Microcontroller

June 2024

Raluca-Mihaela Adam, Eduard-Paul Cavași

PROJECT SUPERVISOR: Ing. Diana Pop

Contents

1. Specifications

The aim of this project is to design a fully embedded 8-bit microcontroller for the Artix-7 100T FPGA trainer board. It will feature a single block of RAM used to form a ROM store for a program of up to 256 instructions. To run the program on the FPGA board, an assembler will be implemented.

The microcontroller features:

- 16 general purpose 8-bit registers without default priorities.
- Carry and Zero flags.
- Execution of basic arithmetic and logical operations.
- Execution of Jump instructions, both conditional and unconditional.
- Display of data of interest for the user on two SSD displays.
- asynchronous reset signal, which enables the user to reinitialize the processor and jump to address x00 in the program memory.

For debugging purposes, the program ROM consists of a small script to test the implemented instructions. The results are displayed on two SSD`s on the FPGA board. The frequency of the clock signal from the board is drastically decreased in order to enable the user to observe the results of each operation.

Remark: The execution of all instructions, under all conditions, expands over 3 clock cycles.

2. Microcontroller Instruction Set

This section lists a complete instruction set representing all operation-codes.

1. “X” and “Y” refer to the definition of the storage registers “s” in range 0 to F.

2. “kk” represents a constant value in range 00 to FF.

3. “aa” represents an address in range 00 to FF.

4. “pp” represents a port address in range 00 to FF

Program Flow Control Instructions

JUMP aa
JUMP Z, aa
JUMP NZ, aa
JUMP C, aa
JUMP NC, aa

Logical Instructions

LOAD sX, kk
AND sX, kk
OR sX, kk

XOR sX, kk
LOAD sX, sY
AND sX, sY
OR sX, sY
XOR sX, sY

Arithmetic Instructions

ADD sX, kk
ADDCY sX, kk
SUB sX, kk
SUBCY sX, kk
ADD sX, sY
ADDCY sX, sY
SUB sX, sY
SUBCY sX, sY

Shift and Rotate Instructions

SR0 sX
SR1 sX
SRX sX
SRA sX
RR sX
SL0 sX
SL1 sX
SLX sX
SLA sX
RL sX

Program Control Group

JUMP

Normally, the program counter increments to point to the next instruction. The address space is fixed to 256 locations (00 to FF hex); therefore, the program counter is 8-bits wide. The top of the memory is FF hex and increments to 00.
(PC = Program Counter)

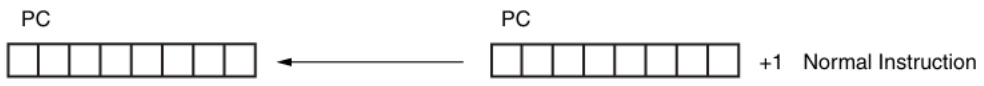


FIGURE 1. INCREMENTING INSTRUCTIONS IN THE PC

The JUMP instruction is used to modify the address sequence, by specifying a new address. The instruction can be conditional. A conditional JUMP is only performed if a test performed on either the ZERO flag or CARRY flag is valid. The JUMP instruction has no effect on the status of the flags.

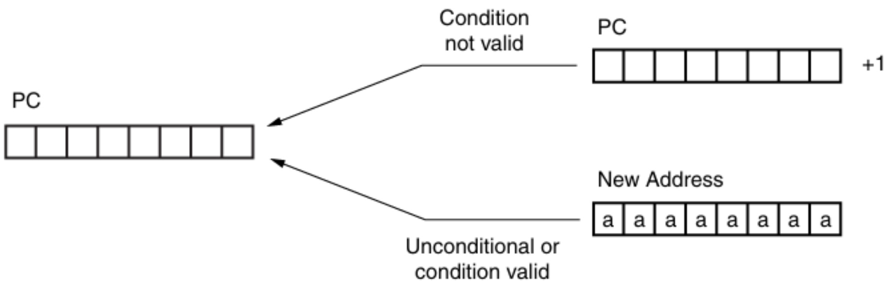
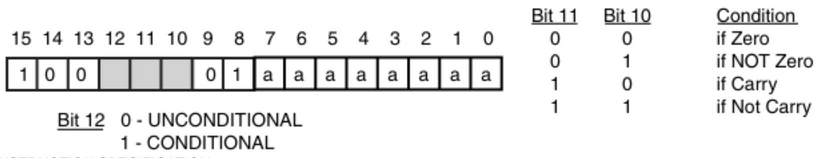


FIGURE 2. JUMP INSTRUCTION

The JUMP instruction has the following specification:



Bit 11	Bit 10	Condition
0	0	if Zero
0	1	if NOT Zero
1	0	if Carry
1	1	if Not Carry

FIGURE 3. JUMP INSTRUCTION SPECIFICATION

Remark:
For all the following instructions involving two operands we note that: the first operand is any register, and it is this register that is assigned the result of the operation.

Logical Group

LOAD

The LOAD instruction specifies the contents of any register. The new value is either a constant or the contents of any other register. The LOAD instruction has no effect on the status of the flags.

- LOAD s0,s0

Loading any register with its own contents achieves nothing and hence is a NO OPERATION consuming three clock cycles. This may be used to cause a delay in the program.

- LOAD sX,00

Loading zero is the equivalent of clearing the register.

The LOAD instruction has the following specification:

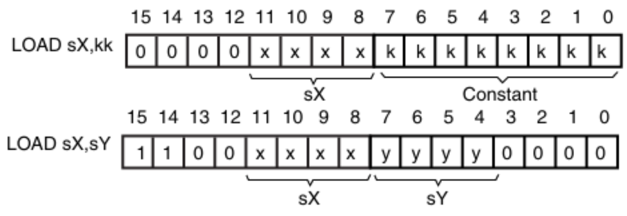


FIGURE 4. LOAD INSTRUCTION SPECIFICATION

AND

The AND instruction performs a bit-wise logical AND operation between two operands (two registers, or a register and a constant). Flags are affected by this operation. The ZERO flag is set if all bits of result are zero and reset in all other cases.
The AND instruction has the following specification:

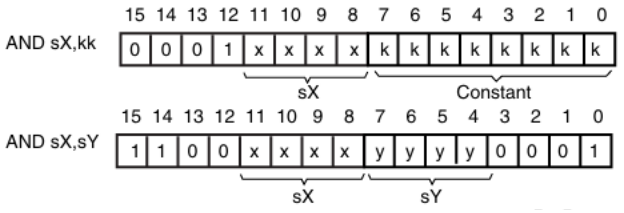


FIGURE 5. AND INSTRUCTION SPECIFICATION

OR

The OR instruction performs a bit-wise logical OR operation between two operands (two registers, or a register and a constant). Flags are affected by this operation. The ZERO flag is set if all bits of result are zero and reset in all other cases.
The OR instruction has the following specification:

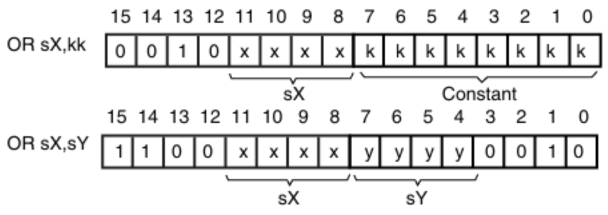


FIGURE 6. OR INSTRUCTION SPECIFICATION

XOR

The XOR instruction performs a bit-wise logical XOR operation between two operands (two registers, or a register and a constant). Flags are affected by this operation. The ZERO flag is set if all bits of result are zero and reset in all other cases.
The XOR instruction has the following specification:

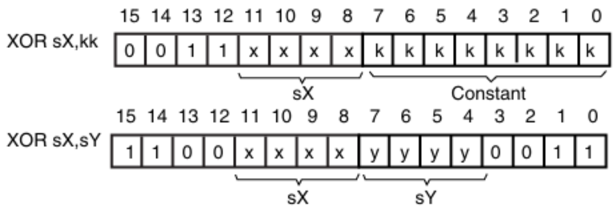


FIGURE 7. XOR INSTRUCTION SPECIFICATION

Arithmetic Group

ADD

The ADD instruction performs an 8-bit addition of two operands (two registers, or a register and a constant). Flags are affected by this operation. The CARRY flag is set if result of addition exceeds FF and reset in all other cases. The ZERO flag is set if all bits of result are zero and reset in all other cases.
The ADD instruction has the following specification:

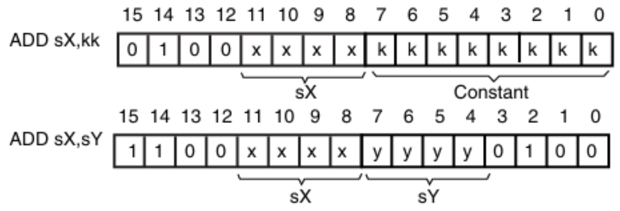


FIGURE 8. ADD INSTRUCTION SPECIFICATION

ADDCY

The ADDCY instruction performs an addition of two 8-bit operands together with the contents of the CARRY flag. Flags are affected by this operation. The CARRY flag is set if result of addition exceeds FF and reset in all other cases. The ZERO flag is set if all bits of result are zero and reset in all other cases.
The ADDCY instruction has the following specification:

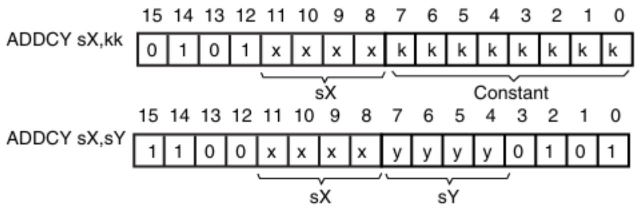


FIGURE 9. ADDCY INSTRUCTION SPECIFICATION

SUB

The SUB instruction performs an 8-bit subtraction of two operands. Flags are affected by this operation. The CARRY flag is set if result is negative and reset in all other cases. The ZERO flag is set if all bits of result are zero and reset in all other cases.
The SUB instruction has the following specification:

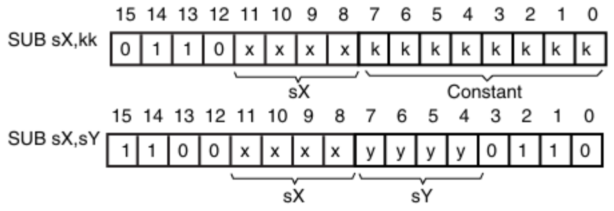


FIGURE 10. SUB INSTRUCTION SPECIFICATION

SUBCY

The SUBCY instruction performs an 8-bit subtraction of two operands together with the contents of the CARRY flag. Flags are affected by this operation. The CARRY flag is set if result is negative and reset in all other cases. The ZERO flag is set if all bits of result are zero and reset in all other cases.
The SUBCY instruction has the following specification:

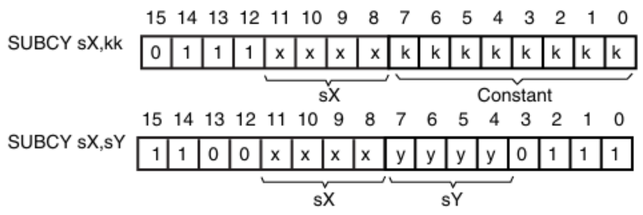


FIGURE 11. SUBCY INSTRUCTION SPECIFICATION

Shift and Rotate Group

SR0, SR1, SRX, SRA, RR

The shift and rotate right group all modify the contents of a single register. All instructions in this group affect the flags.

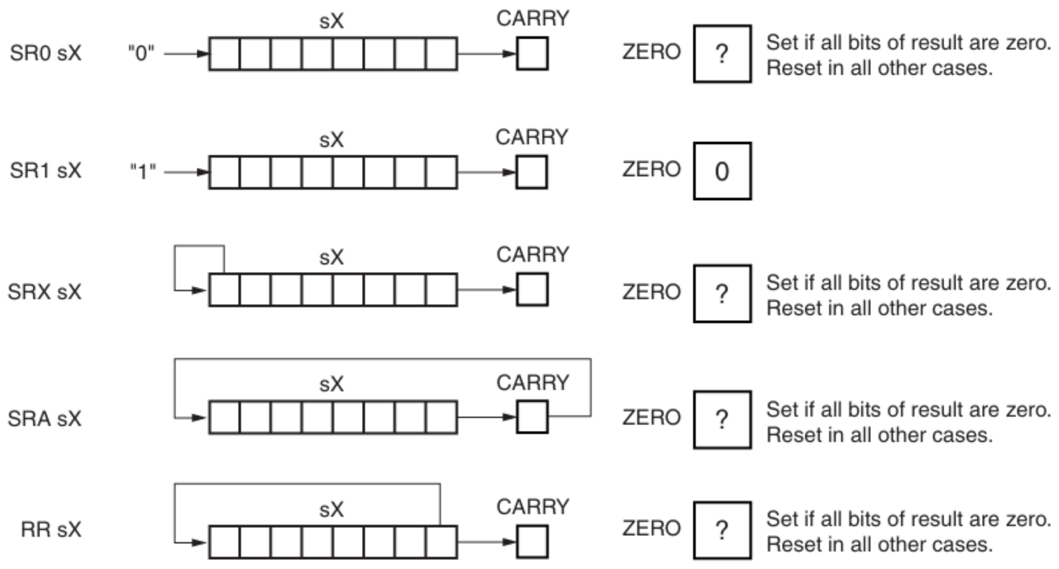
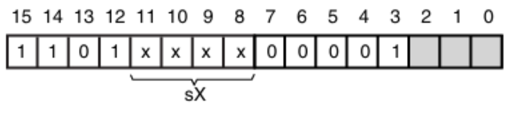

SR0 sX
"0"
sX
CARRY
ZERO
?
Set if all bits of result are zero.
Reset in all other cases.
SR1 sX
"1"
sX
CARRY
ZERO
0
SRX sX
sX
CARRY
ZERO
?
Set if all bits of result are zero.
Reset in all other cases.
SRA sX
sX
CARRY
ZERO
?
Set if all bits of result are zero.
Reset in all other cases.
RR sX
sX
CARRY
ZERO
?
Set if all bits of result are zero.
Reset in all other cases.

FIGURE 12. SHIFT AND ROTATE RIGHT GROUP

Instruction specification:



Bit 2	Bit 1	Bit0	Instruction1
1	1	0	SR0 sX
1	1	1	SR1 sX
0	1	0	SRX sX
0	0	0	SRA sX
1	0	0	RR sX

FIGURE 13. SHIFT AND ROTATE RIGHT INSTRUCTION SPECIFICATION

SL0, SL1, SLX, SLA, RL

The shift and rotate left group all modify the contents of a single register. All instructions in this group affect the flags.

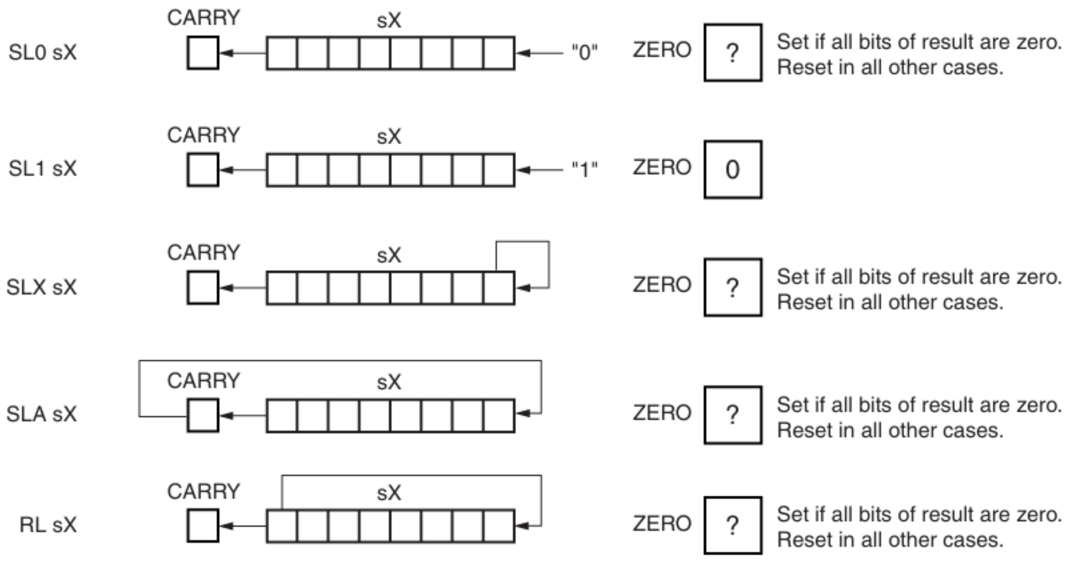
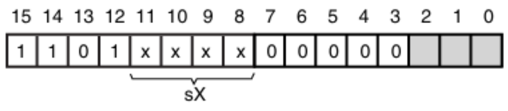

SL0 sX
CARRY
sX
"0"
ZERO
?
Set if all bits of result are zero.
Reset in all other cases.
SL1 sX
CARRY
sX
"1"
ZERO
0
SLX sX
CARRY
sX
ZERO
?
Set if all bits of result are zero.
Reset in all other cases.
SLA sX
CARRY
sX
ZERO
?
Set if all bits of result are zero.
Reset in all other cases.
RL sX
CARRY
sX
ZERO
?
Set if all bits of result are zero.
Reset in all other cases.

FIGURE 14. SHIFT AND ROTATE LEFT GROUP

Instruction specification:



Bit 2	Bit 1	Bit0	Instruction1
1	1	0	SL0 sX
1	1	1	SL1 sX
1	0	0	SLX sX
0	0	0	SLA sX
0	1	0	RL sX

FIGURE 15. SHIFT AND ROTATE LEFT INSTRUCTION SPECIFICATION

3. Design

3.1. Black Box

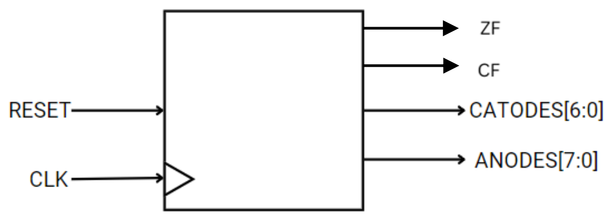


FIGURE 16. BLACK BOX OF THE SYSTEM

The figure above illustrates the Black Box of the Microcontroller. Its Input / Output ports are described in greater detail in section 3.2.2. Input and Output Mapping.

3.2. Control and Execution Unit

3.2.1. Control Unit

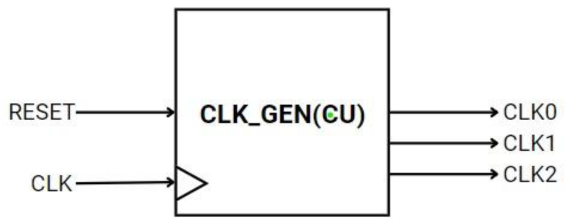


FIGURE 17. CONTROL UNIT

Processing each instruction must be broken down into several steps. For this purpose, the system has 3 states: “100” (CLK0), “010” (CLK1), “001” (CLK2). The Control Unit generates the states and controls the transitions in between them.
CLK0 & CLK1 & CLK2 represent the current state of the whole system.
State “100” –instruction fetching.
State “010” – current instruction decoding.
State “001” – register loading and next state calculation

3.2.2. Input and Output Mapping

Inputs:

The control inputs are CLK and RESET.

The CLK runs at a 100MHz frequency and is generated by the FPGA board.

The RESET is asynchronous.

Outputs:

The CATODES and ANODES are used for display purposes. The result of each arithmetic / logical instruction being displayed on 2 SSDs (for debugging purposes we only display the content of register 0).

The FLAGS serve the purpose of signaling different things depending on the operation which is performed.

ZF = Zero Flag

CF = Carry Flag

3.2.3. Resources (breakdown of the EU)

Frequency Divider

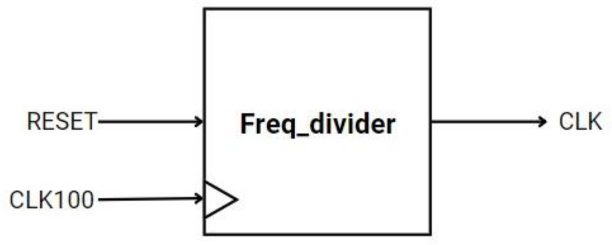


Figure 18. Frequency divider

The Frequency Divider is used for debugging purposes only, in order to ensure that the instructions are delayed long enough for the human eye to perceive them. It divides the FPGA board clock (CLK100), running at 100MHz, into 1Hz (CLK).

8-bit Register

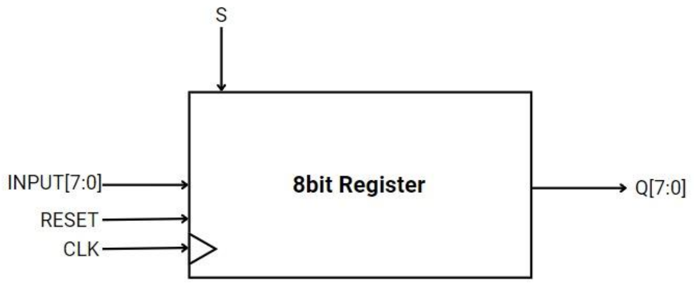


FIGURE 19. 8-BIT REGISTER

There are 16 8-bit registers.
Each of them performs LOAD and HOLD operations, based on selection input S.
RESET is asynchronous.
CLK = CLK2 (register loading)

Program ROM

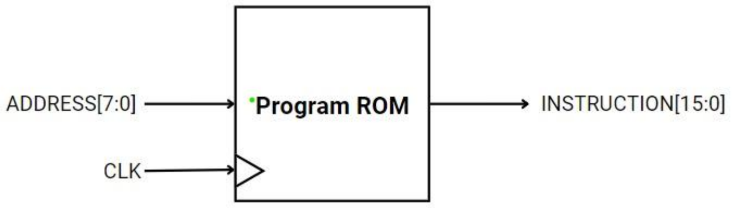


FIGURE 20. PROGRAM ROM

The sequence of instructions to be executed is stored in the Program ROM. Each instruction is manually introduced by the user at the desired address.
Instruction fetching is done in state “100”, so CLK = CLK0.

Arithmetic Logic Unit

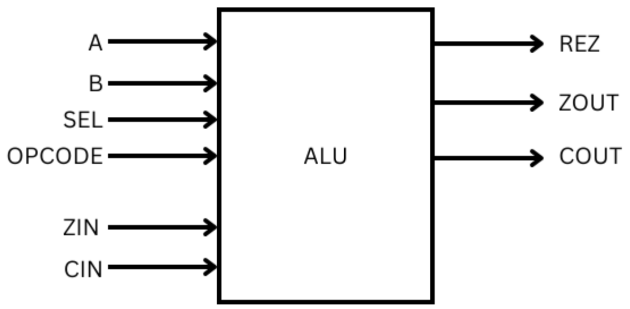


FIGURE 21. ALU

This combinational component executes the arithmetic, logical and load instructions.
In the case of the shift instruction SEL = “1101” and the type of the shift will be dictated by OPCODE (shifting will be done on operand A).
In the case of all the other instructions SEL dictates the operation (between operands A and B).

Instruction Decoder

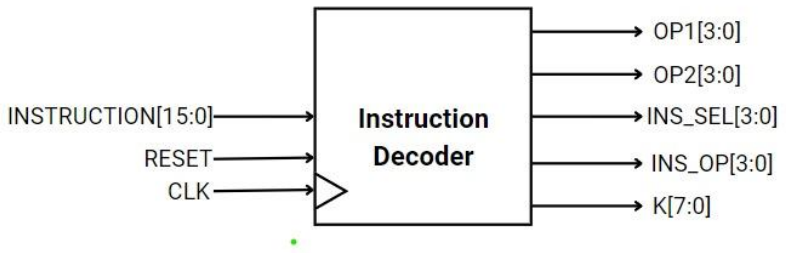


FIGURE 22. INSTRUCTION DECODER

The Instruction Decoder breaks down instructions into the addresses of the operands (OP1, OP2), the value of the constant when the operation requires it (K) and the type of the operation which will be encoded into (INS_SEL, INS_OP).

Jump Control

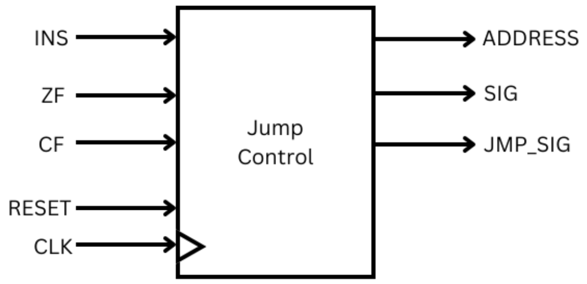


FIGURE 23. JUMP CONTROL UNIT

The Jump Control decodes INS and determines whether it represents a JUMP instruction. If true, it checks if the JUMP is conditional or unconditional and activates signals SIG and JMP_SIG.
Output signal SIG will activate LOAD in the Program Counter if a jump is performed and ADDRESS will be loaded in the Program Counter in that case.
CLK = CLK1 (instruction decoding) – this component works in parallel with the instruction decoder.

Program Counter

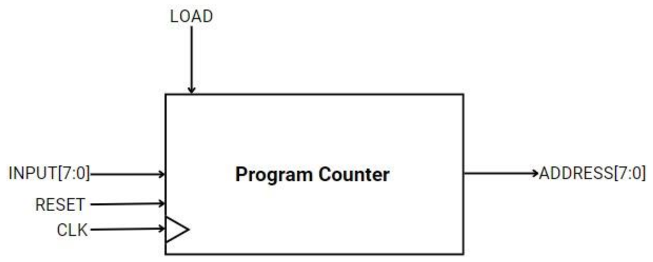


FIGURE 24. PROGRAM COUNTER

The Program Counter is a direct counter on 8 bits, with synchronous LOAD and asynchronous RESET.
It counts starting from address x00 up to x"FF" each time the Microcontroller is initialized.
CLK = CLK2 (next state calculation).

Flags Register

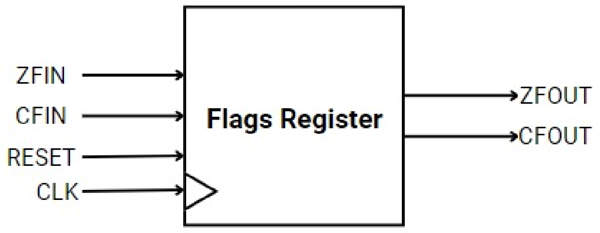


FIGURE 25. FLAGS REGISTER

The ALU operation results affect the ZERO and CARRY flags. Their values are therefore stored in a register.
This information determines the execution sequence of the program using conditional program flow control instructions such as conditional JUMPS.
CLK = CLK2 (register loading)

Combinational Circuit (register / constant use)

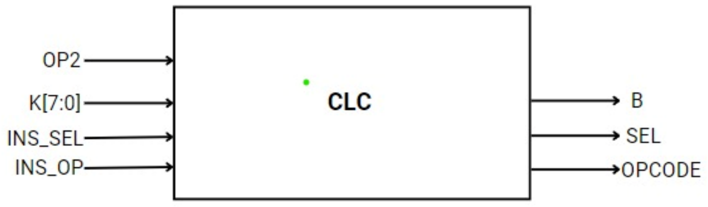


FIGURE 26. CLC

If the operation is done with a constant, then INS_SEL dictates the operation. Otherwise, when using a register, INS_SEL will always be “1100”, and the INS_OP will dictate the instruction.
Thus, B will receive the value of the second operand (either stored in a register or a constant) and SEL will receive the operation code.
OPCODE is important only in case of shifting when SEL = “1101”.

3.2.4. Register Loading

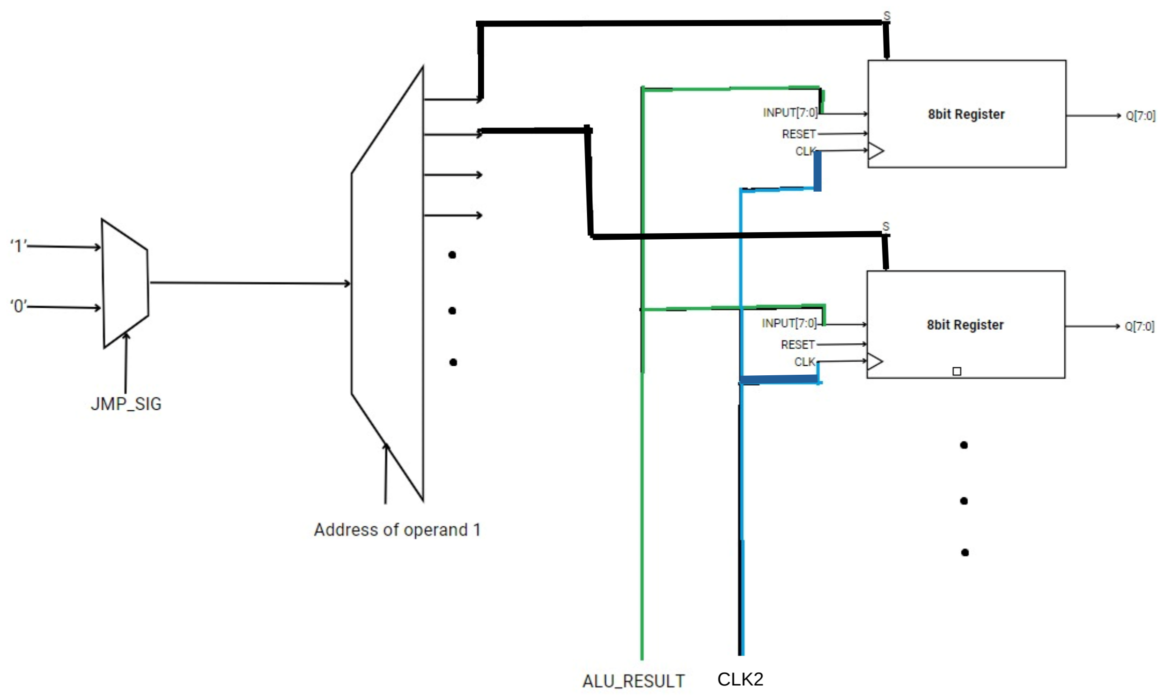
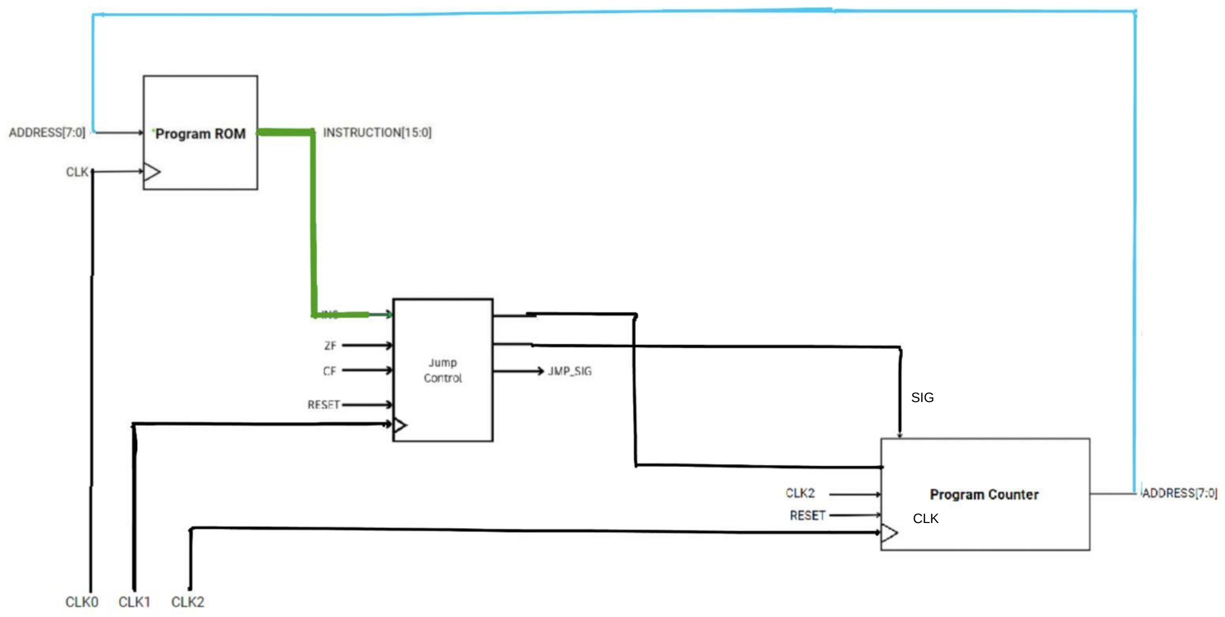


FIGURE 27. REGISTER LOADING DIAGRAM

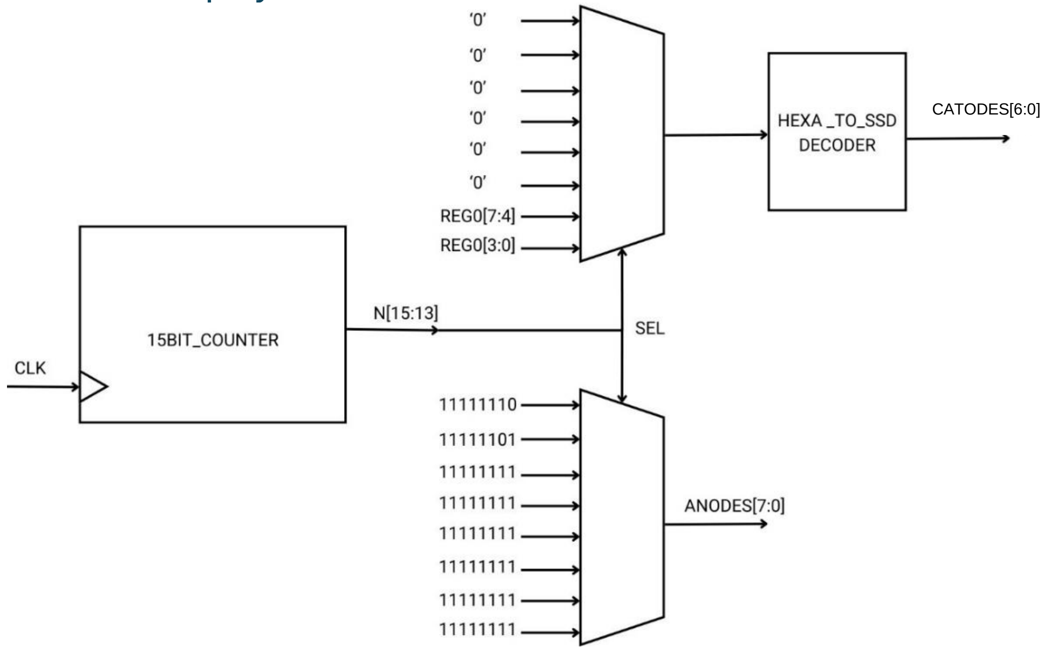
In state “010”, after decoding the instruction the ALU performs the operation, storing the result in ALU_RESULT. Afterwards, in state “001”, the previously calculated result is loaded into the corresponding register. Every register has on its INPUT the ALU_RESULT. Using a 1:16 DEMUX, the desired register’s load signal is activated (all the other load signals are deactivated). This only happens when a previous operation has been done. Thus in the case of JUMP instructions (signaled by JMP_SIG), when no operation is performed, all the register’s load signals are deactivated and no load is performed.

3.2.5. Next Instruction Handling



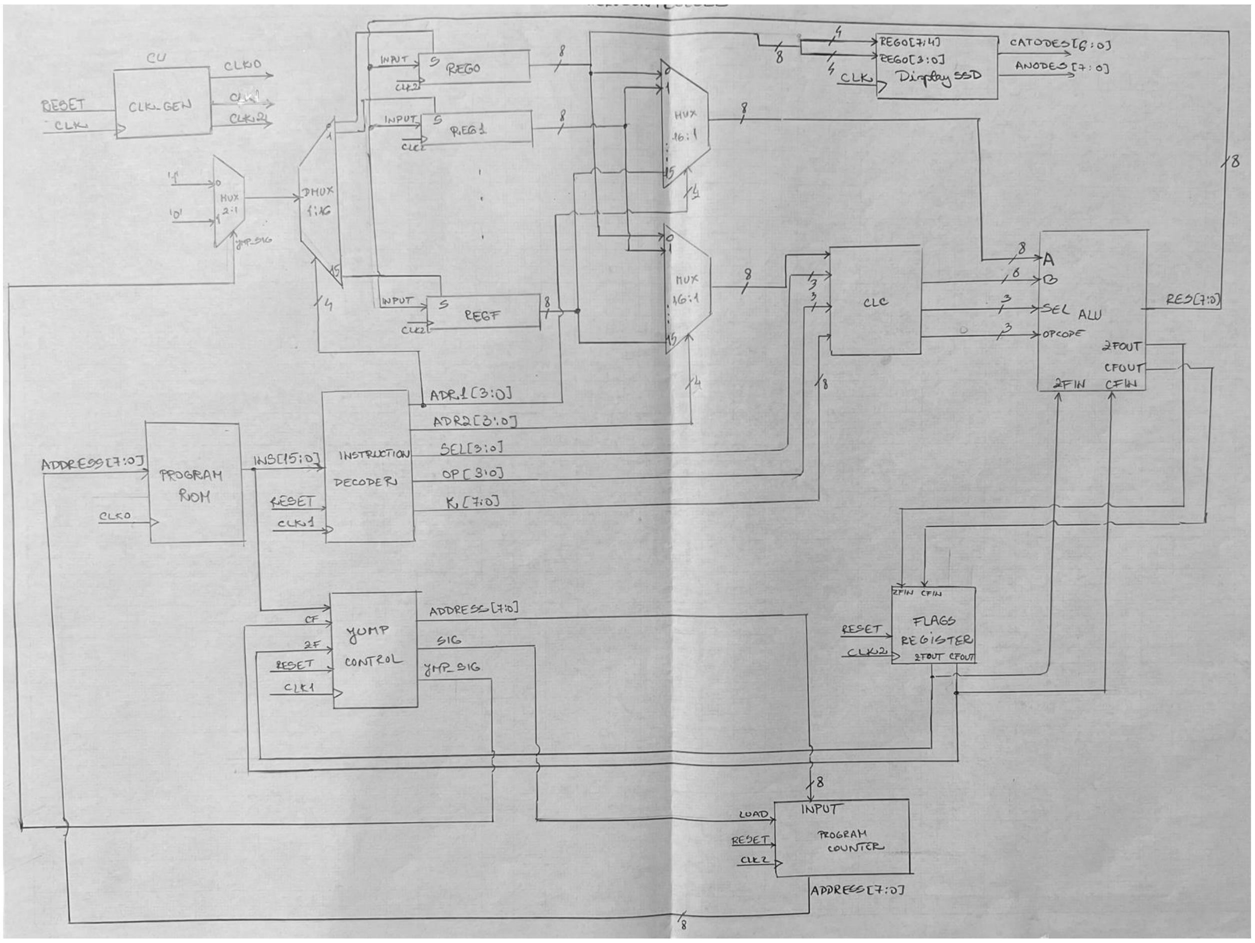
The Jump Control analyses the current instruction and if it is a jump instruction and the conditions for it are satisfied, the signal SIG becomes ‘1’ and activates the load mode of the Program Counter, loading it with the address specified by the jump. Otherwise, the program counter just increments the current address.

3.2.6. Display SSD



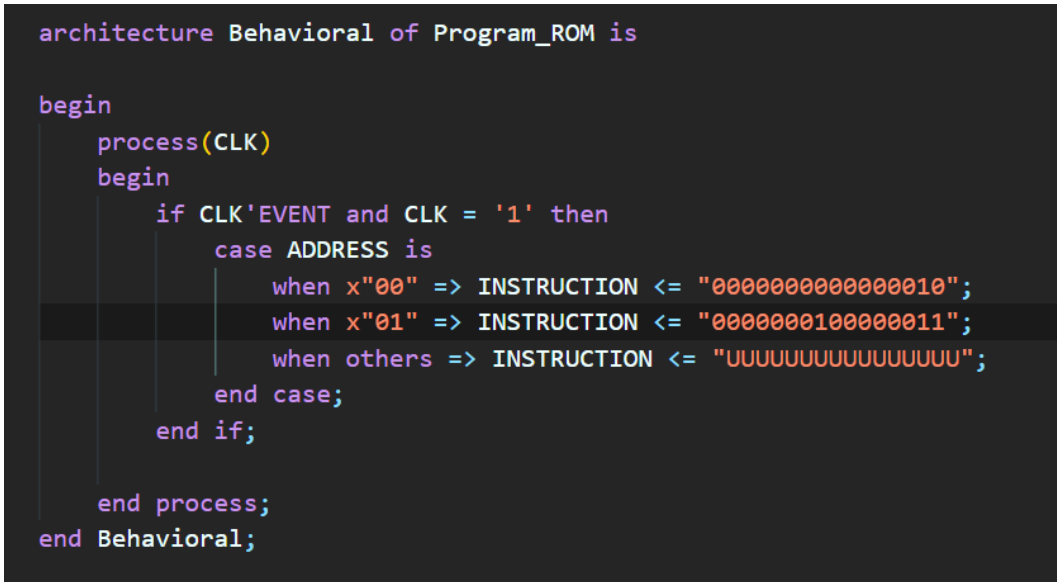
The DisplaySSD module is used to verify the results of the instructions and observe them on the FPGA board.
The content of Register0 is displayed in hexadecimal. In order to do this, 2 SSD's are used, being activated alternatively at a very high frequency, thus creating the illusion that they display simultaneously. Information is encoded in the CATODES signal, and the activation of the SSD's is done using the ANODES signal.

3.2.7. Detailed Diagram of the Project



4. User Manual

The user must be familiar with the instruction format for each operation (a short description can be found in section 2. Microcontroller Instruction Set).



```vhdl
architecture Behavioral of Program_ROM is

begin
    process(CLK)
    begin
        if CLK'EVENT and CLK = '1' then
            case ADDRESS is
                when x"00" => INSTRUCTION <= "0000000000000010";
                when x"01" => INSTRUCTION <= "0000000100000011";
                when others => INSTRUCTION <= "UUUUUUUUUUUUUUUU";
            end case;
        end if;

    end process;
end Behavioral;
```

FIGURE 28. PROGRAM ROM CONTENT

The desired instructions (up to 256) are written in the Program ROM as shown above. Afterwards, the bitstream is generated and operation results can be seen on the FPGA board.

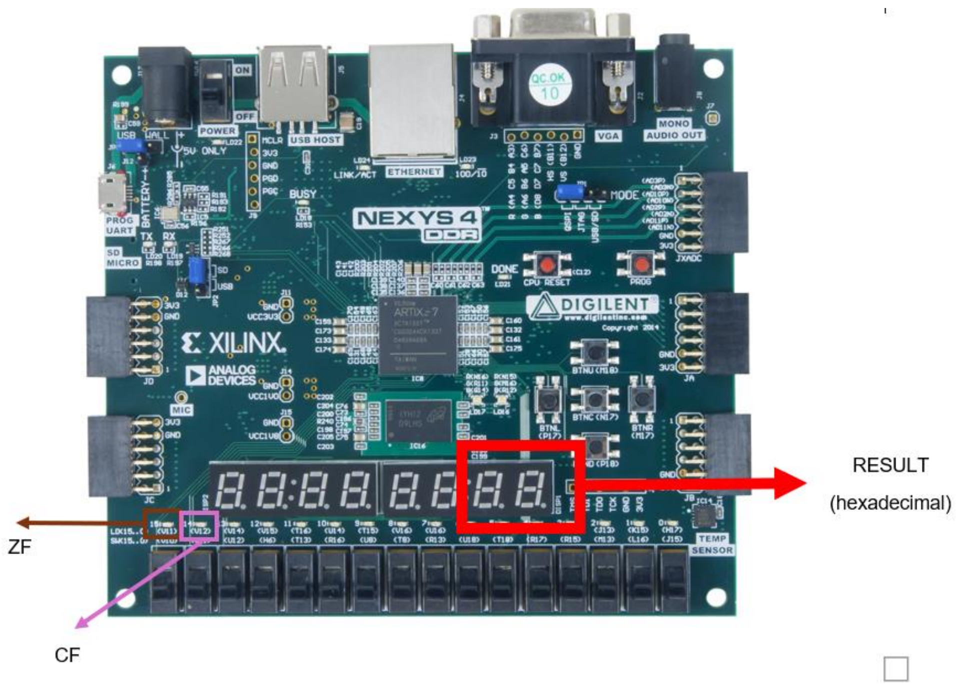


FIGURE 29. FPGA BOARD

5. Future Developments

To further develop the Microcontroller, several functionalities can be implemented:

- Function Calls - which are similar to JUMPS, but they store the current address and restore it after executing the function.
- Interrupts – special functions implemented in the Microcontroller.
- Input / Output ports, to enhance user interaction.

6. References

- PicoBlaze 8-Bit Microcontroller for Virtex-E and Spartan-II/IIE Devices; Author: Ken Chapman, XAPP213 (v2.1) February 4, 2003.
- Introduction to VHDL (lab manual).