Application Report
SPRA573

Extracting Equivalent Hex Values From a COFF File

Arun Chhabra and Ramesh Iyer *Digital Signal Processing Solutions*

Abstract

During compilation and linking of code, the assembler and linker create object and output files respectively. These files are in the common object file format (COFF). The initialized sections (.text, .data and .sect) are placed in the processor memory when the program is loaded. The uninitialized sections (.bss and .usect) do not have any actual contents, but instead they reserve space in memory. At runtime the program uses this reserved space to create and store variables. The COFF-to-hex extraction utility extracts the equivalent hex values of the COFF executable file.

Contents

Extracting Data from a COFF File.....2

Extracting Data from a COFF File

The appended COFF-to-hex extraction utility converts a COFF file (versions 1 and 2) to an equivalent data dump in hex format.

The utility shown here is intended to parse a 16-bit COFF format executable file and print out the extracted hex code. The code is designed to run on a PC platform running Windows 95™. To better understand invoking of this utility, consider the following example consisting of the sample assembly file:

Example 1. Sample Assembly File

```
       ************************
       Example file:  SAMPLE.asm
       ************************
                   .def  main
                   .mmregs
                   .text

       main:       STM 2h, 002ch
                   LD #0h, A
                   LD #1h, A
       loop:
                   ADD #2h, A
                   SUB #1h, A
                   NOP

                   LD #10h, B
                   SUB A, B

                   BC loop, BNEQ
                   .end
                      ************************
                      End of sample.asm
                      ************************
*****************************************
*    FILENAME:  VECTORS.ASM for kernal.asm
*
*    AUTHOR:  ARUN CHHABRA
*
*    DATE:  10/15/97
*
*****************************************



         .ref  start
         .ref  main
```

```
  .sect ".vectors"
  .mmregs

B       start
NOP
NOP
```

Assemble and link this file with the appropriate vectors and linker command file. The coff utility will be applied on the generated "sample.out" file. On the DOS command line, type in the following:

```
"coff_both -out sample.out"
```

The generated output containing the extracted hex code will be entitled "sample.out.c" and its contents will appear as shown in Example 2.

Example 2. Generated Output Containing Extracted Hex Code

```
**************************
Contents of sample.out.c
**************************

Section = .text
src_addr = 0x0
length = 0xD (13)
dest_addr = 0x80
space = 0

    0x772C,  0x0002,  0xE800,  0xE801,
    0xF000,  0x0002,  0xF010,  0x0001,
    0xF495,  0xE910,  0xF520,  0xF84C,
    0x0084,

checksum = 0x6751

Section = .vectors
src_addr = 0x0
length = 0x4 (4)
dest_addr = 0xFF80
space = 0

    0xF073,  0x0080,  0xF495,  0xF495,

checksum = 0xF0F3
         ************************
         End of sample.out.c
         ************************
```

```
*************************************************************************
SOURCE CODE BEGINS HERE
*************************************************************************
// FILENAME:    Coff_both.c
// DESCRIPTION: Convert a COFF file (versions 1 and 2) to an equivalent
//              data dump in hex format

// ARGUMENTS:
//              a) Input: TI executable .out file (e.g. “foo.out”)
//              b) Output:   “foo.out.c”

*************************************************************************

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include "coff.h"
#include <errno.h>

int out = 0;

*************************************************************************
SOURCE CODE – function swapushort() BEGINS HERE
*************************************************************************
// FUNCTION NAME:  swapushort()
// DESCRIPTION: Obtains hex values of data in a byte-wise manner

// ARGUMENTS:
//              a) Input: ushort data
//              b) Output:   tmp

*************************************************************************

ushort swapushort (ushort data)
{
    ushort tmp;
    tmp = (data & 0x00FF);
         tmp|= (data & 0xFF00);

    return tmp;
}


*************************************************************************
SOURCE CODE – function xor() BEGINS HERE
*************************************************************************
// FUNCTION NAME:  xor()
// DESCRIPTION: Print checksum output in specified format

// ARGUMENTS:
//              a) Input: ushort a, ushort b
//              b) Output:   (~(a|b)^(~a|~b))
//                     i.e. [a <XOR> b]

*************************************************************************

ushort xor (ushort a, ushort b)
```

```
{
    return (~(a|b)^(~a|~b));
}


*************************************************************************
SOURCE CODE – function Download() BEGINS HERE
*************************************************************************
// FUNCTION NAME:  Download()
// DESCRIPTION: Format output appearance and call two other routines that
//              output result and perform checksum.  The file pointer "fp"
//              corresponds to the target output file.  The file pointer is
//              passed as a parameter from the main() routine (passed as
//              file pointer “ofp”) to Download() (referred to here as
//              “fp”).

// ARGUMENTS:
//              a) Input: uint src_addr, uint length
//                     uint dest_addr, int space, struct buffer buf

// STEPS:
//              The following steps outline the procedure for Download():
//              i) Call swapushort()
//              ii)Call xor()

*************************************************************************

void
DownLoad (uint src_addr, uint length, uint dest_addr,
           int space, struct buffer buf, FILE *fp)
{
    int i;
    int wrap = 0;
    ushort checksum = 0;

    if (out)
         {
         fprintf(fp, "src_addr = 0x%X \n", src_addr);
         fprintf(fp, "length = 0x%X (%d) \n", length, length);
         fprintf(fp, "dest_addr = 0x%X\n", dest_addr);
         fprintf(fp, "space = %d\n", space);
         fprintf(fp, "    \n");
         for (i = 0; i < length; i++)
         {
            if (4 <= wrap)
         {
               wrap = 0;
               fprintf(fp, "\n    ");
         }

             fprintf(fp, "0x%04X,  ", swapushort(buf.data[i]));
             checksum = xor(checksum, swapushort(buf.data[i]));
               /* prints the output in the specified format and performs
                  the checksum */
             wrap++;
         }
         fprintf(fp, "\n\n");
         fprintf(fp, "checksum = 0x%4X\n\n", checksum);
```

```
        }
    return;
}


*************************************************************************
SOURCE CODE – Main() routine BEGINS HERE
*************************************************************************
// FUNCTION NAME:  Main()

// ARGUMENTS:
//             a) Input: int argc, char *argv[]
//             b) Output:   entryaddr
// STEPS:
//       The following steps outline the procedure for Main():
//          i)    Read command line arguments
//          ii)   Enter appropriate 'if' loop depending on COFF format
//                specified
//          iii)  Get COFF file header
//          iv)   Get COFF section header
//          v)    Print content of each section by calling Download()

*************************************************************************

int
main (int argc, char *argv[])
{
    char    *targetfile;
    char    sectionname[9] = "";
    long    entryaddr = 0;
    long    tempentry;
    FILE*      infile;
    long    sectlen = 0;
    long    remaining = 0;
    double     headersize;
    unsigned char  *file2struct_ptr;
    struct fheader fileheader;
         struct sheader1 sectheader1;
         struct sheader2 sectheader2;
    struct buffer  inbuff;
    long    currentloc = 0;
    int     i, value;
    char    ofname[256], line[2];
    FILE    *ofp;

         if (argc > 2)
            {
/* Check if number of items (entries) on the command line */
/* are greater than 2 */

         if (0 == strcmp(argv[1], "-out"))
            /* if the second entry on the command line is "-out" */
         {
             out = 1;                    /*  variable out = true */

             targetfile = argv[2];
/* 'targetfile' points to the third entry on command line, which */
/* is the input to the "coff_new" routine; in this case, it is   */
```

```
/* the executable generated by TI tools as a .out file           */
        }
        else
        {
            printf("usage: coff [-out] <coff_image>\n");
            exit(1);
/* if "-out" condition not satisfied, then print error in usage message */
        }
        }
        else if (argc > 1)
              /* if number of entries on command line are greater  */
              /* than 1 */
        {
        targetfile = argv[1];
/* make 'targetfile' point to the second entry on command line       */
        }
        else
        {
        printf("usage: coff [-out] <coff_image>\n");
                        /* else print error in usage message *      /
        exit(1);
        }

        if (out)
/* if second entry on command line = "-out" */
        {
        sprintf(ofname, "%s.c", targetfile);
/* prints out the filename pointed to by targetfile into a */
/* string "ofname" */

        if (NULL == (ofp = fopen(ofname, "w")))
/* file pointer "ofp" is assigned a pointer to the file "ofname".  */
/* If "ofname" doesn't exist then, print error message below */
        {
            printf("cannot open file %s.  errno = %d\n", ofname, errno);
            return;
        }
        }

        printf("Which type of COFF output file format are you using?\n");
printf("\tEnter '2' if using COFF2, enter '1' if using COFF1...followed by
'Enter'\n\n");

        fgets(line, sizeof(line), stdin);
        sscanf(line, "%d", &value);

        /***** FOR COFF1 FORMAT *****/

        if (value==1)
        {
        /* Get File header */

        file2struct_ptr = (unsigned char *)&fileheader;
/* assigns the address of 'fileheader' to file2struct_ptr ; 'fileheader' */
/* is a structure of type 'struct fheader' */

        headersize = sizeof(fileheader);
/* headersize is assigned the size of structure 'fileheader' */
```

```
          if ((infile = fopen(targetfile, "rb")) == NULL)
          {
                    /* if 'targetfile' does not exist then, print error */

printf("Cannot open %s file\n", targetfile);
             return(-1);
          }

          fread(file2struct_ptr, headersize,1,infile);
/* the 'file2struct_ptr' will read 1 entry of 'headersize' bytes from */
/* the file 'infile' ;*/

          printf("App File: %s   %s", targetfile, ctime(&fileheader.stamp));
             /* prompt info while invoking utility */

                              /* Skip optional header, if any */

printf("Optional header = %d \n", fileheader.optbytes);
if(fileheader.optbytes)
             {
          fseek(infile,fileheader.optbytes-2, SEEK_CUR);
/* Set the file position for "infile" to optbytes-2 characters from where
it was left off by the previous fread statement */

          printf("COFF (.out) file\n");
          }

    /* Get section header */

          headersize = sizeof(sectheader1);
/* 'headersize' is assigned the size of structure sectheader1 */

printf("\nSection\t\tAddress\t\tSize\tPage\n");
printf("-------------------------------------------\n");
fflush(stdout);

          for(i=1; i<= fileheader.sections;i++)
          {
/* for the total number of sections in the input file do the following */

          file2struct_ptr = (unsigned char *)&sectheader1;
/* assigns the address of 'sectheader1' to 'file2struct_ptr' ; */
/* 'sectheader1' is a structure of type 'struct sheader' */

          fread(file2struct_ptr, headersize,1, infile);
/* 'file2struct_ptr' reads 'headersize' number of bytes from 'infile' */

          strncpy(sectionname,sectheader1.name,8);
/* copy 8 characters from the 'name' field of struct 'sectheader1' */
/* into file 'sectionname' */

          strncat(sectionname,"\0",1);
/* terminate the string with a null char */

          if(sectheader1.rawdataptr == 0x0)
          {
                    sectheader1.sectsize =0;
```

```
/* determining section size */
         }

         if ((!strcmp(sectheader1.name,".text")) && i==1)
         {
         entryaddr = sectheader1.phyaddr;
/* enter loop if it is the first iteration and if the name of the */
/* section is ".text" */
         }

currentloc = sizeof(fileheader) + fileheader.optbytes +
(i* sizeof(sectheader1)) - 2;

                  /* this covers:
                  - the fileheader length,
                  - the optional fileheader length
- the length of the sectheader1 for the number of sections read
(reflected by 'i' value of loop) - (-2), to give the current section ptr
location */


         file2struct_ptr = (unsigned char *)&inbuff;
/* file2struct_ptr points to structure inbuff of type struct buffer */

         fseek(infile,sectheader1.rawdataptr,SEEK_SET);
         sectlen=0;
         printf("%s\t\t",sectionname);
         printf("%4.4xh\t\t", sectheader1.phyaddr);
         printf("%4.4xh\t", sectheader1.sectsize);
         if ((int)sectheader1.page == 0)
         {
         printf("PROG\n");
         }
         else
         {
         printf("DATA\n");
         }

         /* Print the content of each section */

         if(sectheader1.sectsize > 0 && strcmp(sectheader1.name,".xref"))
         {
               /* If the sectionsize is non-zero and if the sectionname */
               /* is not ".xref" then enter the loop */

         if (out)
         {
         fprintf(ofp, "Section = %s\n", sectionname);
/* recall, "ofp" is a pointer to the opened file 'targetfile.c' with */
/* write option */
         }

         for(; sectlen < sectheader1.sectsize; )
         {
         remaining = sectheader1.sectsize - sectlen;
         if(remaining >= HPIbuffsize)
         {
         fread(file2struct_ptr, HPIbuffsize *2, 1, infile);
```

```
/* read contents of targetfile and store them in buffer */
/* "file2struct_ptr"...since this is a pointer to inbuff, then the */
/* contents are stored in inbuff */

         DownLoad(HPIbuf,HPIbuffsize,
               (int)(sectheader1.phyaddr+sectlen),
                              sectheader1.page, inbuff, ofp);
         /* array inbuff is an argument for function Download */

         remaining=HPIbuffsize;
         }
         else
         {
         fread(file2struct_ptr,(size_t)remaining*2,1, infile);
                  DownLoad(HPIbuf,(int)remaining,
(int)(sectheader1.phyaddr+sectlen),
                           sectheader1.page, inbuff, ofp);
         }
         sectlen +=remaining;
         }
         }
         fseek(infile,currentloc, SEEK_SET);
         fflush(stdout);
            }
         #if 0
         if(strlen(entrypt) >= 1)
            {
         if((tempentry =Find_Label(infile, entrypt, fileheader)) == -1)
         {
printf("\n\"%s\" label not found...default to .text section\n",entrypt);
         }
         else
         {
            entryaddr= tempentry;
         }
         }
         #endif
         printf("\nEntry Point is: %4.4x\n\n", entryaddr);

         fclose(infile);
         if (out)
            {
         printf("Output file is %s\n", ofname);
         fclose(ofp);
            }
         return(entryaddr);
         }

         /***** FOR COFF2 FORMAT *****/

         else if (value==2)
         {
            /* Get File header */

         file2struct_ptr = (unsigned char *)&fileheader;
/* assigns the address of 'fileheader' to file2struct_ptr; 'fileheader' is
a structure of type 'struct fheader' */
```

```
         headersize = sizeof(fileheader);
            /* headersize is assigned the size of structure 'fileheader' */

         if((infile = fopen(targetfile, "rb")) == NULL)
         {
            /* if 'targetfile' does not exist then, print error */

            printf("Cannot open %s file\n", targetfile);
            return(-1);
         }

         fread(file2struct_ptr, headersize,1,infile);
/* the 'file2struct_ptr' will read 1 entry of 'headersize' bytes from the
file 'infile' ;*/

         printf("App File: %s   %s", targetfile, ctime(&fileheader.stamp));
            /* prompt info while invoking utility */

/* Skip optional header, if any */

         printf("Optional header = %d \n", fileheader.optbytes);
         if(fileheader.optbytes)
         {
         fseek(infile,fileheader.optbytes-2, SEEK_CUR);
/* Set the file position for "infile" to optbytes-2 characters from */
/* where it was left off by the previous fread statement */

         printf("COFF (.out) file\n");
         }

/* Get section header */

         headersize = sizeof(sectheader2);
         /* 'headersize' is assigned the size of the structure sectheader2
*/

         printf("\nSection\t\tAddress\t\tSize\tPage\n");
         printf("-------------------------------------------\n");
         fflush(stdout);

         for(i=1; i<= fileheader.sections;i++)
         {
/* for the total number of sections in the input file do the following */

         file2struct_ptr = (unsigned char *)&sectheader2;
/* assigns the address of 'sectheader2' to 'file2struct_ptr'; 'sectheader2'
is a structure of type 'struct sheader' */

         fread(file2struct_ptr, headersize,1, infile);
/* 'file2struct_ptr' reads 'headersize' number of bytes from 'infile' */

         strncpy(sectionname,sectheader2.name,8);
/* copy 8 characters from the 'name' field of struct 'sectheader2' */
/* into file 'sectionname' */

         strncat(sectionname,"\0",1);
/* terminate the string with a null char */
```

```
         if(sectheader2.rawdataptr == 0x0)
         {
            sectheader2.sectsize =0;
            /* determining section size */
         }

         if ((!strcmp(sectheader2.name,".text")) && i==1)
         {
         entryaddr= sectheader2.phyaddr;
         }
/* enter loop if it is the first iteration and if the name of the */
/* section is ".text" */

         currentloc = sizeof(fileheader) + fileheader.optbytes +
                   (i* sizeof(sectheader2)) - 2;
               /* this covers:
                  - the fileheader length,
                  - the optional fileheader length
- the length of the sectheader2 for the number of
         sections read (reflected by 'i' value of loop)- (-2)
         to give the current section ptr location  */


         file2struct_ptr = (unsigned char *)&inbuff;
/* file2struct_ptr points to structure inbuff of type struct buffer */

         fseek(infile,sectheader2.rawdataptr,SEEK_SET);
         sectlen=0;
         printf("%s\t\t",sectionname);
         printf("%4.4xh\t\t", sectheader2.phyaddr);
         printf("%4.4xh\t", sectheader2.sectsize);
         if((int)sectheader2.page == 0)
         {
            printf("PROG\n");
         }
         else
         {
            printf("DATA\n");
         }

         /* Print the content of each section */

         if(sectheader2.sectsize > 0 && strcmp(sectheader2.name,".xref"))
         {
               /* If the sectionsize is non-zero and if the sectionname */
               /* is not ".xref", then enter the loop */

         if (out)
         {
         fprintf(ofp, "Section = %s\n", sectionname);
/* recall, "ofp" is a pointer to the opened file 'targetfile.c' with */
/* write option */
         }

         for(; sectlen < sectheader2.sectsize; )
         {
         remaining =sectheader2.sectsize - sectlen;
         if(remaining >= HPIbuffsize)
```

```
          {
          fread(file2struct_ptr, HPIbuffsize *2, 1, infile);
/* read contents of targetfile and store them in buffer */
/* "file2struct_ptr"...since this is a pointer to inbuff, then the */
/* contents are stored in inbuff */
          DownLoad(HPIbuf,HPIbuffsize,
                   (int)(sectheader2.phyaddr+sectlen),
sectheader2.page, inbuff, ofp);
/* array inbuff is an argument for function Download */

          remaining=HPIbuffsize;
          }
          else
          {
          fread(file2struct_ptr,(size_t)remaining*2,1, infile);
                      DownLoad(HPIbuf,(int)remaining,
                            (int)(sectheader2.phyaddr+sectlen),
                               sectheader2.page, inbuff, ofp);
          }
          sectlen +=remaining;
          }
          }
          fseek(infile,currentloc, SEEK_SET);
          fflush(stdout);
          }
          #if 0
          if (strlen(entrypt) >= 1)
          {
          if ((tempentry =Find_Label(infile, entrypt, fileheader)) == -1)
          {
          printf("\n\"%s\" label not found...default to .text
section\n",entrypt);
          }
          else
          {
          entryaddr= tempentry;
          }
          }
          #endif
          printf("\nEntry Point is: %4.4x\n\n", entryaddr);

          fclose(infile);
          if (out)
          {
          printf("Output file is %s\n", ofname);
          fclose(ofp);
          }
          return(entryaddr);
          }
}



*******
COFF.H
*******
/** File: coff.h  **/
```

```
typedef unsigned int  uint;
typedef unsigned long ulong;
typedef unsigned shortushort;

#define STRMAX  2000
#define HPIbuf     0
#define HPIbuffsize   2000

struct buffer{
         ushort    data[STRMAX];
};

struct   fheader {
    unsigned short version;
    unsigned short sections;
    long    stamp;
    long    symptr;
    long    symbols;
    unsigned short optbytes;
    unsigned short flags;
    unsigned short magic;
};

struct   sheader1 {
    char  name[8];
    long  phyaddr;
    long  virtaddr;
    long  sectsize;
    long  rawdataptr;
    long  relocationptr;
    long  linnumptr;
    unsigned short relocationents;
    unsigned short linnums;
    unsigned short flags;
    char  resv;
    char  page;
};

struct   sheader2 {
    char  name[8];
    long  phyaddr;
    long  virtaddr;
    long  sectsize;
    long  rawdataptr;
    long  relocationptr;
    long  linnumptr;
    unsigned long  relocationents;
    unsigned long  linnums;
    unsigned long  flags;
    shortresv;
    char  page;
};

struct symentry {
    union {
         char  symname[4];
         long  strlowint;
    } symbolname;
```

```
    union {
         char   symnamehalf2[4];
         long   strhighint;
    } symbolnamehalf;
    long  symval;
    short sectnum;
    unsigned short dervtype;
    char  storclass;
    char  auxentries;
};
```

TI Contact Numbers

INTERNET

TI Semiconductor Home Page
www.ti.com/sc

TI Distributors
www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas
Phone +1(972) 644-5580
Fax +1(972) 480-7800
Email sc-infomaster@ti.com

Europe, Middle East, and Africa
Phone
- Deutsch +49-(0) 8161 80 3311
- English +44-(0) 1604 66 3399
- Español +34-(0) 90 23 54 0 28
- Francais +33-(0) 1-30 70 11 64
- Italiano +33-(0) 1-30 70 11 67

Fax +44-(0) 1604 66 33 34
Email epic@ti.com

Japan
Phone
- International +81-3-3344-5311
- Domestic 0120-81-0026

Fax
- International +81-3-3344-5317
- Domestic 0120-81-0036

Email pic-japan@ti.com

Asia
Phone
- International +886-2-23786800
- Domestic
  - Australia 1-800-881-011
    - TI Number -800-800-1450
  - China 10810
    - TI Number -800-800-1450
  - Hong Kong 800-96-1111
    - TI Number -800-800-1450
  - India 000-117
    - TI Number -800-800-1450
  - Indonesia 001-801-10
    - TI Number -800-800-1450
  - Korea 080-551-2804
  - Malaysia 1-800-800-011
    - TI Number -800-800-1450
  - New Zealand 000-911
    - TI Number -800-800-1450
  - Philippines 105-11
    - TI Number -800-800-1450
  - Singapore 800-0111-111
    - TI Number -800-800-1450
  - Taiwan 080-006800
  - Thailand 0019-991-1111
    - TI Number -800-800-1450
  - Fax 886-2-2378-6808
  - Email tiasia@ti.com

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE (“CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty, or endorsement thereof.