CS151B Winter 2022

Discussion (Week 1)

CS151B Teaching Team

Today's Agenda

- Icebreaker
- Logistics
- Course Intro
- Concept Review
- Practice Questions
- Q&A

Let's introduce ourselves!

My name: **Sang-joon Lee (he/him)**

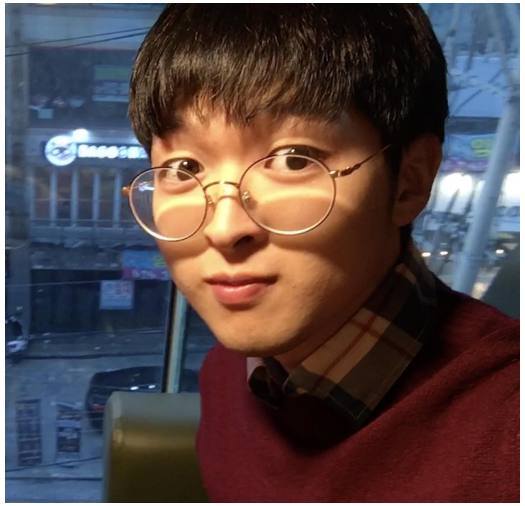
My email: **sangjoonlee@cs.ucla.edu**

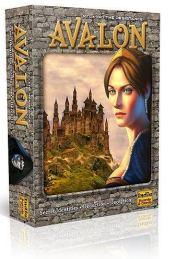
Office Hours: **Wed 2:00am-4:00pm**

Interesting Fact: **I love KBBQ, play Drum Set**

I'm a M.S. student in CS Department (RoMeLa, under Prof. Dennis Hong)

I also did my undergrad in UCLA, and I took this class CS151B under Prof. Reinman





Logistics

- Grading
  - 10% homework (graded based on completion)
  - 40% midterm
  - 50% final
- Piazza is used for all the course related discussions, you can access it via BruinLearn (make sure to use a less restrictive browser)
- Gradescope is used for all the course assignments (also accessible via BruinLearn), if you have issues accessing them, please contact **Salekh** ASAP (salekh@cs.ucla.edu)
- **HW1 has been posted and it's due tonight (11:59 PM PST)!!!**

Tips to succeed in this class

- Watch the pre-lecture videos and show up to the lectures prepared
- **Ask** and discuss the most challenging topics during the lectures
- Post any confusions on Piazza
- Be on the top of the homework assignments
- Show up to the discussions for review and additional practice
- Take advantage of the office hours (we are always here to help)

Welcome to CS151B again, hopefully this will be a meaningful experience for you

(This quarter we will be using a coordinated approach for discussions to make them consistent)

Why do we learn computer architecture ?

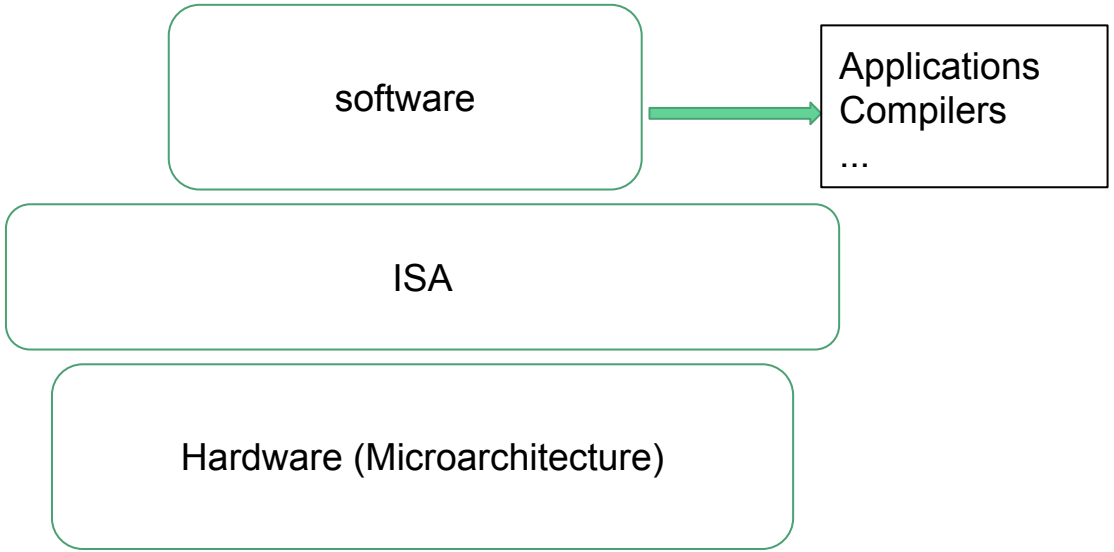
- Write better code
- Understand how hardware executes our high-level code, and its performance
- How software interacts with hardware to make it do its instructions
- Better understand performance
- Learn how to design better systems to achieve energy efficiency
- Sequential vs parallel execution

……..

ISA and microarchitecture

- **ISA:** defines what set of instructions/registers you can use and the device can execute (it's an abstract model of a computer)
- **Microarchitecture:** the hardware implementation of an ISA

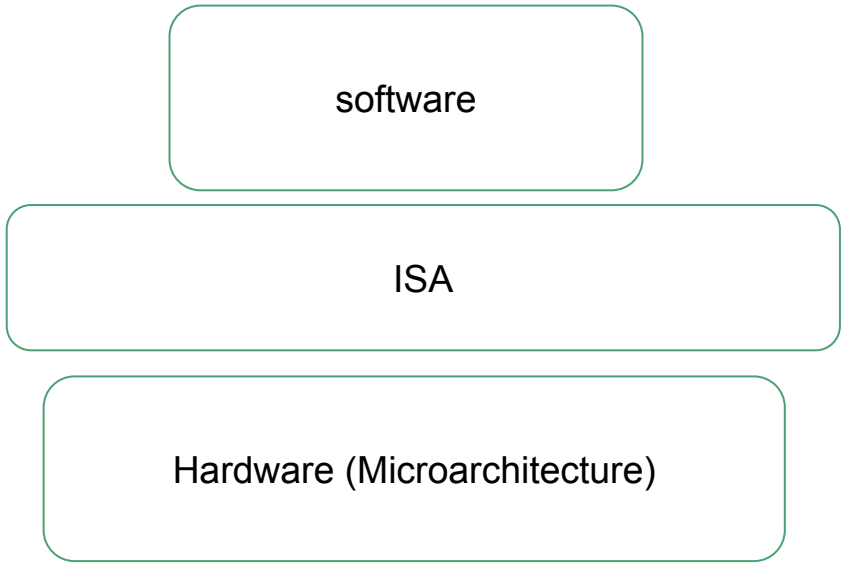
Interface between low-level software and hardware



ISA is a *contract* between software and the hardware, similar to the concept of an API

ISA and microarchitecture

- Different processors can have different implementations of same ISA
  - e.g. Intel and AMD both implements x86 ISA, but have different microarchitecture



How to measure performance ?

- **Throughput** : number of tasks completed per unit time
- **ET**: Execution Time of a given program
- **IC**: Instruction Count
- **CT**: Cycle Time
- **CPI**: Cycles Per Instruction. Average number of clock cycles per instruction for a program or program fragment.

$$ET = IC \times CPI \times CT$$

CPI and **CT** might be at odds: Faster **CT** might increase **CPI** for an instruction

*Always bear in mind that the only complete and reliable measure of computer performance is **time**.*

What can affect performance ?

The performance of a program depends on the **algorithm**, the **language**, the **compiler**, the **architecture (ISA)**, and the actual **hardware**.

Algorithm : - IC, possible CPI

Language : - IC, CPI

Compiler : - IC, CPI

Architecture (ISA) : - IC, CPI, CT

Hardware : - CPI, CT

IC: Instruction Count
CPI: Cycles Per Instruction
CT: Cycle Time
ISA: Instruction Set Architecture

What Affects Performance

Component	SW/HW	Affects	Description
Algorithm	SW	Instruc. Count & CPI	Determines how many instructions & which kind are executed
Programming Language	SW	Instruc. Count & CPI	Determines constructs that need to be translated and the kind of instructions
Compiler	SW	Instruc. Count & CPI	Efficiency of translation affects how many and which instructions are used
Instruction Set	HW	Instruc. Count, CPI, Clock Cycle	Determines what instructions are available and what work each instruction performs
Microarchitecture	HW	CPI, Clock Cycle	Determines how each instruction is executed (CPI, clock period)

CISC: Complex Instruction Set Computers
RISC: Reduced Instruction Set Computers

CISC vs RISC

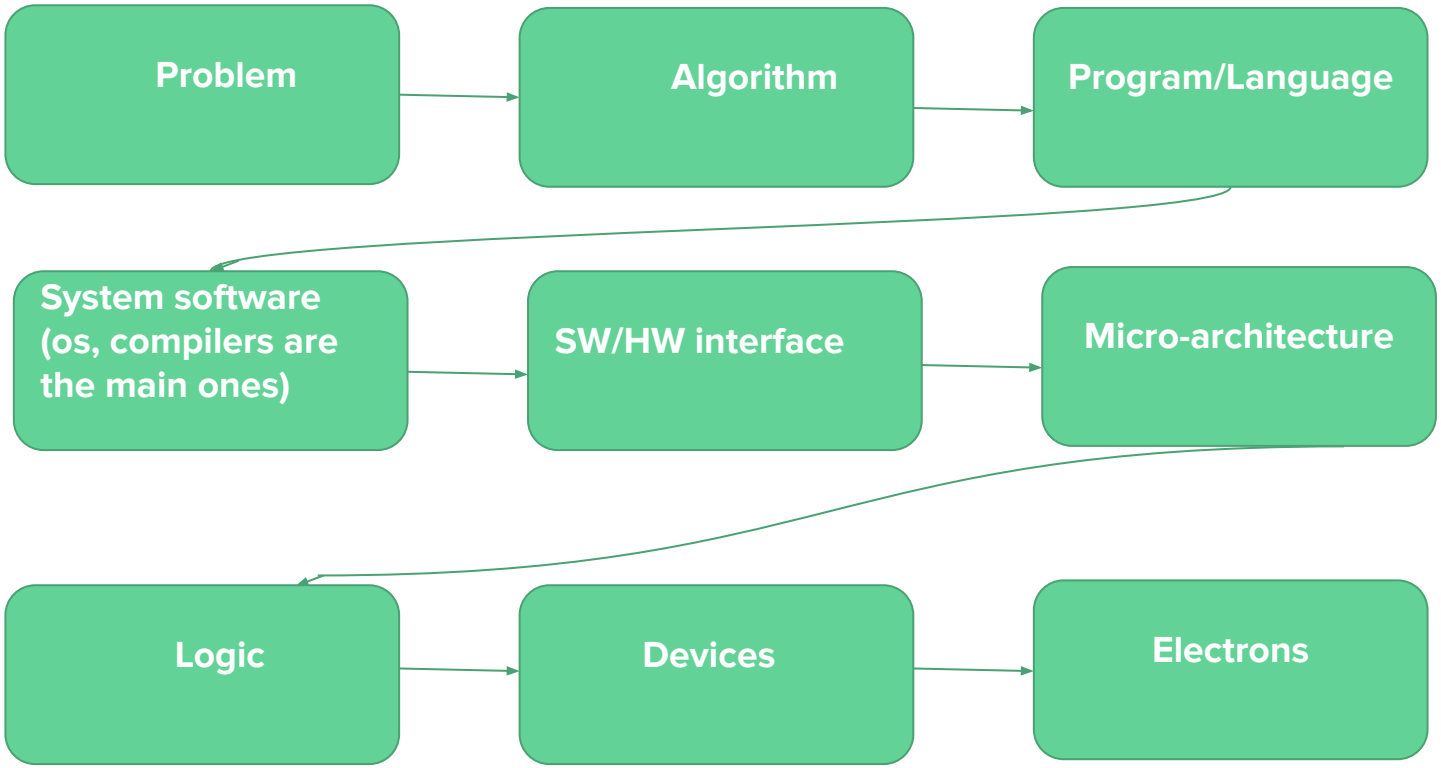
CISC (was designed when memory was an expensive resource)

- more complex instructions
- One instruction can do many things (like MAM (axb+cxd) in one single instruction), so it leads to more compact code
- Makes hardware difficult to design and verify (more on this later)
- Examples: x86, AMD

RISC

- Simpler instructions
- Easier to exploit pipelining & parallelism at software/hardware level
  - Software side, can schedule instructions taking longer time earlier
  - Hardware side, can do pipelining of multiple instructions
- Examples: ARM, MIPS, RISC-V

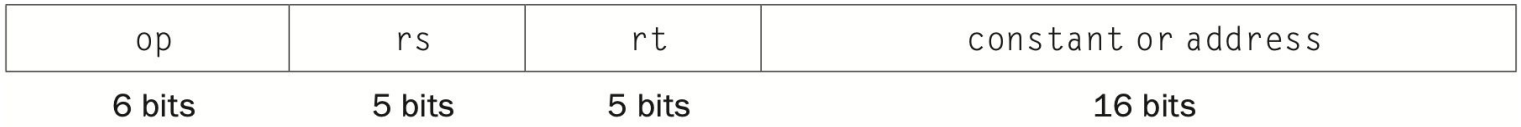
Steps to solve a problem using a computer



Instruction Formats

(A form of representation of an instruction composed of fields of binary numbers)

I type:



R type:

J type:

Q : How does the machine distinguish between different formats ?
A: By the opcode at the beginning.
Q: What's the opcode for R-type instructions?
A: 0
Q: How to distinguish different R-type instructions?
A: the funct field

MIPS Reference Sheet

Some basic MIPS instructions (summary)

Instruction Type	Example Instructions
R-type	add, sub, and, or, nor, slt, jr ...
I-type	addi, andi, ori, slti, lw, sw, beq, bne, ...
J-type	j, jal

Some basic MIPS instructions

- Arithmetic instructions (performs mathematical operations)
  - **add**
    - Example : `g = h + k`, `In MIPS: add $s0,$t0,$t1`
  - **sub**
    - Example: `g = h - k`, `In MIPS: sub $s0,$t0,$t1`
- Data transfer instructions (moving data between memory and registers)
  - **lw**
    - Example: `g = A[8]`, `In MIPS: lw $t0,32($s0)`
  - **sw**
    - Example: `A[12] = h`, `In MIPS: sw $t1, 48($s0)`

Note: for arithmetic operations, operands can only be retrieved from some registers (not directly from memory)

Some basic MIPS instructions

- Control instructions
  - **bne** : branch not equal (I-type)
    - Example: `if (h==j) : PC = PC + 4 else PC = PC + 4 + imm`
      - MIPS: `bne $rs, $rt, imm`
  - **beq** : branch equal (I-type)
    - Example: `if (h != j) : PC = PC + 4 else PC = PC + 4 + imm`
      - MIPS: `beq $rs, $rt, imm`

Q: How far can we jump with these instructions?

Some basic MIPS instructions

- Jump instructions
  - j : jump (J-type)
    - Example: `j LABEL`
  - jal : jump and link (J-type)
    - Example: `jal LABEL`
  - jr : jump register (R-type)
    - Example: `jr $ra`
  - jalr: jump and link register (R-type)
    - Example: `jalr $t0`
- Set instructions
  - slt : set less then (R-type)
    - Example: `slt $t0,$t1,$t2`

R-type format:

Example

Translate **`add $t0,$s1,$s2`** into machine code:

0	17	18	8	0	32

Register encodings: $s1 -> 17, $s2 -> 18, $t0 -> 8

Other fields: (opcode)R-type -> 0, shift amount-> 0, (funct)add -> 32

Binary translation: 0 -> 000000, 17 -> 10001, 18 -> 10010, 8 -> 01000, 0 -> 00000, 32 -> 100000

Final Format:

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

The final format is the machine code format, which looks different from the assembly format

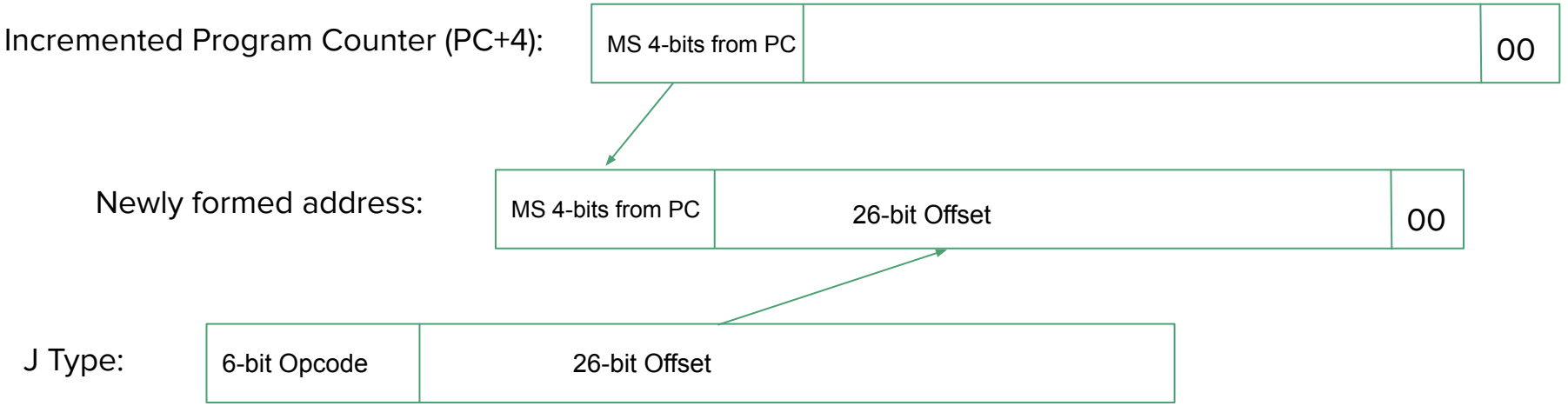
Checkpoints (Think about these questions)

1. Why don’t we simply have more registers ?
2. Why do we not have a very high clock frequency?
3. Pros and cons of fixed and variable length instructions ?
4. What does IC depend on ? (architecture not the exact implementation)
5. CPI varies by application, as well as among implementations with the same instruction set, why so?
6. Is the minimum CPI equal to 1?

Remark: *registers take less time to access and have higher throughput than memory, making data in registers both faster to access and simpler to use. Accessing registers also uses less energy than accessing memory. To achieve highest performance and conserve energy, an instruction set architecture must have a sufficient number of registers, and compilers must use registers efficiently.*

Confusion Point

Addressing using upper 4 bits and 26 bits of the immediate and 2 lower bits are 0's.



Homework Clarification

- 1.8b:
  - Assume CPIs in 1.8b are the same as the CPIs in 1.8a
  - Question was asked incorrectly, it should be **“how much faster is B compared to A”** , not the other way around.

Practice Questions

https://webwhiteboard.com/board/fPl737qjcpUpdwGzpGbRd1qJgKAAsQDv/

P1: (ISA)

1)Translate the below code to MIPS assembly code:

```
A[300] = h+ A[300]
```

(Assume $t1 has the base of the array A, and $s2 has the value of h, A is an int array, $t2, $t3 are temporary registers)

```
lw $t2, 1200($t1)
add $t3, $s2, $t2
sw $t3, 1200($t1)
```

2)What instruction formats are there ?

R-type(add) and I-type(lw, sw)

P2: (ISA)

In MIPS only NOR logical operation is supported, how can we achieve the effects of the NOT operation using NOR?

NOR:

A	B	result
T	T	F
F	T	F
T	F	F
F	F	T

A: **x NOR 0**

Think about the following:

x NOR 0 ⇔ NOT (x OR 0) ⇔ NOT x

(⇔ indicates equivalence)

X NOR X also works!

Q: Why not just a single NOT instruction instead ?

A: Design decision to keep with the three-operand format (R).

P3: (ISA)

How can we simulate the **mv** instruction using an **add** instruction in MIPS?

Answer: we can do **add $s1 $s2 $zero,** this move $s2's content to $s1

P4: Compile the below C code *(optional)*

In the following code segment, f, g, h, i, and j are variables. If the five variables f through j correspond to the five registers $s0 through $s4, what is the compiled MIPS code for this C *if* statement?

C-code:

if (i == j) f = g + h; else f = g – h;

MIPS:

```
        bne $s3, $s4, Else      # go to Else if i ≠ j
        add $s0, $s1, $s2       # f = g + h (skipped
if i ≠ j)
        j Exit     # go to Exit
Else:     sub $s0, $s1, $s2           # f = g – h
(skipped if i = j)
Exit:
```

How far can we jump by bne instruction ?

P5: (performance)

Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much? (assuming we use the same compiler)

Answer:
same program, compiler, ISA => IC stays the same
$ET_A = IC \times CPI_A \times CT_A = IC \times 500ps/instruction$
$ET_B = IC \times CPI_B \times CT_B = IC \times 600ps/instruction$
$ET_B / ET_A = 1.2$

P6: (performance)

A program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

P6: (performance) Solution

Answer:

$$CR_B = \frac{1}{CT_B} = \frac{1}{\left(\frac{ET}{CPI_B \; X \; IC}\right)} = \frac{CPI_B \; X \; IC}{ET_B} = \frac{1.2 CPI_A \; X \; IC}{ET_B} = \frac{1.2 \; (ET_A / CT_A)}{ET_B}$$

$$= \frac{1.2}{ET_B} X \; ET_A \; X \; CR_A = \frac{1.2}{6 \; s} \; X \; 10 \; s \; X \; 2 \; GHz = 4 \; GHz$$

P7: (Performance)

A compiler designer is trying to decide between two code sequences for a particular computer. The hardware designers have supplied the following facts:

	CPI for each instruction class		
	A	B	C
CPI	1	2	3

Compiler designer is considering two code sequences with the below instruction counts:

Code Sequence	Instruction Counts For Each Instruction Class		
	A	B	C
1	2	1	2
2	4	1	1

1) Code 2 has more instructions

2) Total cycle$_1$ = 2x1+1x2+2x3 = 10
   Total cycle$_2$ = 4x1+1x2+1x3 = 9
   So code 2 is faster

3) CPI$_1$ = 10 / 5 = 2
   CPI$_2$ = 9 / 6 = 1.5

Question: Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

P8: (Performance)

A given application written in Java runs 15 seconds on a desktop processor. A new Java compiler is released that requires only 0.6 as many instructions as the old compiler. Unfortunately, it increases the CPI by 1.1. How fast can we expect the application to run using this new compiler?

Answer:

$$ET_B = IC_B * CPI_B * CT_B = (0.6\, IC_A) * (1.1\, CPI_A) * CT_A$$

$$= 0.6 * 1.1 * (IC_A * CPI_A * CT_A) = 0.6 * 1.1 * ET_A$$

$$= 0.6 * 1.1 * 15\, sec = 9.9\, sec$$

P9:(performance)

A program has 2 billion instructions and runs on a computer with 4GHz clock (CT=0.25ns). The types of instructions and their cycles are shown below:

Mix	instructions	cycles
20%	LW	7
60%	R-type	5
10%	SW	4
10%	BNE, BEQ	3

We introduce the following modifications:

- A new **LAW** instruction can perform **LW** and **ADD** in a single instruction.
- **LAW** only takes 8 cycles.
- If there is a **LW** followed by **ADD** in the program, they will be replaced by **LAW**.
- 50% of **LW** in the program has the above pattern.

Q2. What's the new CPI? What's the new execution time? (assuming we have same cycle time)

This question was collected from a previous midterm exam

P9:(performance) solution

Important: note the instruction mix has changed now!

Original Table

Mix	IC	instructions	cycles
20%	0.2 * N	LW	7
60%	0.6 * N	R-type	5
10%	0.1 * N	SW	4
10%	0.1 * N	BNE, BEQ	3

New Table

Mix	IC	instructions	cycles
1/9	10% * 2 billion = 200 million	LAW	8
1/9	10% * 2 billion = 200 million	LW	7
5/9	50% * 2 billion = 1 billion	R-type	5
1/9	10% * 2 billion = 200 million	SW	4
1/9	10% * 2 billion = 200 million	BNE, BEQ	3

Original instruction count **N** = 2 billion

New instruction count **N** = 1.8 billion

```
CPI = (1/9)x8 + (1/9)x7 + (5/9)x5 + (1/9)x4 + (1/9)x3 = 47/9
ET = IC x CPI x CT = 1.8x10^9 x 47/9 x  0.25x10^(-9) = 2.35s
```

This question was collected from a previous midterm exam

Acknowledgment

Patterson, David, and John Hennessy. *Computer Organization and Design MIPS Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. 5th ed., Morgan Kaufmann, 2013.

Credit to Prof. (Glenn) Reinman

Practice questions are partially extracted from the previous midterm exams

Questions ? Feedback?

Thanks!