
University of Trento
Department of Physics

# Developments of a FPGA-based Intel 8080 Mockup Soft Microprocessor

*Leonardo Cattarin*

Laboratory of Advanced Electronics
*Professor: Ricci Leonardo*

## 1 Introduction

*Motivation, what a microprocessor is, what is the Von Neumann architecture vs Harvard architecture* *Description of the General Features*

## 2 Components and Implementation

(Block diagram) The whole computing system is composed by three main modules (CPU, RAM and LCD debug module) and by various other auxiliary modules. The CPU comprises various 8-bit registers and uses them to fetch data from the RAM module, process it and, eventually write it on the RAM. The LCD Debug driver allows to access both the stored RAM bytes and the CPU registers from the LCD screen. North and East pushbuttons are stabilized and used along with a switch to change the CPU registers or RAM values shown on the screen. Both the CPU and RAM run on three different clocks: a 50 MHz "Master" synchronization clock, a 1kHz debug clock used to comunicate with the LCD driver independently from the other operations and a last clock, sourced via the South pushbutton, and used to govern the actual computations and input/output operations between CPU and RAM. The RAM is pre-loaded with the data and instructions during the FPGA configuration phase.

### 2.1 CPU Module

*Block scheme/image of CPU internal registers and buses* *scheme of states for some instructions? (initial 3 states, then example of other instructions)*

The CPU module behaves like a state machine which fetches and executes one instruction at time from the RAM in the form of single-byte operation codes ("opcodes"). If the instruction requires it, the CPU can also fetch other bytes as parameters.. Each instruction execution is divided into several states, kept track by the `state` register.

Various other 8-bit registers are present to help store, process and write data. At each `clk_in` edge risetime, the module behaviour is determined by the current instruction and state. When the last state corresponding to an instruction is reached, `state` register is reset to 0 and the process restart by fetching the next instruction.

In general, the instruction fetch-execution cycle behaves as follows:

- `state` 0-2: The next instruction is fetched from the memory location pointed by the Program Counter ( `PC` ) and placed in the Instruction Register (`IR`).
- `state` 3- : The instruction is executed, eventually by reading or writing other bytes from memory or by operating on the internal registers. At the end, the `PC` address is incremented by one and `state` is reset to 0. The cycle then restarts

The following is a list of the internal CPU module registers:

- `PC` : (8-bit) Program Counter, contains the memory address to the next instruction.
- `IR` : (8-bit) Instruction Register, Contains the byte corresponding to the current instruction in execution
- `A` : (8-bit) Accumulator, Register used to store fetched data and used by default as result recipient by the ADD instruction
- `B,C` : (8-bit) General purpose registers
- `W,Z` : (8-bit) Temporary registers for internal instruction operations, not directly accessible via instructions
- `H,L` : (8-bit) Registers used to store memory addresses, for instance used to write from register to referenced memory location
- `data_addr,data_out,write_en` : Registers connected to output wirebuses, used to communicate read/write informations to RAM
- `flg_carry,flg_sign,flg_zero,flg_parity,flg_auxiliary` : Various flags registers used to keep track of the results of the instructions. `flg_auxiliary` is used as general-purpose flag to reduce the number of required instructions.

As a side note, the input/output operations usually require additional "buffering" times (blank states) to allow address informations delivery or data fetches.

## 2.2 RAM Module

The `Module_BRAM_256_byte` module is based on the usage of the "Block Ram" modules present in the Xilinx Spartan 3A. From the logical and Verilog code points of view, the Block RAM cells can be declared as arrays of $n$-bit registers. In this specific module, the Block ram used is organized as an array of 256 8-bit memory cells, corresponding to a memory with 8-bit addresses. The memory content is initialized using the Verilog command:

```
initial
begin
$readmemh("memory.data", RAM, 0, 255);
end
```

which prescribes how to load the RAM content from the file `memory.data` during the device configuration phase.

At each `clk_in` tick, the RAM either reads or writes the content at address `addr` according to `write_en` by using the wires/registers `data_in` , `data_out`

## 2.3 LCD driver Module

The `LCD_Driver_Dbg` module is used to visualize the content of both the RAM module and the internal CPU registers. The informations to be shown on display are inputed via the `addrInput, dataInput, CPU_interface, dbg_reg` wirebuses. The user can select via switch one of the two devices, and further use the North and East buttons to move between the memory addresses or CPU registers (implemented via a "Ladder" counter). These informations are then passed to the here considered module via `addrInput` and `dbg_reg` .

The LCD screen comprises a display data RAM used to store the shown characters, and communicates with the FPGA board via the following lines:

- `LCD_DB` : an 8-bit "Data bit" wirebus. Here is used in 4-bit mode, therefore the lower 4 bits are set to high. 4-bit mode requires to perform all the operations by sending separately the upper and lower nible of each 8-bit signal.
- `LCD_E` : single-bit Read/write enable pulse, used to comunicate to the display that the other buses are rady for operating.
- `LCD_RS` : single-bit register select.
- `LCD_RW` : read/write control, here permanetly set to 0 for write-only mode

Operating the LCD screen via Driver requires the following steps:

- Power-on initialization: initialization phase, the data bits are set and `LCD_E` is pulsed according to required patterns and timings.
- Display configurations: Here multiple commands are issued to configure the device functioning:
    - Function set
    - Entry mode set: address counter is set to auto-increment
    - Display on, cursor and blinking are set off
    - Clear Display
- Character writing

Disclaimer: the LCD driver module here used is based on a modified version of code not originally writtend by the author.

# 3 Instruction Set and Code Development

## 3.1 Instructions

## 3.2 Basic Examples

*test JUMP, etc*

### 3.3 Example: For Loop

The following example loops over various instructions to generate the natural numbers up to a certain value `??` and write them on memory starting from a chosen location *code and explaination*

# 4 Conclusion

# 5 Appendix:

### 5.1 Sample Subsection

Sample Text [1]

# References

[1]N. Surname, *Text Title*, `https://sampletext.noex`, 2021.