COMP 2655 Assignment 4: Branching, Arrays and Structures

Given:	Wednesday, October 29, 2025
Due:	Monday, November 17, 2025 before 11:59pm
	electronic submission on D2L
Weight:	6%

Objectives:

- To gain more experience with assembly language and the 68000.
- To continue to practice translation of high-level algorithms to assembly language.
- To learn about conditional and unconditional branching
- To learn about how arrays, structures and arrays of structures are represented at the low-level.
- To understand how to use the "register indirect with offset" and "indexed register indirect with offset" addressing modes.
- To continue to gain experience with testing, debugging and documenting low-level code.

Overview

You are being given a C program that solves a problem that involves an array of structures. The objective of the assignment is to translate the given solution into 68000 assembly language. **The given solution must be used!** The provided program consists of four files. The **`mancala.c`** file contains the main program that you are to convert to assembly language. The files **a4lib.o** and **a4lib.h** are an I/O library. You are only being provided the object format version since you are not required to implement it in assembly language. The **types.h** file contains a number of the #defines from FIRST.S, but it also contains that structure declaration used in both modules. You will need to convert the structure declaration into the corresponding assembly language items, according to the method discussed in class. These files are available in the assignments/a4 subdirectory of the class directory on INS. When compiling on INS both executable files must include in the compile command in order to create an executable program, i.e.

```
gcc mancala.c a4lib.o -o mancala
```

Your first step is to determine how this program works. All input to the program comes from the keyboard. An assembly language version of the I/O library module will also be provided; it too will only be in object format.

You are to convert the given solution as is, i.e. you are not allowed to change the structure of the code. Minor adjustments to the selection and/or repetition structures for efficiency is allowed. In the event of doubt consult with your professor. The array of structures MUST be converted to 68000 assembly code using the method shown in class. Your assembly language solution MUST use this array of structures as specified. Changing the array of structures in anyway will be equivalent to not doing the assignment!!

Your assembly language program must use either *address register indirect with offset* or *indexed address register indirect with offset* when appropriate, i.e. when accessing array elements in the array of structures.

The Problem

Mancala is a two-player strategy board game. The board consists of two rows of the same number of pits, for example 4 and two pits, stores, on each end.

Player 1

Player 1 store	4	3	2	1	Pit Number
Pit Number	1	2	3	4	Player 2 store

Player 2

The player's store is always at the right end of their row of pits. Each player starts with the same number of stones in each of their pits, but there are no stones in their store.

Game Play

Player 1 always starts and turns alternate. On a player's turn they pick one of their pits and that pit must not be empty. The stones in that pit are distributed in a counter-clockwise direction with one stone being deposited in all subsequent pits in your row. At the end of your row a stone is placed in your store. If there are stones remaining, they are then deposited in your opponent's pits continuing counter-clockwise. The only exception is that a stone is never placed in your opponent's store. For example, assuming the initial number of stones is 3, then the original layout is:

Player 1

Player 1 store	4	3	2	1	Pit Number
0	3	3	3	3	0
	3	3	3	3	
Pit Number	1	2	3	4	Player 2 store

Player 2

After Player 1 selects pit 3 the board will look like this, with the updated pit values in red:

Player 1

Player 1 store	4	3	2	1	Pit Number
1	4	0	3	3	0
	4	3	3	3	
Pit Number	1	2	3	4	Player 2 store

Player 2

Stones that are placed in a store will always remain there as a player can't select the store on their turn.

Capturing stones:

If the last stone of your turn is placed in an empty pit on your side of the board, you capture all of the stones in your opponent's pit directly across from the empty pit and the stone placed in the empty pit. The captured stones are places in your store. For example, if the board contents were as follows and it was Player 1's turn:

Player 1

Player 1 store	4	3	2	1	Pit Number
1	0	3	0	3	0
	4	4	3	3	
Pit Number	1	2	3	4	Player 2 store

Player 2

Player 1 selects pit 1 resulting in the following situation:

Player 1

Player 1 store	4	3	2	1	Pit Number
1	1	4	1	0	0
	4	4	3	3	
Pit Number	1	2	3	4	Player 2 store

Player 2

Since Player 1's last stone was placed in and empty pit, they capture the 4 stones in Player 2's pit 1 and their stone in pit 4. Notice that while pit 2 was previously empty the stone place in was NOT the last stone, so it is NOT captured. The resulting board would be:

Player 1

Player 1 store	4	3	2	1	Pit Number
6	0	4	1	0	0
	0	4	3	3	
Pit Number	1	2	3	4	Player 2 store

Player 2

Winning:

The game ends when all pits on one side of the board are empty. The player with the most stones in their store wins the game, regardless of which player ended the game.

There are numerous descriptions of Mancala on the web, one can be found at

https://en.wikipedia.org/wiki/Mancala

Your task is to translate the given interactive Mancala C program into 68000 assembly language.

The only input error checking required is that the Player's input for the pit to be used is both a valid pit number and the pit is not empty.

I/O and Subroutines

In order to allow you to focus on implementing the game, selection and repetition in assembly language you are being provided with an I/O module called **A4LIB.BIN**. It contains the exact same subroutines as used in the C version. These are the ONLY subroutines that can be used for this assignment – you are NOT allowed to create your own subroutines.

IMPORTANT: the subroutines being provided are NOT the way subroutines should be implemented. However, because of where we are in the course this is the best that can be done. The correct method will be presented later in the course and that method will be required in assignment 5.

The difference from the C routines is how the required parameters are provided to each subroutine in assembly by the calling program.

read_char	the user's input value is stored in **d0**, with the character in the low order byte and the random value in byte 2, i.e. just like FIRST.C.
write_char	the character to be output is stored in the low order byte of **d0**
print_string	**a0** holds the address of a null-terminated string. All characters of the string, excluding the null terminator will be output to the current screen location. This routine does NOT output a newline.
print_number	**d1** holds a word sized unsigned integer that will be output to the current screen location. This routine does NOT output a newline.
print_board	**d1** holds the number of Pits per side, this excludes the Player's store and **a0** holds the address of the board array.

These routines are found in the library module called, **A4LIB.BIN**. This is a binary object file that has been assembled without debugging information. Thus, stepping into these routines in the debugger will not provide meaningful information and it is recommended that you use **CTRL-t** to treat the routine as a single instruction. This file can be found in the ***assignments/A4*** folder in the class directory on **Acadshare**, and the **I:** drive on the MRU PC network.

Coding Style and Documentation

You must follow good coding style and documentation practices, as discussed in class. The Coding Standards can be found in the Assignment section of the course D2L site.

Submission & Grading

Your 68000 assembly source file is to be submitted electronically on D2L in the Assessment section for this course. Name your file **`username_a4.s`**, where **`username`** is your MRU username

Your grade will depend not only on program correctness, but also on several other factors, including quality of documentation, the accuracy of translation from C to 68000 assembly language, utilizing the required addressing modes and coding style.