CS2000, Introduction to Computer Systems (6-6)

Textbook

Learning Objectives – Introduction to Computer Systems:

1. Be able to integrate foundational principles of computer science, including computer architecture, algorithms, compilers, operating systems, programming languages, and software engineering, by constructing a general-purpose computer system from the ground up.

2. Be able to balance tradeoffs between key design constraints.

3. Be able to write fast programs by understanding what makes computers go fast, how processors are designed, what is inside a modern processor, and the interface between software and physical devices.

4. Be able to characterize how elements of a general-purpose processor and its associated memory hierarchy are organized to execute instructions and perform functions.

5. Be able to describe and explain the dynamics of procedure invocation and evaluation.

6. Be able to describe and explain how program components are linked (bound together) at run time.

7. Be able to apply the principle of abstraction to solve problems in the design of computer systems.

8. Be able to apply the von Neumann Model to the construction of a general-purpose computer system.

9. Be able to apply Boolean logic to the design of digital circuits.

10. Be able to construct complex systems from primitive digital circuits.

11. Be able to apply the concepts of cache memory, virtual memory, memory segmentation, paging, and address translation to the construction of a general-purpose computer system

A weekly schedule is on the next page.
<table>
<thead>
<tr>
<th>Week</th>
<th>Topic</th>
<th>Subtopics</th>
</tr>
</thead>
</table>
| 1    | How machines compute                      | Demonstration of system to be built, abstraction/implementation paradigm, cost metrics, historical development, models of program execution, Moore’s Law  
*Project 1: Elementary Logic Gates* |
| 2    | Data representation                        | Conversion between bases, signed integer representation, floating-point representation, character codes, error detection and correction  
*Project 2: Combinational Chips*    |
| 3    | Basic logic circuits                       | Boolean function evaluation, Boolean algebra for circuit descriptions, logic gates  
*Project 3: Sequential Chips*       |
| 4    | Hardware organization                      | Instruction decoding, functional units, clock strategies, race conditions, interrupts  
*Project 4: Machine Language Programming* |
| 5    | Basic machines and instruction execution   | Instruction set, basic CPU operations, assembly programming, register and stack organization, fetch-decode-execute cycle  
*Project 5: Computer Architecture* |
| 6    | A closer look at instruction set architectures (ISAs) | Instruction formats, instruction types, addressing, real-world examples of ISAs  
*Project 6: The Assembler*          |
| 7    | Memory                                    | Caches, associativity, locality principle, memory hierarchy, virtual memory  
*Project 6: The Assembler*          |
| 8    | Execution dynamics                         | Managing storage, stack, procedure call and return, read-only instruction code |
| 9    | Speed-up                                  | Pipelining, branch prediction, prefetching, out-of-order execution, superscalar processors  
*Project 9: High Level Programming* |
| 10   | Input/Output and Storage Systems           | I/O and performance, Amdahl’s Law, I/O architectures, data transmission modes, storage hierarchy, file systems |
| 11   | Linking and loading                        | Creating executable files from separately compiled components, single-job vs. concurrent operations, services and facilities, organization and types of operating systems, tradeoffs of different OSes, bootstrapping |