ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CNTT & TRUYỀN THÔNG

BÁO CÁO FINAL PROJECT
THỰC HÀNH
KIẾN TRÚC MÁY TÍNH

Giáo viên hướng dẫn: Đỗ Công Thuần
Sinh viên thực hiện:
Nguyễn Thị Hạ - 20225622
Bùi Thị Xuân – 20225957
Lớp: 147796 – HEDSPI - Kỳ: 2023.2

Nhóm 13

Project 01. Curiosity Marsbot
Sinh viên thực hiện: Nguyễn Thị Hạ

Project 10. Phân tích thuật toán block replacement trong bộ nhớ đệm
Sinh viên thực hiện: Bùi Thị Xuân

1. Curiosity Marsbot

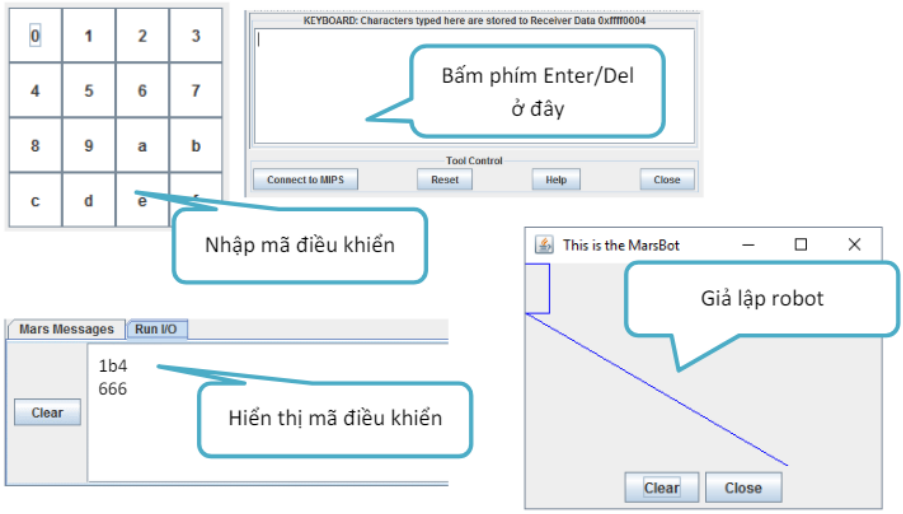
Xe tự hành Curioisity Marsbot chạy trên sao Hỏa, được vận hành từ xa bởi các lập trình viên trên Trái Đất. Bằng cách gửi đi các mã điều khiển từ một bàn phím ma trận, lập trình viên điều khiển quá trình di chuyển của Marbot như sau:

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. ***Gợi ý:*** *Marsbot được lập trình để nhớ lại toàn bộ lịch sử các mã điều khiển và khoảng thời gian giữa các lần đổi mã. Vì vậy, nó có thể đảo ngược lại lộ trình để quay về điểm xuất phát.*

Khi khởi động chương trình thì Curioisity Marsbot sẽ tự động di chuyển theo một quỹ đạo (tuỳ chọn) rồi dừng lại chờ nhận mã điều khiển. Sau khi nhận mã điều khiển, Curioisity 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

Hãy lập trình để Marsbot có thể hoạt động như đã mô tả. Đồng thời 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.



***Mã nguồn:**

```
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012
.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014
.eqv KEY_CODE 0xFFFF0004          # ASCII code from keyboard, 1 byte
.eqv KEY_READY 0xFFFF0000    # =1 if has a new keycode ?
                #Auto clear after lw

.eqv HEADING 0xffff8010           # Integer: An angle between 0 and 359
                # 0 : North (up)
                # 90: East (right)
                # 180: South (down)
                # 270: West (left)
.eqv MOVING 0xffff8050            # Boolean: whether or not to move
.eqv LEAVETRACK 0xffff8020   # Boolean (0 or non-0):
                # whether or not to leave a track
.eqv WHEREX 0xffff8030            # Integer: Current x-location of
MarsBot
.eqv WHEREY 0xffff8040            # Integer: Current y-location of
MarsBot



# Key value
#0-3
      .eqv KEY_0 0x11
      .eqv KEY_1 0x21
      .eqv KEY_2 0x41
      .eqv KEY_3 0x81
#4-7
      .eqv KEY_4 0x12
      .eqv KEY_5 0x22
      .eqv KEY_6 0x42
      .eqv KEY_7 0x82
```

```
#8-b
      .eqv KEY_8 0x14
      .eqv KEY_9 0x24
      .eqv KEY_a 0x44
      .eqv KEY_b 0x84
#c-f
      .eqv KEY_c 0x18
      .eqv KEY_d 0x28
      .eqv KEY_e 0x48
      .eqv KEY_f 0x88

#------------------------------------------------------------------------
------
.data
       #Funtion code
      ChuyenDong: .asciiz "1b4" # command code
      DungLai: .asciiz "c68"
      ReTrai: .asciiz "444"
       RePhai: .asciiz "666"
      DeVet: .asciiz "dad"
      DungDeVet: .asciiz "cbc"
      DiNguoc: .asciiz "999"
      MaLoi:  .asciiz "Ma khong hop le!"

      going: .word 0
      tracking: .word 0

      InputCode: .space 8
      CodeLong: .word 0
      InputCode1: .space 8
      #chia Path thanh:
```

```
      toado_x: .word 0 : 16
      toado_y: .word 0 : 16
      huongdi: .word 0 : 16
      PathLong: .word 4
      a_current: .word 0

.text
main:
li $k0, KEY_CODE
      li $k1, KEY_READY
#-----------------------------------------------------------
# Enable the interrupt of Keyboard matrix 4x4 of Digital Lab Sim
#-----------------------------------------------------------
      li $t1, IN_ADRESS_HEXA_KEYBOARD
      li $t3, 0x80 # bit 7 = 1 to enable
      sb $t3, 0($t1)
#-----------------------------------------------------------

#cai dat vi tri ban dau
setting:

      # x = 0; y = 0; a = 90
      lw $t7, PathLong # PathLong += 4
      addi $t7, $zero, 4
      sw $t7, PathLong

      li $t7, 90
      sw $t7, a_current
      jal ROTATE
      nop
```

```
      sw $t7, huongdi # huongdi[0] = 90

      j waitForKey

#-------------------------------------------------------

Error:
      li $v0, 4
      la $a0, MaLoi
      syscall
#-------------------------------------------------------

Print:
      li $v0, 4
      la $a0, InputCode
      syscall
      j resetInput

resetInput:
      jal Delete
      nop

#-------------------------------------------------------

waitForKey:
      lw $t5, 0($k1)   # $t5 = [$k1] = KEY_READY
      beq $t5, $zero, waitForKey  # if $t5 == 0 -> Polling
      nop
      beq $t5, $zero, waitForKey

readKey:
```

```
	lw $t6, 0($k0)   # $t6 = [$k0] = KEY_CODE
	beq $t6, 0x8, resetInput  # if $t6 != 'ENTER' -> Polling

	bne $t6, 0x0a, waitForKey
	nop
	bne $t6, 0x0a, waitForKey

#Cac code chuc nang
#-------------------------------------------------------------------------
-------------------------
#Chuong trinh con thuc hien ma code vua nhan tu digital lab sim
checkInput:
	lw $s2, CodeLong   # CodeLong != 3 -> invalid code
	bne $s2, 3, Error

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

	la $s3, DungLai
	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
      jal Equal
      beq $t0, 1, CodeTrack

      la $s3, DungDeVet
      jal Equal
      beq $t0, 1, CodeUntrack

      la $s3, DiNguoc
      jal Equal
      beq $t0, 1, CodeReturn
      nop

      j Error

#-----------------------------------------
CodeGo:
      jal strcpy
      jal GO
      j Print
#-----------------------------------------
CodeStop:
      jal strcpy
      jal STOP
      j Print
#-----------------------------------------
CodeTrack:
      jal strcpy
      jal TRACK
      j Print
#-----------------------------------------
```

```
CodeUntrack:
      jal strcpy
      jal UNTRACK
      j Print

#-----------------------------------------
CodeRight:
      jal strcpy
      lw $t7, going
      lw $s0, tracking

      jal STOP
      nop
      jal UNTRACK
      nop

      la $s5, a_current
      lw $s6, 0($s5)  # $s6 is heading at now
      addi $s6, $s6, 90 # increase alpha by 90*
      sw $s6, 0($s5)  # update a_current

      jal storePath
      jal ROTATE

      beqz $s0, noTrack1
      nop
      jal TRACK
      noTrack1: nop

      beqz $t7, noGo1
      nop
```

```
      jal GO
noGo1:
      nop
      j Print

#-------------------------------------------
CodeLeft:
      jal strcpy
      lw $t7, going
      lw $s0, tracking

      jal STOP
      nop
      jal UNTRACK
      nop

      la $s5, a_current
      lw $s6, 0($s5)  # $s6 is heading at now
      addi $s6, $s6, -90 # decrease alpha by 90*
      sw $s6, 0($s5)  # update a_current

      jal storePath
      jal ROTATE

      beqz $s0, noTrack2
      nop
      jal TRACK
      noTrack2: nop

      beqz $t7, noGo2
      nop
```

```
      jal GO
noGo2:
      nop
      j Print

#-----------------------------------------
CodeReturn:
      jal strcpy
      li $t7, IN_ADRESS_HEXA_KEYBOARD # Disable interrupts when going
backward
      sb $zero, 0($t7)

      lw $s5, PathLong  # $s5 = CodeLong
      jal UNTRACK
      jal GO

begin:
      addi $s5, $s5, -4   # CodeLong--
      lw $s6, huongdi($s5)  # $s6 = huongdi[CodeLong]
      addi $s6, $s6, 180  # $s6 = the reverse direction of alpha
      sw $s6, a_current
      jal ROTATE
      nop

      lw $t9, toado_x($s5)  # $t9 = toado_x[i]
      lw $t7, toado_y($s5)  # $t9 = toado_y[i]

Go_to_first_point_of_edge:
      li $t8, WHEREX   # $t8 = x_current
      lw $t8, 0($t8)
```

```
      bne $t8, $t9, Go_to_first_point_of_edge # x_current == toado_x[i]
      nop
      bne $t8, $t9, Go_to_first_point_of_edge

      li $t8, WHEREY   # $t8 = y_current
      lw $t8, 0($t8)

      bne $t8, $t7, Go_to_first_point_of_edge # y_current == toado_y[i]
      nop
      bne $t8, $t7, Go_to_first_point_of_edge # y_current == toado_y[i]

      beq $s5, 0, finish # PathLong == 0
      nop    # -> end

      j begin  # else -> turn

finish:
      jal STOP
      sw $zero, a_current  # update heading
      jal ROTATE

      addi $s5, $zero, 4
      sw $s5, PathLong  # reset PathLong = 0

      j Print

#-------------------------------------------------------------
#luu lai vi tri hien tai va huong di cua marsbot
#-------------------------------------------------------------

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)

lw $s1, WHEREX   # s1 = x
lw $s2, WHEREY   # s2 = y
lw $s4, a_current  # s4 = a_current

lw $t3, PathLong  # $t3 = PathLong
sw $s1, toado_x($t3)  # store: x, y, alpha
sw $s2, toado_y($t3)
sw $s4, huongdi($t3)

addi $t3, $t3, 4   # update long
sw $t3, PathLong

#restore
```

```
      lw $s4, 0($sp)
      addi $sp, $sp, -4
      lw $s3, 0($sp)
      addi $sp, $sp, -4
      lw $s2, 0($sp)
      addi $sp, $sp, -4
      lw $s1, 0($sp)
      addi $sp, $sp, -4
      lw $t4, 0($sp)
      addi $sp, $sp, -4
      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

#=========================================================================
======
#Cac chuc nang cua Marsbot
#-------------------------------------------------------------------------
----
# GO procedure, to start running
# param[in] none
#-------------------------------------------------------------
GO:
        #backup
      addi $sp, $sp, 4
      sw $at, 0($sp)
      addi $sp, $sp, 4
```

```
      sw $k0, 0($sp)

      li $at, MOVING   # change MOVING port
      addi $k0, $zero, 1  # to logic 1,
      sb $k0, 0($at)   # to start running

      li $t7, 1   # going = 0
      sw $t7, going

      #restore
      lw $k0, 0($sp)
      addi $sp, $sp, -4
      lw $at, 0($sp)
      addi $sp, $sp, -4

      jr $ra
#-------------------------------------------------------------
# STOP procedure, to stop running
# param[in] none
#-------------------------------------------------------------
STOP:
        #backup
      addi $sp, $sp, 4
      sw $at, 0($sp)

      li $at, MOVING   # change MOVING port to 0
      sb $zero, 0($at)  # to stop

      sw $zero, going  # going = 0

      #restore
```

```
      lw $at, 0($sp)
      addi $sp, $sp, -4

      jr $ra

#-------------------------------------------------------------
# TRACK procedure, to start drawing line
# param[in] none
#-------------------------------------------------------------
TRACK:
        #backup
      addi $sp, $sp, 4
      sw $at, 0($sp)
      addi $sp, $sp, 4
      sw $k0, 0($sp)

      li $at, LEAVETRACK  # change LEAVETRACK port
      addi $k0, $zero,1  # to logic 1,
      sb $k0, 0($at)   # to start tracking

      addi $s0, $zero, 1
      sw $s0, tracking

      #restore
      lw $k0, 0($sp)
      addi $sp, $sp, -4
      lw $at, 0($sp)
      addi $sp, $sp, -4

      jr $ra
```

```
#-----------------------------------------------------------
# UNTRACK procedure, to stop drawing line
# param[in] none
#-----------------------------------------------------------
UNTRACK:
        #restore
      addi $sp, $sp, 4
      sw $at, 0($sp)

      li $at, LEAVETRACK # change LEAVETRACK port to 0
      sb $zero, 0($at) # to stop drawing tail

      sw $zero, tracking

      #backup
      lw $at, 0($sp)
      addi $sp, $sp, -4

      jr $ra

#-----------------------------------------------------------
# ROTATE_RIGHT procedure, to control robot to rotate
# param[in] HuongDi variable, store heading at present
#-----------------------------------------------------------
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)

	li $t1, HEADING # change HEADING port
	la $t2, a_current
	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



#-------------------------------------------------------------
#Code check xem ma code nhan duoc va ma code theo dau bai cho co dung format hay khong
#Luc nay $s3 se chua dia chi co cac code chuc nang theo format da cho
#$t0 la output neu code nhan vao dung format se tra ve 1, nguoc lai la 0
Equal:
	addi $sp, $sp, 4   # back up
	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)

      xor $t0, $zero, $zero  # $t0 = return value = 0
      xor $t1, $zero, $zero  # $t1 = i = 0

Equal_loop:
      beq $t1, 3, Equal_equal  # if i = 3 -> end loop -> equal
      nop

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

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

      beq $t2, $t3, Equal_next  # if $t2 == $t3 -> continue the loop
      nop

      j Equal_end

Equal_next:
      addi $t1, $t1, 1
      j Equal_loop

Equal_equal:
      add $t0, $zero, 1  # i++

Equal_end:
        #restock
      lw $t3, 0($sp)
      addi $sp, $sp, -4
      lw $t2, 0($sp)
```

```
      addi $sp, $sp, -4
      lw $s1, 0($sp)
      addi $sp, $sp, -4
      lw $t1, 0($sp)
      addi $sp, $sp, -4

      jr $ra

#-------------------------------------------------------------
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
      lw $t3, CodeLong   # $t3 = CodeLong
      addi $t1, $zero, -1  # $t1 = -1 = i

Delete_loop:
      addi $t1, $t1, 1   # i++
      sb $zero, InputCode  # InputCode[i] = '\0'

      bne $t1, $t3, Delete_loop # if $t1 <=3 resetInput loop
```

```
      nop

      sw $zero, CodeLong  # reset CodeLong = 0

Delete_end:
        #restore
      lw $s2, 0($sp)
      addi $sp, $sp, -4
      lw $t3, 0($sp)
      addi $sp, $sp, -4
      lw $s1, 0($sp)
      addi $sp, $sp, -4
      lw $t2, 0($sp)
      addi $sp, $sp, -4
      lw $t1, 0($sp)
      addi $sp, $sp, -4

      jr $ra

#-------------------------------------------------------------
strcpy:
        #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
	li $t2, 0
	la $s1, InputCode1
	la $s2, InputCode

strcpy_loop:
	beq $t2, 3, strcpy_end

	lb $t1, 0($s2)
	sb $t1, 0($s1)

	addi $s1, $s1, 1
	addi $s2, $s2, 1
	addi $t2, $t2, 1

	j strcpy_loop

strcpy_end:
	  #restore
	lw $s2, 0($sp)
	addi $sp, $sp, -4
	lw $t3, 0($sp)
	addi $sp, $sp, -4
	lw $s1, 0($sp)
	addi $sp, $sp, -4
	lw $t2, 0($sp)
	addi $sp, $sp, -4
	lw $t1, 0($sp)
	addi $sp, $sp, -4
```

```
      jr $ra

#-----------------------------------------------------------------------



#Nhan code input
#-----------------------------------------------------------------
#=================================================================================
======
# GENERAL INTERRUPT SERVED ROUTINE for all interrupts
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.ktext 0x80000180
#---------------------------------------------------------
# SAVE the current REG FILE to stack
#---------------------------------------------------------
backup:
      addi $sp,$sp,4
      sw $ra,0($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)
      addi $sp,$sp,4
      sw $a0,0($sp)
      addi $sp,$sp,4
      sw $at,0($sp)
      addi $sp,$sp,4
      sw $s0,0($sp)
```

```
      addi $sp,$sp,4
      sw $s1,0($sp)
      addi $sp,$sp,4
      sw $s2,0($sp)
      addi $sp,$sp,4
      sw $s4,0($sp)
      addi $sp,$sp,4
      sw $t4,0($sp)
      addi $sp,$sp,4
      sw $s3,0($sp)
#-------------------------------------------------------
# Processing
#-------------------------------------------------------
get_cod:
      li $t1, IN_ADRESS_HEXA_KEYBOARD
      li $t2, OUT_ADRESS_HEXA_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

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
case_d:
      li $s0, 'd'
      j store_code
case_e:
      li $s0, 'e'
      j store_code
case_f:
      li $s0, 'f'
      j store_code

store_code:
      la $s1, InputCode
      la $s2, CodeLong
      lw $s3, 0($s2)   # $s3 = strlen(InputCode)
      addi $t4, $t4, -1   # $t4 = i

store_code1:
      addi $t4, $t4, 1
      bne $t4, $s3, store_code1
      add $s1, $s1, $t4  # $s1 = InputCode + i
      sb $s0, 0($s1)   # InputCode[i] = $s0

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

      addi $s3, $s3, 1
      sw $s3, 0($s2)   # update CodeLong

#-------------------------------------------------------
```

```
# Evaluate the return address of main routine
# epc <= epc + 4
#-------------------------------------------------------------
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
#-------------------------------------------------------------
# RESTORE the REG FILE from STACK
#-------------------------------------------------------------
restore:
      lw $s3, 0($sp)
      addi $sp, $sp, -4
      lw $t4, 0($sp)
      addi $sp, $sp, -4
      lw $s2, 0($sp)
      addi $sp, $sp, -4
      lw $s1, 0($sp)
      addi $sp, $sp, -4
      lw $s0, 0($sp)
      addi $sp, $sp, -4
      lw $at, 0($sp)
      addi $sp, $sp, -4
      lw $a0, 0($sp)
      addi $sp, $sp, -4
      lw $t3, 0($sp)
      addi $sp, $sp, -4
      lw $t2, 0($sp)
      addi $sp, $sp, -4
      lw $t1, 0($sp)
      addi $sp, $sp, -4
```

```
      lw $ra, 0($sp)
      addi $sp, $sp, -4
return: eret # Return from exception
```

*Ý tưởng:

- Tạo chương trình nhận mã lệnh từ Digital Lab Sim sau đó được kich 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 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.

*Các bước thực hiện:

B1: Nhập mã lệnh từ Digital Lab Sim

B2: Chờ nhận tín hiệu phím từ bàn phím vào Keyboard & Display MMIO Simulator

+ Nếu nhấn phím Enter sẽ nhảy đến bước thực thi lệnh

+ Nếu nhấn phím Delete sẽ xóa hết mã lệnh đang nhập.

B3: Nếu 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 thực hiện 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.

*Phân tích:

1. Bộ nhớ:

- `IN_ADRESS_HEXA_KEYBOARD`: Địa chỉ để kích hoạt ngắt bàn phím.
- `OUR_ADRES_HEXA_KEYBOARD`: Địa chỉ để đọc dữ liệu phím đã nhấn từ bàn phím.
- `KEY_CODE`: Địa chỉ lưu trữ mã của phím đã nhấn gần đây nhất.
- `KEY_READY`: Địa chỉ cho biết phím có được nhấn (0) hay không (1).
- `HEADING`: Địa chỉ lưu trữ hướng di chuyển hiện tại của Marsbot (0: Bắc, 90: Đông, 180: Nam, 270: Tây).
- `MOVING`: Địa chỉ điều khiển chuyển động của Marsbot (0: Dừng, 1: Di chuyển).
- `LEAVETRACK`: Địa chỉ điều khiển việc đặt dấu vết khi di chuyển (0: Tắt, 1: Bật).
- `WHEREX`: Địa chỉ lưu trữ tọa độ X vị trí hiện tại của Marsbot.
- `WHEREY`: Địa chỉ lưu trữ tọa độ Y vị trí hiện tại của Marsbot.

2. Dữ liệu:

- Biến:

`+ChuyenDong,Dung,ReTrai,RePhai,DeVet,DungDeVet,DiNguoc` : Lưu trữ các chuỗi mã chức năng cho các lệnh di chuyển.

`+InputCode,InputCode1`: Mảng byte để lưu trữ mã do người dùng nhập.

`+CodeLong,CodeLong1`: Biến để lưu trữ độ dài mã do người dùng nhập.

`+HuongDi`: Biến để lưu trữ hướng di chuyển hiện tại của Marsbot.

`+Path`: Mảng byte để lưu trữ thông tin đường đi (bao gồm vị trí và hướng) của Marsbot.

Chia Path thành tọa độ x, y và hướng.

`+PathLong`: Biến để lưu trữ độ dài dữ liệu đường đi (tính bằng byte).

3. Chương trình chính:

- Hàm main:
  - Bật ngắt cho ma trận bàn phím để có thể nhận tín hiệu từ bàn phím.
  - Lặp liên tục để:
    - Chờ người dùng nhấn phím.
    - Đọc mã phím được nhấn.
    - Xử lý các phím đặc biệt (Xóa, Lặp lại).
    - Gọi hàm CheckInput để xử lý mã lệnh do người dùng nhập.
    - In mã lệnh đã nhập ra màn hình.
- Hàm `CheckInput`:
  - Lưu trữ thông tin đường đi hiện tại trước khi xử lý mã lệnh mới.
  - So sánh mã lệnh do người dùng nhập với các mã chức năng được định nghĩa sẵn.
  - Gọi hàm tương ứng với mã chức năng được nhập (`CodeGO,CodeStop,CodeLeft,CodeRight,…`) để thực hiện hành động mong muốn.
- Các hàm khác:
  - `GO`: Điều khiển Marsbot di chuyển về phía trước.
  - `STOP`: Dừng Marsbot di chuyển.
  - `TRACK`: Bật chế độ để lại dấu vết khi Marsbot di chuyển.
  - `UNTRACK`: Tắt chế độ để lại dấu vết.
  - `ROTATE`: Xoay Marsbot theo hướng trái hoặc phải.
  - `Print`: In ra lệnh vừa thực thi
  - `Error`: In thông báo lỗi khi mã lệnh do người dùng nhập không hợp lệ.
  - `Delete`: Xóa mã lệnh hiện tại khỏi bộ nhớ.
  - `storePath`: Lưu vị trí và hướng hiện tại của Marsbot vào mảng Path.

- `Equal:` So sánh mã lệnh do người dùng nhập với mã chức năng được định nghĩa sẵn.

4. Xử lý đầu vào bàn phím:

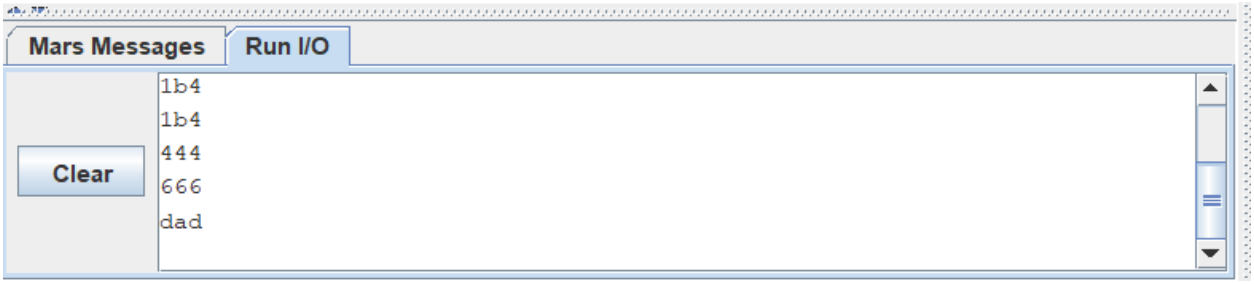
Người dùng có thể nhập các lệnh khác nhau để điều khiển Marsbot di chuyển về phía trước, dừng lại, xoay, để lại dấu vết khi di chuyển,… Mã cũng bao gồm chức năng xử lý lỗi để đảm bảo rằng người dùng nhập mã lệnh hợp lệ.

- Khi có phím được nhấn, mã sẽ quét từng hàng của bàn phím để xác định phím nào được nhấn.
- Mã sau đó sẽ đọc mã phím tương ứng với phím đã nhấn và thực hiện hành động phù hợp.

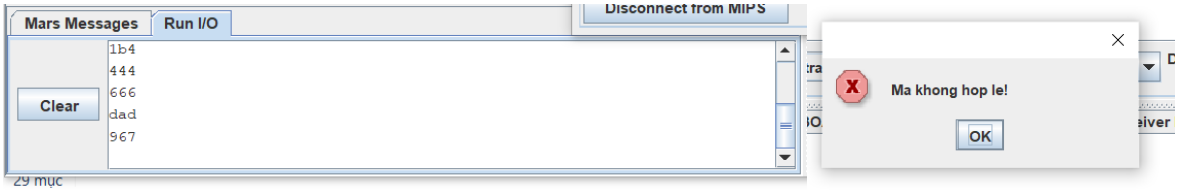
    - `get_code, get_code_in_char`: đọc keypad được nhập từ DigitalLabSim
    - `store_code:` Lưu kí tự vừa được nhập vào InputCode và tăng chiều dài CodeLong lên 1, thêm kí tự kết thúc chuỗi “\0” vào cuối chuỗi.

***Demo:**

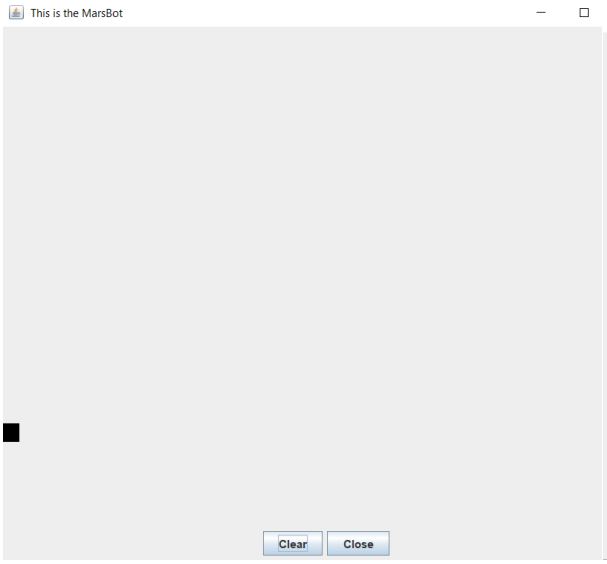
- Khi nhập vào các mã điều khiển hợp lệ sẽ in ra màn hình console:

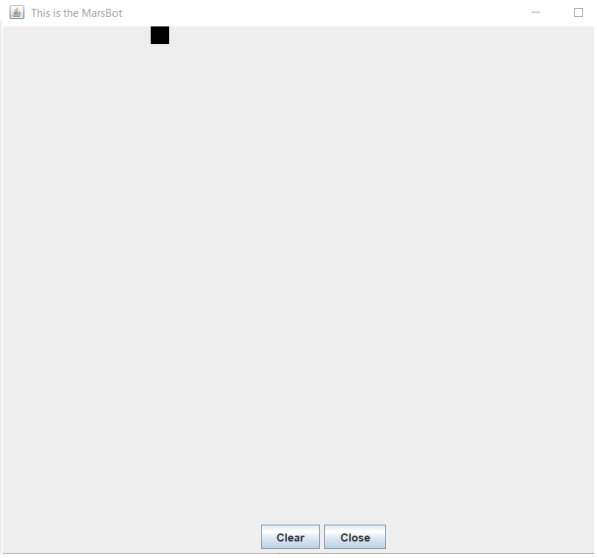


- Khi nhập mã không hợp lệ sẽ in ra màn hình và thông báo “Ma khong hop le!”:



- Marbot thực thi:


This is the MarsBot
Clear
Close


This is the MarsBot
Clear
Close

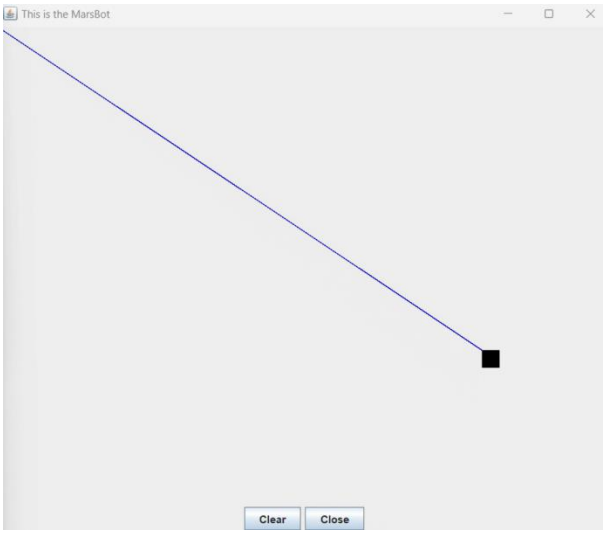

This is the MarsBot
Clear
Close

Table of Contents

I. BÀI 10: PHÂN TÍCH THUẬT TOÁN BLOCK REPLACEMENT TRONG BỘ NHỚ ĐỆM .....
1. Giới thiệu bộ nhớ đệm.....
2. Thuật toán Block replacement Random.....
a. Cơ sở lý thuyết.....
b. Hiệu quả thuật toán.....
c. Phân tích ưu/ nhược điểm thuật toán.....
3. Thuật toán Block replacement LRU.....
a. Cơ sở lý thuyết.....
b. Hiệu quả thuật toán…….
c. Phân tích ưu nhược điểm thuật toán
4. Đánh giá chung

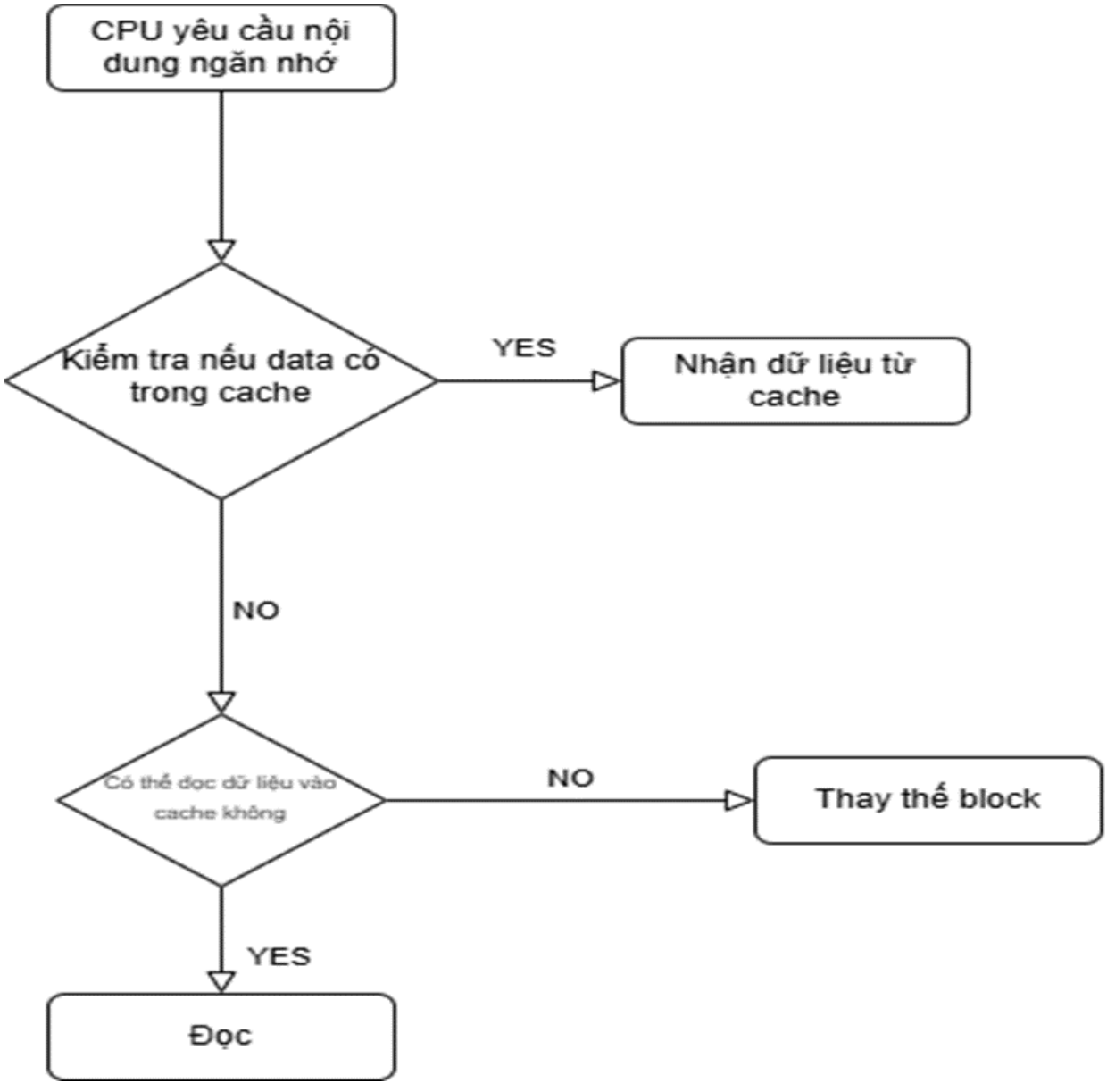
I. BÀI 10: PHÂN TÍCH THUẬT TOÁN BLOCK REPLACEMENT TRONG BỘ NHỚ ĐỆM

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

➢ **Thao tác của Cache**



B1: CPU yêu cầu nội dung của ngăn nhớ

B2: CPU kiểm tra sự tồn tại của dữ liệu này trên cache

B3:

- Nếu có, CPU nhận dữ liệu từ Cache (Nhanh).

- Nếu không có, đọc Block nhớ chứa dữ liệu từ bộ nhớ chính.
  - + Nếu Cache chưa đầy: Thêm khối dữ liệu mới vào Cache
  - + Nếu Cache đầy: Sử dụng thuật toán Block Replacement để thay thế khối dữ liệu.
-

B4: Cập nhật Cache: Sau khi khối dữ liệu mới được thêm vào, Cache được cập nhật với dữ liệu mới.

2. Thuật toán Block replacement Random

a. Cơ sở lý thuyết

➢ **Nguyên lý:** Khi cần thay thế một khối, thuật toán chọn ngẫu nhiên một khối trong Cache để thay thế

➢ **Mô tả hoạt động của thuật toán**:

B1: Xác định số khối trong cache, mỗi khối đánh địa chỉ 0 đến N-1

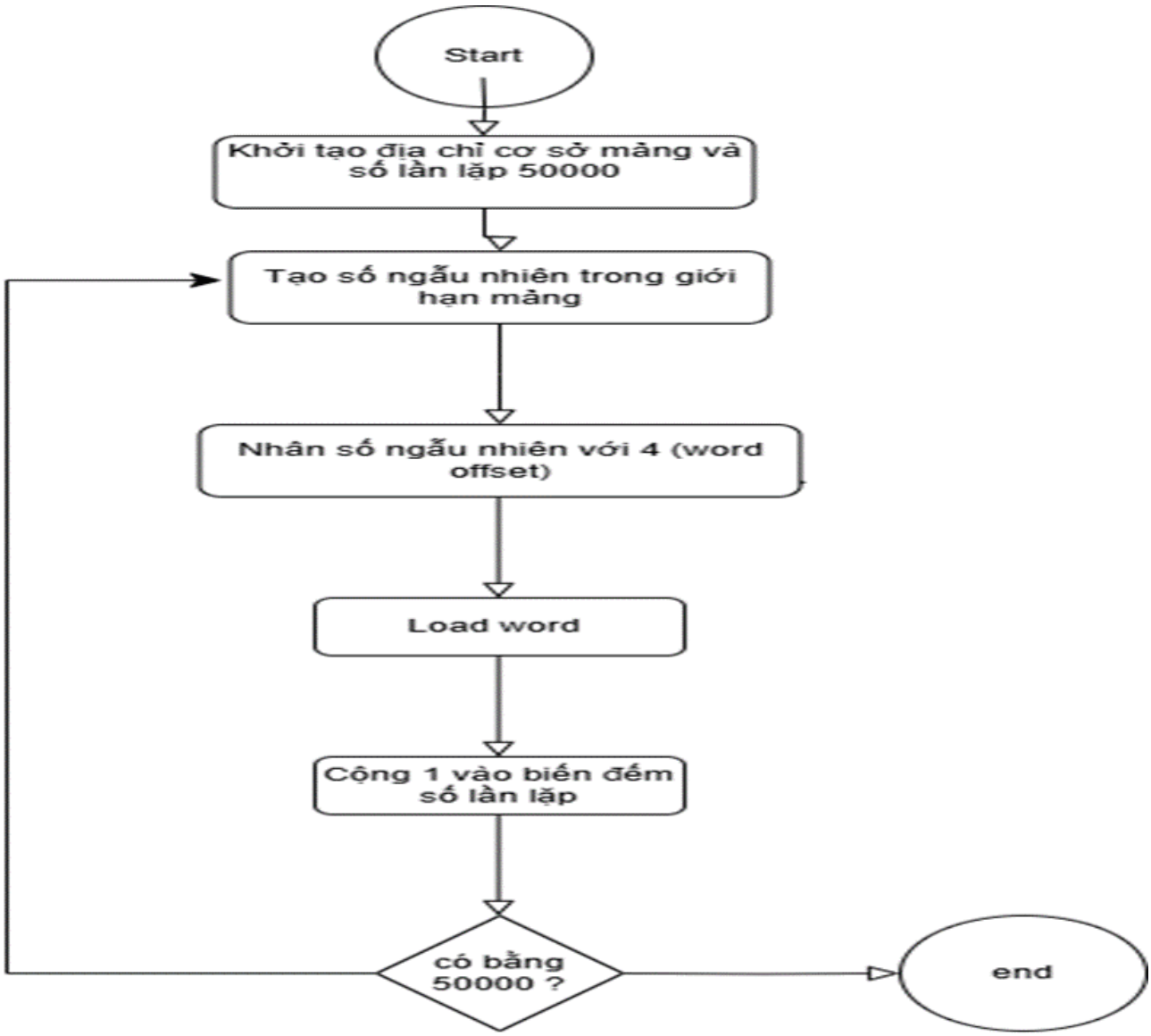
B2: Sử dụng một hàm sinh số ngẫu nhiên trong khoảng từ 0 đến N- 1.

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

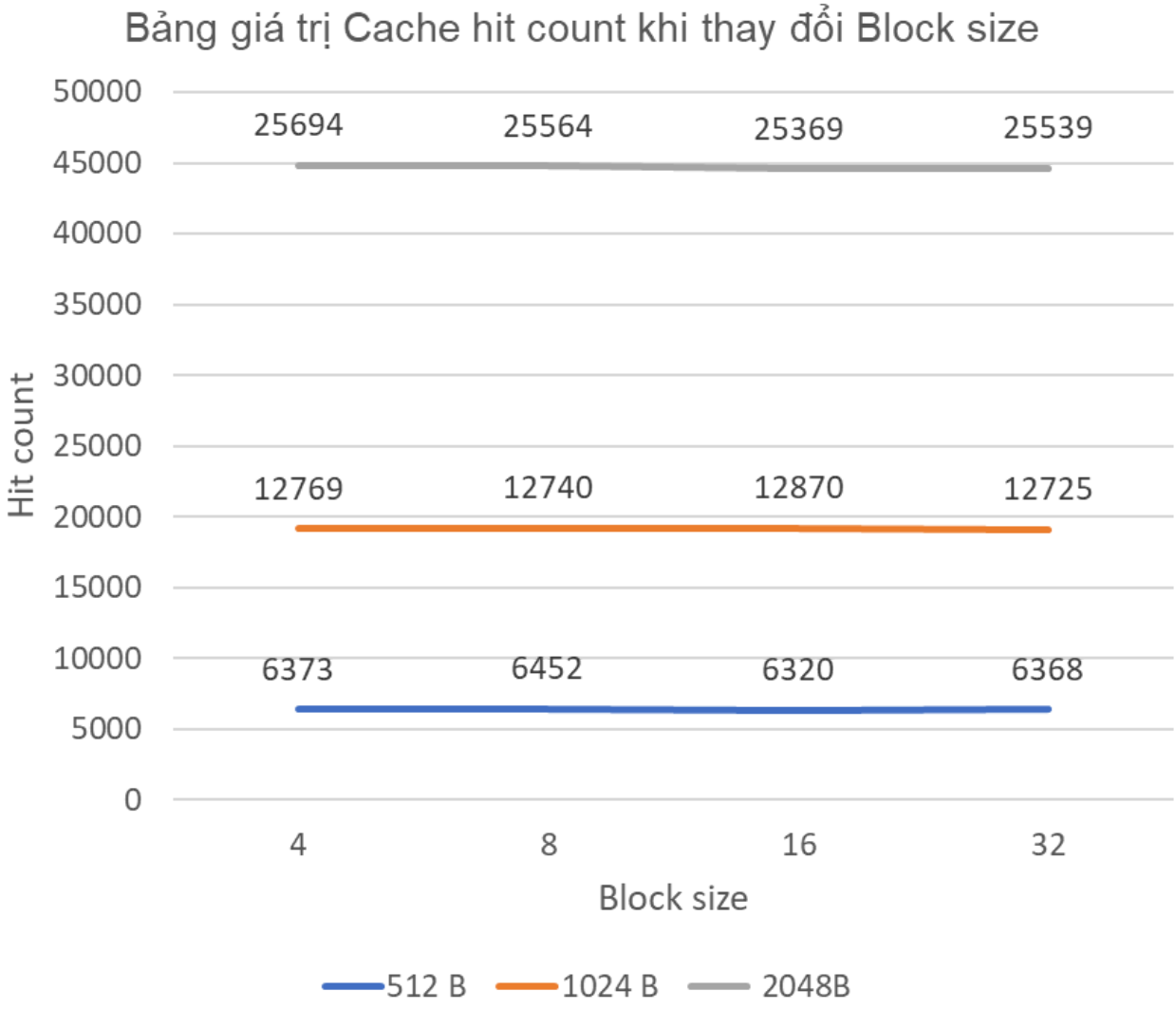
b. Hiệu quả thuật toán

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



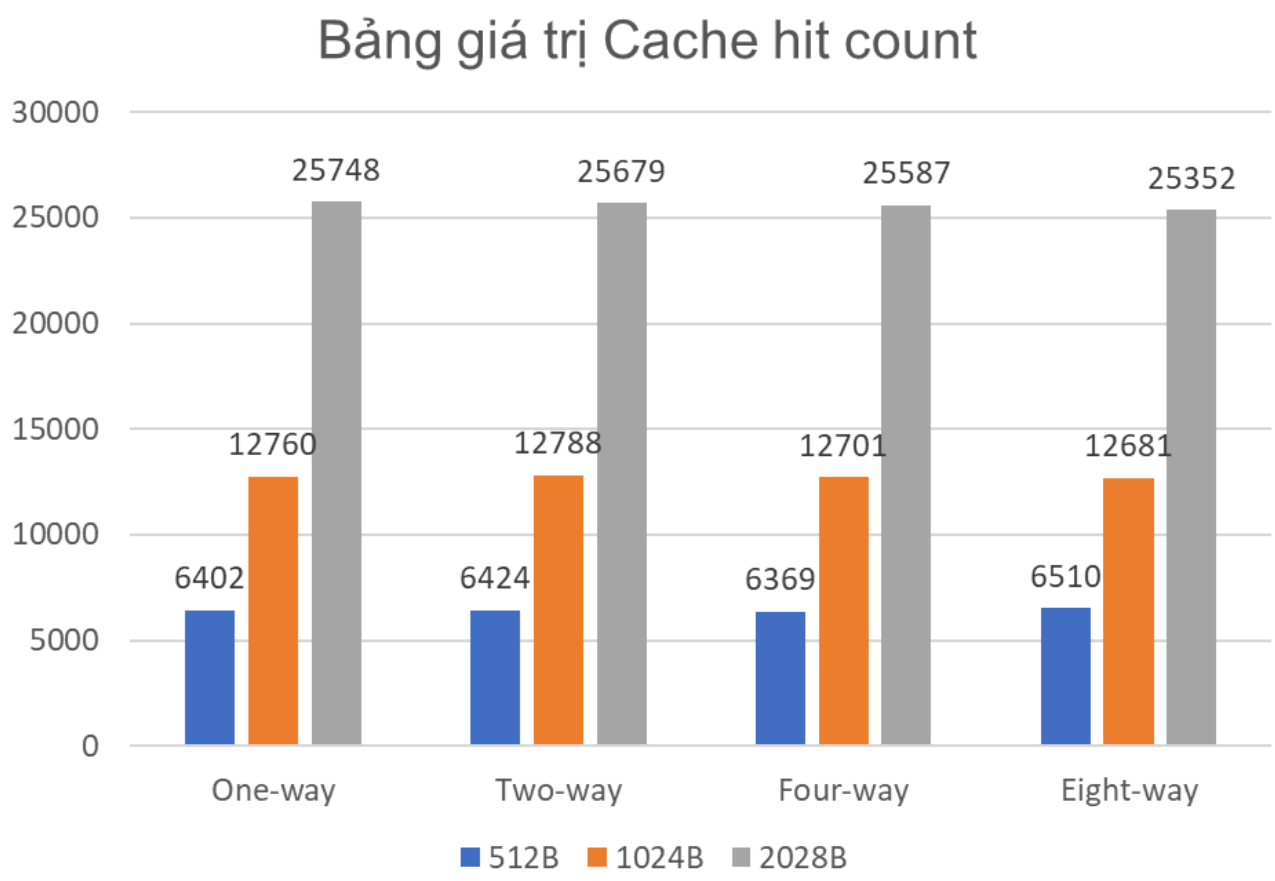
➢ **Hiệu quả thuật toán khi thay đổi các thông số trong Cache**

❖ **Khi thay đổi Block Size**



- **Ưu điểm:** Ban đầu, tăng kích thước khối sẽ giảm tỉ lệ cache miss vì 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, giảm số lần cache miss.
- **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. Điều này có nghĩa là ít dữ liệu khác nhau có thể được lưu trữ trong cache tại một thời điểm, dẫn đến nhiều cache miss hơn khi dữ liệu bị đẩy ra thường xuyên để nhường chỗ cho dữ liệu mới.

❖ **Khi thay đổi Set Size**



- **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.

c. 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

3. Thuật toán Block replacement LRU

a. 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 tới.

➢ **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.

☐ **Ví dụ**: Với cache có phương pháp tổ chức bộ nhớ ánh xạ liên kết 2 đường với 2 Set và sử dụng thuật toán LRU

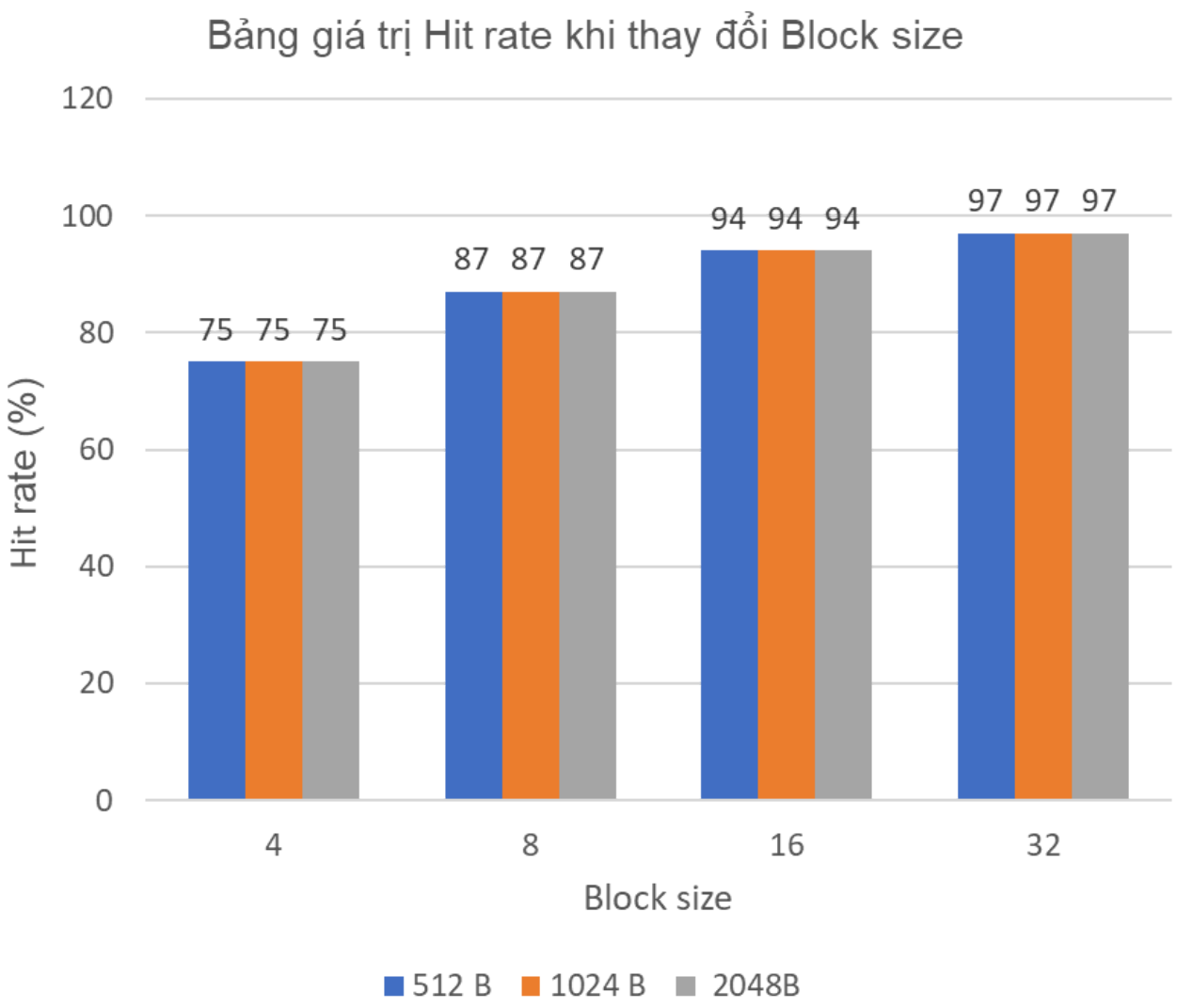
Địa chỉ của khối được tham chiếu	Hit or miss	Nội dung của cache blocks sau khi tham chiếu			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

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

b. Hiệu quả thuật toán

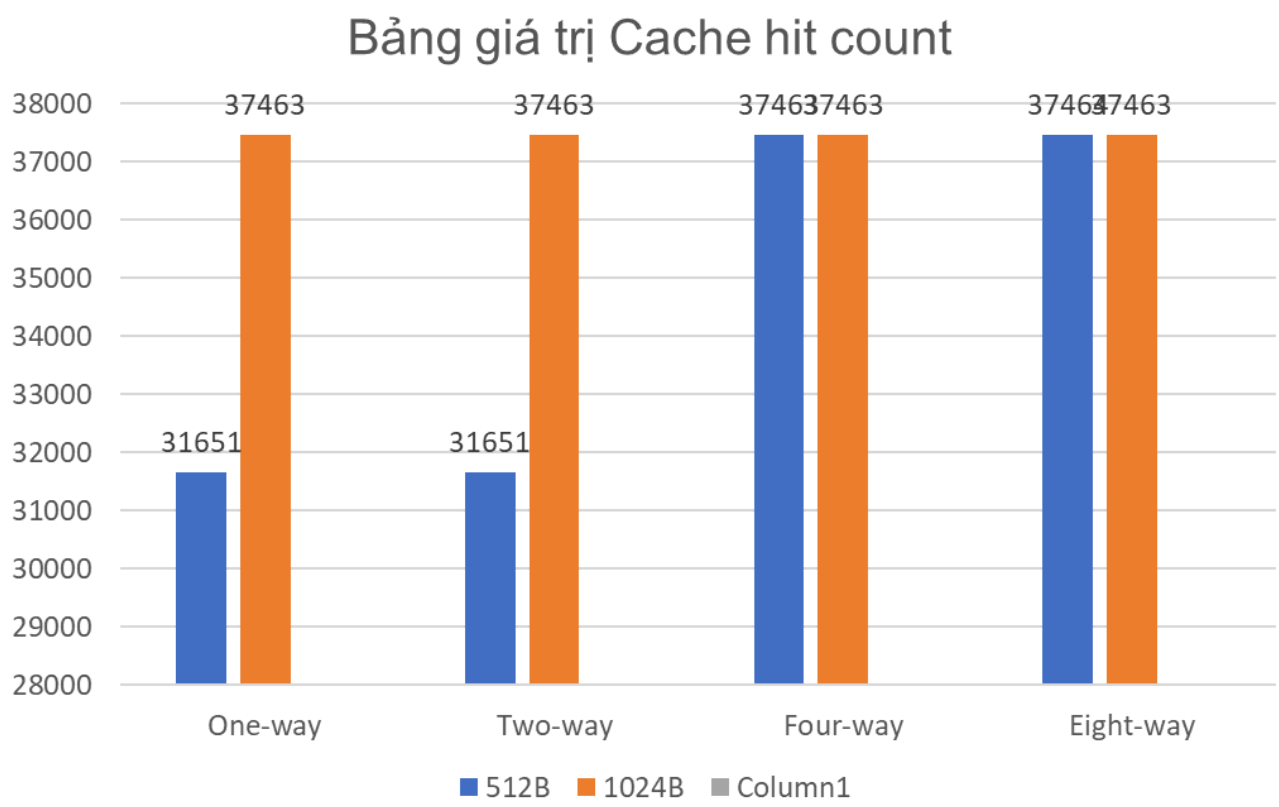
➢ **Hiệu quả thuật toán khi thay đổi các thông số trong Cache**

❖ **Khi thay đổi Block Size**



- **Ưu điểm**: Ban đầu, tăng kích thước khối sẽ giảm tỉ lệ cache miss vì 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, giảm số lần cache miss.
- **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. Điều này có nghĩa là ít dữ liệu khác nhau có thể được lưu trữ trong cache tại một thời điểm, dẫn đến nhiều cache miss hơn khi dữ liệu bị đẩy ra thường xuyên để nhường chỗ cho dữ liệu mới.

❖ **Khi thay đổi Set Size**



- **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.

c. 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.

➢ **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:*** Với cache có tổ chức bộ nhớ ánh xạ 4-way, 8-way, triển khai LRU có thể trở nên phức tạp hơn do cần lưu trữ và quản lý lịch sử tham chiếu của nhiều khối trong nhiều Set khác nhau.

★ ***Không hiệu quả với mô hình sử dụng phân phối đồng đều:*** Trong một số trường hợp, đặc biệt là khi phân phối dữ liệu không tuân theo mô hình sử dụng gần đây (ví dụ như phân phối đều), LRU có thể không hiệu quả.

4. Đánh giá chung

Lựa chọn giữa LRU và Random block replacement phụ thuộc vào yêu cầu cụ thể của hệ thống. Nếu hiệu suất và tối ưu hóa là yếu tố chính, LRU là lựa chọn tốt hơn. Nếu đơn giản và tiết kiệm tài nguyên là yếu tố quan trọng, Random block replacement có thể là một lựa chọn hợp lý.