Interfacing C and TMS320C6713 Assembly Language (Part II)

ABDULLAH A. WARDAK,
Southampton Solent University, Southampton,
UNITED KINGDOM,

Abstract— In this paper, an interfacing of C and the assembly language of TMS320C6713 is presented. Similarly, interfacing of C with the assembly language of Motorola 68020 (MC68020) microprocessor is also presented for comparison. However, it should be noted that the way the C compiler passes arguments from the main function in C to the TMS320C6713 assembly language subroutine is totally different from the way the C compiler passes arguments in a conventional microprocessor such as MC68020. Therefore, it is very important for a user of the TMS320C6713-based system to properly understand and follow the register conventions and stack operation when interfacing C with the TMS320C6713 assembly language subroutine. This paper describes the application of special registers and stack in the interfacing of these programming languages. Working examples of C and their implementation in the TMS320C6713 assembly language are described in detail. Finally, the concept presented in this paper has been tested extensively by examining different examples under various conditions and has proved highly reliable in operation.

Keywords—Interfacing, high-level language, aseembly language.

Received: March 31, 2021. Revised: April 21, 2021. Accepted: April 23, 2021. Published: April 27, 2021.

1. Introduction

In many real-time applications, execution-time is crucially important. In some applications, the use of a high-level programming language cannot satisfy the application requirements, and therefore, assembly language programming becomes necessary. To achieve the real-time requirements, algorithms need to be initially developed and tested using a high-level language and then most of the time-consuming and highly-repetitive processing functions may be implemented in assembly language, which can then be called from within the high-level language program [1-3].

The way in which compilers pass arguments from a main function in C to the assembly language subroutine in a particular micro-based system varies from one system to another [1-6]. Therefore, thorough understanding of how compilers pass arguments among various functions in a particular system plays an important role in interfacing high-level and assembly language. In many micro-based systems, the most efficient way of passing arguments among various functions is through stack [2,3]. However, the way the C compiler passes arguments from the main function in C to a TMS320C6713 assembly language subroutine is totally different from the way the C compiler passes arguments in a conventional microprocessor such as: MC68020 [1-6]. Hence, it is very important for a user of a TMS320C6713-based system to properly understand and follow the register conventions when interfacing C with the TMS320C6713 assembly language subroutine.

2. Interfacing C and MC68020 Assembly

Stack of the MC68020 microprocessor plays an important role in interfacing C and MC68020 assembly language subroutines. The MC68020 stack is used as a tool for passing various arguments from the main function in C to the MC68020 assembly language subroutine. Stack pointer of the MC68020 microprocessor (A7) always points to the last item pushed onto the stack. When an argument is pushed onto the MC68020 stack, the stack pointer is pre-decremented by the size of the arguments and then the arguments is pushed onto the stack; and when an argument is popped off the stack, the stack pointer is then post-incremented by the size of the argument. Fig. 1a describes the manner in which the C compiler pushes the arguments onto the MC68020 stack.

In Example 1, the C function (asmf) is translated into MC68020 assembly language subroutine as shown in Fig. 1b.

Example 1

```
asmf (int  a, int  b, int  *c)
{
    a = a + b;
    b = b + a;
   *c = *c + b;
}
```

```
#include <stdio.h>
extern   asmf ( );
main( )
{
    int  i,j,k;
    i=5;
    j=6;
    k=8;
    asmf(i, j, &k);
}
```

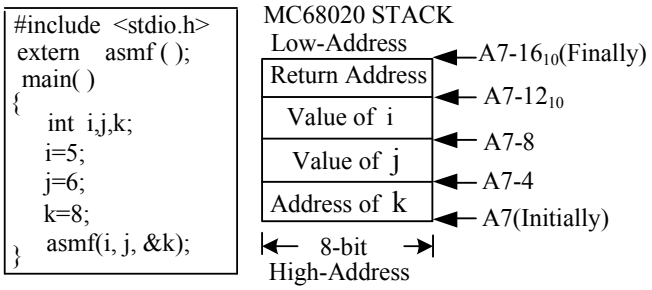


Fig.1a Show how the C compiler places arguments on the MC68020 stack

```
        text        2
        global      asmf
asmf:   MOVE.L  4(A7),D0    ;D0=value of i=a=5
        MOVE.L  8(A7),D1    ;D1=value of j=b=6
        MOVE.L  12(A7),A0   ;A0= Address of k
        MOVE.L  (A0),D2     ;D2=k=*c=8
        ADD.L   D1,D0       ;D0=a=a+b=11
        ADD.L   D0,D1       ;D1=b=b+a=17
        ADD.L   D1,D2       ;D2=*c=*c+b=25
        MOVE.L  D2,(A0)     ;k=*c is pushed back
        RTS                 ;Return from subroutine
```

Fig.1b Implementation of the C function (asmf) into MC68020 assembly

3. Interfacing C and TMS320C6713 Assembly

The C compiler passes arguments from the main function in C into TMS320C6713 assembly language subroutine using **THREE** different techniques. In the case of pure C programming, the users of the TMS320C6713-based system do not need to know how the C compiler passes arguments among various C functions. However, in the case of interfacing C with the TMS320C6713 assembly language subroutine, it is vitally important for a user to understand how the C compiler passes the arguments from a C function into a TMS320C6713 assembly language subroutine. For more information regarding the TMS320C6713 digital signal processor, refer [6,11-14].

3.1 Passing Arguments Through Registers Only

In this method, the C compiler places the arguments inside special registers in a particular manner. The user of the TMS320C6713-based system needs to be aware of this fact and use it correctly when interfacing C with the TMS320C6713 assembly language. This method is presented comprehensively in [1].

In the following sections, the second method (passing arguments through registers and stack**)** is presented in detail.

3.2 Passing Arguments Through Registers and Stack

In this case, the C compiler places arguments in designated registers and also on the stack of the TMS320C6713-based system. An argument that is not declared in the prototype function and whose size is less than the size of **integer** is passed as an **int**. An argument that is a **float** is passed as double if it has no prototype declared. A structure argument is passed as the address of the structure. For a function declared with an **ellipsis** indicating that it is called with varying numbers of arguments, the convention is slightly modified. The last explicitly declared argument is placed on the stack, so that its stack address can act as a reference for accessing the undeclared arguments.

In Example 2, the C compiler places the value of **i** in register A4 and the values of **j, k** and the address of **m** on the stack (see Fig. 2a). As can be seen from the stack-layout, each of the arguments (value of **j**, value of **k** and the address of **m**) occupy 4 bytes on the stack, starting with the first left argument (address of **m** in this case). The difference between the use of parenthesis and the use of square bracket should be noted in the implementation of the assembly language subroutine (see Fig. 2b).

The run-time stack grows from the high addresses to the low addresses as shown in Fig. 2a. The C compiler uses register **B15** as a stack pointer **(SP)** to manage the stack and it points to the next unused location on the stack. Note also, that during push, the stack pointer is post-decremented and during pull, the stack pointer is pre-incremented [4].

As shown in **example-3**, the first three arguments (the values of **i**, **j**, **k**) are placed in registers **A4**, **B4** and **A6** respectively; and the remaining three arguments (address of **m,** values of **n** and **p)** are placed on the stack (see Fig. 3a). The use of ***ellipsis*** in the ***prototype*** function (Fig. 3a) indicates that the C Compiler will certainly place some arguments onto the TMS320C6713 stack. As shown in the ***prototype***, the last explicitly declared argument will be the start of the arguments which the C compiler will place onto the stack. In this case, the first 3 arguments will be placed in A4, B4 and A6 respectively; and the remaining 3 arguments will be placed onto the stack (see Fig. 3a).

For better understanding, the C function (**asmf**) is converted into the TMS320C6713 assembly language subroutine as shown in Fig. 3b. The return address to the calling function is placed in **B3** and for this reason a branch to **B3** needs to be performed at the end of the assembly language subroutine. It is worth mentioning that the way the C compiler passes arguments from the calling function to the called function in the TMS320C6713-based environment is totally different from the way the C compiler passes arguments in a conventional microprocessor such as MC68020 [1-3]. It should be noted that this example gives the same correct result when the TMS320C6713 DSK board is operated either in little-endian or in big-endian mode.

In example-4, the floating-point values of **x** and **y** are placed in registers **A4** and **B4** respectively; while the floating-point value of **m** and the address of **z** are placed on the stack (see Fig. 4a). Four arguments are passed to the C function **(asmf)** and only the types of three arguments are explicitly declared in the ***prototype*** function, therefore, the C compiler places the last two arguments on the stack as shown in Fig. 4a. The C function (**asmf**) is converted into the TMS320C6713 assembly language subroutine as shown in Fig. 4b. It should be noted that this example works correctly and produces the correct result in both little-endian and big-endian mode of the TMS320C6713 DSK board.

Example-5 demonstrates how the C compiler places the floating-point values of the arguments **x** and **y** in registers **A4** and **B4** respectively; and places the remaining arguments on the stack. It should be noted that the floating-point value of **z** occupy 4 bytes on the stack and this is because the type of **z** is explicitly declared in the ***prototype*** function; however, the floating-point value of **m** occupy 8 bytes on the stack and it is stored as 32-LSB/32-MSB as shown on the stack. The address

of **z** occupy 4 bytes as it is the address. Appropriate TMS320C6713 assembly language instructions such as single-precision are used for floating-point data manipulation. The conversion of the C function (**asmf**) into the TMS320C6713 assembly language is presented in Fig. 5b. It should be noted that in this example, the endianness of the TMS320C6713 DSK board also does not matter.

In example-6, the C compiler places the floating-point values of the arguments **x** and **y** in registers **A4** and **B4** respectively; and places the remaining arguments on the stack. It should be noted that the floating-point value of **z** occupy 4 bytes on the stack and this is because the type of **z** is explicitly declared in the ***prototype*** function; however, **m** occupy 8 bytes on the stack and it is stored as 32-MSB/32-LSB as shown on the stack. The address of **z** occupy 4 bytes as it is the address. Appropriate TMS320C6713 assembly language instructions such as single-precision are used for floating-point data manipulation. The conversion of the C function (**asmf**) into the TMS320C6713 assembly language is presented in Fig. 5b. It should be noted that in this example, the endianness of the TMS320C6713 DSK board also does not matter.

In example-7, the address of the double-precision values of the arguments **n** and the double-precision value of **y** are placed in register **A4** and in register pair **B5:B4** respectively; while the double-precision values of **y**, **z** and **m** are placed onto the stack (see Fig. 7a). Appropriate assembly language instructions such as double-precision addition (ADDDP) and double-precision load (LDDW) are employed for data manipulation. The reader needs to pay attention to the way the final double-precision value of **n** is stored into the memory when the TMS320C6713 board is operated in the little-endian mode (see Fig. 7b).

Example-8 demonstrates the stack layout and the implementation of **asmf** function in big-endian mode. It should be noted that there are different layouts of the TMS320C6713 stack in little-endian and in big-endian modes. The reader needs to pay attention to the way the final double-precision value of **z** is stored into the memory when the TMS320C6713 board is operated in big-endian mode. Thorough comparison of examples 7 and 8 will clarify the difference using the two modes of the board.

In **example-9**, the long values of the arguments **x** and **y** are placed as 64-bits in register pairs A5:A4 and B5:B4 respectively; and the long values of **z** and **m** and the address of **n** is placed on the stack as shown in Fig. 9a. Appropriate assembly language instructions are employed for data manipulation. The reader is encouraged to pay lots of attention to the implementation of the C function (asmf) into the TMS320C6713 assembly language as shown in Fig. 9b, especially to the way the final long value of **z** is stored in the memory in little-endian mode.

Finally, **example-10**, implements the C function (**asmf**) into TMS320C6713 assembly language. It should be noted that there are different layouts of the TMS320C6713 stack in little-endian and in big-endian modes. The reader needs to pay attention to the way the final long-value of **n** is stored into the memory when the TMS320C6713 board is operated in big-endian mode. Thorough comparison of examples 9 and 10 will highlight the difference using the two modes of the board.

Example 2

```
asmf (int  a, int  b, int  c, int *d)
{
   a = a + b;
   b = b + a;
   c = c + b;
  *d = *d + c;
}
```

```
extern  asmf (int , int , ...);
main( )
{
  int i, j, k,m;
  i=8;
  j=10;
  k=12;
  m=20;
  asmf(i,j,k,&m);
}
```

Prototype

A4
stack
stack
stack

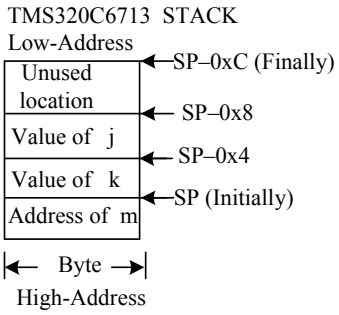
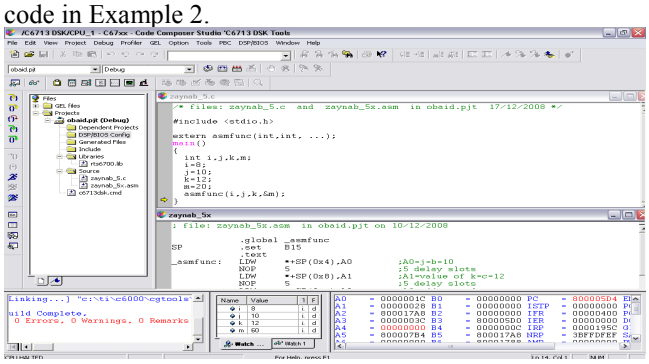


Fig.2a Shows how the C compiler places arguments in registers and on C6713 stack.

```
                .global   _asmf
SP              .set      B15
                .text
_asmf:  LDW     *+SP(0x4),A0   ;A0=j=b=10
        NOP     5              ;5 delay slots
        LDW     *+SP(0x8),A1   ;A1=Value  of k=c=12
        NOP     5              ;5 delay slots
        LDW     *+SP(0xC),A2   ;A2=Address  of m
        NOP     5              ;5 delay slots
        LDW     *A2,A3         ;A3=m=*d=20
        NOP     5
        ADD.D1  A4,A0,A4       ;A4=a=a+b=8+10=18
        NOP     5
        ADD.D1  A0,A4,A0       ;A0=b=b+a=10+18=28
        NOP     5
        ADD.D1  A1,A0,A1       ;A1=c=c+b=12+28=40
        NOP     5
        ADD.D1  A3,A1,A3       ;A3=m=*d=*d+c=20+40=60
        NOP     5
        STW.D1  A3,*A2         ;Store the final value of m
        NOP     5
        B       B3             ;return to the calling function
        NOP     5              ;5 delay slots for branch
```

Fig.2b Translation of the above C function (**asmf**) into C6713 assembly language.

Following is the screen-shot of the CCS **after** running the code in Example 2.



Example 3

```
asmf (int  a, int  b, int  c, int *d, int e, int  f)
{
      a = a + b;
      b = b + a;
      c = c + b;
     *d = *d + c;
      e = e + d;
      f = f + e;
}
```

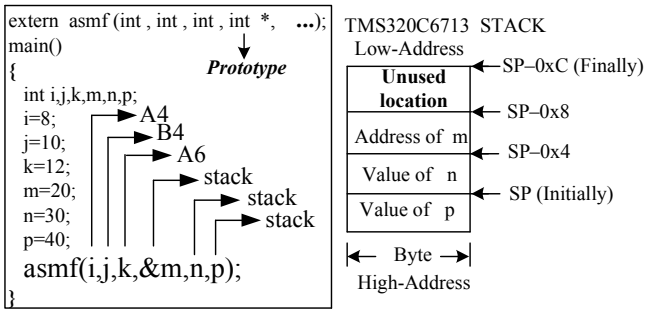


Fig.3a Shows how the C compiler places arguments in registers and onto TMS320C6713 stack.

```
                    .global  _asmf
SP                  .set     B15
                    .text
_asmf:  LDW         *+SP(0x4),A0    ;A0=Address of m
        NOP         5               ;5 delay slots
        LDW         *A0,A1          ;A1=m=d=20
        NOP         5               ;5 delay slots
        LDW         *+SP(0x8),A2    ;A2=value of n=e=30
        NOP         5               ;5 delay slots
        LDW         *+SP(0xC),A3    ;A3=Value  of p=f=40
        NOP         5               ;5 delay slots
        ADD.L1X     A4,B4,A4        ;A4=a=a+b=8+10=18
        NOP         5               ;5 delay slots
        ADD.L2X     B4,A4,B4        ;B4=b=b+a=10+18=28
        NOP         5
        ADD.L1X     A6,B4,A6        ;A6=k=c=c+b=12+28=40
        NOP         5               ;5 delay slots
        ADD.D1      A1,A6,A1        ;A1=d=d+c=20+40=60
        NOP         5
        ADD.D1      A2,A1,A2        ;A2=n=e=e+d=30+60=90
        NOP         5
        ADD.D1      A3,A2,A3        ;A3=p=f=f+e=40+90=130
        NOP         5
        STW.D1      A1,*A0          ;Store the final value of m=60
        NOP         5
        B           B3              ;return to the calling function
        NOP         5               ;5 delay slots for branch
```

Fig.3b Translation of the above C function (**asmf**) into C6713 assembly language.

Example 4

```
asmf (int  a, int  b, int  c, int *d)
{
    a = a + b;
    b = b + a;
    c = c + b;
   *d = *d + c;
}
```

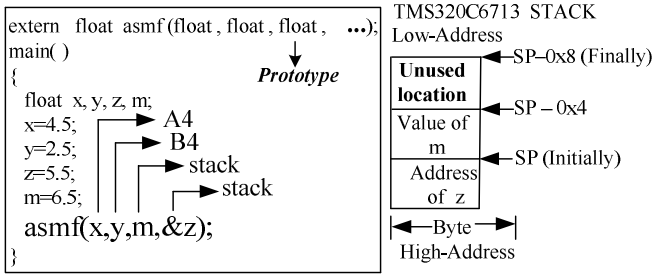
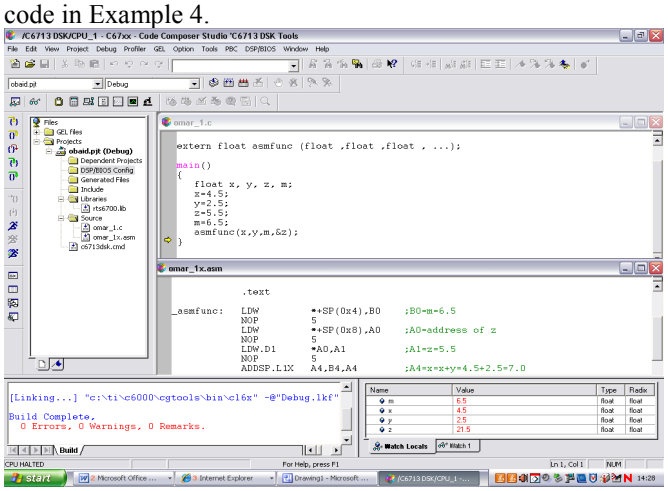


Fig.4a Shows how the C compiler places arguments in registers and onto C6713 stack.

```
                    .global  _asmf
SP                  .set     B15
                    .text
_asmf:  LDW         *+SP(0x4),B0    ;B0=m=6.5
        NOP         5               ;5 delay slots
        LDW         *+SP(0x8),A0    ;A0=Address of z
        NOP         5               ;5 delay slots
        LDW.D1      *A0,A1          ;A1=z=5.5
        NOP         5               ;5 delay slots
        ADDSP.L1X   A4,B4,A4        ;A4=x=x+y=4.5+2.5=7.0
        NOP         5               ;5 delay slots
        ADDSP.L2X   B4,A4,B4        ;B4=y=y+x=7+2.5=9.5
        NOP         5
        ADDSP.L2    B0,B4,B0        ;B0=m=m+y=6.5+9.5=16.0
        NOP         5
        ADDSP.L1X   A1,B0,A1        ;A1=z=z+m=5.5+16=21.5
        NOP         5
        STW.D1      A1,*A0          ;Store the final value of z
        NOP         5
        B           B3              ;return to the calling function
        NOP         5               ;5 delay slots for branch
```

Fig.4b Translation of the above C function (**asmf**) into C6713 assembly language.

Following is the screen-shot of the CCS **after** running the code in Example 4.



Example 5

```
asmf (float  a, float  b, float  c, float  d, float *e)
{
    a = a + b;
    b = b + a;
    c = c + b;
   d = d + c;
   *e = *e + d;
}
```

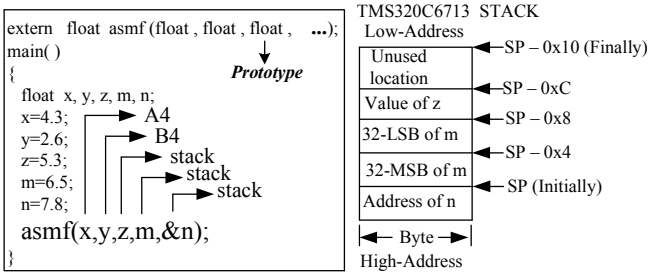


Fig.5a Indicates how the C compiler places arguments in registers and onto TMS320C6713 stack.

```
SP          .global  _asmf
            .set     B15                          N.B (Little-Endian)
            .text
_asmf:  LDW         *+SP(0x4),B0      ;B0=z=5.3
        NOP         5                 ;5 delay slots
        LDW         *+SP(0x8),A0      ;A0=32-LSB of m
        NOP         5                 ;5 delay slots
        LDW         *+SP(0xC),A1      ;A1=32-MSB of m
        NOP         5                 ;5 delay slots
        LDW         *+SP(0x10),A2     ;A2=Address of n
        NOP         5                 ;5 delay slots
        MV          A2,A5             ;Save address of n
        NOP         5
        LDW.D1      *A2,A3            ;A3=n=7.8
        NOP         5
        ADDSP.L1X   A4,B4,A4          ;A4=x=x+y=4.3+2.6=6.9
        NOP         7                 ;8 delay slots
        ADDSP.L2X   B4,A4,B4          ;B4=y=y+x=2.6+6.9=9.5
        NOP         7
        ADDSP.L2    B0,B4,B0          ;B0=z=z+y=5.3+9.5=14.8
        NOP         7
        SPDP.S2     B0,B1:B0          ;B1:B0=z=5.3
        NOP         7
        ADDDP.L1X   A1:A0,B1:B0,A1:A0  ;A1:A0=m=m+z=6.5+14.8=21.3
        NOP         8
        DPSP.L2     B1:B0,B0          ;B0=z=5.3
        NOP         7
        SPDP.S1     A3,A3:A2          ;A3:A2=z=5.3
        NOP         7
        ADDDP.L1    A3:A2,A1:A0,A3:A2  ;A3:A2=n=n+m=7.8+21.3=29.1
        NOP         8
        DPSP.L1     A3:A2,A2          ;A2=n=29.1
        NOP         7
        STW.D1      A2,*A5            ;Store the final value of n
        NOP         5
        B           B3                ;return to the calling function
        NOP         5                 ;5 delay slots for branch
```

Fig.5b Translation of the above C function (**asmf**) into TMS320C6713 assembly language.

Example 6

```
asmf (float  a, float  b, float  c, float  d, float *e)
{
    a = a + b;
    b = b + a;
    c = c + b;
   d = d + c;
   *e = *e + d;
}
```

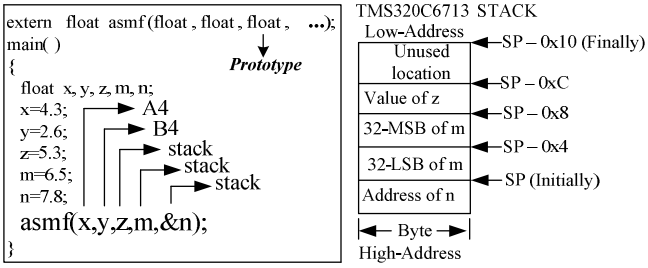


Fig.6a Describes how the C compiler places arguments in registers and onto TMS320C6713 stack.

```
SP          .global  _asmf
            .set     B15                          N.B (Big-Endian)
            .text
_asmf:  LDW         *+SP(0x4),B0      ;B0=z=5.3
        NOP         5                 ;5 delay slots
        LDW         *+SP(0x8),A0      ;A0=32-LSB of m
        NOP         5                 ;5 delay slots
        LDW         *+SP(0xC),A1      ;A1=32-MSB of m
        NOP         5                 ;5 delay slots
        LDW         *+SP(0x10),A2     ;A2=Address of n
        NOP         5                 ;5 delay slots
        MV          A2,A5             ;Save address of n
        NOP         5
        LDW.D1      *A2,A3            ;A3=n=7.8
        NOP         5
        ADDSP.L1X   A4,B4,A4          ;A4=x=x+y=4.3+2.6=6.9
        NOP         7                 ;8 delay slots
        ADDSP.L2X   B4,A4,B4          ;B4=y=y+x=2.6+6.9=9.5
        NOP         7
        ADDSP.L2    B0,B4,B0          ;B0=z=z+y=5.3+9.5=14.8
        NOP         7
        SPDP.S2     B0,B1:B0          ;B1:B0=z=5.3
        NOP         7
        ADDDP.L1X   A1:A0,B1:B0,A1:A0  ;A1:A0=m=m+z=6.5+14.8=21.3
        NOP         8
        DPSP.L2     B1:B0,B0          ;B0=z=5.3
        NOP         7
        SPDP.S1     A3,A3:A2          ;A3:A2=z=5.3
        NOP         7
        ADDDP.L1    A3:A2,A1:A0,A3:A2  ;A3:A2=n=n+m=7.8+21.3=29.1
        NOP         8
        DPSP.L1     A3:A2,A2          ;A2=n=29.1
        NOP         7
        STW.D1      A2,*A5            ;Store the final value of n
        NOP         5
        B           B3                ;return to the calling function
        NOP         5                 ;5 delay slots for branch
```

Fig.6b Implementation of the above C function (asmf) into TMS320C6713 assembly language.

Example 7

```
asmfunc (double *a, double b, double c, double d, double e)
{
  b = b + c;
  c = c + b;
  d = d + c;
  e = e + d;
 *a = *a + e;
}
```

```
extern double asmf (double *, double , double , ...);
main()                                   Prototype
{
  double x, y, z, m, n;
  x=4.3;   → A4
  y=2.6;   → B5:B4
  z=5.3;   → stack
  m=6.5;   → stack
  n=7.8;   → stack
  asmf(&n,x,y,z,m);
}
```

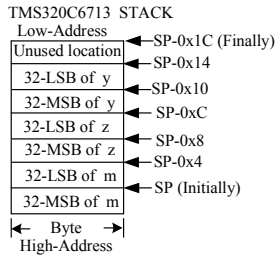
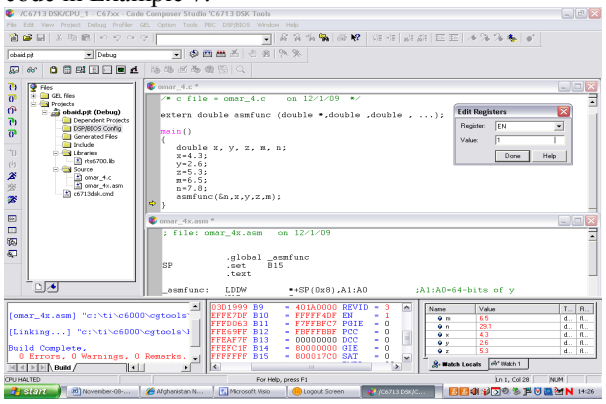


Fig.7a Describes how the C compiler places arguments in registers and onto TMS320C6713 stack

```
                    .global  _asmf
SP                  .set     B15                    N.B (Little Endian)
                    .text
_asmf:  LDDW                 *+SP(0x8),A1:A0    ;A1:A0=64-bits of
        NOP                  7
        LDDW                 *+SP(0x10),A3:A2   ;A3:A2=64-bits of z
        NOP                  7
        LDDW                 *+SP(0x18),A7:A6   ;A7:A6=64-bits of m
        NOP                  7
        LDDW                 *A4,A9:A8          ;A9:A8=n=7.8
        NOP                  7
        ADDDP.L2X            B5:B4,A1:A0,B5:B4  ;B5:B4=x=x+y=4.3+2.6=6.9
        NOP                  7
        ADDDP.L1X            A1:A0,B5:B4,A1:A0  ;A1:A0=y=y+x=2.6+6.9=9.5
        NOP                  7
        ADDDP.L1             A3:A2,A1:A0,A3:A2  ;A3:A2=z=z+y=5.3+9.5=14.8
        NOP                  7
        ADDDP.L1             A7:A6,A3:A2,A7:A6  ;A7:A6=m=m+z=6.5+14.8=21.3
        NOP                  8
        ADDDP.L1             A9:A8,A7:A6,A9:A8  ;A9:A8=n=n+m=7.8+21.3=29.1
        NOP                  8
        STW.D1               A8,*A4++           ;Store 32-LSB of final value of n
        NOP                  5
        STW.D1               A9,*A4             ;Store 32-MSB of final value of n
        NOP                  5
        B                    B3                 ;return from func to addr in B3
        NOP                  5                  ;5 delay slots for branch
```

Fig.7b Implementation of the above C function (**asmf**) into TMS320C6713 assembly language

Following is the screen-shot of the CCS **after** running the code in Example 7.



Example 8

```
asmfunc (double *a, double b, double c, double d, double e)
{
  b = b + c;
  c = c + b;
  d = d + c;
  e = e + d;
 *a = *a + e;
}
```

```
extern double asmf (double *, double , double , ...);
main()                                   Prototype
{
  double x, y, z, m, n;
  x=4.3;   → A4
  y=2.6;   → B5:B4
  z=5.3;   → stack
  m=6.5;   → stack
  n=7.8;   → stack
  asmf(&n,x,y,z,m);
}
```

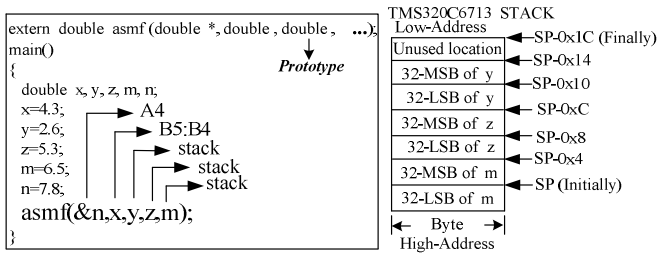


Fig.8a Describes how the C compiler places arguments in registers and onto TMS320C6713 stack.

```
                    .global  _asmf
SP                  .set     B15                    N.B (Big Endian)
                    .text
_asmf:  LDDW                 *+SP(0x8),A1:A0    ;A1:A0=64-bits of
        NOP                  7
        LDDW                 *+SP(0x10),A3:A2   ;A3:A2=64-bits of z
        NOP                  7
        LDDW                 *+SP(0x18),A7:A6   ;A7:A6=64-bits of m
        NOP                  7
        LDDW                 *A4,A9:A8          ;A9:A8=n=7.8
        NOP                  7
        ADDDP.L2X            B5:B4,A1:A0,B5:B4  ;B5:B4=x=x+y=4.3+2.6=6.9
        NOP                  7
        ADDDP.L1X            A1:A0,B5:B4,A1:A0  ;A1:A0=y=y+x=2.6+6.9=9.5
        NOP                  7
        ADDDP.L1             A3:A2,A1:A0,A3:A2  ;A3:A2=z=z+y=5.3+9.5=14.8
        NOP                  7
        ADDDP.L1             A7:A6,A3:A2,A7:A6  ;A7:A6=m=m+z=6.5+14.8=21.3
        NOP                  8
        ADDDP.L1             A9:A8,A7:A6,A9:A8  ;A9:A8=n=n+m=7.8+21.3=29.1
        NOP                  8
        STW.D1               A9,*A4++           ;Store 32-MSB of final value of n
        NOP                  5
        STW.D1               A8,*A4             ;Store 32-LSB of final value of n
        NOP                  5
        B                    B3                 ;return from func to addr in B3
        NOP                  5                  ;5 delay slots for branch
```

Fig.8b Translation of the above C function (**asmf**) into TMS320C6713 assembly language.

Example 9

```
asmf (long a, long b, long c, long d, long *e)
{
  a = a + b;
  b = b + a;
  c = c + b;
  d = d + c;
 *e = *e + d;
}
```

```
extern long asmf (long , long , long , ...);
main()                                   Prototype
{
  long x, y, z, m, n;
  x=4;   → A5:A4
  y=2;   → B5:B4
  z=5;   → stack
  m=6;   → stack
  n=7;   → stack
  asmf(x,y,z,m,&n);
}
```

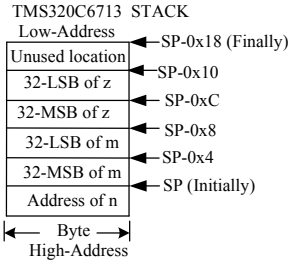


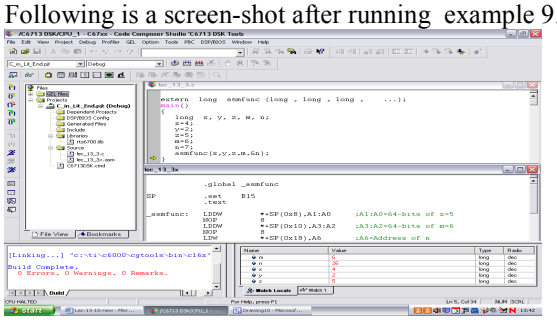
Fig.9a Shows how the C compiler places arguments in registers and onto the C6713 stack.

```
                    .global  _asmfunc                    N.B (Little Endian)
SP                  .set     B15
                    .text

_asmf:  LDDW        *+SP(0x8),A1:A0     ;A1:A0=64-bits of z=5
        NOP         8
        LDDW        *+SP(0x10),A3:A2    ;A3:A2=64-bits of m=6
        NOP         8
        LDW         *+SP(0x18),A6               ;A6=Address of n
        NOP         5
        MV          A6,A8
        NOP         5
        LDDW        *A6,A7:A6           ;A7:A6=64-bits of n=7
        NOP         7
        ADD.L1X     B4,A5:A4,A5:A4      ;A5:A4=x=x+y=4+2=6
        NOP         7
        ADD.S1X     B5,A5,A5            ;A5:A4=x=x+y=4.3+2.6=6.9
        NOP         7
        ADD.L2X     A4,B5:B4,B5:B4              ;B5:B4=y=y+x=2+6=8
        NOP         7
        ADD.S2X     A5,B5,B5            ;B5:B4=y=y+x=2+6=8
        NOP         7
        ADD.L1X     B4,A1:A0,A1:A0              ;A1:A0=z=z+y=5+8=13
        NOP         7
        ADD.S1X     B5,A1,A1            ;A1:A0=z=z+y=5+8=13
        NOP         7
        ADD.L1      A2,A1:A0,A3:A2              ;A3:A2=m=m+z=6+13=19
        NOP         8
        ADD.S1      A3,A1,A1            ;A3:A2=m=m+z=6+13=19
        NOP         8
        ADD.L1      A2,A7:A6,A7         :A6;A7:A6=n=n+m=7+19=26
        NOP         8
        ADD.S1      A3,A7,A7            ;A7:A6=n=n+z=7+19=26
        NOP         8
        STW.D1      A6,*A8++            ;Store final value of n
        NOP         5
        STW.D1      A7,*A8                      ;Store final value of n
        NOP         5
        B           B3                  ;return from func to addr in B3
        NOP         5                    ;5 delay slots for branch
```

Fig.9b Translation of the above C function (**asmf**) into C6713 assembly language.

Following is a screen-shot after running example 9.



Example 10

```
asmf (long  a, long  b, long  c, long  d, long  *e)
{
   a = a + b;
   b = b + a;
   c = c + b;
   d = d + c;
   *e = *e + d;
}
```

```
extern  long  asmf (long , long , long ,  ...);
main()
{                                          Prototype
  long  x, y, z, m, n;
  x=4;                    A5:A4
  y=2;                    B5:B4
  z=5;                    stack
  m=6;                    stack
  n=7;                    stack
  asmf(x,y,z,m,&n);
}
```

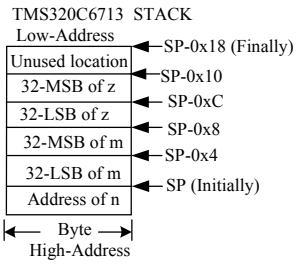


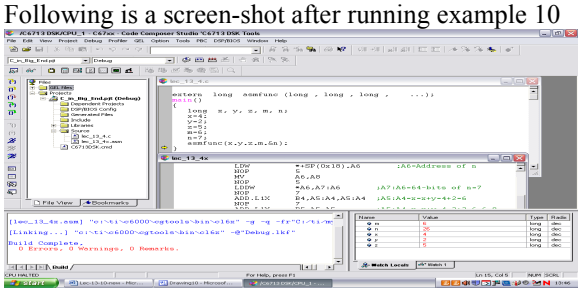
Fig.10a Shows how the C compiler places arguments in registers and onto C6713 stack.

```
                    .global  _asmf                   N.B (Big Endian)
SP                  .set     B15
                    .text

_asmf:  LDDW        *+SP(0x8),A1:A0     ;A1:A0=64-bits of z=5
        NOP         8
        LDDW        *+SP(0x10),A3:A2    ;A3:A2=64-bits of m=6
        NOP         8
        LDW         *+SP(0x18),A6               ;A6=Address of n
        NOP         5
        MV          A6,A8
        NOP         5
        LDDW        *A6,A7:A6           ;A7:A6=64-bits of n=7
        NOP         7
        ADD.L1X     B4,A5:A4,A5:A4      ;A5:A4=x=x+y=4+2=6
        NOP         7
        ADD.S1X     B5,A5,A5            ;A5:A4=x=x+y=4.3+2.6=6.9
        NOP         7
        ADD.L2X     A4,B5:B4,B5:B4              ;B5:B4=y=y+x=2+6=8
        NOP         7
        ADD.S2X     A5,B5,B5            ;B5:B4=y=y+x=2+6=8
        NOP         7
        ADD.L1X     B4,A1:A0,A1:A0              ;A1:A0=z=z+y=5+8=13
        NOP         7
        ADD.S1X     B5,A1,A1            ;A1:A0=z=z+y=5+8=13
        NOP         7
        ADD.L1      A2,A1:A0,A3:A2              ;A3:A2=m=m+z=6+13=19
        NOP         8
        ADD.S1      A3,A1,A1            ;A3:A2=m=m+z=6+13=19
        NOP         8
        ADD.L1      A2,A7:A6,A7         :A6;A7:A6=n=n+m=7+19=26
        NOP         8
        ADD.S1      A3,A7,A7            ;A7:A6=n=n+z=7+19=26
        NOP         8
        STW.D1      A7,*A8++            ;Store final value of n
        NOP         5
        STW.D1      A6,*A8                      ;Store final value of n
        NOP         5
        B           B3                  ;return from func to addr in B3
        NOP         5                    ;5 delay slots for branch
```

Fig.10b Translation of the above C function (**asmf**) into C6713 assembly language.

Following is a screen-shot after running example 10



4. Conclusions

The concept of interfacing C with the TMS320C6713 assembly language has been fully described. The concept presented in this paper will be essential and of great interest to many users who are employing a micro-based system for their applications; and especially for those users who want to use the TMS320C6713-based system for assembly language programming and signal processing.

It is strongly recommended to the users of the TMS320C6713-based systems to properly understand and follow the register conventions and the use of C6713 stack when interfacing C with the TMS320C6713 assembly language subroutine.

The presented software and concept have been tested thoroughly by examining different types of examples under various conditions and has proved highly reliable in operation.

References

[1] A.A. Wardak, "Interfacing C and TMS320C6713 Assembly Language (Part-I)". CESSE 2009 : International Conference on Computer, Electrical, and Systems Science, and Engineering, January 28-30, 2009, Dubai, United Arab Emirates

[2] A. A. Wardak, G.A. King, R. Backhouse, "Interfacing high-level and assembly language with microcodes in 3-D image generation", Journal of Microprocessors and Microsystems, vol. 18, no.4, May 1994.

[3] A.A. Wardak, "Real-Time 3-D Image Generation with TMS320C30 EVM", Journal of Microcomputer Applications, vol. 18, pp. 355-373, 1995, Academic Press Limited.

[4] TMS320C6000 Optimizing C Compiler User's Guide, SPRU187K, Texas Instruments, Dallas, TX, 2002. Section 8, p. 4.

[5] Kyle A., Ashan K., "Data Movement Between Big-Endian and Little-Endian Devices". Freescale Semiconductor, Inc. Freescale Semiconductor, Inc. San Jose, CA Austin, TX, 2008

[6] R Chassaing, Digital Signal Processing and Applications with the 6713 and C6416 DSK, Wiley, New York, 2005, Chapter 1.

[7] TMS320C6000 Programmer's Guide, SPRU198G, Texas Instruments, Dallas, TX, 2002.

[8] R. Bannatyne and C. Viot, "Introduction to Microcontroller - Part 1", Wescon '98 : IEEEconference proceedings, Anaheim Convention Center, Anaheim, California, September 15-17, 1998, pp. 350-360.

[9] R. Bannatyne and C. Viot, "Introduction to Microcontroller - Part 2", Northcon '98 : IEEE conference proceedings, Washington State Convention Center, Seattle, Washington, October 21-23, 1998, pp. 250-354.

[10] S. Menhart, "Transitioning a Microcontroller Course from Assembly Language to C", *Proceedings of the 2004 American Society for Engineering Education, Midwest Section Conference*

[11] TMS320C6211 Fixed-Point Digital Signal Processor– TMS320C6711 Floating-Point Digital Signal Processor, SPRS073C, Texas Instruments, Dallas, TX, 2000.

[12] TMS320C6713 Floating Point Digital Signal Processor, Literature Number: SPRS186L -December 2001 - Revised November 2005

[13] TMS320C6000 Code Composer Studio Tutorial, SPRU301C, Texas Instruments, Dallas, TX, 2000.

[14] TMS320C6000 Assembly Language Tools, User's Guide,