Chapitre 4 : Le processeur

1. Introduction

Le **processeur** (noté **CPU**, pour Central Processing Unit) est le cerveau de l'ordinateur. C'est un circuit électronique cadencé au rythme d'une horloge interne, qui envoie des impulsions, appelées « top ».

A chaque top d'horloge le processeur exécute une **instruction** (opération élémentaire) ou une partie d'instruction. La puissance du processeur est ainsi caractérisée par le nombre d'instructions qu'il est capable de traiter par seconde. Les instructions sont stockées dans la mémoire centrale, en vue d'être traitée par le processeur.

Un processeur est défini par :

1) la largeur de ses registres internes de manipulation de données (8, 16, 32, 64, 128 bits);
2) la cadence de son horloge exprimée en MHz ou GHz ;
3) le nombre de noyaux de calcul (core) ;
4) son jeu d'instructions (ISA en anglais, Instructions Set Architecture) dépendant de la famille (CISC, RISC, etc) ;
5) sa finesse de gravure exprimée en nm (nanomètres, 10^{-9} mètres, soit un milliardième de mètre).

Quelques exemples de processeurs

2. Rôle du processeur, calcul de CPI (Cycle per Instruction), les processeurs CISC et RISC.

Rôle du processeur

- Le rôle fondamental de la plupart des processeurs, indépendamment de la forme physique qu'ils prennent, est d'exécuter une série d'instructions stockées appelée programme.
- Les instructions (parfois décomposées en micro-instructions) et les données transmises au processeur sont exprimées en mots binaires (code machine). Elles sont généralement stockées dans la mémoire.
- Le séquenceur ordonne la lecture du contenu de la mémoire et la constitution des mots présentés à l'UAL qui les interprète.

Calcul de CPI (Cycle par Instruction)

- A chaque top d'horloge, le processeur exécute une action, correspondant à une instruction ou une partie d'instruction.
- L'indicateur appelé **CPI** (*Cycles Par Instruction*) permet de représenter le nombre moyen de cycles d'horloge nécessaire à l'exécution d'une instruction sur un microprocesseur.
- La puissance du processeur peut ainsi être caractérisée par le nombre d'instructions qu'il est capable de traiter par seconde.
- L'unité utilisée est le **MIPS** (Millions d'Instructions Par Seconde) correspondant à la fréquence du processeur que divise le *CPI*.
- La moyenne des cycles par instruction dans un processus donné est définie comme suit:

$$CPI = \frac{\Sigma_i (IC_i)(CC_i)}{IC}$$

Où ICi est le nombre d'instructions pour un type d'instruction donné i, CCi est les cycles d'horloge pour ce type d'instruction et IC=Σi(ICi) est le nombre total d'instructions. La sommation résume tous les types d'instructions pour un processus d'analyse comparative donné.

- La puissance du processeur peut ainsi être caractérisée par le nombre d'instructions qu'il est capable de traiter par seconde. L'unité utilisée est le MIPS (Millions d'Instructions Par Seconde) correspondant à la fréquence du processeur que divise le CPI.

Exemple 1:

Pour le MIPS multi-cycle, il existe cinq types d'instructions :

1. Load (5 cycles)
2. Store (4 cycles)
3. R-type (4 cycles)
4. Branch (3 cycles)
5. Jump (3 cycles)

Si un programme a:

- 50% load instructions
- 25% store instructions
- 15% R-type instructions
- 8% branch instructions
- 2% jump instructions

Alors le CPI est égale à **4.4**

$$\text{CPI} = \frac{5 \times 50 + 4 \times 25 + 4 \times 15 + 3 \times 8 + 3 \times 2}{100} = 4.4$$

Exemple 2

Un processeur à 400 MHz a été utilisé pour exécuter un programme de référence avec le mélange d'instructions et le nombre de cycles d'horloge suivants:

Instruction TYPE	Instruction count	Clock cycle count
Integer Arithmetic	45000	1
Data transfer	32000	2
Floating point	15000	2
Control transfer	8000	2

Determine the effective CPI, MIPS (Millions of instructions per second)rate, and execution time for this program.

$$\text{CPI} = \frac{45000 \times 1 + 32000 \times 2 + 15000 \times 2 + 8000 \times 2}{100000} = \frac{155000}{100000} = 1.55$$

$400Mhz = 400,000,000Hz$

since: $MIPS \propto 1/CPI$ and $MIPS \propto clockFrequency$

$$\text{Effective processor performance} = \text{MIPS} = \frac{\text{clock frequency}}{\text{CPI}} \times \frac{1}{1\text{ Million}} = \frac{400,000,000}{1.55 \times 1000000} = \frac{400}{1.55} = 258\,\text{MIPS}$$

Therefore:

$$\text{Execution time}(T) = \text{CPI} \times \text{Instruction count} \times \text{clock time} = \frac{\text{CPI} \times \text{Instruction Count}}{\text{frequency}} = \frac{1.55 \times 100000}{400 \times 1000000} = \frac{1.55}{4000} = 0.0003875\,\text{sec} = 0.3875\,\text{ms}$$

Les processeurs CISC et RISC

Ce qui caractérise principalement un processeur est la famille à laquelle, il appartient :

- **Le processeur à architecture CISC (Complex Instruction Set Code)** est un processeur dont le jeu d'instructions possède les propriétés suivantes :
  - Il contient beaucoup de classes d'instructions différentes.
  - Il contient beaucoup de type d'instructions différentes complexes et de taille variable.
  - Il se sert de beaucoup de registres spécialisés et de peu de registres généraux.

- **Le processeur à architecture RISC (Reduced Instruction Set Code)** est un processeur mis en place par IBM dans les années 70, dont le jeu d'instructions possède les propriétés suivantes :
  - Le nombre de classes d'instructions différentes est réduit par rapport à un CISC.
  - Les instructions sont de taille fixe.
  - Il se sert de beaucoup de registres généraux.
  - Il fonctionne avec un pipeline

Architecture RISC	Architecture CISC
Instructions simples ne prenant qu'un seul cycle	Instructions complexes prenant plusieurs cycles
Instructions au format fixe	Instructions au format variable
Décodeur simple (câblé)	Décodeur complexe (microcode)
Beaucoup de registres	Peu de registres
Seules les instructions LOAD et STORE ont accès à la mémoire	Toutes les instructions sont susceptibles d'accéder à la mémoire
Peu de modes d'adressage	Beaucoup de modes d'adressage
Compilateur complexe	Compilateur simple

Tableau comparatif entre l'architecture RISC et l'architecture CISC

3. Le microprocesseur MIPS R3000

L'**architecture MIPS** (de l'anglais : ***m****icroprocessor without* ***i****nterlocked* ***p****ipeline* ***s****tages*) est une architecture de processeur de type Reduced instruction set computer (RISC) développée par la société *MIPS Technologies*.

Les premières versions de l'architecture MIPS (R2000, R3000) étaient 32-bits (autant au niveau des registres que des chemins de données), mais par la suite, des versions 64-bits sont apparues telle que le R4000 sorti en 1991(le premier processeur 64 bits).

Le processeur MIPS est capable de terminer l'exécution d'une instruction à chaque cycle d'horloge. Pour obtenir ce type de performances, il est nécessaire d'avoir des instructions de taille constante et de construire un pipeline. Cette structure permet d'exécuter chaque instruction en plusieurs cycles, mais de terminer l'exécution d'une instruction à chaque cycle. En plus d'un banc de registres, le MIPS possède des unités séparées pour l'extraction des instructions, le décodage des instructions, l'exécution et les accès mémoire.

Les processeurs MIPS sont notamment utilisés dans des stations de travail (Silicon Graphics, DEC…), de nombreux systèmes embarqués (Palm, modems, imprimantes,…), et également des consoles de jeux (Nintendo 64, Sony PlayStation 2, etc.).

4. Structure externe du processeur MIPS R3000

L'architecture externe est le niveau d'abstraction nécessaire à l'écriture de programmes assembleur, de la partie génération de code d'un compilateur, et du programmeur de systèmes d'exploitation multiprocessus (et/ou multitâches). Nous allons traiter, en particulier, dans cette section :

- les registres visibles du logiciel ;
- l'adressage de la mémoire ;
- les appels systèmes.

Les jeux d'instructions seront traités dans une section à part.

Le processeur possède deux modes :

- le mode utilisateur (ou user) pour exécuter les applications, et
- le mode noyau (ou kernel) pour exécuter le système.

Ces 2 modes sont nécessaires à l'exécution sure de plusieurs processus sur un même processeur.

1) les registres visibles du logiciel :

Les registres du MIPS R3000 visibles du logiciel, c.-à-d. qui sont manipulés par les instructions implicitement ou explicitement, ont tous une taille de 32 bits.
Le MIPS R3000 visant par construction l'exécution de multiples processus, des mécanismes de protections sont mis en œuvre pour l'accès aux registres relatifs au système. Ces derniers registres appartiennent à un coprocesseur système dit coprocesseur 0 ou *cop0*. Les accès à ce coprocesseur ne peuvent avoir lieu qu'en mode noyau.
Le processeur possède deux modes de fonctionnement (Utilisateur/Superviseur) imposant ainsi d'avoir deux catégories de registres :

a) **Registres non protégés (Registres du processeur):** Le processeur possède **35** registres manipulés par les instructions standards, dont :

- 32 registres de **32 bits** numérotés **$0; : : : ; $31**. Les registres peuvent être accédés soit par leur numéro soit par leur nom.

Nom	Numéro	Description
$zero	$0	constante zéro
$at	$1	réserve pour l'assembleur
$v0	$2	retour de fonction
$v1	$3	retour de fonction
$a0	$4	argument de fonction
$a1	$5	argument de fonction
$a2	$6	argument de fonction
$a3	$7	argument de fonction
$t0	$8	temporaire
$t1	$9	temporaire
$t2	$10	temporaire
$t3	$11	temporaire
$t4	$12	temporaire
$t5	$13	temporaire
$t6	$14	temporaire
$t7	$15	temporaire

Nom	Numéro	Description
$s0	$16	sauvegardé
$s1	$17	sauvegardé
$s2	$18	sauvegardé
$s3	$19	sauvegardé
$s4	$20	sauvegardé
$s5	$21	sauvegardé
$s6	$22	sauvegardé
$s7	$23	sauvegardé
$t8	$24	temporaire
$t9	$25	temporaire
$k0	$26	pour noyau système
$k1	$27	pour noyau système
$gp	$28	pointeur global
$sp	$29	pointeur de pile
$fp	$30	pointeur de frame
$ra	$31	registre d'adresse

- **pc , *program counter*** ce registre contient l'adresse de l'instruction à exécuter. Sa valeur est modifiée par toutes les instructions ;
- **ir , *instruction register*** ce registre contient l'instruction en cours d'exécution. Il n'est pas directement accessible
- **hi** et **lo** ces registres contiennent le résultat de la multiplication sur 64 bits, ou le résultat de la division euclidienne (quotient dans **lo** et reste dans **hi**).

Registres non protégés MIPS R3000

Exemples :
Pour les divisions, on peut donner l'exemple suivant :

```
addi $t3, $0, 37
addi $t4, $0, 2
div $t3, $t4
mflo $t0 #équivaut à 37 / 2 équivaut à 18 (le quotient)
mfhi $t1 #équivaut à 37 % 2 équivaut à 1 (le reste)
```

Et pour les multiplications, on peut donner l'exemple suivant :

```
addi $t3, $0, 37
addi $t4, $0, 2
mult $t3, $t4
mflo $t1 #on y trouve le produit mais il faut que le produit soit sur 32 bits
```

b) Registres protégés (Registres du coprocesseur 0) : Ces registres concernent la gestion des exceptions, interruptions et appels systèmes :

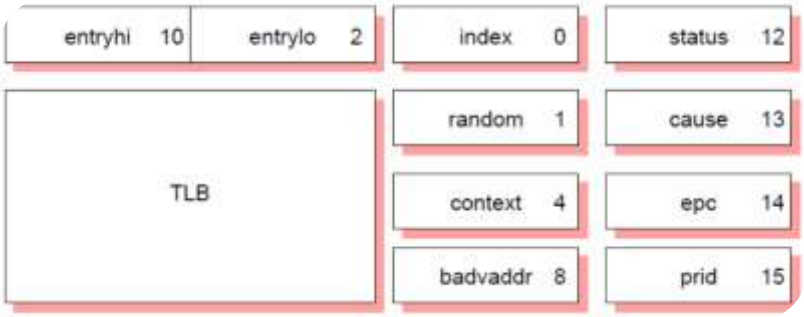
- **badvaddr ,** ***bad virtual address*** ce registre contient l'adresse fautive en cas d'exception de type « adresse illégale » ;
- **SR Registre d'état (Status Register)** c'est le registre d'état. Il contient les masques d'interruption et le mode ;
- **CR Registre de cause** c'est le registre qui contient la cause de l'exception ;
- **EPC Registre d'exception (*Exception Program Counter)*** ce registre contient l'adresse de retour en cas d'interruption et l'adresse de l'instruction fautive en cas d'exception ou d'appel système.

Les registres suivants concernent la gestion de la mémoire virtuelle.

- **TLB,** ***Transaltion Lookaside Buffer*** c'est la mémoire associative pour la traduction adresse virtuelle vers adresse physique ;
- **index** registre contenant l'index de la **TLB** dans lequel faire les accès ;
- **random** registre contenant un index aléatoire valide pour les accès à la **TLB** ;
- **context** registre utilisé partiellement par le logiciel (une partie sert à pointer sur la structure des pages du système d'exploitation) et par le matériel (l'autre partie

contient les poids forts de l'adresse fautive lors d'une traduction qui échoue) pour faciliter l'écriture en logiciel de la gestion de la mémoire virtuelle ;

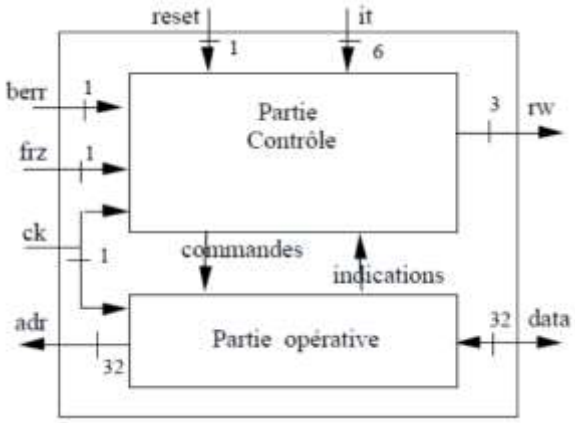
- **entryhi** et **entrylo** valeur à comparer aux entrées de la **TLB** pour savoir s'il y a une erreur de page (***page fault***).



Registres protégés (du coprocesseur 0) du MIPSR3000

5. Structure interne du processeur MIPS R3000

- L'architecture interne du processeur se décompose en une partie opérative et une partie contrôle.
- La partie opérative (PO) contient les registres et les opérateurs. Elle réalise des transferts élémentaires de données entre un ou plusieurs registres sources et un registre destination.
- Un transfert élémentaire est exécuté en un cycle.
- La partie opérative est commandée par la partie contrôle (PC).
- La partie contrôle est chargée de définir, pour chaque cycle d'horloge, les transferts élémentaires qui doivent être réalisés par la partie opérative.
- La partie contrôle contient principalement un séquenceur décrit comme un automate d'états finis (automate de MOORE).
- Les entrées de cet automate sont les signaux indicateurs envoyés par la partie opérative, et dont les sorties sont les signaux de commande constituant la micro-instruction.
- Cet automate est décrit de façon graphique. Les nœuds (ellipses) représentent les états, les flèches représentent les transitions entre états, qui peuvent dépendre des signaux provenant de la partie opérative.
- Chaque état est identifié par un nom.
- A chaque état de l'automate est associée une micro-instruction, qui porte le nom de cet état.
- Toute instruction du langage machine se décompose en une séquence de microinstructions.
- Toutes les instructions commencent par la même micro-instruction, qui réalise l'incrémentation du compteur ordinal.
- Ensuite le comportement de l'automate dépend de la valeur du code opération (IR [31:26]), qui fait partie des entrées de l'automate.
- Enfin, par convention, la dernière micro-instruction d'une instruction i effectue la lecture de l'instruction i+1.



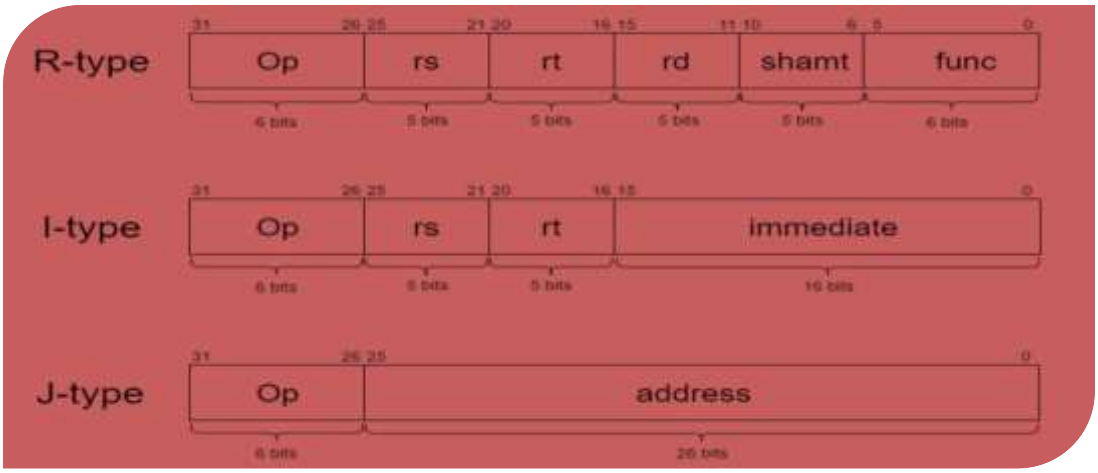
L'architecture interne du processeur MIPS R3000

6. Jeu d'instructions, Formats et programmation du MIPS R3000

Trois (3) formats d'encodage sont possibles dans MIPS :

Chaque instruction MIPS est contenue sur **32 bits**, l'encodage des **3** types d'instruction possibles est décrit plus haut.

1) **Les instructions de type R (R** pour **R**egistre**)** utilisent **3** registres pour les opérations arithmétiques et logiques, **2** registres sources comme opérandes et un registre destination pour le résultat, **5 bits** sont utilisés pour coder le numéro du registre utilisé.
2) **Les instructions de type** I (**I** pour **I**mmédiate) utilisent aussi **2** opérandes et un registre pour le résultat, la différence avec le type **R** est que l'un de ces opérandes est une immédiate sur **16 bits**, ils sont utilisés pour différents types d'instructions incluant aussi des opérations arithmétiques et logiques.
3) **Les instructions de type J (J** pour **J**ump, ou saut**)** sont réservées pour le saut.



Types d'instruction MIPS

MIPS R3000 possède un jeu de 57 instructions très simples qui se répartissent en 4 classes :

1) 32 instructions arithmétiques/logiques entre registres (calcul)
2) 12 instructions de branchement (saut)
3) 8 instructions de lecture/écriture mémoire (transfert)
4) 5 instructions systèmes

1) Instructions de calcul

Ces instructions lisent la valeur de 0, 1 ou 2 registres dits **arguments**, effectuent un calcul, puis écrivent le résultat dans un registre dit **destination**.
Un même registre peut figurer plusieurs fois parmi les arguments et destination.

Exemples :

- Les instructions de calcul nullaires

- **Ecriture d'une constante** (Load Immediate) :

li dest, constant
Produit la constante *constant*.
On a également : **la** (load adrdess)**.**

- Les instructions de calcul unaires

- **Addition d'une constante** (Add Immediate):

addi dest, source, constant
Produit la somme de la constante (de 16 bits) *constant* et du contenu du registre *source*.

- **Déplacement** (Move) :

move dest, source
Produit le contenu du registre *source*. Cas particulier de *addi!*

- **Négation** (Negate) :

neg dest, source
Produit l'opposé du contenu du registre *source.* Cas particulier de *sub!*

- Les instructions de calcul binaires

- **Addition** (Add):

add dest, source1, source2
Produit la somme des contenus des registres *source1* et *source2*.
On a également : **sub, mul, div.**

- **Comparaison** (Set On Less Than):

slt dest, source1, source2
Produit 1 si le contenu du registre *source1* est inférieur à celui du registre *source2* ; produit 0 sinon.
On a également : **sle, sgt, sge, seq, sne**.

2) Instructions de saut

On distingue les instructions de saut selon que :

- Leurs destinations possibles sont au nombre de 1 (saut **inconditionnel)** ou bien 2 (saut **conditionnel**);
- Leur adresse de destination est **constante** ou bien **lue** dans un registre ;
- Une **adresse de retour** est sauvegarde´ e ou non.

Exemples :

- Saut inconditionnel

- **Saut** (Jump):

j address

Saute à l'adresse constante *address*. Celle-ci est en général donnée sous forme symbolique par une *étiquette* que l'assembleur traduira en une constante numérique.

- Saut conditionnel

- **Saut conditionnel unaire** (Branch on Greater Than Zero):

bgtz source, address

Si le contenu du registre *source* est supérieur à zéro, saute à l'adresse constante *address*.

On a également **bgez, blez, bltz.**

- **Saut conditionnel binaire** (Branch On Equal):

beq source1, source2, address

Si les contenus des registres *source1* et *source2* sont égaux, saute à l'adresse constante *address*.

On a également **bne.**

- Saut avec retour

- **Saut avec retour** (Jump And Link):

jal address

Sauvegarde l'adresse de l'instruction suivante dans le registre *ra*, puis saute à l'adresse constante *address.*

- Saut vers adresse variable

- **Saut vers adresse variable** (Jump Register)

jr target

Saute à l'adresse contenue dans le registre *target*. L'instruction **jr $ra** est typiquement employée pour rendre la main à l'appelant à la fin d'une fonction ou procédure.

3) Instruction de transfert

- **Lecture** (load word):

lw dest, offset(base)

On ajoute la constante (de 16 bits) *offset* à l'adresse contenue dans le registre *base* pour obtenir une nouvelle adresse ; le mot stocké à cette adresse est alors transféré vers le registre *dest*.

On a également: **lb** (load byte)**, lbu** (load byte unisgned)**, lh** (load half)**, lhu** (load half unisgned)**.**

- **Ecriture** (store word):

sw source , offset(base)

On ajoute la constante (de 16 bits) *offset* à l'adresse contenue dans le registre *base* pour obtenir une nouvelle adresse ; le mot stocké dans le registre *source* est alors transféré vers cette adresse.

On a également: **sb** (store byte)**, sh** (store half)**.**

4) Instructions système

- Un appel système se fait par l'instruction **syscall**.
- Les simulateurs fournissent un ensemble de services par l'intermédiaire de l'instruction d'appel **syscall**, dont le comportement dépend de la valeur du registre **$v0** :
  - si **$v0 = 1**, alors **syscall** affiche l'entier contenu dans le registre **$a0** ;
  - si **$v0 = 4**, alors **syscall** affiche la chaîne de caractère dont l'adresse est contenue dans le registre **$a0** ;
  - si **$v0 = 5**, alors **syscall** lit un entier à l'écran et met sa valeur dans le registre **$v0**.

7. Programmation du MIPS R3000

a) Syntaxe d'un programme MIPS:

- Les commentaires commencent par le symbole # et se terminent à la fin de la ligne.
- Un identificateur est une séquence de caractères alphanumériques, de soulignés (_) et de points (.), qui ne commence pas par un chiffre.
- Les codes opération d'instruction sont des mots réservés qui ne peuvent pas être utilisés comme identificateurs.
- Les étiquettes sont déclarées en les plaçant au début d'une ligne et en les faisant suivre du symbole (**:**).
- Les nombres sont en base **10** par défaut. S'ils sont précédés de **0x** ils sont interprétés comme hexadécimaux.
- Les chaînes de caractères sont encadrées par des doubles apostrophes (**"**).
- Certains caractères spéciaux dans les chaînes de caractères suivent la convention **C**:
- Retour-chariot : **\n**
- Tabulation : **\t**
- Guillemet : **\"**

b) La structure de base d'un programme MIPS

- section « **.data** », contient les déclarations de données, c.-à-d. les données globales manipulées par le programme utilisateur. Elle est implantée conventionnellement à l'adresse 0x10000000. Sa taille est fixe et calculée lors de l'assemblage. Les valeurs contenues dans cette section peuvent être initialisées grâce à des directives contenues dans le programme source en langage d'assemblage ;
- La section « **.text** », (code du programme) contient le code exécutable en mode utilisateur. Elle est implantée conventionnellement à l'adresse 0x00400000. Sa taille est fixe et calculée lors de l'assemblage. La principale tâche de l'assembleur consiste à générer le code binaire correspondant au programme source décrit en langage d'assemblage, qui sera chargé dans cette section ;
- « **main** », début du programme.
- Pour sortir du programme **MIPS**, on fait un appel system « **li $v0,10 syscall** ».
- Les commentaires permettent de donner plus d'explication sur le code. Ils commencent par un « # » ou un « **;** ».

Exemple d'un simple programme MIPS

```
1   # Exemple0
2   .data
3   # Il n'y aura pas de données à déclarer, on utilisera le mode immédiat
4   .text
5   main:
6   li $t0,3 # charger la valeur 3 dans le registre t0
7   li $t1,8 # charger la valeur 8 dans le registre t1
8   add $t2,$t1,$t0 # Faire l'addition de t0+t1 et mettre le résultat dans le registre t2
9   #Fin du programme
10  li $v0,10
11  syscall
```

c) Quelques directives

.ascii str	Enregistre en mémoire la chaîne de caractères str, mais ne la termine pas par un caractère nul.
.asciiz str	Enregistre en mémoire la chaîne de caractères **str** et la termine par un caractère nul.
.data<@>	Les éléments qui suivent sont enregistrés dans le segment de données. Si l'argument optionnel **@** est présent, les éléments qui suivent sont enregistrés à partir de l'adresse **@**.
.byte b1; : : : ;bn	Enregistre les **n** valeurs dans des octets consécutifs en mémoire.
.word w1; : : : ;wn	Enregistre les **n** quantités **32 bits** dans des mots consécutifs en mémoire.
.float f1; : : : ;fn	Enregistre les **n** nombres flottants simples précision dans des emplacements mémoire consécutifs.
.text <@>	Les éléments qui suivent sont placés dans le segment de texte de l'utilisateur. Dans **SPIM**, ces éléments ne peuvent être que des instructions ou des mots. Si l'argument optionnel **@** est présent, les éléments qui suivent sont enregistrés à partir de l'adresse **@**.
.globl sym	Déclare que le symbole **sym** est global et que l'on peut y faire référence à partir d'autres fichiers.

d) Appel de procédure - Conventions

- Par convention, lors de l'appel de procédure, les registres **$t0,: : :,$t9** sont sauvegardés par l'appelant et peuvent donc être utilisés sans problème par l'appelé. Les registres **$s0,: : :,$s7** doivent quant à eux être sauvegardés et restitués exact par l'appelé.
- La pile croit des adresses hautes vers les adresses basses : on soustrait à **$sp** pour allouer de l'espace dans la pile, on ajoute à **$sp** pour rendre de l'espace dans la pile.
- Les déplacements dans la pile se font sur des mots mémoire entiers (multiples de quatre octets).
- Lors du passage de paramètres : tout paramètre plus petit que **32 bits** est automatiquement promu sur **32 bits**.
- Les quatre premiers paramètres sont passés par les registres **$a0,: : :,$a3**. Les paramètres supplémentaires sont passés dans la pile.
- Toute valeur de format inférieur ou égal à **32 bits** est retournée par le registre **$v0** (sur **64 bits $v1** est utilisé avec **$v0**).

8. Conclusion

Nous avons présenté dans ce chapitre le rôle du processeur, comment calculer de CPI (Cycle per Instruction), les architectures CISC et RISC ainsi que quelques détails sur le

microprocesseur MIPS R3000 (Structure interne, structure externe, jeu d'instructions) et autres.
D'autres détails concernant la gestion des exceptions et des interruptions, les entrées-sorties ainsi que les instructions systèmes du MIPS R3000 seront détaillés dans le chapitre suivant.