Cache

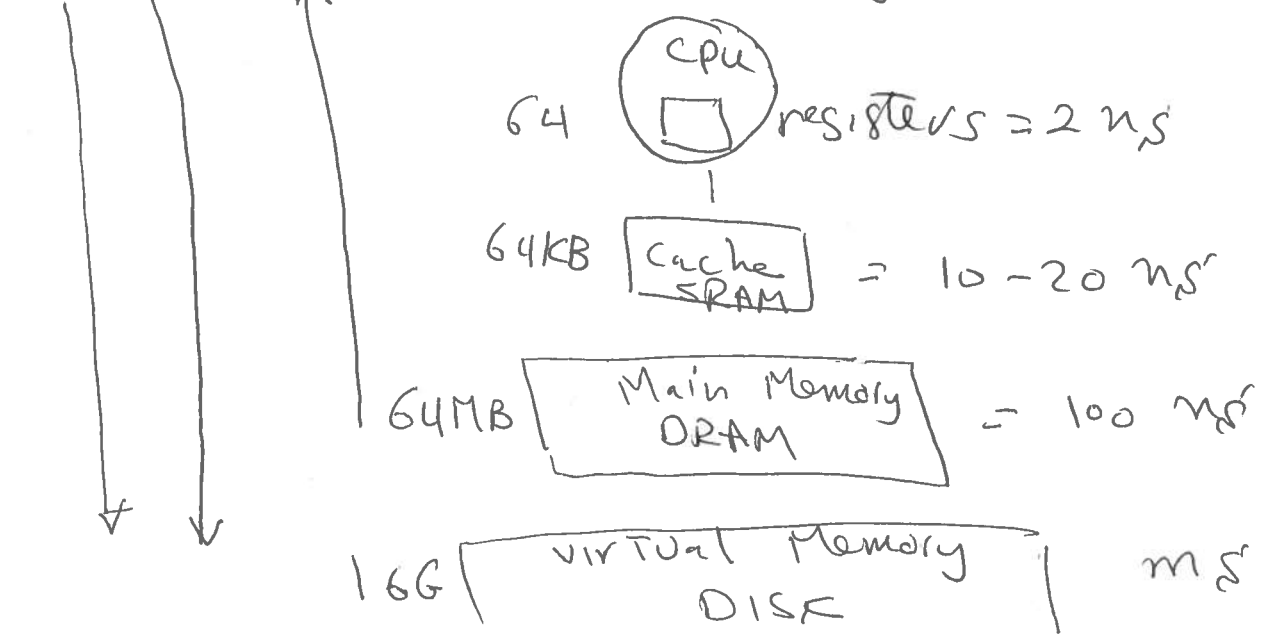
1- Principle of Locality
2- Terminology
3- Direct Mapped Cache

Cache Concept

- cpu executes 10% of code 90% of the time (locality).
- cpu is very fast (2 ns for 500 MHz) DRAM is slow (100 ns), each memory access takes 50 cpu cycles.
- Small parts of code & Data could be stored in fast SRAM, so cpu will execute at rated speed.
- ∴ Cache works because of principle of locality.

Memory Hierarchy

price size speed



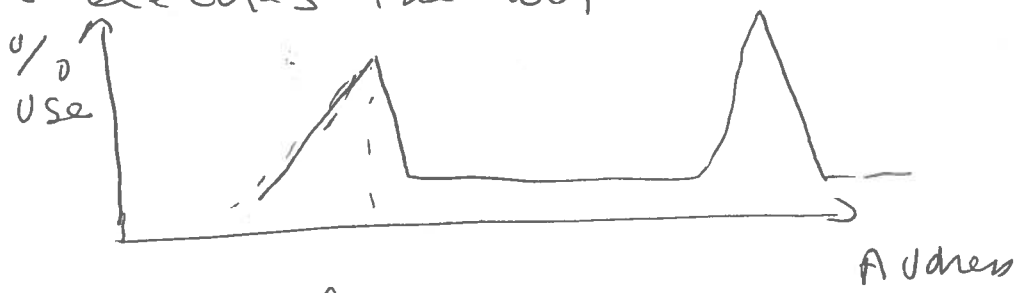
Cache Concept

Principle of Locality

90% of Time executing 10% of Code

Example: for (i=0; i<1000; i++)
a[i]=0;

few instructions executes the loop 1000 times.



If we move mfu references to fast SRAM (cache), we can improve performance of 90% of CPU Time by a factor of DRAM speed / SRAM speed
≈ 10

→ How about the 10%? Amdahl's Law?

Two types of Localities!:

1- Spatial locality: (near in space)
references close by in space (addresses) are more likely to be referenced.
a[i] and a[i+1]

2- Temporal locality: (in time)
items referenced now are more likely to be referenced again soon
a[i] = i instructions.

Cache terminology

Hit rate: percentage of references found in cache

Miss rate: = (1 − hitrate): references not found in cache and must be retrieved from main memory

Hit Time: Access time to cache + time to determine hit/miss

Miss penalty: Time to replace a block in cache from main memory + time to deliver reference to processor

Block: Unit of Information to be transfered (present or not present) between two levels

Example of a cache parameters:

Cache size = 32 Kbyte, hit rate = 95%,
Cache Time = 10 ns, miss penalty = 100ns
Block size = 32 Bytes.

Direct Mapped Cache

- Mapping, finding a block in cache
- Organization
- Performance

Mapping

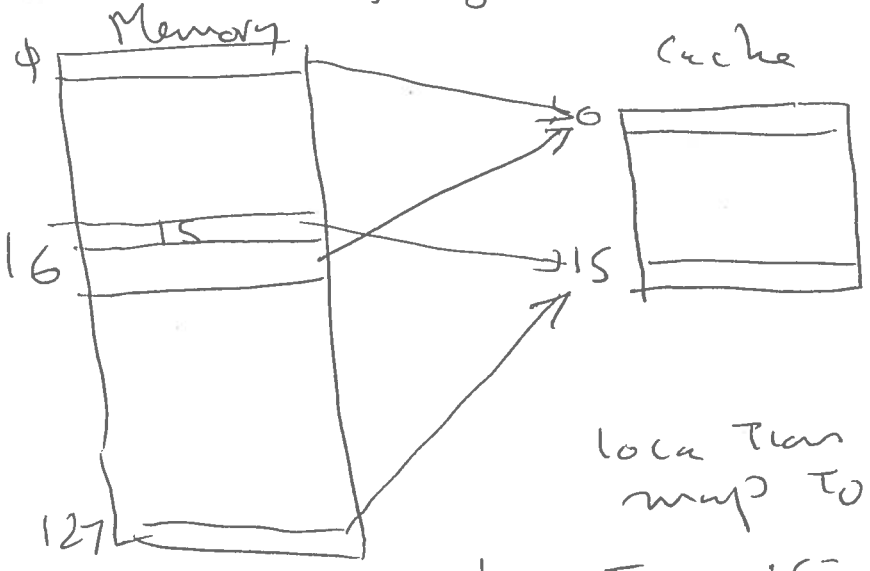
Each item in memory maps to one location in cache.

This location = (Memory Address) modulo of number of cache blocks

Finding a block in Cache:

Use a tag for each block, and compare new access Tag with the stored Tag in cache. If it is the same = cache hit

Example memory size = 128, cache size = 16



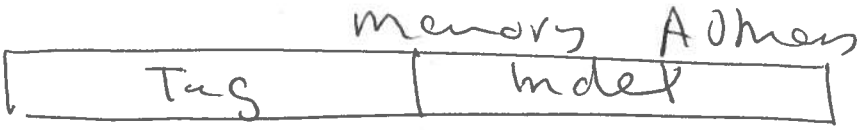
location 0, 16, 32, ... map to cache loc 0

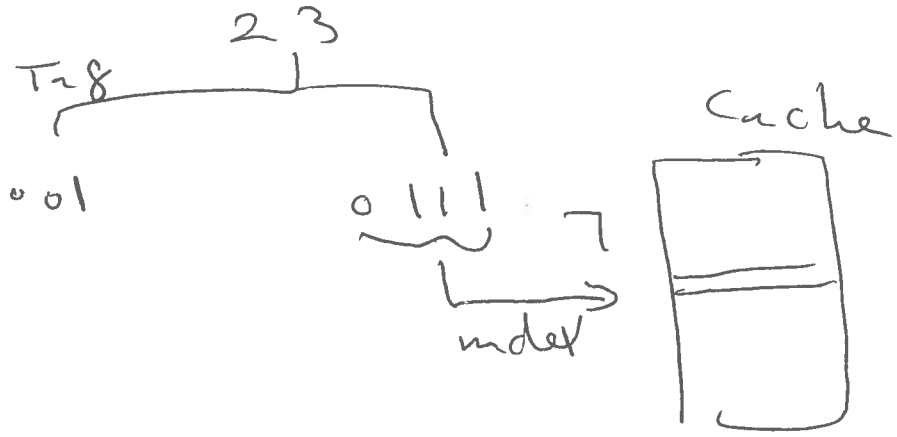
location 15, 31, ... map to 15

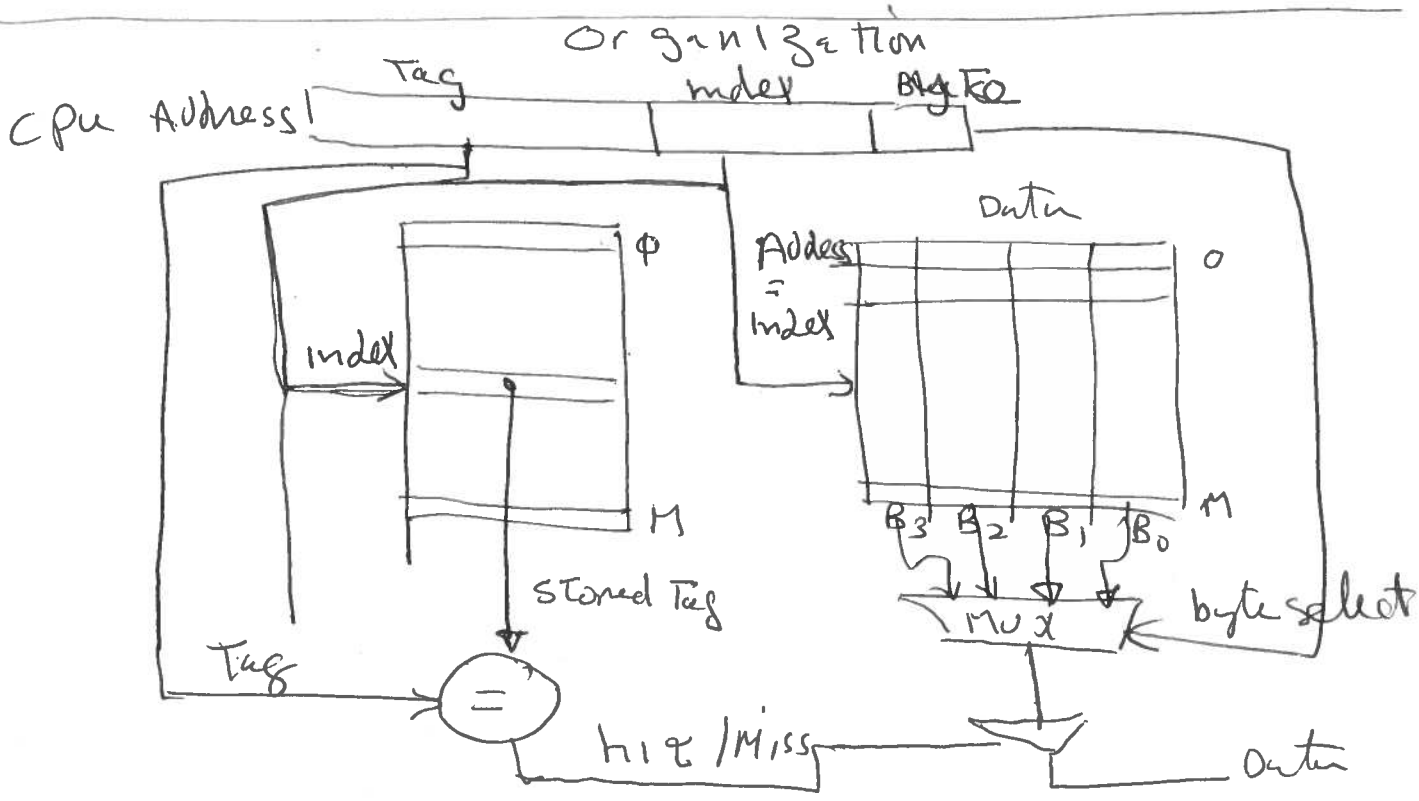
# of Pages or Frames = $\frac{128}{16} = 8$

Address 23 maps To 23 Module 16
= 7

Find if block in cache using Tag (Page #)







operation

If Tag(index) = Tag of current address
    hit = true
    data = Data(index)

both Tag, Data are SRAM cache memory.

    select byte by byte sel

else
    hit = false
    data = M[Tag | index | byte]
    cache Data(index) = M[Address]
    also Tag(index) = Tag new

Example what is the cache organization for 1K byte direct mapped, blk = 32 byte, Find Tag, index of loc 14020 hex

---

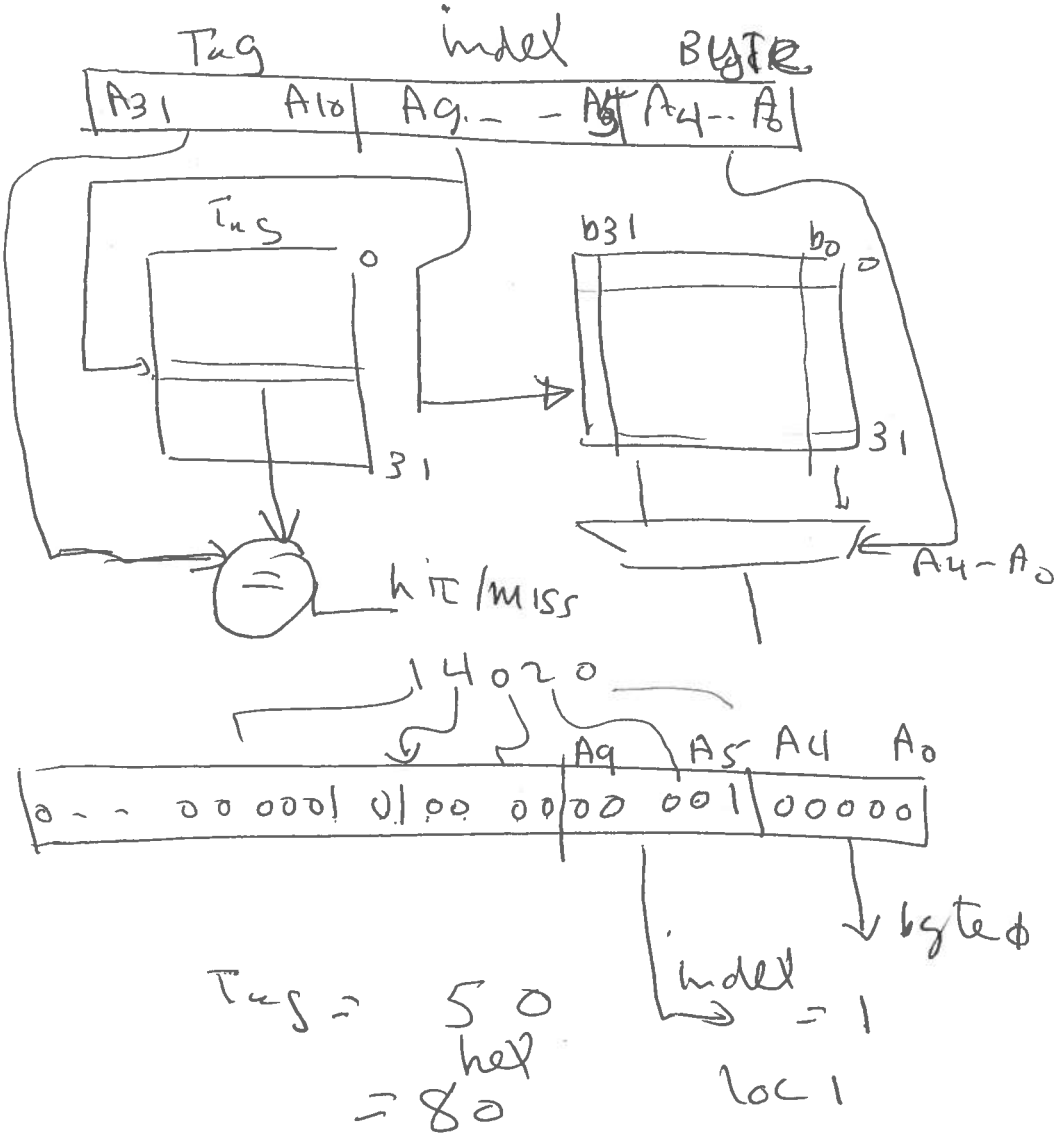
# of blocks = $\frac{\text{size}}{\text{block size}} = \frac{1024}{32} = 32$ blocks

# of Address bits for byte sel = $\log_2 32 = 5$

# of Address bits for index = $\log_2 32 = 5$

# of Address bits for Tag = 32 − 5 − 5 = 22 bits

Assuming a 32 bit CPU Address



Cache Performance

Average memory Access Time
= Hit Time + Miss rate × Miss penalty

Miss rate = (1 − hit rate)

HIT Time = Time to access cache + Time To Find HIT/MISS

Miss Penalty = time to access Main Memory + Time To Transfer block To Cache

Time of Transfer = $\frac{\text{Block size}}{\text{bus size}} \times \frac{1}{\text{Bus Speed}}$

Example: Find Average access Time for a cache system if hit rate = 90%, cache is 10 ns, processor is 200 MHz, DRAM access Time = 100 ns, cache block size = 8 words, BUS width = 1 word, speed = 100 MHz.

Average Time = Cache hit Time
+ Miss rate * (DRAM access + transfer Time)
= 10 + 0.10 * (100 + 10 * 8) = 10 + 18 = 28 ns
≈ about 5 cpu cycles

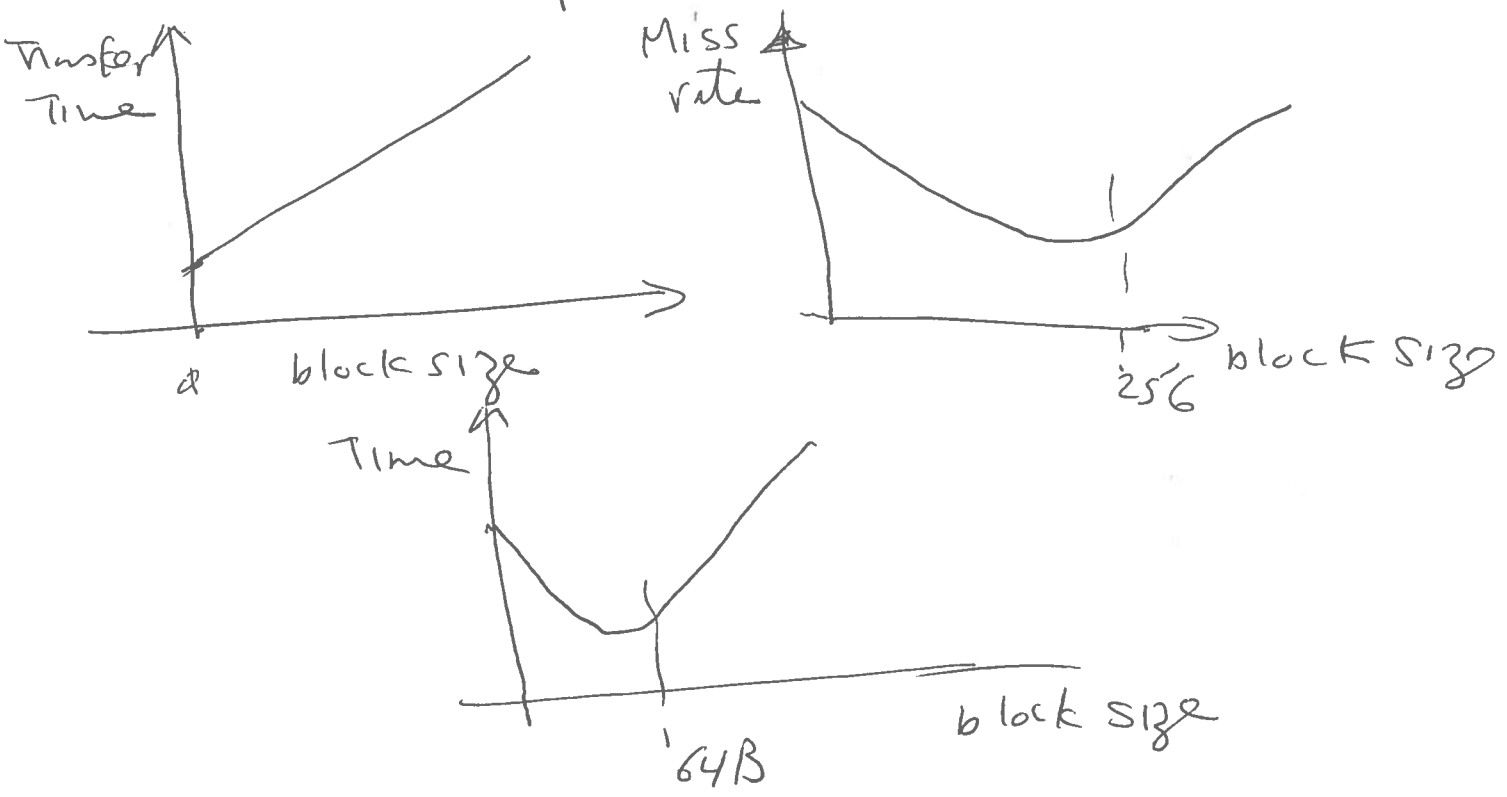
If instructions, Data have different Miss rates

Time = % Instruction (hit Time + Inst. Miss * Memory penalty for Inst))
+ % data * (data hit Time + Data Miss * Memory penalty)

→ Memory penalty For Inst might be diff than for Data (if I cache, D cache have different Block size)

Block Size Trade-off

- Increasing block size improves hit rate because of spatial locality (to a degree)
- Increasing block size increases Transfer Time
- There is a point of optimum performance



Two Way Associative Cache

- Motivations
- Mapping
- organization
- Advantages and disadvantages

Two Way set Associative

MOTIVATIONS:- high Conflict Misses on direct mapped cache because there is only one location for each address, if taken it has to miss

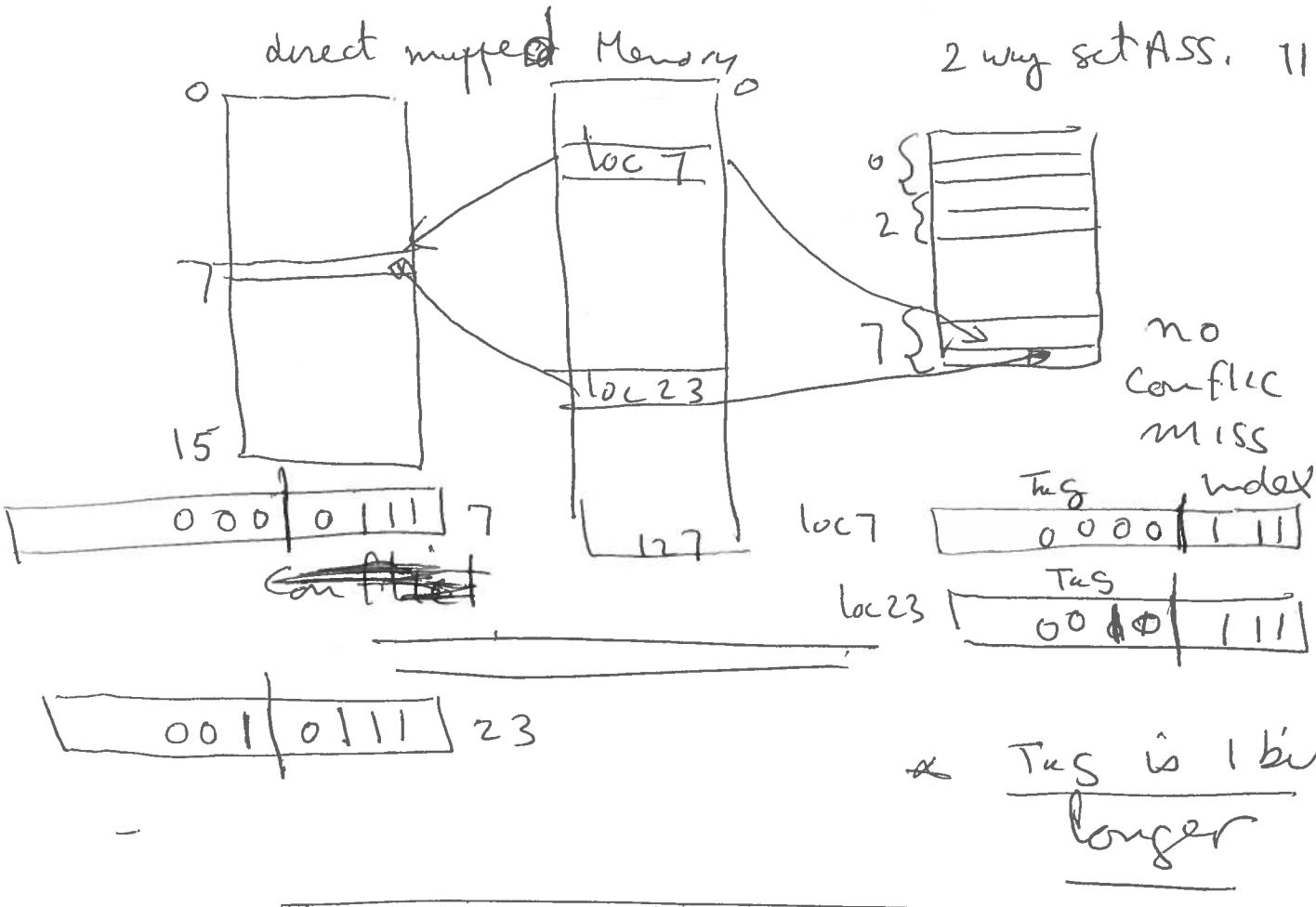
SOLUTION: Use Two way set associative cache. Each address has Two locations to map to.

In a Loop, if 2 instructions map to same cache location in direct mapped cache, with Two way set associative, the 2 instructions will map to diff cache blocks.

Example Assume Memory size = 128
Cache size = 16
Compare direct mapped to 2 way set Associative for locating Loc 7, Loc 23

1- direct mapped LOC 7 maps to 7 modulo 16 = 7
LOC 23 maps to 23 modulo 16 = 7

2- 2 way set Ass. 7 modulo 8 = 7 1st
23 modulo 8 = 7 2nd

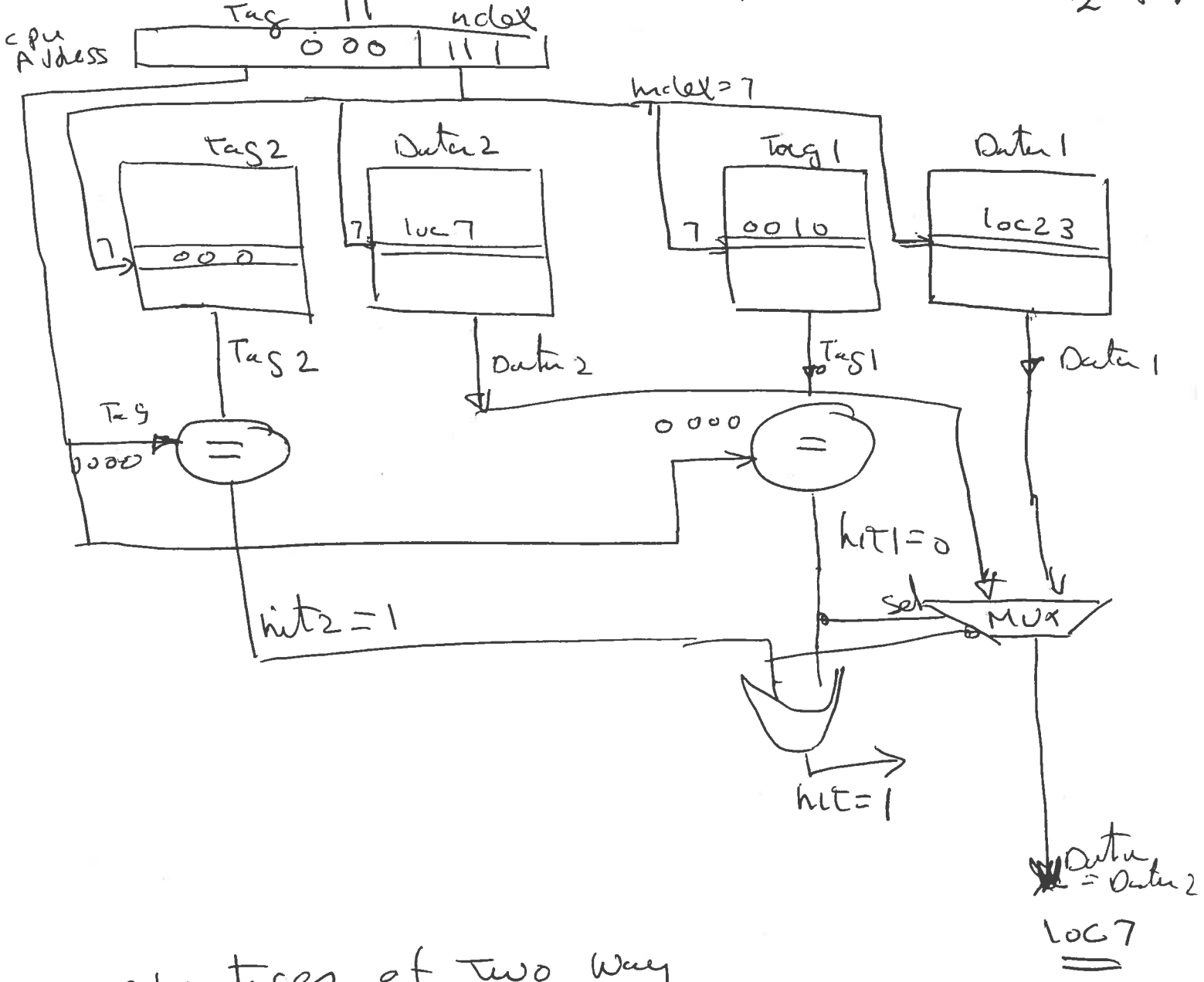


Two way set Associative
cache organization

2 direct mapped cache in
parallel

Two Way Set Associative organization

2 direct mapped cache in parallel (each ½ size)



Advantages of Two Way Set Associative

- Low miss rate (Conflict miss)

Disadvantages of Two way Set Associative: ✓3

1- Larger Tag size (1 bit X # of loc)

2- Use of 2 Comparators (one per each tag)

3- Search both Tags (slow)

4- Using a MUX in the critical Time path (slower)

5- Must wait for hit/miss before can output Data from MUX (compared to direct mapped that can output Data while finding hit/miss)

Example: Find miss rate of the following sequence: 0, 8, 0, 6, 8 for 2 cache size = 4 Loc., if it is a direct mapped and if it is a two way set Associative.

1- Direct mapped

A- Access 0 maps at 0 modulo 4 = 0
- First Time ⟶ miss
Cache has Loc 0 at 0

B - access 8 mapps to $\frac{8}{4} = 2, \phi \longrightarrow \phi$
(miss) cache has loc 8 at ϕ

C - ϕ access mapps to loc ϕ
miss $\longrightarrow$ cache will have loc ϕ

D - 6 mapps to $\frac{6}{4} = 1, 2 \longrightarrow$ loc 2
First time = miss (loc2 has 6)

E - 8 mapps to ϕ ; but ϕ has loc 8
miss

100 % miss rate

Two Way set ASSOCIATIVE

A - 0 $\longrightarrow$ Tag 1 loc ϕ Tag 1

000	0

B - 8 miss $\longrightarrow$ loc ϕ of Tag 2

100	0

C - 0 miss $\longrightarrow$ loc ϕ of Tag 1
hit ✓

D - loc 6 $\longrightarrow$ 1

011	0

tag = 3
miss

	loc		Tag 1	Tag 2
A	0	-	000	-
B	0	8		100
C	ϕ	8	000	100
D	0	6	000	011
E	8	6	100	011

QUICT ?? — LRU

E - loc 8 $\longrightarrow \phi$ miss

100	0

Tag 2

QUICT Tag 1

miss rate 80% compared to 100%

Replacement policy

1- No replacement for Direct mapped cache (only one block to be replaced "No choice")

2- Need a replacement policy for Associative Cache, Each memory location have more than one choice (N = Ass).

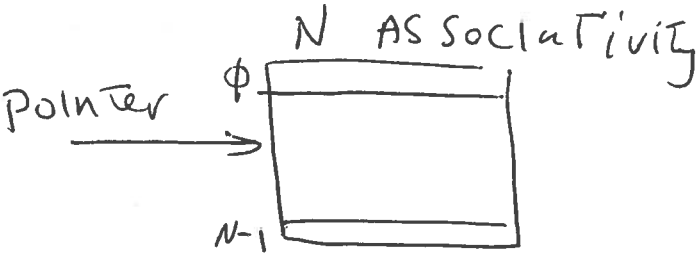
Question? which block to throw out? for the new coming access.

A- Random replacement.
cost less but it is not effective.
The cache block to be replaced is selected in random

B- Least recently used block LRU
throw block that has not been used for longest time (Temporal locality).

Need Hardware to keep track of access History.

Implementation For a LRU



Use a pointer that always point TO LRU

Protocol: Each Time, an access is used, The access moves Pointer away To Next entry.

If access is not used -- do not move Pointer (LRU)

Two way set associative pointer = 1 bi

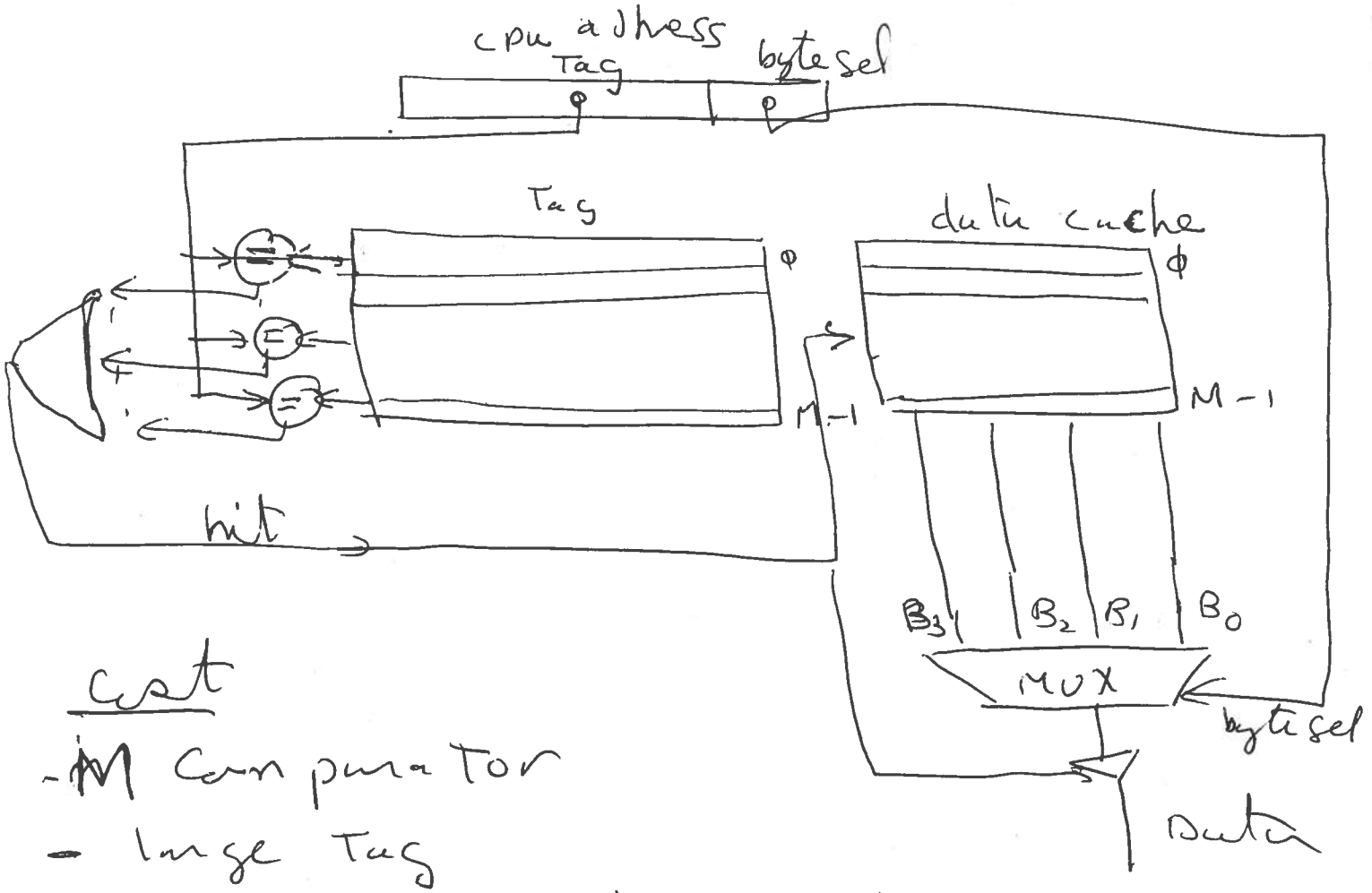
Fully Associative Cache

Concept: any memory location can maps in any cache location (Conflict miss is reduced to zero).

This means that the index field = 0 and Tag = length of Memory Address

---

Fully Associative Cache organization



cost

- M Comparator
- large Tag
- parallel search of all Comparators (slow)

Example Design Fully Associative cache = 1KB if Block size = 32 byte.

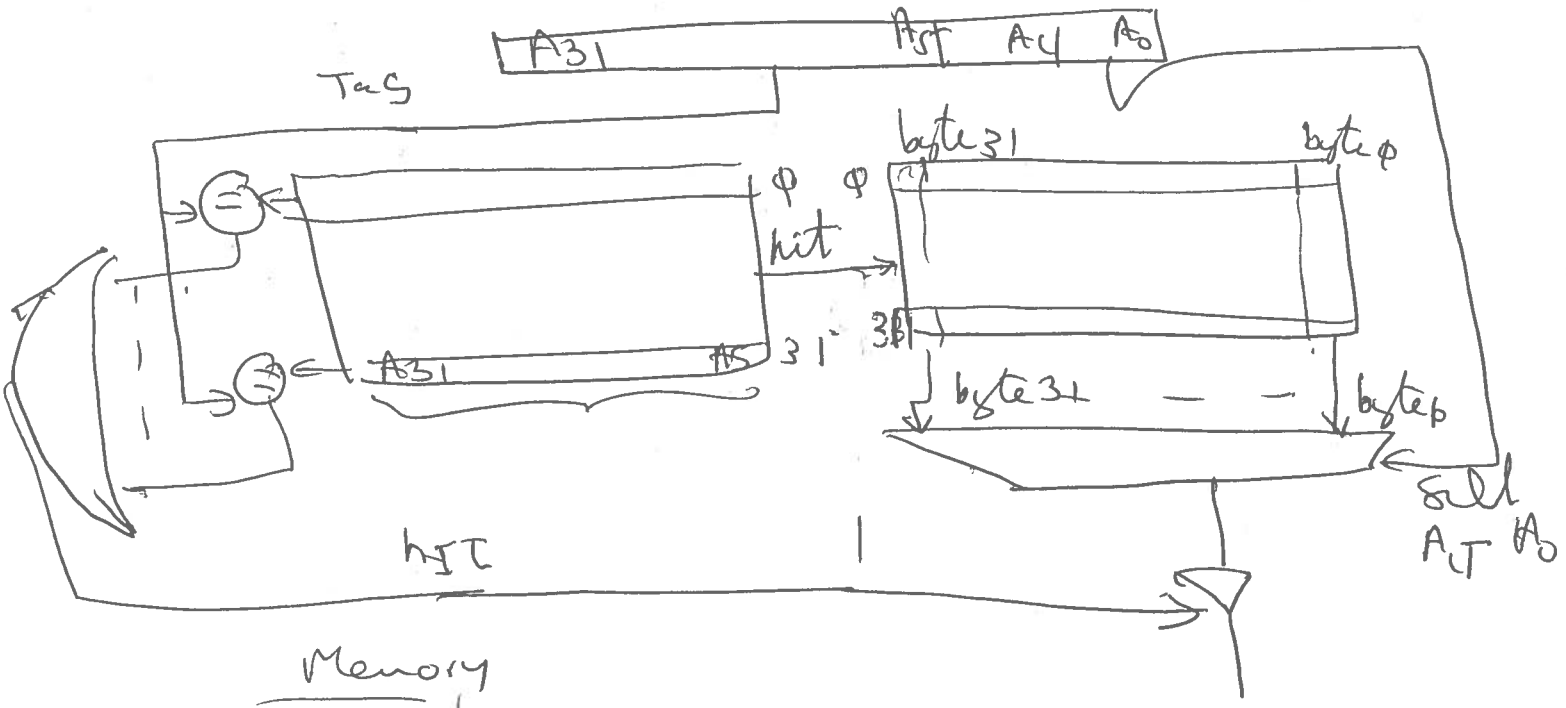
# of byte sel bits = $\log_2 32 = 5$ bits

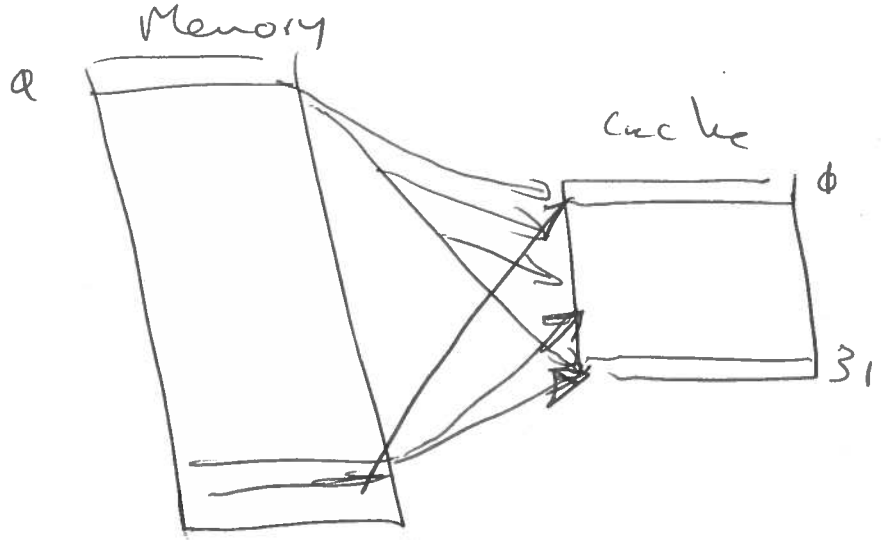
Tag = 32 − 5 = 27 bits

# of blocks in Cache = $\frac{1024}{32} = 32$ blocks

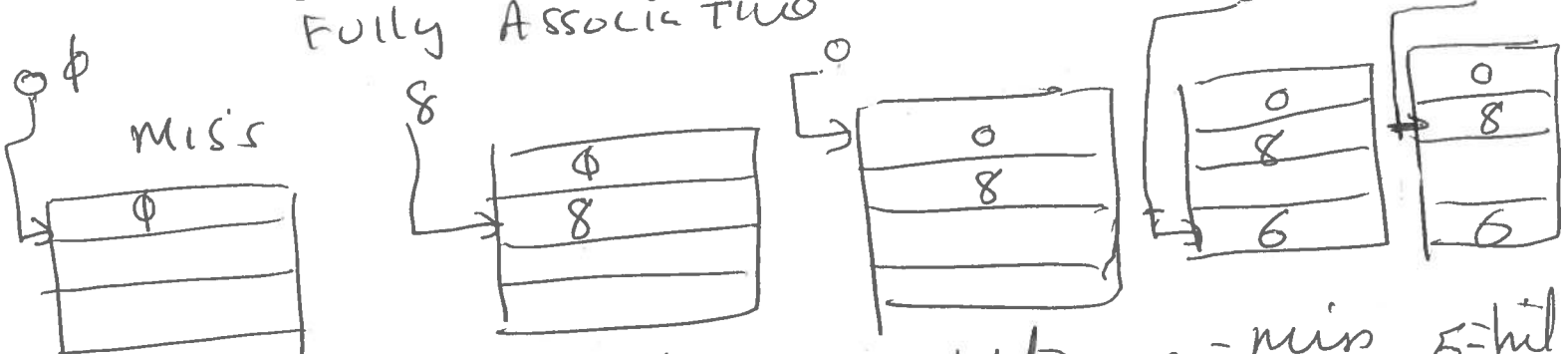
Cache Comparators = 32

Tag = 32 X 27, Data = 32 X 32 byte





Example program uses addresses 0, 8, 0, 6, 8
cache size = 4 locations, use Fully Associative



miss = 60% only

Cache write policy

Read: if hit ⟶ read from cache

if miss ⟶ read from memory
replace (LRU block) by
data from memory
if replaced block is modified (dirty) need
to write it back to memory.

write: if hit — two methods

write through: write to cache and memory

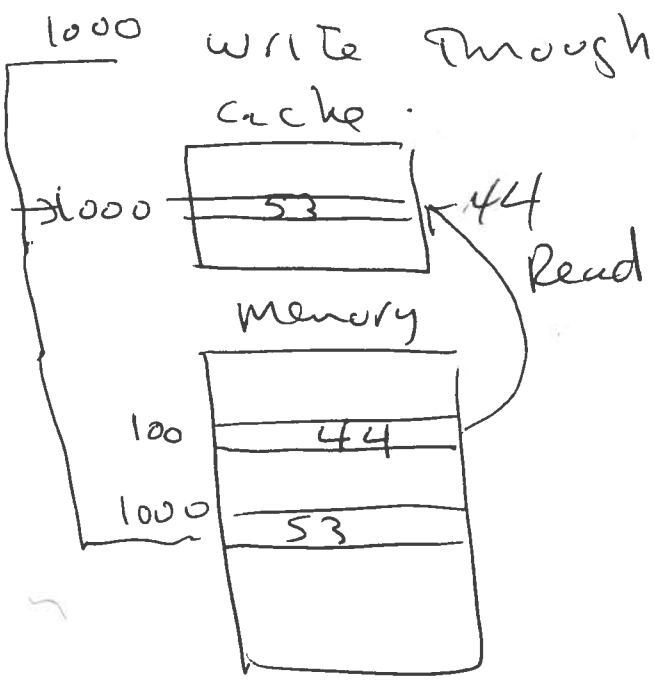
Advantages: cache content is the same
(Coherent with memory) in memory.

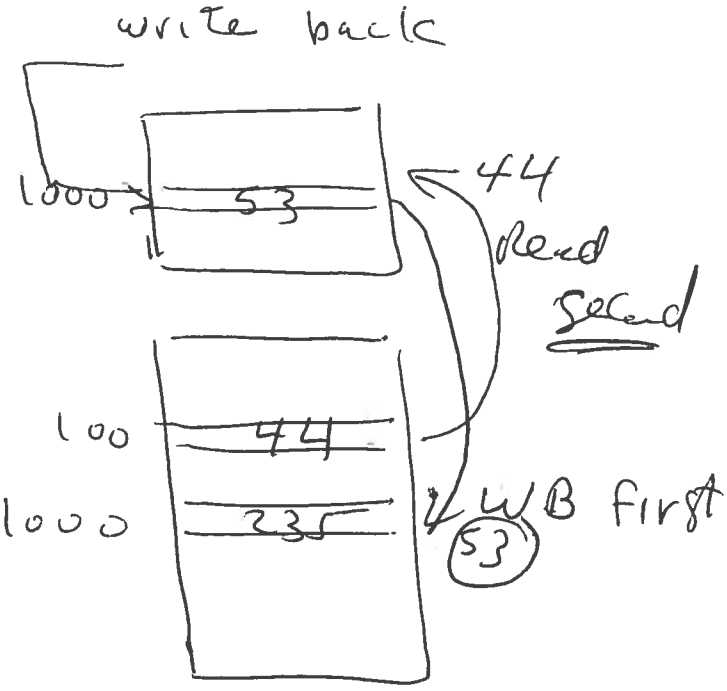
disadvantage: write to memory is slow

write back: only write to cache, mark
it dirty, so when it is
replaced (in read-miss), it must
be written back to memory.

cost: 1 dirty bit per block

Example:
write 53 at 1000 ; read 100





Write Miss

NWA
No write allocate

write only to memory
(block is NOT in cache)
Miss rate is worse

WA
Write allocate

write to memory;
read miss to
move block to
cache
improve miss
rate

Improving Cache Performance

Average memory access Time
= Hit time + Miss rate x Miss Penalty

Improving cache performance with Improving:
1- Hit Time
2- Miss rate
3- Miss Penalty

- Split Cache (I , D) versus unified cache Page 384

Example Compare miss rate of split cache (16KB I + 16KB D) To 32KB unified?
If 16KB I cache miss rate = .64%
16KB D cache miss rate = 6.47 %
32KB Unified miss rate = 1.99%
Assume % Instructions = 75%
Data = 25 % of Progra

Average miss rate for split cache
= .75 x .64 + .25 x 6.47 = .48 + 1.62
= 2.09

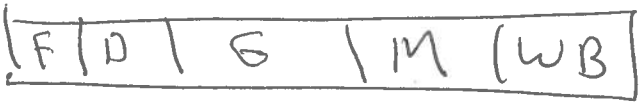
split cache gives worse miss rate than unified.

But performance of unified cache might be worse!
Assume that unified cache hit Takes an extra clock cycle, using pipelining

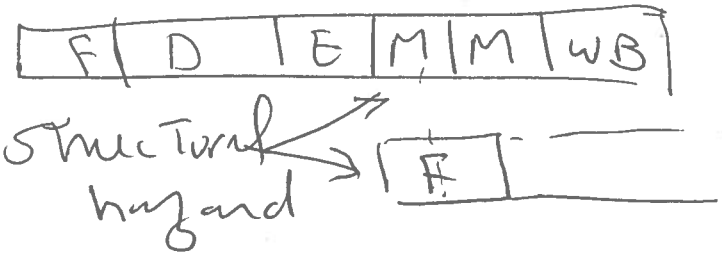
The unified cache leads to a structural hazard (one port). Find Average memory Access Time for each. Assume miss penalty = 50 clock cycle, write is using a buffer with no stalls in writes.

Pipelining with Read/Write operation

1 - split



2- unified (Read/Write) only



Performance of split cache

= % Instruction (hit time + miss rate × miss penalty)
+ % Data (hit time + miss rate × miss penalty)

$= .75(1 + \frac{1.61}{100} \times 50) + .25(1 + \frac{6.47}{100} \times 50)$

$= 2.05$ clock

Unified Cache ? ⟵ not the same ⟶ ?

$= .75(1 + \underline{1.99} \times 50) + .25(1 + 1 + \underline{1.99} \times 50)$

$= 2.24$

slower but has better hit rate

Method 1 Reducing Cache Misses using Larger Block size

types of misses?

1 - Compulsory - Cold start misses for first reference misses (Block)

2 - Capacity - Cache cannot contain all blocks (size)

3 - Conflict - blocks map to same location in cache (Ass)

Larger Block size reduce Complusory misses because of spacial locality.

Problems increase Transfer Time and may increase Conflict misses (few blocks in cache)

Example Assume a 1KB cache with the following miss rate:
Block size 16B = 15.05%, Block 32B = 13.34%
Block 64B = 13.76%, Block 128B = 16.64
if memory Access Time = 40 cycles,
Transfer Time = 16B for 2 clocks.
Find The Best Block size for memory performance.

Average memory access Time =
HIT Time + miss rate * miss penalty
miss penalty = memory Access Time + Transfer Time

TM for 16 B = 1 + .1505 * (40+2) = 7.32 cycles 241

✓✓ 32B = 1 + .1334 * (40+4) = 6.87 cycles

64B = 1 + .1376 * (40+8) = 7.605 cycles

128B = 1 + .1664 * (40+16) = 10.318 cycles

Method 2: Higher Associativity

Improve conflict miss. For size N direct mapped cache, 2-way set associative of size $\frac{N}{2}$ has about same miss rate

Example assume a 32 KB cache, with the following miss rates

1-way = .020, 2-way = .014, 4-way = .013

8-way = .013

miss penalty = 50 cycles, hit time for

1-way = 1 cycle, 2-way = 1.10, 4-way = 1.12

and 8-way = 1.14

Find best cache design.

Average memory Access Time = Hit Time + Miss rate * miss penalty

1-way = 1.0 + .02 X 50 = 2.0

2-way = 1.10 + .014 X 50 = 1.8

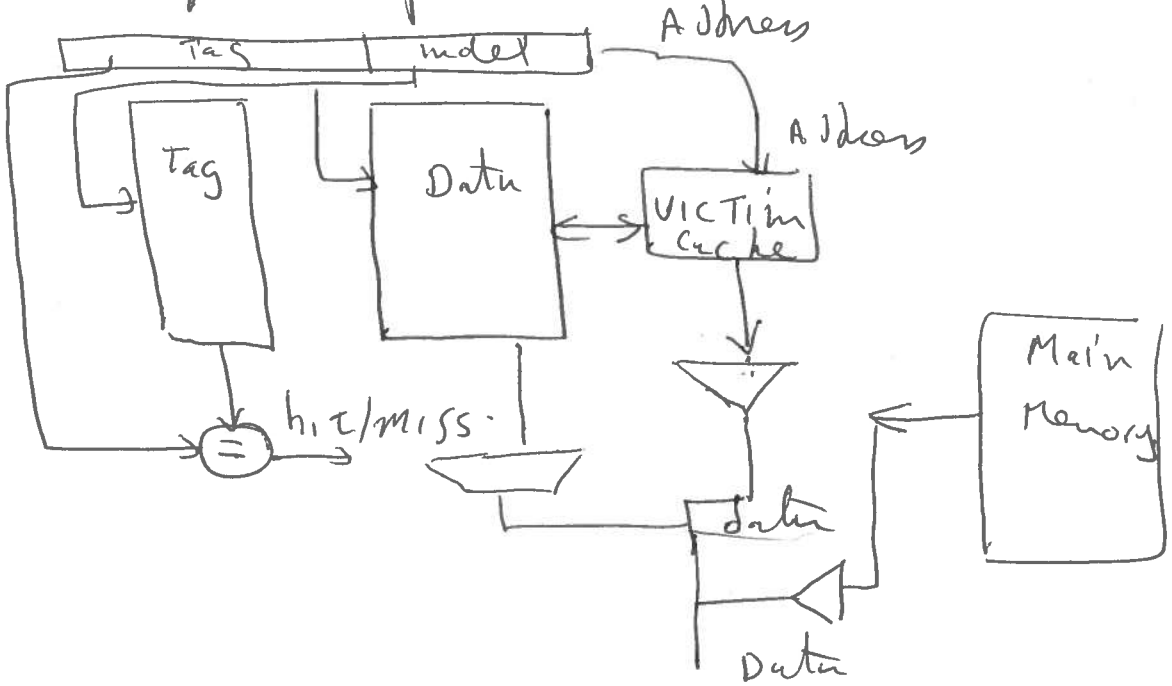
4-way = 1.12 + .013 X 50 = (1.77) ✓✓

8-way = 1.14 + .013 X 50 = 1.79

Method 3 Using Victim Cache

- Reduce miss rate without affecting clock cycle Time (Assoc), or miss penalty (Block).

Add a small fully Associative victim cache for replaced blocks.



ON a cache miss, check victim cache rather than going to lower level (main memory). If victim cache = hit

Swap block with one in cache

It will be a cache hit, replaced block on miss is transfered to victim cache.

It improves Conflict miss by 20 — 95 %.

Method 5: Hardware Prefetching of Instructions and Data

Instruction Prefetch: on a miss fetch 2 blocks, place one in cache and other in stream buffer. Instruction stream buffer improves hit rate by 50%.

Data prefetch: use stream buffer and prefetch data at different addresses

- Need accurate prediction for preFetch.
- Possible Precomputation.
  while waiting for PreFetch, processor can proceed and cache can continue supplying data (Nonblocking cache).

Method 6: Using Compiler

- PreFetch by inserting instructions to request data before they are needed.

has some overhead, of the inserted instructions must not exceed the benefits of prefetch.

Example: Assume 8KB direct mapped with BLK size=16B. Using write back with WA.
assume a[3][100], b[101][3] each element =8B find miss rate, prefetch instructions to improve miss rate for

Reducing Cache Miss Penalty

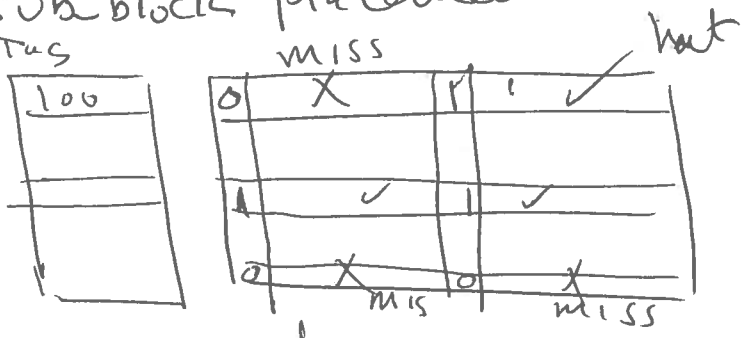
important as cost of miss penalty increases due to speed gap between CPU/DRAM

Method 1: Giving priority to Read Miss over writes

write through
using write buffer, and allow read miss to continue even write buffer is not empty as long as there is no conflicts (Read same location).

write back on read miss with dirty block to be written back to a buffer, then read memory, then write buffer back to memory.

Method 2: sub block placement



on miss transfer only sub-blocks, mark it valid. need a bit for each sub-block

For a hit ⇒ tag match and valid bit = 1

reduce transfer time of large blocks.

Reducing Cache Miss Penalty

Method 3: Early restart and critical word first.

- No Hardware is required
- only one word in a block is required by CPU

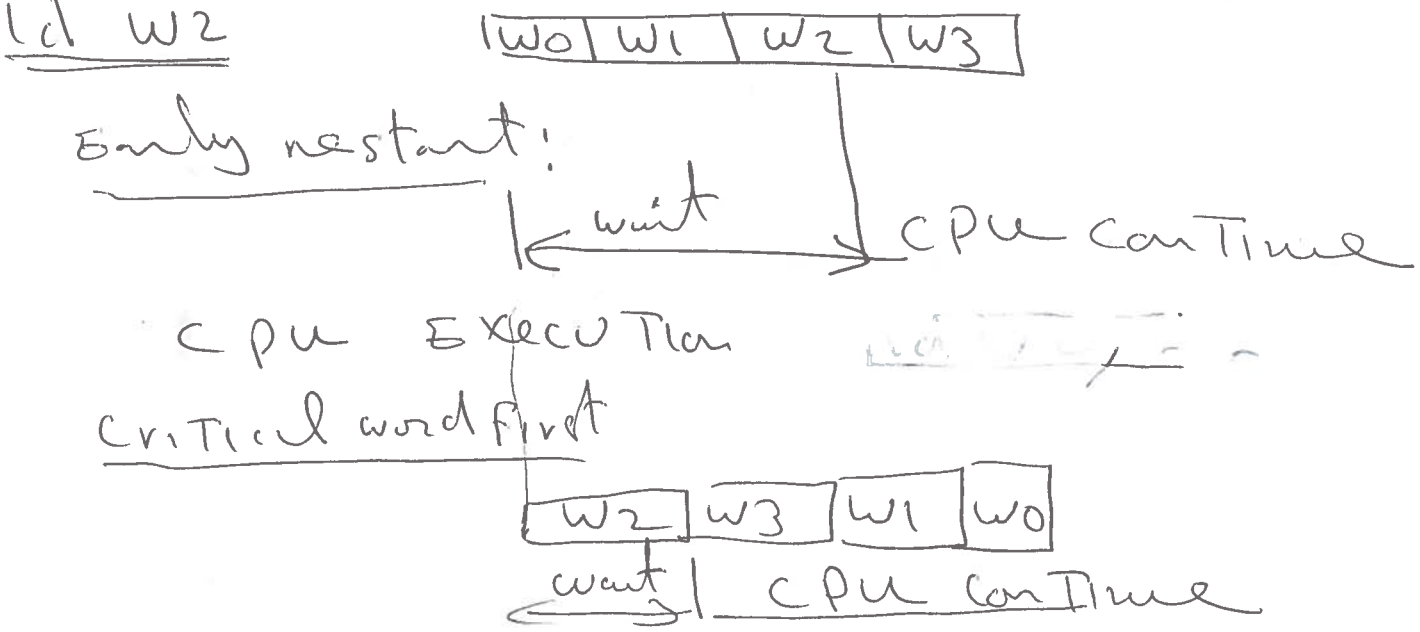
Two methods

1- Early restart: CPU waits for requested word, let CPU continue execution when word arrives

2- Critical word first: send requested word first from memory, let CPU continue execution when word arrives (use wrapped fetch).

Example

ld W2



Method 4: Non Blocking Caches

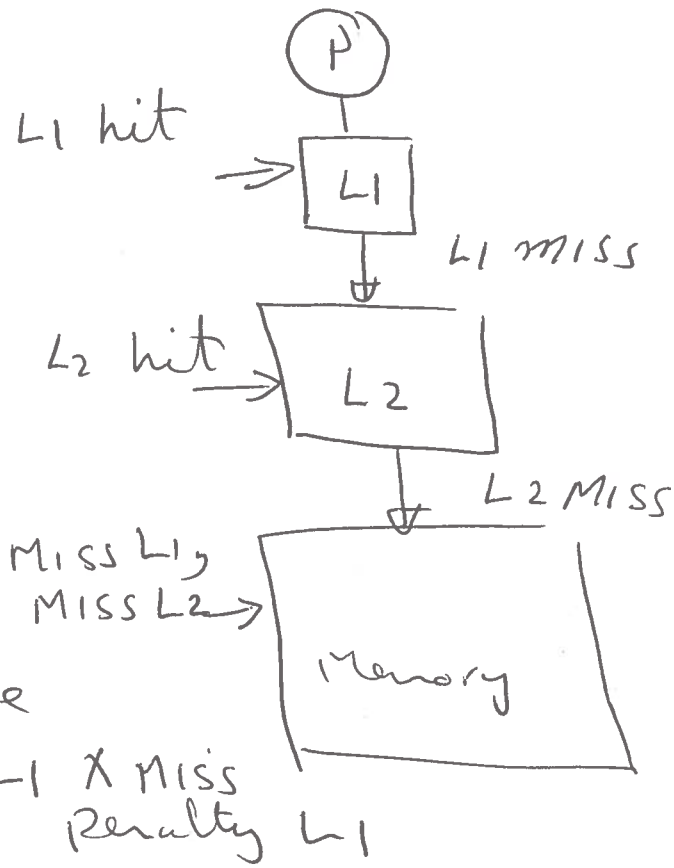
For pipelined machines that allows out of order execution and completion, CPU could continue fetching instructions from instruction cache while waiting for data cache miss. It also allows Data Cache to supply data while waiting for a miss data to arrive from memory.

- Need Scoreboard or Tomasulo Control.

Method 5: Second Level Caches

on a first level cache miss, can use a faster level 2 cache and not the memory.

only misses from L1 could be found in L2.



Average memory access time
= Hit time L1 + Miss L1 x Miss Penalty L1

Miss Penalty L1 = Hit Time L2 + Miss L2 x Miss Penalty L2

Average memory access Time

= Hit Time L1 + Miss L1 * Hit Time L2

+ Miss L1 * Miss L2 * Memory Latency

Local miss rate: Miss L1

Miss L2

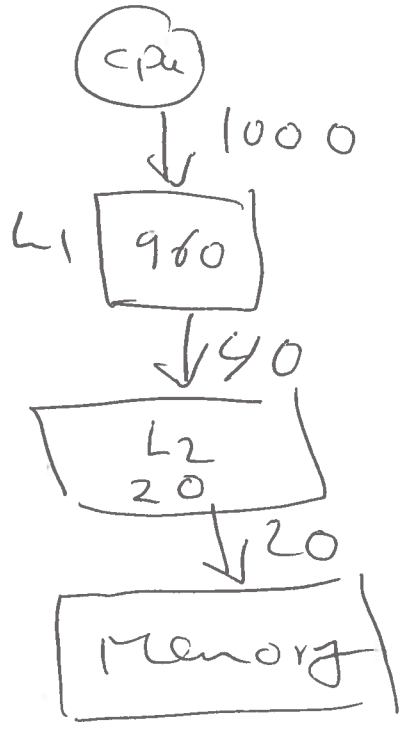
Global miss rate = Miss L1 * Miss L2

Example: Assume 1000 memory references that have 40 misses in L1, 20 misses in L2, find various miss rates

Answer: Miss L1 = $\frac{40}{1000}$ = 4%

Miss L2 = $\frac{20}{40}$ = 50%

Global Miss = $\frac{4}{100} \times \frac{50}{100}$ = 2%



Characteristics of second level cache:

1- Larger Than L1
2- slower than L1.
3- Multi level inclusion property
: data in first level cache
exist in second level.
important in multi processors,
can snoop and check second level
cache
4- might need larger block size than L1
" inclusion can be maintained at a cost "
5- could use write through L1 if L2
is a write back (speed of L2).

Reducing Hit Time

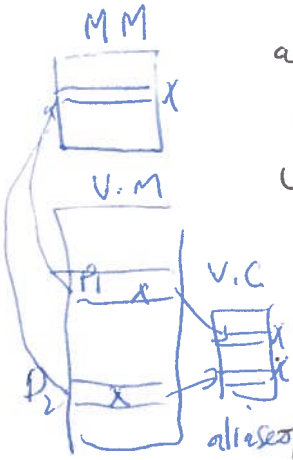
- affects cpu clock rate (very critical)

<u>method 1</u> : small single cache

- on chip cache (smaller is faster)
- direct mapped cache is simpler
can overlap tag check with data
transmission (only one data set)

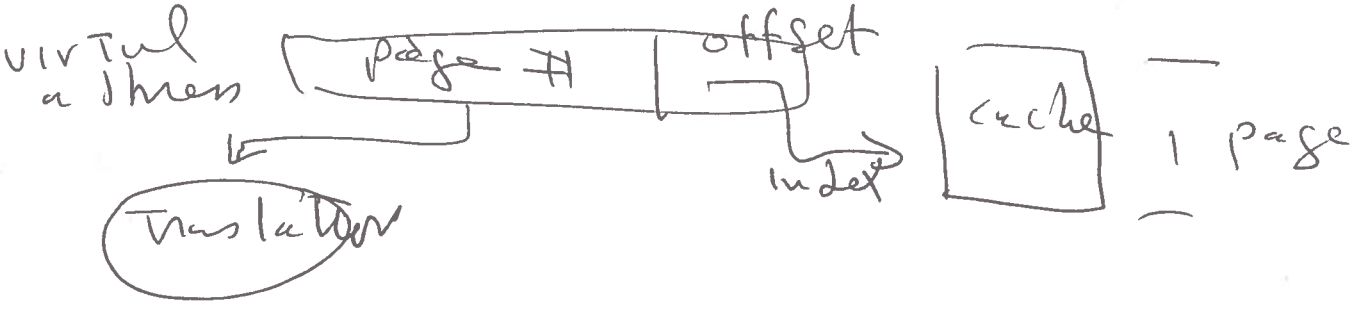
Method 2: Using Virtual addresses to access Cache

- For fast and no translation, use virtual cache
- Problem in process switch, must flush cache (Cache contains different physical addresses)



- virtual addresses of two different addresses might use the same physical address (aliases), this will result in having two copies in virtual cache for same data. CPU then could modify one copy, the other copy will have the wrong value.

Could use both virtual and physical Address by using offset index to index the cache



Limitation: Cache = page size

Access cache while translation of virtual to physical.