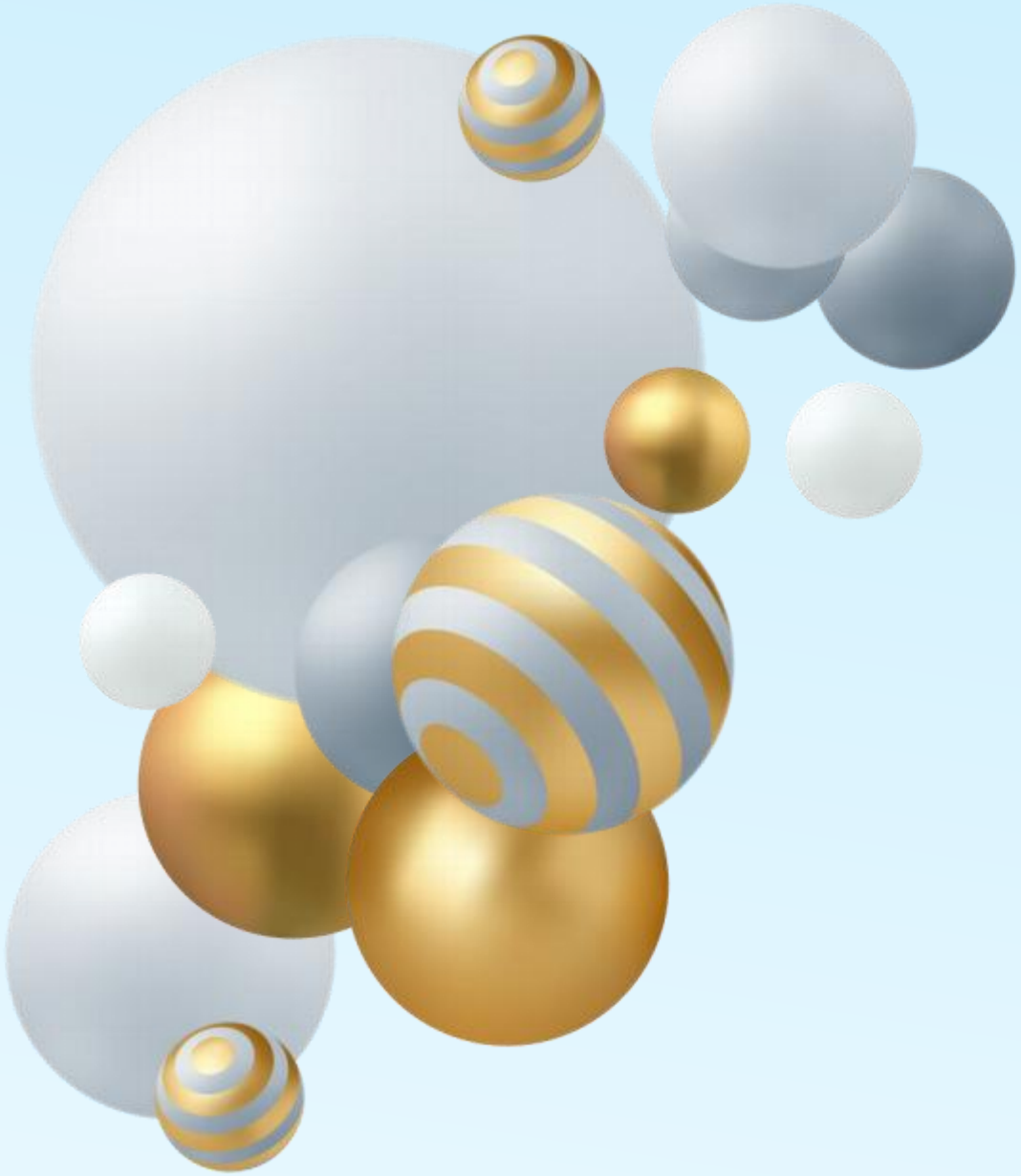
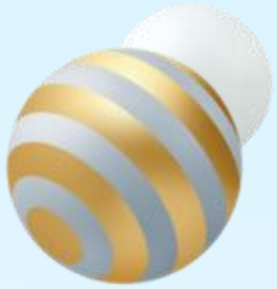


Computer Organization

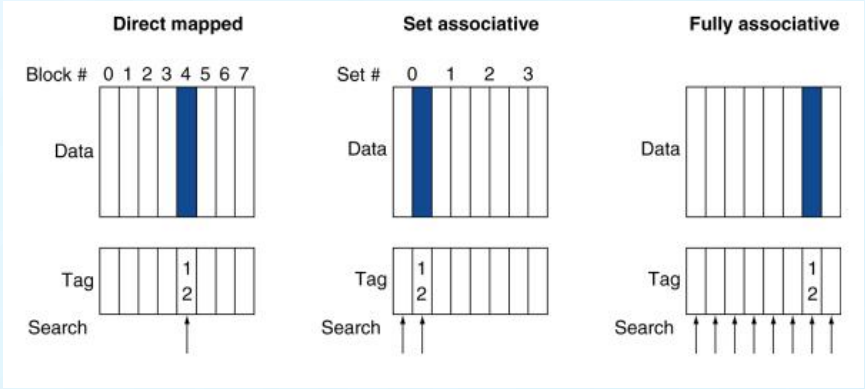
Lab14 Cache

Performance



Topic

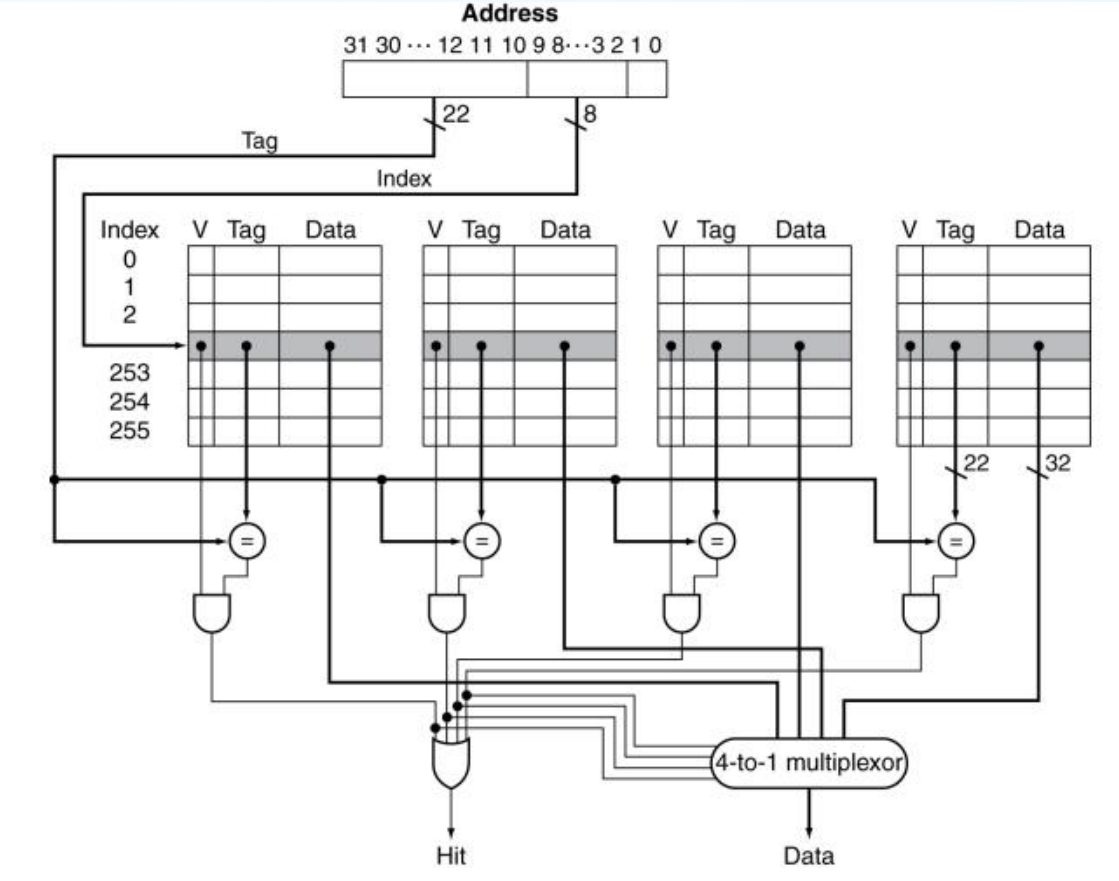
- Cache: Types and Performance
  - Direct Mapped Cache
  - Fully Associative Cache
  - N-way Set Associative Cache



- To achive better cache performance
  - Suggestions on prgramming

- Vivado suggestion (optional)

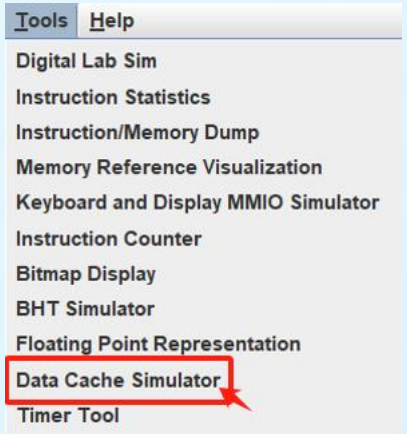
3 types of Cache

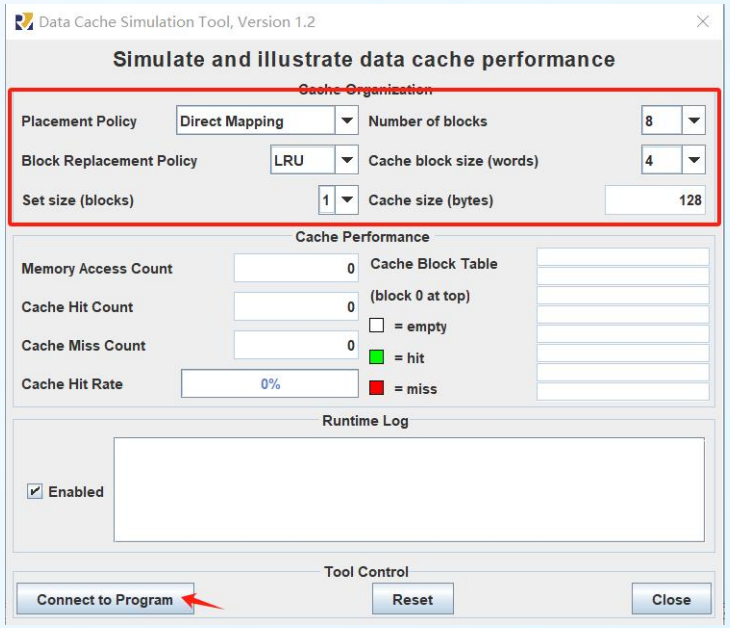


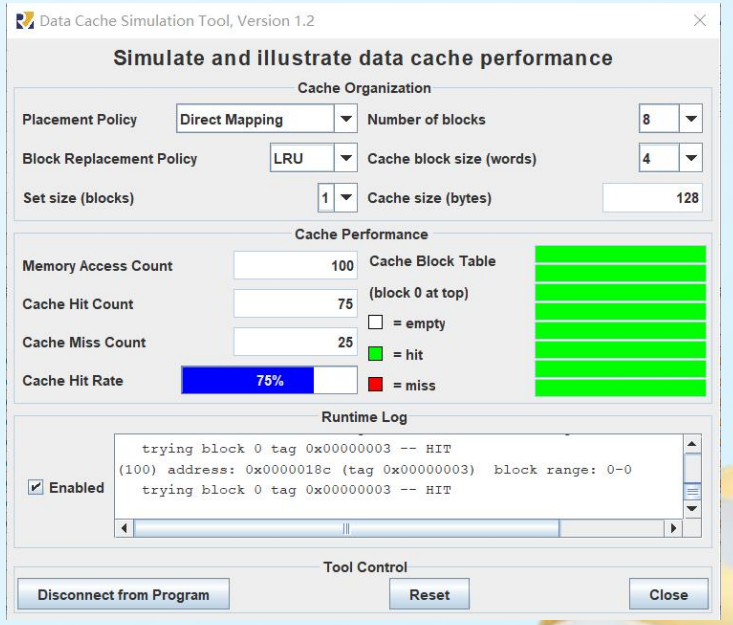
- **N-way Set Associative**
  - each set contains **n** entries
  - Block number determines which set
  - Search all entries in a given set at once
  - **n** comparators(less expensive)
- **Direct Mapped**
  - a given block only mapped to 1 cache entry
  - search **1** entry at once
  - **1** comparator(least 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 Rars

- 1. Open an assembly source file in **Rars**(a Simulator on RISC-V)
- 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 Program**' in left bottom of '**Data Cache Simulator**'
- 6. Run the current program



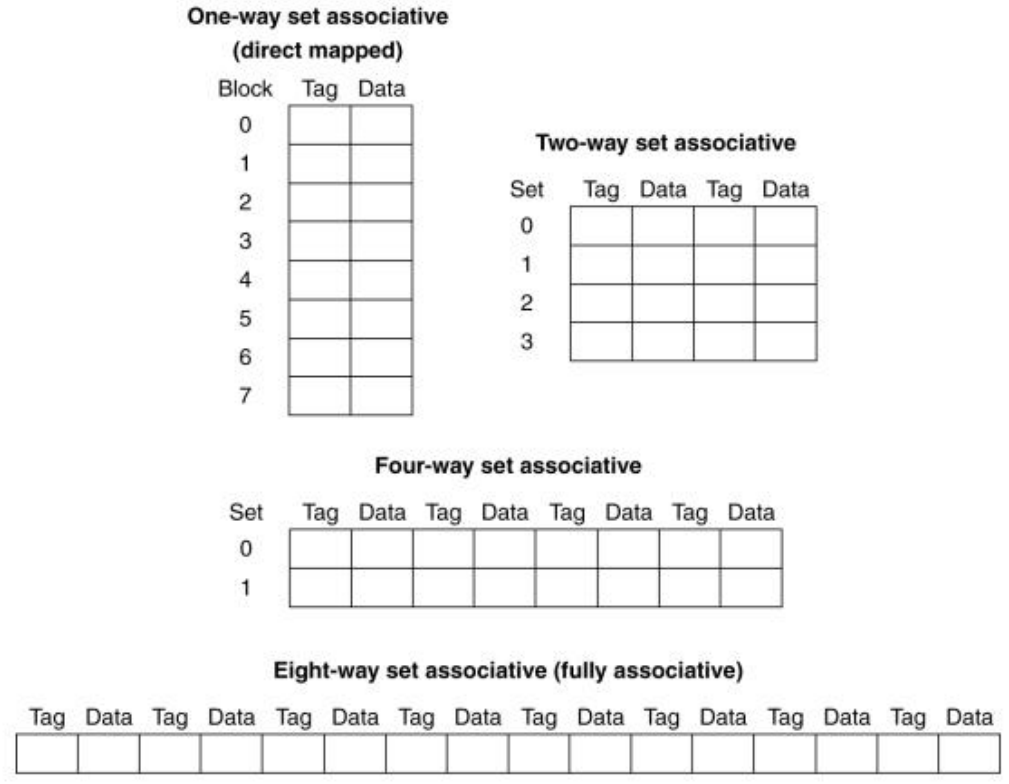


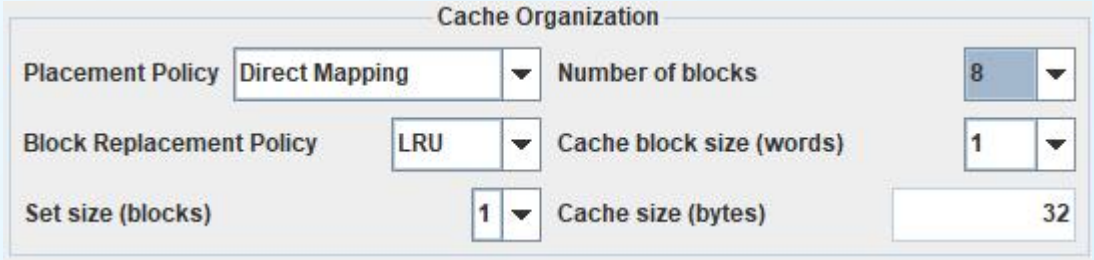


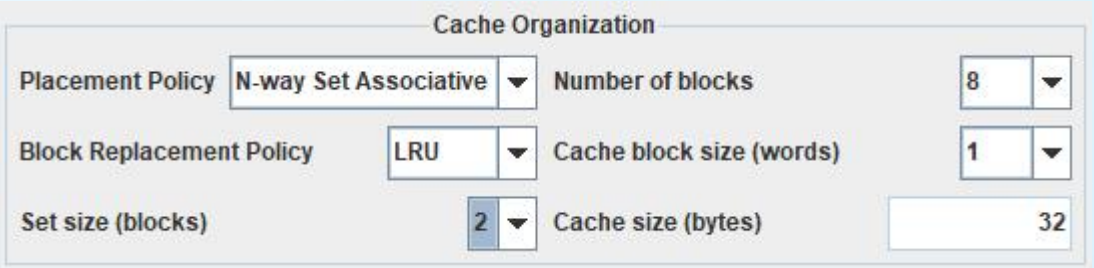
Set the ‘Cache Organization’ of ‘Data Cache Simulator’

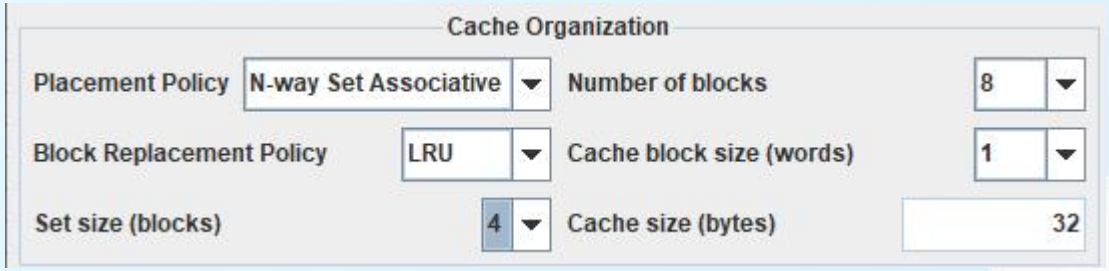
Here are settings on different ‘Cache Organization’, the ‘Cache block size(word)’ are assumed to be 1

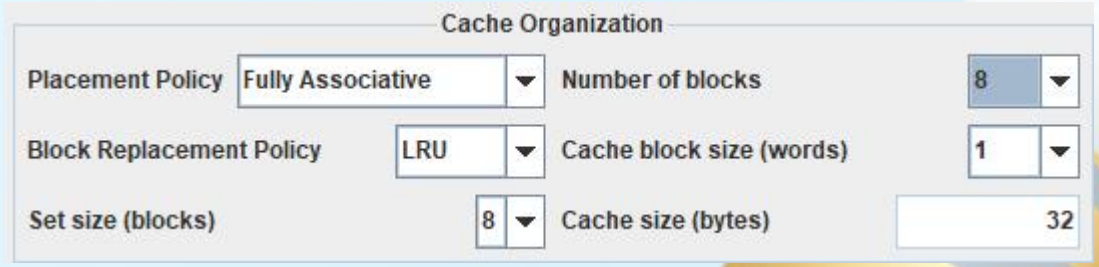
For a cache with 8 blocks











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 a7, 10
ecall
```

- **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 =**

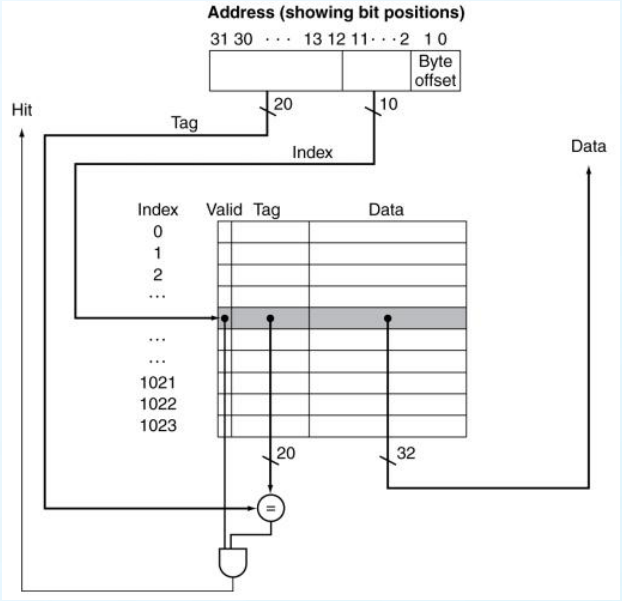
  16 Blocks * 8 words/every block * 4 Bytes/every word

  - 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,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)

        add t2,t2,t0
        sll t2,t2,s0

        sw t2,0(t6)

        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```



Tips: “**la t5,blk0**” , “**la t6,blk1**” used here are pseudo instructions provide by **Rars**

Q1. While running the demo on the RISC-V CPU simulator(Rars), 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

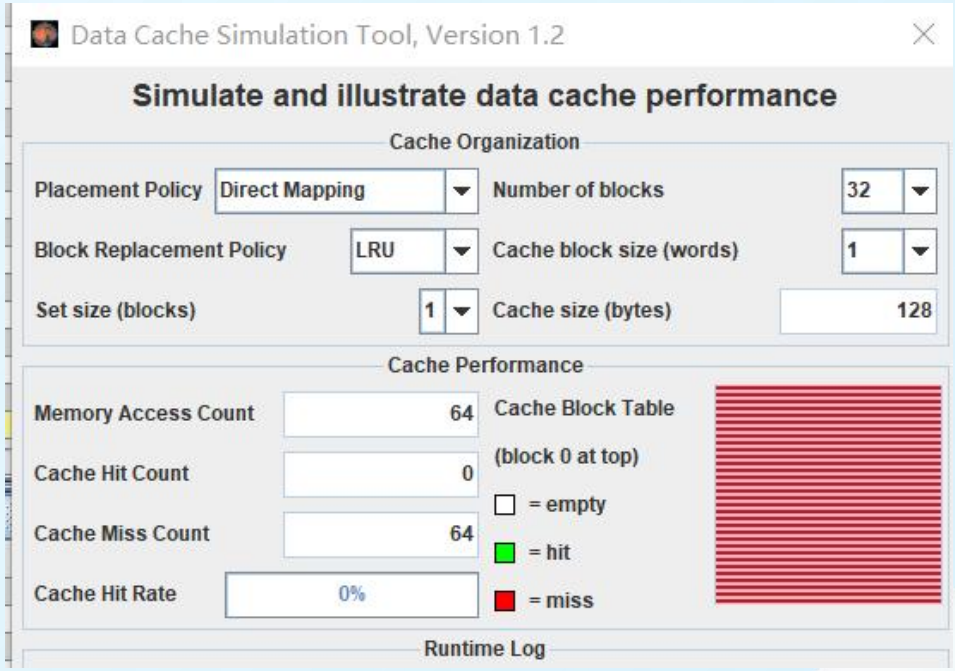
Direct Mapped Cache continued

```
.data
        blk0: .word 1:32
        blk1: .word 0:32
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)

        add t2,t2,t0
        sll t2,t2,s0

        sw t2,0(t6)

        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

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?



Direct Mapped Cache continued

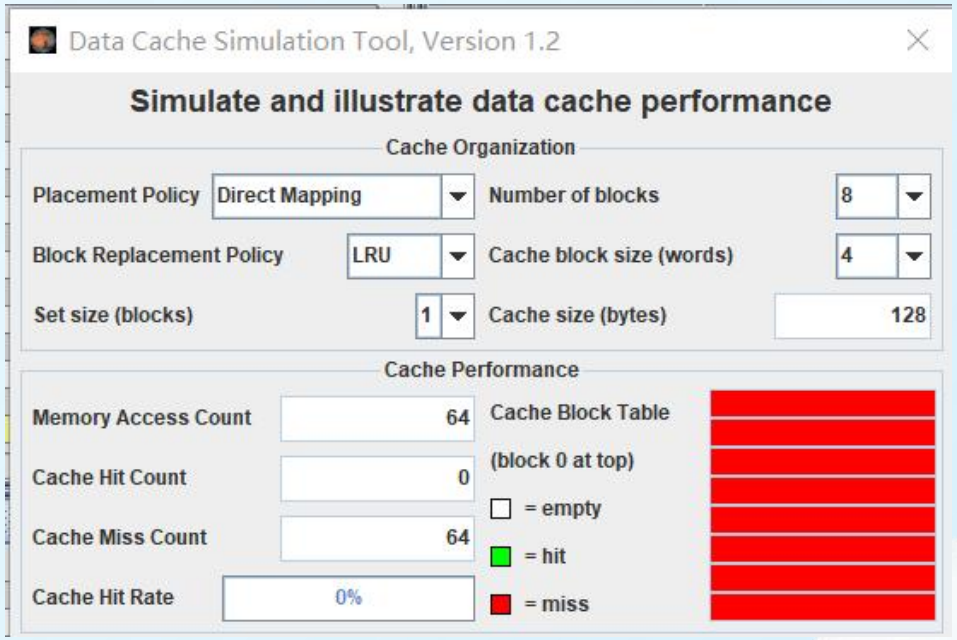
```
.data
        blk0: .word 1:32
        blk1: .word 0:32
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)

        add t2,t2,t0
        sll t2,t2,s0

        sw t2,0(t6)

        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

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,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)

        add t2,t2,t0
        sll t2,t2,s0

        sw t2,0(t6)

        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

- 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

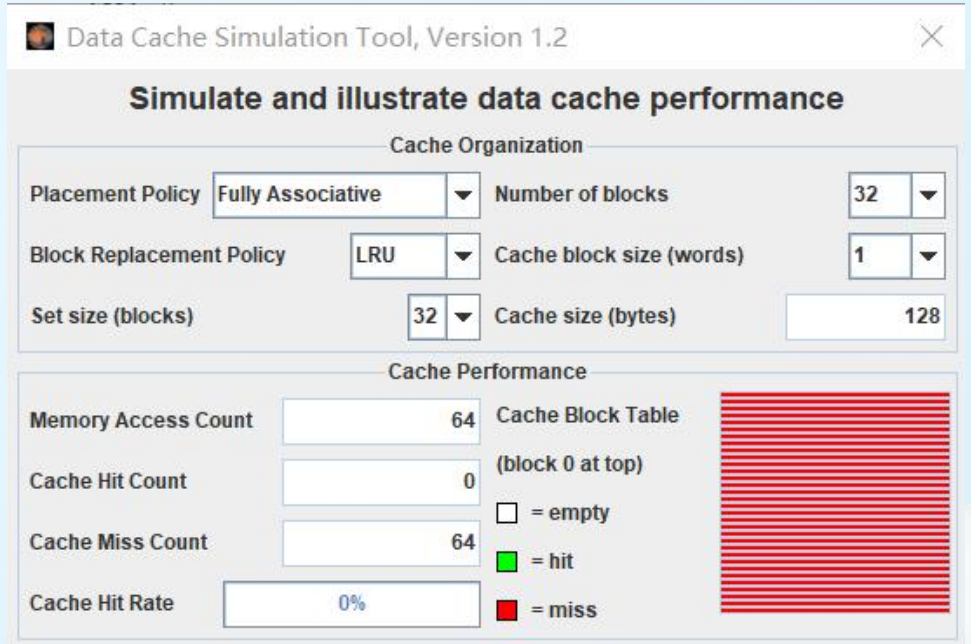
```
.data
        blk0: .word 1:32
        blk1: .word 0:32
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)

        add t2,t2,t0
        sll t2,t2,s0

        sw t2,0(t6)

        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

Would Fully associative cache bings 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?



Fully associative Cache continued

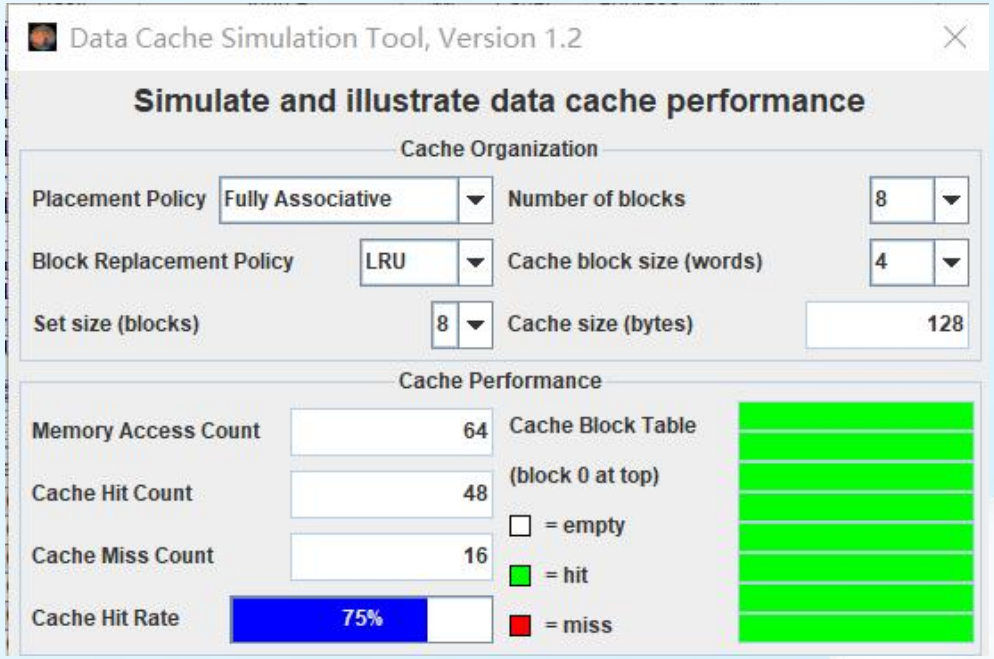
```
.data
        blk0: .word 1:32
        blk1: .word 0:32
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)

        add t2,t2,t0
        sll t2,t2,s0

        sw t2,0(t6)

        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

Would Fully associative cache bings higher cache hit rate?

Fully associative Cache
size： 128Byte
Feature1)
ByteOffset： 4 bit-width

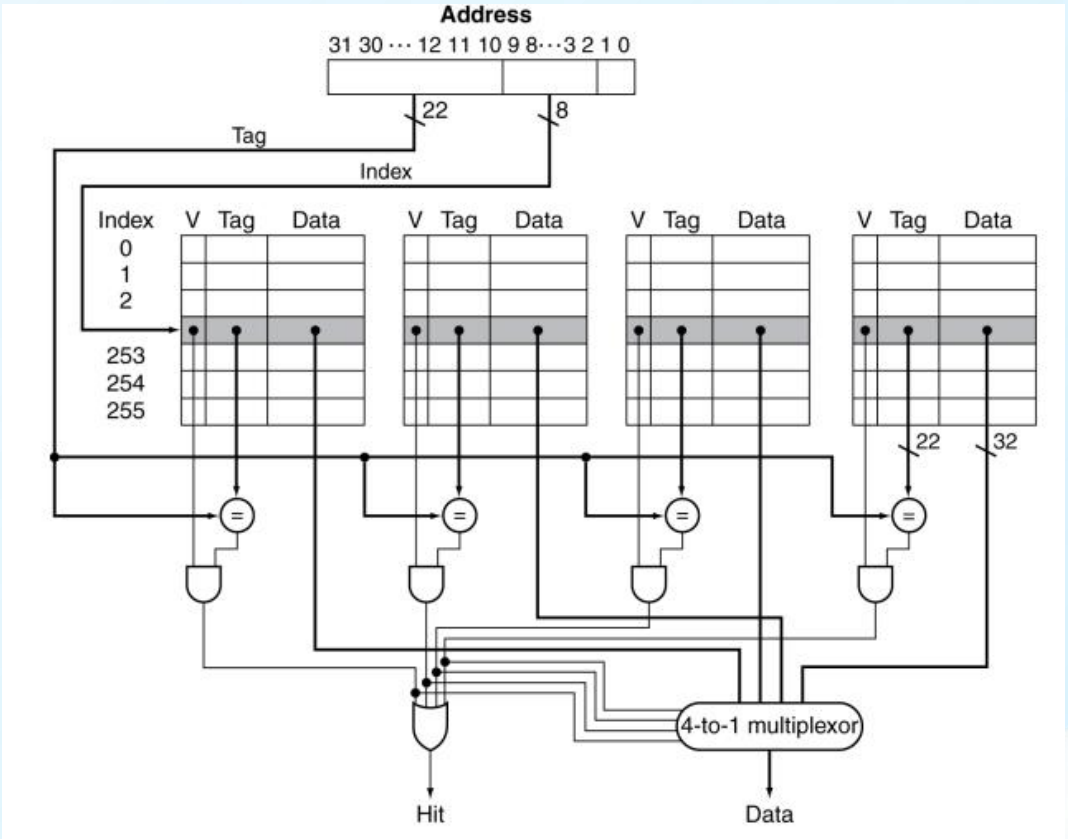
cache hit rate is 94%!!

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

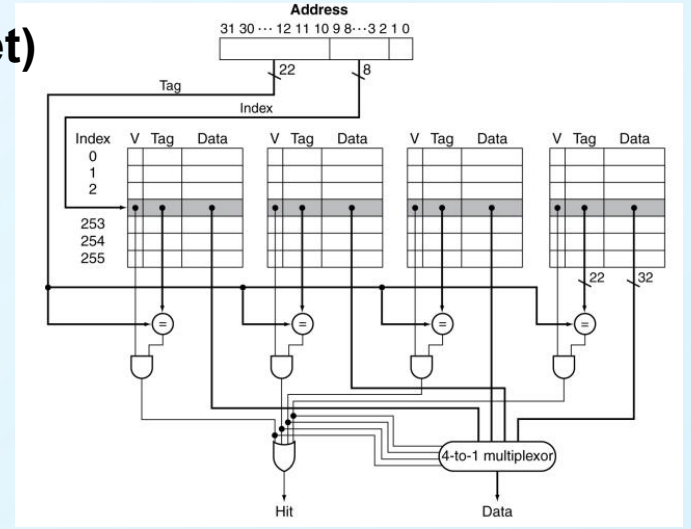
2-way set associative cache : 'Set size' is 2(There are 2 blocks in a set)

```
.data
        blk0: .word 1:32
        blk1: .word 0:32
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)

        add t2,t2,t0
        sll t2,t2,s0

        sw t2,0(t6)

        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

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 Block Table	
Cache Hit Count	0	(block 0 at top)	
Cache Miss Count	64	= empty	
Cache Hit Rate	0%	= hit	
		= miss	

N-way Set Associative Cache continued

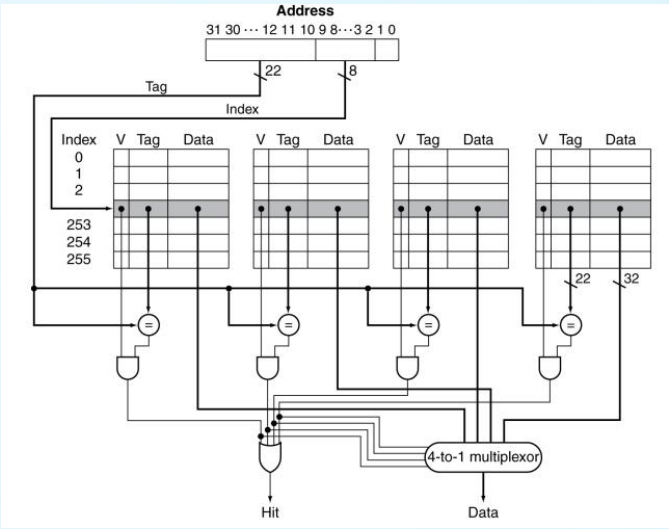
2-way set associative cache : Set size is 2(There are 2 blocks in a set)

```
.data
        blk0: .word 1:32
        blk1: .word 0:32
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)

        add t2,t2,t0
        sll t2,t2,s0

        sw t2,0(t6)

        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

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

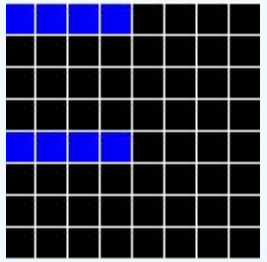
Memory Access Count	64	Cache Block Table (block 0 at top)	
Cache Hit Count	48	= empty	
Cache Miss Count	16	= hit	
Cache Hit Rate	75%	= miss	

Achive better cache performance by programming

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

Demo1

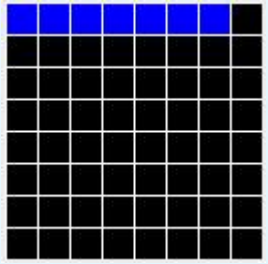
```
int a[size];
int b[size];
```



```
.data  #Demo1
        blk0: .word 0:32
        blk1: .word 0:32
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5, blk0
        la t6, blk1
loop:
        lw t2,0(t5)
        add t2,t2,t0
        srli t2,t2,31
        sw t2,0(t6)
        addi t0,t0,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

Demo2

```
struct merge{
int a;
int b;
};
struct merge marr[size];
```



```
.data  #Demo2
        mblk: .word 0:64
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t5,mblk
loop:
        lw t2,0(t5)
        add t2,t2,t0
        srli t2,t2,31
       addi t0,t0,4
       addi t5,t5,4
        sw t2,0(t5)
        addi t0,t0,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

Achive better cache performance by programming continued

```
# Demo 1 : 1/2
.data
        blk0: .word 0:32
        blk1: .word 0:32
        blk2: .word 0:32
.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32
        la t4, blk0
        la t5, blk1
loop:
        lw t2,0(t4)

        add t2,t2,t0
        srli t2,t2,31

        sw t2,0(t5)

        addi t0,t0,4
        addi t4,t4,4
        addi t5,t5,4
        addi s0,s0,1
        bne s0,t1, loop
```

```
# Demo 1 : 2/2

add t0,x0,x0
add s0,x0,x0
la t4, blk0
la t6, blk2

loop2:
        lw t2,0(t4)

        add t2,t2,t0
        srli t2,t2,31

        sw t2,0(t6)

        addi t0,t0,4
        addi t4,t4,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop2

        li a7,10
        ecall
```

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

```
                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?

```
# Demo 2 : 1/2

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

.text
        add t0,x0,x0
        add s0,x0,x0
        addi t1,x0,32

        la t4,blk0
        la t5,blk1
        la t6,blk2
```

```
# Demo 2 : 2/2

loop:
        lw t2,0(t4)

        add t2,t2,t0
        srli t2,t2,31

        sw t2,0(t5)
        sw t2,0(t6)

        addi t0,t0,4
        addi t4,t4,4
        addi t5,t5,4
        addi t6,t6,4
        addi s0,s0,1
        bne s0,t1,loop

        li a7,10
        ecall
```

Achive better cache performance by programming continued

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

for( i=0;i<2;i++ ){
        for( int j=0;j<32;j++ )
                matrix[i][j] = (i*32+j)*4;
}
```

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

```
.data      #Demo1 :  1/3
 # 32*2 word (rows: 2, columns: 32)
matrix: .space 256

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

```
 #Demo1 :  2/3
.text
addi  t0,x0,0  #i
addi  s0,x0,2

addi  t1,x0,0  #j
addi s1,x0,32
la t6, matrix
```

```
#Demo1:  3/3
loopi:
beq t0,s0,loopiend

addi  t1,x0,0

loopj:
beq t1,s1,loopjend
getindex(a0,t0,t1)
complete here
sw t5,0(t5)
addi t1,t1,1
j loopj

loopjend:
addi t0,t0,1
j loopi

loopiend:
li a7,10
ecall
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000000	0x00000004	0x00000008	0x0000000c	0x00000010	0x00000014	0x00000018	0x0000001c
0x00000020	0x00000020	0x00000024	0x00000028	0x0000002c	0x00000030	0x00000034	0x00000038	0x0000003c
0x00000040	0x00000040	0x00000044	0x00000048	0x0000004c	0x00000050	0x00000054	0x00000058	0x0000005c
0x00000060	0x00000060	0x00000064	0x00000068	0x0000006c	0x00000070	0x00000074	0x00000078	0x0000007c
0x00000080	0x00000080	0x00000084	0x00000088	0x0000008c	0x00000090	0x00000094	0x00000098	0x0000009c
0x000000a0	0x000000a0	0x000000a4	0x000000a8	0x000000ac	0x000000b0	0x000000b4	0x000000b8	0x000000bc
0x000000c0	0x000000c0	0x000000c4	0x000000c8	0x000000cc	0x000000d0	0x000000d4	0x000000d8	0x000000dc
0x000000e0	0x000000e0	0x000000e4	0x000000e8	0x000000ec	0x000000f0	0x000000f4	0x000000f8	0x000000fc
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x00000000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Achive better cache performance by programming continued

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

for( j=0;j<32;i++ ){
        for( int i=0;i<2;j++ )
                matrix[i][j] = (i*32+j)*4;
}
```

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

```
.data      #Demo2 :  1/3
 # 32*2 word (rows: 2, columns: 32)
matrix: .space 256

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

```
 #Demo2 :  2/3
.text
addi  t0,x0,0  #i
addi  s0,x0,2

addi  t1,x0,0  #j
addi s1,x0,32
la t6, matrix
```

```
#Demo2:  3/3
loopj:
beq t1,s1,loopjend

addi  t0,x0,0
loopi:
beq t0,s0,loopiend
getindex(a0,t0,t1)
complete here
sw t5, 0(t5)
addi t0,t0,1
j loopi
loopiend:
addi t1,t1,1
j loopj

loopjend:
li a7,10
ecall
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000000	0x00000004	0x00000008	0x0000000c	0x00000010	0x00000014	0x00000018	0x0000001c
0x00000020	0x00000020	0x00000024	0x00000028	0x0000002c	0x00000030	0x00000034	0x00000038	0x0000003c
0x00000040	0x00000040	0x00000044	0x00000048	0x0000004c	0x00000050	0x00000054	0x00000058	0x0000005c
0x00000060	0x00000060	0x00000064	0x00000068	0x0000006c	0x00000070	0x00000074	0x00000078	0x0000007c
0x00000080	0x00000080	0x00000084	0x00000088	0x0000008c	0x00000090	0x00000094	0x00000098	0x0000009c
0x000000a0	0x000000a0	0x000000a4	0x000000a8	0x000000ac	0x000000b0	0x000000b4	0x000000b8	0x000000bc
0x000000c0	0x000000c0	0x000000c4	0x000000c8	0x000000cc	0x000000d0	0x000000d4	0x000000d8	0x000000dc
0x000000e0	0x000000e0	0x000000e4	0x000000e8	0x000000ec	0x000000f0	0x000000f4	0x000000f8	0x000000fc
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x00000000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Practice

Q1. Complete the code for page18 and page19.

Q2. Choose any of these three questions(on page16, page17, page18/page19), answer the questoin and using the simulator('**Data Cache Simulator**' of '**Rars**') 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 Rars' cache simulation tool and verify your conclusion through simulation.

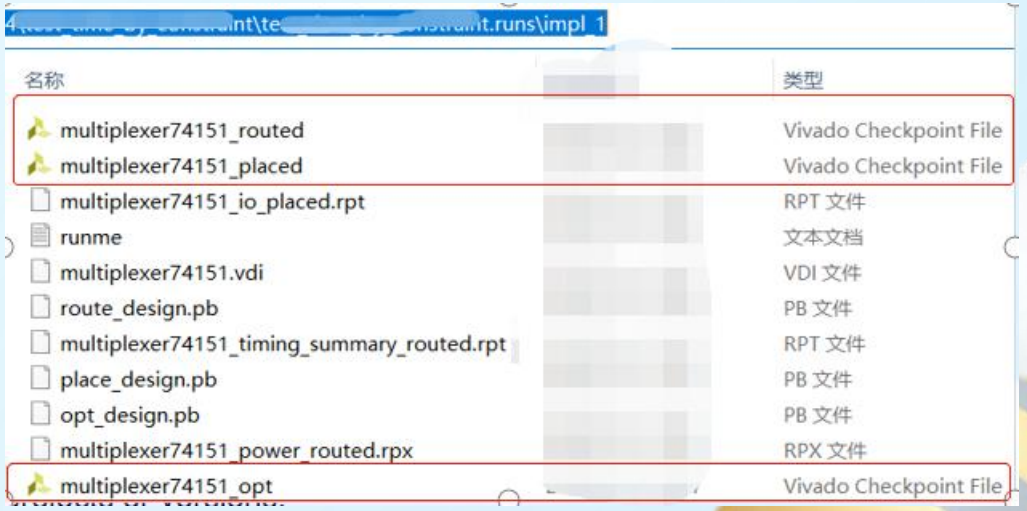
Vivado suggestion-increamental Implement

- **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.
  - "xxxx**routed.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.

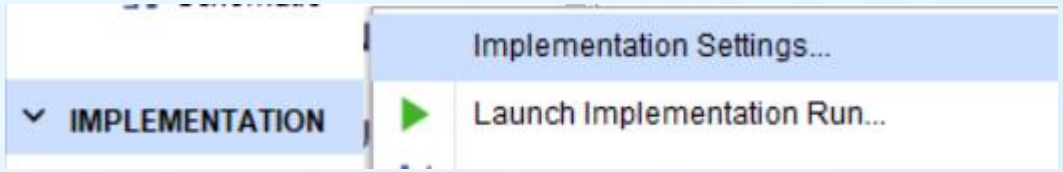


Vivado suggestion-increamental Implement continued

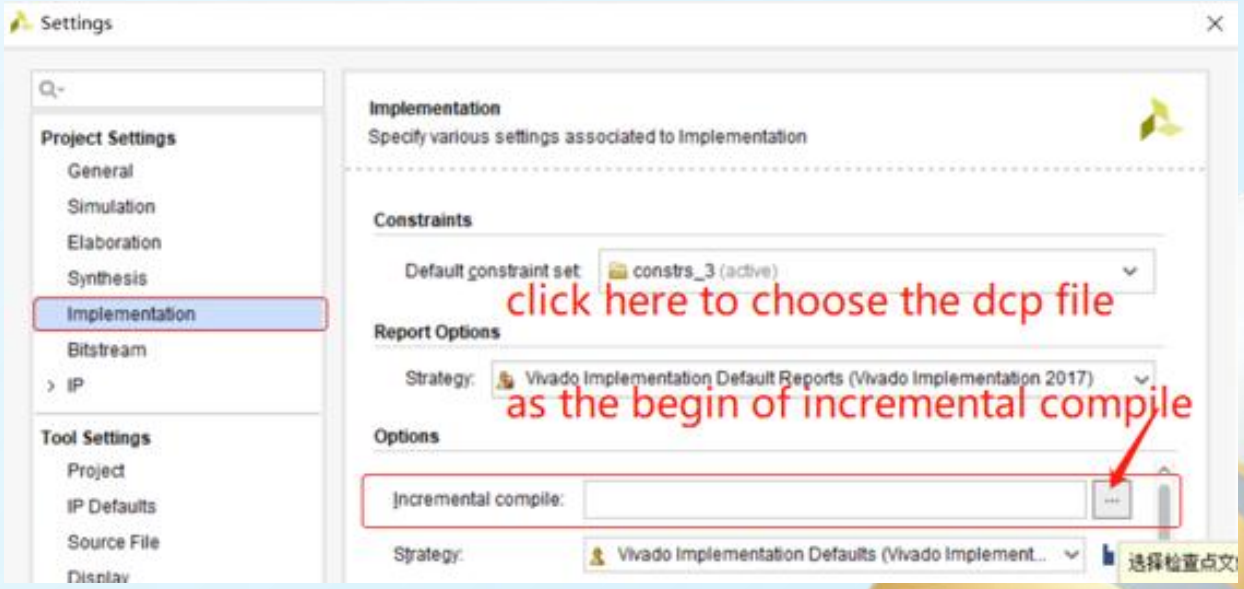
Increamental implementation steps:
1. Do the initial implementation in vivado project to generate the dcp files

2. Copy the dcp files to another new directory(such as project/increamental_compile)

3. set the implementation
  3-1: **Right click** "**implementation**" in "Flow navigator" window, click "**Implementation Settings**" to invoke the "Settings" window



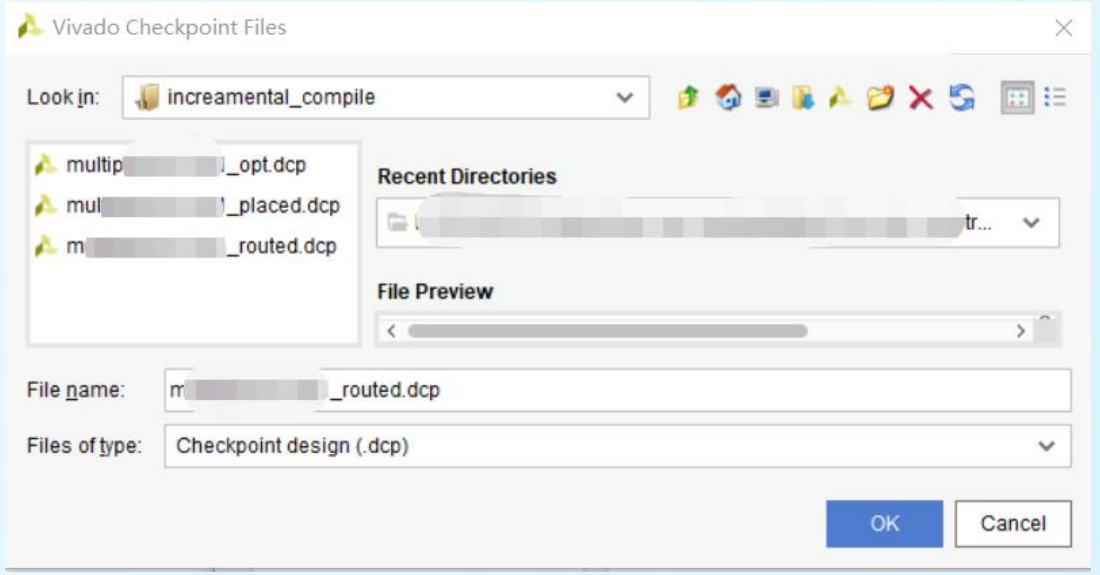
  3-2: In "**Settings**" window, click "**Implementation**", click the button on the end of "**Increamental compile**" to choose the **dcp file**(in the new directory) as the begin of incremental compile.



Vivado suggestion-increamental Implement continued

Increamental 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.**