

**JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**DEPARTMENT:** COLLEGE OF PURE AND APPLIED SCIENCE

**UNIT NAME:** COMPUTER ARCHITECTURE

**UNIT CODE:** BCT 2408

**NAME:** WARWINU PHILLIP KAMAU

**REG NO:** SCT212-0048/2021

**LECTURER NAME:** DR. MULANG ONANDO

**TASK:** LAB1

**1. After graduating, you are asked to become the lead computer designer at Hyper Computers Inc. Your study of usage of high-level language constructs suggests that procedure calls are one of the most expensive operations. You have invented a scheme that reduces the loads and stores normally associated with procedure calls and returns. The first thing you do is run some experiments with and without this optimization. Your experiments use the same state-of-the-art optimizing compiler that will be used with either version of the computer. These experiments reveal the following information:**

**• The clock rate of the unoptimized version is 5% higher.**

**• 30% of the instructions in the unoptimized version are loads or stores.**

**• The optimized version executes 2/3 as many loads and stores as the unoptimized version. For all other instructions the dynamic counts are unchanged.**

**• All instructions (including load and store) take one clock cycle.**

**Which is faster? Justify your decision quantitatively.**

Step 1: Define the performance metrics  
Execution time is given by:

Since CPI=1 for both versions, simplify to:

Step 2: Compute Instruction Counts  
Assume the unoptimized version executes *I*=100 instructions:

* Loads/stores in unoptimized version:
* Other instruction: 70

Optimized version:

* Loads/stores reduced to
* Other instruction remains 70

Step 3: Compare Execution Times  
Let Then:

* Unoptimized clock rate = 1.05f
* Optimized clock rate = f

Execution times:

Speed Calculation:

The optimized version is hence 5.8 % faster.

**Conclusion:** the optimized version is 5.8 % faster due to fewer loads/stores, despite its 5% lower clock rate.

**2. Several researchers have suggested that adding a register-memory addressing mode to a load-store machine might be useful. The idea is to replace sequences of:**

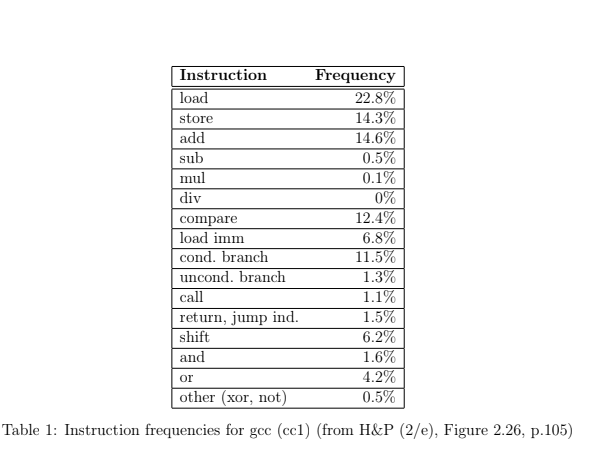
**LOAD Rx,0(Rb)**

**ADD Ry,Ry,Rx**

**by**

**ADD Ry,0(Rb)**

**Assume this new instruction will cause the clock period of the CPU to increase by 5%. Use the instruction frequencies for the gcc benchmark on the load store machine from Table 1.**



**The new instruction affects only the clock cycle and not the CPI.**

**1. What percentage of the loads must be eliminated for the machine with the new in-struction to have at least the same performance?**

Original Execution Time:

Where I = total instructions, f = clock rate.

New Execution Time

* Let *x*= fraction of loads replaced.
* Loads replaced:
* Instructions saved:
* New instruction count:
* New clock rate:
* New execution time:

Condition for Equal performance

Solving for x:

Percentage of load to be eliminated for at least same performance =10.4 %

**2. Show a situation in a multiple instruction sequence where a load of a register (say Rx) followed immediately by a use of the same register (Rx) in an ADD instruction, could not be replaced by a single ADD instruction of the form proposed.**

LOAD Rx, 0(Rb)

ADD Ry, Ry, Rx

SUB Rz, Rz, Rx

The replacement fails when the loaded register (Rx) is used in multiple instructions after the load. Since ADD Ry,0(Rb) does not store Rx, subsequent instructions that rely on its value would break.

**3. In the early years of the RISC versus CISC dispute, the total number of different instructions and their variations in the ISA was a common indication of the “simplicity” of an ISA (lesser the number, greater the simplicity). Modern RISC instruction sets contain almost as many instructions as old CISC instruction sets. Discuss whether modern “RISC” processors are**

**no longer RISC (as envisioned in the 80’s). If they are still RISC, then what features in the instruction set best define the simplicity of an ISA? (e.g. memory access instructions, fixed and simple instruction encoding, register-oriented instructions, simple data types, etc.?)**

Yes, but evolved. While modern RISC ISAs have more instructions than early RISC designs, they retain core principles:

* Register-based execution - Most instructions still operate on registers, with memory access limited to explicit load/store operations.
* Fixed-length instructions - Modern RISC architectures like ARM and RISC-V still use fixed instruction sizes, simplifying decoding.
* Efficient pipelines - Instructions execute in one or a few cycles, supporting high-speed execution.

However, modern RISC also includes:

* More instructions = More Complexity – Modern RISC architectures now include vector processing, floating-point operations, and specialized instructions (e.g., ARM's NEON, MIPS DSP extensions)
* Some micro-coding - Some processors now use microcode, blurring the line between RISC and CISC.
* Variable-length encoding in some cases - While core RISC designs use fixed instructions, compressed instruction sets (e.g., ARM Thumb, RISC-V C extension) introduce variable-length encoding, a feature once associated with CISC.

**4. Even though the Intel x86 ISA is a clear example of a CISC ISA, modern implementations of it (e.g. Core and Xeon) use many RISC ideas: register-based micro-instructions, pipelining, simple branch micro-instructions, fixed length micro-instructions, etc. Some say that, since at the low level the the latest Intel processors behave like a RISC, it is RISC. Others say that, since at the software interface (compiler) they are seen like a CISC, they are CISC.**

**Discuss at what level we should measure the complexity of ISA? What are the implications of considering the ISA at each level? Are the latest Intel processors RISC?**

Modern x86 behaves like RISC at the hardware level but remains CISC in software. Key observations:

Internally, x86 uses RISC-like micro-operations and pipelining:

* Micro-operations break complex instructions into simpler, fixed-size instructions.
* Pipelining and Out-of-Order Execution improve performance as modern x86 CPUs use deep pipelines and execute instructions in parallel.
* Register-based execution is prioritized that is, while x86 started with memory-to-memory operations, modern CPUs rely heavily on register-based execution (like RISC). Externally, x86 still has variable-length CISC instructions and legacy complexity.
* Variable-length instructions (software still sees CISC instructions).
* Legacy compatibility (decades-old features remain).
* Complex decoders are required for instruction translation.

x86 is RISC-like at the microarchitecture(hardware) level but remains CISC at the instruction set(software) level.