Notes: The Marie Simulator

The Accumulator (AC) is the register where calculations are performed.
To add two numbers together,

a) load the first number into the accumulator with a `Load` instruction

b) Add the second number to the accumulator using an `Add` instruction

c) Most of the time, you will want to store the result of a calculation somewhere using a `Store` command, or display the result using the `Output` instruction,

Notes: The Marie Simulator

Instruction Number		Instruction	Notes
Binary	Hexadecimal		
0001	1	Load *X*	Take the value that is stored at address *X* and Load it into the Accumulator (AC)
0010	2	Store *X*	Take the contents of the accumulator, and store it in memory location *X*
0011	3	Add *X*	Take the contents of memory address X and add it to the contents of the accumulator – the result of the calculation is remains in accumulator (AC)
0100	4	Subt *X*	Take the contents of memory address X and subtract it from the contents of the accumulator – the result of the calculation is remains in the accumulator (AC)
0101	5	Input	Accept a value from the keyboard and put it into the accumulator (AC)
0110	6	Output	Copy the value of the accumulator (AC) and display it as the next line of output.
0111	7	Halt	Terminate the program
1000	8	Skipcond	Possibly skip the next instruction depending on some condition
1001	9	Jump *X*	Load the value of X into the Program Counter (PC). This is the address of the next instruction to be processed.

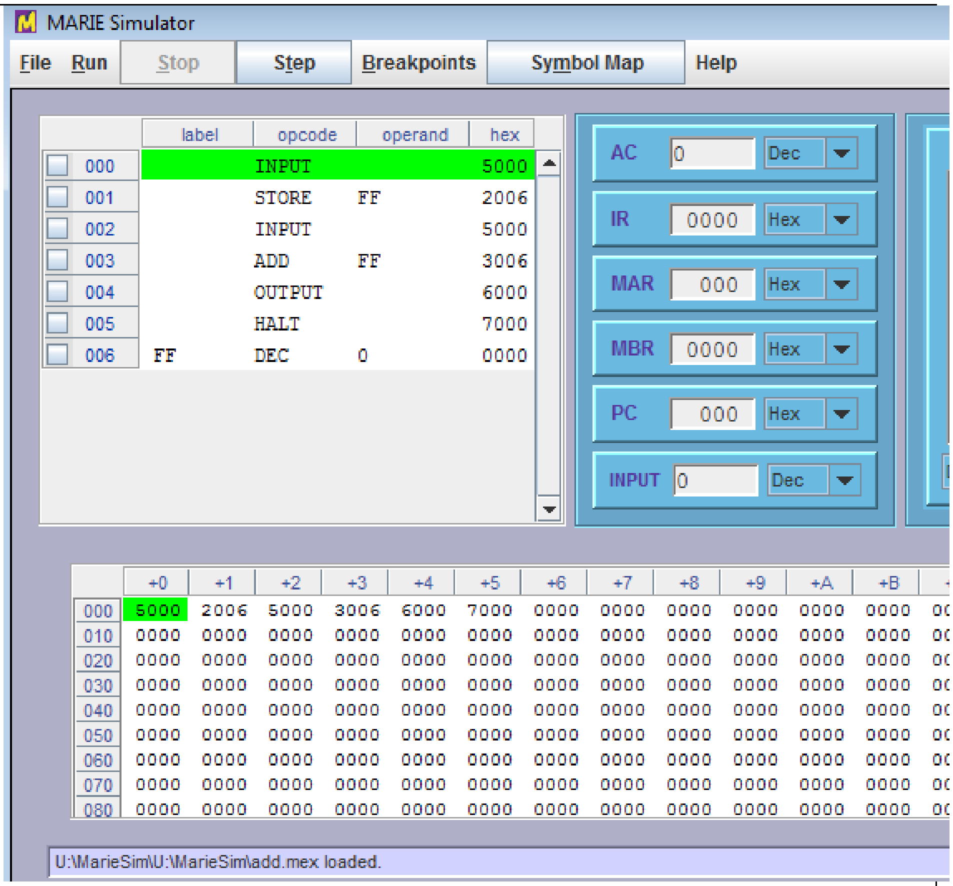
Example 1: Add two numbers together that are specified by the user, and output the result.

Pseudo Code	Marie Assembly Code	Machine Code	
		HEX	BIN
Get the first number from the user – which will put the value into the Accumulator	Input	5000	0101000000000000
Store the first number in memory location FF	Store FF	20FF	0010000011111111
Get the second number from the user – which puts the second value into the Accumulator	Input	5000	0101000000000000
Add the first number that was stored in memory location FF to the Accumulator	Add FF	30FF	0011000011111111
Output the sum – which is now in the Accumulator	Output	6000	0110000000000000
Halt the program	Halt	7000	0111000000000000

HWMath π

Notes: The Marie Simulator

File
  Load
    add.mex

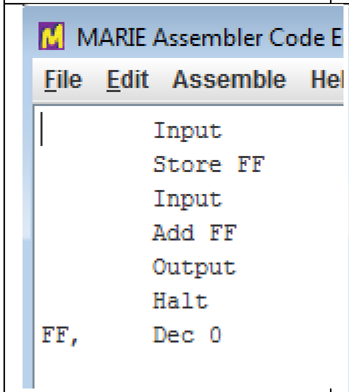


```
     label   opcode   operand   hex
000          INPUT              5000
001          STORE    FF        2006
002          INPUT              5000
003          ADD      FF        3006
004          OUTPUT             6000
005          HALT               7000
006  FF      DEC      0         0000
```

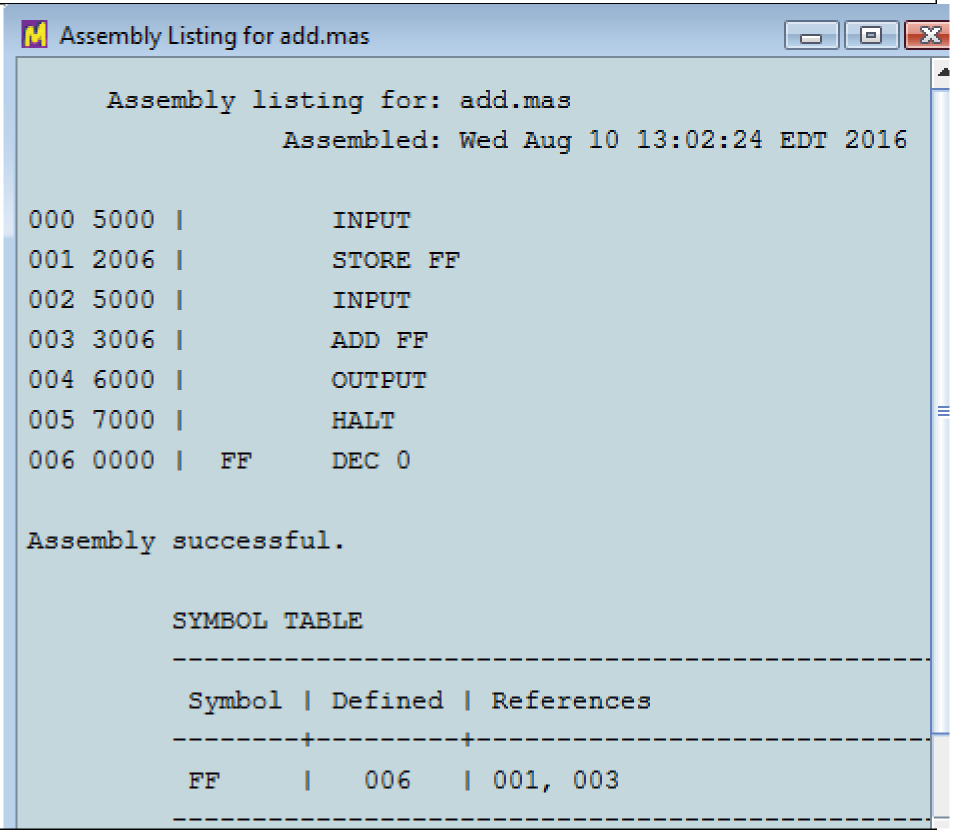
AC 0 Dec
IR 0000 Hex
MAR 000 Hex
MBR 0000 Hex
PC 000 Hex
INPUT 0 Dec

```
     +0   +1   +2   +3   +4   +5   +6   +7   +8   +9   +A   +B
000  5000 2006 5000 3006 6000 7000 0000 0000 0000 0000 0000 0000
010  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
030  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
050  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
070  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
```

U:\MarieSim\U:\MarieSim\add.mex loaded.



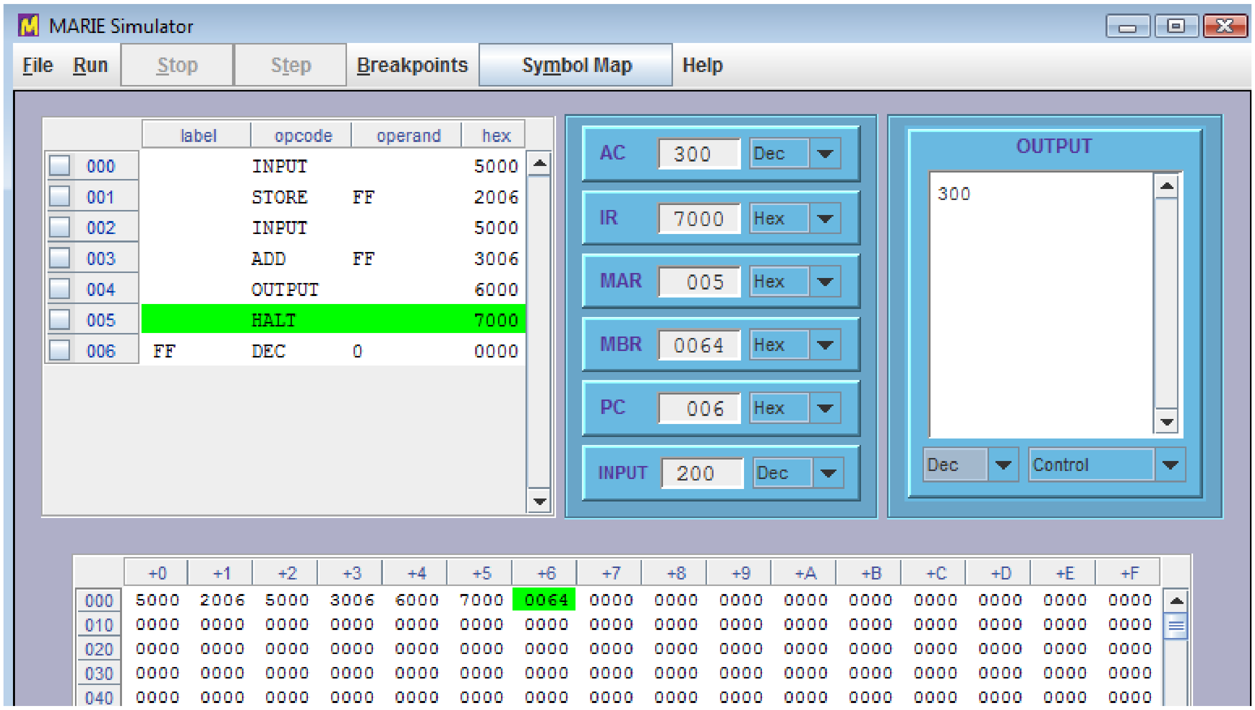
```
         Input
         Store FF
         Input
         Add FF
         Output
         Halt
FF,      Dec 0
```



```
    Assembly listing for: add.mas
              Assembled: Wed Aug 10 13:02:24 EDT 2016

000 5000 |          INPUT
001 2006 |          STORE FF
002 5000 |          INPUT
003 3006 |          ADD FF
004 6000 |          OUTPUT
005 7000 |          HALT
006 0000 |    FF    DEC 0

Assembly successful.

           SYMBOL TABLE
           ------------------------------------------------
            Symbol | Defined | References
           --------+---------+-----------------------------
            FF     |   006   | 001, 003
           ------------------------------------------------
```

Notes: The Marie Simulator

Program execution using decimal numbers 100 and 200



	label	opcode	operand	hex
□ 000		INPUT		5000
□ 001		STORE	FF	2006
□ 002		INPUT		5000
□ 003		ADD	FF	3006
□ 004		OUTPUT		6000
□ 005		HALT		7000
□ 006	FF	DEC	0	0000

AC	300	Dec
IR	7000	Hex
MAR	005	Hex
MBR	0064	Hex
PC	006	Hex
INPUT	200	Dec

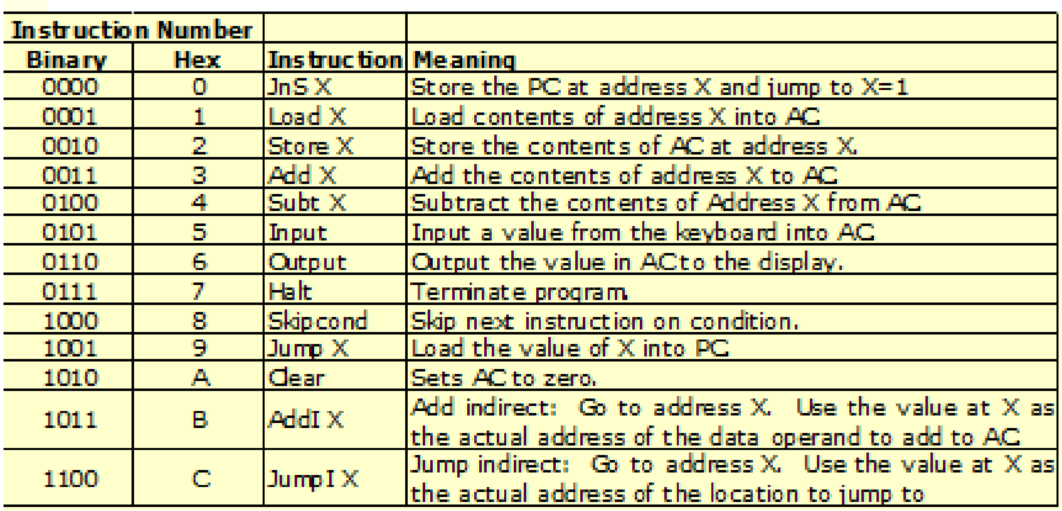
OUTPUT

```
300
```

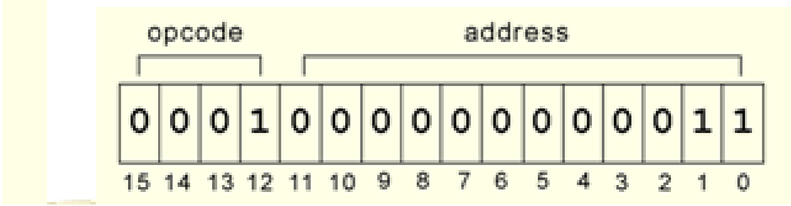
Dec Control

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	5000	2006	5000	3006	6000	7000	0064	0000	0000	0000	0000	0000	0000	0000	0000	0000
010	0000	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

Notes: The Marie Simulator



Instruction Number			
Binary	Hex	Instruction	Meaning
0000	0	JnS X	Store the PC at address X and jump to X=1
0001	1	Load X	Load contents of address X into AC
0010	2	Store X	Store the contents of AC at address X.
0011	3	Add X	Add the contents of address X to AC
0100	4	Subt X	Subtract the contents of Address X from AC
0101	5	Input	Input a value from the keyboard into AC
0110	6	Output	Output the value in ACto the display.
0111	7	Halt	Terminate program.
1000	8	Skipcond	Skip next instruction on condition.
1001	9	Jump X	Load the value of X into PC
1010	A	Clear	Sets AC to zero.
1011	B	AddI X	Add indirect: Go to address X. Use the value at X as the actual address of the data operand to add to AC
1100	C	JumpI X	Jump indirect: Go to address X. Use the value at X as the actual address of the location to jump to



Instruction Number		Instruction	Notes
Binary	Hexadecimal		
0000	0	JnS *X*	Store the program counter at memory location X, which would have been the next instruction to perform, and then jump (branch) to memory location X +1. This essentially stores the "return address" for making a "call" to the program block starting at X. Location X is where to save the return address, location X+1 is where the coding begins for a "called" subroutine. The subroutine will probably finish with a JumpI X command.
1010	A	Clear	Set the Accumulator to zero
1011	B	AddI X	Add indirect – go to address X and use the value at X as the actual address of the data operant to add to AC. In higher level languages this is considered a pointer to an integer
1100	C	JumpI X	Jump Indirect – go to address X, use the value at X as the actual address of the location to jump to; This is good for branching back from a subroutine call.
1101	D	LoadI X	Load Indirect – go to address X, use the value at X as the actual address of the location to load into the accumulator AC. AC = Mem[X]
1110	E	StoreI X	Store Indirect – go to address X, use the value at X as the actual address of the location of where to store the value in the accumulator. Mem[X] = AC

each instruction for MARIE consists of 16bits. The most significant 4 bits, bits 12–15, make up the opcode that specifies the instruction to be executed (which allows for a total of 16 instructions). The least significant 12 bits, bits 0–11, form an address, which allows for a maximum memory size of 212–1. The instruction format for MARIE is shown in Figure 4.10.

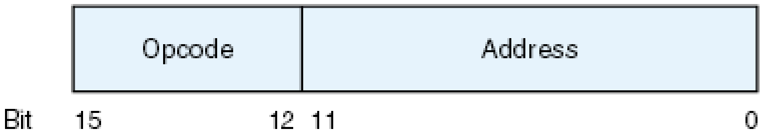


FIGURE 4.10 **MARIE's Instruction Format**

AC Accumulato

Input Register

IR Instruction Regiser

Input Register

MAR Memory Address Register

MBR - Memory Buffer Register

Memory

Output Register

PC Program Counter

ALU Arithmetic Logic Unit

Notes: The Marie Simulator

Glossary

instruction set architecture (ISA) of a machine specifies the instructions that the computer can per-form and the format for each instruction.