Combinational Logic Circuits

Learning Outcomes

At the end of this lab, you should be able to:

1. differentiate combinational and sequential elements in a processor;
2. implement combinational logic circuit elements in VHDL;

Contents

1 Resources

- Video: https://youtu.be/H2TmmSFds-M.
- Source Codes: https://git.io/JU3al

2 Discussion

The discussion in this handout aims only to provide an outline of what is in the video lecture. It is recommended that you watch the video in its entirety.

The computer can only execute machine language that it understands. This language is part of its architectural design or the Istruction Set Architecture (ISA). Machine language is nothing more than an encoding of bits. The computer can will do nothing unless instructions are given to it. These instructions will come from programmers. However, it is very difficult for programmers to use machine language directly. Assembly language is instead used which is more comprehensible for humans. The assembler translates the assembly language program to machine language.

2.1 Definitions

Here are some terms worth knowing and remembering in this lab.

- *Datapath* - The component of the processor that performs arithmetic operations

- *Control* - The component of the processor that commands the datapath, memory, and I/O devices according to the instructions of the program
- *Combinational elements* - Logic elements in the datapath that operate on data values
- *State elements* - contains state, has some internal storage
- *Edge-triggered clocking* - that any values stored in a sequential logic element are updated only on a clock edge, which is a quick transition from low to high or high to low
- *Clock cycle length* - time to propagate the signal from state element 1, through the combinational element, and state element 2

2.2 Single-Cycle Datapath and Control

A processor is composed of the datapath and control unit. Figure 1 shows an example of a single-cycle datapath and control. The main functional units are the following:

- *Program Counter (PC)* - holds the address of the next instruction to be executed
- *Instruction Memory (IM)*-holds the instructions/machine language
- *Register File (RF)*- fast storage
- *Arithmetic and Logic Unit (ALU)*- performs computations
- *Data Memory (DM)*- holds data used by programs
- *Control Unit (CU)*- drives the execution

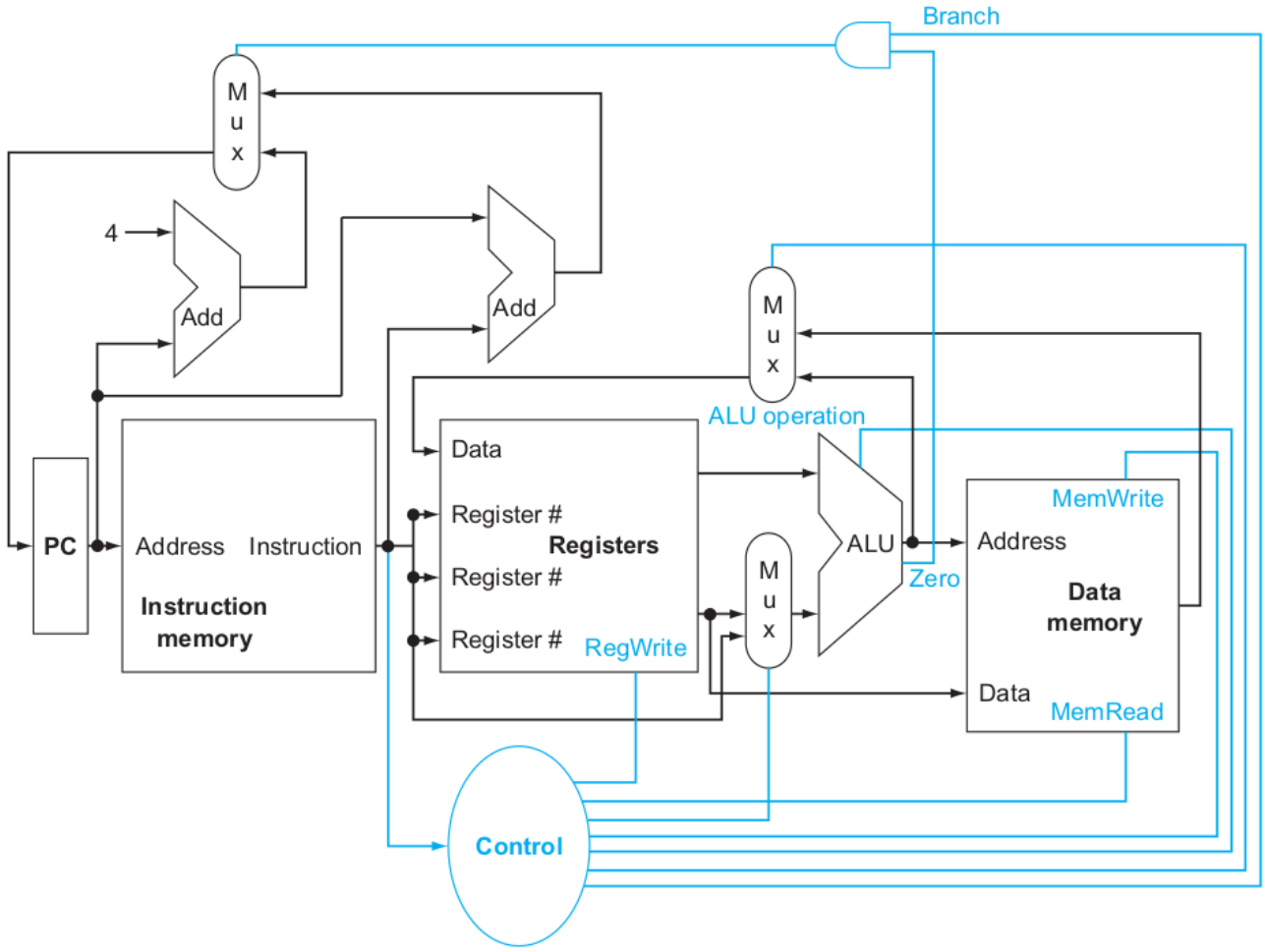


Figure 1: Single-Cycle Datapath and Control.

The blue lines indicate control signals most of which originate from control based on the instruction being executed. You can think of the Control Unit as the 'brain' and the rest as the body parts.

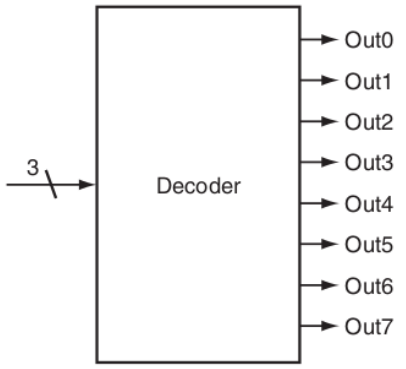
Each of the above functional components are implemented using combinational and sequential logic circuits as building blocks.

Instruction execution follows the following cycle or states:

- *Instruction Fetch (IF)* - instruction is fetch from IM at address specified by PC
- *Instruction Decode (ID)* - current instruction is examined for opcode and operands
- *Execute (EX)* - perform operations based on opcode
- *Memory (MEM)* - need to store data to DM
- *Writeback (WB)* - result will be used stored in register or used in the next instruction

2.3 Decoders

A decoder accepts an n-bit input and produces 2^n outputs but only one output is asserted. Figure 2 shows an example 3-to-8 decoder. The VHDL implementation is also given below.



Inputs			Outputs							
12	11	10	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Figure 2: 3-to-8 decoder.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------
ENTITY decoder_3to8 IS
   PORT (
      I2, I1, I0: IN STD_LOGIC;
      Out7, Out6, Out5, Out4, Out3, Out2,
         Out1, Out0: OUT STD_LOGIC
   );
END decoder_3to8;
---------------------------------
ARCHITECTURE pure_logic OF decoder_3to8 IS
BEGIN
   Out0 <= NOT I2 AND NOT I1 and NOT I0;
   Out1 <= NOT I2 AND NOT I1 and I0;
   Out2 <= NOT I2 AND I1 and NOT I0;
   Out3 <= NOT I2 AND I1 and I0;
   Out4 <= I2 AND NOT I1 and NOT I0;
   Out5 <= I2 AND NOT I1 and I0;
   Out6 <= I2 AND I1 and NOT I0;
   Out7 <= I2 AND I1 and I0;
END pure_logic;
```

2.4 Multiplexers/Selectors

A multiplexer select which input will be allowed to pass through as output. Figure 3 show a mux with the selecter S and two inputs A and B. It is implemented using two AND gates and one OR gate. To support multiple bits, this single bit multiplexer can be cascaded as an array as shown in Figure 4.

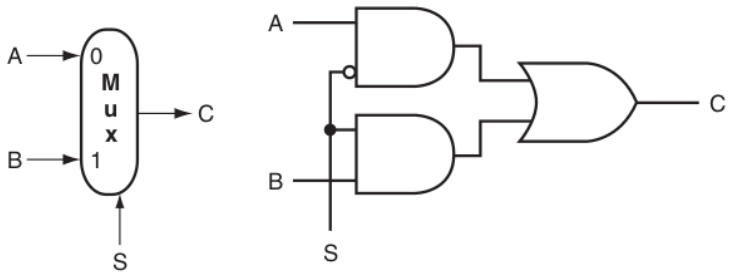


Figure 3: 2-to-1 Multiplexer.

The VHDL code for the 2-to-1 multiplexer is shown below.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------
ENTITY mux_2to1 IS
   PORT (A, B, S: IN STD_LOGIC;
   C: OUT STD_LOGIC);
END mux_2to1;
---------------------------------

ARCHITECTURE pure_logic OF mux_2to1 IS
BEGIN
--- C <= (A AND NOT S) OR (B AND S);
   C <= A WHEN S='0' ELSE B;
END pure_logic;
```

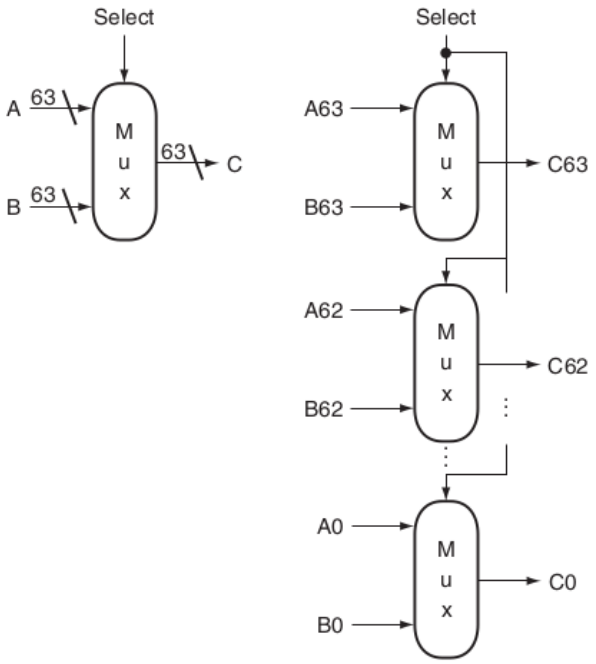


Figure 4: An array of 1-bit multiplexers to select from two 64-bit inputs.

2.5 Designing a 1-bit ALU

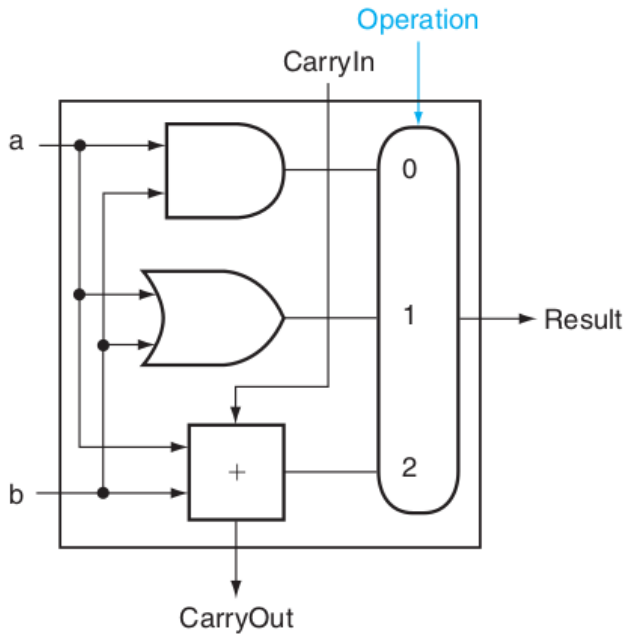


Figure 5: A 1-bit ALU design.

We can design a 1-bit ALU that can support the *AND*, *OR*, and *ADD* operations. The diagram is shown in Figure 5. The full discussion of its design and implementation is discussed in the video. The complicated component in this ALU is the Full Adder represented by the rectangle with the plus('+') sign. The VHDL

code for the 1-bit ALU is shown below.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------
ENTITY alu IS
   PORT (a, b, CarryIn : IN STD_LOGIC;
   Operation: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
   Result, CarryOut: OUT STD_LOGIC);
END alu;
---------------------------------
ARCHITECTURE behavioral OF alu IS
   COMPONENT mux_4to1 IS
      PORT (A, B, C, D, S0, S1: IN STD_LOGIC;
      E: OUT STD_LOGIC);
   END COMPONENT;
   COMPONENT and_gate IS
      PORT (A, B: IN STD_LOGIC; C: OUT STD_LOGIC);
   END COMPONENT;
   COMPONENT or_gate IS
      PORT (A, B: IN STD_LOGIC; C: OUT STD_LOGIC);
   END COMPONENT;
   COMPONENT full_adder is
      PORT ( a : IN STD_LOGIC; b : IN STD_LOGIC;
         CarryIn : IN STD_LOGIC;Sum : OUT STD_LOGIC;
         CarryOut : OUT STD_LOGIC);
   END COMPONENT;
   SIGNAL u, v, w, x: STD_LOGIC;
BEGIN
   u1: and_gate port map(a, b, u);
   u2: or_gate port map(a, b, v);
   u3: full_adder port map(a, b, CarryIn, w, CarryOut);
   u4: mux_4to1 port map(u, v, w, x, Operation(0),
               Operation(1), Result);
END behavioral;
```

3 Summary

We discussed some of the combinational elements that are useful in the design of a processor. We also showed the design of a 1-bit ALU that supports a limited set of operations.

4 Learning Activities

Download the source codes for this lab then try experimenting by adding more test cases in the testbenches. Submit a PDF document that shows screenshots of your modifications and runs.

5 Deliverable

Your final deliverable for this lab is described in the accompanying exercise handout.

6 Further Reading

- https://www.electronics-tutorials.ws/logic/logic_1.html

References

[1] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface, ARM Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, arm edition, 2017.