MONASH University

FIT1047 Applied Session Week 2

ASSEMBLY LANGUAGE PROGRAMS (USING MARIE) AND COMBINATIONAL LOGIC CIRCUITS

OBJECTIVES

- The purpose of this applied session is getting to know MARIE assembly code programs and Combinational logic circuits.

INSTRUCTIONS

- Use MARIE simulator to write assembly language programs, and use logisim to draw combinational logic circuits (adders).
- You may work in a small group.

Activity 1: MARIE Basics

Use the MARIE simulator to try out the program given below and answer the following questions. Copy the codes in the simulator input window, assemble your program to check for any errors, and then run or step through the lines of your codes to observe the changes taking place in CPU registers and various memory locations.

Line No.	Code	Comments
1	Load 004	/ Load value from address 004 into AC
2	Add 004	/ Add value from address 004 to value in AC
3	Store 004	/ Store AC into address 004
4	Halt	/ Stop execution
5	DEC 42	/ This is address 004, initialise with value 42

(a) Can you locate your program In MARIE memory? What are the memory addresses of the first and the last line of your program?

(b) In what format is your program stored in MARIE memory? How is your data “DEC 42” stored?

(c) What is the first and the last address of MARIE memory? Can you determine the size of MARIE memory?
(d) Using a variable to refer to "DEC 42", we can modify the line no. 5 as follows:

5	num, DEC 42	/num is a variable and num = 42

Rewrite the program using the variable "num" instead of using the memory address to refer to the data "DEC 42."

Sample Solution:

(a) memory addresses of the first line of your program: 000_{16}
memory addresses of the last line of your program: 004_{16}
(b) In machine code (shown in hexadecimal format). Converted to hexadecimal format.
(c) 000_{16} - FFF_{16} , 4K Word, (1 Word = 16 bits)
(d) Load num
Add num
Store num
Halt
num, DEC 42

Activity 2: Data input, output and storage in MARIE

Write a MARIE program to input any data from the keyboard and display in the MARIE simulator output window. Assemble your program to check for any errors, and then run or step through the lines of your codes and answer the following questions.

(a) How many different input and output modes of data are available in MARIE?
(b) In which register of the CPU is your input data stored?
(c) Add the following instruction "ORG 020" at the beginning of your program, and assemble your code again. Now, you will find that MARIE has relocated your machine code to a new address. What is this address? Is it the same as the contents of the CPU register PC? Why? Step through your program to see the changes happening in the PC.
(d) Add two variables to your program as follows:

```
numIn,      DEC 0
One,        DEC 1
```

Note: add these lines at the end of your program.

Modify your program to store the input data in "numIn" and add '1' to it before displaying it in the output. Please input decimal numbers (from the keyboard) for this task.

Sample Solution:

(a) Four: (decimal, hex, binary, unicode/ASCII) of data in MARIE
(b) AC
(c) 020, PC = 020, PC changes to the next number as the program steps through.

(d)
```
Input
Store  numIn

Load   numIn
add    One
Store  numIn

Load   numIn
Output
Halt

numIn,      DEC 0
One,        DEC 1
```

Activity 3: Branch Instructions in MARIE and use of labels

Extend your program from Activity 2, by adding labels, such as:

```
begin,      Load   num1
            Add    One
            ……..
```

(a) Further modify your program by adding jump instructions to repeat the data input, modify and display activities endlessly.
(b) Using "skipcond" instruction, exit the program when the user inputs data '0'.

Sample Solution:

(a)
```
begin, Input
       Store  numIn

       Load   numIn
       add    One
       Store  numIn

       Load   numIn
       Output
       Jump   begin
       Halt

numIn,      DEC 0
One,        DEC 1
```

(b)
```
begin, Input
       skipcond 400
       jump cont
       jump end
cont,  Store  numIn

       Load   numIn
       add    One
```

```
        Store  numIn

        Load   numIn
        Output
        Jump   begin
end,    Halt

numIn,      DEC 0
One,        DEC 1
```

Activity 4: Branch Instructions in MARIE and use of labels

A conditional statement allows the flow of a program to depend on data. In high-level programming languages, these are typically achieved using (i) if-then-else or (ii) while-do statements such as the following (in pseudocode):

(a) if-then-else

```
if (X > Y)
        X = X + X
else
        Y = Y + Y
```

(b) while-do

```
X = 1
while (X < 10) do
        X = X + 1
endwhile
```

Write MARIE assembly language programs implementing the conditional statements.

Sample Solution:

```
(a) begin, Input
        Store  numX
        Input
        Store  numY

        Load   numX
        Subt   numY
        skipcond 800
        jump   else
        Load   numX
        add    numX
        Store  numX
        Jump   end


else,   Load   numY
        add    numY
        Store  numY
end,    Halt
```

```
        numX,       DEC 0
        numY,       DEC 0

(b)     Load    One
        Store   numX

        begin, Load   numX
        Subt    numY
        Skipcond 000
        Jump    end
        Load    numX
        Output
        Add     One
        Store   numX
        Jump    begin

        end,    Halt

        numX,       DEC 0
        numY,       DEC 10
        One,        DEC 1
```

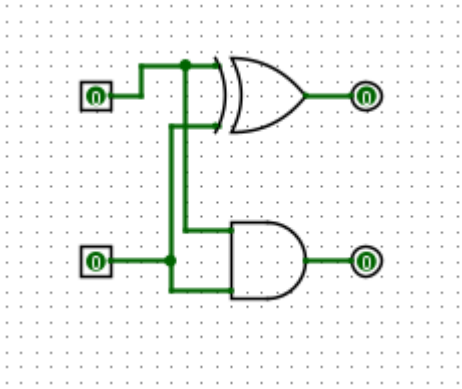
Activity 5: Build a Half adder, a Full Adder and a 4-bit Ripple Carry Adder (RCA) in Logisim

Half-adders, full-adders and ripple-carry adders are described in the lessons. You will now use this knowledge to build a 4-bit adder/subtractor in Logisim.

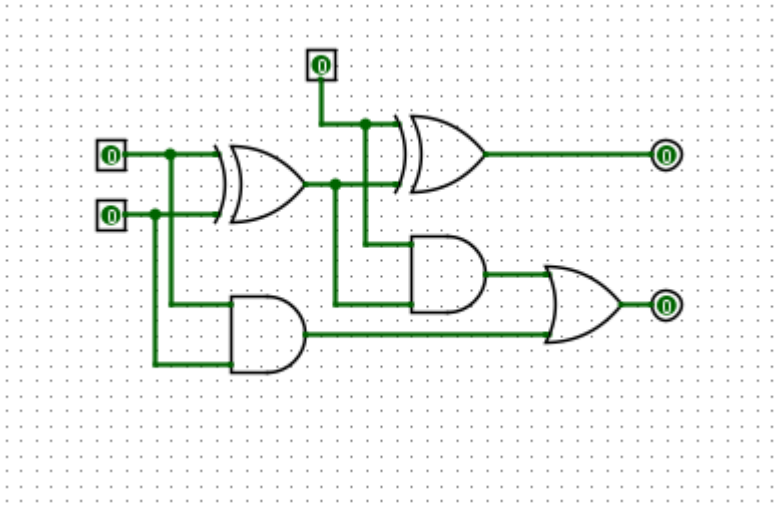
(a) Draw and simulate a simple half-adder and full-adder in Logisim. Discuss the truth tables for these two circuits and how they correspond to the circuit diagrams.
(b) Once your full adder circuit is working, combine four full adders into a 4-bit ripple carry adder. It should have eight inputs (two 4-bit numbers A and B) and five outputs (the 4-bit result and the final carry bit). The carry-in for the first full adder has to be fixed to 0 (use the Ground symbol in the Wiring folder). Test your circuit by adding the following numbers:(i) 3 + 1, (ii) 2 + 3
(c) Now let's assume that we are using the 2's complement representation for negative numbers. What is the range of numbers this adder will operate without any overflow? Simulate a few computations, such as 5 + (-7) and (-3) + (-4).

Sample Solution:

A half-Adder



A Full-Adder



A Ripple Carry Adder (RCA)