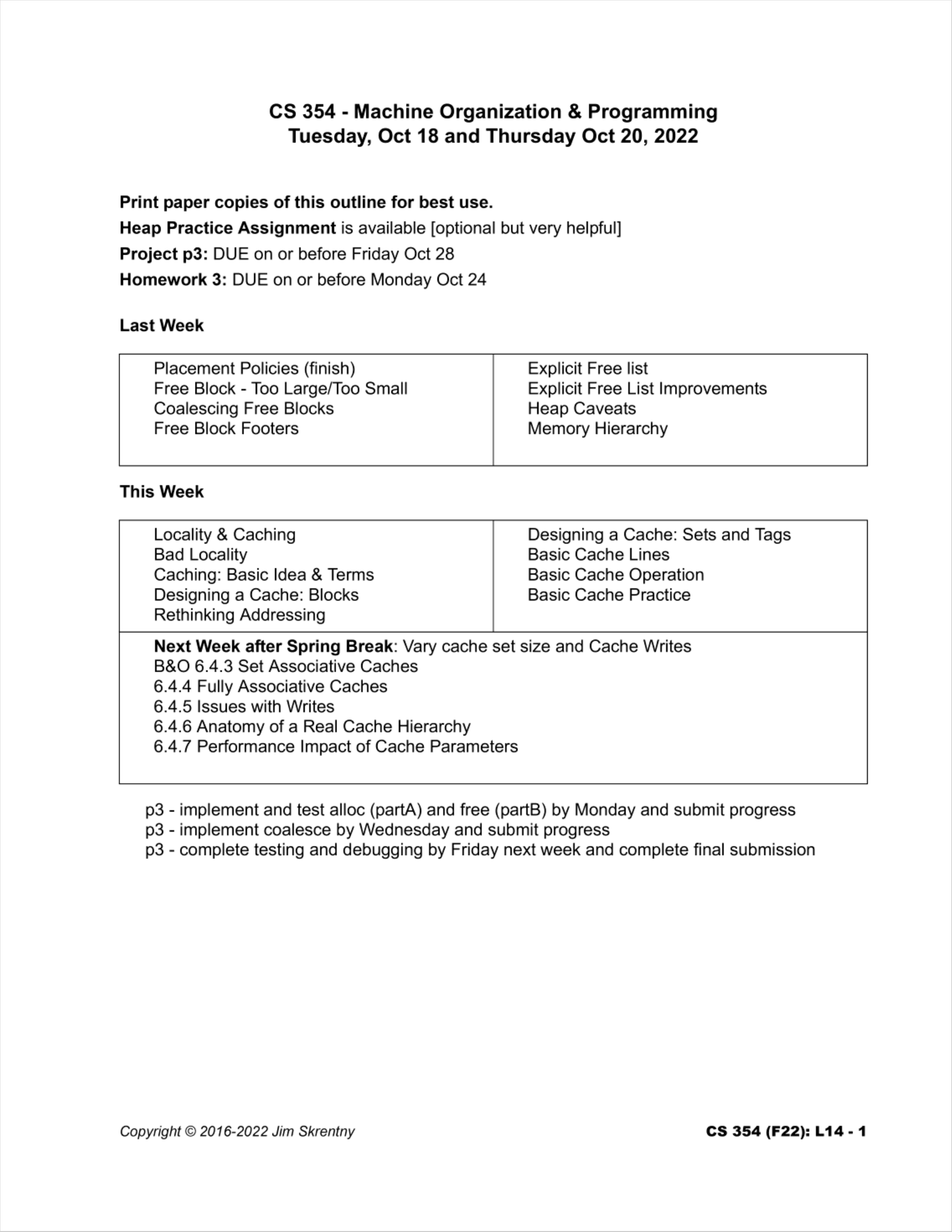
outlineL14-w7TR-student

Monday, October 31, 2022 6:41 PM

outlineL14-w7TR-student



CS 354 - Machine Organization & Programming
Tuesday, Oct 18 and Thursday Oct 20, 2022

Print paper copies of this outline for best use.

Heap Practice Assignment is available [optional but very helpful]

Project p3: DUE on or before Friday Oct 28

Homework 3: DUE on or before Monday Oct 24

Last Week

- Placement Policies (finish)
- Free Block - Too Large/Too Small
- Coalescing Free Blocks
- Free Block Footers
- Explicit Free list
- Explicit Free List Improvements
- Heap Caveats
- Memory Hierarchy

This Week

- Locality & Caching
- Bad Locality
- Caching: Basic Idea & Terms
- Designing a Cache: Blocks
- Rethinking Addressing
- Designing a Cache: Sets and Tags
- Basic Cache Lines
- Basic Cache Operation
- Basic Cache Practice

Next Week after Spring Break: Vary cache set size and Cache Writes
B&O 6.4.3 Set Associative Caches
6.4.4 Fully Associative Caches
6.4.5 Issues with Writes
6.4.6 Anatomy of a Real Cache Hierarchy
6.4.7 Performance Impact of Cache Parameters

p3 - implement and test alloc (partA) and free (partB) by Monday and submit progress
p3 - implement coalesce by Wednesday and submit progress
p3 - complete testing and debugging by Friday next week and complete final submission

Locality & Caching

What?

temporal locality: when recently used memory is accessed in the near future

spatial locality: when recently used memory is followed by accessing nearby memory

locality is designed into Hardware, OS, and applications

Example

```
int sumArray(int a[], int size, int step) {
   int sum = 0;
   for (int i = 0; i < size; i += step)
      sum += a[i];
   return sum;
}
```

→ List the variables that clearly demonstrate temporal locality. Sum, I, Steps, Size

→ List the variables that clearly demonstrate spatial locality. array a[i] if the step size is small

stride: step size in words between sequential access

good spatial locality ~ 1 word / stride length

✵ *The caching system uses localityto predict what the cpu will need in the near future.*

How? Caching system anticipates 2 things:

1. - data will be reused, save in a cache
2. - nearby data will be used, save a block of data in cache

cache block: unit of memory transferred between MM and cache levels

IA-32 block size = 32 bytes/block

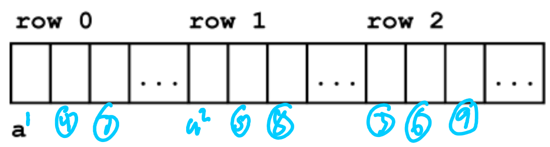
✵ *Programs with good localityrun faster since they work better with the caching system!*

Why? Programs with good locality maximize the use of data at TOP of mem hierarchy

Bad Locality

Why is this code bad?

Bad Locality



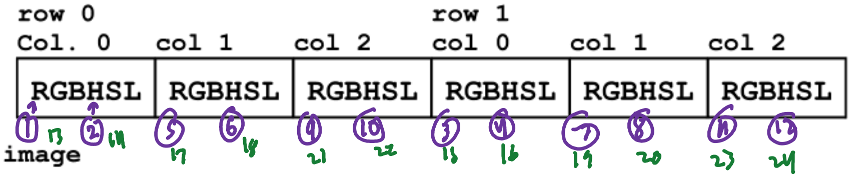
```
       ?      ?
int a[ROWS][COLS];

for (int c = 0; c < COLS; c++)
   for (int r = 0; r < ROWS; r++)
      a[r][c] = r * c;
```

→ How would you improve the code to reduce stride? flip rows + columns loops

Key Questions for Determining Spatial Locality:

1. What does the memory layout look like for the data? 2D SAA is in row-major order
2. What is the stride of the code across the data? # of columns → smaller stride would be better

Why is this code bad?



```
struct {
   float rgb[3];
   float hsl[3];
} image[HEIGHT][WIDTH];
        2      3

for (int v = 0; v < 3; v++)
   for (int c = 0; c < WIDTH; c++)
      for (int r = 0; r < HEIGHT; r++) {
         image[r][c].rgb[v] = 0;
         image[r][c].hsl[v] = 0;
      }
```

stride is 3, 15, 3, 15

➢ How would you improve the code to reduce stride? Change loops:
1. row
2. for each col
3. [for each v rgb]
   [for each v hsl]

Better spatial locality

jumping one instead of 3

Good or bad locality?

- Instruction Flow:
  - sequencing? Good spatial locality, bad temporal locality
  - selection? Bad spatial + temporal
  - repetition? good if small stride, temporal is good
- Searching Algorithms:
  - linear search Array → Good spatial, array access is bad so bad temporal
  - Linked-list Bad spatial + Bad temporal
  - binary search Array - Bad spatial (jumping around), same temporal to track progress too

Copyright © 2016-2022 Jim Skrentny

Caching: Basic Idea & Terms

Assume: Memory is divided into 32 byte blocks and all blocks are already in main memory.
Cache L1 has 4 locations to store blocks and L2 has 16 locations to store blocks.

→ Update the memory hierarchy below given blocks are accessed in this sequence:
~~22~~,~~11~~,~~22~~,~~44~~,~~11~~,~~33~~,~~11~~,~~22~~,~~55~~,~~27~~,~~44~~ , ~~11~~
C C H1 C H1 C H1 H1 X F H2 H1

(M) *cache miss* Block not found in cache, must search lower

(C) cold miss available locations, empty or invalid

(X) capacity miss no available locations, cache is too small for working set

(F) conflict miss when 2 or more blocks map to the same location.

(H) *cache hit* faster memory access, occurs when block is in the cache

(P) *placement policies*

P1 1. Unrestricted

P2 2. Restrictive

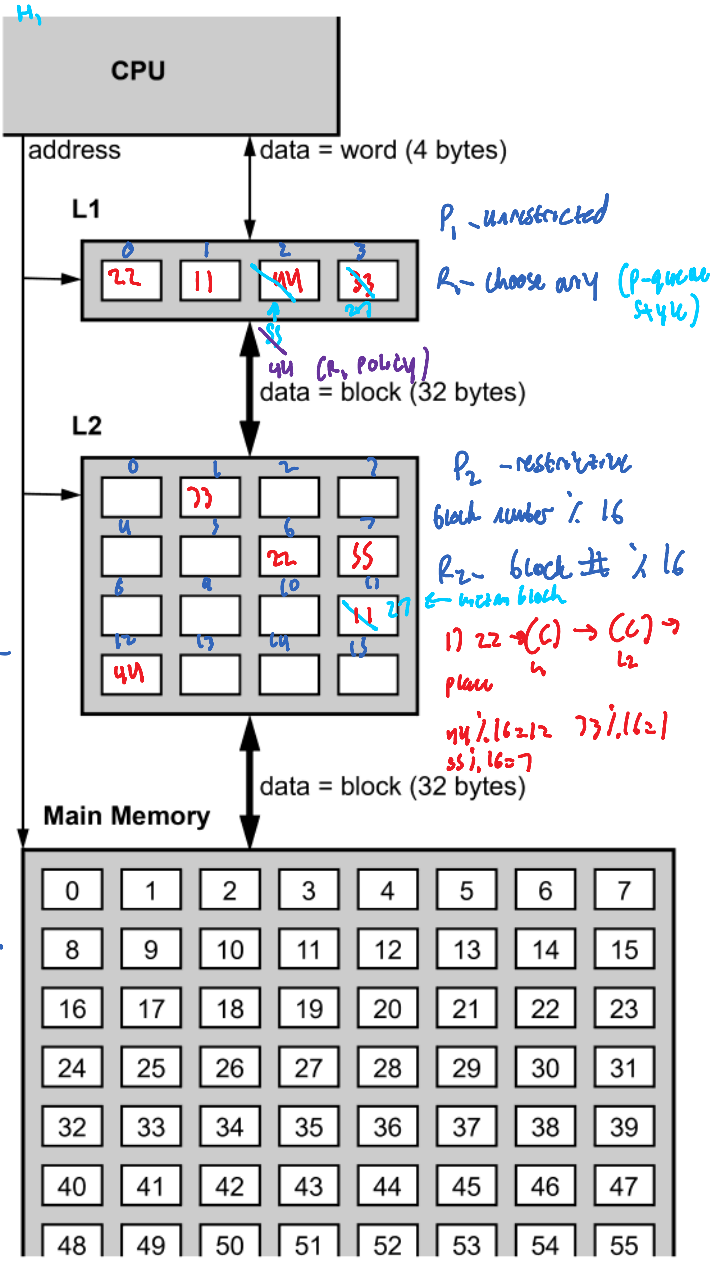
replacement policies

R1 1. choose any location

R2 2. no choice, block maps to specific loc.

victim block Cache block chosen to be replaced

(W) *working set* set of blocks used during some interval of time



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55

P1 - unrestricted

R1 - choose any (P-queue style)

P2 - restrictive
block number % 16

R2 - block # % 16

← victim block

1) 22 → (C) → (C) →
L1 L2
place

44 % 16 = 12 33 % 16 = 1
55 % 16 = 7

Copyright © 2016-2022 Jim Skrentny

Designing a Cache: Blocks

❋ *The bits of an address* are used to look up if block containing that addr is in cache.

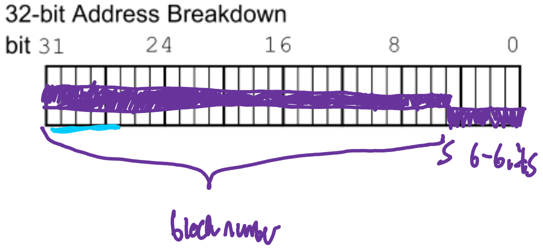
How many bytes in an address space?

Let M be # bytes in addr. space.

IA-32 AA=4GB 4GB = 2^{32} $\log_2(4GB) = 32$

$M = 2^m$
$m = \log_2 M$

Thus m is num of bits in an addr


32-bit Address Breakdown

How big is a block? # bytes/block

❋ *Cache blocks must be big enough* to capture spatial locality
but small enough to minimize latency.

Let B be # bytes/block IA-32 is 32 bytes/block

$B = 2^b = 32$
$b = \log_2 B = 5$ bits

b bits: # of addr bits needed to determine which addr in the block

word offset identifies which word in block contains desired byte

byte offset identifies which byte within that word

➢ What is the problem with using the most significant bits (left side) for the b bits?

Can't utilize spatial locality

How many 32-byte blocks of memory in a 32-bit address space?

$\frac{A.S.}{D.B} = \frac{2^{32}}{2^5} = 2^{(32-5)} = 2^{27} = 2^7 \cdot 2^{20} = 128$ M "32-byte" blocks in 2^{32} A.S.

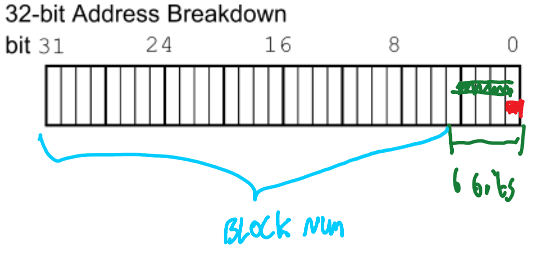
134,217,728

Recall = $K = 2^{10}$ $M = 2^{20}$ $G = 2^{30}$

❋ *The remaining bits of an address* encode the block number

Rethinking Adressing

- An address identifies which byte in VAS to access
- An address is divided into parts to access memory in steps



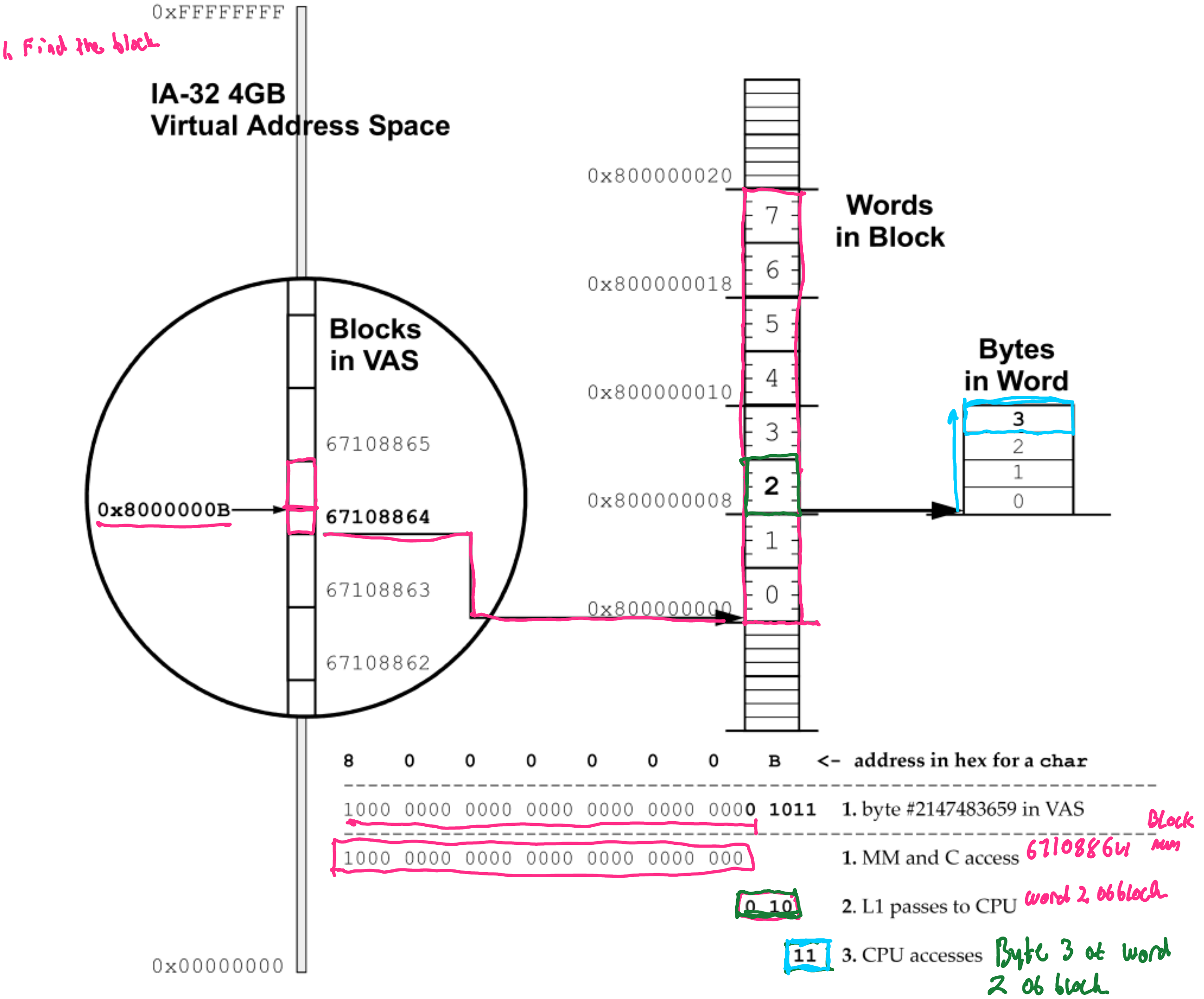
Memory Access in Caching System

step 1. Identify which BLOCK in VAS

step 2. Identify which WORD in the block

step 3. Identify which BYTE in WORD

1. Find the block



```
8    0    0    0    0    0    0    B     <- address in hex for a char
-----------------------------------------------------------------
1000 0000 0000 0000 0000 0000 0000 1011   1. byte #2147483659 in VAS
-----------------------------------------------------------------
1000 0000 0000 0000 0000 0000 000         1. MM and C access 67108864 Block num
                                 0 10     2. L1 passes to CPU word 2 of block
                                      11  3. CPU accesses Byte 3 of word 2 of block
```

Designing a Cache: Sets & Tags

- *A cache must be searched* if unrestricted placement policies is used

→ Problem? Slow O(N) where N is # locations in the cache

Improvement? Limit where block can be stored

set: Of locations where block is uniquely mapped in cache

- *The block number bits of an address* are divided into parts

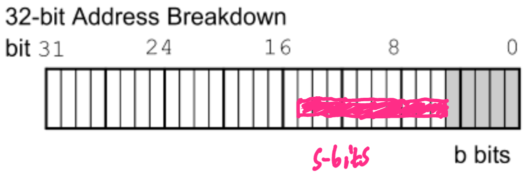
1. maps block number to a set number, set num
2. uniquely identify a particular block, tag

How many sets in the cache?

Let S be # sets in the cache

$S = 2^s$ = 1024

$s = \log_2 S$ = 10 bits



s bits: the bits of an addr that identity which set the block maps to

➤ What is the problem with using the most significant bits (left side) for the s bits?

LOSE SPATIAL LOCALITY

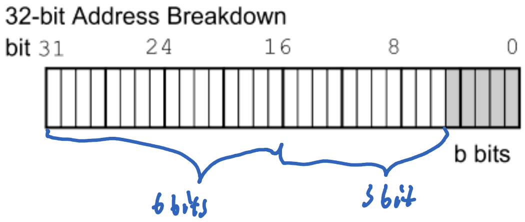
→ How many blocks map to each set for a 32-bit AS and a cache with 1024 sets? 8192 sets?

$$\# \text{ blocks} = \frac{A.S.}{B} \quad \frac{2^{32}}{2^5} = 2^{27}$$

$$\# \text{ blocks/set} = \frac{\# \text{ blocks}}{\# \text{ sets}} = \frac{2^{27}}{2^{10}} = 2^{27-10} = 2^{17} = 2^7 \cdot 2^{10} = 128K$$ blocks mapped to each set.

Since different blocks map to the same set how do we know which block is in a set?

use the remaining bits as a "tag"



t bits: the bits of addr that identify the blocks

- *When a block is copied into a cache* its t-bits are also stored as it's tag

Copyright © 2016-2022 Jim Skrentny

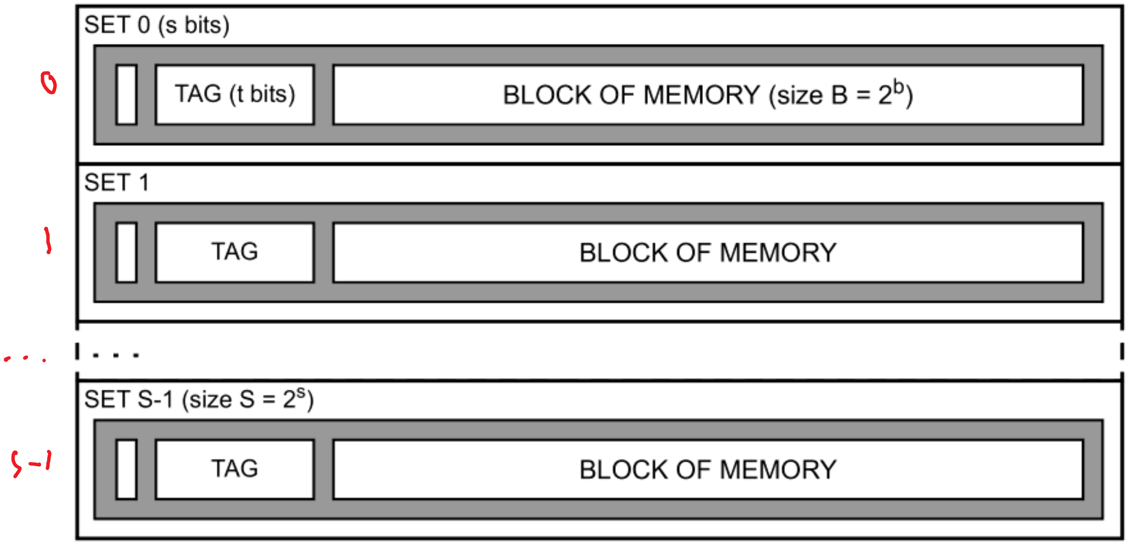
Basic Cache Lines

What? A *line* is

- a location in cache that stores one block of memory
- composed of storage for block bytes and info needed for cache operations,

⋇ *In our basic cache each cache set* has only 1 line.

Basic Cache Diagram



→ How do you know if a line in the cache is used or not? use a status bit (valid bit)

if $V==0$ not valid cache line

if $V==1$ valid line for current process

→ How big is a basic cache given S sets with blocks having B bytes?

$S=1024$

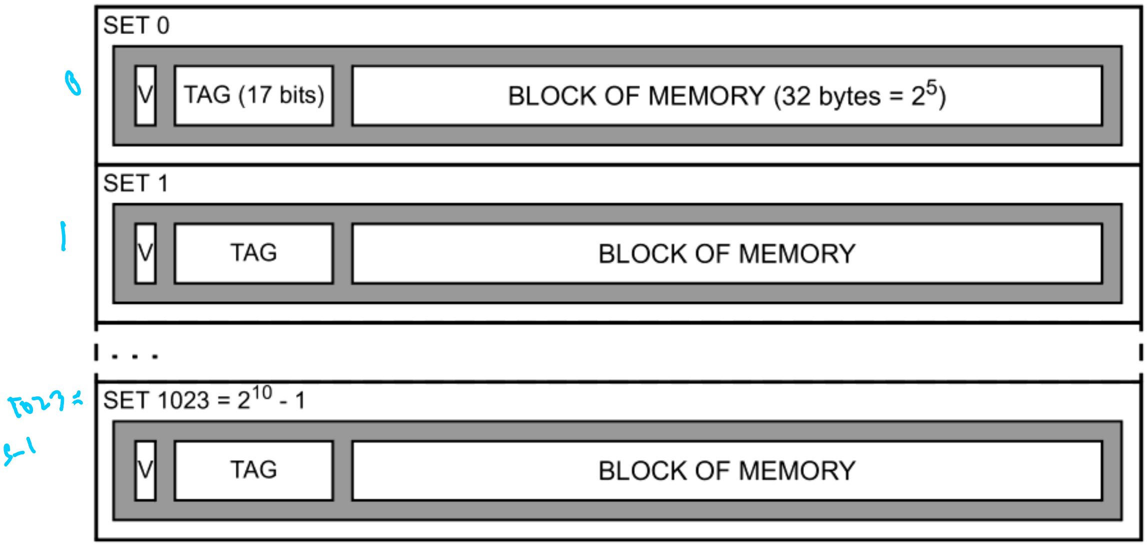
$B=32$

$C=$ cache size $= S \times B$

$= 2^{10} \cdot 2^5 = 2^{15} = 32K$

Basic Cache Operation

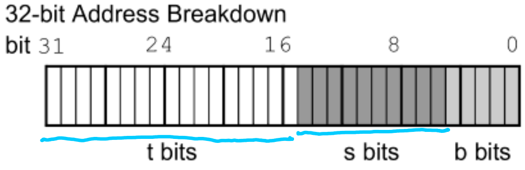
Basic Cache Diagram



→ How big is this basic cache?

$S \times B = 1024 \cdot 32 = 32K = 32{,}768$ bytes

How does a cache process a request for a word at a particular address?



1. *Set Selection* Identify set, extract the s bits and shift to — to get int index

2. *Line Matching* extract t-bits, compare t-bits with stored tag

if no match or valid bit is 0

Cache Miss!

fetch from next lower cache level

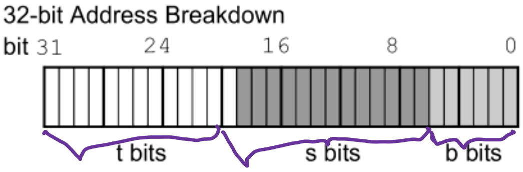
if match and valid bit is 1

Cache hit!

For L1 cache only - Must now extract block words, and then bytes from the word

Basic Cache Practice

You are given the following 32-bit address breakdown used by a cache:



→ How big are the blocks?

$B = 2^6 = 64$ bytes

→ How many sets?

$S = 2^5 = 2^{13} = 2^3 \cdot 2^{10} = 8K$ sets = 8192 sets/ cache

→ How big is this basic cache?

$C = S \cdot B = 2^6 \cdot 2^{13} = 2^{19} = 512K$

Assume the cache design above is given the following specific address: `0x07515E2B`

. turn hex into binary → 0b 0 7 5 1 5 E 2 B

$\underbrace{0000\ 0111\ 0101\ 0}_{T\text{-Bits}}\underbrace{001\ 0101\ 1110\ 00}_{S\text{ Bits}}\underbrace{10\ 1011}_{B\text{ Bits}}$

→ Which set should be checked given the address above? → S-Bits

001 0101 1110 00 ⟹ 1400 = S 0-1399 set at address 1400

(set were looking for)

→ Which word in the block does the L1 cache access for the address?

$1010 \to 10$ word 10 of block

➢ Which byte in the word does the address specify?

$11 \to 3$ byte 3 of word 10 in block

Assume address above maps to a set with its line having the following V status and tag.

→ Does the address above produce a hit or miss?

$\underbrace{0\,1000}\underbrace{0\,111\,0}\underbrace{1010}$

V tag

1.) 1 0x0750 miss, wrong tag

2.) 0 0x0750 miss, wrong tag

3.) 1 0x00EA hit, correct

4.) 0 0x00EA miss, invalid tag