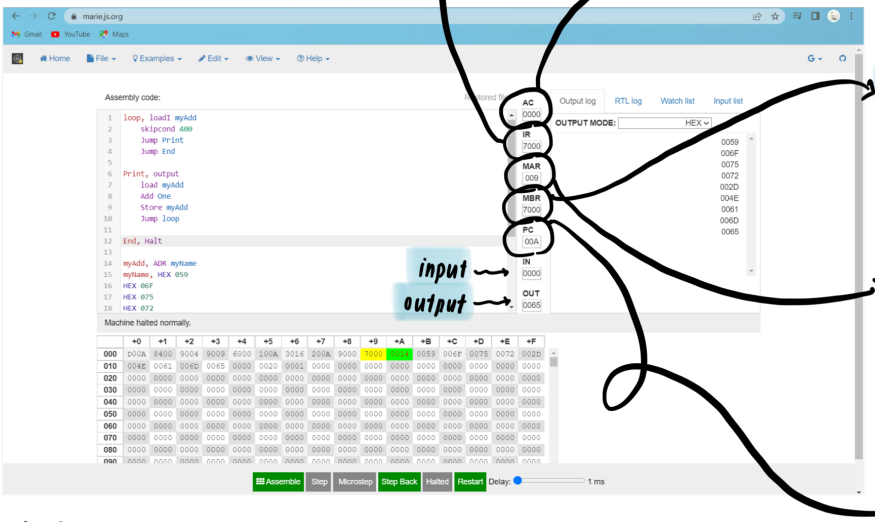
Marie programming :

- words = 16 bits wide
- 16 different instructions
- 1 instruction = 16 bits wide
  - ↳ 4-bit opcode
  - ↳ 12-bit address
- 1 general-purpose register
- Example :



Instruction Register (IR):
contains currently executed instruction

Accumulator (AC) : general-purpose register

(MBR) memory buffer register:
holds the data read from or written to memory

(MAR) memory Address Register:
holds memory address of a word that needs to be read from or written to memory

Program Counter (PC):
contains address of the next instruction

- leftmost 4-bits represents opcode
  - ↳ tell what kind of instruction it is
- remaining 12-bits represents address of a memory location the instruction will work with
  - Eg. 0001 000000000100
    - 0001 → opcode
    - 000000000100 → address to load from
      - ↳ instruction : load data from memory

* 000000000100 = binary number 4

CPU execute instruction : load the current value stored at memory address 4 & put into AC register inside CPU

- Assembly code = mnemonic opcode (easily remembered & recognised)
- Assembler = tool to translate assembly code program → real machine code
  - ↳ simple compiler

Here's an overview of part of the MARIE instruction set. The `X` in the instructions stands for the address part.

Opcode	Mnemonic	Explanation
`0001`	`Load X`	Load value from location `X` into `AC`
`0010`	`Store X`	Store value from `AC` into location `X`
`0011`	`Add X`	Add value stored at location `X` to current value in `AC`
`0100`	`Subt X`	Subtract value stored at location `X` from current value in `AC`
`0101`	`Input`	Read user input into `AC`
`0110`	`Output`	Output current value of `AC`
`0111`	`Halt`	Stop execution
`1010`	`Clear`	Set `AC` to `0`

Notice how the instructions use the `AC` register as temporary storage.

* pseudocode
  - ↳ program written in a "programming language" that doesn't exists
  - ↳ program planning
  - ↳ write general structure

In this case, the program consists of four steps that are supposed to be executed in this order:

```
Load number from memory address 4 into AC register
Add number from memory address 5 to AC register
Store result from AC register into memory address 6
Stop execution
```

Using the table of mnemonics above, it is now easy to translate this into MARIE assembly code:

```
Load 4
Add 5
Store 6
Halt
```

So in order to e.g. output a value that is stored in memory, we first have to Load it into the AC, and then use the Output instruction.

In a real computer, similar instructions would be used to read data from a hard disk or the network, or to send data to a screen, a sound device or a motor of a robot.

* MARIE : CPU starts executing instructions in an orderly fashion until it reaches HALT.

MARIE Instructions

Type	Instruction	Summary
Arithmetic	Add X	Adds value in AC at address X into AC, AC ← AC + X
	Subt X	Subtracts value in AC at address X into AC, AC ← AC - X
	AddI X	Add Indirect: Use the value at X as the actual address of the data operand to add to AC
	Clear	AC ← 0
Data Transfer	Load X	Loads Contents of Address X into AC
	Store X	Stores Contents of AC into Address X
I/O	Input	Request user to input a value
	Output	Prints value from AC
Branch	Jump X	Jumps to Address X
	Skipcond (C)	Skips the next instruction based on C: if (C) = - 000: Skips if AC < 0 - 400: Skips if AC = 0 - 800: Skips if AC > 0
Subroutine	JnS X	Jumps and Store: Stores value of PC at address X then increments PC to X+1
	JumpI X	Uses the value at X as the address to jump to
Indirect Addressing	StoreI	Stores value in AC at the indirect address. e.g. StoreI addresspointer Gets value from addresspointer, stores the AC value into the address
	LoadI	Loads value from indirect address into AC e.g. LoadI addresspointer Gets address value from addresspointer, loads value at the address into AC
	Halt	End the program

MARIE - Indirect Addressing

LoadI X	Loads value stored at address of address X into AC
JnS X	Stores PC at address X and jumps to X+1
JumpI X	Uses value at X as the address to jump to

1. What does the following MARIE program do?

```
        Load X
        Add Y
        Output
        Halt

X,      Dec 2
Y,      Dec 3
```

- It performs the addition between X (value 2) and Y (value 3) then outputs the result.

2. Write a program that subtracts value Y from X, then store it in a new variable and output the variable.

```
        Load X
        Subt Y
        Store Z
        Load Z
        Output
        Halt

X,      Dec 8
Y,      Dec 2
Z,      Dec 0
```

3. Write a program that accepts two 2 number inputs then output the larger number.

Pseudocode:

```
x = input()
y = input()
if x > y:
    print(x)
else:
    print(y)
```

MARIE code:

```
Input           / Get user input into AC
SkipCond 800    / Skip next instruction if AC > 0
Halt            / Halt (if AC not greater than 0!)
Output          / Output AC
Jump 0          / Jump back to beginning of the program
```

```
    //x = input()
    Input
    Store X
    //y = input()
    Input
    Store Y

    //if x > y:   ## x - y > 0
    Load X
    Subt Y
    Skipcond 800 // skip next line if > 0 == if x > y: goto next
    line
    Jump PrintY
    Jump PrintX

    PrintX, Load X
    Output
    Halt

    PrintY, Load Y
    Output
    Halt

    X, Dec 0
    Y, Dec 0
```