INSTRUCTIONS
16 TOTAL
Each splits the bytes into what's required
Pergorms their opperation
(IF needed increment the program counter)

KEY:
rx = register, a valuw 0-7 for our
bb = a byte
aaa = an address

Registers
8 bit r0-r7
Program counter: 16 bit reg, stores address of current instruction

Timer: 8 bit reg for storing timer value

Address - 16 bit reg for storeing an address

Memort, single bit flag that determines whether memory should be ram or ROM-- 0 ram 1 rom

Ram
TOTAL 4kb
 4096 bytes
32768 bits

Rom starts at 0x0 , reads only, throws an error if a write operation is attempted in rom

CPU 500 instructions a second
each cycle the computer reads 2 bytes from the ROM at program counter and p+1 and passes to CPU TO execute
the 2 bytes make 1 instruction
p should always be even

All numbers are base 16

Timer, when it's not 0 , the cpu decrements the value by 1 every 16ms

Registers
+register: Array<Ubyte>[size]
+constructor(int regSize)
+read(){return interpreted byte}
+write(byte){write the byte to bits}

Bytes will be

Timer
-time:Short (0 defualt) max 255
+decrementTime(){time--}
+getTIme()
+seTime(int){}

4 kilobytres
8 bits each

Ram
+memory: 2d Array<UByte>[4096]

UI

Console
+computer: Computer()
+update(){ takes a computer screens and prints out the entire 2d array or promt

Sends TO

Computer
+programCounter: Register
+timer: Timer
+address: Register
+memory:boolean
+ram: Ram
+rom:Rom
-registers: List<Registers>[8]
+display:display
+invoker:Invoker
+invokeInstruction(instruction){
invoker.excecute

ROM
+memory:List<>
- private set that's disabled after creation
+readRom():returns

CPU
+instructionQue: List<Byte>
+run(){loop, till end of instruction que}
+countTimer(){decrement if not 0}
+clock(){runs 500 instructions every 1 second}
+callInstruction()

Read Instructions

<<interface>>
instruction
+
performInsruction(2byteInstruction)

Executes

Invoker
+computer:computer
-instructionset: List<instrucitons>
+commandQue: List<commands>
+
+execute(romInstruction){
executes instruction with given 2byte instruction from rom}

Screen
+screen: 2d Array<char>
+buffer: Array[64] (bytes for asciii)
+refresh(){
60hz}

8x8 Display , with a frame buffer of 64 bytes

Abstract
BaseInstruction
+abstract id: base 16 representation of command
+computer:Computer
+ performINstruction(instruction){
split
perform op
incrementPcount
}
+ open/abstract splitBytes(){with defualt implementation}
+abstract performOperation()
+abstract incrementPcount(){optional, defualt pCount +2}

Store
+id: 0
+override performOperation()

Add
+id: 1
+override performOperation(){stores in reg1}

Sub
+id: 2
+override performOperation(){stores in reg1 (r1-r2) }

Read
+id: 3
+override performOperation(){reads address, puts in register }

Write
+id: 4
+override performOperation()

Jump
+id:5
+override performOperation(){jump to address
+override incrementPcount(){do nothing}

ReadKeyboard
+id: 6
+override performOperation(){
gets input, stores in register (empty = 0)
}

SwitchMemory
+id: 7
+performOperation(){ M = !M}

SkipEqual
+id:8
+performOperation(){ compare reg, calls increment an additional time if true}

SkipNot
+id 9
-performOperation(){double increment conter if r1 and r1 not equal}

SetA
+id A (or 10)
-performOperation(){A = adressvalue}

SetT
+id B (or 11)
-performOperation(){set computer.timer.setTime(bytetime)}
-override splitBytes(){take middle two half bytes for time value}

ReadT
+id C (12)
-performOperation(){reg value = T value}

ConvertB10
+id: D (13)
+performOpperation(){ convert to base ten , place in a1 2 and 3

A for hundreds
A2 for tens
A3 for ones

ByteToAscii
+id:E (14)
+performOperation(){

Draw
+id:F
+performOpperation(){