Lab-Week-4

Friday, August 28, 2020 11:40 AM

FIT1047-Lab-Week-...

FIT1047 Tutorial 4 & 5

Topics
- Memory addressing
- Programming MARIE assembly code
- Boolean Algebra, Digital Circuit logic, K-Map &
- **Assignment 1 Discussion**

Instructions

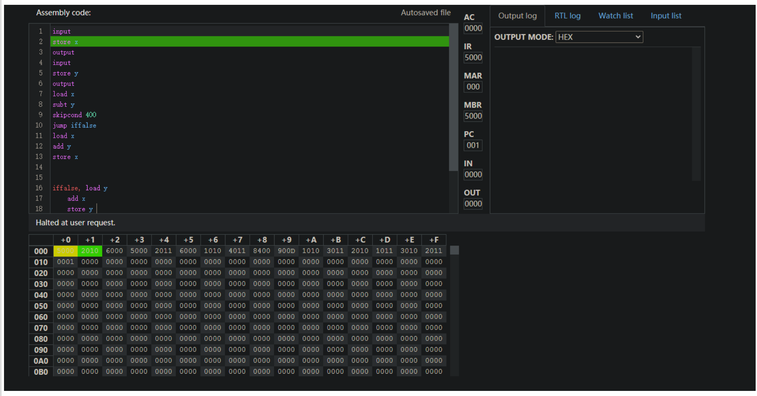
The tasks are supposed to be done individually. You will need the *MARIE* simulator and logisim evolution simulator for this lab tasks.

Task 1: Conditionals

A **conditional statement** allows the flow of a program to depend on data. In a high-level programming language, this is typically achieved using `if-then-else` statements such as the following (this is Python syntax):

```
if X == Y:
   X = X + Y
 else:
   Y = Y + x
```

Implement this piece of code using MARIE assembly. Make use of labels!



	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	0000	2010	6000	5000	2011	6000	1010	4011	8400	900D	1010	3011	2010	1011	3010	2011
010	0001	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
040	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
050	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
060	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
070	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
080	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
090	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0A0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0B0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Task 2: Subroutines

An important concept in programming is that of a **procedure**, **function**, or **subroutine**, a piece of code that has a fixed purpose and that needs to be executed over and over again.

As a simple example, you saw last week that MARIE doesn't have an instruction for multiplication. Now imagine a large program: you will probably need the multiplication routine in hundreds of different places!

Of course, you could just copy and paste the piece of code into the place where you need it. **Discuss why that's a bad idea!**

The subroutine can quote while you need to use it, if not use he subroutines the code will increase lots of lines, it will confused other if writing a large program. Meanwhile, the complex code will cause bug and error and it is hard to repair because it has to be rewritten a large part.

Most ISA (Instruction Set Architectures) have some level of support for writing subroutines. In MARIE, there's the `JnS X` instruction ("jump and store"): It stores the value of the `PC` at memory address `X` and then jumps to address `X+1`. The value stored at `X` is called the *return address*, i.e., the address where execution should continue once the subroutine has finished its job.

To **return** from a subroutine, the last instruction in the subroutine should be a jump back to the return address. This can be achieved using the `JumpI X` instruction: it jumps to the address stored at address `X` (compare that to `Jump X` which jumps to the address `X`).

1. Explain how you can use `JnS X` and `JumpI X` to implement subroutines. In particular, think about why `JnS` stores the value of the `PC`, not `PC+1`.
2. Implement a simple subroutine that computes $2 \times X$, i.e., it takes the value in a memory location `X`, doubles it, and returns to where the original program left off.
3. Take the multiplication code from last week and convert it into a subroutine.
4. Write the following code segment in MARIE assembly language:(Optional task)

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

- Jus can seem like the jump to the funtion, but Jumpi is "return" of the funtion

```
Input
store x
output
Jnz double
Jnz double
load x
output
Halt

double, Dec 100
    Load x
    add x
    store x
    JumpI double

x, Dec 0
```

```
Input
store A
output
Input
store B
output
Jnz multiply
load C
output
Halt


multiply, Dec 200
CLEAR
STORE C
Loop,   Load A
    Skipcond 800
    JumpI multiply
multiply, Dec 200
CLEAR
STORE C
Loop,   Load A
    Skipcond 800
    JumpI multiply
    Load C
    Add B
    Store C
    Load A
    Subt One
    Store A
    Jump Loop


A, Dec 0
B, Dec 0
```

Task 3: Boolean Algebra

Given the function $F(X,Y,Z) = Y(\overline{X}Z + \overline{Z}) + \overline{Y}(XZ + \overline{Z})$

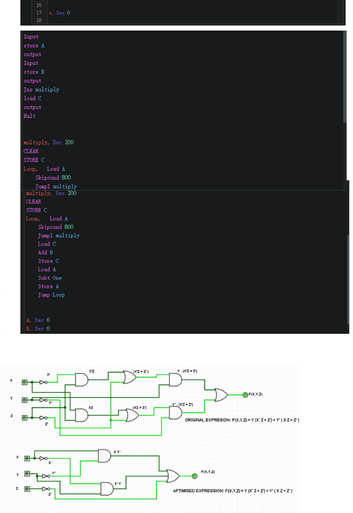
3.a List the truth table for F.

3.b Draw the logic diagram using the original Boolean expression.

3.c Simplify the expression using a Karnaugh map.

3.d List the truth table for your answer in part 3.c.

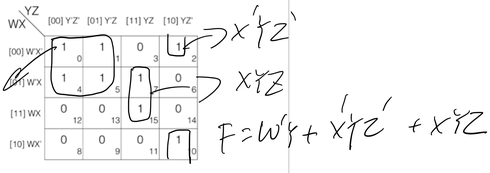
3.e Draw the logic diagram for the simplified expression in part 3.c.



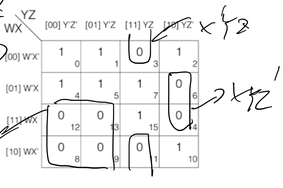
Digital Logic

Task-4: Exercise 1: Learn to reduce **Sum of Products (SOP)** using Karnaugh Map.

Reduction rules for SOP using K-map



Task-4: Exercise 2: Learn to reduce Product of Sum (POS) using Karnaugh Map.



Task-4: Exercise 3:
Given the following truth table:

A	B	C	Output (F)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

a) Write a Boolean expression in the Sum-of-Products standard format that represents the logic function performed by this circuit.

b) From the truth table: convert the logic function into the Product-of-Sums standard format