Παραδείγματα Assembly Λύσεις 1ου Διαγωνίσματος και 1ης Άσκησης (Κωδικοποίησης Hamming)

(Εργαστήριο 3)

Άσκηση 1 (α)

Γράψτε ένα μικρό πρόγραμμα (1-3 εντολές) με το οποίο μπορείτε να ανιχνεύσετε την τιμή του δυαδικού ψηφίου στη θέση 7 της θέσης μνήμης 0X23450009 εάν το λιγότερο σημαντικό ψηφίο είναι αυτό που βρίσκεται στη θέση 0.

lb $t1, 0x23450009

andi $t1, $t1, 0x80 # 0x80 = 1000 0000

Άρα με έλεγχο στον $t1 αποφασίζω αν είναι 0 ή 1.

Άσκηση 1 (β)

Γράψτε με 2 διαφορετικούς τρόπους (addressing modes) μικρά προγράμματα (1-3 εντολές το καθένα) που να φορτώνουν μόνο το περιεχόμενο της θέσης μνήμης 0X18000081 στον καταχωρητή $t1.

1. lb $t1, 0X18000081

2. li $t2, 0X18000081
lb $t1, 0($t2)

Άσκηση 1 (γ)

Συμπληρώστε τις εντολές srl, και and, έτσι ώστε **η κάθε εντολή** να μηδενίζει τον καταχωρητή $t1.

srl $t1 $t1, 32 # ή 0x20
and $t1, $t1, $zero

Άσκηση 2 (α)

Εξηγήστε την εντολή srl $t5, $t4, 8 εάν τα περιεχόμενα των καταχωρητών πριν την εκτέλεση της είναι ως ακολούθως: $t4 = 00F0A012 και $t5 = 03706 B2E. Ποιο θα είναι το αποτέλεσμα;

Κάνει δεξιά ολίσθηση κατά 8 μπιτς στο περιεχόμενο του $t4 και αποθηκεύει το αποτέλεσμα στον $t5.

00F0A012 >> 8 bits, άρα 2 ψηφία Hex:

$t5 = 0000F0A0

Άσκηση 2 (β)

Δίνονται τα περιεχόμενα των καταχωρητών ως ακολούθως: $t6 = 1200F0A0 και $t7 = 0B23706E.
Εξηγήστε την εντολή **andi $t6, $t7, 0X17** και υπολογίστε τα περιεχόμενα του καταχωρητή στον οποίο θα αποθηκευτεί το αποτέλεσμα.

Λογική πράξη "and" μεταξύ του περιεχομένου του $t7 και της τιμής 0X17. Αποθήκευση του αποτελέσματος στον $t6.

0B23 706E **0110 1110**
0000 0017 **and όπου XX είναι: 0001 0111 and**
0000 00XX **0000 0110**

Άρα $t6 = 0X00000006

Άσκηση 2 (γ)

Δίνονται τα περιεχόμενα των καταχωρητών ως ακολούθως: $t3 = 12F400A0, $t4 = 7890B800, $t5 = 00007800.
Εξηγήστε την εντολή xor $t3, $t4, $t5 και υπολογίστε τα περιεχόμενα του καταχωρητή στον οποίο θα αποθηκευτεί το αποτέλεσμα.

Λογική πράξη "xor" ανα μπίτ. Αν το 1 μπιτ είναι μηδέν η έξοδος = με το άλλο μπίτ. Άρα με xor μεταξύ $t4 και $t5, το αποτέλεσμα θα είναι: 7890XX00,
όπου XX = xor μεταξύ B8 και 78 που δίνει C0.
Το αποτέλεσμα θα αποθηκευτεί στον $t3 και είναι:

7890 **C0** 00

Άσκηση 3 (α)

Εάν οι θέσεις μνήμης 0X34560000 έως 0X34560009 έχουν αποθηκευμένες τις δεκαεξαδικές τιμές A0, A1, A2, A3, A4, A5, B6, B7, B8 και B9 αντίστοιχα, υπολογίστε τα περιεχόμενα των καταχωρητών μετά την εκτέλεση των εντολών:

lb $t5, 0X34560007 **$t5 = XX XX XX B7**
lw $t7, 0X34560004 **$t7 = B7 B6 A5 A4**

Άσκηση 3 (β)

Ποιες διευθύνσεις μνήμης θα αλλάξουν περιεχόμενο μετά την εκτέλεση των πιο κάτω εντολών; Καθορίστε επίσης το νέο τους περιεχόμενο σε δεκαεξαδική μορφή εάν **$t8 = 0Xa12bc34d.**

1. sb $t8, 0X20000004
2. sw $t8, 0X20000008

1. Η θέση μνήμης 0X20000004 θα γίνει 4d

2. Οι θέσεις μνήμης 0X20000008, 0X20000009, 0X2000000A, και 0X2000000B, θα γίνουν 4d, c3, 2b, και a1 αντίστοιχα

Άσκηση 3 (γ)

Εξηγήστε την εντολή sll $t5, $t4, 0X10, εάν τα περιεχόμενα των καταχωρητών πριν την εκτέλεση της είναι ως ακολούθως: $t4 = 00F0A012 και
$t5 = 03706B2E

Αριστερή ολίσθηση του περιεχομένου του **$t4** κατά **16 θέσεις** και αποθήκευση του αποτελέσματος στον **$t5.**

Λόγω του ότι το κάθε δεκαεξαδικό ψηφίο είναι 4 μπιτς, το περιεχόμενο του $t4 θα ολισθήσει αριστερά κατά 4 ψηφία και θα εισαχθούν από δεξιά 4 μηδενικά. **$t5 = A012 0000**

Άσκηση 4 (α)

- Γράψτε ένα πρόγραμμα (με loop) που να πολλαπλασιάζει την τιμή του καταχωρητή $t5 3 φορές με το 2 και μετά από κάθε πολλαπλασιασμό να εκτυπώνει την τιμή του στην οθόνη (με κλήση συστήματος). **Η αρχική τιμή του καταχωρητή $t5 να είναι 10. Το loop να εκτελεστεί μόνο 3 φορές,** άρα στην οθόνη αναμένεται να εκτυπωθούν 3 αριθμοί (20, 40 και 80).

Άσκηση 4 (α)

```
1.          li $t5, 0xa ή li $t5, 10
2.          li $t0, 0
3.          li $t1, 3
4.    loop: beq $t0, $t1, end_p
5.          sll $t5, $t5, 1
6.          li $v0, 1
7.          move $a0, $t5
8.          syscall
9.          addi $t0, $t0, 1
10.         b loop
11.   end_p:
```

Άσκηση 4 (β)

Γράψτε ένα πρόγραμμα που να φορτώνει στον καταχωρητή $t1 την τιμή 0X50 και στην συνέχεια (με loop) να τον διαιρεί 2 φορές με το 2 και να εκτυπώνει το αποτέλεσμα της κάθε διαίρεσης στην οθόνη ως integers (άρα θα εκτυπωθούν 2 αριθμοί, ο 0X50/2 και ο 0X50/4).

Άσκηση 4 (β)

```
1.          li $t1, 0x50  # ή li $t1, 80
2.          li $t0, 0
3.          li $t1, 2
4.  loop:   beq $t0, $t1, end_p
5.          srl $t5, $t5, 1
6.          li $v0, 1
7.          move $a0, $t5
8.          syscall
9.          addi $t0, $t0, 1
10.         b loop
11. end_p:
```


Άσκηση 4 (γ)

Γράψτε ένα πρόγραμμα που να φορτώνει στον καταχωρητή $t8 τα περιεχόμενα του $t7 και στην συνέχεια να πραγματοποιεί xor μεταξύ των 3 τελευταίων (λιγότερο σημαντικών) μπιτς του $t8 (με τη χρήση loop). Το αποτέλεσμα (1 ή 0) να αποθηκευτεί στη διεύθυνση μνήμης: 0XAB123456.

Άσκηση 4 (γ)

```
1.          move $t8, $t7 # addi $t8, $t7, 0
2.          li $t0, 0
3.          li $t1, 3
4.  loop:   beqz $t1, end_p
5.          andi $t6, $t8, 1
6.          xor $t0, $t0, $t6
7.          srl $t8, $t8, 1
8.          addi $t1, $t1, -1
9.          b loop
10. end_p:  sb $t0, 0XAB123456
```

Άσκηση 1
(Κωδικοποίηση Hamming)

```
# Hamming Encoder (7,4) to (8,4) with extra parity bit
.data
input: .word 0xfedcba98 # 0x76543210
code_name: .asciiz"Hamming Code Program\n"
input_num: .asciiz"Your Input Number: "
encoded: .asciiz"gives a Hamming Encoded Num of :"
text3:.asciiz" Wrong Number!Number must be between 0-
              255\nTry again.\n"
newline:.asciiz"\n"
```


Άσκηση 1

```
.text
main:
   move $s1, $zero      # initialise $s1 with 0
   move $t1, $zero      # initialise $t1 with 0
   move $t2, $zero      # initialise $t2 with 0
   move $t4, $zero      # initialise $t4 with 0
   move $t8, $zero      # initialise $t8 with 0
   lw $t0, input        # load the 32-bit num into $t0

   li $v0, 4
   la $a0, code_name    # print the name of the code
   syscall
```


Άσκηση 1

```
next_num:
   andi $t9, $t0, 15   # masking to separate the 4 least significant bits

   li $v0, 4
   la $a0, input_num # print the first number to be encoded
   syscall

   li $v0, 1
   move $a0, $t9      # move the 4-bit number in $a0 to be printed
   syscall

      # [p1, p2, d1, p3, d2, d3, d4, p4] transmited bits
      # [t1,  t2, t3,  t4,  t5,  t6, t7,  t8]  their registers
```


Άσκηση 1

```
andi $t7, $t9, 1        # masking to check d4
xor $t1, $t7, $zero     # for p1
xor $t2, $t7, $zero     # for p2
xor $t4, $t7, $zero     # for p3
xor $t8, $t7, $zero     # for p4

srl $t6, $t9, 1         # shift 1 bit to the right
                        # the original number
andi $t6, $t6, 1        # masking to check d3
xor $t2, $t2, $t6       # for p2
xor $t4, $t4, $t6       # for p3
xor $t8, $t8, $t6       # for p4
```

Άσκηση 1

```
srl $t5, $t9, 2     # shift 2 bit to the right the original num
andi $t5, $t5, 1          # masking to check d2
xor $t1, $t1, $t5         # for p1
xor $t4, $t4, $t5         # for p3
xor $t8, $t8, $t5         # for p4

srl $t3, $t9, 3     # shift 3 bit to the right the original num
andi $t3, $t3, 1          # masking to check d1
xor $t1, $t1, $t3         # for p1
xor $t2, $t2, $t3         # for p2
xor $t8, $t8, $t3         # for p4
xor $t8, $t8, $t1         # for P4
xor $t8, $t8, $t2         # for P4
xor $t8, $t8, $t4         # for P4
```



```
# constract the encoded number     [p1, p2, d1, p3, d2, d3, d4, p4]
# using the weight of each position [t1, t2,  t3,  t4,  t5,  t6, t7,  t8]
        move $s0, $zero       # initialise $s0 to keep the result
        beqz $t8, skip1
        addi $s0, $s0, 1      # only if $t8=1, add 1 to the encoded num
skip1:  beqz $t7, skip2
        addi $s0, $s0, 2      # only if $t7=1, add 2 to the encoded num
skip2:  beqz $t6, skip4
        addi $s0, $s0, 4      # only if $t6=1, add 4 to the encoded num
skip4:  beqz $t5, skip8
        addi $s0, $s0, 8      # only if $t5=1, add 8 to the encoded num
skip8:  beqz $t4, skip16
        addi $s0, $s0, 16     # only if $t4=1, add 16 to the encoded num
skip16: beqz $t3, skip32
        addi $s0, $s0, 32     # only if $t3=1, add 32 to the encoded num
skip32: beqz $t2, skip64
        addi $s0, $s0, 64     # only if $t2=1, add 64 to the encoded num
skip64: beqz $t1, skip_end
        addi $s0, $s0, 128    # only if $t1=1, add 128 to the encoded num
```



```
skip_end:

    li $v0, 4
    la $a0, encoded       # print the encoded number message
    syscall

    li $v0, 1
    move $a0, $s0         # move the 4-bit number in $a0 to be printed
    syscall

    li $v0, 4
    la $a0, newline       # print an empty line
    syscall

    srl $t0, $t0, 4       # shift 4bits to the right the 32-bit word
    addi $s1, $s1, 1      # increment counter of numbers to be encoded
    bne $s1, 8, next_num # branch to next_num until 8 numbers are encoded

    li $v0, 10                  # exit programme
    syscall
```