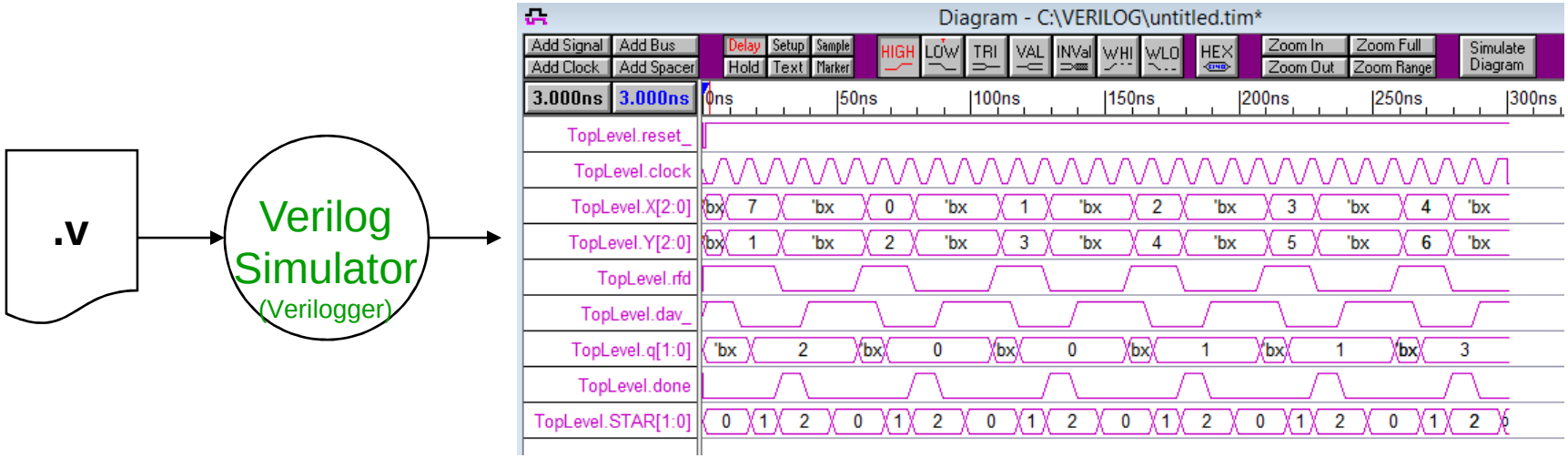
Esercitazione 1
Verilog e MIPS

http://www.dii.unisi.it/~giorgi/didattica/arcal1

Verifica Logica



File Verilog

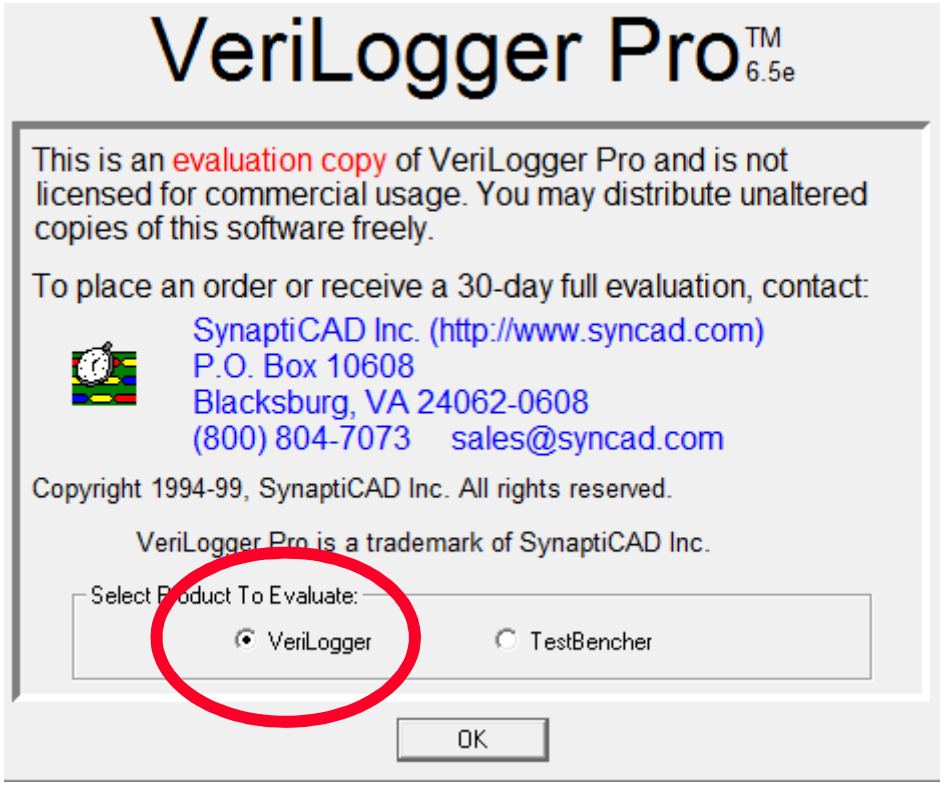
Diagramma Temporale

Istallazione di Verilogger

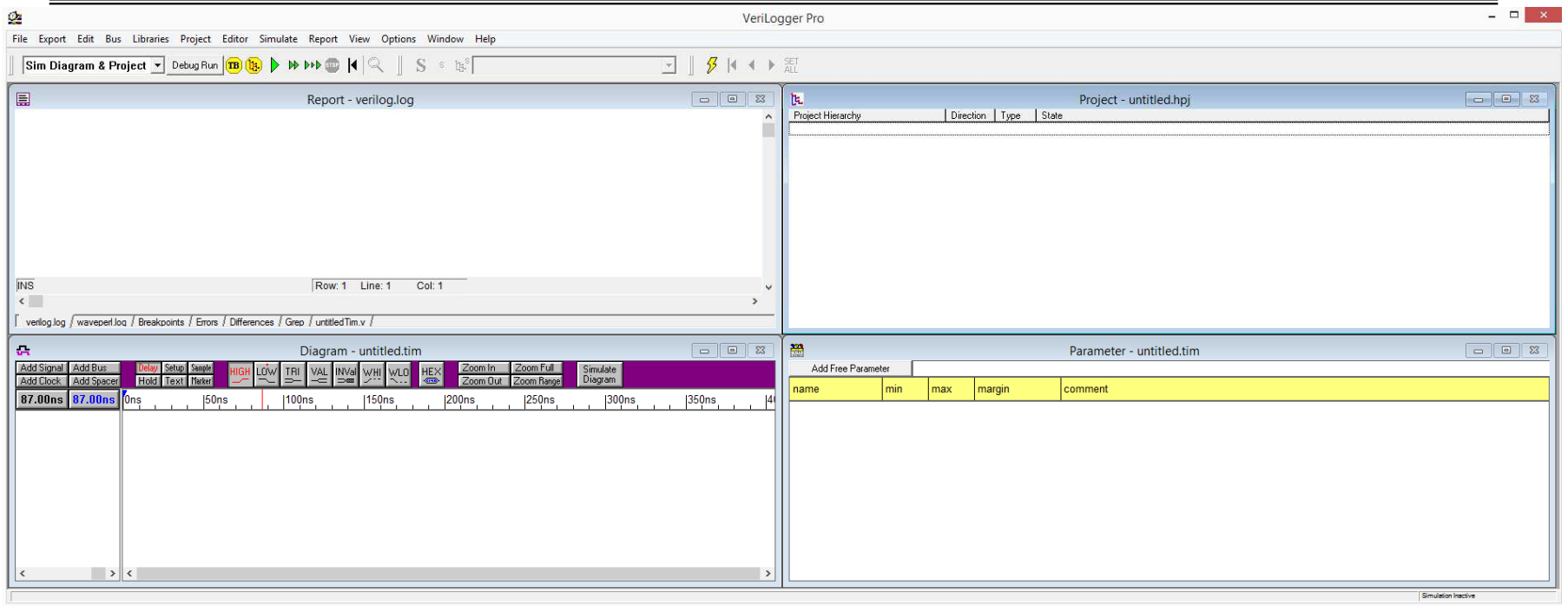
- Distribuibile gratuitamente
- Istruzioni per l'istallazione sul proprio PC (Windows)
  - http://www.dii.unisi.it/~giorgi/didattica/tools/verilog.exe
  - Eseguire (scompatta in C:\VERILOG)
  - Crearsi un link (es. dal Desktop) al file C:\VERILOG\vlogev.exe
- Nel Laboratorio
  - Individuare l'icona relativa

Uso di Verilogger: un primo esempio

- Lanciare il programma Verilogger (vlogev.exe)
  - Selezionare nella prima finestra "VeriLogger"

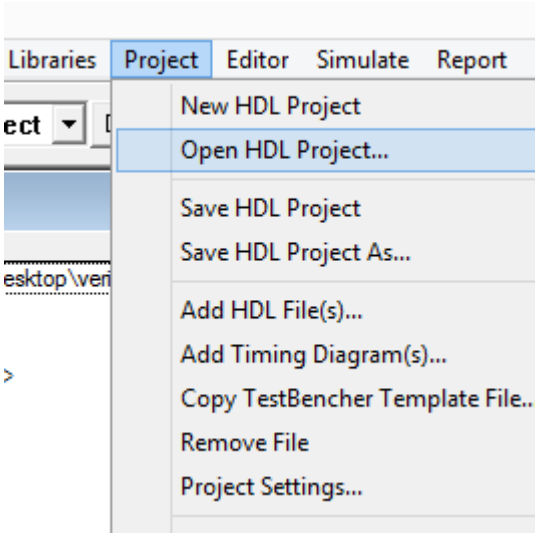


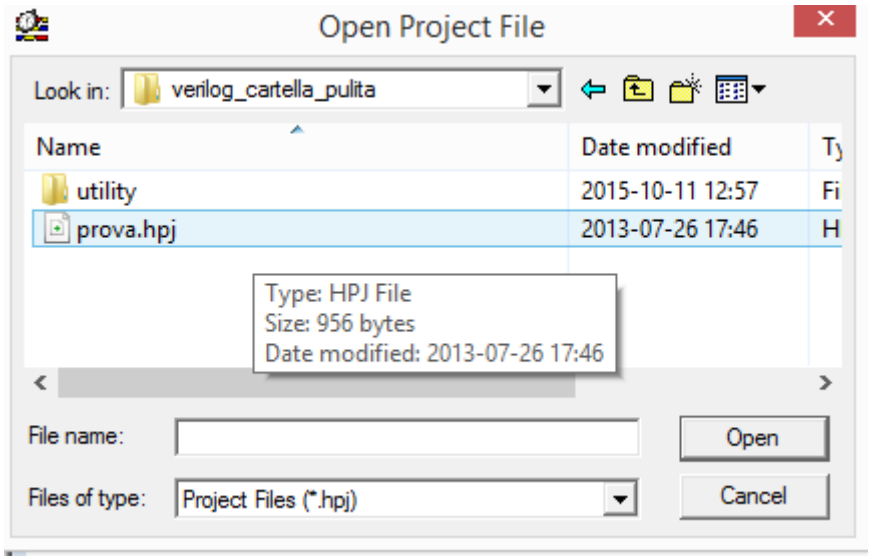
Verilogger: schermata iniziale



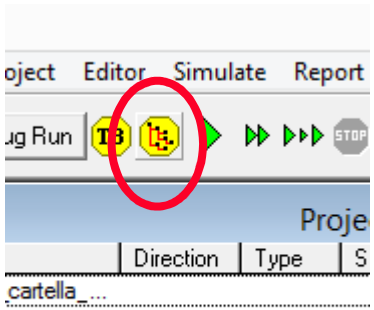
- Sono presenti 4 finestre
  - Report
  - Project
  - Parameter
  - Diagram

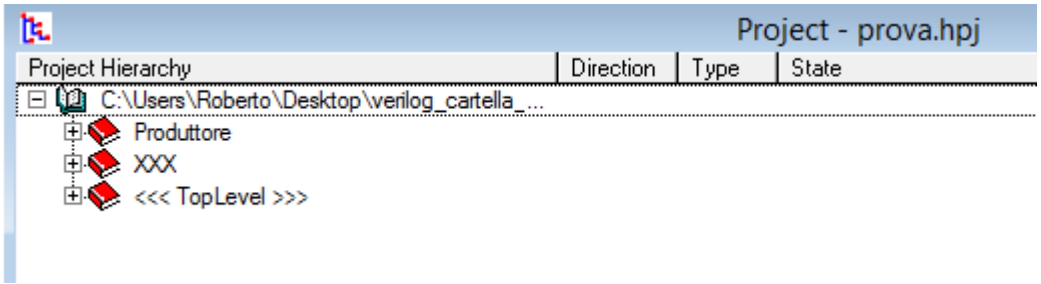
Selezione/Creazione di un progetto

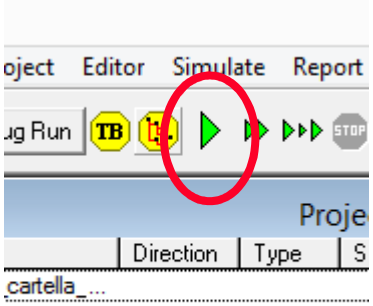




Compilazione e generazione del diagramma temporale







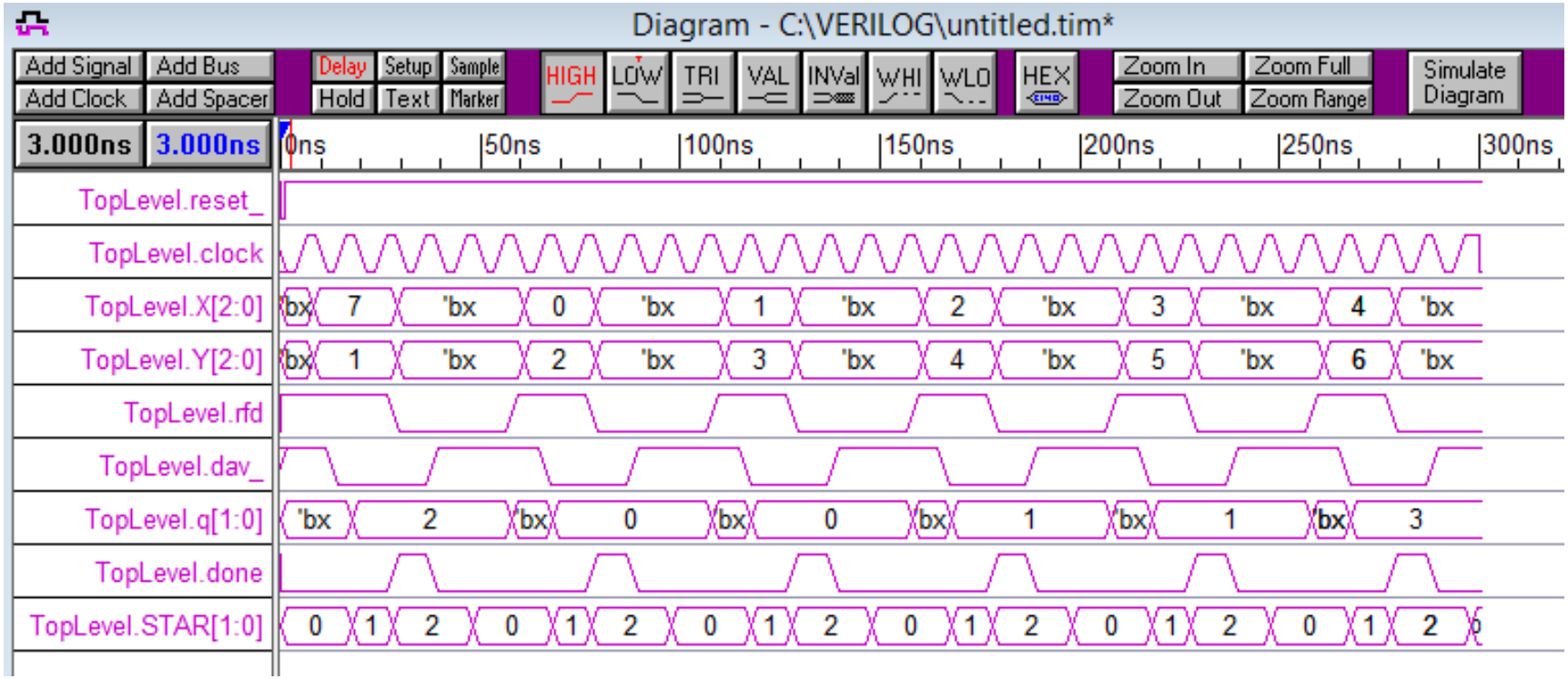
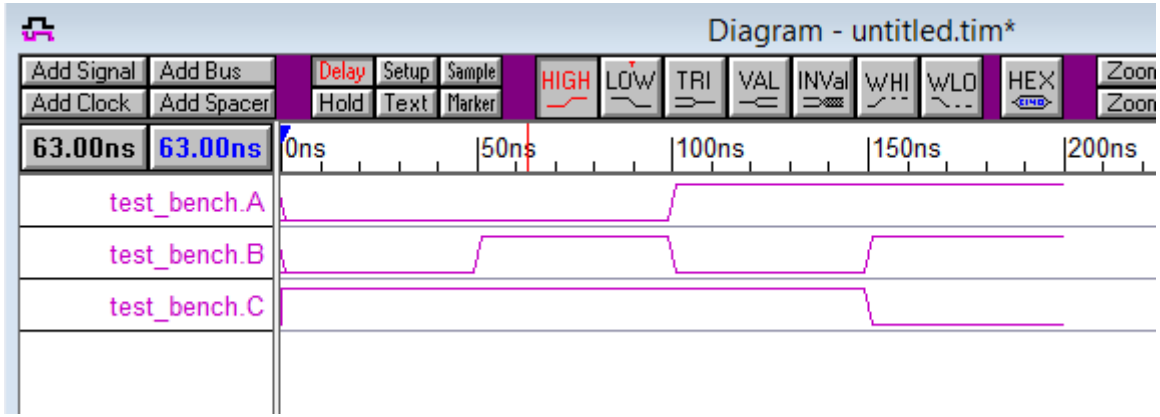


Diagram - C:\VERILOG\untitled.tim*
Add Signal
Add Bus
Add Clock
Add Spacer
Delay
Setup
Sample
Hold
Text
Marker
HIGH
LOW
TRI
VAL
INVal
WHI
WLO
HEX
Zoom In
Zoom Full
Zoom Out
Zoom Range
Simulate Diagram
3.000ns
3.000ns
0ns
50ns
100ns
150ns
200ns
250ns
300ns
TopLevel.reset_
TopLevel.clock
TopLevel.X[2:0]
TopLevel.Y[2:0]
TopLevel.rfd
TopLevel.dav_
TopLevel.q[1:0]
TopLevel.done
TopLevel.STAR[1:0]

Porta nand

```
`timescale 1ns / 1ps
//create a NAND gate out of an AND and an Inverter
module some_logic_component (c, a, b);
       // declare port signals
  output c;
  input a, b;
      // declare internal wire
  wire d;
      //instantiate structural logic gates
  and a1(d, a, b); //d is output, a and b are inputs
  not n1(c, d);     //c is output, d is input
endmodule

//test the NAND gate
module test_bench; //module with no ports
  reg A, B;
  wire C;

  //instantiate your circuit
  some_logic_component S1(C, A, B);

  //Behavioral code block generates stimulus to test circuit
  initial
    begin
      A = 1'b0; B = 1'b0;
      #50; $display("A = %b, B = %b, NAND output C = %b \n", A, B, C);
      A = 1'b0; B = 1'b1;
      #50 $display("A = %b, B = %b, NAND output C = %b \n", A, B, C);
      A = 1'b1; B = 1'b0;
      #50 $display("A = %b, B = %b, NAND output C = %b \n", A, B, C);
      A = 1'b1; B = 1'b1;
      #50 $display("A = %b, B = %b, NAND output C = %b \n", A, B, C);
    end
endmodule
```

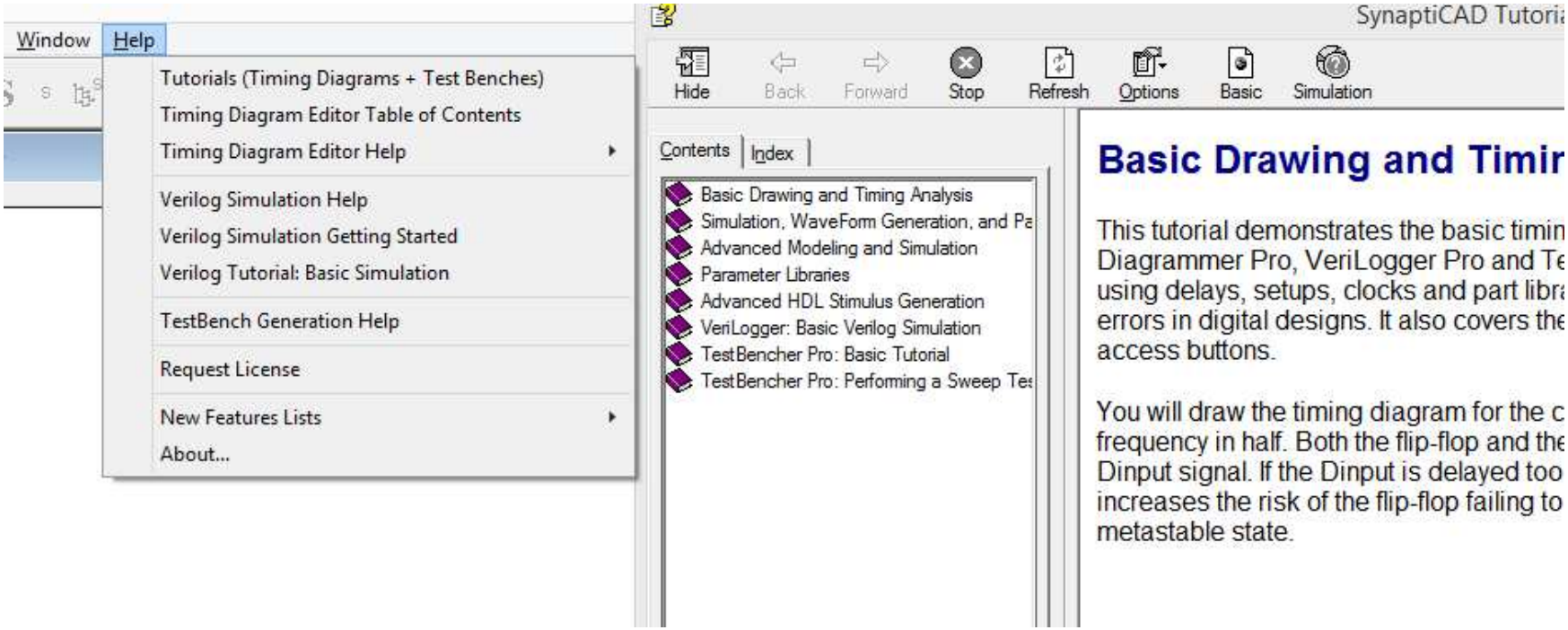
NAND: diagramma e report



Report - C:\VERILOG\verilog.log

```
Finding handle to syncad_top.test_bench.C
Compile Complete
.
Running...
A = 0, B = 0, Nand output C = 1

A = 0, B = 1, Nand output C = 1

A = 1, B = 0, Nand output C = 1

A = 1, B = 1, Nand output C = 0

0 Errors, 0 Warnings
Compile time = 0.02000, Load time = 0.28400, Execution time = 0.02400
```

Documentazione

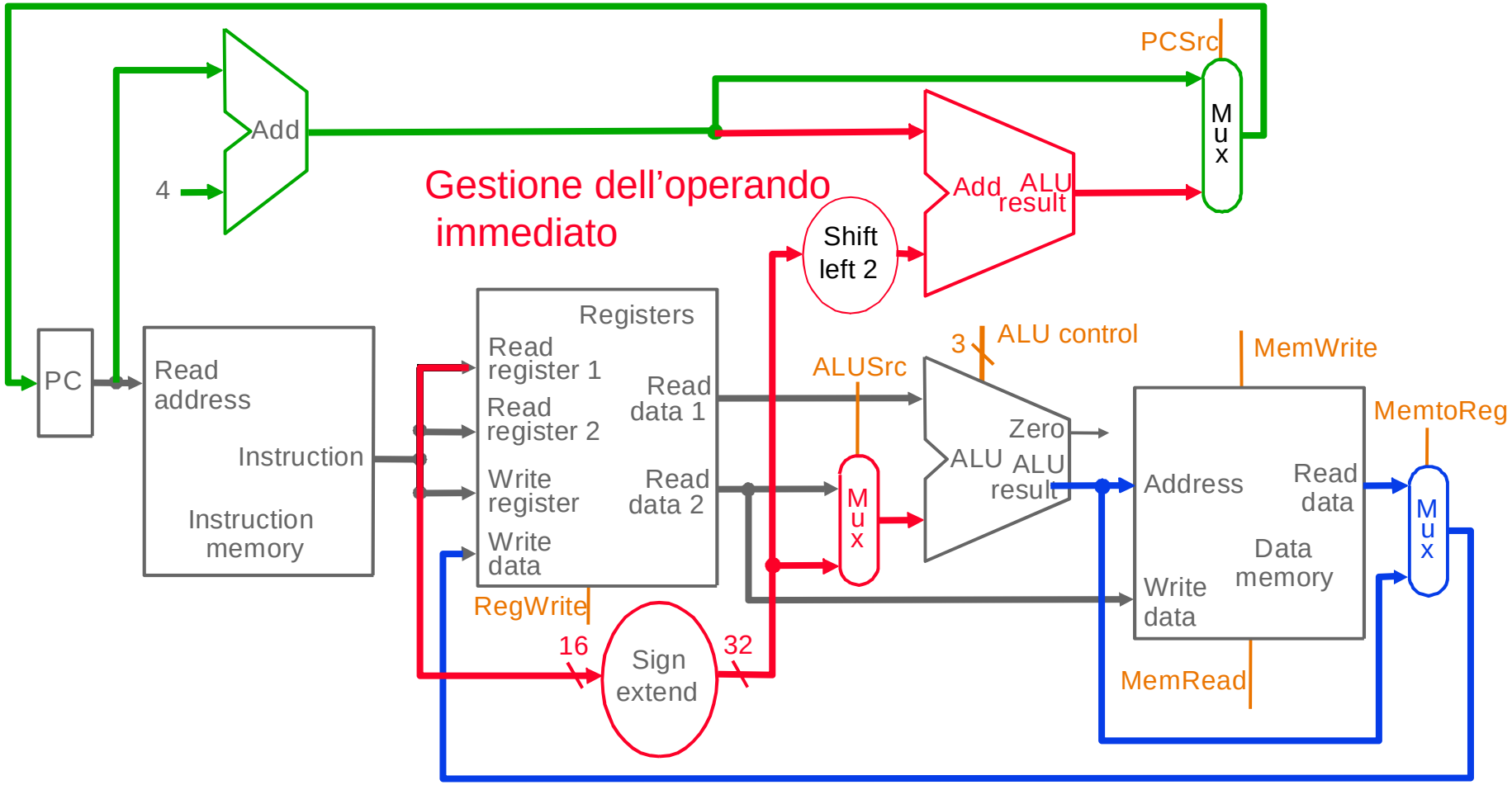


ESEMPIO: COSTRUZIONE DEL DATAPATH DEL MIPS

Costruzione del Datapath

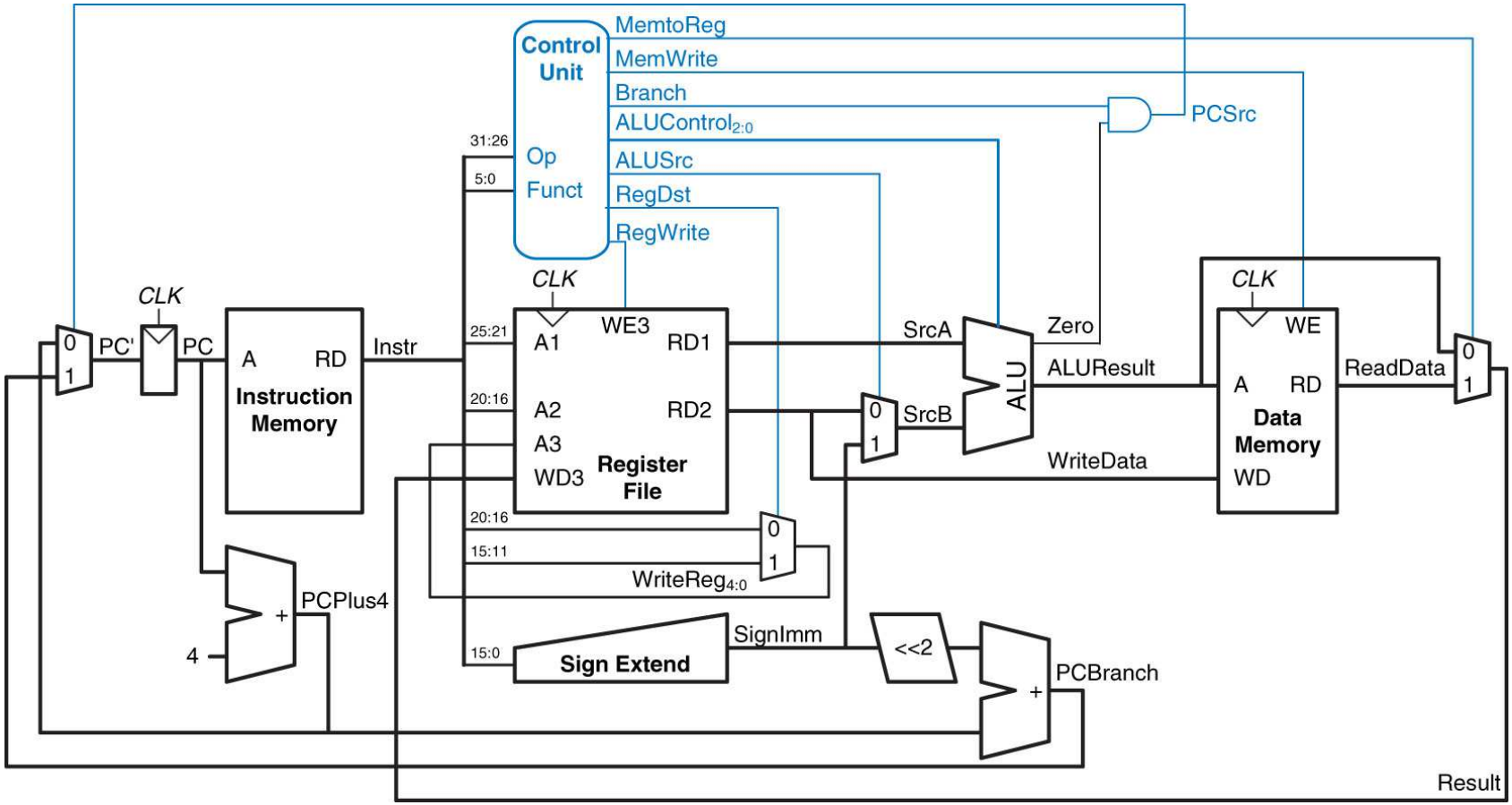
Gestione indirizzo istruzione successiva

I segnali di controllo vengono pilotati in base a “op-code” e “funct” dell’istruzione:



Gestione del risultato della ALU o dato dalla memoria

Datapath piu' preciso

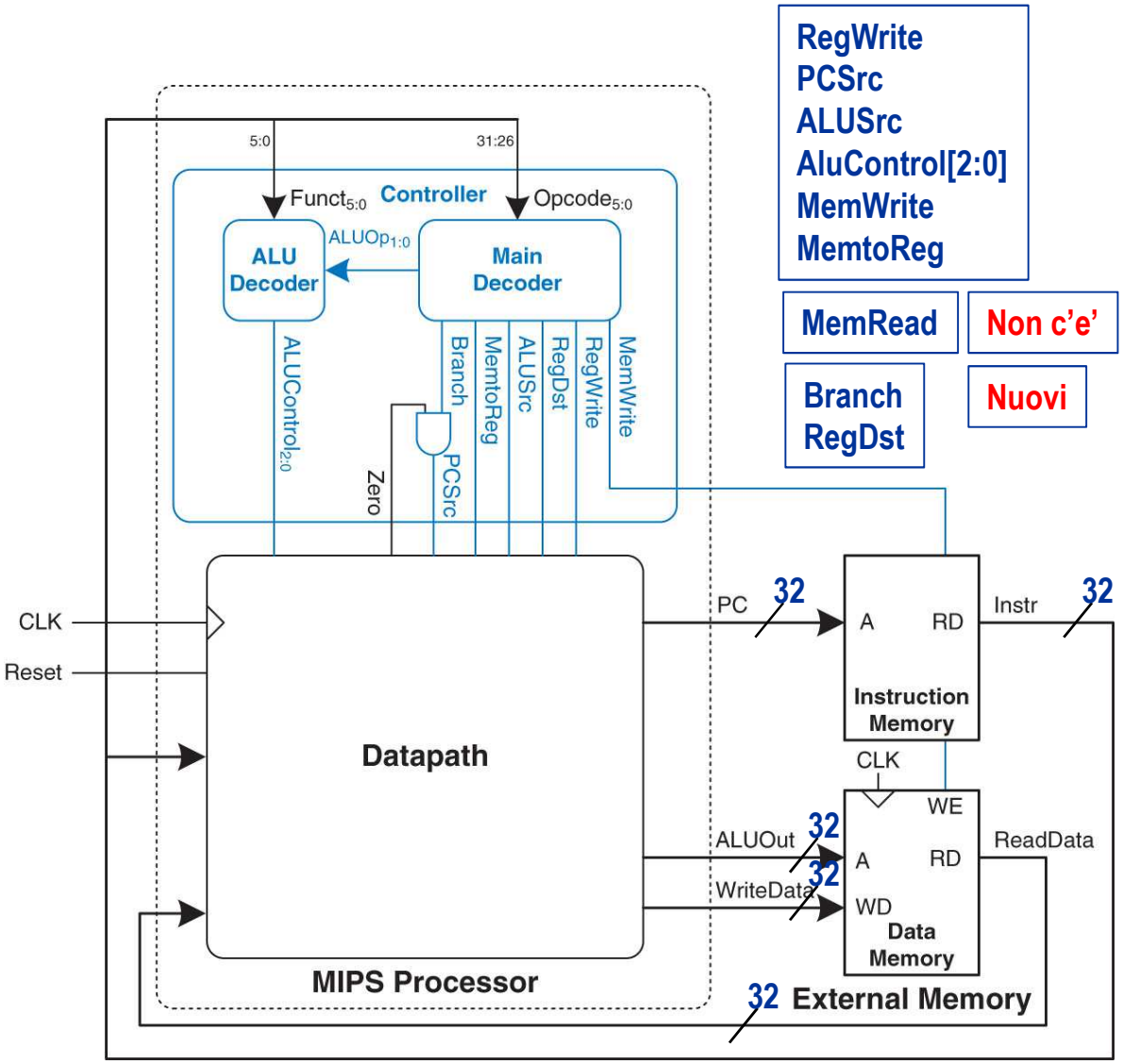


RegWrite
PCSrc
ALUSrc
AluControl[2:0]
MemWrite
MemtoReg

MemRead | Non c'e'

Branch
RegDst | Nuovi

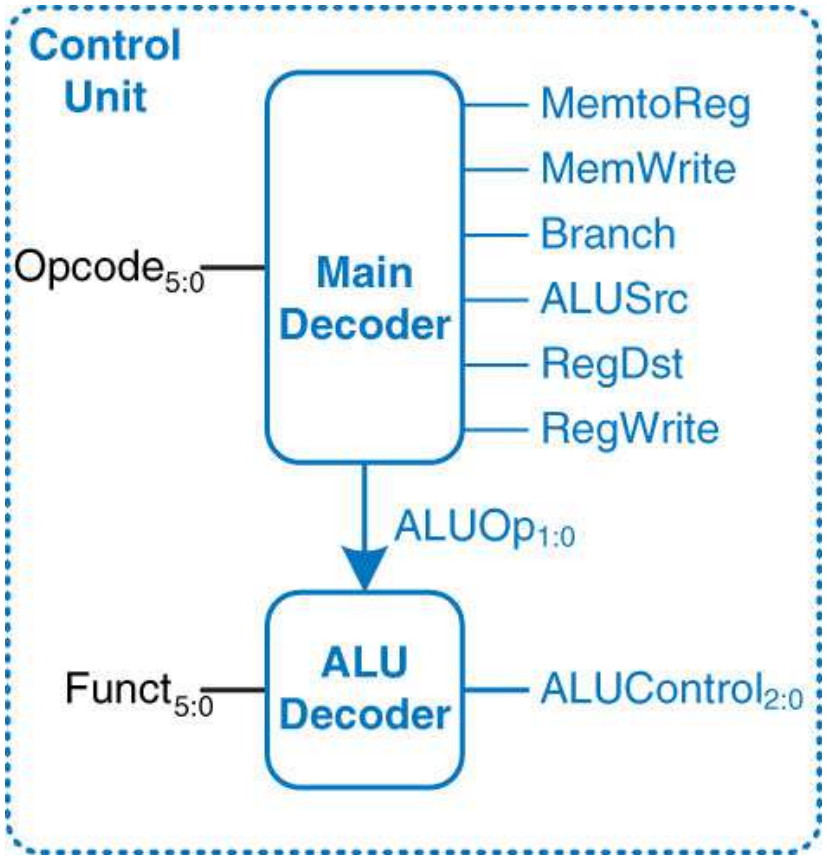
MIPS processor




```
module mips(clk, reset, readdata, instr
   pc, memwrite, aluout, writedata);
input clk, reset;
input [31:0] readdata;
input [31:0] instr;
output [31:0] pc;
output memwrite,
output [31:0] aluout, writedata;

wire memtoreg, alusrc, regdst,
       regwrite, pcsrc, zero;
wire [2:0] alucontrol;

controller c(instr[31:26], instr[5:0], zero,
             memtoreg, memwrite, pcsrc,
             alusrc, regdst, regwrite,
             alucontrol);

datapath dp(clk, reset, memtoreg, pcsrc,
             alusrc, regdst, regwrite,
             alucontrol,
             zero, pc, instr,
             aluout, writedata, readdata);

endmodule
```

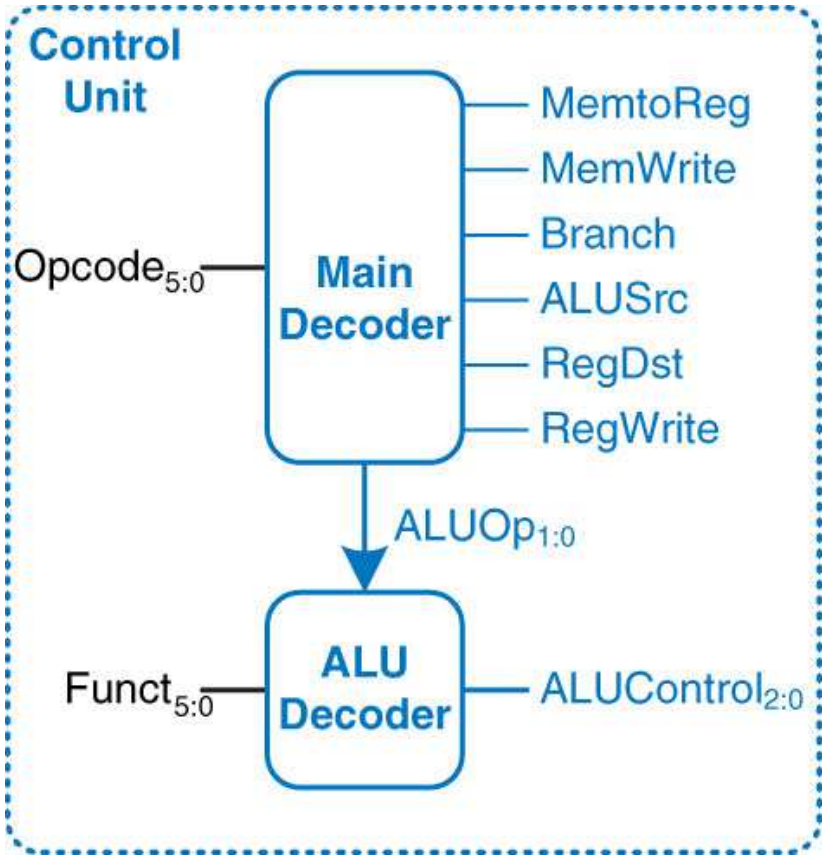
MIPS Controller

```
module controller(opcode, funct, zero,
   zero, memtoreg, memwrite, pcsrc, alusrc,
   regdst, regwrite, alucontrol);

input [5:0] opcode, funct;
input zero;
output memtoreg, memwrite;
output pcsrc, alusrc;
output regdst, regwrite;
output [2:0] alucontro;

wire [1:0] aluop;
wire branch;

maindec md(opcode, memtoreg, memwrite, branch,
alusrc, regdst, regwrite, aluop);

aludec ad(funct, aluop, alucontrol);

assign pcsrc = branch & zero;

endmodule
```



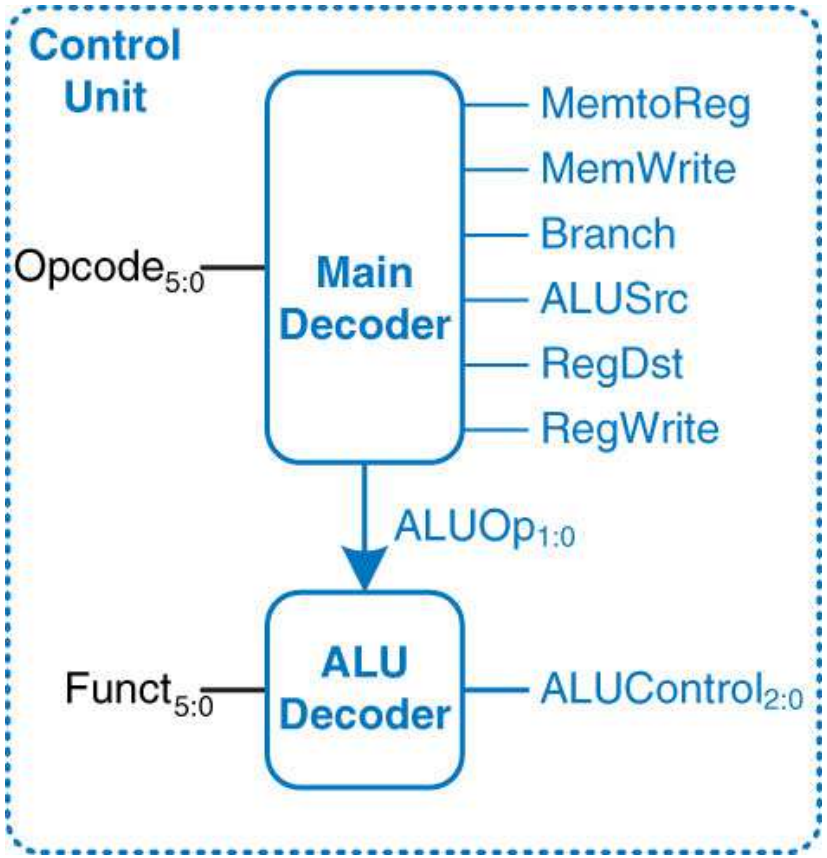
Main Decoder

```
module maindec(opcode,   memtoreg, memwrite,
  branch, alusrc, regdst, regwrite, aluop);

input [5:0] opcode;
output memtoreg, memwrite;
output branch, alusrc;
output regdst, regwrite;
output [1:0] aluop;

wire [7:0] controls;
assign {regwrite, regdst, alusrc, branch, memwrite,
memtoreg, aluop} = controls;

always @(opcode)
  case(opcode)
    6'b000000: controls <= 8'b11000000; // RTYPE
    6'b100011: controls <= 8'b10100100; // LW
    6'b101011: controls <= 8'b00101000; // SW
    6'b000100: controls <= 8'b00010001; // BEQ
    6'b001000: controls <= 8'b10100000; // ADDI
    default: controls <= 8'bxxxxxxxxx; // illegal op
  endcase
endmodule
```



ALU Decoder

```
module aludec(funct, aluop,   alucontrol);

input [5:0] funct,
input [1:0] aluop,
output [2:0] alucontrol;

always @(aluop)
  case(aluop)
    2'b00: alucontrol <= 3'b010; // add (for lw/sw/addi)
    2'b01: alucontrol <= 3'b110; // sub (for beq)
    default: case(funct) // R-type instructions
        6'b100000: alucontrol <= 3'b010; // add
        6'b100010: alucontrol <= 3'b110; // sub
        6'b100100: alucontrol <= 3'b000; // and
        6'b100101: alucontrol <= 3'b001; // or
        6'b101010: alucontrol <= 3'b111; // slt
        default: alucontrol <= 3'bxxx; // ???
     endcase
  endcase
endmodule
```



MIPS Datapath

Nota: I seguenti esempi vanno adattati alla sintassi Verilog !

```
module datapath(input clk, reset, input memtoreg, pcsrc, input alusrc, regdst, input regwrite, jump,
                input [2:0] alucontrol,output zero, output [31:0] pc, input [31:0] instr, output [31:0] aluout,
                writedata, input [31:0] readdata);

wire [4:0] writereg; wire [31:0] pcnext, pcnextbr, pcplus4, pcbranch; wire [31:0] signimm, signimmsh;
wire [31:0] srca, srcb; wire [31:0] result;

// next PC logic
flopr #(32)   pcreg(clk, reset, pcnext, pc);
adder         pcadd1(pc, 32'b100, pcplus4);
sl2           immsh(signimm, signimmsh);
adder         pcadd2(pcplus4, signimmsh, pcbranch);
mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28], instr[25:0], 2'b00}, jump, pcnext);

// register file logic
Regfile       rf(clk, regwrite, instr[25:21], instr[20:16], writereg, result, srca, writedata);
mux2 #(5)   wrmux(instr[20:16], instr[15:11], regdst, writereg);
mux2 #(32)  resmux(aluout, readdata, memtoreg, result);
signext       se(instr[15:0], signimm);

// ALU logic
mux2 #(32)  srcbmux(writedata, signimm, alusrc, srcb);
alu           alu(srca, srcb, alucontrol, aluout, zero);
endmodule
```

MIPS Registerfile and Adder

```
module regfile(input clk, input we3, input [4:0] ra1, ra2, wa3, input [31:0] wd3, output [31:0] rd1, rd2);

  reg [31:0] rf[31:0];
  // three ported register file
  // read two ports combinationally
  // write third port on rising edge of clk
  // register 0 hardwired to 0
  // note: for pipelined processor, write third port
  // on falling edge of clk

  always @(posedge clk)
     if (we3) rf[wa3] <= wd3;

  assign rd1 = (ra1 ! = 0) ? rf[ra1] : 0;
  assign rd2 = (ra2 ! = 0) ? rf[ra2] : 0;
endmodule
```

```
module adder(input [31:0] a, b, output [31:0] y);
   assign y = a + b;
endmodule
```

Shift-Left-2, Sign-Extension, Resettable Flip-flop, MUX2

```
module sl2(input [31:0] a, output [31:0] y);
   // shift left by 2
   assign y = {a[29:0], 2'b00};
endmodule
```

```
module signext(input [15:0] a, output [31:0] y);
   assign y = {{16{a[15]}}, a};
endmodule
```

```
module flopr #(parameter WIDTH = 8)(input clk, reset,
        input [WIDTH-1:0] d,
        output [WIDTH-1:0] q);
  always @(posedge clk, posedge reset)
     if (reset) q <= 0;
     else q <= d;
endmodule
```

```
module mux2 #(parameter WIDTH = 8) (input [WIDTH-1:0] d0, d1,
          input s, output [WIDTH-1:0] y);
   assign y = s ? d1 : d0;
endmodule
```

Instruction Memory and Data Memory

```
module imem(input [5:0] a, output [31:0] rd);
  reg [31:0] RAM[63:0];

  // initial $readmemh("memfile.dat", RAM);

  assign rd = RAM[a]; // word aligned
endmodule
```

```
module dmem(input clk, we, input [31:0] a, wd, output [31:0] rd);
  reg [31:0] RAM[63:0];

  assign rd = RAM[a[31:2]]; // word aligned
  always @(posedge clk)
  if (we) RAM[a[31:2]] <= wd;
endmodule
```

Top module

```
module top(input clk, reset, output [31:0] writedata, dataadr, output memwrite);
  reg [31:0] pc, instr, readdata;

  // instantiate processor and memories
  mips mips(clk, reset, pc, instr, memwrite, dataadr, writedata, readdata);
  imem imem(pc[7:2], instr);
  dmem dmem(clk, memwrite, dataadr, writedata, readdata);

endmodule
```

Esercizi

- Provare a descrivere in Verilog le reti combinatorie notevoli che abbiamo visto a lezione