# CS M151B Nevin Liang

## Lecture 1; Thursday, October 1, 2020

**Inside Thoughts and Notes:** This class is p cool ngl everyone here is white for some reason. I thought there wuda been more Asians. This is kinda weird.

Text

Description automatically generated

Look at this holy shit a 75% is an A- how hard is this class wtaf. No midterms lol but 6 quizzes. I think I prefer that yea. Final is only 30% I think it was more for 33. Hw is 20% which is bless bc it was 5% for other classes.

We’re doing the outline of what the difference btwn ISA (instruction set arch aka cs33 stuff) and Machine Org (all the cs m51a stuff I learned). We’re all just waiting here bc the teachers having technical difficulties lmao. At least this teacher is really passionate u can see. At least compared to my 33 teacher I think. Ppl say his class is rly bad tho riprip. He’s like 1-2 stars on bruinwalk or smth. It’s the first day tho so everything is just super high level. 2 straight hrs of hardcore learning is gonna b rough lol.

Wats rly good is that I can hear and understand him hes a lot better than the 33 teacher in that regard. Even though they have a similar accent. He just accents his words rly clearly.

I love this class the way he explains things is so good oml. Its been 1 hr and im not even fazed as opposed to cs33 or something. Everything here is so interesting this is what im interested in.

Cool stats: Processor speed: 60% a yr, but Memory (DRAM) speed: 7% a yr (capacity doubles every 1.5 yrs tho so that’s cool).

**Class Material Notes:**

* ISA examples: RISC, MIPS, x86, (RISC-V = most recent one and is still living, and the one we are dealing with in this course)
* MIPS R3000 ISA -> 6 instruction categories:
  + load/store, computational, jump and branch, floating point (coprocessor), memory management, special
  + Also defines the registers available, R0-R31, PC, HI, LO
  + Format of instructions in terms of bits: OP, rs, rt, rd, sa, funct (32 bits total)
    - OP, rs, rt, immediate (another example)
  + This is how the hardware manages because the hardware needs a solid backbone for interpreting instructions
* 1 ISA can have multiple types of machine organizations: machine organization is how the diagrams are set up:
  + Diagram

    Description automatically generated
  + This is for Pentium 4 I think
* High Level View of. Computer
  + Datapath + Control make up the Processor:
  + Memory stores all data
  + Input+Output are needed for connection w the outside world
* Data Storage:
  + Volatile main memory: DRAM (4Gigs, 8Gigs, 16Gigs)
    - Volatile because when power off -> loses all data
  + Non-volatile secondary memory: SSD, USB Drive, Hard Drive
* Networks (I/O): LAN, WAN, Wireless
* Integrated Semiconductor Chips
  + Semiconductor Tech: Switches -> gates and stuff, Conductors, Insulators
  + Yield: % dies per wafer. A die is: a small block of semiconducting material on which a given functional circuit is fabricated
  + Chiplets = fraction of a chip that can be assembled later on to form a chip
* Bigger the processor the better performance it is
* Performance is not increased by a higher clock speed anymore because then power density too high and processor too hot. Now, it is increased by increasing the number of cores
* Area scaling -> transistors are x10 every 2 yrs or so: moore’s law.
* Computer Performance:
  + Response time (latency): How long does it take to execute a job
    - Also called execution time; measured in time units/job
    - Limited by physics aka speed of light; improved by shorter transistor distances or faster disk rotation
  + Throughput: How many jobs can the machine run at once
    - Also called bandwidth; measured in jobs/time unit
  + Throughput IS NOT inverse of response time
  + Increasing GHZ of a processor will increase both latency and throughput
  + Doubling cores will only improve throughput and not latency
* CPU time = CPU cycles executed \* cycle time
  + CPU Execution time = instruction count \* CPI (cycles/instruction) \* clock cycle time (seconds/cycle)
* CPU cycles = # instructions \* CPI (average cycles per instruction)
  + For ex: I/O or multiplication takes far more cycles than something like and or addition
  + CPI is what heavly depends on the makeup of the processor
    - CPI depends on the makeup of the instructions (what percent of instructino list is multiplication or what percent addition). Could change if different machine organizations.
  + CPI = sum of (CPI of every single instruction clas\* the number of cycles per instruction of that class)
* Benchmarks:
  + Microbenchmarks: measures 1 feature aka memory accesses or communication speed
  + Kernel: measures most computational-intensive part of app
  + Full App: account for everything
    - Default: SPEC benchmark (System Performance Evaluation Cooperative)

## Discussion 1; Thursday, October 2, 2020

**Inside Thoughts and Notes:**

TA Discussion sessions are insanely helpful we didn’t learn anything about ISA in the actual lectures because it was more holistic but in the discussion session we’re going thru all the important parts of the RISCV ISA which is rly nice cuz he’s going thru the examples and everything.

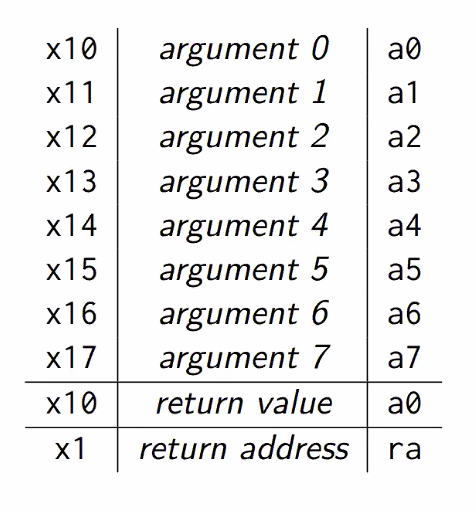
Also what I said above about 75% being an A- is nothing I just went to CS97 and a 50% is a B+ like bruh what is wrong with these classes lmao.

**Class Notes:**

Links:

* <http://www.riscvbook.com/greencard-20181213.pdf>
* <https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/>

ISA

* Destination register is actually put before the operand registers in RISC-V
  + Add rd, rs1, rs2 # add : rd <- rs1 + rs2
* Loading an integer into a register is easy all you have to do is add 0 to the immediate and load it into a register
  + However, when you want to load a big, 32 bit number, instructions are already 32 bits so it requires two steps: add and load upper immediate or (lui)
    - For example, lui x3, 1 and then addi x3, 1 will set x3 to 4097 which is actually bigger than what is allowed in a single addi instruction (size of the imm operand; will learn later)
* Load and store actually have opposite parameter lists like the +imm is on the src vs the dest for each so be careful ab that
* Also imm is measured in bytes so 1(x4) would be 1 byte after memory val at x4 not 1 int if u want 1 int ud do 4 bytes
* is 32 bits: halfword is 16 bits: 1 bytes is 8 bits
* 
* Save values on the stack (sp = stack pointer) (x2 normally)
  + Contains address of last inserted element
  + Pushing rs1 onto the stack
    - Addi sp, sp, -4
    - Sw rs1, 0(sp)
  + Popping stack top element and putting it into rd
    - Lw rd, 0(sp)
    - Addi sp, sp, 4
* GO LOOK AT FUNCTION CALLS AND REVIEW HOW TO WRITE ASSEMBLY FOR THAT/UNDERSTAND
* A picture containing table

  Description automatically generated
* Recursive function calls essentially need 3 flags, the initial function flag, the recursion loop flag, and the return output flag: use branch statement and jump statements to rn the recursive loop
  + Make sure to have a jump with x0 at the end to quit out
  + MAKE SURE to change the value of sp back

Exercises

* X1 = a, x2 = b, -> x1 = 4a + b + 1, x2 = b
  + Slli x1, xi, 2
  + Add x1, x1, x2
  + Addi x1, x1, 1
* Loop
  + Addi x3, x0, 0
  + Loop:
    - Beq x1, x0, end
    - Add x3, x3, x2
    - Subi, x1, x1, 1
    - Jal x0, loop
  + End:
* Recursive Example
  + Graphical user interface, text, application

    Description automatically generated

## Lecture 2; Tuesday, October 6, 2020

**Random Notes and Stuff**

* Text

  Description automatically generated
* Bruh wtf ^^

**Class Material Notes**

* Amdahl’s law:
  + exe time after improvement = exe time affected / amt of improvement + exe time unaffected
  + make the common case fast (aka the thing that happens most often)
* Power = capacitive load \* voltage^2 \* frequency
  + Capacitive load depends on chip mechanics and circuitry
  + FIX PROBLEM 4 IN HW 1
* Multicore Processors (industry moving in this direction because ^^ too hard to implement)
  + More than 1 processor per chip: requires explicit parallel programming
* Constraints of an ISA: performance, power, area, cost.
* Von Neumann Architecture: instructions + data stored in memory
  + Control:
    - Text

      Description automatically generated
  + Datapath:
    - Text

      Description automatically generated
* ISA sits right below firmware and right above instruction set processor
  + Visible to the assembly programmer who needs to undersatnd the ISA
* Instruction: Instruction fetch -> Instruction decode -> Operand fetch -> execute -> result store -> next instruction
* RISC has a fixed instruction set of 4 bytes (unlike x86-64 which is variable)
  + Also has 32 registers
* Types of machines (w.r.t operands on register/memory)
  + Stack machine, Acculumator Machine, Register-Memory Machine, Load-Store Machine (RISC 5, MIPS)
  + For a + b = c
    - Stack: push a, push b, add, pop c
    - Accumulator: load a, add b, store c
    - Register-Memory: add c, a, b
    - Load-store: load r1, a; load r2, b; add r3, r1, r2; store c, r3
* RISC V has 32 64-bit registers
  + 32-bit data is called a word
  + A picture containing text

    Description automatically generated
* RISC V is little endian
  + DOES NOT REQUIRE REQUIRE WORD ALIGNMENT
* R-Format Instructions
  + Funct7, rs2, rs1, funct3, rd, opcode
  + 7, 5, 5, 3, 5, 7 (bits each)
  + .
  + A picture containing table

    Description automatically generated
* I-Format instructions
  + Immediate, rs1, funct3, rd, opcode
    - Immediate in 2’s complement
  + 12, 5, 3, 5, 7 (bits each)
* Note that for R and I format instructions, the position of rs1, rd, and opcode are at the same place!!
* S-Format instructions
  + Imm[11:5], rs2, rs1, funct3, imm[4:0], opcode
  + 7, 5, 5, 3, 5, 7 bits each
  + Rs1: base address register number, rs2: source operand register number
  + Immediate is offset to keep rs1 and rs2 same place
* X5-x7, x28-x31: temporary registers, not preserved by callee
* X8-x9, x18-x27: saved registers (callee saved and restores)

## Lecture 3; October 8, 2020

**Class Notes:**

RISC-V Encoding Summary

* Table

  Description automatically generated

ALU Techniques

* Carry into MSB xor Carry out of MSB
  + If 0, no overflow. If 1, overflow.

## Discussion 3; October 9, 2020

**Class Notes:**

* Clock cycle time = seconds per clock cycle

## Lecture 4; October 13, 2020

Class Notes:

* Section 3.5 in the Textbook
* IEEE 754 Floating Point Encoding
  + Text

    Description automatically generated
  + How to normalize a binary decimal. Shift to the right/left until only a single 1 to the left of the decimal point. Then drop the one.
  + Bias + whatever exponent you want to store = E. E is always unsigned and positive.
    - Single Precision: Bias = 127
    - Double Precision: Bias = 1023
  + Table

    Description automatically generated
* Floating point vs Fixed point
  + Location of the decimal point = fixed or not
    - 123.45, 23.67, 2333893.00 (fixed)
      * If I wanted to represent 123.456, not possible must truncate
      * Don’t need to store location of decimal point
      * Arithmetic similar to integer arithmetic (ALU)
    - Floating point: allows larger range of numbers that you can represent
      * Must store location of point
* Underflow: # too small to represent
* For example: if x and y are representable but small, x – y can equal 0 because too small (10^-39) and thus x == y but that is wrong.
* INTEGER ARITHMETIC CANNOT UNDERFLOW

FP Arithmetic

* Addition:
  + Text

    Description automatically generated
* Text

  Description automatically generated
* Multiplication:
  + Text

    Description automatically generated
* Rounding Modes:
  + A picture containing chart, scatter chart

    Description automatically generated
* Extra Bits:
  + Guard, Round, Sticky
    - Sticky: once it sees a one, it sticks to one.
    - A picture containing table

      Description automatically generated
  + Important for the different types of policies^^^ for rounding
* F0-f31 are registers 64-bits wide, and all floating point registers for RISC-V
  + Floating point instructions can only use floating point registers.
  + Single-precision uses the lower 32 bits of these registers
  + Flw, fld, fsw, fsd (load and store word/doubleword instructions)

## Lecture 5; October X, 2020

Class Notes:

* Chapter 4 (including next few lectures)
* Today: Datapath Design
* RISCV Register File: 32 64-bit registers, 3 ports:
  + 2 output buses A & B for reading, 1 input bus for writing
  + (max read from a command is 2 registers)
  + Diagram

    Description automatically generated register
  + Diagram

    Description automatically generated memory
* Single cycle datapath: forces CPI for every instruction is 1.
  + (cycle time will be long because memory instructions take a long time)
* SIDENOTE: mutliply the immediate by 2 before adding to program counter when branching :?????
* Diagram

  Description automatically generated

## Lecture 6; October X, 2020

* Chapter 4 still:
* Pipelined Datapath:
  + Graphical user interface, text

    Description automatically generated

## Lecture 7; October X, 2020

* Types of hazards: Data Hazard, Control Hazard, Structure Hazard
* Data Hazard: result is needed in pipeline before available
  + Register file bypass: first half vs second half of cycle. Write then read ☺
  + Software: no-ops, hardware: data forsurwarding/insert bubbles