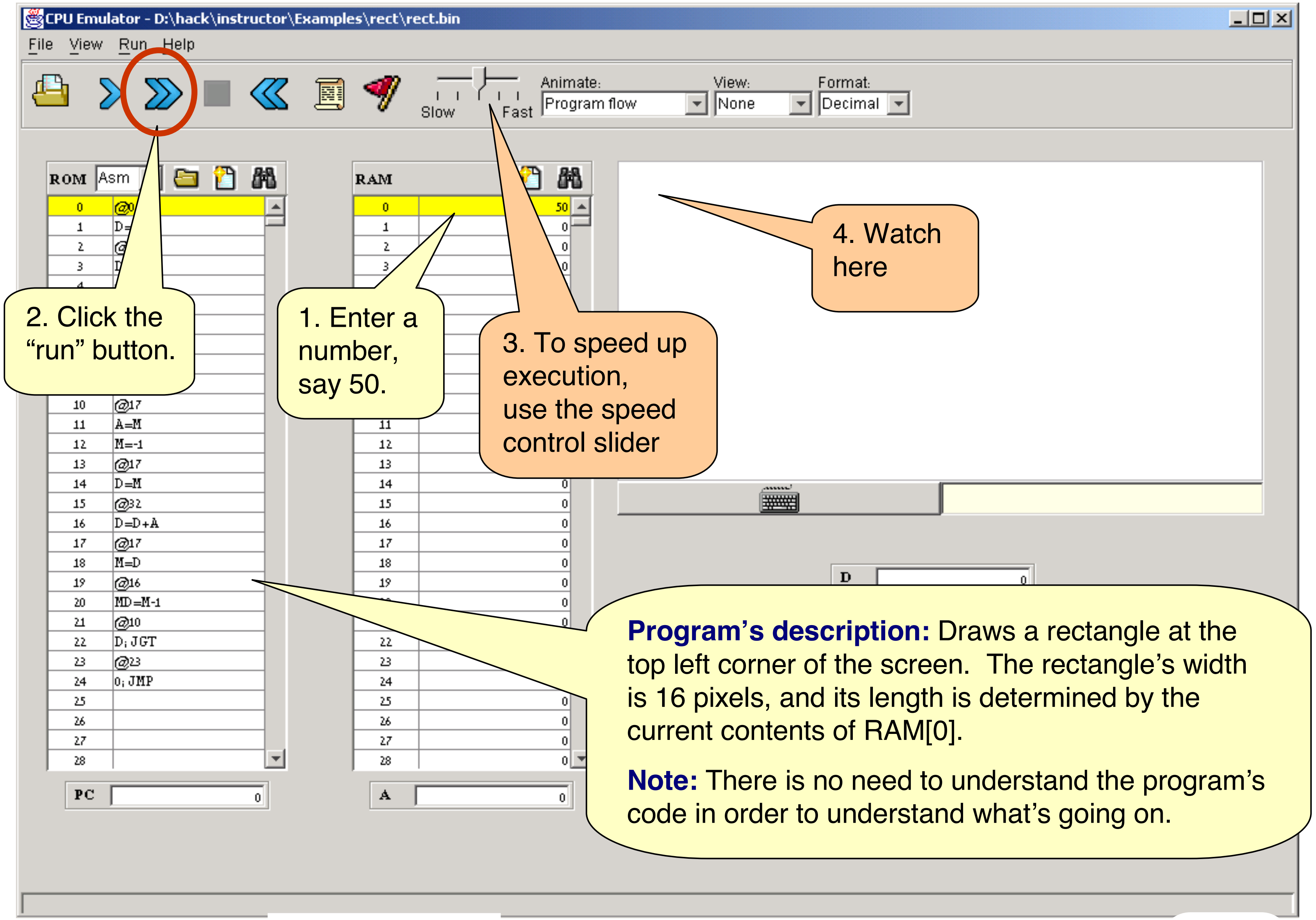
D Input : 0

M/A Input : 0

ALU output : 0

Can switch from binary to symbolic representation

## Running a program



```
ROM
0   @0
1   D=
2   @
3   D
10  @17
11  A=M
12  M=-1
13  @17
14  D=M
15  @32
16  D=D+A
17  @17
18  M=D
19  @16
20  MD=M-1
21  @10
22  D;JGT
23  @23
24  0;JMP
```

```
RAM
0   50
1   0
2   0
3   0
14  0
15  0
16  0
17  0
18  0
19  0
25  0
26  0
27  0
28  0
```

1. Enter a number, say 50.

2. Click the “run” button.

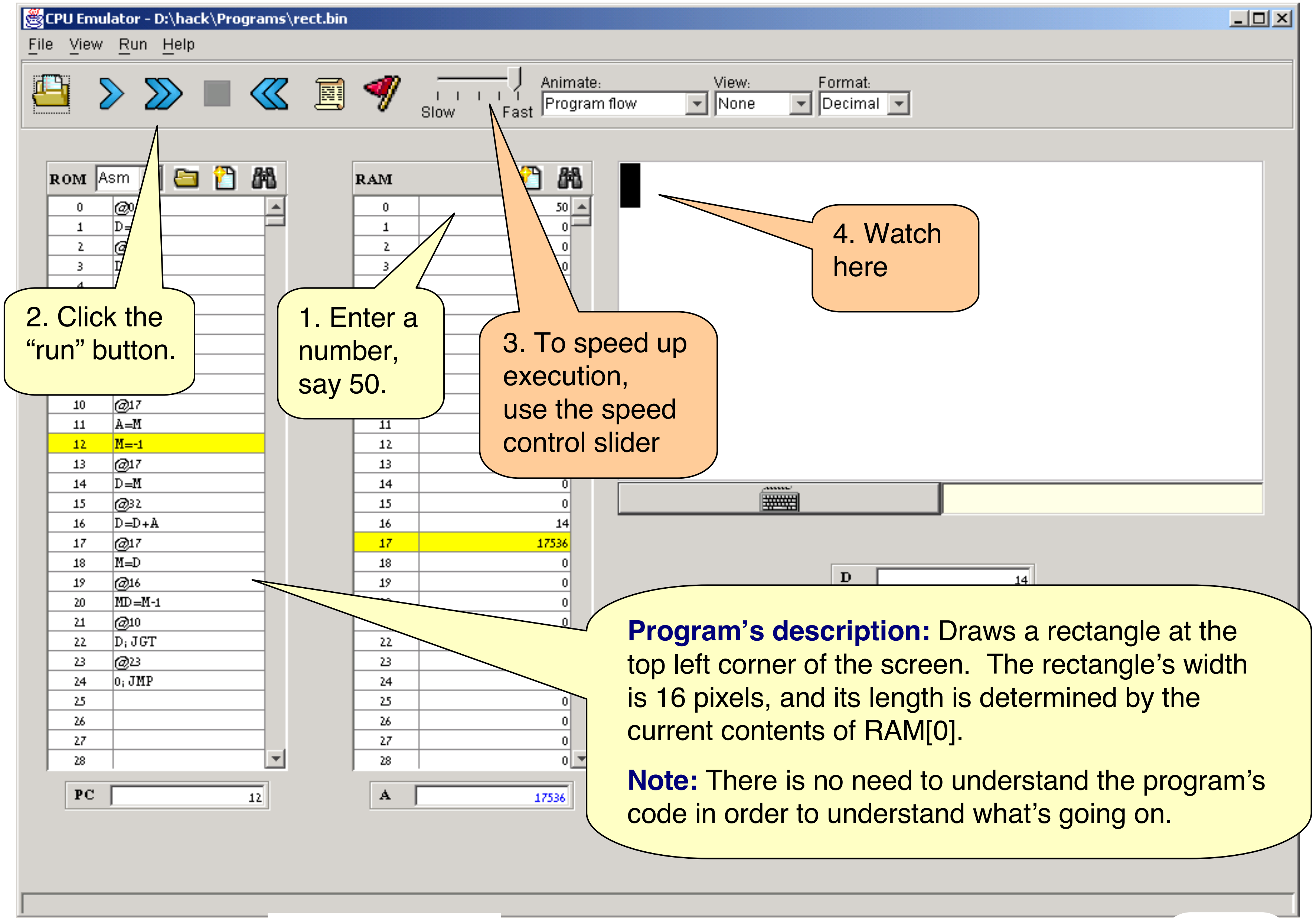
3. To speed up execution, use the speed control slider

4. Watch here

**Program’s description:** Draws a rectangle at the top left corner of the screen. The rectangle’s width is 16 pixels, and its length is determined by the current contents of RAM[0].

**Note:** There is no need to understand the program’s code in order to understand what’s going on.

# Running a program



Animate: Program flow
View: None
Format: Decimal

ROM:

```
0   @0
1   D=
2   @
3   D
10  @17
11  A=M
12  M=-1
13  @17
14  D=M
15  @32
16  D=D+A
17  @17
18  M=D
19  @16
20  MD=M-1
21  @10
22  D;JGT
23  @23
24  0;JMP
```

RAM:

```
0      50
1       0
2       0
3       0
14      0
15      0
16     14
17  17536
18      0
19      0
25      0
26      0
27      0
28      0
```

PC 12

A 17536

D 14

1. Enter a number, say 50.

2. Click the “run” button.

3. To speed up execution, use the speed control slider

4. Watch here

**Program’s description:** Draws a rectangle at the top left corner of the screen. The rectangle’s width is 16 pixels, and its length is determined by the current contents of RAM[0].

**Note:** There is no need to understand the program’s code in order to understand what’s going on.