Hack assembly III: Input and output
COMSM1302 Overview of Computer Architecture

John Lapinskas, University of Bristol

Input and output in Hack

In Hack, the only input peripheral is the keyboard and the only output peripheral is the monitor.

Input and output in Hack

In Hack, the only input peripheral is the keyboard and the only output peripheral is the monitor.

All input/output (**I/O**) is **memory-mapped**. This means on a hardware level, the CPU writes to the monitor the same way it writes to memory, and reads from the keyboard the same way it reads to memory.

This makes for easy assembly code!

Input and output in Hack

In Hack, the only input peripheral is the keyboard and the only output peripheral is the monitor.

All input/output (**I/O**) is **memory-mapped**. This means on a hardware level, the CPU writes to the monitor the same way it writes to memory, and reads from the keyboard the same way it reads to memory.

This makes for easy assembly code!

Recall Hack has 32KB of physical memory divided into 16-bit words, so an address can be held in $2^{18}/2^4 = 2^{14}$ bits: addresses 0x0000 to 0x3FFF.

Anything written to addresses 0x4000 to 0x5FFF will appear on screen. If a key is held on the keyboard, its value appears in 0x6000.

Keyboard input

32:	space	56:	8	80:	P	104:	h	127:	DEL
33:	!	57:	9	81:	Q	105:	i	128:	newLine
34:	"	58:	:	82:	R	106:	j	129:	backSpace
35:	#	59:	;	83:	S	107:	k	130:	leftArrow
36:	$	60:	<	84:	T	108:	l	131:	upArrow
37:	%	61:	=	85:	U	109:	m	132:	rightArrow
38:	&	62:	>	86:	V	110:	n	133:	downArrow
39:	'	63:	?	87:	W	111:	o	134:	home
40:	(	64:	@	88:	X	112:	p	135:	end
41:	)	65:	A	89:	Y	113:	q	136:	pageUp
42:	*	66:	B	90:	Z	114:	r	137:	pageDown
43:	+	67:	C	91:	[	115:	s	138:	insert
44:	,	68:	D	92:	/	116:	t	139:	delete
45:	-	69:	E	93:	]	117:	u	140:	esc
46:	.	70:	F	94:	^	118:	v	141:	f1
47:	/	71:	G	95:	_	119:	w	142:	f2
48:	0	72:	H	96:	`	120:	x	143:	f3
49:	1	73:	I	97:	a	121:	y	144:	f4
50:	2	74:	J	98:	b	122:	z	145:	f5
51:	3	75:	K	99:	c	123:	{	146:	f6
52:	4	76:	L	100:	d	124:	|	147:	f7
53:	5	77:	M	101:	e	125:	}	148:	f8
54:	6	78:	N	102:	f	126:	~	149:	f9
55:	7	79:	O	103:	g			150:	f10
								151:	f11
								152:	f12

Source: Nisan and Schocken Appendix 5

The keyword `KBD` is mapped to 0x6000 (= 24576) in the same way that e.g. `R1` is mapped to 0x0001. For example, if “d” is being held, then `@KBD` will load 24576 into A and 100 into M.

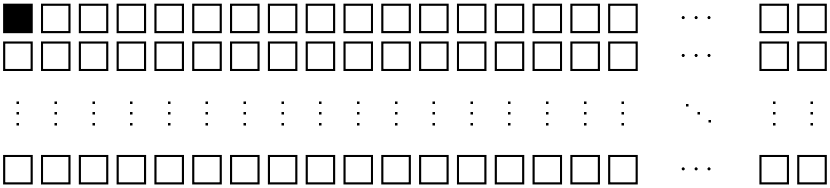
If no key is being pressed, then 0x6000 contains 0.

Warning: There’s no way to detect more than one key being pressed at the same time. (Modern keyboards have trouble with this too!)

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



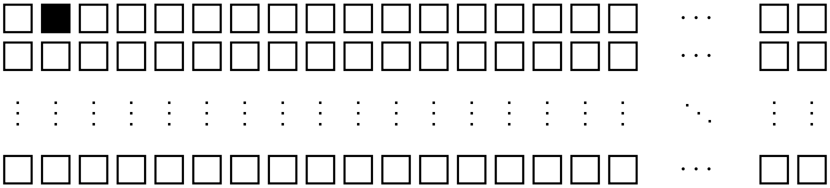
Pixel number 1 RAM[0x4000] Contents 000000000000000**1**

The i'th pixel displays black if the i'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":

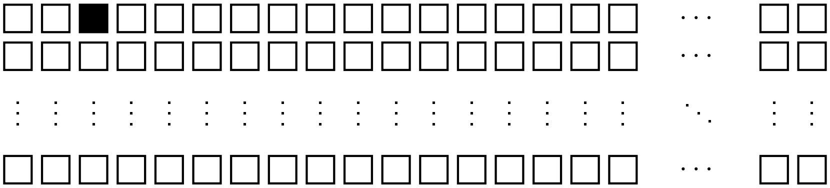


Pixel number 2   RAM[0x4000] Contents 00000000000000**1**0

The *i*'th pixel displays black if the *i*'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:



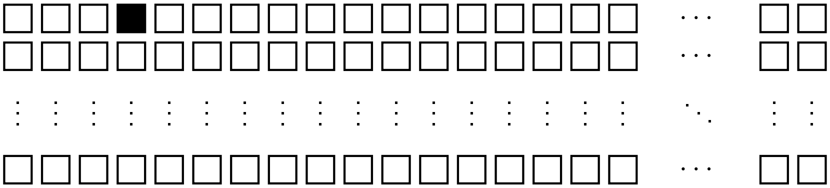
Pixel number 3 RAM[0x4000] Contents 0000000000000**1**00

The *i*’th pixel displays black if the *i*’th bit in memory counting from the lsb of address 0x4000 is 1, and white if it’s 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:



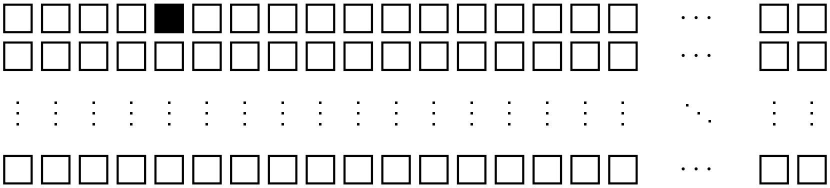
Pixel number 4   RAM[0x4000] Contents 0000000000001000

The *i*’th pixel displays black if the *i*’th bit in memory counting from the lsb of address 0x4000 is 1, and white if it’s 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:



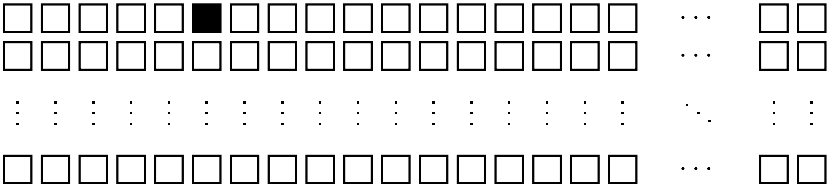
Pixel number 5 RAM[0x4000] Contents 0000000000010000

The i'th pixel displays black if the i'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



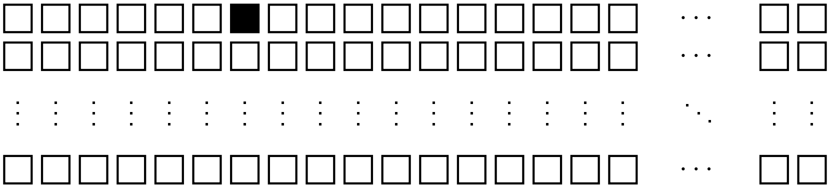
Pixel number 6 RAM[0x4000] Contents 0000000000**1**00000

The i'th pixel displays black if the i'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:

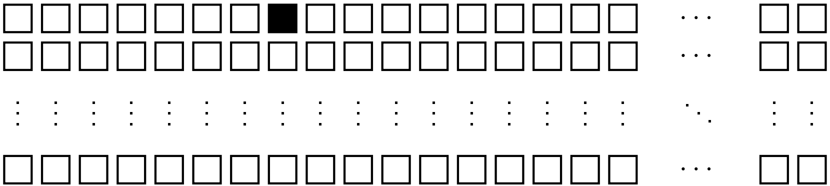


Pixel number 7 RAM[0x4000] Contents 000000000**1**000000

The i’th pixel displays black if the i’th bit in memory counting from the lsb of address 0x4000 is 1, and white if it’s 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



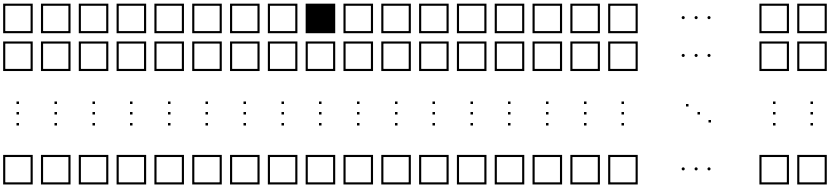
Pixel number 8 RAM[0x4000] Contents 0000000010000000

The i'th pixel displays black if the i'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



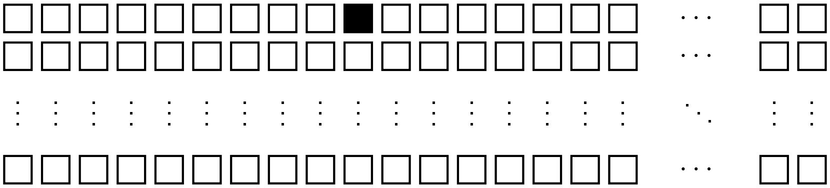
Pixel number 9 RAM[0x4000] Contents 0000000**1**00000000

The *i*'th pixel displays black if the *i*'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:



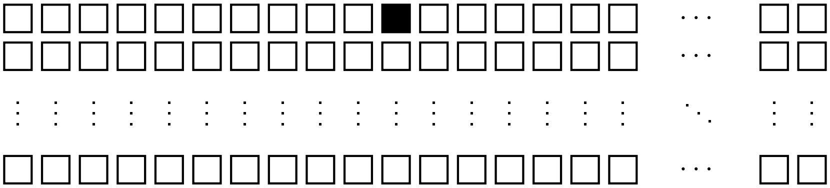
Pixel number 10 RAM[0x4000] Contents 000000**1**000000000

The i’th pixel displays black if the i’th bit in memory counting from the lsb of address 0x4000 is 1, and white if it’s 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:



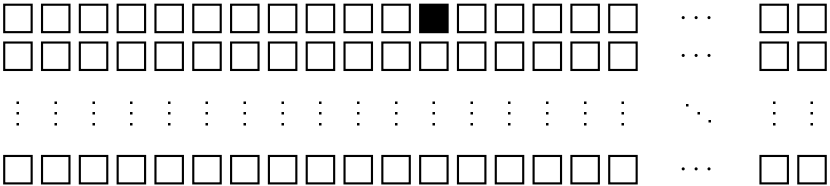
Pixel number 11 RAM[0x4000] Contents 0000010000000000

The *i*’th pixel displays black if the *i*’th bit in memory counting from the lsb of address 0x4000 is 1, and white if it’s 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



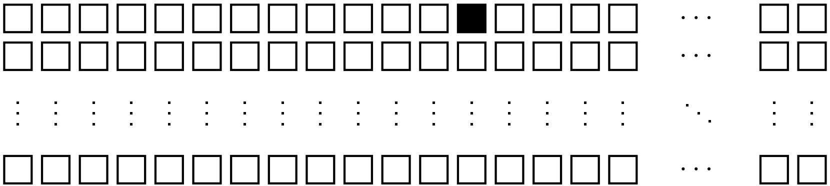
Pixel number 12 RAM[0x4000] Contents 0000**1**00000000000

The *i*'th pixel displays black if the *i*'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



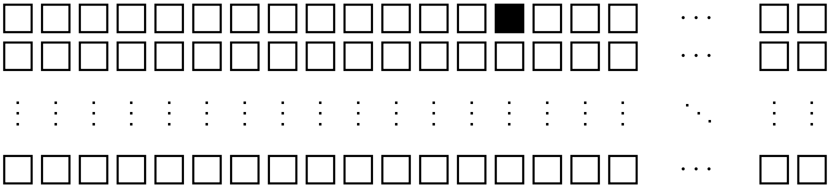
Pixel number 13   RAM[0x4000] Contents 000**1**000000000000

The *i*'th pixel displays black if the *i*'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:

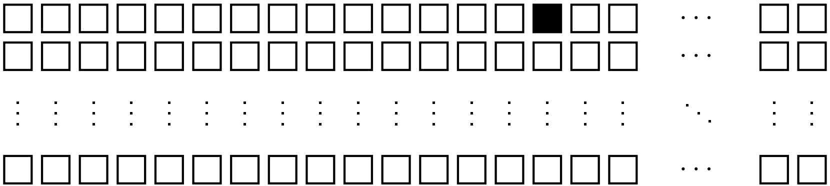


Pixel number 14 RAM[0x4000] Contents 00**1**0000000000000

The *i*’th pixel displays black if the *i*’th bit in memory counting from the lsb of address 0x4000 is 1, and white if it’s 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:



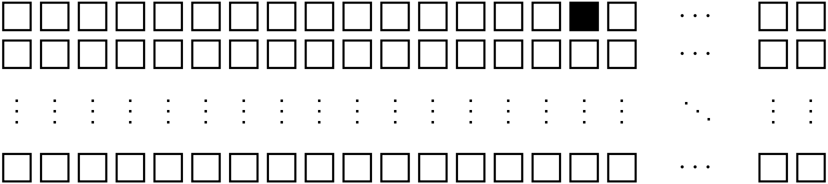
Pixel number 15   RAM[0x4000] Contents 0**1**00000000000000

The *i*'th pixel displays black if the *i*'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



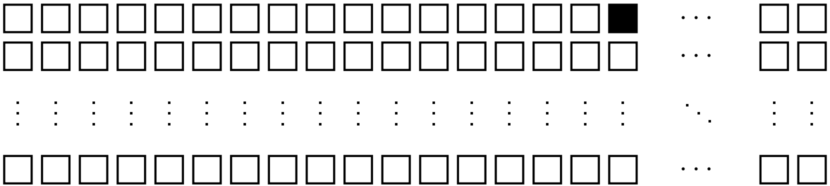
Pixel number 16 RAM[0x4000] Contents **1**000000000000000

The *i*'th pixel displays black if the *i*'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



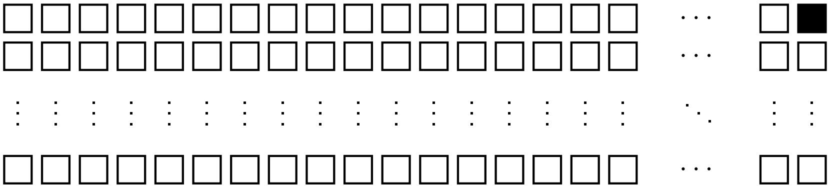
Pixel number 17 RAM[0x400**1**] Contents 000000000000000**1**

The *i*'th pixel displays black if the *i*'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



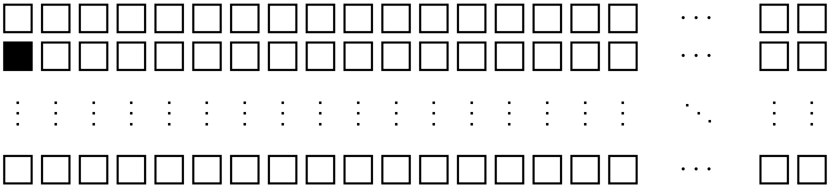
Pixel number 512 RAM[0x40**1F**] Contents **1**000000000000000

The *i*'th pixel displays black if the *i*'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



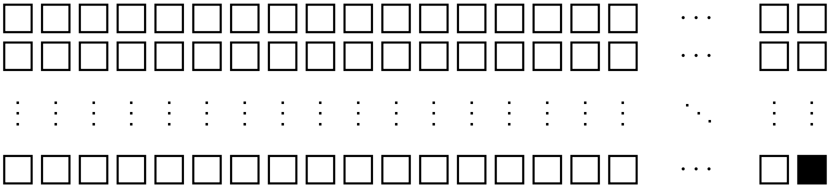
Pixel number 513 RAM[0x40**20**] Contents 000000000000000**1**

The i'th pixel displays black if the i'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in "book order":



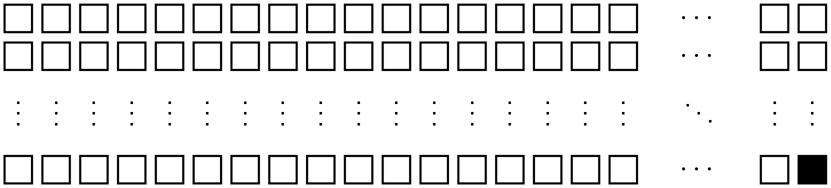
Pixel number 8192   RAM[0x**5FFF**] Contents **1**000000000000000

The i'th pixel displays black if the i'th bit in memory counting from the lsb of address 0x4000 is 1, and white if it's 0. So each word in 0x4000–0x5FFF controls not one pixel, but 16!

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:



Pixel number 8192 RAM[0x5FFF] Contents 1000000000000000

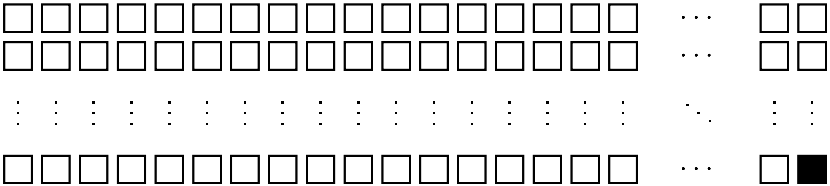
Equivalently, the pixel at row r from the top and column c from the left (both counting from zero) is controlled by the $(c \% 16)$'th bit **from the right** at address $0x4000 + 32r + (c/16)$ (with integer division).

Here $r = 255$ and $c = 511$.

Monitor output

Hack works in a **resolution** of 512x256, i.e. 256 rows of 512 pixels per row.

Pixels are numbered from left to right and top to bottom in “book order”:



Pixel number 8192 RAM[0x5FFF] Contents 1000000000000000

Another way of writing the address you need is

$$010\underbrace{11111111}_{r\text{ in binary}}\overbrace{11111}^{\text{first 5 bits of }c\text{ in binary}}.$$

Exercise: Why are these all equivalent?

Tricks and traps

Warning: The `@` command only works on values of up to 15 bits!

Tricks and traps

Warning: The `@` command only works on values of up to 15 bits!

If you want to e.g. colour the first 16 pixels black, i.e. write 0xFFFF into address 0x4000, then `@65535` won't load 0xFFFF into A. You'll instead need to write e.g. `@0` followed by `A=!A`.

Tricks and traps

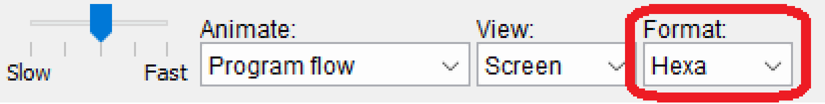
Warning: The `@` command only works on values of up to 15 bits!

If you want to e.g. colour the first 16 pixels black, i.e. write 0xFFFF into address 0x4000, then `@65535` won't load 0xFFFF into A. You'll instead need to write e.g. `@0` followed by `A=!A`. Or just `A=-1`.

Tricks and traps

Warning: The `@` command only works on values of up to 15 bits!

If you want to e.g. colour the first 16 pixels black, i.e. write 0xFFFF into address 0x4000, then `@65535` won't load 0xFFFF into A. You'll instead need to write e.g. `@0` followed by `A=!A`. Or just `A=-1`.

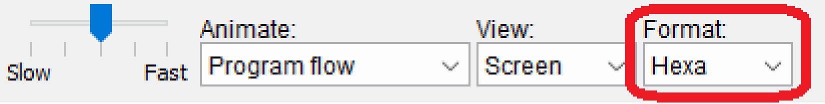


To help with this, by changing this setting in the CPU emulator, you can see register and memory values in hex or binary (rather than decimal).

Tricks and traps

Warning: The `@` command only works on values of up to 15 bits!

If you want to e.g. colour the first 16 pixels black, i.e. write 0xFFFF into address 0x4000, then `@65535` won't load 0xFFFF into *A*. You'll instead need to write e.g. `@0` followed by `A=!A`. Or just `A=-1`.



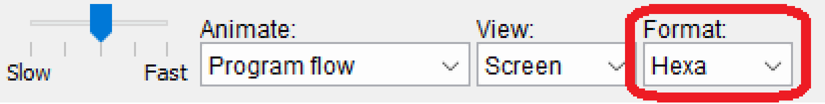
To help with this, by changing this setting in the CPU emulator, you can see register and memory values in hex or binary (rather than decimal).

The keyword `SCREEN` is mapped to 0x4000. For example, `@SCREEN` followed by `M=1` would colour the first pixel black.

Tricks and traps

Warning: The `@` command only works on values of up to 15 bits!

If you want to e.g. colour the first 16 pixels black, i.e. write 0xFFFF into address 0x4000, then `@65535` won't load 0xFFFF into *A*. You'll instead need to write e.g. `@0` followed by `A=!A`. Or just `A=-1`.



To help with this, by changing this setting in the CPU emulator, you can see register and memory values in hex or binary (rather than decimal).

The keyword `SCREEN` is mapped to 0x4000. For example, `@SCREEN` followed by `M=1` would colour the first pixel black.

You can read from the screen as well as writing to it! E.g. if the top-left 16 pixels are all black, then `@SCREEN`, e.g. `D=M` will store 0xFFFF in *D*.

Example: Filling the screen

Fill.asm fills every pixel of the screen black. While any key is held, the screen is instead filled white.

[See video for live coding and explanation.]