

1 Introduction

In this lab, you will write a program that will calculate the checksum of a block of memory. Calculating checksums is a common way to test a ROM device and is described in Appendix A of this description. You will also be introduced to the BasicIO routines and use these routines for a simple user interface. This lab tests your ability, to create flow diagrams, design a subroutine with parameters, and use the BasicIO module.

2 Requirements

The program must meet the following requirements:

1. It must consist of the main program and a checksum subroutine, *ChkSum()*, which is described below. The main program handles the user interface and passes the memory block parameters to the checksum subroutine.
2. The main program must do the following:
 - a. Prompt the user to start the test by selecting 'enter' to run the test, 'q' or 'Q' to quit.
 - b. Wait for a 'enter' or 'q' from the terminal keyboard. All other key presses must be ignored and not echoed back to the terminal.
 - c. If 'enter' then calculate the checksum by calling the checksum routine and display the checksum on the CRT as follows:
CS: XXXX←
where, **XXXX** is the resulting checksum and ← is a carriage return.
 - d. Return to a.
 - e. Else if 'q' or 'Q' then execute an 'swi' to return to the D-Bug12 monitor.
3. The checksum function must be a general purpose subroutine with the following parameter specification:
 - a. The starting address of the block is passed in ACCD. The ending address of the block is passed on the stack.
 - b. The calculated checksum must be returned in ACCD.

There are three data objects required for this program – A variable that contains the resulting checksum, *ChkResult*, and two stored constants for the starting, *StartAddr*, and ending, *EndAddr*, addresses of the memory block to be tested. These are the values to be passed to the checksum subroutine. The program must be written with *StartAddr* = \$C000 and *EndAddr* = \$FFFF (The Flash block).

Note that the starting and ending addresses are stored constants so they must be created using the *dc* directive.

3 Testing the Checksum Routine

To test your program, verify that checksums agree with those listed below. The two blocks shown will test both even and odd block sizes along with checking for the terminal count bug.

Test Block	Checksum
\$C000-\$FFFF	\$90D2
\$C001-\$FFFF	\$9086

To change the starting address for the second block, use D-Bug12 to change the memory location in which starting address constant is stored.

4 Write-up

The source code for your program must be sent via email by midnight on the due date. The write-up must be turned in before 5:00pm on the due date and include the following material:

- Introduction**
- Program Description including Flow Diagrams**
- Program Listing (hardcopy)**
- Comments and Conclusions**

Due Dates: Source-May 16, 2008; Write-up-May 19, 2008

Appendix A

In this lab, you will write a program that tests a ROM memory block by verifying its contents. If a ROM test is performed when the MCU is first powered-up, one can avoid more catastrophic failures during normal operation.

There are two common ways to test ROM contents, the checksum (or sumcheck) and the Cyclical Redundancy Check (CRC). The checksum is the simplest to perform while the CRC is a better test due to a more randomized result. In this lab, you will write a program to calculate the checksum of the memory block defined by two stored constants.

A checksum is defined as the 16-bit sum of each byte in the block. That is, an unsigned 8-bit addition of each byte with an unsigned 16-bit result. For example:

If the contents of a 5-byte EPROM block are: \$1F, \$FE, \$13, \$05, \$CD

The resulting 16-bit checksum of the 5-byte block is: \$0202

Carries out of the 16-bit result are ignored.

The resulting checksum is a number that can be used to identify the contents of memory. Of course, a 16-bit number can not be unique for every combination of the memory contents. Even if the number is not unique, it is rare for an error in memory that results in the same checksum value.

For example, if second location in the EPROM block above failed to hold its charge, the \$FE would change to \$FF. This failure would result in a checksum of \$0203 so the error would be detected. If however, this error was coupled with an error that caused the third location to go to \$12, the checksum would be the same and the errors would not be detected. This type of error is rare. However, in these cases, a CRC would detect both errors and would be a better, but more complex, solution.