RUTGERS

CS211 Computer Architecture

Fall 2020

Recitation 10

Today

- HW5
- Next week:
  - Caches
  - DLD

HW5 Notes

- You are writing an assembly interpreter
  - What this means is that you are going to “interpret” an assembly file (list of assembly instructions) and print as necessary

HW5 Notes

- What you need to implement – interpret.c
  - Read an .asm file
    - This is not any different than a text file, just read using fopen() as you have
    - Will not be more than 100 lines (use this information to your advantage)
    - All lowercase

HW5 Notes

- Information about the files
  - Will have instructions using 4 registers
    - *Signed* 16-bit – this is important!
    - Will only use register names: `ax, bx, cx, dx`
  - Your job is to read every line, and follow the appropriate instruction (in order)
    - Maintain what’s stored in the registers – this is similar to hw3 where you read and follow instructions until EOF

HW5 Notes

- Information about the files
  - Instruction types
    - mov □ format is `mov x y`
    - Arithmetic
    - Jumps **
      - This is trickier to implement than everything, but do not overthink it!
      - `L` □ line number of the file (think about how you will refer to line numbers in the file)

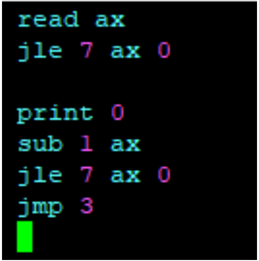
HW5 Notes

- Information about the files
  - when you see
    - `read ax` □ this is asking for user input (stdin) to store a value into `ax`
    - `print ax` □ this is asking to print the value of `ax`
  - Note that because you are asked to input a value manually, your output should match the output determined by order of the instructions
    - Verify this by doing the assembly instructions by hand and checking if your output is correct!

HW5 Notes

- Hints
  - There are many ways to go about this, but your priority should be to figure out how you will **store** each instruction + its arguments (if any)
  - Figure out the **data structure** that's best for you
    - It's easy to go line by line, but implementing jumps are going to need a little more thought put into them
      - The values of registers will of course affect whether to jump or not
        - » If the jump condition is not satisfied, just move onto the next line
    - It's as simple as using arrays

Example – Provided Testcase



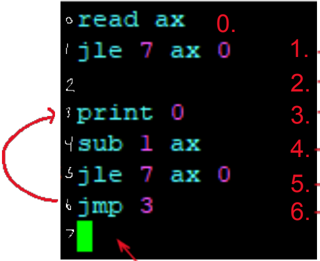
```
read ax
jle 7 ax 0

print 0
sub 1 ax
jle 7 ax 0
jmp 3
```

- ex2.asm – let’s go through it

Example – Autograder Testcase

Example: ax = 8



```
0 read ax
1 jle 7 ax 0
2
3 print 0
4 sub 1 ax
5 jle 7 ax 0
6 jmp 3
7
```

0.

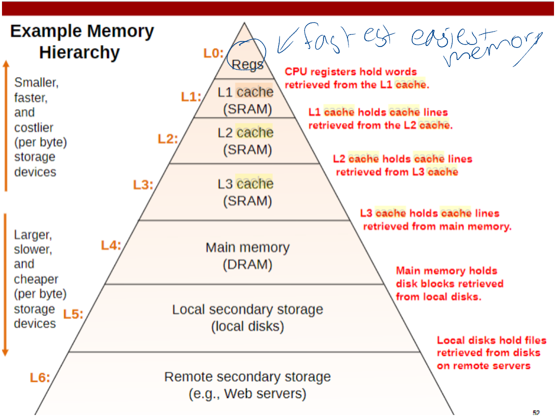
1. Move to line 7 if ax <= 0
2. Just continue
3. Print “0”
4. ax = 7
5. Move to line 7 if ax <= 0
6. Go back to 3

Once you reach line 7, just exit

Next topic:

CACHES

Memory



Example Memory Hierarchy

Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices

L0: Regs

L1: L1 cache (SRAM)

L2: L2 cache (SRAM)

L3: L3 cache (SRAM)

L4: Main memory (DRAM)

L5: Local secondary storage (local disks)

L6: Remote secondary storage (e.g., Web servers)

fastest easiest memory

CPU registers hold words retrieved from the L1 cache.

L1 cache holds cache lines retrieved from the L2 cache.

L2 cache holds cache lines retrieved from L3 cache

L3 cache holds cache lines retrieved from main memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote servers

52

Memory

- **Random access memory (RAM)**
  - Random access: information can be accessed without accessing the memory before
  - **Static** RAM:
    - Does not need refreshers (for data maintenance)
      - Will retain data as long as power is supplied; will lose data when power is removed
    - Faster – small access time
    - Expensive
  - **Dynamic** RAM:
    - **Main memory**
    - Many refreshes needed
      - Expensive
    - Slower – large access time
    - Less expensive

retain what is stored

Caches

- What is a cache?
  - Subset of memory storage that stores data in SRAM
  - Allows for faster retrieval of data
    - Reduces need to access deeper levels of memory
    - However it is more expensive
- **Caching** – storing data about previous accesses to make future retrievals faster (makes predictions)
  - If you access data from address X recently, it will store it in the cache in case that you want to access it again in future – don't have to go to deeper in storage again (since that increases access time)
  - **Cache hit** – requested data was found within cache
  - **Cache miss** – requested data not found; need to look deeper to find it

Levels

- L1
  - Fastest access
  - Highest priority
- L2
  - Bigger in size
  - CPU will access this level next
- L3
  - Slowest access
  - Biggest size

Q&A

- Will continue next week with more info about caches