CPU Emulator

The Bootloader

- bootable image consists of a 256 byte bootloader and a program
- bootloader begins with 2 bytes (eb ff) and ends with 2 bytes (55 aa)
- the 3rd byte defines what kind of program space to start with (default is the setup loop doesn't run)
- the 4th byte indicates if or if not the loop is entered if the setup is entered
- fa indicates loop will be entered f0 indicates loop will not be entered

-
  ```
  eb ff dd f0 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa
  ```

- Sample bootloader with no declaration
- the bootloader is used to define all used variables and their size default is 16 bit

Program Space

- after the bootloader is an infinite amount of program space (only limited to your max memory including ram and drive space)

- there are 2 types of program space:

name	signature	description
setup	dd	setup signature
loop	de	loop signature

- end of each program space is declared with 55 aa and begins with eb xx while xx being the signature of the program space

- the setup only runs once
- example of the setup space:
  ```
  eb dd 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  *
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa\
  ```

- the 3rd and 4th byte of the loop indicate how many times the program in the loop space is executed
- if the 3rd and 4th byte are ff ff the loop runs indefinitely
- example of the loop space:
  ```
  eb d0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  *
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa\
  ```

RAM

- the ram consists of a list with n words. words being bytes
  - the default value if words is 65535 or a size of ~64kb
    - max address is FFFF and min address is 0000 while 0000 is a function address and works like a dump meaning 0000 will always be 00000000
    - **all data send to 0000 will be lost!!**

- sample:
  - [[0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0],...]
  - accessing a specific location in RAM uses the hex address of the word
    - e.g aa, faa, f, ...
  - accessing a specific bit in RAM uses the hex address of the word and the bit offset in hex separated by a '**:**'
    - a:8
      - reads the 8th bit of the word at address a -ff:2
      - reads the 2nd bit of the word at address ff

- the RAM can also be configured to represent the values in DEC or HEX
  - **reading specific bits in a word will then be disabled!!**
  - e.g. [[0x41],[0x0],[0x14],[0xff],...]
  - e.g. [[1],[123],[255],[33],...]

Registers

- there are 4 registers:
  - register a:
    - modification register
      - only has 16 words with each word having 9 bits
        - [x,0,0,0,0,0,0,0,0]
        - x is used to indicate positive or negative numbers if x is 1 number is negative
        - when in DEC or HEX mode the register looks like this:
          - [x,192] or [x,0xff]
      - used for modifying data e.g. adding, subtracting

- register b:
  - communication register
    - used for communication between ram, cpu and gpu
    - 64 words with 8 bits each

- register c:
  - logic register
  - used for logic operations: AND, OR, XOR and CMP (see flag register)
  - 4x 2 words with each having 8 bits

- register d:
  - multipurpose register
  - 32 words with each having 16 bits

- flag register
  - used for setting flags
  - 4 words with each having 2 bits
    - word 1
      - register for flag from CMP
      - [0,0] : default
      - [0,1] : a smaller than b
      - [1,0] : a bigger than b
      - [1,1] : equal
    - word 2
      - interupt
      - [0,0] : continue
      - [0,1] : stop the program until enter is pressed (not used for input)

        - word 3
            - user settable flag
            - get flag bit using 3:1 for bit on the right and 3:2 for bit on the left
        - word 4
            - exit code
            - [0,0] : no error
            - [0,1] : buffer overflow (variable used more RAM than it should or tried to use more RAM than available)
            - [1,0] : user initiated error
            - [1,1] : general error

GPU

- the GPU has a line buffer with the size of bits according to the amount of pixel on the x axis
- the GPU's frame buffer is sized according to this formula (x_pixels*y_pixels/pixel_size)
- the GPU also includes a general purpose ram with a size of 32 bits
- the GPU doesn't do any calculations
    - instead the CPU does all the work
    - the GPU is only used for drawing pixels and text

Instruction set

cpu instructions

HEX code	Opcode	Description	Usage
0x0	/	does absolutely nothing	/
0x1	/	Reserved for GPU	/
0x2	INC	adds 1 to data at address or value	< data / address >
0x3	DEC	decreases data at address or value by 1	< data / address >
0x4	ADD	add	< int or addr > to < int or addr > < output addr >
0x5	SUB	subtract	< int or addr > from < int or addr >

HEX code	Opcode	Description	Usage
			< output addr >
0x6	AND	bit wise and operation	< int or addr > < int or addr > < output addr >
0x7	OR	bit wise or operation	< int or addr > < int or addr > < output addr >
0x8	XOR	bit wise xor operation	< int or addr > < int or addr > < output addr >
0x9	CMP	compares 2 values and sets flag	< int or addr > < int or addr >
0xA	PUSH	overwrites value at memory address	< int or addr > < dst >
0xB	POP	clears data at memory address	< dst >
0xC	MOV	switches values at src and dst	< src > < dst >
0xD	IN	waits for input, input gets saved in var	< var >
0xE	BS	bit shift	< opperant <<< for left >>> for right > < var / addr >
0xf	BR	exits / stops the program and sets flag	< exit code / description >

gpu instructions

HEX code	Opcode	Description	Usage
0x11	GPUINIT	initialises the gpu	/
0x12	GPUPRN	prints text to display	< str in "" / mem addr / data >
0x13	GPUPIX	draws a pixel on the monitor	< position in () > < 0 for white and 1 for balck pixel >
0x14	GPUCLS	clears the monitor	/
0x15	GPUDMPLB	dumps the line buffer to the monitor	< line index default is 0 >

HEX code	Opcode	Description	Usage
0x16	GPUDMPFB	dumps the frame buffer to the monitor	/
0x17	GPUUPD	refreshes the monitor	/