Unit 1 Reflections

This week I have learnt about the architecture and functionality of a computer. As a software developer it was interesting to see what low level aspects have parrels at a high level and what aspects are abstracted away.

Many of the control structures provided by high level programming languages map directly to low level arithmetic and logic operations provided by the CPU. The AND, OR and many bitwise operations can be directly executed by CPU. This direct mapping allows for the operations to execute efficiently.

The RISC vs CISC CPU architecture competition is an interesting case of how, as other technologies have evolved the demands for the CPU have changed. CISC architecture aims to allow the execution of complex instructions written in high level languages directly. RISC focuses on executing a minimal instruction set efficiently.

My immediate thought while reading about the CISC design principle was “Keep It Simple Stupid”, while this is good design advice, in case there is more to it.

KISS emphasizes "Make everything as simple as possible but not simpler". In the 1970s CPU designers found that the design of CISC processors had become difficult. This is also what tends to happen when software developers do not apply KISS. As a result of this CPU design complexity increased and it became harder and harder to optimize this type of design. RISCs more limited set of operations was easier to optimize.

In reality, it is not as clear cut. Researching this further I found that early RISC designs required the developers to write more efficient code. In doing so, this meant the development process was made more difficult for developers. Therefore, CISC processors allowed developers to produce, shorter, simpler code by reducing the burden of optimization. The CISC CPUs took on the responsibility of breaking apart complex instructions for the developer. CISC CPUs where also better suited for machines with slower RAM as less communication was need due to the CPU storing more data in its own registers.

As compliers became more efficient, they then began to handle this optimization on the developer's behalf. With the compilers optimizing the code, RISC architecture no longer added any additional complexity. And therefore, became popular again, due to the hardware being easier to optimize. Nowadays modern RISC and CISC architectures are closer aligned in their design.

It appears that the early adoption of CISC architecture benefited software developers at the cost of more complex CPU design development. I think this was a good decision. Whereas with compliers now handling optimization complexity, the simpler RISC design allows for better instruction optimization on RISC architecture, making more suitable for modern, power limited, mobile devices.

Additional reading material:

https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/#:~:text=RISC%2Dbased%20machines%20execute%20one,than%20one%20cycle%20to%20execute.&text=The%20CISC%20architecture%20can%20execute,at%20once%2C%20directly%20upon%20memory.

https://cs.stackexchange.com/questions/269/why-would-anyone-want-cisc/284 https://www.interaction-design.org/literature/article/kiss-keep-it-simple-stupid-a-design-principle