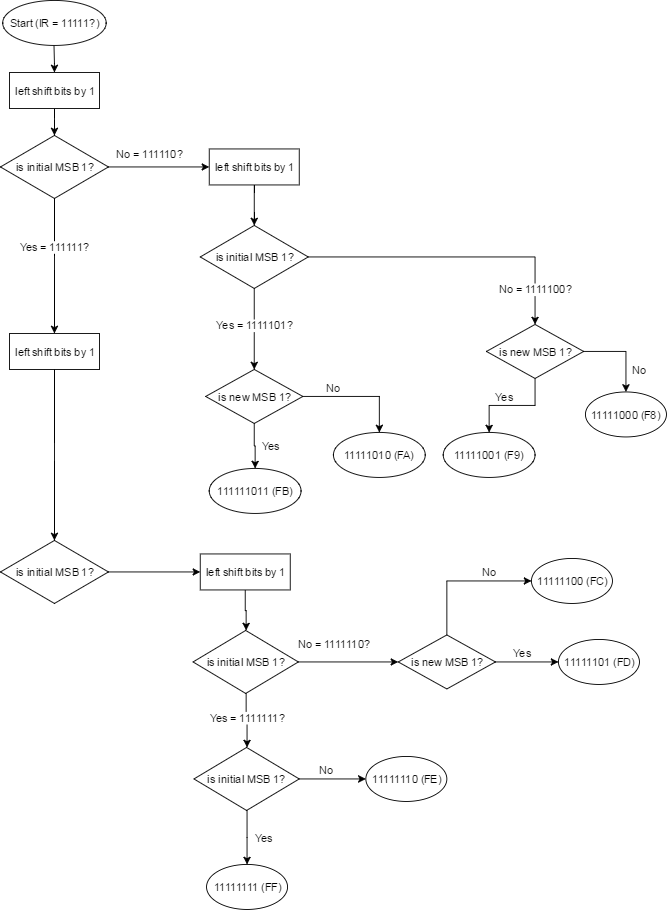
**Assignment 4 Documentation**

**Name:** Jak Hall

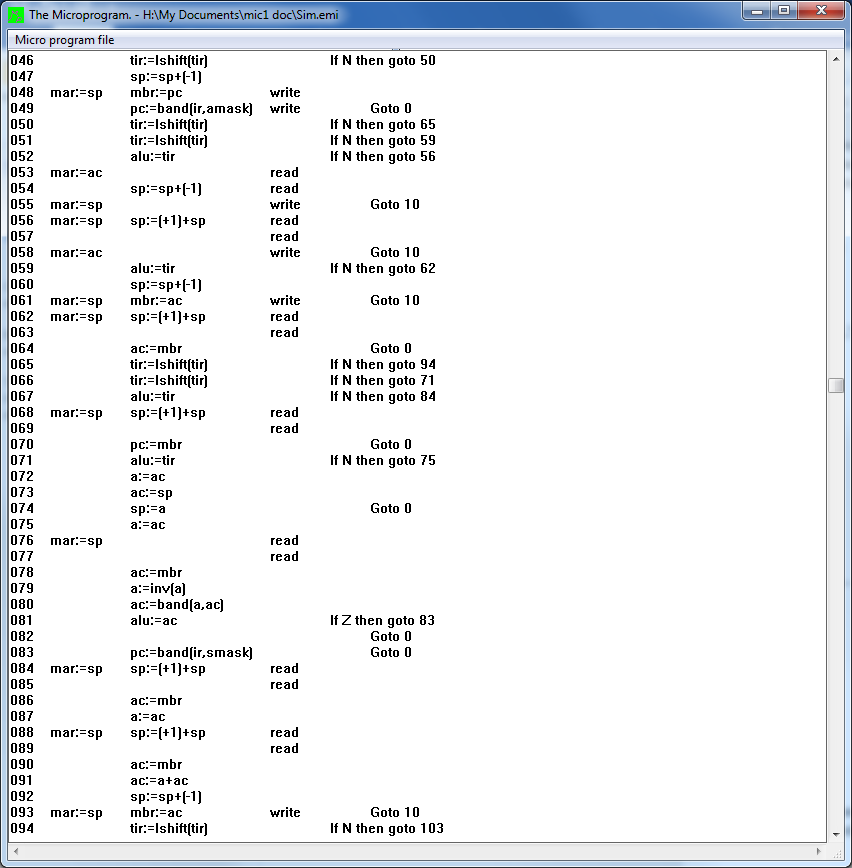
**Student Number:** 15826673

**Adding Opcodes**

The first problem with editing the microprogram is that there is only space for one more opcode without creating additional branching since DJNZ’s hexadecimal address in the high byte of the Instruction register is FF (or 1111 1111) meaning the only option for adding the other 3 opcodes is to either extend the size of the IR register or creating more specific branching to locate each specific binary address. To do this I had to add additional shifting and bit checking into the microprogram, but only after line 65 since only 3 extra spaces are required. In order to understand this I created a flow chart diagram:

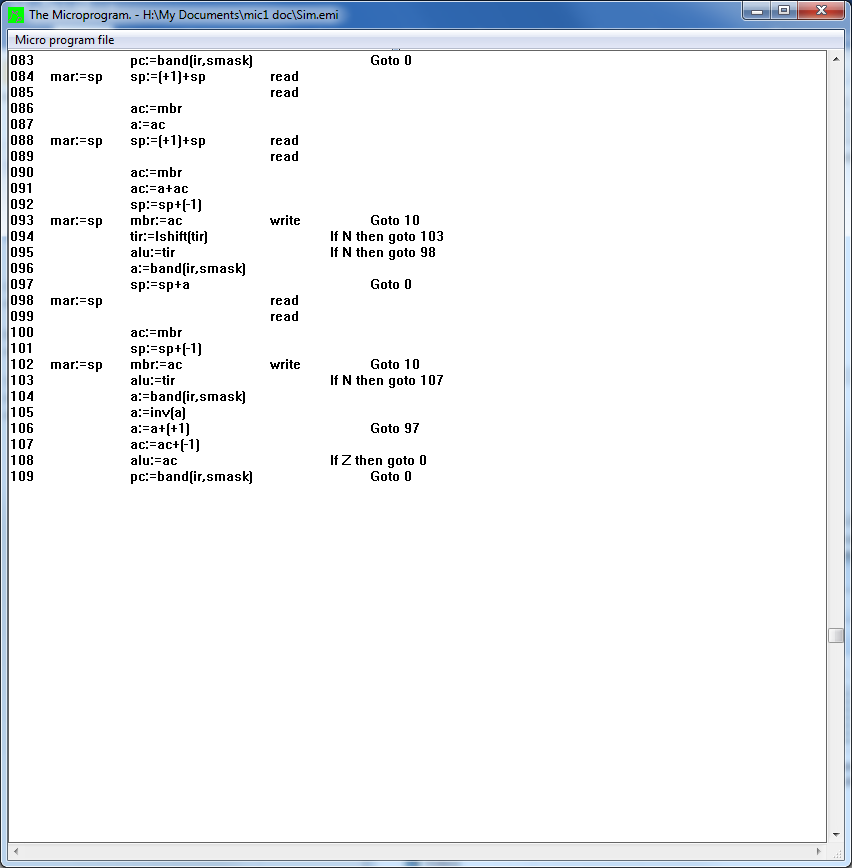


Left shifts were done to check every bit to see if it was a 1 or a 0, depending on each situation it will point towards a different opcode, for example this is done with this microcode:



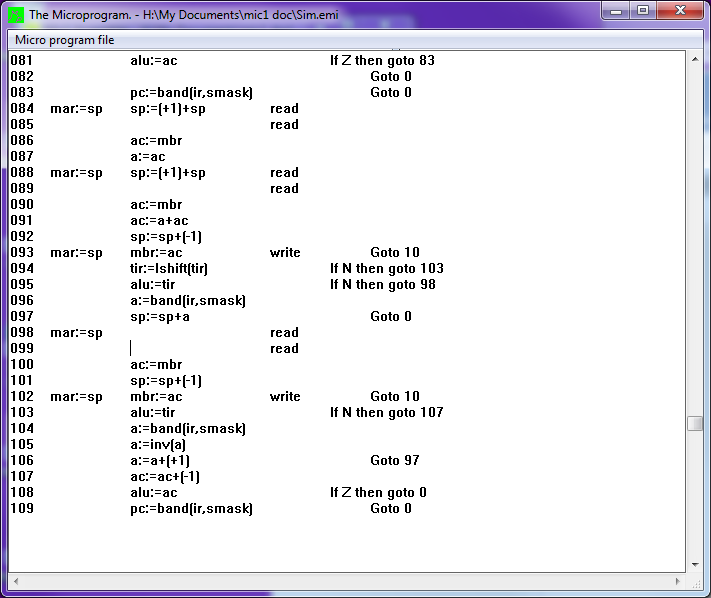
**DJNZ (FF00)**

Decrement, jump if not zero will first decrement the value of the accumulator (ac) register and then check if the ac is equal to 0, if it is then it will return to 0 and stop the opcode, otherwise it will jump to the specified program count. Here is my microcode:



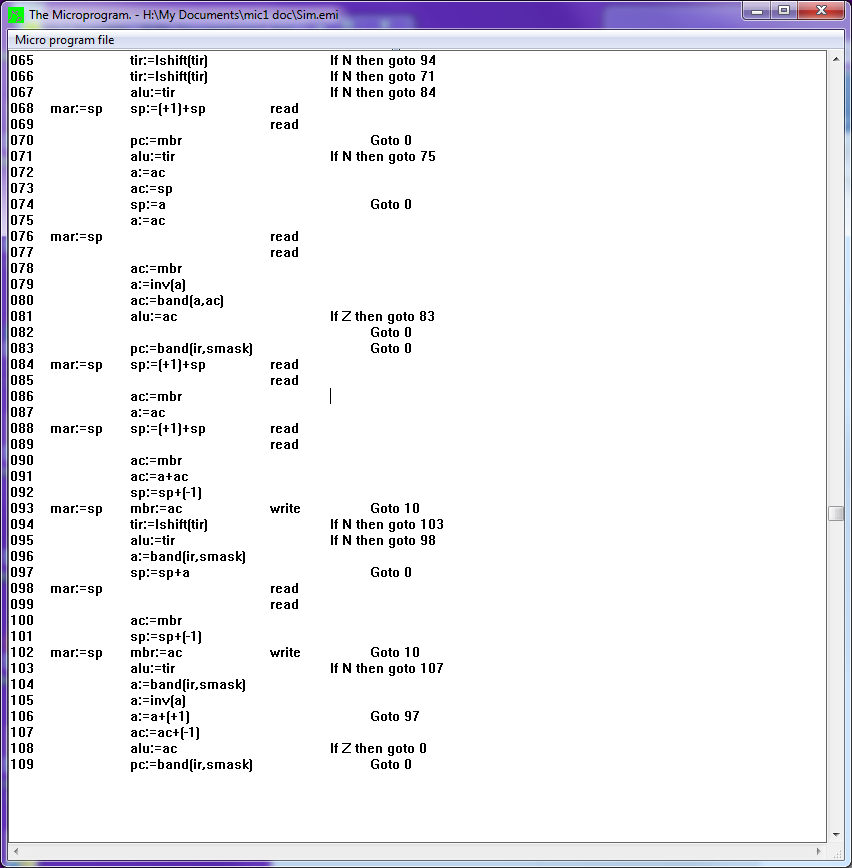
**DUP (FD00)**

This should look at the value at the top of the stack, make a copy of it and then push it onto the top of the stack so there are two duplicates of it on the stack. Here is my microcode:



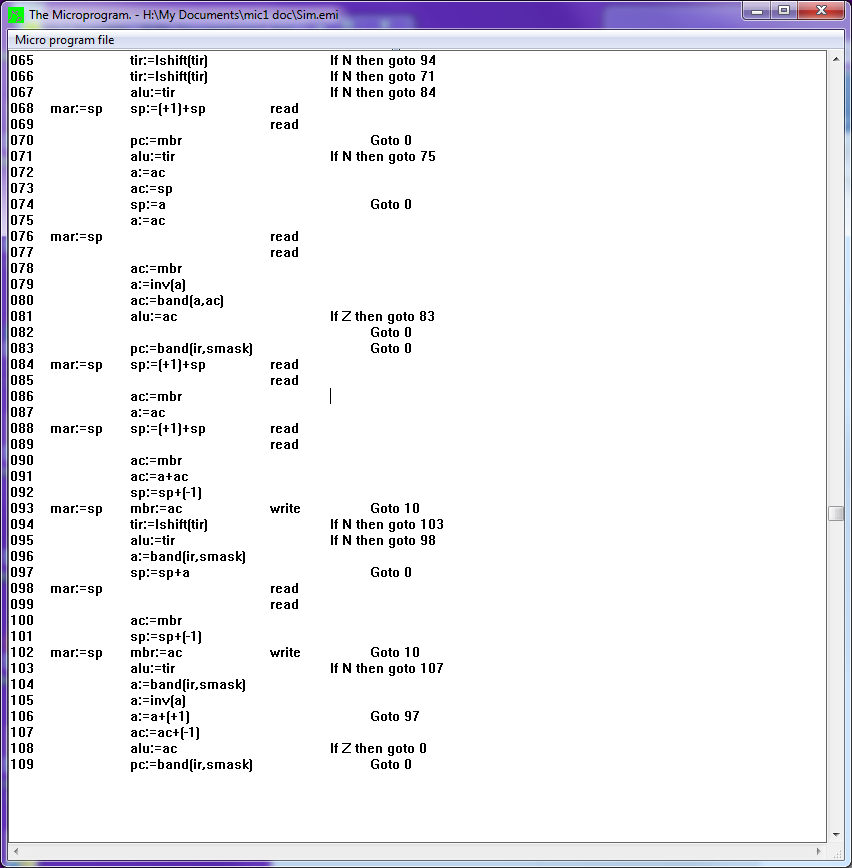
**CMP (FB00)**

CMP will always compare the value at the head of the stack to the value of the accumulator, if they are the same then it will branch, if they are not then it won’t. Here is my microcode:



**IADD (F900)**

The opcode IADD will take the pop the top two values on the stack, add them together and then push the sum onto the stack again. Here is my microcode:



**Summary**

Creating the opcodes turned out to be a big challenge, it was very hard to understand how to input the opcodes and then locate them via there hexadecimal address. Once it was clearer that the IR register was split into two parts the High byte which store the address and the low byte that stores the instruction it was much easier to work out how to branch the microprogram correctly in order to differentiate between opcodes. Implementing the actual opcodes was relatively easy as they are very basic logical instructions, I spent a long time researching assembly instructions that would be helpful to a programmer, and I believe the three that I chose are fulfil this requirement. In the process I have learnt a lot about microprogramming as well as how assembly language is implemented, I should find low level architecture easier in the future.