Cameron Nottingham

Dr. Bralick

Computer Organization

23 November 2020

Final Exam

1. Instruction Set Architecture has impact on the processor’s microarchitecture. Explain that impact — include thoughts on how the reverse (microarchitecture impacts ISA) is also true.

The design and implementation of the microarchitecture of a processor depends on its instruction set architecture. The instruction set architecture lays out the operations that computer support without defining exactly how these operations are implemented. The microarchitecture consists of the parts of the processor that interconnect to implement the ISA. The microarchitecture is the layer in which the operations defined by the instruction set architecture are implemented in the physical hardware of the system. Instructions within the ISA may require the processor to find words in memory, retrieve them, and then storing the result of the operation back into memory (Tanenbaum). These operations take time and space and it is the microarchitecture that decides how this time and space is handles. To make these operations as efficient as possible, the microarchitecture must be designed with the ISA in mind.

The microarchitecture must be designed within the context of the instruction set architecture because the microarchitecture must have all the available pieces to implement the functions in the ISA. Therefore, it must be aware of what instructions the ISA consists of so that it can set its data and control paths in the proper manner. It is important to remember that different microarchitectures can support the same instruction set architecture. This allows a program written in the chose ISA to run consistently and smoothly across many different machines. Since the ISA layer is a layer of abstraction above the microarchitecture, it is much easier to design an instruction set architecture first and then create differing microarchitectures to implement the specified instruction set. This has the added benefit of creating cross compatibility of instruction set architectures across varying microarchitectures. This allows companies to cut costs and then pass those savings down to the consumer.

Now, this does not mean that the design of an ISA is a free for all. The designers of an ISA have to be cognizant of the microarchitecture when designing the ISA. First, the instructions that the ISA contains must be able to be implemented efficiently in the microarchitecture. The ISA design team can have an instruction that performs a function X but if that X cannot be implemented cleanly and reasonably in microarchitecture, the microarchitecture design team will not be pleased. The ISA designers must also be aware of how to efficiently take advantage of the microarchitecture available. The designers of ISA must generally know what microarchitecture controls are available to them and how they affect the speed of the processes involved. Microarchitecture is dependent on the physical chip present on the system, and the ISA must support microarchitectures across generations. This is to increase backwards compatibility and to prevent the need for a hardware change every time a new ISA is released. Backwards compatibility is a major benefit to consumers and is a priority of many companies. Therefore, instruction set architectures should designed with backwards compatibility of the microarchitecture in mind.

1. Some people have argued that with increasing capacity of memory storage per chip, virtual memory is an idea whose time has passed and they expect to see it dropped from future computers. Do you agree or disagree? Why or why not?

One of the most important developments in early computing was the introduction of the concept of virtual memory. Virtual memory allows an abstraction of the actual resources that are available on the system. It frees the programmer from the annoyances of bookkeeping and allows a program to treat the system as if it has more memory than it does. With the advent of cheaper and higher capacity memory, some have called for an elimination of the concept of virtual memory due to the operating system overhead that is required to manage it. However, despite the increase in memory capacity and the greater availability of computer memory, virtual memory still has its place in computing. Beyond providing the illusion of more memory than is actually present, virtual memory allows for an increase in memory security and frees individual applications from having to manage sharing a limited physical address space.

The most obvious advantage of virtual memory is giving programs the ability to use more memory than is physically present. As memory availability has increased, so have the programs that need excessive amounts of memory. In addition, the price of mass storage has decreased faster than the price of memory (hblock.net). Programs that deal with machine learning, artificial intelligence, or another data intensive task often require resource far beyond what the computer had. These programs still need the concept of virtual memory even with large amount of available physical memory. After all, if there was no concept of virtual memory, then this memory-hungry program would crash out when it exceeded the amount of physical memory in the system or the programmer would have to split up the program into many overlays. A crash is undesirable and the amount of effort and time that would be required to manually build bookkeeping into the programming would be excessive (Tanenbaum). While having a virtual space for programs that do not need it does cause additional runtimes do to page table lookups, this time is worth it in order to serve the programs that do need it.

In addition to providing programs access to more memory than physical present, virtual memory also increases the security of memory because it isolates the memory used by each program. The virtual address space allows the memory manager of the operating system to provide each program with its own space of memory isolated from other programs. This allows programs to be loaded into physical memory when needed and have their spaces isolated within the virtual address table. This keeps programs from interfering with each other and makes it harder for processes to overwrite each other’s memory, increasing reliability and security.

The virtual memory model also allows each program to view itself as if it inhabited a single contiguous block of memory. This has the benefit of the program not needing to manage the physical address space by itself. This allows multiple programs to share the physical address space without having to worry about managing the memory themselves. Thus, they can carry on as if they are in a contiguous block even if their pages are scattered across the actual address space. While this may lead to programmers being lazy and using more memory than they need, the benefits outweigh the costs.

1. The principle of "locality of reference" (both spatial and temporal) was an empirical observation made a half-century ago - long before object-oriented programming. Caches exist to take advantage of this phenomenon. Given that programming styles have evolved, are caches still useful? Why or why not?

Caches allow smaller amounts of frequently accessed data to be stored in faster memory channels than ordinary data. The advantage of cached data is that it has very low latency—it does not take much time for data to be accessed by the CPU. A lot of caching is based on the principle of locality of reference—both temporal and spatial. Temporal locality of reference is the idea that if a location in memory is accessed then it will be accessed in the near future. Spatial locality of refence goes a step further. It asserts that if a location in memory is accessed then it is likely that memory locations around it will be accessed soon too. Thus, it is beneficial to bring both the data word being requested and the data round it into the cache to speed up future requests for data. This idea works best for data that is stored contiguously or the structure of the program being run is neatly laid out in memory. However, with the advent of object-oriented programming and virtual memory, it is becoming rarer for programs to be stored linearly or for the data of a program to be laid out in contiguous chunks. Thus, the principle of locality of reference begins to fail. Because of this, there is a growing question about the usefulness of caches Despite the development of many new styles of programming and memory allocation, caches are still useful to the modern system because they still provide inherent speed benefits.

To combat the decrease in locality, most modern systems implement a cache hierarchy and have increased the size of their cache lines. The cache hierarchy allows for the most used, most local data to be stored in the caches closest to the CPU with less used, less local data stored in the caches further away. The cache closes to the CPU, the L1 cache, is the fastest and the smallest. Subsequent caches, such as the L2 and L3 caches, allow for more data to be stored, but at higher latencies. The implementation of a cache hierarchy allows the system to prioritize bits of information based on its use, rather than having a single cache that can become jumbled.

Caches can also provide great benefits to the system beyond data retrieval for programs. System resources that the operating system needs to function can be brought into the caches to speed up system response time by not taking up precious time on the data bus. In addition, most modern systems operate a split cache, where the caches for data and instructions are kept physically separate. The advantage of this kind of cache design is that it allows both instruction and data fetches to occur at the same time. Instructions and data are accessed in different ways and at different times, so by having a cache for both, the overall latency for both can be reduced. In addition, since the split cache allows for instructions to be caches separately, it doesn’t matter if the data is highly localized for there to be a speed boost in instruction access time. So, while the benefits of the data cache may decrease if it has poor temporal locality, there will still be performance benefits because of the instruction cache.

1. Redcode Warrior Commenting

;redcode-94

;name DoTheSplits

;author N/A

;strategy Scanner-type warrior that scans for other warriors

;strategy when another warrior is found, use splits to distract it

;strategy Once the opposing warrior is stunned, kill it with DAT

ORG 1

;This is the scanning routine

;It looks for other warriorss

SUB.F $ 11, $ 1 ;Takes the A and B operands from the dat and subtracts from the compare

CMP.I $ 125, $ 119 ;Looks for addresses six apart and skips th next instruction if equal

SLT.A # 18, $ -1 ;If the value 18 is less than the A field of the previous instruction, skip

DJN.F $ -3, < -273 ;Jump back to the beginning if nothing has been found yet

;This moves splits into the warrior found

;This causes it to waste time executing useless splits

MOV.AB # 8, $ 2 ;Moves 8 into the B field of the DJN.B 2 addresses ahead

MOV.I $ 4, > -4 ;Move the split into the address pointed to by the B field of the CMP.I

DJN.B $ -1, # 0 ;Move back 1 if the B field of this operand is 0 after decrementing it

;After moving all the splits start moving DAT

;This is to kill the stunned warrior

SUB.AB # 8, $ -6 ;Subtracts 8 from the original compare

JMN.B $ -8, $ -8 ;If the first line (the SUB.F) is not zero, jump back there

SPL.A # 0, $ 0 ;The SPL to be moved

MOV.I $ 1, < -4 ;Now that the SPL have been placed, start moving DAT to kill

DAT.F < -42, < -42 ;The DAT to be moved

References

Structured Computer Organization, Tanenbaum

https://hblok.net/blog/posts/2017/12/17/historical-cost-of-computer-memory-and-storage-4/