Barbusoft 9000

User Manual

Jacob Barbulescu

Welcome to the future! This manual will help you change the world with the Barbusoft 9000, a cutting edge tool that has reshaped the computing industry. Feel free to read cover to cover or simply find what you need with the table of contents.

The Architecture

The Barbusoft 9000 is a computer with 4 general purpose registers, 16-bit instructions, and an LED display. The computer also supports the use of 4-bit immediate numbers in its calculations.

The computer is capable of addition, subtraction, loading in memory, storing in memory, directly setting registers, and displaying the value of a register.

There are also two RAMs. One is the InstructionRam, which stores the instructions of the program. The other is the DataRam, which stores the data used throughout the program. The InstructionRam is set externally by the user, but the DataRam can only be modified by the CPU via program instructions.

The 4 registers each store 8 bits of information, and are represented as x0, x1, x2, and x3 in the assembly code. The binary representation of each registers is:

00 –> x0, 01 –> x1, 10 –> x2, 11 –> x3

Each instruction contains the binary reference or 3 registers, and they will be represented as Xn, Xm, and Xd in this manual. Xn and Xm are typically used in

calculations, while Xd is the register either being modified, stored in memory, or displayed.

The LED display can represent one register value at a time, and the display will persist until a new register value is displayed. The LED display represents the register's value through a hexadecimal representation.

As stated, each instruction is 16-bits. 6 are used to choose the operation, 4 is used to represent an immediate number, and the last 6 are used to represent 3 different registers (2 for each register).

END- While this is 1, the program will run. When it equals 0, the CPU will do nothing and the PC will no longer increment (effectively, the program is ended).

DISP- When set to 1, the LED display is now set to be overwritten by the ALU's output.

SET- When set to 1, the program will set the ALU's output to either Xm or the immediate number.

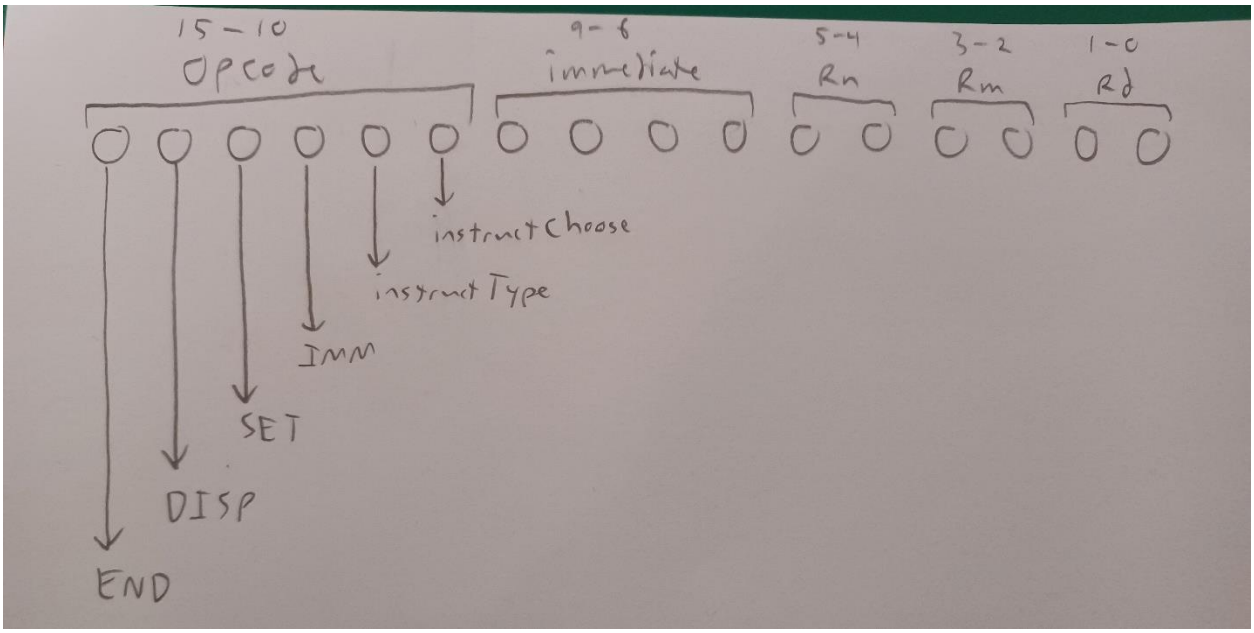
IMM- When set to 1, Rm in the ALU is effectively replaced by the immediate number.

InstructType- When set to 0, the ALU will select the output of the ADD/SUB calculation. When set to 1, the ALU will select Xd (LDR/STR).

InstructChoose- When set to 0, the ALU will choose the output of ADD/LDR (based on InstructType). When set to 1, the ALU will choose the output of SUB/STR (based on InstructType).

InstructType	InstructChoose	Operation
0	0	ADD
0	1	SUB
1	0	LDR
1	1	STR

Every instruction will have this form:



The Instructions

The Barbusoft 9000 supports 6 instructions: ADD, SUB, LDR, STR, SET, and DIS. Every instructions can either use Xm or a 4-bit immediate number in its calculations.

The following is a list of all instructions and their immediate number variations:

ADD Xn, Xm, Xd- Sets Xd to Xn+Xm (Opcode: 100000)

ADD Xn, immediate, Xd- Sets Xd to Xn+immediate (Opcode: 100100)

SUB Xn, Xm, Xd- Sets Xd to Xn-Xm (Opcode: 100001)

SUB Xn, immediate, Xd- Sets Xd to Xn-immediate (Opcode: 100101)

LDR Xn, Xm, Xd- Loads the data at address Xn+Xm in DataRam to Xd (Opcode: 100010)

LDR Xn, immediate, Xd- Loads the data at address Xn+immediate in DataRam to Xd (Opcode: 100110)

STR Xn, Xm, Xd- Stores the data in Xd in address Xn+Xm of DataRam

(Opcode: 100011)

STR Xn, immediate, Xd- Stores the data in Xd in address Xn+immediate of DataRam

(Opcode: 100111)

SET Xm, Xd- Sets Xd to Xm (Opcode: 101000)

SET immediate, Xd- Sets Xd to immediate (Opcode: 101100)

DIS Xd- Display's Xd's value on the LED display (Opcode: 110010)

DIS immediate- Displays the immediate number's value on the LED display

(Opcode: 111110)

Comments in the code are represented with //.

Ex. ADD X0, 5, X1 //This is a comment

//Comments can also be on their own line

Writing Your Code

All Barbusoft programs are saved in the .barb format. Inside these files, each instruction goes on its own line. Comments can either be appended to an instruction or on its own separate line. Blank lines are also permitted. Capitalization does not matter, and neither do spaces. However, be sure to place a comma between each parameter for every instruction.

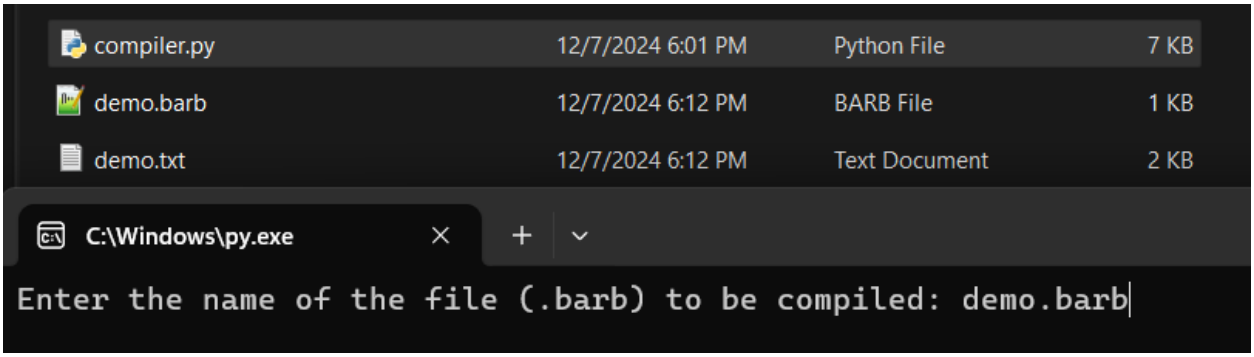
```
//Inititalizes x0 and x1 to 4 and 3, respectively
Set 4, x0
SET 3, X1

DIS 15      //Displays 15 (0f), just for funsies

ADd x0, x1, x2 //X2 = x0+x1
DIS x2           //Shows the value of x2 on the LED display
STR x0, 5, x2   //Stores x2 (7) in address x0+5 (9) of the DataRam

LDR x0, 5, x3   //Loads address 9 of DataRam to x3
SUB x3, 5, x3   //x3 = x3-5
DiS x3
ADD x3, x3, x3  //x3=x3+x3
DIS x3

str x0, x1, x3  //Stores x3 at address x0+x1 (7) of the DataRam
```

The following demo program shows how a typical Barbusoft program will look when written out.

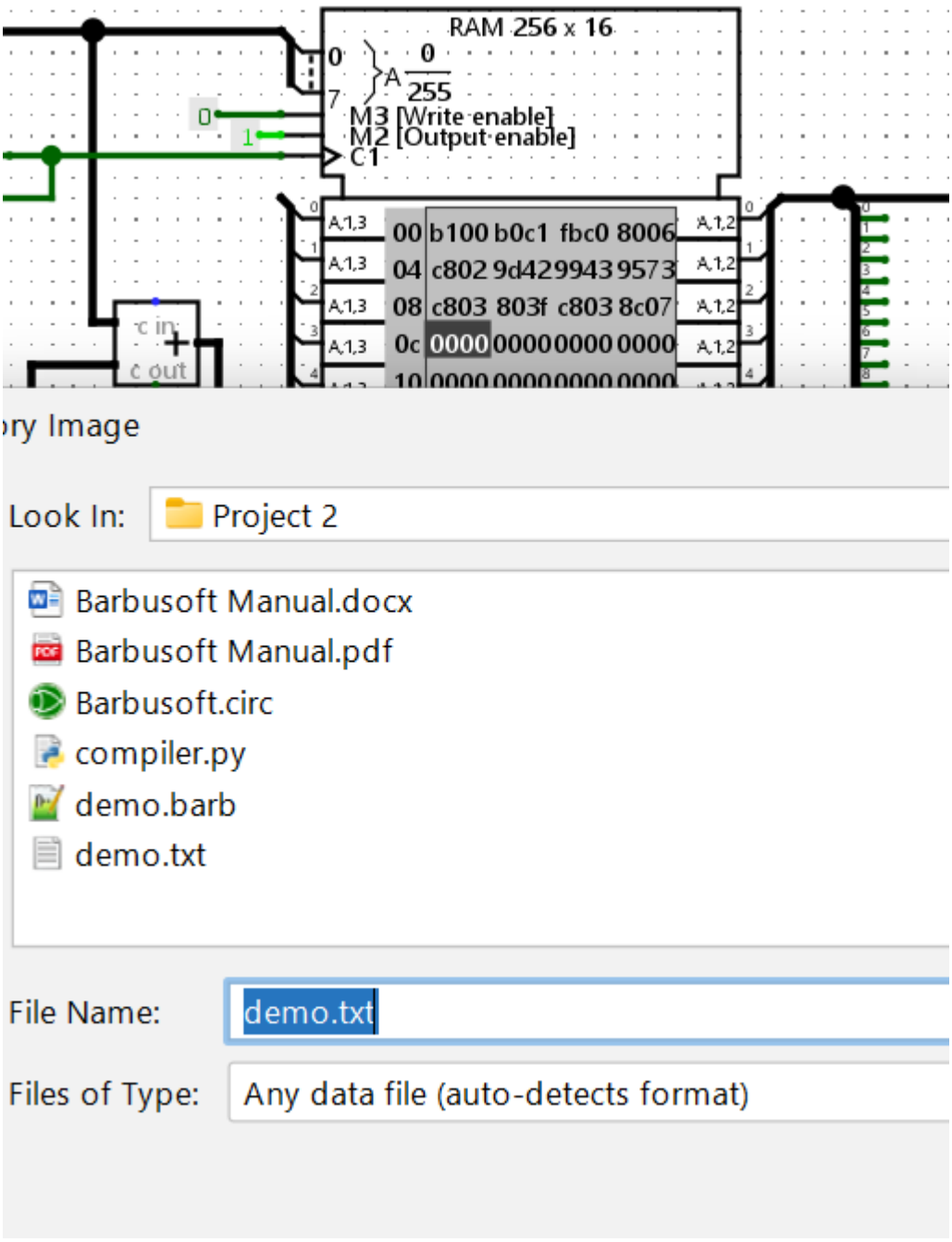
Running Your Code

Once you have your code written, it's time to compile and run the program and have Barbusoft work its magic!

First, import the compiler.py file into whatever directory your code is in. From there, run the compiler file. You will be prompted to enter your file name. Simply type in your program's name (including the .barb extension) and press enter. The compiler will automatically create an image txt file with your program's name.



In the Barbusoft computer, enter the "Instruction_Register" subcircuit. From here, be sure that the simulation is fully reset, then right click the RAM. Click "Load Image", the find and select your program's txt image. The RAM should have loaded your instructions.



Finally, go back to the "main" circuit and have the clock run. You can either have the clock activate manually by clicking it, or by enabling auto-tick. Your program should now run!

The program will end after the last instruction, meaning that the PC will no longer iterate through the Instruction Ram. The computer will be stuck doing nothing until you reset the computer (via resetting the simulation).