LC4 Design and PennSim

CIS 2400 Recitation 4

Recitation Outline

- LC4 Design
  - Review
  - Practice
- VM Demo
  - Terminal
  - PennSim

LC4 Design

LC4 ISA

- All LC4 instructions are associated with
- Handout located on the course website
  - https://www.seas.upenn.edu/~cis2400/current/documents/LC4Instructions.pdf
- Will be provided as references on the exam

LC4 Review

- All code can be deconstructed down to instructions
- These instructions can do many of the basic operations we are used to seeing in code
  - Example: how would we write

    ```
    int R0 = 0;
    R0--;
    ```

    In LC4?

```
CONST R0, #0
ADD R0, R0, #-1
```

PRO TIP: look at the LC4 Instruction sheet

LC4 Review: If & Loops in LC4

- Not all programming constructs have direct LC4 instructions

- How would we implement

```
if (R0 >= 3)
  R1 = R0;
```

```
START
     CMPI R0, #3
     BRn AFTER_IF
     ADD R1, R0, #0
AFTER_IF  ...
```

LC4 Review: If & Loops in LC4

- Not all programming constructs have direct LC4 instructions

- How would we implement

```
if (R0 != R2) {
   R1 = R2;
} else {
   R1 = 0;
}
```

```
START
   CMP R0, R2
   BRz ELSE
   ADD R1, R2, #0
   JMP AFTER
ELSE CONST R1, #0
AFTER
    ...
```

LC4 Review: If & Loops in LC4

- Not all programming constructs have direct LC4 instructions

- How would we implement

```
for (R0 = 0; R0 < R6; R0++) {
   // ...
}
```

```
   CONST R0, #0
START_LOOP
   CMP R0, R6
   BRzp AFTER_LOOP
   ; ...
   ADD R0, R0, #1
   JMP START_LOOP
AFTER_LOOP
   ...
```

Assembly Programming Strategy

- One approach
  - Start by writing a pseudocode program
    - Try to keep code “simple”
      - don’t overuse variables, avoid recursion, etc
    - Comment while you do this
  - Translate each variable to a register
  - Translate each line/piece of code to assembly
  - Test your assembly to make sure it works

Practice: Fibbonacci

- Write an LC4 assembly program that computes the nth Fibonacci number where n is provided in R0 and the output number is produced in R1.
  - You can assume that the value provided in R0 will be greater than or equal to 2.
  - Note:
    - Fibb(0) = 0
    - Fibb(1) = 1
    - Fibb(2) = 1
    - Fibb(n) = Fibb(n-1) + Fibb(n-2)

Practice: Fibonacci

- Pseudocode

```
iter = 2
prev = 1
curr = 1
while(iter != N) {
    tmp = curr + prev
    prev = curr
    curr = tmp
    iter++
}
result = curr
```

```
        CONST R2, #2
        CONST R3, #1
        CONST R4, #1
LOOP        CMP R2, R0
        BRz DONE
        ADD R5, R3, R4
        ADD R3, R4, #0
        ADD R4, R5, #0
        ADD R2, R2, #1
        JMP LOOP
DONE        ADD R1, R4, #0
END
```

Practice: Prefix Sum

- Write an LC4 assembly program that computes the prefix sum for a given list. A pointer to the initial list is given in R0, with its length in R1, and a pointer to an output list is in R2.
    - You can assume that the length of the list is at least 1 and that the output list is the same length as the input list.
    - Example:
        - Prefix_Sum([1, 3, -2, 4], 4)
        - = [1, 1+3, 1+3+-2, 1+3+-2+4]
        - = [1, 4, 2, 6]

Practice: Prefix Sum

```
iters = 0
sum = 0
while(iters < len) {
   temp = mem[input_ptr]
   sum += temp
   mem[output_ptr] = sum
   output_ptr++
   input_ptr++
   iters++
}
```

```
    CONST R3, #0
    CONST R4, #0
LOOP    CMP R3, R1
    BRzp END
    LDR R5, R0, #0
    ADD R4, R4, R5
    STR R4, R2, #0
    ADD R2, R2, #1
    ADD R0, R0, #1
    ADD R3, R3, #1
    JMP LOOP
END
```

Terminal + PennSim Demo

Linux Command Line

- Why do we need it?
  - Allows for greater control of the computer
  - Can run and combine programs in ways that we lack with the GUI

Linux Commands Reference

- **cd <path>**
  - Changes what directory you are currently in to the one specified by the path
- **ls <path>**
  - Lists all entries in the specified directory, or current directory if path is not specified
- **cp <source> <destination>**
  - Copies the source file to the specified destination file
- **mv <source> <destination>**
  - Like cp, but moves instead of copies
- **rm <path>**
  - Removes a specified file

Linux Commands Reference

- **touch <path>**
  - Creates an empty file
- **mkdir <path>**
  - Creates a directory at the specified path
- **sudo <command>**
  - Runs the specified command as super user/administrator
  - (**S**uper **U**ser **DO**)

- All of these commands have optional input flags that provide other functionality

Linux Commands Reference (Advanced)

- **nano <path>**
  - Opens the specified file in the terminal with the simple text editor “nano”
- **vim <path>**
  - Opens the specified file in the terminal with a more complex text editor “vim”. (Travis uses for almost everything)
- **emacs <path>**
  - Like vim, but a different editor
- **find**
  - Used for finding a specified file
- **grep <regex>**
  - Searches through some input for anything matching the regex

Linux Commands

- There’s a lot more commands and ways to combine them!
- If you ever forget a command, Google!

PennSim

- Java .jar file
  - Distributable Java program that should work system-independent
- Provides a place for you to test, debug, and run LC4 code
- Will be used in some future homework assignments

PennSim Commands

- **reset**
  - resets memory, registers, and breakpoints
- **as <input_asm> <output_obj>**
  - Assembles ("compiles") the asm file to an object file
- **ld <obj_file>**
  - Loads the specified object file into PennSim
- **set <register> <value>**
  - Set specified register to specified value
- **break <cmd> <label>**
  - Can be used to set or remove breakpoints
- **trace -on <output_file>**
  - Writes the trace to an output file

That’s all we have for today!

Reminders:

- TA-lead recitations will take place on
  - Tuesdays 6:30-8:00pm in Moore 100A
  - Wednesday 12:00-1:30pm in Moore 100C
- Check the course website for OH times
- Check-in 04 is due WEDNESDAY
- HW4 is due this Friday