Project 2

<Mastermind>

CSC-11-48598

Name: He, Zizhuo

Date: 12/7/2014

Introduction

This is the Mastermind game.

This game is an old code-breaking game. As an intellective game, the rules of the game are simple, but can test people's preciseness and patience. Moreover, it exercises player's ability of logical thinking.

As stated above, the rules of the game are simple.

The player is asked to guess three numbers from 0 to 6.

Those numbers are generated by the program.

When there is 1 number the user guessed has the correct value and in the right position, the counter RP (Right Position) will mark as 1. For example, assume the user guesses:

1 2 3

And the generated numbers are:

1 5 4

We can see that the number in the first spot matches, therefore, in this situation the game will display "RP: 1".

Meanwhile, if the player guesses a number that has the correct value of one or even more generated number but located in the wrong spot, the counter WP (Wrong Position) will mark as 1. For example, assume the user guesses:

1 2 3

And the generated numbers are:

4 1 6

We can see that the number in the first spot of player guessed matches the number in the second spot of generated. Therefore, in this situation the game will display "WP: 1".

As part of the challenge, they game will only allow the play to try a certain amount of tries. If the player do not guess all the three correct numbers when there is no more tries, the play lose.

Summary

Project size

Actual lines of code: about 213

Spacing: about 37

Comments: about 46

Variables: about 13

Because I do not write a write-up for my project1, and project2 is the extension of project1, here I am going to mention both of them.

It took about 2 days for project1 compare to 4 days of midterm because most of the concepts I had tested out during the coding of midterm. In the process of project1, I did write a couple test files to make sure each single piece is working. I do leave some problems that I was not able to solve in both midterm and project1, but I solve them in project 2. The most time consuming part for project 1 is actually figure out the logic of the AI. I spent about 6 hours drawing flowcharts. While I was writing the AI part of the project, I finish a complete flowchart before I start to code, which is different from other part that I develop and fix the logic during coding.

When I start project2, I simply use most of the code of project1. After several weeks study I understand the concept better, therefore I use another method to do the same things in Project2, which eventually makes the actual code shorter than in Porject1. However, I spend more time on writing testing files than on doing Project2.

Overall, in both Project 1 and 2, I utilize Predication inside the AI part. The motivation was because I was too lazy to use labels and branches. Meanwhile, for Project2, I apply some concepts may or may not been went over in class. I noticed that there is a built in division for float so instead of using Divmod I develop a way to utilize float. The process involves convert float to integer and integer to float. The total amount of time on Project 2 is about 15 hours. I still have some problems for Project2, which I feel cannot be solved by now with the limited knowledge.

The following sentences should be noticed:

Due to many reasons, including class issues such as other classes' final. I was not able to fix the AI part of this game. I do, however, spend a lot of time searching and thinking. The AI of this game is not completed. It is giving out wrong notations when there are multiple numbers repeating. It's such a pity that I eventually run out of time but still not able to figure it out. I understand that this is going to affect my grade, but at this point I have nothing can do.

Description

Concepts

This project includes many concepts we went over this semester.

Including:

1. Basic arithmetic, such as ‘mov’ ‘add’ ‘sub’.
   They are located everywhere. Loops, calculations in the AI.s file.
2. Load & Store
   Mostly in main.s file. They are used for printing string, saving values into variables.
3. Conditional branches, such as ‘cmp’, ‘bne’, ‘beq’, etc.
   A number of them are used in mian for loops, and in AI for logical process.
4. Control structures, such as “If, else”, “do, while”, etc.
   By utilizing other concepts I set up these structures successfully. Most of them are located in the beginning and the ending of main.
5. Loops, such as “i = 0 ; i < n-2 ; i++”, etc.
   To more detailed, loops can be find right before asking for input or right after outputting result.
6. Shifted operand, such as “lsl”, “lsr”, etc.
   Utilized to convert the generated random number form rand function, its purpose is to make sure the number does not overflow. It is located in init_Array.s file.
7. Arrays and structures
   In init_Array.s file. Been used to store three random numbers.
8. Functions & stack, such as “push {lr}” , “pop {lr}”, “bl function”, etc.
   Mainly used to store sp values enables the functions return to main correctly. Can be found at the very beginning and very end of each function.
9. Predication, such as “addeq”, “subne”, etc.
   Used in AI part. Avoiding some branches will increase the efficiency of processing, also save time of coding.
10. Floating point numbers

    - VFPv2 Registers, such as “s12”, “d0”, etc.
    - Arithmetic operations, such as “vadd.f32 “, etc.
    - Load and store, such as “vstr”, “vldr”.
    - Movements between registers, such as “vmov s2, r3”.
    - Conversions, such as “vcvt.f64.f32 d0, s0”.

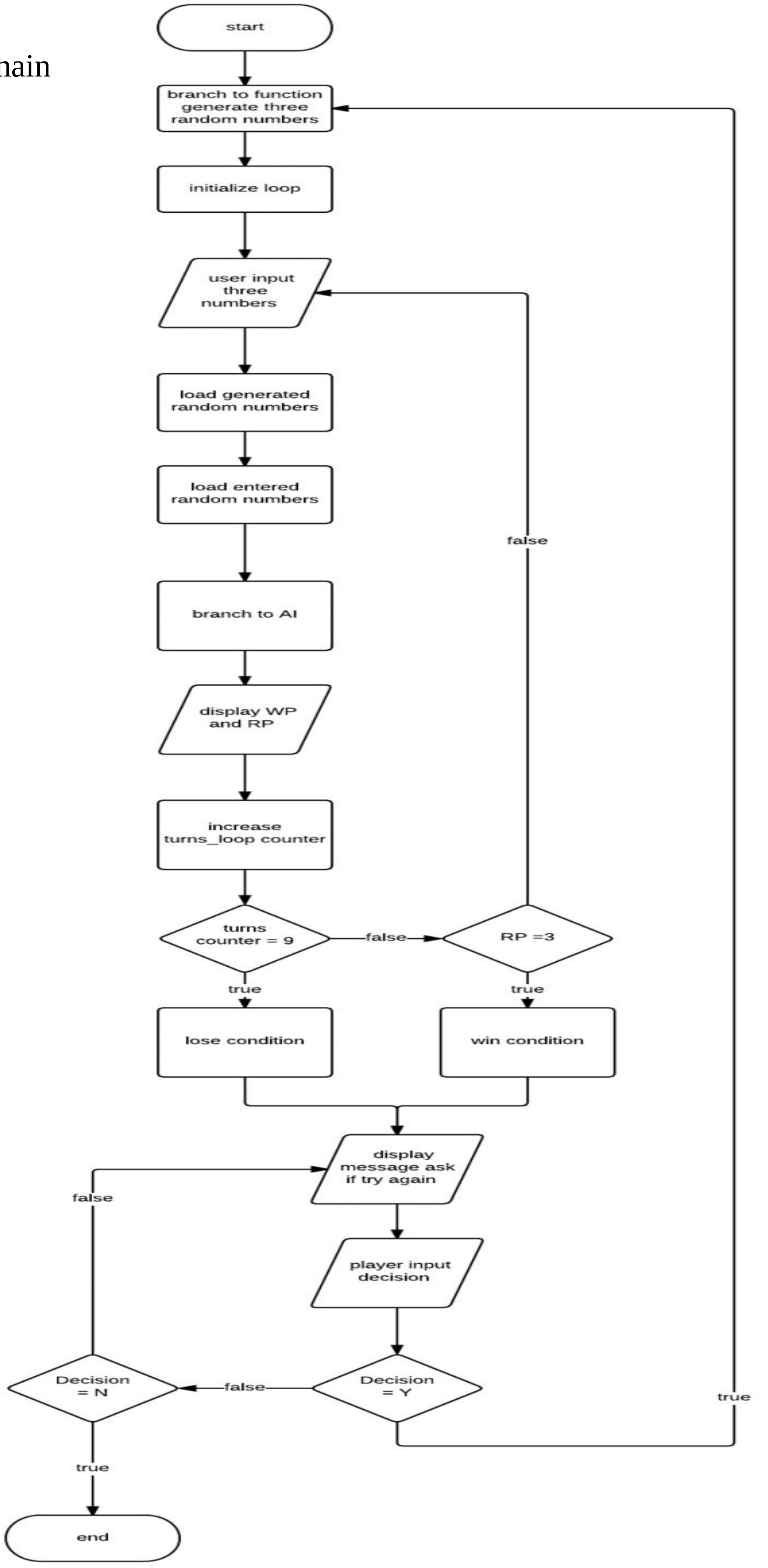
    Used for initializing array, generating 3 random numbers in place of DivMod.

11. Passing data to functions, such as passing an array.
    Can be seen in main.s file. All the value stored in the array are passed into main for further calculation.

Variables

Type	Variable name	Description	Location
	array	The array that contains three random number	.data of main
Asciz string	Message1	State the beginning of game	.data of main
Asciz string	Message2	Display the two counters of the game	.data of main
Asciz string	Message3	The winning information	.data of main
Asciz string	Message4	The nth turns and turns left	.data of main
Asciz string	Message5	The losing information	.data of main
Asciz string	Msg6	Ask if the player want to try again	.data of main
Asciz string	Msg7	Display when the user enter invalid value	.data of main
Asciz string	Msg8	Display the restarting of the game	.data of main
Asciz srting	Msg9	Display the exiting of the game	.data of main
integer	Spot1_read	Store the first entered number	.data of main
integer	Spot2_read	Store the second entered number	.data of main
integer	Spot3_read	Store the third entered number	.data of main

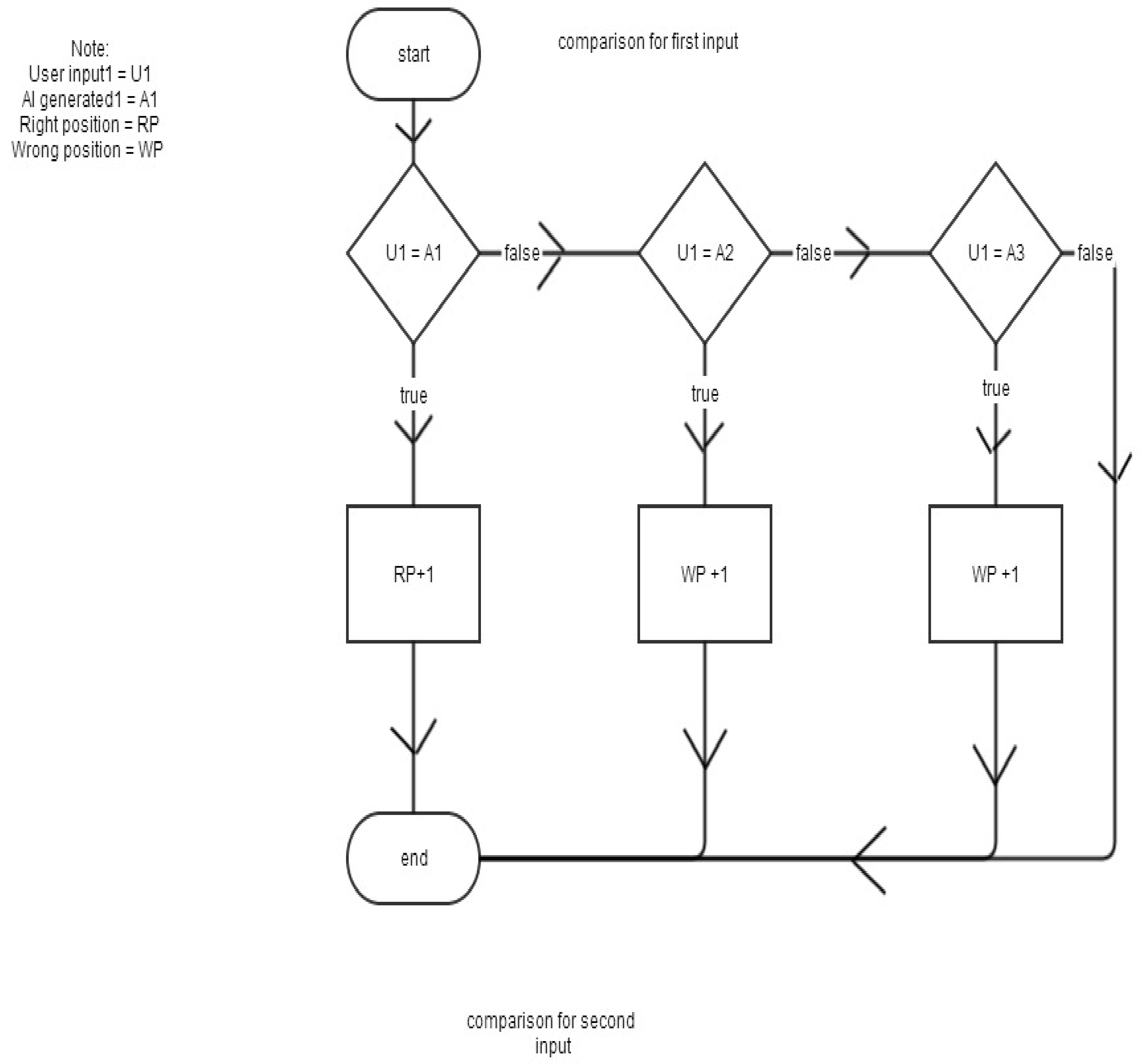
Flowchart main



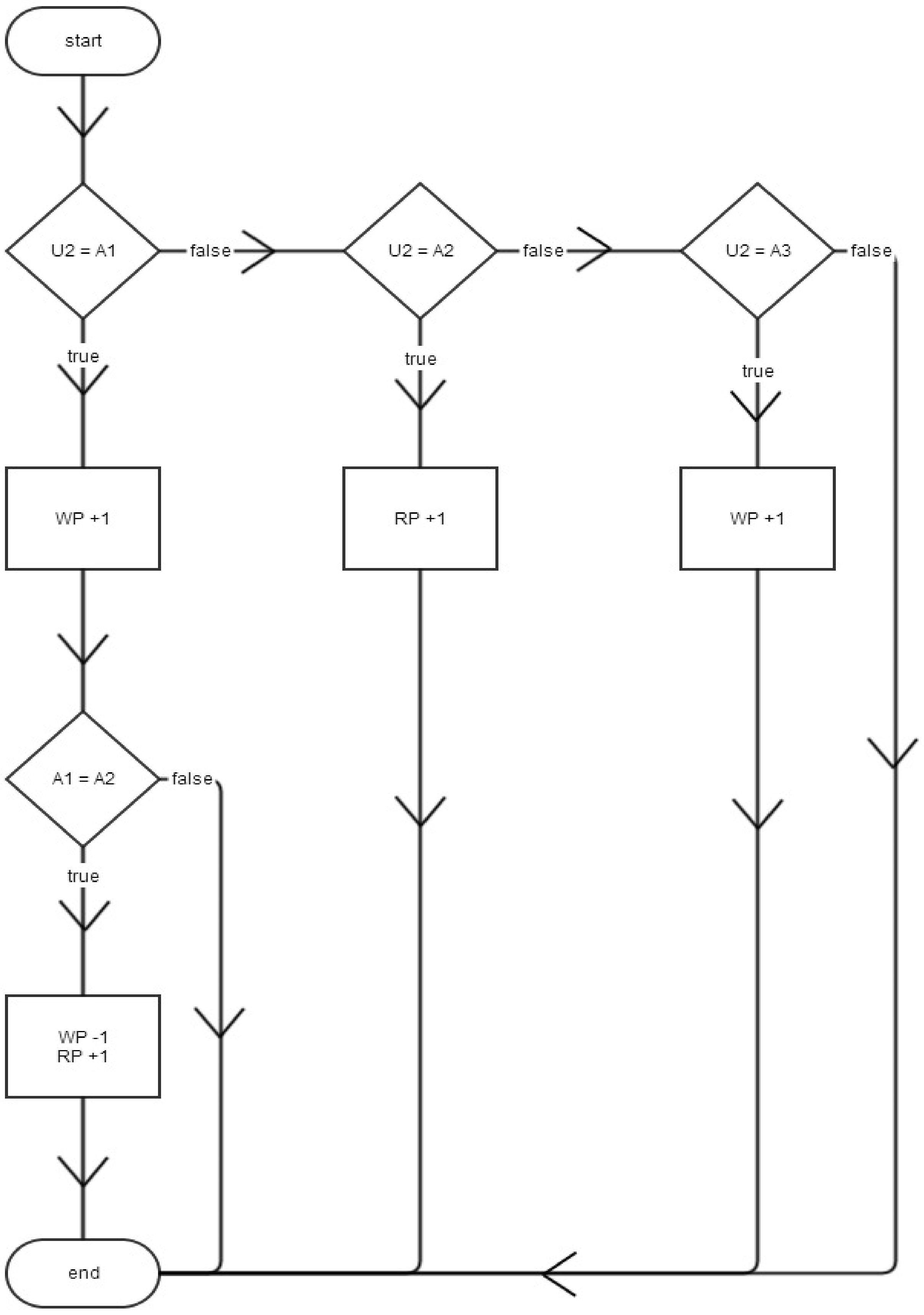
Pseudo Code of main

```
Initialize
Generate three random numbers and save to array
Int turns counter equals to 0
Do
    Do
        Load the random numbers and entered numbers
        Branch to AI, compare
        Display result
        Turns counter add one
    While (turns counter <9 && Right position !=3)
    If (Right position == 3)
        Display winning message
    Else
        Display losing message
    Do
        Ask users if try again, Y or N
        cin decision
    While (Decision!= Y && Decision!= N )
While (Decision == Y)
End
```

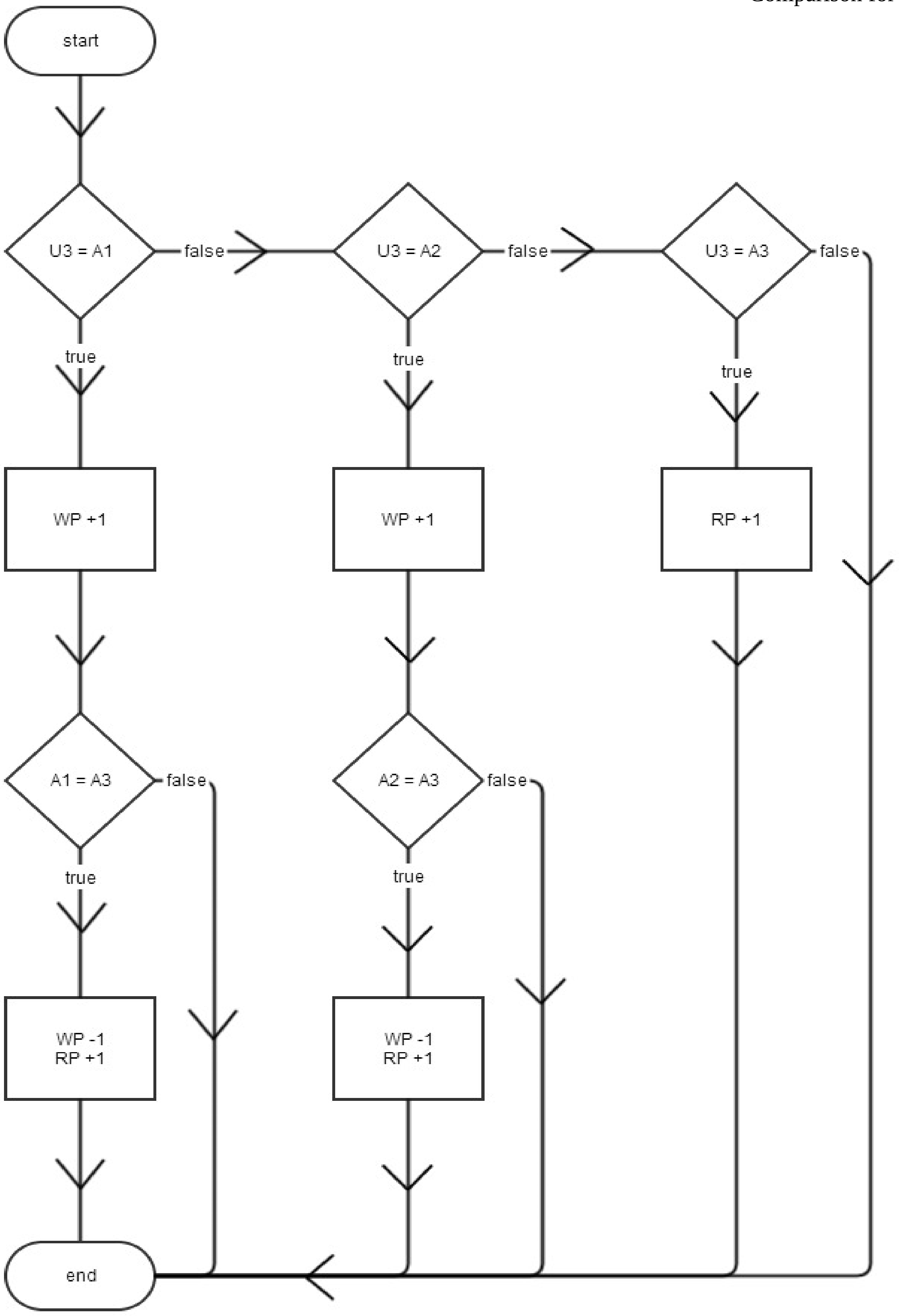
Flow chart AI

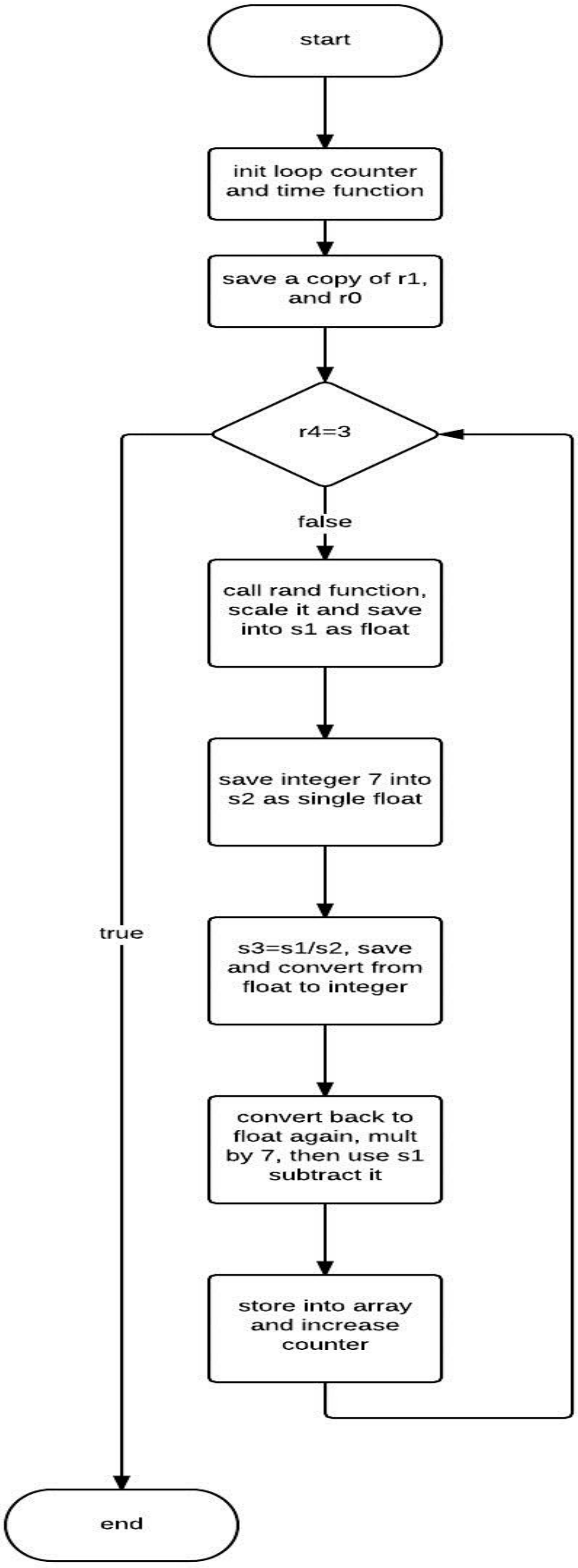


comparison for second input



Comparison for third part




start
init loop counter and time function
save a copy of r1, and r0
r4=3
false
call rand function, scale it and save into s1 as float
save integer 7 into s2 as single float
s3=s1/s2, save and convert from float to integer
convert back to float again, mult by 7, then use s1 subtract it
store into array and increase counter
true
end

Init_Array.s

Reference

Think in geek ARM assembler in Raspberry Pi

Raspberry Pi Assembly Language Raspbian Beginners (text book)

Github LehrMark_CSC11_48598

Program

project2_main_2.s

```
.data
/* saving 3 integers into an array, each integer takes 4 bytes,
 therefore align 3*4=12 bytes. Adding extra 4 bytes so 12+4=16 bytes */
.align 4
array: .skip 16
.align 4
message1: .asciz "========This is a Mastermind game========\n========Please enter 3 numbers between 0 and 6 for each,\n========Press 'enter' key after each number========\n"
.align 4
message2: .asciz "The result is RP: %d, WP: %d\n"
.align 4
scan_format: .asciz " %d %d %d"
/* prepare variables for user input */
.align 4
spot1_read: .word 0
.align 4
spot2_read: .word 0
.align 4
spot3_read: .word 0
.align 4
message3: .asciz "========You have guessed all the number correctly, congratulations!========\n"
.align 4
message4: .asciz "========This is the %d turns, you still have %d turns left========\n"
.align 4
message5: .asciz "========You lose!========\n"
.align 4
msg6: .asciz "Do you want to play the game again? Enter 'Y' for yes or 'N' for no:"
.align 4
scan_format2: .asciz " %c"
.align 4
option: .word 0
.align 4
msg7: .asciz "You entered an invalid value: %c\n"
.align 4
msg8: .asciz "You entered: %c. The game is about to restart!\n"
```

```
.align 4
msg9: .asciz "You entered: %c. Exiting the game, byebye!\n"
.text
.align 4
.global main
main:
        push {lr}
start:
        ldr r0, =message1
        bl printf
/* generate three random number through utilizing floating numbers */
generate_random_number:
        mov r0, #3 /* load the # of items */
        ldr r1, =array /* load the address of array */
        bl init_array /* call function to fill array */
/* set up loop counter here to limit the # of times user can try */
        mov r9, #1 /* the turns counter */
        mov r10, #8 /* turns left counter */
guess:
        ldr r0, =scan_format /* load the scan format */
        ldr r1, =spot1_read /* load r1 address of variable4 */
        ldr r2, =spot2_read /* load r2 address of variable5 */
        ldr r3, =spot3_read /* load r3 address of variable6 */
        bl scanf /*call scanf */
        /* compare*/
        mov r0, #0 /* init 'right position counter' zero */
        mov r5, #0 /* init 'wrong position counter' zero */
        ldr r6, =array /* load the address of array */
        ldr r1, [r6] /* load r1 the first generated random number */
        mov r7, #1 /* increase the increment by 1 */
        ldr r2, [r6, r7, lsl #2] /* load r2 the second generated random number */
        mov r7, #2 /* increase the increment by 2 */
        ldr r3, [r6, r7, lsl #2] /* load r3 the third generated random number */
        ldr r4, =spot1_read /* load the address of the first entered # */
        ldr r4, [r4] /* load r4 the first number user entered */
        bl cmp1 /* call AI to compare */
        ldr r4, =spot2_read /* load the address of the second entered # */
        ldr r4, [r4] /* load r4 the second  number user entered */
        bl cmp2 /* call AI to compare */
        ldr r4, =spot3_read /* load the address of the third entered # */
        ldr r4, [r4] /* load r4 the third  number user entered */
        bl cmp3 /* call AI to compare */
        mov r4, r0 /* save a copy of 'RP' to r4 */
        mov r1, r0 /* save a copy of 'RP' to r1 */
        mov r2, r5 /* save a copy of 'WP' to r5 */
        ldr r0, =message2
        bl printf
        ldr r0, =message4
        mov r1, r9 /* move r9 to r1, to show how many turns the user has tried */
         sub r10, r10, #1 /*sub r10, to show how many turns left */
        mov r2, r10
        bl printf
        add r9, r9, #1 /*add 1 to turns counter after each try */
        cmp r9, #9
        beq lose /* The user loses when the loop reaches the 9th turns */
        cmp r4, #3 /* set up loop for second, third,... tries */
        beq win /* if 'RP' reaches 3 means the user wins */
        b guess /* loop back to input when user did not make the correct guess */
lose:
        ldr r0, =message5
        bl printf /* display losing message */
        b end
win:
        ldr r0, =message3
        bl printf /* display winning message */
end:
        ldr r0, =msg6
        bl printf
        ldr r0, =scan_format2
        ldr r1, =option
        bl scanf
        ldr r1, =option
        ldr r1, [r1]
        cmp r1, #'Y'
        beq case_Y
        cmp r1, #'y'
        beq case_y
        cmp r1, #'N'
        beq case_N
        cmp r1, #'n'
        beq case_n
        ldr r0, =msg7
        bl printf
        b end
exit:
        pop {lr}
        bx lr
case_Y:
        ldr r0, =msg8
         ldr r1, =option
         ldr r1, [r1]
         bl printf
        b start
case_y:
        ldr r0, =msg8
         ldr r1, =option
         ldr r1, [r1]
         bl printf
        b start
case_N:
        ldr r0, =msg9
        ldr r1, =option
        ldr r1, [r1]
        bl printf
        b exit
case_n:
        ldr r0, =msg9
         ldr r1, =option
         ldr r1, [r1]
         bl printf
         b exit
.global printf
.global scanf
```

init_array.s

```
.data
.text
.global init_array
init_array:
        /* r0: # of items
           r1: address of array */
        push {r4, lr}
        mov r4, #0 /* the loop counter */
        mov r6, r1 /* save a copy of address of array */
        mov r5, r0 /* save a copy of # of items */
        mov r0,#0                     /* Set time(0) */
        bl time                       /* Call time */
        bl srand                      /* Call srand */
        b check_loop_array
loop_array:
        bl rand         /* Call rand */
        mov r2, r0, lsr #7     /* Left shit r0 7 times to make sure the float does not overflow */
/* The calculation part start from here */
/* concept:
        Since the idea behind utilizing DivMod is to get the remainder as the needed 1 bit decimal
random number, the same calculation can be represent by float.
        Let #1 represents the random number generated from rand function.
        Let #2.#3 represents the result of dividing #1 by 7.
        #2 will be the integer part of the reuslt, while #3 is the fractional part.
        The remainder is from subtracting the original number #1 by the result of multing the integer
part #2 with 7.
        Let #4 be the remainder, #4 equals #1-#2*7; #4=#1-#2*7.
        In the following calculation:
        1. Convert the random number from rand function to a single-precision float (s_float).
        2. Use the div command to divide by float 7.
        3. Convert the s_float to s_integer, which should only keep the integer part of it.
        4. Then convert the ineger back to s_float again, multiply by 7.
        5. Finally, subtract the original number by it to get the remainder. */
        vmov s1, r2    /* Bit copy from integer register r2 to s1 */
        vcvt.f32.s32 s1, s1 /* Converts s1 signed integer value to a single-precision value and
stores it in s1 */
        mov r2, #7      /* The goal is to generate 7 intege so the denominator is 7 */
        vmov s2, r2    /* Bit copy from integer register r2 to s2 */
        vcvt.f32.s32 s2, s2   /* Converts s2 signed integer value to a single-precision value and 
stores it in s2 */
        vdiv.f32 s3, s1, s2 /* s3 is the result of dividing by 7; s3=s1/s2 */
        vcvt.s32.f32 s3, s3 /* Converts s3 single-precision value to a signed integer and stores it 
in s3, in other words only keep the integer part of the float */
        vmov r3, s3 /*Bit copy from s3 to r3 */
        vmov s4, r3 /* Bit copy from integer register r3 to s4 */
        vcvt.f32.s32 s4, s4 /* Converts s4 signed integer value to a single-precision value and 
stores it in s4 */
        vmul.f32 s5, s4, s2 /* multiply the integer part of the float of division with float 7' 
s5=s4*s2 */
        vsub.f32 s6, s1, s5 /* subtract the integer part of the float of division will leave us the 
remainder; s6=s1*s5 */
        vcvt.s32.f32 s6, s6 /* Convert s6 single-precision value to a signed integer and stores it in 
s6 */
        vmov r1, s6 /* Bit copy from s6 to r1 */
        str r1, [r6, r4, lsl #2] /* store the integer into array */
        add r4, r4, #1 /* increase the loop counter by 1 */
check_loop_array:
        cmp r4, r5 /* compare with # of items */
        bne loop_array /* if not equal branch back */
        pop {r4, lr}
        bx lr
```

AI.s

```
/* all of
the
comparision
functions
are here */
/* Note that user entered numbers are labled as 'U1' 'U2' 'U3' */
/* Also the generated numbers are labled as 'A1' 'A2' 'A3' */
.text
.global cmp1
cmp1:
        push {lr}
        cmp r4, r1 /* compare if U1 is equal to A1 */
        addeq r0, r0, #1 /* if U1 = A1, RP +1 */
        beq end1 /* then there is nothing more to do */
        cmp r4, r2 /* compare if U1 is equal to A2 */
        addeq r5, r5, #1 /* if U1 = A2, WP + 1 */
        beq end1 /* then there is nothing more to do */
        cmp r4, r3 /* compare if U1 is equal to A3 */
        addeq r5, r5, #1 /* if U1 = A3, WP + 1 */
end1:
        pop {lr}
        bx lr
.global cmp2
cmp2:
        push {lr}
        cmp r4, r1 /* compare if U2 = A1 */
        beq U2_equal_A1 /* if U2 = A1, branch to lable */
        cmp r4, r2 /* compare if U2 = A2 */
        addeq r0, r0, #1 /* if equal RP + 1 */
        beq end2 /* then there is nothing more to do */
        cmp r4, r3 /* compare if U2 = A3 */
        addeq r5, r5, #1 /* if equal WP + 1 */
end2:
        pop {lr}
        bx lr
U2_equal_A1:
        add r5, r5, #1 /* WP + 1 when U2 = A1 */
        cmp r1, r2 /* compare if A1 = A2 */
        addeq r0, r0, #1 /* if equal, that means U1 = A1 = A2, RP +1 */
        subeq r5, r5, #1 /* so, there is no need to add 1 to WP */
        b end2
.global cmp3
cmp3:
        push {lr}
        cmp r4, r1 /* compare if U3 = A1 */
        beq U3_equal_A1 /* if equal, branch to lable */
        cmp r4, r2 /* compare if U3 = A2 */
        beq U3_equal_A2 /* if equal, branch to lable */
        cmp r4, r3 /* compare if U3 = A3 */
        addeq r0, r0, #1 /* if equal, RP + 1 */
end3:
        pop {lr}
        bx lr
U3_equal_A2:
        add r5, r5, #1 /* when U3 = A1, WP + 1 */
        cmp r2, r3 /* compare if A1 = A3 */
        addeq r0, r0, #1 /* if equal, U3 = A1 = A3, RP +1 */
        subeq r5, r5, #1 /* so, there is no need to add 1 to WP */
        b end3
U3_equal_A1:
        add r5, r5, #1 /* when U3 = A2. WP + 1 */
        cmp r2, r3 /* compare if A2 = A3 */
        addeq r0, r0, #1 /* if equal, U1 = A2 =A3, RP + 1 */
        subeq r5, r5, #1 /* so, there is no need to add 1 to WP */
        b end3
```