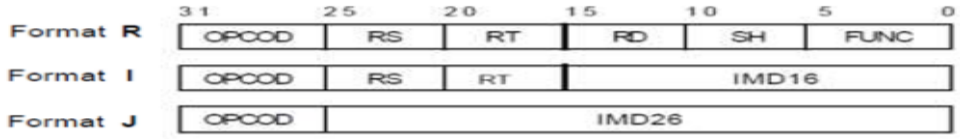
Université Larbi Ben M'Hidi – Oum El Bouaghi
Département des Mathématiques et Informatique
Corrigé type examen : Architecture des ordinateurs
Niveau : deuxième année informatique
Responsable de la matière : Saighi Asma

Réponses des questions de cours :

R1. Toutes les instructions ont une longueur de 32 bits et possèdent un des trois formats suivants : (1 pt)



- Le format **R** : est utilisé par les instructions nécessitant deux registres sources (RS et RT) et un registre résultat (RD). Exemple :

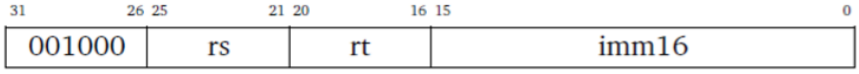
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$00000010001100100100000000100000_2 = 02324020_{16}$

- Le format **I**: est utilisé par les instructions de lecture/écriture mémoire, par les instructions utilisant un opérande immédiat, ainsi que par les branchements conditionnels. Exemple :



L'action effectuée par cette instruction est une opération d'addition entre un registre et une valeur immédiate.

- Le format **J**: n'est utilisé que pour les branchements inconditionnels. Exemple : Le format j est utilisé pour les instructions de saut (ex: Jump (j), Jump and link (jal)). Elle possède deux champs : Opcode sur 6 bits et Address sur 26 bits.

R 2. Une ***interruption*** permet d'arrêter l'exécution du programme en cours afin d'exécuter une tâche jugée plus urgente. Elles sont classées en : Interruption matériel qui est signalée au processeur par un signal électrique sur une borne spéciale et les interruptions logicielles qui sont émises par des programmes (1 pt).

R 3. Les mémoires caches permettent l'**amélioration des performances de l'ordinateur. (1 pt)**
Principe de fonctionnement pour la lecture d'un mot est décrit par l'algorithme :

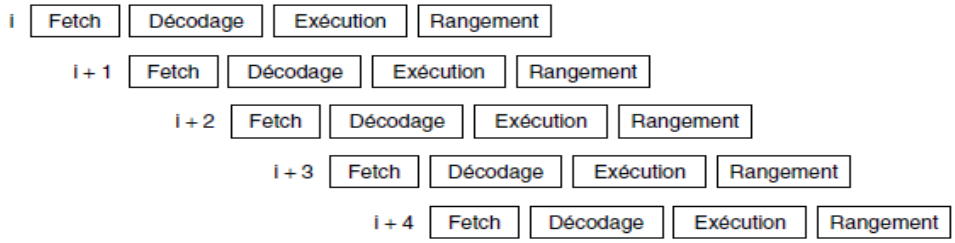
```
si mot présent   // succès (cache hit).
alors charger processeur avec le mot;
sinon            //   échec (cache miss).
     si cache plein
     alors charger cache(remplacer);
          charger processeur;
    sinon
          charger cache;
```

```
        charger processeur;
    finsi
```

L'algorithme suivant décrit le fonctionnement d'une écriture :

```
si mot présent
alors
    modifier cache;
    modifier mémoire principale;
sinon
    modifier mémoire principale;
finsi
```

R 4. Technique de ***pipeline*** **(1 pt)** : La figure ci-dessous montre que l'exécution d'une instruction peut être décomposée en plusieurs phases qui s'exécutent indépendamment les unes des. Dans notre exemple l'exécution est ainsi décomposée en 4 phases, chaque phase étant prise en charge par une unité fonctionnelle différente, créant ainsi un *pipeline à 4 étages*. Dans notre exemple la phase k de l'instruction i s'exécute en même temps que la phase k – 1 de l'instruction i + 1, que la phase k – 2 de l'instruction i + 2, etc. Dans le cas idéal chaque phase est réalisée en un cycle d'horloge et si une instruction peut être décomposée en n phases alors n instructions peuvent être exécutées en parallèle. Ainsi pour notre pipeline à 4 étages si le cycle horloge est de 2 nanosecondes, alors il faut 8 nanosecondes pour exécuter une instruction et cette machine.



Exercice 1 : (5 pt)

```
.data
msg : .asciiz "entrer un nombre"
.text
main :
li $v0, 4
la $a0 , msg
syscall
li $v0, 5
syscall
move $s0, $v0
li $t0 , 1
li $s1, 1
loop : beq $s0, $t0 , exit
mul $s1,$s1, $s0
sub $s0,$s0, 1
j loop
exit:
li $v0, 1
move $a0, $s1
syscall
li $v0,10
syscall
jr $ra
```

Exercice 2 : (4 pt)

```
.data
arr : .word 256
msg : .asciiz "combien d'elements ? \n"
msg2 : .asciiz "entrez un nombre \n"
msg3 : .asciiz " , "
.text
main :
li $v0 , 4
la $a0, msg
syscall
li $v0, 5
syscall
move $t0,$v0
li $t1,0
la $t3,arr
move $t4,$t3
la $a0 ,msg2
startreading: beq $t1 ,$t0 ,finishreading
li $v0 , 4
syscall
li $v0 ,5
syscall
sw $v0,0($t3)
addi $t1,$t1,1
addi $t3 ,$t3, 4
j startreading
finishreading:
move $t3,$t4
li $t1,0
move $t3,$t4
startsorting : beq $t1,$t0,finishsorting
addi $t6,$t1 ,1
move $s2 ,$t3
lw $t5 ,0($t3)
addi $s7 , $t3 ,4
innersorting: beq $t6 , $t0 ,finishinner
        lw $s5 , 0($s7)
        check :bge $s5 ,$t5 , dont
                move $s2 ,$s7
                move $t5, $s5
        dont:
        addi $t6,$t6,1
        addi $s7 ,$s7 ,4
j innersorting
finishinner:
lw $s4 , 0($t3)
lw $s5 , 0($s2)
sw $s4 ,0($s2)
sw $s5 , 0($t3)
addi $t3,$t3,4
addi $t1,$t1,1
j startsorting
finishsorting:
li $t1 , 0
move $t3,$t4
```

```
startprinting : beq $t1,$t0,finishprinting
lw $a0, 0($t3)
li $v0,1
syscall
la $a0, msg3
li $v0 ,4
syscall
addi $t1,$t1,1
addi $t3 ,$t3,4
j startprinting
finishprinting:
li $v0,10
syscall
jr $ra
```

Exercice 3 : (4 pt)

```
.data
bin : .byte 256
msg :.asciiz "donnez un nombre \n"
msg2 : .asciiz " le nombre en binaire est : "
.text
main :
li $v0,4
la $a0,msg
syscall
li $v0,5
syscall
la $t0,bin
move $t1,$v0
li $t2,-1
loop : beqz $t1,endloop
addi $t2,$t2,1
rem $t3,$t1,2
div $t1,$t1,2
add $t4 ,$t0,$t2
sb $t3,0($t4)
j loop
endloop :
la ,$a0 ,msg2
li $v0 ,4
syscall
li $v0 ,1
 startperint : beq $t2,-1,end
 lb $a0,0($t4)
 sub $t2 ,$t2, 1
 add $t4,$t0,$t2
 syscall
 j startperint
 end :
 li , $v0 , 10
 syscall
 jr $ra
 jr $ra
```

Exercice 4: (3 pt)

```
.data
in: .space 32
out: .space 32
msg: .asciiz "entrer une chaine\n"
.text
main:
        la      $a0,msg
        li      $v0,4
        syscall
        li      $v0, 8
        la      $a0, in
        li      $a1, 32
        syscall
        jal     strlen
        move    $t2, $a0 #first element's address
reverse:
        li      $t0, 0
        li      $t3, 0
        reverseLoop:
                add     $t3, $t2, $t0
                lb      $t4, 0($t3)
                beqz    $t4, exit
                sb      $t4, out($t1)
                sub     $t1, $t1, 1
                addi    $t0, $t0, 1
                j       reverseLoop
exit:
        li      $v0, 4
        la      $a0, out
        syscall
        li      $v0, 10
        syscall
strlen:
        li      $t0, 0
        li      $t2, 0
        strlen_loop:
                add     $t2, $a0, $t0
                lb      $t1, 0($t2)
                beqz    $t1, strlenExit
                addiu   $t0, $t0, 1
                j       strlen_loop
        strlenExit:
                sub     $t0, $t0, 1
                move    $t1, $t0 #length
                li      $t0, 0
                jr      $ra
```

N.B: D'autres solutions sont possibles pour résoudre les exercices.