BIT.C MACHINE (REV 3.3)

A simple, Turing-complete and easy to recreate CPU architecture.

SPECS

Registers:

- 16, 64-bit general purpose registers all initialized to zero. Can be accessed with the notion `rA`, `rB`, `rC`, ..., `rP`.
- 3 special registers: `LAST`, `PC` and `ERR`. `LAST` can only be set by instruction outputs and not directly, And cannot be read. `PC` can be set and read using the `pc` instruction. `ERR` will only be set to an error code in case of an error and followed immediately by a CPU halt.

Memory:

By default, The bit machine has `131,072` bytes of random-access memory. `16,384` bytes of this memory starting from `0` are for the system code. The usual, Non-magic-addressed memory starts at 20,001. And inbetween `16,384 … 20,000` exist the magic addresses needed to interact with modules outside the CPU and RAM. Extra memory could theoretically be achieved using 3rd-party input devices.

Addressing modes:

`1 <constant>`: A constant value. Cannot be used in `addr` mode for addressing modes.

`01 <register>`: If used as a destination, It will write to the specified register. Otherwise, It will read the value from said register.

`00 <register>`: If used as a destination, It will write to the memory address pointed to by the value of the specified register. Otherwise, It will read from the same memory address.

INSTRUCTIONS:

`00`: Manager instruction (2^{x+3})

- `00` Halt: This instruction will stop all execution immediately forever. Should be preceded by a shutdown preparation.
- `11` (2^{3+3}) 64-bit mode: Puts the machine in 64-bit mode. In this setting, Each instruction is 64 bits and an address mode is 30 bits.
- `10` (2^{2+3}) 32-bit mode: Puts the machine in 32-bit mode. In this setting, Each instruction is 32 bits and an address mode is 14 bits.
- `01` (2^{1+3}) 16-bit mode: The smallest instruction size. In this mode, Each instruction is 16 bits and an address mode is 6 bits. Useful for saving space when working only with regsiters or pre-set registers containing a memory address.

`01`: MOV instruction

- Syntax:
  - mov(**data** from, **addr** to)
- Moves data between two places, Which each could be a register, A memory address-containing register, etc.
- **Notice**: This instruction can be written using the math instruciton as follows:
- ```
  macro mov(SRC, DEST)
    math nand DEST, $0   ; Becomes all 1s
    math nand DEST, DEST ; Becomes all 0s
    math add DEST, SRC   ; 0 + x == x
  end mov
  ```

`10`: PC (program counter) instruction

- Syntax:
  - pc(**bit** get/set, **bit[3]**? flags, **data** target)
  - pc(<set>, **addr** target)
  - jmp(**bit(3)** flags, **data** target) == pc(get, …)
- If the first bit is set, The machine will jump to said target in memory (Sets `PC` to target) if any of the flags match the `LAST` register.
- If the first bit is not set, The target type changes to an `addr`. The flags are ignored and the current value of `PC` incremented by 1 is written to `target`.
- The `LAST` register is a special register that cannot be read from the code and can only be set by the code and used by the JMP instruction. It is set to `from` in `mov`, `0` in `manager`, Doesn't change in `jmp` and set to the result in `mth`.

`11`: MATH instruction

- Syntax:
  - math(**bit** operation, **addr** left, **data** right)
- If the first bit is set, The operation is `nand`. Equivalent C code would be `left = ~(left & right);`
- If it is not set, The operation is `add`. Equivalent C code would be `left += right;`
  ```