NAME: SANDRA LUNG’AHI

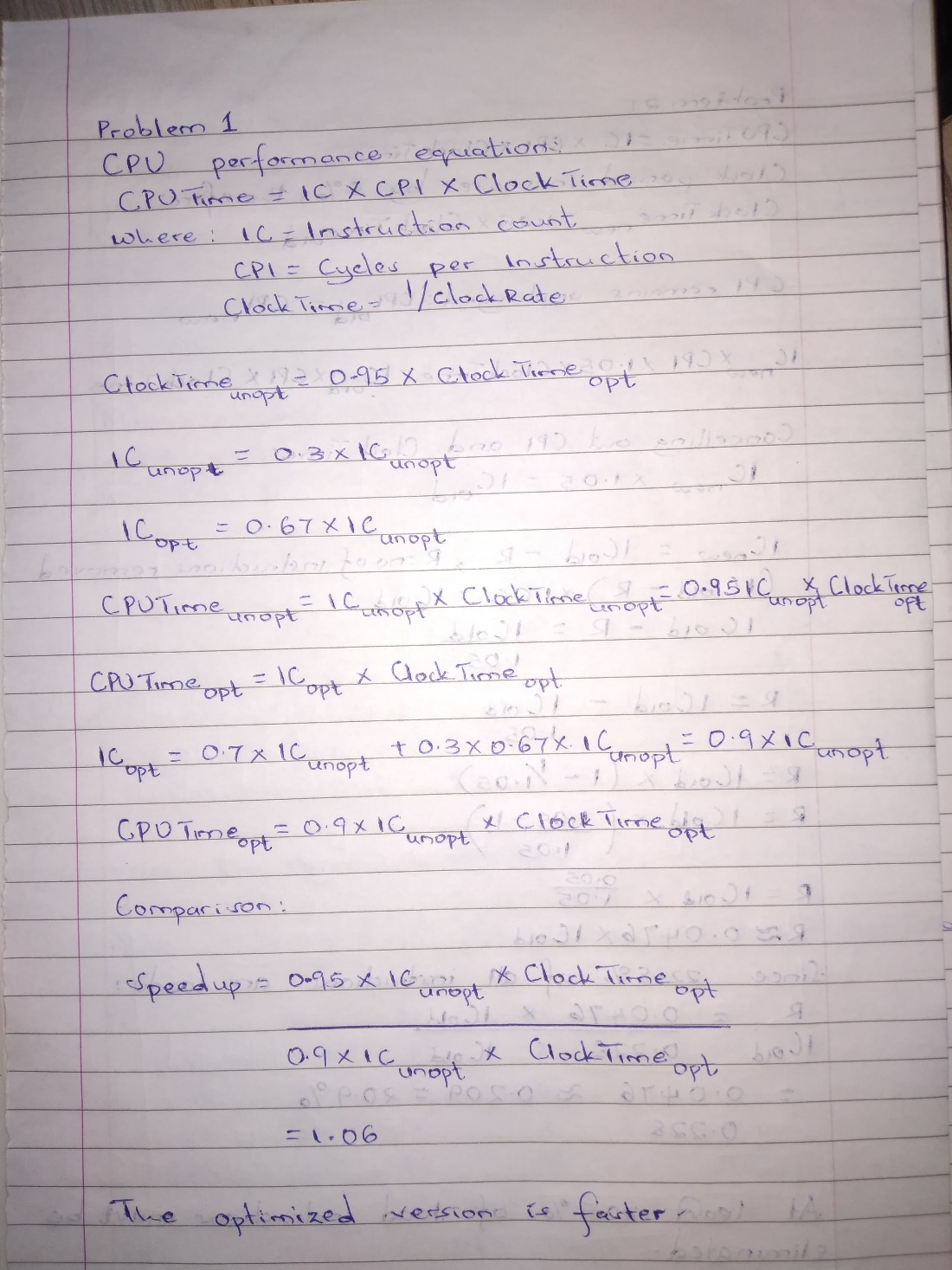
REG NO: SCT212-0072/2021

LAB 1 ASSIGNMENT

Problem 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.

Solution:



Problem 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:

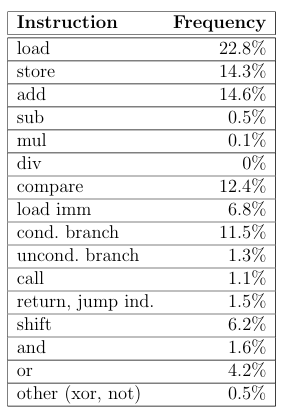
LOAD Rx,0(Rb)

ADD Ry,Ry,Rx

by

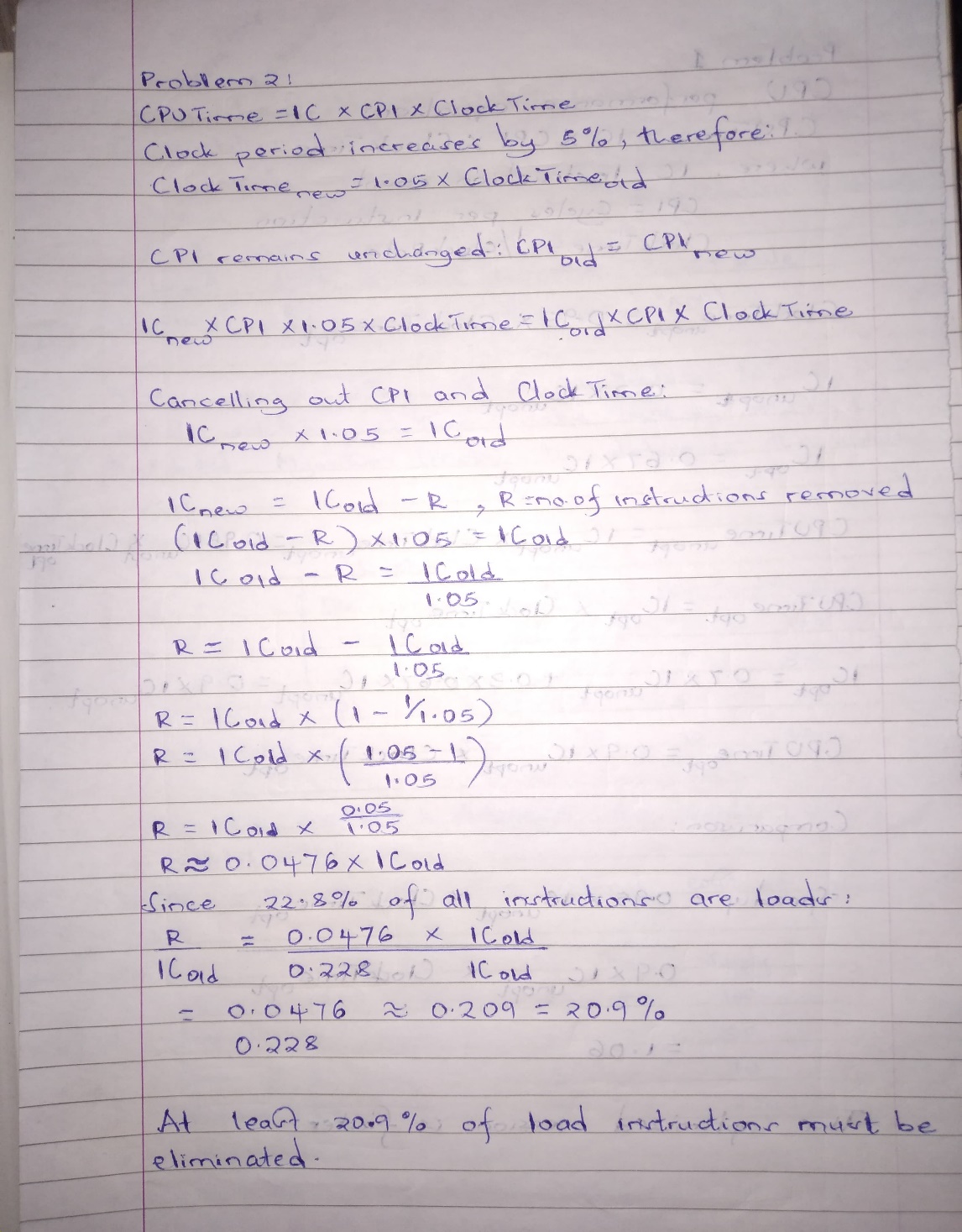
ADD Ry,0(Rb)

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.



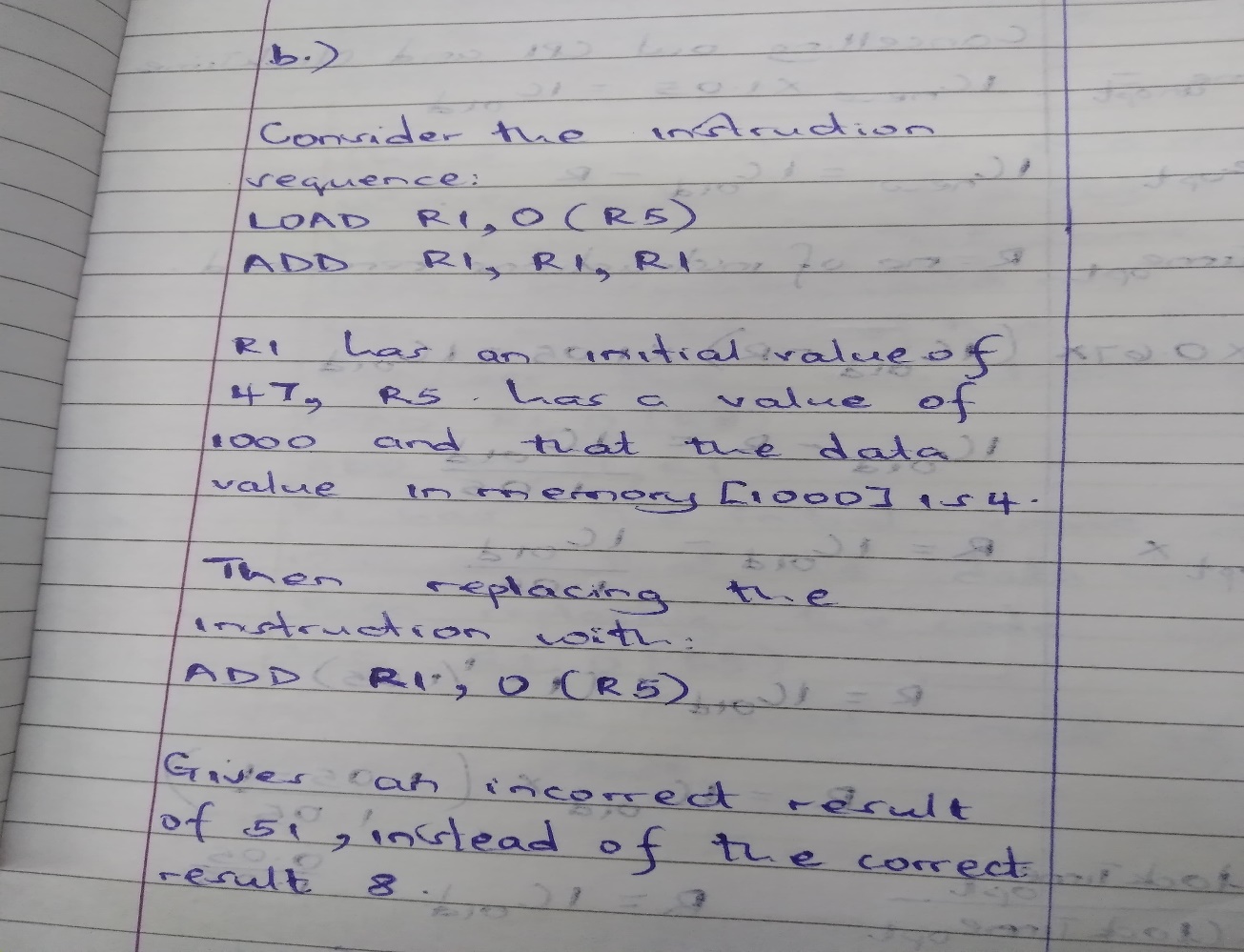
1. What percentage of the loads must be eliminated for the machine with the new instruction to have at least the same performance?

Solution:



2. Show a situation in a multiple instruction sequence where a load of a register (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.

Solution:



*In the early years of the RISC versus CISC dispute, the total number of different instructions and their variations in the IS 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 no longer RISC (as envisioned in the 80s). 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, modern RISC processors retain the core principles of RISC, despite their instruction sets having expanded. The original RISC philosophy emphasized simplicity in instruction design and execution, not merely a minimal instruction count.

RISC key distinctions from CISC remain intact and they are:

Fixed-Length Instruction Encoding: Modern RISC ISAs such as ARMv8 and RISC-V maintain fixed-length instructions, simplifying decoding and pipelining.

Load-Store Architecture: Only explicit load/store instructions access memory; arithmetic/logic operations work solely on registers.

Hardware Execution Without Microcode: Instructions are executed directly by hardware, avoiding CISC-style microcode translation.

Focus on Compiler-Friendly Design: Large register sets and orthogonal instructions (no arbitrary register restrictions) aid efficient compilation.

Despite the growth of instruction counts for SIMD, cryptography, or atomic operations, additions are modular or optimised for specific tasks without compromising core RISC principles. Complexity is managed through optional extensions thus keeping the base ISA simple.

The simplicity of a RISC ISA is best defined by the following characteristics:

Fixed and Simple Instruction Encoding: Fixed-length instructions with uniform fields enable fast decoding and predictable pipeline stages.

Load-Store Architecture: Restricts memory access to dedicated load/store instructions, avoiding complex memory operands in arithmetic operations.

Register-Centric Operations: Most instructions operate on registers, minimizing memory interactions and promoting efficient register usage.

Limited Addressing Modes: Simple addressing reduces hardware complexity compared to CISC’s register-indirect or scaled modes.

Orthogonal Instruction Set: All registers are interchangeable in instructions, eliminating special-case rules.

Simple Data Types: Support for basic types like integers and floats in hardware, with complex types handled via software.

Single-Cycle Execution: Most instructions complete in one cycle, enabling deep pipelining and high clock rates.

*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 latest Intel processors behave like a RISC, they are RISC. Others say that, since at the software interface (compiler) they are seen as CISC, they are CISC. Discuss at what level ISA complexity should be measured. What are the implications of considering the ISA at each level? Are the latest Intel processors RISC?*

ISA complexity should be measured at the software-visible interface (the architectural level), not the microarchitectural implementation. The ISA defines the instructions, addressing modes, data types, and registers exposed to programmers and compilers. While modern Intel processors internally translate complex x86 instructions into RISC-like micro-operations, this is an implementation detail hidden from software. The ISA's complexity is determined by:

Instruction diversity: Variable-length instructions and multi-operation commands such as PUSHAD in x86.

Addressing modes: Complex memory access patterns such as scaled-index addressing.

Register usage: Special-purpose registers and implicit operands.

2. Implications of Considering the ISA at Each Level

Software/Compiler Level (CISC ISA):

Complexity: Compilers must handle variable-length instructions, intricate addressing modes and a large instruction set.

Code Density: CISC instructions can encode more operations per byte, potentially reducing code size.

Backward Compatibility: Maintains support for legacy software, a key strength of x86.

Microarchitectural Level (RISC-like Execution):

Decoding Overhead: Front-end complexity increases due to translating CISC instructions into micro-ops.

Execution Efficiency: Simpler micro-ops enable pipelining, out-of-order execution and superscalar designs, improving performance.

Power and Area Trade-offs: Decoders and micro-op caches add hardware complexity, but streamlined execution units enhance throughput.

3. Are the Latest Intel Processors RISC?

No, the latest Intel processors are CISC at the ISA level but employ RISC-like microarchitectural techniques.

The x86 ISA retains CISC traits such as variable-length instructions, multi-cycle operations and complex memory access which define its classification. Internally, processors decompose CISC instructions into fixed-length micro-ops, enabling RISC-style execution (pipelining, single-cycle operations). While the execution engine behaves like RISC, the ISA remains the defining factor. The term "CISC" describes the software interface, not the hardware implementation.