1 Data Dependencies

- **True Dep** = RAW
- **Anti Dep** = WAR
- **Output Dep** = WAW

2 Cache

2.1 Memory Access Time

Hit rate, **Hit time** = cache access time, **miss rate** = 1 - hit rate, **miss penalty** = time to replace block cache + hit time
Average access time = hit rate × hit time + miss rate × miss penalty.

2.2 Write Policy

- **Write-through**, write data both to the cache & main memory. Problem: write will operate at speed of main memory. Solution: write buffer b/w cache and main memory.
- **Write-back** cache, only write to cache, write to main memory only when cache block is replaced (evicted). Problem: wasteful to write back every evicted cache. Solution: Use a "dirty bit" if cache content is changed. Write back only if dirty bit is set.

2.3 Types of Cache Misses

Compulsory misses on first access to a block, **Conflict misses** on collision, **Capacity misses** when blocks are discarded as cache is full

2.4 Handling Cache Misses

Read Miss: load data from memory to cache and then to register.
Write miss:

- **Write-allocate**: load complete block to cache, change the required word in cache, write to main memory (write policy).
- **Write-around**: no loading to cache, write to main memory only

2.5 Direct Mapped Cache – (index, valid, tag)

How to id: tag match with only 1 block.
Cache block size = 2^N bytes, **no of cache blocks** = 2^M
Offset = N bits, **Index** = M bits, **Tag** = $32-(N+M)$ bits

2.6 N-way Set Associative Cache

How to id: tag match for all the blocks within the set.
A block maps to a unique set (each set having n "cache blocks")
Cache block size = 2^N bytes, **no of cache blocks** = $\frac{\text{size of cache}}{\text{size of block}}$,
no of sets = $\frac{\text{no of blocks}}{\text{n in n-way}} = 2^M$, **Offset** = N bits, **Set Index** = M bits

2.7 Fully Associative Cache – capacity miss, no conflict miss

How to id: tag match for all the blocks in the cache.
A memory block can be placed in any location in the cache.
Cache block size = 2^N bytes, **no of cache blocks** = 2^M
Offset = N bits, **Tag** = $32-N$ bits (block number = tag)

2.8 Block Replacement Policy

- **Least Recently Used (LRU)**: Replace the block which has not been accessed for the longest time. For temporal locality. **Problem:** hard to keep track if there are many choices.
- **FIFO**, **Random Replacement**, **Least Frequently Used**

3 Performance

3.1 Execution Time

Avg Cycle/Instruction: **CPI** = $\frac{\text{(CPU time×clock rate)}}{\text{Instruction count}} = \frac{\text{clock cycles}}{\text{instruction count}}$

CPU time = $seconds = instructions \times \frac{cycles}{instruction} \times \frac{seconds}{cycle}$

$CPI = \sum_{k=1}^{n} (CPI_k \times F_k)$ where $F_k = \frac{I_k \text{(Instruction frequency)}}{\text{Instruction count}}$

3.1.1 Amdahl's Law (1967)

Speedup of parallel execution is limited by the fraction of the algo that can't be parallelised

- $0 \le f \le 1$ = sequential fraction (fixed-workload performance)
- $S_p(n) = \frac{T_{best\_seq}(n)}{f \times T_{best\_seq}(n) + \frac{1-f}{p} T_{best\_seq}(n)}$
- Implication: manufacturers are discouraged from making large parallel computers, more research attn shifted towards developing parallelising compilers that reduces f
- **Rebuttal**: in many computing problems, f is not constant, dependent on problem size n. An effective parallel algorithm is $\lim_{n\to\infty} f(n) = 0$, thus speedup $\lim_{n\to\infty} S_p(n) = \frac{p}{1+(p-1)f(n)} = p$, thus Amdahl's law can be circumvented for large problem size

3.1.2 Gustafson's Law (1988)

- Main constraint is exec time, then higher computing power is used to improve accuracy/better result
- If f decreases when n increases, then $S_p(n) \le p$, $\lim_{n\to\infty} S_p(n) = p$
- Let $T(N)$ = exec time on N processors, $ser(N)$ = exec time of seq portion, $par(N)$ = exec time of par portion
- $T(N) = ser(N) + par(N) = 1$
- $T(1) = ser(N) + N \times par(N)$
- Speedup = $\frac{T(1)}{T(N)} = ser(N) + N \times (1 - ser(N)) = N - (N-1) \times ser(N)$