COMPUTER ARCHITECTURE REVIEW

ARDEN RASMUSSEN

MAY 9, 2019

1. Processor Architecture

1.1. **Data Alignment.** Primitive objects of K bytes must have an address that is a multiple of K.

K	Types
1	`char`
2	`short`
4	`int`,`float`
8	`long`,`double`,`char *`

Structs must satisfy 8 byte alignment, so we add additional bytes to meet this requirement.

1.2. The Y86-64 Instruction Set Architecture.

1.2.1. *Programmer-Visible State.*

Number	Register
0	`rax`
1	`rcx`
2	`rdx`
3	`rbx`
4	`rsp`
5	`rbp`
6	`rsi`
7	`rdi`
8	`r8`
9	`r9`
A	`r10`
B	`r11`
C	`r12`
D	`r13`
E	`r14`
F	`NULL`

1.2.2. *Y86-64 Instructions.*

Instruction	Bytes			
`halt`	00			
`nop`	10			
`rrmovq rA, rB`	20	rA	rB	
`irmovq V, rB`	30	F	rB	V
`rmmovq rA, D(rB)`	40	rA	rB	D
`mrmovq D(rB), rA`	50	rA	rB	D
`OPq rA, rB`	6fn	rA	rB	
`jXX Dest`	7fn	Dest		
`cmovXX rA, rB`	2fn	rA	rB	
`call Dest`	80	Dest		
`ret`	90			
`pushq rA`	A0	rA	F	
`popq rA`	B0	rA	F	

Operations	
`addq`	60
`subq`	61
`andq`	62
`xorq`	63

Branches	
`jmp`	70
`jle`	71
`jl`	72
`je`	73
`jne`	74
`jge`	75
`jg`	76

Moves	
`rrmovq`	20
`cmovle`	21
`cmovl`	22
`cmove`	23
`cmovne`	24
`cmovge`	25
`cmovg`	26

1.3. **Logic Design and the Hardware Control language HCL.**

1.3.1. *Logic Gates.* There are three basic boolean logic gates, `AND`, `OR`, and `NOT`. HCL expressions are written using C operations.

Gates	Expression
`AND`	`&&`
`OR`	`||`
`NOT`	`!`

Multiple logic gates can be combined into a larger gate, and then further combined to compute any process.

1.4. **Sequential Y86-64 Implementations.**

1.4.1. *Organizing Processing into Stages.*

Fetch: THe fetch stage reads the bytes of an instruction from memory, using the program counter as the memory address.

Decode: The decode stages reads up to two operands from teh register file, giving values `valA` and `valB`.

Execute: In the execution stage, the arithmetic/logic is preformed.

Memory: The memory stages may write data to memory, or it may read data from memory.

Write back: The write-back stage writes up to two results to the register file.

PC update: The PC is set to the address of the next instruction.

1.5. **General Principles of Pipelining.** Pipelining allows for processes to be paralleled, and proceed in unison with other processes.

The clock cycle of a pipelined system, will be the delay of the longest stage, plus the delay of the `Reg` stage(20ps). The delay will be the clock cycle multiplied by the number of stages. This means that more stages will not necessarily be faster.

1.6. **Pipelined Y86-64 Implementations.** Hazards can occurs, when an operation requires data that is still being computed by a previous operation. This can be resolved by stalling. This is just waiting for the previous operation to complete before executing the new one. It can also be better avoided by forwarding, this is when the previous operation completes its execution, its write back stage directly writes it to the current operation. This means that there still may be a need for some stalling, but less of it is necessary.

1.6.1. *Performance.*

$$\begin{aligned} ClockCycle &= \sum Stages + PipelineRegister \\ Latency &= Pipelines \cdot ClockCycle \\ Throughput &= \frac{1000}{ClockCycle} \end{aligned}$$

2. The Memory Hierarchy

2.1. **Locality.** Locality is the concept of referencing data that is spatially or temporally near by. By improving locality, performance is often improved due to caching. The smaller the loop body and the greater the number of loop iterations, the better the locality.

2.2. **The Memory Hierarchy.** The lower the level, the faster and smaller the cache is. The larger the level, the larger space the cache is, but it is also much slower.

Level	Name
L0	Regs
L1	L1 cache (SRAM)
L2	L2 cache (SRAM)
L3	L3 cache (SRAM)
L4	Main memory (DRAM)
L5	Local secondary storage (local disks)
L6	Remote secondary storage (distributed file systems, Web servers)

2.3. **Cache Memories.**

2.3.1. *Generic Cache Memory Organization.* A cache is made of sets, where each set has a number of lines. The lines are indexed by a tag, and has a valid bit. For each line there is a cache block of data actually stored there. Caches are usually mapped to directly by memory address. This is such that the first t bits are the tag, the next s bits are the set bits and the final b bits denote the block offset.

2.3.2. *Direct-Mapped Caches.* There caches only have one line, and the set directly corresponds to the value in the memory address.

2.3.3. *Set Associative Caches.* These caches have multiple lines for each set. This way each set can store two caches misses based on the memory address.

2.3.4. *Fully Associative Caches.* Fully associative cache consists of a single set that contains all the cache lines.

Having larger block size, allows for better spacial locality, but mean that there will be a smaller number of cache lines. Higher associativity (lines per set) can increase the hit time, because it must check more and more tags in the set.

2.4. **Cache Parameters.**

Parameter	Description
$S = 2^s$	Number of sets
E	Number of lines per set
$B = 2^b$	Block size(bytes)
$m = \log_2(M)$	Number of physical address bits
$M = 2^m$	Maximum number of unique memory addresses
$s = \log_2(S)$	Number of set index bits
$b = \log_2(B)$	Number of block offset bits
$t = m - (s + b)$	Number of tag bits
$C = B \times E \times S$	Cache size(bytes), not including overhead

3. Virtual Memory

3.1. **Address Spaces.** Virtual memory, is a method of storing little used blocks of data on the disk to free up more space on the physical faster memory.

3.2. **VM as a Tool for Caching.** Virtual pages are chunks of data that are in one of three states

Unallocated: Pages that have not yet been allocated by the VM system. These pages have no data

Cached: Allocated pages that are currently cached in physical memory.

Uncached: Allocated pages that are not cached in physical memory.

The pages can be cached, and there will be page hits and misses.

3.3. **Dynamic Memory Allocation.** Dynamic memory allocation grows the heap with calls to a C allocator, such a malloc. Allocators allocate memory only in the heap, and the align the blocks of memory for better access. A good free function, will Coalesce freed blocks, so if two freed blocks are adjacent to one another, then they will be joined into a larger block.