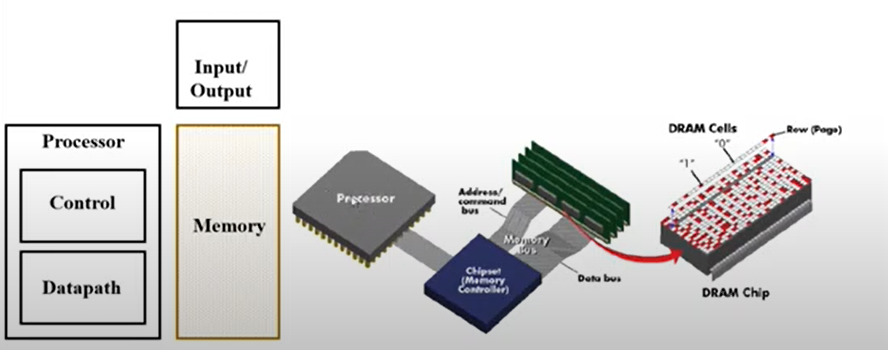
**MEMORY HIERARCHY AND CACHE DESIGN**

The Big Picture: Where are We Now?

The Five Classic Components of a Computer

Memory is usually implemented as:

* Dynamic Random Access Memory (DRAM) – for main memory
  + You have to update the contents synchronously at each time period. You dynamically fill the bits which are 1. Otherwise, 1s are converted to 0 in time. They cannot hold their charges forever. Disadvantage in terms of speed.
  + DRAMs are advantageous because it requires less number of transistors per bit.
* Static Random Access Memory (SRAM) – for cache



Any address of the memory can be accessible at the same time 🡪 That’s why it is random. You don’t have to access sequential memory locations evenly.

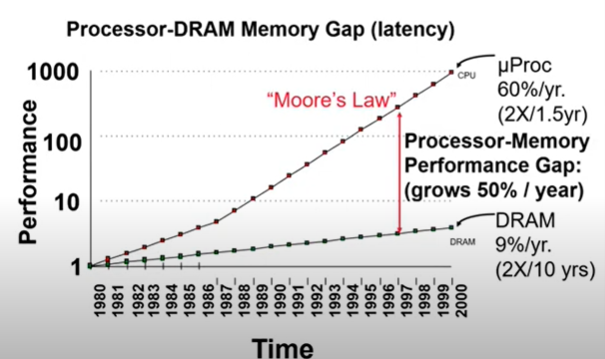
Technology Trends

Table

Description automatically generated

If you make transistor smaller, it gets faster. Both memory and logic uses the same transistors. Why can we increase logic faster?

* Main reason is capacity of memory. While you increase the speed of transistors, at the same time you increase the capacity of the memory. If you have a larger memory, it becomes difficult to access a specific data in that memory.



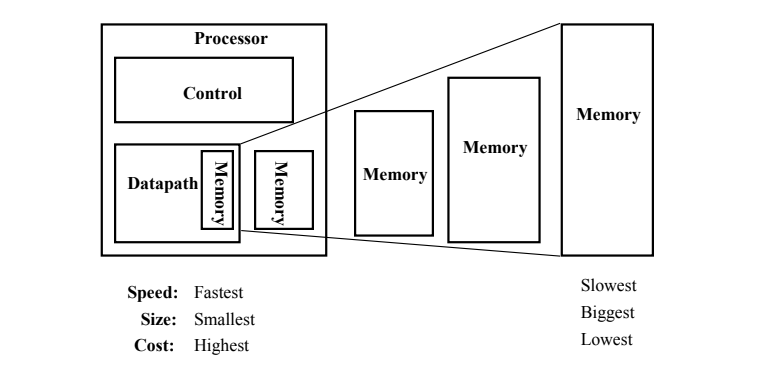
**Today’s Situation: Microprocessors**

Rely on caches to bridge gap

Cache is a high-speed memory between the processor and main memory

1980: no cache in proc;  
1997: 2-level cache, on Alpha 21164 proc

An Expanded View of the Memory System



Cache’e data her zaman blok halinde taşınır, tek bir word olarak taşınmaz. Spatial locality’den dolayı. Array[0]’a access ettiyse 1’e, 2’ye de access edecektir.

L1 cache’deki her şey L2 cache’de bulunmak zorunda.

Taking Advantage of Locality

Memory hierarchy

Store everything on disk

Copy recently accessed (and nearby) items from disk to smaller DRAM memory

* Main memory

Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory

* Cache memory attached to CPU

Temporal Locality (Locality in Time)

Keep the most recently accessed data items closer to the processor

Spatial Locality (Locality in Space)

Move blocks consists of contiguous words to the upper levels

Diagram, table

Description automatically generated

Upper Level 🡪 CPU’ya yakın

Memory Hierarchy: Terminology

Hit: If the data requested by a processor appears in some block in the upper level

* Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
  + Time required to read data from L1 cache
  + Determines the cycle period for MEM stage
* Hit Rate: The fraction of memory access found in the upper level

Miss: If the data is not found in the upper level

* Miss Rate = 1 – (Hit Rate)
* Miss Penalty: Time to replace a block in the upper level + Time to deliver the block the processor

Hit Time << Miss Penalty

Memory Hierarchy of a Modern Computer System

By taking advantage of the principle of locality:

* Present the user with as much memory as is available in the cheapest technology
* Provide access at the speed offered by the fastest technology

Diagram

Description automatically generated

How is the hierarchy managed?

* Registers <-> Memory
  + by instructions and by CPU (lw – sw)
  + Compiler decides which register holds which register content
* Cache <-> Memory
  + by the hardware
* Memory <-> Disks
  + by the hardware and operating system (virtual memory)
    - speed is as slow as allowing the software to come into play
  + by the programmer (files)

Random Access:

* “Random” is good: access time is the same for all locations
* DRAM: Dynamic Random Access Memory
  + High density, low power, cheap, slow
  + Dynamic: need to be “refreshed” regularly
  + A couple of transistors (2-3) are enough to hold 1 bit
* SRAM: Static Random Access Memory
  + Low density, high power, expensive, fast
  + Static: content will last “forever” (until lose power)
  + You need more than 5 transistors to hold 1 bit

“Non-so-random” Access Technology:

* Access time varies from location to location and from time to time
* Examples: Disk, CDROM

Sequential Access Technology: access time linear in location (e.g.,Tape)

General Principles of Memory

Locality

* Temporal Locality : referenced memory is likely to be referenced again soon (e.g. code within a loop)
* Spatial Locality : memory close to referenced memory is likely to be referenced soon (e.g., data in a sequentially access array)

Definitions

* Upper : memory closer to processor
* Block : minimum unit that is present or not present
  + Bazı durumlarda bir block tek word’ten de oluşabilir. O zaman spatial locality’den çok faydalanmamış olursun.
* Block address : location of block in memory
* Hit : Data is found in the desired location
* Hit time : time to access upper level
* Miss rate : percentage of time item not found in upper level

Locality + smaller HW is faster = memory hierarchy

* Levels : each smaller, faster, more expensive/byte than level below
* Inclusive : data found in upper level also found in the lower level
  + L3 cachedeki tüm data main memory’de bulunması gereken data.
  + A picture containing text

    Description automatically generatedL1 cachede değişiklik yaptın, bu değişiklik her yerde yapılmalı o yüzden inclusive

Differences in Memory Levels (2005)

Table

Description automatically generated



MEM stagei birçok cycle’dan oluşacak şekilde düzenleyebiliriz.

Pagefold: Main memory’de istediğini bulamamak.

Kat farkları günümüz için de doğru. Main memory-cacheler arası 5-10 kat hız farkı var.

Memory Technology (~2015)

Static RAM (SRAM)

* 0.5ns – 2.5ns, $2000 – $5000 per GB
* Clock hızı: 1 ns kullanırsak 1GHz, 0.5 ns kullanırsak 2 GHz (kullanabileceğimiz max clock hızı).

Dynamic RAM (DRAM)

* 50ns – 70ns, $20 – $75 per GB

Magnetic disk

* 5ms – 20ms, $0.20 – $2 per GB

Ideal memory

* Access time of SRAM
* Capacity and cost/GB of disk

Four Questions for Memory Hierarchy Designers

* Q1: Where can a block be placed in the upper level?

(Block placement)

* Q2: How is a block found if it is in the upper level?

(Block identification)

lw’ün verdiği adres main memory adresi (öyle düşünelim), cache’deki yeri nasıl bulacağız?

* Q3: Which block should be replaced on a miss?

(Block replacement)

* Q4: What happens on a write?

(Write strategy)

Cache’de değiştirdim datayı. Main memory’de bu değişikliği yapmazsam, cache’deki data yeni gelen block tarafından silinirse bir daha değiştirdiğim yere access etmek istediğimde değişmemiş haline ulaşırız. Main memory’den eski halini çekeriz. Bu yazma işini nasıl yapacağız?

**Q1: Where can a block be placed?**

Direct Mapped: Each block has only one place that it can appear in the cache.

* Her koltuk numaralı olsun. Her insana numara verelim. O koltuğa otursun.

Fully associative: Each block can be placed anywhere in the cache.

* Salon boş olduğu halde başka insana da aynı numaralı koltuk çıkarsa iki insan kavga edecek.
* Herkes istediği koltuğa oturabilir.

Set associative: Each block can be placed in a restricted set of places in the cache.

* If there are n blocks in a set, the cache is called n-way set associative
* Her sıranın numarası olsun, o sıradaki istediği (en uzun süredir oturulan) koltuğa otursun.
* Bir sırada 4 koltuk varsa 🡪 4-way set associative

What is the associativity of a direct mapped cache?

* 1-way set associative ----> her sırada 1 koltuk varmış gibi

Direct Mapped Caches

Mapping for direct mapped cache (Neye göre numara veriyoruz?):

* (Block address) MOD (Number of blocks in the cache)
* Aşağıdan alacağımız blockun adresi var. Normalde lw word adresini verir. O word adresten blockun adresi elde edilir (we’ll see).

Chart, radar chart

Description automatically generated

Cacheimizde 8 farklı yer var yani mod 8’ini alacağız.

Cachedeki gri block memorydeki 1. gri de olabilir, 2. de, 3. de, 4. de; nasıl anlayacağız? Bir blockun datası cache’e konduğunda, block adresin indexte kullanılmayan kalan kısmı da (00001 ise 000) cache’e koyulmalı (tag).

Eğer koltuk doluysa, aynı koltuk çıkmışsa, oturan gider yeni gelen oturur. Ama ya oturan da yeni geldiyse? Ondan uzun süre faydalanamadık.

Full associative olsaydı, eğer salon dolu olsaydı, en uzun süre oturan kaldırılırdı. Zaten CPU’nun onunla işi bitmiştir. Temporal locality’den daha fazla faydalanırız.

Direct map’in iyi yanı aradığın şeyi kolaylıkla bulabilmendir. Yani hit time oldukça düşüktür.

Direct map miss rate’i olumsuz etkiler, daha fazla miss rateler ile karşılaşabiliriz. Çünkü aradığımızın bulunamama ihtimali yüksek.

L1 cache’de 1 numaralı önceliğimiz hit timedır. Çünkü L1 cachein hit timeına göre clock speedimizi ayarlıyoruz (L1 cache MEM stageinde). O yüzden L1 cache genelde direct map kullanır.

L2 ve L3 cachelerde ve özellikle main memoryde optimize etmeyi istediğimiz şey miss rate olur ki aradığımızı bulalım, diske kadar gitmeyelim. O yüzden main memory full associativedir.

L2 cachei 8-way, L3 cachei 16-way set associative yapabilirsin.

Cache Example

Table

Description automatically generated

V: valid bit.

Datalar rastgele olabilir ama şu an benim programıma ait olmadığı için valid bitleri hep no. Valid bit 0’sa o içerik sana ait değil demektir.

Table

Description automatically generated

Datayı wordlere bölmen lazım ve MUX ile seçmen lazım. lw hangisine access etmek istiyorsa onu okuyacak.

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

18’i koymak için 26’yı yerinden kaldırdık. Daha sonra 26’yı almak istese miss olacak ve 18’i yerinden kaldıracak.

Bi 26, bi 18, bi 26, bi 18… diye gidiyorsa sürekli miss alacağız. Miss rate %100 olacak.

Blockların her birinin 4 wordden oluştuğunu düşünürsen arrayin 0. elemanına access ettikten sonra 8 block (32 word) sonrasına, sonra 8 block gerisine, 8 block sonrası, 8 block gerisi… diye gidersen miss rate yine çok kötü olur. Yani arrayin bi 0., bi 32., bi 0. …. elemanına erişmek çok kötü olur, sürekli miss olur.

* 8 blocklu cachede her 8 blockta bir aynı adres denk gelir.

Daha büyük blocklu cacheler kullanılabilir.

Burada 22’yi (10110) istiyoruz diyelim. Geliyoruz 110’a bakıyoruz index tutuyor. Tag (10) de tutuyor. Direkt datayı alıp kullanıyoruz.

Miss durumunda MEM stageindeki cache’e istenilen datalar çekilene kadar tüm instructionlar durdurulur. Her miss programı yavaşlatır.

Associativity Examples

Diagram

Description automatically generated

Set associative’de set sayısına göre mod alınır.

Table

Description automatically generatedExample:

Table

Description automatically generated

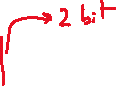
lw sana block adres değil, word adres (32 bitlik sayının adresi) veriyor. Blockun aslında belki 8 tane 32-bit wordten oluşuyor. O block adresinin de adresi olması lazım çünkü o block adresinden mod alarak cachedeki blockun yerini buluruz. Word adresten nasıl block adres bulacağız?

**Q2: How is a Block Found?**



Text

Description automatically generated



byte offset 2 bit çünkü 1 wordte 4 byte var, 2 bitle seçebiliriz.

The address can be divided into 2 main parts:

* Block offset: selects the data from the block
  + offset size = log2(block size)
* Block address: tag + index
  + index: selects set in cache
    - index size = log2(#blocks/associativity)
  + tag: compared to tag in cache to determine hit
    - tag size = address size – index size – offset size

Each block has a valid bit that tells if the block is valid – the block is in the cache if the tags match and the valid bit is set.

Word adreslerimiz 4’ün katı yani son 2 bitleri hep 0.

Wordün içindeki farklı bytelara access etmek istiyorsak, kalan 30 biti alıyoruz, 1 nolu byte’a access edeceksek 01 diyoruz, 2 nolu byte’a access edeceksek 10 diyoruz, 3 nolu byte’a access edeceksek 11 diyoruz.

Yani son 2 bite byte offset deniliyor. Buranın görevi byte’ı seçmek.

Word adresten block adres elde etmemiz için:

* block adresimiz olacak
* block adresin least significant kısmı da o blockta hangi wordü istediğimizi, hatta o word içinde hangi byte’ı istediğimizi söyleyecek
* blockta kaç byte varsa offsetin size’ını o belirler
  + block = 8 word diyelim
  + block = 32 byte
  + Herhangi bir byte’a işaret edebilmek için 5 bite ihtiyacım var.
  + lw (veya lb) ile verdiğim 32 bitlik adresin least sig 5 biti o block içerisindeki bir byte’ı seçmek için kullanılıyor. Bu 5 bite block offset deriz.
  + Bu 5 bitin least sig 2 biti byte seçmek için (lw’se zaten 00), 3 biti de wordlerden birini seçmek için.
  + Bu örnekte word adresten block adrese gidebilmek için least sig 5 bit atılır, kalan adres block adresidir. Çünkü o adres o blocktaki tüm wordler için sabit olacak.
  + Block offsetteki most sig 3 biti değiştirerek farklı wordlere access edebiliriz. Least sig 2 biti değiştirerek de farklı bytelara access edebiliriz.
  + Blocktaki bütün wordler için block address aynı kalacak.

Block addressten sonra cache’de gideceğimiz yeri nasıl biliyoruz ve buluyoruz?

* block adresin modunu alarak
  + diyelim 32 block varsa mod 32 (25) alırsın yani son 5 bite bakarsın 🡪 index kısmı 5 bit

A 4-KB Cache Using 1-word (4-byte) Blocks

Diagram, engineering drawing

Description automatically generated

Block size ve word size aynı. Block offset = Byte offset. Spatial localityden faydalanılmamış.

Least sig 2 bit block içerisindeki farklı byteları seçmek için kullanılacak. Word adres verilmişse zaten offset 00.

Blockların 4 byte oluşundan byte offsetin 2 bit olacağını bulduk.

Adreslerim 32 bit.

4 KB = 212 byte 🡪 cache datası

22 byte 🡪 1 blockun size’ı çünkü her blockta 1 word taşıyorum

Toplam block = 210

Cache direct mapse (ki örnekte öyle), mod 210 alacağım. Bu son 10 bite bakmak demek. Yani index 10 bit.

Data 🡪 1 word = 32 bit

210 block var. Her block için 32 bit (data) + 20 bit (tag) + 1 bit (valid) = 53 bit tutuyorum:

* 1024 x 53 = cachein bit cinsinden sizeı
* 128 x 53 byte = 6784 byte

Yani 4 KB cache için aslında ~6 KB saklanması lazım. Çünkü sadece data yok, tag ve valid de var.

Two-way Set-associative Cache

Diagram, engineering drawing

Description automatically generated

Two-way set associative 🡪 her kümede 2 block var.

1024 kümem var, her kümede 2 block var. 🡪 toplam 2048 block

Her block yine 4 byte’dan (1 word) oluşuyor. 🡪 2048 x 4 bytes (cache’in data size’ı) = 8 KB

Her blocktaki data 4 byte olduğuna göre block offset (byte offset de diyebiliriz çünkü block sadece 1 byte’dan oluşuyor) yine 2 bit.

Indexe karar veren şey bu sefer küme sayısı, block sayısı değil. Çünkü set associative’de mod alınırken küme sayısına göre mod alınır.

* mod 1024 = mod 210 = son 10 bite bak 🡪 index 10 bit

2048 \* 53 bit = total cache size

Associativity arttıkça multiplexer sayısı da artar. Böylece hit time artar.

Example: Alpha 21064 Data Cache

The data cache of the Alpha 21064 has the following features:

* 8 KB of data
* 32 byte blocks (8 word)
* Direct mapped placement
* Write through (no-write allocate, 4-block write buffer)
* 34 bit physical address composed of
  + 5 bit block offset
  + 8 bit index
  + 21 bit tag

Diagram

Description automatically generated

Burada 1 word = 64 bit = 8 byte.

256 bit = 4 word

5 bitlik offsetin 2 biti 4 wordden birini seçmek için kullanılır. 3 biti wordün 8 byte’ından birini seçmek için kullanılır.

**Q3: Which Block Should be Replaced on a Miss?**

Easy for Direct Mapped – only on choice

Aradığımız blocku cache’de bulamadık. Miss penalty ödeyeceğiz.

Set associative or Fully associative:

* Random – easier to implement
  + direct map’e göre daha az conflict olabilir
* Least Recently Used (the block has been unused for the longest time) – harder to implement

Miss rates for caches with different size, associativity and replacement algorithm

Text, table

Description automatically generated



For caches with low miss rates, random is almost as good as LRU (Least Recently Used).

CPU’ya çok yakın cachelerde LRU kullanmak mantıklı olabilir. Çünkü size küçük.

Cache Misses

On cache hit, CPU proceeds normally

On cache miss

* Stall the CPU pipeline
* Fetch block from next level of hierarchy
* Instruction cache miss
  + Restart instruction fetch
* Data cache miss
  + Complete data access

**Q4: What Happens on a Write?**

Write demek sw instructionı demek. Bir block 8 tane wordten oluşuyorsa, sw sadece 1 wordünü değiştirir.

Write through: The information is written to both the block in the cache and to the block in the lower-level memory.

Main memory çok yavaş. Her yazmada main memory’yi beklemek çok cycle beklememe neden olur.

Write back: The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

* is block clean or dirty? (add a dirty bit to each block)
  + datanın değişmiş olup olmadığını anlamak için
  + dirty bit 1’se datada değişiklik yapılmış, aşağı seviyelere yazılmalı
* Defalarca datayı değiştirsen de, sadece data cache’den replace edilirken aşağı seviyelere yazmış oluyorsun.

**Write Through**

On data-write hit, could just update the block in cache

* But then cache and memory would be inconsistent

Write through: also update memory

But makes writes take longer

* e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
  + Effective CPI = 1 + 0.1×100 = 11

Solution: write buffer

* Holds data waiting to be written to memory
* CPU continues immediately
  + Only stalls on write if write buffer is already full

**Write-Back**

Alternative: On data-write hit, just update the block in cache

* Keep track of whether each block is dirty

When a dirty block is replaced

* Write it back to memory
* Can use a write buffer to allow replacing block to be read first

Pros and cons of each:

Write through

* Read misses cannot result in writes to memory,
  + write back’te read misslerde (lw) de aşağıya yazmak zorunda kalabilirsin. Cache’de silinen datanın aşağıya yazılmasına sebep olunabilir. Buffer kullanabilirsin.
* Easier to implement
* Always combine with write buffers to avoid memory latency

Write back

* Less memory traffic
* Perform writes at the speed of the cache
* write allocate always

Since data doesn’t have to be brought into the cache on a write miss, there are 2 options:

* Write allocate
  + The block is brought into the cache on a write miss
  + Used with write-back caches
  + Hope subsequent writes to the block hit in cache
* No-write allocate
  + The block is modified in memory, but not brought into the cache
  + Used with write-through caches
  + Writes have to go to memory anyway, so why bring the block into the cache

Text

Description automatically generated

Örneğin looplarda sürekli aynı instructionı çağırıyoruz. Bu da instruction cache’in miss rate’inin data cache’in miss rate’inden çok daha az olmasını sağlıyor.

Ayrıca insturctionlar sequential olarak çalıştırılır. Yani sıra sıra instructionlar çalıştırılacak (loop ve functionlar dışında).

Yani instruction cachelerde hem temporal, hem spatial locality çok yüksek.

Diagram

Description automatically generated

Her word 4 byte, bir blockta 16 tane word var. Bir blockun boyutu 16 \* 4 = 64 byte = 26. Offset kısmı son 6 bit.

Her blockta 16 word olduğu için MUX 16x1.

***Examples***:

Text

Description automatically generated

Text

Description automatically generated

**SUMMARY**

CPU-Memory gap is major performance obstacle for achieving high performance

Memory hierarchies

* Take advantage of program locality
* Closer to processor => smaller, faster, more expensive
* Further from processor => bigger, slower, less expensive

4 questions for memory hierarchy

* Block placement, block identification, block replacement, and write strategy

Cache parameters

* Cache size, block size, associativity