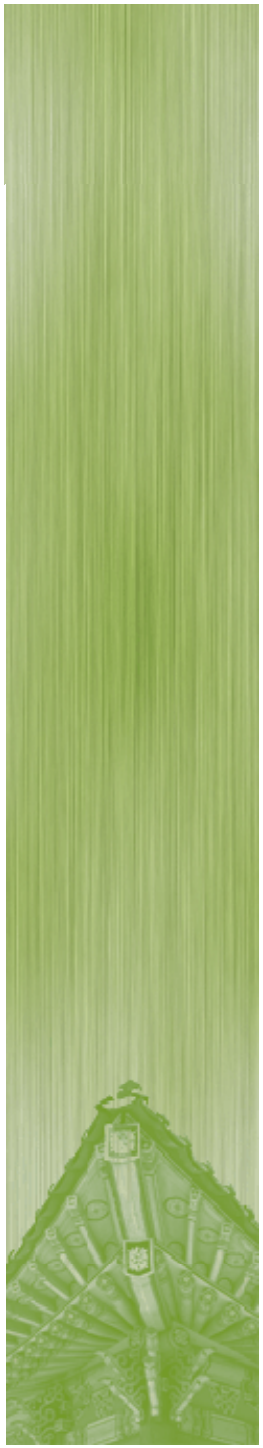


Caches

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Memory Technology

- **Static RAM (SRAM)**
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- **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

Principle of Locality

- **Programs access a small portion of their address space at any time**
- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon
  - e.g., sequential instruction access, array data

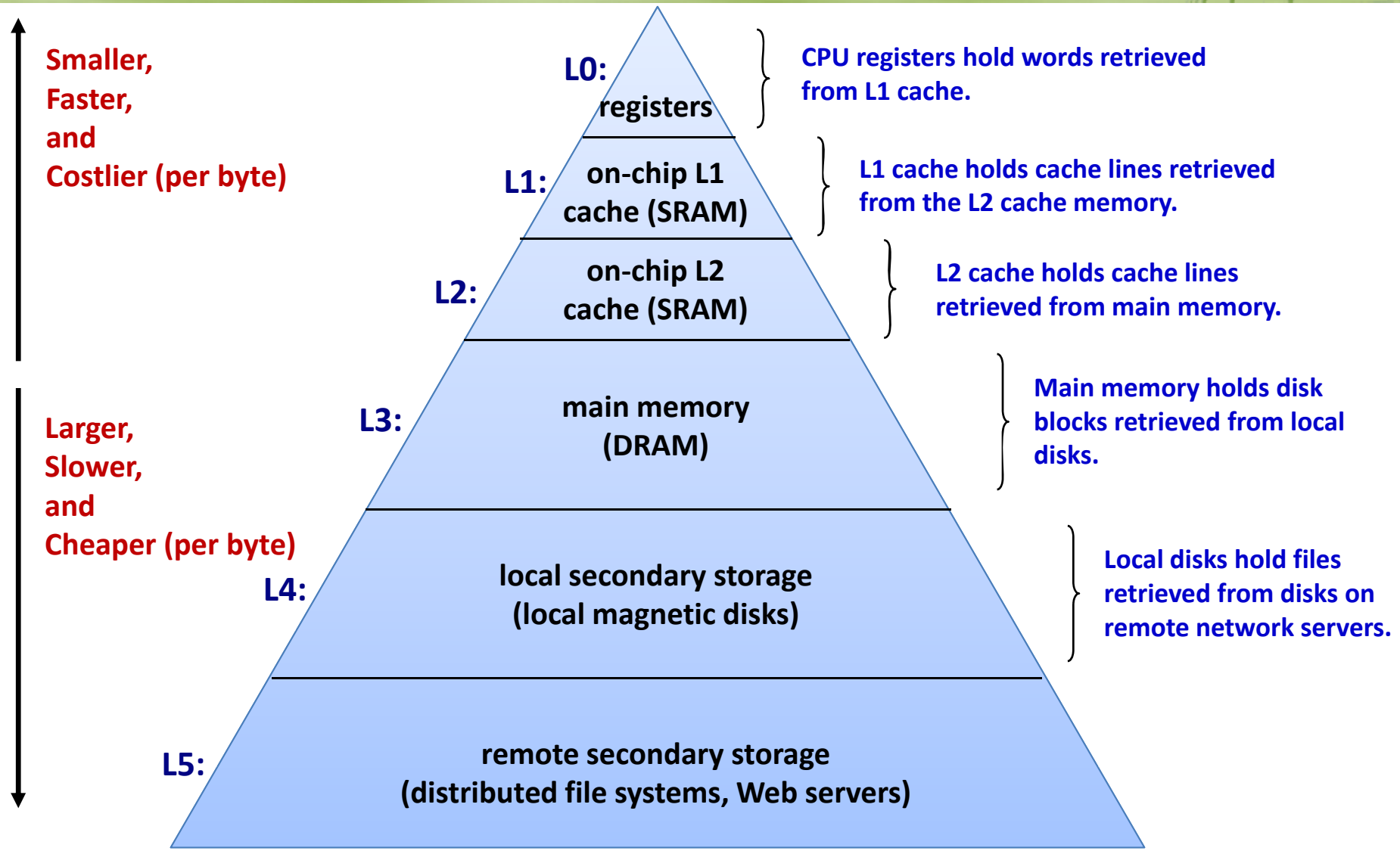
Memory Hierarchy (1)

"We are therefore forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."

-- A. W. Burks, H. H. Goldstein, J. von Neumann, Preliminary Discussion of the Logical Design of Electronic Computing Instrument, June 1946.

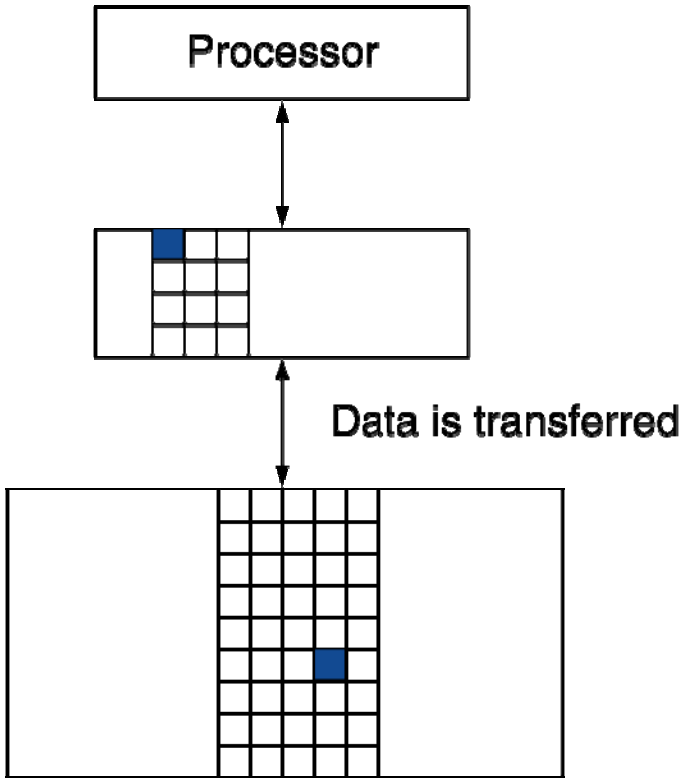
- **Taking advantage of locality**
  - Store everything on disk
  - Copy recently accessed (and nearby) items from disk to smaller DRAM memory (main memory)
  - Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory (cache memory)

Memory Hierarchy (2)



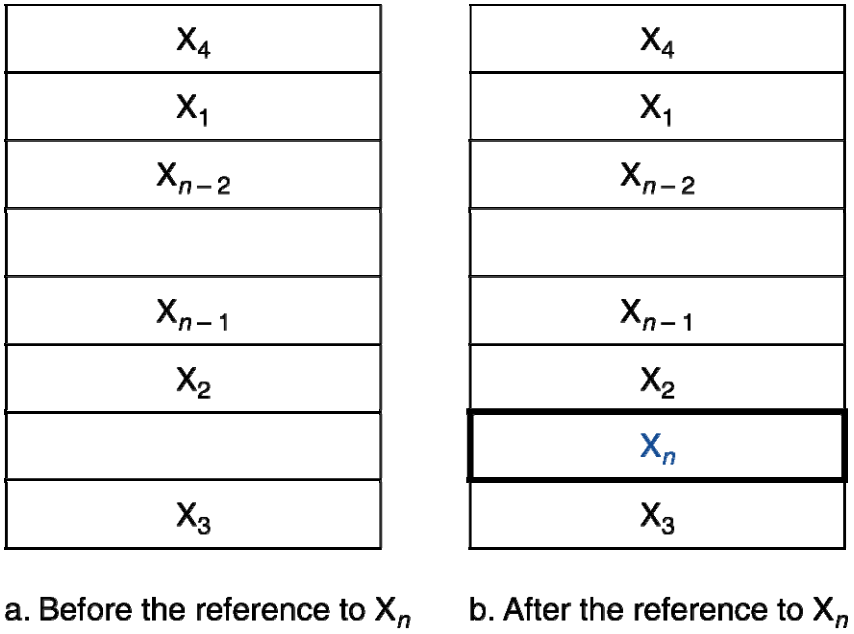
Memory Hierarchy (3)

- **Terminologies**
  - Block (aka line): unit of copying
    - May be multiple words
  - **Hit**: If accessed data is present in upper level, access satisfied by upper level
    - Hit ratio: hits/accesses
  - **Miss**: If accessed data is absent, block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 - hit ratio



Cache Memory (1)

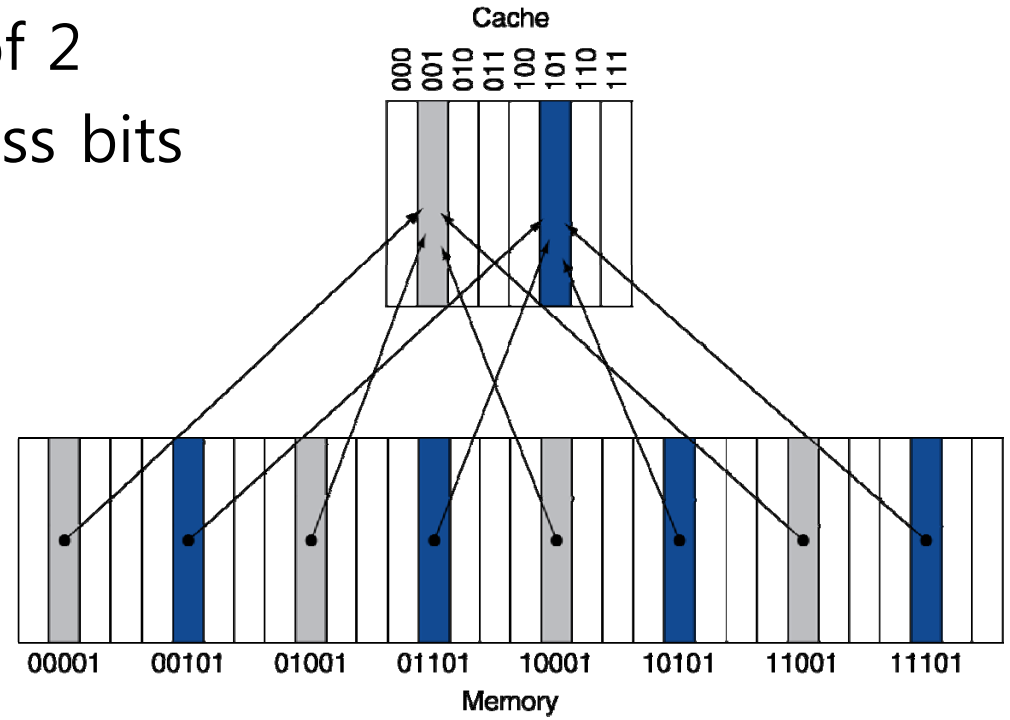
- **Cache memory**
  - The level of the memory hierarchy closest to the CPU
  - Given accesses X_1, …, X_{n-1}, X_n
    - How to we know if the data is present?
    - Where do we look?



a. Before the reference to X_n b. After the reference to X_n

Cache Memory (2)

- **Direct mapped cache**
  - Location determined by address
  - Direct mapped: only one choice
    - (Block address) modulo (#Blocks in cache)
  - #Blocks is a power of 2
  - Use low-order address bits



Cache Memory (3)

- **Tags and valid bits**
  - How do we know which particular block is stored in a cache location?
    - Store block address as well as the data
    - Actually, only need the high-order bits
    - Called the tag

  - What if there is no data in a location?
    - Valid bit: 1 = present, 0 = not present
    - Initially 0

Cache Example (1)

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example (2)

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example (3)

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (4)

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (5)

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

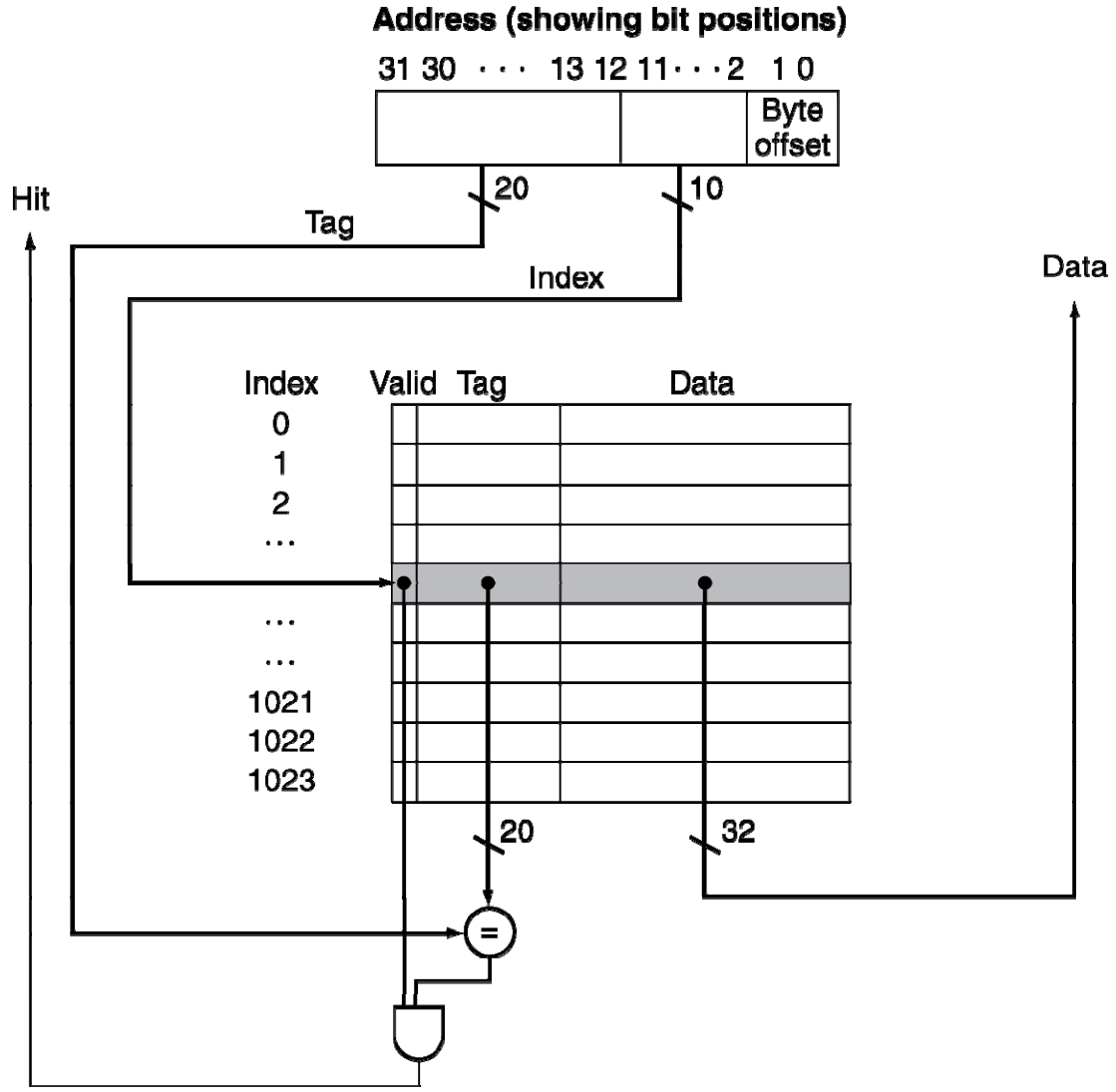
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (6)

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

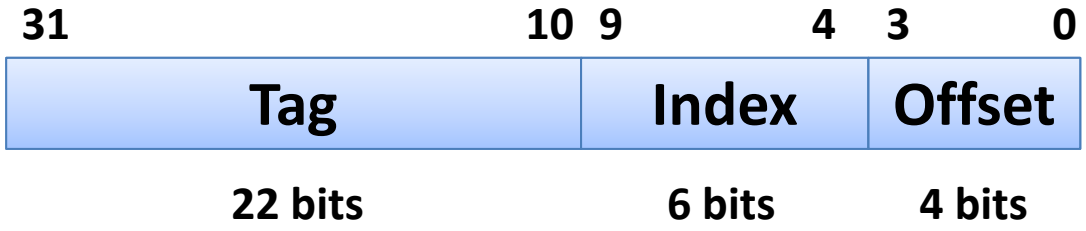
Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision



Block Size (1)

- **Example: larger block size**
  - 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
  - Block address = ⌊1200/16⌋ = 75
  - Block number = 75 modulo 64 = 11



Block Size (2)

- **Considerations**
  - Larger blocks should reduce miss rate
    - Due to spatial locality
  - But in a fixed-sized cache
    - Larger blocks ⇒ fewer of them
      - » More competition ⇒ increased miss rate
    - Larger blocks ⇒ pollution
  - Larger miss penalty
    - Can override benefit of reduced miss rate
    - Early restart and critical-word-first can help

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

Write Policies (1)

- **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 CPU = 1, 10% of instructions are stores, write to memory takes 100 cycles
      - » Effective CPU = 1 + 0.1 x 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 Policies (2)

- **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

Write Policies (3)

- **Write allocation**
  - What should happen on a write miss?
  - Alternatives for write-through
    - Allocate on miss: fetch the block
    - Write around: don't fetch the block
      - » Since programs often write a whole block before reading it (e.g., initialization)
  - For write-back
    - Usually fetch the block

Example: Intrinsity FastMATH

- **Embedded MIPS processor**
  - 12-stage pipeline
  - Instruction and data access on each cycle
- **Split cache: separate I-cache and D-cache**
  - Each 16KB: 256 blocks x 16 words/block
  - D-cache: write-through or write-back
- **SPEC2000 miss rates**
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Example: Intrinsity FastMATH