**Malcom Chiaji**

**SCT212-0063/2021**

**Lab1**

1.  
-The execution time is given by:

Execution Time = Instruction Count × CPIClock Rate

Clock rate

-Let ***I*** be the total instruction count in the unoptimized version.

30% of these instructions are loads and stores, so the number of load/store instructions in the unoptimized version is

0.3I

-The optimized version executes only 2/3 of the load/store instructions, so the number of load/store instructions in the optimized version is:

2/3\*0.3*I* = 0.2*I*

-The number of other instructions remains unchanged, meaning the total instruction count in the optimized version is:

*(I-*0.3*I)* + 0.2*I*

=0.9*I*

-Thus, the optimized version executes 90% as many instructions as the unoptimized version.

The clock rate of the unoptimized version is 5% higher than the optimized version. Let the clock rate of the optimized version be f, then the clock rate of the unoptimized version is:

1.05f

To compare performance, compute the ratio of execution times:

Execution time(optimized)/execution time(unoptimized)

= (0.9*I /* f) / (*I* / 1.05f)

= 0.945

Hence the optimized version is 5.5% faster

2. a)

Execution Time = Instruction Count × CPIClock Rate

Clock rate

Let:

* fL ​ be the fraction of instructions that are loads (22.8% or 0.228).
* fA ​ be the fraction of instructions that are adds (14.6% or 0.146).
* x be the fraction of loads eliminated.
* New clock time = 1.05 \* old clock time

If we eliminate xxx fraction of loads, the new instruction count reduces by 0.228x, making the new instruction count:

(1-0.228x)\*1.05 <= 1

1−0.228x <= 0.9524

0.228x >= 0.0476

X >= 0.228/0.0476​

≈0.209

At least **20.9% of load instructions** must be eliminated to maintain the same performance.

b)

One key scenario where the transformation cannot occur is when the loaded value is modified between the LOAD and ADD instructions.

3.

Modern RISC processors have more instructions than earlier RISC architectures, making them look complex. However, the fundamental principles of RISC still apply:

* Simple instruction encoding (fixed-length, regular formats).
* Register-oriented operations (most operations work on registers, not memory).
* Load-storearchitecture (memory access only via LOAD/STORE).

4.

The complexity of an ISA can be evaluated at two primary levels:

**a. Software Interface (Compiler/Programmer Level)**

* At this level, the ISA is viewed as the interface between the hardware and the software

Implications:

Easier for programmers/compilers to write compact code.

Higher-level abstractions reduce the number of instructions needed for a task.

Complexity is pushed to the hardware for decoding and execution.

**b. Hardware Implementation (Microarchitecture Level)**

* At this level, the ISA is implemented in hardware using micro-operations and other low-level techniques.

Implications:

Easier to design high-performance, pipelined processors.

Simpler hardware control logic.

Complexity is pushed to the software (compiler) for instruction scheduling and optimization.

* Modern Intel processors (e.g., Core, Xeon) exhibit characteristics of both RISC