



# Computer Organization

---

## Lab14 Cache(2)



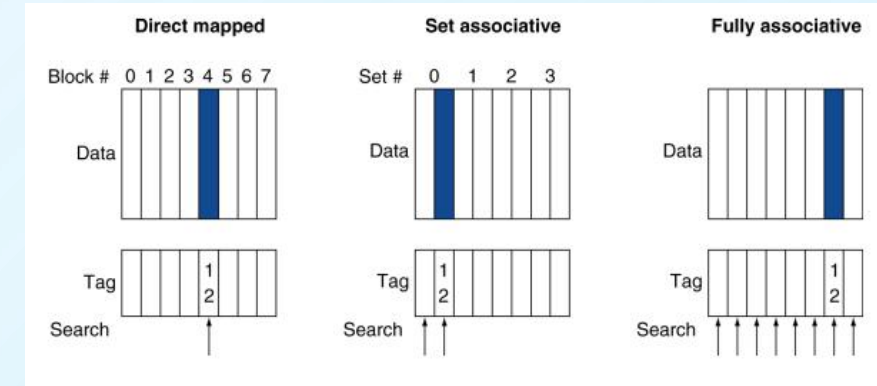
Performance



## Topic

### ➤ Cache: Types and Performance

- Direct Mapped Cache
- Fully Associative Cache
- N-way Set Associative Cache



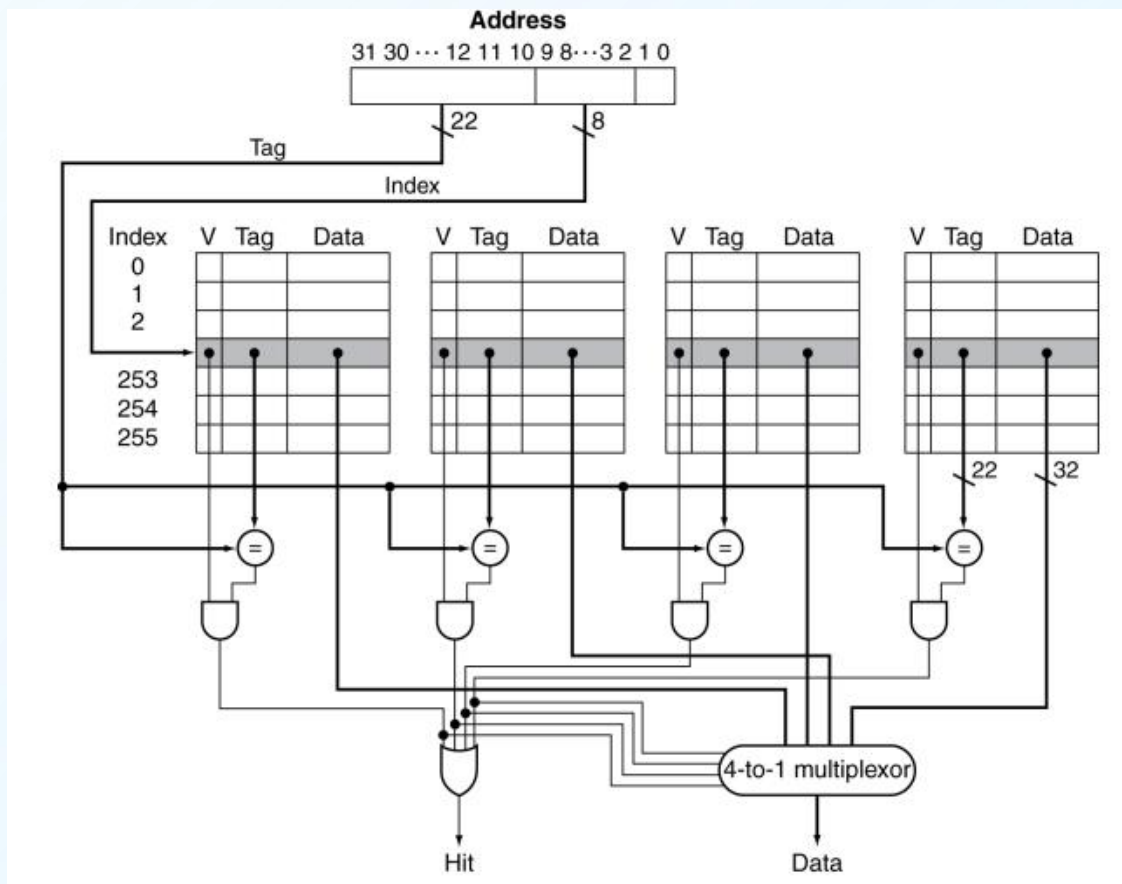
### ➤ To achieve better cache performance

- Suggestions on programming

### ➤ Vivado suggestion (optional)



# 3 types of Cache



## ➤ N-way Set Associative

- each set contains **n** entries
- Block number determines which set
- Search all entries in a given set at once
- **n** comparators (less expensive)

## ➤ Direct Mapped

- a given block only mapped to 1 cache entry
- search **1** entry at once
- **1** comparator (least expensive)

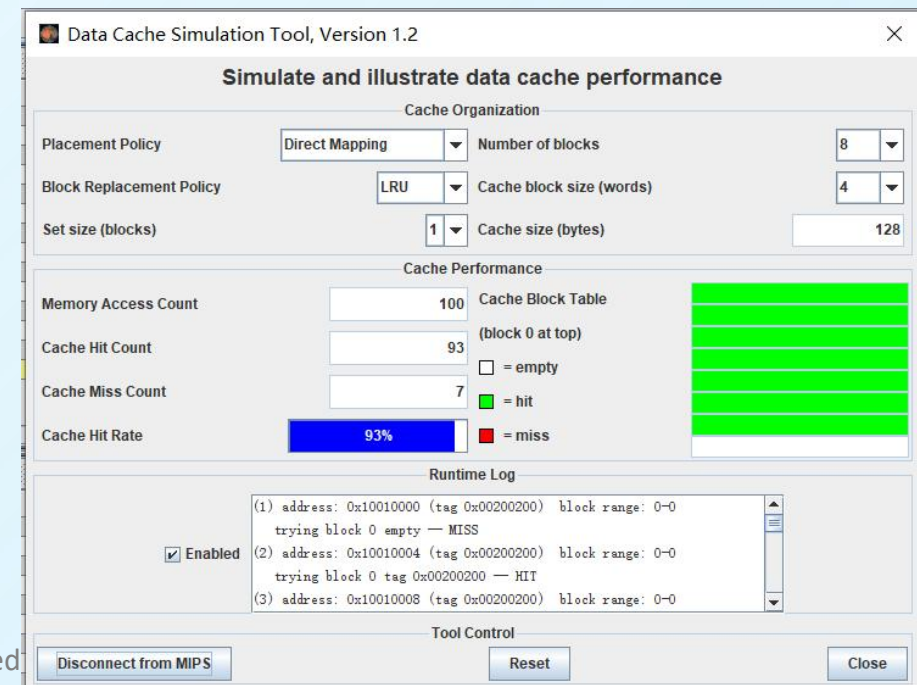
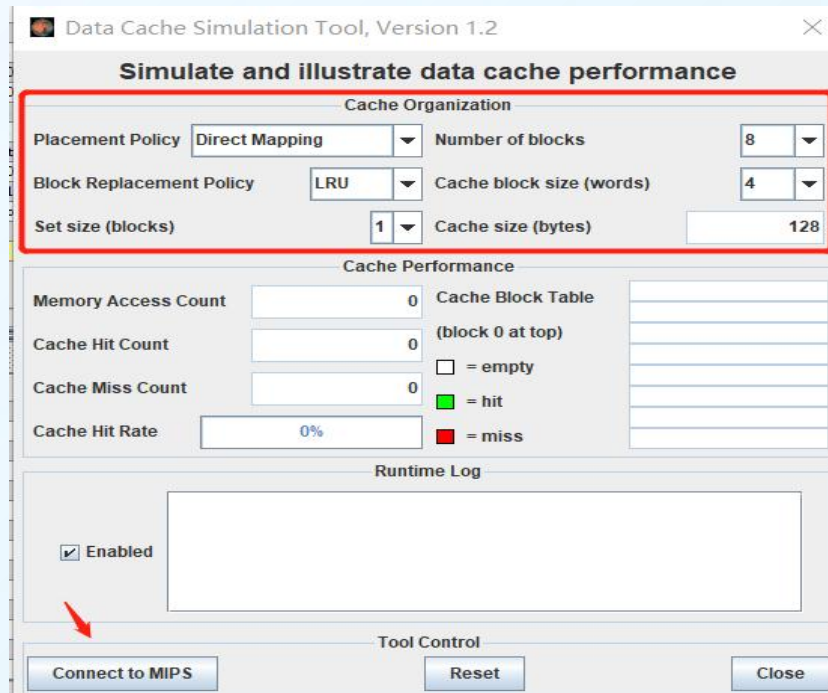
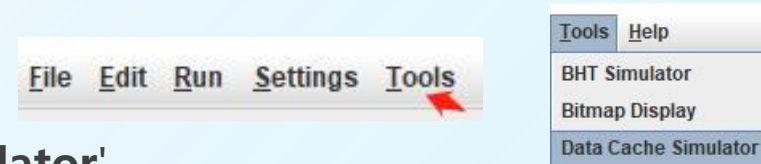
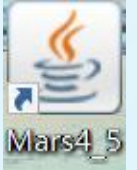
## ➤ Full Associative

- allow a given block to go in **any** cache entry
- requires **all** entries to be searched at once
- comparator per entry (most expensive)



# 'Data Cache Simulator' of Mars

- 1. Open an assembly source file in **Mars**(a Simulator on MIPS)
- 2. Assemble the asm file.
- 3. Open '**Data Cache Simulator**' of '**Tools**'
- 4. Set the '**Cache Organization**' of '**Data Cache Simulator**'
- 5. Click '**Connect to MIPS**' in left bottom of '**Data Cache Simulator**'
- 6. Run the current program





# Set the 'Cache Organization' of 'Data Cache Simulator'

Here are settings on different 'Cache Organization', the 'Cache block size(word)' are assumed to be 1

For a cache with 8 blocks

One-way set associative  
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Cache Organization

Placement Policy: Direct Mapping Number of blocks: 8

Block Replacement Policy: LRU Cache block size (words): 1

Set size (blocks): 1 Cache size (bytes): 32

Cache Organization

Placement Policy: N-way Set Associative Number of blocks: 8

Block Replacement Policy: LRU Cache block size (words): 1

Set size (blocks): 2 Cache size (bytes): 32

Cache Organization

Placement Policy: N-way Set Associative Number of blocks: 8

Block Replacement Policy: LRU Cache block size (words): 1

Set size (blocks): 4 Cache size (bytes): 32

Cache Organization

Placement Policy: Fully Associative Number of blocks: 8

Block Replacement Policy: LRU Cache block size (words): 1

Set size (blocks): 8 Cache size (bytes): 32



# Direct Mapped Cache performance

```
.data
array: .word 1,1,1
tmp: .word 0 : 100
.text
la $t0, array
li $t1, 25
loop:
    lw $t3, 0($t0)
    lw $t4, 4($t0)
    lw $t5, 8($t0)

    add $t2, $t3, $t4
    add $t2, $t2, $t5

    sw $t2, 12($t0)

    addi $t0, $t0, 16
    addi $t1, $t1, -1
    bgtz $t1, loop

li $v0, 10
syscall
```

➤ **512Byte =**

**32 Blocks \* 4 words/every block \* 4 Bytes/every word**

➤ There are totally 25 miss and **75** hit in 100 accessing, cache hit rate is **75%**.

➤ **512Byte =**

**16 Blocks \* 8 words/every block \* 4 Bytes/every word**

➤ There are totally 13 miss and **87** hit in 100 accessing, cache hit rate is **87%**.

--> Here **bigger** size of **cache block** lead to **higer** cache hit rate.



# Direct Mapped Cache continued

```
.data
    blk0: .word 1:32
    blk1: .word 0:32

.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32

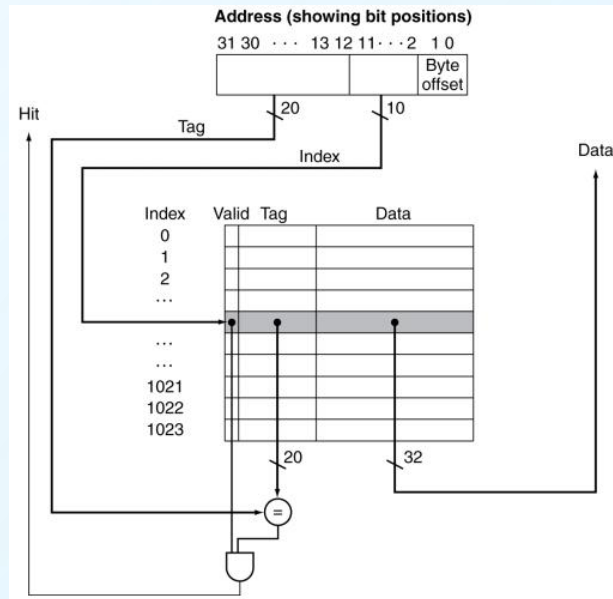
loop:
    lw $t2,blk0($t0)

    add $t2,$t2,$t0
    sllv $t2,$t2,$s0

    sw $t2,blk1($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop

    li $v0,10
    syscall
```



Q1. While running the demo on the MIPS CPU simulator(Mars), How many time of memory access?

Q2. While there is a **Direct Map Cache(size: 128Byte)** work with the CPU, what's the cache hit rate on the following settings?

## Feature1)

ByteOffset: 2 bit-width  
index: 5 bit-width

## Feature2)

ByteOffset: 4 bit-width  
index: 3 bit-width

Tips: “lw \$t2,blk0(\$t0)” , “sw \$t2,blk1(\$t0)” used here are pseudo instructions provide by **Mars**



# Direct Mapped Cache continued

```
.data
    blk0: .word 1:32
    blk1: .word 0:32

.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32

loop:
    lw $t2,blk0($t0)

    add $t2,$t2,$t0
    sliv $t2,$t2,$s0

    sw $t2,blk1($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop

    li $v0,10
    syscall
```

## Direct Map Cache

size: 128Byte

Feature1)

**ByteOffset: 2 bit-width**

**Index: 5 bit-width**

cache hit rate is 0!!

Would wider the size of cache  
block bring better cache hit rate?

Data Cache Simulation Tool, Version 1.2

**Simulate and illustrate data cache performance**

Cache Organization

Placement Policy: Direct Mapping Number of blocks: 32

Block Replacement Policy: LRU Cache block size (words): 1

Set size (blocks): 1 Cache size (bytes): 128

Cache Performance

Memory Access Count: 64 Cache Block Table (block 0 at top)

Cache Hit Count: 0

Cache Miss Count: 64

Cache Hit Rate: 0%

Legend: ☐ = empty ☒ = hit ☒ = miss

Runtime Log





# Direct Mapped Cache continued

```
.data
    blk0: .word 1:32
    blk1: .word 0:32

.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32

loop:
    lw $t2,blk0($t0)

    add $t2,$t2,$t0
    sllv $t2,$t2,$s0

    sw $t2,blk1($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop

    li $v0,10
    syscall
```

## Direct Map Cache

size: 128Byte

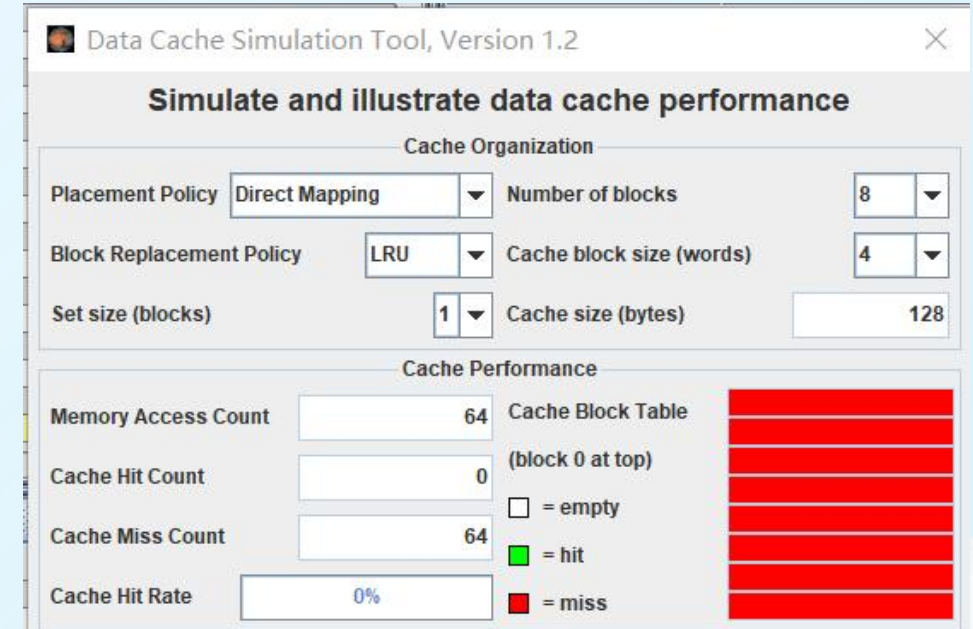
Feature2)

**ByteOffset:** 4 bit-width

**Index:** 3 bit-width

cache hit rate is 0!!

Would wider the size of cache  
block bring better cache hit  
rate?





# Fully associative Cache

```
.data
    blk0: .word 1:32
    blk1: .word 0:32

.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32

loop:
    lw $t2,blk0($t0)

    add $t2,$t2,$t0
    sllv $t2,$t2,$s0

    sw $t2,blk1($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop

    li $v0,10
    syscall
```

- Fully associative Cache
  - **Allow a given block to go in ANY cache entry**
  - Requires all entries to be searched at once
  - Comparator per entry

Q1. While there is a **Fully associative Cache(size: 128Byte)** work with the CPU, what's the cache hit rate on the following settings?

Feature1(cache size: 128B)		Feature2(cache size: 128B)	
ByteOffset	2 bit-width	ByteOffset	4 bit-width

Tips: 'index' is meaningless in fully associative cache



# Fully associative Cache continued

```
.data
    blk0: .word 1:32
    blk1: .word 0:32

.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32

loop:
    lw $t2,blk0($t0)

    add $t2,$t2,$t0
    sllv $t2,$t2,$s0

    sw $t2,blk1($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop

    li $v0,10
    syscall
```

Would Fully associative cache bring higher cache hit rate?

## Fully associative Cache

size: 128Byte

Feature1)

ByteOffset: 2 bit-width

cache hit rate is 0!!

Would wider the size of cache  
block bring better cache hit  
rate in the cache?

Data Cache Simulation Tool, Version 1.2

### Simulate and illustrate data cache performance

Cache Organization

Placement Policy: Fully Associative Number of blocks: 32

Block Replacement Policy: LRU Cache block size (words): 1

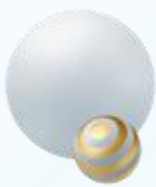
Set size (blocks): 32 Cache size (bytes): 128

Cache Performance

Memory Access Count: 64 Cache Hit Count: 0 Cache Miss Count: 64 Cache Hit Rate: 0%

Cache Block Table (block 0 at top)

Legend: ☐ = empty ☒ = hit ☒ = miss



# Fully associative Cache continued

```
.data
    blk0: .word 1:32
    blk1: .word 0:32
.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32
loop:
    lw $t2,blk0($t0)

    add $t2,$t2,$t0
    sllv $t2,$t2,$s0

    sw $t2,blk1($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop

    li $v0,10
    syscall
```

Would Fully associative cache bring higher cache hit rate?

## Fully associative Cache

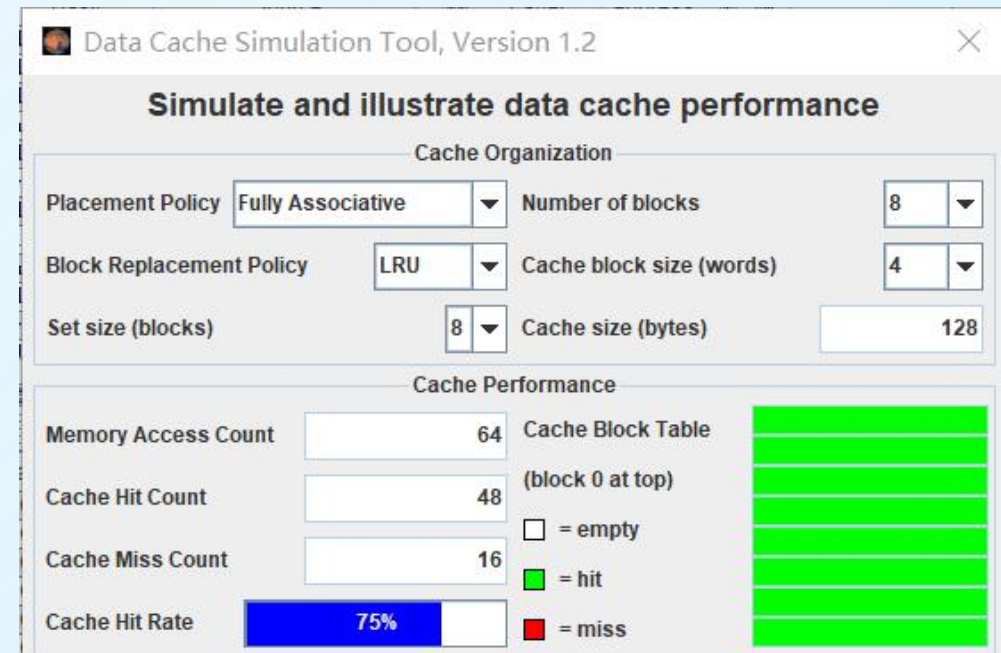
size: 128Byte

Feature1)

ByteOffset: 4 bit-width

cache hit rate is 94%!!

Would wider the size of cache  
block bring better cache hit  
rate in the cache?

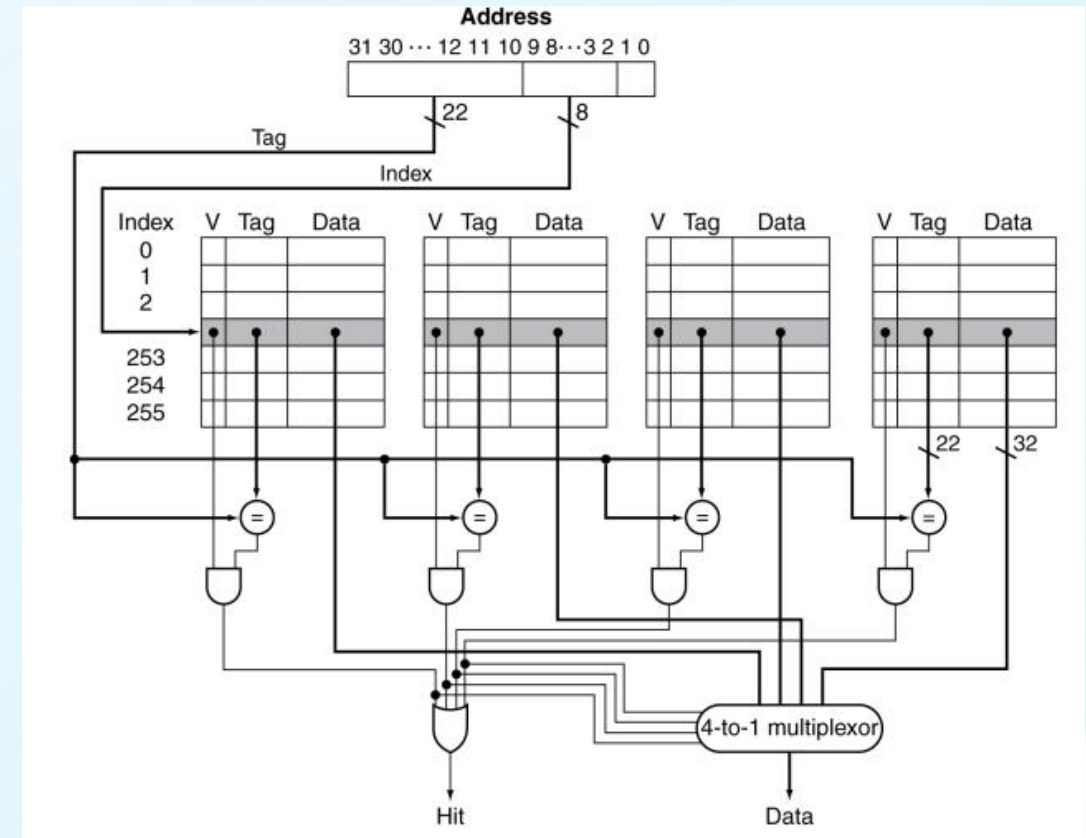




# N-way Set Associative Cache

- N-way set associative Cache
  - Each set contains **n** entries
  - Block number determines which set
    - (Block number) modulo (#sets in cache)
  - Search all entries in a given set at once
  - **n** comparators

Feature1 (128B, 2-way)		Feature2(128B, 2-way)	
ByteOffset	2 bit-width	ByteOffset	4 bit-width
set Index	4 bit-width	set Index	2 bit-width



Fully associative ← N-way Set associative → Direct Mapping



# N-way Set Associative Cache continued

**2-way set associative cache : 'Set size' is 2(There are 2 blocks in a set)**

```
.data
    blk0: .word 1:32
    blk1: .word 0:32

.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32

loop:
    lw $t2,blk0($t0)

    add $t2,$t2,$t0
    sllv $t2,$t2,$s0

    sw $t2,blk1($t0)

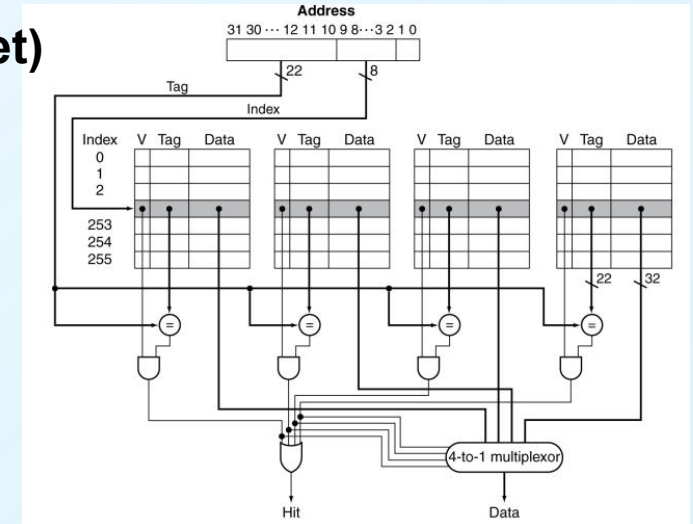
    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop

    li $v0,10
    syscall
```

**N-way set associative Cache**  
size: 128Byte  
Feature1)  
**2-way set associative**  
**ByteOffset: 2 bit-width**  
set Index: 4 bit-width

cache hit rate is 0%!!

Would wider the size of cache  
block bring better cache hit  
rate in the cache?



**Simulate and illustrate data cache performance**

**Cache Organization**

Placement Policy: **N-way Set Associative** Number of blocks: **32**

Block Replacement Policy: **LRU** Cache block size (words): **1**

Set size (blocks): **2** Cache size (bytes): **128**

**Cache Performance**

Memory Access Count: **64** Cache Hit Count: **0**

Cache Miss Count: **64** Cache Hit Rate: **0%**

Cache Block Table (block 0 at top)

Legend: ☐ = empty, ☒ = hit, ☒ = miss



# N-way Set Associative Cache continued

**2-way set associative cache : Set size is 2(There are 2 blocks in a set)**

.data

blk0: .word 1:32

blk1: .word 0:32

.text

add \$t0,\$0,\$0

add \$s0,\$0,\$0

addi \$t1,\$0,32

loop:

lw \$t2,blk0(\$t0)

add \$t2,\$t2,\$t0

sllv \$t2,\$t2,\$s0

sw \$t2,blk1(\$t0)

addi \$t0,\$t0,4

addi \$s0,\$s0,1

bne \$s0,\$t1,loop

li \$v0,10

syscall

## N-way set associative Cache

size: 128Byte

Feature2)

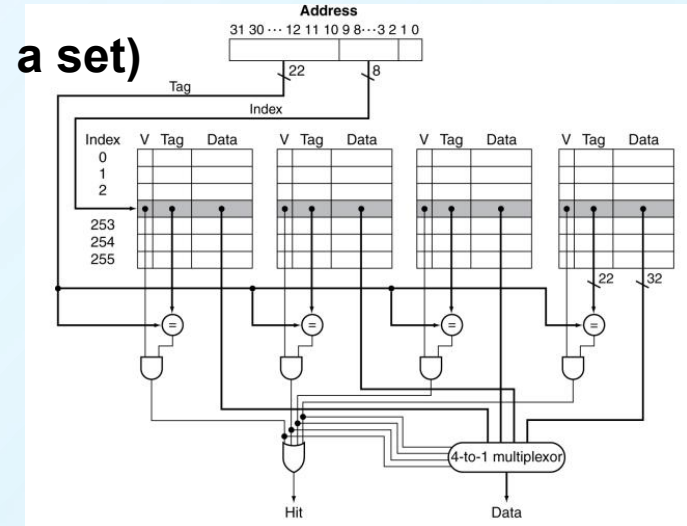
**2-way set associative**

**ByteOffset: 4 bit-width**

**Set Index: 2 bit-width**

cache hit rate is 75%!!

Would wider the size of cache  
block bring better cache hit  
rate in the cache?



## Simulate and illustrate data cache performance

### Cache Organization

Placement Policy	N-way Set Associative	Number of blocks	8
Block Replacement Policy	LRU	Cache block size (words)	4
Set size (blocks)	2	Cache size (bytes)	128

### Cache Performance

Memory Access Count	64	Cache Block Table	
Cache Hit Count	48	(block 0 at top)	
Cache Miss Count	16	<input type="checkbox"/> = empty	
Cache Hit Rate	75%	<input checked="" type="checkbox"/> = hit	
		<input type="checkbox"/> = miss	

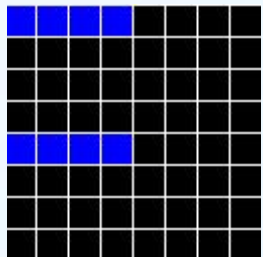


# Achieve better cache performance by programming

Which one has better cache performance? Demo1 or Demo2? Why?

Demo1

```
int a[size];  
int b[size];
```



```
.data #Demo1
```

```
blk0: .word 0:32
```

```
blk1: .word 0:32
```

```
.text
```

```
add $t0,$0,$0
```

```
add $s0,$0,$0
```

```
addi $t1,$0,32
```

```
loop:
```

```
lw $t2,blk0($t0)
```

```
add $t2,$t2,$t0
```

```
srl $t2,$t2,31
```

```
sw $t2,blk1($t0)
```

```
addi $t0,$t0,4
```

```
addi $s0,$s0,1
```

```
bne $s0,$t1,loop
```

```
li $v0,10
```

```
syscall
```

Demo2

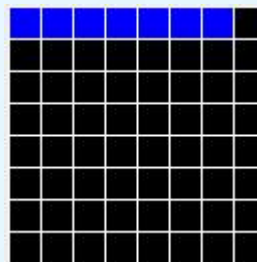
```
struct merge{
```

```
int a;
```

```
int b;
```

```
};
```

```
struct merge marr[size];
```



```
.data #Demo2
```

```
mblk: .word 0:64
```

```
.text
```

```
add $t0,$0,$0
```

```
add $s0,$0,$0
```

```
addi $t1,$0,32
```

```
loop:
```

```
lw $t2,mblk($t0)
```

```
add $t2,$t2,$t0
```

```
srl $t2,$t2,31
```

```
addi $t0,$t0,4
```

```
sw $t2,mblk($t0)
```

```
addi $t0,$t0,4
```

```
addi $s0,$s0,1
```

```
bne $s0,$t1,loop
```

```
li $v0,10
```

```
syscall
```





# Achieve better cache performance by programming continued

Demo1  
**for(i=0;i<size;i++)**  
    **B[i] = A[i];**  
**for(i=0;i<size;i++)**  
    **C[i] = A[i];**

```
add $t0,$0,$0
add $s0,$0,$0

loop2: # Demo 1 : 2/2
    lw $t2,blk0($t0)
    add $t2,$t2,$t0
    srl $t2,$t2,31

    sw $t2,blk2($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop2

li $v0,10
syscall
```

```
.data # Demo 1 : 1/2
    blk0: .word 0:32
    blk1: .word 0:32
    blk2: .word 0:32

.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32

loop:
    lw $t2,blk0($t0)
    add $t2,$t2,$t0
    srl $t2,$t2,31

    sw $t2,blk1($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop
```

Demo2  
**for(i=0;i<size;i++){**  
    **B[i] = A[i];**  
    **C[i] = A[i];**  
**}**

Which one has  
better cache  
performance?  
Demo1 or Demo2?  
Why?

```
.data
    blk0: .word 0:32
    blk1: .word 0:32
    blk2: .word 0:32

.text
    add $t0,$0,$0
    add $s0,$0,$0
    addi $t1,$0,32

loop:
    lw $t2,blk0($t0)
    add $t2,$t2,$t0
    srl $t2,$t2,31

    sw $t2,blk1($t0)
    sw $t2,blk2($t0)

    addi $t0,$t0,4
    addi $s0,$s0,1
    bne $s0,$t1,loop

li $v0,10
syscall
```



# Achieve better cache performance by programming continued

```
.data #Demo1P1/2
# 32*2 word (rows: 2, columns: 32)
matrix: .space 256

.macro getindex(%ans,%i,%j)
    sll %ans,%i,complete here
    add %ans,%ans,%j
    sll %ans,%ans,complete here
.end_macro

.text
addi $t0,$0,0 #i
addi $s0,$0,2

addi $t1,$0,0 #j
addi $s1,$0,32
```

```
loopi: #Demo1P2/2
beq $t0,$s0,loopiend

addi $t1,$0,0

loopj:
beq $t1,$s1,loopjend
getindex($a0,$t0,$t1)
lw $v0,matrix($a0)
addi $t1,$t1,1
j loopj

loopjend:
addi $t0,$t0,1
j loopi

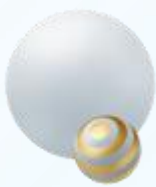
loopiend:
li $v0,10
syscall
```

Demo1

```
int matrix[2][32];

for( i=0;i<2;i++){
    for( int j=0;j<32;j++ )
        ...matrix[i][j] ...
}
```

Which one has better  
cache performance?  
Demo1 on page18 or  
Demo2 on page19?  
Why?



# Achieve better cache performance by programming continued

```
.data #Demo2P1/2
# 32*2 word (rows: 2, column: 32)
matrix: .space 256

.macro getIndex(%ans,%i,%j)
    sll %ans,%i,complete here
    add %ans,%ans,%j
    sll %ans,%ans,complete here
.end_macro

.text
addi $t0,$0,0 #i
addi $s0,$0,2

addi $t1,$0,0 #j
addi $s1,$0,32
```

```
loopj: #Demo2P2/2
beq $t1,$s1,loopjend

addi $t0,$0,0
loopi:
beq $t0,$s0,loopiend
getIndex($a0,$t0,$t1)
lw $v0, matrix($a0)
addi $t0,$t0,1
j loopi

loopiend:
addi $t1,$t1,1
j loopj

loopjend:
li $v0,10
syscall
```

## Demo2

```
int matrix[2][32];

for( j=0;j<32;j++ ){
    for( int i=0;i<2;i++ )
        ...matrix[i][j] ...
}
```

**Which one has better cache performance? Demo1 on page18 or Demo2 on page19? Why?**



## Practice

Q1. Answer questions on p16 and p17, complete the code for p18 and p19.

Q2. Choose any of these three questions, using the simulator('Data Cache Simulator' of 'Mars' ) to verify your conclusion through simulation.

NOTE: you are suggested to use the specified **4-way associate cache** (cache block size is 2word , cache size 128bytes).

2-1: Please determine the offset, index, and tag bit widths in the 32-bit address.

2-2: Complete the relevant configuration in Mars' cache simulation tool and verify your conclusion through simulation.





# Vivado suggestion-incremental Implement

## ➤ **Incremental implementation** in vivado:

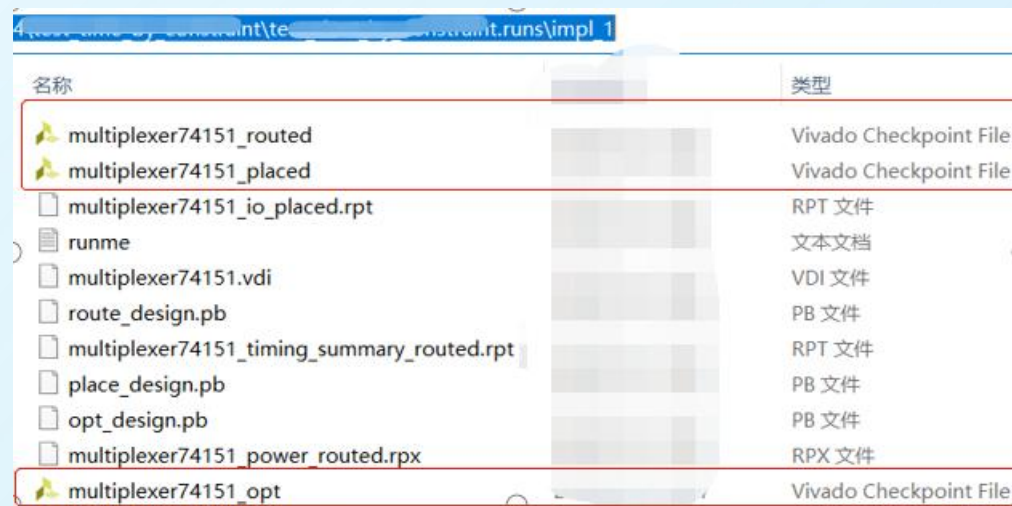
The incremental implementation in Vivado will reuse existing layout and wiring data to shorten runtime and generate predictable results. When the design has a **similarity of over 95%**, the **running time** of incremental layout cabling will be **reduced**, **otherwise** the incremental compiling is **not suggested**!

## ➤ The incremental implementation in Vivado needs **DCP**(abbreviated as Design CheckPoint)**file**.

➤ Dcp file is an encrypted, compressed binary file type that contains complete design information such as instantiation hierarchy, resource usage, temporal analysis data, constraints, and other important information.

➤ “xxxxrouted.dcp” file are more frequently applied in incremental implementations.

TIP: After implementation, the dcp file could be found in the “**runs\impl\_1**” of vivado project direcotry.



名称	类型
multiplexer74151_routed	Vivado Checkpoint File
multiplexer74151_placed	Vivado Checkpoint File
multiplexer74151_io_placed.rpt	RPT 文件
runme	文本文档
multiplexer74151.vdi	VDI 文件
route_design.pb	PB 文件
multiplexer74151_timing_summary_routed.rpt	RPT 文件
place_design.pb	PB 文件
opt_design.pb	PB 文件
multiplexer74151_power_routed.rpx	RPX 文件
multiplexer74151_opt	Vivado Checkpoint File



# Vivado suggestion-incremental Implement continued

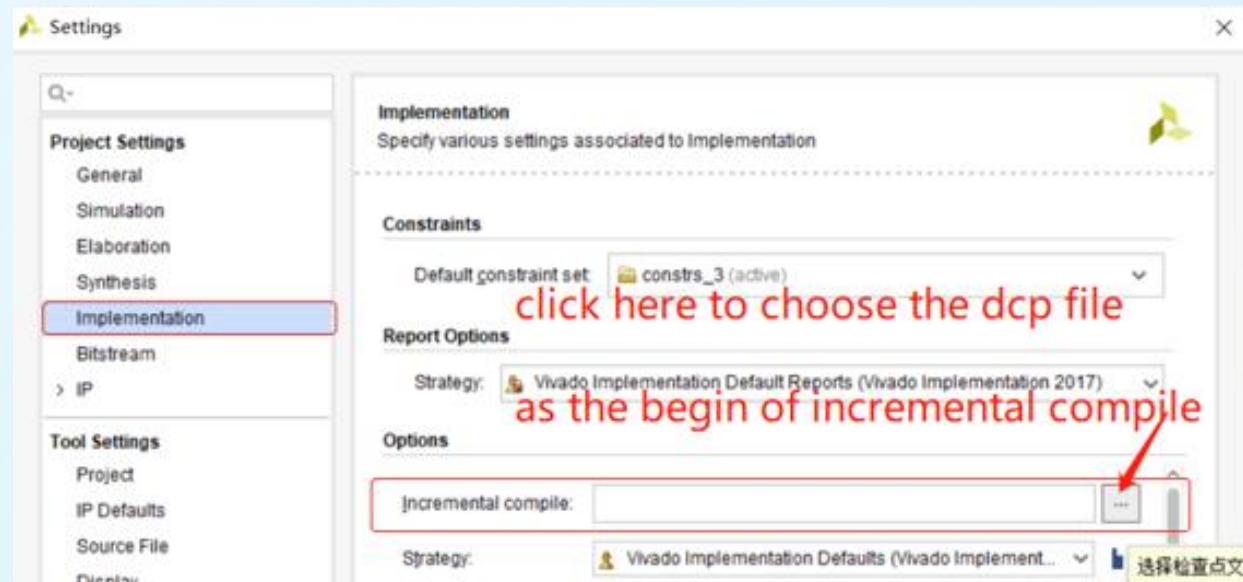
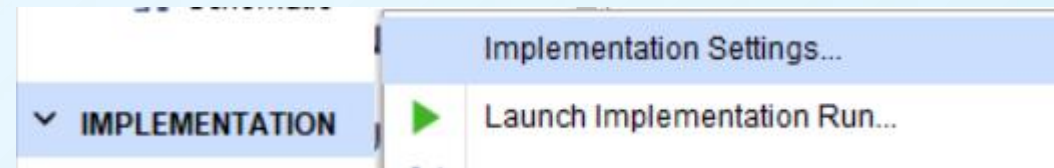
Incremental implementation steps:

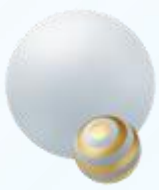
1. Do the initial implementation in vivado project to generate the dcp files
2. Copy the dcp files to another new directory(such as project/incremental\_compile)

3. set the implementation

3-1: **Right click “implementation”** in “Flow navigator” window, click **“Implementation Settings”** to invoke the “Settings” window

3-2: In **“Settings”** window, click **“Implementation”**, click the button on the end of **“Incremental compile”** to choose the **dcp file**(in the new directory) as the begin of incremental compile.



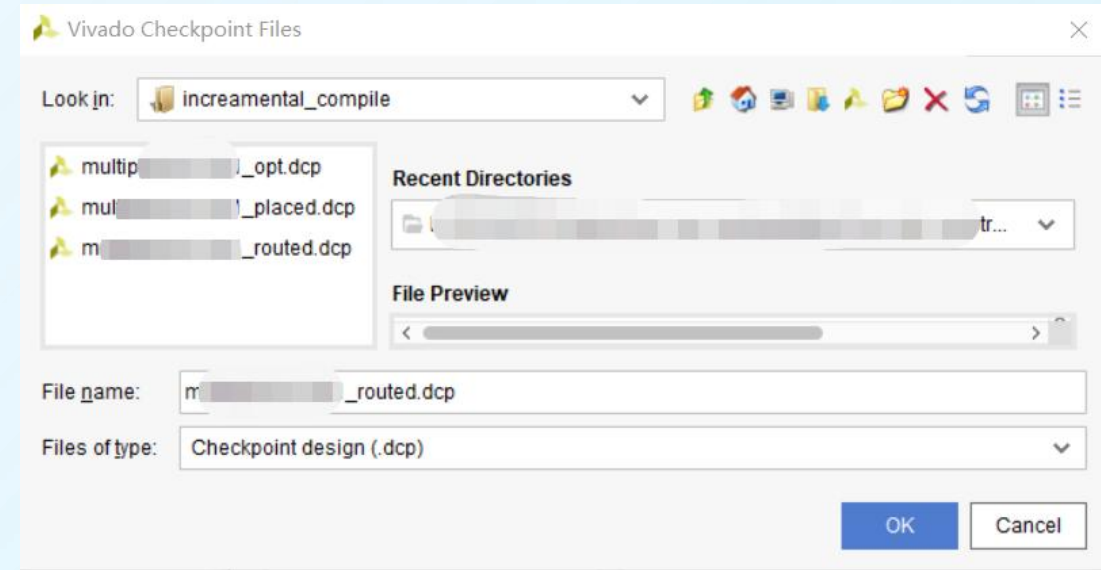


# Vivado suggestion-incremental Implement continued

Incremental implementation steps:

3-3. choose the “xxx\_route.dcp” file in the new directory(see step2 on last page) as the begin of incremental compile.

NOTE: DONOT use the “xxx\_route.dcp” in “**runs\impl\_x**” directory because the file would be updated durning the implementation.



4. do the implementation.

**NOTE: Only the updated code's similarity is over 95%, the running time of incremental implementation would be reduced, otherwise incremental implementation is not suggested.**