D5700 CPU

The D5700 is a computer that I invented for this class. Some of its features include:

- 8-bit, 500Hz CPU with 16 instructions
- 4kb of RAM
- Read programs from ROM
- 8x8 ASCII display
- ASCII keyboard input

This computer is very limited, but you can still make some very interesting programming that run on it.

Architecture

Registers

The D5700 CPU has 8 general purpose, 8-bit registers (`r0` - `r7`).

It also has 4 special registers

- `P` rogram Counter. A 16-bit register that stores the address of the current instruction.
- `T` imer - an 8-bit register for storing timer value
- `A` ddress - a 16-bit register for storing an address
- `M` emory - a single bit flag that determines whether memory operations should occur on RAM or ROM: 0 for RAM, 1 for ROM

Timer

While the `T` register is not 0, the CPU decrements the value by 1 at 60hz (every 16ms).

RAM

The D5700 has 4kb of RAM.

ROM

The D5700 reads each instruction from ROM starting at address 0x0. On the physical hardware, ROM would be loaded into the computer via a cartridge.

All D5700 ROM chips are 4kb in size.

The currently running program is terminated with an error if a write operation is attempted on ROM. However, in the future, some cartridges will include writable chips and the CPU should be future-proof for when that happens.

CPU

The computer executes instructions at 500hz (500 times per second)

The computer reads 2 bytes from ROM every cycle at `P` and `P` + 1 and gives them to the CPU to execute. These bytes form the instruction.

This means the `P` should always be an even number; operations that increment the program counter should increment it by 2

SCREEN

The screen is an 8x8 ASCII display. The screen has 64 bytes of internal RAM that serves as the frame buffer for the ASCII character to display at each position.

Instruction Set

The first nibble of each 2-byte instruction maps to one of the instructions. There are 16 instructions in total.

Each instruction does the following:

1. Splits the bytes up into the registers, address, bytes etc... required by the operation
2. Performs the operation
3. (Optional) increments the program counter.

Use the following to understand each instruction:

- `rX` = register, a value 0-7
- `bb` = a byte
- `aaa` = an address

ALL NUMBERS SHOULD BE INTERPRETED AS BASE-16 NUMBERS

STORE (0, rX, bb)

Stores byte `bb` in register `r`

- Example `00FF` - stores the value `FF` in register `0`

ADD (1, rX, rY, rZ)

Adds the value in `rX` to the value in `rY` and stores in `rZ`

- Example `1010` - adds the values in r0 and r1 and stores in r0

SUB (2, rX, rY, rZ)

Adds the value in `rY` from the value in `rX` and stores in `rZ`.

- Example `2010` - subtracts the value in r1 from r0 and stores in r0

READ (3, rX, 00)

Reads the value in memory at the address stored in `A` and store in register `rX`;

Reads from ROM if `M` = 1

- Example `3700` stores the value in memory at address `A` and store in `r7`.

WRITE (4, rX, 00)

Writes the value in `rX` to memory at the address stored in `A`.

Will attempt to write to ROM if `M` = 1. This will fail for most ROM chips. But, some ROMs have a writable chips. For future proofing, the instruction should attempt to write to ROM.

- Example `4300` - stores the value in `r3` in memory at address `A`

JUMP (5, aaa)

Sets `P` to the value of `aaa`

Terminates the program with an error if `aaa` is not divisible by 2.

Program counter is not incremented after this instruction.

- Example `51F2` - sets the program counter to `1F2`

READ_KEYBOARD (6, rX, 00)

Pauses the program and waits for keyboard input. Only the base-16 digits 0-F are allowed as input. Input should be parsed as a number and stored in rX. Only up to 2 digits (one byte) are read, the rest are ignored. Stores 0 if input is the empty string.

- Example `6200` - stores the number the user typed in `r2`

SWITCH_MEMORY (7000)

Toggles the `M` register - sets to 1 if `M` is 0 and sets to 0 if `M` is 1.

SKIP_EQUAL (8, rX, rY, 0)

Compares the values in `rX` and `rY` and skips the next instruction if they are equal.

- Example `8120` - compares `r1` with `r2` and skips the next instruction if they are equal.

SKIP_NOT_EQUAL (9, rX, rY, 0)

Compares the values in `rX` and `rY` and skips the next instruction if they are NOT equal.

- Example `9120` - compares `r1` with `r2` and skips the next instruction if they are NOT equal.

SET_A (A, aaa)

Sets the value of `A` to be `aaa`.

- Example `A255` - sets `A` to be 255.

SET_T (B, bb, 0)

Sets the value of `T` top be `bb`.

- Example `B0A0` - sets `T` to be 0A (or just A).

READ_T (C, rX, 00)

Reads the value of `T` and stores in `rX`.

- Example `C000` - reads `T` and stores in `r0`

CONVERT_TO_BASE_10 (D, rX, 00)

Converts the byte stored in `rX` to base-10 and stores the 100s digit in `A` the, 10s digit in `A+1` and the 1s digit in `A+2`.

- Example `D200` - converts value `r2` and stores the digits in `A`, `A + 1`, and `A+2`.

CONVERT_BYTE_TO_ASCII (E, rX, rY, 0)

Takes the digit (0-F) stored in `rX` and converts it to the ASCII value for the digit and stores it in `rY`

Terminates the program with an error if the byte stored in `rX` is greater than `F` (base-16)

- Example `E010` - Takes the value in `r0` and stores the ASCII value in `r1`.

DRAW (F, rX, rY, rZ)

Draws the ASCII character for the byte stored in `rX` at row `rY`, and column `rZ` to the screen. (Row and column are 0 based.)

It accomplishes this by writing the ASCII character to the screens internal RAM. `rY` and `rZ` are converted to the address in the RAM before writing.

Terminates the program with an error if the value in `rX` is > 7F (127 in base-10)

- Example `F123` - draws the ASCII character stored in `r1` at row `r2` and column `r3`