/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

devernay.free.fr/hacks/chip8/C8TECH10.HTM#0.0

chip-8 roms:

test1.bin

dos>cd C:\Users\Alifa\Desktop\lc3-backup\eclipse\LC3Compiler\ant

dos>1.init.bat

dos>2.compile.bat src\chip8\1-chip8emu.lct

dos>3.run.bat out\1-chip8emu.lct\_asm.txt out\1-chip8emu.lct\_asm.obj hex

dos>3.run.bat ... > C:\Users\Alifa\Desktop\lc3-backup\eclipse\LC3Chip8Emulator\lc3-obj\chip8out\out.txt

Notes about lea ld and st:

lea x y

put address of y into x

i.e. x = address of y

ld x y

put mem[y] into x

i.e. x = mem[y]

st x y

put y into mem[x]

i.e mem[x] = y

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

set global variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

global {

public gRetParam = 0

public gXPassParam = 0

public gYPassParam = 0

public gLenPassParam = 0

public gHeiPassParam = 0

public gDebug = 0

public gVRamAddr = 0

public gVRamSize = 0

public gVRamWidth = 0

public gVRamHeight = 0

public gVRamScale = 15

public gORXORAddr = 0xff09

public gORVal = 0x5a

public gXORVal = 0x5b

public gData1Addr = 0xff07

public gData2Addr = 0xff08

public gSleepAddr = 0xff0a

public gReset = 0

public gFontSetAddr = 0

}

function main {

/\*\*\*\*\*\*

chip-8

\*\*\*\*\*\*\*/

let filename = "test2.bin"

let outfile = "memory.bin"

let START\_ADDRESS = 0x200

let FONTSET\_START\_ADDRESS = 0x50

setpublic gFontSetAddr FONTSET\_START\_ADDRESS

let memsize = 0xfff

uninitarray mem memsize

uninitarray reg 16

uninitarray stack 16

let regStart = 0

let stackStart = 0

lea regStart reg

lea stackStart stack

let flag = 15

let COLS = 64

setpublic gVRamWidth COLS

let ROWS = 32

setpublic gVRamHeight ROWS

let SCALE = 0

getpublic SCALE gVRamScale

let WIDTH = 0

// vramSize is COLS \* ROWS

let vramSize = 0x800

setpublic gVRamSize vramSize

WIDTH = COLS

WIDTH \* SCALE

let HEIGHT = 0

HEIGHT = ROWS

HEIGHT \* COLS

uninitarray vram vramSize

let vramtemp = 0

lea vramtemp vram

setpublic gVRamAddr vramtemp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

There is also a 16-bit register

called I. This register is

generally used to store memory

addresses:

let ir = 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

let ir = 0

let sp = 0

let Rr0 = 0

let Rr1 = 0

lea Rr0 filename

lea Rr1 mem

param0 = Rr0

param1 = Rr1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I'm using 'RES' (reserved) opcode 0xD (13)

to load a (chip-8 rom) program who's file

name starts at reg r0 (index). Reg r1 stores

the location/address WHERE the program will

be loaded to, plus offset (r3). Register 2

will then store the size (in bytes)

of the loaded program (chip 8 rom).

E.g res r0 r1 r3 IN

lc3.vm.chip.CPU class:

map.put(Vars.OP\_RES, () -> chip8prog.resLdPrg());

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

<asm> ld r3 START\_ADDRESS-main </asm>

<asm> res r0 r1 r3 IN </asm>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

register 2 stores the file size

in bytes. See lc3.vm.datatransfer.ResLoadProg

class. Need to call getparam2 immediately after

rom is loaded otherwise r2 is used for other

operations like add.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

let filesize = 0

filesize = getparam2

let pc = 0

pc = Rr1

FONTSET\_START\_ADDRESS + Rr1

pc + START\_ADDRESS

/\*\*\* String variables for debugging output \*\*\*/

let opcodeSTR = " | opcode = "

let space = " "

let regOpen = " reg[ "

let ldOpen = " LD[ "

let brackClose = " ]"

let unimplemented = " UNIMPLEMENTED CODE"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Declare more variables

var opcode = instruction >> 12;

var r1 = (instruction >> 8) & 0xf;

var r2 = (instruction >> 4) & 0xf;

var lowbyte = instruction & 0xff;

var last4bits = instruction & 0xf;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

let delayTimer = 0

let soundTimer = 0

let HI = 0

let LO = 0

let instruction = 0

let opcode = 0

let r1 = 0

let r2 = 0

let lowbyte = 0

let last4bits = 0

let sleep = 500

let debug = 0

getpublic debug gDebug

let sleepAddr = 0

getpublic sleepAddr gSleepAddr

let bitwiseORXORAddr = 0xff09

let bitwiseORVal = 0x5a

let data1Addr = 0xff07

let data2Addr = 0xff08

let rShiftBy12 = 12

let rShiftBy8 = 8

let rShiftBy4 = 4

let retVal12 = 0

let retVal8 = 0

let retVal4 = 0

let lShiftby7 = 7

let retVal7 = 0

let maskbyte = 0xff

let masknibble = 0xf

let caseE0 = 0xE0

let caseEE = 0xEE

let case9E = 0x9E

let case1A = 0x1A

let case1E = 0x1E

let case15 = 0x15

let case29 = 0x29

let case33 = 0x33

let case55 = 0x55

let case65 = 0x65

let caseA1 = 0xA1

let byte80 = 0x80

let counter = 0

let tempvarA = 0

let tempvarB = 0

let tempvarC = 0

let tempvarD = 0

let tempvarE = 0

let tempvarF = 0

let tempvarG = 0

let tempvarH = 0

let tempvarI = 0

let tempvarJ = 0

let tempvarK = 0

let tempvarL = 0

let tempvarM = 0

let tempvarN = 0

let tempvarO = 0

let tempvarP = 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

fontset = [ ... ];

Load fontset starting at

index of mem[] + FONTSET\_START\_ADDRESS

WARNING: Do not leave space after

comment. //1 is ok. // 1 not ok!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

<fillintarray> fontset

0xF0, 0x90, 0x90, 0x90, 0xF0, //0

0x20, 0x60, 0x20, 0x20, 0x70, //1

0xF0, 0x10, 0xF0, 0x80, 0xF0, //2

0xF0, 0x10, 0xF0, 0x10, 0xF0, //3

0x90, 0x90, 0xF0, 0x10, 0x10, //4

0xF0, 0x80, 0xF0, 0x10, 0xF0, //5

0xF0, 0x80, 0xF0, 0x90, 0xF0, //6

0xF0, 0x10, 0x20, 0x40, 0x40, //7

0xF0, 0x90, 0xF0, 0x90, 0xF0, //8

0xF0, 0x90, 0xF0, 0x10, 0xF0, //9

0xF0, 0x90, 0xF0, 0x90, 0x90, //A

0xE0, 0x90, 0xE0, 0x90, 0xE0, //B

0xF0, 0x80, 0x80, 0x80, 0xF0, //C

0xE0, 0x90, 0x90, 0x90, 0xE0, //D

0xF0, 0x80, 0xF0, 0x80, 0xF0, //E

0xF0, 0x80, 0xF0, 0x80, 0x80 //F

</fillintarray>

let fonsetsize = 0

let fonsetaddress = 0

lea fonsetaddress fontset

ld fonsetsize fonsetaddress

let i = 0

let val = 0

do

fonsetaddress + 1

ld val fonsetaddress

st FONTSET\_START\_ADDRESS val

FONTSET\_START\_ADDRESS + 1

i + 1

loop i < fonsetsize

//...then Print to make sure it's correct:

if debug == 1 {

let tempFS = 0x50

i = 0

val = 0

tempFS + Rr1

do

ld val tempFS

print val

print " "

i + 1

tempFS + 1

loop i < fonsetsize

println ""

}

else

endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Init LCD when debug = 0

Order of calls matter!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if debug == 0 {

setupVramForLCD

lcdInit

lcdTest

let sleepfor2seconds = 2000

st sleepAddr sleepfor2seconds

clearVRam

}

else

endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Set shift bits by

calling 2ToPower

function each time

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

setpublic gXPassParam rShiftBy12

2ToPower

getpublic retVal12 gRetParam

setpublic gXPassParam rShiftBy8

2ToPower

getpublic retVal8 gRetParam

setpublic gXPassParam rShiftBy4

2ToPower

getpublic retVal4 gRetParam

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main loop starts here

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

label mainloop {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Remeber that getparam2

stores the xQuotient

value after a divide

operation (which I use

for bit shifting here).

getparam2 also stores the

filesize after a load rom

operation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ld HI pc

pc + 1

HI & maskbyte

setpublic gXPassParam rShiftBy8

2ToPower

getpublic retVal8 gRetParam

HI \* retVal8

ld LO pc

LO & maskbyte

st data1Addr HI

st data2Addr LO

st bitwiseORXORAddr bitwiseORVal

ld HI data1Addr

pc + 1

//opcode

instruction = HI

instruction / retVal12

opcode = getparam2

//r1

instruction = HI

instruction / retVal8

r1 = getparam2

r1 & masknibble

//r2

instruction = HI

instruction / retVal4

r2 = getparam2

r2 & masknibble

//lowbyte

instruction = HI

instruction & maskbyte

lowbyte = instruction

//last4bits - lower nibble

instruction = HI

instruction & masknibble

last4bits = instruction

instruction = HI

if debug == 1 {

print counter

print " - "

print instruction

print ": "

print opcode

print space

print r1

print space

print r2

print space

print lowbyte

print space

print last4bits

}

else

endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

call sleep function

if needed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//st sleepAddr sleep

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

switch case conditions start...

Note that jump continue

acts similar to a break in

Java switch case statements

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if opcode == 0 {

if lowbyte == caseE0 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print " CLS"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 0 E0:

vram = new Array(COLS \* ROWS);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

clearVRam

}

endif

jump continue

}

else

endif

if lowbyte == caseEE {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print " RET"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 0 EE:

pc = stack[--sp];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sp - 1

tempvarA = stackStart

tempvarA + sp

ld pc tempvarA

}

endif

jump continue

}

else

endif

//default

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print " IGNORE"

}

else

endif

jump continue

}

else

endif

if opcode == 1 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE3"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 1:

pc = instruction & 0xfff;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & memsize

pc = tempvarA

pc + Rr1

}

endif

jump continue

}

else

endif

if opcode == 2 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE4"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 2:

var addr = instruction & 0xfff;

stack[sp++] = pc;

pc = addr;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & memsize

tempvarB = sp

tempvarB + stackStart

st tempvarB pc

sp + 1

pc = tempvarA

pc + Rr1

}

endif

jump continue

}

else

endif

if opcode == 3 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE5"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 3:

var b = instruction & 0xff;

pc = reg[r1] == b ? pc + 2 : pc;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & maskbyte

tempvarB = r1

tempvarB + regStart

ld tempvarB tempvarB

if tempvarB == tempvarA {

pc + 2

}

else

endif

}

endif

jump continue

}

else

endif

if opcode == 4 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE6"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 4:

var b = instruction & 0xff;

pc = reg[r1] != b ? pc + 2 : pc;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & maskbyte

tempvarB = r1

tempvarB + regStart

ld tempvarB tempvarB

if tempvarB != tempvarA {

pc + 2

}

else

endif

}

endif

jump continue

}

else

endif

if opcode == 5 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE7"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 5:

pc = reg[r1] == reg[r2] ? pc + 2 : pc;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

if tempvarA == tempvarB {

pc + 2

}

else

endif

}

endif

jump continue

}

else

endif

if opcode == 6 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE8"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 6:

var b = instruction & 0xff;

reg[r1] = b;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & maskbyte

tempvarB = r1

tempvarB + regStart

st tempvarB tempvarA

}

endif

jump continue

}

else

endif

if opcode == 7 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE9"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 7:

var b = instruction & 0xff;

reg[r1] += b;

reg[r1] &= 0xff;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & maskbyte

tempvarB = r1

tempvarB + regStart

tempvarC = tempvarB

ld tempvarB tempvarB

tempvarB + tempvarA

tempvarB & maskbyte

st tempvarC tempvarB

}

endif

jump continue

}

else

endif

if opcode == 9 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE10"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 9:

pc = reg[r1] != reg[r2] ? pc + 2 : pc;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

if tempvarA != tempvarB {

pc + 2

}

else

endif

}

endif

jump continue

}

else

endif

if opcode == 10 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE11"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case A:

ir = instruction & 0xfff;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & memsize

ir = tempvarA

}

endif

jump continue

}

else

endif

if opcode == 11 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE12"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case B:

pc = (instruction & 0xfff) + reg[0];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & memsize

ld tempvarB regStart

tempvarA + tempvarB

pc = tempvarA

pc + Rr1

}

endif

jump continue

}

else

endif

if opcode == 12 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE13"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case C:

reg[r1] = Math.floor(Math.random \* 255) & (instruction & 0xff);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & maskbyte

<asm> res r0 r1 r3 RANDB </asm>

tempvarB = getparam0

tempvarB & tempvarA

tempvarC = r1

tempvarC + regStart

st tempvarC tempvarB

}

endif

jump continue

}

else

endif

if opcode == 13 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print "COMPUTE14"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case D:

var b = instruction & 0xf;

let x = reg[r1];

let y = reg[r2];

for(i=0;i<b;i++) {

let row = y+i;

var sprite = mem[i+ir];

for(j=0;j<8;j++) {

let col=x+j;

if((sprite & 0x80) == 0x80)

reg[flag] = lcdSetPixel(col,row) ? 1 : 0;

sprite <<= 1;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = instruction

tempvarA & masknibble

tempvarB = r1

tempvarB + regStart

ld tempvarB tempvarB

tempvarC = r2

tempvarC + regStart

ld tempvarC tempvarC

tempvarG = ir

tempvarG + Rr1

tempvarK = flag

tempvarK + regStart

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

x = tempvarB = reg[r1]

y = tempvarC = reg[r2]

b = tempvarA

i = tempvarD

j = tempvarE

row = tempvarF:

y + i

tempvarC + tempvarD

sprite = tempvarH

col = tempvarI:

x + j

tempvarB + tempvarE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarD = 0

do {

tempvarE = 0

tempvarF = tempvarC

tempvarF + tempvarD

tempvarH = tempvarG

tempvarH + tempvarD

ld tempvarH tempvarH

do {

tempvarI = tempvarB

tempvarI + tempvarE

tempvarJ = tempvarH

tempvarJ & byte80

if tempvarJ == byte80 {

setpublic gXPassParam tempvarI

setpublic gYPassParam tempvarF

lcdSetPixel

getpublic tempvarJ gRetParam

st tempvarK tempvarJ

}

else

endif

tempvarH \* 2

tempvarE + 1

}

loop tempvarE < 8

tempvarD + 1

}

loop tempvarD < tempvarA

}

endif

jump continue

}

else

endif

if opcode == 8 {

if last4bits == 0 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8A"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 0:

reg[r1] = reg[r2];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

st tempvarA tempvarB

}

endif

jump continue

}

else

endif

if last4bits == 1 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8B"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 1:

reg[r1] |= reg[r2];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

getpublic tempvarC gData1Addr

st tempvarC tempvarA

getpublic tempvarC gData2Addr

st tempvarC tempvarB

getpublic tempvarC gORVal

getpublic tempvarD gORXORAddr

st tempvarD tempvarC

getpublic tempvarC gData1Addr

ld tempvarC tempvarC

tempvarA = r1

tempvarA + regStart

st tempvarA tempvarC

}

endif

jump continue

}

else

endif

if last4bits == 2 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8C"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 2:

reg[r1] &= reg[r2];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

tempvarA & tempvarB

tempvarB = r1

tempvarB + regStart

st tempvarB tempvarA

}

endif

jump continue

}

else

endif

if last4bits == 3 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8D"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 3:

reg[r1] ^= reg[r2];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

getpublic tempvarC gData1Addr

st tempvarC tempvarA

getpublic tempvarC gData2Addr

st tempvarC tempvarB

getpublic tempvarC gXORVal

getpublic tempvarD gORXORAddr

st tempvarD tempvarC

getpublic tempvarC gData1Addr

ld tempvarC tempvarC

tempvarA = r1

tempvarA + regStart

st tempvarA tempvarC

}

endif

jump continue

}

else

endif

if last4bits == 4 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8E"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 4:

var res = reg[r1] + reg[r2];

reg[r1] = res & 0xff;

reg[flag] = res > 0xff ? 1 : 0;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

tempvarA + tempvarB

tempvarA & maskbyte

tempvarB = r1

tempvarB + regStart

st tempvarB tempvarA

tempvarC = flag

tempvarC + regStart

if tempvarA > maskbyte {

tempvarD = 1

st tempvarC tempvarD

}

else {

tempvarD = 0

st tempvarC tempvarD

}

endif

}

endif

jump continue

}

else

endif

if last4bits == 5 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8F"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 5:

var res = reg[r1] - reg[r2];

reg[flag] = reg[r1] > reg[r2] ? 1 : 0;

reg[r1] = res & 0xff;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

tempvarC = flag

tempvarC + regStart

if tempvarA > tempvarB {

tempvarD = 1

st tempvarC tempvarD

}

else {

tempvarD = 0

st tempvarC tempvarD

}

endif

tempvarA - tempvarB

tempvarA & maskbyte

tempvarB = r1

tempvarB + regStart

st tempvarB tempvarA

}

endif

jump continue

}

else

endif

if last4bits == 6 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8G"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 6:

reg[flag] = ((reg[r1] & 0xff) << 7) == 0x80 ? 1 : 0;

reg[r1] >>= 1;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarA & maskbyte

setpublic gXPassParam lShiftby7

2ToPower

getpublic retVal7 gRetParam

tempvarA \* retVal7

tempvarB = flag

tempvarB + regStart

if tempvarA == byte80 {

tempvarD = 1

st tempvarB tempvarD

}

else {

tempvarD = 0

st tempvarB tempvarD

}

endif

tempvarA = r1

tempvarA + regStart

ld tempvarB tempvarA

tempvarB / 2

tempvarB = getparam2

st tempvarA tempvarB

}

endif

jump continue

}

else

endif

if last4bits == 7 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8H"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 7:

var res = reg[r2] - reg[r1];

reg[flag] = reg[r2] > reg[r1] ? 1 : 0;

reg[r1] = res & 0xff;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarB = r2

tempvarB + regStart

ld tempvarB tempvarB

tempvarC = flag

tempvarC + regStart

if tempvarB > tempvarA {

tempvarD = 1

st tempvarC tempvarD

}

else {

tempvarD = 0

st tempvarC tempvarD

}

endif

tempvarB - tempvarA

tempvarB & maskbyte

tempvarA = r1

tempvarA + regStart

st tempvarA tempvarB

}

endif

jump continue

}

else

endif

if last4bits == 14 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "COMPUTE-8I"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case 8 0xE

reg[flag] = ((reg[r1] & 0xff) >> 7) == 1 ? 1 : 0;

reg[r1] <<= 1;

reg[r1] &= 0xff;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarA & maskbyte

setpublic gXPassParam lShiftby7

2ToPower

getpublic retVal7 gRetParam

tempvarA / retVal7

tempvarA = getparam2

tempvarB = flag

tempvarB + regStart

if tempvarA == 1 {

tempvarD = 1

st tempvarB tempvarD

}

else {

tempvarD = 0

st tempvarB tempvarD

}

endif

tempvarA = r1

tempvarA + regStart

ld tempvarB tempvarA

tempvarB \* 2

tempvarB & maskbyte

st tempvarA tempvarB

}

endif

jump continue

}

else

endif

//default

if debug == 1 {

print opcodeSTR

print opcode

print space

print last4bits

print space

print "Err NO OPCPDE"

}

else

endif

jump continue

}

else

endif

/\* case 0xE \*/

if opcode == 14 {

if lowbyte == case9E {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print unimplemented

}

else

endif

jump continue

}

else

endif

if lowbyte == caseA1 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print unimplemented

}

else

endif

jump continue

}

else

endif

//default

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print " Err NO OPCPDE"

}

else

endif

jump continue

}

else

endif

/\* case 0xF \*/

if opcode == 15 {

if lowbyte == 7 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print ldOpen

print r1

print brackClose

print ","

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case F 0x7:

reg[r1] = delayTimer;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

st tempvarA delayTimer

}

endif

jump continue

}

else

endif

if lowbyte == 10 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print unimplemented

}

else

endif

jump continue

}

else

endif

if lowbyte == case15 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print " LD delaytimer"

print regOpen

print r1

print brackClose

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case F 0x15:

delayTimer = reg[r1]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld delayTimer tempvarA

}

endif

jump continue

}

else

endif

if lowbyte == 8 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print " LD soundtimer"

print regOpen

print r1

print brackClose

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case F 0x8:

soundTimer = reg[r1]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld soundTimer tempvarA

}

endif

jump continue

}

else

endif

if lowbyte == case1E {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print space

print "COMPUTE15"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case F 0x1E:

ir += reg[r1];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

ir + tempvarA

}

endif

jump continue

}

else

endif

if lowbyte == case29 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print space

print "COMPUTE16"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case F 0x29:

var b = reg[r1];

ir = FONTSET\_START\_ADDRESS + (5 \* b);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

tempvarA \* 5

getpublic tempvarB gFontSetAddr

//tempvarB + Rr1 (Not a memory access...hmm)

tempvarB + tempvarA

ir = tempvarB

}

endif

jump continue

}

else

endif

if lowbyte == case33 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print space

print "COMPUTE17"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case F 0x33:

var b = reg[r1];

mem[ir + 2] = b % 10;

b /= 10;

b=parseInt(b);

mem[ir + 1] = b % 10;

b /= 10;

b=parseInt(b);

mem[ir] = b % 10;

Warning: Remainder has bug!

Do no use it after division.

Remainder = getparam0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = r1

tempvarA + regStart

ld tempvarA tempvarA

// Ones-place

tempvarD = tempvarA

tempvarB = ir

tempvarB + Rr1

tempvarB + 2

tempvarA / 10

tempvarC = getparam2

tempvarA = getparam2

tempvarC \* 10

tempvarD - tempvarC

st tempvarB tempvarD

// Tens-place

tempvarD = tempvarA

tempvarB = ir

tempvarB + Rr1

tempvarB + 1

tempvarA / 10

tempvarC = getparam2

tempvarA = getparam2

tempvarC \* 10

tempvarD - tempvarC

st tempvarB tempvarD

// Hundreds-place

tempvarD = tempvarA

tempvarB = ir

tempvarB + Rr1

tempvarA / 10

tempvarC = getparam2

tempvarC \* 10

tempvarD - tempvarC

st tempvarB tempvarD

}

endif

jump continue

}

else

endif

if lowbyte == case55 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print space

print "COMPUTE18"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case F 0x55:

for(i=0; i<=r1; i++)

mem[i+ir] = reg[i];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = 0

do {

tempvarB = Rr1

tempvarB + tempvarA

tempvarB + ir

tempvarC = regStart

tempvarC + tempvarA

ld tempvarC tempvarC

st tempvarB tempvarC

tempvarA + 1

}

loop tempvarA <= r1

}

endif

jump continue

}

else

endif

if lowbyte == case65 {

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print space

print "COMPUTE19"

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

case F 0x65:

for(i=0; i<=r1; i++)

reg[i] = mem[i+ir];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tempvarA = 0

do {

tempvarB = Rr1

tempvarB + tempvarA

tempvarB + ir

ld tempvarB tempvarB

tempvarC = regStart

tempvarC + tempvarA

st tempvarC tempvarB

tempvarA + 1

}

loop tempvarA <= r1

}

endif

jump continue

}

else

endif

//default

if debug == 1 {

print opcodeSTR

print opcode

print space

print lowbyte

print space

print "Err NO OPCPDE"

}

else

endif

jump continue

}

else

endif

//end for if opcode == 15

/\* Here? No opcode!

default

\*/

if debug == 1 {

print "Error: No opcode found"

}

else

endif

}

//end label mainloop

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jump to mainloop

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

label continue {

if debug == 1 {

println ""

if filesize > 0 {

filesize - 1

counter + 1

jump mainloop

}

else {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Done! Print chip8

memory to file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

lea Rr0 outfile

param0 = Rr0

param1 = Rr1

<asm> ld r3 memsize-main </asm>

<asm> res r0 r1 r3 OUT </asm>

}

endif

}

else {

jump mainloop

}

endif

}

//end label continue

}

/\*\*\*\* HELPER FUNCTIONS \*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function 2ToPower:

See binary.lct for

explanation.

Returns 2^gXPassParam

Returns 2 if gXPassParam

is 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function 2ToPower {

let iLocal = 1

let powerOf2 = 2

let numofloops = 0

getpublic numofloops gXPassParam

while iLocal < numofloops

iLocal + 1

powerOf2 \* 2

endwhile

setpublic gRetParam powerOf2

powerOf2 = 2

iLocal = 1

return

}

//end function 2ToPower

function lcdInit {

let debug = 0

getpublic debug gDebug

let lcdInitAddr = 0xff0b

let lcdInitVal = 0x5c

st lcdInitAddr lcdInitVal

if debug == 1 {

//println "LCD Init"

}

else

endif

return

}

function clearVRam {

let start = 0

let debug = 0

let ivramsize = 0

getpublic start gVRamAddr

getpublic debug gDebug

getpublic ivramsize gVRamSize

let clear = 0

do

if debug == 1 {

//print "Clearing vram at: "

//println start

}

else

endif

st start clear

start + 1

ivramsize - 1

loop ivramsize > 0

return

}

function setupVramForLCD {

let start = 0

let w = 0

getpublic start gVRamAddr

getpublic w gVRamWidth

param0 = start

param1 = w

<asm> ld r3 gVRamHeight-global </asm>

<asm> res r0 r1 r3 VRAM </asm>

<asm> ld r0 gVRamScale-global </asm>

<asm> res r0 r1 r3 SCALE </asm>

return

}

function lcdSetPixel {

let x = 0

let y = 0

let COLS = 0

let ROWS = 0

let start = 0

let bitwiseORXORAddr = 0

let bitwiseXORVal = 0

let data1Addr = 0

let data2Addr = 0

let val = 0

let one = 1

let true = 1

let false = 0

getpublic x gXPassParam

getpublic y gYPassParam

getpublic start gVRamAddr

getpublic COLS gVRamWidth

getpublic ROWS gVRamHeight

getpublic bitwiseORXORAddr gORXORAddr

getpublic bitwiseXORVal gXORVal

getpublic data1Addr gData1Addr

getpublic data2Addr gData2Addr

if x > COLS

x - COLS

jump lcdSetPixel01

else

endif

if x < 0

x + COLS

else

endif

label lcdSetPixel01

if y > ROWS

y - ROWS

jump lcdSetPixel02

else

endif

if y < 0

y + ROWS

else

endif

label lcdSetPixel02

y \* COLS

x + y

x + start

ld val x

st data1Addr val

st data2Addr one

st bitwiseORXORAddr bitwiseXORVal

ld val data1Addr

/\*

vram[x + (y \* COLS)] ^= 1;

\*/

st x val

/\*

return vram[x + (y \* COLS)] != 1;

\*/

if val != 1

setpublic gRetParam true

else

setpublic gRetParam false

endif

return

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

lcdTestRender:

Only use it for testing.

Status: Working :)

Warning: Has infinite loop

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function lcdTestRender {

let keepgoing = 1

let x = 0

let y = 0

let sleep = 1000

let sleepaddr = 0

getpublic sleepaddr gSleepAddr

while keepgoing == 1 {

<asm> res r0 r1 r3 RAND </asm>

x = getparam0

y = getparam1

setpublic gXPassParam x

setpublic gYPassParam y

lcdSetPixel

<asm> res r0 r1 r3 RAND </asm>

x = getparam0

y = getparam1

setpublic gXPassParam x

setpublic gYPassParam y

lcdSetPixel

st sleepaddr sleep

}

endwhile

return

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

func: lcdDrawRectangle

draws a shape at pos x, y

of lenght len and height hei

function lcdDrawRectangle(x,y,len,hei) {

for(i=0; i < len; i++)

for(j=0; j < hei; j++)

lcdSetPixel(x+i,y+j);

}

Status: Working :)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function lcdDrawRectangle {

let x = 0

let y = 0

let xtemp = 0

let ytemp = 0

let len = 0

let hei = 0

let i = 0

let j = 0

getpublic x gXPassParam

getpublic y gYPassParam

getpublic len gLenPassParam

getpublic hei gHeiPassParam

getpublic i gReset

getpublic j gReset

do {

do {

xtemp = x

ytemp = y

xtemp + i

ytemp + j

setpublic gXPassParam xtemp

setpublic gYPassParam ytemp

lcdSetPixel

j + 1

}

loop j < hei

i + 1

j = 0

}

loop i < len

return

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Taken from:

austinmorlan.com/posts/chip8\_emulator/

Although he had a typo:

instead of (7,6) it should be

(7,3). I emailed him about it:

mail@austinmorlan.com

lcdDrawRectangle(1,1,10,4);

lcdDrawRectangle(6,6,8,2);

lcdDrawRectangle(7,3,3,4);

Status: Working :)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function lcdTest {

let x = 0

let y = 0

let len = 0

let hei = 0

x = 1

y = 1

len = 10

hei = 4

setpublic gXPassParam x

setpublic gYPassParam y

setpublic gLenPassParam len

setpublic gHeiPassParam hei

lcdDrawRectangle

x = 6

y = 6

len = 8

hei = 2

setpublic gXPassParam x

setpublic gYPassParam y

setpublic gLenPassParam len

setpublic gHeiPassParam hei

lcdDrawRectangle

x = 7

y = 3

len = 3

hei = 4

setpublic gXPassParam x

setpublic gYPassParam y

setpublic gLenPassParam len

setpublic gHeiPassParam hei

lcdDrawRectangle

return

}