22/12/2015

Αναφορά Εργαστηρίου 6

Ομάδα

Κολομβάκης Χρήστος (Α.Μ.: 2013030103)
Ζαχαριουδάκης Χρήστος (Α.Μ.: 2014030056)

Προεργασία

Κατά την προεργασία του εργαστηρίου μας ζητήθηκε η χρήση της τεχνικής Polling και η χρήση interrupts (διακοπών) και interrupt handler (κώδικα που διαχειρίζεται τα interrupts)για την επικοινωνία με τα περιφερειακά .

Συγκεκριμένα στο πρώτο μέρος της εργαστηριακής άσκησης μας ζητήθηκε :

Η δημιουργία ενός προγράμματος σε Assembly το οποίο να δέχεται μια συμβολοσειρά και να μετατρέπει τα πεζά σε κεφαλαία. Η ιδιαιτερότητα του προγράμματος ήταν στο γεγονός ότι τόσο το διάβασμα των δεδομένων από το πληκτρολόγιο, όσο και το γράψιμο των δεδομένων στην κονσόλα, δεν γινόταν με την χρήση syscall, αλλά με την τεχνική Poling.

Ο αλγόριθμος του προγράμματος είναι ο παρακάτω:

- Τύπωση του μηνύματος καλωσορίσματος.
- Εισαγωγή συμβολοσειράς από τον χρήστη.
- Εκτύπωση της αρχικής συμβολοσειράς γράμμα – γράμμα (echo).
- Εκτύπωση της νέας συμβολοσειράς.

Από την θεωρία μάθαμε, ότι για τον transmitter (οθόνη) και για τον receiver (πληκτρολόγιο) αντιστοιχεί ένας καταχωρητής control και ένας καταχωρητής data.

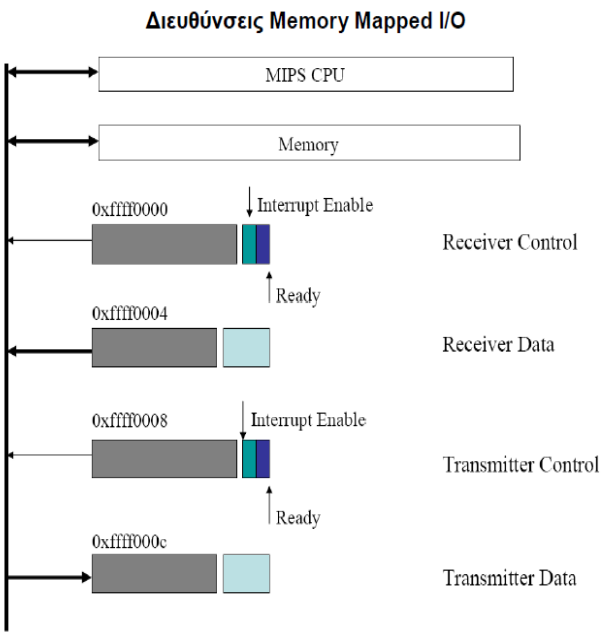
Για την εκτύπωση χαρακτήρων κάναμε τα εξής:

Ελέγχαμε το Ready Bit του καταχωρητή Transmitter Control για να δούμε αν είναι 1. Εφόσον γίνει 1, μεταφέραμε έναν χαρακτήρα του εκάστοτε πίνακα στον καταχωρητή Transmitter Data. Ο χαρακτήρας γράφεται στην λιγότερο σημαντική θέση του καταχωρητή Transmitter Data.

Για την ανάγνωση χαρακτήρων από το πληκτρολόγιο:

Ελέγχαμε το Ready Bit του Receiver Control για να δούμε αν είναι 1. Εφόσον γίνει 1, σημαίνει ότι ο καταχωρητής Receiver Data είναι έτοιμος να δεχτεί έναν καινούργιο χαρακτήρα. Ο χαρακτήρας που πατάμε στο πληκτρολόγιο αποθηκεύεται αρχικά

στον καταχωρητή $v0 και έπειτα γίνεται η μεταφορά στην λιγότερο σημαντική θέση του Receiver Data.



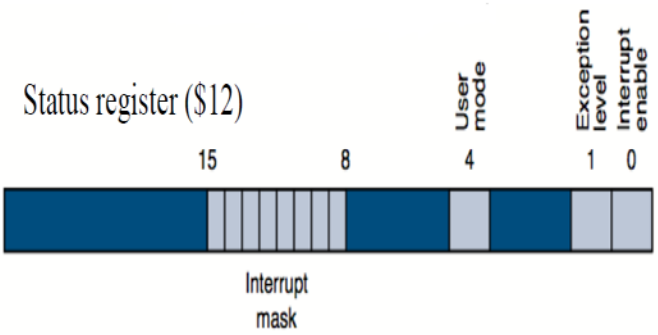
Τόσο για το διάβασμα, όσο και για το γράψιμο, έπρεπε κάθε φορά να ελέγχουμε το αντίστοιχο Ready Bit. Σε αυτόν τον έλεγχο έγκειται η διαδικασία Poling.

Στο δεύτερο μέρος της εργαστηριακής άσκησης μας ζητήθηκε :

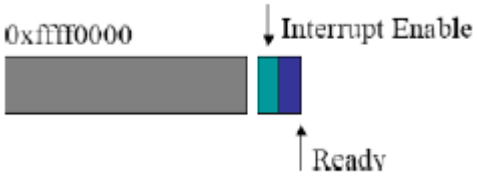
Η άσκηση ζητούσε την εκτύπωση ενός απλού μενού επιλογών με την χρήση syscall και την εκτύπωση των μηνυμάτων «You pressed 1» ή «You pressed 2» , αν πατήθηκε το 1 ή το 2 και την έξοδο από το πρόγραμμα αν πατήθηκε Space . Σε περίπτωση εισαγωγής αλλού χαρακτήρα, το πρόγραμμα το αγνοεί και ξανατυπώνει το μενού. Η εκτύπωση των μηνυμάτων γίνεται με χρήση syscall, άλλα η ανάγνωση του εισαγόμενου χαρακτήρα γίνεται με την χρήση των interrupts και του interrupt handler .

Συγκεκριμένα στο βασικό κώδικα του προγράμματος απαιτήθηκε :

A. Η δημιουργία δύο περιοχών μνήμης cdata και cflag , τις οποίες ο interrupt handler φορτώνει με την τιμή που εισήχθη από τον πληκτρολόγιο (cdata) και με 0 ή 1 ανάλογα αν το πληκτρολόγιο έχει τιμή να δώσει ή όχι (cflag).



B. Η ενεργοποίηση του interrupt enable (bit 1) του status register ($12) του coprocessor (κυκλώματος που αναλαμβάνει την διαχείριση των διακοπών) και των διακοπών πληκτρολογίου (bit 11) του status register .



Επιπλέον χρειάζεται και ενεργοποίηση των interrupts του πληκτρολογίου με αλλαγή του bit 1 του control register του πληκτρολογίου (0xffff0000) .

C. Στην συνέχεια εκτυπώνεται το menu και εάν η τιμή του cflag είναι 1 , γίνεται ανάγνωση των δεδομένων από το cdata και ανάλογα με το περιεχόμενα εκτυπώνει το αντίστοιχο μήνυμα ή απλώς ξανατυπώνει το menu αν το cdata δεν είναι 1,2 ή Space. Αν τιμή του cflag είναι 0 , ξαναγίνεται ανάγνωση του cflag , μέχρι η τιμή του να γίνει 1 .

<u>Στον κώδικα του interrupt handler απαιτήθηκε :</u>

Η προσθήκη κώδικα στο σημείο του exceptions.s όπου γράφει :

 # Interrupt-specific code goes here!
 # Don't skip instruction at EPC since it has not executed.

Ο κώδικας αυτός αναλαμβάνει την ανάγνωση και αποθήκευση των δεδομένων από τον data register του πληκτρολογίου (**0xffff0004**) στο **cdata** και η αλλαγή του **cflag** από 0 σε 1 ,ώστε ο βασικός κώδικας να γνωρίζει πότε υπάρχουν δεδομένα και να εκτελέσει τον interrupt handler . Στον interrupt handler γίνεται χρήση μόνο των καταχωρητών $k0 , $k1 .

Ο κώδικας που διαβάζει τα περιεχόμενα του data register του πληκτρολογίου

```
li $k0,0
lw $k1,0xffff0004($k0)
```

Ο κώδικας που αποθηκεύει τα περιεχόμενα του data register στην περιοχή μνήμης cdata .

```
la $k0,cdata
sb $k1,0($k0)
```

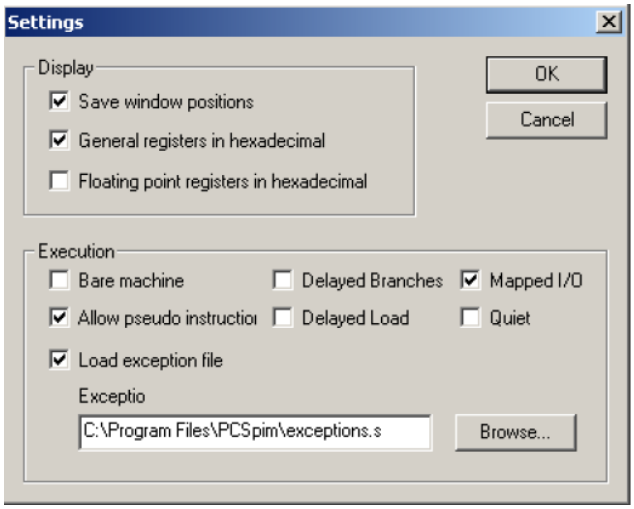
Τέλος ο κώδικας που αλλάζει την περιοχή μνήμης cflag σε 1 .

```
li $k1,1
la $k0,cflag
sw $k1,0($k0)
```

Περιγραφή Ζητούμενων

Σκοπός του 6ου εργαστηρίου ήταν η απόκτηση εμπειρίας στον τρόπο με τον οποίο ο υπολογιστής επικοινωνεί με τις περιφερειακές του συσκευές και συγκεκριμένα την επικοινωνία της μνήμης του υπολογιστή με το πληκτρολόγιο και την κονσόλα . Στο πρώτο μέρος μας ζητήθηκε η χρήση της τεχνικής Polling και στο δεύτερο η χρήση interrupts (διακοπών) για την επικοινωνία με τα περιφερειακά .

Περιγραφή της Εκτέλεσης



Εκτελέσαμε τον κώδικα της Assembly στο πρόγραμμα PCSPIM , όπου εξετάστηκε η ορθή λειτουργία του και η γνώση μας στην εντολές τις Assembly . Πριν την εκτέλεση του προγράμματός μας απαιτήθηκε η αλλαγή των ρυθμίσεων του PCSPIM όπως φαίνεται στην διπλανή εικόνα και επιπλέον στο δεύτερο μέρος της εργαστηριακής άσκησης η επιλογή από το ίδιο μενού το τροποποιημένο exceptions.s , το οποίο περιέχει τον κώδικα του interrupt handler. Αφού ολοκληρώθηκε η σωστή λειτουργία του , εξεταστήκαμε στην κατανόηση του τρόπου λειτουργίας του κώδικα .

Συγκεκριμένα εξεταστήκαμε :

Πρώτο Ερώτημα :

```
Συνάρτηση
addi $sp,$sp,-8
sw $ra,0($sp)
sw $s0,4($sp)
…..
Κλήση συνάρτησης (jal functionname)
…..
lw $ra,0($sp)
lw $s0,4($sp)
addi $sp,$sp,8
```

1. Στον παραπάνω κώδικα γίνεται αποθήκευση (PUSH) και επαναφορά (POP) του περιεχομένου του $**ra** .Ο καταχωρητής $**ra** αλλάζει τιμή κατά την εκτέλεση της εντολής **jal** και περιέχει την διεύθυνση της εντολής μετά την **jal** ,όπου θα πρέπει να επιστρέψει ο PC μετά την ολοκλήρωση της συνάρτησης . Έτσι είναι απαραίτητη η αποθήκευση του όταν καλούμε συνάρτηση μέσα σε συνάρτηση , ώστε η αρχική συνάρτηση να επιστρέψει στο σωστό σημείο της main (ή της συνάρτησης που την κάλεσε) όταν εκτελεστεί η εντολή **jr $ra .** Ο καταχωρητής $s0 αποθηκεύεται και επαναφέρεται καθώς σύμφωνα με τις συμβάσεις της Assembly MIPS , οι καλούμενες συναρτήσεις δεν επιτρέπεται να αλλάξουν τις τιμές $s_ καταχωρητών .

2. Η συνάρτηση TurnToCaps μετατρέπει τα γράμματα της συμβολοσειράς σε κεφαλαία . Αυτό επιτυγχάνεται με τον ακόλουθο κώδικα :

```
beq $t1, 32, repeat
blt $t1, 97, repeat
addi $t1, $t1, -32
```

Αφού γίνει έλεγχος ότι ο χαρακτήρας που έχει αποθηκευτεί στο $t1 , δεν είναι ήδη κεφαλαίο ή Space , ο χαρακτήρας μειώνεται κατά 32 καθώς τα κεφαλαία με τα πεζά γράμματα έχουν διαφορά 32 με τα κεφαλαία να αρχίζουν από τον ASCII κωδικό 65 και τα πεζά από τον ASCII κωδικό 97 . Με την αφαίρεση αυτή επιτυγχάνεται η μετατροπή του χαρακτήρα από πεζό σε κεφαλαίο .

3. Στις συναρτήσεις διαβάσματος από το πληκτρολόγιο και γραψίματος στην κονσόλα.

   Για το γράψιμο στην κονσόλα, δημιουργήσαμε δύο συναρτήσεις, τις PrintString και Write_ch:

   Η **Write_ch** είναι η συνάρτηση που εκτελεί το Poling: Ελέγχει την θέση 0xffff0008 για να δει αν το Ready Bit είναι 1. Ωστόσο, θέλουμε το λιγότερο σημαντικό Bit, οπότε πριν κάνουμε τον έλεγχο, πρέπει να κάνουμε AND του byte που φορτώνουμε, με το 1, έτσι ώστε να

κρατήσουμε μόνο το Ready Bit. Μόλις γίνει 1, στέλνουμε το όρισμά της στην διεύθυνση 0xffff000c για να γίνει η εκτύπωση.
Η Write_ch καλείται μέσω της **PrintString** και συνεχίζει να καλείται για κάθε στοιχείο του πίνακα που βάζουμε κάθε φορά ως όρισμα στην PrintString, μέχρι να βρει το NULL character.

Το διάβασμα από το πληκτρολόγιο γίνεται μέσω των συναρτήσεων ReadString και Read_ch:

Η **Read_ch** είναι η συνάρτηση του Poling: Ελέγχει την θέση **0xffff0000** για το Ready Bit. Η διαδικασία διαβάσματος του Ready Bit είναι όπως και στην **Write_ch**. Μόλις το Ready Bit γίνει 1, μεταφέρουμε τον χαρακτήρα που πατήθηκε – ο οποίος έχει αποθηκευτεί στον καταχωρητή $v0 – στην διεύθυνση **0xffff0004**.
Ο έλεγχος έπειτα γυρνάει στην **ReadString.** Για να γίνει το echo, καλεί την Write_ch με όρισμα τον χαρακτήρα που μόλις διαβάστηκε. Η διαδικασία αυτή συνεχίζεται μέχρι να πατήσει ο χρήστης enter, οπότε το διάβασμα σταματάει. Να σημειωθεί ότι το NULL character προστίθεται στο τέλος της συμβολοσειράς.

Δεύτερο Ερώτημα :

Μας ζητήθηκε η επεξήγηση του κώδικα , όπως περιγράφεται στην προεργασία και η σημασία των cflag και cdata για επικοινωνία του κώδικα του προγράμματος με τον interrupt handler .

Παράδειγμα Εκτέλεσης:

Πρώτο Ερώτημα :

Give the string: hello WorlD
HELLO WORLD

Δεύτερο Ερώτημα :

1.Choice 1
2.Choice 2
Press Space to exit the software.

You pressed 1.

MENU

You pressed 2.

MENU
You have exited the software.

Συμπεράσματα

Κατανοήσαμε τον τρόπο αλληλεπίδρασης υπολογιστή – περιφερειακών , μια διαδικασία η οποία έως τώρα μας φαινόταν αυτονόητη . Διδαχθήκαμε ότι ακόμα και η πληκτρολόγηση την οποία χρησιμοποιούμε καθημερινά , απαιτεί μια αρκετά σύνθετη διαδικασία στο παρασκήνιο , ώστε ο χαρακτήρας που επιλέξαμε να περαστεί στην μνήμη του υπολογιστή καθώς και να εμφανιστεί στην οθόνη όταν τον πληκτρολογούμε . Επιπλέον έχουν ενδιαφέρον οι διαφορετικοί τρόποι με τους οποίους επιτυγχάνεται η διαδικασία αυτή (polling και interrupts) , καθώς και η διαφορές στην εκτέλεση και την αποδοτικότητά τους .

Παράρτημα - Κώδικας

A) ***Υλοποίηση σε Assembly***

ΕΡΩΤΗΜΑ ΠΡΩΤΟ

```
.data

Welc: .asciiz "Give the string:"
err: .asciiz "Error"

array: .align 2
      .space 81

.text

.globl main

main:

la $a0, Welc
jal PrintString

la $a0, array
jal ReadString

la $a0, array
jal TurnToCaps

la $a0, array
```

```
jal PrintString

li $v0, 10
syscall

################################################################

PrintString:

addi $sp,$sp,-8
sw $ra,0($sp)
sw $s0,4($sp)
move $s0,$a0

while1:

lb $a0, 0($s0)
beq $a0,$zero,end_Label

jal Write_ch
addi $s0,$s0,1

b while1

end_Label:

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

################################################################

Write_ch:

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

while2:

lb $s0, 0xffff0008
andi $s0, $s0, 1

beq $s0, $zero, while2

sb $a0, 0xffff000c
```

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

jr $ra

################################################################

Read_ch:

addi $sp, $sp, -4
sw $ra, 0($sp)

while3:

lb $t0, 0xffff0000
andi $t0, $t0, 1

beq $t0, $zero, while3

lb $v0, 0xffff0004

lw $ra, 0($sp)
addi $sp, $sp, 4

jr $ra

################################################################

ReadString:

addi $sp, $sp, -8
sw $ra, 0($sp)
sw $s0, 4($sp)
move $s0,$a0

while4:

jal Read_ch

sb $v0, 0($s0)
move $a0, $v0

jal Write_ch

addi $s0, $s0, 1
```

```
bne $v0, 10, while4
sb $zero, 0($s0)

lw $ra, 0($sp)
lw $s0, 4($sp)
addi $sp, $sp, 4

jr $ra

#################################################################

TurnToCaps:

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

move $s0, $a0

while5:

lb $t1, 0($s0)

beq $t1, $zero, exit_el
beq $t1, 32, repeat
blt $t1, 97, repeat

addi $t1, $t1, -32
sb $t1,0($s0)
addi $s0, $s0, 1

b while5

repeat:
addi $s0, $s0, 1
b while5

exit_el:

move $v0, $s0
lw $ra, 0($sp)
lw $s0, 4($sp)
addi $sp, $sp, 8

jr $ra
```

##

ΕΡΩΤΗΜΑ ΔΕΥΤΕΡΟ (BONUS)

```
.data
.globl cflag
.globl cdata
.align 2
cflag: .space 4
.align 2
cdata: .space 4

menu: .asciiz "\n1.Choice 1\n2.Choice 2\nPress Space to exit the software.\n"
choice_1: .asciiz "\nYou pressed 1\n"
choice_2: .asciiz "\nYou pressed 2.\n"
exit_message: .asciiz "\nYou have exited the software!\n"

.text
.globl main

main:

sw $zero,cflag # Initialize cflag to zero

mfc0 $t0,$12 # Move status register content to $t0
ori $t0,$t0,1 # Interrupt Enable = 1 (0 bit of status register) (1 = 2^0)
ori $t0,$t0,2048 # Διακοπές Πληκτρολογίου (Interrupt Mask) = 1 (11 bit of status
register) (2048 = 2^11)
mtc0 $t0,$12 # Store new value to the status register

lw $t0,0xffff0000 # Keyboard (Receiver) | Control Register | Interrupt Enable
ori $t0,$t0,2 # (2^1 = 2)
sw $t0,0xffff0000

menu_loop :

li $v0,4 # Print the menu
la $a0,menu
syscall

cflag_loop : #

lw $t1 , cflag # Read cflag
beq $t1 , $zero , cflag_loop

lw $t2 , cdata
sw $zero,cflag # Set cflag memory to 0
```

```
bne $t2,49,Ch2_label  # 49 : Ascii Code for 1

#Choice 1

li $v0,4
la $a0,choice_1
syscall

Ch2_label :
bne $t2,50,Exit_label # 50 : Ascii Code for 2
#Choice 2

li $v0,4
la $a0,choice_2
syscall

Exit_label :
bne $t2,32,menu_loop # 32 : Ascii Code for Space //Step 8

#Exit choice

li $v0,4
la $a0,exit_message
syscall

li $v0,10
syscall
```

Ο κώδικας που προσθέσαμε στο αρχείο exceptions.s :

```
# Interrupt-specific code goes here!
# Don't skip instruction at EPC since it has not executed.
li $k0,0
lw $k1,0xffff0004($k0) #Receiver Data

la $k0,cdata #cdata = input character
sb $k1,0($k0)

li $k1,1 #cflag = 1
la $k0,cflag
sw $k1,0($k0)
rfe
```