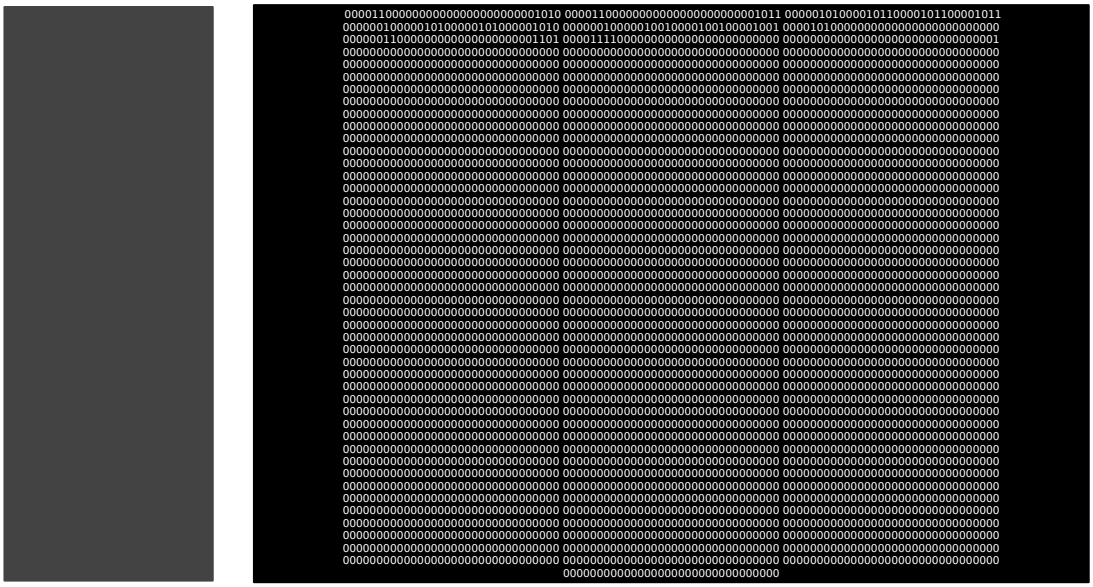
# system/a

general
computing
platform



# operating manual

# design

system/a is a 16 - 32 bit computer designed to conduct simple computations on programs loaded exclusively into main memory. In the 32 bit configuration there are up to 256 memory addresses for code or data. In the 16 bit configuration there are only 16 possible memory addresses. This memory of course can be augmented by external input containing instructions and data that can then be processed by external infrastructure.

To start up a system/a machine you first load a program into the memory of the computer and then you initiate processing. The system will then read the instructions contained within the memory starting at address 0 and incrementing up from there (ignoring JMP statements).

Each instruction is formatted as follows :

```
    0000      0000      0000      0000
[instruction] [arg addr1] [arg addr2] [target addr]
```

Where the `[instruction]` is something like a jump, addition statement, flip statement, or something else, `[arg addr1]` is the address for the first argument, `[arg addr2]` is the address for the second argument, and `[target addr]` is the address of the target of the operation where the value is placed or the system jumps to.

In a physical implementation system/a would be a fully mechanical computer operating via punch cards.

These are the eight instructions that system/a can process :

## FLP = 00001101

- flips the bits of the value held in `[arg addr1]` and places it in `[target addr]`, allowing the user to quickly convert positive numbers to negative numbers

## ADD = 00000010

- adds the value held in `[arg addr1]` to `[arg addr2]` and places it in `[target addr]`, which allows the user to add two values

## JMP = 00001010

- jumps unconditionally to the memory address located in `[target addr]`. Use this to skip data stores that are local to the code being processed.

## JIE = 00000101

- jumps to the memory address located in `[target addr]` if the values stored in `[arg addr 1]` and `[arg addr 2]` are equal.

## WRT = 00000011

- writes the contents of the `[arg addr1]` to the output buffer

## RED = 00001100

- writes the contents of the input buffer to `[target addr]`

## MOV = 00000100

- moves the contents of [arg addr1] to [target addr]

## STP = 00001111

- stops the machine. This is included because without a stop command the machine would run off of the remaining data in memory, executing any code that may inadvertently exist there and mutilating punch cards.

## NUM

serves as a marker to designate an address as storing a number. **note** : is only used by the assembler to properly parse numbers, it is has no final representation in the machine code and thus if you are not careful your numbers can be read as instructions.

This would be an example of a simple addition program that takes the number 3, turns it negative, and then adds the number 4 and then writes it to the output buffer if it is equal to 4, if it is not it writes.

| | system/a ASM: | | system/a machine code: |
|---|---|---|---|
| bc 00 | FLP 0A 0A | 0000 | 1101 1010 0000 1010 |
| 01 | ADD 0B 0A 0A | 0001 | 0010 1011 1010 1010 |
| 02 | ADD 0A 09 08 | 0010 | 0010 1010 1001 1000 |
| 03 | JIE 08 09 05 | 0011 | 0101 1000 1001 0101 |
| 04 | WRT 08 | 0100 | 0011 1000 0000 0000 |
| 05 | STP | 0101 | 1111 0000 0000 0000 |
| 06 | | 0110 | 0000 0000 0000 0000 |
| 07 | 0 | 0111 | 0000 0000 0000 0000 |
| 08 | 4 | 1000 | 0000 0000 0000 0100 |
| 09 | 3 | 1001 | 0000 0000 0000 0011 |
| 0A | 1 | 1010 | 0000 0000 0000 0001 |
| 0B | 0 | 1011 | 0000 0000 0000 0000 |
| 0C | | 1100 | 0000 0000 0000 0000 |
| 0D | | 1101 | 0000 0000 0000 0000 |
| 0E | | 1110 | 0000 0000 0000 0000 |
| 0F | | 1111 | 0000 0000 0000 0000 |

I: nothing
O: 0000 0000 0000 0001 (1)

This would be an example of a simple multiplication program that multiplies two numbers from the input buffer together and writes the output to the output buffer.

| | system/a ASM: | | 32 bit system/a machine code: |
|---|---|---|---|
| 00 | RED 0F;input 0 | 00000000 | 00001100 00001111 00000000 00000000 |
| 01 | RED 0E;input 1 | 00000001 | 00001100 00001110 00000000 00000000 |
| 02 | JIE 0E 0D 07; jumps to wrt if eq | 00000010 | 00001010 00001100 00001101 00000000 |
| 03 | ADD 0F 0F 0C; adds 0f to 0f | 00000011 | 00000110 00001000 00000000 00000000 |
| 04 | ADD 0B 0D 0D; adds 1 to counter | 00000100 | 00000101 00000101 00000000 00000000 |
| 05 | WRT 0C | 00000101 | 00000010 00001111 00001111 00001111 |
| 06 | JMP 00 | 00000110 | 00000110 00000011 00000000 00000000 |
| 07 | STP | 00000111 | 00000010 00001111 00001100 00001111 |
| 08 | | 00001000 | 00000011 00001100 00000000 00000000 |
| 09 | | 00001001 | 00001111 00000000 00000000 00000000 |
| 0A | | 00001010 | 00000000 00000000 00000000 00000000 |
| 0B | NUM 1 | 00001011 | 00000000 00000000 00000000 00000000 |
| 0C | ;this is where the output is | 00001100 | 00000000 00000000 00000000 00000000 |
| 0D | ;this is the counter value | 00001101 | 00000000 00000000 00000000 00000000 |
| 0E | ;this is where input 0 is | 00001110 | 00000000 00000000 00000000 00000000 |
| 0F | ;this is where input 1 is | 00001111 | 00000000 00000000 00000000 00000000 |

I: 0000 0000 0000 1000 (8), 0000 0000 0000 1001 (9)
O: 0000 0000 0100 1000 (72)