Mastermind

Project1

CSC- 11 – 48598 Assembly

Tsz,Kwan

29 – October – 2014

Content

1.Introduction………………………………………..….3  
Rule and Gameplay  
Thoughts after Program

2. Development………………………..…………………4  
Approach Strategy

3. Research…………………………………………..…..4

time, srand, rand

strcmp

4. Variables list…………………………….………….…5

5. Global External…………….………………………….5

6. Topic Covered (Checklist)……………………………6

7. Pseudo Code………………………………………...7-8

8. Flowchart…………………………………………..9-14

9. Code……………………………………………...15-28

1. Introduction

**Rules and Gameplay**

This program is like the original mastermind. The original mastermind is let the player guess certain time of the colors in 4slots. Each time after the player guess, it will tell the player how many colors are at the right place, and how many colors are the right colors excluded the right place. I change the colors to numbers (0-9). The player has 10 times to guess to number. If they can’t guess the right number in 10 times, the program will end and announce the player lose and the correct number.

**Thoughts after Program**

The game was simple. The main ability the player need is logical thinking because the next guess is based on the result of the previous guesses. The player needs to use the previous results and cross comparison to inference the right digits.

It really makes me happy when the program runs successfully. However, there are some improvements which is using array store the input as a string which means the player doesn’t have to input 3 digits with space. There are only 3functions which are random number, modified, and checking functions. I can put them back to main by using branch, but it is hard to debug. If I make them as functions, I can comment them out to check whether the main function works or not. Then put the functions back one by one.

I was going to make 4 digits number; however, I didn’t understand how to use array in assembly, so I make 3 digit number. If I had known how to use array, it would have been much easier, and the code would have been less.

2. Development

Approach Strategy

I change my topic to mastermind because it is easy to write a very simple mastermind even though I don’t know how to use array.

The way to check the result is check the right place first. If they are match, the program will replace the correct digit to -1 to avoid it matches again and increase counter. After checking the place, the program will check the numbers which the same places of original numbers aren’t -1. If the numbers are match the other places, the original digits will be change to -1 and increase another counter. After finish the comparison, it counts the number of guesses, displays how many digits at the right place and how many digits are correct. If the right place counter is equal to 3 that mean the player input the correct answer, the program will display the correct number and how many times the player guesses. After that, it asks the player whether he wants to play again.

3. Research

1. time, srand, rand – I need to use a random number generator to make a 3 digit number, but I don’t know how to do it in assembly. I found that I can use printf and scanf in assembly, so I try to use time, srand, and rand which are c function in assembly language. It is easy than set the clock and make my own generator.
2. strcmp – This is c function too. I need to check the y/n answer to show the correct number and run the program again. I tried, but I didn’t know how to use binary to compare the input. Therefore, strcmp really helps a lot.

4. Variables list

|  |  |
| --- | --- |
| **Registers / Variables** | **Description** |
| r7-r9 | store original 3digit number for recover |
| r10 | counter (10 chances) |
| r1-r3 | 3 digit input number and used to compare |
| r4-r6 | use to compare to input number (will change and recover) |
| r10 | correct place counter, before end of check function mov to r1 |
| r11 | correct number counter, before end of check function mov to r2 |
| a | store first input digit |
| b | store second input digit |
| c | store third input digit |
| ans | store input y/n |
| y | character ‘y’, use to compare to ans |
| n | character ‘n’, use to compare to ans |
| scan | scan format of 3digit |
| scanYN | scan format of string |

5. Global external

* printf
* scanf
* strcmp
* time
* srand
* rand

6. Topic Covered (Checklist)

|  |  |  |  |
| --- | --- | --- | --- |
| description | code | source | line |
| start 4 bytes boundary | .balign 4 | proj1.s | 7 |
| scan format | scanYN: .asciz “%s” | proj1.s | 28 |
| string | y: .asciz “y” | proj1.s | 30 |
| emit 4 bytes | a: .word 0 | proj1.s | 37 |
| label | askShow | proj1.s | 59 |
| str lr for return | push {lr} | proj1.s | 47 |
| get lr from return | pop {lr} | proj1.s | 193 |
| exit stage right | bx lr | proj1.s | 194 |
| load address into regiester | ldr r0, addr\_intro | proj1.s | 57 |
| call printf | bl printf | proj1.s | 58 |
| call scanf | bl scanf | proj1.s | 64 |
| call strcmp | bl strcmp | proj1.s | 67 |
| put value into register | mov r10, #0 | proj1.s | 56 |
| add | add r0, r0, r3 | divMod.s | 29 |
| subtraction | sub r1, r1, r2 | divMod.s | 30 |
| Logical shift left | mov r3, r3, lsl #1 | divMod.s | 43 |
| arithmetic shift right | mov r1, r0, asr #1 | randN.s | 15 |
| put registers into stack | push {r7-r9, lr} | proj1.s | 54 |
| pull register out from stack | pop {r7-r9, lr} | proj1.s | 86 |
| load the pointer into register | ldr r1, [r1] | proj1.s | 98 |
| compare and update flag | cmp r1, #0 | proj1.s | 110 |
| branch | b loop | proj1.s | 80 |
| branch if equal | beq loop | proj1.s | 72 |
| branch if not equal | bne askShow | proj1.s | 73 |
| branch if less than | blt invalid | proj1.s | 111 |
| branch if greater or equal to | bge lose | proj1.s | 150 |
| call function | bl randN | proj1.s | 50 |
| address | addr\_a: .word a | proj1.s | 196 |
| external | .global scanf | proj1.s | 215 |
|  | .global printf | proj1.s | 216 |
|  | .global strcmp | proj1.s | 217 |
|  | .global time | randN.s | 29 |
|  | .global srand | randN.s | 30 |
|  | .global random | randN.s | 31 |

7. Pseudo Code

do{

generate 3 digit number

store at r7-r9

push r7-r9 into stack

do{

ask for showing answer (easy to check)

if(yes)

display answer

else if (! no)

display invalid

}while(invalid)

reset counter

do{

do{

game start

input 3 digit

if(anyone of 3 digits isn’t 0-9)

invalid

}while(invalid);

reset place, num counter

if(r1==r4)

place++

r4=-1

if(r2==r5)

place++

r5=-1

if(r3==r6)

place++

r6=-1

if(place==3)

counter++

display counter and answer

else

if(r4!=-1)

compare r1to r4 and r5

change r4 or r5 if matched

num++

if(r5!=-1)

compare r2to r4 and r6

change r4 or r6 if matched

num++

if(r6!=-1)

compare r3to r4 and r5

change r4 or r5 if matched

num++

counter ++

display place and num

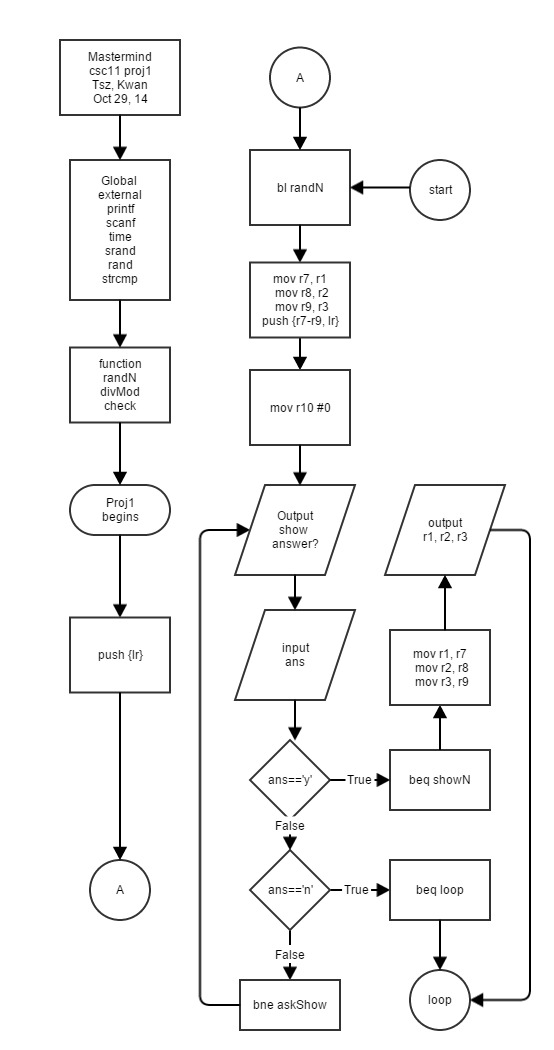
}while(counter<10)

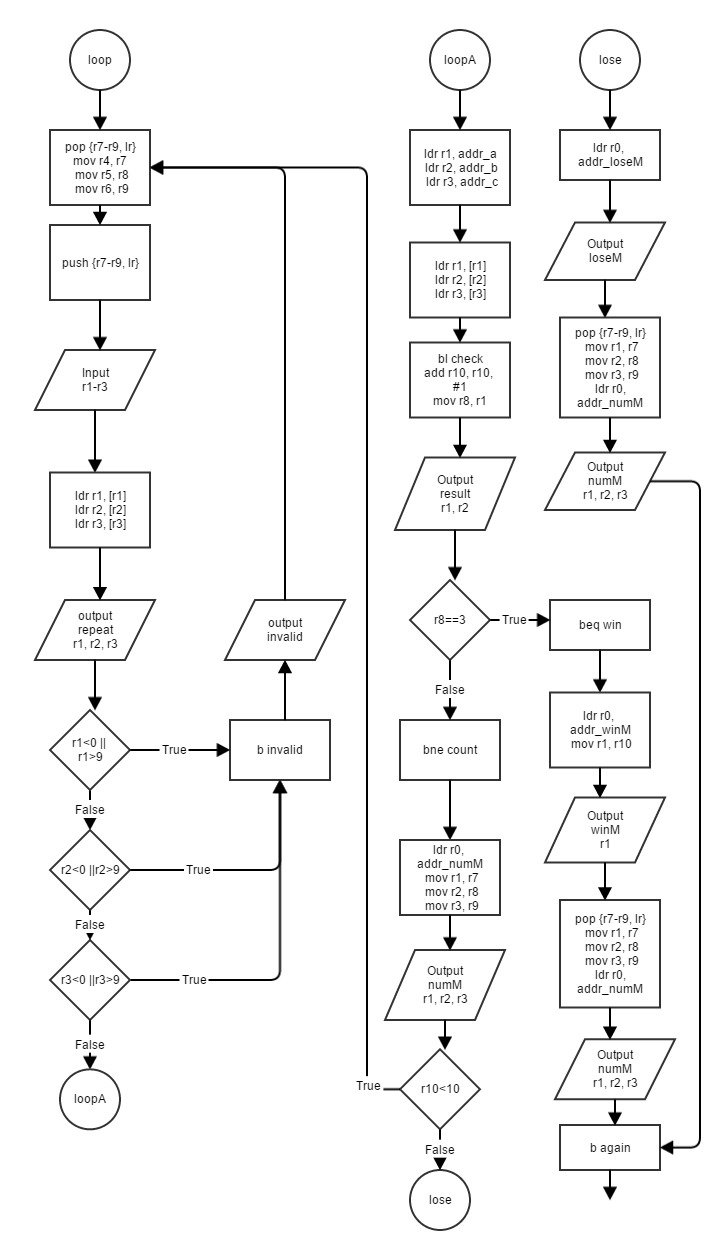
ask whether want to play again

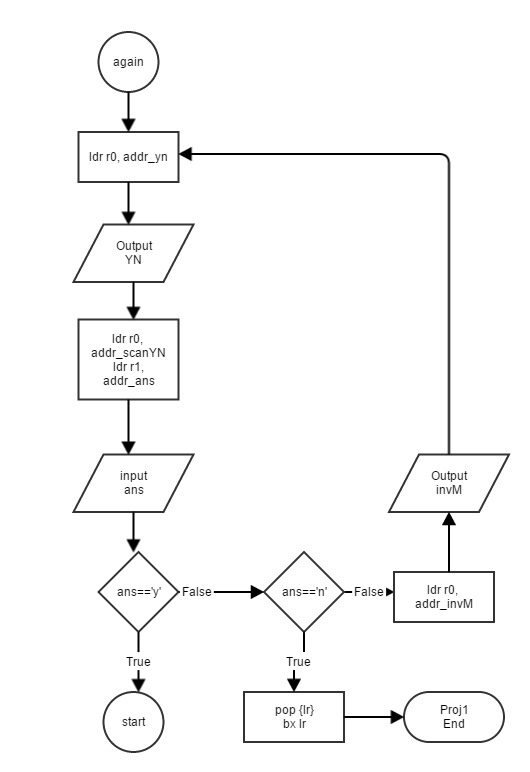
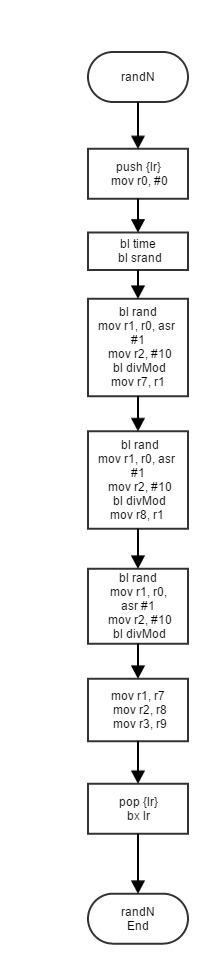
}while(yes)

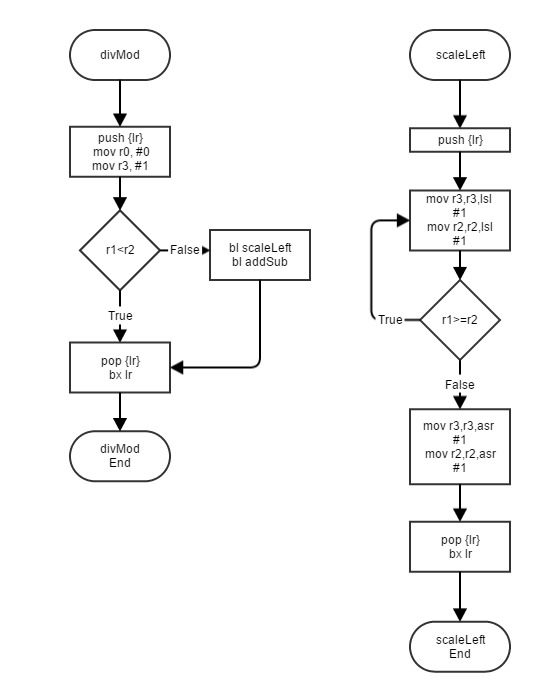
Program Ends

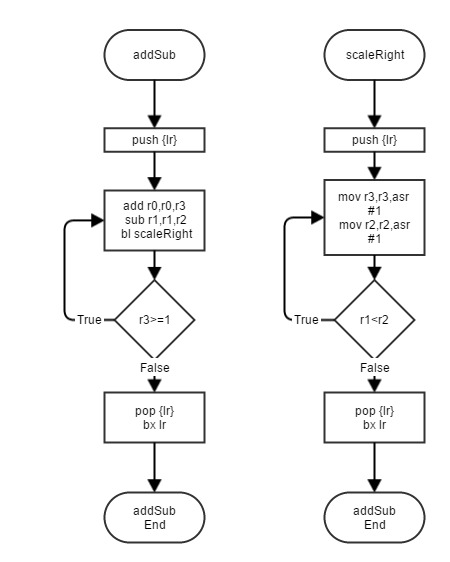
8. Flowchart

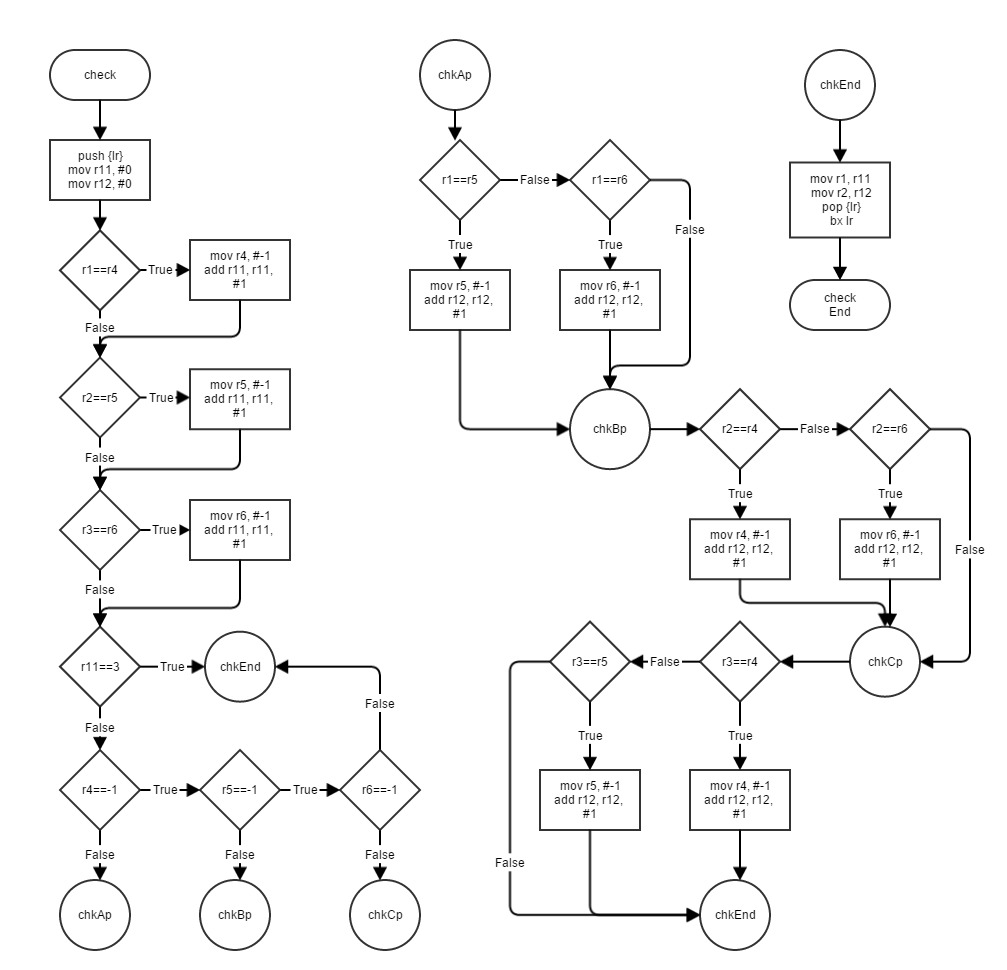












9. Code

/\* mastermind \*/

/\* csc 11 project 1 \*/

/\* author: Tsz, Kwan \*/

/\* Oct 29, 14 \*/

.data

/\* store input r1-r3 \*/

/\* store correct num in r4-r6 \*/

/\* r7 = counter \*/

.balign 4

intro: .asciz "Mastermind!\nIt is three digit number\n You have 10 chances\n"

.balign 4

show: .asciz "Do you want to show answer? (y/n)\n"

.balign 4

inM: .asciz "Please enter 3 digit number with space\n"

.balign 4

winM: .asciz "You win!\nYou guess %d times\n"

.balign 4

loseM: .asciz "You lose!\n"

.balign 4

repeat: .asciz "Your input is %d%d%d\n"

.balign 4

numM: .asciz "The number is %d%d%d\n"

.balign 4

scan: .asciz "%d %d %d"

.balign 4

invM: .asciz "Invalid input\n"

.balign 4

result: .asciz "%d digit(s) are at the right place\n%d digit(s) are correct number\n"

.balign 4

scanYN: .asciz "%s"

.balign 4

y: .asciz "y"

.balign 4

n: .asciz "n"

.balign 4

yn: .asciz "Do you want to play again? (y/n)\n"

/\* 3 digit input \*/

.balign 4

a: .word 0 /\* first input digit \*/

.balign 4

b: .word 0 /\* second input digit \*/

.balign 4

c: .word 0 /\* third input digit \*/

.balign 4

ans: .word 0 /\* y/n answer \*/

.text

.global main

main:

push {lr}

start:

/\* gen 3 digit \*/

bl randN

mov r7, r1 /\* original at r7-r9 \*/

mov r8, r2

mov r9, r3

push {r7-r9, lr}

/\* set counter \*/

mov r10, #0

ldr r0, addr\_intro

bl printf

askShow:

ldr r0, addr\_show /\* ask show answer \*/

bl printf

ldr r0, addr\_scanYN

ldr r1, addr\_ans

bl scanf /\* show or not \*/

ldr r0, addr\_ans

ldr r1, addr\_y

bl strcmp

beq showN

ldr r0, addr\_ans

ldr r1, addr\_n

bl strcmp

beq loop

bne askShow

showN:

ldr r0, addr\_numM

mov r1, r7

mov r2, r8

mov r3, r9

bl printf

b loop

loop:

pop {r7-r9, lr}

mov r4, r7 /\* recover \*/

mov r5, r8

mov r6, r9

push {r7-r9, lr}

ldr r0, addr\_inM /\* print input message \*/

bl printf

ldr r0, addr\_scan /\* scan input \*/

ldr r1, addr\_a

ldr r2, addr\_b

ldr r3, addr\_c

bl scanf

/\* repeat input \*/

ldr r1, addr\_a

ldr r2, addr\_b

ldr r3, addr\_c

ldr r1, [r1]

ldr r2, [r2]

ldr r3, [r3]

ldr r0, addr\_repeat

bl printf

ldr r1, addr\_a

ldr r2, addr\_b

ldr r3, addr\_c

ldr r1, [r1]

ldr r2, [r2]

ldr r3, [r3]

/\* check validation \*/

cmp r1, #0

blt invalid

cmp r1, #9

bgt invalid

cmp r2, #0

blt invalid

cmp r2, #9

bgt invalid

cmp r3, #0

blt invalid

cmp r3, #9

bgt invalid

/\* call check function \*/

ldr r1, addr\_a

ldr r2, addr\_b

ldr r3, addr\_c

ldr r1, [r1]

ldr r2, [r2]

ldr r3, [r3]

bl check

add r10, r10, #1

/\* print result \*/

mov r8, r1

ldr r0, addr\_result

bl printf

cmp r8, #3

beq win

bne count

invalid:

ldr r0, addr\_invM

bl printf

b loop

count:

ldr r0, addr\_numM

mov r1, r7

mov r2, r8

mov r3, r9

bl printf

cmp r10, #10

bge lose

blt loop

win:

ldr r0, addr\_winM

mov r1, r10

bl printf

pop {r7-r9, lr}

mov r1, r7

mov r2, r8

mov r3, r9

ldr r0, addr\_numM

bl printf

b again

lose:

ldr r0, addr\_loseM

bl printf

pop {r7-r9, lr}

mov r1, r7

mov r2, r8

mov r3, r9

ldr r0, addr\_numM

bl printf

b again

again:

ldr r0, addr\_yn /\* play again? \*/

bl printf

ldr r0, addr\_scanYN

ldr r1, addr\_ans

bl scanf

ldr r1, addr\_ans

ldr r0, addr\_y

bl strcmp

beq start

ldr r1, addr\_ans

ldr r0, addr\_n

bl strcmp

beq end

bne invalidYN

invalidYN:

ldr r0, addr\_invM

bl printf

b again

end:

pop {lr}

bx lr

addr\_a: .word a

addr\_b: .word b

addr\_c: .word c

addr\_inM: .word inM

addr\_winM: .word winM

addr\_loseM: .word loseM

addr\_numM: .word numM

addr\_invM: .word invM

addr\_result: .word result

addr\_yn: .word yn

addr\_scan: .word scan

addr\_scanYN: .word scanYN

addr\_ans: .word ans

addr\_y: .word y

addr\_n: .word n

addr\_show: .word show

addr\_intro: .word intro

addr\_repeat: .word repeat

/\* external \*/

.global scanf

.global printf

.global strcmp

/\* random number generator \*/

/\* store in r1 \*/

.global randN

randN:

push {lr}

mov r0, #0

bl time

bl srand

bl rand

mov r1, r0, asr #1

mov r2, #10

bl divMod

mov r7, r1

bl rand

mov r1, r0, asr #1

mov r2, #10

bl divMod

mov r8, r1

bl rand

mov r1, r0, asr #1

mov r2, #10

bl divMod

mov r1, r7

mov r2, r8

mov r3, r9

pop {lr}

bx lr

.global time

.global srand

.global random

/\* divMod \*/

.text

.global scaleRight

scaleRight:

push {lr}

doWhile\_r1\_lt\_r2:

mov r3,r3,asr #1

mov r2,r2,asr #1

cmp r1,r2

blt doWhile\_r1\_lt\_r2

pop {lr}

bx lr

.global addSub

addSub:

push {lr}

doWhile\_r3\_ge\_1:

add r0,r0,r3

sub r1,r1,r2

bl scaleRight

cmp r3,#1

bge doWhile\_r3\_ge\_1

pop {lr}

bx lr

.global scaleLeft

scaleLeft:

push {lr}

doWhile\_r1\_ge\_r2:

mov r3,r3,lsl #1

mov r2,r2,lsl #1

cmp r1,r2

bge doWhile\_r1\_ge\_r2

mov r3,r3,asr #1

mov r2,r2,asr #1

pop {lr}

bx lr

.global divMod

divMod:

push {lr}

mov r0,#0

mov r3,#1

cmp r1,r2

blt end

bl scaleLeft

bl addSub

end:

pop {lr}

bx lr

/\* r8=correct place, r9=correct num \*/

.global check

check:

push {lr}

mov r11, #0 /\* reset counter \*/

mov r12, #0

chkA:

cmp r1, r4

beq aChange

bne chkB

aChange:

mov r4, #-1

add r11, r11, #1

b chkB

chkB:

cmp r2, r5

beq bChange

bne chkC

bChange:

mov r5, #-1

add r11, r11, #1

b chkC

chkC:

cmp r3, r6

beq cChange

bne chkP

cChange:

mov r6, #-1

add r11, r11, #1

b chkP

chkP:

cmp r11, #3

beq chkEnd

cmp r4, #-1

bne chkAp

cmp r5, #-1

bne chkBp

cmp r6, #-1

bne chkCp

chkAp:

cmp r1, r5

moveq r5, #-1

addeq r12, r12, #1

beq chkBp

cmp r1, r6

moveq r6, #-1

addeq r12, r12, #1

b chkBp

chkBp:

cmp r2, r4

moveq r4, #-1

addeq r12, r12, #1

beq chkCp

cmp r2, r6

moveq r6, #-1

addeq r12, r12, #1

b chkCp

chkCp:

cmp r3, r4

moveq r4, #-1

addeq r12, r12, #1

beq chkEnd

cmp r3, r5

moveq r5, #-1

addeq r12, r12, #1

b chkEnd

chkEnd:

mov r1, r11

mov r2, r12

pop {lr}

bx lr