SERIAL COMMUNICATIONS ASSIGNMENT

8th of December

2011

Computer Systems Architecture
UFCEHV-20-1
Rob Williams

Sami Giacaman
UWE Student ID Number:10023253

James Johns
UWE Student ID Number:11004840

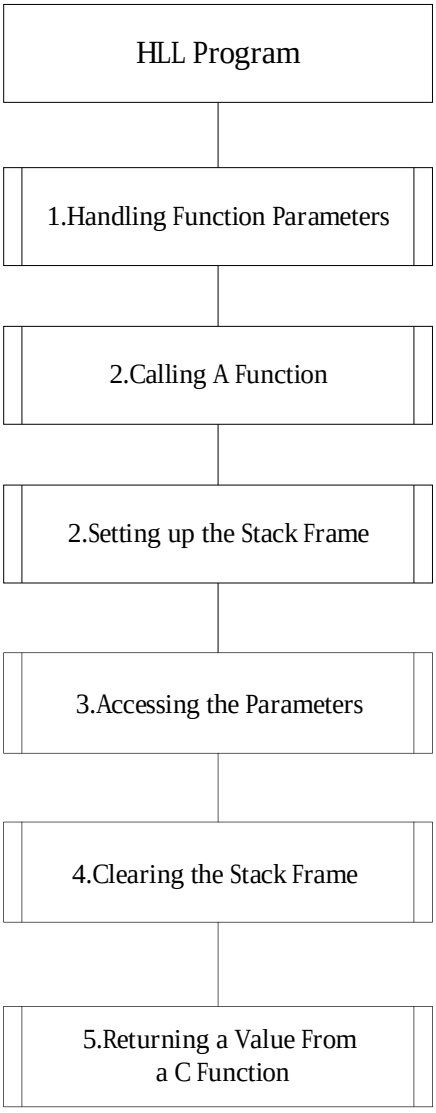
CONTENTS

REPORT

In order for our HLL Program to call an assembler coded function, using two parameters and a single return value, the implementation of the following steps in the attached diagram are required:

Structure Chart Design

For Calling an Assembler Coded Function From an HLL Program Using Two Parameters and a Single Return Value.



❖ Handling Function Parameters:

Calling an assembly function in C code is not that complicated, however we need to understand the basic mechanisms behind C function calling, which occur in Assembly.

If a C function requires parameters, they are pushed onto the stack before calling the function. They must be pushed in the reverse order of the C function definition.

E.g. To call: printf ("String: %s.", stringToPrint), stringToPrint must be pushed onto the stack first, followed by the string literal, and finally call the printf function.

The function then executes its own code, using the parameters as it needs to and returns.

❖ Calling a Function:

Whenever calling a function, the processor overwrites the current instruction pointer (EIP) register.

To prevent losing the location in memory to return to after executing the called function, EIP is pushed onto the stack. This is done automatically as part of the CALL instruction in assembly.

After storing the current EIP value, the processor then changes the EIP register to the location of the function being called and continues execution.

❖ Setting up the Stack Frame:

The first thing a function should do upon entry is to setup the stack frame. This involves pushing the EBP register onto the stack, and moving ESP into EBP. This preserves the location of the stack frame of out caller, and also enables easier access to the function parameters.

❖ Accessing the Parameters:

After setting up the stack frame, accessing parameters are simple. As the EBP register contains our original stack position after setting up the stack frame, we know that we are 8 bytes away from our parameter data. These 8 bytes contain the EIP and EBP values pushed earlier by calling the function and setting up the stack frame.

Therefore to reach the address of our parameter list, we need to add 8 to EBP. Then all we need to do is dereference this address, and we can obtain the value of the first parameter. For each subsequent parameter we simply need to add 4 to the EBP register.

E.g.:

- mov eax, [ebp+8] ; access the first parameter on the stack and store its value in eax.
- mov ebx, [ebp+12] ; access the second parameter and store it in ebx.
- mov ecx, [ebp+16] ; access the third parameter and store it in ecx.

❖ Clearing the Stack Frame:

When returning from a C function, the calling function's stack frame must be restored. To restore the stack frame, we only need to pop EBP off of the stack. This recovers the value stored earlier when the current stack frame was created. Restoring a previous stack frame removes our current stack frame.

❖ Returning a Value From a C Function:

It is optional to return a value from a C function. If the function does return a value, it is stored the EAX register before returning from the function. This limits the return value to 4 bytes (8 bytes on 64 bit CPUs). If more information is required to be returned, there are 2 ways of doing so; returning a pointer to a data structure or modifying the contents of a parameter.

Returning a pointer is bad practice, as it is not always clear if the allocated memory needs to be deallocated by the programmer using the function, or if it is deallocated by the programmer who wrote the function (or possibly library).

Modifying the contents of a parameter is more common. If a parameter of a function is a pointer, the programmer can modify the contents of memory at the address pointed to by the parameter. This is preferential, as it forces the programmer calling the function to handle all memory allocation/deallocation.

The value can then be returned from the function using the instruction RET. This instruction pops EIP off of the stack and returns it to the EIP register which then points to the instruction after the function call and execution can continue.

❖ Description of the Operation of Our Encryption Technique:

Our encryption algorithm has 2 steps; obfuscation and encryption.

The obfuscation step rotates the byte of data by 4 bits, before encrypting it. This achieves an encoding which runs equally fast on both encoding and decoding the value, preventing synchronisation issues during long transmissions.

The encryption step uses a user specified password to modify the data before transmission. When the program is run, the user is required to input a password up to 15 characters long. Any passwords longer than this are truncated to 15 characters, passwords shorter than 15 letters are left unmodified. This password is then used to encrypt the data by using the exclusive OR operation. This is achieved by stepping through the password for each byte of data being sent, and XORing one character with the data.

The benefit of an XOR encryption is that the data can be recovered using the same mask. This means that if the receiver program has had the same password input into it, then it only needs to follow the same algorithm as the encryption in order to decrypt it.

TRANSMITTER ASSEMBLER CODE

```
/*****************************************************************************************
 *
*Author: James Johns
*Purpose:  Serial communications program to send a data file encrypted over COM port.
*
 * ****************************************************************************************/
#include <stdio.h>
#include <string.h>
#include <conio.h>

#define EOT 0x80

int main(int argc, const char *argv[]) {

	char portOpenType[] = "wb";
	char fileOpenType[] = "rb";

	char portName[] = "COM1";
	char msgFileName[] = "W:/profile/j2-johns/random.txt";

	char errorMsgCOM[] = "Failed to open COM port\n";
	char errorMsgFile[] = "Could not load message file\n";
	char successMsg[] = "Transfer complete!\n";

	char passwordRequest[] = "Please enter password: ";
	char passScanString[] = "%15s";
	char password[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};    /* max 15 letter password */
	int passLength = 0;

	FILE *comport, *srcfile;

	_asm {
		lea eax, passwordRequest
		push eax
		call DWORD PTR [printf]		/* print password request */
		add esp, 4
		lea eax, password
		push eax
		lea eax, passScanString
		push eax
		call DWORD PTR[scanf]		/* scan in password string, scanf(passScanString,
						   password) */
		add esp, 4			/* scrub scan format string from stack, keep
						   password pointer on stack */
		call DWORD PTR[strlen]		/* get length of password by strlen(password) */
		add esp, 4			/* clean password pointer from stack */
		mov passLength, eax		/* save password length */

openFile:
		lea eax, fileOpenType
		push eax
		lea eax, msgFileName
		push eax
		call DWORD PTR [fopen]		/* open message source file, fopen(msgFileName,
						   fileOpenType) */
		add esp, 8			/* stack scrub */
```

```
                or eax, eax
                jz fileError                    /* if eax == 0, error opening file */
                mov srcfile, eax                /* otherwise, save file pointer */
                jmp openPort                    /* continue to open COM port */
fileError:
                lea eax, errorMsgFile
                push eax
                call DWORD PTR [printf]         /* print file error message, printf(errorMsgFile) */
                add esp, 4
                jmp end                         /* exit early */

openPort:
                lea eax, portOpenType
                push eax
                lea eax, portName
                push eax
                call DWORD PTR [fopen]          /* open COM port, fopen(portName,
                                                   portOpenType) */
                add esp, 8
                or eax, eax
                jz portError
                mov comport, eax                /* save comport file pointer */
                xor ecx, ecx                    /* clear ecx ready for loop. ecx will be our offset into
                                                   password. */
                jmp L1

portError:
                mov eax, DWORD PTR [errorMsgCOM]
                push eax
                call DWORD PTR [printf]         /* print com port error message,
                                                   printf(errorMsgCom) */
                add esp, 4
                jmp end                         /* exit early */

L1:
                push ecx
                mov eax, sf                     /* load up stack to call fgetc(sf) */
                push eax
                call DWORD PTR [fgetc]
                add esp, 4
                pop ecx

                cmp eax, EOF                /* end execution if we have hit end of file */
                je end

                ror al, 4                   /* effective nibble swap for obfuscation */
                lea ebx, password           /* get password string so we can xor with one of the
                                               characters for security */
                add ebx, ecx                /* ecx is our offset into password */
                xor al, [ebx]

                                            /* al is now encrypted */
                push ecx                    /* save ecx, as it is our current offset into password[] */
                push cp
                push eax                        /* load up stack to call fputc(c, cp) */
                call DWORD PTR [fputc]
                add esp, 8

                push cp                         /* load up stack to call fflush(cp) */
                call DWORD PTR [fflush]
                add esp, 4
                pop ecx                         /* restore ecx for modification */
```

```
                inc ecx                         /* increment through to the next character
                                                   in password */

                mov eax, passLength
                cmp ecx, eax                    /* if we have reached the end of the password */
                jge resetPassCount              /* reset to beginning of password string to endlessly
                                                   use the string */
                jmp L1                          /* repeat loop endlessly. */

                                                /* reset our offset into password by clearing ecx */
resetPassCount:
                xor ecx, ecx                    /* ecx is out offset into password. */
                jmp L1

                                                /* finish up by sending EOT, flushing and closing files */
end:
                                                /* send EOT to receiver */
                mov al, EOT
                ror al, 4                       /* effective nibble swap for obfuscation. */
                lea ebx, password               /* get password string so we can xor with one of the
                                                   characters for security. */
                add ebx, ecx                    /* ecx is our offset into password. */
                xor al, [ebx]                   /* dereference ebx to get the value within the
                                                    password array. */

                push comport
                push eax                        /* load up stack to call fputc(EOT, comport),
                                                    when EOT has been encrypted */
                call DWORD PTR [fputc]
                add esp, 8

                lea eax, successMsg
                push eax
                call DWORD PTR [printf]         /* print transmission complete message,
                                                    printf(successMsg) */
                add esp, 4

                push comport
                call DWORD PTR [fclose]         /* close comport file, fclose(comport) */
                add esp, 4

                push srcfile
                call DWORD PTR [fclose]          /* close srcfile, fclose(comport) */
                add esp,4
        }
        return 0;
}
```

RECEIVER ASSEMBLER CODE

```
/* **************************Receiver Program*****************************
*
*Author: Sami Giacaman
*Date Created: 12/11/2011
*Date Modified:  30/11/2011
*Purpose: A Program to act as a Receiver for Another Computer Transmitter.
*
********************************************************************** */

/* **************************Include Part************************** */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#define EndT 0x80        /* To Be Used for Knowing When it is the End of Transmitting. */

/* ********************************************************************** */

/* **************************Main Program************************** */

int main(void)

{

char rdmode[] ="r";                                    /* Used For Reading Data From the COM Port. */
char wtmode[] ="w";                                    /* Used For Writing Data to File. */
char pname[]  ="COM1";
char error1[] ="\nFail to Open COM Port.\n";                    /* Error Message One. */
char error2[] ="\nFail to Open Data File.\n";                   /* Error Message Two. */
char fname[]  ="C:/CSA/data.txt";                              /* File Location. */
char Receive[] ="\n\nReceiving Completed!.\n";                  /* Received Message. */
char PasswordMessage[] = "\nPlease Enter Password:\n";          /* Password Request. */
char Password[16] ={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};          /* 15 Char Password. */
char PasswordScan[] ="%15s";                                   /* Password Length Scan. */

char Intro[] ="        ******************Receiver   Program*****************\n";    /* Intro Display. */

          FILE *portptr;
          FILE *fileptr;
          int  PassLength = 0;
          int c;


           _asm
                {
                   lea     eax,Intro                    /* Display an Into Message. */
                  push   eax
                  call    DWORD PTR (printf)          /* Display Message. */
                  add     esp, 4
                   lea     eax,PasswordMessage         /* Display an Input Password Message. */
                  push   eax
                  call    DWORD PTR (printf)          /* Display Message. */
                  add     esp, 4
                   lea     eax, Password
```

```
                push   eax
                lea    eax, PasswordScan
                push   eax
                call   DWORD PTR (scanf)        /* Scan the Input Password. */
                add    esp, 4                   /* Keep Password Pointer on Stack.  */
                call   DWORD PTR (strlen)       /* Get The Password Length. */
                add    esp, 4                   /* Clean Password Pointer of the Stack. */
                mov    PassLength, eax          /* Save Password Length. */

/* ********************************************************************
* Open File. Make Sure its Open, to Write on it the Received Data.
******************************************************************** */

                lea    eax,wtmode
                push   eax
                lea    eax,fname
                push   eax
                call   DWORD PTR (fopen)        /* Open Data File For Writting. */
                add    esp,8
                mov    fileptr, eax
                or     eax,eax

                jnz    PortCheck                /* Jump to PortCheck, to Open the COM Port, after
                                                 * Opening the File. */

                lea    eax,error2               /* Fail to Open File, Will Display an Error and
                                                 *  Shutdown the program. */
                push   eax
                call   DWORD PTR (printf)       /* Display Error Message. */
                add    esp, 4
                jmp    Endit


/* ********************************************************************
*
*       if ((portptr = fopen("COM1", "r"))== NULL)
*         {
*                 printf("Fail to Open COM Port\n");
*
******************************************************************** */

PortCheck:

                lea    eax, rdmode
                push   eax
                lea    eax,pname
                push   eax
                call   DWORD PTR (fopen)        /* Open COM Port For Reading. */
                mov    portptr,eax
                add    esp,8
                xor    edx, edx
                or     eax,eax


                jnz    FPOK                     /* Jump to File/Port OK Loop, to Start Receiving Data,
                                                 *  After Check the Port. */

                lea    eax,error1               /* Fail to Open Port, Will Display an Error and
                                                 *  Shutdown the program. */
                push   eax
                call   DWORD PTR (printf)       /* Display Error Message. */
                add    esp,4
```

```
                    jmp    Endit


/* ********************************************************************* *
*}
*   while ((c= fgetc(dp)) != EOF) {
*   putch( c);
*}
********************************************************************** */

FPOK:

                    push   edx
                    mov    eax, portptr              /* Get the Next Char, In the COM Port. */
                    push   eax
                    call   DWORD PTR (fgetc)
                    add    esp,4
                    pop    edx


/* ********************************************************************* *
*
*Decryption:
*
*char Decrypt(char byte)
*
*{
*  char Code= 0x00;
*
*  byte^=0xAD
*
*  Code = (byte &0x0f) <<4;
*  Code|= ((byte &0xf0) >>4;
*}
*
* And Password XORing with data and Characters.
* Also Checking if it's the end of transmitting.
*
/* ***************************Decryption***************************** */

                    lea    ebx, Password       /* Get Password String so We Can XOR it With One of the
                                                  * Characters for Security, edx is our Offset into Password. */

                    add    ebx, edx            /* edx is Our Offset Into Password. */

                    xor    al,[ebx]
                    ror    al, 4                /* Effective Nibble Swap for Obfuscation.



                    cmp    al,EndT             /* Compare al to 0x80, to Check if there are
                                                  * No more Chars to be Sent.*/
                    je     end                 /* If True, End Program. */

/* ********************************************************************* */

                    push   edx                 /* Save edx,, as its our Offset Into Password. */
                    mov    ecx, eax            /* Move eax to ecx, to be Compared,
                                                  * To know if it's the end of transmitting.
                                                  * Then also save its value to the file, and
                                                  * display it on the consol. */
```

```
                cmp     ecx, EOF
                je      end                     /* Escape Loop if NO More File to Process. */

                mov     eax, fileptr
                push    eax
                push    ecx
                call    DWORD PTR (fputc)       /* Save Char to File. */
                pop     ecx
                add     esp,4

                push    ecx
                call    DWORD PTR (putch)       /* Display Char to Screen. */
                add     esp,4
                pop     edx                     /* Restore edx to be modified. */
                inc     edx                     /* Increment for the next Char in Password. */
                mov     eax, PassLength         /* To Save The Password Length. */
                cmp     edx, eax                /* If we Have reached the End of the Password,
                                                 * Reset to the Beginning of Password String to
                                                 * Endlessly, Use String. */
                jge     Resetedx

                jmp     FPOK                    /* Loop Again until the End Of File. */


Resetedx:

                xor edx,edx                     /* edx is Our Offset into Password. Reset it by
                                                 * Clearing edx*/
                jmp  FPOK


end:
                lea     eax,Receive             /* Display Received Message. */
                push    eax
                call    DWORD PTR (printf)      /* Display Message. */
                add     esp, 4

                mov     eax, fileptr
                push    eax
                call    DWORD PTR (fclose)      /* Close File. */
                add     esp, 4

                mov     eax, portptr
                push    eax
                call   DWORD PTR (fclose)       /* Flush & Close COM Port. */
                add    esp, 4

                xor    eax,eax                  /* Clear eax. */
                xor    ecx,ecx                  /* Clear ecx. */
Endit:
                NOP
}
/* ********************************************************************** */

                return 0;
}

/* ********************************************************************** */

/* ********************************************************************** */
```

REFERENCE LIST

❖ Computer Engineering 2, 2003. *Laboratory Notes, Chapter 8 C Programming.* [Online] Available at: <http://courses.engr.illinois.edu/ece390/books/labmanual/c-prog-mixing.html> [Accessed on 22nd November 2011].

❖ Williams, R., 2011.*Intel Worksheet*. [E-book] Available at: <http://www.cems.uwe.ac.uk/~rwilliam/CSA_ufcEHV-20-1/> [Accessed 11th November 2011].

❖ Banister Fiend, 2008. *Calling an Asm Function from C.* [Online] Available at: <http://banisterfiend.wordpress.com/2008/08/15/calling-an-asm-function-from-c/> [Accessed on 24th November 2011].

❖ Microsoft, 2011. *MSDN.* [Online] Available at: < http://msdn.microsoft.com/en-gb/ms348103> [Accessed on 23rd November 2011].

❖ Unixwiz, 2010. *Inter x86 Function-Call Conventions.* [Online] Available at: <http://unixwiz.net/techtips/win32-callconv-asm.html> [Accessed on 22nd November 2011].

❖ Williams, R., 2011.*Serial Communications Sheet*. [E-book] Available at: <http://www.cems.uwe.ac.uk/~rwilliam/CSA_ufcEHV-20-1/> [Accessed 6th November 2011].