Credits: Dr. Asanovic (MIT), Dr. Yousif (Intel), Dr. Park (Virginia Tech), Dr. Patterson (UCB)

CpE 313: Microprocessor Systems Design

Handout 07
Memory Hierarchy and Caches

September 21, 2004

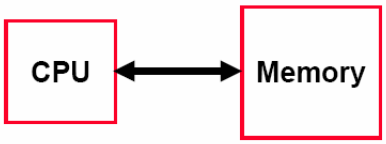
Shoukat Ali

shoukat@umr.edu

UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

1

CPU-Memory Bottleneck



Performance of high-speed computers is usually limited by memory *bandwidth* & *latency*

- ***Latency (time for a single access)***
  Memory access time >> Processor cycle time
- ***Bandwidth (number of accesses per unit time)***
  if fraction ***m*** of instructions access memory,
  ⇒ **1+*m*** memory references / instruction
  ⇒ **CPI = 1 requires 1+*m*** memory refs / cycle

2

Multilevel Memory

Strategy: Hide latency using small, fast memories called caches.

Caches are a mechanism to hide memory latency based on the empirical observation that the stream of memory references made by a processor exhibits locality

	PC
...	96
loop: ADD r2, r1, r1	100
SUBI r3, r3, #1	104
BNEZ r3, loop	108
...	112

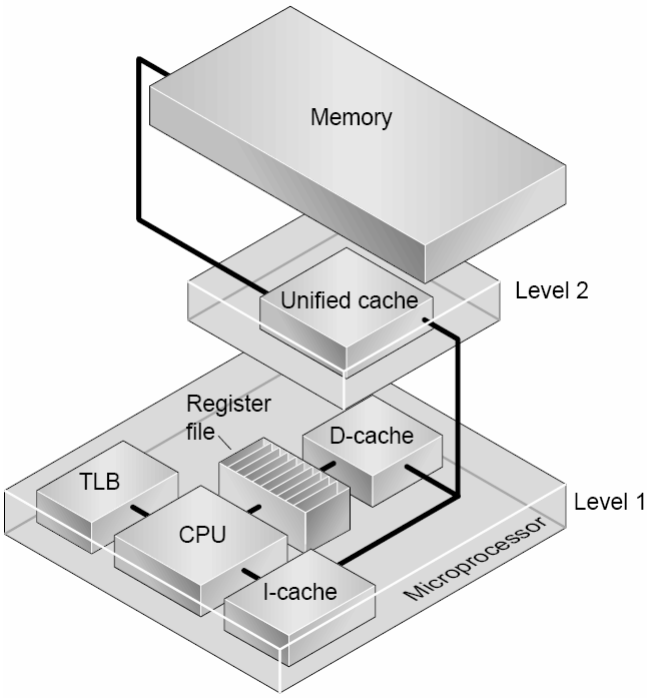
What is the pattern of instruction memory addresses?

3

Principle of Locality

- Principle of locality: Programs access a relatively small portion of their address space at any instant of time
  - Temporal locality (time):
    - If an item is referenced, it will tend to be referenced again soon
      - so keep most recently accessed data items closer to the processor in **cache**
  - Spatial locality (space):
    - If an item is referenced, items whose addresses are close by will tend to be referenced soon
      - so when CPU asks for a word from memory, move a whole ***block*** consisting of contiguous words to the **cache**
- memory hierarchy
  - main memory has all data (like a bookshelf)
  - cache keeps copies of “local” data (like your desk)

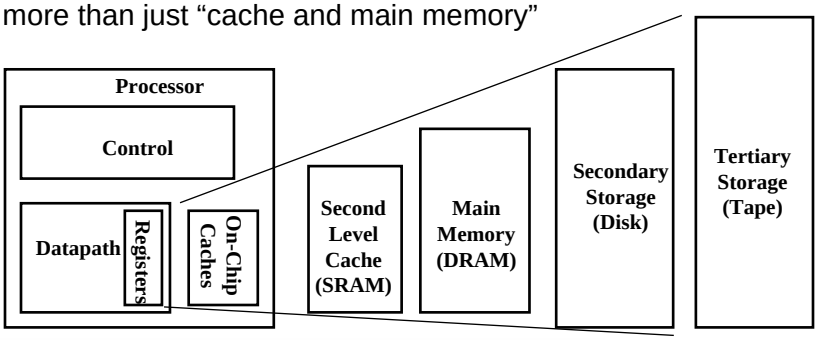
4



5

Memory Hierarchy of a Modern Computer System

- more than just “cache and main memory”



- *size*: Register << SRAM << DRAM *why?*
- *latency*: Register << SRAM << DRAM *why?*
- *bandwidth*: on-chip >> off-chip *why?*

- SRAM costs $70 per megabyte --- $286,720 for MIPS
- DRAM costs $2 per megabyte --- $8,192 for MIPS
- Disk storage: $0.005 per megabyte --- $20 for MIPS

6

Some Examples

Processor	L1 cache
MIPS R5000 MIPS R10000	32K instruction + 32K data
MIPS R20K	32K instruction + 32K data
Alpha 21164	8K instruction + 8K data
Alpha 21264	64K instruction + 64K data
Power 603e	16K instruction + 16K data
Power 604e	32K instruction + 32K data
Power 740 Power 750	32K instruction + 32K data
HP PA-8500 HP PA-8600	512K instruction + 1,024K data
HP PA-8700	768K instruction + 1,536K data

- Pentium II
  - 16 Kbyte L1 I-cache, 16 Kbyte L1 D-cache

7

Reference Slide: Technologies Used in Memory Hierarchies

- Static random access memory (SRAM)
  - Used for caches
  - Low density, high power, expensive, fast
  - Static: content will last until power is lost
- Dynamic random access memory (DRAM)
  - Used for main memory
  - High density, low power, cheap, slow (factor of 5 to 10)
  - Dynamic: need to be “refreshed” regularly
    - Value is stored as a charge on a capacitor (must be refreshed)
- Disk memory (hard disk)
  - slowest, cheapest
  - content will last “forever” (until disk physically fails)

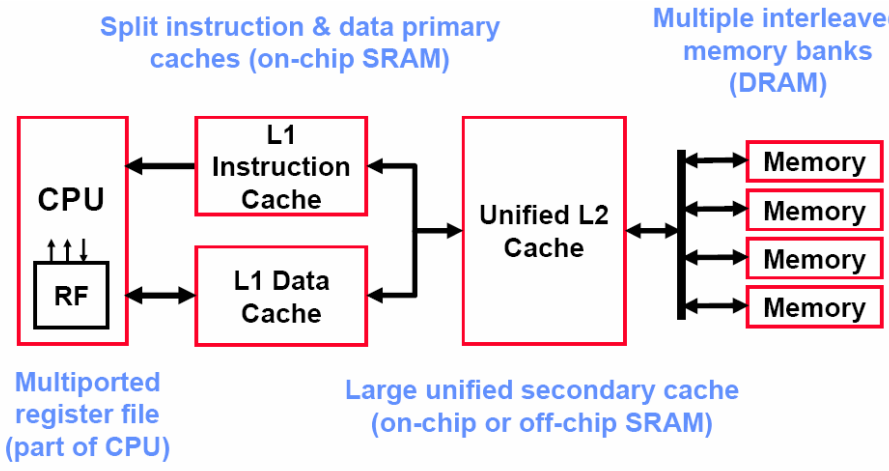
8

Management of Memory Hierarchy

- Software managed, e.g., registers
  - part of the software-visible processor state
  - software in complete control of storage allocation
    - » but hardware might do things behind software's back, e.g., register renaming
- Hardware managed, e.g., caches
  - not part of the software-visible processor state
  - hardware automatically decides what is kept in fast memory
    - » but software may provide "hints", e.g., don't cache or prefetch

9

Split Caches: Why? … Split Memory: Why?



10

Caches: Terminology

- *Block*: Minimum unit of information that is transferred from main memory to cache (why is it not just one word?)
- *Hit*: Data requested by CPU is in some block in the cache
  - *Hit Rate*: The fraction of memory accesses found in cache
  - *Hit Time*: Time to determine if CPU request is a hit PLUS time to read data from cache

    should data be read after "hitting" is confirmed?
- *Miss*: data requested by CPU is not in cache, and must be brought from main memory into cache and then word within block be delivered to CPU
  - *Miss rate*: 1 – Hit Rate
  - *Miss Time* (Miss Penalty): Time to bring the block from main memory to cache PLUS Time to deliver the word to processor
- Hit Time << Miss Penalty

11

Cache Design Issues

UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

12

4 Qs for Cache Designers

- Q1: On a miss, when a new block is brought from memory, where can the block be placed in the cache?
  (Block placement)
- Q2: On a cache access, how does the HW know if the requested block is in the cache?
  (Block identification)
- Q3: On a miss, which block should be replaced to make room for the new block?
  (Block replacement)
- Q4: What happens on a write?
  (Write strategy)

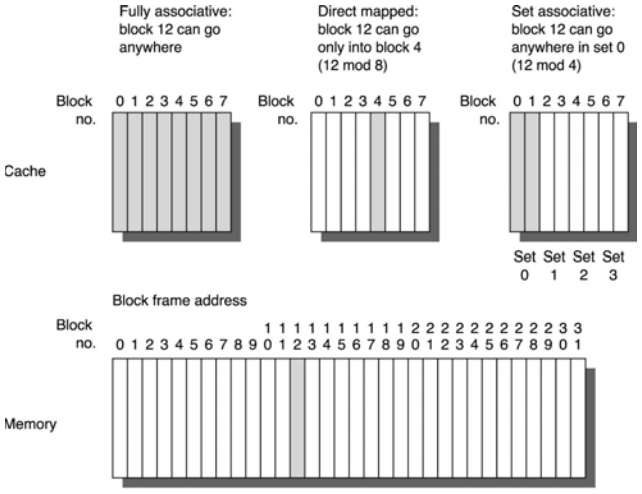
13

Q1: Where Can a Block Be Placed?

- categories of cache organization are based on block placement restriction
  - direct mapped: each block has only one place that it can appear in the cache
    - a place in cache is called a frame or a block frame or just a block
    - frame no. = (Block address) mod (# of frames in cache)
  - fully associative: each block can be placed anywhere in the cache
    - frame number = <<no frame number>>
  - set associative: each block can be placed in a restricted set of frames in the cache
    - a block is first mapped to a set, and then placed anywhere in set
      - set number = (Block address) mod (# of sets in cache)
    - if there are *n* frames in a set, the cache is called n-way set associative cache

14

Example of Set Associativity in Caches



32 blocks in the main memory

Cache size is 8 blocks
Where does Block 12 from memory go?

Fully associative:
Block 12 can go anywhere

Direct mapped:
frame no. =
(Block address) mod (# of frames in cache)
Block 12 can go only into block 4 (because 12 mod 8 = 4)

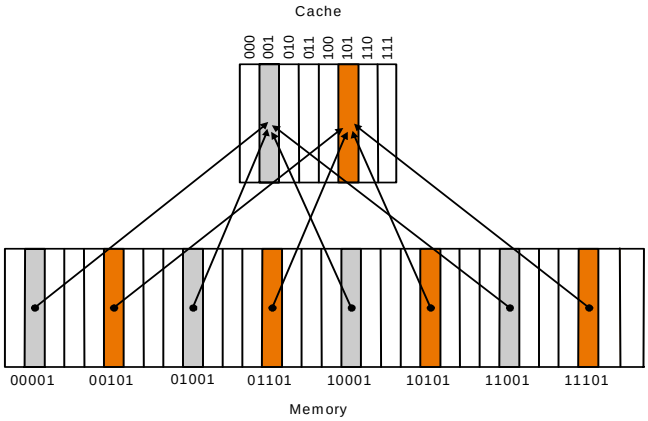
2-way set associative:
Set no. =
(Block address) mod (# of sets in cache)
Block 12 can go in any frame in set 0

why set 0? 12 mod 4 = 0

15

DMC: Memory Block → Cache Frame Number Mapping

- frame number is also called cache index
- cache index = mod or remainder of (block address) / (# of frames in cache)
  - trick: index = lower *n* bits of block address where $n = \log_2$(# of frames in cache)



- 1/8, 9/8, 17/8, 25/8 all have remainder of 1
- so blocks 1, 9, 17 and 25 all get stored in frame number 1
  - ***one at a time***
- what happens when block 25 is in cache, and CPU asks for block 9?

16

Example

why are last 3 bits raised?

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$

what happened here?

hit rate = ?

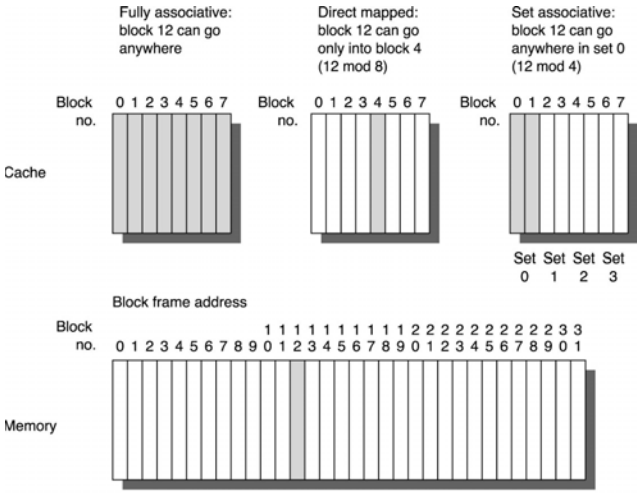
miss rate = ?

what do I do in real-life to find miss rate of a program?
address traces
special miss counters in some microprocessors

17

Cache Index Size: In General

- cache frame number and its size in bits?



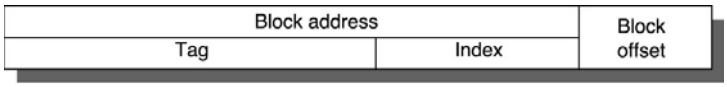
- Z is cache size ***in frames***
- FAC
  - no number
  - 0 bits
- DMC
  - baddr mod Z
  - $\log_2(Z)$ bits
- N-way associative
  - any in set S; S = baddr mod (Z/N)
  - $\log_2(Z/N)$ bits

18

Q2: How Is A Block Found If It Is In Cache?

PRELIMINARY INFO

- CPU generates address A for a word (not for a block)
  - location of word within a block is specified by bits called *block offset*
  - block addr = "A" without lower $\log_2$(block size) bits
- lower I bits of baddr are index bits
  - index bits select a set in cache (for DMC, consider each frame to be a set)
    - index size = $\log_2$ (cache size in frames / associativity)
- ***tag: A without index bits and without block offset bits***

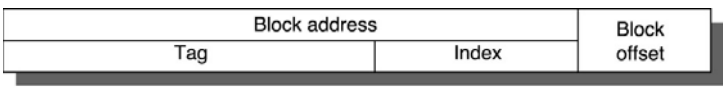


- tag-index boundary moves to right with _____ associativity

19

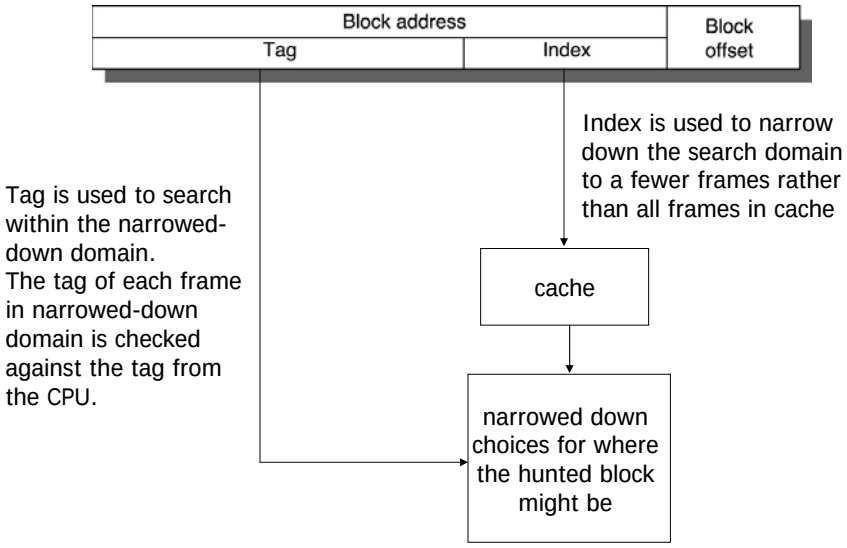
Q2: How Is A Block Found If It Is In Cache? - Detail

- when caches store a memory block in a cache frame, they also store the tag of the memory block in a part of cache called "tags"
- when the CPU asks for a word at address A,
  - the bits of A are split in three fields:
    - block-offset$_A$, i.e., lower $\log_2$(block size) bits of A
    - index$_A$, i.e., lower n bits of baddr
      - n = $\log_2$ (cache size in frames / associativity)
    - tag$_A$ = remaining bits of A
  - cache set at index$_A$ is accessed
  - tags of all frames in set are searched to see if tag$_A$ = stored tag
    - if yes, the block sought by CPU is in cache



20

Q2: How Is A Block Found If It Is In Cache? - Outline



21

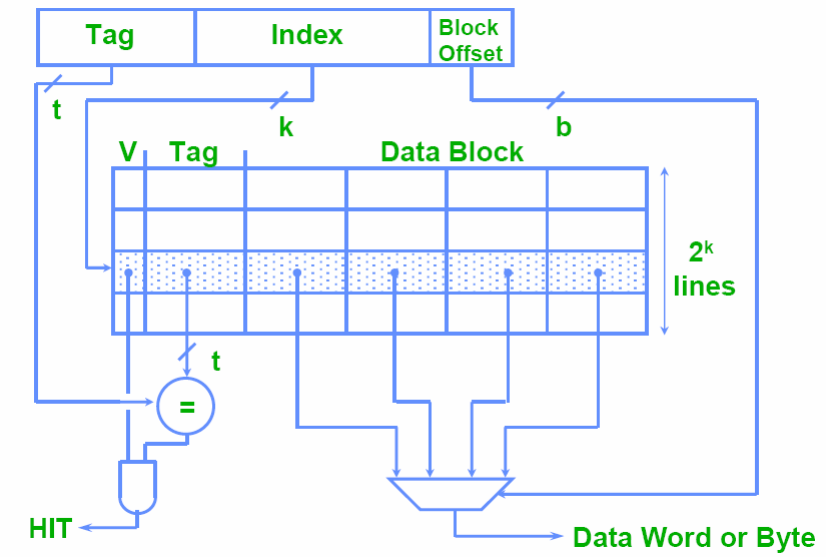
Q2: Wait …

- some cache entries might contain junk (e.g., after processor startup)
- how can we determine whether a cache entry contains valid information?
- a valid bit is added to the tag of a cache to indicate whether the block is valid

- revised block search procedure
- when the CPU asks for a word at address A,
  - <blah blah from previous slide>
  - tags of all frames in set are searched to see if tag_A = stored tag
    - block sought by CPU is in cache if tag_A = stored tag ***and*** the valid bit is set

22

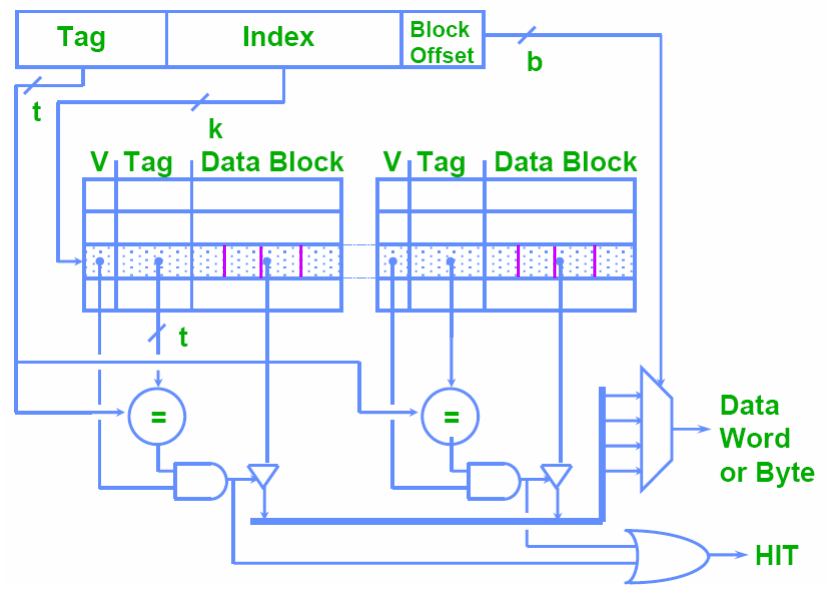
Example: Direct Mapped Cache



Hit time = Time to determine if CPU request is a hit
PLUS time to read data from cache

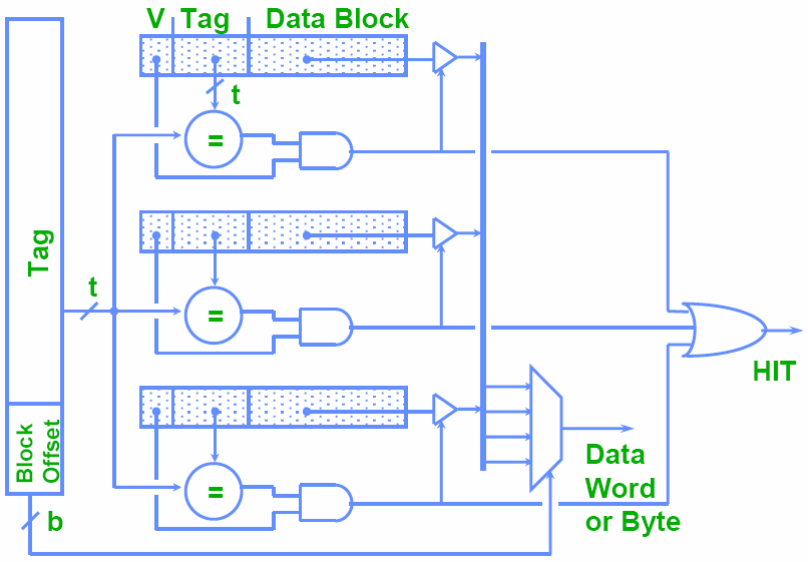
23

Example: 2-Way Associative Cache



24

Example: Fully Associative Caches

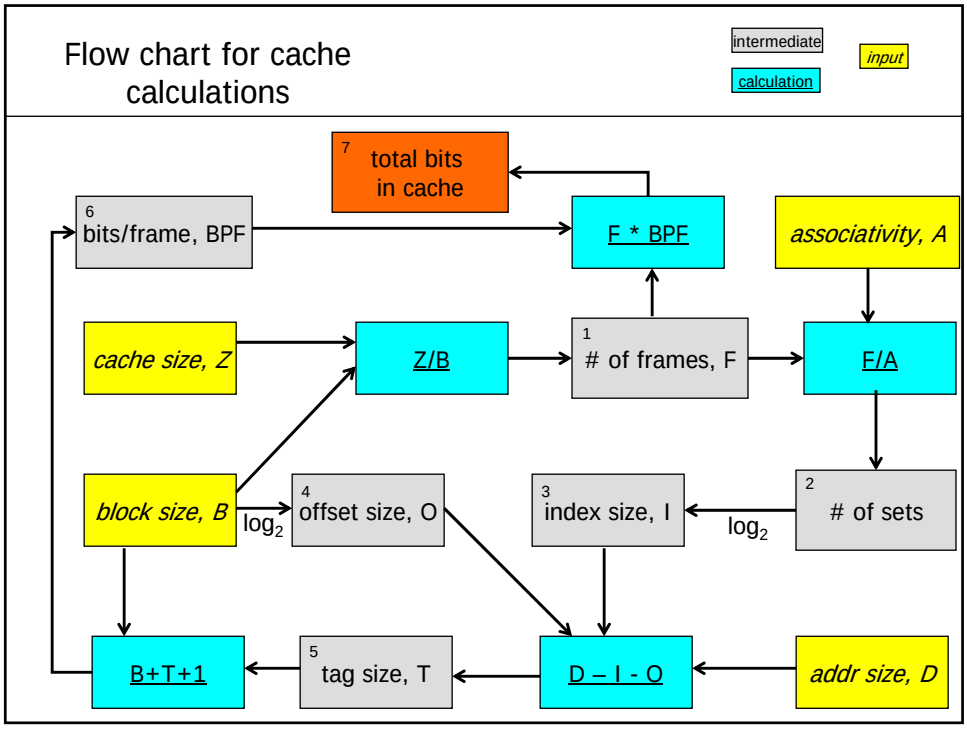


25

Calculating Bits in Cache: Example 1

- How many total bits are needed for a direct-mapped cache with 64 KBytes of data and one word blocks, assuming a 32-bit address?
  - #frames = cache size / block size = (64 * 1024) /4 = 16384 = 2^{14} frames
  - #sets = # frames / associativity = $2^{14}/1 = 2^{14}$
    - index size = $\log_2$(#sets) = 14 bits
  - block size = 4 bytes
    - offset size = $\log_2$(block size) = 2 bits
  - tag size = address size - index size - offset size
    = 32 - 14 - 2 = 16 bits
  - bits/frame = data bits + tag bits + valid bit
    = 32 + 16 + 1 = 49
  - total bits in cache = #frames x bits/frame
    = 2^{14} x 49 bits = 49 x 2^1 x 2^{10} x 2^3 = 98 Kbytes

26



Calculating Bits in Cache: Example 2

- How many total bits would be needed for a 4-way set associative cache to store the same amount of data?
  (64 KBytes of data and one word blocks)
  - #frames = 2^{14} = same as in Example 1
  - #sets = #frames /4 = $(2^{14})/4 = 2^{12}$
    - index size = $\log_2$(#sets) = 12 bits
  - offset size = 2 bits = same as in Example 1
  - tag size = address size - index size – offset
    = 32 - 12 - 2 = 18 bits
  - bits/frame = data bits + tag bits + valid bit
    = 32 + 18 + 1 = 51
  - bits in cache = # frames x bits/frame = 2^{14} x 51 bits = 102 Kbytes
- *lesson: increasing associativity increases total cache size*

28

Calculating Bits in Cache: Example 3

- how many total bits are needed for a direct-mapped cache with 64 KBytes of data and 8 word blocks, assuming a 32-bit address?
  - #frames = cache sz / blk sz = (64 x 1024)/32 = 2 x 1024 = 2^{11} frames
  - #sets = # frames / associativity = $2^{11}/1 = 2^{11}$
    - index size = $\log_2$(#sets) = 11 bits
  - block size = 32 bytes
    - offset size = $\log_2(32)$ = 5 bits
  - tag size = address size - index size - offset size
    = 32 - 11 - 5 = 16 bits
  - bits/ frame = data bits + tag bits + valid bit
    = 8x32 + 16 + 1 = 273 bits
  - bits in cache = # frames x bits/ frame = 2^{11} x 273 bits
    = 68.25 Kbytes
- *lesson: increasing block size decreases total bits in cache*
- is there another benefit to larger block sizes?
- what happens when the block size is too large?

29

Issues Related to Block Size

- Miss rate might increase if block size becomes significant fraction of cache size because
  - # of blocks that can be held in the cache will become small
  - This will increase the competition for those blocks
  - As a result, a block will be bumped out of the cache before many of its words are accessed
- Is there another problem associated with just increasing the block size?
  - Cost of a miss increases: Miss penalty increases with larger block sizes

30