# ECE220: Computer Systems and Programming

## Spring 2016 - Midterm Exam 1

### February 15, 2016

- This is a closed book exam except for 1 sheet of notes
- You may not use any personal electronic devices, such as cellphone and calculator
- Absolutely no interaction between students is allowed
- Manage your time properly to get the most points
- This booklet should be handed in at the end of the exam
- All files must be committed via svn

Name:_____

NetID:_____

Room:_____

## Problem 1 (30 points): Phone Number

Determine if a user input is a valid phone number or not. A valid phone number is in the following format (where x means any digit 0-9):

xxxxxxxxxx

Each input must also have **no leading or trailing whitespace**. (There should be no tabs or spaces at the beginning or end).

After the user types their input and presses enter, your program should then print "Valid Phone Number." if the phone number is valid otherwise print "Invalid Phone Number."

**Your program should not print whether the input is a valid phone number or not until after the user presses enter**.

### Part 1 – Getting inputs without using TRAP (10 points):

For this part you must write code that gets the user input and echos it to the screen without using any trap subroutines. **The user inputs should be stored starting at memory address x5000.** You cannot use IN, GETC, OUT, etc. in Part 1. Think back to how LC-3 reads input with the KBSR and KBDR and prints output with the DSR and DDR. If you cannot successfully do part 1 you can use GETC/IN/OUT but will lose all points for this part.

### Part 2 – Checking whether inputs are valid (20 points):

For this part you must determine if the phone number entered is valid or not and print the corresponding message to the console. A valid phone number has exactly 10 digits between 0 to 9. You can use TRAP in Part 2.

Example:
0123456789   – is a valid phone number
21730#          – is not a valid phone number
217300  0000 – is not a valid phone number

Write your program in **phone_num.asm**. You are not required to use subroutines in this problem, but you may write subroutines if you like.

## Problem 2 (30 points): Print Number in Base 7

Write a program in **base7.asm** to print a positive value stored in R3 in base 7 format.

**Algorithm**
1) Divide value stored in R3 by 7. Store quotient in R3 and push remainder to stack.
2) If quotient (value in R3) is not 0, go to step 1
3) Pop values off the stack one at a time till the stack is empty. Add ASCII offset for '0' and print to screen. You do not need to print the 'x' in front.

You can use any TRAP you find useful. The DIV, PUSH and POP subroutines are given to you. PUSH and POP subroutines are the same as the ones given in lab and MP. Stack starts at x4000 and ends at x3FF0, which means the first available spot on the stack is at x4000 and the last available spot at x3FF0. You are not required to use subroutines in this problem, but you may write subroutines if you like.

**Testing:**

Assembly your code using the command:

*~$ lc3as base7.asm*

You may test your code using the following command and set R3 to 9:

*~$ lc3sim base7.obj*

*register R3 9*

*finish*

## Problem 3 (30 points): Reverse Characters

You will be provided n characters, where n is a positive number stored in memory location x4FFF. These characters are stored in sequential memory addresses, beginning at x5000. Your code should swap the order of the characters, so the last character appears at x5000, the second to last character at x5001, etc.

**SWAPMEM:** Implement this subroutine at the label SWAPMEM. The inputs are R0 and R1, which contain memory addresses. If mem[R0]=A and mem[R1]=B, then after the subroutine, mem[R0]=B and mem[R1]=A.

**REVERSE:** Implement this subroutine at the label REVERSE. This code should reverse the order of the characters in memory, so memory address x5000 is swapped with x5009, x5001 with x5008, and so on. SWAPMEM subroutine must be be called here to swap addresses, not in the main user program.

**Details:**

- Code in LC-3 assembly in **reverse.asm**. You may define extra labels and values as needed.

- You may use any TRAPs your find useful.

- You must use subroutines, with the JSR and RET instructions, or you may lose points.

**Testing:**

Assemble your code and the test input file using the command
*~$ lc3as reverse.asm*
*~$ lc3as input.asm*
You may test your code using the following commands for reversing 10 characters:
*~$ lc3sim*
*file input.obj*
*file reverse.obj*
*finish*
Check your output using the command
*dump x5000*

## Expected Output

```
4FF8:                                        0030 0039 0038 0037           0987
5004: 0036 0035 0034 0033 0032 0031 0000 0000 0000 0000 0000 0000  654321......
5010: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ............
```

| Memory Address | Value |
|----------------|-------|
| x5000 | x0030 |
| x5001 | x0039 |
| x5002 | x0038 |
| x5003 | x0037 |
| x5004 | x0036 |
| x5005 | x0035 |
| x5006 | x0034 |
| x5007 | x0033 |
| x5008 | x0032 |
| x5009 | x0031 |

## Problem4: Concepts (10 points)

In **prob4.txt**, record your answer to the following short answer questions. Please limit each answer to no more than 3 sentences.

1.  In LC-3, what is the size of MAR and MDR? And explain why. (5 points)

2.  What is the defining characteristic of a stack (in terms of how it is being accessed)? (5 points)

## Table E.2    The Standard ASCII Table

| ASCII Character | Dec | Hex | ASCII Character | Dec | Hex | ASCII Character | Dec | Hex | ASCII Character | Dec | Hex |
|---|---|---|---|---|---|---|---|---|---|---|---|
| nul | 0 | 00 | sp | 32 | 20 | @ | 64 | 40 | ` | 96 | 60 |
| soh | 1 | 01 | ! | 33 | 21 | A | 65 | 41 | a | 97 | 61 |
| stx | 2 | 02 | " | 34 | 22 | B | 66 | 42 | b | 98 | 62 |
| etx | 3 | 03 | # | 35 | 23 | C | 67 | 43 | c | 99 | 63 |
| eot | 4 | 04 | $ | 36 | 24 | D | 68 | 44 | d | 100 | 64 |
| enq | 5 | 05 | % | 37 | 25 | E | 69 | 45 | e | 101 | 65 |
| ack | 6 | 06 | & | 38 | 26 | F | 70 | 46 | f | 102 | 66 |
| bel | 7 | 07 | ' | 39 | 27 | G | 71 | 47 | g | 103 | 67 |
| bs | 8 | 08 | ( | 40 | 28 | H | 72 | 48 | h | 104 | 68 |
| ht | 9 | 09 | ) | 41 | 29 | I | 73 | 49 | i | 105 | 69 |
| lf | 10 | 0A | * | 42 | 2A | J | 74 | 4A | j | 106 | 6A |
| vt | 11 | 0B | + | 43 | 2B | K | 75 | 4B | k | 107 | 6B |
| ff | 12 | 0C | ' | 44 | 2C | L | 76 | 4C | l | 108 | 6C |
| cr | 13 | 0D | - | 45 | 2D | M | 77 | 4D | m | 109 | 6D |
| so | 14 | 0E | . | 46 | 2E | N | 78 | 4E | n | 110 | 6E |
| si | 15 | 0F | / | 47 | 2F | O | 79 | 4F | o | 111 | 6F |
| dle | 16 | 10 | 0 | 48 | 30 | P | 80 | 50 | p | 112 | 70 |
| dc1 | 17 | 11 | 1 | 49 | 31 | Q | 81 | 51 | q | 113 | 71 |
| dc2 | 18 | 12 | 2 | 50 | 32 | R | 82 | 52 | r | 114 | 72 |
| dc3 | 19 | 13 | 3 | 51 | 33 | S | 83 | 53 | s | 115 | 73 |
| dc4 | 20 | 14 | 4 | 52 | 34 | T | 84 | 54 | t | 116 | 74 |
| nak | 21 | 15 | 5 | 53 | 35 | U | 85 | 55 | u | 117 | 75 |
| syn | 22 | 16 | 6 | 54 | 36 | V | 86 | 56 | v | 118 | 76 |
| etb | 23 | 17 | 7 | 55 | 37 | W | 87 | 57 | w | 119 | 77 |
| can | 24 | 18 | 8 | 56 | 38 | X | 88 | 58 | x | 120 | 78 |
| em | 25 | 19 | 9 | 57 | 39 | Y | 89 | 59 | y | 121 | 79 |
| sub | 26 | 1A | : | 58 | 3A | Z | 90 | 5A | z | 122 | 7A |
| esc | 27 | 1B | ; | 59 | 3B | [ | 91 | 5B | { | 123 | 7B |
| fs | 28 | 1C | < | 60 | 3C | \ | 92 | 5C | | | 124 | 7C |
| gs | 29 | 1D | = | 61 | 3D | ] | 93 | 5D | } | 125 | 7D |
| rs | 30 | 1E | > | 62 | 3E | ^ | 94 | 5E | ~ | 126 | 7E |
| us | 31 | 1F | ? | 63 | 3F | _ | 95 | 5F | del | 127 | 7F |

NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

## ADD
| 0001 | DR | SR1 | 0 | 00 | SR2 |

ADD DR, SR1, SR2 — DR ← SR1 + SR2, Setcc

## ADD
| 0001 | DR | SR1 | 1 | imm5 |

ADD DR, SR1, *imm5* — DR ← SR1 + SEXT(imm5), Setcc

## AND
| 0101 | DR | SR1 | 0 | 00 | SR2 |

AND DR, SR1, SR2 — DR ← SR1 AND SR2, Setcc

## AND
| 0101 | DR | SR1 | 1 | imm5 |

AND DR, SR1, *imm5* — DR ← SR1 AND SEXT(imm5), Setcc

## BR
| 0000 | n | z | p | PCoffset9 |

BR{nzp} *PCoffset9* — ((n AND N) OR (z AND Z) OR (p AND P)): PC ← PC + SEXT(PCoffset9)

## JMP
| 1100 | 000 | BaseR | 000000 |

JMP BaseR — PC ← BaseR

## JSR
| 0100 | 1 | PCoffset11 |

JSR *PCoffset11* — R7 ← PC, PC ← PC + SEXT(PCoffset11)

## TRAP
| 1111 | 0000 | trapvect8 |

TRAP *trapvect8* — R7 ← PC, PC ← M[ZEXT(trapvect8)]

## LD
| 0010 | DR | PCoffset9 |

LD DR, *PCoffset9* — DR ← M[PC + SEXT(PCoffset9)], Setcc

## LDI
| 1010 | DR | PCoffset9 |

LDI DR, *PCoffset9* — DR ← M[M[PC + SEXT(PCoffset9)]], Setcc

## LDR
| 0110 | DR | BaseR | offset6 |

LDR DR, BaseR, *offset6* — DR ← M[BaseR + SEXT(offset6)], Setcc

## LEA
| 1110 | DR | PCoffset9 |

LEA DR, *PCoffset9* — DR ← PC + SEXT(PCoffset9), Setcc

## NOT
| 1001 | DR | SR | 111111 |

NOT DR, SR — DR ← NOT SR, Setcc

## ST
| 0011 | SR | PCoffset9 |

ST SR, *PCoffset9* — M[PC + SEXT(PCoffset9)] ← SR

## STI
| 1011 | SR | PCoffset9 |

STI SR, *PCoffset9* — M[M[PC + SEXT(PCoffset9)]] ← SR

## STR
| 0111 | SR | BaseR | offset6 |

STR SR, BaseR, *offset6* — M[BaseR] + SEXT(offset6)] ← SR

**End of ECE 220 Midterm Exam 1**