ΨΗΦΙΑΚΟΙ ΥΠΟΛΟΓΙΣΤΕΣ

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΟΥ 6

ΜΙΧΑΛΗΣ ΓΑΛΑΝΗΣ 2016030036
ΓΙΩΡΓΟΣ ΒΙΡΙΡΑΚΗΣ 2016030035

Προεργασία

Το 6ο εργαστήριο είχε ως θέμα τη διαχείριση μονάδων εισόδου/εξόδου χρησιμοποιώντας **Memory Mapped IO.** Για την επίτευξη του παραπάνω, κληθήκαμε να εφαρμόσουμε δυο διαφορετικές τεχνικές, την **polling** και την **interrupt.**

Για την πρώτη, μας ζητήθηκε να κατασκευάσουμε ένα πρόγραμμα το οποίο διαβάζει μια συμβολοσειρά απο το πληκτρολόγιο, εναλλάσσει τα πεζά και κεφαλαία του γράμματα και εκτυπώνει την ανανεωμένη συμβολοσειρά στην οθόνη.

Στη δεύτερη περίπτωση έπρεπε να κατασκευάσουμε ένα παρόμοιο πρόγραμμα, με τη διαφορά οτι η εναλλαγή γίνεται άμεσα, χαρακτήρα προς χαρακτήρα.

Περιγραφή Ζητούμενων

Αυτή η εργαστηριακή άσκηση αποσκοπούσε στην κατανόηση μας για το πως διαβάζονται «χειροκίνητα» χαρακτήρες απο το πληκτρολόγιο και πως αυτοί εκτυπώνονται στην κονσόλα χωρις την εντολή **syscall.**

ΣΥΝΤΟΜΗ ΘΕΩΡΙΑ POLLING

Οι απλές συσκευές IO όπως πληκτρολόγιο (**Receiver**) και κονσόλα (**Transmitter**) διαθέτουν όλες 2 καταχωρητές **Control** και **Data.** Ο καταχωρητής Control αποτελείται απο το **Ready Bit.** Η διαδικασία του Polling στηρίζεται στον επαναληπτικό έλεγχο της συσκευής μέχρι το Ready Bit να πάρει την τιμή 1, που τότε σημαίνει ότι είναι έτοιμη για χρήση (receive or transmit data).

ΣΥΝΤΟΜΗ ΘΕΩΡΙΑ INTERRUPTS

Σε αντίθεση με την τεχνική Polling, εδώ η κάθε συσκευή ειδοποιεί τον επεξεργαστή ότι είναι έτοιμη οπότε η κανονική ροή του προγράμματος σταματά και ξεκινά η εκτέλεση του **interrupt handler** που αναλαμβάνει την εξυπηρέτησή της. Η διαφορά είναι ότι δε χρειάζεται ο ίδιος ο επεξεργαστής να «ανησυχεί» για τη διαχείριση των συσκευών οπότε μπορεί να ασχοληθεί με πιο χρήσιμα πράγματα.

Περιγραφή της Εκτέλεσης

Κατά τη διάρκεια του προγράμματος χρησιμοποιήσαμε κάποιες σταθερές οι οποίες έκαναν πιο ευανάγνωστο τον κώδικά μας.

```
#"CONSTANTS"
NEXT_LINE           = 10

RECEIVER_CONTROL    = 0xffff0000   READY_BIT           = 0x1
RECEIVER_DATA       = 0xffff0004   INTERRUPT_BIT       = 0x2
TRANSMIT_CONTROL    = 0xffff0008
TRANSMIT_DATA       = 0xffff000c   ENABLE_INTERRUPT_BITS = 0x801
```

- **POLLING**

Αρχικά, έπρεπε να δημιουργήσουμε τις δυο συναρτήσεις **READ_CH** & **WRITE_CH** που ελέγχουν επαναληπτικά εάν η συσκευή είναι έτοιμη για λειτουργία.

```
#"READ_CH"
READ_CH:
    li          $t0, RECEIVER_CONTROL           #"t0 is the address of RECEIVER_CONTROL"
    lw          $t1, ($t0)                      #"t1 is the value of RECEIVER_CONTROL"

check_read:                                     #"Checks repeatedly if ready bit is 1"
    bne         $t1,NULL, read_char             #"Go to read_char if it's ready"
    lw          $t1, ($t0)                      #"t1 is the value of RECEIVER_CONTROL"
    andi        $t1,$t1, READY_BIT              #"t1 becomes AND(t1,READY_BIT)"
    j           check_read                      #"loop again"

read_char:
    lw          $v0, RECEIVER_DATA              #"READ_CH returns v0, which is the value of RECEIVER_DATA"
    jr          $ra                             #"Function exits"



#"WRITE_CH"
WRITE_CH:
    lw          $t1, TRANSMIT_CONTROL           #"t0 is the address of TRANSMIT_CONTROL"
    bne         $t1,NULL, write_char            #"Go to read_char if t1 != null"
    andi        $t1,$t1, READY_BIT              #"t1 becomes AND(t1,READY_BIT)"
    j           WRITE_CH                        #"loop again"

write_char:
    sw          $a0, TRANSMIT_DATA              #"value of TRANSMIT_DATA gets saved in a0"
    jr          $ra                             #"Function exits"
```

Ύστερα, κατασκευάσαμε δυο συναρτήσεις **READ_STR** & **WRITE_STR** που ήταν υπεύθυνες για την ανάγνωση και εγγραφή αντίστοιχα ολόκληρων συμβολοσειρών καλώντας επαναληπτικά τις **READ_CH** & **WRITE_CH.** Η συναρτήσεις αυτές έκαναν χρήση stack memory και η συμβολοσειρά κάθε φορά αποθηκευόταν σε έναν πίνακα χαρακτήρων. Για επιβεβαίωση της λειτουργίας, χρησιμοποιήσαμε την WRITE_STR ακόμα και για έτοιμα μηνύματα πληροφόρησης του χρήστη.

Είχαμε μια ακόμα συνάρτηση, την **CONVERTER,** η οποία σε ένα string μετέτρεπε τα μικρά γράμματα σε κεφαλαία και αντίστροφα. Η υλοποίησή της δεν ήταν δύσκολη, καθώς είχαμε κάνει ένα μεγάλο μέρος της στο 4ο εργαστήριο.

Τέλος, η main καλούσε τις συναρτήσεις με την ακόλουθη σειρά (δε συμπεριλαμβάνονται μηνύματα πληροφόρησης χρήστη):

(READ_STR) -> **(WRITE_STR)** -> **(CONVERTER)** -> **(WRITE_STR)**

- **INTERRUPTS**

Ο interrupt handler βρίσκεται στο exceptions.s αρχείο του PCSpim. Στην κανονική ροή του προγράμματος, τα interrupts ενεργοποιούνται από την αρχή στη main επικοινωνώντας με τον καταχωρητή 12 (coprocessor) με τον ακόλουθο τρόπο:

```
#"Enable interrupts"
mfc0 $t0, $12
ori  $t0, ENABLE_INTERRUPT_BITS
mtc0 $t0, $12

lw      $t0, RECEIVER_CONTROL
ori     $t0, INTERRUPT_BIT
sw      $t0, RECEIVER_CONTROL
```

Η WRITE_CH για εκτύπωση χαρακτήρων χρησιμοποιούσε τεχνική Polling άρα παρέμεινε ίδια. Η READ_CH όμως χρησιμοποιούσε interrupts με τη βοήθεια των θέσεων μνήμης **cflag, cdata.**

```
#"READ_CH"
READ_CH:                                    #"USES INTERRUPTS METHOD TO READ CHARACTER FROM KEYBOARD"
    lw      $t1, cflag                      #"Loads cflag to $t1"

check_read:
    bne     $t1, $zero, read_char           #"if cflag was 0, continue looping, else goto read_char"
    lw      $t1, cflag                      #"Loads cflag to $t1"
    andi    $t1, $t1, READY_BIT             #"ti becomes AND(t1, READY_BIT)"
    j check_read                            #"loop again"

read_char:
    lw      $v0, cdata                      #"Load cdata value to $v0"
    sw      $zero, cflag                    #"We just wrote the char, cflag becomes zero again"
    jr      $ra                             #"Function exits"
```

Σε κάθε κλήση της READ_CH η ροή σταματά, και εκτελείται ο κώδικας του interrupt handler που έχουμε εμείς ορίσει (εκεί γίνεται η εναλλαγή πεζών-κεφαλαίων γραμμάτων).

Επειδή η εκτέλεση του interrupt handler είναι εμβόλιμη να χρησιμοποιήσαμε τους **$k** καταχωρητές για να μην επηρεαστούν οι τιμές των άλλων καταχωρητών. Στο αρχείο exceptions.s επίσης πρέπει να ορίσουμε τα cflag,cdata

```
#"====================================="
#"Defining flag data"

    .data
.globl cflag
.globl cdata

cflag:              .word 0
cdata:              .word 0
#"====================================="

#"====================================="
    #"Reads byte"
    li          $k0, RECEIVER_DATA              #"k0 = address"
    lw          $k1, ($k0)                      #"k1 = value"

    blt         $k1,97,else_label               #"Old is Lowercase"
    bgt         $k1,122,else_label              #"Old is Lowercase"

    addi        $k1,$k1,-32                     #"Convert to Uppercase"

    j           after_cond

else_label:
    blt         $k1,65,after_cond               #"Old is Uppercase"
    bgt         $k1,90,after_cond               #"Old is Uppercase"

    addi        $k1,$k1,32                      #"Convert to Lowercase"

after_cond:
    #"Writes byte"
    sw          $k1, cdata                      #"Stores k1 (value of RECEIVER_DATA) to cdata"
    lw          $k1, cflag                      #"Loads cflag to k1"
    ori         $k1, READY_BIT                  #"k1 becomes OR (k1, READY_BIT)"
    sw          $k1, cflag                      #"Stores k1 to cflag"
#"====================================="
```

Συμπέρασμα

Ολοκληρώνοντας αυτό το εργαστήριο, έχουμε πλέον μια ολοκληρωμένη γνώση για το πως επικοινωνεί ο επεξεργαστής με τις περιφερειακές συσκευές όπως το πληκτρολόγιο και την κονσόλα. Καταλήξαμε στο συμπέρασμα ότι η τεχνική interrupts, φαίνεται μεν πολυπλοκότερη, θεωρείται δε πιο πρακτική και αποτελεσματική καθώς ελευθερώνει πόρους στον επεξεργαστή.

Σας ευχόμαστε καλές γιορτές!