Cache
Memory
Cache Memory

* Why? Temporal Locality (Access Again)

CPU \[\rightarrow\] Memory

\[\leq 0.5\ \text{ns}\]
Cycle time

2 GHz

Hundreds of cycles

* How?

CPU \[\leftarrow\] Cache \[\rightarrow\] Memory

* Granularity

- Line Size (Block Size)
  (Spatial Locality)
The Abstraction (Physically Addressed)

VIRTUAL ADDR

TLB

1. To get thePFN

PHYSICAL ADDR

TAG STORE

2. To see if it is there

YES! A HIT

DATA STORE

3. To get it

Three sequential accesses to storage. Can we do better?
First Example:
Memory: 256 B
Cache: 64 B
Line Size: 8 B
Direct Mapped
FURTHER EXAMPLE (INCREASE ASSOCIATIVITY) Cache/4

MEMORY: 256MB
CACHE: 64KB
LINE SIZE: 8B
4-WAY SET ASSOC

WHY?

Tag Store
Cache A

Cache D
Data

Hit/Miss

INDEX
To Cache Line

Tag Store Entry:
(Bookkeeping Information)

< LRU

TAG V D Replacement

Do You Need It Always?
Note: Tag Store and Data Store accessed at the same time (no longer 3 sequential accesses)
Can we do it in one access?

Virtual/Physical

VA TAG INDEX

Tag Store Logic

Data Store

Plus: Don’t wait for translation

Minuses: Same VA refers to two things
One PA stored in two places

How do we handle this
1. Flush on context switch (cold start)
2. Include process ID in tag store
Can we do it in one access?

Virtual/Physical

VA | PAGE NO | INDEX
---|---------|------
    |         |      
TLB

PA | FRAME NO | INDEX
---|---------|------
    |         |      
    |         |      

Use unmapped bits for access index

Use frame number for tag

Tag Store

Data Store

Logic

Plus: Don't wait for translation

Minus: Limits size of cache
   (although you can increase associativity)
Can we do it in one access time?

Virtually indexed / Physically tagged

VA  Page Number  Page Offset

INDEX

TLB

TAG STORE

DATA STORE

PFN

PA TAGS COMPARE

Plus: Don't wait for translation

Minus: Synonym problem
  (Block can be in more than one place)
Characteristics

* Set Associative (Set Size)
  - Fully Associative
  - Direct Mapped
  - Hit Ratio = \( \frac{\text{Hits}}{\text{Hits} + \text{Misses}} \)

* Write Back, Write Through

* Replacement Algorithm
  - LRU
  - FIFO
  - Random

* Instructions/Data

* Supervisor/User

* Virtual/Physical
Write Through/Write Back

```
PROC → CACHE → MEM
```

```
PROC → CACHE → MEM
```

**Issues**

* Simplicity of Design
* Bus Traffic
* Application Environment (Stack Frame)
* Allocate on Write Miss
  - Sector Cache
**Replacement**

* Two-Way (LRU)

Set | Way | Way | Way

One bit to tell you which way is LRU

* Four-Way (Pseudo-LRU)

1. [Diagram of cache with ways A, B, C, D and three bits per set A/B, C/D]

2. Victim, Next Victim

<table>
<thead>
<tr>
<th>Way</th>
<th>V</th>
<th>NV</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Access to B

We flip a coin to determine A or C is now victim.
The Microprocessor

* First Cache - Motorola 68020
  - mid-1980s, one level, 256 bytes I-Cache

* Today - several levels (Cache Hierarchy)

- Separate L1 Inst/L1 Data

- Inclusion Property (Important for Coherence)

- Prefetch
  - Makes Cache Visible (Prefetch Instruction)
  - Hardware - (Not Visible)
Coherence

* When we study multi-processors
  - Cache coherence (Hardware)
  - Memory consistency (Software)

* Even in a uniprocessor
  (If) we have intelligent controllers