machine language	assembly	x16joy	meaning	example
0000 0000 0000 0000	end	0	finish running code	end
0001 vvvv vvvv rrrr	sv_ r v	4096	reg[r] = v	sv_ r12 22
0100 AAAA rrrr ssss	ALU r s	16384	reg[r] = reg[r] ALU reg[s]	add r13 r10
1000 xxxx rrrr ssss	ld_ r s	32768	reg[r] = RAM[reg[s]]	ld r2 r10
1010 xxxx rrrr ssss	st_ r s	40960	RAM[reg[s]] = reg[r]	st r2 r10
1100 xxxx xxxx rrrr	jp_ r	49152	clc = reg[r]	jp r12
1111 xxxx cccc rrrr	jpcc r	61440	if (c) -> clc = reg[r]	jpns r1
0010		8192		
0011		12288		
0101 xxxx xxxx xxxx	co_	20480	clear output (pixel grid)	co_
0110 xxxx rrrr ssss	dp_ r s	24576	if reg[r] > 0 draw px(reg[s])	dp_ r1, r2
0111 rrgg bbaa ssss	dpc rg ba s	28672	draw px reg[rg] reg[ba] reg[s]	dp_ r2, r3, r4
1001 xxxx xxxx rrrr	prc r	36864	print char reg[r]	prc r10
1011 xxxm mmmm rrrr	prv r m	45056	print int (m=num system)	prv r10 _16
1101		53248		
1110 xxxx xxxx rrrr	gik r	57344	Reg[r] = last event.key	gik r10

The following notation is used:

- vvvv vvvv is an 8 bit value that is embedded into the instruction. This is also called an ‘immediate value’ as it’s immediately available from the instruction.
- X stands for “don’t care.” These bits can be anything
- rrrr, ssss, rrgg, bbaa stand for register numbers (0-15)
- AAAA stands for ALU operation. The following operations are supported

ALU			
bits	instruction name	description	operation
0000	NOP	No OPeration	A = B
0001	OR	bitwise OR	A |= B
0010	AND	bitwise AND	A &= B
0011	XOR	bitwise XOR	A ^= B
0100	ADD	addition	A = A + B
0101	SUB	subtraction	A = A – B
0110	LSL	Logical Shift Left	A = A << B
0111	LSR	Logical Shift Right	A = A >> B
1000	ASR	Arithmetic Shift Right	A = A >>> B
1001	NOT	logic inverse (1-comp)	A = ~B
1010	NEG	Negative (2-comp	A = -B
1011	MUL	multiplication	A = A * B
1100	DIV	division	A = A / B
1101	POW	exponent	A = A ^ B
1110	???		
1111	???		

- cccc stands for condition. Every time a calculation is done by the ALU, multiple flags are set according to the result. The following conditions can be checked:

conditions			
bits	condition short	condition long	meaning
0000	c	carry	carried bit was 1
0001	nc	no-carry	no carried bit

0010	z	zero	result = 0
0011	nz	not-zero	result != 0
0100	S	sign	result < 0
0101	ns	no-sign	result >=0
0110	o	overflow	result > 16bit max
0111	no	no-overflow	result <= 16bit max

prv – print value - Prints value in the given register to terminal/console in provided numeral system. Examples: [:] prv r10 _16 [:] prints the value inside of register 10 in hexadecimal, or base 16 format. [:] prv r6 _10 [:] prints the value inside of register 6 in decimal, or base 10 format.

prc – print char - Prints char matching the value inside the given register. The char is printed in the terminal/console without any whitespace or new lines characters attached. Characters use binary to ASCII char conversion. Decimal 97 or hexadecimal 0x61 would be equal to char 'a', 98 to 'b' and so on.

dp_ - draw pixel - Draws pixel on output 16x16 canvas. Pixel can either be black or white. Register 1 gives the color value. If register 1 value is greater than 0, set color to max rgb(white). If register 1 is equal or less than 0, set color to min rgb(black). The value of the second register acts as coordinates. The 4 least significant bits act as the y coordinate while the next 4 more significant bits act as the x coordinate. Register value is as follows: 0000 0000 xxxx yyyy. Coordinates are from 0 to 15. This results in a 16x16 grid of pixels.

dpc – draw pixel (with) color - Draws pixel on output 16x16 canvas. Pixel use rgba values stores in register 1 and 2. Register 1 gives the red and green color values. (rrrr rrrr gggg gggg). Register 2 gives the blue and alpha values. (bbbb bbbb aaaa aaaa). The value of the third register acts as coordinates. The 4 least significant bits act as the y coordinate while the next 4 more significant bits act as the x coordinate. Register value is as follows: 0000 0000 xxxx yyyy. Coordinates are from 0 to 15. This results in a 16x16 grid of pixels.

co_ - clear output - Sets all pixels on the 16x16 output grid to rgba(0,0,0,0)

gik - get input key - Loads the value of the event.key of the last held button (and currently held) and stores it in the specified register. If you press and hold a button, that event key will be stored in the special input register. When you let go of a button, that event key will be cleared. If you press another button while still holding the old one, the new key will override the old. When that happens, the input register will be cleared only when you let go of the *new* key.