CS061 – Lab 08
Emulating a Compiler

1 High Level Description

The purpose of this lab is to do some simple case-conversion, print a table of instructions & opcodes, and—if you are awesome enough—to implement a basic opcode checker, which emulates one basic element of a compiler.

2 Our Objectives for This Week

1. Exercise 01 ~ Converting a string to uppercase
2. Exercise 02 ~ Subroutine: Prints out LC3 instructions & op-codes
3. Exercise 03 ~ Subroutine: Op-code parser
   (to get full points for this lab, you must have at least made a serious attempt at exercise 03 before the end of lab)

Exercise 01

1. Write the following subroutine:

```
;-------------------------------------------------------------------------------------------------------------
--
; Subroutine: SUB_TO_UPPER
; Parameter (R0): Address to store a string at
; Postcondition: The subroutine has allowed the user to input a string,
;                terminated by the [ENTER] key, has converted the string
;                to upper-case, and has stored it in a null-terminated array that
;                starts at (R0).
; Return Value: R0 □ The address of the now upper case string.
;-------------------------------------------------------------------------------------------------------------
--
```

Hints:

- The conversion of a letter to uppercase can be done with a total of two lines of LC3 code. Look at the difference in the hexadecimal values of a lowercase vs. an uppercase letter.
- Use bit-masking.

Test Harness:

Write the following test harness (feel free to paste the description in your code)

```
;-------------------------------------------------------------------------------------------------------------
--
; Test Harness for SUB_TO_UPPER subroutine:
;
; (1) R0 □ Some address where we will store a user-input string
; (2) Call SUB_TO_UPPER subroutine
; (3) Trap x22 (i.e. print out the now-uppercase string)
;-------------------------------------------------------------------------------------------------------------
--
```

Exercise 02

Write the following subroutine:

```
;-------------------------------------------------------------------------------------------
; Subroutine: SUB_PRINT_OPCODES
; Parameters: None
; Postcondition: The subroutine has printed out a list of every LC3 instruction
;                and corresponding opcode in the following format:
;                ADD = 0001
;                AND = 0101
;                BR = 0000
;                …
; Return Value: None
;-------------------------------------------------------------------------------------------
```

Specifications:

- The data block of the subroutine must contain:
  - An array of decimal values (not strings), each one representing an LC3 opcode (i.e. #1, #5, #0, …)
  - An array of strings, each one representing an LC3 instruction (i.e. “ADD”, “AND”, “BR”, …)

Hints:

- Store the array of opcodes normally
- To implement the array of strings:
  - See the last page of the lab
  - Write a series of .STRINGZ pseudo ops, one for each instruction
  - Terminate the “array” of strings with a .FILL #-1
- To iterate through the two arrays, keep a pointer to each array
  - Iterate through the opcodes one memory location at a time
  - Iterate through the array of strings, printing each letter as you get to it (Trap x21), stopping at the #0 (don’t use Trap x22; it’s harder that way).
  - You will know to stop iterating when you reach the end (i.e. the #-1) of the array of strings (because that will flag that there are no more strings to print)

Test Harness:

Write the following test harness (feel free to paste the description in your code)

```
;-------------------------------------------------------------------------------------------
; Test Harness for SUB_PRINT_OPCODES subroutine:
; (1) Call SUB_TO_UPPER subroutine
;-------------------------------------------------------------------------------------------
```

Fair Warning:
If you use .STRINGZ to simply store “ADD = 0001” (or any similar cheating hack-job) etc and print it out that way, you will not only get no credit for the lab, you will also receive a heavy sigh and will be walked away from in tired dismissal by the TA.

Exercise 03
(Must finish before the end of your lab section to get extra credit points)

Modify the subroutine from Exercise 02 so that the user can repeatedly type in instruction names (example: “ADD”, “JSR”, “BR”) and be told whether the instruction is valid (whether the instruction exists).

Specifications:

- The subroutine now allows the user to type an [ENTER]-terminated string.
- The input string is compared with the array of LC3 instructions.
- If the input string matches one of the instructions, then that line from the opcode table is printed out. Otherwise “Invalid instruction” is printed.

Example:

- The user types “JSRR[ENTER]”
    - The subroutine prints “JSRR = 0100”
- The user types “AMD[ENTER]”
    - The subroutine prints “Invalid instruction”

“Gee, I wonder how many people were responsible enough to bring their book to lab?” The TA muttered to himself.

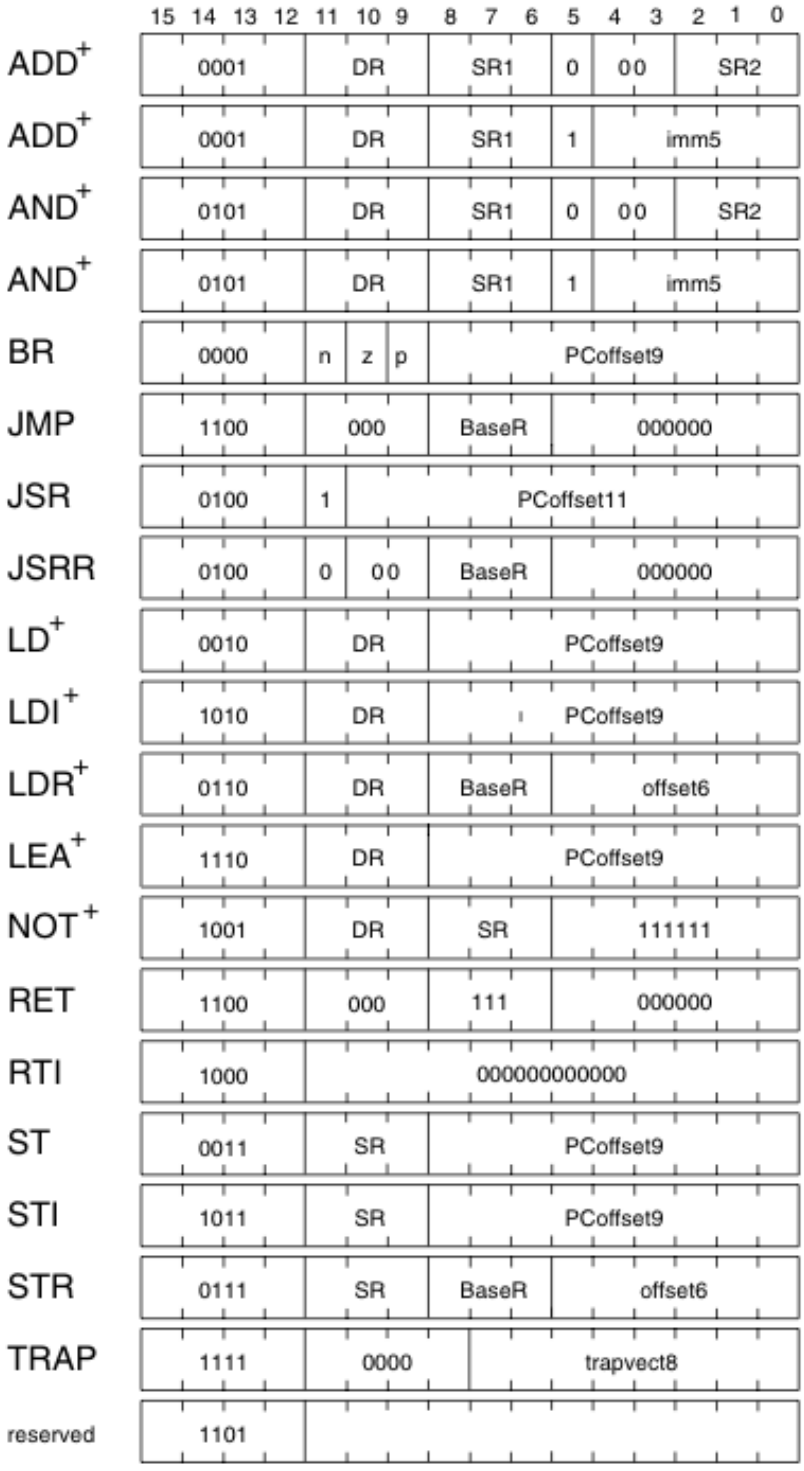

Figure A.2 Format of the entire LC-3 instruction set. **Note:** + indicates instructions that modify condition codes