JOINT INSTITUTE
交大密西根学院

ADAC LAB

ECE3700J RC
Cache 1

Presenter: Ruan Renjian 阮仁剑

SJTU

Locality

Temporal locality: Items that are accessed recently are likely to be accessed again soon

Example: constant, variable, and counter in LOOP

```
// Compute sum of an int array
int  a[N] = {2, 5, 3, 7, ...};
int sum=0;

for(i=0; i<N; i++)
      sum = sum + a[i];
```

Spatial locality: Items **near those that** are accessed recently are like to be accessed soon

Example: array, memory, stack

1. (10 points) The following code is written in C, where elements within the same row are stored contiguously. Assume each word is a 32-bit integer.

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

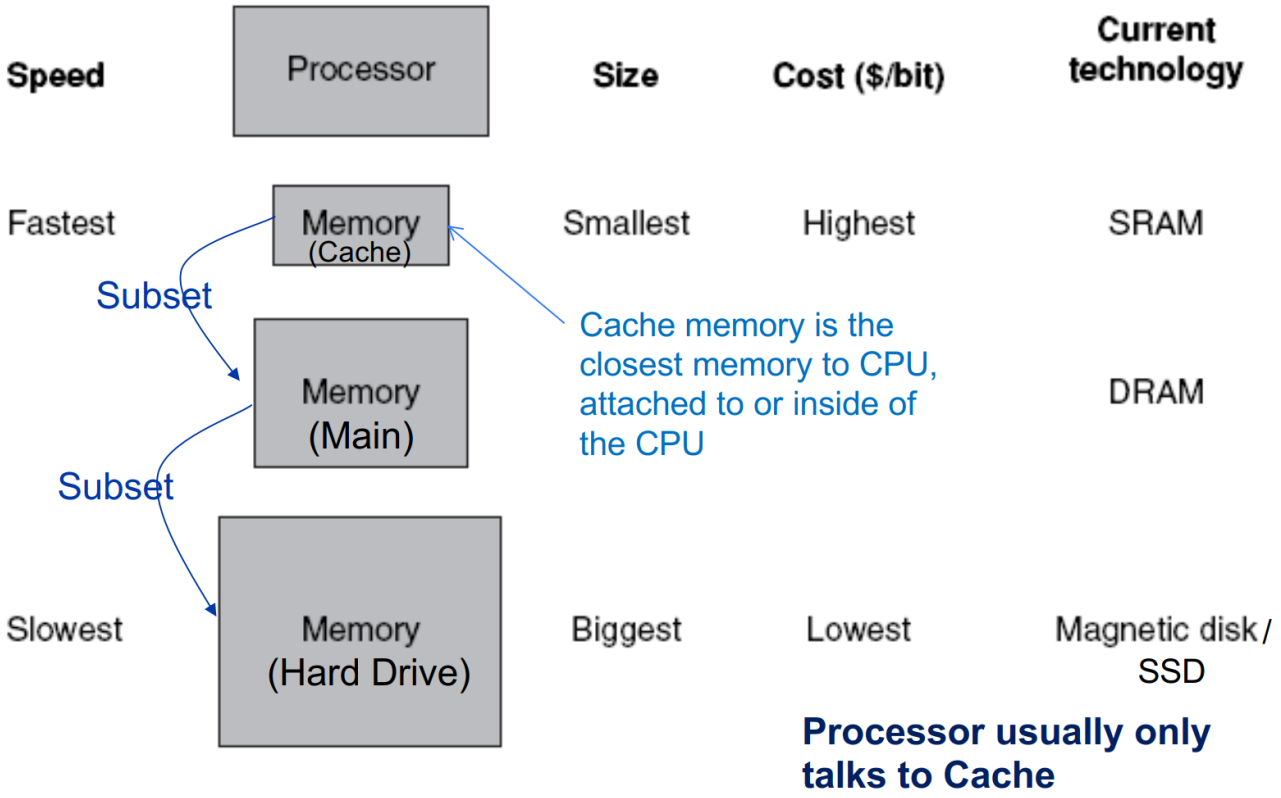
(1) Which variable references exhibit temporal locality? (5 points)

(2) Which variable references exhibit spatial locality? (5 points)

Temporal: I, J, B[I][0]
Spatial: A[I][J]

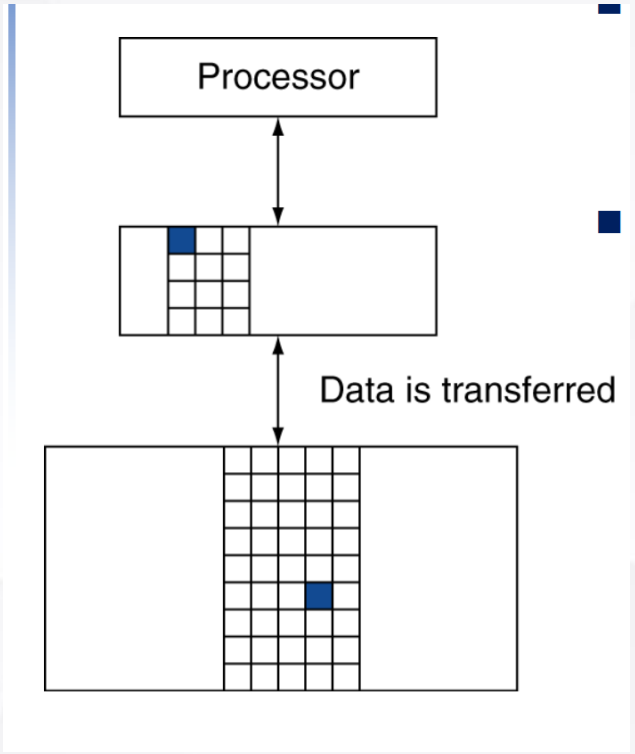
Memory Hierarchy

Memory Hierarchy



Unit of Data transfer: **block**

- May be one or multiple words
- Also has an address
- Take advantage of temporal and spatial locality



Address

Suppose a 10-bit address and each block has 4 words.

1 word = 4 bytes

Byte Address (Word Number[9:2]+Byte Offset[1:0])	Contents
11111100 00	Byte1
11111100 01	Byte2
11111100 10	Byte3
11111100 11	Byte4

1 block = 2^[Word Offset] words

Word Number (Block Number[7:2]+Word Offset[1:0])	Contents
111111 00	Word1
111111 01	Word2
111111 10	Word3
111111 11	Word4

Hit & Miss

- **Hit**
  - If accessed data is present in upper level, access satisfied by upper level
- Hit rate: hits/accesses
- Hit time: time to access a memory including
  - Time to determine whether a hit or miss, and
  - Time to pass block to requestor

- **Miss**
  - If accessed data is absent
  - Block copied from lower to higher level
  - Then accessed data supplied from upper level
- Miss penalty: time taken
- Miss rate: misses/accesses = 1 – hit rate

- Miss (time) penalty
  - Time to fetch a block from lower level upon a miss, including
    - Time to access the block
    - Time to transfer it between levels
    - Time to overwrite the higher level block
    - Time to pass block to requestor (CPU)

Cache Example

First miss, then write in 0xFFF0002, (then hit), then write into R5

lw R2 ← mem[26]

Requested mem addr	Block (word) Addr	Hit/miss	Cache block
11010 00	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**0x5DC60007**
011	N		
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

Cache memory

Word Addr	Data
00000 (0)	0x81230431
00001 (1)	0xABCD3305
…	…
10101 (21)	0x12345678
10110 (22)	0x05ACF011
…	…
11010 (26)	**0x5DC60007**
…	…
11110 (30)	0x00000000
11111 (31)	0x00000000

Main memory

Requested mem Addr	Block (word) Addr	Hit/miss	Cache block
1000000(lw)	10 000	Miss	000
0001100(sw R5)	00 011	Miss	011
1000000(lw)	10 000	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**0x8765ABCD**
001	N		
010	Y	11	(R4)
011	**Y**	**00**	**0xFFFF0002→(R5)**
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

Word Addr	Data
00000(0)	0x81230431
00001(1)	0xABCD3305
00010(2)	0xFFFF0001
00011(3)	**0xFFFF0002**
…	…
10000(16)	**0x8765ABCD**
…	…
10101(21)	0x12345678
10110(22)	0x05ACF011
…	…
11010(26)	0x5DC60007
…	…
11110(30)	0x00000000
11111(31)	0x00000000

HW5

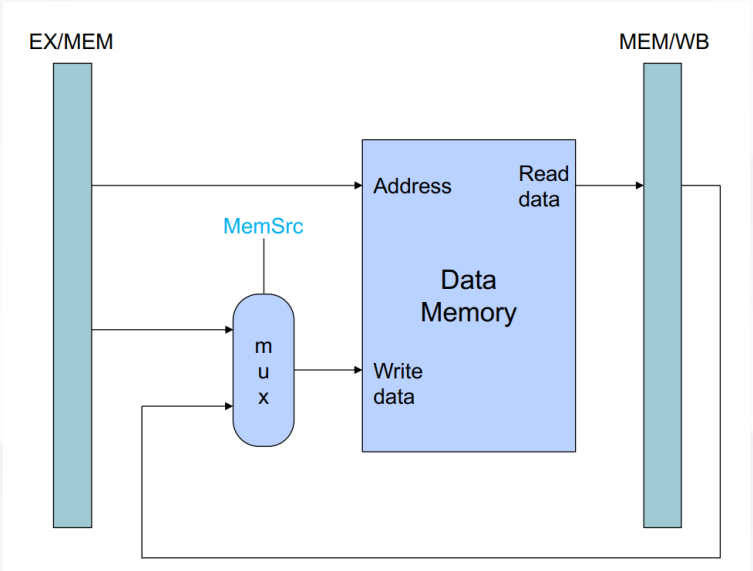
d) What could potentially be the smallest number of clock cycles to execute the instructions correctly? What would you do to achieve that? (15 points)

MEM-to-MEM data hazard could be completely removed by a new forwarding path. (5 points)

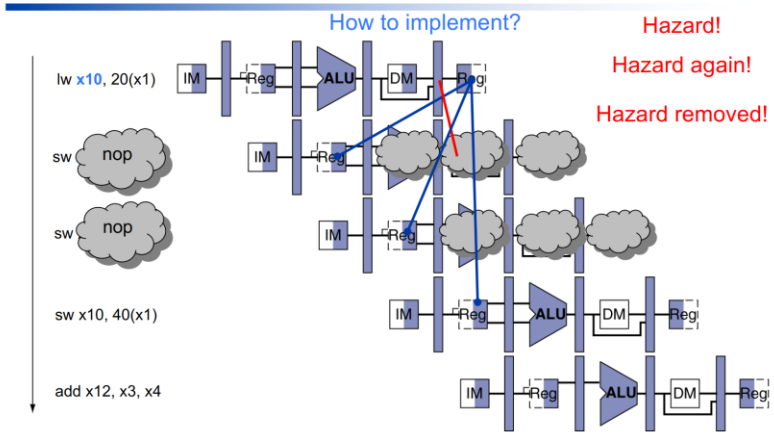
```
L1: add  x8,x9,x10
L2: lw   x3,8(x8)
L3: sw   x3,-12(x8)
L4: lb   x6,1(x3)
nop
L5: or   x8,x9,x6
```

In this case, the smallest number of instructions is 6, thus the smallest number of clock cycles is 6+4 = 10. (3 points)

To do this, the following need be done. (7 points)



Example: Another Hazard



This delays the execution too much, can it be fixed with forwarding?

39