Lecture 23:
Subroutines in C

Today's Goals

- Use multiple files to write a C program
- Share variables and labels between assembly files
- Discuss how to pass parameters to C functions
- Discuss how to return values from C functions
- Call a subroutine written in assembly from a C program

Compile and Link

- Compiler
  - A compiler translate the English-like source code that human can understand into binary codes (object files) that a computer can understand.
  - When multiple source files are used, there is no cross-reference between source files while they are being compiled.
    - Some information cannot be filled such as addresses of subroutines.
- Linker
  - A linker reads the object file(s) and combines them to an executable file.
  - Uncompleted information is filled during linking process.

Sharing Labels between Files in Assembly

XDEF and XREF

- To share a label, two things must happen.
  - The file that declares the label must state to the assembler that it will be global,
  - Only one file can do so for a unique label. Any file that wants to use the predefined global label must explicitly ask for it.

- Three steps to do this
  - The file that creates the label declares it normally, i.e. by labeling a line in a subroutine, a DS.B statement, etc.
  - The file that creates the label makes it global with the
  - All other files that wish to use the label's global value link to it with

Example

File1.asm

```
            XDEF        SUB1,TEMP
TEMP        ORG         $1000
SUB1        DS.B        4
            ORG         $2800
            CLRA
            RTS
```

File2.asm

```
            XREF        SUB1
Main        ORG         $2000       ; jumps to $2800
            LDS         #$3600
            JSR         SUB1
            SWI
```

File3.asm

```
            ORG         $2900
SUB1        CLRB        $2100       ; Jumps to $2900
            RTS         #$3600
            ORG         SUB1
            LDS
            JSR
            SWI
```

Subroutines

- Return value
  - A C subroutine may return one value, or “void” if there is no return value needed.

.

- Definition / declaration
  - The subroutine must be in the file that uses it BEFORE any code that calls it.
- Prototype
  - A prototype shows
- Location
  - The subroutine itself does not need to be in the same file as the caller.

Example

```
int answer;
int myequation(int, int, int);

void main(void)
{
    int my_num = -10;
    answer = myequation(5,  my_num, 0x0007);
}

int myequation(int num1, int num2, int num3);
{
    return num1*num2+num3;
}
```

Using Assembly Subroutines in C

- Inline assembly instructions
  - `asm (“cli”); /* enable interrupt globally */`

- Assembly subroutines are often written in separate files so that inline assembly is not used.

- Basic steps for using assembly subroutines in C
  - Write the subroutine in an assembly file, such as *subfile.asm*.
  - In the assembly file, use an XDEF directive for the name of the subroutine.
  - Write the calling C program in a C file, such as *main.c.*
  - In the C file, use a one line function declaration with the same name as the assembly subroutine.

- Notes:
  - The C compiler determines how and where parameters are passed. The assembly subroutine must retrieve parameters and return the result as dictated by the C compiler.

Parameter Passing

```
int myequation(int num1, int num2, int num3);
{
    return num1*num2+num3;
}
```

- Parameter passing convention
  - Parameters are pushed into the stack.
- Parameter passing order
  - C style:

    .
  - Pascal style:

    .
  - Several variations
    - stdcall (WIN32 API)
      - A variation of Pascal style: Right to Left and the callee cleans the stack before the function call returns.

CodeWarrior

- CodeWarrior is an IDE that is developed by Freescale
- We are going to use CodeWarrior in the last lab session.

CodeWarrior’s Parameter Passing

- Pascal convention for parameter passing
- The last parameter (i.e. the rightmost) is passed by register if the parameter is four bytes or less.
- The result, if there is one, is passed in register.
- The list below shows which registers are used for this
  - One Byte: B
  - Two Bytes: D
  - Three Bytes: B:X
  - Four Bytes: D:X

Example 1

Convert signed char to signed int

```
; assembly file
        XDEF    char_s
        ORG     $2800
char_s  PSHC    ; something a C subroutine won’t do
        CLRA    ; B already contains byte to convert
        TSTB
        BPL     endsub
        LDAA    #$FF
endsub  PULC
        RTS     ; D now has signed int

/* c file */
int char_s(signed char);
char tinynum;
int shortnum;

….
shortnum = char_s(tinynum);
```

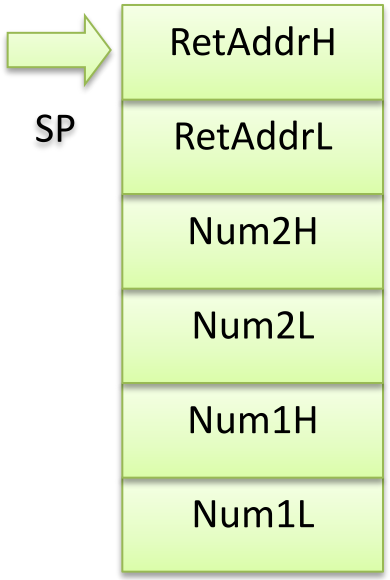
Example 2

Add two unsigned integers

Write a subroutine that adds two unsigned integers, generates the answer, and returns 0 for no overflow and 1 for overflow.
Note: the addition should be returned through a parameter (pass by reference).

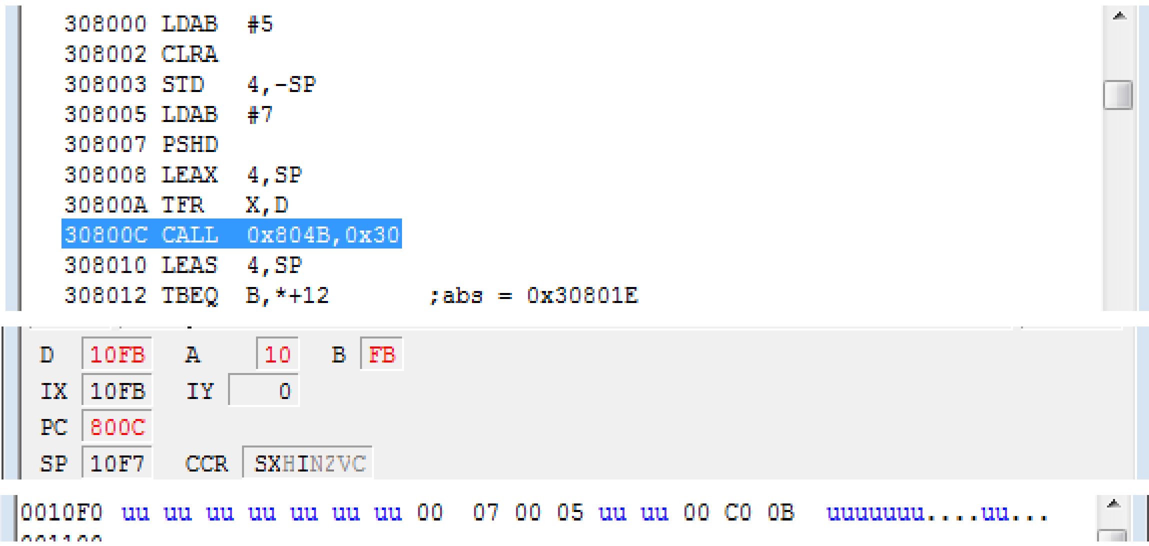
```
; ---------------------------------------------------
; assembly file
          XDEF      add_uint;
add_uint  LDY       #0        ; for the result
          TFR       D, X
          LDD       5,SP      ; load first number
          ADDD      3,SP
          BCC       skip
          LDY       #1
skip      STD       0,X
          TFR       Y,B
          RTS

/* ************************************************************ */
/* c file that calls subroutine                                 */
/* ************************************************************ */
/* passing Num1, Num2, Answer */
char add_uint(unsigned int, unsigned int, unsigned int*);
unsigned int onenum = 5;
unsigned int twonum = 7;
unsigned int answer;
…
// exits calling program if overflow is detected since in C
// 0 means false, anything else means true
if( add_uint(onenum,  twonum, &answer) return 1;
…
```



```
char add_uint(unsigned int, unsigned int, unsigned int*);
unsigned int onenum = 5;
unsigned int twonum = 7;
unsigned int answer;

if( add_uint(onenum,  twonum, &answer)
    return 1;
```

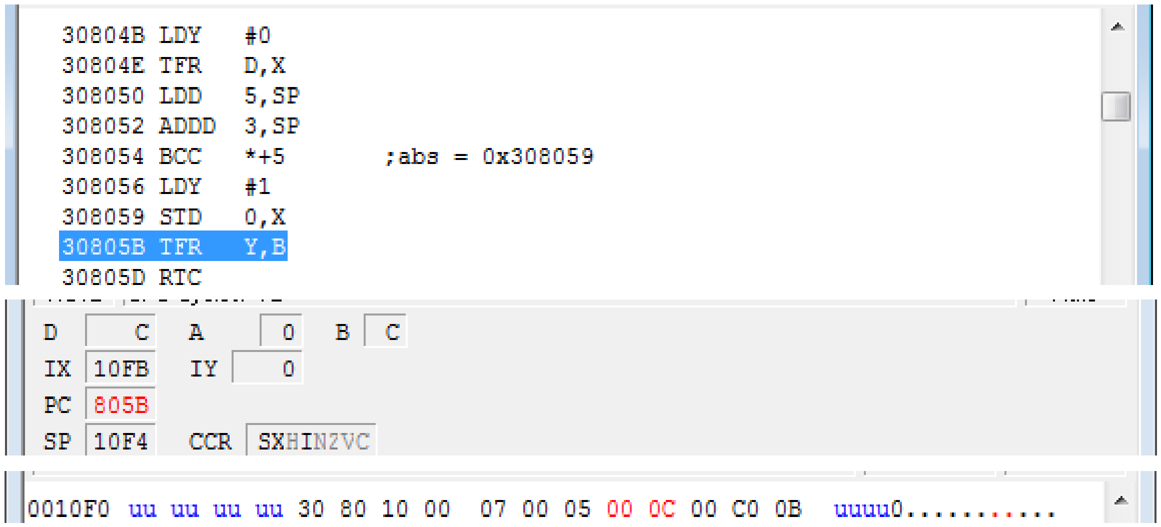


```
308000 LDAB  #5
308002 CLRA
308003 STD   4,-SP
308005 LDAB  #7
308007 PSHD
308008 LEAX  4,SP
30800A TFR   X,D
30800C CALL  0x804B,0x30
308010 LEAS  4,SP
308012 TBEQ  B,*+12       ;abs = 0x30801E
```

```
D   10FB   A     10   B  FB
IX  10FB   IY     0
PC  800C
SP  10F7   CCR  SXHINZVC
```

```
0010F0 uu uu uu uu uu uu uu 00  07 00 05 uu uu 00 C0 0B  uuuuuuu....uu...
```

```
add_uint   LDY       #0
           TFR       D, X
           LDD       5,SP
           ADDD      3,SP
           BCC       skip
           LDY       #1
skip       STD       0,X
           TFR       Y,B
           RTS
```



```
30804B LDY    #0
30804E TFR    D,X
308050 LDD    5,SP
308052 ADDD   3,SP
308054 BCC    *+5       ;abs = 0x308059
308056 LDY    #1
308059 STD    0,X
30805B TFR    Y,B
30805D RTC
```

```
D      C   A      0   B  C
IX  10FB   IY     0
PC  805B
SP  10F4   CCR  SXHINZVC
```

```
0010F0 uu uu uu uu 30 80 10 00  07 00 05 00 0C 00 C0 0B  uuuu0...........
```

Questions?

Wrap-up

What we’ve learned

What to Come