

ECE 379: Introduction to Computer Engineering

Spring 2006

Professor Lipasti

TA: Payam Karbassi

Programming Assignment #3

Due: Friday, May 5, 2006 in class (last day of class)

Instructions: Please complete this programming assignment in a team of 2-3 students.

Problem statement: You are to write a program in LC-3 assembly language that prompts the user to enter 8 bits of data (entered as binary 0/1 characters). The program then displays the eight bits of data, along with 4 check bits ($c_8c_4c_2c_1$) and a parity bit, and prompts the user to retype the data, check bits, and parity bit, possibly introducing one or more bit errors. In response to the retyped data, your program should report either: no error detected, single bit error detected, or dual bit error detected. In the first two cases, the program should then print the correct data, check bits, and parity bit (either unchanged or corrected). In the third case, the program should print the corrupted data, check bits, and parity bit. The program should then prompt for another 8 bits of data, until the user types 'x' to quit.

In summary, the program allows you to correct single-bit errors and detect double-bit errors that occur in the communication path from your eyeballs (reading the bits off screen) to your fingers (typing in the matching pattern of zeroes and ones).

Your program does not need to deal correctly with errors in the input data. That is, if the user types unexpected characters (anything other than 0 or 1, or x as the first character), your program need not do anything reasonable; it can just fail.

Your program should be written in a modular fashion, using subroutines where appropriate. For example, you should reuse your SECEDED generation code from programming assignment two.

The figure below shows an example session (underlined characters are typed by the user and echoed by the program); your program should closely mimic this behavior (but your program won't use underlining).

```

Welcome to ECE 379 Program #3.
Please enter 8 bits of data (x to quit): 00000000
Data: 00000000 Check bits: 1111 Parity bit: 1
Please retype data, check, and parity bits: 0000000011111
Result: no errors detected
Data: 00000000 Check bits: 1111 Parity bit: 1
Please enter 8 bits of data: 00000001
Data entered: 00000001 Check bits: 0011 Parity bit: 0
Please retype data, check, and parity: 0000000000110
Result: single bit error detected and corrected
Data: 00000001 Check bits: 0011 Parity bit: 0
Please enter 8 bits of data (x to quit): 00000011
Data: 00000011 Check bits: 0111 Parity bit: 0
Please retype data, check, and parity bits: 0000000001110
Result: double error detected, uncorrectable data
Data: 00000000 Check bits: 0111 Parity bit: 0
Please enter 8 bits of data: x
Goodbye!

```

Please refer to the handout for programming assignment #2 for background on SECDED codes.

Background on Forward Error Correction: Given the SECDED code described in programming assignment #2, your program can detect and correct a single bit error in either the data, the ECC check bits, or the parity bit, or it can detect (but not correct) a double-bit error. We will first explain the algorithm for detecting single- and double-bit errors, and then go over the algorithm for correcting a single-bit error.

Detecting errors: Errors are detected by recomputing the check bits and the parity bit for the retrieved data bits, and then comparing the recomputed check bits against the bits that were retrieved (in your case, retyped by the user). If all the bits match, no errors were detected. If the bits do not match, the following table summarizes the possible outcomes.

Check bits	Parity bit	Outcome
Match	Match	No single- or double-bit errors
Match	Mismatch	Parity bit corrupted, correctable
Mismatch	Mismatch	Single bit corrupted, correctable
Mismatch	Match	Double-bit error (uncorrectable)

Correcting a single-bit error: The third case in the table corresponds to the case where a single-bit error has occurred: both the parity bit and the check bits have a mismatch between the recomputed and retrieved bits. In this instance, the difference in check bits is called the *syndrome*, and can be used to identify the erroneous bit. The following table (slightly rearranged from the handout for programming assignment two) identifies with X

each of the bits that is used to compute each of the check bits. The X's correspond to a '1' in the binary bit index for that column. Hence, a mismatch in a given check bit indicates that one of the columns with an X bit in that row is incorrect (for example, a mismatch for C₈ indicates that a bit in column 8, 9, 10, 11, or 12 is corrupted, since each of those indices has a '1' in the 8's digit). Hence, by examining all four of the check bits, one can identify the exact column that experienced the corrupted bit.

For example, a mismatch in check bits C₁ and C₈, but a match in check bits C₂ and C₄, indicates that the error must occur in an odd-numbered column (since C₁ didn't match), eight or higher (since C₈ didn't match), but not ten or eleven (otherwise C₂ would not have matched), and not twelve (otherwise C₄ would not have matched). This leaves only one possibility: column 9. As it turns out, this is the binary number expressed by the syndrome, if the syndrome is written with a '1' for each mismatched check bit, and a '0' for each matching check bit (so, for our example where C₈ and C₁ are mismatched, the syndrome would be 1001, or a binary 9).

Bit index	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100
	0	1	2	3	4	5	6	7	8	9	10	11	12
Bit name	p	c ₁	c ₂	D ₁	c ₄	d ₂	d ₃	d ₄	c ₈	d ₅	d ₆	d ₇	d ₈
C ₁		X		X		X		X		X		X	
C ₂			X	X			X	X			X	X	
C ₄					X	X	X	X					X
C ₈									X	X	X	X	X
P		X	X	X	X	X	X	X	X	X	X	X	X
Example	0	1	0	0	1	1	1	0	1	0	1	1	0

This *syndrome* can be computed using the logical XOR (exclusive-or) of the retrieved check bits and the recomputed check bits, and can then be used as the bit index that identifies the corrupted bit (as shown in the table above). Your forward-error correction logic will then need to flip that bit, assuming the conditions for a single-bit error have been satisfied. Note that if the parity bit is stored in column 0, this will automatically capture the case of the parity bit being corrupted as well (simply flip the 0th bit to correct the error).

Your program should start at location **0x3000** in memory.

Submitting your program: You should submit a programming project report that includes a listing of your program. The report should describe the problem and your solution, identify problems you encountered, and documenting the algorithm you chose to implement. The assembly language program listing must be commented so it is easily readable by the grader. Finally, your report should also include a section that describes how you tested your program, including a table that lists the input testcases you used and the corresponding outputs.

The written report is due by in class on Friday, May 5, 2006 (last day of class).

Grading: This assignment will be graded according to the following table. You must include a statement of work that clearly specifies the work done by each of the team members; this statement must be signed by all team members.

	Points possible	Points earned
Written report	40	
- Introduction/overview	5	
- Problem description	5	
- Solution description	5	
- Testing approach	15	
- Report quality	5	
- Statement of work	5	
Program listing	35	
- Functionality	20	
- Readability	5	
- Comment quality	10	
Total	75	