More Cache Stuff
AND
Languages and
Regex and
DFAs and NFAs

CS230 Tutorial 08
PSA about mental health

So in light of today’s suicide on campus, please take care of yourselves. There are always options, and don’t feel ashamed to seek help if you need it. You’re all worth it.

Here’s a list of available resources both on campus and in general:

UWMates: https://uwaterloo.ca/campus-wellness/counselling-services/uw-mates-peer-support
Counselling services: https://uwaterloo.ca/campus-wellness/counselling-services
Group therapy: https://uwaterloo.ca/campus-wellness/counselling-services/group-therapy

24/7 help lines:
Here 24/7: 1-844-437-3247
Good2Talk: 1-866-925-5454
TTY: 1-877-688-5501

This is a post on the UW reddit describing the various services:
https://www.reddit.com/r/uwaterloo/comments/827tw0/mental_health_resources_at_uw/
Multi-Level Cache

- Recall our memory hierarchy
  - Registers -> Cache (SRAM) -> Main Memory (DRAM) -> SSD/Hard drive
  - Closer to CPU is smaller and more expensive
  - End up with bug cliffs in speed

- What if we add more layers
  - L1 and L2 caches
    - L1 also called primary cache
  - Still big cliff to memory
  - More stuff closer to CPU
  - Faster on average
Multi-Level Cache: Calculating CPI

When a memory access is attempted, what are the possible results?
Multi-Level Cache: Calculating CPI

We have three possibilities, find the formula for each:

- **L1 Cache Hit Cost**
  - This is considered part of normal execution. CPI cost is zero.

- **L2 Cache Hit Cost**
  - We missed L1 cache but found the data in the L2 cache
  - This is $L1\text{-miss\text{-}penalty} \times L1\text{-miss\text{-}chance}$

- **L2 Cache Miss Cost**
  - We missed L1 and L2 cache, have to go all the way to memory
  - This is $\text{main-memory\text{-}access\text{-}time} \times \text{global\text{-}cache\text{-}miss\text{-}chance}$
    - $\text{global\text{-}cache\text{-}miss\text{-}chance}$ could be calculated as
      - $L1\text{-miss\text{-}chance} \times L2\text{-miss\text{-}chance}$

- **CPI Formula is**: $\text{base\text{-}CPI} + L2\text{-cache\text{-}hit\text{-}CPI} + L2\text{-cache\text{-}miss\text{-}CPI}$
Multi-Level Cache: CPI Example

Consider a processor running at 5GHz with a Base-CPI of 2.4. It has a two-level cache. The L1 miss penalty is 2ns. The main memory access time is 60ns.

The L1 cache hit rate is 85%. The L2 cache hit rate is 90%.

Calculate the total CPI of this processor with and without an L2 cache.
Multi-Level Cache: CPI Example Solution

A 5GHz processor has a cycle time of 0.2ns.
L2 cache hit CPI = 2ns * 0.15 = 0.3ns = 1.5 cycles
L2 cache miss CPI = 60ns * 0.15 * 0.10 = 0.9ns = 4.5 cycles
So the effective CPI of this processor is: 2.4 + 1.5 + 4.5 = 8.4 CPI

If there was no L2 cache this would be:
L1 cache miss = 60ns * 0.15 = 9ns = 45 cycles
So the effective CPI of this processor would have been: 2.4 + 45 = 47.4 CPI

The speedup is therefore: 47.4/8.4 = ~5.6 times faster!
Input/Output (I/O)

- The only result of a program that actually matters is I/O
  - I/O can be on files, on network sockets, on the screen, on USB devices, on headphones, etc…

- How do we actually do I/O? Two methods:
  - Memory-Mapped I/O
    - A special set of memory addresses is reserved for I/O
    - When you do a \texttt{lw/sw} on those addresses, instead of memory the data goes to the device
    - This is how our MIPS programs worked: \texttt{0xFFFF000C} was a memory-mapped I/O address
  - I/O Instructions
    - Specific instructions are used to do I/O
I/O Workings

● How do we react to input?
  ○ When a user presses a key or moves the mouse etc. how does the electrical signal in the USB connection tell the CPU that something happened?
  ○ How does the computer know when a piece of a file arrived over the network?

● Two methods:
  ○ Polling
    ■ The CPU continuously asks the device: “anything ready yet” over and over again quickly
    ■ When there is something ready the device responds
  ○ Interrupts
    ■ Special signal connected directly to CPU
    ■ When the signal is set, the CPU pauses what it was doing to handle the data
I/O Busses

How is the data actually transmitted? The Bus! A set of wires between all the devices and the CPU.

- **Serial vs. Parallel**
  - A serial bus has few wires and sends data one bit at a time
  - A parallel bus has many wires and sends many bits at once

- **Synchronous vs. Asynchronous**
  - Synchronous bus shares a clock with all devices
    - Protocol determines who sends data on what tick
    - Limited number of devices because of clock skew
  - Asynchronous bus has handshake setup
    - Example: CPU request, device acknowledges it, device response, CPU acknowledges
    - No clock sharing needed
    - More devices can fit in one bus
Compilers

What is a compiler? Basically a **translator**.

- You know the computer reads binary
- You write code
- How does the code turn into the corresponding binary?
- .... this is where the compiler comes in

A compiler serves to translate a higher level language (like C, C++, etc) into the corresponding binary file.
Stages of Compilation

In general, the steps the compiler follows to translate your code into something the computer can actually run go as follows (in this order):

1. **Scanning/Lexing** -- read your code and turn it into *tokens* (this is basically an intermediate representation of your code, and more abstract than your code -- tokens are in terms of the language rules)

2. **Parsing/Syntax analysis** -- go through the tokens and make sure they follow the rules of the language (does your code make a correct program?)

3. **Semantic Analysis** -- more information from your code (variable names, types, etc) that is needed to actually run the program

4. **Code Generation** -- this is where you produce the output language (we were talking about binary before, but this could be another language)
More info on Tokens

Tokens are what are produced by the Scanning/Lexing phase of compilation.

What are tokens? Basically think symbolic representation of different parts of the program. They match to the rules of the programming language.

- keywords (think MIPS: add, addi, jr, etc)
- operators (think math: +, -, *, =, etc)
- delimiters (how do we divide parts of the program? usually spaces, tabs, new lines, etc)
- and others
Regular Languages

There are various types of formal languages… you can wikipedia them if you want to, but the category we care about here is the regular languages.

All formal languages are represented by the following:

- an alphabet of symbols that are used to make up the “words” of the language (this just means a list of the smallest building blocks in a language, that you use to make up every program)
- a list of rules that says how the symbols in the alphabet are allowed to combine (basically, the syntax rules of your language)

Regular languages can be represented in various ways that we will go over now: DFAs, NFAs, and Regular Expressions
Regular Expressions (Regex)

This way of representing languages uses **metacharacters** - these are just characters that mean something specific and aren’t just ASCII hanging out.

Here are some of the most important ones to know:

- **|** -- OR
  - example: a | b -- this represents a OR b
- **()** -- grouping, basically like regular parentheses in math
  - example: m(i | a)lk -- this represents ‘milk’ OR ‘malk’
- **[]** -- matches one character in the list specified in the brackets
  - example: [abc] -- this represents a OR b OR c
  - can use - to specify a range in the brackets, i.e. [a-c] is the same as [abc]
- **^** -- NOT
- **.** -- any ONE character
  - example: a.c -- this represents a[every ascii character]c
More metacharacters...

● ?  -- the previous character 0 or 1 time
  ○ example: aa?  -- this means a OR aa
● +  -- the previous character 1 or more times
  ○ example: a+  -- this matches a OR aa OR aaa OR aaaa OR ….. infinite a’s
● *  -- the previous character 0 or more times
  ○ example: a*  -- this matches epsilon OR a OR aa OR aaa OR …… infinite a’s
● \  -- e s c a p e: you use this when you want to treat one of the metacharacters as a normal ASCII character
  ○ example: a\*  -- this means a* literally, not aa or aaa or …
  ○ example: \  -- this means the backslash character
Regex can be used to represent regular languages

Remember, a language is just a bunch of rules saying what you can represent with a particular character set.

Example: Represent the language of any combination of the letters a and b.

Solution:

Example: Represent the language of any number of a’s, followed by an even number of c’s.

Solution:

Example: Describe the language represented by: a+[0-9]*xy?

Solution:
Regex can be used to represent regular languages

Remember, a language is just a bunch of rules saying what you can represent with a particular character set.

Example: Represent the language of any combination of the letters a and b.

   **Solution:** (a | b)*

Example: Represent the language of any number of a’s, followed by an even number of c’s.

   **Solution:** a*(cc)*

Example: Describe the language represented by: “a+[0-9]*xy?”

   **Solution:** 1 or more a’s, followed by any number of digits (remember, digit is any number from 0-9), followed by x, and then 0 or 1 y.
Deterministic Finite Automata (DFAs)

You know now that you can represent regular languages with regex. You can also represent them with a DFA.

Note: DFA can also be referred to as DFSM (deterministic finite state machine).

Don’t be scared off by the acronym! DFA is basically just a flow chart for the language.

The idea is that you can take a string (i.e. a list of characters) and check if it is “accepted” by (i.e. fits in) the language by seeing if you can follow along the flow of the DFA and reach the end state by the end of the string.
DFA Examples (same as regex examples)

Any language you can represent with regex, you can also represent with an equivalent DFA.

Example: Draw a DFA for the language of any combination of the letters a and b.

Solution:

Example: Draw a DFA for the language of any number of a’s, followed by an even number of c’s.

Solution:

Example: Draw a DFA for the language represented by: “a+[0-9]*xy?”

Solution:
DFA Solutions

Example: Draw a DFA for the language of any combination of the letters a and b.

**Solution:** Remember this is \((a \mid b)^*\)
Example: Draw a DFA for the language of any number of a’s, followed by an even number of c’s.

Solution: Remember this is $a^*(cc)^*$
DFA Solutions

Example: Draw a DFA for the language represented by: “a+[0-9]*xy?”

Solution:

```
start
q0  a  q1
   0-9

q1  a  q3
   0-9

q3

q4  x  y  q5
```

23
Non-Deterministic Finite Automata (NFAs)

What is the difference between a DFA and an NFA?

Look at the definition of “deterministic”: with DFAs, there is no ambiguity at all, and only one possible path between states for a given string. With NFAs, there are situations where there could be more than one choice for a move from state to state while parsing a string.

In detail, NFAs can have (which DFAs do not have):

- epsilon transitions (i.e. state transitions on the empty character)
- multiple transitions on the same state for a particular character (for example, on one state, there could be 2 arrows out on the same character)

NFAs can always be translated to an equivalent DFA.
Let’s look at one of the examples we’ve been looking through so far:

Example: Draw a NFA for the language represented by: “a+[0-9]*xy?”

Recall that the DFA is as follows:
NFA solution

Example: Draw a NFA for the language represented by: “a+[0-9]*xy?”

Solution:

Note: there are more than one possible NFA for this problem, and I am purposely showing many NFA features and not going for optimal NFA construction.
Assignment reminders

● Submit a *.txt* XOR a *.pdf* for each question
  ○ Do not submit both for the same question!
  ○ You may submit a *.pdf* for one question and a *.txt* for a different question

● Make sure your diagrams and tables are clear and easy to read
  ○ Make sure to leave enough space

● Assignment solutions for A1-A3 inclusive will be posted simultaneously, after A3 is due!