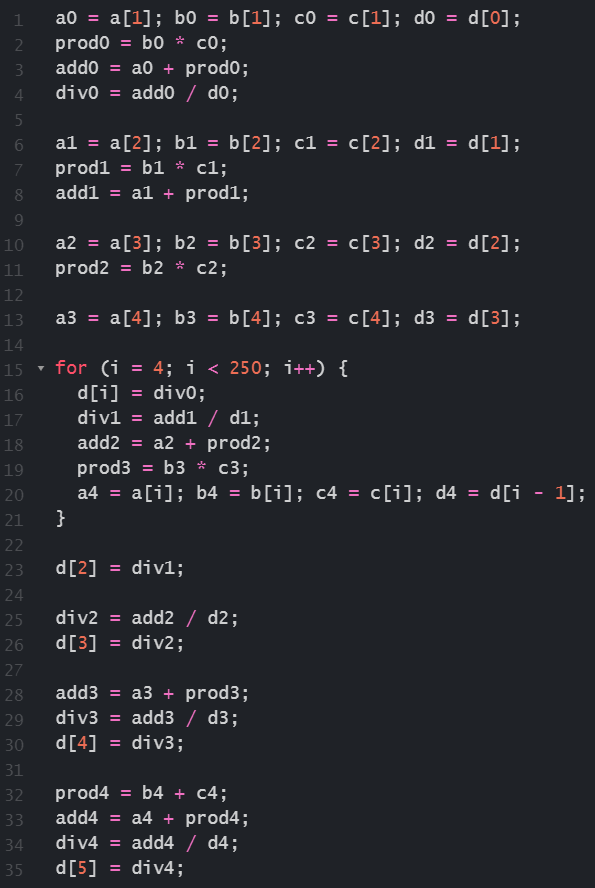
Jared Raiola

4290 HW2

6/30/20

**Section 1**

1. The biggest roadblock in extracting ILP from a program is dependence chains. This is a roadblock because it limits the amount of parallel execution that can be done as code motion needs to be able to maintain its data dependencies. This results in more registers needing to be allocated and more branches to be done when future instructions rely heavily on prior instructions.
2. Loop Unrolling VS Software Pipelining
   1. Loop Unrolling: Transforming an M-iteration loop into a loop with M/N iterations
      1. Pros
         1. Less loop overhead (lowers instruction numbers), less branches
         2. Better scheduling of instructions
         3. Gets rid of small loops, removing branches
         4. Less load latency
      2. Cons
         1. Code is larger (code bloat, more complicated to understand)
         2. N may not be a multiple of M
         3. N may not be known
         4. Exit condition issues in while loop
         5. Needs a lot of unrolling to hide all load latency
         6. Muls are slow and critical
   2. Software Pipelining: Each instruction in the loop body executes operations from different logical iterations of the loop
      1. Break up loop body into operations that can be formed independently
      2. Pros
         1. Less loop overhead
         2. Faster execution than loop unrolling
         3. Less load latency
         4. Removes branching
      3. Cons:
         1. Code gets bigger
         2. Eats up registers
         3. Need to know exit condition early on
         4. Works only for loops
3. Optimized Code below



**Section 2**

1. Bit values:
   1. Offset = 13 bits
   2. VPN = 19 bits
   3. PPN = 11 bits
2. 8-Way Set-Associative TLB will have the best performance because it allows for the most efficient splitting of memory with the lowest access time and highest hit rate. When calculating Average Memory Access Time, it has the lowest value.
3. Combining the 4-Way Set-Associative TLB with the fully associative TLB would result in the lowest average memory access time, causing it to outperform the 8-Way Set-Associative TLB.
4. Size:
   1. Fully Associative:
   2. 8-Way:
   3. 4-Way:

**Section 3**

1. Caches take the ideas behind spatial locality, if a data item is needed now, nearby data is likely to be needed in the near future, and temporal locality, if a data item is needed now, it is likely to be needed again in the near future, and act as a smaller, faster storage mechanism that allows for programs to essentially jump from point to point in deeper memory instead of having to traverse one by one. The main idea is that recently used data is kept in fast memory close to the processor, along with the nearby data for instant lookup, and if the data needed does not fall under the principles of locality, then do a slow lookup.
2. Write vs No-write
   1. A piece of C code that would benefit from write-allocate would be code that requires you to write to the same memory location multiple times. This means that when we miss, the block will still be in the cache for us to access again very quickly.
   2. A piece of C code that would benefit from no-write-allocate would be the opposite, where we’re constantly writing to different locations in memory and it is slower to bring the whole block into the cache every single time we miss.
3. Compare and Contrast
   1. Virtually Indexed Cache:
      1. Pros
         1. Lower latency than physically indexed – no need to check TLB
      2. Con
         1. Must flush cache on process change
         2. Same virtual address in different processes can map to different data
         3. Aliasing issue where several virtual pages can map to the same physical page.
   2. Physically Indexed Cache:
      1. Pros
         1. No aliasing issue
         2. Cache does not need to be flushed upon process change
      2. Cons
         1. TLB lookup and cache access is serialized
4. No, the physically indexed cache will never be accessed at the same latency as the virtually cached index, because the physically indexed cache must always check the TLB before lookup.
5. 140 bits
6. 140 bits
7. Compare Trace vs Victim caches:
   1. Trace cache:
      1. Stores instructions either after decoding or as they are retired
      2. Having the instructions stored and tracked, they will not need to be decoded again and can be just used from the trace cache
   2. Victim:
      1. Recently kicked-out blocks are kept in small cache
      2. If we miss on those blocks we can get them quickly
      3. It’s primarily to protect against conflict misses

**Section 4**

1. The destructive read property of DRAM is the issue found that when a state of a capacitor is read, it is discharged, causing the state to require a write-back to not lose the data. DRAM deals with this issue by reading each row periodically in order to restore the charge to the capacitor.
2. Memory level parallelism is the ability to perform multiple memory operations at the same time. The way DRAM stores and reads memory its through the use of row column pairs of cells, as a bit cell array which reduces overall latency. This allows for quick memory access, loading multiple instructions almost simultaneously.
3. Compare Policies
   1. Open page:
      1. Keep the row open until a conflict
   2. Closed page:
      1. Close the row buffer after access
4. The goal of a scheduling process is to maximize row buffer hit rate in order to maximize DRAM throughput. A memory controller would use the first ready, first come first served scheduling property in order to maximize the amount of commands executed.

**Section 5**

1. SECDED provides both error correction and error detection through the use of putting a parity code on top of base SEC. It’s designed correct a single error using the general hamming code algorithm with a parity code used to track if the number of ones of a code is odd or even. Through both correcting errors and detecting double errors at the same time, SECDED can help in the error removal and error forecasting portions of improving memory reliability.
2. The purpose of RAID is to improve disk performance by having multiple smaller disks perform the role of one big disk.
   1. RAID 0:
      1. Uses striping to improve performance
      2. All disks can work in parallel
      3. Low latency by having a low queuing delay for each disk
      4. No redundancy, meaning if ANY disk in an array fails there will be a problem
      5. Mean time to failure (MTTF) is 1/(1-(1-P)^2) where P is the probability that a drive will die for one hour.
      6. Baseline: P=0.01
         1. 100 hours for one drive
         2. 50.25 hours for 2-drive
         3. 25.38 for 4-drive
   2. RAID 1:
      1. Uses disk mirroring, keeping disks paired up with identical data
         1. An issue is that a write must update BOTH disks
         2. A read, however can read any of the two copies
      2. Improved performance as more reads can happen per unit time.
      3. Writes are not improved
      4. Data is backed up on mirror disk
      5. MTTF is 1/(P^2), but only if we replace failed drives in the same hour.
         1. Baseline: P=0.01
            1. 100 hours for one drive
            2. 10,000 for 2-drive
            3. 100,000,000 hours for 4-drive
   3. RAID 4:
      1. Uses block-interleaved parity where:
         1. One disk is a parity disk
         2. A read can only access the data disk where the data is
         3. A write must update the data block AND its parity block
      2. Can recover from an error on any ONE disk by using parity and other disks to restore lost data
      3. N disks, N-1 data disks and one parity disk
      4. Write performance is significantly worse because all writes must read and then write the parity disk.
   4. RAID 5:
      1. Uses distributed block-interleaved parity:
         1. Similar to RAID 4, but parity blocks are spread out over all disks
         2. All disks share parity block update load
      2. RAID 5 and RAID 4 share the same reliability.
      3. For RAID 5 (and RAID 4) with 4 drives:
         1. Probably that none fails: (1-P)^4
         2. Probably that one fails: 4\*P\*(1-P)^3
         3. Probably that 2 or more fail: 1-[(1-P)^4+4\*P\*(1-P)^3]
      4. Baseline P=0.01:
         1. 100 hours for a single drive
         2. 10,000 hours for 2-drive
         3. 1,689 hours for 4-drive