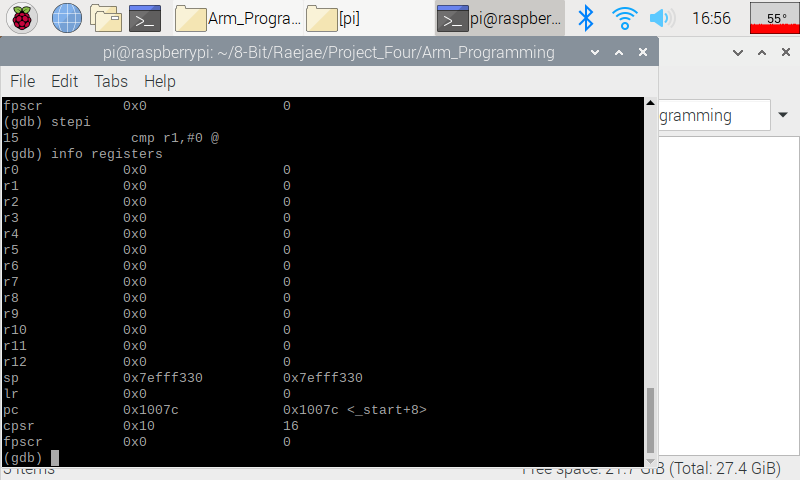
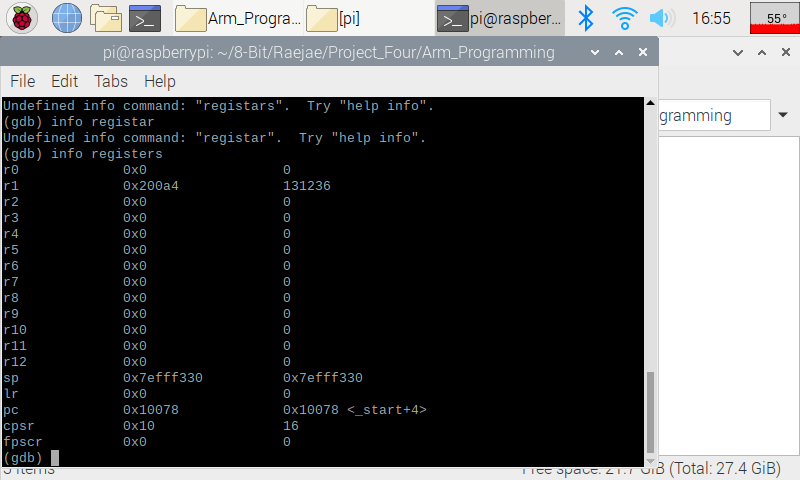
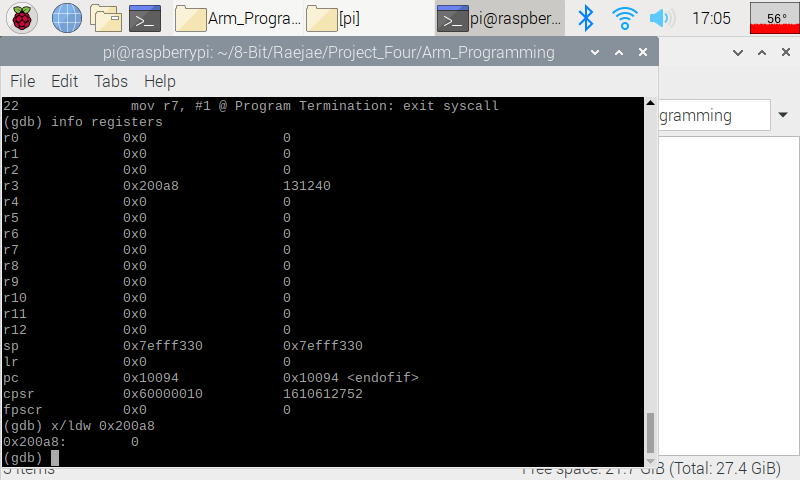
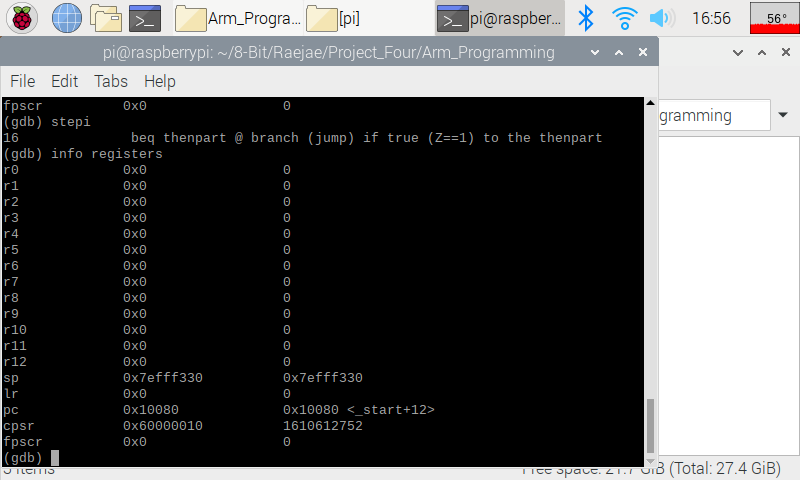
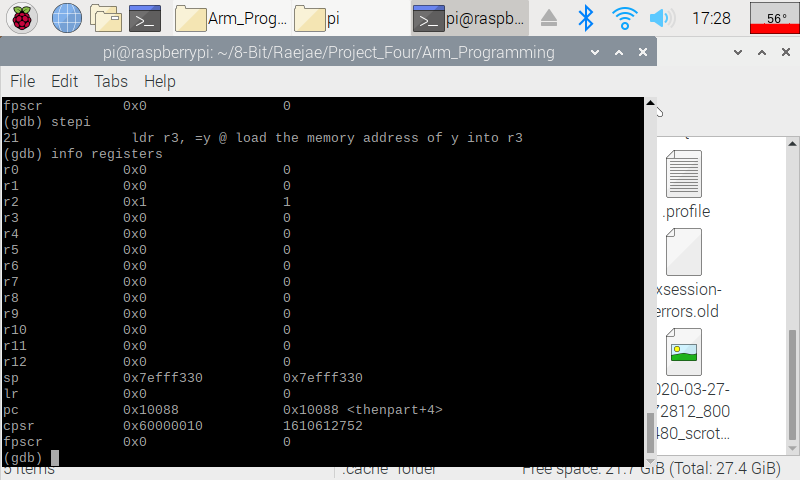
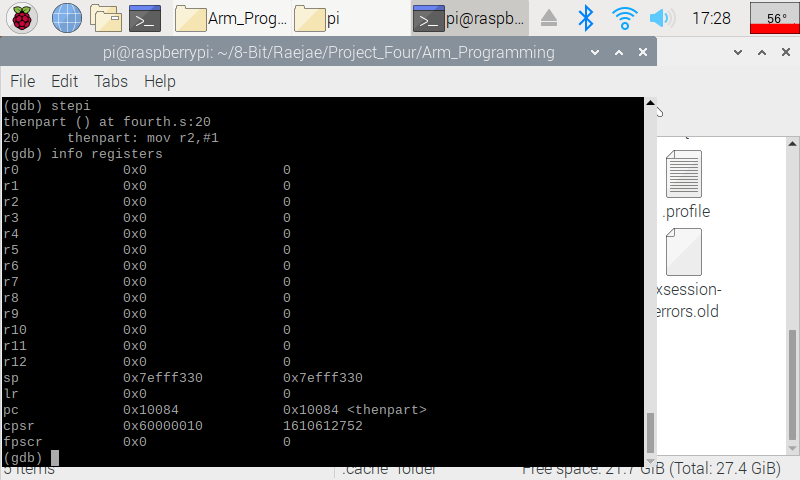
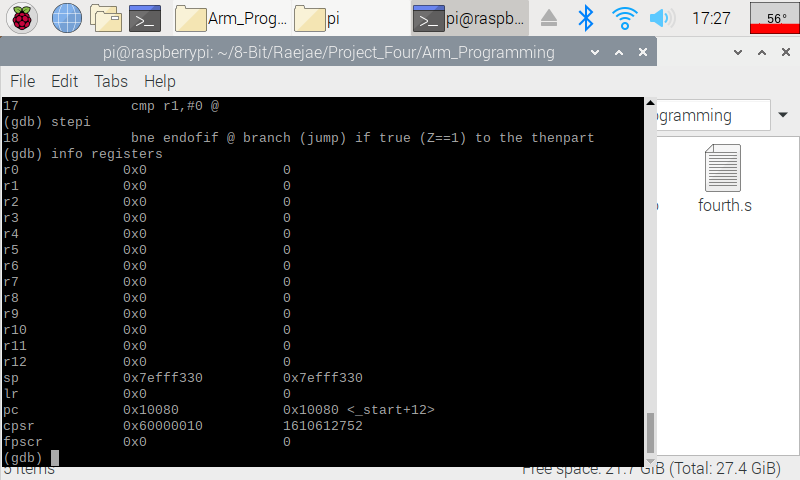
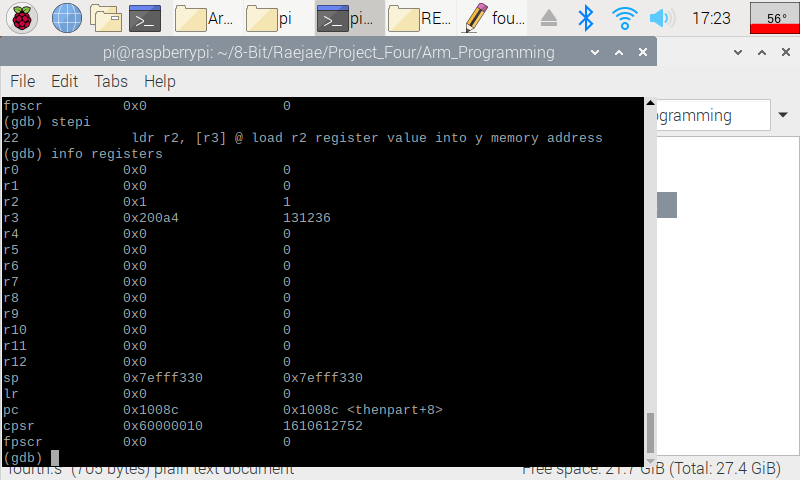
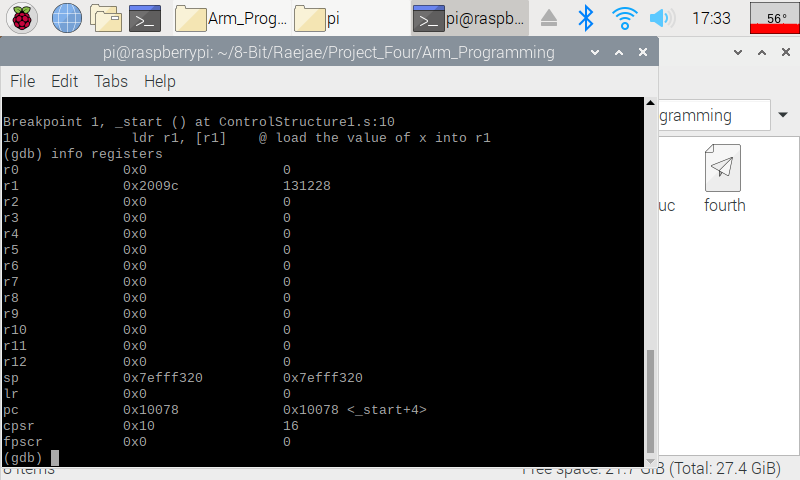
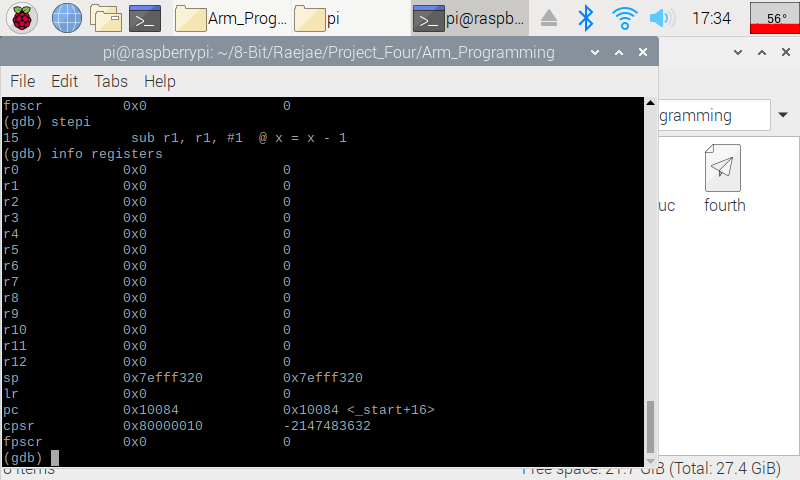
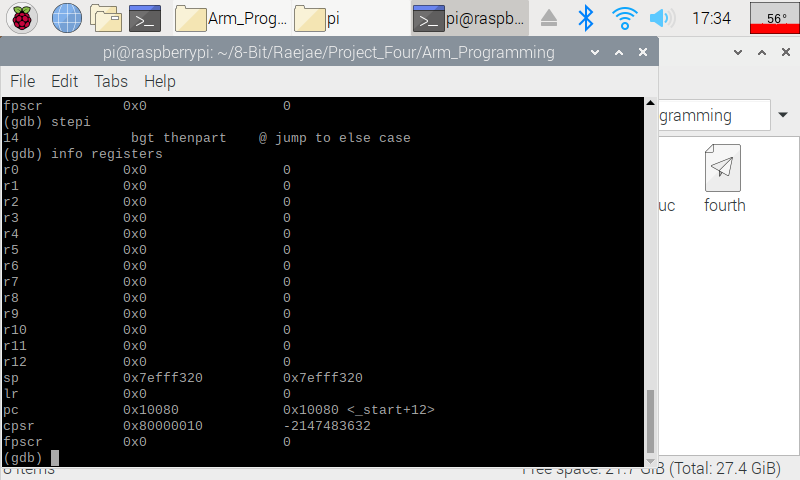
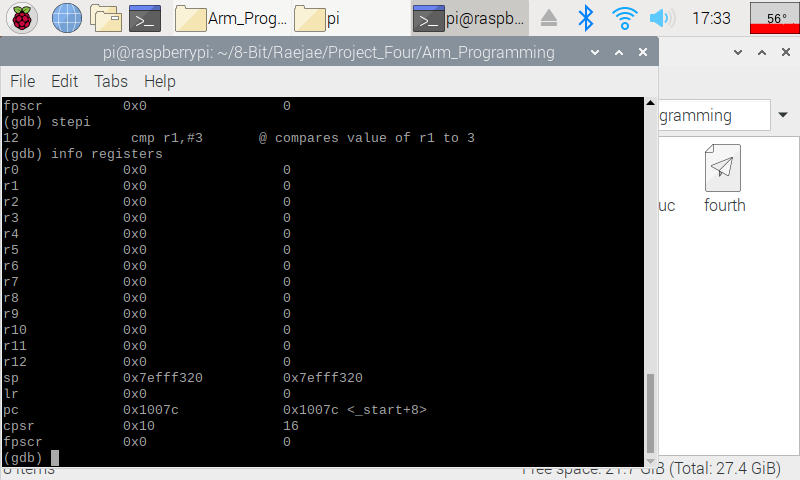
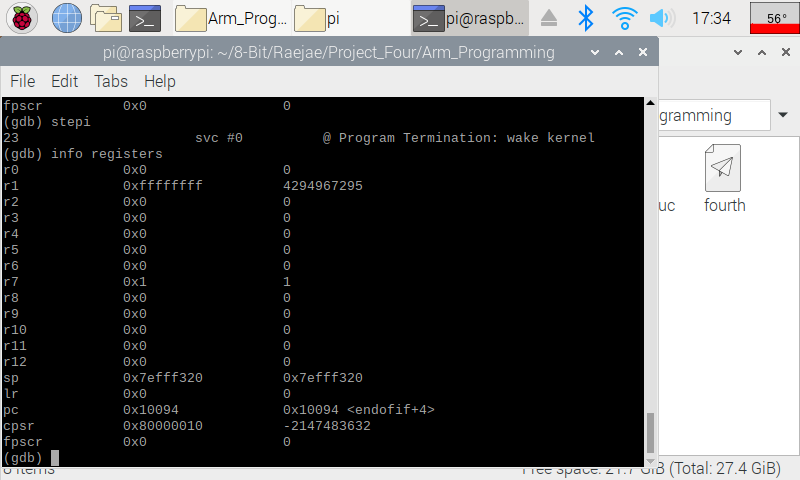
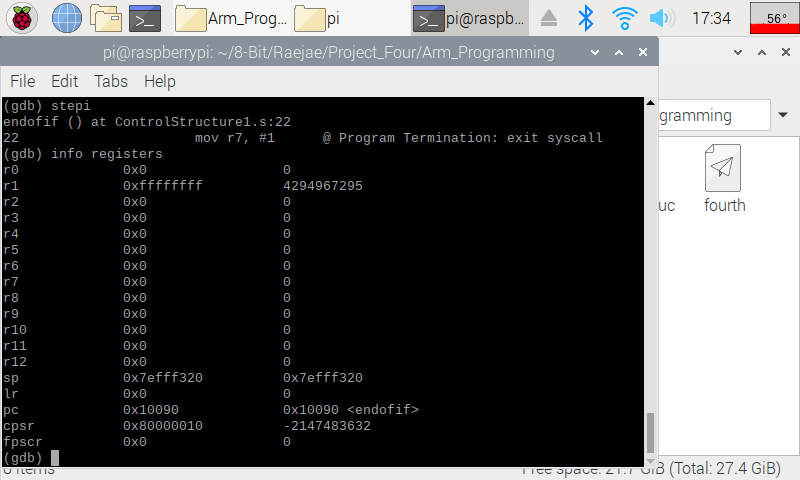
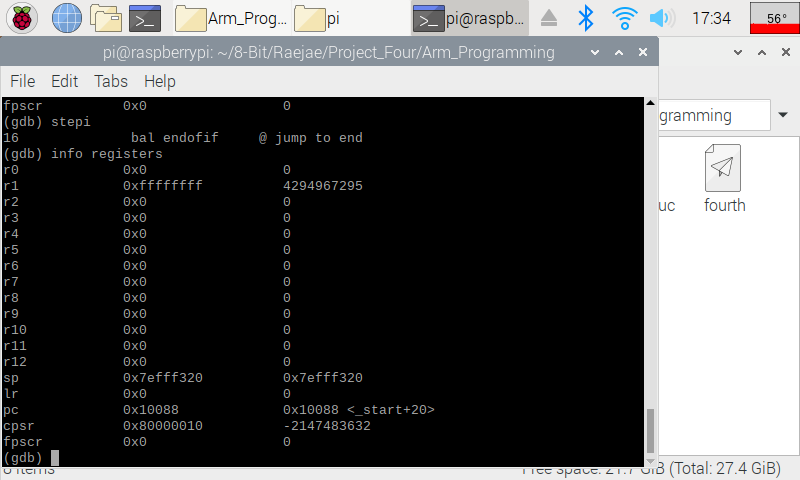
  
  
A lot happens in this code but the one thing we want to remember is that if the value is equal to 0 then y will be 1. Deviations from this shows we have an issue, and there were discrepancies over the execution of this code within our group as there may have been an error. However the made thing to focus on is how the code handles if statements with their version of the zero flag and jumps.  
  
  
Here we’re simply loading our numbers, and if we pay attention we see the addresses being filled.   
  
  
  
--------  
Improved   
  
  
  
I felt it would be easier to explain by showing the improved version and then explaining the intended output of the code. In the improved version the main thing we need to look at and understand is beq vs ben in the code jumping. The CPSR flag is our Zero Flag determinant and will dictate how these operate and since the results are equal to zero on both we will get our code flowing. What we want is for the code after line 18 to take our result and update and only access the in-between code if the resultants are not equivalent to zero. We get this result albeit we have to make sure we store properly in registers after our jumps.

Control:

  
First thing first we want to make sure we set our values and memory addresses up correctly. If we pull it up we will indeed get one.

  
  
The important thing to notice here is that we’re comparing R1 to a value. We can sub any value but if the difference is there we will branch or not. We don’t have a value greater than the compared so we will set our breakpoint elsewhere.   
  
  
Our code will execute the following based on what is compared above jumping to either the elseCase or the EndCase and in this case jumping out.