Laboratory 3 8086/8088 Programming

Last Revised: June 2000

1 Objectives

With the completion of this lab, you should be able to:

- Write subroutines.
- Use the debugging features of the MUN-88 Monitor to locate any bugs in your code.
- Use assembly data directives to allocate memory for your data.

2 Introduction

In this lab, you will write a game called *Rat-bashing*, in which the objective is to quickly respond to a lighted LED by flipping the appropriate DIP switch. You will implement the key functions of the program as subroutines that you will call in your main program. Carefully read all of the specifications before starting your design – don't start coding until you have a good idea of exactly what is required. Writing brilliant code is great, but of little use if it does not solve the given problem.

As you write your program, take care to document your code using comments. Include a comment block at the beginning that includes information about your program, such as its title, your names, the date, a description of the program, and brief instructions. Each key assembly code instruction should have a brief comment that states its purpose. Use the EQU command to establish sensible labels, which can be used to self-document the code. You may wish to include the following code near the beginning of your .ASM file:

```
TRUE   equu   -1
```

This allows the use of the MUN-88 header file. Further details about the file can be found in the MUN-88 Monitor Reference Manual in section 6.2 (page 30).

3 Rat Bashing

3-1 Problem Statement

Write an assembly language program for the MUN-88 that simulates a carnival *rat bashing* game. The purpose of the real game is to use a play hammer to smash a wooden rat as it pops up through one of several holes. The rat only remains up for a moment before it goes back down. If the player hits enough rats, he or she wins a prize.

3-2 Specification

When the player types **go** at the MUN-88 Monitor prompt, the program must indicate it is about to start by blinking all the LEDs on and off three times. It must then wait for about 1 second with no LEDs lit.

Next, the program must show a single LED representing the rat to be bashed. It must then wait (for about a second) for the player to hit the rat by switching the corresponding DIP switch. Register a hit if the player flips *only* the correct DIP - if another switch is also flipped, consider it to be a miss. When a hit has been registered, the lit LED should quickly blink (off-on) 3 times. On a miss, do nothing special. Follow either case with approximately a second of all lights extinguished.

Repeat this for 8 rounds, with a variety of different LEDs lit. The pattern should not be predictable within a single game, but does not have to vary each time the game is played.

When done, display on the LEDs the rats that were hit. The LED number should represent the round. For example, if the player hit the rat on rounds 1, 2, 5, 7, and 8, then lights L1, L2, L5, L7, and L8 should light. If all 8 rats were hit, show a fancy display – something worthy of the feat that the player has accomplished.

The program must include at least the following three subroutines; you have some flexibility with your implementation, and may include more functionality than listed here:

DisplayRat	- displays the next rat on the LEDs
HammerWait	- waits for a limited time for the player to flip a switch
ShowScore	- displays the rats that were hit during the game

3-3 Implementation Notes & Suggestions

Your code should consist of a main block of code, in which you call various subroutines. The main code should first initialize any registers or memory locations, and then enter a loop. After looping, finish by displaying the score or fancy display on the LEDs.

If you wish to store all 8 LEDs in memory using DB, and then manually set them at assembly time, that is fine. On the other hand, if you can determine a way to randomly generate the next rat location, please do so!

When you design your subroutines, you may require that a certain register or memory location be set before the main code calls the routine. For example, your *DisplayRat* routine might require register DI to point to the memory location where the next rat is stored, or perhaps register AL will store the value to display.

3-4 Debugging

Use the debugging skills you acquired in Lab 2 to hunt down any anomalies in your program. Do **not** call over a TA as soon as you realize that the program is not working as you expect. Use your list code, and the Monitor commands **single** and **brk**, to pinpoint what instructions are behaving differently than you would like. And make sure that the code on the MUN-88 board is of the same version as the code you are looking at in the .LST file!

4 Submission

When you have finished your program, demonstrate it to a TA. When the TA is satisfied that it is working correctly, print off your *list file* and he or she will sign it for you. If you submit a version of your program that does not meet the specifications, you will not be able to receive full marks on the lab.

If you wish to submit your .ASM file to overcome layout issues with the list file, you may, but it is not required.

As your report, **submit a short document** describing your program. Include a brief overview of your program (what it is and how to use it). Explain any major design decisions. List all of subroutines, giving their names, brief descriptions, any registers or memory locations that must be set prior to calling, and any registers or memory locations that will be set after the return instruction.