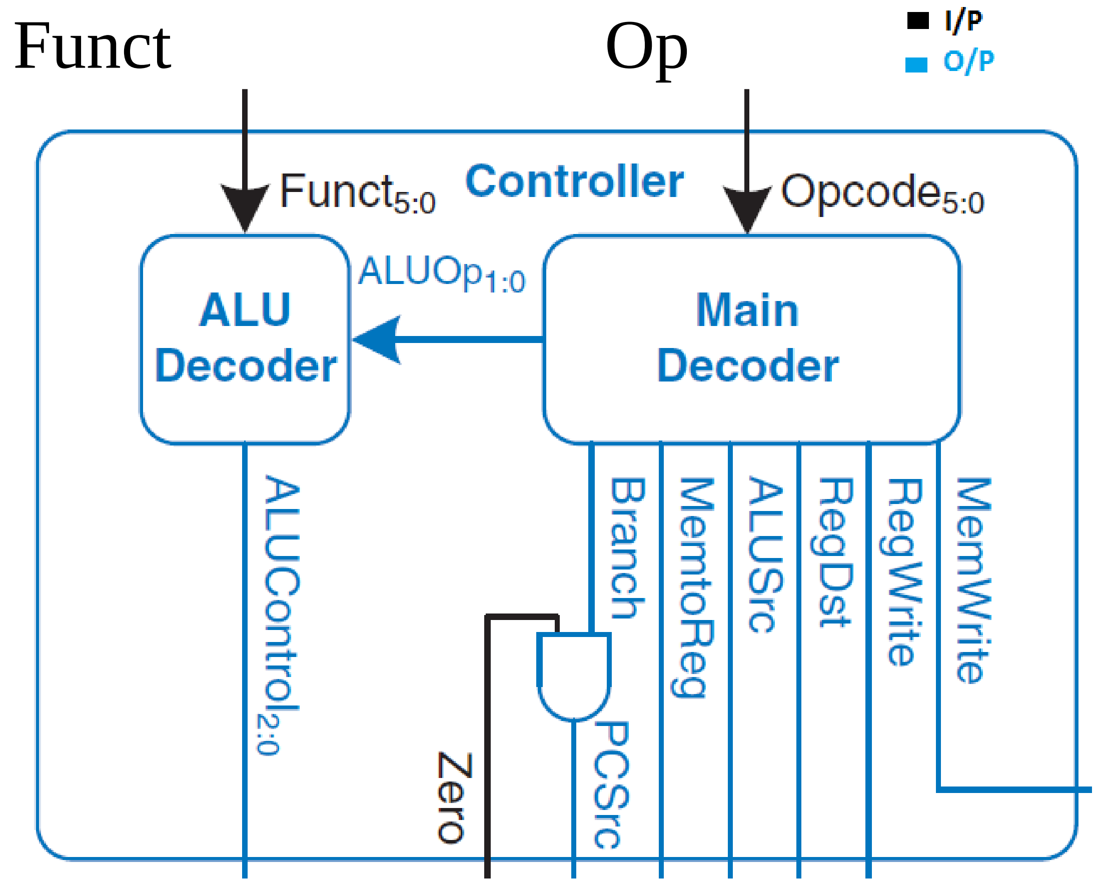
COMPUTER ARCHITECTURE LAB7 MIPS PROCESSOR

FCIS Ainshams University
Spring2021

AGENDA

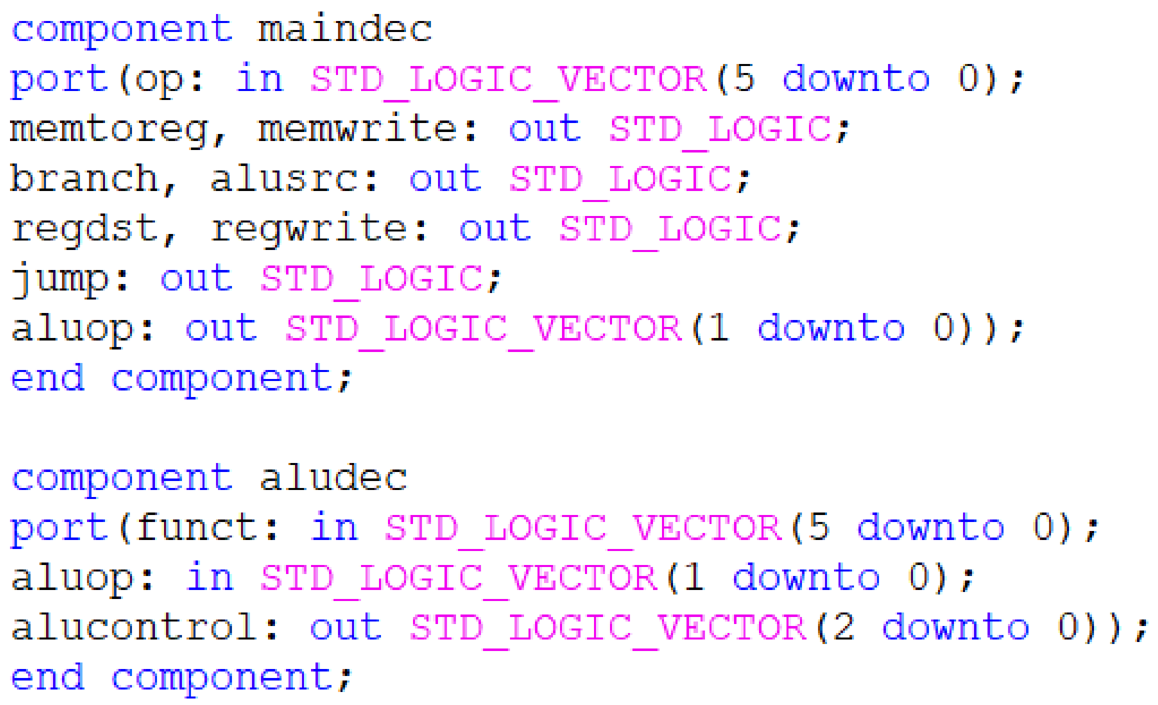
- Hands-On 1: Implementing Control Unit (CU) (Controller)
- Hands-On 2: Update MIPS Datapath to include J instruction.
- Hands-On 3: Implementing MIPS processor.

CONTROLLER/ CONTROL UNIT



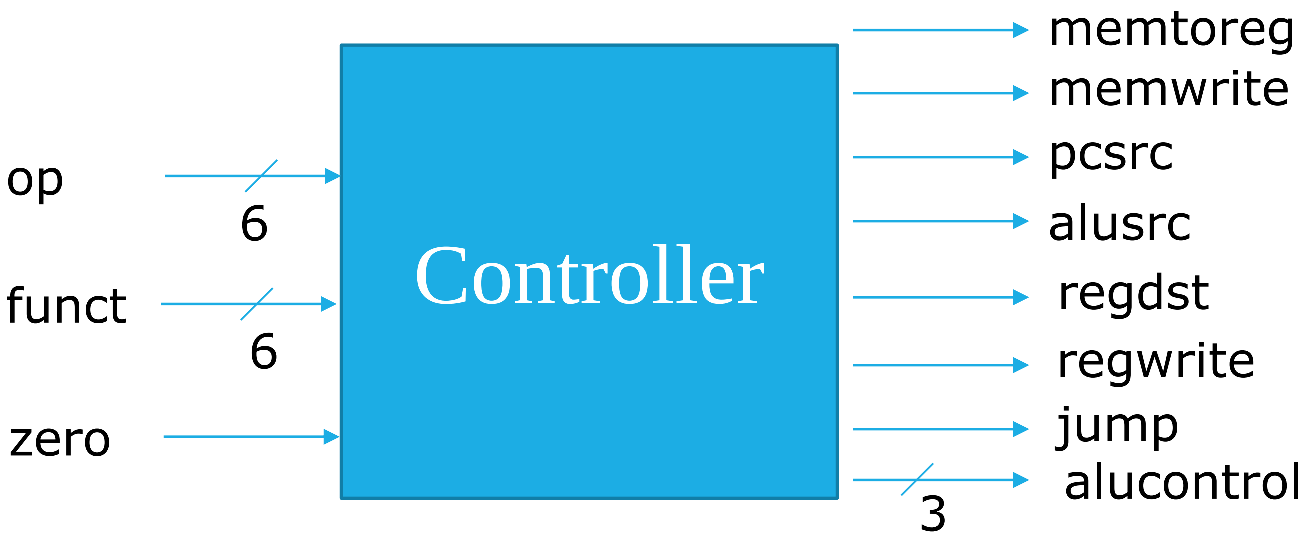
1) Create a project
2) Add maindec, aludec module to your project
3) In your package file, add 2 components for maindec and aludec
4) Create Controller module

CONTROLLER/ CONTROL UNIT



```
component maindec
port(op: in STD_LOGIC_VECTOR(5 downto 0);
memtoreg, memwrite: out STD_LOGIC;
branch, alusrc: out STD_LOGIC;
regdst, regwrite: out STD_LOGIC;
jump: out STD_LOGIC;
aluop: out STD_LOGIC_VECTOR(1 downto 0));
end component;

component aludec
port(funct: in STD_LOGIC_VECTOR(5 downto 0);
aluop: in STD_LOGIC_VECTOR(1 downto 0);
alucontrol: out STD_LOGIC_VECTOR(2 downto 0));
end component;
```

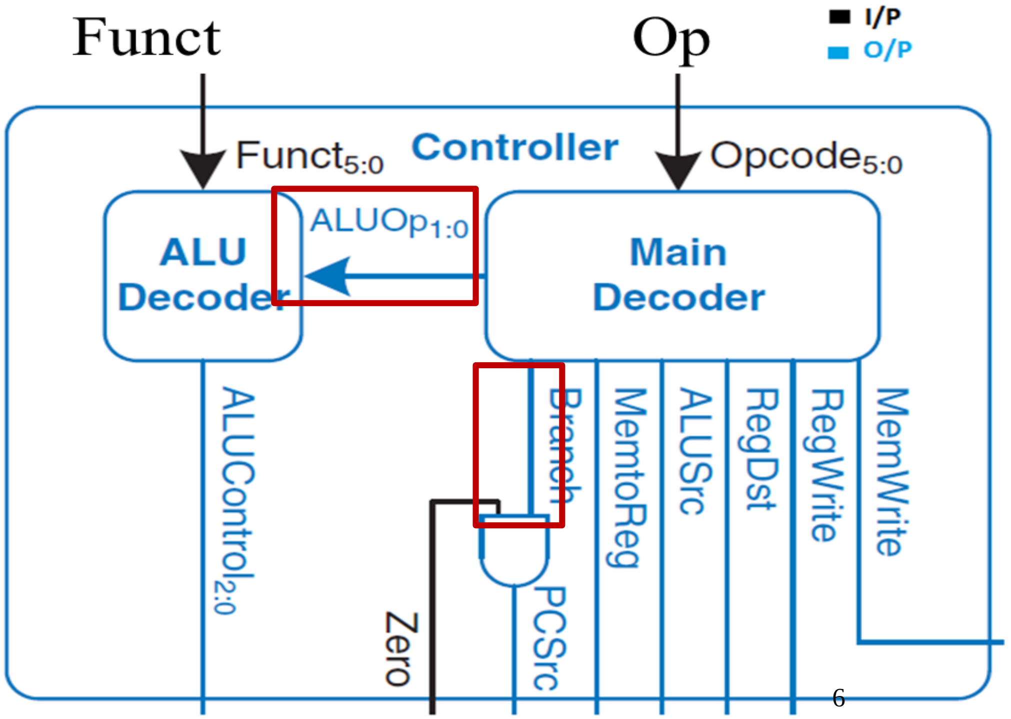
HANDS-ON1: CONTROLLER ENTITY



```
entity controller is -- Alu Controller
port(op, funct: in STD_LOGIC_VECTOR(5 downto 0);
zero: in STD_LOGIC;
memtoreg, memwrite: out STD_LOGIC;
pcsrc, alusrc: out STD_LOGIC;
regdst, regwrite: out STD_LOGIC;
jump: out STD_LOGIC;
alucontrol: out STD_LOGIC_VECTOR(2 downto 0));
end;
```

HANDS-ON 1:CONTROLLER (START UP)

```
-- Hands-On 2 steps
-- 1. add main and alu decoders' modules into controller project
-- 2. add main and alu decoders as components in architecture declaration section
-- 3. write ALU controller logic

architecture struct of controller is

signal aluop: STD_LOGIC_VECTOR(1 downto 0);
signal branch: STD_LOGIC;

begin


end;
```



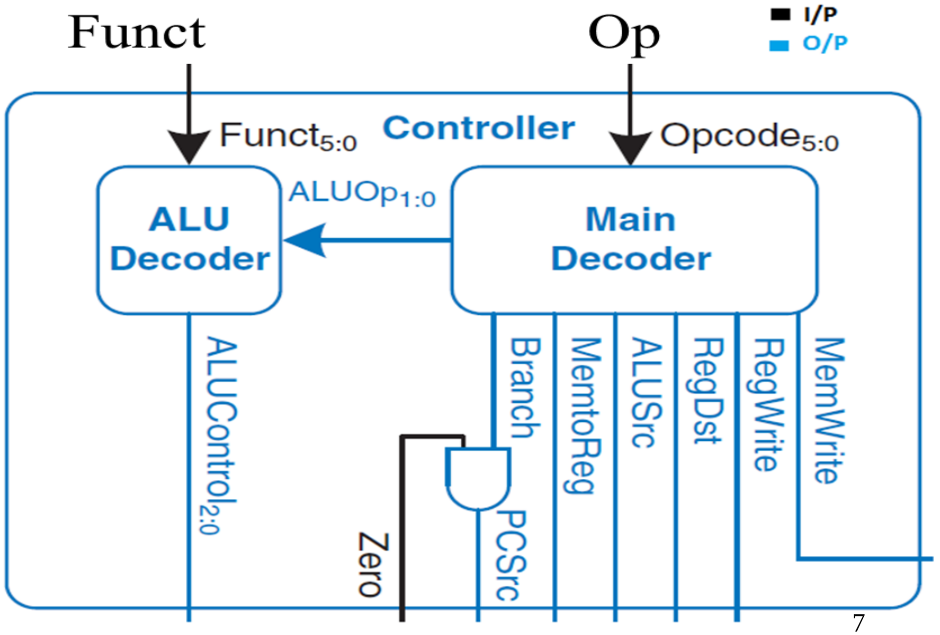
HANDS-ON 1: CONTROLLER (LOGIC)

A. Port Map the maindec
- I/P of maindec is “op” I/P of controller
- O/P of maindec is outputs of controller except: Aluop, branch (sig)

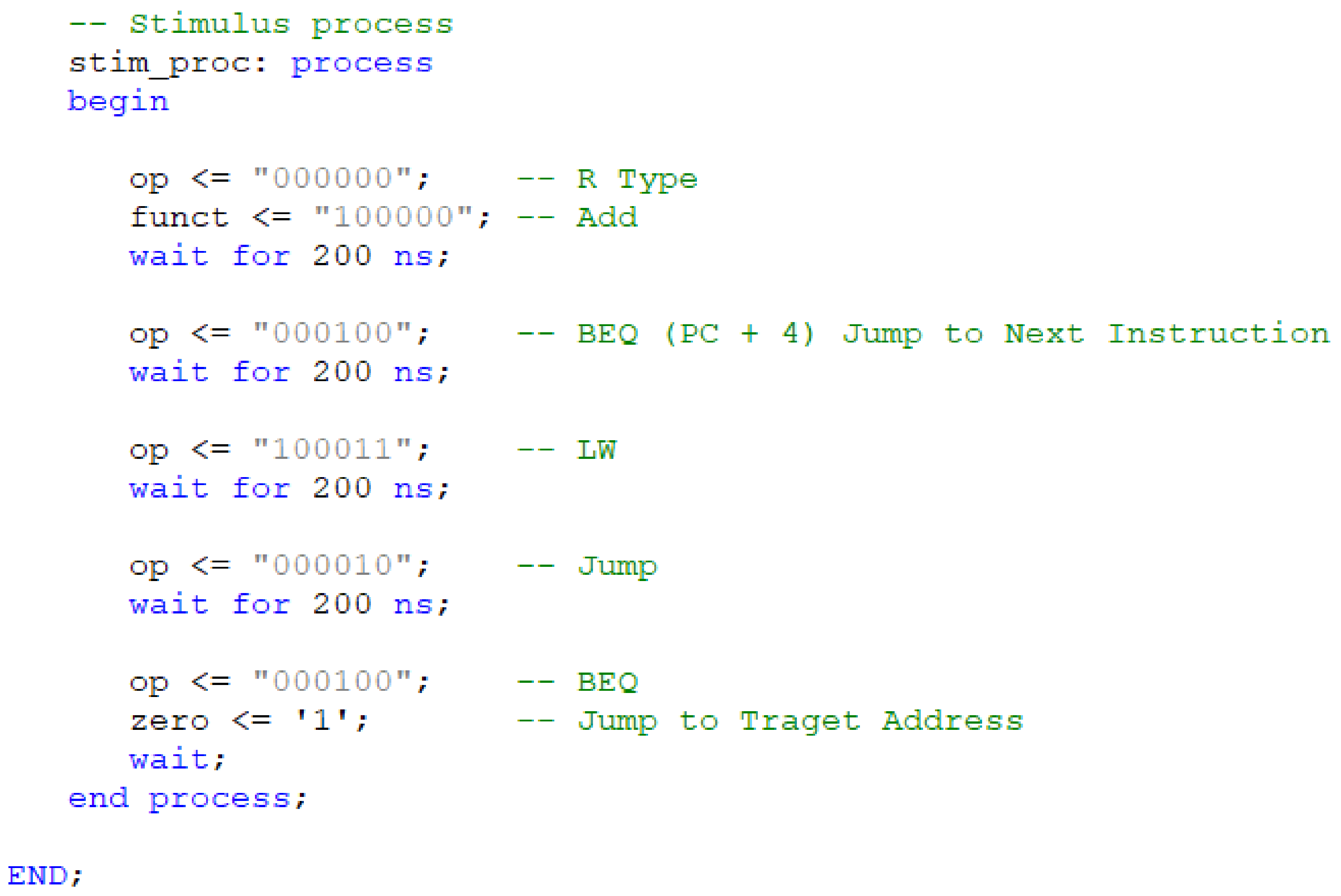
B. Port Map the aludec
- I/P of aludec are (“funct” I/P of controller) , (“Aluop” signal)
- O/P of maindec is output of controller

C. Generate PCsrc signal

and output out of controller

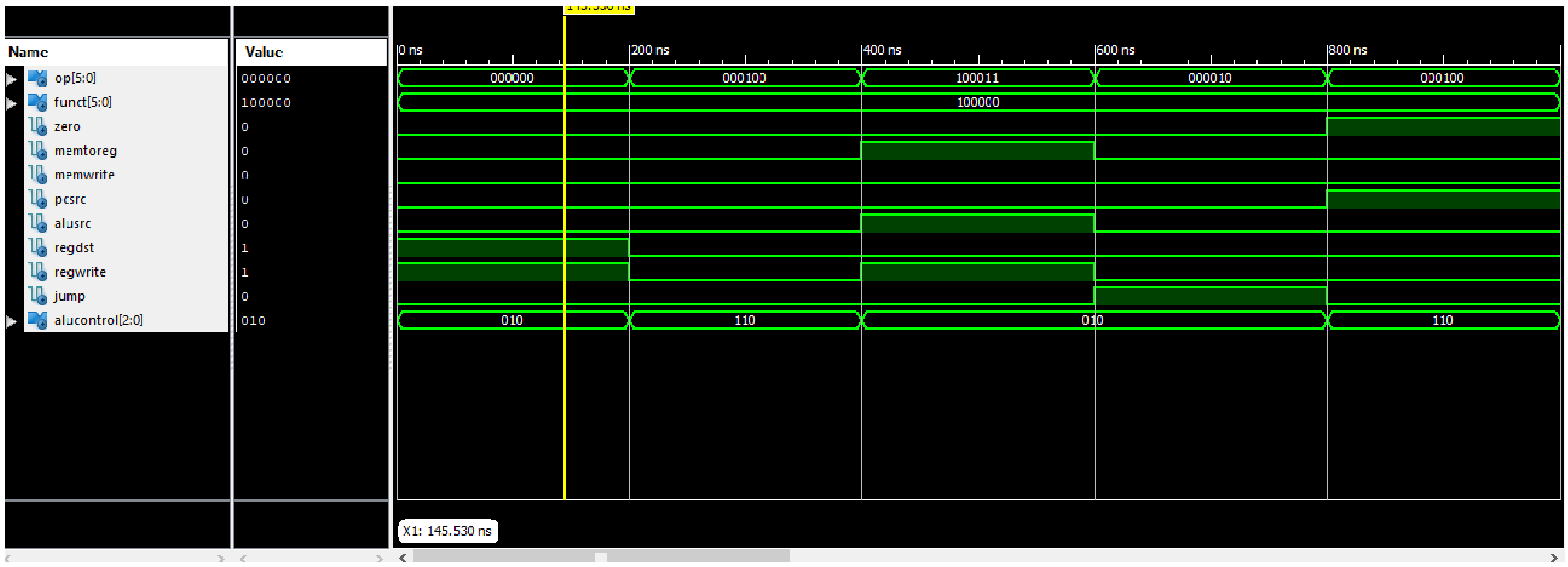


HANDS-ON 1: ALU TEST CASE

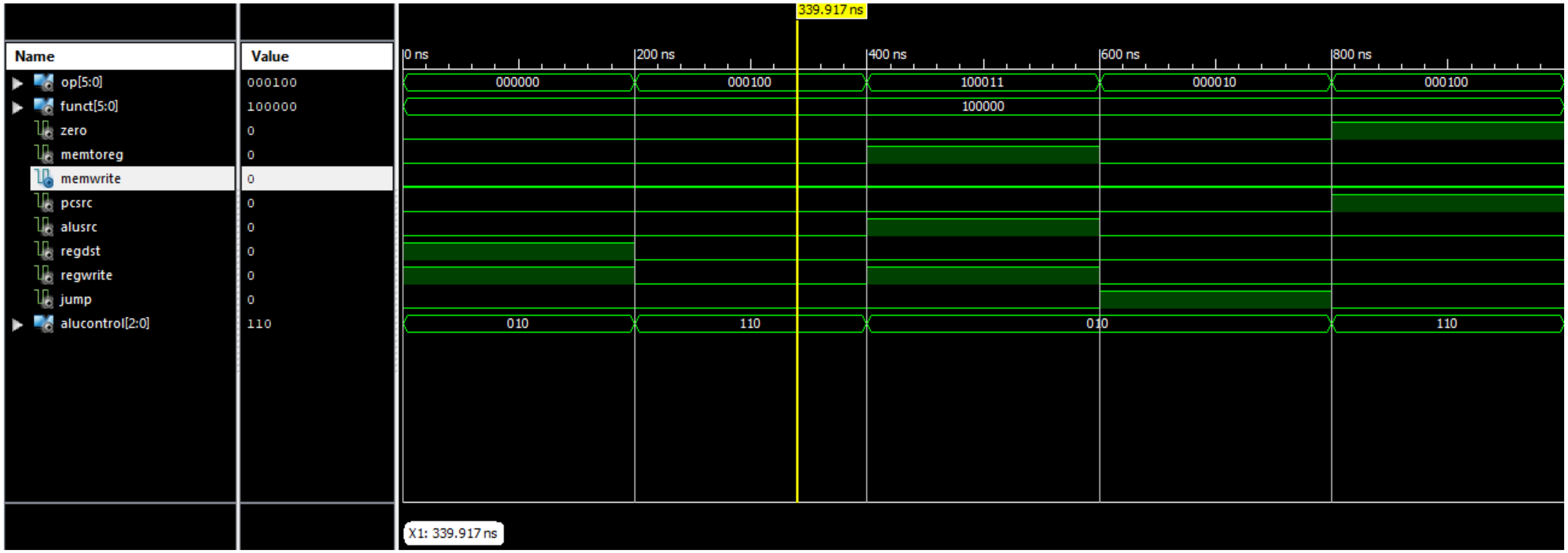


```
   -- Stimulus process
   stim_proc: process
   begin

      op <= "000000";     -- R Type
      funct <= "100000";  -- Add
      wait for 200 ns;

      op <= "000100";     -- BEQ (PC + 4) Jump to Next Instruction
      wait for 200 ns;

      op <= "100011";     -- LW
      wait for 200 ns;

      op <= "000010";     -- Jump
      wait for 200 ns;

      op <= "000100";     -- BEQ
      zero <= '1';        -- Jump to Traget Address
      wait;
   end process;

END;
```

HANDS-ON 1: ALU SIM. RESULT (R-TYPE ADD)



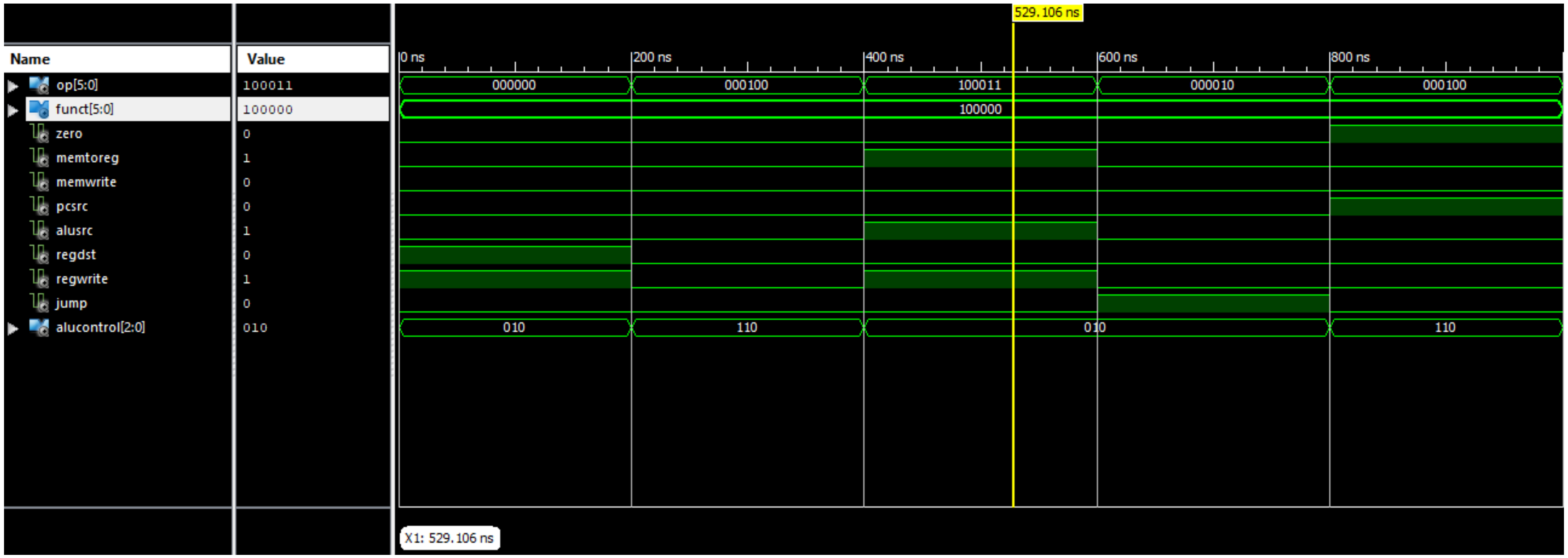
HANDS-ON 1: ALU SIM. RESULT (BEQ - PC)



Name	Value
op[5:0]	000100
funct[5:0]	100000
zero	0
memtoreg	0
memwrite	0
pcsrc	0
alusrc	0
regdst	0
regwrite	0
jump	0
alucontrol[2:0]	110

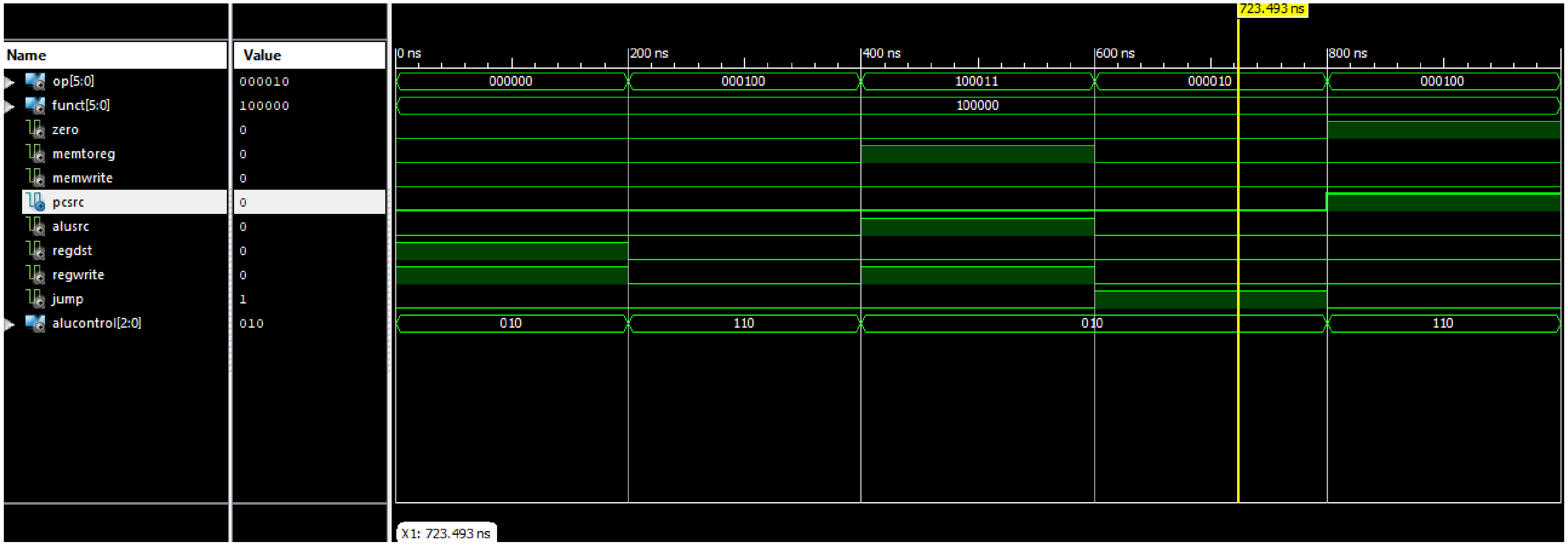
```
op[5:0]          000000 | 000100 | 100011 | 000010 | 000100
funct[5:0]       100000
alucontrol[2:0]  010    | 110    | 010             | 110
```

HANDS-ON 1: ALU SIM. RESULT (LW)



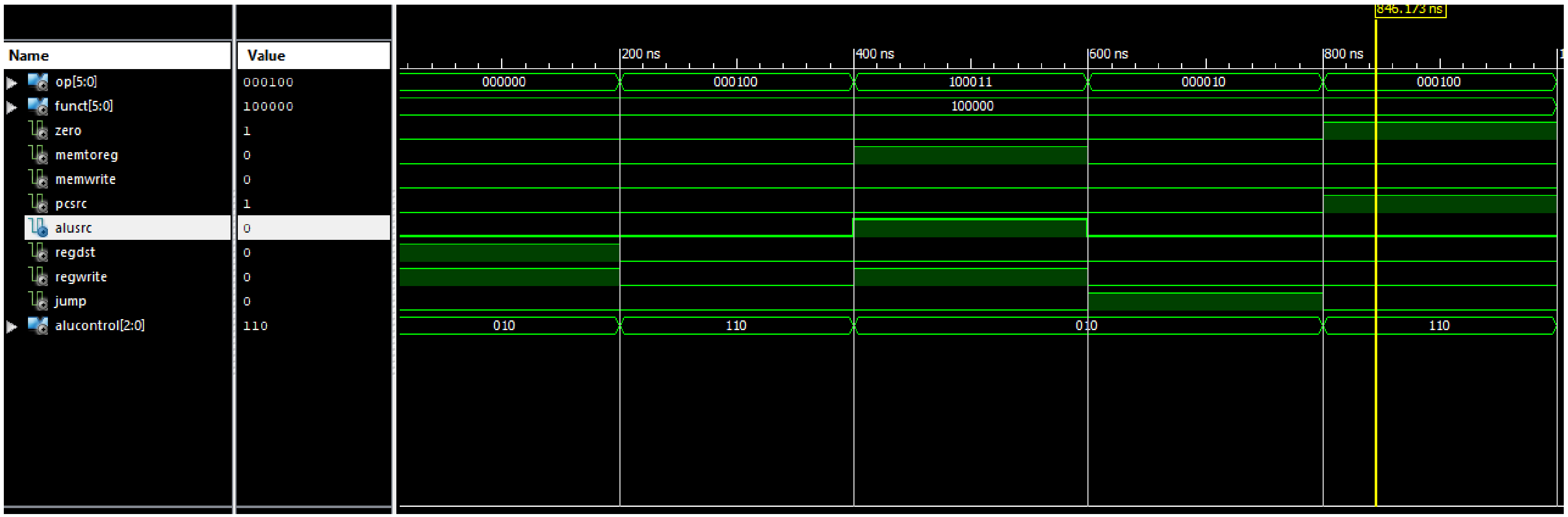
Name	Value
op[5:0]	100011
funct[5:0]	100000
zero	0
memtoreg	1
memwrite	0
pcsrc	0
alusrc	1
regdst	0
regwrite	1
jump	0
alucontrol[2:0]	010

HANDS-ON 1: ALU SIM. RESULT (JUMP)



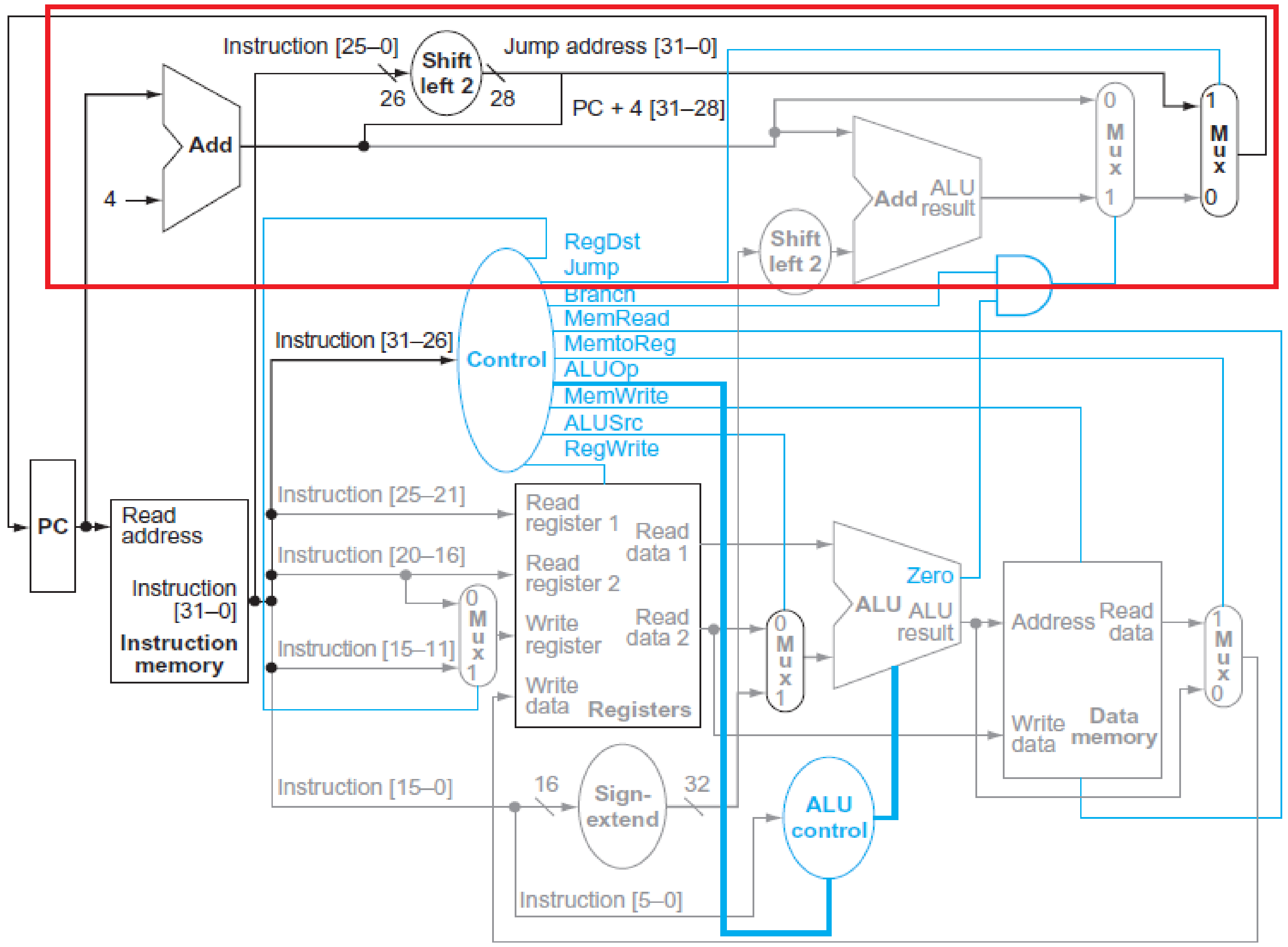
```
op[5:0]          000010   000000 | 000100 | 100011 | 000010 | 000100
funct[5:0]       100000   100000
zero             0
memtoreg         0
memwrite         0
pcsrc            0
alusrc           0
regdst           0
regwrite         0
jump             1
alucontrol[2:0]  010      010 | 110 | 010 | 110
```

HANDS-ON 1: ALU SIM. RESULT (BEQ – TARGET ADDRESS)



Name	Value
op[5:0]	000100
funct[5:0]	100000
zero	1
memtoreg	0
memwrite	0
pcsrc	1
alusrc	0
regdst	0
regwrite	0
jump	0
alucontrol[2:0]	110

HANDS-ON 2: DATAPATH UPDATE



HANDS-ON 2: DATAPATH UPDATE

```
entity datapath is -- MIPS datapath
    port(clk, reset: in STD_LOGIC;
    readdata: in STD_LOGIC_VECTOR(31 downto 0);
    instr: in STD_LOGIC_VECTOR(31 downto 0);
    memtoreg, pcsrc,alusrc,regwrite, regdst, jump: in STD_LOGIC;
    aluoperation: in STD_LOGIC_VECTOR(2 downto 0);
    zero: out STD_LOGIC;
    pc: buffer STD_LOGIC_VECTOR(31 downto 0);
  aluout, writedata: buffer STD_LOGIC_VECTOR(31 downto 0))
end;
```

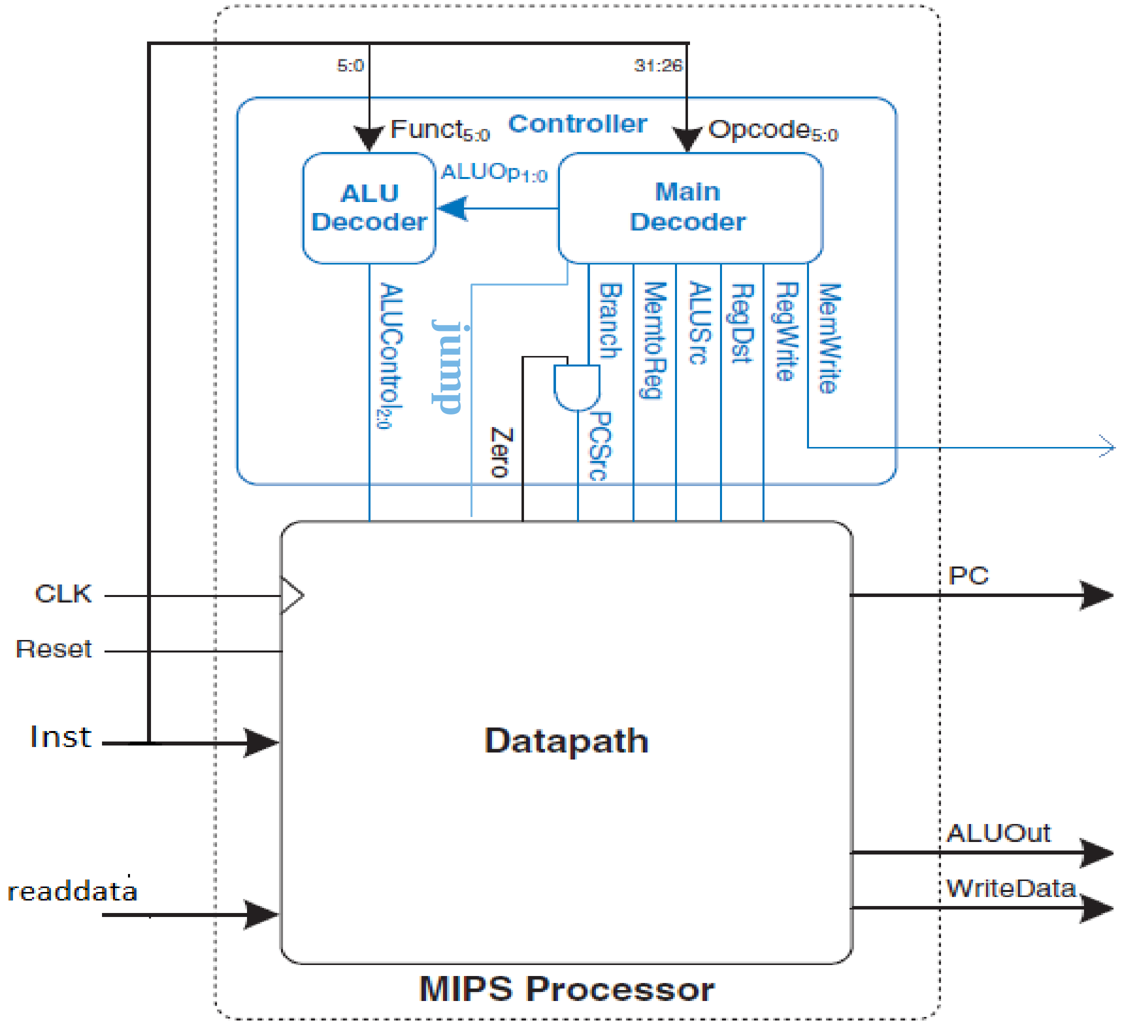
Do update it in datapath.vhd file

HANDS-ON 2: DATAPATH UPDATE

```
architecture struct of datapath is
signal writereg: STD_LOGIC_VECTOR(4 downto 0);
signal pcjump, pcnext, pcnextbr, pcplus4,
pcbranch: STD_LOGIC_VECTOR(31 downto 0);
signal signimm, signimmsh: STD_LOGIC_VECTOR(31 downto 0);
signal srca, srcb, result: STD_LOGIC_VECTOR(31 downto 0);
Begin

— next PC logic
pcjump <= pcplus4(31 downto 28) & instr(25 downto 0) & "00";
pcreg: flopr generic map(32) port map(clk, reset, pcnext, pc);
pcadd1: adder port map(pc, X"00000004", pcplus4);
immsh: sl2 port map(signimm, signimmsh);
pcadd2: adder port map(pcplus4, signimmsh, pcbranch);
pcbrmux: mux2 generic map(32) port map(pcplus4, pcbranch,
pcsrc, pcnextbr);
pcmux: mux2 generic map(32) port map(pcnextbr, pcjump, jump,
pcnext);
```

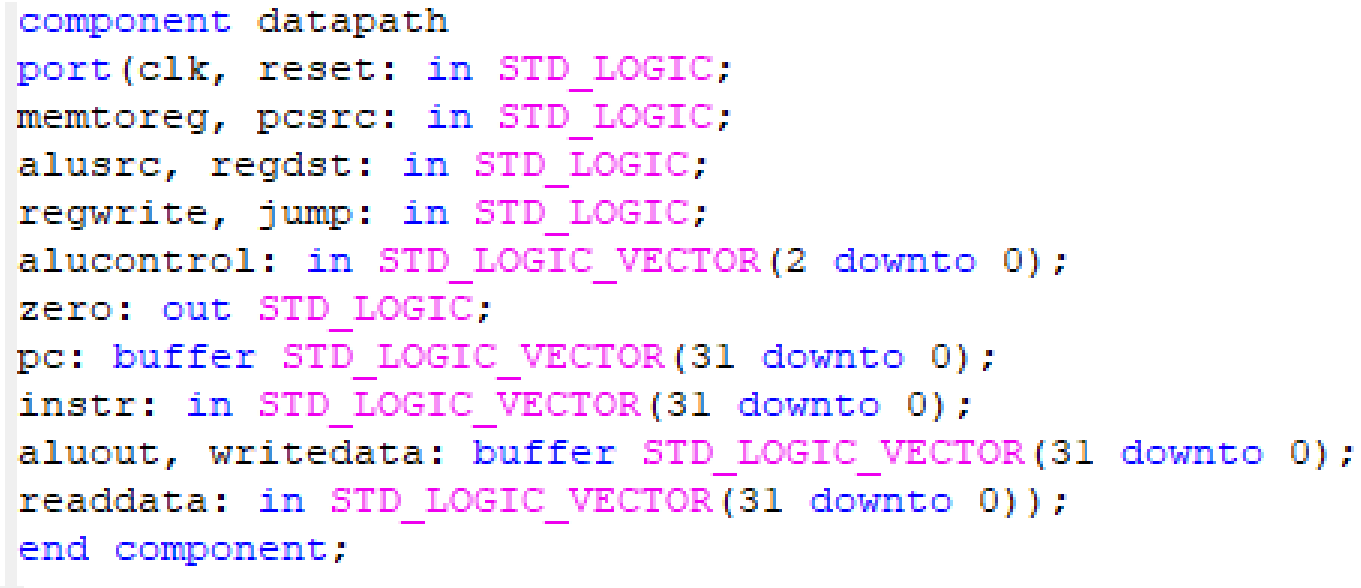
Do update it in datapath.vhd file

HANDS-ON 3: MIPS PROCESSOR

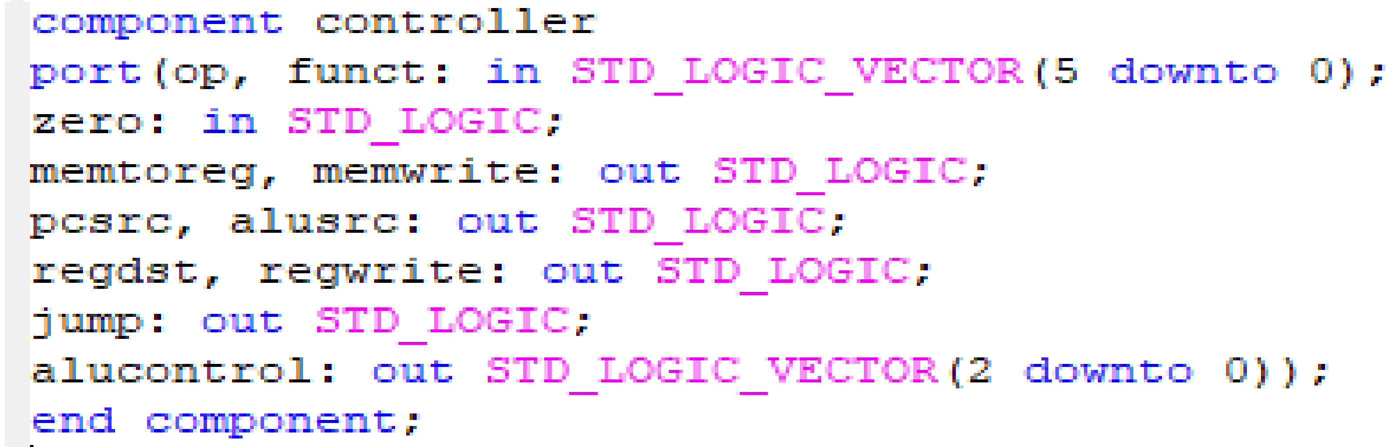


HANDS-ON 3: MIPS PROCESSOR

I. Add the Datapath and Controller Modules as components in your package.



```
component datapath
port(clk, reset: in STD_LOGIC;
memtoreg, pcsrc: in STD_LOGIC;
alusrc, regdst: in STD_LOGIC;
regwrite, jump: in STD_LOGIC;
alucontrol: in STD_LOGIC_VECTOR(2 downto 0);
zero: out STD_LOGIC;
pc: buffer STD_LOGIC_VECTOR(31 downto 0);
instr: in STD_LOGIC_VECTOR(31 downto 0);
aluout, writedata: buffer STD_LOGIC_VECTOR(31 downto 0);
readdata: in STD_LOGIC_VECTOR(31 downto 0));
end component;
```



```
component controller
port(op, funct: in STD_LOGIC_VECTOR(5 downto 0);
zero: in STD_LOGIC;
memtoreg, memwrite: out STD_LOGIC;
pcsrc, alusrc: out STD_LOGIC;
regdst, regwrite: out STD_LOGIC;
jump: out STD_LOGIC;
alucontrol: out STD_LOGIC_VECTOR(2 downto 0));
end component;
```

HANDS-ON3: MIPS PROCESSOR

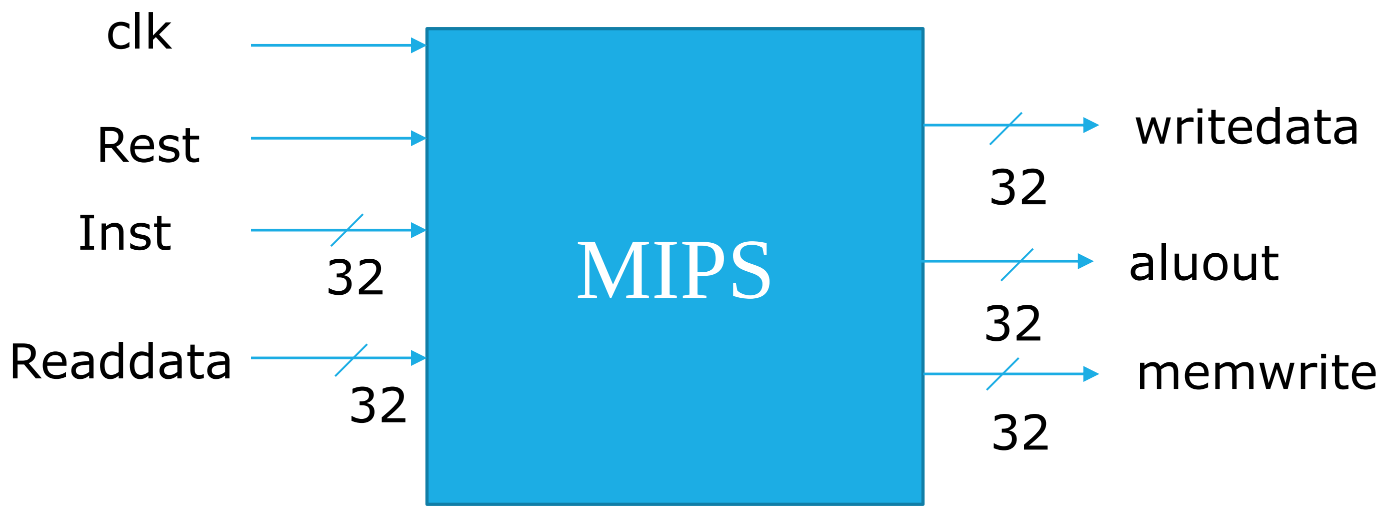
All vhd files and components till now are:

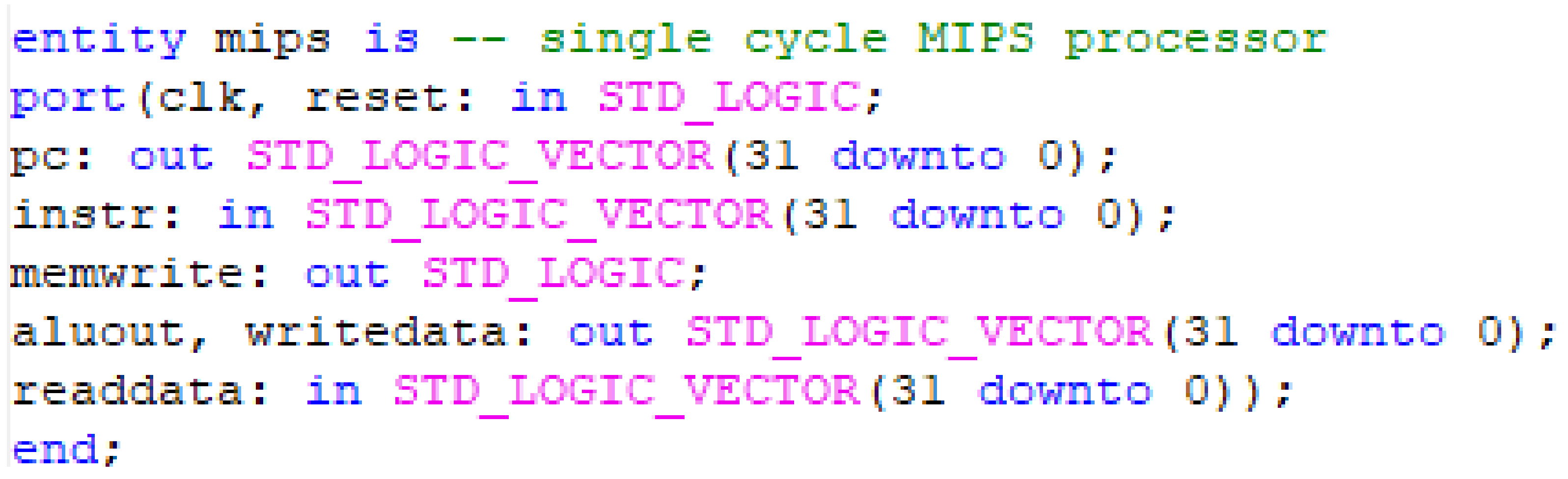
- Controller
- Maindec
- Aludec
- Datapath
- Registerfile
- ALU
- Mux2
- Sl2
- Adder
- Signext
- Flopr

HANDS-ON3: MIPS PROCESSOR

II. Now: Create main module for MIPS

HANDS-ON3: MIPS PROCESSOR(ENTITY)

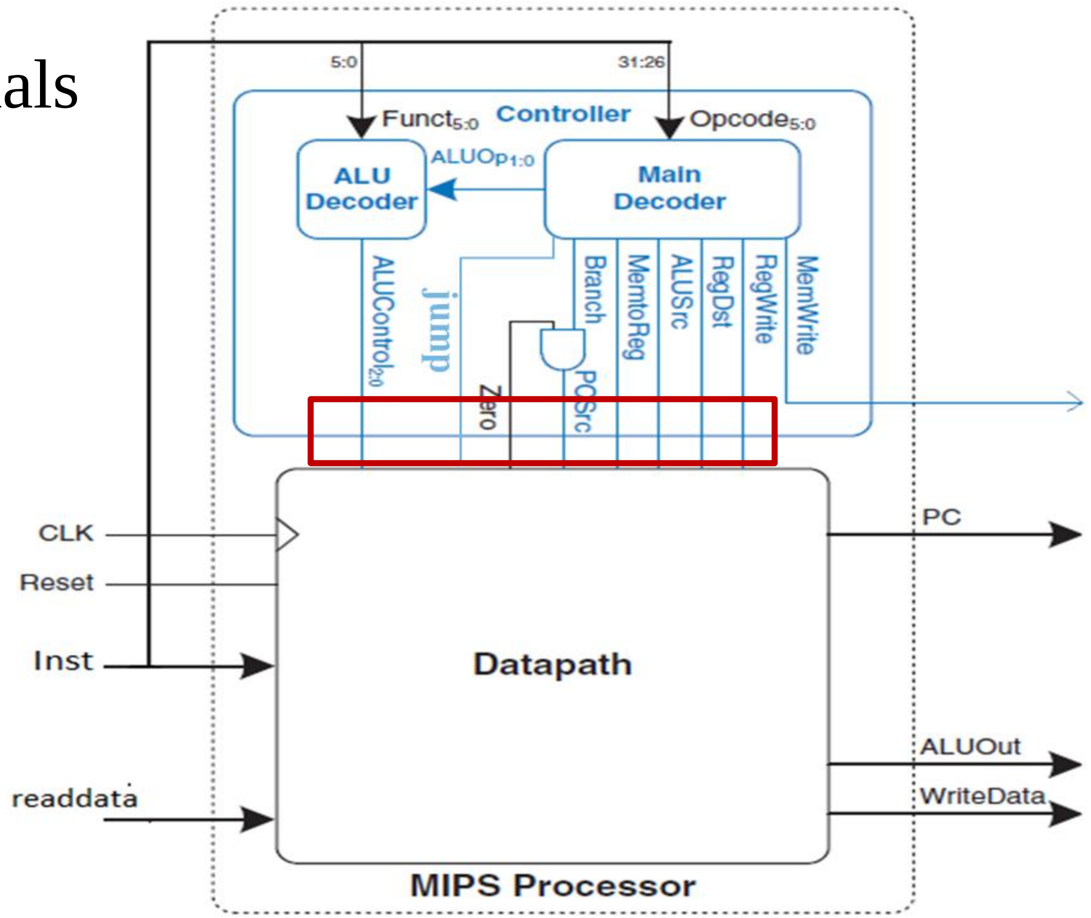




```
entity mips is -- single cycle MIPS processor
port(clk, reset: in STD_LOGIC;
pc: out STD_LOGIC_VECTOR(31 downto 0);
instr: in STD_LOGIC_VECTOR(31 downto 0);
memwrite: out STD_LOGIC;
aluout, writedata: out STD_LOGIC_VECTOR(31 downto 0);
readdata: in STD_LOGIC_VECTOR(31 downto 0));
end;
```

HANDS-ON3: MIPS PROCESSOR(ARCHI)

We will need some signals



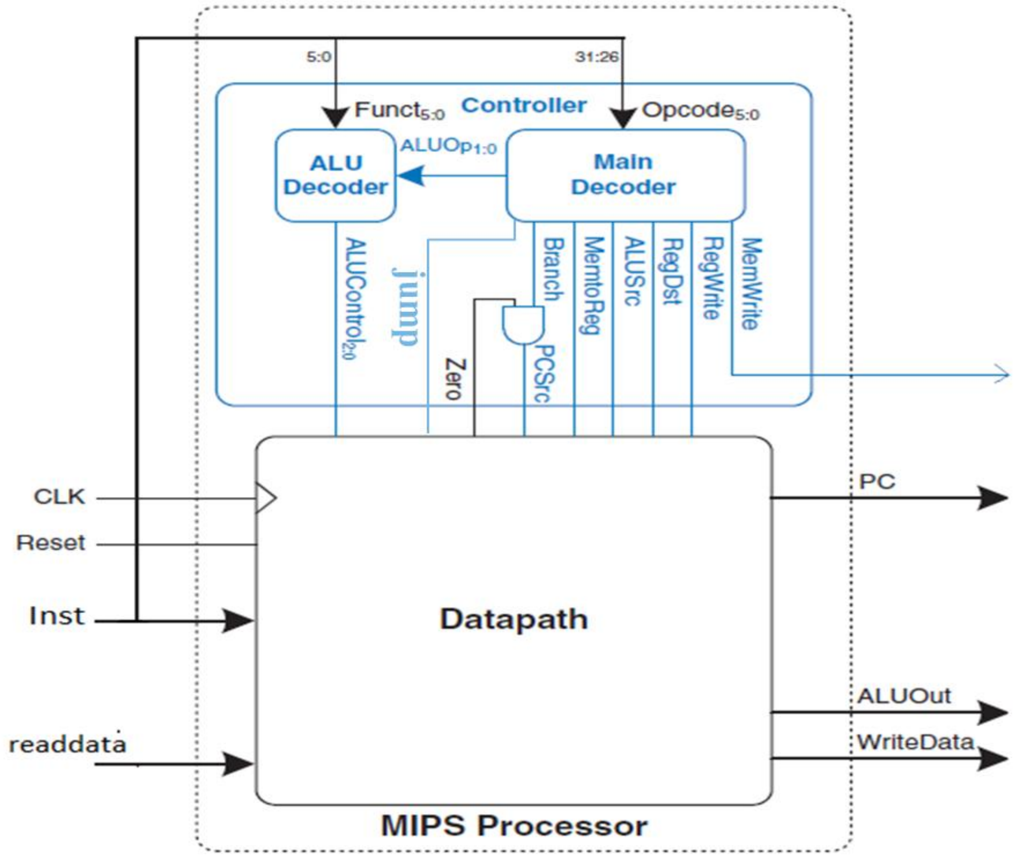
```
signal memtoreg, alusrc, regdst, regwrite, jump, pcsrc: STD_LOGIC;
signal zero: STD_LOGIC;
signal alucontrol: STD_LOGIC_VECTOR(2 downto 0);
```

HANDS-ON3: MIPS PROCESSOR(ARCHI)

Behavior:

A. Port MAP the controller
   - I/Ps are |"Instr" parts
   - O/Ps are signals except "Memwrite" (O/P of MIPS)

B. Port Map the datapath
   - Data I/Ps are MIPS I/Ps
   - Control I/Ps are signals
   - O/Ps are MIPS O/Ps



Don’t Forget to bring your project next lab to test the processor.

Thanks