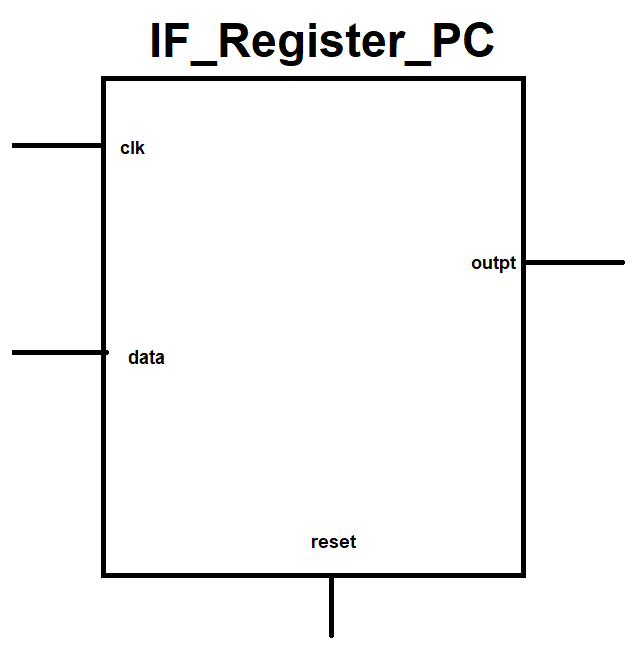
**Modules/Entities Report**

1. **IF\_Register\_PC**

* Black-Box:



* Entity/Module Ports:  
    
  inputs:

clk – Input clock of the Register (active in positive/rising edge).

data – 32[Bits] synchronous data input.  
reset – Asynchronous reset signal (active HIGH).

Outputs:

outpt – 32[Bits] synchronous output data (read from the input data)

* Architecture Description:

The module read the data input only in rising edge of the clock input.

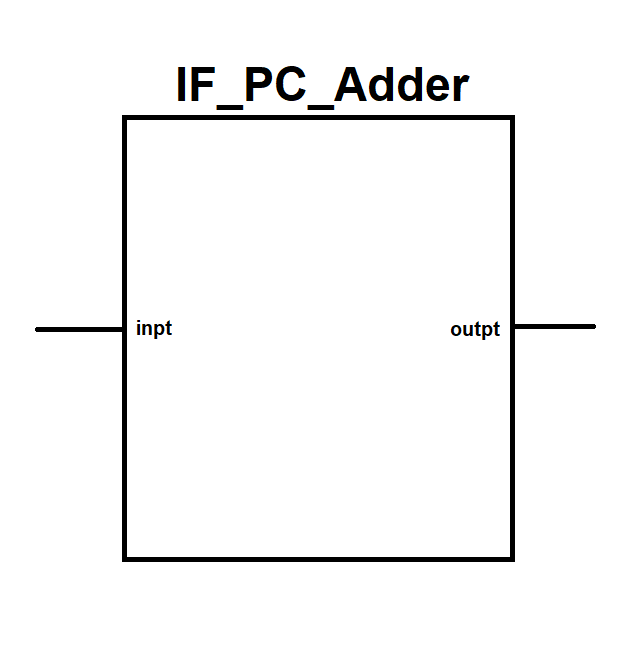
The module can be reset to zero when the reset input gets logic ‘1’.

* Target:

The module holds the address of the current cell in the instruction memory of the executed instruction.

1. **IF\_PC\_Adder**

* Black-Box:



* Entity/Module Ports:

Inputs:

inpt – 32[Bits] input signal.

Outputs:

outpt – 32[Bits] output signal.

* Architecture Description:

The output of the module will be according to the formula:

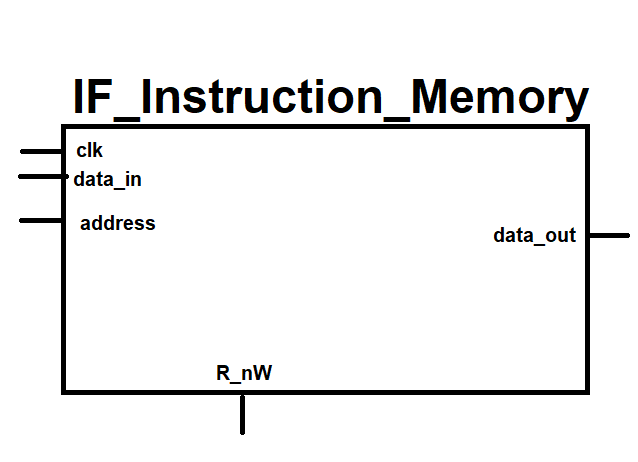
The module increment the input.

* Target:

The input of the module wired with the output of the PC Register and the output of the module go to the input of the PC Register.  
It helps to the PC Register to move forward to the next cell in the instruction memory.

1. **IF\_Instruction\_Memory**

* Black-Box:



* Entity/Module Ports:

Inputs:

clk – input clock signal (active rising edge).

data\_in – 32[Bits] data input of the memory (this port use only by the debugger).

address – 32[Bits] address input.

R\_nW – when the input is ‘1’ the module in Read state else the module in Write state.

Outputs:

data\_out – 32[Bits] data output port of the memory (the output get the encode instruction that holds in the memory).

* Architecture Description:

Instruction memory (number of cells = 2^32 | size of cell = 32[Bits]) in every rising edge of the clock the module check the R\_nW input port:  
when ‘1’: Read state – The data\_out port of the module get the data from the cell that address input port point.  
when ‘0’: Write state – The data\_out port get ‘Z’ high impedance.  
we can write data to the memory in a specific address with the data\_in input port.

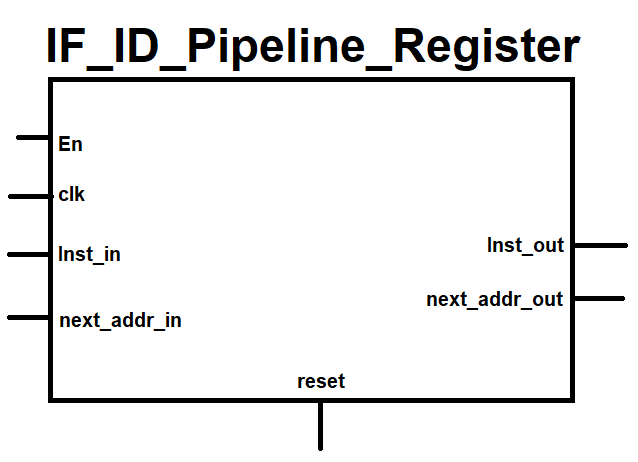
* Target:  
  The module is used as an instruction memory.  
  every cell holds an instruction that encoded from assembler to machine language.

We can store data to the memory from external device.

The memory will always will be in read state when the CPU control this module (without interference of external devices).

1. **IF\_ID\_Pipeline\_Register**

* Black-Box:



* Entity/Module Ports:

Inputs:

clk – input clock signal (active rising edge).

En – enable input port (active high – ‘1’).

Inst\_in – 32[Bits] data input instruction encode.

next\_addr\_in – 32[Bits] data input of the next address in the program memory.

reset – input reset asynchronous signal (active HIGH).

Outputs:

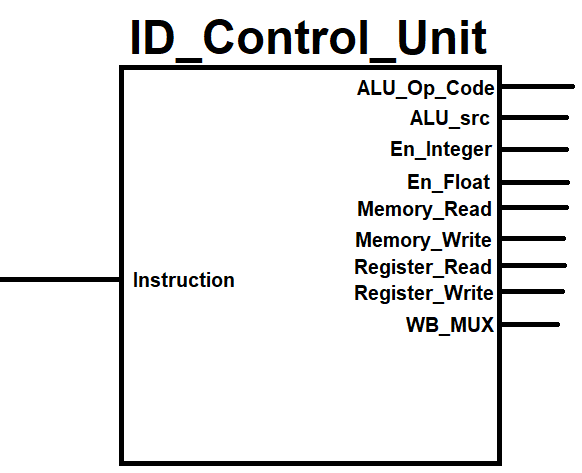
Inst\_out – 32[Bits] data output instruction encode output

next\_addr\_out – 32[Bits] data output of the next address in the program memory.

* Architecture Description:  
  The module is register that holds 2 parameters:  
  1. The instruction encode.  
  2. The next address of the program counter.  
  The register has an enable input signal that synchronous to the clk and asynchronous reset input.
* Target:  
  The module help to sync the first stage of the pipeline with the second stage.

1. **ID\_Control\_Unit**

* Black-Box:



* Entity/Module Ports:

Inputs:

Instruction – 32[Bits] data input (get the instruction from the pipeline register).

Outputs:

ALU\_Op\_Code – n[Bits] control signal output that choose the operation of the ALU.  
ALU\_src – control signal output that choose the input source of the ALU between register value and immediate value.

En\_Integer – Control signal output that enable the UI\_ALU (active HIGH).

En\_Float – Control signal output that enable the FPU\_ALU (active HIGH).

Memory\_Read – Control signal output that enable the read function from the data memory (active HIGH).

Memory\_Write – Control signal output that enable the write function from the data memory (active HIGH).

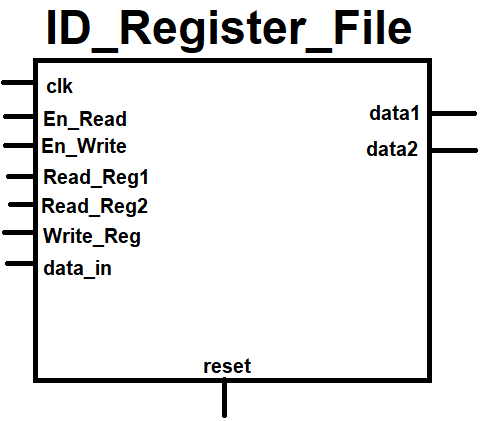
Register\_Read – Control signal output that enable the read function from the CPU registers.  
Register\_Write – Control signal output that enable the write function from the CPU registers.  
WB\_MUX – Control signal output that choose the data we want to write back to the register (‘1’ – MEMORY, ‘0’ - Register).

* Architecture Description:

The control unit is combinational logic module that send control signals to all of the modules in the CPU.

1. **ID\_Register\_File**

* Black-Box:



* Entity/Module Ports:

Inputs:

clk – input clock signal (active **falling edge**).

reset – input asynchronous reset signal (active HIGH).

En\_Read – control signal that enable the read function from the registers.  
En\_Write – control signal that enable the write function to the registers.

Read\_Reg1 – 5[Bits] address of the first register we want to read.

Read\_Reg2 – 5[Bits] address of the second register we want to read.

Write\_Reg – 5[Bits] address of the register we want to write.  
data\_in – 32[Bits] of data input to the register we want to write.

Outputs:

data1 – 32[Bits] of data from the first register we read.

data2 – 32[Bits] of data from the second register we read.

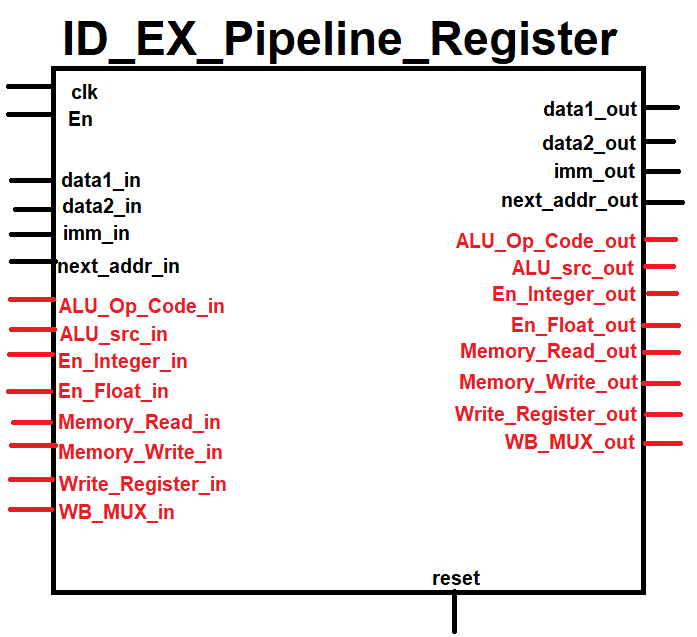
* Architecture Description:

The module includes 32 registers (they are the CPU Registers) every register has: en, clk, reset signals and input/output data ports.  
The register size is: 32[Bits].

Note: The register r0 contain only 0 value and can’t be change.

1. **ID\_EX\_Pipeline\_Register**

* Black-Box:



* Entity/Module Ports:

Inputs:

clk – input clock signal (active rising edge).  
En – input enable signal (synchronous).

reset – input reset signal (asynchronous).

next\_addr\_in – get the next address of the PC Register from the previous pipeline register.

data1\_in – 32[Bit] get the data that read from the first register.

data2\_in – 32[Bit] get the data that read from the second register.

Imm\_in – 16[Bit] get the immediate value from the instruction encode.

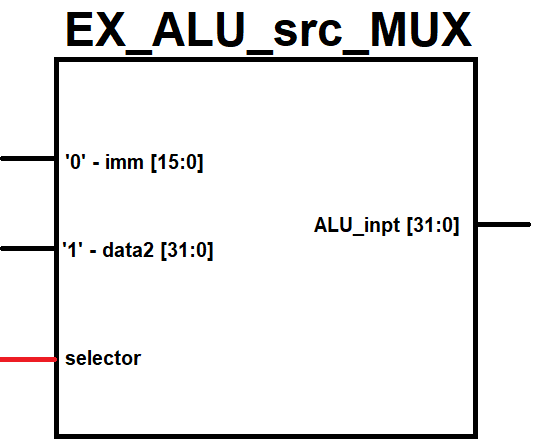
Control Signals (RED) – Continue to send the control signals to the other modules in the CPU architecture.

Outputs:  
all the outputs are the same like the inputs (the outputs update its value by the rising edge of the clk).

* Architecture Description:  
  Register that holds the value all the data inputs at the output.  
  the output update after rising edge of the clock.
* Target:  
  The target of this module is to sync between the second and the third stages in the pipeline.

1. **EX\_ALU\_src\_MUX**

* Black-Box:



* Entity/Module Ports:  
    
  Inputs:

imm – 16[Bit] input of the immediate value.  
data2 – 32[Bit] input of the Register 2 value.

selector – input that select which input will go to the output.

Outputs:

ALU\_inpt – 32[Bit] output of the multiplexer.

* Architecture Description:

This module is mux2x1 that select between the operands data2 (come from register) and imm (come from immediate value in the instruction encode).  
**Note:** that if selector ‘0’ the output will be imm and the 16[MSBs] need to be zero.

* Target:

This module enable us to perform operations with immediate values and registers.

1. **EX\_ALU\_IU**

* Black-Box:
* Entity/Module Ports:

Inputs:

inpt1 – 32[Bit] input operand.

inpt2 – 32[Bit] input operand.

Op\_Code – 6[Bit] input selector.

Outputs:

outpt – 32[Bit] the output result.  
xxx\_flag – output flag that calculate the BEQ operation (if ‘1’ the condition exists).  
(There are 4 flags, each one has different )

* Architecture Description:

The module behave like the following LUT (Look Up Table):

|  |  |
| --- | --- |
| Op\_Code | Operation |
| 000100 | ADD (+) |
| 000101 | SUB (-) |
| 000110 | MUL (\*) |
| 000111 | DIV (/) |
| 010100 | ADDU (+ Unsigned) |
| 010101 | SUBU (- Unsigned) |
| 010110 | MULU (\* Unsigned) |
| 010111 | DIVU (/ Unsigned) |
| 001000 | AND (&) |
| 001001 | OR (|) |
| 001010 | NOR (~|) |
| 001011 | XOR (^) |
| 011000 | SLL |
| 011001 | SRL |
| 011010 | SLA |
| 011011 | SRA |
| 001100 | BEQ (FLAG) |
| 001101 | BNE (FLAG) |
| 001110 | BGT (FLAG) |
| 001111 | BLE (FLAG) |

Note: If the MSB of the Op\_Code equal to ‘1’ the operation executed with the 16 high bits.  
Note: If the Op\_Code(4) also equal to one (in addition to the MSB) the operation that executed is unsigned operation

* Target: