Mastermind

Project 2

CSC- 11 – 48598 Assembly

Tsz,Kwan

06 – December – 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…………………………………………..9-15

9. Code……………………………………………...16-32

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 four slots. 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), and it is only 3 digits. The player has ten times to guess to number. If they can’t guess the right number in ten 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 are input three digits form keyboard to array directly and store the input as a string which means the player doesn’t have to input three digits with space. There are only three functions which are random number generator, 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 remove the branches to check whether the main function works or not, which is easier to comment the whole thing. Then put the functions back one by one. In addition, I was going to make four-digit number; however, I didn’t know how to check the right digit in better way instead of hard code it, and using the index in assembly is confusing, so I make three digit number only.

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** |
| r0-r3 | temporary |
| r4 | address of original number array |
| r5 | address of input number array |
| r6 | address of copy number array |
| r7 | try counter |
| r8 | index counter |
| r10 | correct place counter |
| r11 | correct digit counter |
| input | temp storage input to array |
| original | original number array(answer) |
| copy | array copy from original |
| inarr | input number array |
| onef | float number 1 |
| tenf | float number 10 |
| y | asciz character y |
| n | asciz character n |
| ansYN | store play again answer |
| scan | int scan format |
| scanYN | string scan format |

5. Global external

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

6. Topic Covered (Checklist)

|  |  |  |  |
| --- | --- | --- | --- |
| description | code | source | line |
| scan formay (int) | scan: .asciz“%d” | proj2.s | 21 |
| scan format (string) | scanYN: .asciz“%s” | proj2.s | 22 |
| start 4 bytes boundary | .align 4 | proj2.s | 25 |
| emit 4 bytes | input: .word 0 | proj2.s | 26 |
| array | original: .skip 12 | proj2.s | 28 |
| float | onef: .float 1 | proj2.s | 34 |
| string | y: .asciz“y” | proj2.s | 38 |
| label | intro: | proj2.s | 51 |
| str lr for return (8 bytes) | push {r4, lr} | proj2.s | 49 |
| get lr for return (8 bytes) | pop {r4, lr} | proj2.s | 212 |
| exit stage right | bx lr | proj2.s | 213 |
| ldr address to register | ldr r0, addr\_intro1 | proj2.s | 52 |
| str valud to address | str r1, [r6, r8, lsl #2] | proj2.s | 81 |
| call printf | bl printf | proj2.s | 53 |
| call scanf | bl scanf | proj2.s | 98 |
| call strcmp | bl strcmp | proj2.s | 192 |
| put value into register | mov r7, #1 | proj2.s | 62 |
| add | add r8, r8, #1 | proj2.s | 83 |
| subtraction | sub r7, r8, r7 | proj2.s | 169 |
| Logical shift left | ldr r1, [r4, r8, lsl #2] | proj2.s | 81 |
| arithmetic shift right | mov r1, r0, asr #1 | randN.s | 12 |
| load the pointer into register | ldr r1, [r4] | proj2.s | 76 |
| compare and update flag | cmp r8, #3 | proj2.s | 84 |
| branch | b lose | proj2.s | 146 |
| branch if equal | beq start | proj2.s | 193 |
| branch if not equal | bne invalidYN | proj2.s | 198 |
| branch if less than | blt copy\_real | proj2.s | 85 |
| branch if greater or equal to | bge doWhile\_r3\_ge\_1 | divMod.s | 24 |
| call function | bl chk | proj2.s | 131 |
| translate a int to float | vcvt.f32.s32 s0, s14 | proj2.s | 172 |
| move value to s register | vmov s14, r7 | proj2.s | 171 |
| ldr a value s register | vldr s14, [r1] | proj2.s | 174 |
| float division | vdiv.f32 s31, s0, s14 | proj2.s | 175 |
| convert 32bit to 64 bit | vcvt.f64.f32 d5, s31 | proj2.s | 176 |
| put a double to registers | vmov r2, r3, d5 | proj2.s | 176 |
| prediction | moveq r2, #-1 | chk.s | 22 |
| update indexing modes | ldr r2, [r4, #4] | proj2,s | 77 |
| address | addr\_intro1: .word intro1 | proj2.s | 216 |
| external | .global scanf | proj2.s | 243 |
|  | .global printf | proj2.s | 244 |
|  | .global strcmp | proj2.s | 245 |
|  | .global time | proj2.s | 246 |
|  | .global srand | proj2.s | 247 |
|  | .global random | randN.s | 26 |

7. Pseudo Code

do{

Introduction

generate 3 digit number

store at original array

output original for debug

copy original to copy array

reset counter

do{

do{

game start

output counter

input 3 digit

if(anyone of 3 digits isn’t 0-9)

invalid

}while(invalid);

save input to inarr (input array)

reset place, num counter

for(int i=0;i<3;i++){

if(inarr[i]==copy[i]){

place++

copy[i]=-1

}

}

if(place==3)

display win message and accuracy( (11-try)/10 )

else

if(copy[0]!=-1)

compare inarr[0]to copy[1] and copy[2]

change copy[1] or copy[2] if matched

num++

if(copy[1]!=-1)

compare inarr[1] to copy[0] and copy[2]

change copy[0] or copy[2] if matched

num++

if(copy[2]!=-1)

compare inarr[2] to copy[0] and copy[1]

num++

counter ++

recover – copy original array to copy array

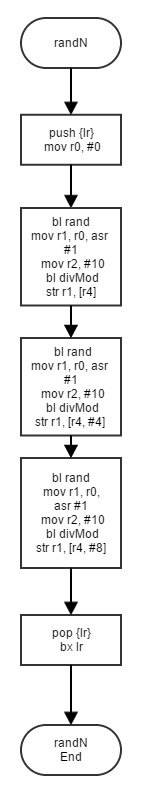
display place and num

}while(counter<10)

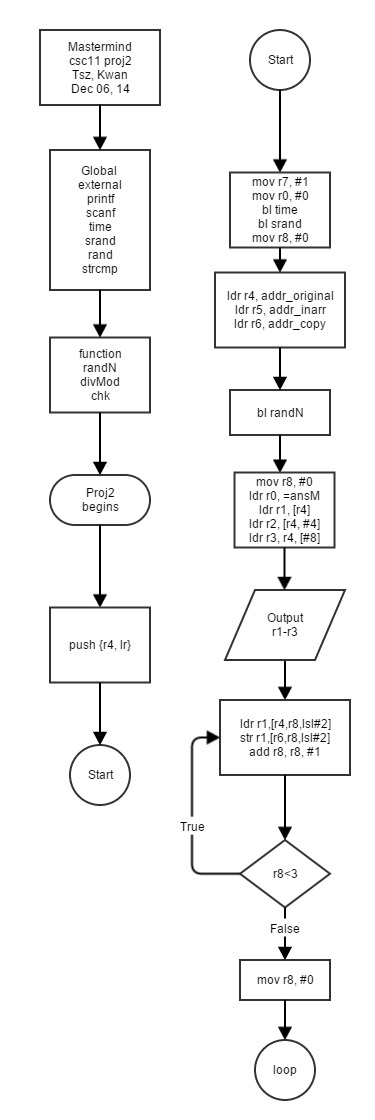
ask whether want to play again

}while(yes)

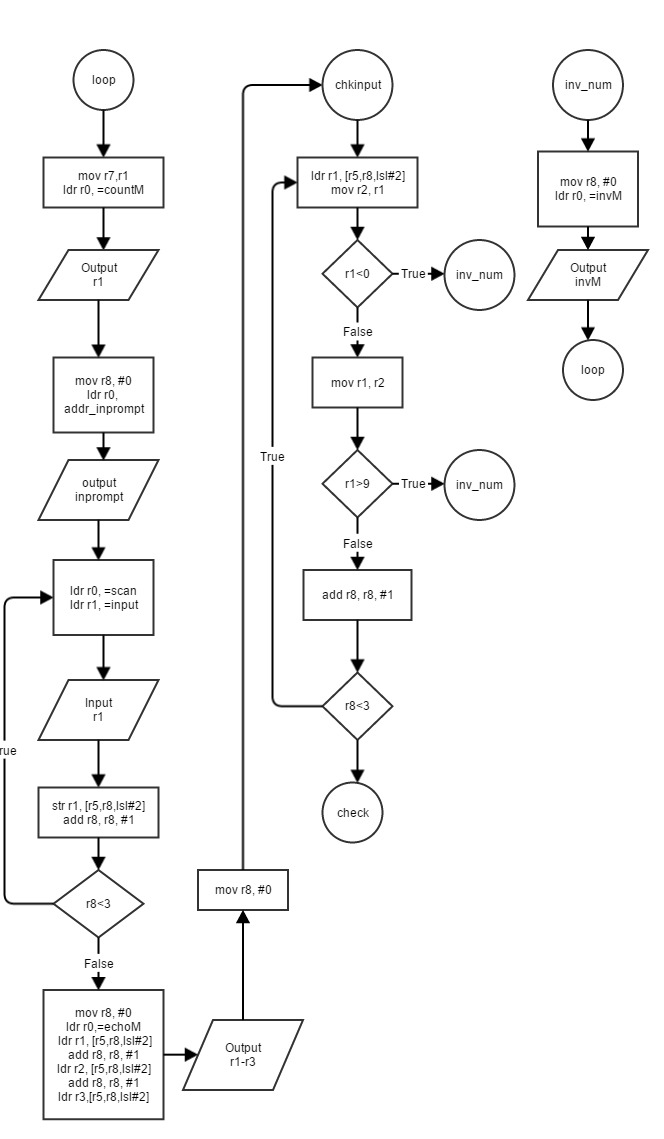
Program Ends

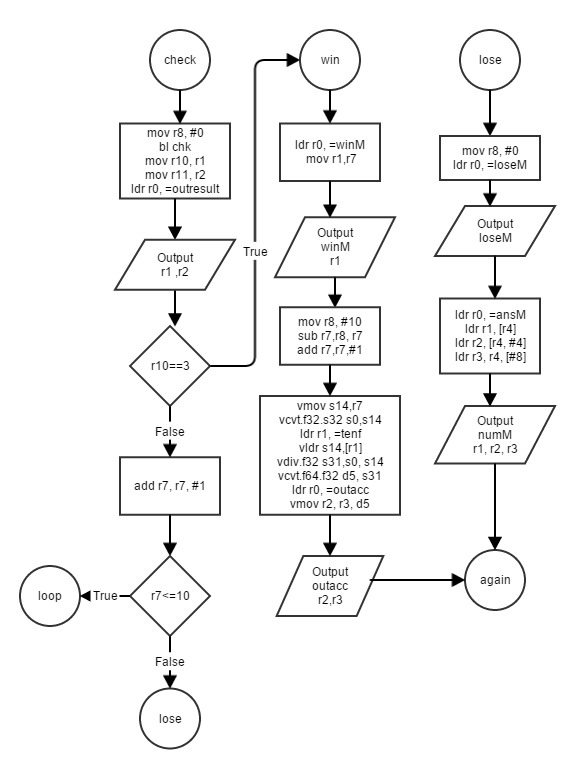
8. Flowchart

Main part 1 and randN

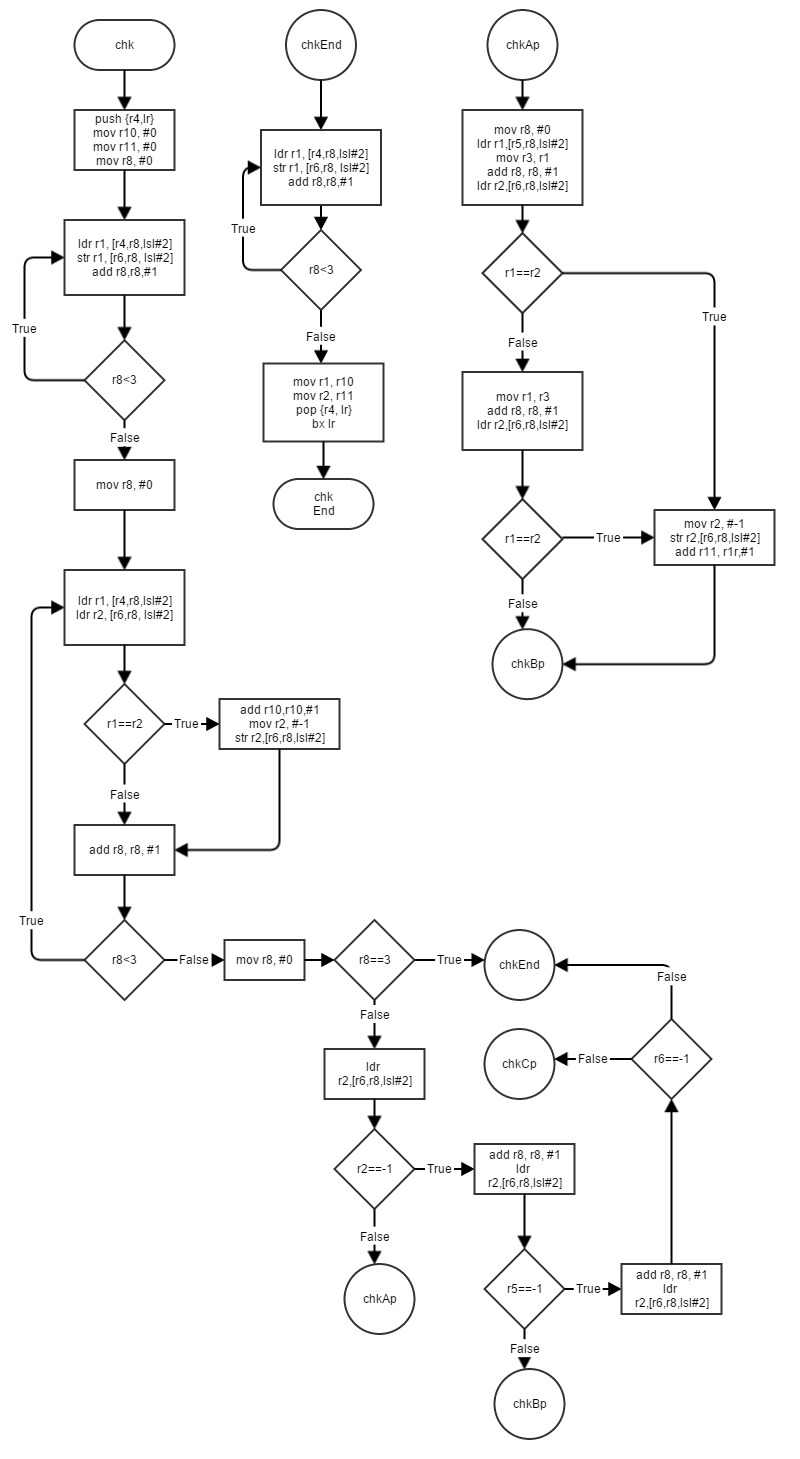


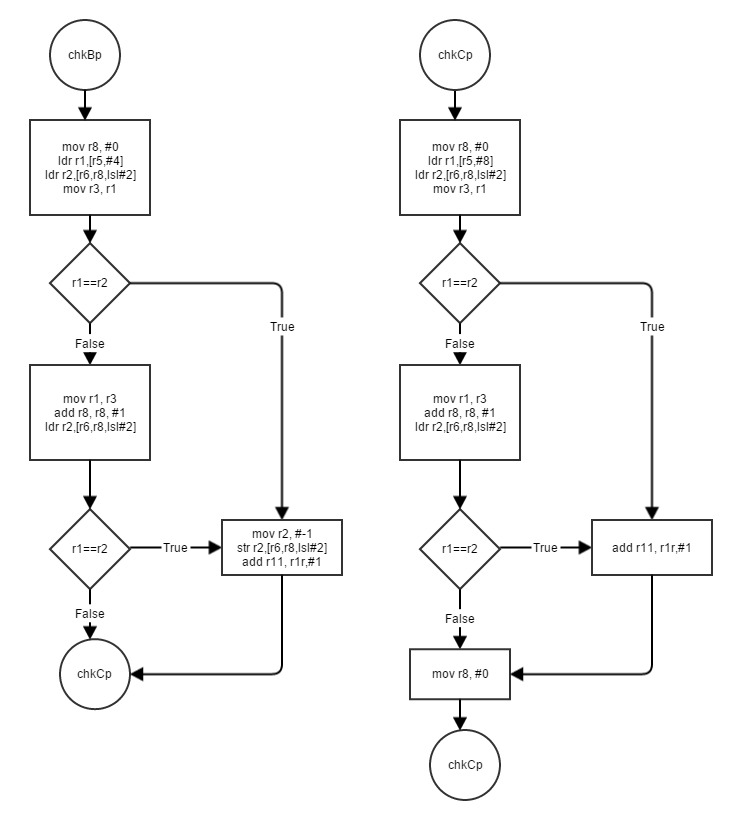
Main part2



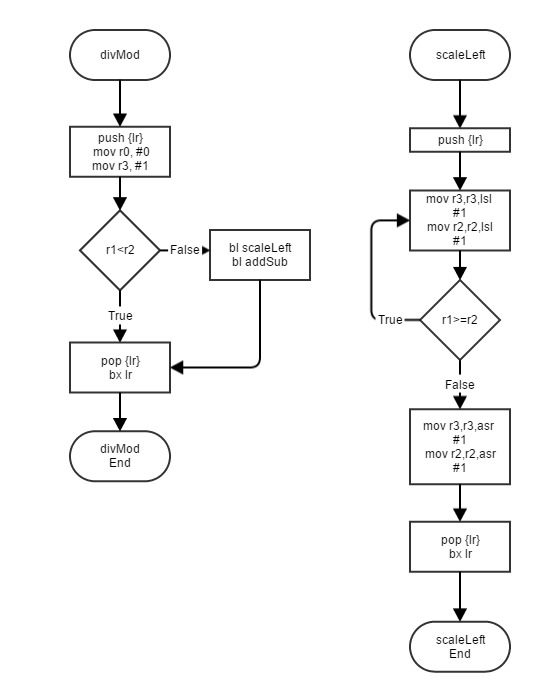
Check Result

chk function part 1

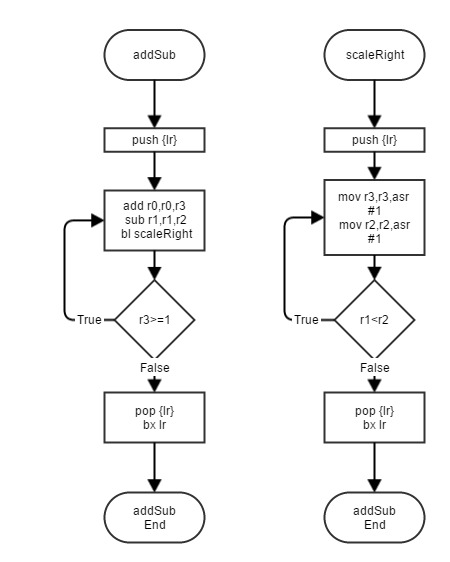


chk function part 2

divMod function and scaleLeft



addSub and scaleRight function



9. Code

/\* CSC 11 Project 2 \*/

/\* Mastermind \*/

.data

/\* message \*/

intro1: .asciz"Mastermind\n"

intro2: .asciz"3 digit number\n"

intro3: .asciz"10 chances\n"

intro4: .asciz"Game start\n"

outresult: .asciz"%d right place, %d right digit\n" @r1=x, r2=o

outacc: .asciz"Accuracy: %f\n" @r2, r3 =float

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

invM: .asciz"Invalid input\n" @invalid message

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

loseM: .asciz "You lose!\n" @lose message

echoM: .asciz "You input %d%d%d\n" @echo message

testM: .asciz "%d\n"

inprompt: .asciz "Please enter 3 digits with space\n" @input prompt

ansM: .asciz"The correct number is %d%d%d\n" @answer message

countM: .asciz"%d try\n" @try counter message

/\* scan format \*/

scan: .asciz"%d" @scan format 1 digit with space

scanYN: .asciz"%s" @string scan format

/\* variables \*/

.align 4

input: .word 0 @input digit 3 loop

.align 4

original: .skip 12 @correct answer, skip 12 bytes 3 int

.align 4

copy: .skip 12 @used to check

.align 4

inarr: .skip 12 @store input array, 3 int, 12byte

.align 4

onef: .float 1 @float counter

.align 4

tenf: .float 10 @float total 10

.align 4

y: .asciz"y" @string y

.align 4

n: .asciz"n" @string n

.align 4

ansYN: .word 0 @store y/n ans

/\* accuracy start from ten one try -1 \*/

/\* r4=ans array, r5=input array \*/

.text

.global main

main:

push {r4, lr} @8 bytes

intro:

ldr r0, addr\_intro1

bl printf

ldr r0, addr\_intro2

bl printf

ldr r0, addr\_intro3

bl printf

start:

ldr r0, addr\_intro4

bl printf

mov r7, #1 @set try counter

/\* r4=input, r5=original, r6=copy \*/

/\* generate 3 digit answer \*/

mov r0, #0

bl time @set clock

bl srand @set time seed

mov r8, #0 @reset index counter

ldr r4, addr\_original @r4=correct answer

ldr r5, addr\_inarr @r5=input array address

ldr r6, addr\_copy @r6=copy

rand\_number:

bl randN @call random function

mov r8, #0 @reset index counter

ldr r0, =ansM

ldr r1, [r4]

ldr r2, [r4, #4]

ldr r3, [r4, #8]

bl printf @out number for debug

copy\_real:

ldr r1, [r4, r8, lsl #2]

str r1, [r6, r8, lsl #2]

add r8, r8, #1

cmp r8, #3

blt copy\_real

mov r8, #0

loop: @try loop

mov r1, r7 @r1=try

ldr r0, =countM

bl printf

mov r8, #0 @reset index counter

ldr r0, addr\_inprompt

bl printf @output input prompt

inputloop:

ldr r0, addr\_scan @%d

ldr r1, addr\_input @one word width

bl scanf

ldr r1, addr\_input @r1=input address

ldr r1, [r1] @r1=input

str r1, [r5, r8, lsl #2] @r5=inarr[r11] address, inarr[r11]=r1

add r8, r8, #1 @index counter++

cmp r8, #3

blt inputloop @r11<3 go inputloop

outinput: @Your input is %d%d%d

mov r8, #0 @reset index counter

ldr r0, addr\_echoM

ldr r1, [r5, r8, lsl #2] @r1=inarr[0]

add r8, r8, #1 @index counter++

ldr r2, [r5, r8, lsl #2] @r2=inarr[1]

add r8, r8, #1 @index counter++

ldr r3, [r5, r8, lsl #2] @addr of input + 8bytes input[3]

bl printf

mov r8, #0 @reset index counter

chkinput: @input validation

ldr r1, [r5, r8, lsl #2] @inarr[r8]

mov r2, r1

cmp r1, #0

blt invalid\_digit

mov r1, r2

cmp r1, #9

bgt invalid\_digit

add r8, r8, #1 @index++

cmp r8, #3

blt chkinput @index<3 go chkinput

/\* check function \*/

/\* chk function r1=X, r2=O \*/

check:

mov r8, #0

bl chk @call chk function

mov r10, r1 @r10=X

mov r11, r2 @r11=O

checkresult: @ ?X ?O

ldr r0, addr\_outresult

bl printf @output result

cmp r10, #3

beq win @r10==3 -> win

addne r7, r7, #1 @incorrect try++

bne countchk @r10<3 ->chk try counter

countchk:

cmp r7, #10

ble loop @try counter<10 loop

b lose @try counter>=10 lose

win:

ldr r0, addr\_winM

mov r1, r7 @r7=counter

bl printf

b accresult

lose:

mov r8, #0

ldr r0, addr\_loseM

bl printf

ldr r0, =ansM

ldr r1, [r4, r8, lsl #2]

add r8, r8, #1

ldr r2, [r4, r8, lsl #2]

add r8, r8, #1

ldr r3, [r4, r8, lsl #2]

bl printf

b again

accresult:

mov r8, #10 @r8=10

sub r7, r8, r7 @r7=10-try

add r7, r7, #1

vmov s14, r7 @s14=10-try (int)

vcvt.f32.s32 s0, s14 @s0=10-try (float)

ldr r1, =tenf @address of float 10

vldr s14, [r1] @s14=10f

vdiv.f32 s31, s0, s14 @s31=(10-try)/10

vcvt.f64.f32 d5, s31 @convert to double d5

ldr r0, addr\_outacc

vmov r2, r3, d5 @store accuracy in r2, r3

bl printf @output accuracy

b again @ask yn question

again:

ldr r0, addr\_yn @again yn question

bl printf

ldr r0, addr\_scanYN

ldr r1, addr\_ansYN

bl scanf @take yn input

ldr r1, addr\_ansYN

ldr r0, addr\_y

bl strcmp

beq start

ldr r1, addr\_ansYN

ldr r0, addr\_n

bl strcmp

beq end

bne invalidYN

invalid\_digit: @invalid input digit

mov r8, #0

ldr r0, addr\_invM

bl printf @output invalid message

b loop @go back to loop

invalidYN:

ldr r0, addr\_invM

bl printf @output invalid message

b again @loop back to yn question

end:

pop {r4, lr}

bx lr @exit stage right

/\* address \*/

addr\_intro1: .word intro1

addr\_intro2: .word intro2

addr\_intro3: .word intro3

addr\_intro4: .word intro4

addr\_outresult: .word outresult

addr\_outacc: .word outacc

addr\_yn: .word yn

addr\_invM: .word invM

addr\_winM: .word winM

addr\_loseM: .word loseM

addr\_echoM: .word echoM

addr\_inprompt: .word inprompt

addr\_scan: .word scan

addr\_scanYN: .word scanYN

addr\_inarr: .word inarr

addr\_original: .word original

addr\_copy: .word copy

addr\_input: .word input

addr\_onef: .word onef

addr\_tenf: .word tenf

addr\_y: .word y

addr\_n: .word n

addr\_ansYN: .word ansYN

/\* external \*/

.global printf

.global scanf

.global strcmp

.global time

.global srand

/\* 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

/\* check method \*/

@r10=x, r11=o

@r5=input, r6=copy

@r8=index

.global chk

chk:

push {r4, lr}

mov r10, #0 @reset

mov r11, #0 @reset

mov r8, #0

re:

ldr r1, [r4, r8, lsl #2]

str r1, [r6, r8, lsl #2]

add r8, r8, #1

cmp r8, #3

blt re

mov r8, #0

chk\_X:

ldr r1, [r5, r8, lsl #2] @r1=in[r8]

ldr r2, [r6, r8, lsl #2] @r1=copy[r8]

cmp r1, r2

moveq r2, #-1 @r1==r2, r2=-1

streq r2, [r6, r8, lsl #2] @copy[r8]=-1

addeq r10, r10, #1 @place++

add r8, r8, #1 @index++

cmp r8, #3

blt chk\_X

mov r8, #0

chk\_O:

cmp r10, #3

beq chkEnd @X==3 -> in==copy

ldr r2, [r6, r8, lsl #2] @copy[0]

cmp r2, #-1

bne chkAp

add r8, r8, #1 @index++

ldr r2, [r6, r8, lsl #2] @copy[1]

cmp r2, #-1

bne chkBp

add r8, r8, #1 @index++

ldr r2, [r6, r8, lsl #2] @copy[2]

cmp r2, #-1

bne chkCp

beq chkEnd

chkAp:

mov r8, #0

ldr r1, [r5, r8, lsl #2] @in[0]

mov r3, r1 @r3=in[0]

add r8, r8, #1 @index++

ldr r2, [r6, r8, lsl #2] @copy[1]

cmp r1, r2

moveq r2, #-1 @r2=-1

streq r2, [r6, r8, lsl #2] @copy[1]=-1

addeq r11, r11, #1 @digit++

moveq r8, #0 @reset index

beq chkBp

mov r1, r3 @recover

add r8, r8, #1 @index++

ldr r2, [r6, r8, lsl #2] @copy[2]

cmp r1, r2

moveq r2, #-1 @r2=-1

streq r2, [r6, r8, lsl #2] @copy[2]=-1

addeq r11, r11, #1 @digit++

moveq r8, #0 @reset index

chkBp:

mov r8, #0

ldr r1, [r5, #4] @in[1]

ldr r2, [r6, r8, lsl #2] @copy[0]

mov r3, r1 @r3=in[1]

cmp r1, r2

moveq r2, #-1 @r1==r2, r2=-1

streq r2, [r6, r8, lsl #2] @copy[0]=-1

addeq r11, r11, #1 @digit++

beq chkCp

mov r1, r3 @recover

add r8, r8, #2 @coz ldr index 2

ldr r2, [r6, r8, lsl #2] @copy[2]

cmp r1, r2

moveq r2, #-1 @r2=-1

streq r2, [r6, r8, lsl #2] @copy[2]=-1

addeq r11, r11, #1 @digit++

chkCp:

mov r8, #0

ldr r1, [r5, #8] @in[2]

ldr r2, [r6, r8, lsl #2] @copy[0]

mov r3, r1 @r3=in[2]

cmp r1, r2

/\* moveq r2, #-1 @r1==r2, r2=-1

streq r2, [r6, r8, lsl #2] @copy[0]=-1\*/

addeq r11, r11, #1 @digit+

moveq r8, #0

beq chkEnd

mov r1, r3 @recover

add r8, r8, #1 @index++ index=1

ldr r2, [r6, r8, lsl #2] @copy[1]

cmp r1, r2

/\* moveq r2, #-1 @r2=-1

streq r2, [r6, r8, lsl #2] @copy[1]=-1\*/

addeq r11, r11, #1 @digit++

mov r8, #0

b chkEnd

chkEnd:

/\* recover \*/

ldr r1, [r4, r8, lsl #2] @r1=real[r8]

str r1, [r6, r8, lsl #2] @copy[r8]=real[r8]

add r8, r8, #1 @index++

cmp r8, #3

blt chkEnd @i<3 chkEnd

mov r1, r10 @r1=X

mov r2, r11 @r2=O

pop {r4, lr}

bx lr

/\* random number generator \*/

/\* store in r1 \*/

.global randN

randN:

push {lr}

bl rand @call random function from c

mov r1, r0, asr #1 @make sure it is positive

mov r2, #10 @set divisor

bl divMod @call divMod function

str r1, [r4] @r1=original[0]

bl rand

mov r1, r0, asr #1

mov r2, #10 @r1=original[1]

bl divMod

str r1, [r4, #4]

bl rand

mov r1, r0, asr #1

mov r2, #10

bl divMod

str r1, [r4, #8]

pop {lr}

bx lr

.global time

.global srand

.global random