**Computer organization homework 2**

**Mohammad Shughri 110510196**

**Q1)** To understand the need behind creating RISC and CISC, we need to look back to the 1950s. Most computers could not exceed more than two registers due to limited hardware capacities that were huge and complex and would be considered poorly designed in our modern age. For instance, 18-bit Lincoln labs had only four instructions and two registers.

These designs were mainly used for basic tools such as addition or subtraction operations.

By the late 1960s, the hardware became less expensive and smaller than it used to be, which gave the designers new capabilities to improve the CPU design. Even though the number of instructions increased to 256, there were many issues in the CPU design. Designers tried to fix the issues using microcode, which broke down the complex instructions into simpler instructions. However, since the system only ran one at a time, the system had to wait for each instruction to finish in order for a new one to start.

As technology has advanced, the CISC approach seemed outdated and not reliable for the future. This was caused by multiple reasons, such as the usage of high-level languages. Even though the system had many instructions, only a few were being used by most programs. It was clear that a change must be made to improve the utilization of the CPUs. Therefore, chip designers created the RISC approach, which was the opposite of the CISC approach. It had a large register set, supported high-level languages, had minimal instruction sets and did not have microcode architecture.

**Q2)**

|  |  |  |
| --- | --- | --- |
|  | RISC | CISC |
| Number and types of instructions in them | Number: Only uses a small set of instructions, and the fixed format is 32 bits. Usually, the number of instructions is less than 100.  Type: the instructions are based on the register—for example, Sparc. | Number: uses a large set of instructions and the works with uneven format from 16-64 bits.  Type: the instructions are not based on the register—for example, Intel x86. |
| Addressing Modes | It Supports simple addressing formats. Moreover, it only allows displacement and base addressing—for example, Microchip PIC processors. | Supports multiple formats for specifying operands. Which can have many displacement, base, and index registers—for example, intel 8051. |
| Data Types | MIPS is a Byte word and half word. For example, Atmel’s AVR. | CISC has more data types such as  byte (8 bits), word (2 bytes), dword (4 bytes), qword (8 bytes). For example, Motorola 68000. |
| Memory Organization | It uses separate hardware to implement instructions since it does not have a memory unit.  Memory to memory.  It can make calculations without the need for external memory.  For example, PowerPC. | It has a memory unit that can process complex instructions.  Register to register.  To make calculations, CISC requires an external memory.  For example, VAX. |
| Registers | It requires a more significant number of registers. For example, Arm processors. | It requires a smaller number of registers. For example, DEC. |
| Virtual Memory Support | MIPS uses a virtual memory size of 2^32 = 4GB. However, most of the memory is not used because the testbin only requires 1.2MB of virtual memory to run—for example, Arm processors. | X86 uses a virtual memory size of 2^48 = 256TB. However, most of the memory is not used because the testbin only requires 1.2MB of virtual memory to run—for example, Intel x86. |

**Q3)**

To tell which architecture will be more efficient, we have to consider the device that we are using and the scenario it is in. Some machines use RISC to execute one instruction per clock. Other machines use CISC, which can have more complex instructions than one cycle to execute. However, in our modern-day, RISC is much better than CISC since it supports high-level languages and is more efficient in utilizing cycles per instruction. In conclusion, CISC is more efficient if we want to use instructions per program. On the other hand, RISC is more efficient if we want to use cycles per instruction. Most modern hardware uses RISC as its microprocessor architecture.

Examples of RISC machines are SPARC, PowerPC, Atmel’s AVR, the Microchip PIC processors, Arm processors, RISC-V.

Examples of CISC machines are Motorola 68000, DEC VAX, PDP-11, several generations of the Intel x86, and 8051.

**Q4)**

In X86 architecture, a translation layer is mainly used to convert a code sequence into a code that runs on the target architecture and caches the resulting sequence. This method can be done using a translator that converts an executable file of code and translates it to run the code on the target architecture without actually running the code. Using this method is considered very complex because some parts of the code cannot be reached. An example of a translator is, universal superoptimizer, which can execute many target pairs with high performance and low price.

**References**

Binary translation. (2020, November 08). Retrieved December 19, 2020, from https://en.wikipedia.org/wiki/Binary\_translation

Aishwaryaagarwal2. (2019, August 22). Difference between RISC and CISC processor: Set 2. Retrieved December 19, 2020, from https://www.geeksforgeeks.org/difference-between-risc-and-cisc-processor-set-2/?ref=rp

Thornton, S., Says, S., Says, S., & Says, J. (2018, January 09). RISC vs. CISC Architectures: Which one is better? Retrieved December 19, 2020, from https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/

Says:, C., Says:, T., Says:, M., Says:, C., Says:, A., Says:, A., . . . Says:, P. (2020, November 09). Difference Between RISC and CISC Architecture and Their Applications. Retrieved December 19, 2020, from https://www.elprocus.com/difference-between-risc-and-cisc-architecture/

OSDEV. (2019, June 11). Historical Notes on CISC and RISC. Retrieved December 19, 2020, from https://wiki.osdev.org/Historical\_Notes\_on\_CISC\_and\_RISC

Wikipedia. (2020, November 08). Binary translation. Retrieved December 19, 2020, from https://en.wikipedia.org/wiki/Binary\_translation