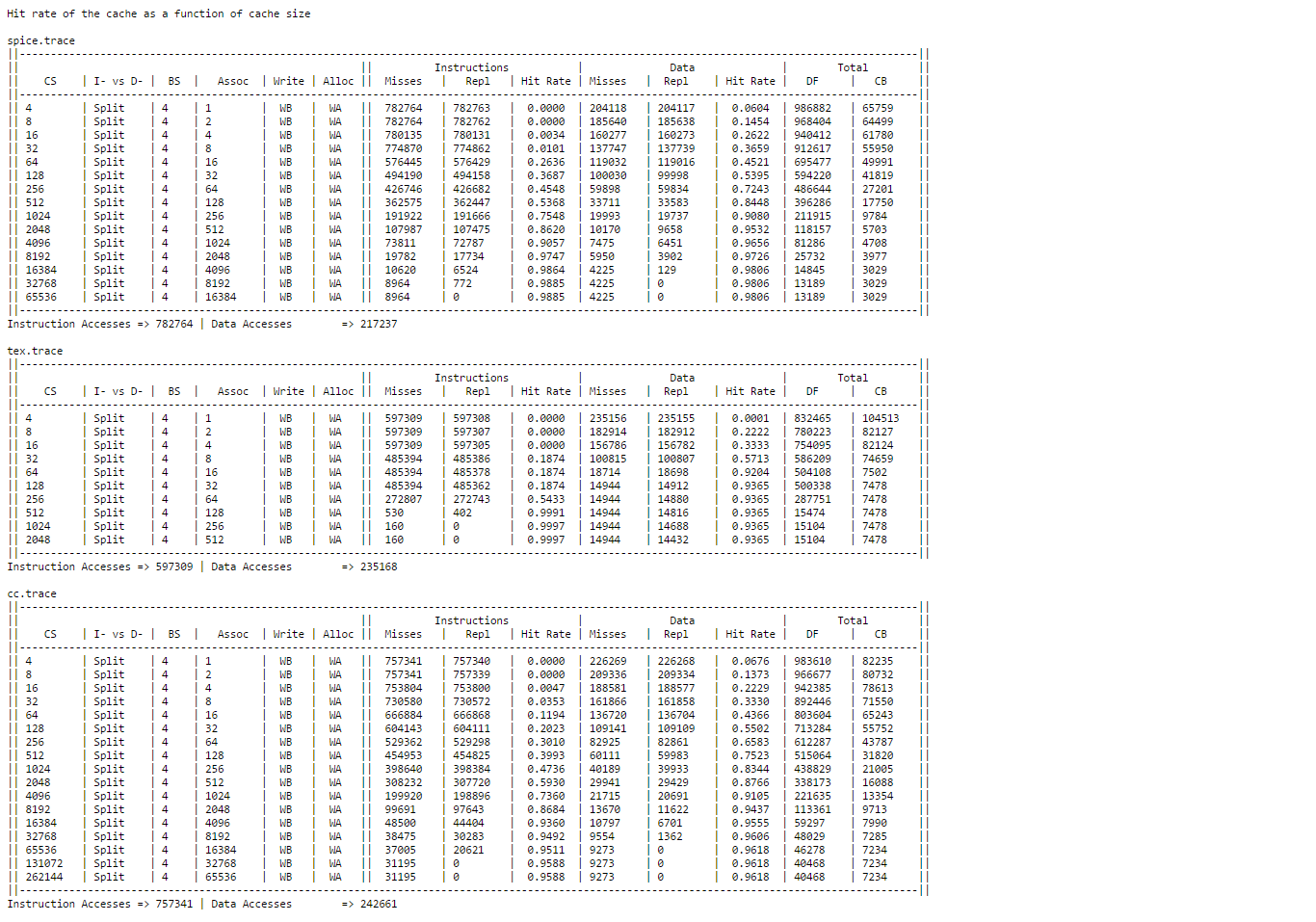
# COP 290: Cache Simulator Report

# *Submitted by: Suyash Agrawal (2015CS10262) Ankesh Gupta (2015CS10435)*

# Analysing Cache Size Effects

## Charts



## Detailed Analysis

1. The focus here is on studying the effect of increasing cache size on hit rate.
2. As seen and intuitive too, increasing cache size increases the hit % and also decreases misses. The larger the Cache size, the lesser is the probability of a conflict arising.
3. But as we keep increasing the block size, it reaches a saturation, upto a point where there are only **compulsory misses** in the Cache. The cache is now, large enough to accommodate all the data accessed during the execution of the program.
4. Here, all the 3 trace files shows same pattern because of aforementioned reasons.
5. Separate data memory and instruction memory behaviour patterns are inline with above observations as both function independently and the misses saturate after a point.

# Analysing Block Size Effects

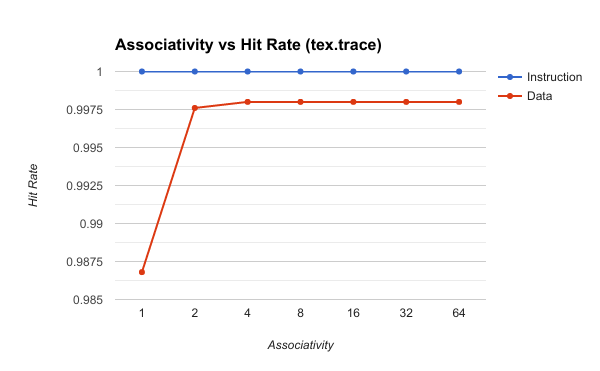
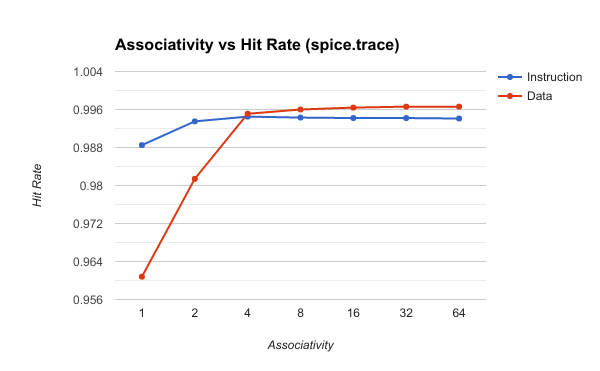
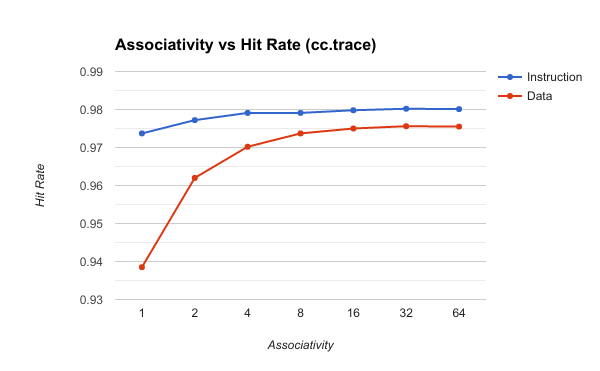
## Charts

## C:\Users\ANKESH GUPTA\Desktop\screencapture-raw-githubusercontent-ozym4nd145-MIPS-Simulator-report-report-tables-block_size-txt-1493230001760.pngDetailed Analysis

1. The focus here is on studying the effect of varying block size on hit rate.
2. As expected, the hit rate follows a **convex pattern** with increasing block size.
3. The initial increase is because small block sizes do not take maximum advantage of **spatial locality.**
4. The hit rate keeps on increasing with increasing block size as more and more spatial locality is exploited.
5. The decrease in the end is because when block size becomes large, there are fewer blocks available giving rise to higher **potential conflicts**.
6. The 2 memories shows different performance here. Increasing size of block in Instruction cache keeps on increasing the hit rate whereas Data Memory shows performance as mentioned above.
7. This is because instruction memory has a continuous access trace (mostly) and thus increasing block size decreases the number of lookup’s in memory. Whole block can be fetched in single access which shows up as improve in performance when compared to word by word transfer.
8. The optimal block size for Data Cache was in range of **64 – 256 bytes** per block whereas Instruction cache showed no such distinctive maxima’s.
9. Hence, we can conclude that Instruction references are much more **ordered and continuous** when compared with Data references.

# Analysing Associativity Effects

## Charts

## C:\Users\ANKESH GUPTA\Desktop\screencapture-raw-githubusercontent-ozym4nd145-MIPS-Simulator-report-report-tables-assoc_num-txt-1493230209477.pngDetailed Analysis

1. Here, the focus is on studying variation of set associativity against hit rate.
2. The pattern is first increasing and then saturates.
3. Highly associative caches have a lower miss rate because of more **flexible placement** of blocks within a set.
4. The fully associative cache are the most performance efficient but constant time search amongst tag bits is too hardware intensive.
5. Varying associativity didn’t improve much on performance of instruction cache whereas significant improvement was seen on performance of data cache.
6. Instruction references poses less conflicts because of ordering in which they are accessed and most miss are **Compulsory Misses.**

# Analysing effect on Memory Traffic

## C:\Users\ANKESH GUPTA\Desktop\screencapture-raw-githubusercontent-ozym4nd145-MIPS-Simulator-report-report-tables-mem_band_wa_wna-txt-1493230397187.png

## Analysing effect of Write back vs Write through

1. Write back has smaller memory traffic then write through.
2. This is because both the policies perform equally on **demand fetches** but write back perform outperforms Write through when studying copies back.
3. This is because Write back keeps both the cache and memory **simultaneously** updated whereas, writes back lazily when it becomes imperative.
4. Generally, write back has smaller memory traffic than write through except cases when:

* Block size of Cache <= 4 bytes (Word Size)
* When a particular index in cache continuously gets written and replaced (because of a conflict).

## Analysing effect of Write allocate vs Write no allocate

1. The aim here is studying effect of Write Allocate and No Write Allocate policies, given Write Back strategy is followed.
2. Here, the performance is mixed and both perform at par with the other when traffic is concerned.
3. When a write miss occurs, and the same address is continuously accessed, write allocate is superior as it would have already bought the data into memory.
4. When we are filling some memory block and need not read it, then bringing it back to memory makes no sense and thus write no allocate reduces traffic by not bringing it back to memory.
5. The above 2 policies are situation dependent because of aforementioned reasons.

