
CDA3101 - Spring 2016 - Exam #1 Review

TOPICS TO BE COVERED ON EXAM 1: (Study these in Textbook and Web Notes)

1. Introductory Material, Computer Abstractions, and Technology
    - 1.1. Introduction and Overview
    - 1.2. Overview of Computer Abstractions
    - 1.3. Overview of Computer Technology Trends
    - 1.4. Digital Logic Design
    - 1.5. Computer Performance Assessment
    - 1.6. Performance Benchmarking

2. ISA, Machine Language, and Number Systems
    - 2.1. ISA and Machine Language
    - 2.2. Instruction Representation
    - 2.3. Decision Instructions and Procedure Support
    - 2.4. Number Representations, Data Types, and Addressing
    - 2.5. MIPS Programs
    - 2.6. Pointers and Arrays

3. Computer Arithmetic
    - 3.1. Arithmetic and Logic Operations
    - 3.2. Arithmetic Logic Units and the MIPS ALU
    - 3.3. Boolean Multiplication and Division

>>> There will be NO LENGTHY MIPS PROGRAMS TO CODE <<<

TYPES OF QUESTIONS:

- Short Answer (1-2 sentences - no dissertations, please)

- Programming (write or debug a MIPS program, given its C equivalent)

- Analysis (calculate the performance of a MIPS program given IC & CPI for each instruction type, and CPUE cycle time)

* Due to student requests from last semester, the exams this semester will have no problems where you have to do matching of questions and predetermined answers.

EXAMPLE QUESTIONS: (answers in blue)

* Why is RISC important, and how is it different from CISC?

RISC uses a small instruction set and very fast, relatively simple hardware to achieve low CPI and low cycle time at the expense of larger IC. CISC uses a large instruction set with many irregular instruction formats and complex hardware to achieve reduced IC, at the expense of large CPI and moderate cycle time. CISC is hardware-intensive, which means higher cost as opposed to RISC, which is software-intensive (compiler translates HLL to a small instruction set).

* What is a compiler, and what does it do?

A compiler is a computer program that translates source code (usually, a HLL) to assembly code or object code.

* What is the von Neumann bottleneck, and what MIPS instructions does it most adversely impact?

The VNB refers to I/O-limited processor speed resulting from insufficient bandwidth between the memory and register file (register-memory architecture) or between the memory and ALU (memory-memory architecture). The VNB most adversely affects those instructions that transfer data to and from memory. In MIPS, these are load/store instructions.

* What are the important technology trends in computer hardware that we discussed in class?

The steady increase in gate density per unit chip area, the increased capacity of memory, and the increased performance (in MFLOPS) of processors. These trends have driven application development, which has in turn motivated development of faster architectures.

* How has RISC design philosophy impacted these technology trends, as well as processor performance?

Faster RISC architectures and faster processors generally depend on faster register technology. Thus, RISC development focused on faster memory devices (e.g., D flip-flops) which results from faster gate technology. This is achieved in several ways: (1) reduced clock cycle time, (2) reduced voltage for lower power consumption and reduced heat dissipation, as well as (3) increased gate density for higher-capacity memory devices.

* What are the implications of Ahmdahl's Law for processor performance optimization?

Ahmdal's Law pertains to diminishing returns from recursive optimization of a processor architecture. This means that, if you're going to optimize hardware, you need to get the highest possible increase in performance the first time you optimize.

* What is Moore's Law, and why is it important? What is the likelihood that it will hold in the future? Explain in detail...

Moore's Law says that the number of transistors per chip has approximately doubled every 1.5 years since the early 1970s. A discussion of its implications, and future constraints, is found in Section 1.3.2 of the Web notes.

* What are the MIPS design principles we have learned thus far? Give an example of how each one is implemented in hardware.

#1: Keep it Simple and Stupid (KISS):

MIPS has 32 32-bit registers, simple ALU and hardware

#2: Simplicity favors regularity:

MIPS has all instructions with 32-bit format, just three instruction types (R, I, J)

#3: Simpler is faster

Simpler ISA design (e.g., RISC instead of CISC) means less hardware required to decode and execute instructions, since the instruction format is simpler and more regular, and there are fewer types of instructions.

#4: Make the common case as fast as you can:

MIPS designers noted that addition and multiplication by small constants occurred frequently in HLL programs. So, they put small constants in registers, and dedicated a register to the value zero ($0).

MIPS designers observed (through analysis of large bodies of source code and exeuction profiling results) that branches are more often not taken than taken. Thus, they compute the branch target address concurrently with evaluating the branch condition. If the branch is taken, then the branch target address is immediately avaiable, and you don't have to waste a few cycles computing it.

Amdahl's Law provides penalties of diminishing returns if you don't make the most frequently occurring case run as fast as possible. Many designers make the mistake of ignoring Amdahl's Law, by supposing that they can make a small part of a program that is not used very often run very fast, and that will make the entire architecture run faster. This is a common mistake early in one's career, and can cost you dearly if your job depends upon improving the overall performance of a processor or system to make it go as fast as possible.

Other examples are given in Section 2 of the Web notes.

* Know how to solve Amdahl's Law problems given (a) the system speedup, and (b) fraction of system enhanced by a speedup, to yield the required speedup for the fraction of the system enhanced (accelerated). See Section 1.5 of Web notes.

* How is RISC design philosophy related to the MIPS architecture design principles that you have learned thus far?

RISC is based on the KISS design principle - a simple instruction set with simpler hardware than CISC. RISC machines try to make the common case fast, using fast

hardware with lots of fast, general-purpose registers to speed data access. RISC attempts to have fast memory-register-memory transfer, to reduce the adverse impact of the von Neumann bottleneck. RISC also tries to have regular instruction formats, with the instructions all the same size (simplicity favors regularity).

* What is an ISA and why is it important in processor design?

An instruction set architecture is the specification that links the hardware structure and function to that of the software. ISAs are important because they clarify processor design and provide a convenient abstraction for hardware/ software interface design, analysis, and maintenance.

* Know how to solve the types of problems we did in class and recitation re: CPUtime = IC x CPI x CycleTime. This is *very* important. Hint: You might be asked to compute the runtime of one or more MIPS programs that you write, so practice on the exercises we used for Homework #1 and #2.

* Also know how to compare the performance of two processors, given CPI and CycleTime: justr use the performance equation:

CPUtime = IC x CPI x CycleTime,

and choose an arbitrary IC that is the same for both machines.

For example, if M1 has CPI = 1.3 and CycleTime = 0.5 nsec, and M2 has CPI = 1.5 and CycleTime = 0.4 nsec. Which is faster?

Solution: Letting IC = 1,000 (for convenience), we have

M1: CPUtime = 1,000 x 1.3 x 0.5 nanosec = 0.65 microsec

M2: CPUtime = 1,000 z 1.5 x 0.4 nanosec = 0.60 microsec

M2 takes less time to run the hypothetical test program, so M2 is faster.

* What are the three MIPS instruction formats?

Answer: R-format, for arithmetic and logic operations
I-format, for load/store and conditional branching,
also operations with constants in the immediate field
J-format, for unconditional branching

Know: MIPS R-format, I-format, and J-format instruction layouts
What each field in each MIPS instruction format is for
What a BEQ instruction is for, and how it works

* How are MIPS programs produced, and what type of code is involved?

Answer: MIPS programs are produced in assembly language by (a) a programmer using an editor, or (b) by a **compiler** translating high-level programming language (HLL) to assembly. An **assembler** translates the assembly language to object code, which is combined with libraries by a **linker**, to produce machine code. A **loader** or **runtime module** puts the machine code into memory and saves the start address A so execution can begin by loading A into the PC.

* What is the difference between assembly and object code?

Answer: Assembly code is written with pseudo-names for the registers, text labels, and has the result register first in the list of operands. The assembler resolves dependencies, codes the register addresses in terms of numbers, and translates the labels to addresses. If libraries are called, the dependencies are resolved by the linker.

* What is a pointer, and what is it used for?

Answer: A pointer is an address that it is used to reference data directly in memory.

* How is pointer arithmetic used in programming practice?

Answer: By incrementing or decrementing pointers, you can efficiently access large amounts of data without actually having to handle the parts of memory that store the data. For example, instead of passing large arrays in the argument list of a function (call-by-value), we pass the pointers to the arrays (call-by-reference).

* How do load/store operations in MIPS relate to pointer arithmetic?

Answer: If x is a variable and p is a pointer, and you have the C-like statement "p = &x", this is like fetching the address of a memory element in MIPS. If you have "x = *p", that is like a load operation in MIPS, because it says "the variable x contains the data referenced by pointer (address) p." Conversely, if you have "*p = x", that is like a store operation, since it means "put the contents of x into memory at address p."

* Know how to perform Boolean multiplication using Booth's Algorithm, and know how to perform Boolean UNSIGNED division using the Restoring Division Algorithm (both discussed in class and recitation). Problems will be limited to 4-5 digits per operand, so as not to cause busywork.

-EOF-