ARM BlackJack

Project 2



# 

# Amul Bham

## CSC 11 :48982

## Date: 12/16/2015

# INTRODUCTION

BlackJack is a simple card game in which the player competes against the dealer to get the higher hand but not over 21. Whoever goes over 21 loses. The player receives their first two cards and the dealer receives one. Based on the player's card total, they must decide to “hit” or “stay”, trying not to go over 21 but still attempting to have a higher hand than the dealer. The dealer is set up in a way where they never “hit” if they are at 17 or higher, therefore it is the player’s decision to hit or stay depending on if they think the dealer will bust or have a low card total.

For example: If the player has a hand of 14, they risk the chance of going over 21 on their next card draw. So they might decide to stay, despite having a low card total, in hopes that the dealer will ultimately bust. If they dealer has a 9 for their first card, they will keep drawing cards till they are at 17 or higher.

Basically the dealer is always playing risky, and it is up to the player to outsmart the dealer based on odds and knowing when to be aggressive or passive. Also note that if the player goes over 21, they lose no matter what because they draw their cards first. If the hand ends in a tie, no one wins. If a user has a blackjack (21), the dealer still has a chance of drawing a tie if they also have a 21.

This game provides a very similar experience to playing BlackJack in a casino and is intended to be a stimulating game based on odds and risk. The reason I choose BlackJack is because it is a relatively simple game to play, however requires a ton of logic to guide the program to the proper outcome. The game is challenging to program and I find joy translating the game to different programming languages. I must say, assembly by far proved to be the trickiest and provided me quite the challenge to get the logic and results correct. In addition, due to the complexity of the logic, the game ensures the use of all the programming concepts in a purposeful way. I often found myself stuck on a particular piece only to be relieved when I realized that we already learned how to fix the error in class, examples such as pushing the link register to the top of the stack for storage etc.

# 

# SUMMARY

Project Size : About 600 lines

Number of Variables : About 30

Number of Methods : 9

## Repository:

https://github.com/amulbham/ASSEMBLY\_CSC11/tree/master/proj/BlackJack\_Proj\_2

My project ended up being much larger than I originally anticipated coming in at about 600 lines of code between all the separate functions. This occurred simply because I under anticipated how difficult it would be to program all the different logic points in Assembly Language, something that proves to be simple in C++ can be exponentially more difficult in Assembly. For example, emulating a deck of cards proved to very difficult. At first, I tried to create an array and store random values between 2-53, and then have a player “draw” from this pool of random values to determine their card. However, I realized that I didn’t quite understand arrays fully to effectively implement this concept. So instead, I used a random number generator to give a value between 2-54 which was assigned to the player drawing. Even though their is bound to be duplicates, I figured it would suffice as Casino decks typically are made of four decks making duplicates a part of the game. Stimulating the concept of randomly drawing from a deck, as well as assigning this value a “face” and a card value proved to be the most difficult part of the game to program. This work can all be found within my get card function which took me around 6-8 hours to code and debug.

I also utilized open source code provided by Dr. Lehr to generate a true random number based on time. In addition, I used my previous C++ BlackJack project as a reference and “road map” for all the different logic points in the game. This proved to be crucial as it also allowed me to more easily visualize what needed to be done as C++ is far easier to interpret. Any time I found myself stuck or not knowing what to do, I would glance over the C++ code to give me some guidance. Although Dr Lehr constantly emphasizes the importance of coding assembly projects in C first, this project truly made me appreciate why as the project would have taken me far longer to code without C++ reference code.

# Sample Input/Output

# Pseudocode:

First the user decides if they want to hear the rules of or not

if so, branch to the greeting function to display the rules

else, branch to the start of the game

The player then receives their first two cards

Cards are randomly selected using random number generator, assignment a suit, a card value and determines if the card is a face. Branch to he get card function

The dealer receives their first card

branch to the get card function

Player must now decide if they want to hit or stay

Program calculates the value of each player's hand after each card draw -> get card

if the user decides to hit, they will continue to receive cards till they are over 21, they decide to stay, or hit a BlackJack.

else, the dealer receives their next cards, and continues until they are at 17 or over.

If player or dealer bust, opposing player wins -> the calculations are ignored if either player busts and the game skips to the output

else, if both players have not gone over 21, a winner is determined.

Program calculates winner depending on who had the higher card total

if user has a higher card total, they win

else if the dealer has a higher card total, or the player has gone over 21, the dealer wins

User is prompted to play again or walk away

if user walks away -> exit the program

else, game loops to the start of the game loop, registers are reset for next hand

# Major Variables:

|  |  |  |
| --- | --- | --- |
| Variable Name | Description | Location |
| pcard\_total | stores the value of the players current card | .global main |
| dcard\_total | stores the value of the dealer's current card | .global main |
| card\_1: | keeps a running total of the player’s hand | .global main |
| card\_2: | keeps a running total of the dealer's hand | .global main |
| pagain | determines if the user would like to play another hand | .global main |
| address\_of\_message  address\_of\_finish  address\_of\_lose  address\_of\_win  address\_of\_tie  address\_of\_plose21  address\_of\_plose21 | displayed various messages regarding the outcome of the game, informs the user if they won, lost, tie, hit a blackjack etc. | .global main |
| address\_of\_return: address\_of\_opening: address\_of\_opening2: address\_of\_continue: address\_of\_continueA:  address\_of\_hitorstay: address\_of\_firstcards: address\_of\_dfirstcards: | displayed various messages throughout the game, including the intro message, informing the user of what point the game currently is at, if they want to hit or stay, etc | .global main |
| output\_hearts  output\_diamonds  output\_cloves  output\_spades | stored the string text of each of the four suits that the card could possibly be | .global getcard, .global dcard |
| output\_jack  output\_queen  output\_king  output\_ace | stored the string text for each of the face cards in the case that the player or dealer draws a face | .global getcard, .global dcard |
| address\_of\_return | used to store the link register | .global main |

# Checklist of Concepts:

|  |  |
| --- | --- |
| Concept | Implementation |
| Using link register | Stored the lr upon entering a function and then popped back the lr upon leaving the function  Can be seen in my greeting function, I also utilized push and pop commands in my get card function |
| Passing registers as Parameters for a function | Before entering a function, I made sure beforehand that the proper variables were stored in R0-R3 to ensure they were passed to the get card and dcard functions appropriately for modification, can also be seen upon entering printf and scanf, I loaded the registers with the appropriate outputs, or format for scanf, prior to using them . |
| Branching : bge, bl, blt, beq, bx,b  compare: cmp | Utilized branching like second nature all throughout my program, used it to make decisions, act like if else statements, switches, basically anywhere I needed the program to make a decision based on the input. I also used the bl command to branch to other functions while also passing the link register.  Based on certain circumstances, I used branching to move fluidly and effectively around my program |
| str and ldr - store and load register | Used str command to preserve my link register upon entering a function. Used the ldr command all throughout my program to store memory addresses in certain registers for modification or passing to functions. |
| Looping using labels and branching | I used labels at the start of major points in the program to save writing excess code, for example the hit or miss part of the program can be looped until the player decides to stay or bust, instead of just rewriting the code, I used a label and a branch statement to continuously reuse this code given the user wants to keep hitting. I used this concept in many other parts of my program including the end when the user decides if they would like to play again, I simply created a label at the start of the game, and branch to that label if the user would like to play again. |
| Functions | Utilized functions to separate logic points in my program to streamline the code and make it more modular. For example, I kept the greeting function separated since it is essentially a wall of text explaining the rules, so instead of having that in my main function, where it takes up space and gets in the way of readability. Also calculating the card value and card total required a ton of code and logic that I kept separated from the main function, this allowed me to focus on getting the logic correct in that block of code instead of having it all jumbled up together in my main function. This allowed my main function to simply guide the program while my outside functions did the more complicated tasks. It also allowed me to more easily detect where a bug was arising from as I could individually debug functions. |
| External Functions | I used scanf and printf to read user input and output text to the screen all throughout my program. In addition, I also used the srand and time functions to bring in the current time and then plant the random number seed inside of the rand function which I used to generate a random number representing a card draw |
| Setting Flags using cmp | I used the cmp to compare numbers against register values in order to guide my program to the next step. For example, the dealer keeps drawing cards till their card total is at or above 17 so to ensure this, I used a cmp R1, #17 to set the flag, if the value was greater than or equal (bge) the program would go to the next step, else blt was executed and the program looped . Or when determining the winner of the game, I simply used cmp R1, R2, in other words I just directly compared the card values of the dealer and player, whoever was high was the winner |
| Addition, subtraction of values in registers | Used arm subtraction and addition all over my program to keep a counter, add the card values together, set loop conditions etc. |

# 

# References:

1. I utilized the DivMod function as well as parts of the random number generator graciously provided by Dr. Lehr. I used these two functions in tandem to simulate drawing a card from a randomly shuffled deck of cards.
   1. <https://github.com/ml1150258/LehrMark_CSC11_48982/blob/master/Class/Week9/randTest.s>
   2. <https://github.com/ml1150258/LehrMark_CSC11_48982/blob/master/Class/Week9/mainDivModFuncV2.s>
2. I previously coded BlackJack in C++ in a previous course; I used this source code as the basis for my project and an outline of sorts when dealing with the logic of the game.
   1. <https://github.com/amulbham/CSC-17A/tree/master/proj/BlackJack_Project_>