ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

1η ΑΣΚΗΣΗ ΣΤΗΝ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Ακ. έτος 2011-2012, 5ο Εξάμηνο Σχολή ΗΜ&ΜΥ

ΜΕΡΟΣ Α

Το παρακάτω πρόγραμμα προσπαθεί να αντιγράψει λέξεις από το σημείο της μνήμης που δείχνει ο $a0 στο σημείο που δείχνει ο $a1. Ταυτόχρονα, το πρόγραμμα μετρά τις λέξεις που αντιγράφονται και αποθηκεύει την τιμή του μετρητή στον καταχωρητή $v0. Η αντιγραφή σταματά όταν το πρόγραμμα βρει μια λέξη ίση με το 0. Αυτή η τελευταία λέξη αντιγράφεται μεν από τη μια τοποθεσία της μνήμης στην άλλη, αλλά δεν πρέπει να μετριέται.

```
Loop:     lw      $v1, 0($a0)        # Διάβασε την επόμενη λέξη
          addi    $v0, $v0, 1        # Αύξησε τον μετρητή
          sw      $v1, 0($a1)        # Αποθήκευσε τη λέξη στον προορισμό
          addi    $a0, $a0, 1        # Μετακίνησε το δείκτη στην επόμενη λέξη
          addi    $a1, $a1, 1        # Μετακίνησε το δείκτη στην επόμενη λέξη
          bne     $v1, $zero, Loop   # Αν η λέξη δεν ήταν 0 επανάλαβε το Loop
```

Υπάρχουν αρκετά λάθη στο πρόγραμμα αυτό. Διορθώστε τα δίνοντας μια bug-free εκδοχή του προγράμματος. [*Δε χρειάζεται στην άσκηση αυτή να διατηρείτε τα περιεχόμενα των καταχωρητών $v1, $a0 και $a1*].

Συνολικά έχουμε 4 bugs:

1. Ο μετρητής $v0 δεν έχει αρχικοποιηθεί.
2. Το πρόγραμμα μετρά και την τελευταία μηδενική λέξη.
3. Ο source pointer μετακινείται κατά 1 byte και όχι μια λέξη.
4. Ο destination pointer μετακινείται κατά 1 byte και όχι μια λέξη.

Μια bug-free εκδοχή θα ήταν η εξής:

```
          addi    $v0, $zero, -1     # Απαιτείται αρχικοποίηση του $v0
Loop:     lw      $v1, 0($a0)
          addi    $v0, $v0, 1
          sw      $v1, 0($a1)
          addi    $a0, $a0, ~~1~~ 4      # Το μήκος της λέξης είναι 4 bytes
          addi    $a1, $a1, ~~1~~ 4
          bne     $v1, $zero, Loop
```

ΜΕΡΟΣ Β

Υλοποιήστε τη διαδικασία (procedure) *itoa* σε assembly MIPS, η οποία μετατρέπει ένα ακέραιο (integer) στο αντίστοιχο null-terminated ASCII string. Η διαδικασία δέχεται ως ορίσματα στον καταχωρητή *$a0* τον integer και στον καταχωρητή *$a1* ένα δείκτη στη μνήμη, όπου αποθηκεύεται το string που παράγεται. Η διαδικασία επιστρέφει στον καταχωρητή *$v0* τον αριθμό των χαρακτήρων (εκτός από το null) του string που αποθηκεύτηκαν στη μνήμη.

Ο παρακάτω C κώδικας υλοποιεί την itoa, όπως περιγράφεται στο βιβίο *The C Programming Language* των Kernighan και Ritchie.

```
void reverse(char* s) {
     int c, i, j;
     for( i = 0, j = strlen(s)-1; i < j; i++, j--) {
          c = s[i];
          s[i] = s[j];
          s[j] = c;
     }
}
void itoa(int n, char* s) {
     int i, sign;
     if( (sign=n) < 0)
          n = -n;
     i = 0;
     do {
          s[i++] = n % 10 + '0';
     } while( (n /= 10) > 0);
     if(sign < 0)
          s[i++] = '-';
     s[i] = '\0';
     reverse(s);
}
```

Μια υλοποίηση σε assembly MIPS είναι η παρακάτω:

```
reverse:                                  #Το string που θα αντιστρέψουμε
                                          #βρίσκεται στον $a0
                                          # s = i = $a0, j = $t2
          addi   $t2, $a0, -1
          lbu    $t3, 1($t2)
          beqz   $t3, end_strlen
strlen_loop:
          addi   $t2, $t2, 1
          lbu    $t3, 1($t2)
          bnez   $t3, strlen_loop
end_strlen:                               # j = &s[strlen(s)-1]
          bge    $a0, $t2, end_reverse    # while (i < j)
reverse_loop:
          lbu    $t3, 0(a$0)
          lbu    $t4, 0($t2)
          sb     $t3, 0($t2)
          sb     $t4, 0($a0)
          addi   $a0, $a0, 1
          addi   $t2, $t2, 1
          blt    $a0, $t2, reverse_loop
end_reverse:
          jr     $ra
itoa:     addi   $sp, $sp, -8
```

```
            sw      $ra, 4($sp)
            add     $t0, $a0, $zero         # sign = n
            add     $t3, $a1, $zero         # save string pointer
            add     $v0, $zero, $zero
            bgez    $a0, non_neg            # if(sign < 0)
            sub     $a0, $zero, $a0         #    n = -n
non_neg:    addi    $t2, $zero, 10
itoa_loop:  div     $a0, $t2
            mfhi    $t1                     # $t1 = n % 10
            mflo    $a0                     # $a0 = n / 10
            addi    $t1, $t1, 48            # $t1 = $t1 + '0'
            sb      $t1, 0($a1)
            addi    $a1, $a1, 1
            addi    $v0, $v0, 1
            bnez    $a0, itoa_loop
            bgez    $t0, non_neg2           # if (sign < 0) add '-'
            addi    $t1, $zero, '-'
            sb      $t1, 0($a1)
            addi    $a1, $a1, 1
            addi    $v0, $v0, 1
non_neg2:   sb      $zero, 0($a1)
            add     $a0, $t3, $zero
            sw      $v0, 0($sp)
            jal     reverse
            lw      $v0, 0($sp)
            lw      $ra, 4(sp)
            addi    $sp,$sp, 8
            jr      $ra
```

ΜΕΡΟΣ Γ

Υλοποιήστε τη διαδικασία *atoi* σε assembly MIPS, η οποία μετατρέπει ένα null-terminated ASCII string σε ακέραιο. Η διαδικασία αυτή δέχεται ως όρισμα στον καταχωρητή *$a0* ένα δείκτη στον πρώτο χαρακτήρα του string και επιστρέφει στον καταχωρητή *$v0* την τιμή του αντίστοιχου ακεραίου. Το string θα πρέπει να περιέχει μόνο συνδυασμούς των ψηφίων 0 έως 9. Εάν εμφανιστεί οποιοσδήποτε άλλος χαρακτήρας, τότε η διαδικασία διακόπτει την εκτέλεση της, επιστρέφοντας στον καταχωρητή $v0 την τιμή -1. Για παράδειγμα, αν ο $a0 δείχνει στη μνήμη στη σειρά bytes με τιμή $50_{<10>}52_{<10>}0_{<10>}$, τότε η διαδικασία θα πρέπει να επιστρέψει στον $v0 την τιμή $24_{<10>}$.

```
atoi:  addi   $sp, $sp, -4
       sw     $ra, 0($sp)
       add    $t6, $zero, '0'
       add    $t7, $zero, '9'
       add    $s0, $zero, $zero
       add    $t0, $a0, $zero
Loop:  lb     $t1, 0($t0)
       slt    $t2, $t1, $t6         # check if less than 0
       bne    $t2, $zero, done
       slt    $t2, $t7, $t1         # check if greater than 9
       bne    $t2, $zero, done
       sub    $t1, $t1, $t6
       beq    $s0, $zero, first
       mul    $s0, $s0, 10
first: add    $s0, $s0, $t1
       addi   $t0, $t0, 1
       j      Loop
done:  add    $v0, $s0, $zero
       lw     $ra, 0($sp)
       addi   $sp, $sp, 4
       jr     $ra
```