CPU Summary

What is pipelined execution?

- Pipelined execution is a technique that enables microprocessor designers to increase the speed at which a processor operates, thereby decreasing the amount of time that the processor takes to execute a program.
    - Chassis example:
        - Build the chassis
        - Drop the engine into the chassis
        - Put the doors, a hood, and coverings on the chassis
        - Attach the wheels
        - Paint the SUV
    - We don't need to speed up the amount of time spent on each stage (assuming one person is doing all the work). We just need to have more people doing more jobs in parallel.
    - It takes time for the pipeline to be filled and therefore take advantage of the entire pipeline. Increasing pipeline breadth allows for increasing a processor's instruction completion rate and therefore runs programs faster.
- Each processor has 4 basic instructions:
    - Fetch – Fetch the instruction from code storage
    - Decode – Decode the instruction.
    - Execute – Execute the instruction.
    - Write – Write the results of the instruction back to the register file.
- A clock cycle is the maximum time it takes to run an instruction through all each of these four phases. For example, if each phase takes 4 nanosecond, the clock period is 4ns and the instruction completion rate is 0.25 per nanosecond.
    - Pipelining does **not** speed up instruction execution time, each instruction still takes 4 nanoseconds. <u>It does increase program execution time by increasing the number of instructions finished per unit time.</u>
        - A four-stage pipeline yields a fourfold speedup in the completion rate.
    - Since each pipeline stage must take one clock cycle, the clock pulse that coordinates all stages must take the amount of time equivalent to the slowest stage in the pipeline.
- Instruction throughput is the number of instructions completed each clock cycle.
    - For pipelined and non-pipelined processors, the maximum number is one instruction per cycle.
    - Higher instruction throughput rate leads to higher instruction completion rate.

What are pipeline stalls?

- There are three states of a pipeline:
    - Full
    - Being filled
    - Stalling

- Instruction stalling reduces average instruction throughput.
- For a 3GHz processor that has a clock cycle time of a fraction of a nanosecond, a 100ns main memory access fetch translates into a few thousand clock cycles of bubbles.
- Instruction latency is the amount of clock cycles it takes an instruction to pass through the pipeline.

What is superscalar execution?

- Superscalar refers to the notion of duplicating the amount of execution units on a single chip to allow for more work to be done in parallel. Allows one to increase the maximum theoretical instruction throughput for a pipelined processor multipipeline execution.
- Alongside SSE came a paradigm shift in execution units.
  - The ALU, which is usually synonymous with integer execution unit, now has been broken into an IU (integer execution unit) and an FPU (floating point execution unit).
  - The front-end of a CPU (fetch and decode) is broken down into two execution units, those being i) the load-store unit (LSU) and ii) the branch execution unit (BEU).
    - The BEU is responsible for conditional and unconditional branch instructions. It determines whether to replace the program counter with the branch target.

What are some superscalar execution hazards?

There are three types of hazards that can cause bubbles in the pipeline:

1. Data hazards – occur when the result of one instruction depends on the other.
   a. This prevents both from being executed in parallel (for example, at the same time by two different ALUs).
      i. This is alleviated via a concept called *forwarding* where the output of the first instruction on ALU 1 is placed back on the input port of ALU 2, bypassing the write stage.
   b. A similar case can be seen when one instruction requires a read from register A and the second requires a write to register A. The read must happen before the second instruction's write.
      i. The CPU alleviates this issue with register renaming, which takes advantage of the additional hardware general purpose registers that may not be present in the programming model. It maps the programmer-visible registers to the internal machine-visible registers; thus writing the second add to a temporary register before then writing it to register A.
         1. The more data is held close to registers, the less loads and stores (since the result of computations can be accessed immediately), and the less memory subsystem traffic.
2. Structural hazards – A systematic limit on the number of data access paths we have to a register file, which contains a contiguous array of registers. Since we can't have a single path between an ALU and each register, the coordination is done by a register file, shortening the amount of wires from N to 2 for reads (2 read ports) and 1 for writes (1 write port).
3. Control hazards – Also known as branch hazards, these arise at a conditional branch when we need to decide which instruction to fetch next. Another problem occurs that once the next

instruction is loaded into the program counter, it takes an additional number of cycles to fetch the next instruction from storage.

    a. Depending on where the instruction is, it make take a few cycles to thousands of cycles to fetch the next instruction.

What is instruction set architecture?

- Instruction set architecture is generally referred to as microarchitecture. It is the set of instructions that allow software to interface with hardware such that the same piece of software can interact with hardware from more than one machine (e.g. x86 instruction set).
    - IBM first introduces this with IBM System/360 which meant that the information the programmer needed to know to program the machine was abstracted from the actual hardware implementation of the machine.
        - This technical innovation was made possible by a microcode engine. It's like a CPU in a CPU. It's job is to translate a particular instruction into a series of commands that controls the internal parts of the chip.
    - This stands in stark contrast to consoles. For example, you cannot play PS2 games on the N64.

What is the branch prediction unit?

- The branch unit works closely with the program counter, steering it to different sections of the code stream, allowing it to place different addresses in the instruction register.
- There are two types of branches:
    - Direct – have the branch target specified explicitly in the instruction (e.g. goto statement)
    - Indirect – Have to load the branch target from a register and have multiple potential targets
- The branch unit is made up of two parts:
    1. The branch execution unit and the branch prediction unit.
        a. An instruction in the instruction register that is decoded by the control unit may be a conditional branch instruction. This instruction is sent to the BU which determines whether or not the branch is taken. If taken, it needs to retrieve the starting address of the next section of code. This address is called the branch target.
    2. Old processors sat idle while the branch condition was to be evaluated. Modern processors use a technique called speculative execution, which involves making an education guess on the direction the branc h will ultimately take, beginning execution at the new branch target before the branch's condition is actually evaluated.
        a. This is used to reduce the amount of delays associated with evaluating branches (which would otherwise introduce bubbles into the pipeline).
        b. These speculatively executed branches cannot commit their write results back to register files until the branch condition is evaluated.

            i. If there is a misevaluation, all instructions currently executing in the pipeline must be flushed.
        c. There are two types of branch prediction:
            i. Static prediction – relies on the assumption that the majority of back pointing branches occur in the context of loops, and that the loop exit condition will be false for the majority of the repeat code
                1. This is fast because it doesn't involve any lookups or statistical heuristics
            ii. Dynamic prediction – involves two types of tables:
                1. The branch history table – creates an entry for each conditional branch that the branch unit has encountered in its last few cycles. For example, it includes some bits that indicate the likelihood that the branch wil be taken based on its past history.
                2. Branch target buffer - when a branch is executed speculatively, it needs to know exactly where in memory the branch is pointing (the branch target).
        d. Usually, static prediction will be used when there isn't a corresponding entry in the branch history table for a conditional branch.
    3. Some processors perform branch folding. The idea is that when an instruction is sitting in the reservation station / instruction queue may already have enough information about it in the branch unit to resolve to its branch target. Hence, we replace the conditional instruction with the branch target.
        a. This eliminates the branch from the code stream.

What is out of order execution?

- Newly decoded instructions are inserted into a special buffer that collects instructions waiting to be executed (called the reservation station). The processor's dynamic scheduling logic examines instructions in the buffer and issues these instructions from the buffer to the execution unit at the most opportune time. This logic has quite a bit of freedom, even allowing instructions to be reordered. Though, there may be limitations to this, like one instruction may be waiting on data from a yet-to-be-executed instruction.
- This out of order execution has two phases, the first I've already detailed (issue buffer).
- The primary purpose of this policy (dynamic execution), alongside the issue buffer, is to allow the processor to squeeze out bubbles of the pipeline prior to the execution phase.
    - This depends on the ability for the front-end (the instruction fetcher) being able to fill up the issue buffer faster than the execution units can consume instructions. Otherwise there will be nothing in the issue buffer and hence nothing to fill in bubbles.
- The second phase is the completion phase. Once an instruction is done executing, its result is placed in a rename register and the instruction itself is placed in a second 'completion' buffer before results are written back to the register file in program order.
    - These results are committed in program order and so they mask the illusion of non-sequential execution. This means that no instruction can be committed until all the instructions ahead of it in the queue were committed.

What are vector execution units?

- A vector execution unit handles SIMD instructions (single instruction, multiple data).
    - SIMD relies on data parallelism, where elements in a dataset can be processed in parallel. This often happens with data of a uniform type, like media files.
        1. Here the same instruction is applied to multiple units at once.
    - A simple CPU operates on scalars one at a time. A superscalar CPU operates on multiple scalars at once, but performs different operations on each instruction. A vector processor lines up a whole row of scalars, all of the same type, and operates on them in parallel as a single unit.

What is 64-bit computing?

- When you hear of 64-bits, what is really being communicated is the number of bits that each processor's general purpose register can hold.
    - This doesn't mean that all data needs to be 64 bits, it just means that this is the widest.

Understanding caching

Level	Access Time	Typical Size	Technology	Managed By
Registers	1-3ns	1kb	Custom CMOS	Compiler
Level 1 Cache	2-8ns	8kb-128kb	SRAM	Hardware
Level 2 Cache	5-12ns	0.5mb to 8mb	SRAM	Hardware
Main Memory	10-60ns	64mb – 1Gb	DRAM	Operating System
Hard Disk	3 Million ns – 10 Million ns	20GB +	Magnetic	Operating System / User

- L1 is a subset of L2. This is part of the cache hierarchy, where we start with the page file on the hard disk and go all the way down to the registers on the CPU.
- The data contained on one level is usually mirrored in the level below it, for a piece of data in L1, there are usually copies of that same data in L2 and main memory.
- When a load instruction is issued to load a piece of data into one of the processor's registers, various hierarchies of the memory subsystem are first scanned. If L1 / L2 cache does not have it, this is a cache miss. If main memory doesn't have it, we are in trouble, since it means this piece of data has to be paged from the hard disk.

Cache misses

- There are three types of cache misses:
    - Compulsory miss – happens when the desired data was never in the cache and therefore must be paged for the first time in a program's execution. It's one of the misses that can't be avoided.
    - Conflict misses – a miss due to a block frame collision, where a given cache block cannot occupy a block frame due to its direct-mapping associated from being occupied by another cache block.

    - Capacity miss – A miss as a result of the eviction of a block that would have otherwise been hit due to L1 cache capacity constraints (bringing in new data)

Locality of reference

Caching works because of a simple property exhibited by all types of code and data: locality of reference.

There are two types of references:

1. Special locality – A fancy word for the rule that states 'If the CPU needs an item from memory at any given moment, its likely to need that item's neighbors next'
    a. Spacial locality of data – Relative chunks of data tend to clump up together in memory. Think of the data needs of a CPU reading minute 1:45 of an iTunes song. It'll most likely need minutes 1:45 and 1:46.
    b. Spacial locality of code – Most well-written code tries to avoid jumps and branches so that the processor can execute through large contiguous blocks of code uninterrupted.
2. Temporal locality – If an item in memory was accessed once, it will most likely be accessed again in the future.
    a. Temporal locality implies special locality, but not vice versa. That is to say that data that is often reused is always related, but data that is related is not always used.

Cache organization

- When a piece of information is requested, the actual piece that is requested is called the critical word. Its sur rounding group of bytes is called the cache line or cache block.
- When a cache block is moved from ram to L1 cache its placed in a block frame. Cache designers can pick different schemes to govern which ram blocks can be stored in which of the cache's block frames. This is called cache placement policy.
    - When a CPU requests information from RAM, it needs to answer these questions:
        1. Is the needed block in the cache (cache hit or miss)?
        2. Where is it in the cache (if cache hit)
        3. What is the location of the critical word within the block (if cache hit)
    - These questions are answered via a tag that each block frame in the cache contains. The tag is stored in a special kind of memory called tag ram that is a very fast form of SRAM. Since these tags need to be searched, they can add unwanted access latency to the cache. The way you use these tags to map ram blocks to block frames will help decrease access latency.
        1. Fully associative mapping – Ram blocks can be stored in any available block frame.
            - With this scheme, you have to check every block frame for your critical word.
        2. Direct mapping – Each block frame can only cache a certain subset of the blocks in main memory.
            - This reduces the potential number of search spaces for any single block in cache and increases search efficiency.

    - If a CPU needs to load blocks from the same working set, it will need to swap out these blocks should the mapped set become full. For example, if it wants to load blocks 0 to 3 and 8 to 11, and blocks 1 and 9 need to go in the same frame, they can't occupy the same frame at once. This is the downside. What's even more egregious is that half the cache is going unused.
        - This is called a collision (the cpu would like to store multiple blocks but cant because they all require the same frame)
3. Four-way set associative mapping – The cache is devided into a set of 4 block frames each. In a cache with 3 sets, for example, only one third of the cache is searched. And within that search, you search only a single flour-block set.
    - **Note**: Increasing the associativity increases the number of tags that must be checkes, increasing cache latency, but also decreasing the chance for a conflict miss.
    - The placement formula, which enables you to compute the set in which an arbitrary block of memory should be stored in an n-way associative cache is *blockaddress* mod *numberOfSetsInCache*

Cache Eviction Policy

- Cache eviction policies dictate which blocks currently in cache will be replaced when a new block is fetched. Most caches use a pseudo-LRU (least recently used) algorithm that approxtimates true LRU by marking cache blocks are dirty the longer they sit unused in the cache.
- What is bound to happen over time is that some blocks that aren't all that dirty get replaced by newer blocks, just because there isn't enough room in cache to accommodate the entire working set.
    - This may later lead to a miss when a block containing needed data has been evicted due to cache capacity. This is called a capacity miss.

Cache Block Sizes

- The larger the cache blocks are, the more likely you are to wind up with wasted space (cache pollution), as fewer bytes of the working set relative to the amount of bytes you draw from memory will be present.
    - Larger block sizes means fewer sets also means that more cache conflict misses (cache collisions) are likely
    - Large block sizes also eat up memory bus bandwidth and hence increase the amount of time of a fetch.

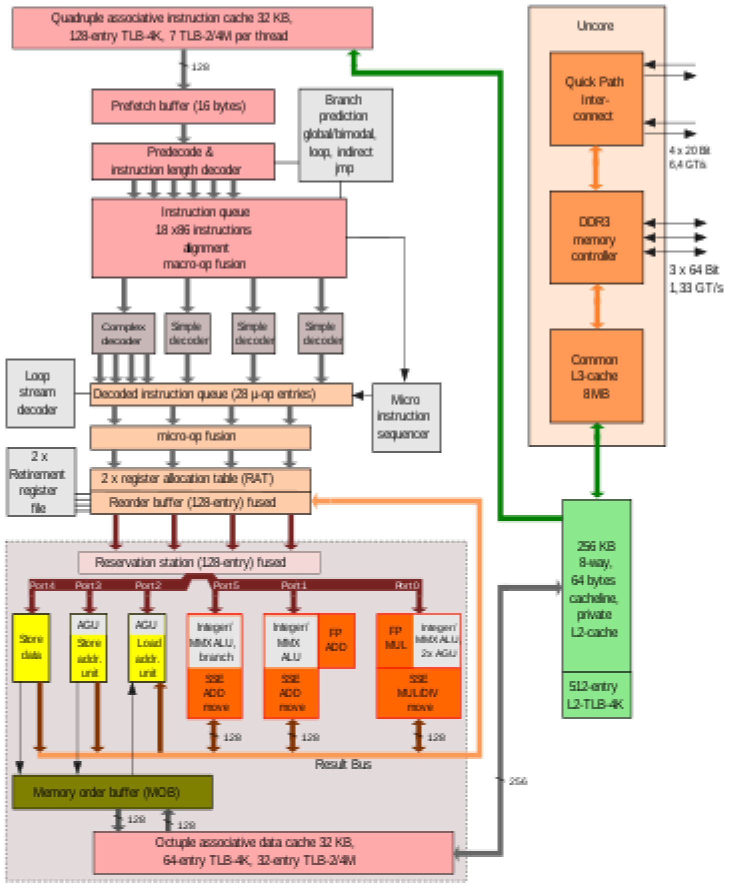
Write Policies

- Once a retrieved piece of data is modified by the CPU, it must be stored (or written back)_ to main memory so that the rest of the system can receive the most up-to-date version of it.
- There are two ways to deal with such writes to memory:

  - Immediate modification (write through) – Immediately update all copies of the data in each level of the cache hierarchy to reflect the latest changes. This ensures all copies of them are current. This is called write-through.
    - This policy is nice for a multiprocessor system where multiple clients of the cache data are reading from memory at once and all need the latest piece of information.
    - On the downside, this can greatly increase memory traffic.
  - Delayed modification (write back) – Changes propagate to the higher levels of the cache only after an updated piece of data is evicted from the lower level.
    - Upside = less memory traffic, downside = harder to keep data in sync, especially in a multiprocessor system.

https://en.wikipedia.org/wiki/Reservation_station

Intel Nehalem microarchitecture



Quadruple associative instruction cache 32 KB, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch buffer (16 bytes)
Branch prediction global/bimodal, loop, indirect jmp
Predecode & instruction length decoder
Instruction queue 18 x86 instructions alignment macro-op fusion
Complex decoder
Simple decoder
Simple decoder
Simple decoder
Loop stream decoder
Decoded instruction queue (28 µ-op entries)
Micro instruction sequencer
micro-op fusion
2 x Retirement register file
2 x register allocation table (RAT)
Reorder buffer (128-entry) fused
Reservation station (128-entry) fused
Port4
Port3
Port2
Port5
Port1
Port0
Store data
AGU
Store addr. unit
AGU
Load addr. unit
Integer/ MMX ALU, branch
SSE ADD move
Integer/ MMX ALU
SSE ADD move
FP ADD
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV move
128
128
128
Result Bus
Memory order buffer (MOB)
128
128
Octuple associative data cache 32 KB, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 memory controller
3 x 64 Bit 1,33 GT/s
Common L3-cache 8 MB
256 KB 8-way, 64 bytes cacheline, private L2-cache
512-entry L2-TLB-4K

GT/s: gigatransfers per second