Documentation for sandbox-assembly

By mayedl10

**The shell**

The shell has 4 commands. These are:

* help .……… shows help on all the commands
* exit ..……. closes the shell
* build ……… creates a .bin file from a given .asm file (any file extension should work. .asm just works best) instead of a filename, a path to a file may be used. (eg.: “files/main.asm”)  
  syntax: build <filename>
* exec ………. Executes a .bin file. instead of a filename, a path to a file may be used.  
  (eg.: “files/main.bin”)  
  syntax: exec <filename>

**Logical operators**

* AND:
  + Syntax: AND <output register address> <register address 1> <register address 2>
  + Operates based on this truth table:

|  |  |  |
| --- | --- | --- |
| Input 1 | Input 2 | Output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* ORR:
  + Syntax: ORR <output register address> <register address 1> <register address 2>
  + Operates based on this truth table:

|  |  |  |
| --- | --- | --- |
| Input 1 | Input 2 | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

* + Is called “ORR” instead of “OR” to stay be at the same length as the other operators.
* NOT:
  + Syntax: AND <output register address> <register address>
  + Operates based on this truth table:

|  |  |
| --- | --- |
| Input 1 | Output |
| 0 | 1 |
| 1 | 0 |

* XOR:
  + Syntax: AND <output register address> <register address 1> <register address 2>
  + Operates based on this truth table:

|  |  |  |
| --- | --- | --- |
| Input 1 | Input 2 | Output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Mathematical operators**

* ADD:
  + Syntax: ADD <destination register address> <second number register address>
  + Adds the number at <second number register address> to   
    <destination register address>
* SUB
  + Syntax: SUB <destination register address> <second number register address>
  + Subtracts the number at <second number register address> from the number at  
    <destination register address>
* MUL
  + Syntax: MUL <destination register address> <second number register address>
  + Multiplies the number at <second number register address> with the number at  
    <destination register address>
* DIV
  + Syntax: DIV <destination register address> <second number register address>
  + Integer divides the number at <destination register address> with the number at <second number register address>
  + Integer division example: 5//2 = 2

**IO operators**

* PNL:
  + Prints a newline character (\n)
* PCI:
  + Syntax: PCI <register address>
  + Prints the ASCII character of the number at <register address>
* INP:
  + Syntax: INP <register address>
  + Requests user input. Input must be an integer in range [0,65535]. Will be stored at <register address>

**Data management operators**

* LDI:
  + Syntax: LDI <register address> <number in range [0,15]>
  + Sets the value of the register at <register address> to <number in…>
* LOD:
  + Syntax: LOD <register address> <memory address>
  + Loads a number from the memory at <memory address> to <register address> in the register
* STR:
  + Same as LOD, but “memory” and “register” swapped

**Other operators**

* JIZ:
  + Syntax: JIZ <line number> <register address>
  + Jumps to <line number> if the number at <register address> == 0
* HLT:
  + Stops code execution.
* END:
  + Not required when writing raw binary code. MUST be at the end of your program. It tells the assembler when to stop. Therefore, it doesn’t have a binary opcode.

**Terminology**

The memory is split into two parts, each having “slots” (register: 0-15 memory: 0-65535) that can hold any whole number in range [0, 65535]. Only the “register” can be used for operations. The other part, “data memory” or short “memory” can only be accessed via LOD and STR. Every slot int each part has a value of 0 at the beginning.

|  |
| --- |
| **Note:** JIZ stands for “Jump if Zero”. It is in no way meant to be an inappropriate word. |