COMPSCI 210

Tutorial Week 6 – LC3 and Assembly 1

- Today we will cover some exercises and questions regarding assembly and the LC-3.
- Keep in mind that these will be useful for solving the problems you may face while doing assignment 2!

Assignment 2

- 1) *The digits of your ID must not be directly visible in their proper order in the assembly language code!*

- E.g. swap letters, reverse string, a substitution cipher or an intermediary encoding with ASCII conversion look-up table.

- 2) How can multiplication and division on LC-3 be performed?

- Answer: iteration.
  - Multiplication: through iterative addition e.g. 2*5 = 2 + 2 + 2 + 2 + 2
  - Division: through iterative subtraction e.g. 10/3 = the number of times you can subtract 2 before the remainder is <= 0, which is 3 remainder 1.

  - What is nice about binary representation of powers of two?

  - What about exponentiation? (This is a very similar process. Example next.)

- 3) How would you calculate powers of ten in binary using only addition?
  - Idea is the same for powers of any base:
  - $A = 10^2 = 10*10 = 10 + 10 + 10 + 10 + 10 +$
  - $10 + 10 + 10 + 10 + 10$
  - $B = A*10 = 10^3 = A + A + A + A + A +$
  - $A + A + A + A + A$

- 4) How could arithmetic to deal with 2s compliment integers larger than 16 bits be handled on LC-3?

- By using an arbitrary precision arithmetic approach.
  - Numbers that are larger than 16-bits will here be referred to as ‘BigNums’.
- A possible approach focuses on manually handling carry bits (feel free to critique it):
- Allocate a few memory locations (e.g. 4 locations = 4*16 bits to give 64 bits, enough for values up to 2^50) to store BigNums across them.
- Process BigNums in blocks:
  - Do this by handling the carries manually when performing LC-3 ADD operations and adding any carries into the next data block.
  - To do this 15 of the 16 bits could be used to store values in each of the memory locations so that the carry could be identified in the 16th bit during addition.
  - LC-3 has 8 registers so loading and storing between registers and memory to deal with two BigNum integers.

- This process would also work with subtraction by filling a BigNum memory location with a divisor value, then taking the 2s complement of the data while zeroing the 16th bits of the first 3 lower data blocks (we want those free for noting the carries).

- Example (on board) using two 4-bit blocks (and a system with 4-bit 2s complement):
  - The large number problem is basically an implementation issue!

REVIEW

- After reviewing the past few exercises on multiplication, division and bignums – it’s probably clear that implementing these features with an ISA like the LC-3 is a lot of work!

- What would be an alternative implementation approach? See next slide.

- 5) Using Look-up tables
- Pre-compute values and store them in a program.
  - Benefit:
    - Reduce programming effort if data can be obtained easily
    - Reduce computational complexity of program
  - But, may impact on code size, or be inflexible if more values or higher data accuracy is needed
  - Example ->

- How could this table be used?

- E.g. (assume all registers cleared, except for R4)
- LEA R5, LookUp10
- ADD R5, R5, R4
- LDR R4, R5, #0
- ADD R0, R0, R4

- R4 stores a number n, 0-9
- R0 stores the result of n * 10.

LookUp10	.FILL	#0
	.FILL	#10
	.FILL	#20
	.FILL	#30
	.FILL	#40
	.FILL	#50
	.FILL	#60
	.FILL	#70
	.FILL	#80
	.FILL	#90

Exercises

- 6) The following program adds the values stored in memory locations A, B, and C, and stores the result into memory. There are two errors in the code. For each, describe the error and indicate whether it will be detected at assembly or run time.

- (on next slide)

Line No.		
1		.ORIG x3000
2	ONE	LD R0, A
3		ADD R1, R1, R0
4	TWO	LD R0, B
5		ADD R1, R1, R0
6	THREE	LD R0, C
7		ADD R1, R1, R0
8		ST R1, SUM
9		TRAP x25
10	A	.FILL x0001
11	B	.FILL x0002
12	C	.FILL x0003
13	D	.FILL x0004
14		.END

- Answer:
  - SUM is not defined – assembly time error.

  - R1 is not cleared before the ADD operation, run time error (R1 might contain a value and should be set to zero).

- 7) The following is an LC-3 program that performs a function. Assume a sequence of integers is stored in consecutive memory locations, one integer per memory location, starting at the location x4000. The sequence of numbers terminates with the number x0000. What does the following program do?
- (on next slide)

Line No.		
1		.ORIG x3000
2		LD R0, NUMBERS
3		LD R2, MASK
4	LOOP	LDR R1, R0, #0
5		BRz DONE
6		AND R5, R1, R2
7		BRz L1
8		BRnzp NEXT
9	L1	ADD R1, R1, R1
10		STR R1, R0, #0
11	NEXT	ADD R0, R0, #1
12		BRnzp LOOP
13	DONE	HALT
14	NUMBERS	.FILL x4000
15	MASK	.FILL x8000
16		.END

- Answer: The program finds all positive numbers, doubles them, and stores the doubled results back to their original locations.
    - The mask x8000 has a one in the MSB, therefore an AND operation with a negative number will give a result >0 and on line 7, wont branch to L1.

- 8) What values do registers R0,..R4 have after this code block is run? Assume all registers are cleared beforehand.
  - For this exercise it's OK to write the equivalent assembly code instead of the hexadecimal for each register.

…		
0xA400	THIS1	LEA R0, THIS1
0xA401	THIS2	LD R1, THIS2
0xA402	THIS3	LDI R2, THIS5
0xA403	THIS4	LDR R3, R0, #2
0xA404	THIS5	.FILL xA400
…		

- Answer:

- R0 = 0xA400
- R1 = LD R1, THIS2
- R2 = LEA R0, THIS
- R3 = LDI R2, THIS5

- 9) What are two possible formats you could use to store integers in LC-3 memory?

- Answer: binary numeral system or ASCII format ….. Or binary coded decimal (BCD)

- E.g. representing 11_{10} in the following formats:
  - Binary: 00001011 (8-bit representation)
  - BCD: 0001 0001 (i.e. two nibbles each with a value of one, each nibble having a max. of 1001
  - ASCII: x3131 (hexadecimal)

- What formats would be useful for your assignment?

- 10) Say you were to write a ‘long’ program on the LC-3 and needed to store a ‘large’ amount of data in your code. You also note that, for example the LD and LDI opcodes use 9 bits to specify Pc-offsets in order to access the program data.
- What could you do to ensure that needed data is always no further than +255 to -256 addresses away (the range for 9-bits 2s compliment)?

- Suggestion: you could intersperse code with data, so that required data isn't more than +255 or -256 addresses away.
  - Using the BR statement would allow for skipping over the data sections.

- 11) What is the benefit of using interrupts to handle I/O? When might you use polling for handling I/O?

- Answer: the processor can spend time performing other tasks instead of polling a device continually.
- Polling can be efficient when events are being received regularly.

- 12) Something worth doing: review the TRAP service routines for LC-3.