Lab2 Report

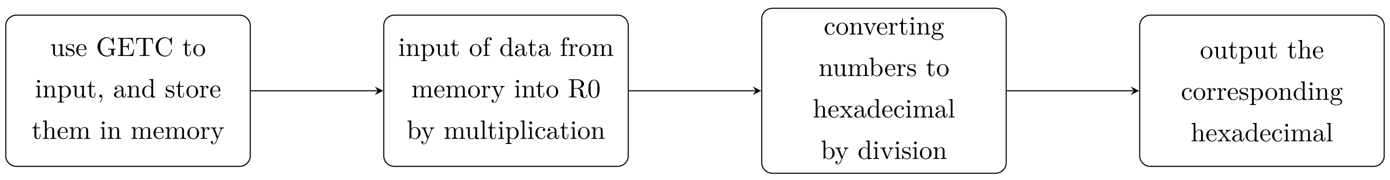
作者: 罗文杰
专业: 计算机科学与技术
学号: 3210102456

1 Introduction

In this lab, we will write a program to convert numbers to hexadecimal.

2 construst

The construsts of the program is as follows.


use GETC to input, and store them in memory
input of data from memory into R0 by multiplication
converting numbers to hexadecimal by division
output the corresponding hexadecimal

3 Improvment

There are two points to improve.

3.1 fast multiplication

First, in step 2, we need use multiplication to take numbers from memory into R0, and it needs cumulative addition to realize multiplication convincing that the input numbers may have many bits. And Repeated addition is too time consuming, so we need to use fast multiplication to achieveit.

When multiplying a number X with some constant, we can decompose the constant into a number of numbers that add up to a base of two, and then multiply X with each of these numbers and add them again. This algorithm reduces the number of operations because X can be multiplied with 2^k by having X shifted left by k bits to achieve this.

In this program, $10 = 2 + 8$, so we can have R0 shifted one place left to add to the number shifted 3 places left. This algorithm requires only 4 operations, which is a significant improvement over the original 10.

The codes is as follows.

```
;The CHAR1 already had the input
DONE    AND R0, R0, #0 ; binary num
        LEA R3, CHAR1
LOOP    LDR R1, R3, #0
```

```
        ADD R3, R3, #1
        ADD R0, R0, R1
        ADD R2, R2, #-1
        BRz LOOPUOT
        ADD R1, R0, R0 ; shift 1 place
        ADD R0, R1, R1 ; shift 2 places
        ADD R4, R0, R0 ; shift 3 places
        ADD R0, R4, R1 ; R0 = 2 * R0 + 8 * R0
        BRnzp LOOP
; R0 has the number
```

3.2 right shift

Second, in step 3, we need to use addition to achieve the division operation, but this is too time consuming.

To cover this problems, we have another solution: Let the data in R0 be cyclically shifted to the right, each time by 4 bits and then let the result be an iso-or operation with x000F to store the result in memory.

However, there is no right shift instruction in LC-3 and we need to use a left shift instead of a right shift. Before each left shift, identify the first position of the data in R0, if it is 1, the result will be added by 1 after the left shift, otherwise it will be shifted directly left without adding 1.

The codes is as follow.

```
LOOPUOT LEA R3, MARK ; output char addressing -1
        ADD R2, R2, #5 ; counter
        AND R5, R5, #0
        LD  R4, PDIV ; mask
; bigin to trans
TRANS   ADD R5, R5, #4
        AND R1, R0, R4
        STR R1, R3, #0
        ADD R3, R3, #1
        ADD R2, R2, #-1
        BRz OUTPUT
RIGHT   ADD R0, R0, #0
        BRzp ZARO
        BRnzp ONE
ZARO    ADD R0, R0, R0
        BRnzp ALOOP
ONE     ADD R0, R0, R0
        ADD R0, R0, #1
ALOOP   ADD R5, R5, #-1
        BRz TRANS
        BRnzp RIGHT
; output
```