Project 2: BSD Checksum Due: Thursday September 30, 2004

Objective

The objective of this programming project is to practice designing your own loops and branching code in assembly language and to gain greater familiarity with the i386 instructions set.

Background

Checksums can be used to detect corrupted files. A file might be corrupted during transmission through a network or because the disk drive where the file is stored is damaged.

The BSD Checksum algorithm uses a 16-bit checksum. Initially, the value of the checksum is zero. For each byte of the file (in sequential order), the checksum is rotated 1 bit to the right and the byte is added to the checksum. The value of the checksum after the last byte of the file has been processed is the checksum of the file.

If the checksum of a file changes, then you know that its contents have been altered. However, it is possible for two different files to have the same checksum (since there are only 64K different values for a 16-bit checksum but many more possible files). So, having the same checksum does not guarantee that the file has not been corrupted. A well-designed checksum algorithm should be able to indicate the most common types of file corruption (e.g., transposed bits, single bits flipped).

Assignment

Write an assembly language program that computes the BSD checksum (algorithm given above) of the `stdin` file. You should output the checksum as a 16-bit binary number to `stdout`. The intention is for you to use Unix input/output redirection:

```
./a.out <ifile >ifile.checksum
```

The value of the checksum can be examined using the `hexdump` command.:

```
hexdump ifile.checksum
hexdump -e '1/2 "%u\n"' ifile.checksum
```

The first `hexdump` command gives the result in hexadecimal. The second `hexdump` command gives the value in decimal. (It is a challenge to alias the second command in Unix.)

Some details:

- Your program must read a block of bytes from the input. You should not read from the input one byte at a time. (It would be terribly inefficient).
- You may assume that when the operating system returns with 0 bytes read that the end of the input file has been reached.
- On the other hand, you may not assume that the end of the file has been reached when the operating system gives you fewer bytes than your block size.

Implementation Notes

- You can check your program using the `sum` command which prints out the BSD checksum of the file in decimal. (No, you may not call the Unix sum command from your program.)
- Look up the rotate right instruction in the Intel manual to make sure that you are using the correct rotate instruction.
- The BSD checksum algorithm involves adding an 8-bit value to a 16-bit value. Make sure you are doing this correctly.

- You will have two nested loops. The outer loop reads blocks from the input until the end of the file. The inner loop processes one character at a time. Decide ahead of time how the loops are controlled, which value is stored in which register or memory location.
- Record some sample runs of your program using the Unix script command.

Turning in your program

Use the UNIX `submit` command on the GL system to turn in your project. You should submit two files: 1) the assembly language program and 2) the typescript file of sample runs of your program. The class name for submit is `cs313_0101`. The name of the assignment name is `proj2`. The UNIX command to do this should look something like:

```
submit cs313_0101 proj2 checksum.asm typescript
```