**Project 1**

Title

**Simple Blackjack Game**

Course

**CSC 11**

Section

**48598**

Due Date

**November 3, 2014**

Author

**Victor Medel**

**Introduction**

Title: Blackjack

This is a simple program that allows any player to quickly play a game of Blackjack. The object of the game is to beat the house by receiving a score of 21 or by getting a higher score than the house without going over 21 with any additional cards. The game begins by dealing two cards to the player; after displaying your score and if your score is less than 21 you will have the option to take another card to add to your total score or hold with your existing score. If you hold or go over 21 after choosing the additional card the program will automatically display the house’s hand and then determine the outcome. Multiple decks of cards are used with the following values:

Cards 2 through 10 = face value points

Jacks = 10 points

Queens = 10 Points

Kings = 10 Points

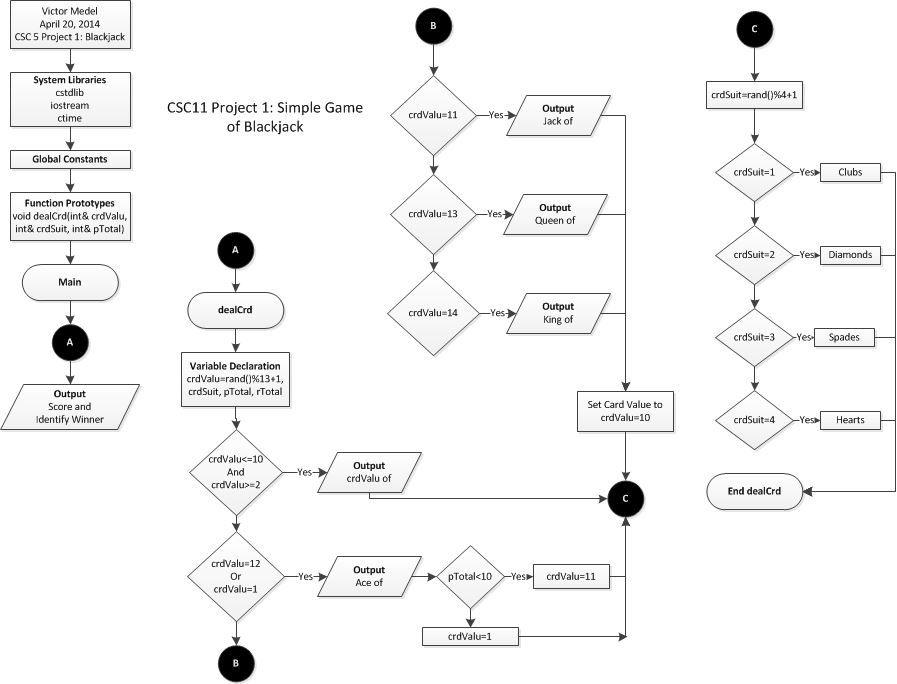
Aces = 1 or 11 are determined by the random number generated

**Summary**

This assembly assignment has been one of the toughest thus far; fortunately having taking C++ last semester I was able to utilize project ideas, notes, and most of the C++ code from that semester. As references I utilized the class textbook, (*Raspberry Pi Assembly Language: Raspbian Beginners*) all available class GitHub repositories and their contents, as well as some of the notes that were discussed in class from the *Think In Geek* website.

I utilized many of the mnemonics covered in class and in the class textbook to develop my assembly program. I also used many of the ideas presented in class such as the random number generation procedure and function utilization. I believe that this program can be improved or refactored to be easier to read and increase performance. I also believe that I required more practice coding assembly language to further improve the outcome of this program, for that reason I feel that my code long and cumbersome.

My plan to improve this program is to streamline the code and add more prompts that would allow the user to easily follow the game just as the C++ version of this application. Overall the program as it is now took about five days to create and prepare for delivery. I think for the time allotted this program fully displays all concepts covered in class.

****

**Flowchart**

**Program Code**

/\*

\* Author: Victor Medel

\* Created on November 1, 2014

\* CSC11 Project 1 - Simple Game of Black Jack

\*

\*/

.data

message0: .asciz "You have been delt the following card(s): "

message1: .asciz "%d of "

message2: .asciz "%d | "

message3: .asciz "%d of "

message4: .asciz "%d"

message5: .asciz "\nYour current score is %d\n"

message6: .asciz "Would you like another card? \n(Enter 0 for yes, anything else for no.): "

message50: .asciz "The house has been delt the following cards: "

message7: .asciz "%d of "

message8: .asciz "%d | "

message9: .asciz "%d of "

message10: .asciz "%d | "

message11: .asciz "%d of "

message12: .asciz "%d\n"

message13: .asciz "\nThe House's score is %d\n"

message14: .asciz "You Win!\n"

message15: .asciz "You Lose\n"

message16: .asciz "Clubs | "

message17: .asciz "Diamonds | "

message18: .asciz "Hearts | "

message19: .asciz "Spades | "

message20: .asciz "Ace of "

message21: .asciz "Jack of "

message22: .asciz "Queen of "

message23: .asciz "King of "

format: .asciz "%d"

.text

scaleRight:

push {lr} @ Push lr onto the stack

doWhile\_r1\_lt\_r2: @ Shift right until just under the remainder

mov r3,r3,ASR #1; @ Division counter

mov r2,r2,ASR #1 @ Mod/Remainder subtraction

cmp r1,r2

blt doWhile\_r1\_lt\_r2

pop {lr} @ Pop lr from the stack

bx lr

addSub:

push {lr} @ Push lr onto the stack

doWhile\_r3\_ge\_1:

add r0,r0,r3

sub r1,r1,r2

bl scaleRight

cmp r3,#1

bge doWhile\_r3\_ge\_1

pop {lr} @ Pop lr from the stack

bx lr

scaleLeft:

push {lr} @ Push lr onto the stack

doWhile\_r1\_ge\_r2: @ Scale left till overshoot with remainder

mov r3,r3,LSL #1 @ scale factor

mov r2,r2,LSL #1 @ subtraction factor

cmp r1,r2

bge doWhile\_r1\_ge\_r2 @ End loop at overshoot

mov r3,r3,ASR #1 @ Scale factor back

mov r2,r2,ASR #1 @ Scale subtraction factor back

pop {lr} @ Pop lr from the stack

bx lr

division:

push {lr} @ Push lr onto the stack

@ Determine the quotient and remainder

mov r0,#0

mov r3,#1

cmp r1,r2

blt end

bl scaleLeft

bl addSub

end:

pop {lr} @ Pop lr from the stack

bx lr

@Suit Selection

suitselect:

cmp r1, #1

ble clubs

bal select

select:

cmp r1, #2

ble diamonds

bal select1

select1:

cmp r1, #3

ble hearts

bal select2

select2:

cmp r1, #4

ble spades

bal exit

clubs:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message16 @ Set message16 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

diamonds:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message17 @ Set message17 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

hearts:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message18 @ Set message18 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

spades:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message19 @ Set message19 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

@End of Suit Selection

@Ace, Jack, Queen, and King Selection

faceselect:

cmp r1, #1

ble ace

bal facesel

facesel:

cmp r1, #14

bge king

bal facesel1

facesel1:

cmp r1, #13

bge queen

bal facesel2

facesel2:

cmp r1, #12

bge jack

bal facesel3

facesel3:

cmp r1, #11

bge ace

bal regular

ace:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message20 @ Set message20 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

jack:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message21 @ Set message21 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

queen:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message22 @ Set message22 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

king:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message23 @ Set message23 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

regular:

push {lr} @ Push lr onto the stack

ldr r0, address\_of\_message1 @ Set message19 as the first parameter of printf

bl printf @ Call printf

pop {lr} @ Pop lr from the stack

bx lr

@ End Ace, Jack, Queen , and King Selection

.global main

main:

push {lr} @ Push lr onto the top of the stack

mov r0,#0 @ Set time(0)

bl time @ Call time

bl srand @ Call srand

mov r4,#0 @ Setup loop counter

.global face1

face1: @ Create a random number

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#14 @ Move 14 to r2

@ We want rand()%14+1 so cal division function with rand()%14

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 14

mov r5, r1

@ldr r0, address\_of\_message1 @ Set message1 as the first parameter of printf

@bl printf @ Call printf

bl faceselect

bl suit1

.global suit1

suit1:

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#4 @ Move 4 to r2

@ We want rand()%4+1 so call division function with rand()%4

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 4

mov r10, r1

@ldr r0, address\_of\_message2 @ Set message2 as the first parameter of printf

@bl printf @ Call printf

bl suitselect

bl face2

.global face2

face2: @ Create a random number

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#14 @ Move 14 to r2

@ We want rand()%14+1 so cal division function with rand()%14

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 14

mov r6, r1

@ldr r0, address\_of\_message3 @ Set message3 as the first parameter of printf

@bl printf @ Call printf

bl faceselect

bl suit2

.global suit2

suit2:

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#4 @ Move 4 to r2

@ We want rand()%4+1 so cal division function with rand()%4

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 4

mov r10, r1

@ldr r0, address\_of\_message4 @ Set message4 as the first parameter of printf

@bl printf @ Call printf

bl suitselect

cmp r5, #11

movgt r5, #10

cmp r6, #11

movgt r6, #10

add r7, r6, r5 @ Add players score and print it out

mov r1, r7

ldr r0, address\_of\_message5 @ Set message5 as the first parameter of printf

bl printf

cmp r7, #21 @ Compare players score with 21

blt ask @ Ask player if the would like another card

bge houseface1 @ Otherwise display house's hand

.global ask

ask:

str lr, [sp,#-4]! @ Push lr onto the top of the stack

sub sp, sp, #4 @ Make room for one 4 byte integer in the stack

@ In these 4 bytes we will keep the number

@ entered by the user

ldr r0, address\_of\_message6 @ r0 <- message6

bl printf @ call to printf

ldr r0, address\_of\_format @ r0 <- scan\_pattern

mov r1, sp @ Set variable of the stack as

bl scanf @ call to scanf

add r1, sp, #4 @ Place sp+4 -> r1

ldr r1, [sp] @ Load the integer b read by scanf into r2

bl compare

add sp, sp, #4 @ Discard the integer read by scanf

ldr lr, [sp], #+4 @ Pop the top of the stack and put it in lr

bx lr @ return from main using lr

.global compare

compare:

cmp r1, #0

beq face3

bne houseface1

.global face3

face3: @ Create a random number

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#14 @ Move 14 to r2

@ We want rand()%14+1 so cal division function with rand()%14

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 14

mov r8, r1

@ldr r0, address\_of\_message1 @ Set message3 as the first parameter of printf

@bl printf @ Call printf

bl faceselect

bl suit3

.global suit3

suit3:

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#4 @ Move 4 to r2

@ We want rand()%4+1 so cal division function with rand()%4

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 4

mov r10, r1

@ldr r0, address\_of\_message2 @ Set message4 as the first parameter of printf

@bl printf @ Call printf

bl suitselect

bal addhand

addhand:

cmp r7, #11

movgt r7, #10

add r7, r7, r8 @ Add players score and print it out

mov r1, r7

ldr r0, address\_of\_message5 @ Set message5 as the first parameter of printf

bl printf

bal houseface1

.global houseface1

houseface1: @ Create a random number

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#14 @ Move 14 to r2

@ We want rand()%14+1 so cal division function with rand()%14

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 14

mov r5, r1

@ldr r0, address\_of\_message7 @ Set message1 as the first parameter of printf

@bl printf @ Call printf

bl faceselec

bl housesuit1

.global housesuit1

housesuit1:

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#4 @ Move 4 to r2

@ We want rand()%4+1 so call division function with rand()%4

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 4

mov r10, r1

@ldr r0, address\_of\_message8 @ Set message2 as the first parameter of printf

@bl printf @ Call printf

bl suitselect

bl houseface2

.global houseface2

houseface2: @ Create a random number

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#14 @ Move 14 to r2

@ We want rand()%14+1 so cal division function with rand()%14

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 14

mov r6, r1

@ldr r0, address\_of\_message9 @ Set message3 as the first parameter of printf

@bl printf @ Call printf

bl faceselect

bl housesuit2

.global housesuit2

housesuit2:

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#4 @ Move 4 to r2

@ We want rand()%4+1 so cal division function with rand()%4

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 4

mov r10, r1

@ldr r0, address\_of\_message10 @ Set message4 as the first parameter of printf

@bl printf @ Call printf

bl suitselect

bal houseface3

.global houseface3

houseface3: @ Create a random number

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#14 @ Move 14 to r2

@ We want rand()%14+1 so cal division function with rand()%14

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 14

mov r7, r1

@ldr r0, address\_of\_message11 @ Set message3 as the first parameter of printf

@bl printf @ Call printf

bl faceselect

bl housesuit3

.global housesuit3

housesuit3:

bl rand @ Call rand

mov r1,r0,asr #1 @ In case random return is negative

mov r2,#4 @ Move 4 to r2

@ We want rand()%4+1 so cal division function with rand()%4

bl division @ Call division function to get remainder

add r1,#1 @ Remainder in r1 so add 1 giving between 1 and 4

mov r10, r1

@ldr r0, address\_of\_message12 @ Set message4 as the first parameter of printf

@bl printf @ Call printf

bl suitselect

bal addhand2

addhand2:

cmp r5, #11

movgt r5, #10

cmp r6, #11

movgt r6, #10

cmp r7, #11

movgt r7, #10

add r9, r5, r6 @ Add players score and print it out

add r9, r9, r7

mov r1, r9

ldr r0, address\_of\_message13 @ Set message5 as the first parameter of printf

bl printf

bal scorecomp0

scorecomp0: @ The following compare numonics are used to compare score and determine winner

cmp r7, #21

ble housescore

bgt youlose

housescore:

cmp r9, #21

ble scorecomp

bgt youwin

scorecomp:

cmp r7, r9

bgt youwin

blt youlose

youwin:

ldr r0, address\_of\_message14

bl printf

bal exit

youlose:

ldr r0, address\_of\_message15

bl printf

bal exit

@add r4,#1

@cmp r4,#1 @ How many hands do you want the dealer to deal?

@blt face1

exit:

pop {lr} @ Pop the top of the stack and put it in lr

bx lr @ Leave main

address\_of\_message0: .word message0

address\_of\_message1: .word message1

address\_of\_message2: .word message2

address\_of\_message3: .word message3

address\_of\_message4: .word message4

address\_of\_message5: .word message5

address\_of\_message6: .word message6

address\_of\_message7: .word message7

address\_of\_message8: .word message8

address\_of\_message9: .word message9

address\_of\_message10: .word message10

address\_of\_message11: .word message11

address\_of\_message12: .word message12

address\_of\_message13: .word message13

address\_of\_message14: .word message14

address\_of\_message15: .word message15

address\_of\_message16: .word message16

address\_of\_message17: .word message17

address\_of\_message18: .word message18

address\_of\_message19: .word message19

address\_of\_message20: .word message20

address\_of\_message21: .word message21

address\_of\_message22: .word message22

address\_of\_message23: .word message23

address\_of\_message50: .word message50

address\_of\_format: .word format

@ External Functions

.global printf

.global time

.global srand

.global rand