



HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

Thực hành:

KIẾN TRÚC MÁY TÍNH

Nhóm 13

Nguyễn Thị Hạ - 20225622

Bùi Thị Xuân - 20225957



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Giảng viên hướng dẫn: TS. Đỗ Công Thuận
Sinh viên thực hiện: Nguyễn Thị Hạ -
20225622

ONE LOVE. ONE FUTURE.



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

1. CURIOSITY MARSBOT

Giảng viên hướng dẫn: TS. Đỗ Công Thuận
Sinh viên thực hiện: Nguyễn Thị Hạ -
20225622

ONE LOVE. ONE FUTURE.

- Xe tự hành Curiosity Marsbot chạy trên sao Hỏa, được vận hành từ xa bởi các lập trình viên Trái Đất.
- Bằng cách gửi đi các mã điều khiển từ một bàn phím ma trận, lập trình viên điều khiển quá trình di chuyển của Marsbot như sau:

Mã điều khiển	Ý nghĩa
1b4	Marbot bắt đầu chuyển động.
c68	Marbot đứng im.
444	Rẽ trái 90 độ so với phương chuyển động gần nhất và giữ hướng mới.
666	Rẽ phải 90 độ so với phương chuyển động gần nhất và giữ hướng mới.
dad	Bắt đầu để lại vết trên đường.
cbc	Chấm dứt để lại vết trên đường.
999	Tự động quay trở lại theo lộ trình ngược lại. Không vẽ vết, không nhận mã khác cho tới khi kết thúc lộ trình ngược.

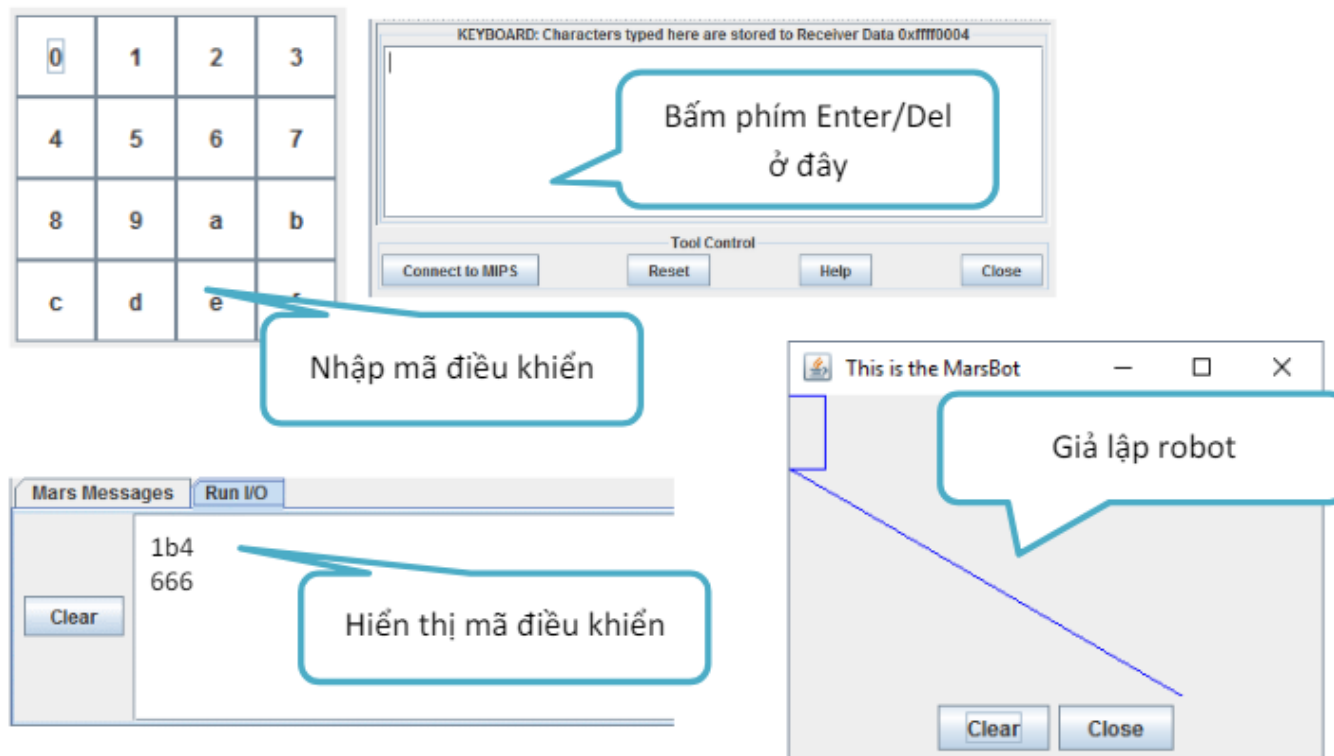
Yêu cầu

- Khi khởi động chương trình thì **Curiosity Marsbot** sẽ tự động di chuyển theo một quỹ đạo (tùy chọn) rồi dừng lại chờ nhận mã điều khiển.
- Sau khi nhận mã điều khiển, **Curiosity Marsbot** sẽ không xử lý ngay, mà phải đợi lệnh kích hoạt mã từ bàn phím **Keyboard & Display MMIO Simulator**. Có 2 lệnh như vậy:

Kích hoạt mã	Ý nghĩa
Phím Enter	Kết thúc nhập mã và yêu cầu Marbot thực thi.
Phím Del	Xóa toàn bộ mã điều khiển đang nhập

Yêu cầu


- Lập trình để **Marsbot** có thể hoạt động như đã mô tả.
- Bổ sung thêm tính năng: mỗi khi gửi một mã điều khiển cho **Marsbot**, hiển thị mã đó lên màn hình console để người xem có thể giám sát lộ trình của xe.



Tạo chương trình nhận mã lệnh từ Digital Lab Sim sau đó được kích hoạt từ Keyboard & Display MMIO Simulator.



Đọc key nhận được từ Keyboard & Display MMIO Simulator sau đó sẽ kích hoạt tương ứng: thực hiện lệnh (Enter), xóa mã lệnh đang nhập (Delete).



Sau đó so sánh mã lệnh nhận được có đúng với mã lệnh đã cho hay không rồi cho Marsbot thực thi.

Giải thích code

Địa chỉ bộ nhớ và các biến

```
#eqv Keyboard

.eqv IN_ADRESS_HEXА_KEYBOARD 0xFFFF0012

.eqv OUT_ADRESS_HEXА_KEYBOARD 0xFFFF0014

.eqv KEY_CODE 0xFFFF0004

.eqv KEY_READY 0xFFFF0000

#eqv Marsbot

.eqv HEADING 0xffff8010

    # 0 : North (up)

    # 90: East (right)

    # 180: South (down)

    # 270: West (left)

.eqv MOVING 0xffff8050

.eqv LEAVETRACK 0xffff8020

.eqv WHEREX 0xffff8030

.eqv WHEREY 0xffff8040

.eqv Digital Lab Sim

#0-9 & a-f
```

```
#Function code

ChuyenDong: .asciiz "1b4"

Dung: .asciiz "c68"

ReTrai: .asciiz "444"

RePhai: .asciiz "666"

DeVet: .asciiz "dad"

DungDeVet: .asciiz "cbc"

DiNguoc: .asciiz "999"

MaLoi: .asciiz "Ma khong hop le!"

InputCode: .space 50

InputCode1: .space 50

CodeLong: .word 0

CodeLong1: .word 0

HuongDi: .word 0

Path: .space 600

PathLong: .word 12
```

Giải thích code

Hàm main:

```
68
69
70 .text
71 main:
72     li $k0, KEY_CODE
73     li $k1, KEY_READY
74     #-----
75     # Enable the interrupt of Keyboard matrix 4x4 of Digital Lab Sim
76     #-----
77     li $t1, IN_ADRESS_HEX_A_KEYBOARD
78     li $t3, 0x80 # bit 7 = 1 to enable
79     sb $t3, 0($t1)
80     #-----
81 loop:    nop
82 WaitForKey: lw $t5, 0($k1)           #$t5 = [$k1] = KEY_READY
83             beq $t5, $zero, WaitForKey    #if $t5 == 0 then Polling
84             nop
85             beq $t5, $zero, WaitForKey
86 ReadKey:   lw $t6, 0($k0)             #$t6 = [$k0] = KEY_CODE
87             beq $t6, 127, Delete         #if $t6 == delete key then r
88                                           #127 is delete key in ascii
89             beq $t6, 32, repeat
90
91             beq $t6, 13, loop           #if $t6 != '\n' then Polling
92             nop
93
94
```

- Kích hoạt ngắt cho ma trận bàn phím.
- Vào vòng lặp liên tục chờ phím nhấn.
- Khi nhận được phím, kiểm tra cụ thể
- Gọi hàm CheckInput để xử lý mã nhận.
- In mã nhận ra bảng điều khiển.

Các hàm xử lý phím bấm

CheckInput:

```
jal storePath

la $s2, CodeLong
lw $s2, 0($s2)

bne $s2, 3, Error

la $s3, ChuyenDong
jal Equal
beq $t0, 1, CodeGO

la $s3, Dung
jal Equal
beq $t0, 1, CodeStop

la $s3, ReTrai
jal Equal
beq $t0, 1, CodeLeft

la $s3, RePhai
jal Equal
beq $t0, 1, CodeRight

la $s3, DeVet
```

Delete:

```
#backup
addi $sp, $sp, 4
sw $t1, 0($sp)
addi $sp, $sp, 4
sw $t2, 0($sp)
addi $sp, $sp, 4
sw $s1, 0($sp)
addi $sp, $sp, 4
sw $t3, 0($sp)
addi $sp, $sp, 4
sw $s2, 0($sp)

#processing
la $s2, CodeLong
lw $t3, 0($s2)
addi $t1, $zero, -1
addi $t2, $zero, 0
la $s1, InputCode
addi $s1, $s1, -1

Delete_loop: addi $t1, $t1, 1
add $s1, $s1, 1
sb $t2, 0($s1)

bne $t1, $t3, Delete_loop #if
```

repeat:

```
#backup
addi $sp, $sp, 4
sw $s1, 0($sp)
addi $sp, $sp, 4
sw $s2, 0($sp)
addi $sp, $sp, 4
sw $s0, 0($sp)
addi $sp, $sp, 4
sw $t1, 0($sp)
addi $sp, $sp, 4
sw $t2, 0($sp)
addi $sp, $sp, 4
sw $t3, 0($sp)

#processing
la $s1, InputCode1
la $s2, InputCode
strcpy2:
add $s0, $zero, $zero # s0 = i
L2:
add $t1, $s0, $s1 # t1 = s0 + a
lb $t2, 0($t1) # t2 = gia tri
add $t3, $s2, $s0 # t3 = dia c
sb $t2, 0($t3) # Gan gia tri c
beq $t2, $zero, end_of_strcpy2
```

Kiểm tra đầu vào và thực hiện chức năng tương ứng

Xóa mã đầu vào hiện tại

Lặp lại mã đầu vào trước đó

Các hàm xử lý dữ liệu:

`storePath:`

```
#backup
addi $sp, $sp, 4
sw $t1, 0($sp)
addi $sp, $sp, 4
sw $t2, 0($sp)
addi $sp, $sp, 4
sw $t3, 0($sp)
addi $sp, $sp, 4
sw $t4, 0($sp)
addi $sp, $sp, 4
sw $s1, 0($sp)
addi $sp, $sp, 4
sw $s2, 0($sp)
addi $sp, $sp, 4
sw $s3, 0($sp)
addi $sp, $sp, 4
sw $s4, 0($sp)

#processing
li $t1, WHEREX
lw $s1, 0($t1)      #s1 = x
li $t2, WHEREY
lw $s2, 0($t2)      #s2 = y

la $s4, HuongDi
lw $s4, 0($s4)      #s4 = now heading
```

`storePath:`

Lưu vị trí và hướng hiện tại của Robot vào cấu trúc dữ liệu Path.

`strcpy1, strcpy2, strcpy3, strcpy4 :`

Sao chép mã đầu vào của người dùng từ bộ đệm này sang bộ đệm khác.

Giải thích code

Các hàm hỗ trợ:

```
Equal:
    #backup
    addi $sp,$sp,4
    sw $t1, 0($sp)
    addi $sp,$sp,4
    sw $s1, 0($sp)
    addi $sp,$sp,4
    sw $t2, 0($sp)
    addi $sp,$sp,4
    sw $t3, 0($sp)

    #processing
    addi $t1, $zero, -1           #$t1 = -1
    add $t0, $zero, $zero
    la $s1, InputCode             #$s1 = InputCode
Equal_loop: addi $t1, $t1, 1       #i++

    add $t2, $s1, $t1             #$t2 = InputCode[i]
    lb $t2, 0($t2)                #$t2 = InputCode[i]

    add $t3, $s3, $t1             #$t3 = s + i
    lb $t3, 0($t3)                #$t3 = s[i]

    bne $t2, $t3, isNotEqual       #if $t2 != $t3

    bne $t1, 2, Equal_loop        #if $t1 <= 2 Delete loop
    nop
```

```
Error: li $v0, 4
        la $a0, InputCode
        syscall
        nop

        li $v0, 55
        la $a0, MaLoi
        syscall
        nop
        nop
        j Delete
        nop
        j Delete

#-----
```

In ra thông báo mã lỗi không hợp lệ

So sánh mã đầu vào với các mã chức năng

Các hàm chức năng:

#Bắt đầu chuyển động

```
CodeGO: jal GO
        j Print
```

#Marsbot dừng im

```
CodeStop: jal STOP
          j Print
```

```
GO:      #backup
        addi $sp,$sp,4
        sw $at,0($sp)
        addi $sp,$sp,4
        sw $k0,0($sp)
        #processing
        li $at, MOVING # change MOVING port
        addi $k0, $zero,1 # to logic 1,
        sb $k0, 0($at) # to start running
        #restore
        lw $k0, 0($sp)
        addi $sp,$sp,-4
        lw $at, 0($sp)
        addi $sp,$sp,-4

        jr $ra
        nop
        jr $ra
```

```
#-----
STOP:    #backup
        addi $sp,$sp,4
        sw $at,0($sp)
        #processing
        li $at, MOVING # change MOVING port to 0
        sb $zero, 0($at) # to stop
        #restore
        lw $at, 0($sp)
        addi $sp,$sp,-4

        jr $ra
        nop
        jr $ra
```

Bắt đầu chuyển động và dừng chuyển động của Marsbot.

Các hàm chức năng

#De lai vet tren duong

CodeTrack: jal TRACK

j Print

#Dung de lai vet tren duong

CodeUntrack: jal UNTRACK

j Print

TRACK: *#backup*

addi \$sp,\$sp,4

sw \$at,0(\$sp)

addi \$sp,\$sp,4

sw \$k0,0(\$sp)

#processing

li \$at, LEAVETRACK *# change LEAVETRACK port*

addi \$k0, \$zero,1 *# to logic 1,*

sb \$k0, 0(\$at) *# to start tracking*

#restore

lw \$k0, 0(\$sp)

addi \$sp,\$sp,-4

lw \$at, 0(\$sp)

addi \$sp,\$sp,-4

jr \$ra

nop

jr \$ra

#-----

"

UNTRACK procedure, to stop drawing line

param[in] none

#-----

UNTRACK: *#backup*

addi \$sp,\$sp,4

sw \$at,0(\$sp)

#processing

li \$at, LEAVETRACK *# change LEAVETRACK*

sb \$zero, 0(\$at) *# to stop drawing tail*

#restore

lw \$at, 0(\$sp)

addi \$sp,\$sp,-4

jr \$ra

nop

jr \$ra

#-----

Kích hoạt để lại dấu vết và dừng để lại dấu vết.

Các hàm chức năng:

```
CodeRight:      la $s5, HuongDi
                lw $s6, 0($s5) # $s6 is heading at now
                addi $s6, $s6, 90 # increase heading by
                sw $s6, 0($s5) # update HuongDi
                jal storePath
                jal ROTATE
                j Print

#Re trai 90 do
CodeLeft:       la $s5, HuongDi
                lw $s6, 0($s5) # $s6 is heading at now
                addi $s6, $s6, -90 # increase heading by
                sw $s6, 0($s5) # update HuongDi
                jal storePath
                jal ROTATE
                j Print

#-----
```

Marsbot đi rẽ phải và rẽ trái.

```
ROTATE:
    #backup
    addi $sp, $sp, 4
    sw $t1, 0($sp)
    addi $sp, $sp, 4
    sw $t2, 0($sp)
    addi $sp, $sp, 4
    sw $t3, 0($sp)
    #processing
    li $t1, HEADING # change HEADING port
    la $t2, HuongDi
    lw $t3, 0($t2) # $t3 is heading at now
    sw $t3, 0($t1) # to rotate robot
    #restore
    lw $t3, 0($sp)
    addi $sp, $sp, -4
    lw $t2, 0($sp)
    addi $sp, $sp, -4
    lw $t1, 0($sp)
    addi $sp, $sp, -4

    jr $ra
    nop
    jr $ra
```


Các hàm chức năng

```
CodeReturn:    la $s7, Path
               la $s5, PathLong
               lw $s5, 0($s5)
               add $s7, $s7, $s5
begin:         addi $s5, $s5, -12      #lui lai 1 structure

               addi $s7, $s7, -12      #vi tri cua thong tin ve canh cuoi cung
               lw $s6, 8($s7)          #huong cua canh cuoi cung
               addi $s6, $s6, 180      #nguoc lai huong cua canh cuoi cung
               #sub $s6, $zero, $s6

               la $t8, HuongDi #marshot quay nguoc lai
               sw $s6, 0($t8)
               jal ROTATE

Go_to_first_point_of_edge:
               lw $t9, 0($s7)          #toa do x cua diem dau tien cua canh
               li $t8, WHEREX          #toa do x hien tai
               lw $t8, 0($t8)

               bne $t8, $t9, Go_to_first_point_of_edge

               lw $t9, 4($s7)          #toa do y cua diem dau tien cua canh
               li $t8, WHEREY          #toa do y hien tai
               lw $t8, 0($t8)
```

Đi theo lộ trình ngược lại.

Giải thích code:

Phần ktext xử lý đầu vào:

```
get_cod:
    li $t1, IN_ADDRESS_HEX keyboard
    li $t2, OUT_ADDRESS_HEX keyboard
scan_row1:
    li $t3, 0x11
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_code_in_char
scan_row2:
    li $t3, 0x12
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_code_in_char
scan_row3:
    li $t3, 0x14
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_code_in_char
scan_row4:
    li $t3, 0x18
    sb $t3, 0($t1)
    lbu $a0, 0($t2)
    bnez $a0, get_code_in_char
```

```
get_code_in_char:
    beq $a0, KEY_0, case_0
    beq $a0, KEY_1, case_1
    beq $a0, KEY_2, case_2
    beq $a0, KEY_3, case_3
    beq $a0, KEY_4, case_4
    beq $a0, KEY_5, case_5
    beq $a0, KEY_6, case_6
    beq $a0, KEY_7, case_7
    beq $a0, KEY_8, case_8
    beq $a0, KEY_9, case_9
    beq $a0, KEY_a, case_a
    beq $a0, KEY_b, case_b
    beq $a0, KEY_c, case_c
    beq $a0, KEY_d, case_d
    beq $a0, KEY_e, case_e
    beq $a0, KEY_f, case_f

    # $s0 store code in char type
```

Quét các ma trận bàn phím và giải mã
xem phím nào được nhấn

Phần ktext xử lý đầu vào

```
case_0: li $s0, '0'
        j store_code
case_1: li $s0, '1'
        j store_code
case_2: li $s0, '2'
        j store_code
case_3: li $s0, '3'
        j store_code
case_4: li $s0, '4'
        j store_code
case_5: li $s0, '5'
        j store_code
case_6: li $s0, '6'
        j store_code
case_7: li $s0, '7'
        j store_code
case_8: li $s0, '8'
        j store_code
case_9: li $s0, '9'
        j store_code
case_a: li $s0, 'a'
        j store_code
case_b: li $s0, 'b'
        j store_code
case_c: li $s0, 'c'
        j store_code
```

Xử lý các phím được nhấn.

```
store_code:
    la $s1, InputCode
    la $s2, CodeLong
    lw $s3, 0($s2)                                # $s3 = strlen(InputCode)
    addi $t4, $t4, -1                             # $t4 = i
    for_loop_to_store_code:
        addi $t4, $t4, 1
        bne $t4, $s3, for_loop_to_store_code
        add $s1, $s1, $t4                         # $s1 = InputCode + $t4
        sb $s0, 0($s1)                            # InputCode[i] = $s0

        addi $s0, $zero, '\n'                    # add '\n' character
        addi $s1, $s1, 1                          # add '\n' character
        sb $s0, 0($s1)                            # add '\n' character

        addi $s3, $s3, 1
        sw $s3, 0($s2)                            # update length of InputCode

next_pc:
    mfc0 $at, $14 # $at <= Coproc0.$14 = Coproc0.epc
    addi $at, $at, 4 # $at = $at + 4 (next instruction)
    mtc0 $at, $14 # Coproc0.$14 = Coproc0.epc <= $at
#-----
```

Lưu trữ ký tự được đọc từ bàn phím vào InputCode, cập nhật độ dài của chuỗi.

Các bước thực hiện

Bước 1

- Nhập mã lệnh từ Digital Lab Sim

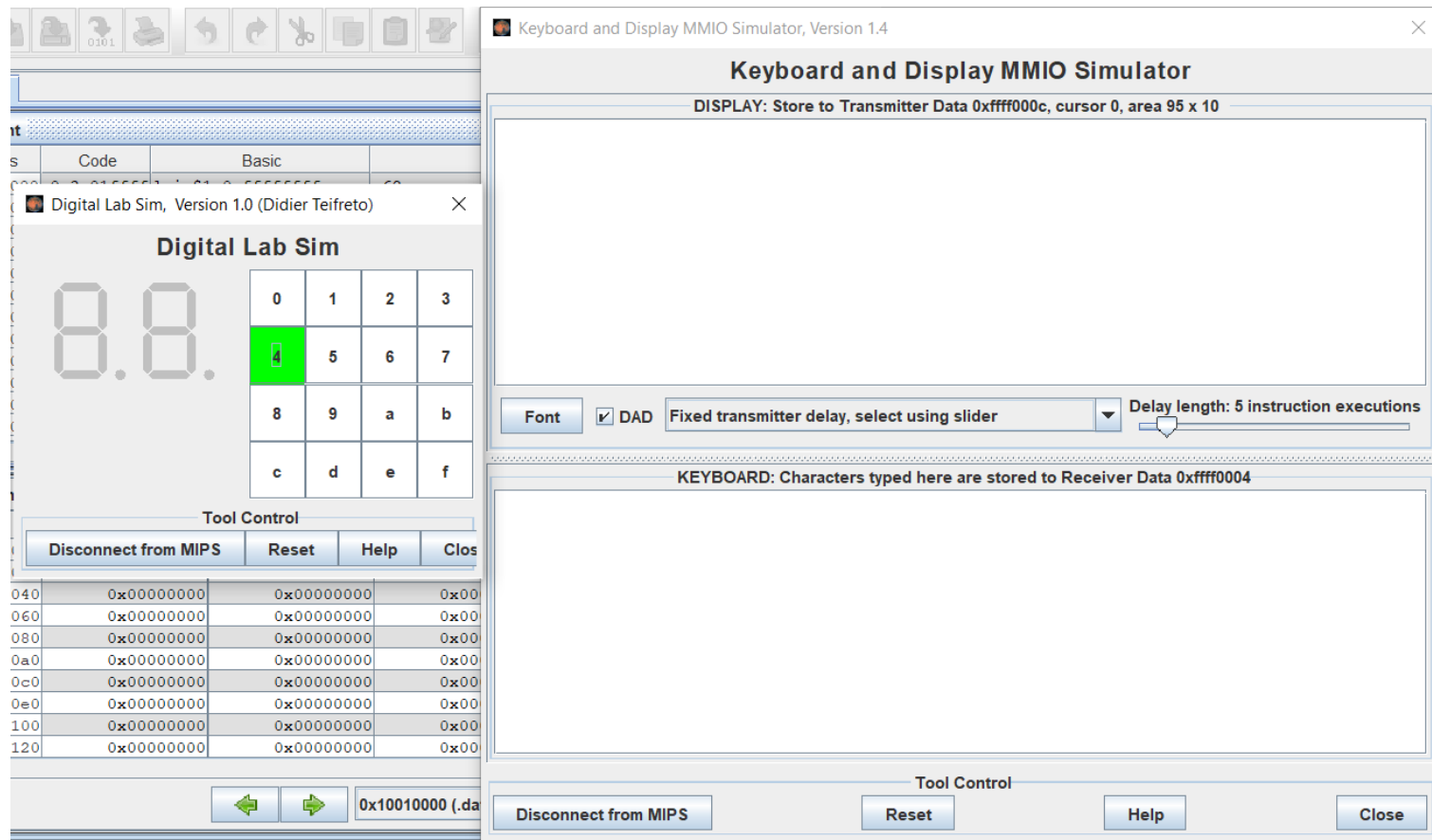
Bước 2

- Chờ nhận tín hiệu phím từ bàn phím vào Keyboard & Display MMIO Simulator

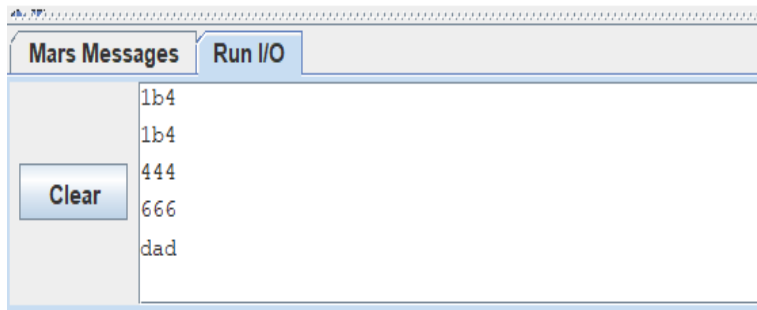
Bước 3

- Ấn Enter và đến bước thực thi lệnh, Marsbot sẽ thực hiện theo yêu cầu sau đó sẽ in ra mã lệnh đã nhập xuống màn hình hiển thị.
- Nếu báo lỗi xong sẽ thực hiện xóa hết mã lệnh đang chứa trong CodeInput để nhận lệnh tiếp theo và sau khi thực thi lệnh xong và in lệnh ra sẽ thực hiện xóa hết lệnh đang chứa trong CodeInput để chờ lệnh tiếp theo.

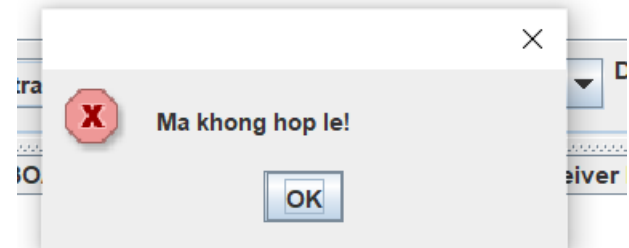
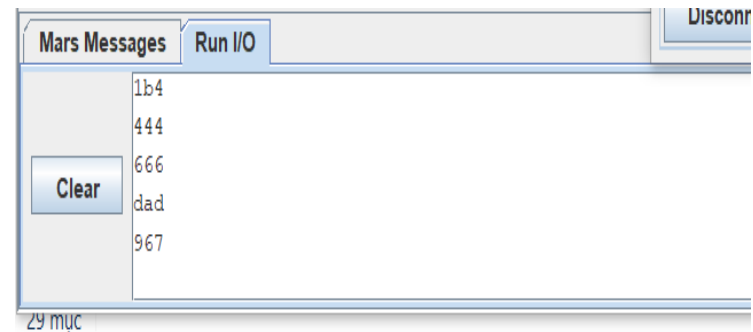
- Nhập mã lệnh từ Digital Lab Sim sau đó đọc key nhận được từ Keyboard & Display MMIO Simulator.



- Khi nhập vào mã điều khiển hợp lệ:

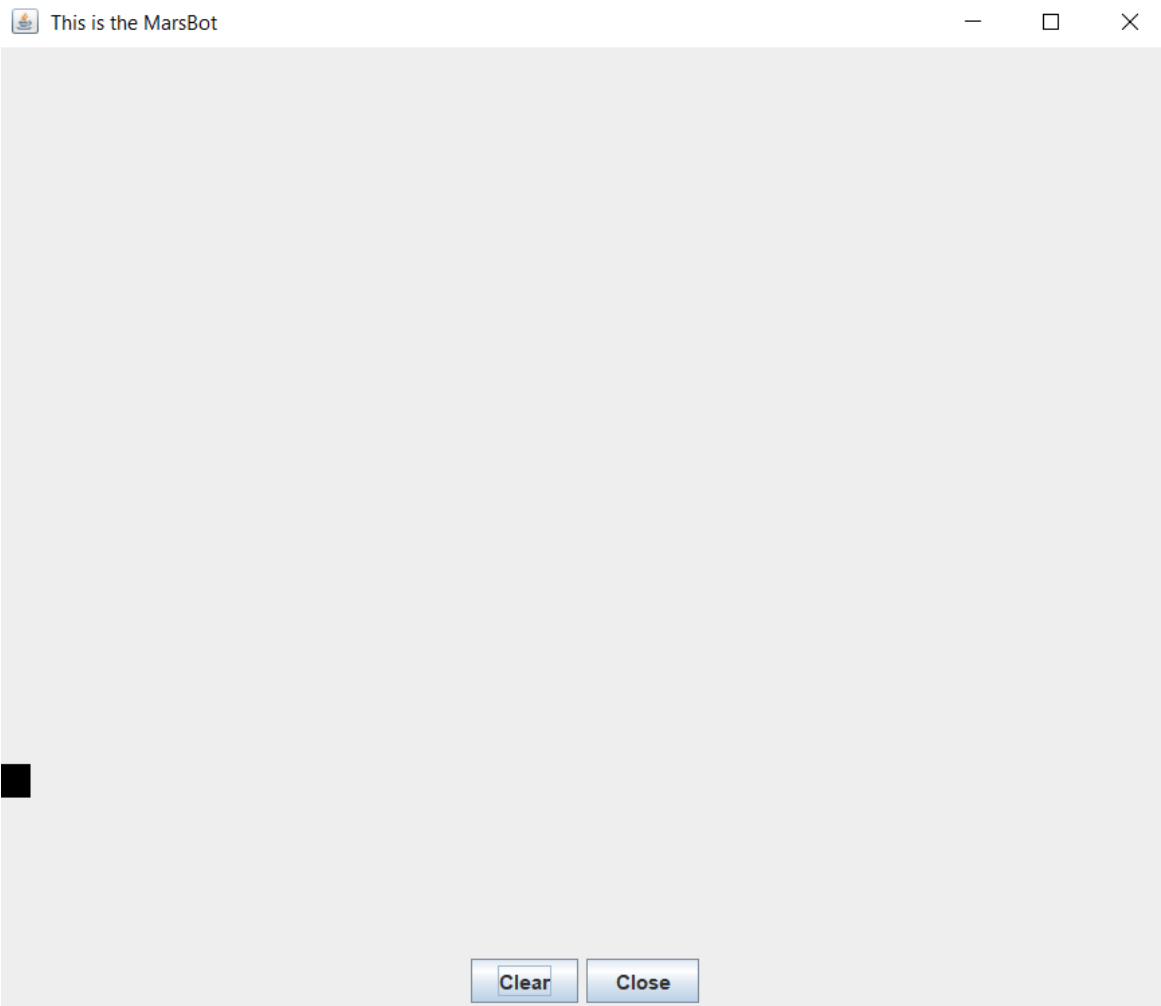


- Khi nhập vào mã điều khiển không hợp lệ:



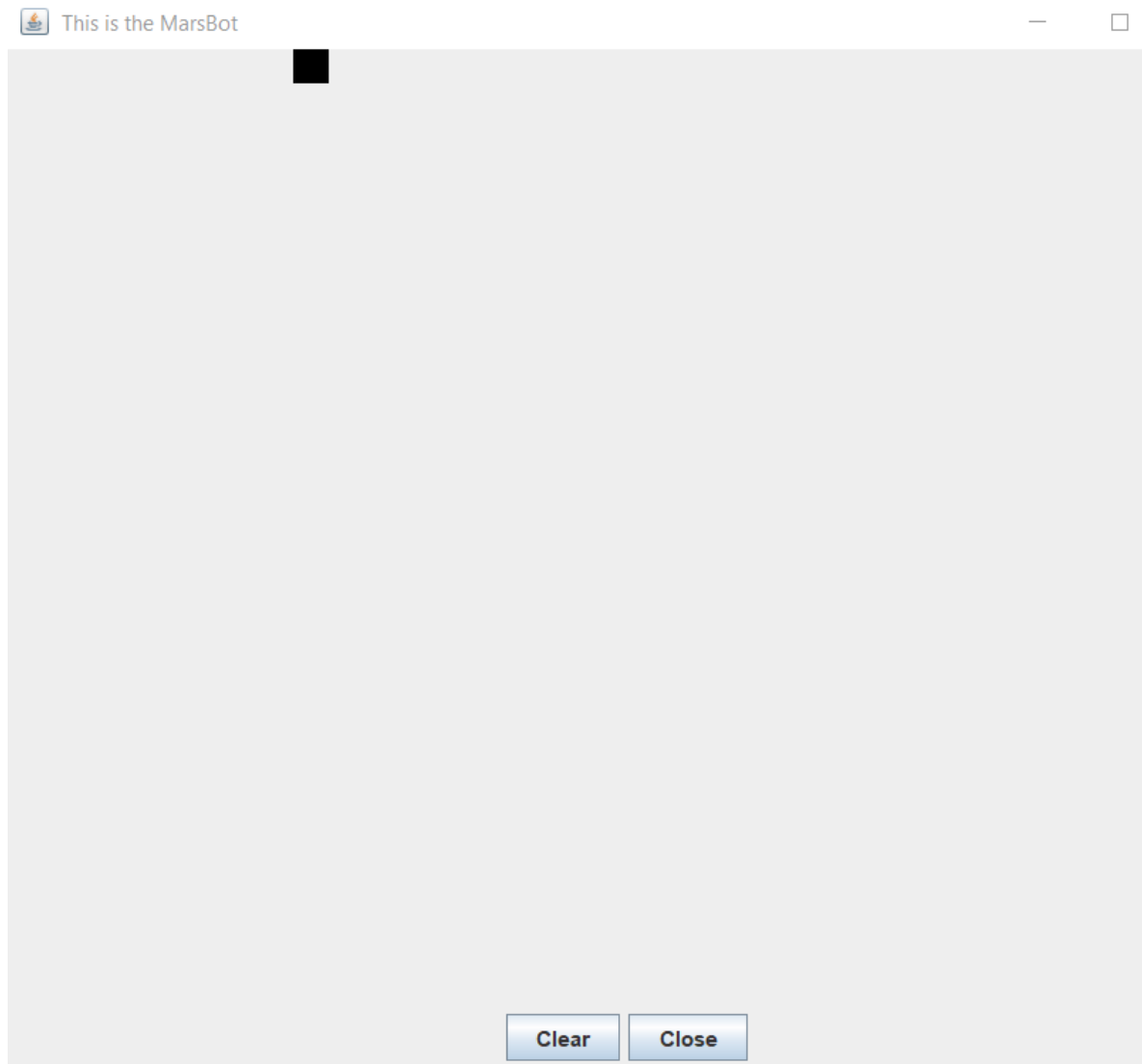
HUST

Demo



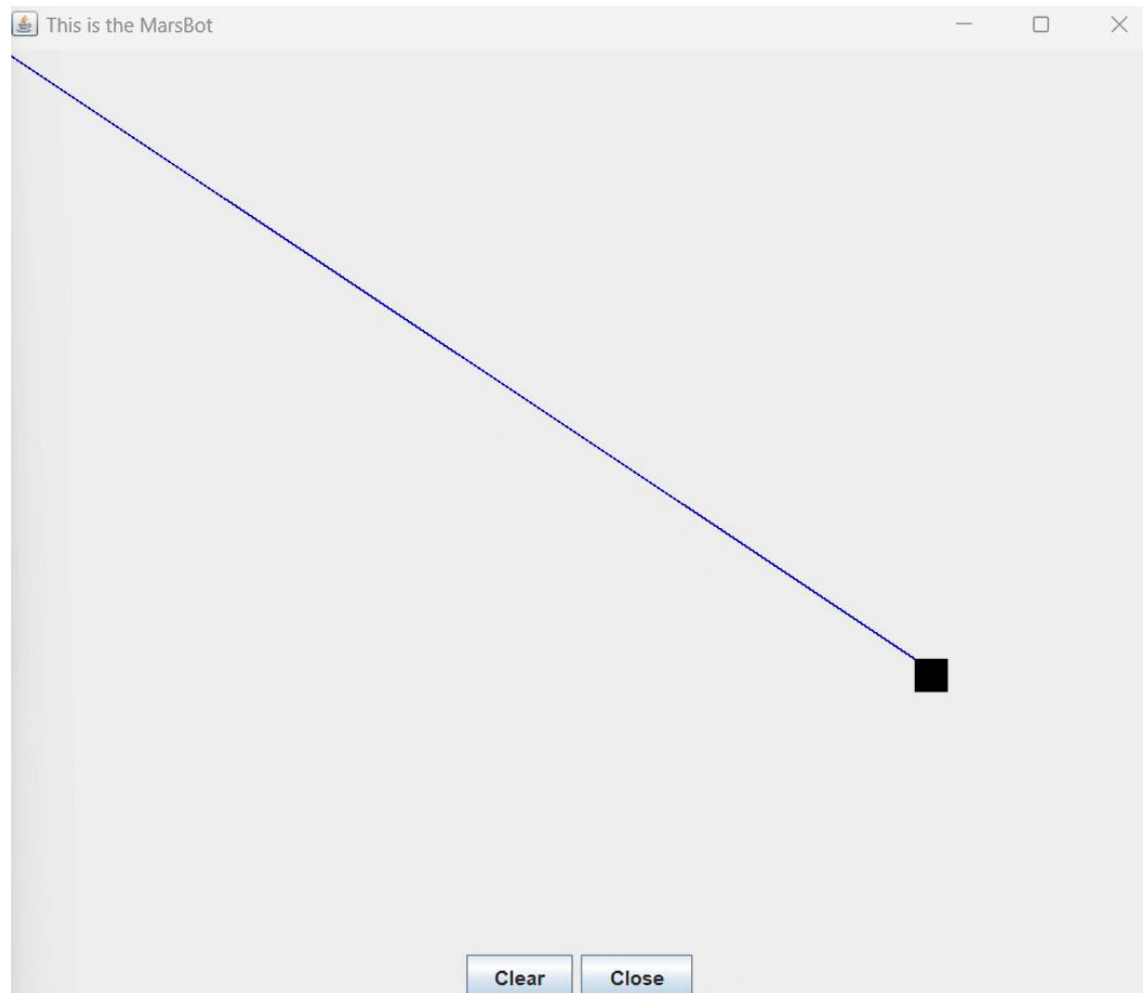
HUST

Demo



HUST

Demo





ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

10. PHÂN TÍCH THUẬT TOÁN BLOCK REPLACEMENT TRONG BỘ NHỚ ĐÊM

Giảng viên hướng dẫn: TS. Đỗ Công Thuận

Sinh viên thực hiện: Bùi Thị Xuân - 20225957

ONE LOVE. ONE FUTURE.

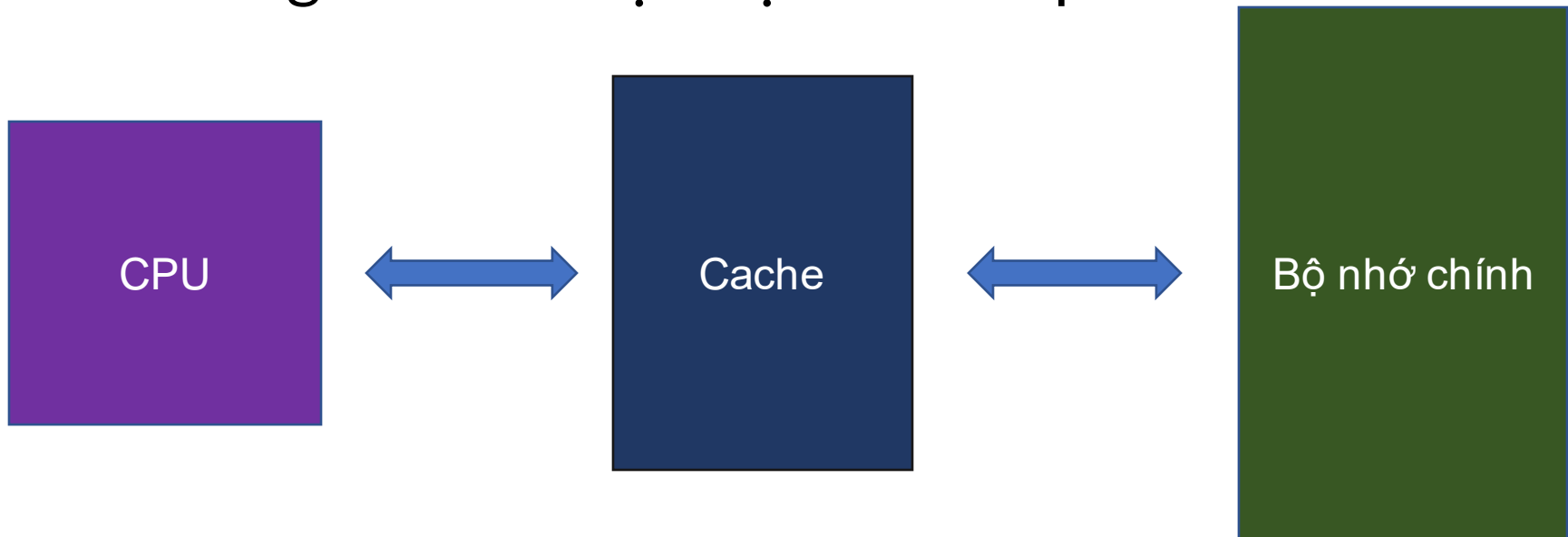
NỘI DUNG

1. Giới thiệu
2. Thuật toán Random
 - a. Cơ sở lý thuyết
 - b. Hiệu quả thuật toán
 - c. Phân tích ưu/nhược điểm
3. Thuật toán LRU
 - a. Cơ sở lý thuyết
 - b. Hiệu quả thuật toán
 - c. Phân tích ưu/nhược điểm
4. Kết luận

1. GIỚI THIỆU BỘ NHỚ ĐỆM

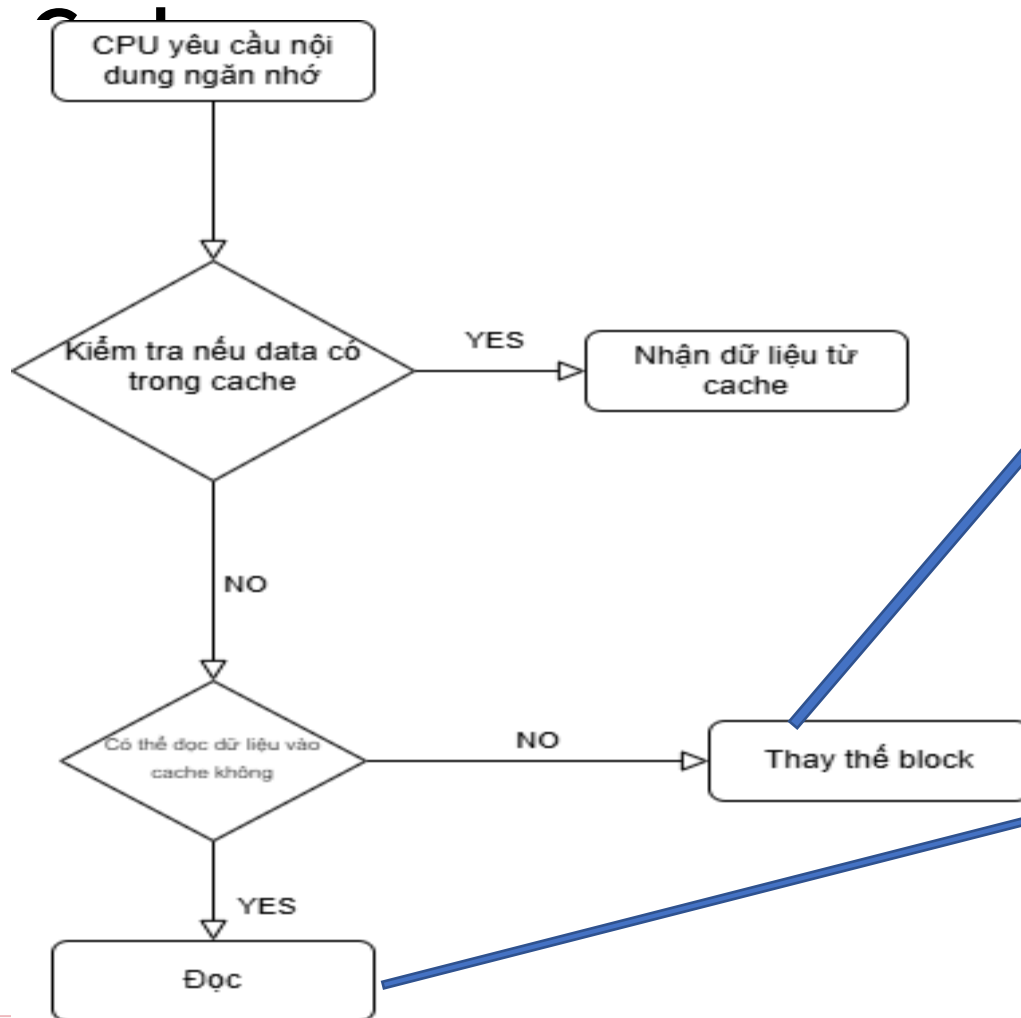
➤ Nguyên tắc chung của Cache:

- Cache có tốc độ nhanh hơn bộ nhớ chính
- Cache được đặt giữa CPU và bộ nhớ chính nhằm tăng tốc độ CPU truy cập bộ nhớ
- Cache cũng có thể được đặt trên chip CPU



1. GIỚI THIỆU BỘ NHỚ ĐỆM

➤ Thao tác của

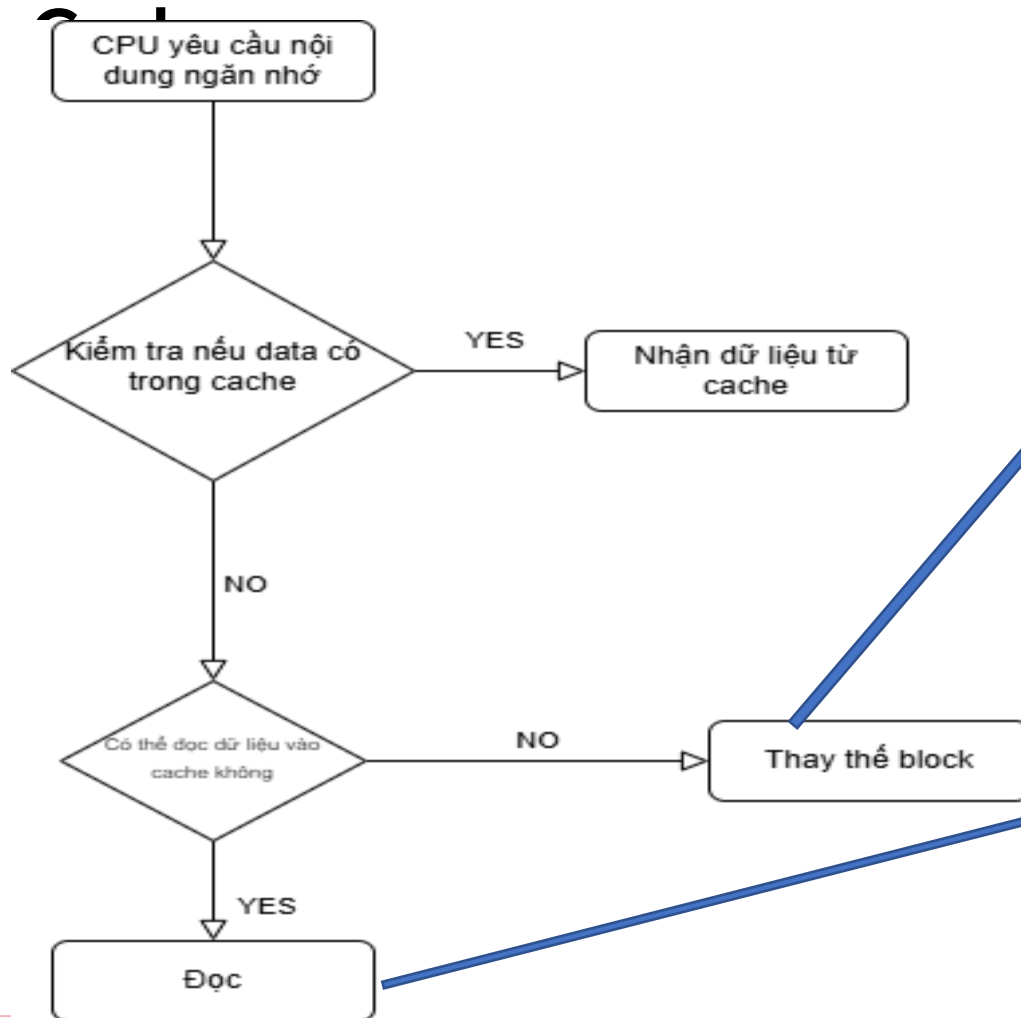


- Thuật toán Random
- Thuật toán LRU
- Thuật toán FIFO
- Thuật toán LFU

- Direct Mapping
- Fully associative mapping
- Set associative mapping

1. GIỚI THIỆU BỘ NHỚ ĐỆM

➤ Thao tác của



- Thuật toán Random

- Thuật toán LRU

- Thuật toán FIFO

- Thuật toán LFU

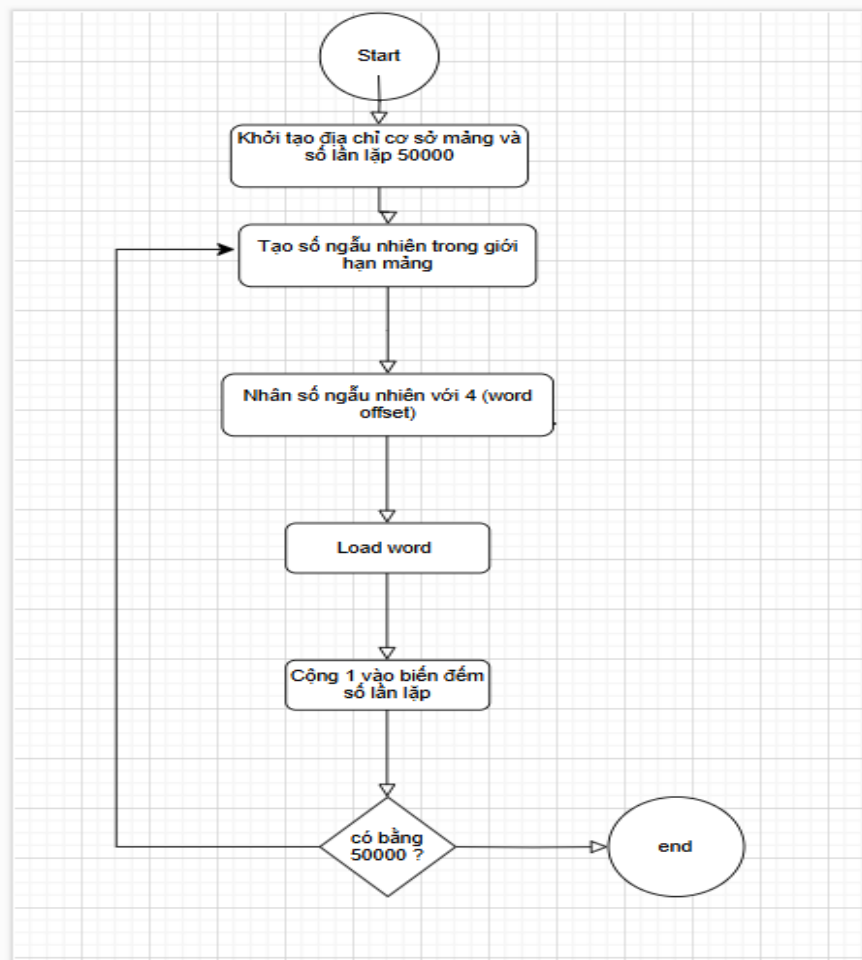
- Direct Mapping

- Fully associative mapping

- Set associative mapping

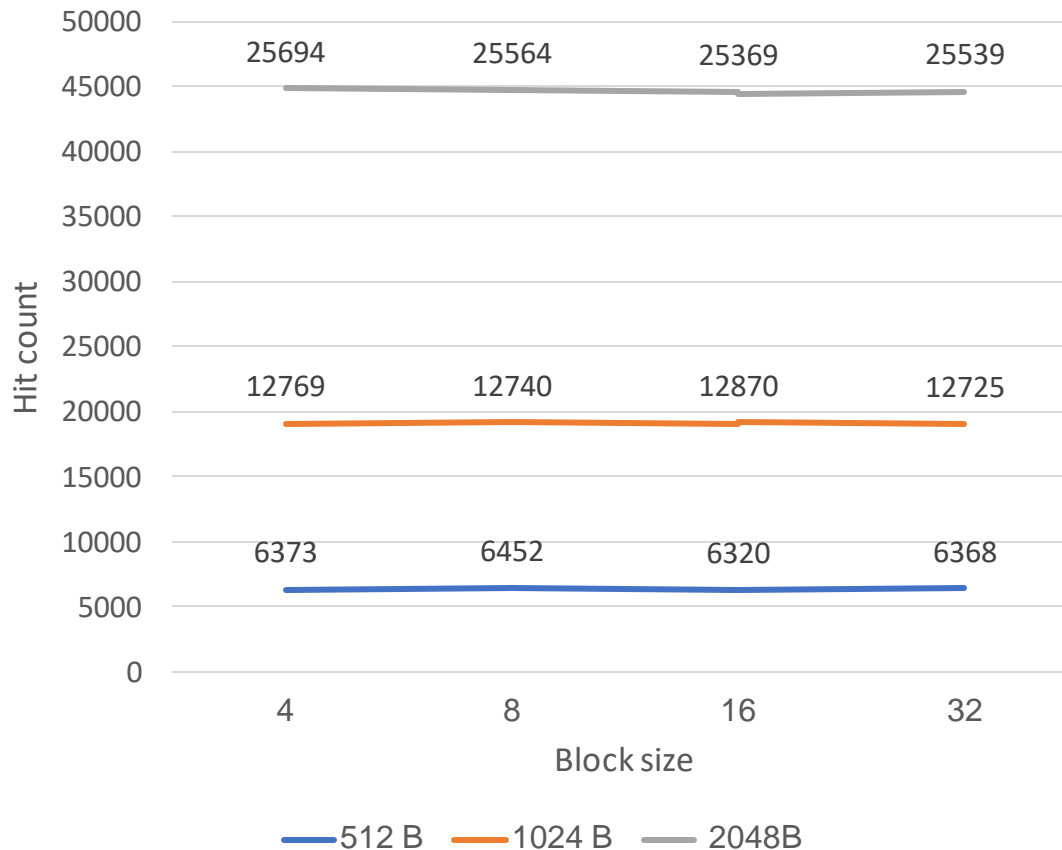
- **Nguyên lý:** khi cần thay thế một khối, thuật toán chọn khối ngẫu nhiên trong Cache để thay thế
- **Mô tả hoạt động của thuật toán:**
 - B1: Xác định số khối trong cache, mỗi khối đánh địa chỉ từ 0 đến $N-1$
 - B2: Sử dụng hàm sinh số ngẫu nhiên trong khoảng từ 0 đến $N-1$.
 - B3: Khối tương ứng với số ngẫu nhiên được tạo sẽ là khối được thay thế
 - B4: Thay thế dữ liệu trong khối đã chọn bằng dữ liệu mới

➤ Bài toán: Truy cập bộ nhớ có địa chỉ ngẫu nhiên



➤ Khi thay đổi Block Size

Bảng giá trị Cache hit count khi thay đổi Block size



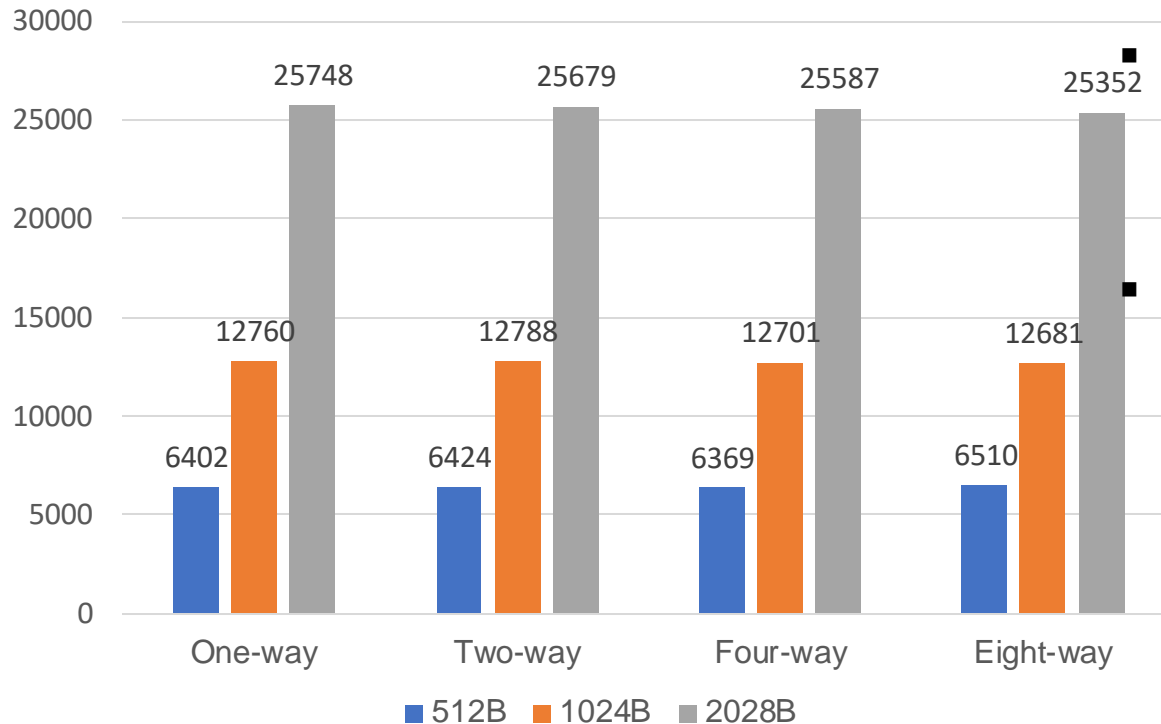
- **Ưu điểm:** Ban đầu, tăng kích thước khối sẽ tăng tỉ lệ cache hit vì các khối lớn hơn đưa vào nhiều dữ liệu liên kề, tận dụng tính cục bộ không gian.

- **Nhược điểm:** Nếu kích thước khối trở nên quá lớn so với tổng kích thước cache, số lượng khối mà cache có thể chứa sẽ giảm.

=> ít dữ liệu khác nhau được lưu trữ trong cache tại một thời điểm.

➤ Khi thay đổi Set Size

Bảng giá trị Cache hit count



- **Ưu điểm:** Giúp giảm tỉ lệ cache miss bằng cách cho phép các khối nhớ cạnh tranh ít hơn cho cùng một vị trí.

- **Nhược điểm:** Tăng Set Size làm tăng chi phí sản xuất và làm chậm thời gian truy cập Cache.

2. PHÂN TÍCH ƯU/ NHƯỢC ĐIỂM THUẬT TOÁN

➤ Ưu điểm:

- ❑ **Tính đơn giản:** Thuật toán Random dễ triển khai và không đòi hỏi tài nguyên tính toán nhiều
- ❑ **Tốc độ nhanh:** Quyết định thay thế được thực hiện nhanh chóng

➤ Nhược điểm:

- ❑ **Không tối ưu:** Vì việc thay thế hoàn toàn ngẫu nhiên, thuật toán có thể thay thế một khối quan trọng hoặc vừa được sử dụng gần đây, dẫn đến hiệu suất thấp hơn

2. THUẬT TOÁN LRU (Least Recently Used) Cơ sở lý thuyết

- **Nguyên lý:** Khi cần thay thế một khối, thay thế khối ở trong Set tương ứng có thời gian lâu nhất không được tham chiếu.
- **Mô tả hoạt động của thuật toán:**
 - B1: Tìm khối dữ liệu có thời gian truy cập cũ nhất
 - B2: Thay thế khối đó bằng khối dữ liệu mới
 - B3: Cập nhật thời gian truy cập của khối mới

2. THUẬT TOÁN LRU (Least Recently Used) Cơ sở lý thuyết

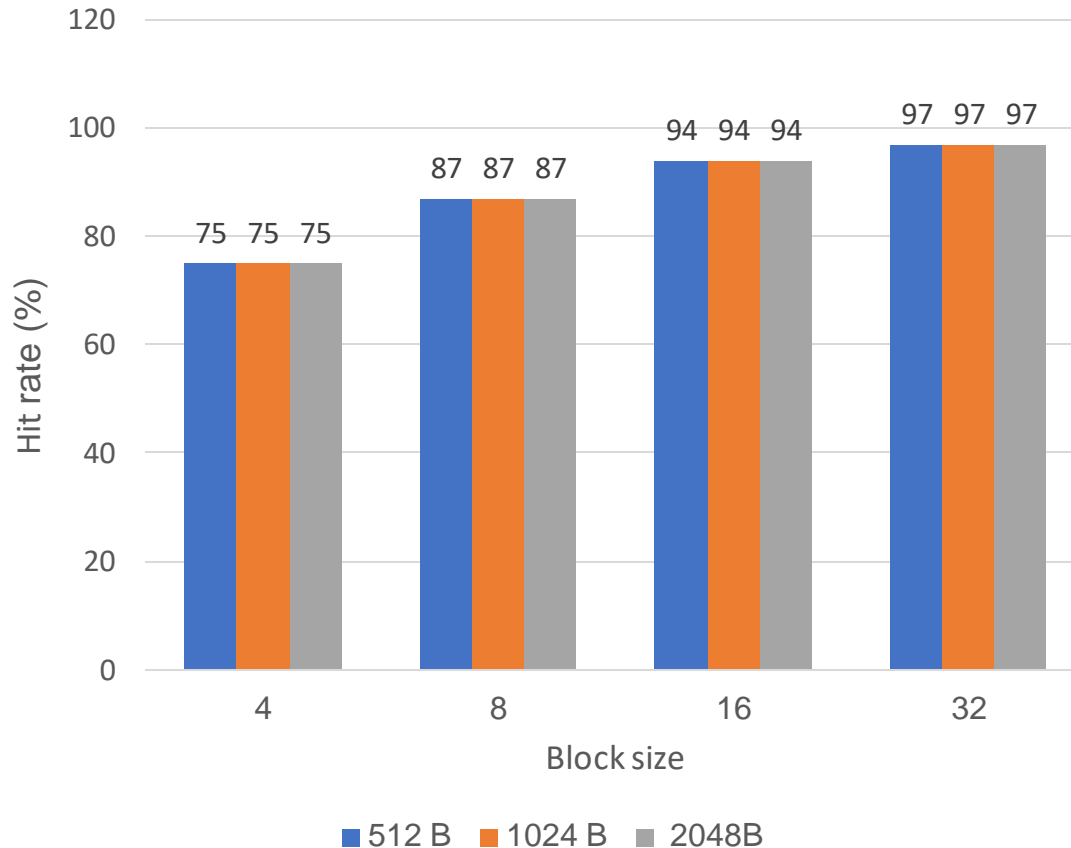
➤ **Ví dụ minh họa:** Cache có phương pháp tổ chức bộ nhớ ánh xạ liên kết 2 đường có 2 Set, sử dụng thuật

Địa chỉ của khối được tham chiếu	Hit or miss	Nội dung của cache blocks sau khi tham chiếu			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

✓ **Nhận xét:** Khi khối 6 được tham chiếu, khối 8 bị thay thế vì được tham chiếu ít gần đây hơn so với khối có địa chỉ 0.

➤ Khi thay đổi Block Size

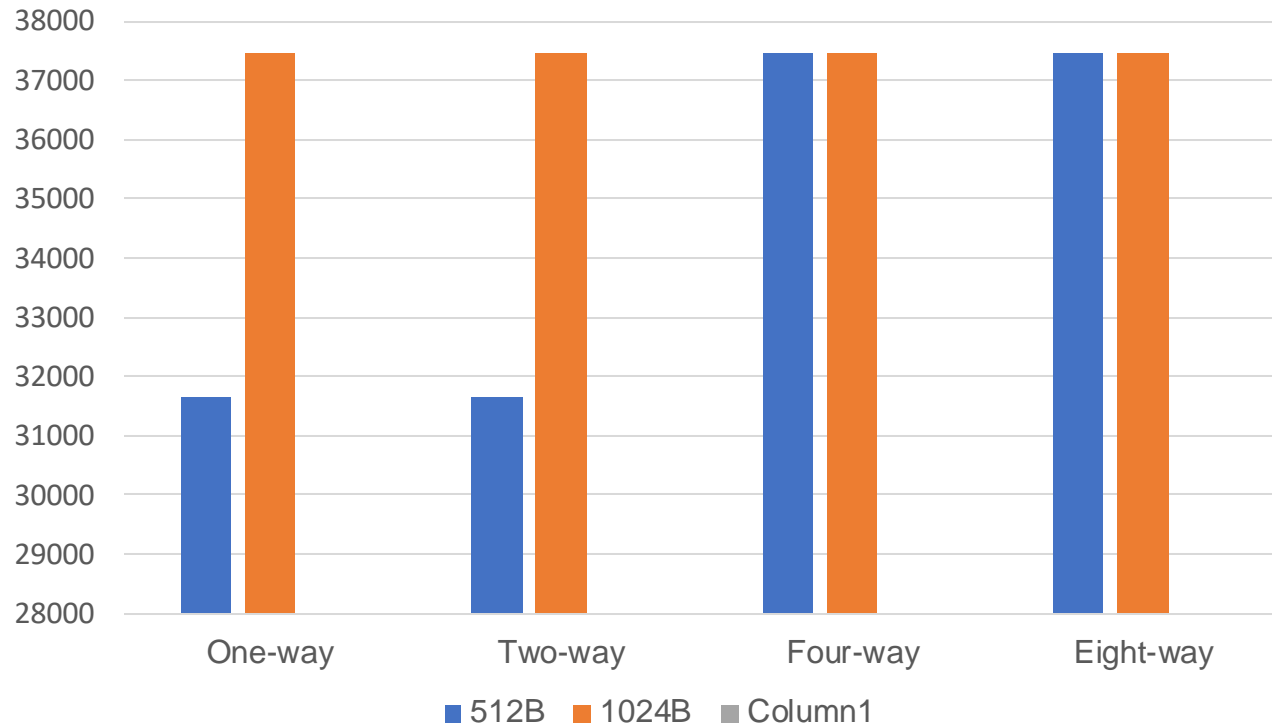
Bảng giá trị Hit rate khi thay đổi Block size



✓ **Nhận xét:** Kích thước khối tăng làm tỉ lệ hit rate tăng. Các khối lớn hơn đưa vào nhiều dữ liệu liền kề, tận dụng tính cục bộ không gian. Điều này có nghĩa là nếu một từ trong khối được truy cập, khả năng các từ khác trong cùng khối cũng sẽ được truy cập sớm, tăng tỷ lệ cache hit.

➤ Khi thay đổi Set Size

Bảng giá trị Cache hit count



- **Ưu điểm:** Giúp giảm tỉ lệ cache miss bằng cách cho phép các khối nhớ cạnh tranh ít hơn cho cùng một vị trí.
- **Nhược điểm:** Tăng Set Size làm tăng chi phí sản xuất và làm chậm thời gian truy cập Cache do tăng các phép so sánh và tính toán.

3. PHÂN TÍCH ƯU/ NHƯỢC ĐIỂM THUẬT TOÁN

➤ Ưu điểm:

- ❑ **Hiệu suất tốt:** Với các bài toán có các dữ liệu được truy cập gần đây sẽ được truy cập lại trong tương lai gần, thuật toán có thể giảm thiểu số lượng cache miss, do đó có hiệu suất tốt.

3. PHÂN TÍCH ƯU/ NHƯỢC ĐIỂM THUẬT TOÁN

➤ **Nhược**

điểm:

- ☐ **Chi phí phần cứng cao:** LRU yêu cầu lưu trữ thời gian tham chiếu cho mỗi khối cache. Việc cần nhiều bit hoặc bộ nhớ để lưu trữ thông tin này có thể tăng chi phí phần cứng của hệ thống.
- ☐ **Khó khăn khi triển khai với Cache có tổ chức bộ nhớ ánh xạ liên kết cao**
- ☐ **Không hiệu quả với mô hình sử dụng phân phối đồng đều**

3. KẾT LUẬN

➤ So sánh và đánh giá chung:

- ❑ **Hiệu quả:** LRU thường cho hiệu quả tốt hơn so với Random block replacement do khả năng tận dụng thông tin lịch sử truy cập. Tuy nhiên, điều này đi kèm với chi phí quản lý cao hơn.
- ❑ **Ứng dụng thực tế:** LRU thích hợp cho các hệ thống có mẫu truy cập có tính lặp lại cao, trong khi Random block replacement có thể được sử dụng trong các hệ thống không yêu cầu hiệu suất cao hoặc có các mẫu truy cập không thể dự đoán.

A large, stylized graphic on the left side of the slide. It consists of a red background with a circular pattern of white dots of varying sizes, creating a sense of depth and movement. The word "HUST" is written in white, bold, sans-serif capital letters in the center of this graphic.

HUST

THANK YOU !