Computer Architecture Lab 1

Name: Eddy Kimani

Reg No.: SCT212-0596/2021

E1: Optimized vs Unoptimized processor version

Step 1

Execution Time = (Instruction Count x CPI) / Clock Rate

where: Instruction Count = Number of instructions executed

CPI = Clock cycles per Instruction

Since all instructions take one cycle i.e. CPI = 1, the execution time becomes:

Execution Time = Total Instructions / Clock Rate

Step 2

Let N be the total instruction count in the unoptimized version

30% of N are loads and stores i.e.

Loads and Stores (Unoptimized) = 0.3N

The optimized version executes 2/3 as many loads and stores i.e.

Loads and Stores (Optimized) = 2/3 x 0.3N = 0.2N

All other instructions remain unchanged, meaning

Other Instructions = 0.7N

Thus, the total instruction count for the optimized version

$N_{optimized} = 0.7N + 0.2N = 0.9N$

Step 3

Unoptimized execution time = N / Unoptimized Clock Rate

The optimized version has 10% fewer total instructions, but its clock rate is 5% lower than the unoptimized version

Optimized Execution Time = 0.9N / (0.95 x Unoptimized Clock Rate)

<u>Step 4</u>

Speedup is given by:

Speedup = Unoptimized Execution Time / Optimized Execution Time

Speedup = (N / Unoptimized Clock Rate) / (0.9N / (0.95 x Unoptimized Clock Rate))

Speedup = N / Unoptimized Clock Rate x (0.95 x Unoptimized Clock Rate / 0.9N)

Speedup = 1 / (0.9 / 0.95) = 0.95 / 0.9 = 1.056

Therefore

Optimized version is 5.6% faster than unoptimized version

E2: Register memory addressing problem

Q1:

Execution Time = (Instruction Count x CPI) / Clock Rate

= Total Instructions / Clock Rate

The clock rate of the new machines decreases by 5%, meaning its execution time increases by 1.05 times unless instruction count is reduced

Let N be the original instruction count

Load instructions make up 22.8% of all instructions

Let X be the fraction of loads eliminated

The new instruction count is:

$N_{new} = N – X (0.228N) = (1 - 0.228X) N$

For performance to be the same:

$(1 – 0.228X) N / 1.05 = N$

Divide both sides by N

$(1 – 0.228X) / 1.05 = 1$

$1 – 0.228X = 1.05$

$-0.228X = 0.05$

$X = 0.05 / 0.228 = 0.219$

Therefore, at least 21.9% of load instructions must be eliminated to maintain the same performance

Q2:

The new instruction ***(Add Ry, 0(Rb))*** replaces

Load Rx, 0(Rb)

Add Ry, Ry, Rx

However, this only works if ***Rx*** is not used elsewhere before being overwritten. This can occur in a case when ***Rx*** is used multiple times before being overwritten. For example:

Load R1, 0(Rb)

Add R2, R2, R1

Sub R3, R3, R1

If we replace ***Load + Add*** with ***Add, R2, 0(Rb)***, we lose ***R1*** for the subtraction, making the optimization invalid. The optimization only works when the loaded register is used immediately and only once before being overwritten

D1: Are modern RISC processors different from the original RISC of the 80s

The early RISC designs of the 1980s was characterized by a small, highly optimized instruction set to achieve faster execution and efficiency. Modern RISC processors have definitely evolved from the original RISC architecture, but they still stick to the core design principles.

The advancements in modern RISC processors was in response to the increased need for improved performance and energy efficiency. The increase in instruction count does not necessarily make them CISC though, because the fundamental principles of RISC are still followed. They include:

1. **Memory Access Instructions (Load-Store Architecture)**

   Modern RISC processors have a strict load-store model where memory access is limited to dedicated load and store instructions.

   CISC architecture allows memory operations to be carried out within arithmetic and logical instructions. For example:

   In RISC, addition using a memory operand requires 2 separate instructions

   LDR R1, [R2]   ; Load value from memory into R1

   ADD R3, R1, R4 ; Perform addition with R1 and R4

   In CISC, a processor like x86 can allow direct addition with the memory operand

   ADD R3, [R2]   ; Direct memory operand in addition

2. **Fixed and simple Instruction encoding**

   Original RISC architectures used fixed-length instruction formats, typically 32-bit. Modern RISC processors have introduced variable-length encodings, but they have retained a level of simplicity by using predictable encoding patterns that help with pipelining and instruction decoding.

3. **Register-Oriented Instructions**

   RISC processors use a large number of general-purpose registers and rely on register-to-register operations rather than memory operands.

   Modern RISC designs still follow this principle with architectures like ARM using register-based instructions extensively

4. **Simple data types and instruction semantics**

   Early RISC instruction sets supported only basic integer and float operations, avoiding complex instructions like string manipulation. Modern RISC instruction sets now include vector instructions, but they regularity in encoding which keeps the instruction set manageable

5. **Efficient pipeline and decoding**

   One of the biggest features that RISC architecture offered was pipelining. Since each instruction being executed was taking the same time, pipelining was possible. This is where one instruction is being executed, the next instruction is being decoded and another instruction is being fetched, all in one clock cycle.

   Modern RISC processors still retain this simple decoding and execution stages, unlike the multi-instruction sequences found in CISC

D2: What level should we measure the complexity of an ISA?

The complexity of an Instruction Set Architecture should ideally be measure at the software (architectural) level rather than the hardware (microarchitectural) level. This is because:

1. **Software and compilers target the ISA**

   The complexity of writing and compiling code for an instruction set is determined by what the software sees, not by how the hardware executes it internally

2. **Backwards compatibility and legacy constraints**

   Modern Intel x86 processors maintain full backwards compatibility with older x86 code. This means that they still support variable-length instructions, memory-to-memory operations and complex addressing nodes

If we classified ISAs based on how they were implemented internally, then x86 architecture could change from CISC to RISC over time, which will contradict the fact that the ISA has not changed from the software's perspective

3. **Microarchitectures can change without changing the ISA**

   Internal implementation details such as decoding x86 instructions into RISC-like operations help optimize the performance but are not part of the ISA itself

   Two different architectures like Intel's Core or AMD's Ryzen can execute the same x86 instructions differently, but if they execute the same instructions, the ISA still remains CISC

As a result, the latest Intel processors still remain CISC processors even though they have RISC-like execution. The processors still support complex instructions, variable-length encoding, and memory operations. They decode x86 instructions into fixed-length RISC-like operations but that does not change the fact that software sees them as x86 CISC instructions.