FIT1047 Tutorial 4

Topics

- Register Transfer Language
- Memory addressing
- Programming MARIE assembly code

Instructions

The tasks are supposed to be done in groups of two or three students. You will need the *MARIE* simulator for this assignment.

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!

Task 2: Extending the instruction set

We are going to extend the MARIE instruction set by the following instruction:

- `SkipIfGreater X` skips the next instruction if the value at memory address `X` is greater than the value in `AC`.

Give the RTL steps needed to implement the instruction.

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!**

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. Implement a subroutine `Exp` that computes x^y for two numbers x and y. This works almost the same way as multiplication, and since you already have a multiplication subroutine, you can call it to implement `Exp`.
5. Bonus task: what if a subroutine wants to call itself? This is called recursion. Discuss the problems with this idea, and what you could do in MARIE to enable recursive calls.