EEAP 282 - PROGRAMMING ASSIGNMENT #2

CYCLIC REDUNDANCY CHECK (CRC)

PURPOSE:
This lab will introduce you to the use of branch and shift instructions as well as program design. You will write a program which computes the CRC for a block of data. For the purposes of this program the block of data will be provided by a series of DC.B statements although, in general, the data would be provided by a data communications buffer.

BACKGROUND:
When you send a sequence of bits (message) from one point to another, you want to know that the message arrived correctly. A common form of insurance is to use a parity bit scheme as discussed in class. Unfortunately, the single parity bit approach mathematically only has about a 50% chance of detecting an error involving more than one bit error.

Most communications protocols use a multibit generalization of the single parity bit called a "cyclic redundancy check" or CRC. In a typical application the CRC is 16 bits long, so that the chance of a random error going undetected is 1 in 2^16, or 1 in 65536. Such an M-bit CRC has the mathematical property of detecting all errors that occur in M or fewer consecutive bits, for any length of message. This latter point is very important as a parity bit is associated with each byte sent whereas a single CRC can be associated with a large block of data, with a consequent savings in communications bandwidth.

Normally CRCs are used for communications. However, there are additional situations where CRCs can be useful. We sometimes need to be able to communicate with low-level hardware or software that expects a valid CRC as part of its input. Or in the manipulation of large amounts of data, it is useful to tag data with a statistically unique key, its CRC. The data can then be identified by comparing only the CRC, a much shorted process. This is possible because identical CRCs (or keys) imply, to a high statistical certainty, identical records.

MATHEMATICS:
The mathematical theory behind CRCs is that of "polynormials over the integers modulo 2." Any binary message can be thought of as a polynormial with coefficients 0 and 1. For example, the message "1100001101" is the polynormial x^9 + x^8 + x^3 + x^2 + 1. Since 0 and 1 are the only integers modulo 2, a power of x in the polynormial is either present (1) or absent (0).

An M-bit long CRC is based on a particular polynormial of degree M, called the generator polynormial. The choice of the polynormial is a matter of convention. For 16-bit CRCs, the CCITT (an international telecommunications standards organization) has adopted the "CCITT polynormial" which is x^16 + x^12 + x^5 + 1.

Given the generator polynomial G of degree M (which can be written either in polynomial form or as a bit-string, e.g., 10001000000100001 for the CCITT standard) you compute the CRC for a sequence of bits S by the following procedure:

- First, multiply S by x^M, that is append M zero bits to it.
- Second, divide G into S*x^M. Keep in mind that the subtractions are done modulo 2, so that there are never any borrows: Modulo 2 subtraction is the same as logical exclusive-or.
- Third, ignore the quotient.
- Forth, when you eventually get to a remainder, it is the CRC, call it C. C will be a polynormial of degree M-1 or less. Therefore, in bit string form, it has M bits, which may include leading zeros.

Here is an example for S=1101011011 and G=10011. Note that the math used in this division is modulo 2 meaning the subtraction is done on a bit by bit basis (no borrow), which is equivalent to logical exclusive-or. This kind of math is common in coding and compression theory. It may look a little strange but it works.

```
             1100001010
           _____________
10011 ) 11010110110000  <---append M=4 zero bits
        10011,,.,,....
        -----,,.,,....
         10011,.,,....
         10011,.,,....
         -----,.,,....
          00001.,,....
          00000.,,....
          -----.,,....
           00010,,....
           00000,,....
           -----,,....
            00101,....
            00000,....
            -----,....
             01011....
             00000....
             -----....
              10110...
              10011...
              -----...
               01010..
               00000..
               -----..
                10100.
                10011.
```

```
-----.
 01110
 00000
 -----
  1110 = Remainder
```

If you worked through the above steps in an example, you will see that most of what you write down in the division is superfluous. You are actually just left-shifting sequential bits of S, from the right, into an M-bit register. Everytime a 1 bit gets shifted off the left end of this register, you zap the register by an EOR with the M low order bits of G (that is, all the bits of G except its leading 1). When a 0 bit is shifted off the left end, you don't zap the register. When the last bit that was originally part of S gets shifted off the left end of the register, what remains is the CRC.

This procedure can be easily implemented in hardware requiring only a shift register with a few hard-wired EOR taps. CRCs are typically computed in communications devices by a single chip (or a small part of one) using a tapped shift register.

PSEUDOCODE:

The following is a pseudo code description of the CRC algorithm.

```
        MASK = %0001000000100001
/* CCITT G without leading 1 */
        CRC = 0
LOOP:   get next available byte (character) C
        CRC = CRC EOR C<<8
/* <<8 is the shift left by 8 bits */
        for i=1 to 8
                if (bit 15 of CRC == 1) then
                        CRC = (CRC<<1) EOR MASK
                else
                        CRC = CRC<<1
                endif
        goto LOOP
```

The trick used is that character bits are EORed into the most significant bits, eight at a time, instead of being fed into the least significant bit, one bit at a time, at the time of the register shift. This works because the EOR is associative and commutative - we can feed in character bits any time before they will determine whether to zap with the generator polynormial.

DATA:
Use your program on the following sets of data:

DATA SET NO.1

```
TBL     DC.W    0,0,402,0,804,0,1205,0,1606,0,2006,0,2404,0
        DC.W    2801,0,3196,0,3590,0,3981,0,4370,0,4756,0
```

```
        DC.W    5139,0,5520,0,5897,0,6270,0,6639,0,7005,0,7366,0
        DC.W    7723,0,8076,0,8423,0,8765,0,9102,0,9434,0,9760,0
        DC.W    10080,0,10394,0,10702,0,11003,0,11297,0,11585,0
        DC.W    11866,0,12140,0,12406,0,12665,0,12916,0,13160,0
        DC.W    13395,0,13623,0,13842,0,14053,0,14256,0,14449,0
        DC.W    14635,0,14811,0,14978,0,15137,0,15286,0,15426,0
        DC.W    15557,0,15679,0,15791,0,15893,0,15986,0,16069,0
        DC.W    16143,0,16207,0,16261,0,16305,0,16340,0,16364,0
        DC.W    16379,0
```

DATA SET NO.2

```
DTBL    DC.W    12867,0,12859,0,12843,0,12820,0,12789,0,12751,0
        DC.W    12704,0,12650,0,12589,0,12519,0,12443,0,12358,0
        DC.W    12267,0,12168,0,12061,0,11948,0,11827,0,11699,0
        DC.W    11564,0,11422,0,11273,0,11117,0,10955,0,10786,0
        DC.W    10611,0,10429,0,10241,0,10046,0,9846,0,9640,0
        DC.W    9427,0,9210,0,8986,0,8758,0,8524,0,8285,0
        DC.W    8040,0,7792,0,7538,0,7280,0,7017,0,6750,0
        DC.W    6479,0,6205,0,5926,0,5644,0,5359,0,5070,0
        DC.W    4778,0,4483,0,4186,0,3886,0,3584,0,3280,0
        DC.W    2973,0,2665,0,2356,0,2044,0,1732,0,1419,0
        DC.W    1104,0,789,0,474,0,158,0
```

DATA SET NO.3

```
        DC.B 'Four score and seven years ago, our fathers'
        DC.B 'brought forth upon this continent a new nation,'
        DC.B 'conceived in liberty and dedicated to the'
        DC.B 'proposition that all men are created equal.  Now'
        DC.B 'we are engaged in a great civil war, testing'
        DC.B 'whether that nation or any nation so conceived'
        DC.B 'and so dedicated can long endure.  We are met'
        DC.B 'on a great battlefield of that war.  We have'
        DC.B 'come to dedicate a portion of that field as a'
        DC.B 'final resting-place for those who have given their'
        DC.B 'lives that that nation might live.  It is altogether'
        DC.B 'fitting and proper that we should do this. But in'
        DC.B 'a larger sense, we cannot dedicate, we cannot'
        DC.B 'consecrate, we cannot hallow this ground. The'
        DC.B 'brave men, living and dead, who struggled here have'
        DC.B 'consecrated it far above our power to add or detract.'
        DC.B 'The world will little note nor long remember what'
        DC.B 'we say here, but it can never forget what they did'
        DC.B 'here.  It is for us the living rather to be dedicated'
        DC.B 'here to the unfinished work which they who fought here'
        DC.B 'have thus far so nobly advanced. It is rather for us'
        DC.B 'to be here dedicated to the great task remaining'
        DC.B 'before us - that from these honored dead we take'
```

```
        DC.B 'increased devotion to that cause for which they gave'
        DC.B 'the last full measure of devotion - that we here'
        DC.B 'highly resolve that these dead should not have died'
        DC.B 'in vain, that this nation under God shall have a new'
        DC.B 'birth of freedom, and that government of the people,'
        DC.B 'by the people, for the people shall not perish'
        DC.B 'from the Earth.'
; Abraham Lincoln, November 19, 1863.
```

PROGRAM

Implement the above CRC algorithm in 68000 assembly using EOR and shift instructions. Run your program on the above three blocks of data. Rerun the program for one of the blocks causing one or more bit errors. Describe what you changed and the resultant CRC.

Your lab writeup should include:

1. signed title page
2. program listing (assembler listing with symbol table)
3. short explanation of how your program works. Be sure to explain how you fetched one byte from the given data sets in your program. (What addressing mode did you use, how does it works)
4. CRCs for the above three blocks of data
5. Answers to the following questions:

QUESTIONS:

1. Hand-calculate the CRC for the message "T" using the algorithm shown in the pseudo code. Show all the intermediate CRCs in Hex and Binary forms. Hand-calculate the CRC for the same message using the direct division. Compare the results.
2. Change one byte from each data set and re-calculate the CRCs for the three given blocks of data. Compare the CRCs with those for the unchanged, original data sets. Discuss the results.
3. Devise a way to automatically calculate the number of bytes in the above data sets using assembler directives. HINT: You can do calculations with labels, or you can use assembler directives.

ADDITIONAL REFERENCES:

1. URL: http://www.paranoia.com/~filipg/HTML/LINK/F_crc_v3.html
2. Tanenbaum, A.S., "Computer Networks", Prentice Hall, 1981
3. Knuth, D.E., "The Art of Computer Programming", Volume 2: Seminumerical Algorithms