Overview

CSC 235 - Computer Organization

References

- Slides adapted from CMU

Outline

- Course theme
- Realities of programming

Course Theme

- Systems Knowledge
  - How hardware combine to support the execution of application programs
  - How you as a programmer can best use these resources
- Useful outcomes from taking CSC 235
  - Become more effective programmers
  - Prepare for later systems courses

It is Important to Understand How Things Work

- Most CS courses emphasize abstraction
    - Abstract data types
    - Asymptotic analysis
- These abstractions have limits
    - Especially in the presence of bugs
    - Need to understand the details of underlying implementations
    - Sometimes the abstract interfaces do not provide the level of control or performance you need

Realities of Programming

- `ints` are not integers and `floats` are not reals
- knowing assembly is useful
- memory matters
- there is more to performance than asymptotic complexity
- computers do more than execute programs

`ints` and `floats`

- Example: is $x^2 \geq 0$?
  - `float`: yes
  - `int`: ?
- Example: is $(x + y) + z = x + (y + z)$?
  - unsigned and signed `ints`: yes
  - `floats`: ?

Computer Arithmetic

- Arithmetic operations have important mathematical properties
- But, we cannot assume all the “usual” mathematical properties
  - due to finiteness of representations
  - integer operations satisfy “ring” properties: commutativity, associativity, and distributivity
  - floating point operations satisfy “ordering” properties: monotonicity and values of signs
- Observation
  - Need to understand which abstractions apply in which contexts
  - Import issues for compiler writers and serious application programmers

Knowing Assembly is Useful

- You will probably never write programs in assembly
  - compilers are typically much better and more patient than you are
- But, understanding assembly is key to the machine-level execution model
  - behavior of programs in the presence of bugs
  - tuning program performance
  - implementing system software
  - creating / fighting malware

Memory Matters

- Memory is not unbounded
  - it must be allocated and managed
  - many applications are memory dominated
- Memory referencing bugs are especially pernicious
  - effects are distant in both time and space
- Memory performance is not uniform
  - cache and virtual memory effects can greatly affect program performance
  - adapting program to characteristics of memory system can lead to major speed improvements

Memory Referencing Bug Example

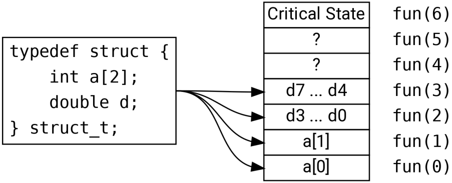
- Code with a bug:

```
typedef struct {
    int a[2];
    double d;
} struct_t;

double fun(int i) {
    volatile struct_t s;
    s.d = 3.14;
    s.a[i] = 1073741824; /* Possibly out of bounds */
    return s.d;
}
```

- What is the result of `fun(6)`?

Memory Referencing Bug Example



```
typedef struct {
    int a[2];
    double d;
} struct_t;
```

Critical State fun(6)

? fun(5)

? fun(4)

d7 ... d4 fun(3)

d3 ... d0 fun(2)

a[1] fun(1)

a[0] fun(0)

Memory Referencing Errors

- C and C++ do not provide any memory protection
  - out of bounds array references
  - invalid pointer values
  - abuses of malloc/free
- Can lead to nasty bugs
  - Whether or not a bug has any effect depends on the system and compiler
- How do we deal with this?
  - program in a memory safe language
  - understand what possible interactions may occur
  - use or develop tools to detect referencing errors

Asymptotic Complexity and Performance

- Constant factors matter
- Exact operation count does not predict performance
  - must optimize at multiple levels: algorithm, data representation, procedures, and loops
- Must understand the system to optimize performance
  - how programs are compiled and executed
  - how to measure program performance and identify bottlenecks
  - how to improve performance without destroying code modularity and generality

Memory System Performance Example

- Slower

```
void cpy(int src[2048][2048], int dst[2048][2048]) {
    for (int j = 0; j < 2048; j++)
        for (int i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```

- Faster

```
void cpy(int src[2048][2048], int dst[2048][2048]) {
    for (int i = 0; i < 2048; j++)
        for (int j = 0; j < 2048; i++)
            dst[i][j] = src[i][j];
}
```

Computers do more than execute programs

- They need to get data in and out
  - I/O system critical to program reliability and performance
- They communicate with each other with each other over networks
  - Many system-level issues arise in the presence of a network
    - concurrent operations by autonomous processes
    - coping with unreliable media
    - cross platform compatibility
    - complex performance issues

Course Perspective

- This course is programmer-centric
    - By knowing more about the underlying system, you can be more effective as a programmer
    - Enable you to write programs that are more reliable and efficient