OSCR-16

Manual

by- Ranak

version – 0.1.1

OSCR – 16

Contents

Preface

OSCR-16 is a CPU emulator made by Ranak. This emulator emulates a CPU with a data width of 16-bits. It supports a custom assembly.

It’s special features are that the program files can be in any format ,and the file can be read by the emulator as a program file. In short you can edit the program in any text editor.

CPU Specifications

The CPU is very capable in terms of power. But simple enough to be used by a new user.

Clock speed:	1khz / 1000 clock cycles
Ram:	262 kb / 65535 * 16 bits
Rom/Program-file:	65535 lines * infinite files
Data width:	16-Bits
Stack:	1024 addresses * 65535 bits

How to use it

The ui / user interface is pretty straight foreword.

When you run it you will be met with a User Interface, that looks like the following-

```
+-------------------------------------------------------------------------------+
|                  WELCOME TO OSCR - 16                                         |
|This is a cpu emulator made by Ranak                                           |
|CPU specifications:                                                            |
| Ram:262kb  | BUS SIZE:16-bits | 65535 lines of program  |
|Clock speed: 1 khz                                                             |
|Please read the manual before using the emulator!                              |
+-------------------------------------------------------------------------------+

Please enter the file name:
>>>
```

The User Interface will ask you for the Name of the program file.

How to add a program file:

1) Type the Name of the program file.
!Note: Please include the file extension as well

2) The file should be in the same location as the emulator.
!Note: If the file is not in the same directory, try to add the directory of the file.

This small introduction should be enough to let you get started ,now you just need to know the assembly language that is supported by OSCR-16.

ASSEMBLY LANGUAGE

The following are the commands supported by OSCR-16

JMP	JMPZ	JMPN	JMPZ-N
MOV	R2R	INP	OUT
DEL	ALU	ALU|ADD	ALU|SUB
ALU|MUL	ALU|DIV	ALU|INC	ALU|DEC
ALU|AND	NOP	ADD	POP
PUSH	POP	CALL	RET

It is to be noted that between sections there is a divider: '|'
for example:

ALU|ADD|0|1|2

The following is a short description of each command.

1. JMP <line>

Description: Unconditionally jumps to a specific line in the program.
Note: Sets the program counter to the target line (adjusted by –1 since the counter is auto-incremented after each cycle).

2. JMPN <line>

Description: Conditional jump if the result of the previous ALU operation was negative.
Note: Checks if the flag register equals 'N' (negative). If so, it sets the program counter to the given line.

3. JMPO <line>

Description: Conditional jump if an arithmetic overflow occurred.
Note: Jumps to the specified line when the flag register equals 'O' (overflow).

4. JMPZ <line>

Description: Conditional jump if the result of the last operation was zero.

Note: If the flag register is 'Z', indicating a zero result, the program counter is set to the provided line.

5. JMPZ-N <line>

Description: Conditional jump if the last result was neither zero nor negative.
Note: Intended to jump when the flag register is neither 'Z' nor 'N'. (Be aware there may be an implementation detail to review in your code.)

6. MOV <register_index> <value>

Description: Moves a constant value into a specified register.
Note: Assigns the given value to the register at the indicated index.

7. OUT <target> <index>

Description: Outputs the contents of a register or a memory location.
Note:
If target is "REG", it prints the value from the register at the provided index.
If target is "RAM", it prints the value stored in RAM at that index.

8. INP <register_index>

Description: Reads user input and stores it in a register.
Note: Prompts the user, then assigns the entered data to the register at the specified index.

9. ALU <operation> <src_reg1> <src_reg2> <dest_reg>

Description: Executes an arithmetic operation (e.g., addition or subtraction) using the ALU.
Note:
Arithmetic Operations:
ALU ADD: Adds values from two source registers and stores the result in the destination register.
ALU SUB: Subtracts the second source register from the first and stores the result.
ALU DIV: Divides from two source registers and stores in a different register
ALU MUL: Multiplies and stores the data in the target register

ALU <operation> <src_and_result_reg>
ALU INC: Increments (adds by 1) and stores the data in the same register

ALU DEC: Decrements (subtract by 1) and stores the data in the same register
ALU LSH: Shifts the data to left by no. of bits assigned
AIU RSH: Shifts the data to right by no. of bits assigned

Boolean logic:

ALU ADD: Checks if the tow data in the registers are same or not and returns TRUE or FALSE in the target register

After the operation, flags are updated as follows:
Sets 'Z' if the result is zero.
Sets 'N' if the result is negative.
Sets 'O' if the result is equal to or exceeds 65535 (overflow).

10. DEL <time_in_seconds>

Description: Introduces a delay in the program execution.
Note: Pauses the CPU emulator for the specified duration (using time.sleep).

11. HALT

Description: Stops the execution of the emulator.
Note: When encountered (or triggered by error conditions such as an out-of-range value), the emulator terminates its main loop.

12. PUSH <register_address>

Description: It adds the data stored in the specified register and increments the stack pointer.

13. POP <register_address>

Description: It returns the value stored on the stack stack pointer is ‘pointing’ and stores in the specified register and decrements the stack pointer.

14. CALL <register_address>

Description: It can be used to make functions, it stores the <current_count> in the stack, and jumps to th specified address.

15. RET

Description: It can be used to return to the last saved location.

Example prgrams

Printing Hello World:

In the state that the emlator is it also accepts string type data into the register ,which means you can directly store the data in the register and then print it.

PROGRAM:

```
MOV|REG|0|HELLO WORLD!
OUT|REG|0
HALT
```

OUTPUT:
>>>HELLO WORLD!

If you are really naughty ,you can print this infinite times.

PROGRAM:

```
MOV|REG|0|HELLO WORLD!
OUT|REG|0
JMP|2              // UNCONDITIONAL JUMP
HALT               // THE HALT IS NEVER REACHED
```

Adding two data and doing all the alu functions on it:

Program:

```
INP|0                       //Accepting the data
INP|1
ALU|ADD|0|1|2               //Doing all the oprations
R2R|RAM|0|2                 // Saving the data in the ram
ALU|SUB|0|1|2
R2R|RAM|1|2
ALU|MUL|0|1|2
R2R|RAM|2|2
ALU|DIV|0|1|2
R2R|RAM|3|2
ALU|AND|0|1|2
R2R|RAM|4|2
ALU|LSH|1|0
R2R|RAM|5|0
ALU|RSH|1|1
R2R|RAM|6|1
NOP                         // Outputing all the data
OUT|RAM|0
OUT|RAM|1
OUT|RAM|2
OUT|RAM|3
OUT|RAM|4
OUT|RAM|5
OUT|RAM|6
HALT
```