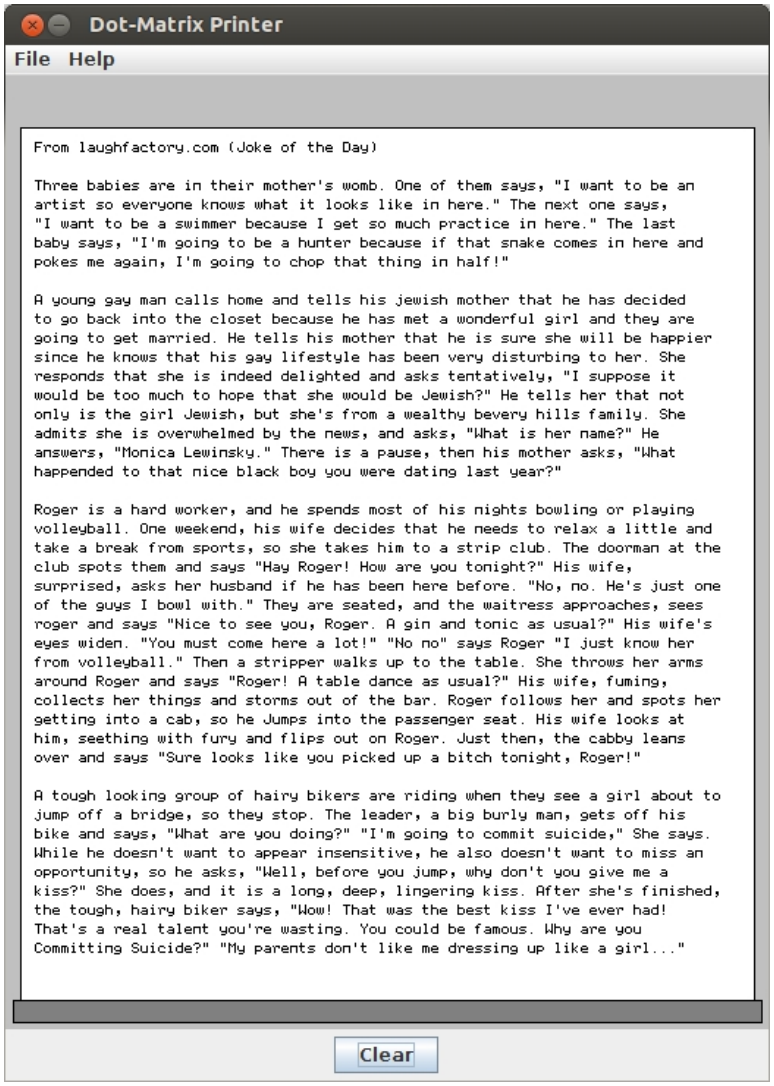
Project 2 - Dot-Matrix Printer

CS 0447 — Computer Organization & Assembly Language

Check the Due Date on the CourseWeb

The purpose of this project is for you to practice writing assembly language to interact with output hardware. The hardware for this project is a primitive dot-matrix printer. The Dot-Matrix Printer (Mars Tool) that we are going to use for this project is shown below. This tool can be found in `DotMatrixPrinterRegister.zip` located in the CourseWeb under this project. Extract all files to your `[..]/mars4_5/mars/tools` directory. If you extract all files to the right directory, when you run the MARS program, you should see "Dot-Matrix Printer (Register) V0.1" under the "Tools" menu.



Introduction to the Dot-Matrix Printer (Mars Tool)

The Dot-Matrix Printer (Mars Tool) is a very primitive printer. The print head consists of only one pin. Therefore, it can only print one dot at a time. This print head only prints when it moves from left to right. Once the print head moves all the way to the right, it will stop printing, move the print head back all the way to the left, feed the paper up just a little bit (equal to one dot), and then continue printing. For this printer, each line consists of 480 dots.

To print to this printer, we have to write a 32-bit data to the register `$t8` and then set the register `$t9` to 1 to tell the printer that the data is available to be printed. When the printer finishes printing, it will set the register `$t9` to 0 to let you know that it is ready to receive the next data. A 32-bit data will be mapped to 32 dots on the paper. A 1 represents a black dot, and a 0 represents no dots. For example if you set the register `$t8` to 0xFF00F00F and set the register `$t9` to 1, the printer will print (from left to right), 8 black dots, 8 white dots (not actually print), 4 black dots, 8 white dots (again, not actually print), and 4 black dots. Again, **DO NOT** forget to wait for the register `$t9` to change back to 0 before you send a new 32-bit data.

Since there are 480 dots in one line, to print the whole line, you need to send the total of 15 32-bit data to the printer (15 * 32 = 480). In other words, one line of dots is equal to 15 words.

The "Clear" button is used to reset the printer. Note that you should stop your program before click the Clear button. Otherwise, the printer may keep printing.

What to Do?

For this program, write a MIPS assembly program that asks user to enter a filename to be printed, reads the file (assuming that it is a text file) and prints it to the printer. Note that you cannot simply send a series of characters to the printer. The printer does not understand characters. It only prints lines of dots. So, you job is to map text file into a series of 32-bit data that can be used to send to the printer.

For this project, a character should consists of 8 lines of dots and the width of each character is 5 dots. The space between two characters is 1 dot and the space between two lines of characters is 5 dots. For example, to print "Hello" in one line and "World" in the next line, your data may look like the Figure 1. Note that if we remove all 0s from the above data, and imaging that 1s are dots on the paper, the result of data in Figure 1 sending to the printer is shown in Figure 2.

For simplicity, we give you a starter file that contains data representations of all characters. In the starter file, there are eight labels in the data segment (`line1` to `line8`). Each line contain 96 bytes of data organized into 24 words. It is also organized according to ASCII table from ASCII number 32 (space character) to ASCII number 127. Each byte represents a line of each character.

For example, suppose we read character `'H'` and the value in ASCII is 72 (in decimal). Subtract 72 by 32 and we get 40 which is the number of bytes away from each line label. That is, at the address `line1` plus 40 will be the data representation of the first line the character `'H'`. At the address `line2` plus 40 will be the data representation of the second line of the character `'H'` and so on. Note that the data representation for each character at each line is 8 bits (1 bytes) but a character width is only 5 dots. So, you only need the top 6 bits to represent the character and one dot space. Note that the last three bits are always 0s.

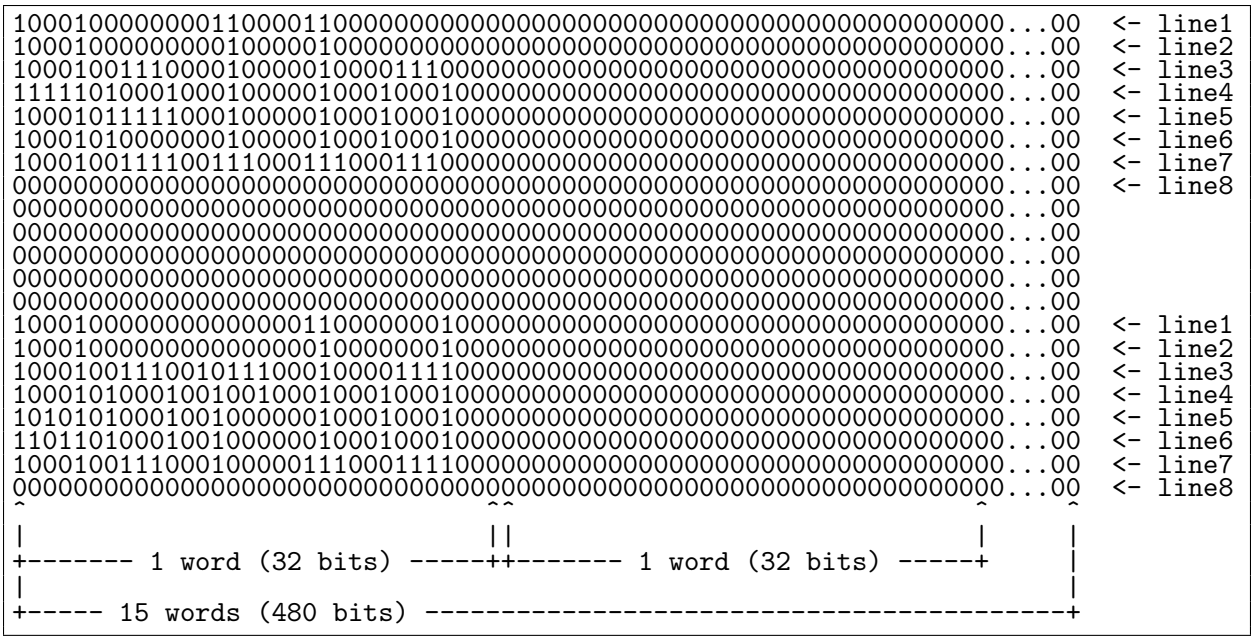


Figure 1: Example of Data

Text Files

We supply you a set of text files. Note that these text files are carefully crafted so that there will be no line that contains more than 80 characters. Each line ends with the new line character (the value of new line character is 10). Note that one character takes 6 dots (5 dots data + 1 dot space between characters). The printer can print only 480 dots. Therefore, one line of text can only fit $480/6 = 80$ characters.

Requirements

The following are requirements that you **must** follow:

- Your program should be named `printer.asm`
- Since the Dot-Matrix Printer (Mars Tool) uses registers `$t8` and `$t9`, **do not use** these registers for any other purpose.
- You must use the following system calls
  - System call 8: read string from keyboard
  - System call 13: open file
  - System call 14: read from file
  - System call 16: close file
- You must separate your program into various functions and each function must follow calling convensions.

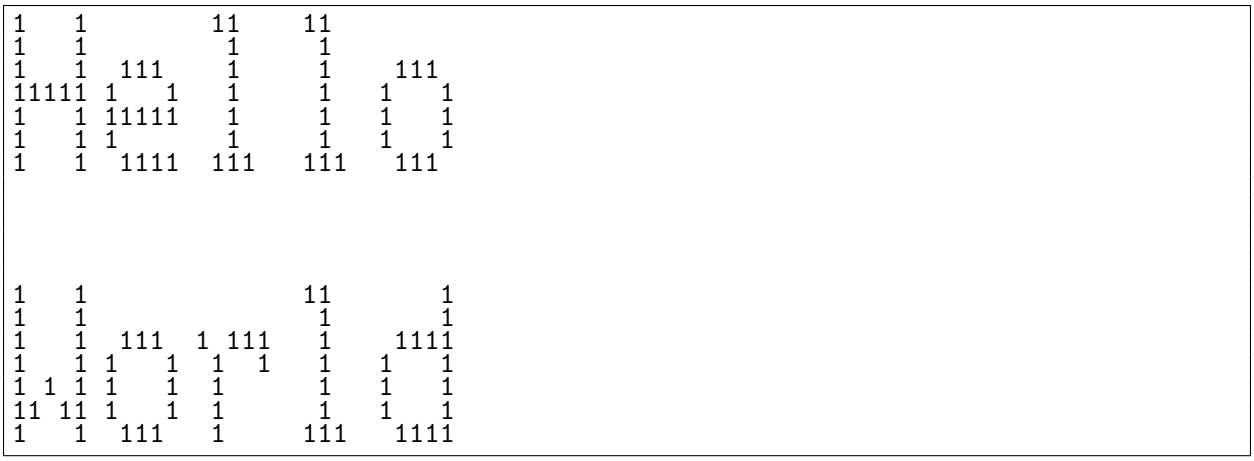


Figure 2: Result on the Printer

HINTS

You should separate your program into various fuctions based on its functionality. For example, the following is a suggestion. You do not have to follow this.

- Function `_readLine`: This function should take two arguments, a file descriptor and an address of 80-byte buffer. It should return a value 1 if it encounters the end of file. Otherwise, return 0. **Note** that this function should always fill up the 80-byte buffer. For example, if a line contains 20 characters, follows by a new line character (ASCII value 10), the first 20 bytes of the buffer should be those characters and the next 60 bytes should be spaces (ASCII value 32). In this situation, the function should return 0. Same situation if it encounters an end of file. For example, if the last line of the file contains 30 characters and that is the end of file, the first 30 bytes of buffer should be those characters and the rest should be spaces. Return 1 for this case. **Hint**: You should read the file one byte at a time. Each time you read a byte, check whether it is an end of file (return value 0 from the syscall). If it is not, check whether it is a new line or a regular character. Handle each situation accordingly.

- Function `_printLine`: The purpose of this function is to print one line of **dots** based on the 80-byte buffer and the line data. This function should take two arguments; (1) the address of 80-byte buffer, and (2) the address of the line data to be used (e.g., the address of labels `line1` to `line8`). This function should send data to printer one word (32 bits) at a time for 15 words.

- Function `_printBuffer`: The purpose of this function is to print 80-byte buffer to printer. Therefore, it takes one argument which is the address of the 80-byte buffer and returns no value. Note that this function should call the function `_printLine` eight times. For example, suppose the value stored in `$s0` is the address of the 80-byte buffer:

```
        add  $a0, $zero, $s0
        la   $a1, line1
        jal  _printLine
        add  $a0, $zero, $s0
        la   $a1, line2
```

```
        jal  _printLine
    :
        add  $a0, $zero, $s0
        la   $a1, line8
        jal  _printLine
```

- Function `_printSpaceBetweenLine`: This purpose of this function is to simply print space between lines. Note that the space between two lines of character should be 5-dot high. So, you simply have to send five fifteen-word data to printer where each word is simply 0. Recall that you can only send one word at a time. This function should take no argument and return no value.

Submission

The due date of this project is stated on the CourseWeb. Late submissions will not be accepted. You should submit the file `printer.asm` via CourseWeb.