

Concepts in Programming Languages

Assignment 4

**Topic:** Analysis of Assembly code generated by (GCC) Complier

**Student Details:**

**Name** :Abhishek Prashant Chandurkar

**Class**:2nd Year 4th semester

**Department** : Computer Science and Engineering

**Enrollment No**. : BT22CSE104

**Analysis of .gimple file**

1)Substraction replaced by addition.For example: if we want to compute (n-1) we will do so as (n+ -1)

2)Use of temporary variable to store return value other than the one defined by programmer.

3)The variable declarations done separately(not cascaded).

Eg:

int a,b; converted to 2 statements int a; and int b;

**Analysis of O3(Level 3 optimised assembly code)**

1)It is a 6 argument register function which is quite big in terms of number of statements.

2)Recursive in nature but consists of many cases unlike the the original code.

3)Probably like

int fib(int first)

{

//Using all parameter registers

int t4,t8,t12,t16,t20;//The temp vars allocated on stack

/\*\*\*The logic comes here\*\*\*/

}

4)Consumes a memory of (56+72) bytes on stack for Activation record (excluding return address )and all the general purpose registers(r10 to r15).Hence not memory optimized.

5)Stack grows from Higher to Lower address.

6)The stack memory .locations can be accessed with offsets as multiple of 4 using stack pointer.

The base address of AR is stored in %rbp(stack base pointer register) and the stack pointer in %rsp.

7)Refer Assembly\_Code\_Analysis3.s file for assembly code.

**Analysis of O1(Level 1 optimised assembly code)**

1)Unlike O3, it is quite simple to read since it has a structure similar to the original c code.

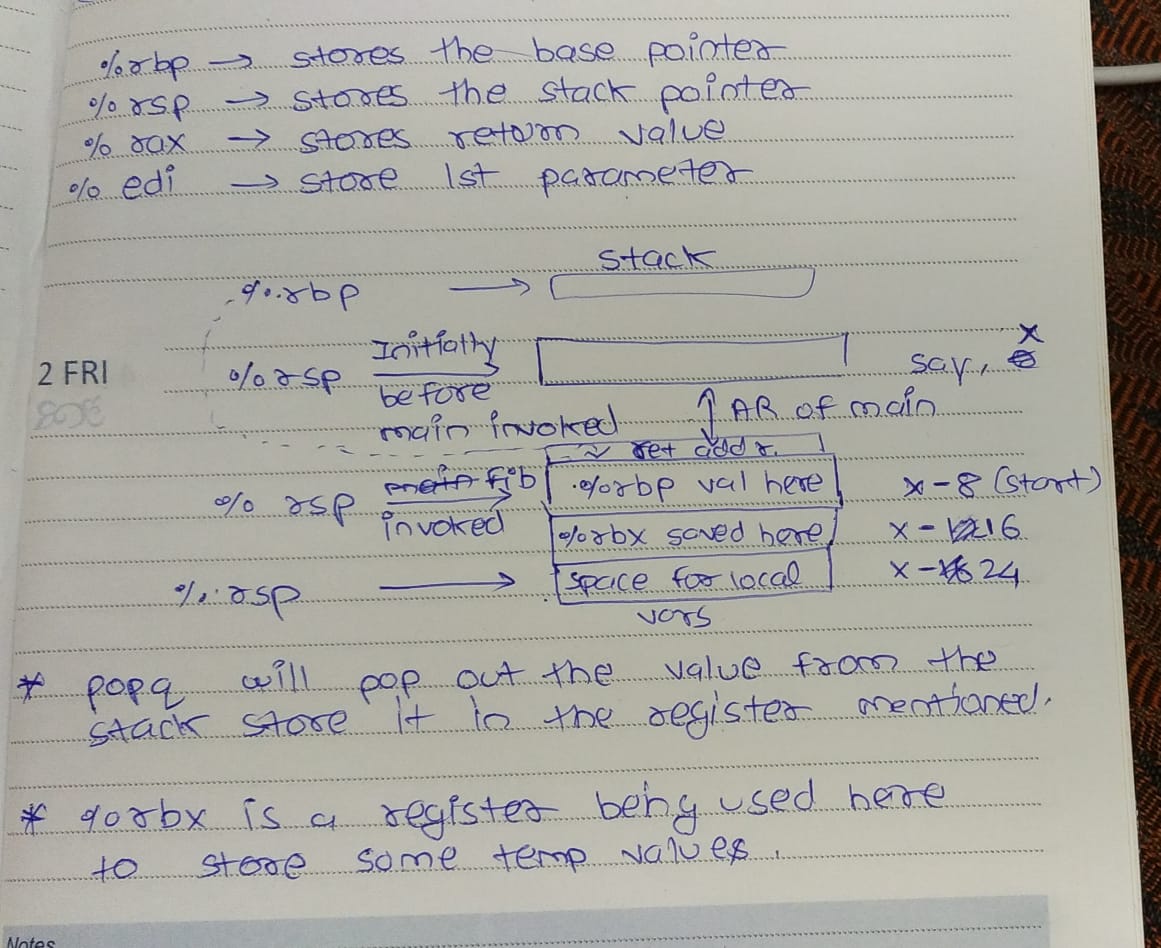
2)The AR size is 32 bytes

3)8 bytes each for %rbp(Caller base pointer) and %rbx(Callee Saved) and 8 for return address and 8 for some other information..

4)A line by line explanation of the assembly is in the file attached(Assembly\_Code\_Analysis1.s).

Note: Here the AR size refers to the size allocated on stack(including the return address which is the pushed on stack by caller ). It includes return address ,local variables (which could not be accommodated in registers).

5)What happens on stack:



The above image just for an idea of what happens on the stack.

The return address is pushed to stack by processor(call).

It is fetched by the processor and stored in %rip during ‘ret’ instruction.

**Some Important Observations about Optimising code with redundant statements**:

1)If a function is being called which affects only its local variable and not affect others ,then the function call is simply removed by the compiler.

2)It is the same for arithmetic expressions.

3)Refer the files OnlyMain.s and TwoFunc.s.

**Technical Details:**

A 64 bit gcc compiler on installed on Ubuntu OS was used.

The x86-64 bit assembly language is used.

**References:**

Some of the main references are:

1)Video by MIT: <https://youtu.be/wt7a5BOztuM>

Just for understanding of x86 language terminology and Stack.

Also helped in choosing the C code with a function definition and function call which would cover as many concepts as possible.

2)Register Names: <https://cs61.seas.harvard.edu/site/2018/Asm1/>

Got to know about the symbols like %rbp,%rsp,etc.

.