CS252
Graduate Computer Architecture
Lecture 2

Review of Cost, Integrated Circuits, Benchmarks, Moore's Law, & Prerequisite Quiz

January 19, 2001
Prof. David A. Patterson
Computer Science 252
Spring 2001

Review #1/3: Pipelining & Performance

- Just overlap tasks; easy if tasks are independent
- Speed Up ≤ Pipeline Depth; if ideal CPI is 1, then:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \cdot \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

- Hazards limit performance on computers:
  - Structural: need more HW resources
  - Data (RAW,WAR,WAW): need forwarding, compiler scheduling
  - Control: delayed branch, prediction
- Time is measure of performance: latency or throughput
- CPI Law:

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

Review #2/3: Caches

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity.
- Write Policy:
  - Write Through: needs a write buffer.
  - Write Back: control can be complex
- Today CPU time is a function of (ops, cache misses) vs. just f(ops): What does this mean to Compilers, Data structures, Algorithms?

Now, Review of Virtual Memory

Basic Issues in VM System Design

- size of information blocks that are transferred from secondary to main storage (M)
- block of information brought into M, and M is full, then some region of M must be released to make room for the new block --> replacement policy
- which region of M is to hold the new block --> placement policy
- missing item fetched from secondary memory only on the occurrence of a fault --> demand load policy

Paging Organization

virtual and physical address space partitioned into blocks of equal size

page frames

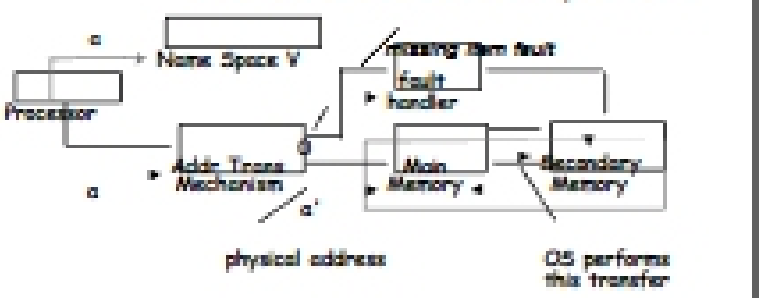
pages

Address Map

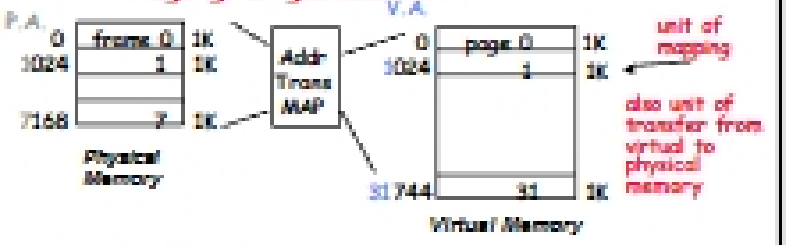
V = {0, 1, . . . , n - 1} virtual address space n > m
M = {0, 1, . . . , m - 1} physical address space

MAP: V --> M ∪ {0} address mapping function

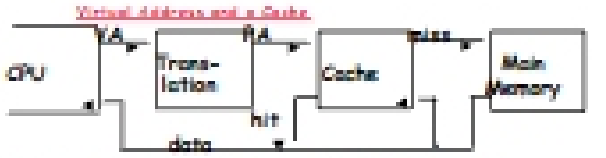
MAP(a) = a' if data at virtual address a is present in physical address a' and a' in M
= 0 if data at virtual address a is not present in M



Paging Organization



Virtual Address and a Cache



It takes an extra memory access to translate VA to PA

This makes cache access very expensive, and this is the "innermost loop" that you want to go as fast as possible

ASIDE: Why access cache with PA at all? VA caches have a problem!

synonym / alias problem: two different virtual addresses map to same physical address => two different cache entries holding data for the same physical address!

for update: must update all cache entries with same physical address or memory becomes inconsistent

determining this requires significant hardware: essentially an associative lookup on the physical address tags to see if you have multiple hits;
or software enforced alias boundary: same lsb of VA & PA > cache size

TLBs

A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is Translation Lookaside Buffer or TLB

Virtual Address	Physical Address	Dirty	Ref	Valid	Access

Really just a cache on the page table mappings

TLB access time comparable to cache access time (much less than main memory access time)

Translation Look-Aside Buffers

Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.

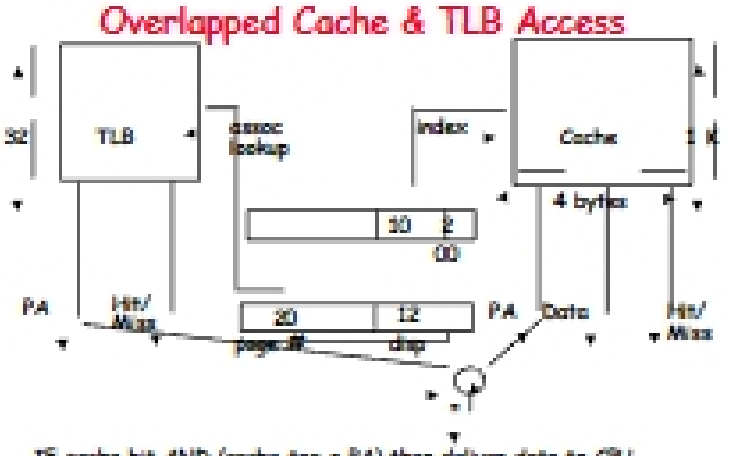
Reducing Translation Time

Machines with TLBs go one step further to reduce # cycles/cache access

They overlap the cache access with the TLB access:

high order bits of the VA are used to look in the TLB while low order bits are used as index into cache

Overlapped Cache & TLB Access

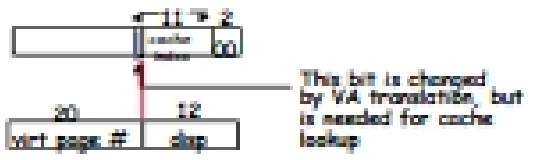


Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache do not change as the result of VA translation

This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:



Solutions:
go to 8K byte page sizes;
go to 2 way set associative cache; or
SW guarantee VA[13]=PA[13]

SPEC: System Performance Evaluation Cooperative

- First Round 1989
  - 10 programs yielding a single number ("SPECmarks")
- Second Round 1992
  - SPECint92 (6 integer programs) and SPECfp92 (14 floating point programs)
    - Compiler Flags unlimited. March 93 of DEC 4000 Model 610:

      spice: unix.c:/def=(sgrev,bsd_bcopy,"bcopy(a,b,c)=memcpy(b,a,c)"
      wave5: /all=(all,dcom=nat) /ag=a/sc=4/sc=200
      nasa7: /noreco/ag=a/sc=4/sc3=200/lc=blas
- Third Round 1995
  - new set of programs: SPECint95 (8 integer programs) and SPECfp95 (10 floating point)
  - "benchmarks useful for 3 years"
  - Single flag setting for all programs: SPECint_base95, SPECfp_base95

SPEC: System Performance Evaluation Cooperative

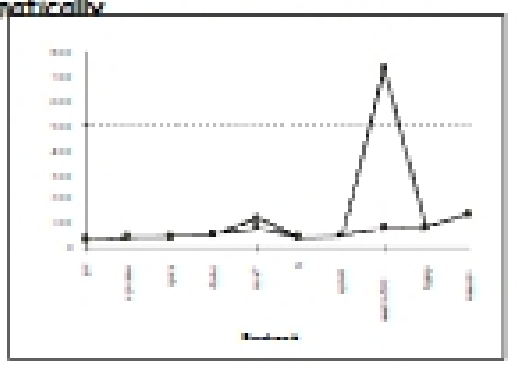
- Fourth Round 2000: SPEC CPU2000
  - 12 Integer
  - 14 Floating Point
  - 2 choices on compilation: "aggressive" (SPECint2000, SPECfp2000); "conservative" (SPECint_base2000,SPECfp_base): flags same for all programs, no more than 4 flags, same compiler for conservative, can change for aggressive
  - multiple data sets so that can train compiler if trying to collect data for input to compiler to improve optimization

How to Summarize Performance

- Arithmetic mean (weighted arithmetic mean) tracks execution time:
  $\Sigma(T_i)/n$ or $\Sigma(W_i*T_i)$
- Harmonic mean (weighted harmonic mean) of rates (e.g., MFLOPS) tracks execution time:
  $n/\Sigma(1/R_i)$ or $n/\Sigma(W_i/R_i)$
- Normalized execution time is handy for scaling performance (e.g., X times faster than SPARCstation 10)
- But do not take the arithmetic mean of normalized execution time, use the geometric mean:
  $(\Pi T_i / N_i)^{1/n}$

SPEC First Round

- One program: 99% of time in single line of code
- New front-end compiler could improve dramatically



Impact of Means on SPECmark89 for IBM 550

	Ratio to VAX:		Time:		Weighted Time:	
Program	Before	After	Before	After	Before	After
gcc	30	29	49	51	6.91	9.22
espresso	35	34	65	67	7.84	7.86
spice	47	47	510	510	5.69	5.69
doduc	46	49	41	38	5.81	5.45
nasa7	78	144	258	140	3.43	1.86
li	34	34	183	183	7.86	7.86
eqntott	40	40	26	26	5.66	5.66
matrix300	78	730	58	6	3.43	0.37
fpppp	90	87	34	35	2.97	3.07
tomcatv	33	138	20	19	2.81	1.94
Mean	54	72	124	108	54.42	49.09
	Geometric		Arithmetic		Weighted Arith	
	Ratio	1.33	Ratio	1.16	Ratio	1.09