BLG 413E - System Programming Project 1

Due 08.11.2021 23:59

Policy:

- In this homework, you will work with NASM on a 32-bit Linux, same as what you have used in class and recitations. Projects, which use another assembler or incompatible with 32-bit Linux platform will not be evaluated.
- Upload your solutions through Ninova. Homework files sent via e-mail and late submissions will not be accepted.
- If you have any questions, please use the message board.

Project Description:

In this project, you are asked to implement basic encoder and decoder modules, which utilize the following hamming coding scheme by using Intel assembly.

$$H = \begin{bmatrix} H_{mr} \\ \bar{I}_r \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Assume that for the given scheme above, you will have 4-bit of parity (r) for every 4-bit of message and you will be using given H_{mr} matrix to determine the parity bits. (Note that matrix H should not have duplicate lines)

- **To encode** your message, you should multiply given input/message with H_{mr} matrix and use the result as your parity. For the example above, data and parity map is calculated as follows:

Data	0000	0001	0010	0011	0100	0101	0110	0111
Parity	0000	1101	1001	0100	0110	1011	1111	0010

Data	1000	1001	1010	1011	1100	1101	1110	1111
Parity	1100	0001	0101	1000	1010	0111	0011	1110

- **To decode** the received message, you should multiply the received message with matrix H and calculate the syndrome. If your syndrome is not 000 (for the given example) that means received message contains errors. You should use error syndrome map to determine which bit contains the error. After you find the syndrome, error case is found by finding the corresponding row of the H matrix, which corresponds to syndrome. Last, received message should XOR with related error in order the correct it before filtering out the parity bits (since you are only going to encounter **single bit errors** in the project, you can just toggle the incorrect bit).

Error	0000 0000	0000 0001	0000 0010	0000 0100	0000 1000	0001 0000	0010 0000	0100 0000	1000 0000
Syndrome	0000	0001	0010	0100	1000	1101	1001	0110	1100

Example messages are given below:

Data: 0000 **Parity:** 0000
Encoded Message: 0000 0000
Corrupted Message: 0**1**00 0000
Syndrome: 0**1**00 0000 x H = 0110
Error: 0100 0000
Corrected Message: 0100 0000 $\bigoplus$ 0**1**00 0000 = 0000 0000

Data: 1100 **Parity:** 1010
Encoded Message: 1100 1010
No corruption this time!
Message: 1100 1010
Syndrome: 1100 1010 x H = 0000
No Error!

Implementation Details

Main C file given to you reads the input files (H matrix and initial message), calls the **encode_data** function (which you are going to write in assembly), which takes H matrix and *initial* message as input, and get the *encoded* message as the function output. Between encoding and decoding, some bits of the encoded data may be corrupted by the main program. After the corruption process, main program will call the **decode_data** function (which you are going to write in assembly), which takes H matrix and *encoded* message as input, and get the decoded message and the error status. See the comments in the C file for details about functions and their parameters. In order to obtain your final executable, you will compile code files separately and link them together at the end as shown in the course/practice session.

You should not modify the given C file, if you think there is an error or something is unclear in this file please contact the course assistant.

Example scenario is given below:

Assume that you read 0000 1100 as input stream and $\begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$ as H. Then in your encode function you should divide your input stream into 4-bit chunks (**First** part: 0000 **Second** part: 1100). After calculating the parity map from H, you should add parity to each part. First byte of the encoded message becomes 0000 0000 and second byte of the encoded message becomes 1100 1010. Then we will corrupt your data accordingly, 0**1**00 0000 1100 1010. In your decode function you will find the syndrome and error and return a bit stream containing 1's for corrupted bit and 0's for uncorrupted bit so your error status output for this scenario should be:
0**1**00 0000
0000 0000

Submission Details

- You are required to implement the given functions in Intel assembly. The main program given to you will handle all file read and standard I/O operations, so you do not need to read or write anything from the assembly.
- You should not write the requested encoder-decoder codes in higher-level language (like C, C++) and then, transform it into an assembly code for your project submission. Accordingly, the codes generated by compiler will not be accepted and this will be considered as cheating.
- Every group member is required to submit source code file(s) through the Ninova system as a zip file.
- Any form of cheating or plagiarism will not be tolerated. This includes actions such as, but not limited to, submitting the work of others as one's own (even if in part and even with modifications) and copy/pasting from other resources (even when attributed). Serious offenses will be reported to the administration for disciplinary measures.