Αναφορά Project

Ομάδα LAB20138024
<Παπαδάμ Διαμαντής Ραφαήλ>
<Τσαρκνιάς Στέργιος>

Προεργασία

Τα προαπαιτούμενα για την υλοποίηση του project ήταν τα εξής:
-Ικανότητα χρήσης C, CLANG και Assembly με ευχέρεια.
-Κατανόηση του τρόπου επικοινωνίας του επεξεργαστή με τα περιφερειακά (πληκτρολόγιο, οθόνη κλπ.)
-Κατανόηση του κώδικα για τα exceptions ούτως ώστε να είναι δυνατή η επεξεργασία αυτού, με σκοπό τον χειρισμό interrupts από το πληκτρολόγιο.

Περιγραφή Ζητούμενων

Ο σκοπός του Project ήταν η περεταίρω εξοικείωση με την αναδρομή, η υλοποίηση αυτής σε Assembly, όπως και η αντίληψη του τρόπου επικοινωνίας με τον επεξεργαστή μέσω polling και interrupts. Το πρώτο ερώτημα (υλοποίηση του project σε C) συνέβαλε στην εξοικείωση με την αναδρομή. Το δεύτερο ερώτημα (υλοποίηση του project σε Assembly) συνέβαλε στην βαθύτερη κατανόηση της γλώσσας προγραμματισμού Assembly (χρήση στοίβας, αναδρομή κλπ.). Το τρίτο (Polling) και το τέταρτο (Interrupts) βοήθησαν στην αντίληψη του τρόπου επικοινωνίας επεξεργαστή-πληκτρολογίου, μέσω διαχείρισης των δεδομένων των θέσεων μνήμης receiver data και receiver control. Το τέταρτο ζητούμενο (Interrupts) συνέβαλε στην κατανόηση του τρόπου με τον οποίο το MIPS χειρίζεται τα exceptions.

Περιγραφής της Εκτέλεσης

Αρχικά κατανοήσαμε τον τρόπο με τον οποίο λειτουργούν οι δοθείσες συναρτήσεις για την εκτύπωση του λαβύρινθου και την εύρεση του βέλτιστου μονοπατιού. Έπειτα υλοποιήσαμε τη συνάρτηση makePlayMove() για την κίνηση του παίκτη. Αφού ολοκληρώσαμε τον κώδικα σε C, τον μετατρέψαμε σε CLANG ούτως ώστε να είναι ευκολότερη η μετάβαση σε assembly. Στην Assembly έγινε χρήση στοίβας στην αναδρομική συνάρτηση makeMove() και στην printLabyrinth() όπως φαίνεται ενδεικτικά παρακάτω:

Δέσμευση μνήμης και αποθήκευση καταχωρητών σε αυτήν

```
makeMove:
        #a1 -> argument
        addi $sp, $sp, -16
        addi $t7, $a1, 0
        sw $t7, 0($sp) #store in stack
        sw $ra, 4($sp) #store in stack

        bge $t7, $0, mV_next_cond_label
        add $v0, $0, $0 #return 0
        addi $sp, $sp, 16
        jr $ra
mV_next_cond_label:
        blt $t7, $t4, no_problem
        add $v0, $0, $0 #return 0
        addi $sp, $sp, 16
        jr $ra

no_problem:
        la $t8, map
        add $t8, $t8, $t7
        lb $t9, ($t8)
        sw $t8, 8($sp)  #storing $t8 (&map[index]) in stack.
        sb $t9, 12($sp) #storing $t9 (map[index]) in stack.
```

Αποδέσμευση μνήμης στοίβας πριν την επιστροφή

```
addi $v0, $0, 1
addi $sp, $sp, 16 #free stack memory
jr $ra
```

Αφού ολοκληρώσαμε την υλοποίηση σε Assembly ασχοληθήκαμε με το polling. Δημιουργήσαμε μία συνάρτηση read_ch κατά την οποία ο επεξεργαστής ρωτάει συνεχώς το πληκτρολόγιο αν έχει εισαχθεί χαρακτήρας μέσω του ready bit του receiver control, και όταν όντως εισάγεται πηγαίνουμε στη διεύθυνση μνήμης του receiver data και παίρνουμε τα τελευταία 8 bits τα οποία είναι ο χαρακτήρας που εισήχθη . Τον επιστρέφουμε μέσω του $v0 στη main και συνεχίζουμε την εκτέλεση του προγράμματος.

```
read_ch:
        la $t5, 0xFFFF0000 #Receiver Control
read_char_loop:
        lw $t7, 0($t5)
        andi $t7, $t7, 1 #keeping only the least significant bit of Transmitter Cotrol.
        beq $t7, 1, ready_bit_is_1
        j read_char_loop

ready_bit_is_1:
        la $t8, 0xFFFF0004 #Receiver Data
        lw $v0, 0($t8) #Put input char in v0
        andi $v0, $v0, 255 #keeping the 8 LS bits of Receiver Data

        jr $ra
```

Τέλος, ασχοληθήκαμε με τα interrupts αφού ενεργοποιήσαμε την λειτουργία στο MIPS ο επεξεργαστής να δέχεται interrupts από το πληκτρολόγιο. Τα interrupts είναι ένας καλύτερος τρόπος από πλευρά απόδοσης να γίνεται η ίδια δουλειά. Σε αυτήν την περίπτωση αντί να έχουμε τον επεξεργαστή να ρωτάει συνεχώς το πληκτρολόγιο αν έχει εισαχθεί κάποιος χαρακτήρας, πραγματοποιεί κανονικά τις λειτουργίες του προγράμματος και το πληκτρολόγιο τον απασχολεί μόνο όταν έχει εισαχθεί κάποιος χαρακτήρας. Ο επεξεργαστής τότε δέχεται το interrupt και το εξυπηρετεί όταν είναι έτοιμος (όταν φτάσει το πρόγραμμα στο σημείο να ελέγξει το cflag). Για να λειτουργήσουν τα παραπάνω τροποποιήσαμε το exceptions.s αρχείο του SPIM προσθέτοντας γραμμές κώδικα για τη διαχείριση του interrupt που προκαλείται από το πληκτρολόγιο.

Exception File Code

```
# Interrupt-specific code goes here!

        #Our Code
        #$v1 -> cflag
        addi $v1, $0, 1 #cflag = 1
        la $t8, 0xFFFF0004 #Receiver Data, $t8 -> cdata
        lw $t8, 0($t8) #Put input char in v0
        andi $t8, $t8, 255 #keeping the 8 LS bits of Receiver Data
        j restore_registers

# Don't skip instruction at EPC since it has not executed.
```

Συμπεράσματα

Το project συνεισέφερε στην βαθύτερη κατανόηση της αναδρομής, της assembly και του τρόπου με τον οποίο ο επεξεργαστής επικοινωνεί με τις περιφερειακές συσκευές. Επίσης, μέσω του project αποκτήσαμε γνώσεις που αφορούν τον τρόπο με τον οποίο το MIPS χειρίζεται τα exceptions.