Support TP2 AO
Les sauts et les appels de fonctions ou procédures

1. Sauts inconditionnels

Jump	**j**
Jump **a**nd **l**ink	**jal**
Jump **r**egister	**jr**
Jump **a**nd **l**ink **r**egister	**jalr**

- **j adr** -> va à l'adresse **adr**
- **jal adr** -> sauvegarde en plus **$pc** dans le registre **$ra**
- **jr $t0** -> va à l'adresse contenue dans le registre **$t0**
- **jalr $t0** -> sauvegarde en plus **$pc** dans le registre **$ra**

Saut avec retour (Jump And Link) :

```
jal address
jr $ra
```

- **jal** : Sauvegarde l'adresse de l'instruction suivante dans le registre ***ra***, puis saute à l'adresse constante **address**.
- L'instruction **jr $ra** est typiquement employée pour rendre la main à l'appelant à la fin d'une fonction ou procédure.

Exemple

```
1  print_int:
2  li $v0, 1        # # Charger le registre $v0 avec le code 1 pour afficher entier (print_int)
3  syscall          # system call  pour print_int
4  jr $ra           # rendre la main à l'appelant à la fin de la fonction/procédure print_int(return)
5  main:
6  li $a0, 15       # On veut enregistrer la valeur 15 dans le registre$a0
7  jal print_int    # Sauvegarde l'adresse de l'instruction suivante dans le registre ra, puis saute à l'adresse print_int
```

2. Sauts conditionnels :

Branch on **eq**ual	**beq**
Branch on **n**ot **e**qual	**bne**
Branch on **l**ess **t**han	**blt**
Branch on **g**reater **t**han	**bgt**
Branch on **l**ess or **e**qual than	**ble**
Branch on **g**reater or **e**qual than	**bge**

- **beq r1, r2, adr** **->** si les contenus des registres **r1** et **r2** sont égaux, saute à l'adresse **adr**

Exemple : Instruction conditionnelle if

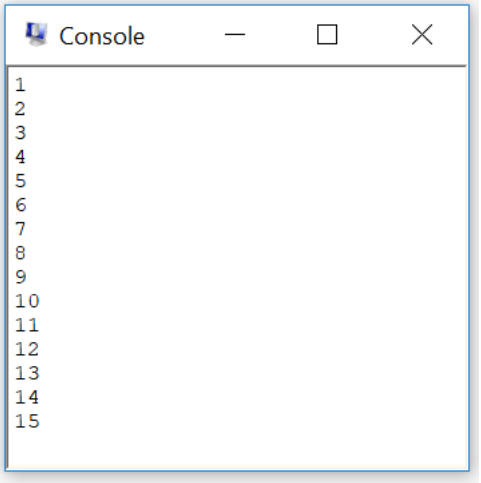
if t1 < t2 then t3 := t1 else t3 := t2

Code MIPS équivalent

```
blt $t1, $t2, Then  # si t1 < t2 saut à Then
move $t3, $t2       # t3 := t2
j End               # saut à End
Then: move $t3, $t1 # t3 := t1
End:                # suite du programme
Li $v0,10
syscall
```

Exercice 1 :

Ecrire le programme MIPS permettant d'afficher les valeurs entières de 0 à 15 comme indiqué ci-dessous sur la console.

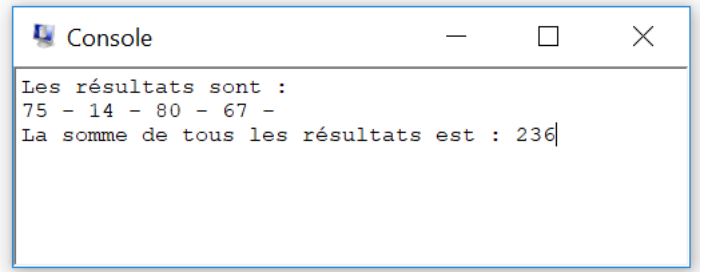


 # Solution de l'exercice 1 :

```
1   .data
2   Newln : .asciiz "\n"
3   .text
4   main :
5   li $t0,1
6   li $t1,15
7   loop : move $a0,$t0
8   li $v0,1
9   syscall
10  li $v0,4
11  la $a0,Newln
12  syscall
13  addi $t0,$t0,1
14  ble $t0,$t1,loop #if ($t0<=$t1 aller à loop)
15  li $v0,10
16  syscall
```

Exercice 2 :

En utilisant l'appel fonction *affiche*, faire un programme MIPS qui calcule et affiche les résultats de : x+y+z ; 2z(x+4) ; (x+8)/5 ; x+y ; (x+y)/(z/2) Tels que x, y, z sont des variables déclarées sur 8 bits et leurs valeurs sont respectivement : 10, 50, 15.
Pour l'affichage il faut avoir : (Voir l'image de la console ci-dessous)
Message1- **Les résultats sont :** *retour à la ligne*
Résultat des calculs : **résultat1 - résultat2 - résultat 3 - résultat 4 -**
Message 2 - **La somme de tous les résultats est :** ***Résultat de la somme***



 # Solution de l'exercice 2 :

```
1   .data
2   x: .byte 10
3   y: .byte 50
4   z: .byte 15
5   message1: .asciiz"Les résultats sont : \n"
6   separateur:.asciiz " - "
7   retour:.asciiz "\n"
8   message2 : .asciiz"La somme de tous les résultats est : "
9   .text
10  main:
11  li$v0,4
12  la $a0,message1
13  syscall
14  lb $t1,x
15  lb $t2,y
16  lb $t3,z
17  add $a0,$t1,$t2
18  add $a0,$a0,$t3
19  move $t0,$a0
20  jal affiche
21  addi $a0,$t1,4
22  move $t5,$a0
23  jal affiche
24  li $t4,8
25  mul $a0,$t1,$t4
26  move $t6,$a0
27  jal affiche
28  add $a0,$t1,$t2
29  li $t4,2
30  div $t4,$t3,$t4
31  add $a0,$a0,$t4
32  move $t7,$a0
33  jal affiche
34  li$v0,4
35  la $a0,retour
36  syscall
37  li$v0,4
38  la $a0,message2
39  syscall
40  add $t4,$t0,$t5
41  add $a0,$t4,$t6
42  add $a0,$a0,$t7
43  li $v0,1
44  syscall
45  j exit
46  exit:
47  li $v0,10
48  syscall
49  affiche:
50  li $v0,1
51  syscall
52  li$v0,4
53  la $a0,separateur
54  syscall
55  jr $ra
```