Übung 8

Assembler-Direktiven

Ziele dieser Übung:
Sie sollen in der Lage sein,

1) Die Assembler-Diraktiven der GNU-Toolchain für einen Cortex-Mx Prozessors anzuwenden.

Verwenden Sie für den Test Ihres Codes den Debugger und das Leguan-Board. Beachten Sie bei der Programmausführung die Register, die Statusbits und das Memory.

Aufgabe 8.1 Verwendung von Assembler-Direktiven

a) Realisieren Sie ein Programm, welches die folgenden Sections verwendet:
   - .bss nicht initialisierte Variablen
   - .data für initialisierte Variablen
   - Section .text für Konstanten
   - Section .text für den Code

b) Schreiben Sie nun Assembler-Code. Definieren Sie folgende nicht initialisierte Variablen in der Section .bss:
   - counter (word)
   - ytab (array mit 8 words)

c) Definieren Sie folgende initialisierte Variablen in der Section .data:
   - value (word=0x4321)

d) Definieren Sie folgende Konstanten in der Section .text:
   - CTAB1 (Array mit 8 Words, wie folgt initialisiert: [0,1,2,3,4,5,6,7]
   - CTAB2 (Array mit 8 Bytes, alle auf 0x55 initialisiert)
   - TEXT (String mit “BFH-TI Biel-Burgdorf“ initialisiert)

e) Codieren Sie folgenden Ablauf in der Section .text:
   - Initialisierung der Variablen “counter“ mit 0x12345678
   - Initialisierung des Array “ytab“ beginnend mit 0, jedes Wort um 1 inkrementiert.
   - Variablenwert “value“ laden, mit 33 multiplizieren und wieder zurückspeichern.

f) Um während dem Debugging zu wissen, auf welchen Adressen die Variablen und Konstanten liegen, generiert der Linker/Locator ein Mapfile. Dieses finden Sie in der Entwicklungsumgebung für jedes Projekt im jeweiligen Debug-Ordner. Um die symbolischen Namen der Variablen und Konstanten im Mapfile wirklich zu sehen, müssen diese in der Assemblerdatei noch exportiert werden, z.B. “.global CTAB1“.

g) Studieren Sie das erzeugte Mapfile und suchen Sie die Adressen Ihrer Variablen und Konstanten.

h) Führen Sie das Programm aus:

- Prüfen Sie nach dem Laden des Programms die Konstanten mit Hilfe eines Memory-Fensters. Prüfen Sie auch die Werte der initialisierten und uninitialisierten Variablen. Variablen können Sie entweder im Memory-Fenster prüfen, oder Sie zeigen diese als “Watch Expression“ an.
- Gehen Sie anschliessend schrittweise durch Ihren Code und prüfen Sie, wie sich die Werte der Variablen ändern.

Aufgabe 8.2 CRC-CCITT16 Tabelle

Die Aufgabenstellung geht davon aus, dass die Grundlagen des CRC bekannt sind. CRC ist ein wirkungsvolles Verfahren, um Fehler während einer Datenübertragung zu erkennen. Die Methode kann mittels Hardwarebausteinen oder rein softwaremässig angewandt werden. Wenn mit einer Softwarelösung gearbeitet wird, kann der Rechenaufwand mit Hilfe einer Tabelle stark reduziert werden.

Für die Sicherung byteweiser Daten kann der CRC-Wert jedes Bytes nach [BIN95] tabellarisch aufgelistet werden:

```
/* CCITT Lookup Table */
unsigned short ccitt_table[256] =
{
/*   0 - */ 0x0000,0x1021,0x2042,0x3063,0x4084,0x50a5,0x60c6,0x70e7,
/*   8 - */ 0x8108,0x9129,0xa14a,0xb16b,0xc18c,0xd1ad,0xe1ce,0xf1ef,
/*  16 - */ 0x1231,0x0210,0x3273,0x2252,0x52b5,0x4294,0x72f7,0x62d6,
/*  24 - */ 0x9339,0x8318,0xb37b,0xa35a,0xd3bd,0xc39c,0xf3ff,0xe3de,
/*  32 - */ 0x2462,0x3443,0x0420,0x1401,0x64e6,0x74c7,0x44a4,0x5485,
/*  40 - */ 0xa56a,0xb54b,0x8528,0x9509,0xe5ee,0xf5cf,0xc5ac,0xd58d,
/*  48 - */ 0x3653,0x2672,0x1611,0x0630,0x76d7,0x66f6,0x5695,0x46b4,
/*  56 - */ 0xb75b,0xa77a,0x9719,0x8738,0xf7df,0xe7fe,0xd79d,0xc7bc,
/*  64 - */ 0x48c4,0x58e5,0x6886,0x78a7,0x0840,0x1861,0x2802,0x3823,
/*  72 - */ 0xc9cc,0xd9ed,0xe98e,0xf9af,0x8948,0x9969,0xa90a,0xb92b,
/*  80 - */ 0x5af5,0x4ad4,0x7ab7,0x6a96,0x1a71,0x0a50,0x3a33,0x2a12,
/*  88 - */ 0xdbfd,0xcbdc,0xfbbf,0xeb9e,0x9b79,0x8b58,0xbb3b,0xab1a,
/*  96 - */ 0x6ca6,0x7c87,0x4ce4,0x5cc5,0x2c22,0x3c03,0x0c60,0x1c41,
/* 104 - */ 0xedae,0xfd8f,0xcdec,0xddcd,0xad2a,0xbd0b,0x8d68,0x9d49,
/* 112 - */ 0x7e97,0x6eb6,0x5ed5,0x4ef4,0x3e13,0x2e32,0x1e51,0x0e70,
/* 120 - */ 0xff9f,0xefbe,0xdfdd,0xcffc,0xbf1b,0xaf3a,0x9f59,0x8f78,
/* 128 - */ 0x9188,0x81a9,0xb1ca,0xa1eb,0xd10c,0xc12d,0xf14e,0xe16f,
/* 136 - */ 0x1080,0x00a1,0x30c2,0x20e3,0x5004,0x4025,0x7046,0x6067,
/* 144 - */ 0x83b9,0x9398,0xa3fb,0xb3da,0xc33d,0xd31c,0xe37f,0xf35e,
/* 152 - */ 0x02b1,0x1290,0x22f3,0x32d2,0x4235,0x5214,0x6277,0x7256,
/* 160 - */ 0xb5ea,0xa5cb,0x95a8,0x8589,0xf56e,0xe54f,0xd52c,0xc50d,
/* 168 - */ 0x34e2,0x24c3,0x14a0,0x0481,0x7466,0x6447,0x5424,0x4405,
/* 176 - */ 0xa7db,0xb7fa,0x8799,0x97b8,0xe75f,0xf77e,0xc71d,0xd73c,
/* 184 - */ 0x26d3,0x36f2,0x0691,0x16b0,0x6657,0x7676,0x4615,0x5634,
/* 192 - */ 0xd94c,0xc96d,0xf90e,0xe92f,0x99c8,0x89e9,0xb98a,0xa9ab,
/* 200 - */ 0x5844,0x4865,0x7806,0x6827,0x18c0,0x08e1,0x3882,0x28a3,
/* 208 - */ 0xcb7d,0xdb5c,0xeb3f,0xfb1e,0x8bf9,0x9bd8,0xabbb,0xbb9a,
/* 216 - */ 0x4a75,0x5a54,0x6a37,0x7a16,0x0af1,0x1ad0,0x2ab3,0x3a92,
/* 224 - */ 0xfd2e,0xed0f,0xdd6c,0xcd4d,0xbdaa,0xad8b,0x9de8,0x8dc9,
/* 232 - */ 0x7c26,0x6c07,0x5c64,0x4c45,0x3ca2,0x2c83,0x1ce0,0x0cc1,
/* 240 - */ 0xef1f,0xff3e,0xcf5d,0xdf7c,0xaf9b,0xbfba,0x8fd9,0x9ff8,
/* 248 - */ 0x6e17,0x7e36,0x4e55,0x5e74,0x2e93,0x3eb2,0x0ed1,0x1ef0
};
```

Zur Nutzung in einem Programm kann die Tabelle manuell eingegeben oder mit einem Programm berechnet werden.

Ein Programm in der Sprache C zur Berechnung der Tabelle finden Sie im nachfolgendem Code:

```c
/*
Beispiel zur Implementierung eines CRC-Verfahrens mit einem 16-Bit CRC nach CCITT
unter Verwendung von tabellierten CRC-Werten. Hierbei wird fuer jedes zu sichernde
Byte ein CRC-Wert aus der Tabelle gelesen.
*/

#include <stdio.h>
#define TABLE_SIZE 256
unsigned short crcTable[TABLE_SIZE];

// Erzeugen der CRC-Tabellenwerte. Dies kann durch Berechnung fuer jede Bitstelle
// erfolgen. Alternativ koennte das Array auch als Konstante definiert werden.
//
// Parameter:
// crcTable[]: Zeiger auf CRC-Tabelle
// tableSize: Groesse der CRC-Tabelle
//
// crcPoly CRC Polynomkoeffizienten (1-Werte des CRC-Polynoms):
// CCITT: 0x1021 (X.25, HDLC/SDLC, ADCCP)
// CCITT rev.: 0x0811
// CRC16: 0x8005 (ARC)
// CRC16 rev.: 0x4003 (LHA)
// XMODEM: 0x8048
//
// Referenz:
// Implementierung aehnlich CRC-CCITT, p.555 aus
// "Practical Algorithms for Programmers"
// A. Binstock/ J.Rex, Addison Wesley Publishing 1995, ISBN 0-201-63208-X

void buildCRCTable(unsigned short crcTable[], int tableSize, unsigned short crcPoly)
{
  int i,j;
  unsigned short crc;

  for (i = 0; i < TABLE_SIZE; i++) {
    crc = i; // CRC Tabellenwert der Position i bestimmen
    for (j = 0; j < 16; j++ ) {
      if (crc & 0x8000) {
        crc = (crc << 1) ^ crcPoly;
      } else {
        crc = crc << 1;
      }
    }
    crcTable[i]=crc;
  }
}

int main(void)
{
  // CRC-Tabelle aufbauen
  buildCRCTable(crcTable, sizeof(crcTable) / sizeof(unsigned short), 0x1021);
  return 0;
}
```

Aufgabe:
Schreiben Sie eine Codesequenz mit .rept-Makros, die eine CRC-Tabelle mit den Werten nach CCITT erzeugt. Die Werte sollen als 16-Bit Datenwerte mit Start beim Label CRC16: im .text-Segment abgelegt werden. Nutzen Sie zur Berechnung die im Assembler zur Verfügung stehenden Operatoren.

Literaturreferenzen:
[*BIN*95] Practical Algorithms for Programmers, Andrew Binstock / John Rex, Addison Wesley Publishing Comp. 1995, ISBN 0-201-63208-X

[*CHA_LD*94] Using ld, The GNU linker,ld version 2, January 1994 Steve Chamberlain, Cygnus Support http://www.gnu.org/software/binutils/manual/ld-2.9.1/html_mono/ld.html