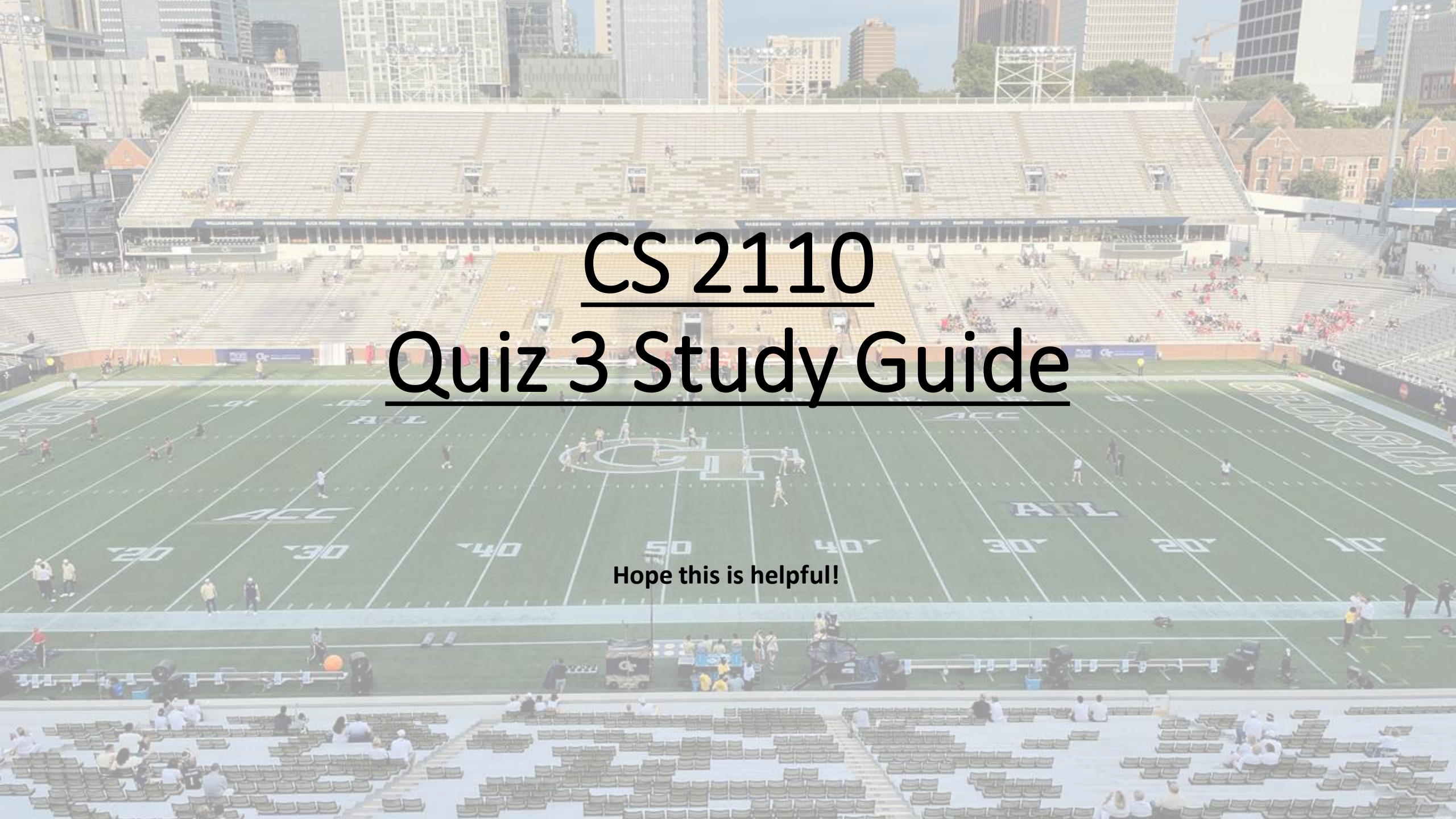

CS 2110
Quiz 3 Study Guide

Hope this is helpful!

Pseudo-Ops

- Pseudo-ops are instructions for the assembler
- 5 main Pseudo-Ops :
  - .orig : Tells the assembler to put the following code at a given memory address. In memory, the address of subsequent lines can be determined based on relation from the .orig address. For example, if .orig x3000 and the next block is a .stringz "BUZZ" followed .end, then 5 blocks in memory are taken with them being (x3000,x3001,x3002,x3003,x3005)
  - .stringz : Tells the assembler to put an array of ASCII values followed by a null terminator(ASCII value of 0). In memory, will take up as much space as length of the string + 1(for null terminator). Essentially an array of ASCII values is placed.
  - .blkw : Tells the assembler to make the next n locations reserved for a specific label
  - .fill : Tells the assembler to put the given label at the next set of addresses
  - .end : Tells the assembler that the end of a given block has been reached, usually in line with the .orig

LC-3 instruction format and assembly/disassembly

- In LC-3 instructions, there are key parts to the Op code based upon different situations different parts will be included.
  - DR --> Refers to a destination register. At the end of an instruction, when the data will end up in a register, DR will be part of the instruction.
  - SR1 --> Refers to source register one. Coming out of the main register there are 2 source registers. SR1 can be thought of as the A.
  - SR2 --> Refers to source register one. Coming out of the main register there are 2 source registers. SR2 can be thought of as the B.
  - BaseR --> Refers to base register. Many operations will need to add some value. Depending on the instruction it will be either a base value or it will be from the PC offset.
- LC-3 Instructions can have different parts do the instructions including:
  - Imm5 --> Last 5 bits of instruction
  - Offset6 --> Sign Extends 6 bits
  - PCoffset9 --> 9 bits
  - PCoffset11 --> 11bits
  - Trapvect8 --> Traps are in-built subroutines with the purpose of simplifying instructions

LC-3 assembly instructions and functions

Instruction(W/ Assembly example) * Indicates sets CC	Function
ADD* (EX: ADD R0,R0,#1)	Adds value from source register to destination register with a constant
AND* (EX: AND R0,R0,#0)	Ands a register and puts it in a destination register
NOT*(EX: NOT R0,R0)	Flips the bits
BR(EX: BRzp [LABELNAME]	When CC matches the Branch condition jumps and continues executing code at given label
JMP	Unconditionally jumps to the given register/label
JSR (EX: JSR Mult)	Branches to the specified subroutine
LEA(EX: LEA R4, Target)	Load the effective address of a specified label. Does not go through memory.
LD*(EX: LD R1, Value)	Loads the value of what was at a given label/register. Does this by sign extending to 16 bits and adding with PC Offset. Goes through memory once
LDR*(EX: LDR R5, #4)	Loads the value of what was at a given label/register. Does this by sign extending to 16 bits and adding with base. Goes through memory once
LDI* (EX: LDI R4, LOADER)	Loads the value of what was at a given label/register. Does this by sign extending to 16 bits and adding with PC Offset. Then repeats using the value as an address. Goes through memory twice.

LC-3 assembly instructions and functions(Continued)

Instruction(W/ Assembly example) * Indicates sets CC	Function
ST versus LD	Main difference between the two types of instructions is that with LD instructions the end goal is to get data into registers. On the other hand, in ST instructions the end goal is get data into memory.
ST (EX: ST R1, Result)	Takes in the specified data from the register and stores in the memory location associated with the specified label
STR (EX: STR R4, R2, #5)	Value of specified register is added to a memory location. The address of the specified value is found by computing by adding sign extension of 8 bits to 16 bits with the value of the second register.
STI (EX: STI R4, TWICE	Sign extends to 16 bits and adds to PC Offset. This gives an address which then we want the value of. That value then become the address of where we want to store our specified value in memory.

Stack

- The Stack uses the LIFO concept to hold essential elements of a subroutine.
- The stack is the region in memory where the temporary program data is stored. This includes the subroutine calls which consists of the arguments, return values, and the previous register values.
  - Sometimes, local variables are needed which are also stored in the Stack when registers are not enough.
- The Stack grows "downward" from the location specified from the beginning of the stack.
- There are 4 main parts to the Stack:
  - R7 --> Return address
  - R6 --> Stack pointer always points to the top of the stack
  - R5 --> Frame pointer is a fixed location so all args can be called in reference from the frame pointer
    - The formula for locations of arguments are R5, (#3+N) where N represents the place in the number of arguments
- Pushing data onto the stack:
  - In order to use data in a subroutine, it must have the data in the stack. So, the arguments must be pushed(added) onto the stack, but because the Stack involves the LIFO principles, it must be done in reverse order.
  - For each argument that becomes pushed onto the stack, the stack becomes "taller" by 1, so R6 must increase by one.
  - Lastly, the given register value needs to be stored in at the R6(stack pointer) address
  - Example:
    - ADD R6, R6, #-1
    - STR R4, R6, #0

Stack (Continued)

- **At the end of the subroutine, all the stuff that was used was specific to this instance of the subroutine call and will no longer be needed for subsequent instances of the subroutine, so it is important to get rid of it by popping of the arguments.**
- **Popping off the stack:**
  - **Essentially, reverse order and reverse instructions of what went on in the pushing segments, so restore the values in the registers. And then move the top of the stack back to where it was**
    - **Example:**
      - **LDR R4, R6, #0**
      - **ADD R6, R6, #1**

Alright, Stack Pop off then

LC-3 Calling Convention

- **With a calling convention, there are two main aspects to it: Caller and callee.**
- **The caller is the one that pushes args in reverse order and calls the necessary subroutines.**
- **The callee stores both the return value and address, the old frame pointer, and makes sure the frame pointer and the stack pointer are referring to the same value. It also makes sure there is the necessary space to save the registers.**

Preprocessor and Macros

- **Preprocessors convey to the assembler whether this is a file inclusion call or a subroutine call. This will**
- **Macros are a directive in C that state the beginning of a new subroutine call.**
  - **Example: #define PI 3.141593**
- **The #include refers to the declarations and definitions, but does not include any executable code. They are technically not part of C, but just preprocessor directives.**