EE 109 Lab 8a – Conversion Experience

1 Introduction

In this lab you will write a small program to convert a string of digits representing a number in some other base (between 2 and 10) to decimal. The user can specify the string of digits and the base. You will then convert it to decimal while also checking for invalid digits (i.e. those not in the range of 0 to base-1).

2 What you will learn

This lab is intended to teach you the basics of writing assembly language code that includes various control structures (loops, conditionals, etc.) as well as how to use appropriate syscalls for input & output via the MARS simulator. Your next lab will build on this kernel of assembly code so please work hard to get it working.

3 Background Information and Notes

Syscall's: A 'syscall' instruction is a mechanism by which user applications can call specific OS routines, usually for I/O purposes. This is accomplished by simply using the instruction 'syscall'. What OS function gets called is determined by the values of specific registers (in MARS it is the value of $v0 and often $a0 and $a1). The value in $v0 indicates which syscall operation should be executed with additional arguments in $a0 and $a1. Table 1 shows several useful syscall's, while the online help in MARS has a list of all available syscalls which you should look over as you do your other labs. Thus before you execute the syscall instruction you would need a few other instructions to initialize $v0, $a0, etc. to the desired values.

Syscall	$v0	Additional arguments	Notes
'read string'	8	$a0 = address of input buffer $a1 = maximum # of characters to read	Leaves the LF character at the end of the string
'read integer'	5	No additional input arguments. The integer read from the keyboard will be written into $v0 (overwriting the 5)	
'print string'	4	$a0 = address of null terminated ASCII string	
'print character'	11	$a0 = ASCII char. to print	
'print integer'	1	$a0 = integer to print	

Table 1- Useful Syscalls

4 Procedure and Program Requirements

The requirements that your program must meet are as follows:

a. Your program shall prompt a user to enter an ASCII string of digits, maximum 10 digits without spaces (the 'Enter' key / LF character followed by the null character '\0' will terminate the string)

b. Receive the input. You can assume the user enters a string composed of 10 or less digits. You need not check for non-digit characters or worry about strings that are too long (though they may be shorter than 10 digits). **Note:** We will not enter a number that results in 32-bit overflow (i.e. is too big for its decimal representation to fit in 32-bits).

c. Convert the number to decimal. Recall a number: $a_{n-1}a_{n-2}\ldots a_1a_0$ in base r can be converted to decimal through the formula:

$$X_{10} = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_1r^1 + a_0r^0 = \sum_{i=0}^{n-1} a_i r^i$$

However, this would require us to compute powers of r. Instead we can use an iterative process to sum up the number multiplying by r each iteration. The above summation can equivalently be shown as:

$$X_{10} = \big((r(r(r(a_{n-1}) + a_{n-2}) + a_{n-3}) + a_{n-4}) \cdots \big)$$

This can be summarized by the recursive formulation:

$$Let\ X_{10}(0) = 0$$

$$X_{10}(i+1) = rX_{10}(i) + a_{n-i-1}$$

Essentially, if we have an n-digit number we can start a sum at 0 (i.e. X(0) = 0) and then repeatedly multiply that value by r and add in the next digit (i.e. a_{n-i-1}) a total of n times.

A C-code version of the program is attached at the end of this document.

Note that we have received ASCII digits (i.e. '0' = 0x30, '1' = 0x31, etc.) and need to convert each digit to a number.

d. You must also add a bit of error checking. Recall that valid digits in base r range from 0 to (r-1). You should check if the current digit being processed is out of range, and if so, start the sum back at 0 (essentially restart your conversion process with the remaining part of the string and discarding the effect of any prior digits).

e. At the end of your computation output the message pointed to by label 'omsg1' followed by outputting the integer representing the decimal equivalent. You should do this with two separate syscall sequences loading the appropriate $v0, $a0 values prior to your 'syscall' instruction.

f. We have provided skeleton code to do most of the I/O at the start of the program. You should NOT change this code.

g. Comment your code enough for the TA/Instructor to understand your approach/intent to solving the problem.

h. Try to organize your code in a coherent and make it readable using helpful labels, good formatting/alignment, etc.

5 Programming and Submission Guidelines

1) A program skeleton can be downloaded from our website. Use it as a baseline. Review the C/C++ implementation to help you.

2) You should be careful about a few things. First, the numbers you will be receiving as input will be *ASCII representations* of 0-9. You should think carefully whether it is beneficial to convert them to their actual 0-9 binary representations or leave them as their ASCII representations. You can convert to integer values 0-9 by subtracting the 0x30 from each ASCII digit. Second, be careful to use appropriately sized load and store instructions. ASCII characters are each a byte.

3) You may use any pseudo-instructions you desire. "li – load immediate", "la – load address", and branch pseudo-instructions will be helpful.

4) We've included a C-code version on the page below. Take the highlighted area and try to take each aspect (i.e. the code in the loop, the code in the if statement, etc. and use the examples from class to translate each piece. Then if you like you can optimize it a bit.

5) Test your program on several input patterns. Consider what would be good test cases…maybe something with the max number of characters, something with a single characters, and something with one or more error conditions (i.e. digits not in the range of the base).

6) Submit your source file using the link on our website.

6 Review

None

See below for sample program executions.

Sample Program Execution(s):

```
Enter a number in another base using at most 10 digits: 746
Enter the base number: 8
486

Enter a number in another base using at most 10 digits: 881178
Enter the base number: 9
525761

Enter a number in another base using at most 10 digits: 0110101
Enter the base number: 2
53
```

Sample Program Execution (with an error):

```
Enter a number in another base using at most 10 digits: 112012110
Enter the base number: 2
6
```

7 Appendix

This is an equivalent C/C++ implementation.

```
#include <iostream>
#include <cstdio>
using namespace std;

int main()
{
  int base, ans, len;
  char num[12];

  cout << "Enter a number in another base using at most 10 digits: ";
  fgets(num,12,stdin);

  cout << "Enter the base number: ";
  cin >> base;
  // This is what you'll need to write in assembly yourself
  int sum = 0;
  char* pstr = num; // pointer to the start of the num array
  while(*pstr != '\n'){
    int dig = *pstr - '0';
    if(dig < base){
      sum *= base;
      sum += dig;
    }
    else {
      sum = 0;
    }
    pstr++;
  }
  cout << "Decimal equivalent: " << sum << endl;
  return 0;
}
```

Student Name: _____

Item	Outcome	Score	Max.
Palindrome check			
• Loops through characters and stops appropriately	Yes / No		4
• Performs conversion algorithm correctly	Yes / No		4
• Handles invalid digits correctly	Yes / No		2
• Prints result correctly	Yes / No		2
Instructor Tests			
• Program works correctly for input set 1	Yes / No		2
• Program works correctly for input set 2	Yes / No		2
• Program works correctly for input set 3	Yes / No		2
• Program works correctly for input set 4	Yes / No		2
SubTotal			20
Late Deductions			
Total			
Open Ended Comments:			