Ve370 Review

exception

Exception

32-bit register in MIPS：

- Exception Program Counter (EPC) = address of interrupted instruction + 4
- CAUSE: Service the exception or interrupt according to the causes
- Cause and EPC not necessarily in EX stage.
- After exception is handled: PC= EPC-4

Vectored Interrupts：

- Single Entry Point: MIPS
- Multiple Entry Point: each entry point for one cause

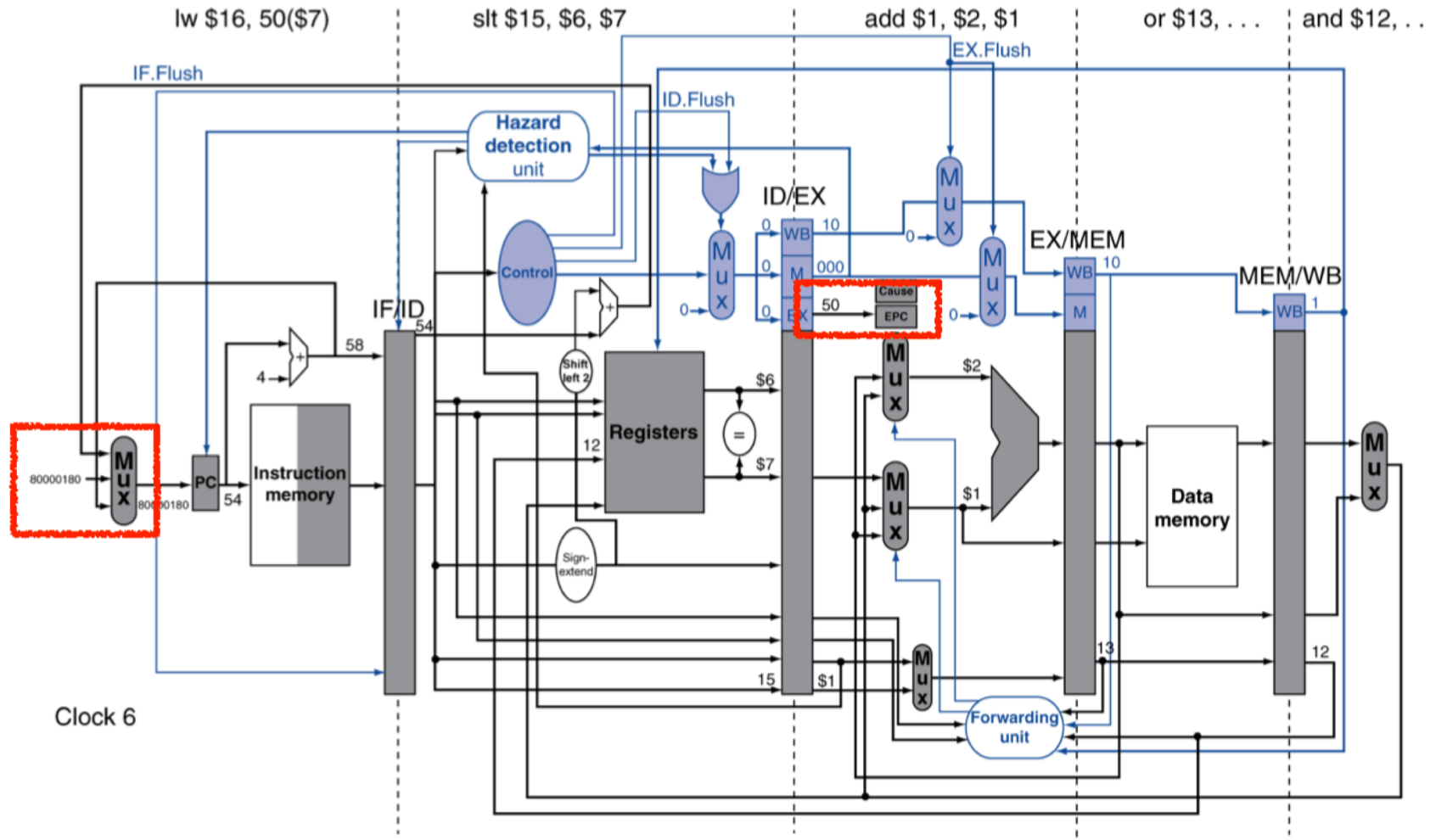
Exception on add in

```
40      sub   $11, $2, $4
44      and   $12, $2, $5
48      or    $13, $2, $6
4C      add   $1,  $2, $1
50      slt   $15, $6, $7
54      lw    $16, 50($7)
…
```

Handler

```
80000180    sw    $25, 1000($0)
80000184    sw    $26, 1004($0)
```

EPC=50
PC=80000180



cache

Locality

Temporal locality

- Items that are accessed recently are likely to be accessed again soon
- e.g., instructions in a loop

Spatial locality

- Items near those that are accessed recently are likely to be accessed soon
- E.g., sequential instruction access, array data

Exercise 5.2

In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row are stored contiguously.

a.

```
for (I=0; I<8; I++)
  for (J=0; J<8000; J++)
    A[I][J]=B[I][0]+A[J][I];
```

b.

```
for (J=0; J<8000; J++)
   for (I=0; I<8; I++)
    A[I][J]=B[I][0]+A[J][I];
```

5.2.2 [5] <5.1> References to which variables exhibit temporal locality?

5.2.3 [5] <5.1> References to which variables exhibit spatial locality?

5.2.2

(a). I, J, B[I][0]

(b). I, J

5.2.3

(a). A[I][J]

(b). A[J][I]

Address

Direct-mapped Cache, 2 words per block, 4 blocks in the cache

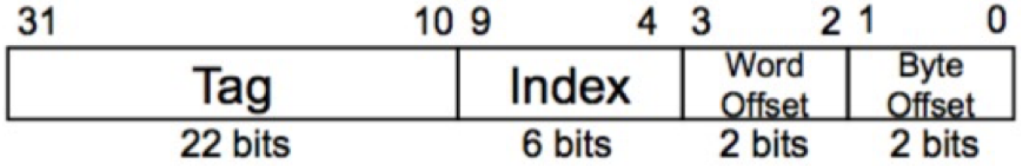
Byte address: 1111_1111_0000_0001

Word address: 1111_1111_0000_00 (2 words/block)

Block address: 1111_1111_0000_0

- Divided by 4 = shift left two bits
- modulo 4 = take the last two bits

- 64 blocks, 4 words/block
  - What cache block number does byte address 1200 map to?
  - Word number = 1200/4 = 300
  - Block (address) number = 300/4 = 75
- Block index in cache = 75 modulo 64 = 11



Cache Design – Direct-mapped Cache

Direct-mapped Cache, 2 words per block, 4 blocks in the cache

Byte address: 1111_1111_0000_0001

Word address: 1111_1111_0000_00 (2 words/block)

Block address: 1111_1111_0000_0

Cache Index: 0_0 (4 block/cache)
Tag: 1111_1111_000
Check Hw7 5.3

Cache index = (Block address in memory) modulo (Number of blocks in cache)

Cache Size in Bits

- Given
  - 32-bit byte address
  - 2^n blocks in cache
  - 2^m words per block, 2^{m+2} bytes
- Size of tag field = 32 – (n + m + 2)
  - n bits to index blocks in cache
  - m bits used to select words in a block
  - 2 bits used to select the 4 bytes in a word
  - Tag field decreases when n and m increase
- Cache size = 2^n × (block size + tag size + valid field size)

Cache Design – Set Associative Cache

Two-way set associative Cache, 2 words per block, a total size of 8 words

Byte address: 1111_1111_0000_0001

Word address: 1111_1111_0000_00 (2 word/block)

Block address: 1111_1111_0000_0

Cache Index: 0 (2 sets/cache)
Tag: 1111_1111_0000
Block Offset: 0

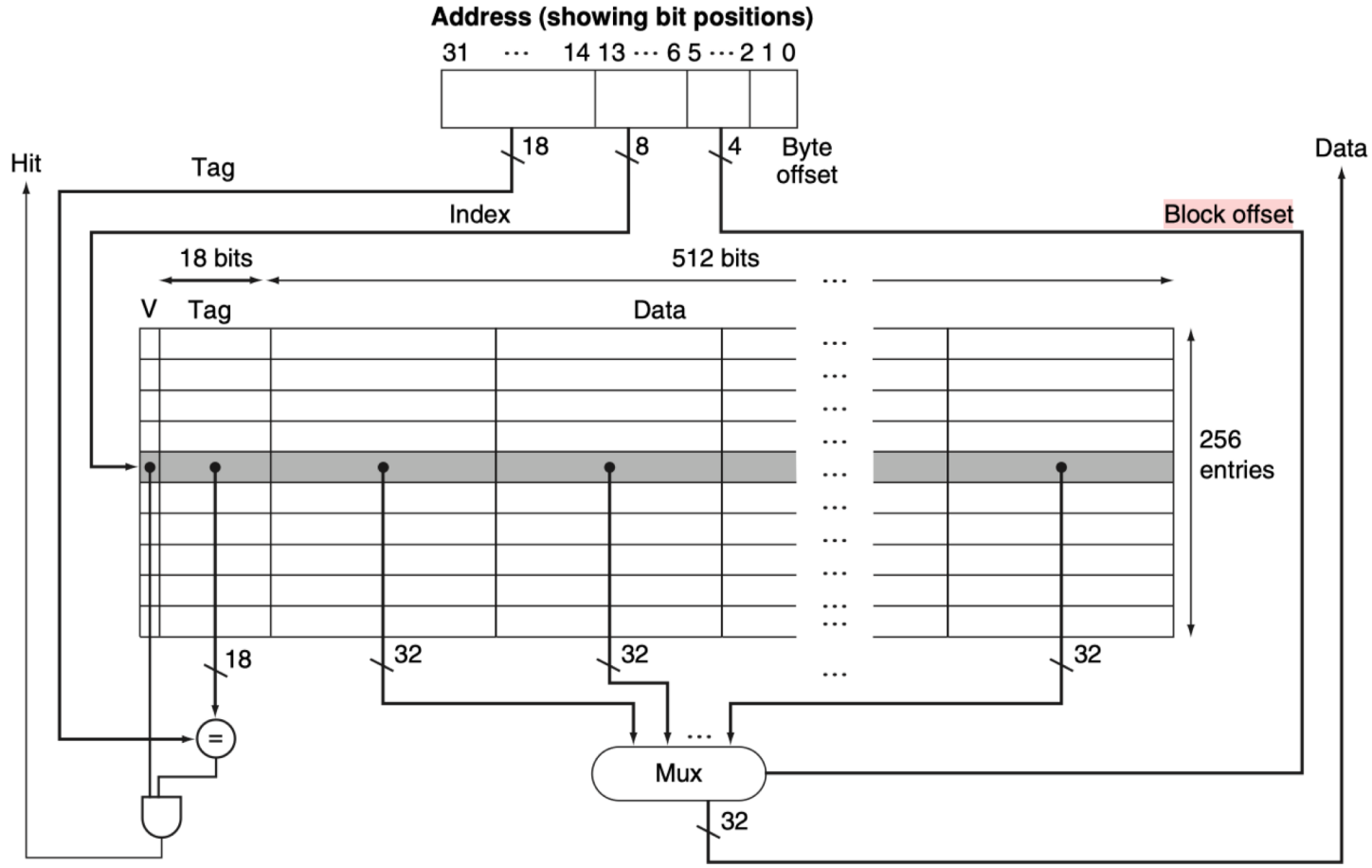
Set address = (Block address) modulo (number of sets in cache)
To locate a block in a set, need to compare n times

The block offset is the address of the desired data within the block.
HW 8 5.8

Replacement: LRU and MRU

Cache Design – Set Associative Cache

The 16 KB cache contains 256 blocks with 16 words per block.



Tag: 18 bits
Cache Index: 8bits
Block Offset: 4 bits
Byte Offset: 2 bits

Cache Design – Block Size

Block Size 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 miss penalty
  - Primarily result of longer time to fetch block
    - Latency to first word
    - Transfer time for the rest of the block
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

A larger cache has a longer access time,
Leading to lower performance
HW7 5.3.4

Cache Design – Keeping Track of Modified Data

Write Through

1. Naive version:
   - Cannot tolerate any inconsistency between cache and lower memory
   - Pay access time to cache and lower memory
   - A write hit actually acts like a miss
2. Write buffer version:
   - The write buffer's job is to keep track of all the pending updates to L2, so that L1 can move on.
   - The write buffer is finite. CPU stalls if buffer is full

Write Back

- dirty bits, update memory only when to be replaced
- A hit required two steps:
  1. check match
  2. write data
- A miss to a dirty block: two accesses to L2. Double miss penalty.
  1. One to let it know about the modified data in the dirty block
  2. Another to fetch the actual missed data.

Cache Design – Write Misses

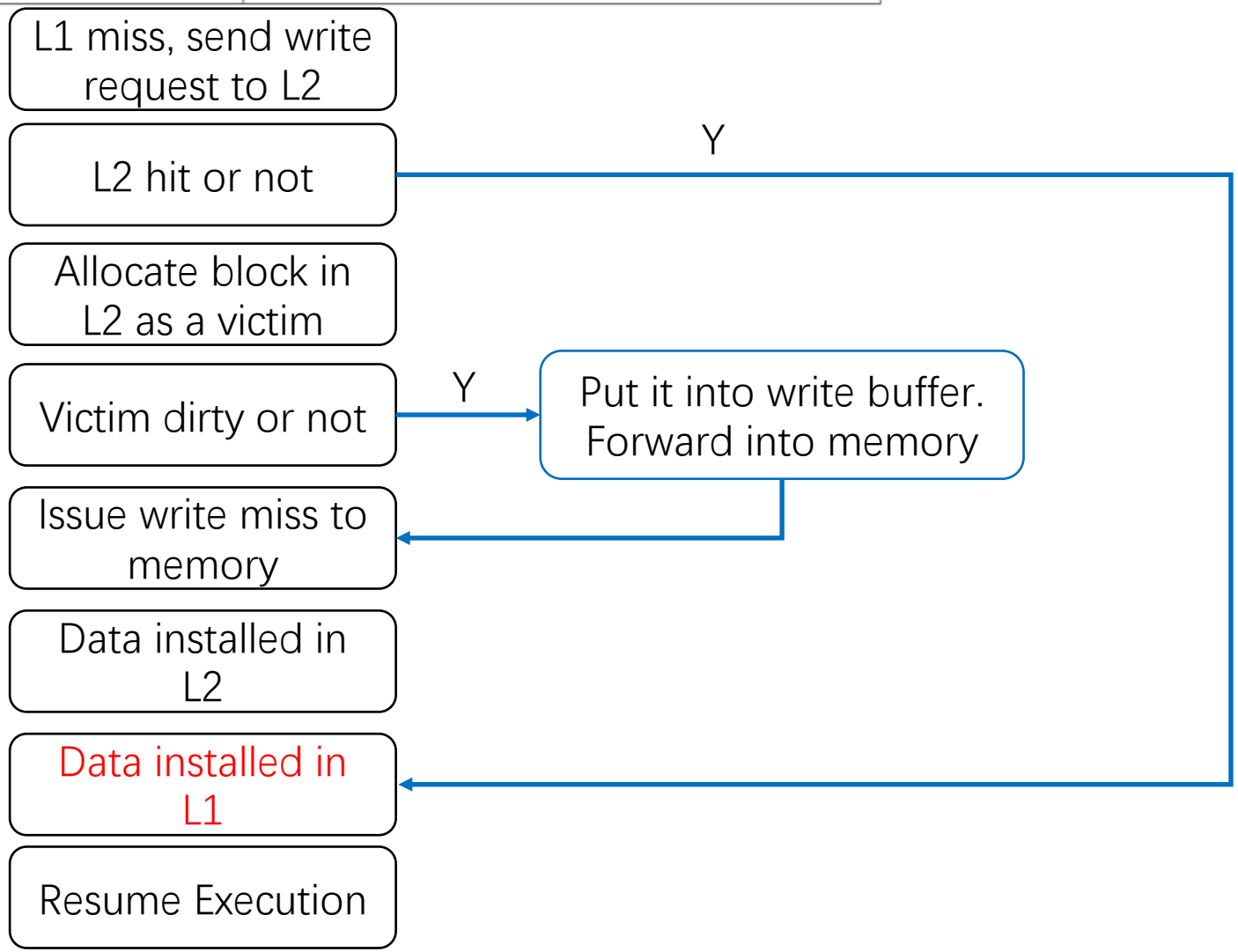
Write Allocation
- Make room for the new data on a write miss.

No Write Allocation
- Write directly to memory

Generally, write-allocate makes more sense for write-back caches and no-write-allocate makes more sense for write-through caches, but the other combinations are possible too.
HW8 5.5.2

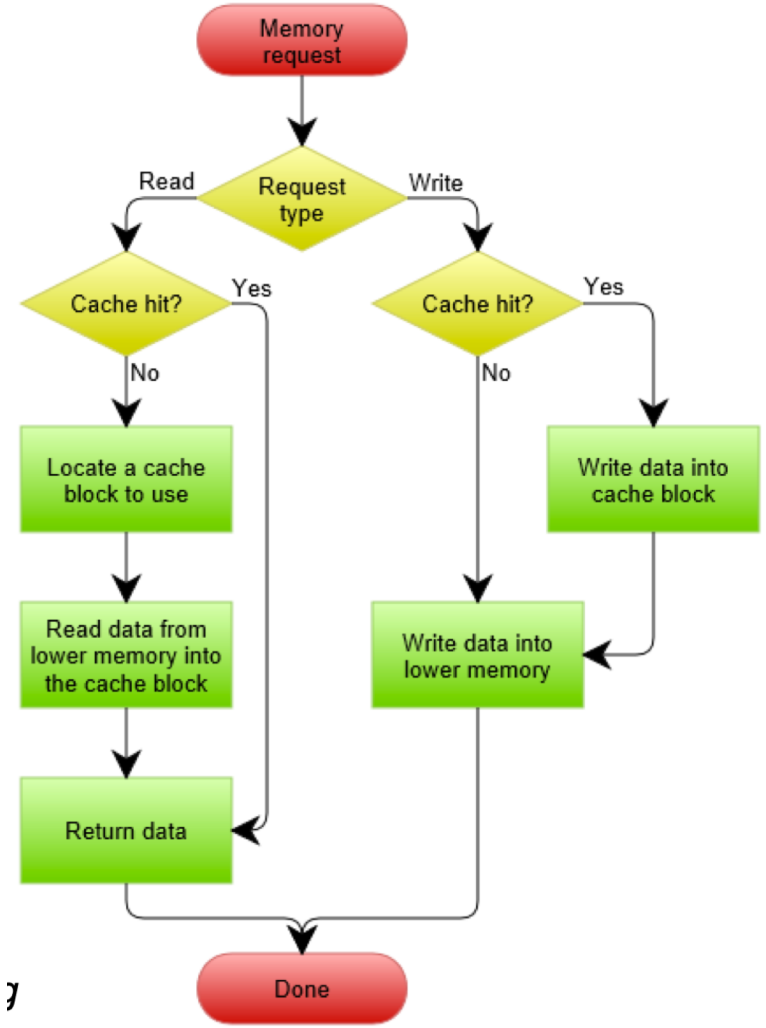
Cache Design – Design Combination

b.	Write through, write allocate	Write back, write allocate

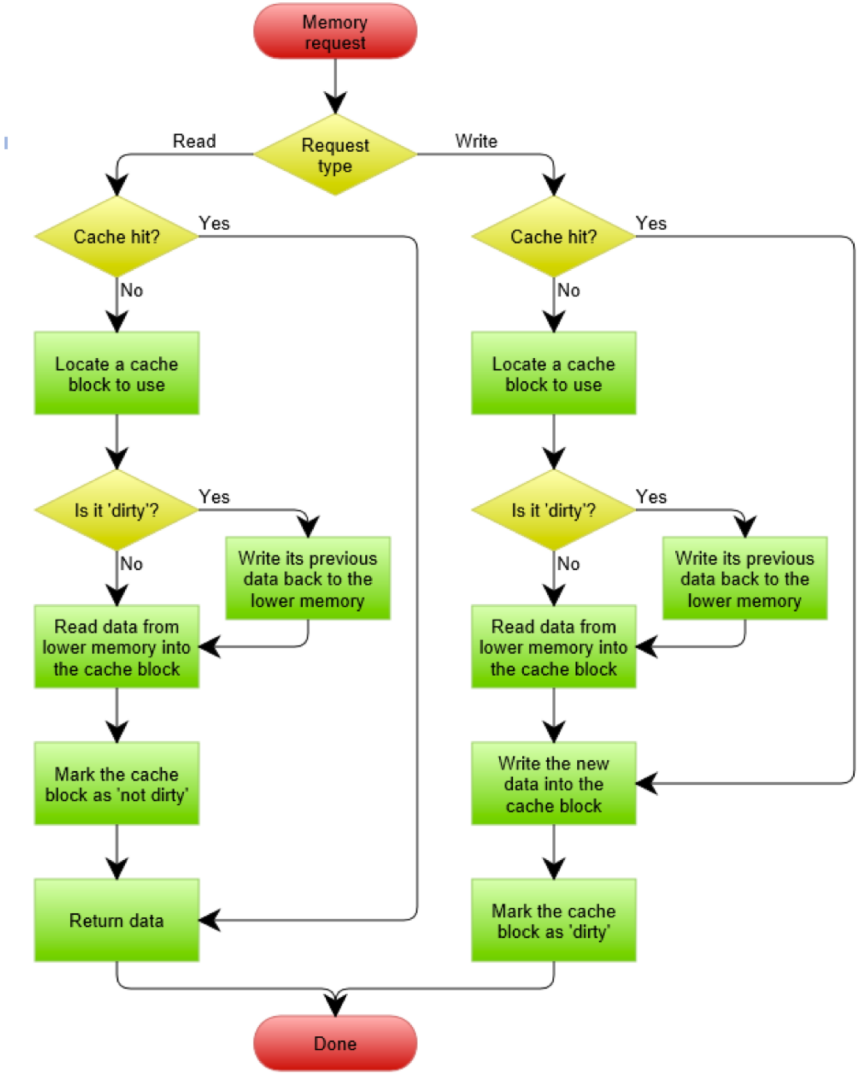


Cache Design – Design Combination

Write through with no write allocation



Write back with write allocation



Cache Design – Miss Penalty

Total CPI = base CPI + Miss (stall) cycles per instruction

Average memory access time (AMAT)
AMAT = Hit time + Miss rate × Miss penalty
HW 8 5.5

Multilevel Cache
CPI = base CPI + L1 miss rate * L2 hit rate * L1 miss penalty (cycles per instruction) + L1 miss rate * L2 miss rate * (L1 miss penalty + L2 miss penalty + main memory hit time)(cycles per instruction)
HW8 5.7, 5.8.4

1. Total CPI = base CPI + memory miss cycles × 1st level cache miss rate
2. Total CPI = base CPI + memory miss cycles × global miss rate w/2nd level direct-mapped cache + 2nd level direct-mapped speed × 1st level cache miss rate
3. Total CPI = base CPI + memory miss cycles × global miss rate w/2nd level 8-way set assoc cache + 2nd level 8-way set assoc speed × 1st level cache miss rate