Άσκηση 1η – Μέρος Α

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
int i, j, tmp;
int *arr, n;

for (i=0; i < n; i++){
    for (j=0; j < n-i-1; j++){
        if (arr[j] > arr[j+1]) {
            tmp      = arr[j];
            arr[j]   = arr[j+1];
            arr[j+1] = tmp;
        }
    }
}
```

```
        add $t0,$zero,$zero
I_LOOP:
        beq $t0,$s3, END
        add $t1,$zero,$zero
J_LOOP:
        sub $t2, $s3, $t0
        addi$t2, $t2, -1
        beq $t2, $t1, NEXT_I
        sll $t2, $t1, 2
        add $t2, $t2, $s1
        lw  $t3, 0($t2)
        lw  $t4, 4($t2)
        slt $t5, $t4, $t3
        beq $t5, $zero, NEXT_J
        sw  $t4, 0($t2)
        sw  $t3, 4($t2)
NEXT_J:
        addi$t1,$t1,1
        jmp J_LOOP
NEXT_I:
        addi$t0,$t0,1
        jmp I_LOOP
END:
```

Άσκηση 1η – Μέρος Β

Ζητούμενο: Υλοποιήστε τις παρακάτω ρουτίνες σε assembly του MIPS υποθέτοντας πως οι μεταβλητές x,y,z,w αποθηκεύονται σε συνεχόμενες θέσεις της μνήμης ξεκινώντας από την 0x10000000.

```
uint32_t x, y, z, w;

void xorshift_init(int _x, int _y,
                   int _z, int _w)
{
    x = _x; y = _y;
    z = _z; w = _x;
}

uint32_t xorshift()
{
    uint32_t t = x;
    t ^= t << 11;
    t ^= t >> 8;
    x = y; y = z; z = w;
    w ^= w >> 19;
    w ^= t;
    return w;
}
```

```
xorshift_init:
    lui $t0, 0x1000
    sw $a0, 0($t0)
    sw $a1, 4($t0)
    sw $a2, 8($t0)
    sw $a3, 12($t0)
    jr $ra

xorshift:
    lui $t9, 0x1000
    lw $t0, 0($t9)   # x
    lw $t1, 4($t9)   # y
    lw $t2, 8($t9)   # z
    lw $t3, 12($t9)  # w
    sll $t4, $t0, 11
    xor $t0, $t0, $t4
    srl $t4, $t0, 8
    xor $t0, $t0, $t4
    sw $t1, 0($t9)
    sw $t2, 4($t9)
    sw $t3, 8($t9)
    srl $t4, $t3, 19
    xor $t3, $t3, $t4
    xor $t3, $t3, $t0
    sw $t3, 12($t9)
    add $v0, $t3, $zero
    jr $ra
```

Άσκηση 1η – Μέρος Γ

Ζητούμενο: Οι παρακάτω C ρουτίνες υλοποιούν τον αλγόριθμο ταξινόμησης **quicksort** για ακέραιους αριθμούς 32-bit. Η quicksort χρησιμοποιεί την *xorshift()* για την παραγωγή ενός τυχαίου αριθμού μεταξύ left και right τον οποίο χρησιμοποιεί σαν pivot. Υλοποιήστε τις ρουτίνες σε assembly του MIPS.

```
void swap(int *m1, int *m2)
{
    int tmp = *m1;
    *m1 = *m2;
    *m2 = tmp;
}

int partition(int *A, int left, int right)
{
    int pivot, i, j;

    pivot = A[left];
    i = left - 1;
    j = right + 1;

    while (1) {
        while (A[++i] < pivot);
        while (A[--j] > pivot);

        if (i < j) swap(&A[i], &A[j]);
        else       return j;
    }
}
```

```
void quicksort(int *A, int left, int right)
{
    if (left >= right) return;

    int pivot = xorshift128() %
                (right - left) + left;
    swap(&A[pivot], &A[left]);

    int q = partition(A, left, right);
    quicksort(A, left, q);
    quicksort(A, q+1, right);
}
```

Άσκηση 1η – Μέρος Γ

```
void swap(int *m1, int *m2)
{
     int tmp = *m1;
     *m1 = *m2;
     *m2 = tmp;
}
```

```
swap:
     lw $t0, 0($a0)
     lw $t1, 0($a1)
     sw $t1, 0($a0)
     sw $t0, 0($a1)
     jr $ra
```

Άσκηση 1η – Μέρος Γ

```
int partition(int *A, int left, int right)
{
    int pivot, i, j;

    pivot = A[left];
    i = left - 1;
    j = right + 1;

    while (1) {
        while (A[++i] < pivot);
        while (A[--j] > pivot);

        if (i < j) swap(&A[i], &A[j]);
        else       return j;
    }
}
```

```
partition:
  addi $sp, $sp, -16
  sw $ra, 0($sp)
  sw $s0, 4($sp)
  sw $s1, 8($sp)
  sw $s2, 12($sp)
```

save registers to stack

```
  sll $t0, $a1, 2
  add $t9, $t0, $a0
  lw  $s0, 0($t9)
  addi $s1, $a1, -1
  addi $s2, $a2, 1
```

```
OUTER_LOOP:
I_LOOP:
  addi $s1, $s1, 1
  sll $t0, $s1, 2
  add $t1, $a0, $t0
  lw $t2, 0($t1)
  slt $t0, $t2, $s0
  bne $t0, $zero, I_LOOP
J_LOOP:
  addi $s2, $s2, -1
  sll $t0, $s2, 2
  add $t1, $a0, $t0
  lw $t2, 0($t1)
  slt $t0, $s0, $t2
  bne $t0, $zero, J_LOOP
```

Άσκηση 1η – Μέρος Γ

```
int partition(int *A, int left, int right)
{
     int pivot, i, j;

     pivot = A[left];
     i = left - 1;
     j = right + 1;

     while (1) {
          while (A[++i] < pivot);
          while (A[--j] > pivot);

          if (i < j) swap(&A[i], &A[j]);
          else       return j;
     }
}
```

```
slt $t0, $s1, $s2
beq $t0, $zero, PARTITION_RETURN
```

```
addi $sp, $sp, -8
sw $a0, 0($sp)
sw $a1, 4($sp)
sll $t0, $s2, 2
add $a1, $a0, $t0
sll $t0, $s1, 2
add $a0, $a0, $t0
jal swap
lw $a1, 4($sp)
lw $a0, 0($sp)
addi $sp, $sp, 8
j OUTER_LOOP
```

```
PARTITION_RETURN:
  add $v0, $s2, $zero
  lw $ra, 0($sp)
  lw $s0, 4($sp)
  lw $s1, 8($sp)
  lw $s2, 12($sp)
  addi $sp, $sp, 16
  jr $ra
```

restore saved registers from stack

Άσκηση 1η – Μέρος Γ

```
void quicksort(int *A, int left, int right)
{
    if (left >= right) return;

    int pivot = xorshift128() %
                (right - left) + left;
    swap(&A[pivot], &A[left]);

    int q = partition(A, left, right);
    quicksort(A, left, q);
    quicksort(A, q+1, right);
}
```

```
quicksort:
  addi $sp, $sp, -8
  sw $ra, 0($sp)
  sw $s0, 4($sp)

  slt $t0, $a1, $a2
  beq $t0, $zero, QUICKSORT_RETURN

  jal xorshift
  sub $s0, $a2, $a1
  divu $v0, $s0
  mfhi $s0
  add $s0, $s0, $a1

  addi $sp, $sp, -8
  sw $a0, 0($sp)
  sw $a1, 4($sp)
  sll $a1, $a1, 2
  add $a1, $a0, $a1
  sll $t0, $s0, 2
  add $a0, $a0, $t0
  jal swap
  lw $a1, 4($sp)
  lw $a0, 0($sp)
  addi $sp, $sp, 8

  jal partition
```

Άσκηση 1η – Μέρος Γ

```
void quicksort(int *A, int left, int right)
{
     if (left >= right) return;

     int pivot = xorshift128() %
                 (right - left) + left;
     swap(&A[pivot], &A[left]);

     int q = partition(A, left, right);
     quicksort(A, left, q);
     quicksort(A, q+1, right);
}
```

```
addi $sp, $sp, -8
sw $a2, 0($sp)
sw $v0, 4($sp)
add $a2, $zero, $v0
jal quicksort
lw $a2, 0($sp)
lw $v0, 4($sp)
addi $sp, $sp, 8
```

```
add $a1, $zero, $v0
addi $a1, $a1, 1
jal quicksort
```

```
QUICKSORT_RETURN:
  lw $ra, 0($sp)
  lw $s0, 4($sp)
  addi $sp, $sp, 8
  jr $ra
```

2η Άσκηση

Δεδομένα

Έχουμε ένα loop...

```
LOOP: LW    $t0,  0($t3)
      LW    $t1,  0($t0)
      LW    $t2,  8($t0)
      ADD   $t2,  $t2,  $t1
      ADD   $t2,  $t2,  $t0
      SW    $t2,  0($t3)
      ADDI  $t3,  $t3, -4
      BNE   $t9,  $t3,  LOOP
```

2η Άσκηση

Δεδομένα

Έχουμε ένα loop...

```
LOOP: LW    $t0,  0($t3)
      LW    $t1,  0($t0)
      LW    $t2,  8($t0)
      ADD   $t2,  $t2,  $t1
      ADD   $t2,  $t2,  $t0
      SW    $t2,  0($t3)
      ADDI  $t3,  $t3, -4
      BNE   $t9,  $t3,  LOOP
```

και αυτή την αρχική κατάσταση

$t3 = 0x1000 = 4096
$t9 = 0x800 = 2048

- Δεν υπάρχει cache miss
- Cache hit σε 1cc
- branches γίνονται resolve στο EX stage

2η Άσκηση

```
LOOP: LW    $t0,  0($t3)
      LW    $t1,  0($t0)
      LW    $t2,  8($t0)
      ADD   $t2,  $t2,  $t1
      ADD   $t2,  $t2,  $t0
      SW    $t2,  0($t3)
      ADDI  $t3,  $t3, -4
      BNE   $t9,  $t3,  LOOP
```

```
$t3 = 4096
$t9 = 2048
```

$t3 = 4096, 4092, …, 2048

Ο βρόχος θα εκτελεστεί 2048 / 4 = 512 φορές.

2η Άσκηση – 1ο ζητούμενο

1ο Ζητούμενο : Για το 1ο LOOP (μέχρι και το lw του 2ου LOOP)

Να δείξετε τα **διάφορα στάδια του pipeline** (διάγραμμα χρονισμού) που περνάνε οι εντολές. Υποθέστε ότι η αρχιτεκτονική δε διαθέτει σχήμα προώθησης.

Κύκλος	1	2	3	4	5	6	7
Εντολή 1	IF	ID	EX	MEM	WB		
Εντολή 2		IF	ID	EX	MEM	WB	
Εντολή 3			…	…	…	…	…
…							

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F																					
`LW $t1, 0($t0)`																						
`LW $t2, 8($t0)`																						
`ADD $t2,$t2,$t1`																						
`ADD $t2,$t2,$t0`																						
`SW $t2,0($t3)`																						
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D																				
`LW $t1, 0($t0)`		F																				
`LW $t2, 8($t0)`																						
`ADD $t2,$t2,$t1`																						
`ADD $t2,$t2,$t0`																						
`SW $t2,0($t3)`																						
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X																			
`LW $t1, 0($t0)`		F	D																			
`LW $t2, 8($t0)`			F																			
`ADD $t2,$t2,$t1`																						
`ADD $t2,$t2,$t0`																						
`SW $t2,0($t3)`																						
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LW $t0, 0($t3)	F	D	X	M																		
LW $t1, 0($t0)		F	D	-																		
LW $t2, 8($t0)			F	-																		
ADD $t2,$t2,$t1																						
ADD $t2,$t2,$t0																						
SW $t2,0($t3)																						
ADDI $t3,$t3,-4																						
BNE $t9,$t3,LOOP																						
LW $t0, 0($t3)																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-																	
`LW $t2, 8($t0)`			F	-	-																	
`ADD $t2,$t2,$t1`																						
`ADD $t2,$t2,$t0`																						
`SW $t2,0($t3)`																						
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X																
`LW $t2, 8($t0)`			F	-	-	D																
`ADD $t2,$t2,$t1`						F																
`ADD $t2,$t2,$t0`																						
`SW $t2,0($t3)`																						
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M															
`LW $t2, 8($t0)`			F	-	-	D	X															
`ADD $t2,$t2,$t1`						F	D															
`ADD $t2,$t2,$t0`							F															
`SW $t2,0($t3)`																						
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LW $t0, 0($t3)	F	D	X	M	W																	
LW $t1, 0($t0)		F	D	-	-	X	M	W														
LW $t2, 8($t0)			F	-	-	D	X	M														
ADD $t2,$t2,$t1						F	D	-														
ADD $t2,$t2,$t0							F	-														
SW $t2,0($t3)																						
ADDI $t3,$t3,-4																						
BNE $t9,$t3,LOOP																						
LW $t0, 0($t3)																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M	W														
`LW $t2, 8($t0)`			F	-	-	D	X	M	W													
`ADD $t2,$t2,$t1`						F	D	-	-													
`ADD $t2,$t2,$t0`							F	-	-													
`SW $t2,0($t3)`																						
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M	W														
`LW $t2, 8($t0)`			F	-	-	D	X	M	W													
`ADD $t2,$t2,$t1`						F	D	-	-	X												
`ADD $t2,$t2,$t0`							F	-	-	D												
`SW $t2,0($t3)`										F												
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LW $t0, 0($t3)	F	D	X	M	W																	
LW $t1, 0($t0)		F	D	-	-	X	M	W														
LW $t2, 8($t0)			F	-	-	D	X	M	W													
ADD $t2,$t2,$t1						F	D	-	-	X	M											
ADD $t2,$t2,$t0							F	-	-	D	-											
SW $t2,0($t3)										F	-											
ADDI $t3,$t3,-4																						
BNE $t9,$t3,LOOP																						
LW $t0, 0($t3)																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M	W														
`LW $t2, 8($t0)`			F	-	-	D	X	M	W													
`ADD $t2,$t2,$t1`						F	D	-	-	X	M	W										
`ADD $t2,$t2,$t0`							F	-	-	D	-	-										
`SW $t2,0($t3)`										F	-	-										
`ADDI $t3,$t3,-4`																						
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M	W														
`LW $t2, 8($t0)`			F	-	-	D	X	M	W													
`ADD $t2,$t2,$t1`						F	D	-	-	X	M	W										
`ADD $t2,$t2,$t0`							F	-	-	D	-	-	X									
`SW $t2,0($t3)`										F	-	-	D									
`ADDI $t3,$t3,-4`													F									
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LW $t0, 0($t3)	F	D	X	M	W																	
LW $t1, 0($t0)		F	D	-	-	X	M	W														
LW $t2, 8($t0)			F	-	-	D	X	M	W													
ADD $t2,$t2,$t1						F	D	-	-	X	M	W										
ADD $t2,$t2,$t0							F	-	-	D	-	-	X	M								
SW $t2,0($t3)										F	-	-	D	-								
ADDI $t3,$t3,-4													F	-								
BNE $t9,$t3,LOOP																						
LW $t0, 0($t3)																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M	W														
`LW $t2, 8($t0)`			F	-	-	D	X	M	W													
`ADD $t2,$t2,$t1`						F	D	-	-	X	M	W										
`ADD $t2,$t2,$t0`							F	-	-	D	-	-	X	M	W							
`SW $t2,0($t3)`										F	-	-	D	-	-							
`ADDI $t3,$t3,-4`													F	-	-							
`BNE $t9,$t3,LOOP`																						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M	W														
`LW $t2, 8($t0)`			F	-	-	D	X	M	W													
`ADD $t2,$t2,$t1`						F	D	-	-	X	M	W										
`ADD $t2,$t2,$t0`							F	-	-	D	-	-	X	M	W							
`SW $t2,0($t3)`										F	-	-	D	-	-	X						
`ADDI $t3,$t3,-4`													F	-	-	D						
`BNE $t9,$t3,LOOP`																F						
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M	W														
`LW $t2, 8($t0)`			F	-	-	D	X	M	W													
`ADD $t2,$t2,$t1`						F	D	-	-	X	M	W										
`ADD $t2,$t2,$t0`							F	-	-	D	-	-	X	M	W							
`SW $t2,0($t3)`										F	-	-	D	-	-	X	M					
`ADDI $t3,$t3,-4`													F	-	-	D	X					
`BNE $t9,$t3,LOOP`																F	D					
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LW $t0, 0($t3)	F	D	X	M	W																	
LW $t1, 0($t0)		F	D	-	-	X	M	W														
LW $t2, 8($t0)			F	-	-	D	X	M	W													
ADD $t2,$t2,$t1						F	D	-	-	X	M	W										
ADD $t2,$t2,$t0							F	-	-	D	-	-	X	M	W							
SW $t2,0($t3)										F	-	-	D	-	-	X	M	W				
ADDI $t3,$t3,-4													F	-	-	D	X	M				
BNE $t0,$t3,LOOP																F	D	-				
LW $t0, 0($t3)																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`LW $t0, 0($t3)`	F	D	X	M	W																	
`LW $t1, 0($t0)`		F	D	-	-	X	M	W														
`LW $t2, 8($t0)`			F	-	-	D	X	M	W													
`ADD $t2,$t2,$t1`						F	D	-	-	X	M	W										
`ADD $t2,$t2,$t0`							F	-	-	D	-	-	X	M	W							
`SW $t2,0($t3)`										F	-	-	D	-	-	X	M	W				
`ADDI $t3,$t3,-4`													F	-	-	D	X	M	W			
`BNE $t9,$t3,LOOP`																F	D	-	-			
`LW $t0, 0($t3)`																						

2η Άσκηση – 1ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LW $t0, 0($t3)	F	D	X	M	W																	
LW $t1, 0($t0)		F	D	-	-	X	M	W														
LW $t2, 8($t0)			F	-	-	D	X	M	W													
ADD $t2,$t2,$t1						F	D	-	-	X	M	W										
ADD $t2,$t2,$t0							F	-	-	D	-	-	X	M	W							
SW $t2,0($t3)										F	-	-	D	-	-	X	M	W				
ADDI $t3,$t3,-4													F	-	-	D	X	M	W			
BNE $t9,$t3,LOOP																F	D	-	-	X	M	W
LW $t0, 0($t3)																					F	D

Σύνολο κύκλων: 20*511 + 22 = 10242

2η Άσκηση – 2ο ζητούμενο

2ο Ζητούμενο : Για το 1ο LOOP (μέχρι και το lw του 2ου LOOP)

Να δείξετε τα **διάφορα στάδια του pipeline** (διάγραμμα χρονισμού) που περνάνε οι εντολές. Υποθέστε τώρα ότι η αρχιτεκτονική **διαθέτει σχήμα προώθησης**.

Κύκλος	1	2	3	4	5	6	7
Εντολή 1	IF	ID	EX	MEM	WB		
Εντολή 2		IF	ID	EX	MEM	WB	
Εντολή 3			…	…	…	…	…
…							

2η Άσκηση – 2ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW $t0, 0($t3)	F	D	X	M										
LW $t1, 0($t0)		F	D	-										
LW $t2, 8($t0)			F	-										
ADD $t2,$t2,$t1														
ADD $t2,$t2,$t0														
SW $t2,0($t3)														
ADDI $t3,$t3,-4														
BNE $t9,$t3,LOOP														
LW $t0, 0($t3)														

2η Άσκηση – 2ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW $t0, 0($t3)	F	D	X	M	W									
LW $t1, 0($t0)		F	D	-	X									
LW $t2, 8($t0)			F	-	D									
ADD $t2,$t2,$t1					F									
ADD $t2,$t2,$t0														
SW $t2,0($t3)														
ADDI $t3,$t3,-4														
BNE $t9,$t3,LOOP														
LW $t0, 0($t3)														

2η Άσκηση – 2ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW $t0, 0($t3)	F	D	X	M	W									
LW $t1, 0($t0)		F	D	-	X	M	W							
LW $t2, 8($t0)			F	-	D	X	M							
ADD $t2,$t2,$t1					F	D	-							
ADD $t2,$t2,$t0						F	-							
SW $t2,0($t3)														
ADDI $t3,$t3,-4														
BNE $t9,$t3,LOOP														
LW $t0, 0($t3)														

2η Άσκηση – 2ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`LW $t0, 0($t3)`	F	D	X	M	W									
`LW $t1, 0($t0)`		F	D	-	X	M	W							
`LW $t2, 8($t0)`			F	-	D	X	M	W						
`ADD $t2,$t2,$t1`					F	D	-	X						
`ADD $t2,$t2,$t0`						F	-	D						
`SW $t2,0($t3)`								F						
`ADDI $t3,$t3,-4`														
`BNE $t9,$t3,LOOP`														
`LW $t0, 0($t3)`														

2η Άσκηση – 2ο ζητούμενο

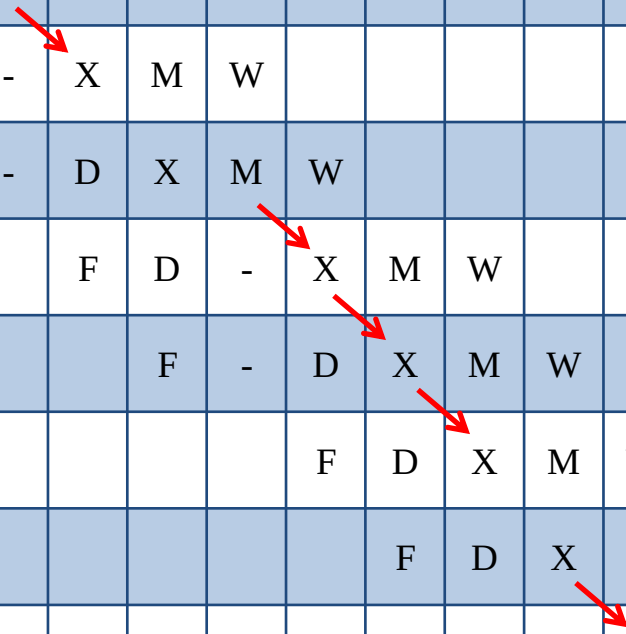
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW $t0, 0($t3)	F	D	X	M	W									
LW $t1, 0($t0)		F	D	-	X	M	W							
LW $t2, 8($t0)			F	-	D	X	M	W						
ADD $t2,$t2,$t1					F	D	-	X	M					
ADD $t2,$t2,$t0						F	-	D	X					
SW $t2,0($t3)								F	D					
ADDI $t3,$t3,-4									F					
BNE $t9,$t3,LOOP														
LW $t0, 0($t3)														

2η Άσκηση – 2ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`LW $t0, 0($t3)`	F	D	X	M	W									
`LW $t1, 0($t0)`		F	D	-	X	M	W							
`LW $t2, 8($t0)`			F	-	D	X	M	W						
`ADD $t2,$t2,$t1`					F	D	-	X	M	W				
`ADD $t2,$t2,$t0`						F	-	D	X	M				
`SW $t2,0($t3)`								F	D	X				
`ADDI $t3,$t3,-4`									F	D				
`BNE $t9,$t3,LOOP`										F				
`LW $t0, 0($t3)`														

2η Άσκηση – 2ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW $t0, 0($t3)	F	D	X	M	W									
LW $t1, 0($t0)		F	D	-	X	M	W							
LW $t2, 8($t0)			F	-	D	X	M	W						
ADD $t2,$t2,$t1					F	D	-	X	M	W				
ADD $t2,$t2,$t0						F	-	D	X	M	W			
SW $t2,0($t3)								F	D	X	M	W		
ADDI $t3,$t3,-4									F	D	X	M		
BNE $t9,$t3,LOOP										F	D	X		
LW $t0, 0($t3)														



2η Άσκηση – 2ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW $t0, 0($t3)	F	D	X	M	W									
LW $t1, 0($t0)		F	D	-	X	M	W							
LW $t2, 8($t0)			F	-	D	X	M	W						
ADD $t2,$t2,$t1					F	D	-	X	M	W				
ADD $t2,$t2,$t0						F	-	D	X	M	W			
SW $t2,0($t3)								F	D	X	M	W		
ADDI $t3,$t3,-4									F	D	X	M	W	
BNE $t9,$t3,LOOP										F	D	X	M	W
LW $t0, 0($t3)													F	D

Σύνολο κύκλων: 12*511 + 14 = 6146

2η Άσκηση – 3ο ζητούμενο

3ο Ζητούμενο : Για το 1ο LOOP (μέχρι και το lw του 2ου LOOP)

Προσπαθήστε να πετύχετε καλύτερη απόδοση τροποποιώντας τον κώδικα, χωρίς όμως να αλλάξετε τη σημασιολογία του προγράμματος.

Κύκλος	1	2	3	4	5	6	7
Εντολή 1	IF	ID	EX	MEM	WB		
Εντολή 2		IF	ID	EX	MEM	WB	
Εντολή 3			…	…	…	…	…
…							

2η Άσκηση – 2ο ζητούμενο (επανάληψη)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW $t0, 0($t3)	F	D	X	M	W									
LW $t1, 0($t0)		F	D	-	X	M	W							
LW $t2, 8($t0)			F	-	D	X	M	W						
ADD $t2,$t2,$t1					F	D	-	X	M	W				
ADD $t2,$t2,$t0						F	-	D	X	M	W			
SW $t2,0($t3)								F	D	X	M	W		
ADDI $t3,$t3,-4									F	D	X	M	W	
BNE $t9,$t3,LOOP														W
LW $t0, 0($t3)													F	D

Σύνολο κύκλων: 12*511 + 14 = 6146

2η Άσκηση – 3ο ζητούμενο

	1	2	3	4	5	6	7	8	9	10	11	12
LW $t0, 0($t3)	F	D	X	M	W							
ADDI $t3,$t3,-4		F	D	X	M	W						
LW $t2, 8($t0)			F	D	X	M	W					
LW $t1, 0($t0)				F	D	X	M	W				
ADD $t2,$t2,$t0					F	D	X	M	W			
ADD $t2,$t2,$t1						F	D	X	M	W		
SW $t2,4($t3)							F	D	X	M	W	
BNE $t9,$t3,LOOP								F	D	X	M	W
LW $t0, 0($t3)											F	D

Σύνολο κύκλων: 10*511 + 12 = 5122

Άσκηση 3η – Μέρος Α

Δίνεται η παρακάτω ακολουθία προσπελάσεων

Διεύθυνση (hex)	Αποτέλεσμα
0x044	Miss
0x042	Hit
0x048	Miss
0x1AF	Miss
0x04A	Miss

- Direct-mapped cache
- Ελάχιστη μονάδα δεδομένων που μπορεί να διευθυνσιοδοτηθεί το 1 byte
- Μήκος διεύθυνσης 9 bits
- Αρχικά η cache είναι άδεια

Υπολογίστε το συνολικό μέγεθος της cache καθώς και το μέγεθος του tag array.

Άσκηση 3η – Μέρος Α

Υπολογίστε το συνολικό μέγεθος της cache καθώς και το μέγεθος του tag array.

Διεύθυνση (hex)	Διεύθυνση (bin)	Αποτέλεσμα
0x044	0 0100 0100	Miss
0x042	0 0100 0010	Hit
0x048	0 0100 1000	Miss
0x1AF	1 1010 1111	Miss
0x04A	0 0100 1010	Miss

Στο ίδιο block → block offset ≥ 3

Σε διαφορετικό block → 1 ≤ block offset ≤ 3

block offset = 3 bits
block size = 8 bytes

Για block size = 8 bytes είναι στο ίδιο block άρα το 0x04A θα έπρεπε να είναι hit → το 0x1AF πρέπει να έχει ίδιο index → index = 1 ή 2 bits

cache size = 2 * 8 = **16 bytes** και tag array = 2 * 5 = **10 bits**
ή cache size = 4 * 8 = **32 bytes** και tag array = 4 * 4 = **16 bits**

Άσκηση 3η – Μέρος Β

Δίνεται το ακόλουθο κομμάτι κώδικα:

```
int i,j;
double A[8][8], B[8][8], C[64];

for (i=0; i < 8; i++)
  for (j=0; j < 8; j++)
    if (j % 2 == 0)
      A[i][j] = A[i][j] + B[i][j] + C[8*i +j];
    else
      A[i][j] = A[i][j] + B[i][j];
```

- ✓ *Κάθε στοιχείο του πίνακα είναι 8 bytes*
- ✓ *1 επίπεδο κρυφής μνήμης, direct-mapped, write-allocate, 32B block, μεγέθους 512 bytes*
- ✓ *Αρχικά η cache είναι άδεια*
- ✓ *Όλες οι μεταβλητές αποθηκεύονται σε καταχωρητές εκτός από τα στοιχεία των πινάκων*
- ✓ *Οι πίνακες αποθηκεύονται στη μνήμη κατά γραμμές και είναι ευθυγραμμισμένοι*
- ✓ *ο Α στη θέση μνήμης 0x00080000.*
- ✓ *Η σειρά με την οποία γίνονται οι αναφορές στην μνήμη είναι Α, Β, {C,} Α*

Άσκηση 3η – Μέρος Β (i)

1ο Ζητούμενο: Βρείτε το συνολικό αριθμό hits και misses για όλη την εκτέλεση του παραπάνω κώδικα. Υποδείξτε τον τύπο των misses.

- 1 block = 32 bytes
- 1 στοιχείο = 8 bytes
- πίνακας αποθηκευμένος κατά γραμμές

→ σε 1 block της cache θα απεικονίζονται 4 διαδοχικά στοιχεία του πίνακα, π.χ. A[i][j],A[i][j+1], A[i][j+2], A[i][j+3]

32 bytes block size = 2^5 → 5 bits offset
512B cache / 32B block → 16 blocks
16 blocks → 16 sets = 2^4 → 4 bits index

όλα στο set 0

A[0][0] → 0x0008 0000 = 0000 0000 0000 0001 0000 0000 0000 0000
B[0][0] → 0x0008 0200 = 0000 0000 0000 0001 0000 0010 0000 0000
C[0][0] → 0x0008 0400 = 0000 0000 0000 0001 0000 0100 0000 0000

A[0][0], B[0][0] και C[0] γίνονται mapped στο ίδιο block

Άσκηση 3η – Μέρος Β (i)

A[0][0] compulsory miss

i=0, j=0

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος B (i)

i=0, j=0

A[0][0] compulsory miss

B[0][0] compulsory miss

0	B[0][0]	B[0][1]	B[0][2]	B[0][3]
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος B (i)

i=0, j=0

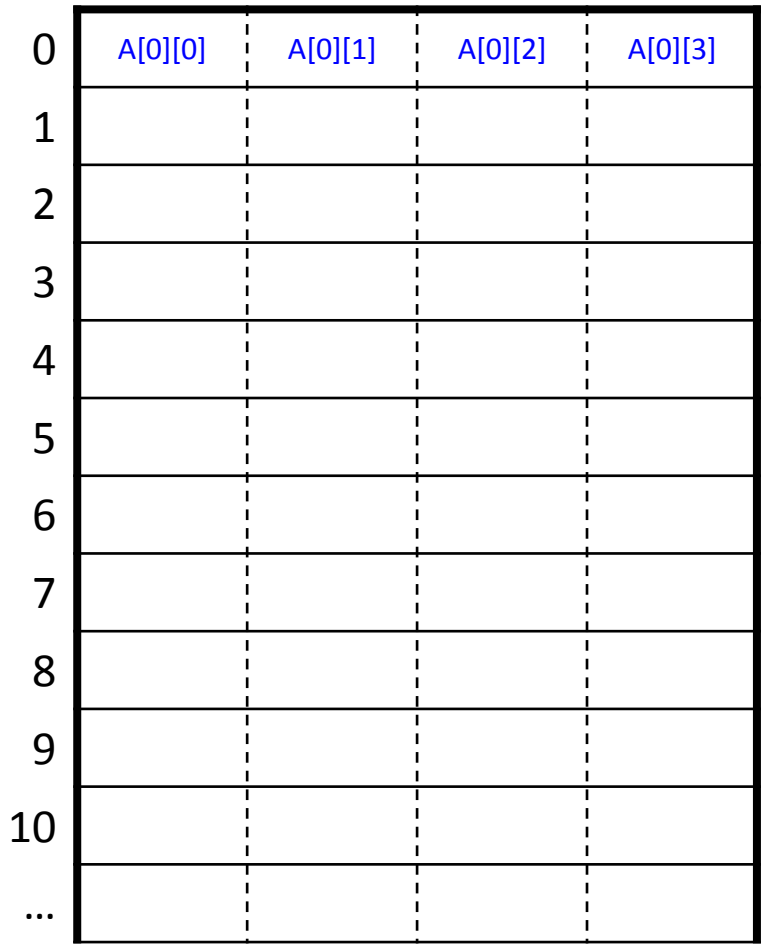
A[0][0]	compulsory miss
B[0][0]	compulsory miss
C[0]	compulsory miss

	C[0]	C[1]	C[2]	C[3]
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος B (i)

i=0, j=0

A[0][0]	compulsory miss
B[0][0]	compulsory miss
C[0]	compulsory miss
A[0][0]	conflict miss



Άσκηση 3η – Μέρος B (i)

i=0, j=0

A[0][0] compulsory miss

B[0][0] compulsory miss

C[0] compulsory miss

A[0][0] conflict miss

A[0][1] hit

i=0, j=1

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				

Άσκηση 3η – Μέρος B (i)

	A[0][0]	compulsory miss
i=0, j=0	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
	A[0][1]	hit
i=0, j=1	B[0][1]	conflict miss

0	B[0][0]	B[0][1]	B[0][2]	B[0][3]
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

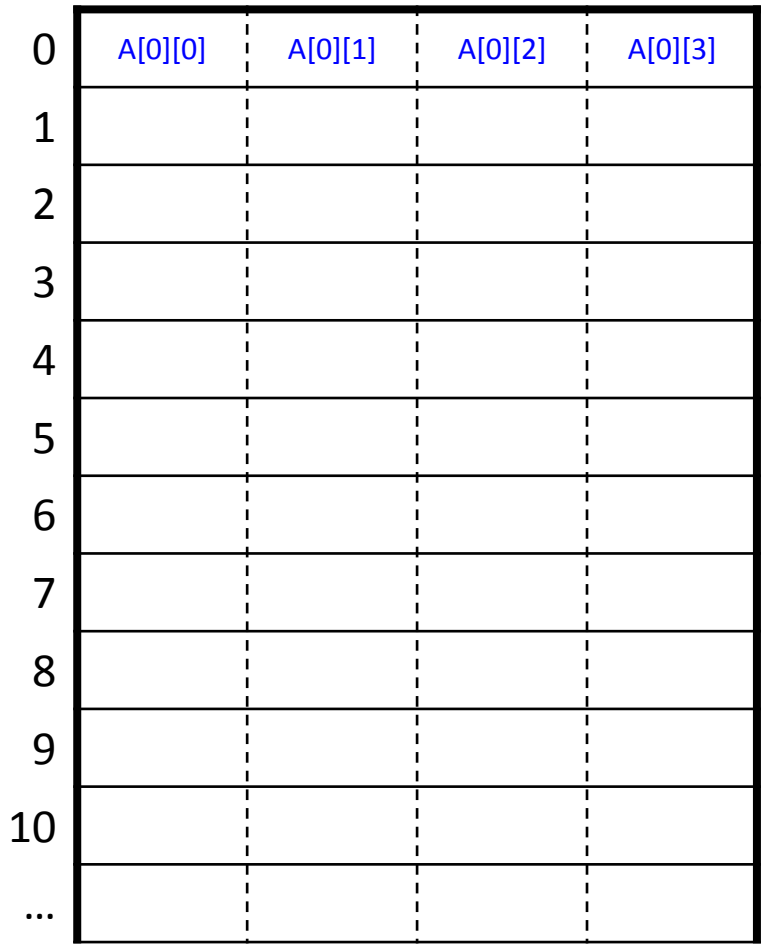
Άσκηση 3η – Μέρος Β (i)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

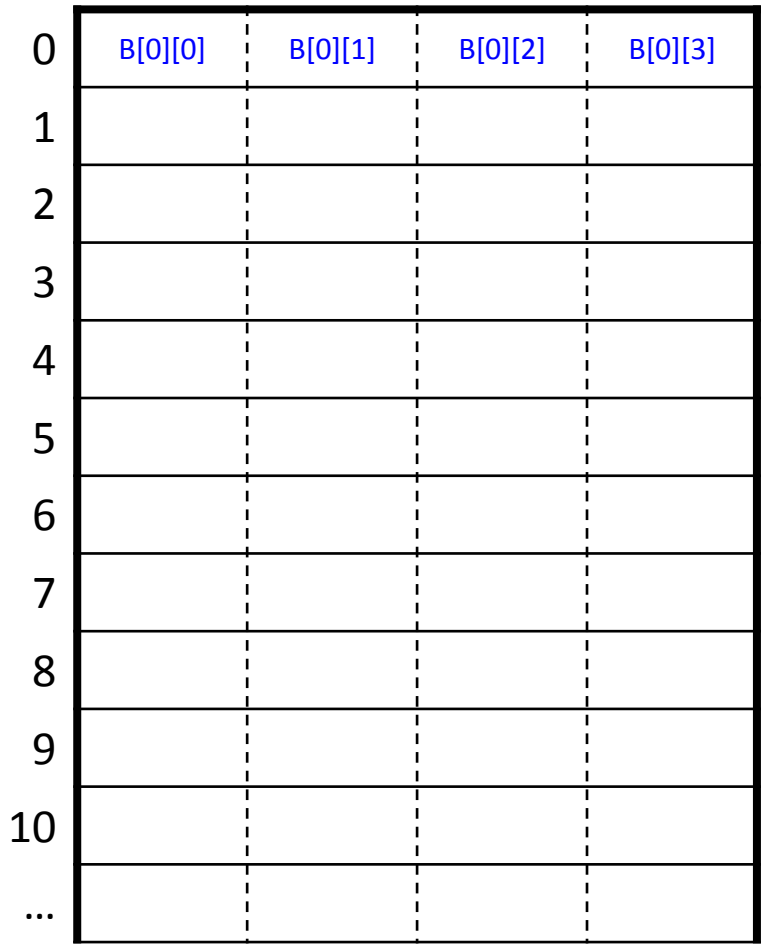
Άσκηση 3η – Μέρος Β (i)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	conflict miss
i=0, j=2	A[0][2]	hit



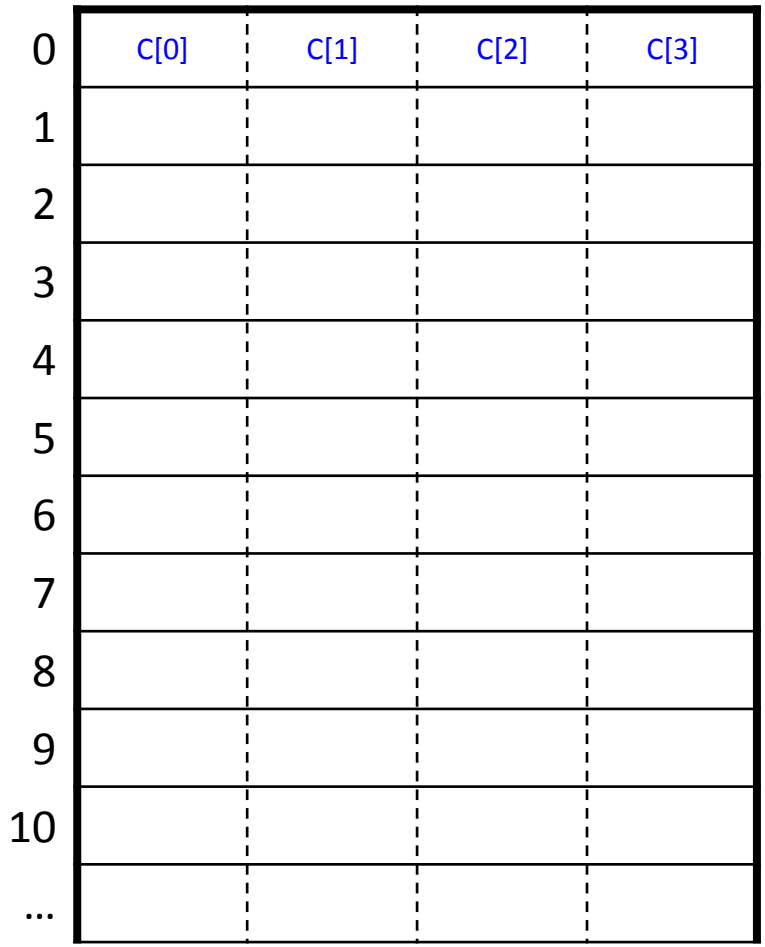
Άσκηση 3η – Μέρος B (i)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	conflict miss
i=0, j=2	A[0][2]	hit
	B[0][2]	conflict miss



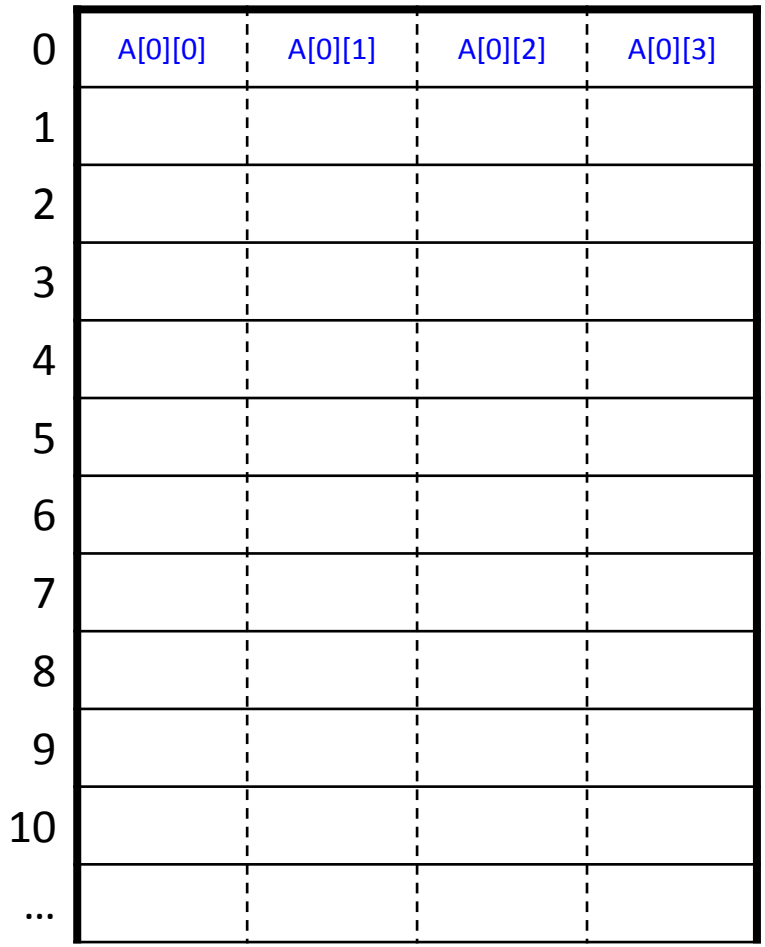
Άσκηση 3η – Μέρος Β (i)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	conflict miss
i=0, j=2	A[0][2]	hit
	B[0][2]	conflict miss
	C[2]	conflict miss



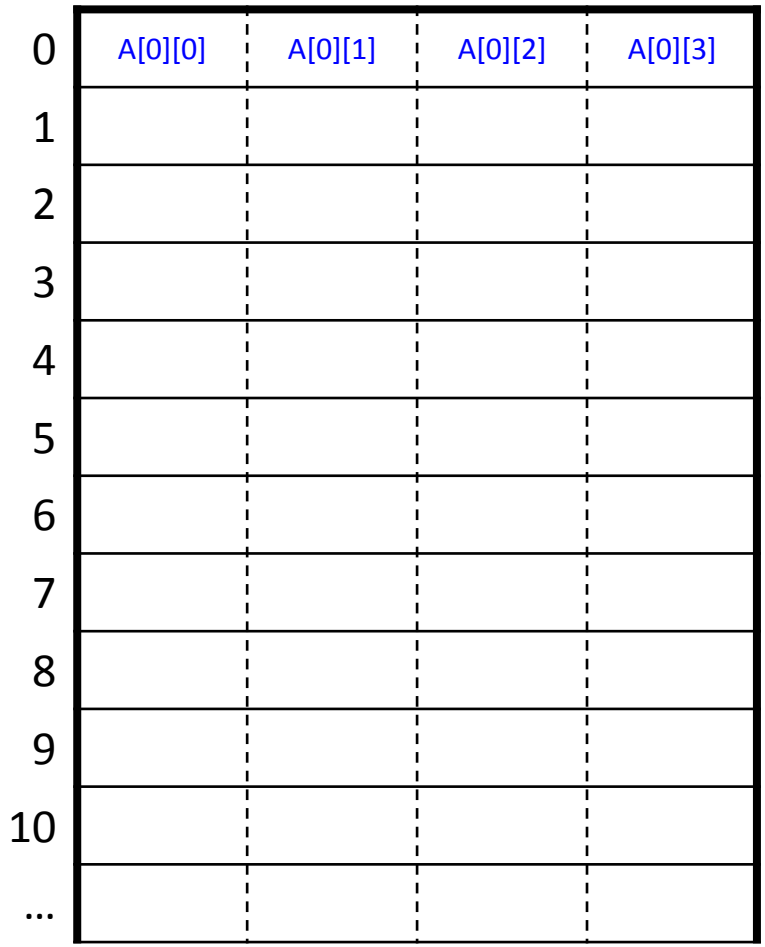
Άσκηση 3η – Μέρος Β (i)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	conflict miss
i=0, j=2	A[0][2]	hit
	B[0][2]	conflict miss
	C[2]	conflict miss
	A[0][2]	conflict miss



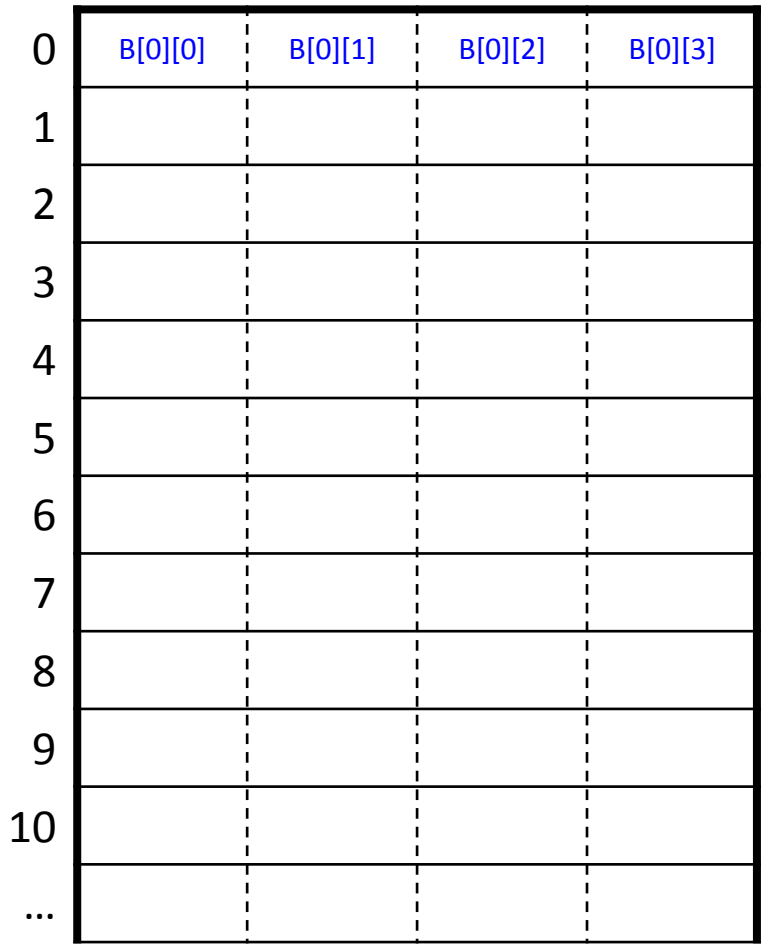
Άσκηση 3η – Μέρος Β (i)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	conflict miss
i=0, j=2	A[0][2]	hit
	B[0][2]	conflict miss
	C[2]	conflict miss
	A[0][2]	conflict miss
i=0, j=3	A[0][3]	hit



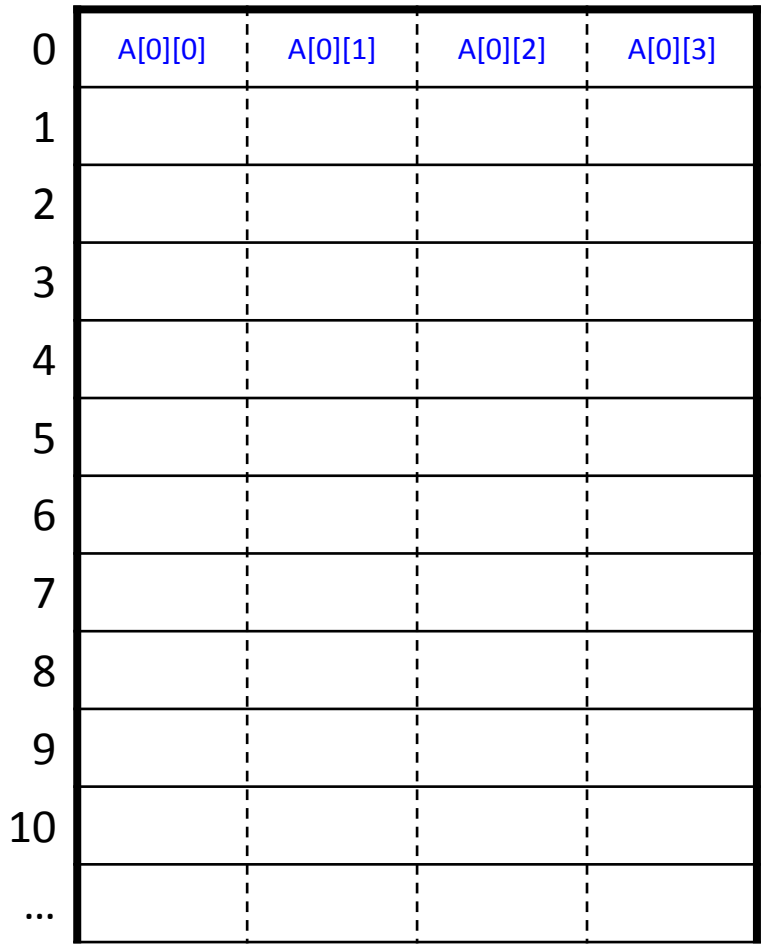
Άσκηση 3η – Μέρος B (i)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	conflict miss
i=0, j=2	A[0][2]	hit
	B[0][2]	conflict miss
	C[2]	conflict miss
	A[0][2]	conflict miss
i=0, j=3	A[0][3]	hit
	B[0][3]	conflict miss



Άσκηση 3η – Μέρος Β (i)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	conflict miss
i=0, j=2	A[0][2]	hit
	B[0][2]	conflict miss
	C[2]	conflict miss
	A[0][2]	conflict miss
i=0, j=3	A[0][3]	hit
	B[0][3]	conflict miss
	A[0][3]	conflict miss



Άσκηση 3η – Μέρος Β (i)

A[0][4] compulsory miss

i=0, j=4

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[0][4]	A[0][5]	A[0][6]	A[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος B (i)

i=0, j=4

A[0][4] compulsory miss

B[0][4] compulsory miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	B[0][4]	B[0][5]	B[0][6]	B[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος B (i)

i=0, j=4

A[0][4]	compulsory miss
B[0][4]	compulsory miss
C[4]	compulsory miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	C[4]	C[5]	C[6]	C[7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος B (i)

i=0, j=4

A[0][4]	compulsory miss
B[0][4]	compulsory miss
C[4]	compulsory miss
A[0][4]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[0][4]	A[0][5]	A[0][6]	A[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος B (i)

i=0, j=4	A[0][4]	compulsory miss
	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
	A[0][5]	hit
i=0, j=5		

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[0][4]	A[0][5]	A[0][6]	A[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

	A[0][4]	compulsory miss
i=0, j=4	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
	A[0][5]	hit
i=0, j=5	B[0][5]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	B[0][4]	B[0][5]	B[0][6]	B[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

	A[0][4]	compulsory miss
i=0, j=4	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
	A[0][5]	hit
i=0, j=5	B[0][5]	conflict miss
	A[0][5]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[0][4]	A[0][5]	A[0][6]	A[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

	A[0][4]	compulsory miss
i=0, j=4	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
	A[0][5]	hit
i=0, j=5	B[0][5]	conflict miss
	A[0][5]	conflict miss
	A[0][6]	hit
i=0, j=6		

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[0][4]	A[0][5]	A[0][6]	A[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

	A[0][4]	compulsory miss
i=0, j=4	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
	A[0][5]	hit
i=0, j=5	B[0][5]	conflict miss
	A[0][5]	conflict miss
	A[0][6]	hit
i=0, j=6	B[0][6]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	B[0][4]	B[0][5]	B[0][6]	B[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

i=0, j=4	A[0][4]	compulsory miss
	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
i=0, j=5	A[0][5]	hit
	B[0][5]	conflict miss
	A[0][5]	conflict miss
i=0, j=6	A[0][6]	hit
	B[0][6]	conflict miss
	C[6]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	C[4]	C[5]	C[6]	C[7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

i=0, j=4	A[0][4]	compulsory miss
	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
i=0, j=5	A[0][5]	hit
	B[0][5]	conflict miss
	A[0][5]	conflict miss
i=0, j=6	A[0][6]	hit
	B[0][6]	conflict miss
	C[6]	conflict miss
	A[0][6]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[0][4]	A[0][5]	A[0][6]	A[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

	A[0][4]	compulsory miss
i=0, j=4	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
	A[0][5]	hit
i=0, j=5	B[0][5]	conflict miss
	A[0][5]	conflict miss
	A[0][6]	hit
i=0, j=6	B[0][6]	conflict miss
	C[6]	conflict miss
	A[0][6]	conflict miss
	A[0][7]	hit
i=0, j=7		

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[0][4]	A[0][5]	A[0][6]	A[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

i=0, j=4	A[0][4]	compulsory miss
	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
i=0, j=5	A[0][5]	hit
	B[0][5]	conflict miss
	A[0][5]	conflict miss
i=0, j=6	A[0][6]	hit
	B[0][6]	conflict miss
	C[6]	conflict miss
	A[0][6]	conflict miss
i=0, j=7	A[0][7]	hit
	B[0][7]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	B[0][4]	B[0][5]	B[0][6]	B[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος Β (i)

i=0, j=4	A[0][4]	compulsory miss
	B[0][4]	compulsory miss
	C[4]	compulsory miss
	A[0][4]	conflict miss
i=0, j=5	A[0][5]	hit
	B[0][5]	conflict miss
	A[0][5]	conflict miss
i=0, j=6	A[0][6]	hit
	B[0][6]	conflict miss
	C[6]	conflict miss
	A[0][6]	conflict miss
i=0, j=7	A[0][7]	hit
	B[0][7]	conflict miss
	A[0][7]	conflict miss

0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[0][4]	A[0][5]	A[0][6]	A[0][7]
2				
3				
4				
5				
6				
7				
8				
9				
10				
…				

Άσκηση 3η – Μέρος B (i)

Σύνολο για i = 0:

	αναφορές στη μνήμη			αποτέλεσμα			
i = 0:							
A[0][0]	B[0][0]	C[0]	A[0][0]	m	m	m	m
A[0][1]	B[0][1]		A[0][1]	h	m		m
A[0][2]	B[0][2]	C[2]	A[0][2]	h	m	m	m
A[0][3]	B[0][3]		A[0][3]	h	m		m
A[0][4]	B[0][4]	C[4]	A[0][4]	m	m	m	m
A[0][5]	B[0][5]		A[0][5]	h	m		m
A[0][6]	B[0][6]	C[6]	A[0][6]	h	m	m	m
A[0][7]	B[0][7]		A[0][7]	h	m		m

28 accesses
22 misses
6 hits

Το ίδιο pattern για i = 1, …, 7

Άσκηση 3η – Μέρος Β (i)

Συνολικά

Accesses : 28 * 8 = 224
Misses : 22 * 8 = 176
Hits : 6 * 8 = 48

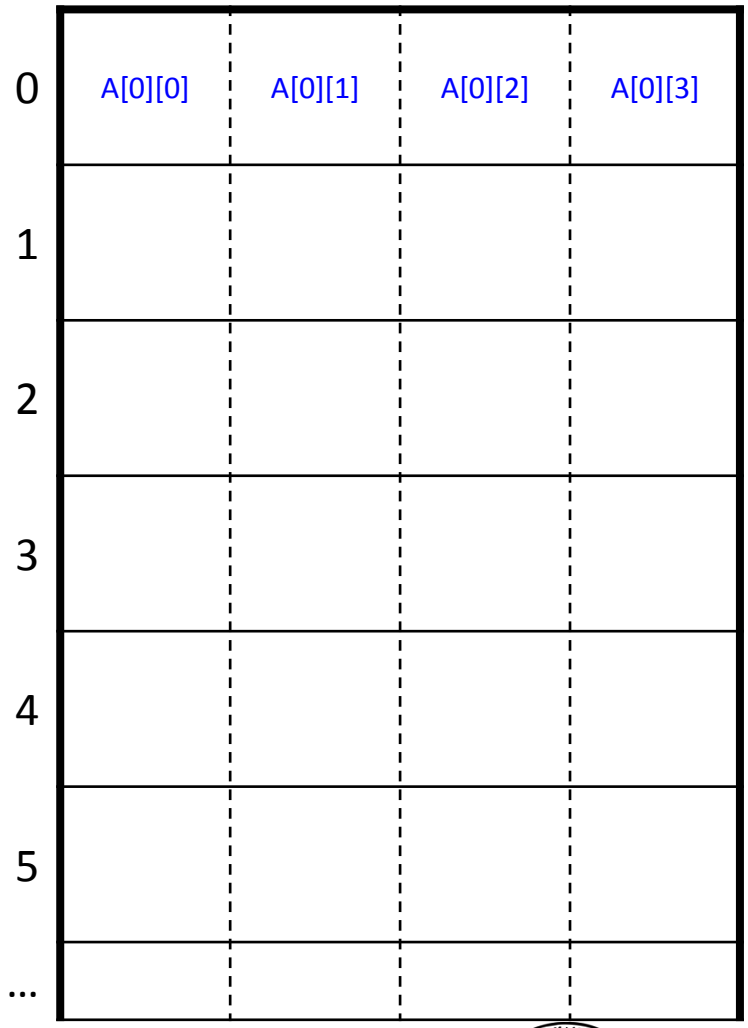
Άσκηση 3η – Μέρος Β (ii)

2ο Ζητούμενο: Σας προτείνουν να αντικαταστήσετε την κρυφή μνήμη με μια άλλη ίδιας χωρητικότητας, 2-way associative, με ίδιο μέγεθος block που χρησιμοποιεί LRU πολιτική αντικατάστασης. Βελτιώνεται η απόδοση του κώδικα; Υπολογίστε τον καινούριο αριθμό hits και misses.

Άσκηση 3η – Μέρος B (ii)

A[0][0] compulsory miss

i=0, j=0

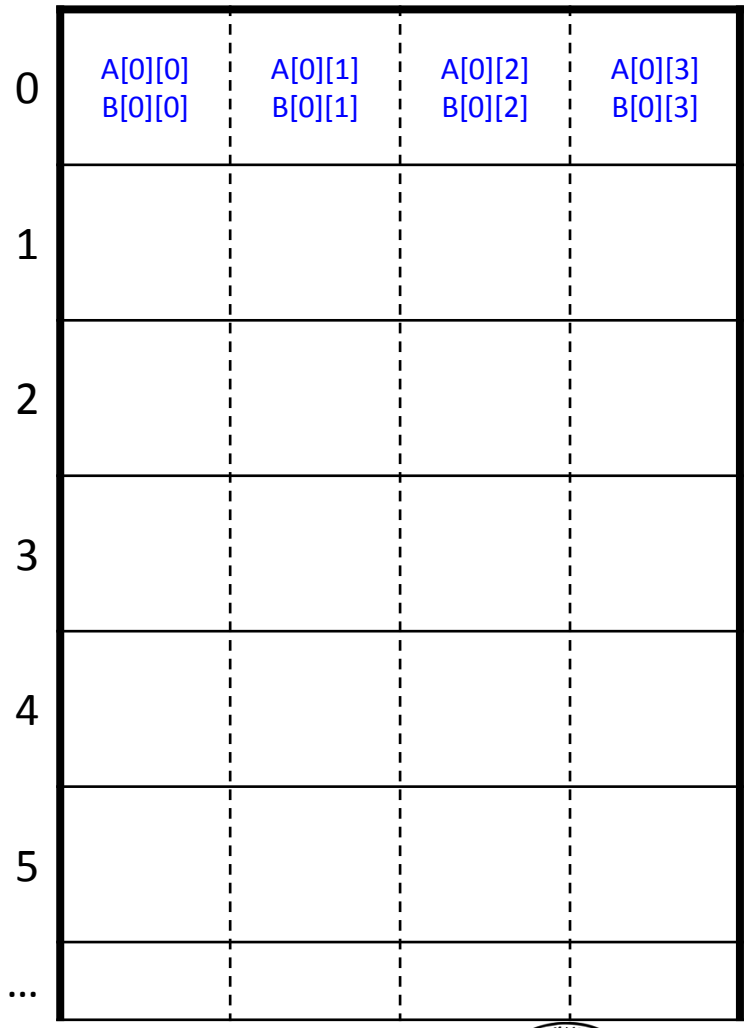


Άσκηση 3η – Μέρος B (ii)

i=0, j=0

A[0][0] compulsory miss

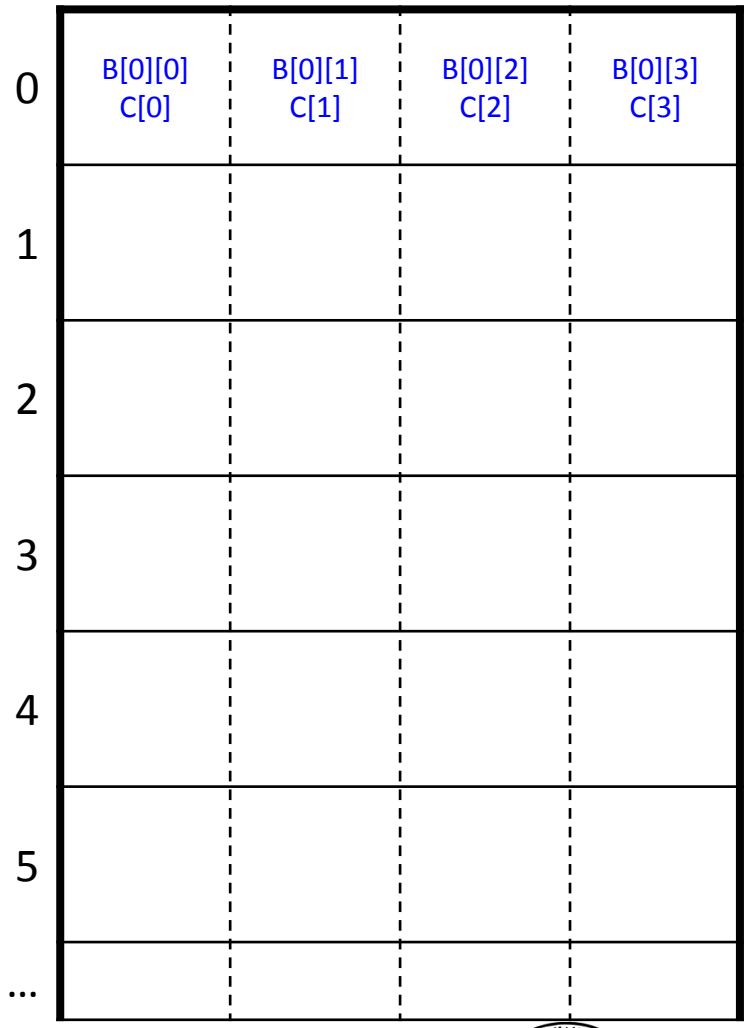
B[0][0] compulsory miss



Άσκηση 3η – Μέρος B (ii)

i=0, j=0

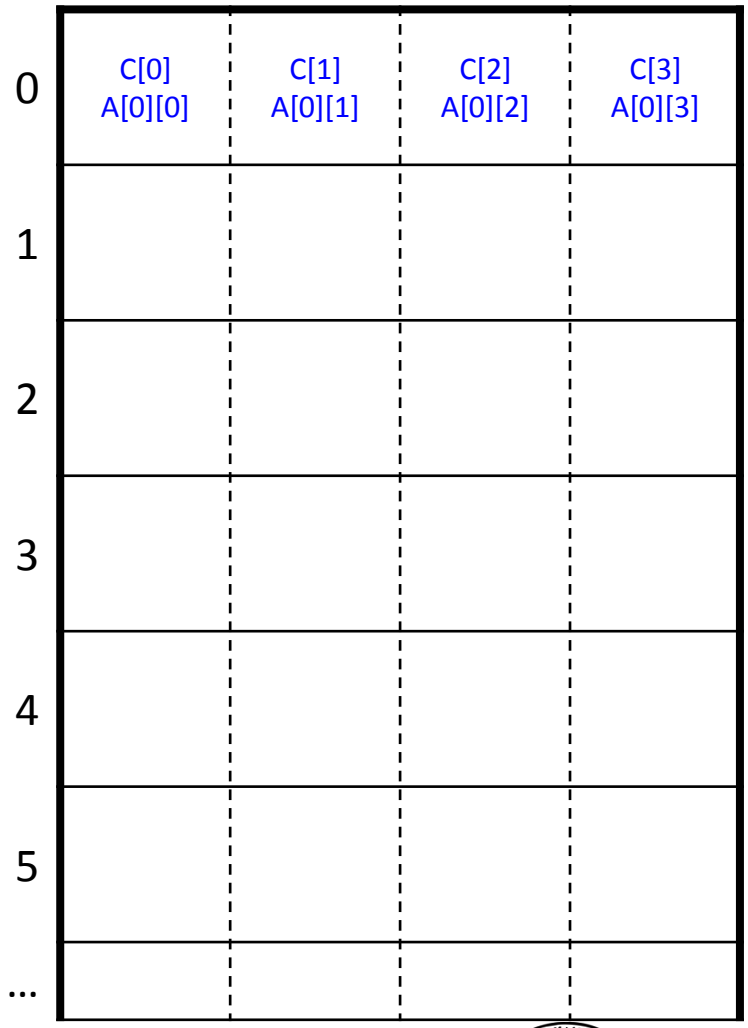
A[0][0]	compulsory miss
B[0][0]	compulsory miss
C[0]	compulsory miss



Άσκηση 3η – Μέρος B (ii)

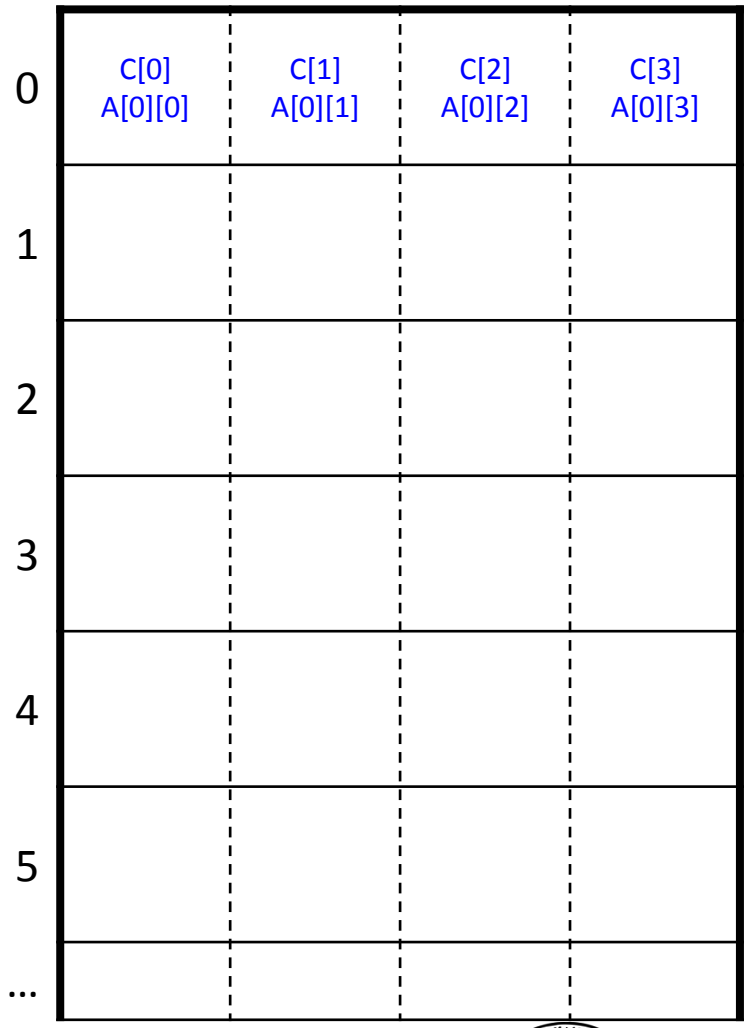
i=0, j=0

A[0][0]	compulsory miss
B[0][0]	compulsory miss
C[0]	compulsory miss
A[0][0]	conflict miss



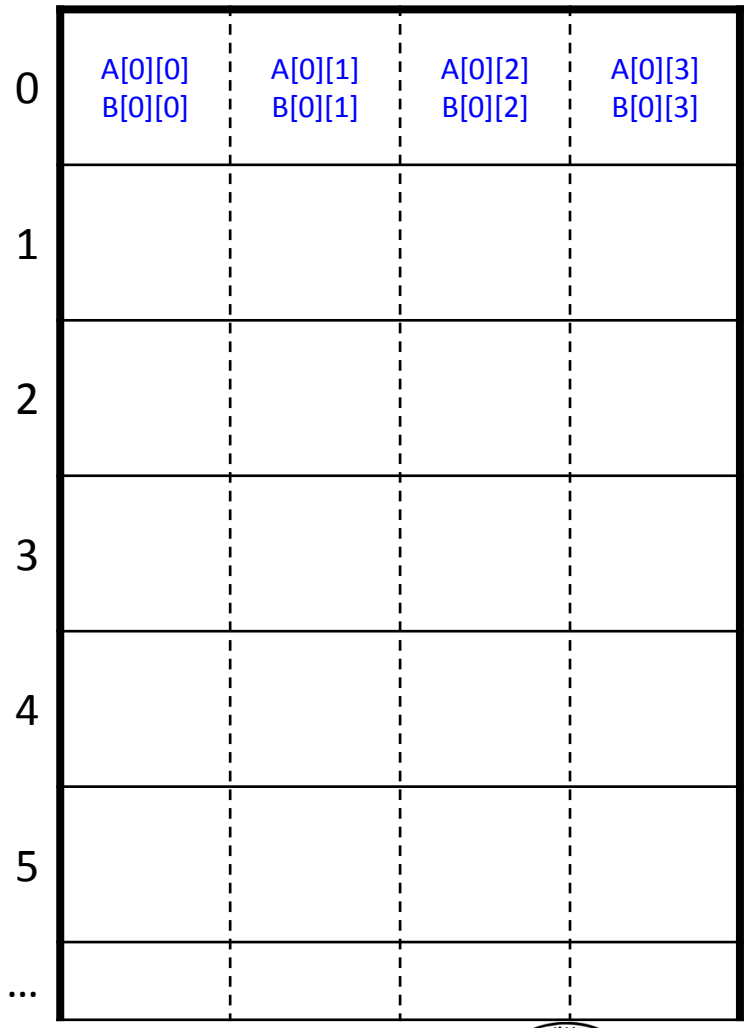
Άσκηση 3η – Μέρος B (ii)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
	A[0][1]	hit
i=0, j=1		



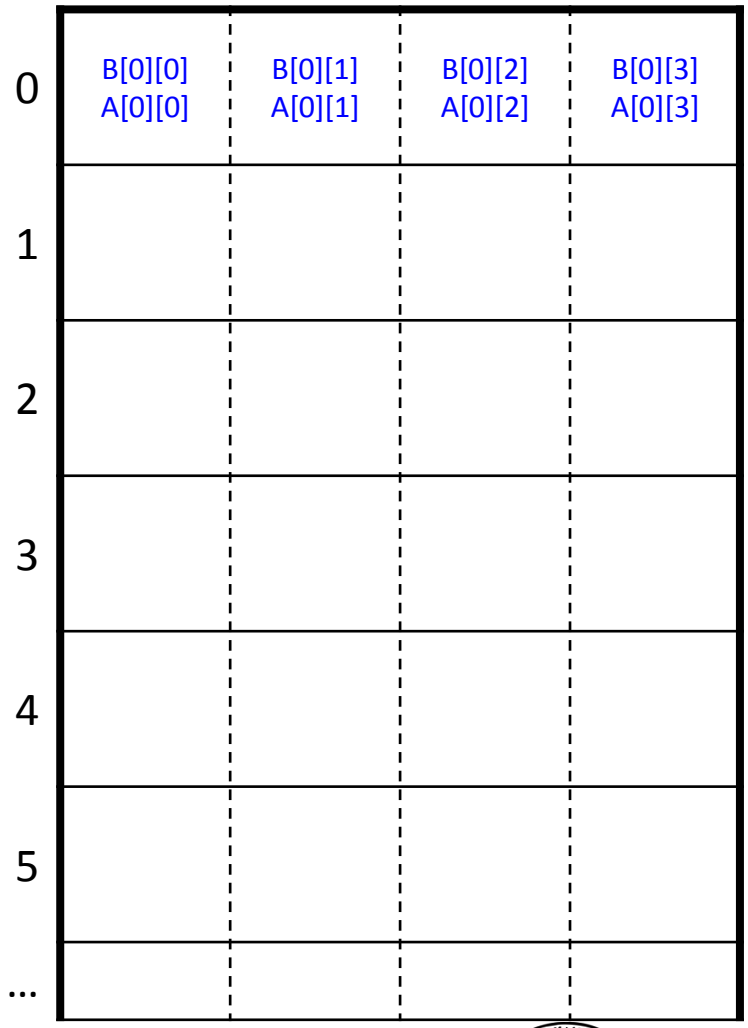
Άσκηση 3η – Μέρος B (ii)

	A[0][0]	compulsory miss
i=0, j=0	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
	A[0][1]	hit
i=0, j=1	B[0][1]	conflict miss



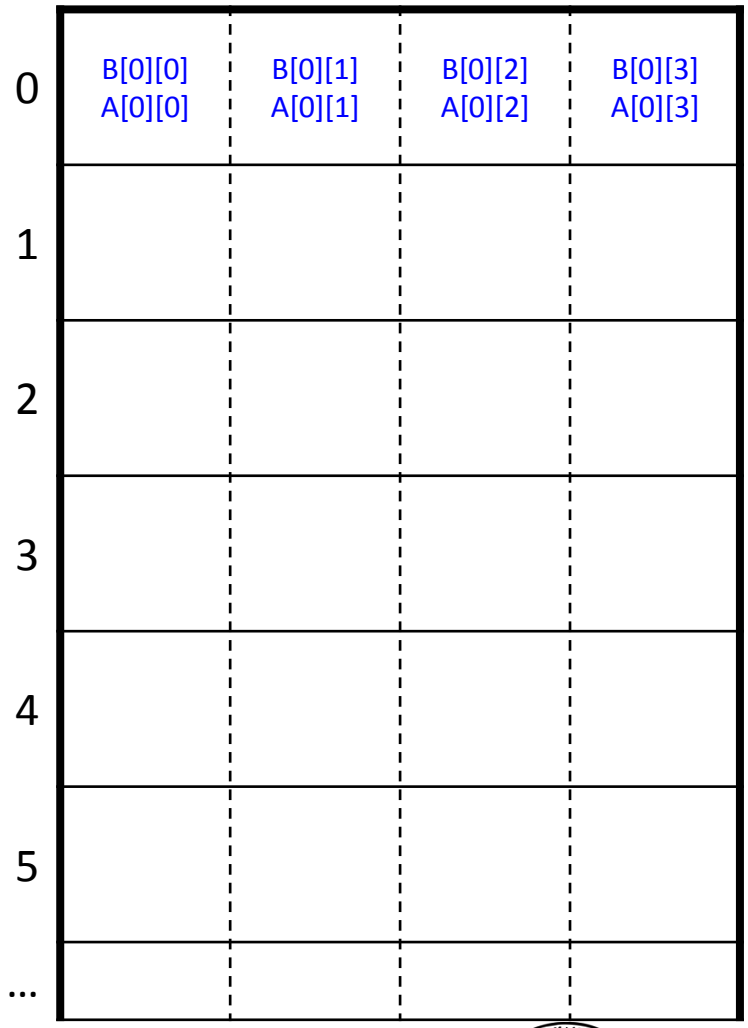
Άσκηση 3η – Μέρος B (ii)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	hit



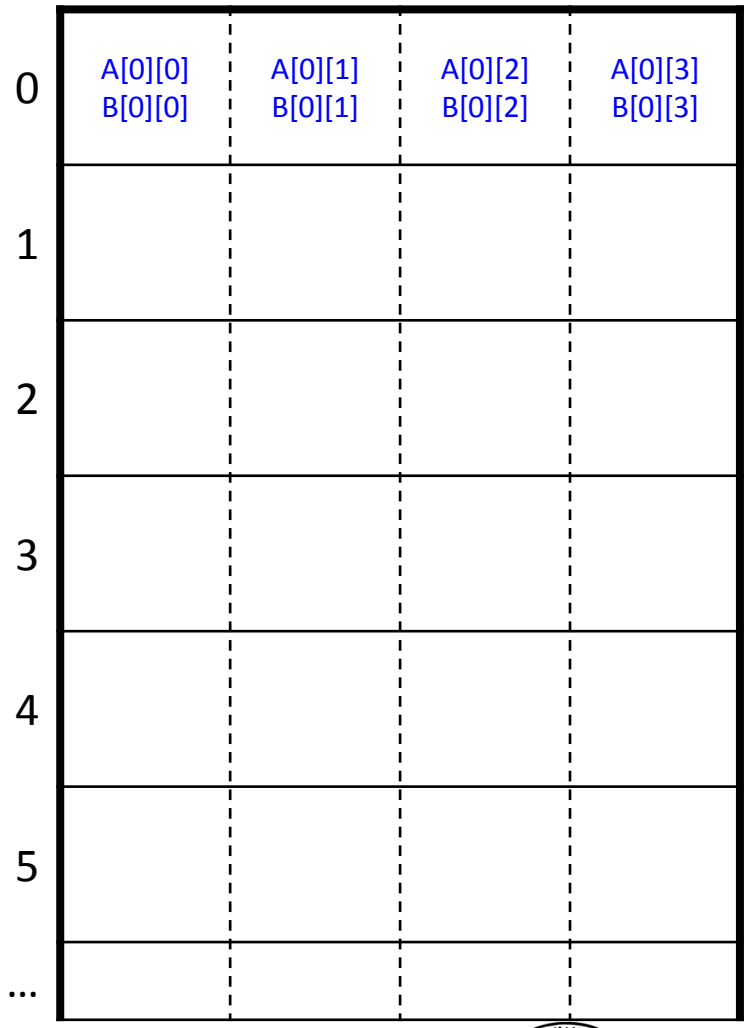
Άσκηση 3η – Μέρος B (ii)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	hit
i=0, j=2	A[0][2]	hit



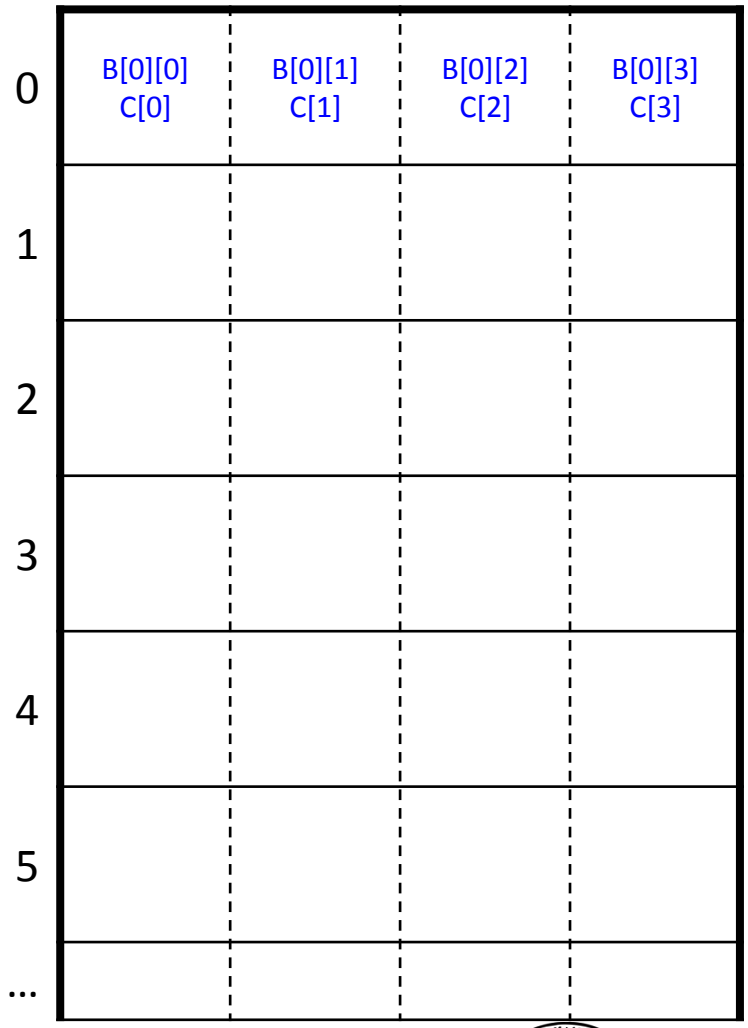
Άσκηση 3η – Μέρος B (ii)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	hit
i=0, j=2	A[0][2]	hit
	B[0][2]	hit



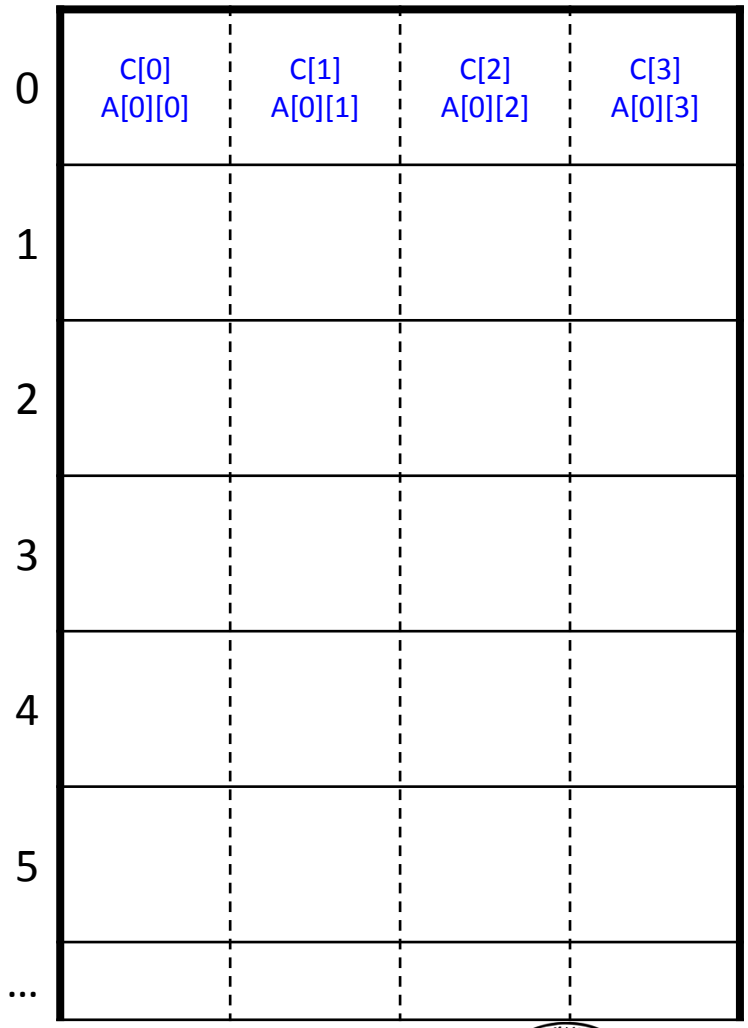
Άσκηση 3η – Μέρος B (ii)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	hit
i=0, j=2	A[0][2]	hit
	B[0][2]	hit
	C[2]	conflict miss



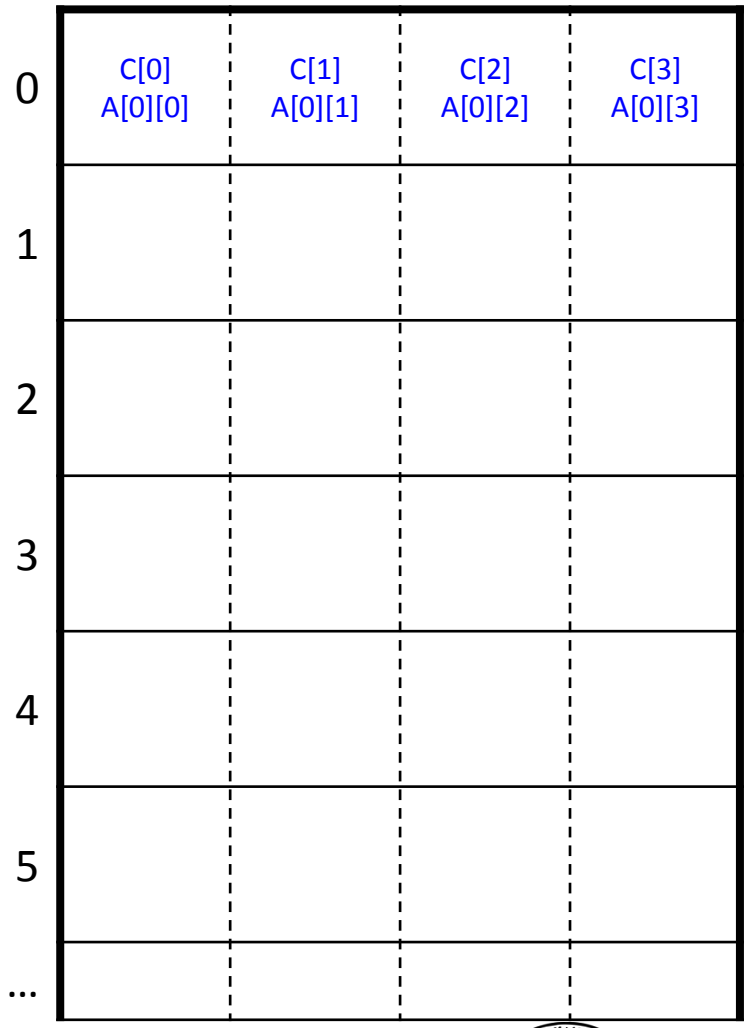
Άσκηση 3η – Μέρος B (ii)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	hit
i=0, j=2	A[0][2]	hit
	B[0][2]	hit
	C[2]	conflict miss
	A[0][2]	conflict miss



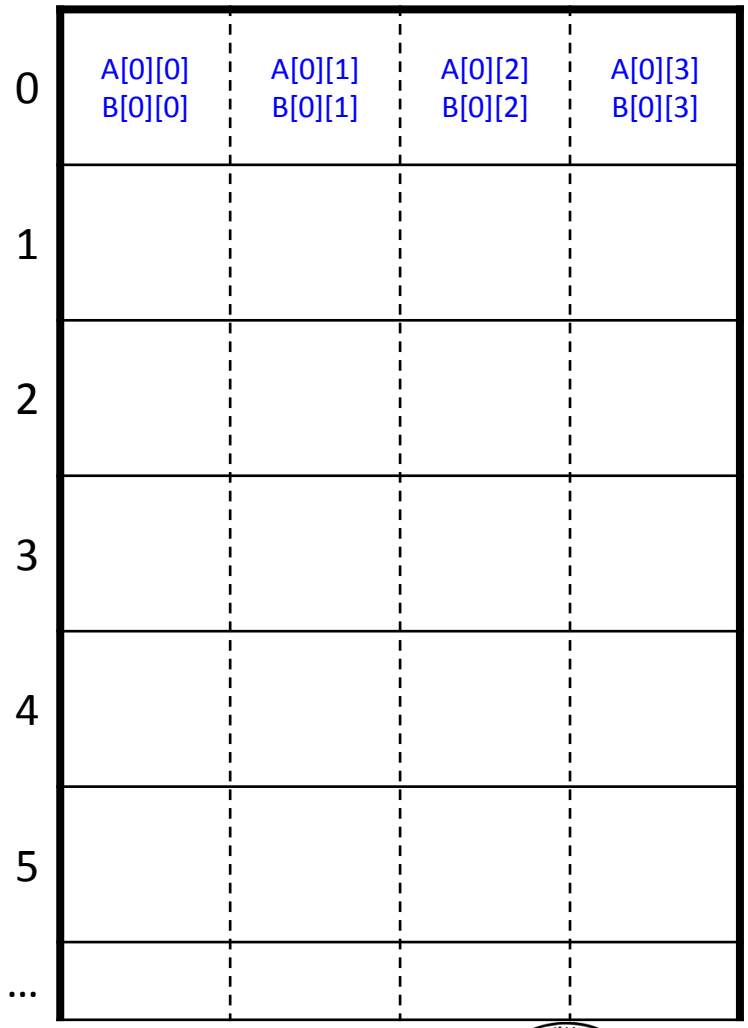
Άσκηση 3η – Μέρος B (ii)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	hit
i=0, j=2	A[0][2]	hit
	B[0][2]	hit
	C[2]	conflict miss
	A[0][2]	conflict miss
i=0, j=3	A[0][3]	hit



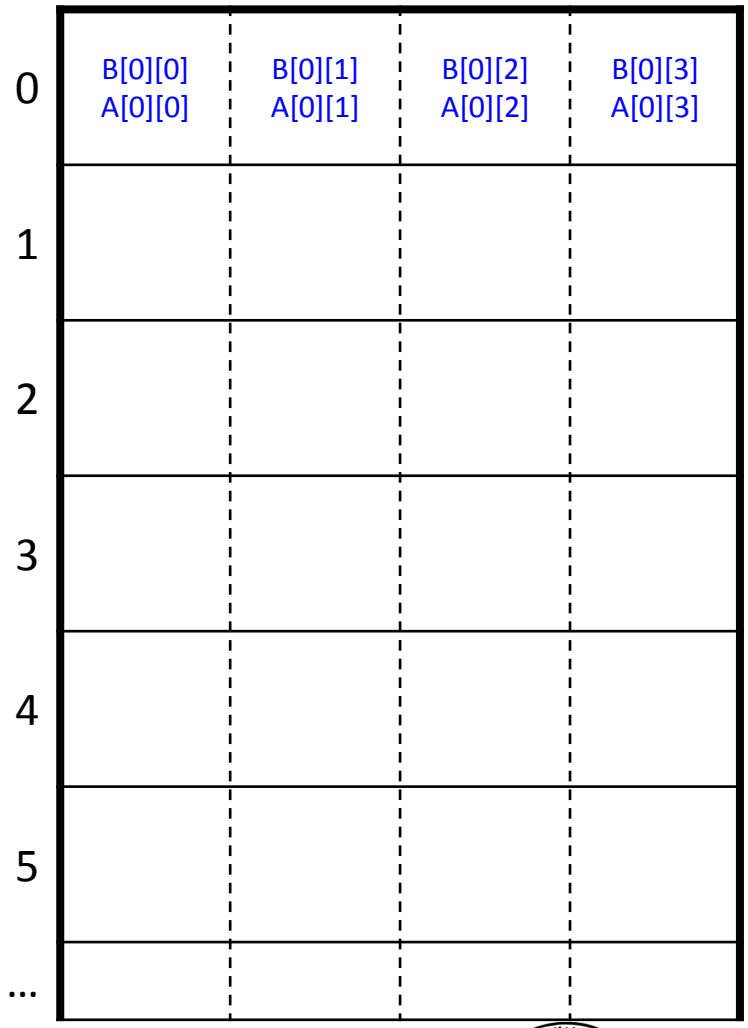
Άσκηση 3η – Μέρος B (ii)

	A[0][0]	compulsory miss
i=0, j=0	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
	A[0][1]	hit
i=0, j=1	B[0][1]	conflict miss
	A[0][1]	hit
	A[0][2]	hit
i=0, j=2	B[0][2]	hit
	C[2]	conflict miss
	A[0][2]	conflict miss
	A[0][3]	hit
i=0, j=3	B[0][3]	conflict miss



Άσκηση 3η – Μέρος B (ii)

i=0, j=0	A[0][0]	compulsory miss
	B[0][0]	compulsory miss
	C[0]	compulsory miss
	A[0][0]	conflict miss
i=0, j=1	A[0][1]	hit
	B[0][1]	conflict miss
	A[0][1]	hit
i=0, j=2	A[0][2]	hit
	B[0][2]	hit
	C[2]	conflict miss
	A[0][2]	conflict miss
i=0, j=3	A[0][3]	hit
	B[0][3]	conflict miss
	A[0][3]	hit



National Technical University of Athens
CSLab

Άσκηση 3η – Μέρος Β (ii)

Σύνολο για i = 0:

	αναφορές στη μνήμη			αποτέλεσμα			
i = 0:							
A[0][0]	B[0][0]	C[0]	A[0][0]	m	m	m	m
A[0][1]	B[0][1]		A[0][1]	h	m		h
A[0][2]	B[0][2]	C[2]	A[0][2]	h	h	m	m
A[0][3]	B[0][3]		A[0][3]	h	m		h
A[0][4]	B[0][4]	C[4]	A[0][4]	m	m	m	m
A[0][5]	B[0][5]		A[0][5]	h	m		h
A[0][6]	B[0][6]	C[6]	A[0][6]	h	h	m	m
A[0][7]	B[0][7]		A[0][7]	h	m		h

28 accesses
16 misses
12 hits

Το ίδιο pattern για i = 1, …, 7

Άσκηση 3η – Μέρος Β (ii)

Direct-mapped (i)

Accesses : 28 * 8 = 224
Misses : 22 * 8 = 176
Hits : 6 * 8 = 48

2-way (ii)

Accesses : 28 * 8 = 224
Misses : 16 * 8 = 128
Hits : 12 * 8 = 96

Άσκηση 3η – Μέρος Β (iii)

2ο Ζητούμενο: Τροποποιήστε κατάλληλα τον κώδικα του loop ώστε να βελτιωθεί η απόδοση του όταν εκτελείται στο σύστημα του ερωτήματος Β. Δώστε τον καινούριο κώδικα καθώς και τον καινούριο αριθμό hits και misses.

Άσκηση 3η – Μέρος Β (iii)

1η βελτιστοποίηση: Αναδιάταξη των προσβάσεων

```
int i,j;
double A[8][8], B[8][8], C[64];

for (i=0; i < 8; i++)
  for (j=0; j < 8; j++)
    if (j % 2 == 0)
      A[i][j] = A[i][j] + B[i][j] + C[8*i +j] ;
    else
      A[i][j] = A[i][j] + B[i][j];
```

Άσκηση 3η – Μέρος Β (iii)

1η βελτιστοποίηση: Αναδιάταξη των προσβάσεων

```
int i,j;
double A[8][8], B[8][8], C[64];

for (i=0; i < 8; i++)
  for (j=0; j < 8; j++)
    if (j % 2 == 0)
      A[i][j] = C[8*i +j] + A[i][j] + B[i][j] ;
    else
      A[i][j] = A[i][j] + B[i][j];
```

Άσκηση 3η – Μέρος Β (iii)

Σύνολο για i = 0:

	αναφορές στη μνήμη			αποτέλεσμα			
i = 0:							
C[0]	A[0][0]	B[0][0]	A[0][0]	m	m	m	h
	A[0][1]	B[0][1]	A[0][1]		h	h	h
C[2]	A[0][2]	B[0][2]	A[0][2]	m	h	m	h
	A[0][3]	B[0][3]	A[0][3]		h	h	h
C[4]	A[0][4]	B[0][4]	A[0][4]	m	m	m	h
	A[0][5]	B[0][5]	A[0][5]		h	h	h
C[6]	A[0][6]	B[0][6]	A[0][6]	m	h	m	h
	A[0][7]	B[0][7]	A[0][7]		h	h	h

28 accesses
10 misses
18 hits

Το ίδιο pattern για i = 1, …, 7

Άσκηση 3η – Μέρος Β (iii)

Συνολικά

Accesses : 28 * 8 = 224
Misses : 10 * 8 = 80
Hits : 18 * 8 = 144

Άσκηση 3η – Μέρος Β (iii)

2η βελτιστοποίηση: Loop distribution

```
int i,j;
double A[8][8], B[8][8], C[64];

for (i=0; i < 8; i++)
  for (j=0; j < 8; j++)
    if (j % 2 == 0)
      A[i][j] = A[i][j] + B[i][j] + C[8*i +j];
    else
      A[i][j] = A[i][j] + B[i][j];
```

Άσκηση 3η – Μέρος Β (iii)

2η βελτιστοποίηση: Loop distribution

```
int i,j;
double A[8][8], B[8][8], C[64];

for (i=0; i < 8; i++) {
  for (j=0; j < 8; j += 2)
    A[i][j] = A[i][j] + B[i][j] + C[8*i +j];
  for (j=1; j < 8; j += 2 )
    A[i][j] = A[i][j] + B[i][j];
}
```

Άσκηση 3η – Μέρος Β (iii)

Σύνολο για i = 0:

	αναφορές στη μνήμη			αποτέλεσμα			
i = 0:							
A[0][0]	B[0][0]	C[0]	A[0][0]	m	m	m	m
A[0][2]	B[0][2]	C[2]	A[0][2]	h	m	m	m
A[0][4]	B[0][4]	C[4]	A[0][4]	m	m	m	m
A[0][6]	B[0][6]	C[6]	A[0][6]	h	m	m	m
A[0][1]	B[0][1]		A[0][1]	h	m		h
A[0][3]	B[0][3]		A[0][3]	h	h		h
A[0][5]	B[0][5]		A[0][5]	h	m		h
A[0][7]	B[0][7]		A[0][7]	h	h		h

28 accesses
16 misses
12 hits

Το ίδιο pattern για i = 1, …, 7

Άσκηση 3η – Μέρος Β (iii)

Συνολικά

Accesses : 28 * 8 = 224
Misses : 16 * 8 = 128
Hits : 12 * 8 = 96

Άσκηση 3η – Μέρος Β (iii)

3η βελτιστοποίηση: Merging arrays

```
int i,j;
double A[8][8], B[8][8], C[64];

for (i=0; i < 8; i++)
  for (j=0; j < 8; j++)
    if (j % 2 == 0)
      A[i][j] = A[i][j] + B[i][j] + C[8*i +j];
    else
      A[i][j] = A[i][j] + B[i][j];
```

Άσκηση 3η – Μέρος Β (iii)

3η βελτιστοποίηση: Merging arrays

```
int i,j;
double C[64];
struct {
  double A;
  double B;
} AB[8][8];

for (i=0; i < 8; i++)
  for (j=0; j < 8; j++)
    if (j % 2 == 0)
      AB[i][j].A = AB[i][j].A + AB[i][j].B + C[8*i +j];
    else
      AB[i][j].A = AB[i][j].A + AB[i][j].B;
```

Άσκηση 3η – Μέρος Β (iii)

Σύνολο για i = 0:

	αναφορές στη μνήμη			αποτέλεσμα			
i = 0:							
AB[0][0].A	AB[0][0].B	C[0]	AB[0][0].A	m	h	m	h
AB[0][1].A	AB[0][1].B		AB[0][1].A	h	h		h
AB[0][2].A	AB[0][2].B	C[2]	AB[0][2].A	m	h	h	h
AB[0][3].A	AB[0][3].B		AB[0][3].A	h	h		h
AB[0][4].A	AB[0][4].B	C[4]	AB[0][4].A	m	h	m	h
AB[0][5].A	AB[0][5].B		AB[0][5].A	h	h		h
AB[0][6].A	AB[0][6].B	C[6]	AB[0][6].A	m	h	h	h
AB[0][7].A	AB[0][7].B		AB[0][7].A	h	h		h

28 accesses
6 misses
22 hits

Το ίδιο pattern για i = 1, …, 7

Άσκηση 3η – Μέρος Β (iii)

Συνολικά

Accesses : 28 * 8 = 224
Misses : 6 * 8 = 48
Hits : 22 * 8 = 176