Memory Hierarchy Design

Principle of Locality
- Temporal Locality
- Spatial Locality

Smaller Hardware is faster

Price/Performance Consideration (Amdahl's Law)

: cache blocks

: page

Memory Hierarchy

Memory Level	Price'99 per MBytes	Speed access time	Size
cache (SRAM)	Expensive $50-200	Fast (8-35ns)	Small
primary memory (DRAM)	M $25-50	M (60-120ns)	M
disk	Cheap $1-2	Slow (8-20ms)	Large

A memory access is said to have a **hit** (***miss***) in a memory level, if the data is found (*can not be found*) in the level.

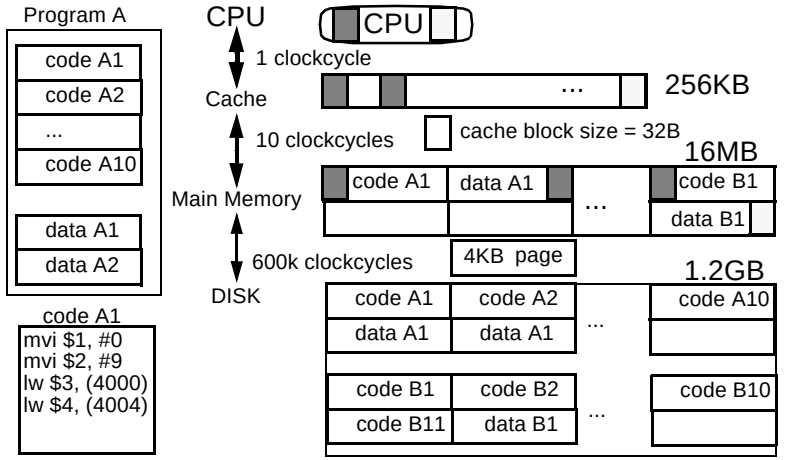
Hit rate (***Miss rate***)—is the fraction of memory accesses (*not*) found in the level.

Hit time—the time to access data in a memory level including the time to decide if the access is a hit or miss.

Miss penalty—the time to replace a block in a level with the corresponding block from the level below, plus the time to deliver the block to CPU
=the time to access the first word on a miss+the transfer time of remaining words

Access Time **Transfer Time**

A Memory Access Scenario in A system with Cache and Virtual Memory (Paging)

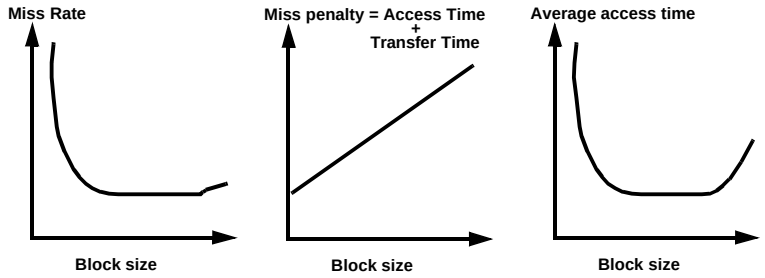


Evaluating Performance of a Memory Hierarchy

Average Memory Access Time is a better measure than the Miss rate.

Average Memory Access Time = Hit time + Miss rate * Miss penalty

Relationship between block size and Average access time, Miss penalty, Miss rate



Goal of Memory Hierarchy:
to reduce execution time, not the no. of misses

⇓

Computer designers favor a block size with the lowest average access time rather than the lowest miss rate.

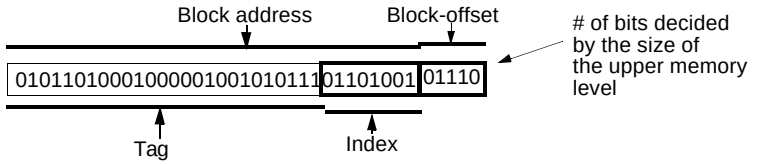
Four Questions for Classifying Memory Hierarchies

Q1: Where to place a block in the upper memory level? (***Block placement***)

Q2: How to find a block in a memory level? (***Block identification***)

Q3: Which block should be replaced on a miss? (***Block replacement***)

Q4: What happens on a write? (***Write strategy***)



Caches

The memory level between CPU and main memory.

Cache: a safe place for hiding or storing things.

Webster's New World Dictionary of the American Language, Second College Edition (1976)

Block (line) size	**4-128 bytes**
Hit time	**1-4 clock cycles (normally 1)**
Miss penalty	**8-32 clock cycle**
(Access time)	**(6-10 clock cycles)**
(Transfer time)	**(2-22 clock cycles)**
Miss rate	**1%-20%**
Cache size	**1KB-256KB**

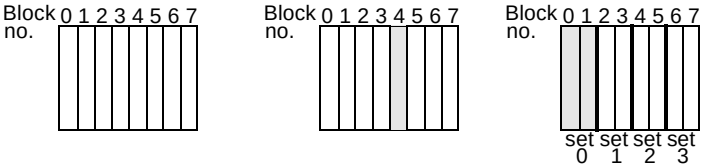
Q1: Where to place a block in a cache? (*Block placement*)

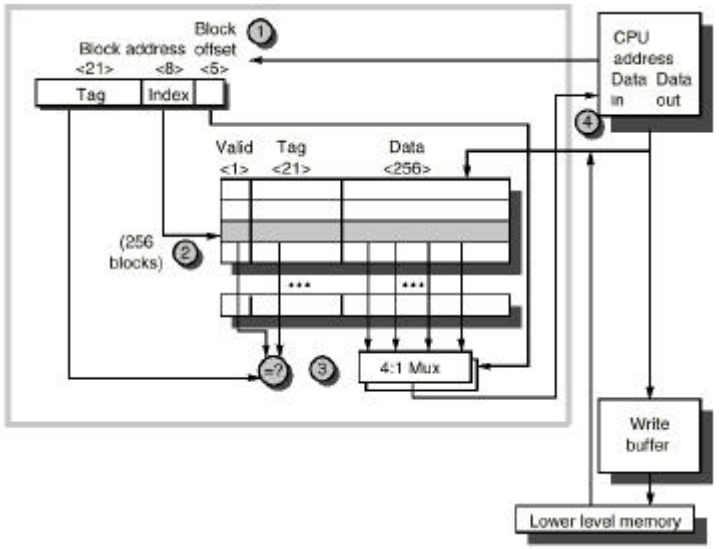
Direct mapped cache—a fixed place for a block to appear in a cache.
e.g., the location = (block-frame address) modulo (no. of blocks in cache).
Fully Associative cache—a block can be placed anywhere in the cache.
Set Associative cache—a block can be placed in a restricted set of places.
If there are n blocks in a set, the cache placement is called *n-way set associative*.

Fully associative: block 12 can go anywhere

Direct mapped: block 12 can go only into block 4 (12 mod 8)

2-way Set associative: block 12 can go anywhere in set 0 (12 mod 4)





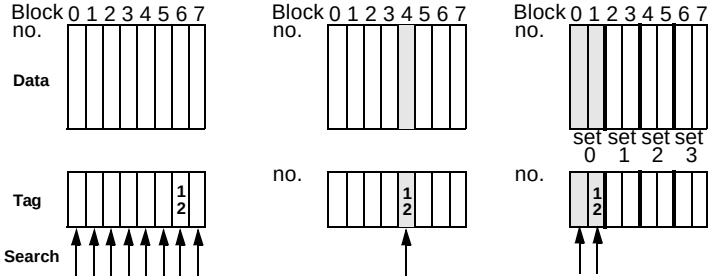
Q2: How to find a block in a cache? (*Block identification*)

Caches include an address tag (which gives part of block address) on each block.
A valid bit is attached to a tag to indicate if the information in the block is valid.

Fully associative: block 12 can go anywhere

Direct mapped: block 12 can go only into block 4 (12 mod 8)

2-way Set associative: block 12 can go anywhere in set 0 (12 mod 4)



Q3: Which block should be replaced on a miss? (*Block replacement*)

For the direct-mapped cache, this is easy since only one block is replaced.

For the fully-associative and set-associative cache, there are two strategies:

- Random
- Least-recently used (LRU)—replace the block that has not been access for a long time. (Principle of temporal locality)

Table 1:The LRU blocks for a sequence of block-frame addresses. Assume there are 4 blocks.

Block-frame address		**3**	**2**	**1**	**0**	**0**	**2**	**3**	**1**	**3**	**0**
LRU block number	**0**	**0**	**0**	**0**	**3**	**3**	**3**	**1**	**0**	**0**	**2**

Figure 5.4

Q4: What happens on a write? (*Write strategy*)

Reads dominate cache accesses. All instructions accesses are reads.

Write policies (options when writing to the cache):

- Write–through—The information is written to both the cache and main mem.
- Write–back—The information is only written to the cache; the modified cache block is written to main memory only when it is replaced.

A block in a write–back cache can be either **clean** or **dirty**, depending on whether the block content is the same as that in main memory.

For the write back–cache,

- uses less memory bandwidth, since multiple writes within a block only requires one write to main memory.
- a read miss (which causes a block to be replaced and therefore) may result in writes to main memory.

For the write–through cache,

- a read miss does not result in writes to main memory.
- it is easier to implement.
- the main memory has the most current copy of the data.

Dealing with Write Miss

There are two options (whether to bring the block into the cache):

- Write–allocate—The block is loaded into the cache, followed by the write-hit actions above.
- No–write–allocate—The block is modified in the main memory and not loaded into the cache.

In general, the write–back caches use write–allocate.

$\Rightarrow$ hoping that there are subsequent writes to the same block.

The write–through caches often use no–write–allocate.

$\Rightarrow$ since the subsequent writes also go to the main memory.

Dealing with CPU write stall

CPU has to wait for the writes to complete during write-through.

This can be solved by having a write buffer and let CPU to continue while the memory is updated using data in write buffer.

Write merging: allow multiple writes to the write buffer to be merged into a single entry to be transferred to the lower level memory.

Figure 5.6

Miss rate vs. block size

Figure 5.7

Split Caches vs. Unified Caches

Assume the percentage of instruction references is about 75%.

Why instruction-only caches have lower miss rates than data-only caches?

Example: Which cache performs better, 32KB split cache(16KB instruction cache+16KB data cache) or 32 KB unified cache? Assume a hit takes one clock cycles, a miss costs 50 clock cycles. A load or store (data cache) hit takes two clock cycles on a unified cache.

Ans: Use average memory access time formula.

Average Memory Access Time (AMAT) = Hit time + Miss rate * Miss penalty.

$AMAT_{split}=AMAT_{split,instruction\ cache} + AMAT_{split,\ data\ cache}$

=75%x(1+0.64%x50) + 25%x(1+6.47%x50)

=(75%x1.32) + (25%x4.235) = 0.990+1.059 = 2.05

$AMAT_{unified}$ = 75%x(1+1.99%x50) + 25%x(2+1.99%x50)

=(75%x1.995) + (25%x2.995) = 1.496+0.749 = 2.24

The split cache, which offers two memory ports per clock cycle, performs better.

Cache Performance

CPU time=(CPU-execution clock cycles+Memory-stall clock cycles)*cycleTime.

CPU time=IC*($CPI_{execution}$+(Memory-stall clock cycles/IC))*cycleTime.
IC: instruction count.

CPU time=IC*($CPI_{execution}$+MAPI*MissRate*MissPenalty)*cycleTime.
where MAPI: Memory Accesses Per Instruction.

Example 1. VAX cache miss penalty is 6 clock cycles. All instructions normally take 8.5 clock cycles (ignoring memory stall). Miss rate=11%. Average 3 memory references per instruction. What is the impact on performance when the behavior of the cache is included?

Answer:
$CPU\ time_{consider\ cache}$=IC*(8.5+3.0*11%*6)*cycleTime=IC*10.5*cycleTime.
$CPU\ time_{did\ not\ consider\ cache}$=IC*8.5*cycleTime.

Example 2. Assume a machIne with lower CPI, CPI=1.5. Cache miss penalty is 10 clock cycles. Miss rate 11%. Average 1.4 memory reference per instruction.

Answer:
$CPU\ time_{consider\ cache}$=IC*(1.5+1.4*11%*10)*cycleTime=IC*3.0*cycleTime.
$CPU\ time_{did\ not\ consider\ cache}$=IC*1.5*cycleTime. → impact on this machine larger

Cache Block Placement Trade-off
Is 2-way associative better than direct-mapped?

2-way associative cache requires extra logic to select the block in the set⇒ longer hit time ⇒ longer CPU clock cycle time.

Will the advantage in lower miss rate offset the slower hit time?

Example (page 387). $CPI_{execution}$=2, DataCacheSize=64KB, Clockcycletime=2ns, Miss Penalty=70ns (35CPUClockcycles), MemoryAccesPerInstuction=1.3.

	CPU**with direct-mapped cache**	CPU**with 2-way associative cache**
ClockCycleTime	2ns	2*1.1=2.2ns
Miss rate(Fig.5.9)	0.014	0.010
Average Mem Access Time	2+0.014*70	2.2+0.010*70
	=2.98ns	=2.90ns

CPU time=IC*($CPI_{execution}$*ClockCycleTime
+MemoryAccesPerInstuction*MissRate*MissPenalty*ClockCycleTime)

CPU time	IC*(2.0*2+1.3*0.014*70)	IC*(2.0*2.2+1.3*0.010*70)
	=IC*5.27	=IC*5.31

Since the CPU time is the bottom line evaluation metric and direct-mapped cache is simpler to build, in this case the direct-mapped cache is preferred.

Improving Cache Performance

Caches can be improved by:

- Reducing miss rate
- Reducing miss penalty
- Reducing hit time

Often there are related, improving in one area may impact the performance in the other areas.

Reducing Cache Misses

Three basic types of cache misses:

- Compulsory - The first access to a block not in the cache. (first reference misses, cold start misses).
- Capacity - since the cache cannot contain all the blocks of a program, some blocks will be replaced and later retrieved.
- Conflict - when too many blocks try to load into its set, some blocks will be replaced and later retrieved.

Figure 5.10

Figure 5.9

Reducing miss rate by Larger Block Size

Larger blocks takes advantage of spatial locality.

Larger blocks increase the miss penalty and reduce the number of blocks.

Figures 5.11 & 5.12 & 5.13

Select the block size that minimizes AMAT

Assume the memory system takes 40 cycles overhead and then delivers 16 bytes every 2 clock cycles. Figure 5.13 shows the results on AMAT.

Example 394: $AMAT_{blocksize=16B,\ cachesize=1KB}$ = 1+(15.05%x42)=7.321 clock cycles.

Figure 5.13

Reducing miss rate by Higher Associativities

Assume the clockcycletime will be stretched to 1.10, 1.12, and 1.14 times of 1-way clockcycletime, for 2-way, 4-way, and 8-way set associative cache.

Using Figure 5.9 miss rate, Figure 5.14 shows the AMAT for set associativities trade-off.

Figure 5.14.

Reducing Miss Rate by Victim Cache

- contains blocks that are discarded from a cache miss
- checked on a miss, if matched, victim block and cache block are swapped.
- A four entry victim cache removed 20% to 95% of the conflict misses in a 4KB direct-mapped data cache.

Figure 5.15.

Reducing miss rate by Hardware Prefetching of Instructions and Data

CPU contains steams buffers (e.g., each 32 byte long). Each time an instruction/ data (e.g.,4 bytes) is fetched from cache, the whole block is loaded from cache into the stream buffers. (e.g., one stream buffer miss followed by 7 consecutive hits if no branch instructions in between.)

For an instruction/data fetch, CPU first looks to see if it is in the stream buffer.

Jouppi [1990] found that a single instruction buffer would catch 15% to 25% of the misses from a 4-KB direct-mapped instruction cache with 16-byte block.

Reducing miss rate by Compiler-Controlled Prefetching

An alternative to hardware prefetching is to let compiler generate prefetch instructions to request for the data before they are needed.

- Register prefetch - load the value into a register.
- Cache prefetch - load the value into the cache.

A ***faulting*** prefetch instruction can cause virtual address faults or protection violation.

A ***nonfaulting*** prefetch instruction does not cause virtual address faults or protection violation, it simply turns into no-ops.

The goal of a nonfaulting cache prefetch design is to overlap the CPU execution with the prefetching of data.

Loops are key targets for compiler-controlled prefetching.

Example (page 403): Assume 8-KB direct-mapped data cache with 16-byte blocks, it is a write-back cache with write allocate. Each element of a or b are double precision floating point number (8 bytes), 3rwos and 100 columns for a and 3 rows and 101 columns for b. How many misses will be generated for the following code?

```
for (i=0; i<3; i++)
    for (j=0; j<100; j++)
        a[i][j] = b[j][0]*b[j+1][0];
```


Ans: In C language, elements in a 2 dimension array are arranged in column major order, e.g., a[0][0] followed by a[0][1]...a[0][99]a[1][0]...a[1][99]a[2][0]... Unlike Fortran which arranged in row major order.

Among the 300 accesses to a, 150 (with even column index) will be miss, the other 150 will be hit.

Assume that cache is large enough without having conflict or capacity misses, the 2*100*3 accesses to b will only have 101 misses. Since after b[0][0]...b[100][0] are loaded into the cache, the remaining accesses to them will be cache hit.

Totally we have 150+101=251 cache misses.

Here is the code that uses nonfaulting cache prefetching:

```
for (j=0; j<100; j++) {          /* due to long miss penalty=50cycles */
    prefetch(b[j+7][0]); /*need to prefetch 7 iterations in advance */
    prefetch(a[0][j+7]); /* actually we only need to fetch 50 times for a*/
    a[0][j] = b[j][0]*b[j+1][0];}
for (i=1; i<3; i++)
    for (j=0; j<100; j++) {
        prefetch(a[i][j+7]);
        a[i][j] = b[j][0]*b[j+1][0];}
```

Only a[i][0,2,4,6] and b[0-6][0] causing cache misses. 3*4+7=19.

We trade 251-19=232 caches with 400 prefetch instructions!

Compiled-Controlled Prefetching

Example (page 404): Calculate the time saved in the above example.

Ignore instruction cache misses.

Assume prefetch can overlap with each other and with cache misses.

The original loop takes 7 clock cycles per iteration.

The first prefetch loop takes 9 clock cycles (one cycle per prefetch instruction)

The second prefetch loop takes 8 clock cycles per iteration.

A miss takes 50 clock cycles.

Ans:

$Time_{original\ loop}$ = instruction execution+cache misses penalty
= 300x7+251x50=14650 clock cycles.

$Time_{prefetching\ loops}$ = instruction execution+cache misses penalty
=(100x9+200x8) + (11x50+8x50)=900+1600+550+400=3450 cycles.

Speedup = 14650/3450 = 4.2.

The prefetch code is 4.2 times faster.

The accesses of array a benefit from spatial locality.

The accesses of array b benefit from temporal locality.

Reducing Miss Rate by Compiler Optimizations 1. Merging Arrays

This technique reduces misses by improving spatial locality.

If we access multiple arrays in the same dimension with the same index at the same time, these accesses can interfere with each other and generate conflict misses.

Solution-> combine elements to form a single compound array. A single cache block can contain the desired elements

```
/*before */                    /* after */
int val[SIZE];                 struct merge {
int key[SIZE];                     int val;
                                   int key;};
for (i=0; i<SIZE; i++)         struct merge m[SIZE];
    val[i] = f(key[i]);        for (i=0; i<SIZE; i++)
                                   m.val[i] = f(m.key[i]);
```

Reducing Miss Rate by Compiler Optimizations
2. Loop Interchange

Reordering of code to maximize the use of data in a cache block.

```
/* before*/                        /* after */
for (j=0; j<100; j++)              for (i=0; i<5000; i++)
    for (i=0; i<5000; i++)             for (j=0; j<100; j++)
        x[i][j] = 2*x[i][j];               x[i][j] = 2*x[i][j];
```

The original code skip through memory in strides of 100 words

while the revised code accesses all the words in the cache block before going to the next one.

It does not affect the number of instructions executed, unlike prior example.

Reducing Miss Rate by Compiler Optimizations
3. Loop Fusion

By fusing multiple loops which access the same arrays into a single loop, the data that are fetched into the cache can be used repeatedly before swapped out.

This technique reduces misses via improved temporal locality. Explain.

```
/* before */                       /* after */
for (i=0; i<N; i++)                for (i=0; i<N; i++)
    for (j=0; j<N; j++)                for (j=0; j<N; j++) {
        a[i][j]=1/b[i][j] *c[i][j];        a[i][j]=1/b[i][j] *c[i][j];
for (i=0; i<N; i++)                        d[i][j]=a[i]j]+c[i][j];}
    for (j=0; j<N; j++)
        d[i][j]=a[i]j]+c[i][j];
```

In the fused loop, the second statement freeloads on the cache accesses of the first statement.

Reducing Miss Rate by Compiler Optimizations
4. Blocking

Modify a loop into a nested loop so that the submatrices referenced in the inner loop can be fit in the cache.

B is called the ***blocking factor***.

Reducing Miss Penalty: Give Priority to Read Misses Over Writes

Example: (Page 411)

Reducing Miss Penalty: Give Priority to Read Misses Over Writes

Solution 1: let read miss wait until the write buffer is empty. This increase the read miss penalty (on MIPS M/1000 is about 1.5).

Solution 2: Check the content of write buffer on a read miss. If there is no conflicts and the memory system is available, let the read miss go head (fetch the block).

In case of write back cache and a read miss will replace a dirty memory block.

Solution 1: write dirty block first, then read block.

Solution 2: copy dirty block to write buffer, then read block, then write buffer to memory.

Reducing Miss Penalty: Sub-block Placement

Dilemma in designing a cache that must fit on the chip:

- Tag too big (can't fit on the chip or too slow), solution is large block.
- But large block increases the miss penalty.

Solution: Sub-block placement.

- A valid bit is added to units smaller than the full block, called sub-blocks.
- Only a single sub-block need to be read in a miss.
- Match of tag and valid bit is set indicate that the word is in the cache.
- Smaller miss penalty than full block.

Figure 5.21

Reducing Miss Penalty: Early Start and Critical Word First

- Early start - As soon as the requested word of the block arrives, send it to the CPU and let CPU continue execution.
- Critical word first - Request the missed word first from the memory, and send it to CPU as soon as it arrives; let CPU continue execution while filling the rest of the words in the block.

Reducing Miss Penalty: Second-level Cache

Add another level of memory between the original cache and memory to capture some of the memory accesses that would go to main memory.

AMAT = HittimeL1+MissrateL1xMissPenaltyL1

MissPenaltyL1=HittimeL2+MissrateL2xMiss PenaltyL2.

AMAT = HittimeL1+MissrateL1x(HittimeL2+MissrateL2xMissPenaltyL2)

Local miss rate: Missrate L2 for the second level cache.

Global miss rate: MissrateL1 x MissrateL2.

example page 417.

Miss rate vs. Cache size

figure 5.32

Example Page 420.

Main Memory

DRAM (Dynamic RAM) vs. SRAM (Static RAM)

Two measures of Memory Latency

- Access Time—the time between T**read is requested** and T**the word arrives**
- Cycle Time—the minimum time between two requests to memory.
  (the address line need to be stable for the next access)

	DRAM	SRAM
need to be refreshed	Yes≈every 8msec	No
	takes $\approx \sqrt{\text{Size}} \times \text{AccessTime}$	
circuit per bit cell	less	more
capacity	4~8x	1x
cycle time	8~16x	1x
price	8~16x	1x
used in (virtually)	Main memory	Cache
	cycle time > access time	cycle time = access time

Ferroelectric Memory—Fast Non-volatile Memory vs. EPROM

page B-30-33

Figure 5.30

Organizations to Achieve Higher Mem. Bandwidth

Memory Bandwidth:No. of bytes that can be transferred in a time unit(clockcycle)?

Four-way Interleaved Memory

Wider Main Memory vs. Interleaved Memory

Assume the basic memory organization takes 1 cycle to send address; 6 cycles to access a word; 1 cycle to transfer a word. For reading a block size of 4 words,

	Time(cycles)	Memory Bandwidth(bytes/cycle)
Basic memory width of 1word	4x(1+6+1)=32	16 bytes/32 cycles = 0.5
Memory with width of 2words	2x(1+6+1)=16	16 bytes/16 cycles = 1
Memory with width of 4words	1x(1+6+1)=8	16 bytes/8 cycles = 2

$CPIaverage = CPI_{execution} + MemoryAccesPerInstuction * MissRate * MissPenalty$

Example: Use CPIaverage to evaluation trade-off between the wide and interleaved memory. Assume that the clock cycle time and instruction counts do not change. For BlockSize(BS in word unit)=1, MemoryBusWidth(MBW in word unit)=1, MissRate=15%,(10%,5%for BS=2,4) MissPenalty=8cycles. MemoryAccessPerInstruction=1.2, **AverageCyclesPerInstruction (ignoring cache miss)**=2

	MissRate	CPIaverage	
BS=1, MBW=1, no interleaving	**15%**	**2+(1.2*15%*8)**	**=3.44**
BS=2, MBW=1, no interleaving	**10%**	**2+(1.2*10%*2*8)**	**=3.92**
BS=2, MBW=1, interleaving	**10%**	**2+(1.2*10%*(1+6+2))**	**=3.08**
BS=2, MBW=2, no interleaving	**10%**	**2+(1.2*10%*1*8)**	**=2.96**
BS=4, MBW=1, no interleaving	**5%**	**2+(1.2*5%*4*8)**	**=3.92**
BS=4, MBW=1, interleaving	**5%**	**2+(1.2*5%*(1+6+4))**	**=2.66**
BS=4, MBW=2, no interleaving	**5%**	**2+(1.2*5%*2*8)**	**=2.96**

Virtual Memory

is a technique to

- share a small physical main memory among many processes.
- facilitate programmers to write programs whose size is larger than main memory. Make physical main memory size invisible to programmers.
- relieve programmers the burden of writing code swap routine to move ***overlay*** program segment in/out of main memory.
- protect memory space for different processes.
- enable relocation of program anywhere in the main memory.
- reduce the time to start a program, why?

Physical main memory is divided into fixed length blocks, called **pages**. A miss is called **page fault**

If blocks are variable length, they are called **segments**. A segment miss is called **segment fault**.

When a page fault happens, the page will be brought in from the disk.

Mapping Virtual Address to Physical Address

Memory Mapping (Address Translation): CPU produces virtual addresses that are translated by a combination of hardware and software to physical addresses which are used in access main memory.

Figure 5.36

Page vs. Segment

Paging VM system: fixed size blocks, called pages. 4KB-64KB

Segment VM system: variable size blocks, called segments. 1B-2^{16}B~2^{32}B

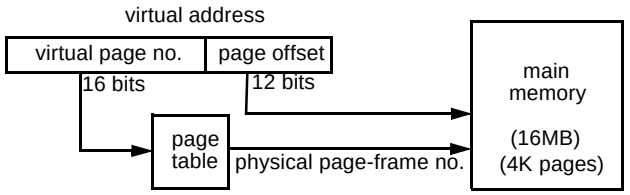
Figures 5.38-9

Q1: Where to place a block in main memory? (*Block placement*)

Because the horrendous cost of a miss (the DRAM speed and the disk speed are 4 orders of magnitude apart), OS designer always pick lower miss rates to allow blocks to be placed anywhere in main memory. (Fully associative)

Figure 5.37

Q2: How to find a block in main memory? (*Block identification*)



Example. Size of **page table**. For a 28-bit virtual address and a 4KB page and 4 bytes per page table entry, the page table has 2^{16}=64K entries and 256KB (equivalent of 64 pages). It is quite big and the address translation is quite slow.

To reduce the size of page table, some machines apply a hashing function on the virtual page no. to get a value btw 0~4K-1 and use an **Inverted page table**—one entry per physical pages in main memory. A 16MB physical memory needs 4B*16MB/4KB=16KB (4 pages) for the inverted page table.

To reduce the address translation time, a special cache called TLB for that.

Q3: Which block should be replaced on a virtual memory miss? *(Block replacement)*

- Replacement on cache miss is controlled by hardware; Replacement on virtual memory miss is controlled by OS.
- OS tries to replace Least-Recent Used (LRU) block.
- To help estimate the "freshness" of the blocks, many OS provide a use bit or reference bit.
  - set it when the page is accessed
  - and periodically clears them.

Q4: What happens on a write?

- Write through is too expensive. 0.7M~6M cycles
- Always use write back.

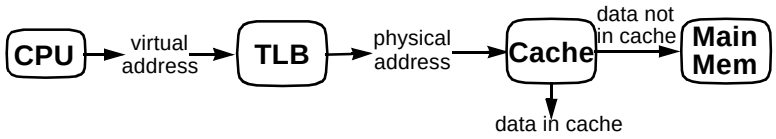
Translation-Lookaside Buffer: TLB

To reduce the virtual-to-physical address translation time, a cache called *Translation-Lookaside Buffer* (*TLB first used in IBM 370*) is used to keep the most recent address translation.

A TLB entry contains

- the tag field that holds the virtual page number, and
- the data field that holds the corresponding physical page-frame number, protect field, use bit, and dirty bit.

Combine cache with virtual memory

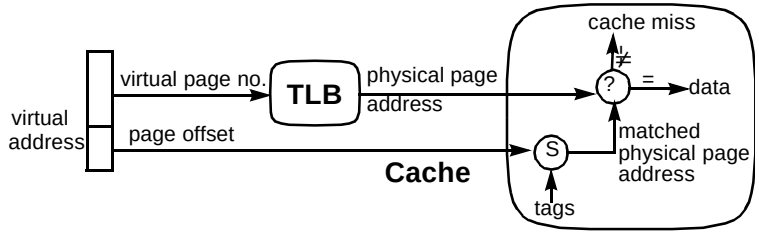


⇒ modified cache hit time = address translation time + original cache hit time

Is there a way to reduce the cache hit time?

How to combine cache with virtual memory

The following technique reduce the hit time. It uses the page offset to search the tags. This allows the search for a matched tag in cache and the address translation in TLB to be performed in parallel.



TLB is smaller and faster than the cache-address-tag memory ⇒ simultaneous TLB reading need not slow down cache hit time.

Drawback of this technique is that the direct-mapped cache can be no bigger than a page. Assume 16KB cache and block size=16bytes ⇒ there are 1K blocks.

If the page offset has 12 bits, then only the upper 8 bit will be used to identify the block location in cache.⇒ only can identify 256 blocks=4KB.

AXP 21064 TLB (32 entries)

Selecting a Page Size

Reasons for large page size:

- smaller page table
- can use page offset to index cache, allow concurrent operations
- Transferring larger pages to or from secondary storage is more efficient.
- The number of TLB entries may be restricted by the chip size/design.

Reasons for smaller page size:

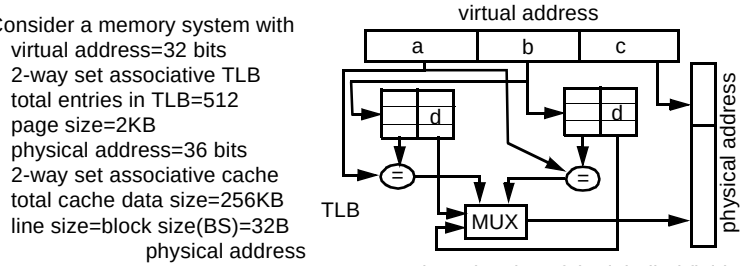
- waste less storage. (many processes are small)
  The text, heap, and stack segments of a program can not fit exactly into a page.
  The term for this unused memory in a page is ***internal fragmentation***.
- start-up time faster. (take less time to load).

AXP 21064 allows multiple page sizes: 8KB, 64KB, 512KB, and 4096KB.

a = TLB tag field
b =TLB index field
c= page offset
d = physical page frame address field
e = cache tag field f = cache index field
g = block offset

Exercise on Cache+Virtual Memory

Consider a memory system with
virtual address=32 bits
2-way set associative TLB
total entries in TLB=512
page size=2KB
physical address=36 bits
2-way set associative cache
total cache data size=256KB
line size=block size(BS)=32B



Show the size of the labelled fields
a=32-b-c=32-8-11=13
b=$\log_2$(entries/associativity)
=$\log_2$(512/2)=$\log_2$(256)=8
c=$\log_2$(page size)=$\log_2$(2^{11})=11
d=36-c=36-11=25
e=36-f-g=36-12-5=19
f=$\log_2$(datasize/(assoc.*BS))
=$\log_2$(2^{18}/(2*2^5))=12
g=$\log_2$(BS)=$\log_2$(32)=5

h=32*8=256bits

Exercise on Cache+Virtual Memory (continue)

- **0.4 data references/instruction -> MAPI=1.4**
  TLB miss rate=0.1%, TLB miss penalty=25 cycles
- **b) Find the average memory access time for a 4W block size. assume 1 clock cycle for the hittime,**

= $AMAT_{TLB}$ cache
= (1+0.001*25)+(1+0.015*6*4)=2.175

Block size with 8W has the lowest AMAT=2.165.

Table 1:

Size

Block	Miss Rate per reference	
1W	**4%**	
2W	**2.2%**	
4W	**1.5%**	
8W	**1%**	
16W	**0.8%**	

There is a trace-driven cache simulator called dinero in ~cs520/dinero/bin
Given a trace (the history of instruction and data reference of a program) and

instruction cache size and data cache size, associativity of the cache, replacement policy, and write policy),

For example, the following command
dinero -b32 -u32K -a1 < eg.din > eg32k.out

size=32btyes using a trace input file eg.din and generated output to eg32k.out.
The man page is in ~cs520/dinero/man. The eg.din is in ~cs520/dinero/demo.

Donot copy them. They are big.
Exercise: use eg.din as a trace and compare the performance of 128KB unified

DEC3100 has a cache with the second design.
There will be an exercise in homework#3 similar to the above exercise.

>dinero -b32 -u64K -a1 < ~cs520/traces/benchmarks/tex.din > tex.cache

```
---Dinero III by Mark D. Hill.
CMDLINE: dinero -b32 -u64K -a1
CACHE (bytes): blocksize=32, sub-blocksize=0, Usize=65536, Dsize=0, Isize=0.
POLICIES: assoc=1-way, replacement=l, fetch=d(1,0), write=c, allocate=w.
CTRL: debug=0, output=0, skipcount=0, maxcount=10000000, Q=0.

---Simulation begins.
Metrics                          Access Type:
(totals,fraction)      Total     Instrn     Data      Read      Write     Misc
-----------------      ------    ------     ------    ------    ------    ------
Demand Fetches         832477    597309     235168    130655    104513    0
                       1.0000    0.7175     0.2825    0.1569    0.1255    0.0000

Demand Misses          1537      130        1407      343       1064      0
                       0.0018    0.0002     0.0060    0.0026    0.0102    0.0000

Words From Memory      12296
(/ Demand Fetches)     0.0148
Words Copied-Back      8648
(/ Demand Writes)      0.0827
Total Traffic (words)  20944
(/ Demand Fetches)     0.0252

---Execution complete.
43.7u 11.1s 0:57 95% 92+470k 950+1io 0pf+0w
```


Cache-Coherency Problem (CPU-I/O)

Data are accessed concurrently by CPU and I/O. For the write-back cache, I/O devices may access the old copies of data in the main memory.

Figure 5.46

Homework # 3

Prob. 1. Use dinero and the cc1.din trace to evaluate the following caches with block size=32bytes, (which one has the lowest missed rate):
a) 256KB unified directed mapped write-through cache. (use -ww option)
b) 256KB unified directed mapped write-back cache.
c) 256KB unified 2-way associated write-back cache.
d) 128KB instruction + 128KB data directed mapped write back cache.
e) 192KB instruction +64KB data directed mapped write back cache.

Prob. 2. Exercise on Cache+Virtual Memory
Consider a memory system with virtual address=34 bits, fully associative TLB, total entries in TLB=1024, page size=8KB, physical address=36 bits, direct map cache, total cache data size=256KB, block size=32B.

a) What are the tag field sizes in TLB and cache?

b) Assume 0.5 data references per instruction, TLB miss rate=0.14%, TLB miss penalty=20 cycles, cache miss penalty=6 cycles+#words in a block, the miss rate for block size of 4 words is 1.4% while the miss rate for block size of 8 words is 1%. Which block size offers better performance?

Prob. 3. Given the following addresses from cache to main memory of DEC3000 model 800. Assume there is no time gap between the accesses.

a) Find the total time it takes to deliver all the blocks, 32 bytes, to the cache.

Assume the main memory are 1024MB and 256bit wide. It uses 16Mx1 page mode DRAMs with 30 ns access time when data is in the column latch is needed (row address matches previous row address,) 120 ns access time when data is not in the column latch (row address does not match previous row address.) It takes 10 ns to deliver addresses to the main memory and 10 ns to transfer data back to cache.

20d

1fc780

7fffccb0

1fd77c

223

54

7fffcc9c

56

6a

b) repeat a) with 4-way interleaved memory bank, 64bit wide bus.

Interesting Project: Memory System Evaluation Program

The attached program, when runs on a computer system, it prints the timing results of memory accesses to an integer array. By varying the stride of accesses to the array, we will get different timing results due to cache hits/ misses. The results allow us to know the size of the cache and the block. The following Figure shows the time results on SPARCstation 1+.

Interesting Project: Memory System Evaluation Program

Since the runtime of the program goes up dramatically when cache sizes above 64K are tried, we conclude that all previous expriments fit in cache and so the cache must be 64 KBytes.

Since the block size of the cache impacts the number of hits a stream of references of a given stride length at which the program begins to exhibit different behavior. We conclude the line size is 64 Bytes.

Potential Semester Project:

Extend the code so that it plots the curves (gnuplot or plot program of your choice) and answer the cache/block size of the system being evaluated. Answer 5.3c. verify the results on DEC3100, SparcClassic.

For graduate students, in additon to the above effort, extend the code to detect the size of the secondary cache, verify the results on DEC3000.

Solution to Homework # 3

Prob. 1. Use dinero and the cc1.din trace to evaluate the following caches with block size=32bytes, (which one has the lowest missed rate):

a) 256KB unified directed mapped write-through cache. (use -ww option)

ans: miss rate = 0.0098

b) 256KB unified directed mapped write-back cache.

ans: miss rate = 0.0098

c) 256KB unified 2-way associated write-back cache.

ans: miss rate = 0.0078

d) 128KB instruction + 128KB data directed mapped write back cache.

ans: miss rate = 0.0097

e) 192KB instruction +64KB data directed mapped write back cache.

ans: miss rate = 0.0085.

The 2-way set associate write back cache has the best performance.

Prob. 2. Exercise on Cache+Virtual Memory

Consider a memory system with virtual address=34 bits, fully associative TLB, total entries in TLB=1024, page size=8KB, physical address=36 bits, direct map cache, total cache data size=256KB, block size=32B.

a) What are the tag field sizes in TLB and cache?

Ans: For fully set assocative TLB, the set associativity is the number of entries in the TLB. There is no index field in the partition of the virtual address. The size of page offset field is $\log_2(8K)=\log_2(2^{13})=13$ bits. The size of tag field = 34-13=21 bits.

The size of block offset field = $\log_2(32)=5$ bits. The size of index field = $\log_2$(#of entries in cache)$=\log_2(256K/32)=\log_2(8K)=13$ bits. The size of tag field=36-13-5=18 bits.

b) Assume 0.5 data references per instruction, TLB miss rate=0.14%, TLB miss penalty=20 cycles, cache miss penalty=6 cycles+#words in a block, the miss rate for block size of 4 words is 1.4% while the miss rate for block size of 8 words is 1%. Which block size offers better performance?

Ans: Let x and y be the hit times of TLB and cache.

For 4 word block, AMAT=(x+0.0014*20)+(y+0.014*(6+4))=x+y+0.168

For 8 word block, AMAT=(x+0.0014*20)+(y+0.01*(6+8))=x+y+0.168.

The two block size offer the same performance based on AMAT.

Prob. 3. Given the following addresses from cache to main memory of DEC3000 model 800. Assume there is no time gap between the accesses.

a) Find the total time it takes to deliver all the blocks, 32 bytes, to the cache.

Assume the main memory are 1024MB and 256bit wide. It uses 16Mx1 page mode DRAMs with 30 ns access time when data is in the column latch is needed (row address matches previous row address,) 120 ns access time when data is not in the column latch (row address does not match previous row address.) It takes 10 ns to deliver addresses to the main memory and 10 ns to transfer data back to cache.

Address	Board Address	Row Address	Column Address	Match?	Deliver Address Time	Memory Access Time	Data Transfer Time
0000020d	**00**	**000**	**20d**	**first access, no**	**10**	**120**	**10**
001fc780	**00**	**1fc**	**780**	**000≠1fc, no**	**10**	**120**	**10**
7fffccb0	**7f**	**ffc**	**cb0**	**first access, no**	**10**	**120**	**10**
001fd77c	**00**	**1fd**	**77c**	**1fc≠1fd, no**	**10**	**120**	**10**
00000223	**00**	**000**	**223**	**1fd≠000, no**	**10**	**120**	**10**
00000054	**00**	**000**	**054**	**000=000, yes**	**10**	**30**	**10**
7fffcc9c	**7f**	**ffc**	**c9c**	**ffc=ffc, yes**	**10**	**30**	**10**
				Sub Total =	**70**	**660**	**70**
				Total Time=	**70+660+70=800 nsec**		

b) repeat a) with 4-way interleaved memory bank, 64bit wide bus.

Address	Board Address	Row Address	Column Address	Match?	Deliver Address Time	Memory Access Time	Data Transfer Time
0000020d	**00**	**000**	**20d**	**first access, no**	**10**	**120**	**40**
001fc780	**00**	**1fc**	**780**	**000≠1fc, no**	**10**	**120**	**40**
7fffccb0	**7f**	**ffc**	**cb0**	**first access, no**	**10**	**120**	**40**
001fd77c	**00**	**1fd**	**77c**	**1fc≠1fd, no**	**10**	**120**	**40**
00000223	**00**	**000**	**223**	**1fd≠000, no**	**10**	**120**	**40**
00000054	**00**	**000**	**054**	**000=000, yes**	**10**	**30**	**40**
7fffcc9c	**7f**	**ffc**	**c9c**	**ffc=ffc, yes**	**10**	**30**	**40**
				Sub Total =	**70**	**660**	**40**
				Total Time=	**70+660+280=1010 nsec**		

Solution to HW#3

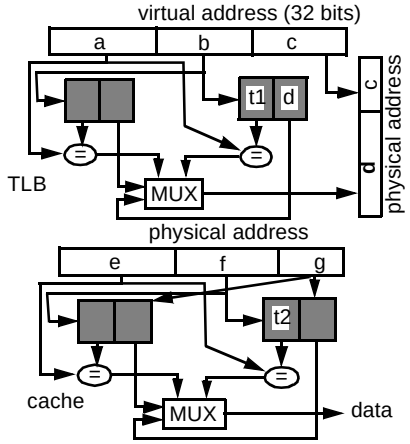
Prob. 1. Use dinero and the tex trace to evaluate the following caches with block size=32bytes, (which one has the lowest miss rate):
a) 256KB unified directed mapped write-through cache. (use -ww option)
b) 256KB unified directed mapped write-back cache.
c) 256KB unified 2-way associated write-back cache.
d) 128KB instruction +128KB data directed mapped write back cache.
e) 192KB instruction +64KB data directed mapped write back cache.

Sol. miss rate of 1a) = 0.16%
miss rate of 1b) = 0.16%
miss rate of 1c) = 0.15%
miss rate of 1d) = 0.16%
miss rate of 1e) = 0.16%
1c) has the lowest miss rate.
Note that write through and write back option will not affect the miss rate
No. of instruction references of the tex trace is higher but the instruction miss rate is lower. Instead of increasing the instruction cache size, we should increase data cache size. It will be interesting to find out, given 256KB cache memory, which organization gives the lowest miss rate.

a = TLB tag field
b =TLB index field
c= page offset
d = physical page frame address field
e = cache tag field f = cache index field
g = block offset

Solution to HW#3

Prob. 2a)



Show the size of the labelled fields
a=32-b-c=32-9-12=11
b=$\log_2$(entries/associativity)
=$\log_2$(1024/2)=9
c=$\log_2$(page size)=$\log_2$(4K)=12
d=36-c=36-12=24
e=36-f-g=36-12-5=19
f=$\log_2$(datasize/(assoc.*blocksize)
=$\log_2$(256K/(2*32))=12
g=$\log_2$(blocksize)=$\log_2$(32)=5

Tag field size of TLB = t1=a=11 bits
Tag field size of cache=e=19 bits.

Solution to HW#3

Prob. 2b) Assume the hittime is 1 cycle for both TLB and cache.
Average Memory Access Time$_{4w}$=AMAT$_{TLB}$+AMAT$_{cache}$
=(1+ 0.0014*22)+(1+ 0.014*(6+4))=2.1708 cycles
Average Memory Access Time$_{8w}$=AMAT$_{TLB}$+AMAT$_{cache}$
=(1+ 0.0014*22)+(1+0.01*(6+8))=2.1708 cycles
Both block sizes have the same performance.

Prob. 2c) CPI=CPIexec+CPITLB+CPIcache=CPIexec+
(MAPI*(hittime+missrate*misspenalty))$_{TLB}$+
(MAPI*(hittime+missrate*misspenalty))$_{cache}$
=1.35+(1+0.5)*2.1708
=4.6062

Caches in SuperSPARC Microprocessor

- Each set of the cache is required to be equal to the minimum MMU (TLB) page size (4KBytes).
- SPARC MBus is a 64-bit multiplexed bus implementing cache coherence protocol with modified, owned, exclusive, shared, and invalid states.
- A basic MBus cacheable memory reference transfer 32-bytes of data.

Caches in Motorola 88110 Superscalar RISC μp

- 40 (40) entry fully associative instruction (data) address TLB
- 32-entry branch target instruction cache.
- 64-bit bus with Multiprocessor cache snooping protocol. (write invalidate)

Caches in HP Precision Architecture RISC μp

- Use off-chip caches which can be very large and help achieve “balanced performance” across wide variety of applications.
- Most μps with on-chip caches still need additional secondary caches.
- Processors which donot cycle their secondary caches with the same frequency suffer large first level cache miss penalties.
- CPU chip contains a Unified TLB with 120 fixed sized full associative entries and 16 variable sized entries.

Evaluating Two-level Caches

Average memory-access time = Hit timeL1+Miss rateL1*
(Hit timeL2+Miss rateL2*Miss penaltyL2)

Example. For 1000 memory references there are 40 misses in the first level and 20 misses in the second level cache.

The miss rateL1=40/1000=4%. The local miss rateL2=20/40=50%. The global miss rate for the second-level cache is 20/1000=2%.

- The speed of the first level cache affects the clock rate of CPU.
- The speed of the second level cache only affects the miss penalty of the first level cache.

Project 16: Design and implement a two-level cache simulator based on the dinero source code.

Page Mode DRAM

DRAM Size and Performance

i486 internal structure

Internal frequency doubling used by 486DX2

To avoid emission electromagnetic waves. (proportion to $freq^4$)

To avoid the speed mismatch between external memory (2nd level cache) and the CPU. A SRAM of 12 ns access time corresponds to a clock frequency of 80MHz. But we have to design a system that tolerates the signal propagation delay.

A 66MHz 486DX2 operates with an external clock of 33 MHz and internally it doubles the frequency to 66 MHz.