

Laboratory Exercise 12

Interrupts, IO programming&Cache

Goals

After this laboratory exercise, you should understand the basic principles of interrupts and how interrupts can be used for programming. You should also know the difference between polling and using interrupts and the relative merits of these methods; the basic principles of cache memories and how different parameters of a cache memory affect the efficiency of a computer system.

Literature

■ Patterson and Hennessy: Chapter 2.7, 2.9, 2.10, 2.13, 5.7, Appendix A.6, A.7, A.10, Chapter 7.1–7.3

Polling or Interrupts

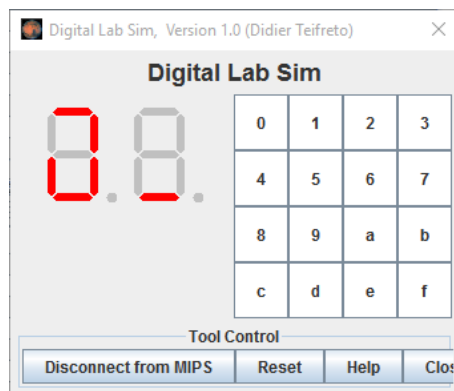
A computer can react to external events either by polling or by using interrupts. One method is simpler, while the other one is more systematic and also more efficient. You will study the similarities and differences of these methods using a simple “toy” example program.

Each peripheral device connects to the CPU via a few ports. CPU uses address to find out the respective port, and after that, CPU could read/write the new value to these port to get/control the device.

Preparation

Study literature and these home assignments before coming into the class.

Assignments at Home and at Lab



Home Assignment 1 - POOLING

Write a program using assembly language to detect key pressed in Digi Lab Sim and print the key number to console.

The program has a unlimited loop, to read the scan code of key button. This is POLLING.

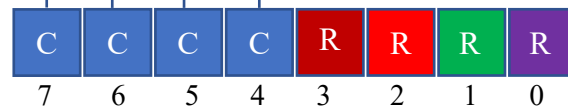
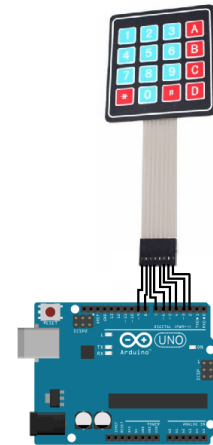
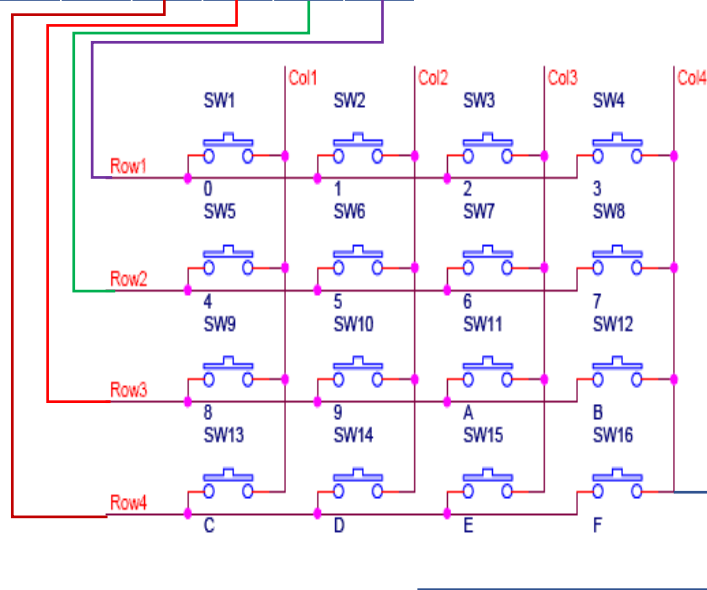
In order to use the key matrix¹, you should

¹ Key matrix animation: http://hackyourmind.org/public/images/keypad12keys_anim.gif

1. assign expected row index into the byte at the address 0xFFFF0012
2. read byte at the address 0xFFFF0014, to detect which key button was pressed

IN_ADRESS_HEXА_KEYBOARD

Address 0xFFFF0012



OUT_ADRESS_HEXА_KEYBOARD

Address 0xFFFF0014

```
#-----
#
#          col 0x1      col 0x2      col 0x4      col 0x8
#
# row 0x1      0          1          2          3
#              0x11      0x21      0x41      0x81
#
# row 0x2      4          5          6          7
#              0x12      0x22      0x42      0x82
#
# row 0x4      8          9          a          b
#              0x14      0x24      0x44      0x84
#
# row 0x8      c          d          e          f
#              0x18      0x28      0x48      0x88
#
#-----
# command row number of hexadecimal keyboard (bit 0 to 3)
# Eg. assign 0x1, to get key button 0,1,2,3
```

```

#      assign 0x2, to get key button 4,5,6,7
# NOTE must reassign value for this address before reading,
# eventhough you only want to scan 1 row
.eqv IN_ADDRESS_HEX_A_KEYBOARD      0xFFFF0012

# receive row and column of the key pressed, 0 if not key pressed
# Eg. equal 0x11, means that key button 0 pressed.
# Eg. equal 0x28, means that key button D pressed.
.eqv OUT_ADDRESS_HEX_A_KEYBOARD      0xFFFF0014

.text
main:          li    $t1,    IN_ADDRESS_HEX_A_KEYBOARD
               li    $t2,    OUT_ADDRESS_HEX_A_KEYBOARD
               li    $t3,    0x08      # check row 4 with key C, D, E, F

polling:       sb    $t3,    0($t1)    # must reassign expected row
               lb    $a0,    0($t2)    # read scan code of key button
               print:    li    $v0,    34      # print integer (hexa)
               syscall
               sleep:    li    $a0,    100      # sleep 100ms
               li    $v0,    32
               syscall
back_to_polling: j    polling      # continue polling

```

Home Assignment 2 - INTERRUPT

Study the following assembly program, which **waits for an interrupt from keyboard matrix, and prints out a simple message**. Go over the code in detail and make sure that you understand everything, especially how to write and install an interrupt routine, how to enable an interrupt, and what happens when an interrupt is activated.

Vietnamese support:

Cũng như các bộ xử lý khác, MIPS có 3 service với cùng một nguyên lý, nhưng khác nhau về mục đích sử dụng

- *Exception: xảy ra khi có lỗi trong quá trình chạy, chẳng hạn tham chiếu bộ nhớ không hợp lệ.*
- *Trap: xảy ra bởi cách lệnh kiểm tra*
- *Interrupt: do các thiết bị bên ngoài kích hoạt*

Cả 3 cơ chế trên đều được gọi chung là Exception.

Cách thức hoạt động: *khi một exception xảy ra*

- *Khi có một Exception xảy ra, MIPS sẽ luôn nhảy tới địa chỉ cố định 0x80000180 để thực hiện chương trình con phục vụ ngắt. Để viết chương trình con phục vụ ngắt, sử dụng chỉ thị **.ktext** để viết code ở địa chỉ 0x80000180 nói trên.*
- *Bộ đồng xử lý C0, thanh ghi \$12 (status) sẽ bật bit 1*
- *Bộ đồng xử lý C0, thanh ghi \$13 (cause) sẽ thay đổi các bit 2~6 cho biết nguyên nhân gây ra ngắt*
- *Bộ đồng xử lý C0, thanh ghi \$14 (epc) sẽ chứa địa chỉ kế tiếp của chương trình chính, để quay trở về sau khi xử lý các đoạn mã Exception xong. (giống như thanh ghi \$ra)*

- Trường hợp thanh ghi \$13 (cause) cho biết nguyên nhân làm tham chiếu địa chỉ bộ nhớ không hợp lệ, thanh ghi \$8 (vaddr) sẽ chứa địa chỉ lỗi đó.
- Nếu không có mã lệnh nào ở địa chỉ 0x80000180 (*.ktext*), chương trình sẽ hiện thông báo lỗi và tự kết thúc.
- Sau khi kết thúc chương trình con, sử dụng lệnh để quay trở lại chương trình chính. Lệnh *eret* sẽ gán nội dung thanh ghi PC bằng giá trị trong thanh ghi \$14 (epc).

Tuy nhiên, lưu ý rằng, trong MARS, thanh ghi PC vẫn chứa địa chỉ của lệnh mà ngắt xảy ra, tức là lệnh đã thực hiện xong, chứ không chứa địa chỉ của lệnh kế tiếp. Bởi vậy phải tự lập trình để tăng địa chỉ chứa trong thanh ghi epc bằng cách sử dụng 2 lệnh mfc0 (để đọc thanh ghi trong bộ đồng xử lý C0) và mtc0 (để ghi giá trị vào thanh ghi trong bộ đồng xử lý C0)

```
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
```

- Các bit 8-15 của thanh ghi Cause, \$13 được sử dụng để xác định nguyên nhân gây ra ngắt. Hãy đọc thanh ghi này, kết hợp với thông tin chi tiết trong hướng dẫn sử dụng của từng thiết bị giả lập để biết được nguồn gốc gây ra ngắt.

Cách thức viết chương trình phục vụ ngắt: để viết chương trình con phục vụ ngắt khi có sự kiện ngắt xảy ra, có thể dùng một trong các phương pháp sau:

1. Viết chương trình con phục vụ ngắt trong cùng một file nguồn
2. Viết chương trình con phục vụ ngắt trong file nguồn độc lập, và lưu trữ trong cùng một thư mục với chương trình chính. Sau đó, sử dụng tính năng trong mục Setting là “Assemble all files in directory”
3. Viết chương trình con phục vụ ngắt trong file nguồn độc lập, và lưu trữ trong cùng một thư mục bất kì. Sau đó, sử dụng tính năng trong mục Setting là “Exception Handler..”

BUG: Ghi nhận các lỗi của công cụ MARS

1. Giữa 2 lệnh syscall và lệnh jump, branch cần bổ sung thêm lệnh nop. Nếu không việc ghi nhận giá trị của thanh ghi PC vào EPC sẽ bị sai
2. Với các công cụ giả lập, nên bấm nút “Connect to MIPS” trước khi chạy giả lập. Nếu không, việc phát sinh sự kiện ngắt sẽ không xảy ra.

```

.equ IN_ADRESS_HEX_A_KEYBOARD 0xFFFF0012

.data
Message: .asciiz "Oh my god. Someone's presed a button.\n"
# ~~~~~
# MAIN Procedure
# ~~~~~
.text
main:
#-----
# Enable interrupts you expect
#-----
# Enable the interrupt of Keyboard matrix 4x4 of Digital Lab Sim
li    $t1, IN_ADRESS_HEX_A_KEYBOARD
li    $t3, 0x80 # bit 7 of = 1 to enable interrupt
sb    $t3, 0($t1)

#-----
# No-end loop, main program, to demo the effective of interrupt
#-----
Loop:  nop
      nop
      nop
      nop
      b     Loop          # Wait for interrupt
end_main:

# ~~~~~
# GENERAL INTERRUPT SERVED ROUTINE for all interrupts
# ~~~~~
.ktext 0x80000180
#-----
# Processing
#-----
IntSR: addi    $v0, $zero, 4 # show message
      la      $a0, Message
      syscall
#-----
# 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
return: eret                  # Return from exception

```

Home Assignment 3 – INTERRUPT & STACK

Study the following assembly program, in which

1. Main program enables 1 interrupt: from key matrix in Data Lab Sim
2. Main program only print a sequence number to console
3. Connect Data Lab Sim. Whenever user press a key button C, D, E, or F, an interrupt raises and print key scan-code to console

```

.eqv IN_ADDRESS_HEX_A_KEYBOARD      0xFFFF0012
.eqv OUT_ADDRESS_HEX_A_KEYBOARD     0xFFFF0014

.data
Message: .asciiz "Key scan code "

# ~~~~~
# MAIN Procedure
# ~~~~~
.text
main:
    #-----
    # Enable interrupts you expect
    #-----
    # Enable the interrupt of Keyboard matrix 4x4 of Digital Lab Sim
    li    $t1,    IN_ADDRESS_HEX_A_KEYBOARD
    li    $t3,    0x80        # bit 7 = 1 to enable
    sb    $t3,    0($t1)

    #-----
    # Loop an print sequence numbers
    #-----
    xor    $s0, $s0, $s0    # count = $s0 = 0
Loop:     addi   $s0, $s0, 1    # count = count + 1
prn_seq:  addi   $v0, $zero, 1
          add    $a0, $s0, $zero    # print auto sequence number
          syscall

prn_eol:  addi   $v0, $zero, 11
          li    $a0, '\n'        # print endofline
          syscall

sleep:    addi   $v0, $zero, 32
          li    $a0, 300        # sleep 300 ms
          syscall
          nop                    # WARNING: nop is mandatory here.
          b     Loop            # Loop
end_main:

# ~~~~~
# GENERAL INTERRUPT SERVED ROUTINE for all interrupts
# ~~~~~
.ktext 0x80000180
    #-----
    # SAVE the current REG FILE to stack
    #-----
IntSR:    addi   $sp, $sp, 4      # Save $ra because we may change it later
          sw     $ra, 0($sp)
          addi   $sp, $sp, 4      # Save $ra because we may change it later
          sw     $at, 0($sp)
          addi   $sp, $sp, 4      # Save $ra because we may change it later
          sw     $v0, 0($sp)
          addi   $sp, $sp, 4      # Save $a0, because we may change it later
          sw     $a0, 0($sp)
          addi   $sp, $sp, 4      # Save $t1, because we may change it later
          sw     $t1, 0($sp)
          addi   $sp, $sp, 4      # Save $t3, because we may change it later
          sw     $t3, 0($sp)
    #-----

```

```

# Processing
#-----
prn_msg: addi    $v0, $zero, 4
        la      $a0, Message
        syscall

get_cod: li      $t1, IN_ADRESS_HEX_A_KEYBOARD
        li      $t3, 0x88          # check row 4 and re-enable bit 7
        sb      $t3, 0($t1)        # must reassign expected row
        li      $t1, OUT_ADRESS_HEX_A_KEYBOARD
        lb      $a0, 0($t1)

prn_cod: li      $v0, 34
        syscall
        li      $v0, 11
        li      $a0, '\n'          # print endofline
        syscall
#-----
# 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      $t3, 0($sp)          # Restore the registers from stack
        addi    $sp, $sp, -4
        lw      $t1, 0($sp)          # Restore the registers from stack
        addi    $sp, $sp, -4
        lw      $a0, 0($sp)          # Restore the registers from stack
        addi    $sp, $sp, -4
        lw      $v0, 0($sp)          # Restore the registers from stack
        addi    $sp, $sp, -4
        lw      $ra, 0($sp)          # Restore the registers from stack
        addi    $sp, $sp, -4

return: eret                                # Return from exception

```

Home Assignment 4 – MULTI INTERRUPT

Vietnamese support:

Thanh ghi số 13, status trong bộ đồng xử lý C0, chứa các thiết lập về tình trạng ngắt.

$$1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0$$
[illegible]

Study the following assembly program, in which:

1. Main program enables 2 interrupts simultaneously: from key matrix and time counter in Data Lab Sim
2. Main program do nothing with a deadlock
3. Connect Data Lab Sim. Whenever **user press any key** or **time interval reaches**, an interrupt raises and print key scan-code to console.

```
.eqv IN_ADDRESS_HEX_A_KEYBOARD 0xFFFF0012
.eqv COUNTER 0xFFFF0013 # Time Counter

.eqv MASK_CAUSE_COUNTER 0x00000400 # Bit 10: Counter interrupt
.eqv MASK_CAUSE_KEYMATRIX 0x00000800 # Bit 11: Key matrix interrupt

.data
msg_keypress: .asciiz "Someone has pressed a key!\n"
msg_counter: .asciiz "Time interval!\n"

# ~~~~~
# MAIN Procedure
# ~~~~~
.text
main:
    #-----
    # Enable interrupts you expect
    #-----
    # Enable the interrupt of Keyboard matrix 4x4 of Digital Lab Sim
    li $t1, IN_ADDRESS_HEX_A_KEYBOARD
    li $t3, 0x80 # bit 7 = 1 to enable
    sb $t3, 0($t1)

    # Enable the interrupt of TimeCounter of Digital Lab Sim
    li $t1, COUNTER
    sb $t1, 0($t1)

    #-----
    # Loop an print sequence numbers
    #-----
Loop:  nop
      nop
      nop
sleep: addi $v0,$zero,32 # BUG: must sleep to wait for Time Counter
      li $a0,200 # sleep 300 ms
      syscall
      nop # WARNING: nop is mandatory here.
      b Loop
end_main:

# ~~~~~
# GENERAL INTERRUPT SERVED ROUTINE for all interrupts
# ~~~~~
.ktext 0x80000180
IntSR: #-----
      # Temporary disable interrupt
      #-----
dis_int:li $t1, COUNTER # BUG: must disable with Time Counter
        sb $zero, 0($t1)
```



```

# no need to disable keyboard matrix interrupt
#-----
# Processing
#-----
get_caus:mfc0    $t1, $13                # $t1 = Coproc0.cause
IsCount:li      $t2, MASK_CAUSE_COUNTER# if Cause value confirm Counter..
        and     $at, $t1,$t2
        beq     $at,$t2, Counter_Intr
IsKeyMa:li      $t2, MASK_CAUSE_KEYMATRIX # if Cause value confirm Key..
        and     $at, $t1,$t2
        beq     $at,$t2, Keymatrix_Intr
others: j        end_process            # other cases

Keymatrix_Intr: li      $v0, 4          # Processing Key Matrix Interrupt
        la      $a0, msg_keypress
        syscall
        j        end_process
Counter_Intr: li      $v0, 4           # Processing Counter Interrupt
        la      $a0, msg_counter
        syscall
        j        end_process
end_process:
        mtc0    $zero, $13            # Must clear cause reg
en_int: #-----
        # Re-enable interrupt
        #-----
        li      $t1, COUNTER
        sb      $t1, 0($t1)
        #-----
        # 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
return: eret                          # Return from exception

```

Home Assignment 5 – KEYBOARD

Vietnamese support:

- Bộ xử lý MIPS cho phép tạo ra ngắt mềm, bằng lệnh *teq*, hoặc *teqi*
- Thiết bị Keyboard không tự động tạo ra ngắt khi có một phím được bấm, mà người lập trình phải tự tạo ngắt mềm.

```

.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 DISPLAY_CODE 0xFFFF000C         # ASCII code to show, 1 byte
.eqv DISPLAY_READY 0xFFFF0008        # =1 if the display has already to do
                                         # Auto clear after sw

.eqv MASK_CAUSE_KEYBOARD 0x0000034   # Keyboard Cause

```

```

.text
        li    $k0, KEY_CODE
        li    $k1, KEY_READY

        li    $s0, DISPLAY_CODE
        li    $s1, DISPLAY_READY

loop:    nop
WaitForKey: lw    $t1, 0($k1)          # $t1 = [$k1] = KEY_READY
        beq   $t1, $zero, WaitForKey # if $t1 == 0 then Polling
MakeIntR: teqi  $t1, 1                # if $t0 = 1 then raise an Interrupt
        j     loop

#-----
# Interrupt subroutine
#-----
.ktext 0x80000180
get_caus: mfc0    $t1, $13              # $t1 = Coproc0.cause
IsCount:  li     $t2, MASK_CAUSE_KEYBOARD # if Cause value confirm
Keyboard..
        and    $at, $t1, $t2
        beq    $at, $t2, Counter_Keyboard
        j      end_process

Counter_Keyboard:
ReadKey:  lw     $t0, 0($k0)            # $t0 = [$k0] = KEY_CODE

WaitForDis: lw    $t2, 0($s1)          # $t2 = [$s1] = DISPLAY_READY
        beq    $t2, $zero, WaitForDis # if $t2 == 0 then Polling

Encrypt:  addi   $t0, $t0, 1            # change input key

ShowKey:  sw     $t0, 0($s0)            # show key
        nop

end_process:
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
return:   eret                          # Return from exception

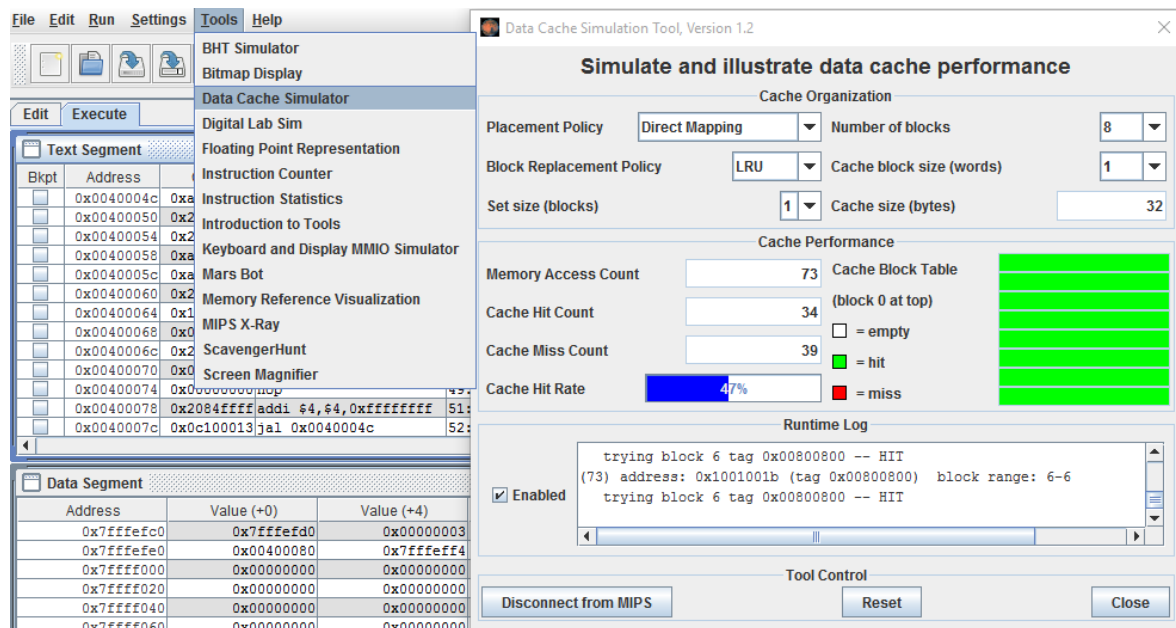
```

Home Assignment 6

Looking for and research information about Cache in CPU.

Try to use Data Cache Simulation Tool in MARS.

Understand terms and change parameters.



Function of a Cache Memory

A cache memory is a memory that is smaller but faster than the main memory. Due to the locality of memory references, the use of a cache memory can have the effect on the computer system that the apparent speed of the memory is that of the cache memory, while the size is that of the main memory.

Cache Efficiency

The actual efficiency gained by using a cache memory varies depending on cache size, block size and other cache parameters, but it also depends on the program and data.

Assignment 1

Create a new project, type in, and build the program of Home Assignment 1.
Upgrade the source code so that it could defect all 16 key buttons, from 0 to F.

Assignment 2

Create a new project, type in, and build the program of Home Assignment 2.
Upgrade the source code so that it could defect all 16 key buttons, from 0 to F.

Assignment 3

Create a new project, type in, and build the program of Home Assignment 3.
Upgrade the source code so that it could defect all 16 key buttons, from 0 to F.

Assignment 4

Create a new project, type in, and build the program of Home Assignment 4.

Upgrade the source code so that it could detect all 16 key buttons, from 0 to F.

Assignment 5

Create a new project, type in, and build the program of Home Assignment 5.

Assignment 6

Run the program in the cache simulator and study how the instruction cache works. Then give *full* answers to the following questions.

- How is the full 32-bit address used in the cache memory?
- What happens when there is a cache miss?
- What happens when there is a cache hit?
- What is the block size?
- What is the function of the tag?

Assignment 7

The parameters of the cache memory can be changed to test the effects of different cases. Investigate the effects of different parameter settings.

- Explain the following: cache size, block size, number of sets, write policy and replacement policy.
- If a cache is large enough that all the code within a loop fits in the cache, how many cache misses will there be during the execution of the loop? Is this good or bad?
- What should the code look like that would benefit the most from a large block size?

Conclusions

Before you finish the laboratory exercise, think about the questions below:

- What is polling?
- What are interrupts?
- What are interrupt routines?
- What are the advantages of polling?
- What are the advantages of using interrupts?
- What are the differences between interrupts, exceptions and traps?
- What is the general idea with cache memory?
- What is a block?
- How does block size affect the efficiency of a cache?
- How fast is a cache memory? How fast is a DRAM?
- Do the optimal cache parameters depend on the program code?
- How can one select good cache parameters?
- Is it possible to change cache size on a PC? On a Mac?