University of Regina

Department of Computer science

CS 201

Introduction to Digital Systems

Course Information Booklet

Created: December 12th, 2011; Dr. Malek Mouhoub

Updated:

November 2014 Dr. Malek Mouhoub

August 2016 Dr. David Gerhard

Calendar Description

Hardware paradigms, logic minimization, sequential and combinational circuits, register transfer notation. Numerical data representation, number bases, floating-point and two's-complement representation, representation of non-numeric data, records and arrays. Von Neumann architecture, control units, instruction sets, assembly language programming, addressing modes, subroutines, basic building blocks, computer components.

Pre-requisite(s):

CS 110

Co-requisite(s): None

Textbook(s):

LOGIC AND COMPUTER DESIGN FUNDAMENTALS
Author: MANO
ISBN: 978-0133760637
Publisher: Prentice Hall
Edition: 5

MIPS ASSEMBLY LANGUAGE PROGRAMMING
Author: BRITTON
ISBN: 978-0131420441
Publisher: Pearson
Edition: 1

Other References:

COMPUTER ARCHITECTURE FROM MICROPROCESSORS TO SUPERCOMPUTERS
Author: PARHAMI
ISBN: 978-0195154559
Publisher: Oxford

Lecture Outline and Schedule

Topics	Description	Lecture Hours
Boolean Logic	Boolean Algebra, Logic Functions Representations, Truth Tables, Logic Gates, Proving the Equivalence of Logic Expressions, Simplification of Logic Expressions, Physical Considerations (gate delays), Canonical and Standard Forms, Minimization through K-maps, Universal Gates (NAND and NOR), XOR and XNOR, Single Error Detection using the Parity Bit.	7
Combinatorial Circuits	Design Procedure, Arithmetic Circuits, Half and Full Adders, Decoders and Encoders, Multiplexers and Demultiplexers, Read Only Memory (ROM), Programmable Logic Array (PLA).	3
Sequential Circuits	SR and S'R' Latches, D Latches, JK flip-flop, Master Slave flip-flop, Sequential Circuit Design, State Table, State Diagram, Registers and Counters.	3
Computer Arithmetic	Bits, Bytes and Words, Unsigned and Signed Number Representations and Number Bases, Sign and Magnitude, 1's and 2's Complement, Addition, Subtraction, Overflow Detection, Pencil and Paper Approach for Multiplication and Division, Hardware Approach for Multiplication and Division, Booth Algorithm.	4
Introduction to MIPS and SPIM	The von Neumann machine, Mini MIPS Instruction Set Architecture, Instruction Fetch, Decode and Execution, Introduction to SPIM, Assembly/Machine Language Programming, Instruction formats, MIPS Program Structure and Statements, System I/O Services, Arithmetic and Logic Instructions, Load and Store Instructions, Jump and Branch Instructions, Addressing Modes.	8
MIPS Procedures and Data	Memory and Processing Subsystems, Multiplication and Division, Logic and Arithmetic Shifts, Simple Procedure Calls, Nested Procedure Calls, Stacks and Recursion, Memory Map, Data Types, Representation of Nonnumeric Data (character codes), Loading and Storing Bytes, Arrays and Pointers, Fixed- and Floating-point Systems.	9

Lab Outline and Schedule

Topics	Description	Lab hours
Introduction to LogicWorks 5	Learn how to build circuits in LogicWorks	2
Combinational Circuits	Practice on Combinational design proceddures	2
Adders and Subtractors	Practice on basic arithmetic circuitry	2
Sequential Design	Practice on building sequential circuits	2
Sequential Adders & Subtractors	Further practice on sequential designs and prepare for building a multiplier circuit	2
Multiplier Circuit	Build a integer multiplier	2
Introduction to SPIM (with basic I/O)	Introduce PCSpim and System Calls students to write MIPS assembly programs	2
Arithmetic Operations and Debugging	Practice on arithmetic operations	2
SPIM Control Flow Structures	Looping and Selection Control Structures	2
SPIM Addressing Modes and Arrays	Review SPIM addressing modes and practice on arrays	2
SPIM Procedure Calls	Practice on SPIM procedure calls	2
Counting Vowels	Run MIPS Assembly Program on Real Hardware	2

Core Hours (Topics in b**old** appear in CS 201, some are repeated/expanded in CS 301; topics in *italics* appear in CS 301)

AR. Architecture and Organization:

AR/Digital Logic and Digital Systems (**3** (of 3) Core-Tier2 hours)

- **Overview and history of computer architecture**
- **Combinational vs. sequential logic/Field programmable gate arrays as a fundamental combinational + sequential logic building block**
- **Multiple representations/layers of interpretation (hardware is just another layer)**
- **Computer-aided design tools that process hardware and architectural representations**
- **Register transfer notation/Hardware Description Language (Verilog/VHDL)**
- **Physical constraints (gate delays, fan-in, fan-out, energy/power)**

AR/Machine Level Representation of Data (**3** (of 3) Core-Tier2 hours)

- **Bits, bytes, and words**
- **Numeric data representation and number bases**
- **Fixed- and floating-point systems**
- **Signed and twos-complement representations**
- **Representation of non-numeric data (character codes, graphical data)**
- **Representation of records and arrays**

AR/Assembly Level Machine Organization (**3** of 6 Core-Tier2 hours)

- **Basic organization of the von Neumann machine**
- **Control unit; instruction fetch, decode, and execution**
- **Instruction sets and types (data manipulation, control, I/O)**
- **Assembly/machine language programming**
- **Instruction formats**
- **Addressing modes**
- **Subroutine call and return mechanisms**
- *I/O and interrupts*
- *Heap vs. Static vs. Stack vs. Code segments*
- *Shared memory multiprocessors/multicore organization*
- *Introduction to SIMD vs. MIMD and the Flynn Taxonomy*

SF. Systems Fundamentals:

SF/Computational Paradigms (**1** of 3 Core-Tier1 hours)

- **Basic building blocks and components of a computer (gates, flip-flops, registers, interconnections; Datapath + Control + Memory)**
- **Hardware as a computational paradigm: Fundamental logic building blocks; Logic expressions, minimization, sum of product forms**
- *Application-level sequential processing: single thread*
- *Simple application-level parallel processing: request level (web services/client-server/ distributed), single thread per server, multiple threads with multiple servers*
- *Basic concept of pipelining, overlapped processing stages*
- *Basic concept of scaling: going faster vs. handling larger problems*

SF/State and State Machines (**4** of 6 Core-Tier1 hours)

- **Digital vs. Analog/Discrete vs. Continuous Systems**
- **Simple logic gates, logical expressions, Boolean logic simplification**
- **Clocks, State, Sequencing**
- **Combinational Logic, Sequential Logic, Registers, Memories**
- *Computers and Network Protocols as examples of state machines*
    - *a) Computers as examples of state machines*
    - *b) Network Protocols as examples of state machines*

Learning Outcomes (refer to core hours for topics list)

AR. Architecture and Organization:

AR/Digital Logic and Digital Systems (**3** (of 3) Core-Tier2 hours)

1. Describe the progression of computer technology components from vacuum tubes to VLSI, from mainframe computer architectures to the organization of warehouse-scale computers. [Familiarity]
2. Comprehend the trend of modern computer architectures towards multi-core and that parallelism is inherent in all hardware systems. [Familiarity]
3. Explain the implications of the "power wall" in terms of further processor performance improvements and the drive towards harnessing parallelism. [Familiarity]
4. Articulate that there are many equivalent representations of computer functionality, including logical expressions and gates, and be able to use mathematical expressions to describe the functions of simple combinational and sequential circuits. [Familiarity]
5. Design the basic building blocks of a computer: arithmetic-logic unit (gate-level), registers (gate-level), central processing unit (register transfer-level), memory (register transfer-level). [Usage]
6. Use CAD tools for capture, synthesis, and simulation to evaluate simple building blocks (e.g., arithmeticlogic unit, registers, movement between registers) of a simple computer design. [Usage]
7. Evaluate the functional and timing diagram behavior of a simple processor implemented at the logic circuit level. [Assessment]
8. Identify the data components and behaviors of multiple abstract data types. [Usage]
9. Implement a coherent abstract data type, with loose coupling between components and behaviors. [Usage]
10. Identify the relative strengths and weaknesses among multiple designs or implementations for a problem. [Assessment]

AR/Machine Level Representation of Data (**3** (of 3) Core-Tier2 hours)

1. Explain why everything is data, including instructions, in computers. [Familiarity]
2. Explain the reasons for using alternative formats to represent numerical data. [Familiarity]
3. Describe how negative integers are stored in sign-magnitude and twos-complement representations. [Familiarity]
4. Explain how fixed-length number representations affect accuracy and precision. [Familiarity]
5. Describe the internal representation of non-numeric data, such as characters, strings, records, and arrays. [Familiarity]
6. Convert numerical data from one format to another. [Usage]
7. Write simple programs at the assembly/machine level for string processing and manipulation. [Usage]

AR/Assembly Level Machine Organization (**3** of 6 Core-Tier2 hours)

1. **Explain the organization of the classical von Neumann machine and its major functional units. [Familiarity]**
2. **Describe how an instruction is executed in a classical von Neumann machine, with extensions for threads, multiprocessor synchronization, and SIMD execution. [Familiarity]**
3. *Describe instruction level parallelism and hazards, and how they are managed in typical processor pipelines. [Familiarity]*
4. *Summarize how instructions are represented at both the machine level and in the context of a symbolic assembler. [Familiarity]*

5. **Demonstrate how to map between high-level language patterns into assembly/ machine language notations. [Familiarity]**
6. **Explain different instruction formats, such as addresses per instruction and variable length vs. fixed length formats. [Familiarity]**
7. **Explain how subroutine calls are handled at the assembly level. [Familiarity]**
8. *Explain the basic concepts of interrupts and I/O operations. [Familiarity]*
9. **Write simple assembly language program segments. [Usage]**
10. **Show how fundamental high-level programming constructs are implemented at the machine-language level. [Usage]**

SF. Systems Fundamentals:

SF/Computational Paradigms (**1** of 3 Core-Tier1 hours)

1. **List commonly encountered patterns of how computations are organized. [Familiarity]**
2. **Describe the basic building blocks of computers and their role in the historical development of computer architecture. [Familiarity]**
3. *Articulate the differences between single thread vs. multiple thread, single server vs. multiple server models, motivated by real world examples (e.g., cooking recipes, lines for multiple teller machines and couples shopping for food). [Familiarity]*
4. *Articulate the concept of strong vs. weak scaling, i.e., how performance is affected by scale of problem vs. scale of resources to solve the problem. This can be motivated by the simple, real-world examples. [Familiarity]*
5. **Design a simple logic circuit using the fundamental building blocks of logic design. [Usage]**
6. **Use tools for capture, synthesis, and simulation to evaluate a logic design. [Usage]**
7. *Write a simple sequential problem and a simple parallel version of the same program. [Usage]*
8. *Evaluate performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved. [Assessment]*

SF/State and State Machines (**4** of 6 Core-Tier1 hours)

1. **Describe computations as a system characterized by a known set of configurations with transitions from one unique configuration (state) to another (state). [Familiarity]**
2. **Describe the distinction between systems whose output is only a function of their input (Combinational) and those with memory/history (Sequential). [Familiarity]**
3. **Describe a computer as a state machine that interprets machine instructions. [Familiarity]**
4. *Explain how a program or network protocol can also be expressed as a state machine, and that alternative representations for the same computation can exist. [Familiarity]*
5. *Develop state machine descriptions for simple problem statement solutions (e.g., traffic light sequencing, pattern recognizers). [Usage]*
6. **Derive time-series behaviour of a state machine from its state machine representation. [Assessment]**

Grading Scheme

	How Many?	Grade Each	Weight	Comments
Assignments	6	40 pts	20%	All assignments are submitted and graded through URCourses
Labs	12	variable pts	20%	12 labs
Quizzes	N/A	N/A	N/A	
Midterm exams	1	20 pts	20%	midterm exam takes one full lecture slot
Final exam	1	45 pts	40%	3 hour exam
Projects	N/A	N/A	N/A	
Presentations	N/A	N/A	N/A	

- To pass the course you must pass the final exam.
- Up to ±5\% instructor discretion may be assigned based on in-class discussion, online forums and general attitude.
- Students are expected to keep up-to-date with all material posted online
- Students are expected to be familiar with University of Regina policies regarding academic integrity.