Faculty of Engineering ZU,

Electronics and Communications

Engineering Dept.

ECE 442b: Electronics Measurements(Lab)

Eng/ Asmaa Tantawy.

Project Report

"8-Bit Simple Processor."

Year: *Spring 2020* ***Semester***: 2^{nd}

Name: *Ahmed Elsayed Ahmed Abdelazeem.*

Sec: *1.*

***Abstract*:**

Based on FPGA, the hardware description language VHDL and the top-to-down design method of modularization are adopted to achieve the design of a simple Microprocessor logic controller, in which the software design and simulation are conducted for each module of the Microprocessor, and then each module is synthesized. The principle analysis is performed for the key function modules of Microprocessor. The function-sequence simulation diagrams are given by means of compilation and adaptation with software Xilinx Spartan 3 xc3s200FT256 using ISE 14.7. ***.*** *The result suggests that the design has a strong flexibility, reliability, and easiness for extension. The Microprocessor can achieve more functions as long as extending or amending the instructions appropriately.*

I. INTRODUCTION:

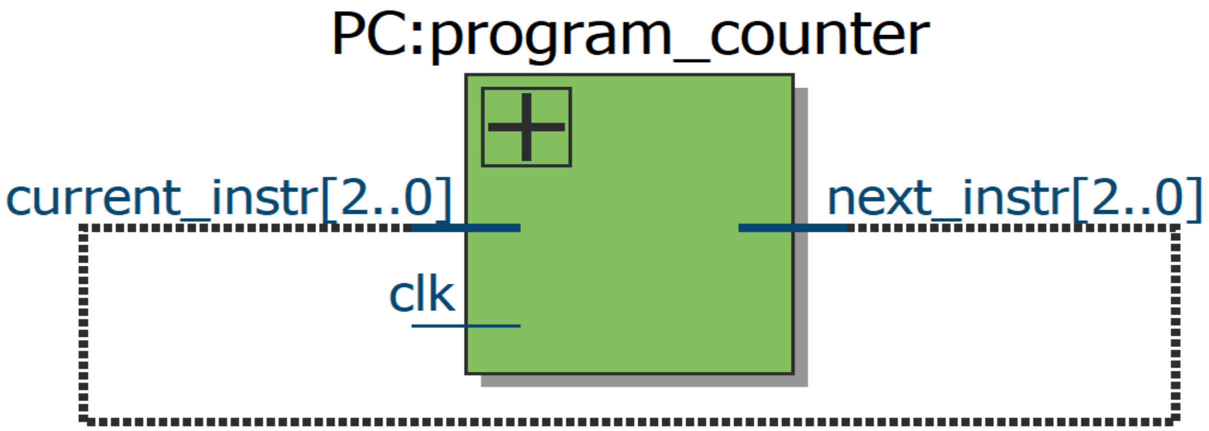
Hardware Description Languages is used to design the behavior of logic circuits. In the older days, early Intel's and other processors were designed by hand, laying out the layers of an IC's substrate masks using regular drafting techniques. There were little or no computer-aided design tools to help the chip developer. This method was so tedious that the least and few people had the patience and skills for such a task. Thankfully, times have changed and designing custom processors are within reach of many designers of such hobby. HDLs give a way to describe and implement a hardware design in a similar manner as developing a software program. Besides, there are similar tools like compilers, debuggers and simulators. There are two predominate HDL language, Verilog and VHDL. VHDL is adopted in this paper.

Based on the relevant knowledge on the principles of the Microprocessor composition, the paper designs a simple Microprocessor by using top-down approach and completed all of the functional modules using the hardware description language VHDL and EDA technology. The functional modules include the control module, arithmetic and logic module, the general register group, the program counter, instruction register, input device, as well as output device. It is designed, compiled and simulated under the integrated development environment of Xilinx ISE 14.7, and after that it is downloaded to the FPGA experimental platform for units and system testing.

II. *Program Counter (PC)*

Send the address of current instruction to Instruction Memory and increase every falling edge of clock.

```
if falling_edge(clock) then
    next_address <= current_address + 1
end if
```



III. *Instruction Memory*

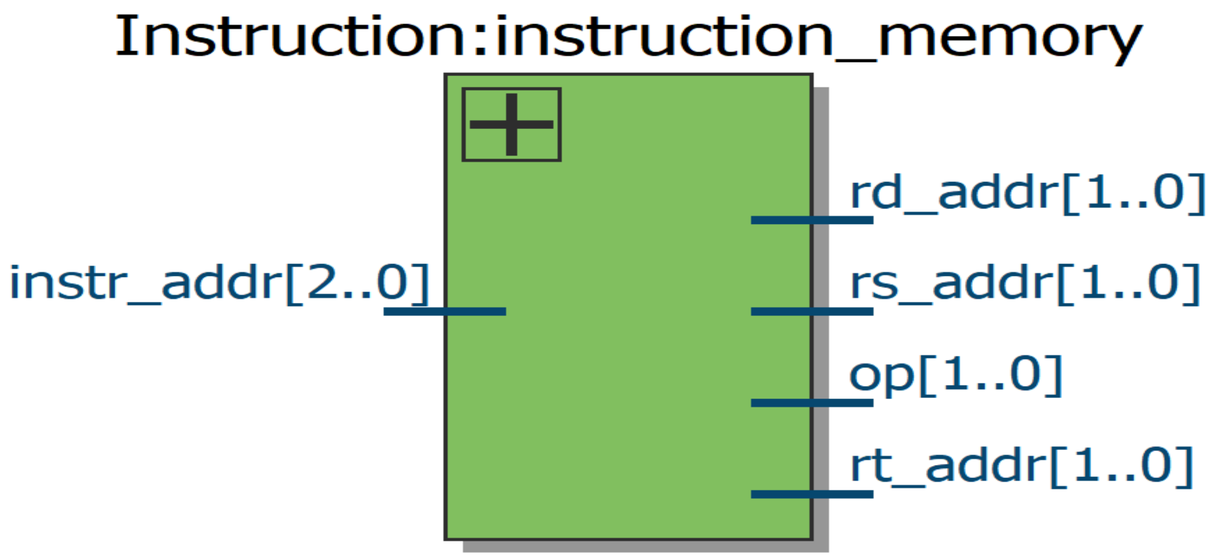
A memory unit to store the 8-bit instructions of a program. Fetch each instruction with address.

instruction <= instruction_set(current_address)

R-type instruction splitted.

- *2-bit operation code send to ALU and Control Unit.*
- *2-bit address of source register 1 send to Registers File.*
- *2-bit address of source register 2 send to Registers File*
- *2-bit address of destination register send to Registers File*

```
op_code <= instruction(7 downto 6 )
src1_address <= instruction(5 downto 4)
src2_address <= instruction(3 downto 2)
dst_address <= instruction(1 downto 0)
```



IV. *Registers File*

4 8-bit registers to store the binary from 00000000 to 11111111 (default 00000000). Fetch each registers with address.

- *8-bit value of source register 1 and 2 send to ALU*

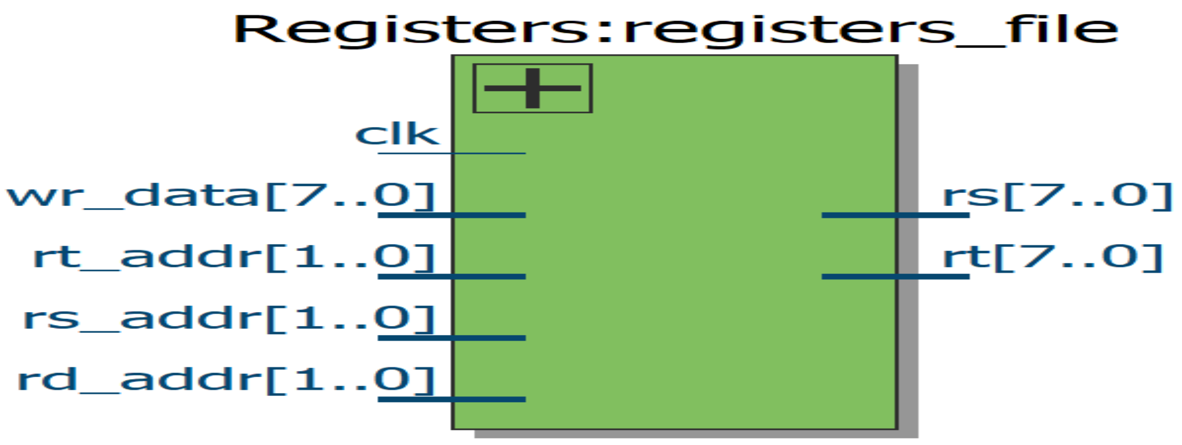
```
 src1_value <= registers(src1_address)
src2_value <= registers(src2_address)
```

- *. Write 8-bit data result from ALU to destination register every falling edge of clock.*

```
if falling_edge(clock) then

    registers(dst_address) <= result

end if
```

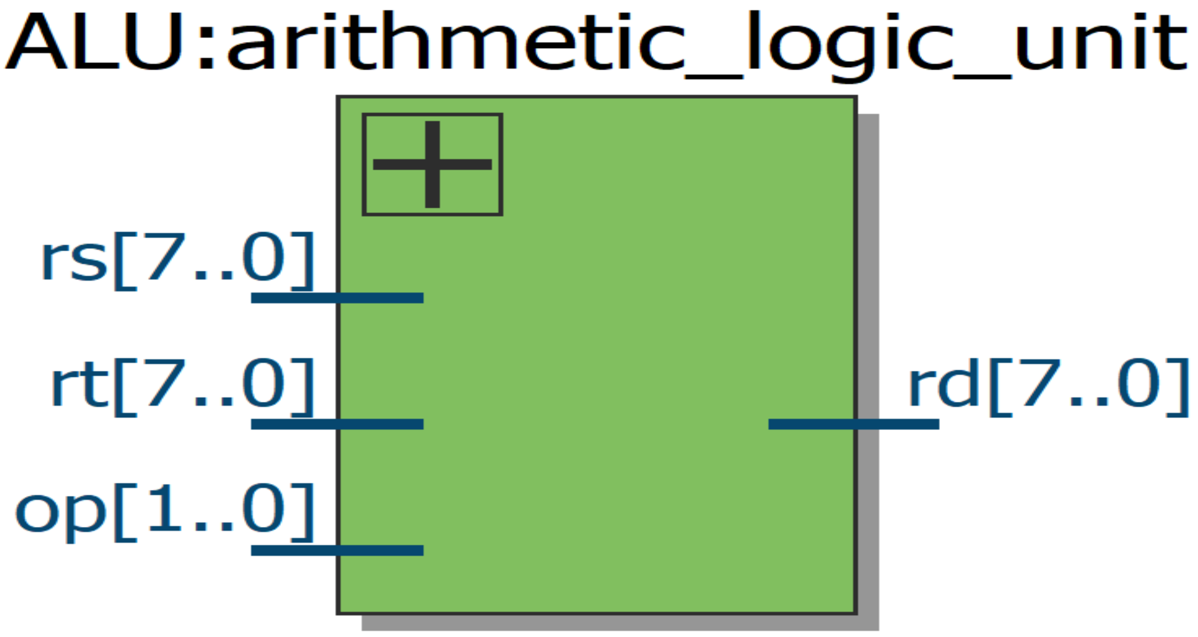


V. *Arithmetic Logic Unit (ALU)*

Calculates arithmetic operations then sent to Registers File. Select each operation with operation code.

- *operation code **00** is and.*
- *operation code **01** is add (addition).*
- *operation code **10** is sub (subtraction).*
- *operation code **11** is addi (addition immediate).*

```
if (op_code = "00") then
  result <= src1_value and src2_value
elsif (op_code = "01") then
  result <= src1_value + src2_value
elsif (op_code = "10") then
  result <= src1_value - src2_value
elsif (op_code = "11") then
  result <= src1_value + data
end if
```

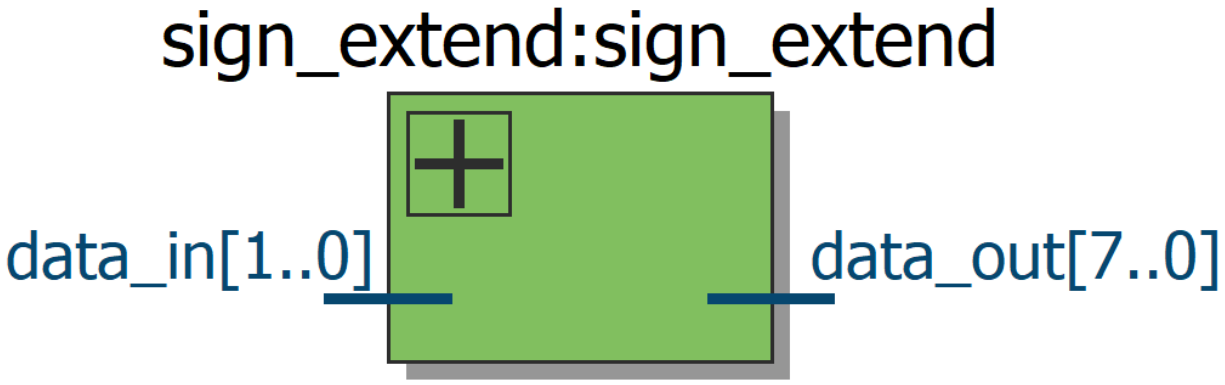


VI. Sign Extend

Special component for addi operation. Extend bits of data from 2 to 8 then send to ALU.

# positive value only

data_out <= "000000" & data_in



VII. MUX

Mux0

Special component for addi operation. Select destination address with reg_dst from Control Unit then send destination to Registers File.

- *reg_dst is 0. Default destiantion address.*
- *reg_dst is 1. Change destination address, use address of source register 2.*

Mux1

Special component for addi operation. Select data with alu_src from Control Unit then sent data to ALU.

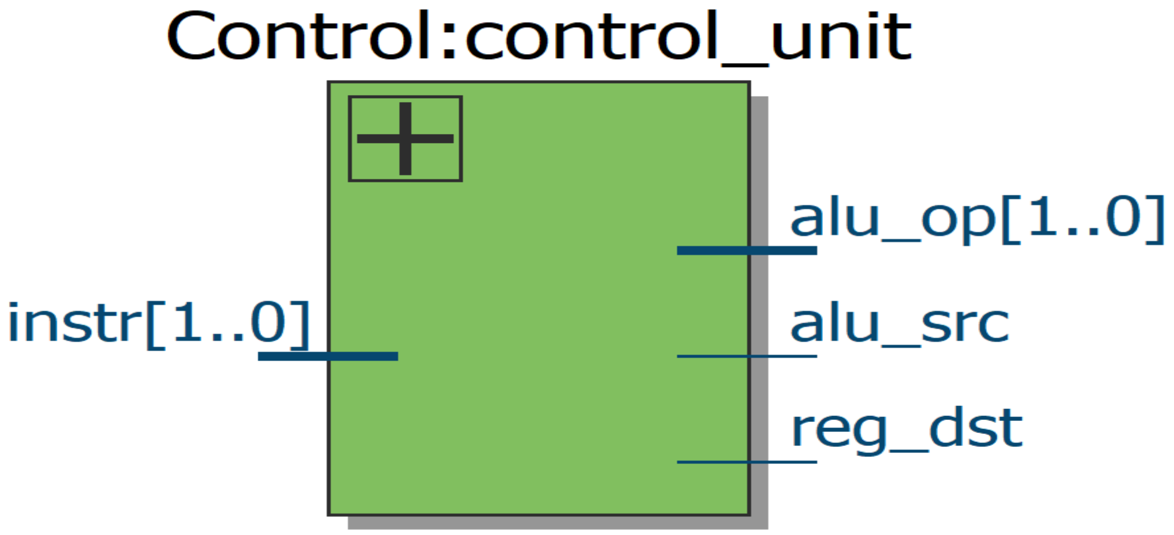
- *alu_src is 0. Default data values from register*
- *alu_src is 1. Change data values, use data from Sign Extend*

VIII. Control Unit

Special component for addi operation. Control others component with operation code.

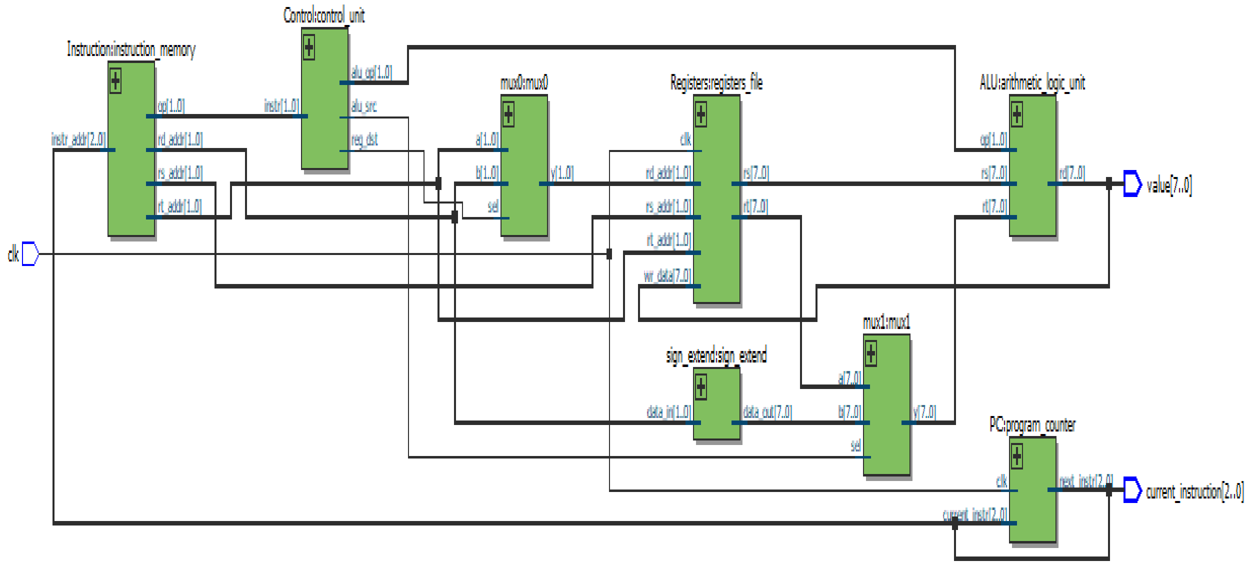
reg_dst is register destination control.
alu_src is ALU source register data control.

- *op_code is 11.*
  reg_dst is 1.
  alu_src is 1.
- *Others.*
  reg_dst is 0.
  alu_src is 0.



8-Bit Simple Processor:

FPGA-based 8-bit simple processor designed using the VHDL language



Instruction:instruction_memory
instr_addr[2..0]
op[1..0]
rd_addr[1..0]
rs_addr[1..0]
rt_addr[1..0]
Control:control_unit
instr[1..0]
alu_op[1..0]
alu_src
reg_dst
mux0:mux0
a[1..0]
b[1..0]
sel
y[1..0]
Registers:registers_file
clk
rd_addr[1..0]
rs_addr[1..0]
rt_addr[1..0]
wr_data[7..0]
rs[7..0]
rt[7..0]
ALU:arithmetic_logic_unit
op[1..0]
rs[7..0]
rt[7..0]
rd[7..0]
value[7..0]
sign_extend:sign_extend
data_in[1..0]
data_out[7..0]
mux1:mux1
a[7..0]
b[7..0]
sel
y[7..0]
PC:program_counter
clk
current_instr[2..0]
next_instr[2..0]
current_instruction[2..0]
clk