William Frazee

CDA5106 HW 1

1. (15 points) Investigate and write a short summary for each of the following features developed in Intel processors.

- **Pipelining and hyper-pipelining**
  Pipelining: Processing an instruction is broken down into various stages. Without pipelining, we have to wait for each instruction to run through all of the stages, before feeding the next instruction into the first stage. With pipelining, we can have each stage process a different instruction simultaneously and sequentially. “Pipelining works best with RISC designs”. For more complex operations, they are broken down into uops, especially on x86. There are four pipeline slots available in intel processors. Pipeline slots are optimized for specific application types, and serve to process uops.
  Source: https://techdecoded.intel.io/resources/understanding-the-instruction-pipeline/

  Hyper pipelining: Intel decided that instead of shrinking the die to improve clock time, they needed to make the CPU do less per clock cycle. In order to accomplish this, they increased the number of their pipelining stages. “This 20-stage pipeline is what Intel is calling their Hyper Pipelined Technology.” The downside to this is that when branch prediction goes awry, they suffer a performance hit due to increased pipeline stages meaning increased bubbles. In order to compensate for incurred penaties for mis-predicted branches, they implemented other features.
  Source: https://www.anandtech.com/show/661/3
- **SMP support**
  A multiprocessor approach where two or more identical processors share the same memory, are controlled by the same OS, and have full access to I/O. SMP systems serve to parallelize tasks in the computer. The downside is that since memory is shared, there will be times when different jobs running on different processors need access to the same memory space.
  Source: https://en.wikipedia.org/wiki/Symmetric_multiprocessing
- **MMX and SSE instruction sets**
  MMX: A form of single instruction, multiple data (SIMD) instruction sets that first debuted in Intel processors in the 1990s. SIMDs are optimized to perform the same instruction on multiple pieces of data in parallel; MMX specifically has “eight processor registers... and operations that operate on each of them”. It provides only integer operations.
  Source: https://en.wikipedia.org/wiki/MMX_%28instruction_set%29

  SSE: Similar to MMX, but supports floating-point math.
  Source: https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions
- **Virtualization support**
  Intel CPU virtualization allows for abstraction of the CPU to a virtual machine. This allows the virtual machine to run software as if it was running natively on a dedicated CPU. This allows for faster performance on a number of virtual machine applications.
  Source: https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html
- **TurboBoost**
  This is dynamic overclocking. Essentially, the CPU can boost it’s own clock speed, within certain electrical and thermal limits, for tasks that it knows will be demanding in performance. The processor determines this by keeping track of the workload.
  Source: https://en.wikipedia.org/wiki/Intel_Turbo_Boost

2. (5 points) Floating point instructions improved to run 2X but only 20% of actual instructions are floating point. Calculate the overall speed up after this improvement.

Using Amdahl’s law:

$\text{Speedup}_{\text{overall}} = 1/((1-F)+(F/S))$

Where:
F = fraction of task that the improvement can enhance = .2
S = Best case speedup from the improvement alone = 2

$\text{Speedup}_{\text{overall}}$ = 1/((1-.2)+(.2/2)) = 1.11 times faster.

3. (10 points) Estimate the performance improvement when we upgrade a system component as follow. On a large system, suppose we can upgrade a CPU to make it 75% faster for $10,000 or upgrade its disk drives for $10,000 to make them 200% faster. Processes spend 75% of their time running in the CPU and 25% of their time waiting for disk service. An upgrade of which component would offer the greater benefit for the lesser cost?

Using Amdahl's law:

$\text{Speedup}_{\text{overall}}$ = 1/((1-F)+(F/S))

For the CPU upgrade:
Where:
F = fraction of task that the improvement can enhance = .75
S = Best case speedup from the improvement alone = 1.75

$\text{Speedup}_{\text{overall}}$ = 1/((1-.75)+(.75/1.75)) = 1.47 times faster

For the Disk Drive upgrade:
Where:
F = fraction of task that the improvement can enhance = .25
S = Best case speedup from the improvement alone = 3

$\text{Speedup}_{\text{overall}}$ = 1/((1-.25)+(.25/3)) = 1.20 times faster

The $10k would be better spent upgrading the CPU.

4. (5 points) What is the speedup of the pipeline if 1/5 of the instructions are branches, and 4/5 of those are correctly predicted? Can you give a general formula for a k-stage pipeline? What other information do you need to know?

Let's assume that we have 100 instructions, using our example 5 stage processor (Where our execution stage is in the middle, stage 3).

Calculating our piplined cycles: 1/5 of those are branch instructions, so 20 branch instructions. 4/5 are correctly predicted, which leaves 4 instructions incorrectly predicted, So we would have 8 bubbles to take care of (since there are 2 stages before the execution stage that we would have to scrub when it turns out the prediction is wrong).

1 cycle * 100 instructions + 8 cycles for the bubbles from the incorrectly predicted branches is 108 cycles.

Non pipelined would take 5 stages * 100 instructions = 500 clock cycles.

So, our pipeline is 500/108 = 4.63 times faster than non-pipelined.

So in general:

speedup = k processor stages * instruction count / (instruction count + (number of incorrectly predicted branches * however many stages there are before exec stage aka the number of bubbles))

So we need to know how many stages exist before the execution stage.

5. (10 points) Consider the following multilevel cache hierarchy with their seek times and miss rates:

- **L1-cache, 0.5 ns, 30%**
- **L2-cache, 1.8 ns, 5%**
- **L3-cache, 4.2 ns, 1%**
- **Main memory, 70 ns, 0%**

In this case, the seek times given refer to the total time it takes to both check whether the requested data is available on the current level of hierarchy, and transmit the data to the level above (or to the CPU). Calculate the Average Memory Access Time.

AMAT = Time for a hit + (miss rate * miss penalty)
We just need to expand the miss penalty for each level of cache,
= L1_Seek_Time + L1_Miss_Rate*L1_Miss_Penalty
= L1_Seek_Time + L1_Miss_Rate*(L2_Seek_Time + L2_Miss_Rate*L2_Miss_Penalty)
= L1_Seek_Time + L1_Miss_Rate*(L2_Seek_Time + L2_Miss_Rate*(L3_Seek_Time+L3_Miss_Rate*L3_Seek_Penalty))
= L1_Seek_Time + L1_Miss_Rate*(L2_Seek_Time + L2_Miss_Rate*(L3_Seek_Time+L3_Miss_Rate*(Main_Mem_Seek_Time)))

Replacing with appropriate values...

= 0.5 + 0.3(1.8 + .05(4.2 + .01(70)))
= 1.1135 ns

6. (5 points) Find the Average memory access time for a processor with a 1 ns clock cycle time, a miss rate of 0.02 misses per instruction, a miss penalty of 15 clock cycles, and a cache access time (including hit detection) of 1 clock cycle. Also, assume that the read and write miss penalties are the same and ignore other write stalls.

AMAT = Time for a hit + (miss rate * miss penalty)
AMAT = 1 cycle + (0.02*15)
AMAT = 1.3 cycles*1 ns/cycle
AMAT = 1.3ns