

**“AZƏRBAYCAN HAVA YOLLARI” CJSC NATIONAL AVIATION ACADEMY**

**Individual Work № 8:**

**Topic: Convert C code to assembly language**

**Subject: System software and Operating systems-2**

**Teacher: Mehemmed Shahmaliyev**

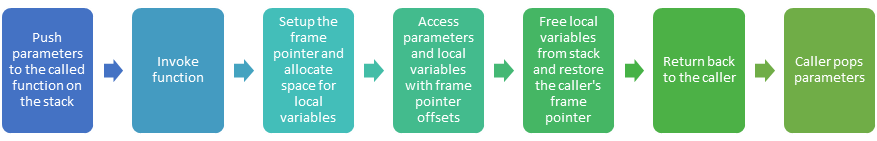
**Group: 1459i Student: Mammadov Ahmad**

**Date: Signature: M.A**

**Baku 2022**

**C to Assembly Translation**

**C to assembly: function calling**



Even though most programming is now carried out in high level languages, a good understanding of the generated assembly code really helps in debugging, performance analysis and performance tuning.

Here we present a series of articles describing C to assembly translation. We will be mapping C code to pseudo-assembly. The concepts learnt here can easily be applied to understand the generated code for any real processor assembler.

In this article, we will discuss the assembly code generated for function calling, parameter passing and local variable management. Before we go any further we need to discuss a few things about the pseudo-assembler.

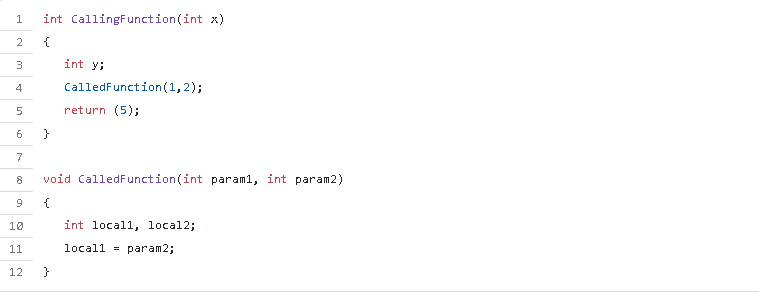
## Pseudo assembler basics

* Processor registers are designated as R0, R1, etc.
* The MOVE instruction has the source on the left side and destination on the right side.
* Register RETURN\_VALUE\_REGISTER is used to return values to the calling function.
* The stack in the pseudo-processor grows from higher address to lower address. Thus a push results in a decrement to the stack pointer. A pop results in an increment to the stack pointer.
* Register STACK\_POINTER is used to point the stack.
* Register FRAME\_POINTER is used as the frame pointer. The frame pointer serves as an anchor between the called and the calling function.
* When a function is called, the function first saves the current value of the FRAME\_POINTER on the stack. It then saves the value of the STACK\_POINTER register in FRAME\_POINTER register. This is followed by decrements the STACK\_POINTER register to allocate space for local variables.
* The FRAME\_POINTER register is used to access local variables and parameters. Local variables are located at a negative offset to the frame pointer. Parameters passed to the function are located at a positive offset to the frame pointer.
* When the function returns, the FRAME\_POINTER register is copied into the STACK\_POINTER register. This frees up the stack used for local variables. The value of FRAME\_POINTER register for the caller of this function is restored from the stack by a pop.

## Function calling

The following block shows the C code and the corresponding generated assembly code.

C Code

****

## Function calling sequence

The generated assembly code is best understood by tracing through the invocation of CalledFunction() from CallingFunction().

### Pushing parameters

CallingFunction() pushes values 2 followed by 1 on the stack. These values correspond to param2 and param1 respectively. (Note that pushing order is reverse of the declaration order.). This is implemented by the PUSH instruction. The PUSH instruction pre-decrements the STACK\_POINTER register and then copies the value to the address pointed to by the STACK\_POINTER.