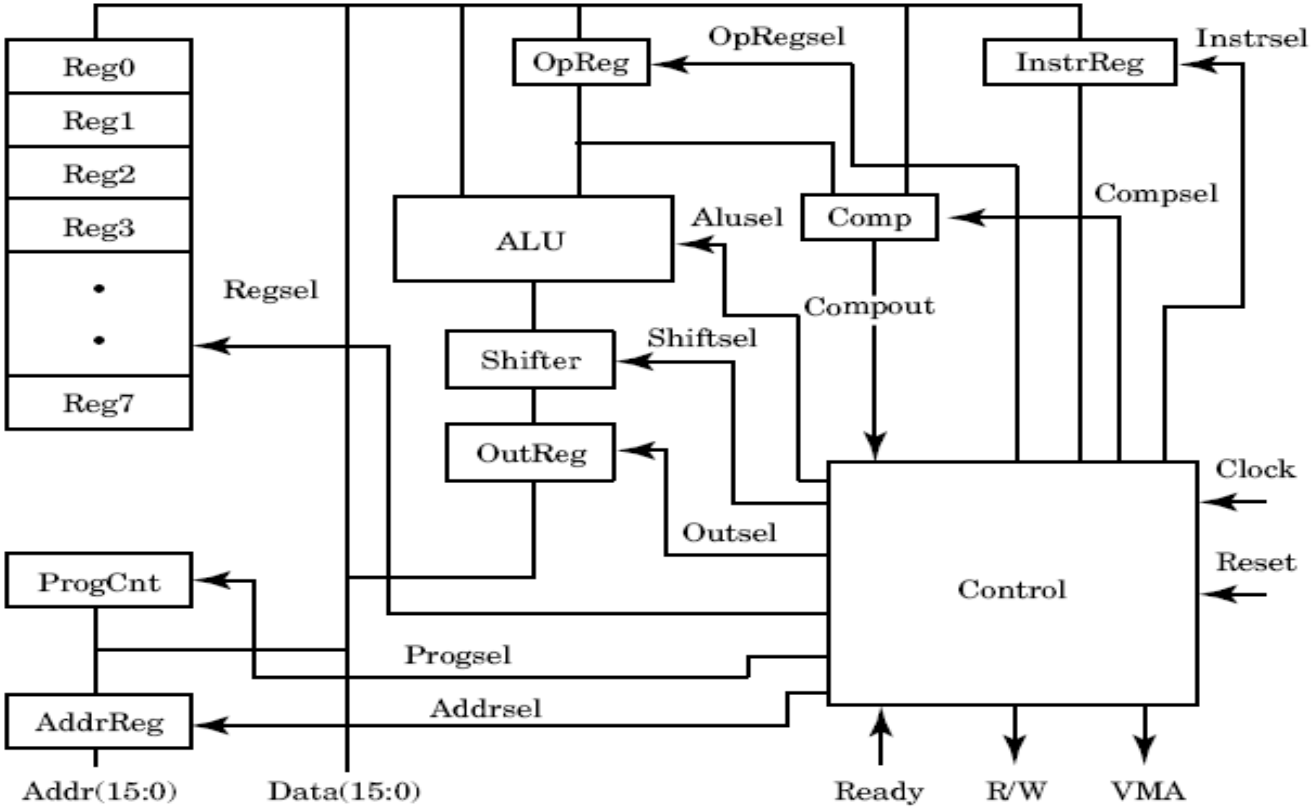
UNIVERSITÀ
DEGLI STUDI DI TRIESTE

dipartimento
di ingegneria
e architettura

15 - High level design flow

A.Carini – Progettazione di sistemi elettronici

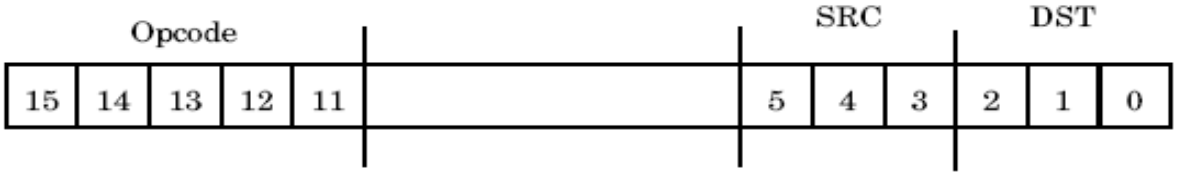
CPU design


Reg0
Reg1
Reg2
Reg3
Reg7
Regsel
OpReg
OpRegsel
InstrReg
Instrsel
ALU
Alusel
Comp
Compsel
Compout
Shifter
Shiftsel
OutReg
Outsel
Clock
Reset
Control
ProgCnt
Progsel
AddrReg
Addrsel
Addr(15:0)
Data(15:0)
Ready
R/W
VMA

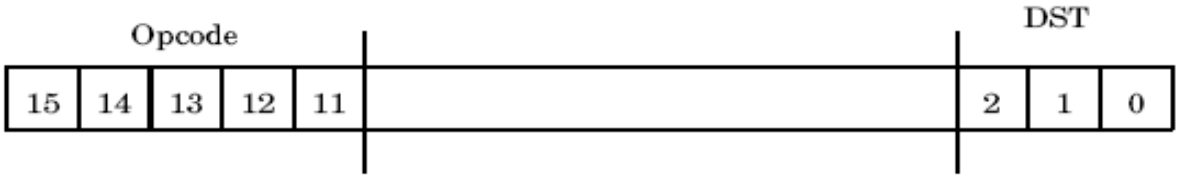
Instructions

- Load
- Store
- Branch
- ALU
- Shift

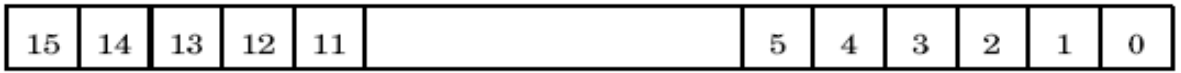
Instruction words



Single Word



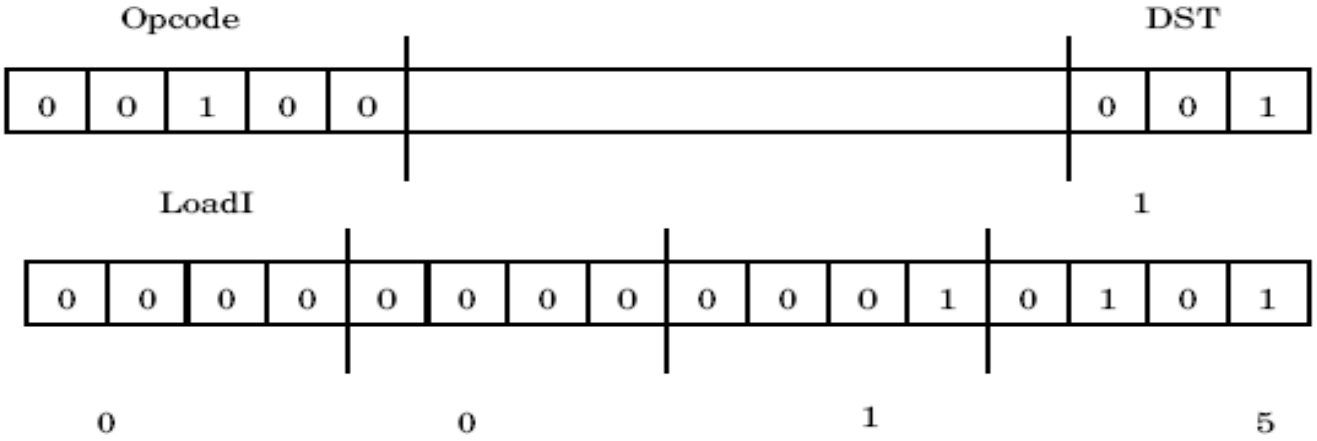
Address or Data



Double Word

Instruction data

```
LoadI 1, 16#15
```



Operation codes

OPCODE	INSTRUCTION	NOTE
00000	NOP	No operation
00001	LOAD	Load register
00010	STORE	Store register
00011	MOVE	Move value to register
00100	LOADI	Load register with immediate value
00101	BRANCHI	Branch to immediate address
00110	BRANCHGTI	Branch greater than to immediate address
00111	INC	Increment
01000	DEC	Decrement
01001	AND	And two registers
01010	OR	Or two registers
01011	XOR	Xor two registers
01100	NOT	Not a register value
01101	ADD	Add two registers
01110	SUB	Subtract two registers
01111	ZERO	Zero a register
10000	BRANCHLTI	Branch less than to immediate address
10001	BRANCHLT	Branch less than
10010	BRANCHNEQ	Branch not equal
10011	BRANCHNEQI	Branch not equal to immediate address
10100	BRANCHGT	Branch greater than
10101	BRANCH	Branch all the time
10110	BRANCHEQ	Branch if equal
10111	BRANCHEQI	Branch if equal to immediate address
11000	BRANCHLTEI	Branch if less or equal to immediate address
11001	BRANCHLTE	Branch if less or equal
11010	SHL	Shift left
11011	SHR	Shift right
11100	ROTR	Rotate right
11101	ROTL	Rotate left

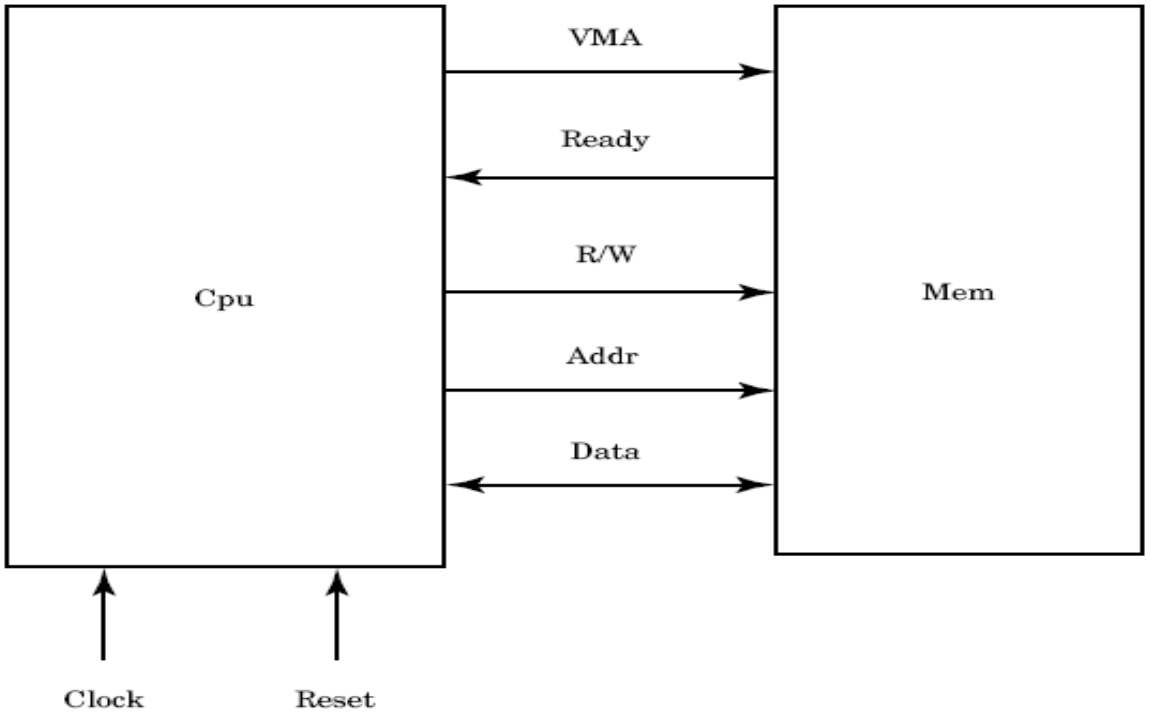
Cpu_lib.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
package cpu_lib is
  type t_shift is (shftpass, shl, shr, rotl, rotr);
  subtype t_alu is unsigned(3 downto 0);
  constant alupass : unsigned(3 downto 0) := "0000";
  constant andOp : unsigned(3 downto 0) := "0001";
  constant orOp : unsigned(3 downto 0) := "0010";
  constant notOp : unsigned(3 downto 0) := "0011";
  constant xorOp : unsigned(3 downto 0) := "0100";
  constant plus : unsigned(3 downto 0) := "0101";
  constant alusub : unsigned(3 downto 0) := "0110";
  constant inc : unsigned(3 downto 0) := "0111";
  constant dec : unsigned(3 downto 0) := "1000";
  constant zero : unsigned(3 downto 0) := "1001";

  type t_comp is (eq, neq, gt, gte, lt, lte);
  subtype t_reg is std_logic_vector(2 downto 0);
  type state is (reset1, reset2, reset3, reset4, reset5, reset6, execute,
                 nop, load, store, move, load2, load3, load4, store2, store3,
                 store4, move2, move3, move4,incPc,
                 incPc2, incPc3, incPc4, incPc5, incPc6, loadPc, loadPc2,
                 loadPc3, loadPc4, bgtI2, bgtI3, bgtI4, bgtI5, bgtI6, bgtI7,
                 bgtI8, bgtI9, bgtI10, braI2, braI3, braI4, braI5, braI6, loadI2,
                 loadI3, loadI4, loadI5, loadI6, inc2, inc3, inc4);
  subtype bit16 is std_logic_vector(15 downto 0);

end cpu_lib;
```

Top.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;
-- use work.cpu_math.all;

entity top is
end top;

architecture behave of top is
  component mem
    port (addr : in bit16;
          sel, rw : in std_logic;
          ready : out std_logic;
          data : inout bit16);
  end component;
  component cpu
    port(clock, reset, ready : in std_logic;
         addr : out bit16;
         rw, vma : out std_logic;
         data : inout bit16);
  end component;
  signal addr, data : bit16;
  signal vma, rw, ready : std_logic;
  signal clock, reset : std_logic := '0';
begin

  clock <= not clock after 50 ns;
  reset <= '1', '0' after 100 ns;

  m1 : mem port map (addr, vma, rw, ready, data);
  u1 : cpu port map(clock, reset, ready, addr, rw, vma, data);
end behave;
```

Top.vhd



Mem2.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.cpu_lib.all;
--use work.cpu_math.all;

entity mem is
  port (addr : in bit16;
        sel, rw : in std_logic;
        ready : out std_logic;
        data : inout bit16);
end mem;

architecture behave of mem is
begin
  memproc: process(addr, sel, rw)
    type t_mem is array(0 to 63) of bit16;
    variable mem_data : t_mem :=
     ("0010000000000001",  --- 0  loadI 1, #   -- load source address
      "0000000000010000",  --- 1  10
      "0010000000000010",  --- 2  loadI 2, #   -- load destination address
      "0000000000110000",  --- 3  30
      "0010000000000110",  --- 4  loadI 6, #   -- load data end address
      "0000000000111111",  --- 5  3F
      "0000100000001011",  --- 6  load 1, 3    -- load reg3 with source element
      "0001000000011010",  --- 7  store 3, 2   -- store reg3 at destination
      "0011000000001110",  --- 8  bgtI 1, 6, # -- compare to see if at end of data
      "0000000000000000",  --- 9  00           -- if so just start over
```

Mem2.vhd

```
"0011100000000001",  --- A  inc 1        -- move source address to next
"0011100000000010",  --- B  inc 2        -- move destination address to next
"0010100000001111",  --- C  braI #       -- go to the next element to copy
"0000000000000110",  --- D  06
"0000000000000000",  --- E
"0000000000000000",  --- F
"0000000000000001",  --- 10       --- Start of source array
"0000000000000010",  --- 11
"0000000000000011",  --- 12
"0000000000000100",  --- 13
"0000000000000101",  --- 14
"0000000000000110",  --- 15
"0000000000000111",  --- 16
"0000000000001000",  --- 17
"0000000000001001",  --- 18
"0000000000001010",  --- 19
"0000000000001011",  --- 1A
"0000000000001100",  --- 1B
"0000000000001101",  --- 1C
"0000000000001110",  --- 1D
"0000000000001111",  --- 1E
"0000000000010000",  --- 1F
"0000000000000000",  --- 20
"0000000000000000",  --- 21
"0000000000000000",  --- 22
"0000000000000000",  --- 23
"0000000000000000",  --- 24
"0000000000000000",  --- 25
"0000000000000000",  --- 26
```

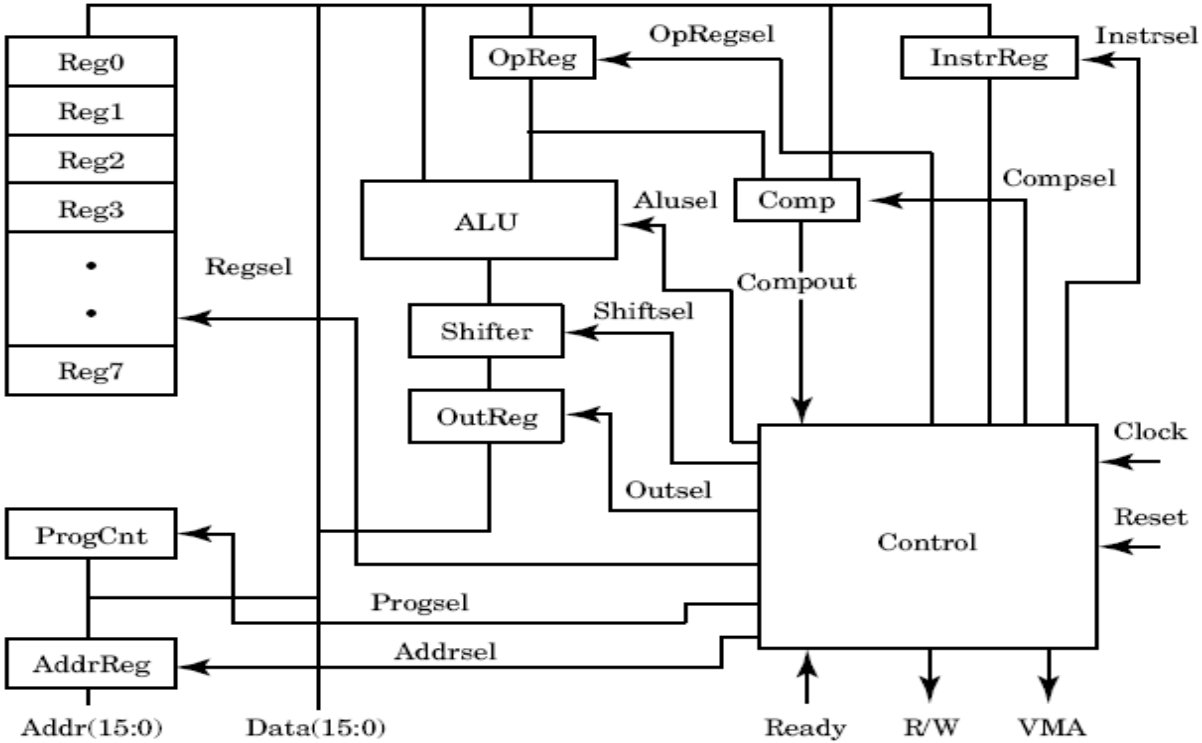
Mem2.vhd

```
        "0000000000000000",  --- 27
        "0000000000000000",  --- 28
        "0000000000000000",  --- 29
        "0000000000000000",  --- 2A
        "0000000000000000",  --- 2B
        "0000000000000000",  --- 2C
        "0000000000000000",  --- 2D
        "0000000000000000",  --- 2E
        "0000000000000000",  --- 2F
        "0000000000000000",  --- 30    -- start of destination array
        "0000000000000000",  --- 31
        "0000000000000000",  --- 32
        "0000000000000000",  --- 33
        "0000000000000000",  --- 34
        "0000000000000000",  --- 35
        "0000000000000000",  --- 36
        "0000000000000000",  --- 37
        "0000000000000000",  --- 38
        "0000000000000000",  --- 39
        "0000000000000000",  --- 3A
        "0000000000000000",  --- 3B
        "0000000000000000",  --- 3C
        "0000000000000000",  --- 3D
        "0000000000000000",  --- 3E
        "0000000000000000"); --- 3F
```

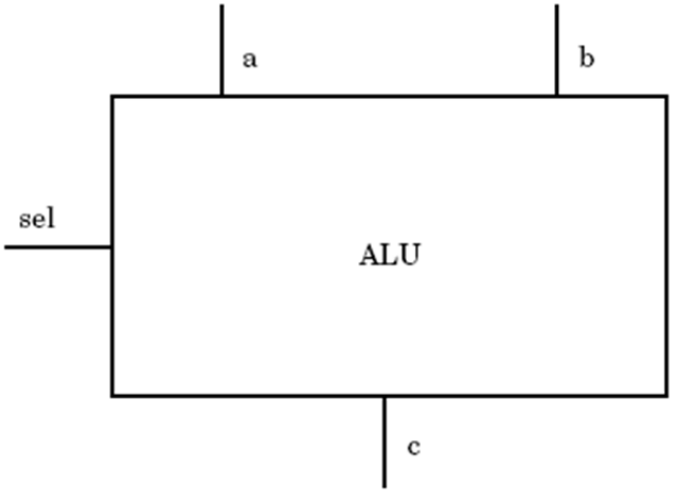
Mem2.vhd

```
  begin
     data <= "ZZZZZZZZZZZZZZZZ";
     ready <= '0';

     if sel = '1' then
       if rw = '0' then
         data <= mem_data(CONV_INTEGER(addr(15 downto 0))) after 1 ns;
         ready <= '1';
       elsif rw = '1' then
         mem_data(CONV_INTEGER(addr(15 downto 0))) := data;
       end if;
     else
       data <= "ZZZZZZZZZZZZZZZZ" after 1 ns;
     end if;
   end process;

end behave;
```

CPU block diagram

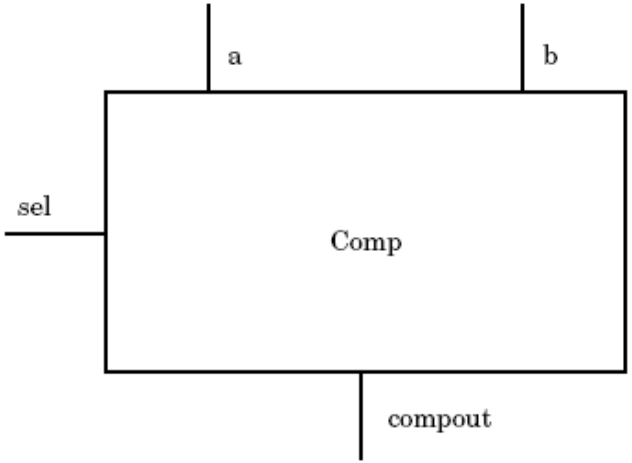


ALU



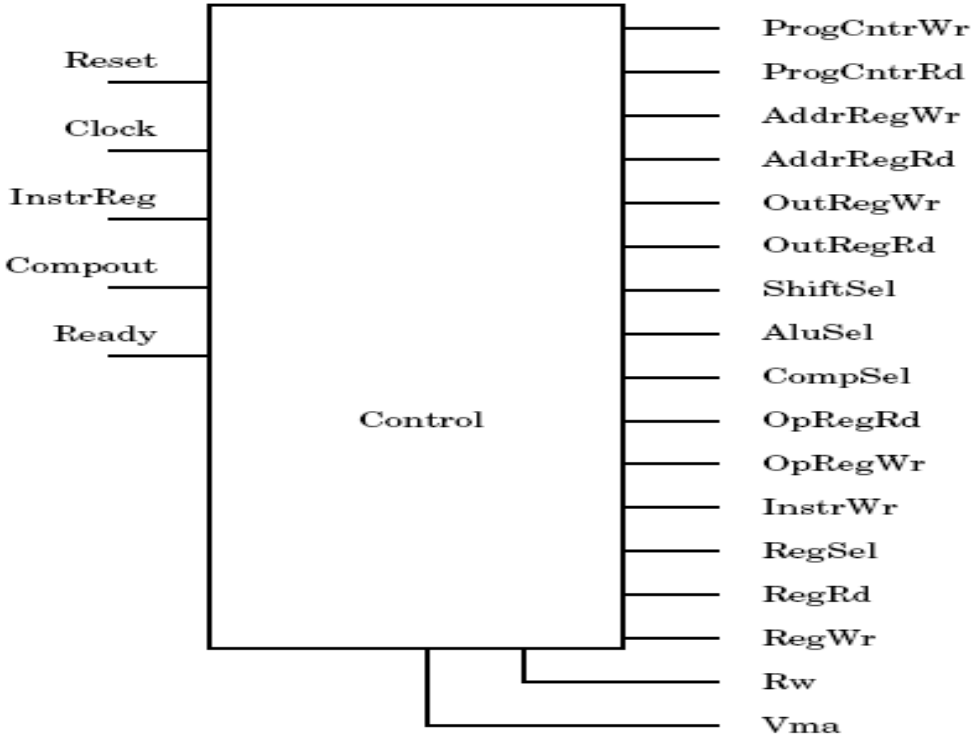
Sel Input	Operation
0000	C = A
0001	C = A AND B
0010	C = A OR B
0011	C = NOT A
0100	C = A XOR B
0101	C = A + B
0110	C = A – B
0111	C = A + 1
1000	C = A –1
1001	C = 0

Comp

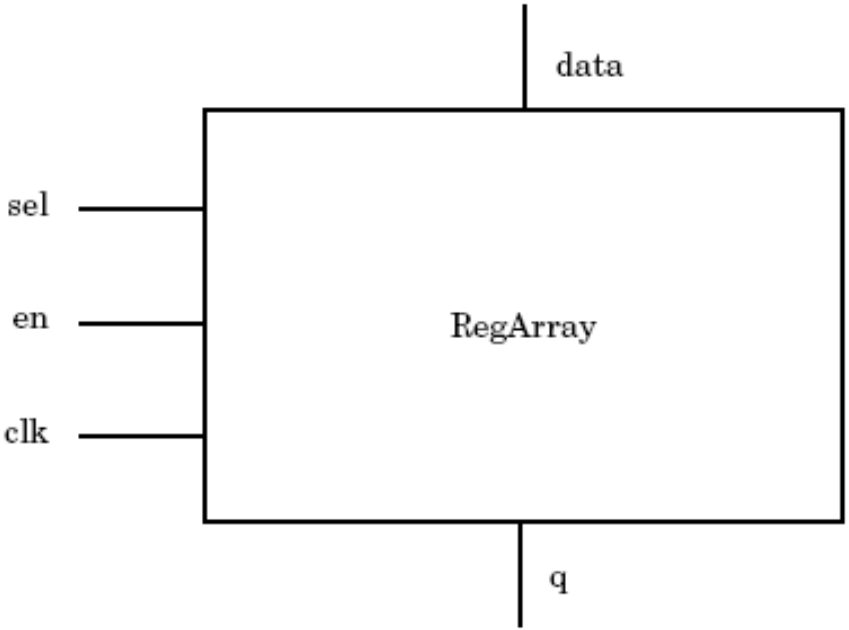


Sel input value	Comparison
EQ	Compout = 1 when a equals b
NEQ	Compout = 1 when a is not equal to b
GT	Compout = 1 when a is greater than b
GTE	Compout = 1 when a is greater than or equal to b
LT	Compout = 1 when a is less than b
LTE	Compout = 1 when a is less than or equal to b

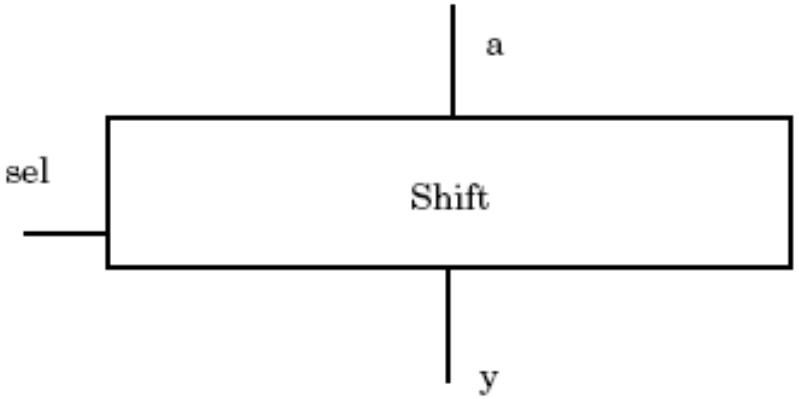
Control


Reset
Clock
InstrReg
Compout
Ready
Control
ProgCntrWr
ProgCntrRd
AddrRegWr
AddrRegRd
OutRegWr
OutRegRd
ShiftSel
AluSel
CompSel
OpRegRd
OpRegWr
InstrWr
RegSel
RegRd
RegWr
Rw
Vma

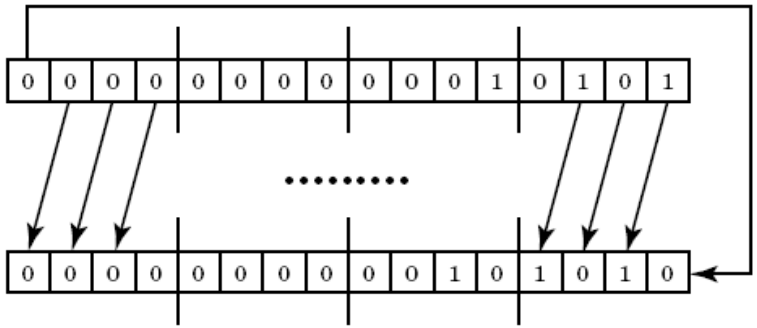
Regarray



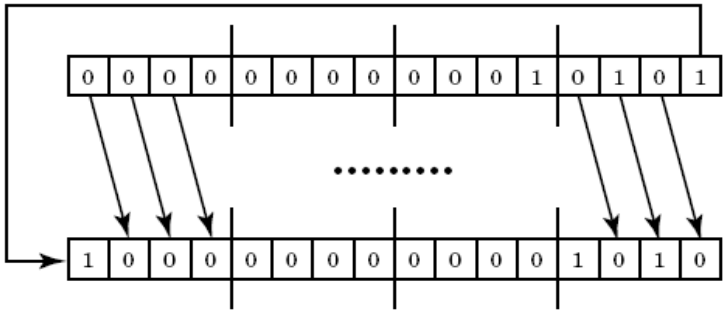
Shift



Rotate

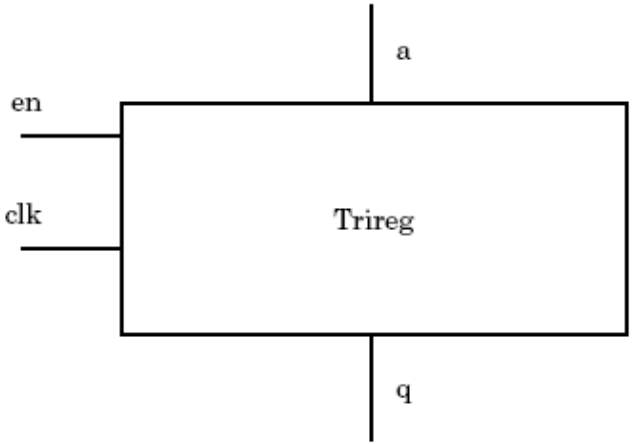


Rotate Left



Rotate Right

Trireg



See:

- Douglas L. Perry, «VHDL programming by example» McGraw Hill,
  - Chapter 12 and 13