T. P 02

Matière : Architecture des ordinateurs

Déclaration de données:

.data

- **.ascii *str***
  This stores **str** in memory, but without a null terminator.
- **.asciiz *str***
  This stores str in memory, but with a null terminator. The "z" refers to zero, which is the ASCII code for the null character. This is how C-style strings are stored.
- **.byte *b1, ..., bn***
  Store n bytes contiguously in memory (you get to pick n). I'll assume the values b1,...,bn can be written in either in base 10 or in hex. I'll also assume commas are needed to separate the values. Finally, I assume that the values can be written on more than one line.
- **.halfword *h1, ..., hn***
  Store n 16-bit halfwords contiguously in memory (you get to pick n). I'll assume the values h1,...,hn can be written in either in base 10 or in hex. I'll also assume commas are needed to separate the values. I assume that the values can be written on more than one line. Finally, I assume the halfwords are half word aligned in memory, i.e., initial byte stored at addresses divisible by 2.
- **.word *w1, ..., wn***
  Store n 32-bit words contiguously in memory (you get to pick n). I'll assume the values w1,...,wn can be written in either in base 10 or in hex. I'll also assume commas are needed to separate the values. I assume that the values can be written on more than one line. Finally, I assume the words are word-aligned in memory, i.e., initial byte stored at addresses divisible by 4.
- **.space *numBytes***
  Reserves numBytes of space in memory.

```
 .data  # Tells assembler we're in the data segment
val:  .word  10, -14, 30
str:  .ascii  "Hello, world"
num:  .byte  0x01, 0x03
arr:  .space 100
```

We have four declarations above. Each starts with a label, which consists of the identifier and a colon, then the "type", then possibly the data.
The assembler tries to store the data in consecutive memory locations, and tries to observe word alignment, if applicable.

Instructions assembleur :

N°	Assembleur	opération	Commentaire	Format
01	bge $rs, $rt, label	Si ($rs>=$rt) goto label	Voir aussi bgt, ble,blt	pseudo
02	la $rd, label	$rd = label	Mettre l'adresse de label dans rd	Pseudo
03	lw $rt, imm($rs)	$rt = Mem[$rs + imm]	Charger un mot de la mémoire	I
04	sw $rt, imm($rs)	Mem[$rs + imm] = $rt	Range $rt en mémoire	I
05	J destination	PC = adresse*4	Saut à destination, adresse = destination/4	J
06	Move $rd, $rs	$rd = $rs	Mettre $rs dans $rd	pseudo

T. P 02

Matière : Architecture des ordinateurs

Programme MIPS correspondant

```
.data
Tab1 : .word 45,128,47,89,324,223,706,72,11186,44


.text
.globl __start
__start:
      la $t3, Tab1        # put address of  list into $t3
      li $t2, 0
      li $t5, 0           # put the index into $t2

while:
            bge $t5,10,EndWhile
            lw   $t4, 0($t3)
            addi $t3,$t3,4
            addi $t5,$t5,1
            li   $v0, 1
            move $a0, $t4
            syscall
            j while

EndWhile:
         li $v0 10
         syscall
```

Soit le programme C suivant :

```
#include <stdio.h>
main()
{
    int a[] = {45,128,47,89,324,223,706,72,11186,44};
    int i=0;
    while (i<10)
    {
        printf("%d",a[i]);
        i++;
     }
 }
```

A faire :

- Exécuter le programme sous votre simulateur (PCSPIM, MARS, …)
- Améliorer l'affichage en séparant les nombres par un espace,
- Déclarer un deuxième tableau nommé Tab2 Qui contient les valeurs de 1 à 10,
- Additionnez les éléments des deux tableaux élément par élément et afficher le résultat de la somme.