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* *Description of an assembly language* *Block scheme of the device*

1.1 Sample Subsection

Sample Text [1]

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?*

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.

comments on the number of minimal instructions

The list of the implemented instructions is presented in Appendix A.

2.2 RAM Module

The `Module_BRAM_256_byte` module is based on the usage of Block Ram, 256 8-bit Block-Ra

2.3 LCD driver Module

Sample Text [1]

3 Instruction Set and Code Development

explaination of the problem *implementation and explaination of the code*

3.1 Sample Subsection

Sample Text [1]

4 Conclusion

5 Appendix

explaination of the problem *implementation and explaination of the code*

5.1 Sample Subsection

Sample Text [1]

References

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