

COMPUTER ARCHITECTURE

LAB 6

CONTROLLER

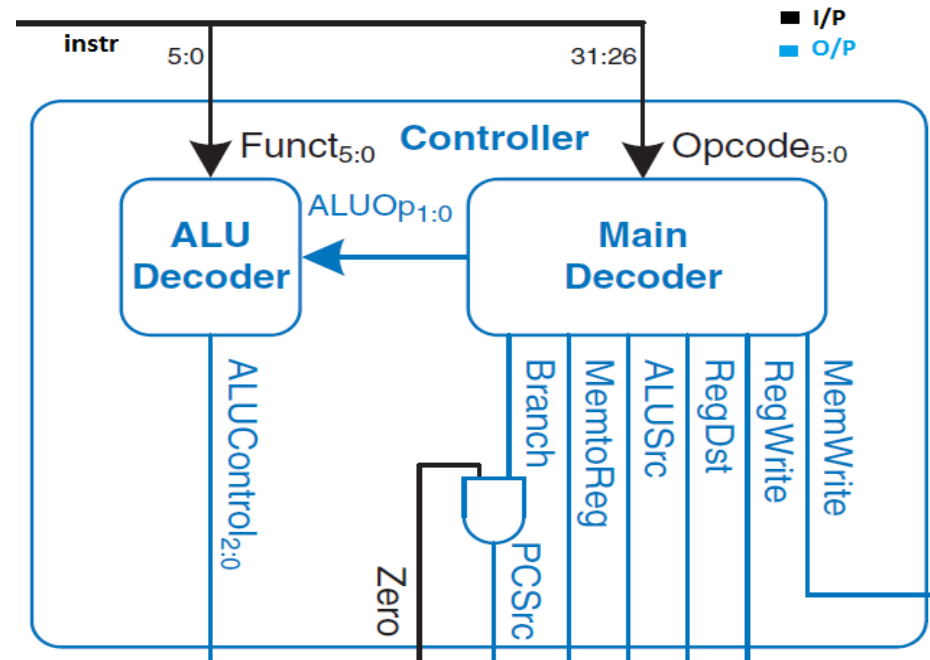
FCIS Ainshams University
Spring 2021

AGENDA

- What is Control Unit (CU).
- Design of Main Decoder.
- Hands On 1 – Implementing Main Decoder Module.
- Design of ALU control.
- Hands-On 2 – Implementing ALU Decoder Module.

CONTROL UNIT (CONTROLLER)

- The control unit (CU) is a component of a computer's **central processing unit** (CPU) that directs the operation of the **processor**.
- It tells the computer's **memory**, ALU, others how to respond to the instructions that have been sent to the **processor**.



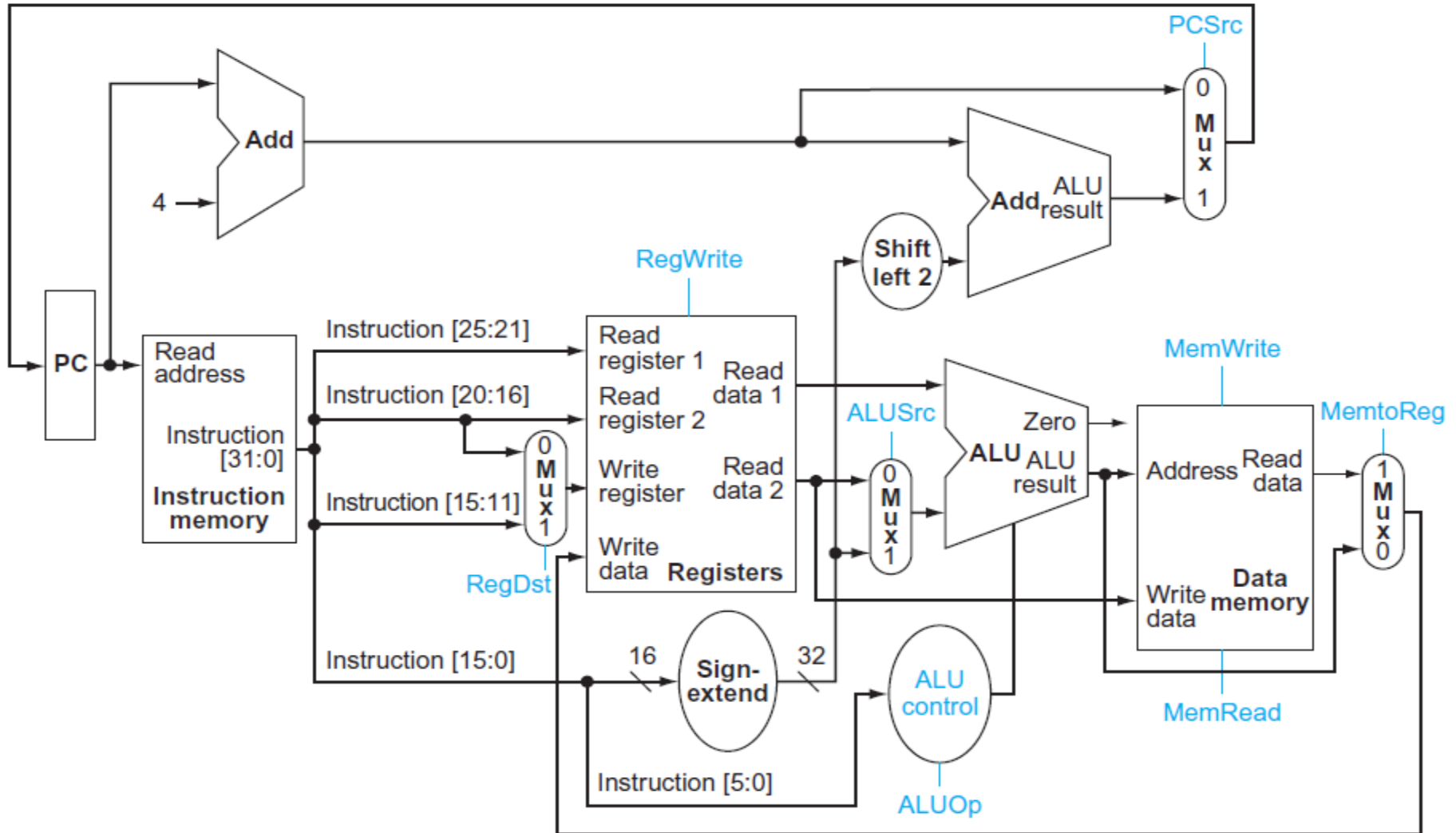


1) Main Decoder

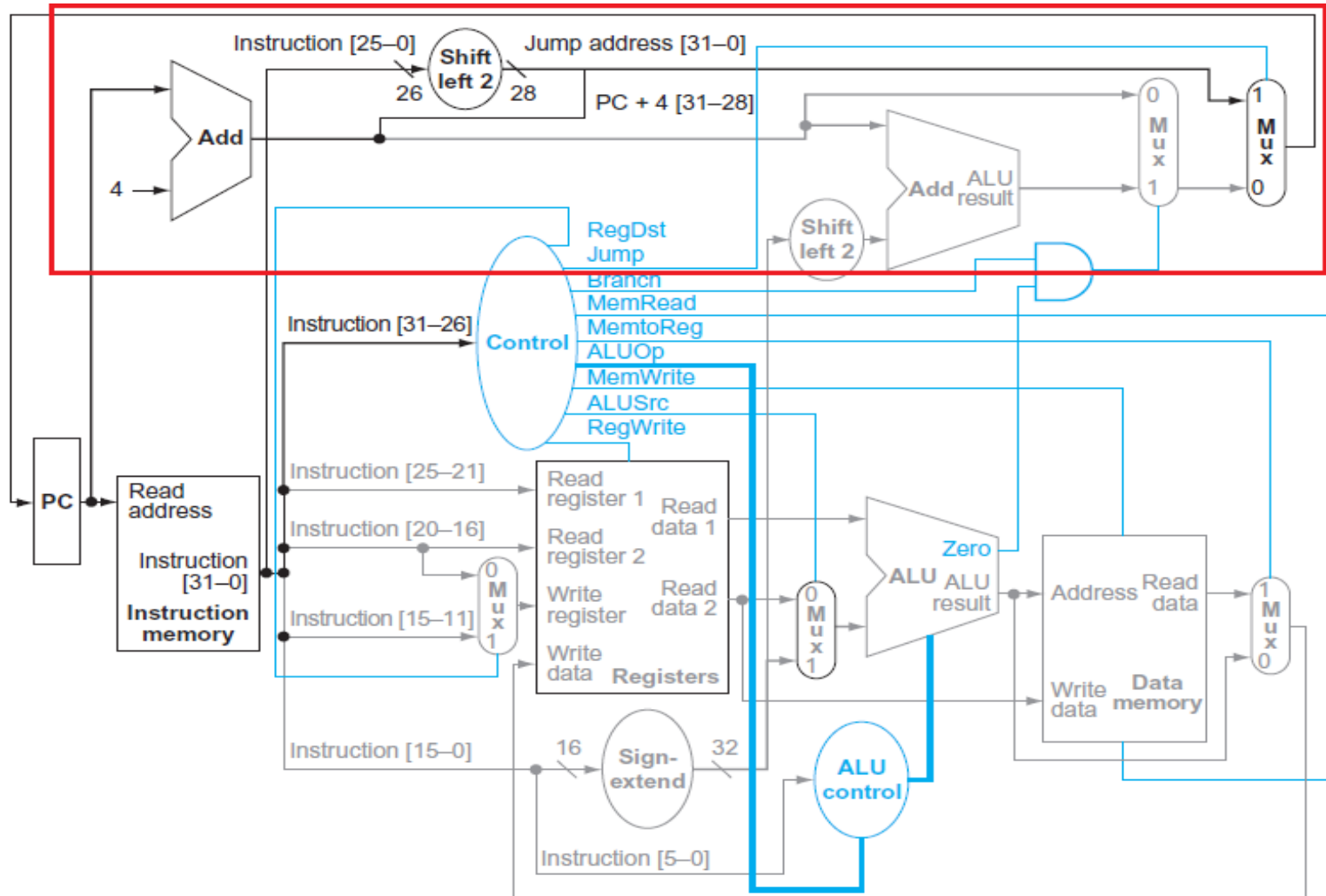
MAIN DECODER SIGNALS

Signal Name	Effect when de-asserted (0)	Effect when asserted (1)
Regwrite	None	The register on the write register input is written with the value on the write data input
Regdst	The register destination number for the write register comes from the rt field (bits 20:16)	The register destination number for the write register comes from the rd field (bits 15:11)
Alusrc	The second ALU operand comes from the second register file output(read data 2)	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
Branch	PC+4 to be placed in PC	Target address of the branch instruction (PC+4 +address×4) is placed in PC
Memwrite	none	Activates write operation to data memory.
memtoReg	The value fed to register write data input comes from the ALU	The value fed to the register write data input comes from data memory
Jump	The output of the branch MUX (target address or PC+4) is placed into PC	The target address of the jump (PC[31-28]:address×4) is placed into PC
ALUop	00 for lw,sw,add, 01 for beq, and 10 for R-type	

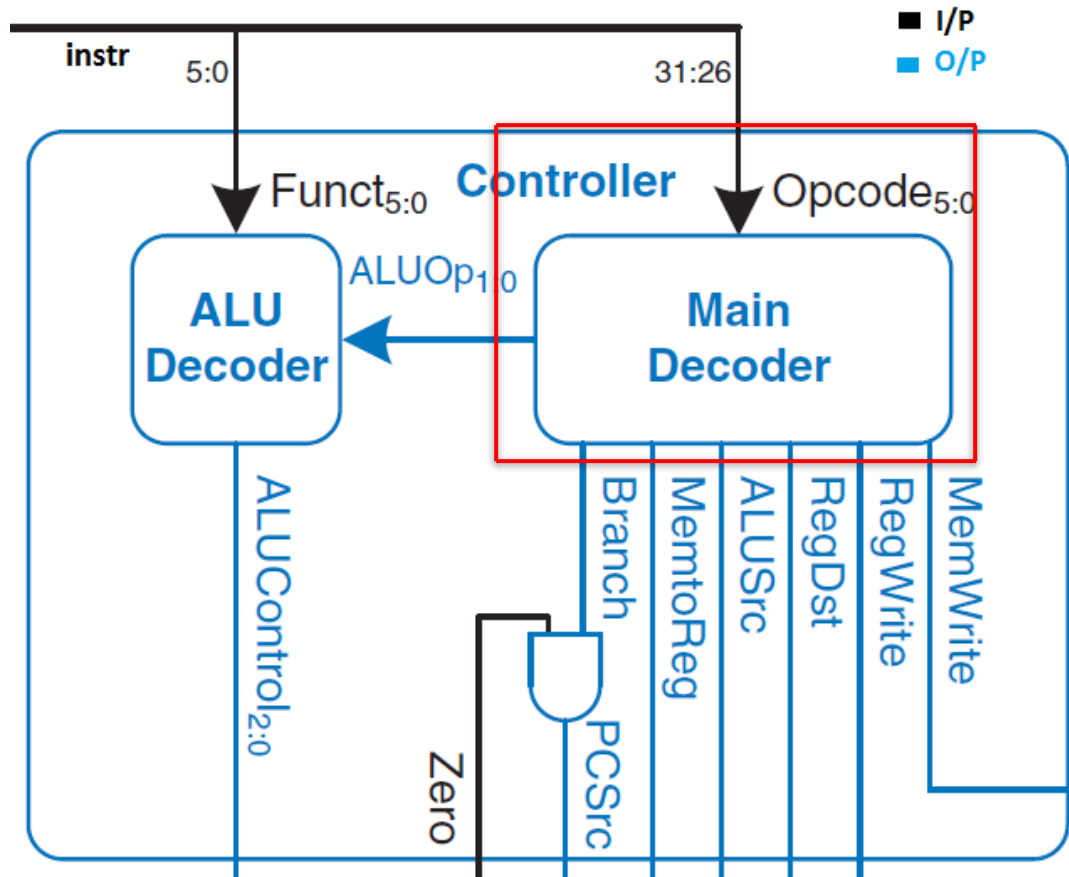
MAIN DECODER SIGNALS



MAIN DECODER SIGNALS (JUMP)



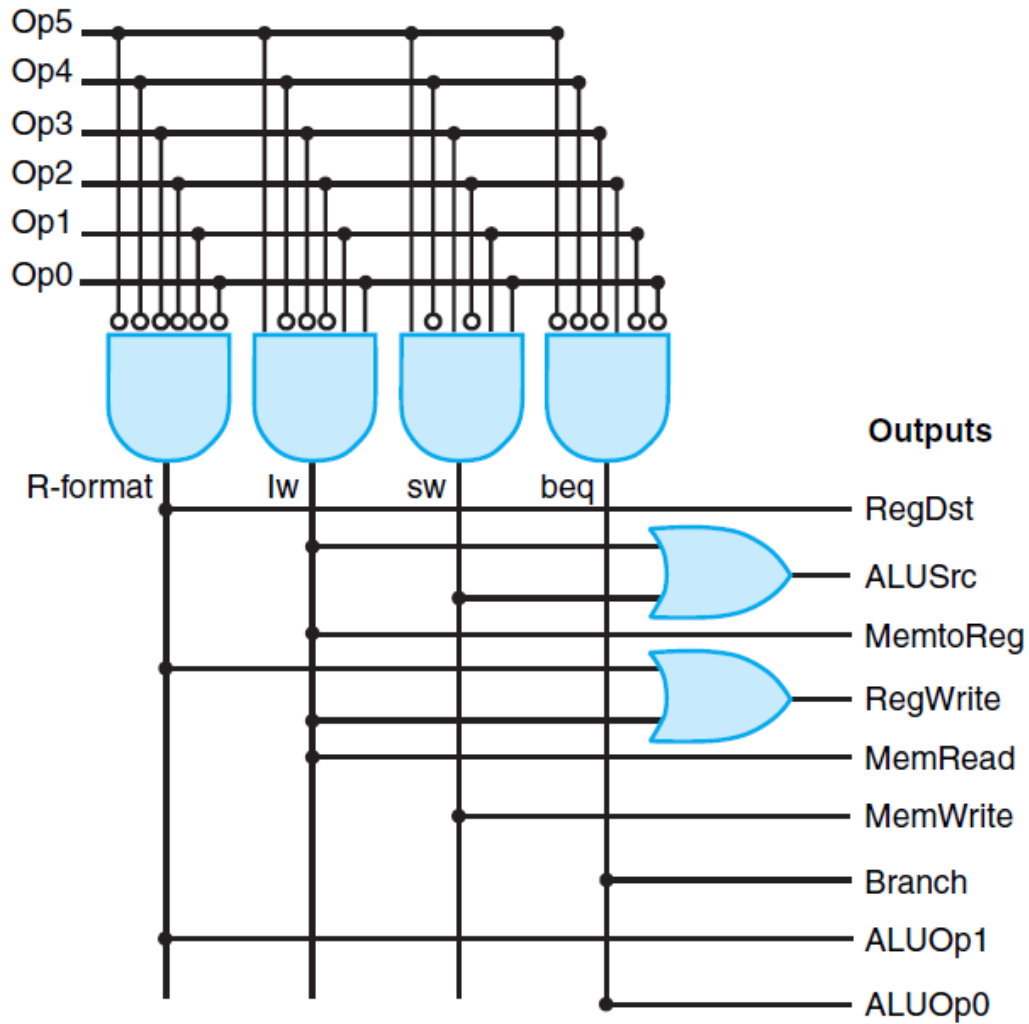
MAIN DECODER BLOCK



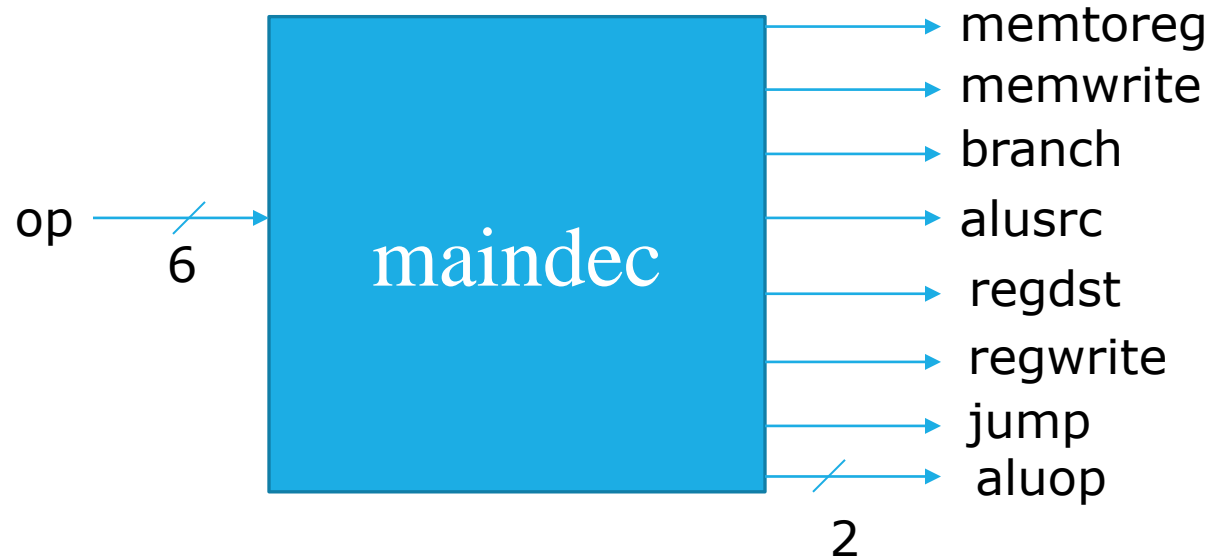
MAIN DECODER FUNCTION TABLE

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

HW DESIGN



HANDS-ON 1: MAIN DECODER ENTITY



```
entity maindec is
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;
```

HANDS-ON1: MAIN DECODER ARCHI

Instruction	op	Regwrite	Regdst	Alusrc	Branch	Mem write	Memtoreg	Jump	Aluop (1)	Aluop (0)
R-type	000000	1	1	0	0	0	0	0	1	0
Lw	100011	1	0	1	0	0	1	0	0	0
Sw	101011	0	0	1	0	1	0	0	0	0
beq	000100	0	0	0	1	0	0	0	0	1
addi	001000	1	0	1	0	0	0	0	0	0
J	000010	0	0	0	0	0	0	1	0	0

Behavior?

1) Separate case statement for each output? Or

An additional signal to hold the outputs

2) Individual assignment with the index bit in the signal? Or

Concatenated assignment

(regwrite, regdst, alusrc, branch, memwrite, memtoreg, jump, aluop(1), aluop(0)) <= signalname;

HANDS-ON1: MAIN DECODER ARCHI

```
architecture maindecLogic of maindec is
signal controls: STD_LOGIC_VECTOR(8 downto 0);
begin

process(op)
begin
case op is
when "000000" => controls <= "110000010"; -- RTYPE
when "100011" => controls <= "101001000"; -- LW
when "101011" => controls <= "001010000"; -- SW
when "000100" => controls <= "000100001"; -- BEQ
when "001000" => controls <= "101000000"; -- ADDI
when "000010" => controls <= "000000100"; -- J
when others => controls <= "-----"; -- illegal op
end case;
end process;

(regwrite, regdst, alusrc, branch, memwrite, memtoreg, jump,
aluop(1), aluop(0)) <= controls;
end;
```

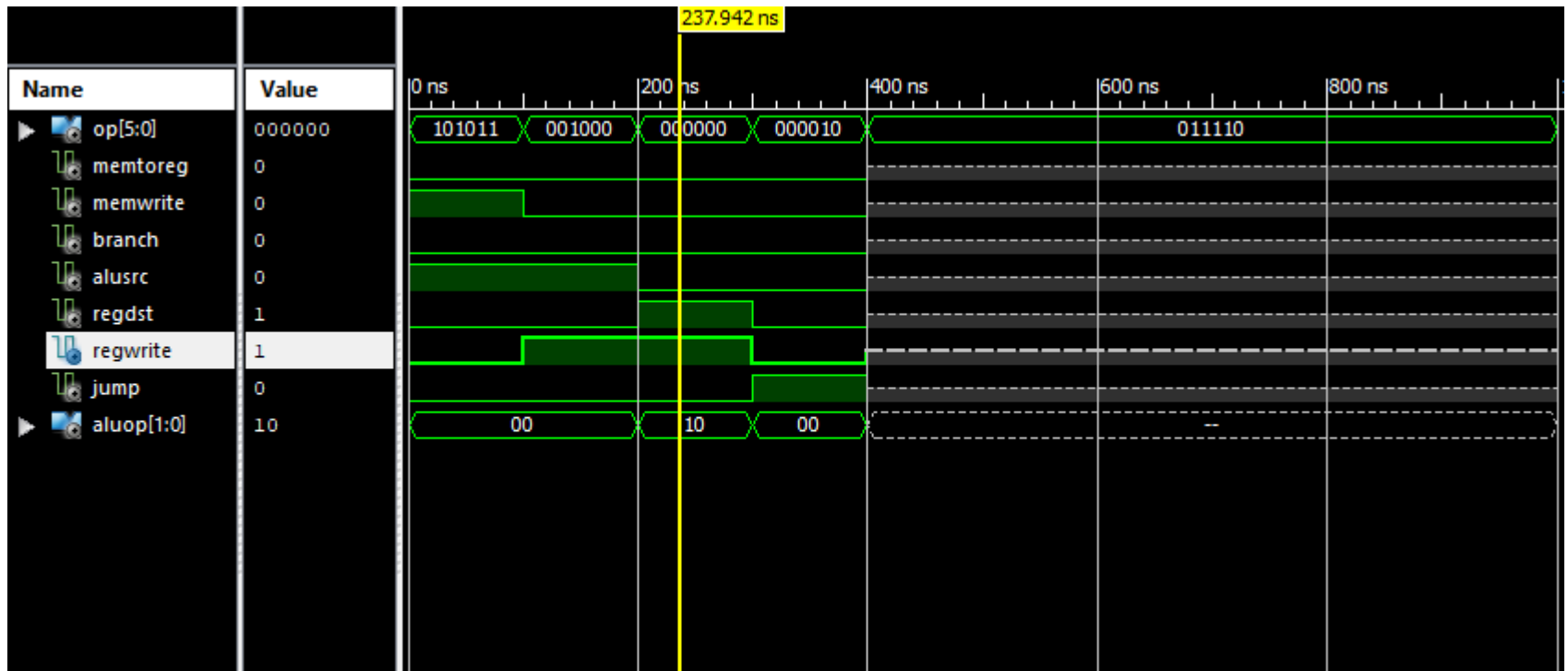
HANDS-ON1: MAIN DECODER TEST CASE

```
-- Stimulus process
stim_proc: process
begin
    -- hold reset state for 100 ns.
    op <= "101011";
    wait for 100 ns;
    op <= "001000";
    wait for 100 ns;
    op <= "000000";
    wait for 100 ns;
    op <= "000010";
    wait for 100 ns;
    op <= "011110"; ← illegal op

    -- insert stimulus here

    wait;
end process;
```

HANDS-ON1: MAIN DECODER SIM. RESULT



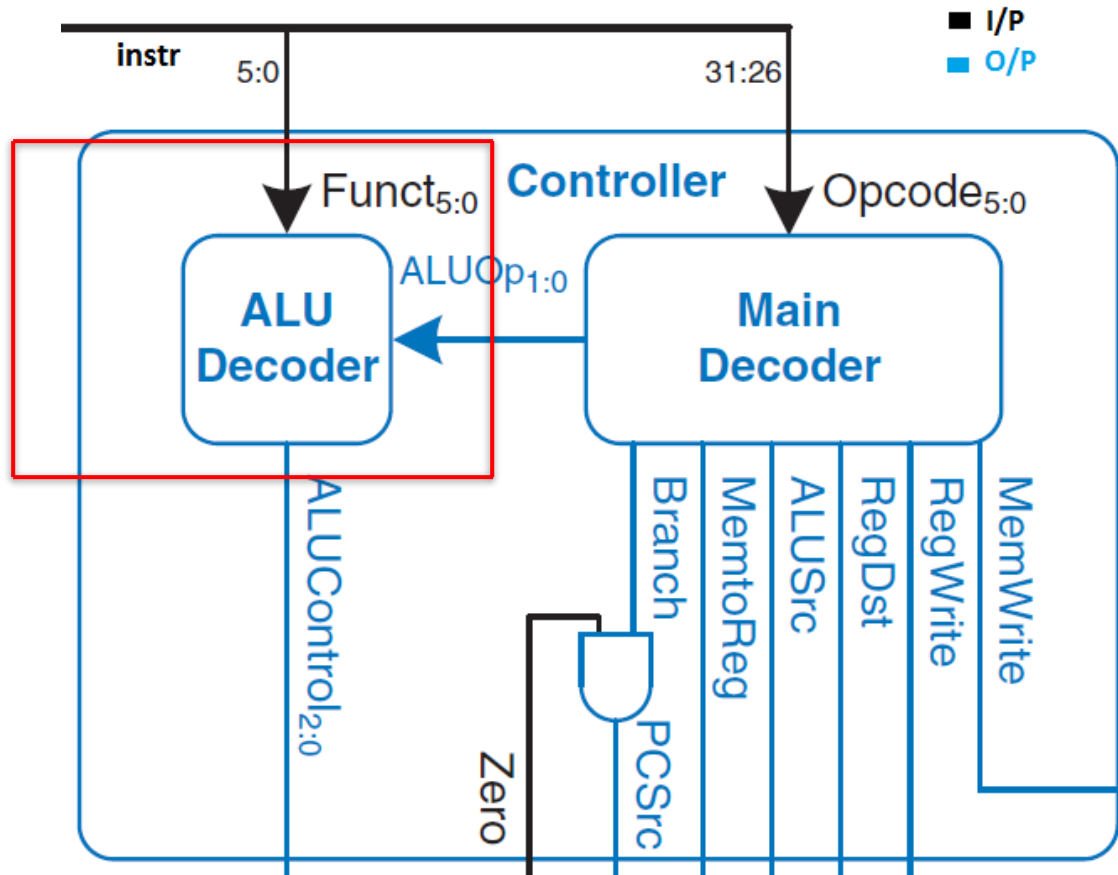


2) ALU Decoder

ALU DECODER/ALU CONTROL

- Decides which operation will be performed by the ALU.
- Input1: *ALUOp*, which is a 2-bit control signal indicating a 00 (add for loads and stores), a 01 (subtract for branches), and a 10 (use the *funct* field).
- Input2: *funct* field. Remember that for R-type instructions, the *opcode* field is always zero and the *funct* field is used to determine the type of operation to perform.
- The output of the ALU control unit is a 3-bit field that is fed into the ALU to select the operation to be performed.

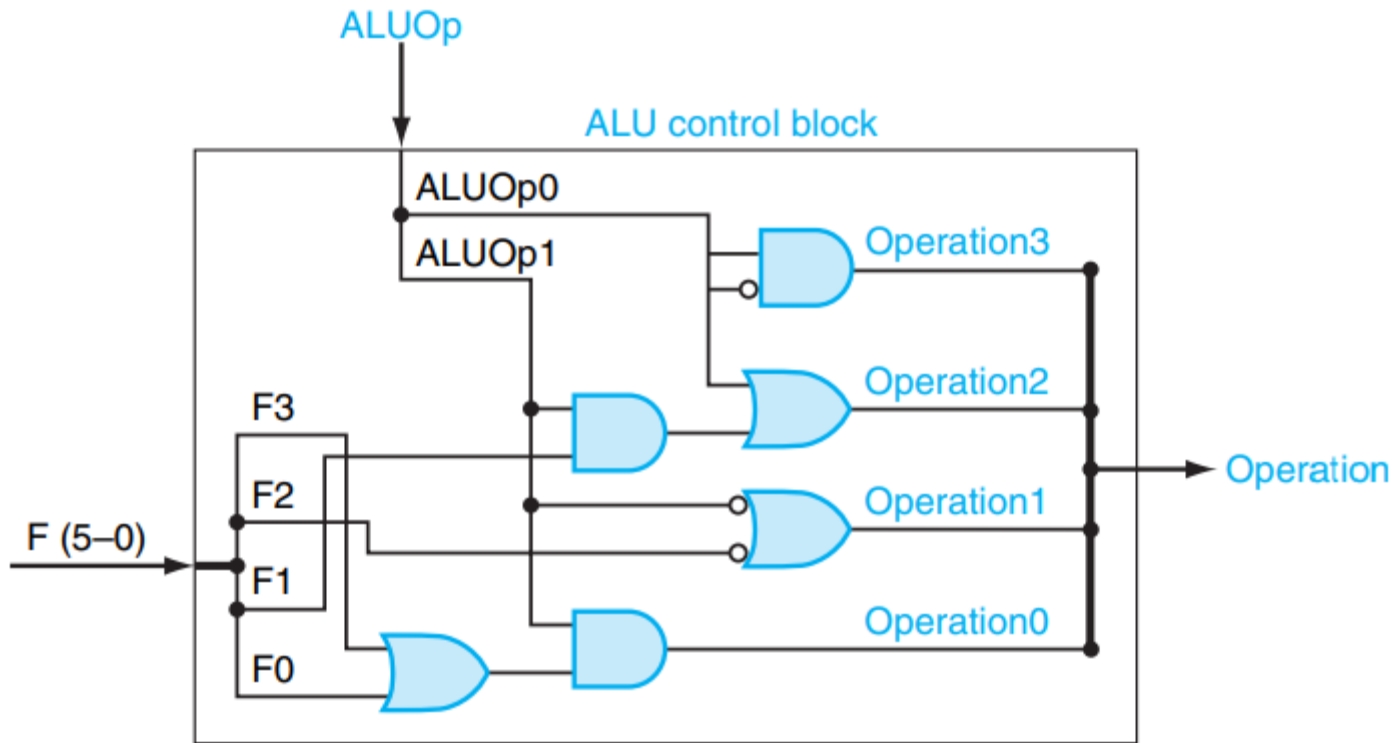
ALU DECODER BLOCK



ALU DECODER FUNCTION TABLE

Instruction opcode	ALUOp	Instruction operation	Funcnt field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	010
SW	00	store word	XXXXXX	add	010
Branch equal	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set on less than	101010	set on less than	111

HW DESIGN



HANDS-ON2: ALU DECODER ENTITY



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- ALU control decoder

entity aludec is
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;
```


HANDS-ON2: ALU DECODER ARCHI.

Instruction	Aluop	funct	alucontrol
w,sw,addi	00		010
Beq	01		110
R-type	10	100000	010
		100010	110
		100100	000
		100101	001
		101010	111

Two nested check cases: 1) Check **aluop** when (lw,sw,addi), or beq
2) Otherwise, check **funct**

HANDS-ON2 : ALU DECODER ARCHI.

```
architecture behave of aludec is
begin
process(aluop, funct)
begin
  case aluop is
    when "00" => alucontrol <= "010"; -- add (for lw/sw/addi)
    when "01" => alucontrol <= "110"; -- sub (for beq)
    when others =>
      case funct is -- R-type instructions
        when "100000" => alucontrol <= "010"; -- add
        when "100010" => alucontrol <= "110"; -- sub
        when "100100" => alucontrol <= "000"; -- and
        when "100101" => alucontrol <= "001"; -- or
        when "101010" => alucontrol <= "111"; -- slt
        when others => alucontrol <= "---"; -- ???
      end case;
    end case;
end process;
end;
```



Bring your package files in addition to maindec.vhd and aludec.vhd with you next labs.



Thanks