Writing Assembly Code for the NetBurner ColdFire Platform

Revision 1.0
January 24, 2007
Released

Table of Contents

Introduction

So you want to write some assembly code for the NetBurner ColdFire platform. This is not meant to be a complete guide to assembly language programming for the ColdFire, but a minimal guide to get started. If you would like more information on assembly language programming for the ColdFire, then you can find references to a couple of books at the end of this document.

Basic Rules

- Parameters are passed on the stack.
- You can modify D0, D1, A0, A1, and the SR registers. Any other registers must be restored before returning.
- Return values are passed in D0.
- Functions must end with `rts`.

Parameter Passing

When a function is called, any input parameters are passed on the stack. The return address is then passed on the stack. The stack is stored in register A7.

Simple Examples

Increment Function

Suppose we want to write the following function:

```
int inc( int i )
{
   return i + 1;
}
```

In assembly, it would be written as the following:

```
.global inc
inc:
   move.l %a7@(4),%d0   // Get the first parameter
   addq.l #1,%d0
   rts
```

Subtraction Function

Suppose we have two parameters:

```
int sub( int a, int b )
{
   return a - b;
}
```

`a` is the first parameter and it will be closest on the stack, followed by `b` being the next closest, etc...

```
.global sub
sub:
   move.l %a7@(4),%d0   // Value of a
   move.l %a7@(8),%d1   // Value of b
   sub.l  %d1,%d0
   rts
```

Passing in Pointer-Type Parameters

Suppose one or more of the parameters are pointers:

```
void asmstrcpy( char *dest, const char *source );
```

```
.global asmstrcpy
asmstrcpy:
   move.l %a7@(4),%a0   // Get the first parameter
   move.l %a7@(8),%a1   // Get the second parameter
asm_loop:
   move.b %a1@+,%d0     // Load char and increment address ptr
   move.b %d0,%a0@+     // Store char and increment address ptr
   bne asm_loop
   rts
```

Accessing Global Variables

```
DWORD getsecs( void );
```

The following accesses the global variable Secs:

```
.global getsecs
.extern Secs            // Define the global variable to access
getsecs:
   move.l Secs,%d0
   rts
```

Calling Another Function

It is easy if the function being called takes no parameters:

```
.extern other_function
   jsr other_function
```

If the function being called takes parameters, then you must pass them clean up after yourself when the function returns.

Take the following test function for example:

```
void ShowParams( int P1, int P2 )
{
   iprintf( "P1 = %08x or %d   P2 = %08x or %d\r\n", P1, P1, P2, P2 );
}
```

Then call it from assembly language:

```
.global TestAsmCall
.extern ShowParams
TestAsmCall:
   move.l #1,%d0
   move.l %d0,%a7@-    // Push the second parameter on the stack
   move.l #2,%d0
   move.l %d0,%a7@-    // Push the first parameter on the stack
   jsr ShowParams
   addq.l #8,%a7       // Take the parameters back off the stack
   rts
```

Inline or as a Separate File

You have been shown some simple code, but have not yet defined how to get it into your program. You can do it in two ways: as inline assembly or as a separate assembly file.

Inline Assembly

Below is a project that defines all of the functions above as inline assembly. Note that the functions must be declared as extern C so that C++ can find them. Also note that all the functions must "live" someplace and that they are all defined in a function named `Holderfunc` (the name of this function does not matter).

```
#include "predef.h"
#include <stdio.h>
#include <ctype.h>
#include <startnet.h>
#include <autoupdate.h>
#include <dhcpclient.h>
#include <bsp.h>


extern "C" {
   void UserMain( void *pd );
}


const char *AppName = "ftest";


extern "C" {
   int inc( int i );
   int sub( int a, int b );
   void asmstrcpy( char *dest, const char *source );
   DWORD getsecs( void );
   void ShowParams( int P1, int P2 );
   void TestAsmCall();
}


void ShowParams( int P1, int P2 ) {
   iprintf( "P1 = %08x or %d   P2 = %08x or %d\r\n", P1, P1, P2, P2 );
}


void Holderfunc() {
   /* inc function */
   asm(".global inc           ");
   asm("inc:                  ");
   asm("   move.l %a7@(4),%d0 "); // Get the first parameter
   asm("   addq.l #1,%d0      ");
   asm("   rts                ");

   /* sub function */
   asm(".global sub           ");
   asm("sub:                  ");
   asm("   move.l %a7@(4),%d0 "); // Get the first parameter
   asm("   move.l %a7@(8),%d1 "); // Get the second parameter
   asm("   sub.l  %d1,%d0     ");
   asm("   rts                ");

   /* asmstrcpy function */
   asm(".global asmstrcpy     ");
   asm("asmstrcpy:            ");
   asm("   move.l %a7@(4),%a0 "); // Get the first parameter
   asm("   move.l %a7@(8),%a1 "); // Get the second parameter
   asm("asm_loop:             ");
   asm("   move.b %a1@+,%d0   "); // Load char & increment address ptr
   asm("   move.b %d0,%a0@+   "); // Store char & increment address ptr
   asm("   bne asm_loop       ");
```

```
    asm("   rts                 ");

    /* getsecs function */
    asm(".global getsecs        ");
    asm(".extern Secs           "); // Define global variable to access
    asm("getsecs:               ");
    asm("   move.l Secs,%d0     ");
    asm("   rts                 ");

    /* TestAsmCall function */
    asm(".global TestAsmCall    ");
    asm(".extern ShowParams     ");
    asm("TestAsmCall:           ");
    asm("   move.l #1,%d0       ");
    asm("   move.l %d0,%a7@-    "); // Push 2nd parameter on the stack
    asm("   move.l #2,%d0       ");
    asm("   move.l %d0,%a7@-    "); // Push 1st parameter on the stack
    asm("   jsr ShowParams      ");
    asm("   addq.l #8,%a7       "); // Take parameters back off the stack
    asm("   rts                 ");
}


void UserMain( void *pd ) {
    // Set up the TCP/IP stack buffers, etc...
    InitializeStack();

    // Get a DHCP address if needed. You may want to add a check for the
    // return value from this function. See the function definition in
    // \Nburn\include\dhcpclient.h.
    GetDHCPAddressIfNecessary();

    // Change our priority from highest to something in the middle.
    OSChangePrio( MAIN_PRIO );

    // Enable the ability to update code over the network.
    EnableAutoUpdate();

    iprintf( "Application started\n" );

    char buffer[20];

    while ( 1 )
    {
        OSTimeDly( 20 );
        iprintf( "inc(25) = %ld\r\n", inc( 25 ) );
        iprintf( "sub(5,4) = %ld\r\n", sub( 5, 4 ) );
        asmstrcpy( buffer, "Test" );
        iprintf( "buffer[%s]\r\n", buffer );
        iprintf( "getsecs = %ld, Secs = %ld\r\n", getsecs(), Secs );
        TestAsmCall();
    }
}
```

Separate Assembly File

To put your assembly code in a separate file, put it all in a file ending with a ".s" extension such as "myasm.s", and then add it to the ASRCS area of your makefile like this:

```
ASRCS := myasm.s
```

The "myasm.s" file:

```
/********** Start of myasm.s **********/

.text

/* inc function */
.global inc
inc:
   move.l %a7@(4),%d0   /* Get the first parameter */
   addq.l #1,%d0
   rts

/* sub function */
.global sub
sub:
   move.l %a7@(4),%d0   /* Get the first parameter */
   move.l %a7@(8),%d1   /* Get the second parameter */
   sub.l  %d1,%d0
   rts

/* asmstrcpy function */
.global asmstrcpy
asmstrcpy:
   move.l %a7@(4),%a0   /* Get the first parameter */
   move.l %a7@(8),%a1   /* Get the second parameter */
asm_loop:
   move.b %a1@+,%d0     /* Load char and increment address pointer */
   move.b %d0,%a0@+     /* Store char and increment address pointer */
   bne asm_loop
   rts

/* getsecs function */
.global getsecs
.extern Secs            /* Define the global variable to access */
getsecs:
   move.l Secs,%d0
   rts

/* TestAsmCall function */
.global TestAsmCall
.extern ShowParams
TestAsmCall:
   move.l #1,%d0
   move.l %d0,%a7@-     /* Push the second parameter on the stack */
   move.l #2,%d0
   move.l %d0,%a7@-     /* Push the first parameter on the stack */
```

```
   jsr ShowParams
   addq.l #8,%a7        /* Take the parameters back off the stack */
   rts

/********** End of myasm.s **********/
```

ColdFire Assembly Programming References

ColdFire Family Programmer's Reference Manual - Revision 3. Motorola, 8 Mar. 2005.

Ford and William Topp. Assembly Language and Systems Programming for the M68000 Family - Second Edition. Lexington: D.C. Heath and Company. 1992.