# Calling ‘C’ Function from Assembly

Main Assembly Function

asm_main.asm

```
    .global  main   ; makes main accessible from outside this file.
    .ref myAddC   ; external label, here ‘C’ function name
    .thumb        ; use thumb, UAL syntax
    .data         ; set memory location to SRAM
    ; put your variables here
    .text         ; set memory location to flash
main:
    MOV r0, #1 ; 1st parameter
    MOV r1, #2 ; 2nd parameter
    MOV r2, #3 ; 3rd parameter
    MOV r3, #4 ; 4th parameter
    BL myAddC
Here:
    B Here
    .end
```

‘C’ Function

filename.c

```
int myAddC(int x1, int x2, int x3, int x4)
{
    int result=0;
    result = x1 + x2 + x3 + x4;
    return(result);
}
```

- Create an C project with empty main.c
- Either delete or rename the main.c e.g. filename.c
- Create a new source file with extesion .asm say asm_main.asm
- Build and debug the code.

# Calling Assembly Function From ‘C’

C Function

main.c

```
extern int sum1toN(int);
int main(void)
{
    int N = 5;
    int sum = sum1toN(N);
    while(1);
}
```

Assembly Function filename.asm

```
    .global sum1toN
    .thumb          ; use thumb, UAL syntax
    .data           ; set memory location to SRAM
     ; put your variables here
    .text           ; set memory location to flash
sum1toN:
    ; Initialize registers
    ; Loop Counter is in r0
    MOV r1, #0 ;Starting result

Loop:
    ADD r1, r0   ; r0 = r1 + r0
    SUBS r0, #1 ; Decrement r0, update flags
    BNE Loop
    MOV r0, r1  ;Store return value
    BX LR
    .end
```

# Inline Assembler

- Using __asm is syntactically performed as a call to a function named __asm, with one string constant argument:

```
__asm("assembler text");
```

- Examples:

```
__asm("    MOV r0, #0");  // Inline Assembly Example

__asm("STR: .byte \"abcd\"");  // Inline Assembly Example
```

- The __asm statement does not provide any way to refer to local variables. If your assembly code needs to refer to local variables, you will need to write the entire function in assembly code.