Memory Hierarchies

Forecast

- Memory (B5)
- Motivation for memory hierarchy
- Cache
- ECC
- Virtual memory

Background

Mem Element	Size	Speed	Price/MB
Register	small	1-5ns	high ??
SRAM	medium	5-25ns	$??
DRAM	large	60-120ns	$1
Disk	large	10-20ms	$0.20

Background

Need basic element to store a bit - latch, flip-flop, capacitor

Memory is logically a 2D array of #locations x data-width

- e.g., 16 registers 32 bits each is a 16 x 32 memory
- (4 address bits; 32 bits of data)
- today's main memory chips are 8M x 8
- (23 address bits; 8 bits of data)

Register File

32 FF in parallel => one register

16 registers

one 16-way mux per read port

decode write enable

can use tri-state and bus for each port

SRAM

Static RAM

- does not lose data like DRAM
- 6T CMOS cell
- pass transistors as switch
- bit lines, word lines

SRAM interface

Today - 2M x 8 in 5-15ns

Typical large implementations (512 x 64) x 8

DRAM

Dense memory

- 1 T cell
- forgets data on read and after a while
- e.g., 16M x 1 in 4k x 4k array
- 24 address bits - 12 for row and 12 for column

Implementation

writeback row to restore destroyed value

Refresh - in background, march through reading all rows

Interface reflects internal orgn. - addr/2, RAS, CAS, data

Optimizations

Give faster access to some bits of row

- static column - change column address
- page mode - change column address & CAS hit (EDO)
- nibble mode - fast access to 4 bits

Bigger changes in future

- bandwidth inside >> external bandwidth
- 8kb/50ns/chip >> 8b/50ns/chip
- 164 Gb/s >> 20 Mb/s
- RAMBUS, IRAM, etc

Motivation for Hierarchy

CPU wants

- memory reference/insn * bytes-per-reference * IPC/Cycle
- 1.2*4*1/2ns = 2.4 GB/s

CPU can go only as fast as memory can supply

Motivation for Hierarchy

Want memory with

- fast access (e.g., one 500 ps CPU cycle)
- large capacity (10 GB)
- inexpensive ($1/MB)

Incompatible requirements

Fortunately memory references are not random!

Motivation for Hierarchy

Locality in time (temporal locality)

if a datum is recently referenced,

it is likely to be referenced again soon

Locality in space (spacial locality)

If a datum is recently referenced,

neighbouring data is likely to be referenced soon

Motivation for Hierarchy

E.g.,

- researching term paper - don’t look at all books at random
- if you look at a chapter in one book
- temporal - may re-read the chapter again
- spatial - may read neighbouring chapters
- Solution - leave the book on desk for a while
- hit - book on desk
- miss - book not on desk
- miss ratio - fraction not on desk

Motivation for Hierarchy

Memory access time = access-desk + miss-ratio * access-shelf

- 1 + 0.05 * 100
- 6 << 100

Extend this to several levels of hierarchy

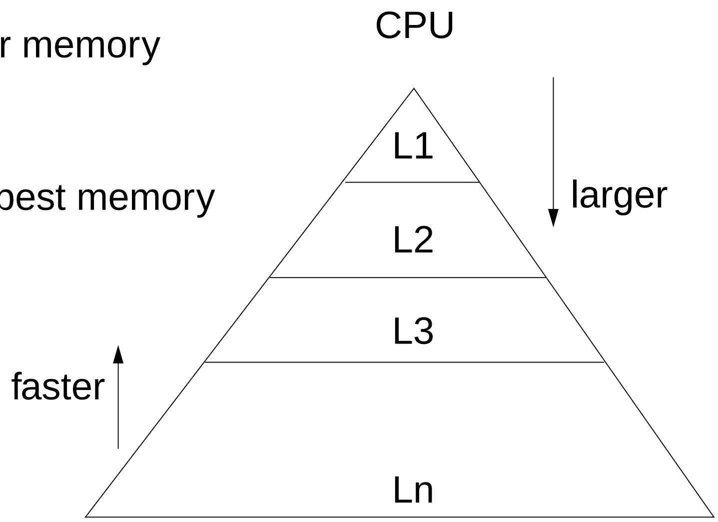
Memory Hierarchy

Small, fast, inexpensive memory

larger, slower, cheaper memory

. . .

largest, slowest, cheapest memory



Memory Hierarchy

Type	**Size**	**Speed (ns)**
Register	< 1 KB	0.5
L1 Cache	< 128 KB	1
L2 Cache	< 16 MB	20
Main memory	< 4 GB	100
Disk	> 10 GB	10×10^6

Memory Hierarchy

Registers <-> Main memory: managed by compiler/programmer

- holds expression temporaries
- holds variables - more aggressive
  - register allocation
  - spill when needed
  - hard!

Memory Hierarchy

Main memory <-> Disk: managed by

- program - explicit I/O
- operating system - virtual memory
  - illusion of larger memory
  - protection
  - transparent to user

Cache

cache managed by hardware

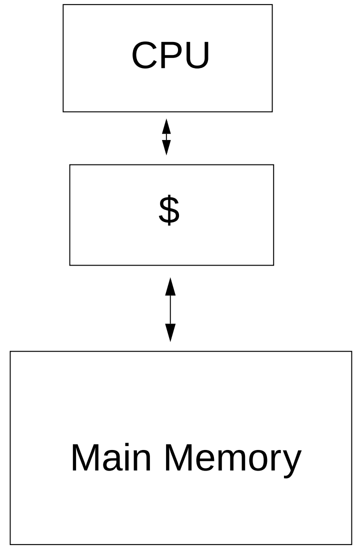
keep recently accessed block

- temporal locality

break memory into blocks (several bytes)

- spatial locality

transfer data to/from cache in blocks



Cache

put block in “block frame”

- state (e.g., valid)
- address tag
- data

Cache

on memory access

- if incoming tag == stored tag then HIT
- else MISS
  - << replace old block >>
  - get block from memory
  - put block in cache
  - return appropriate word within block

Cache Example

Memory words:

0x11c 0xe0e0e0e0

0x120 0xffffffff

0x124 0x00000001

0x128 0x00000007

0x12c 0x00000003

0x130 0xabababab

Cache Example

a 16-byte cache block frame:

- state tag data
- invalid 0x?? ???

lw $4, 0x128

Is tag ox120 in cache? (0x128 mod 16 = 0x128 & 0xfffffff0)

No, get block

- state tag data
- valid 0x129 0xffffffff, 0x1, 0x7, 0x3

Cache Example

Return 0x7 to CPU to put in $4

lw $5, 0x124

Is tag 0x120 in cache?

Yes, return 0x1 to CPU

Cache Example

Often

- cache 1 cycle
- main memory 20 cycles

Performance for data accesses with miss ratio 0.1

mean access = cache access + miss ratio * main memory access

= 1 + 0.01 * 20 = 1.2

Typically caches 64K, main memory 64M

- 20 times faster
- 1/1000 capacity but contains 98% of references

Cache

4 questions

- Where is block placed?
- How is block found?
- Which block is replaced?
- What happens on a write?

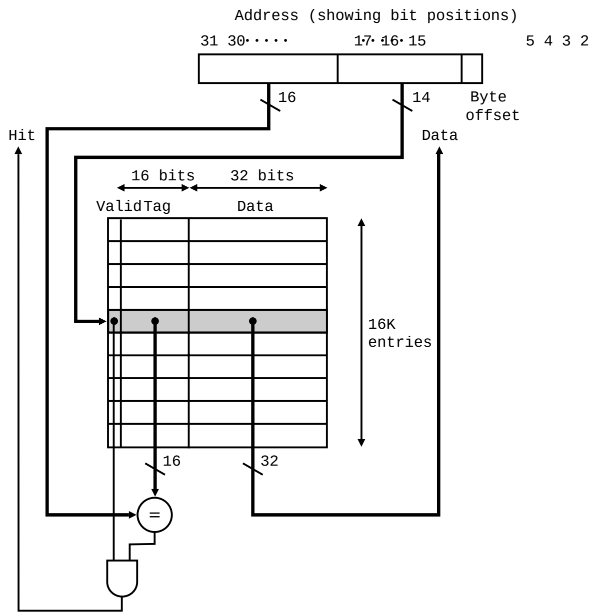
Cache Design

Simple cache first

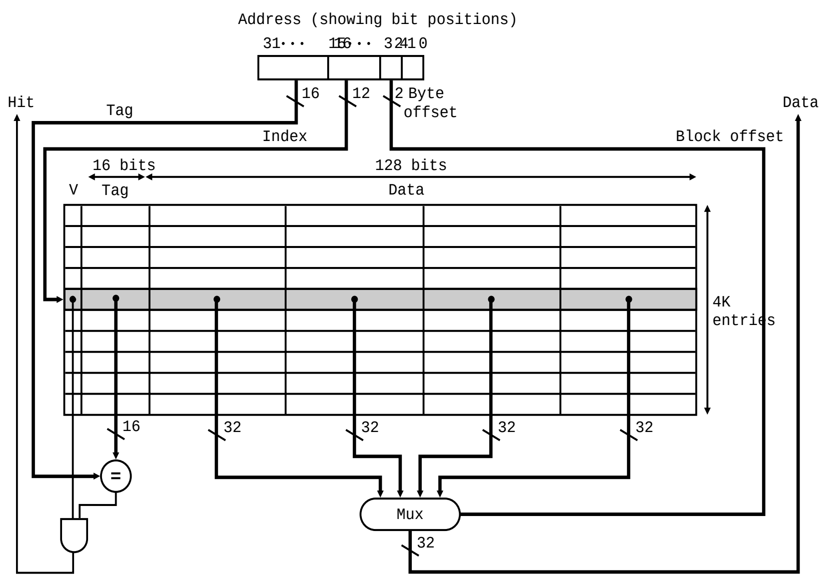
- block size = 1 word
- “direct-mapped”
- 16K words (64KB)
- index - 14 bits
- tag - 16 bits

Consider

- hit & miss
- place & replace



Cache Design w/ 16-byte blocks (7.10)

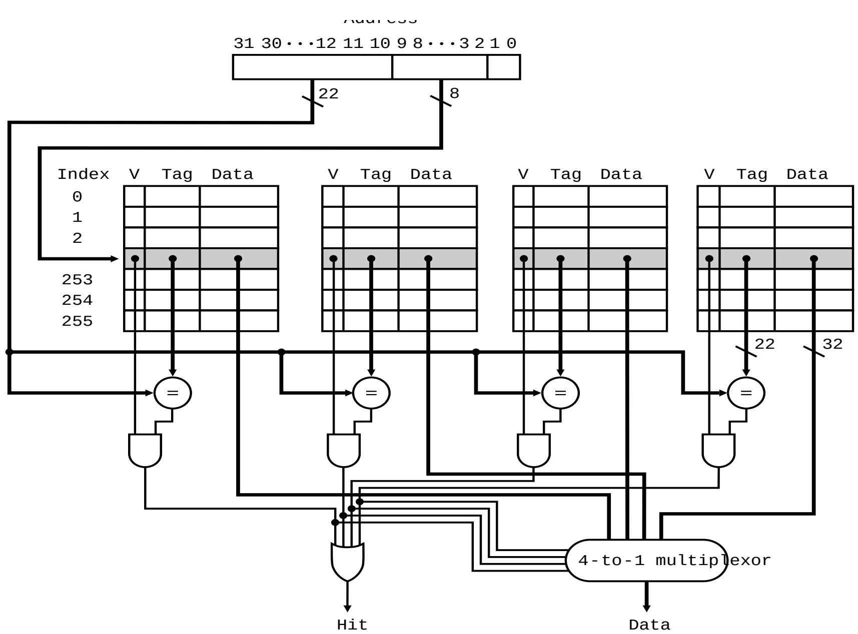


Cache Design

What if blocks conflict?

- Fully associative cache
  - CAM cells hold D and D'; incoming bits B and B'
  - match = AND (B_i*D_i + B'_i*D'_i)
- compromise - set associative cache

Cache Design w/ 4-way set-assoc. (7.19)



Cache Design

3C model

- Conflict
- Capacity
- Compulsory

Q3. Which block is replaced

- LRU
- random

Cache Design

Q4. What happens on a write?

- write hit must be slower
- propagate to memory?
  - immediately - write-through
  - on replacement - write-back

Cache Design

Exploit spatial locality

- bigger block size
- may increase miss penalty

Reduce conflicts

- more associativity
- may increase cache hit time

Cache Design

Unified vs. split instruction and data cache

Example

- consider building 16K I and D cache
- or a 32K unified cache
- let t_{cache} be 1 cycle and t_{memory} be 10 cycles

Cache Design

I and D split cache

- (a) I_{miss} is 5% and D_{miss} is 6%
- 75% references are instruction fetches
- $t_{avg} = (1 + 0.05*10)*0.75 + (1 + 0.06*10) * 0.25 = 1.5$

Unified cache

- $t_{avg} = 1 + 0.04*10 = 1.4$ WRONG!
- $t_{avg} = 1.4 +$ cycles-lost-to-interference
- will cycles-lost-to-interference be < 0.1?
- NOT for modern pipelined processors!

Cache Design

Multi-level caches

Many systems today have a cache hierarchy

E.g.,

- 16K I-cache
- 16K D-cache
- 1M L2-cache

Cache Design

Why?

- Processors getting faster w.r.t. main memory
- want larger caches to reduce frequency of costly misses
- but larger caches are slower!

Solution: Reduce cost of misses with a second level cache

Begin to occur: 3 Cache Levels

Split L1 instruction & data on chip

Unified L2 on chip

Unified L3 on board

CPU and Cache Performance

Cache only

- miss ratio
- average access time

Integrate - assume cache hits are part of the pipeline

Time/prog = insn/prog * cycles/insn * sec/cycle

CPI = (execution cycles + stall cycles)/insn

CPI = execution cycles/insn + stall cycles/insn

CPU and Cache Performance

Stall cycles/insn =

- read stall cycles/insn + write stall cycles/insn

read stall cycles/insn =

- read/insn * miss ratio * read miss penalty

write stall cycles/insn =

- more complex - write through, write back, write buffer?

CPU and Cache Performance

Example

- CPI with ideal memory is 1.5
- Assume IF and write never stall
- How is CPI degraded if loads are 25% of all insns
- loads miss 10% and miss cost is 20 cycles

CPI = 1.5 + 0.25*0.10*20 = 2

- 2/1,5 = 33% slower