## 4. Bref aperçu sur l'architecture des processeurs MIPS R3000

Le circuit intégré du processeur MIPS R3000 contient deux composants qui sont le processeur principal (CPU) et le processeur secondaire (coprocesseur) CP0 (voir la figure 1.1).

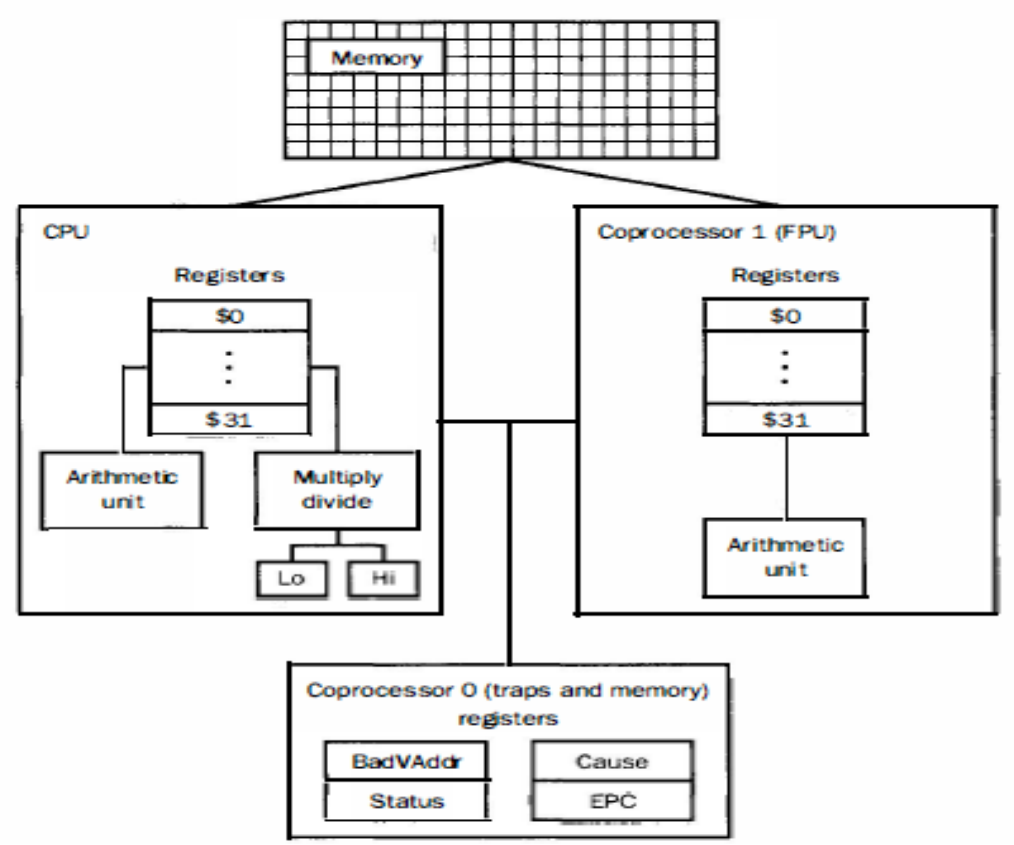


**Figure 1.1. Architecture du processeur MIPS R3000**

| Register Name | Register Number | Register Usage |
|---|---|---|
| $zero | $0 | Hardware set to 0 |
| $at | $1 | Assembler temporary |
| $v0 - $v1 | $2 - $3 | Function result (low/high) |
| $a0 - $a3 | $4 - $7 | Argument Register 1 |
| $t0 - $t7 | $8 - $15 | Temporary registers |
| $s0 - $s7 | $16 - $23 | Saved registers |
| $t8 - $t9 | $24 - $25 | Temporary registers |
| $k0 - $k1 | $26 - $27 | Reserved for OS kernel |
| $gp | $28 | Global pointer |
| $sp | $29 | Stack pointer |
| $fp | $30 | Frame pointer |
| $ra | $31 | Return address |

**Tableau 1.1. Les registres à usage général du CPU.**

Dans ce qui suit on va détailler les registres inclus dans le circuit intégré du processeur MIPS R3000 (le CPU et le CP0).

**Le CPU** contient deux catégories de registres :

**1) La catégorie des registres à usage général** (voir tableau 1.1) qui inclut les 32 registres suivants **(**registre $0 jusqu'au registre $31) :

- **Le registre $0** est mis à 0 par le matériel et ne peut pas être modifié. Les traducteurs (assembleurs et compilateurs) et les systèmes d'exploitation peuvent utiliser les 31 registres restants à leur guise, sachant que tout de même il existe un certain nombre de conventions d'utilisation, destinées essentiellement à permettre de relier plus facilement des procédures rédigées dans différents langages. Ces conventions concernent essentiellement la gestion et l'appel des procédures.

- **Le registre $1** ($at) est utilisé par l'assembleur comme registre temporaire (pour gérer les pseudo-instructions par exemple).

- **Les deux registres $2 et $3** sont utilisés entre autres pour retourner le résultat d'une fonction.

- **Les registres $4, $5, $6 et $7** sont utilisés pour passer les paramètres à une procédure appelée (si ces registres ne suffisent pas, on utilise la pile).

- **Les registres $8 jusqu'au $25** sont utilisés pour recevoir et contenir les données.
  Pour **les registres $8 jusqu'au $15 ainsi que $24 et $25** (*appelés "caller-saved- registers"*), dans le cas où ils sont manipulés par la procédure appelée, l'appelant doit s'occuper de leurs sauvegarde et restitution.
  Pour **les registres $16 jusqu'au $23** (*appelés "callee-saved- registers"*), dans le cas où ils sont manipulés par la procédure appelée, c'est cette même procédure qui doit s'occuper de leurs sauvegarde et restitution.

- **Les deux registres $26 et $27** ($k0 et $k1) sont réservés pour le système d'exploitation.

- **Le registre $28** ($gp) est utilisé pour pointer vers les données globales du programme.

- **Le registre $29** ($sp) est utilisé pour pointer le sommet de la pile du programme.

- **Le registre $30** ($fp) est utilisé pour récupérer les arguments passés dans la pile du programme.

- **Le registre $31** ($ra) est utilisé pour sauvegarder l'adresse de retour lors des appels aux procédures. Dans le cas où une procédure appelée fait elle-même un appel à une autre procédure, elle doit d'abord sauvegarder ce registre dans la pile.

**2) La catégorie des registres à usage spécifique :** qui contient essentiellement les trois registres suivants :

- **Le registre $pc** (compteur ordinale) qui pointe vers l'instruction suivante à exécuter.
- **Les deux registres $hi et $lo** qui sont utilisés dans le contexte des opérations de multiplications et de divisions.

Pour ce qui est du deuxième élément du circuit intégré du MIPS R3000, c'est-à-dire le coprocesseur **CP0**, il contient un ensemble de registres à usage spécifique, utilisés pour la gestion de la mémoire centrale et la gestion des interruptions/exception. On peut citer par exemple les 4 registres suivants (voir la figure 1.1):

- **Le registre $psw (ou $status)** représente le mot d'état du processeur.
- **Le registre $epc** contient l'adresse de l'instruction qui a causée l'exception.
- **Le registre $cause** contient la cause ou le type de l'interruption ou de l'exception.
- **Le registre $Badvaddr** contient l'adresse mémoire dont l'accès a provoquée une exception.

:

| | |
|---|---|
| $pc | Program counter |
| $status or $psw | Status Register |
| $cause | Exception cause register |
| $hi | Used for some multiple/divide operations |
| $lo | |

**Tableau 1.2. Exemple de registres non programmables**

**Remarques :**

1) En plus du coprocesseur CP0, il ya un autre coprocesseur CP1(FPU) spécialisé dans les calcules à virgule flottante (voir la figure 1.1). Ce coprocesseur se trouve en dehors du circuit intégré du MIPS R3000 et se compose de 32 registres 32 bits à usage général et de 5 registres 32 bits à usage spécialisés (pour le contrôle du coprocesseur).

2) Les registres à usage général du CPU et du FPU sont des ***registres programmables***, c'est-à-dire qu'ils peuvent être manipulés directement par le programmeur. Les autres registres à usage spécialisé (appartenant soit au CPU ou aux coprocesseurs CP0 et FPU) ne sont pas programmables, c'est-à-dire qu'ils ne peuvent être modifiés que d'une manière implicite (à travers certaines instructions). Le tableau 1.2 donne des exemples de registres non programmables.

## 5. Disposition d'un programme dans la mémoire centrale

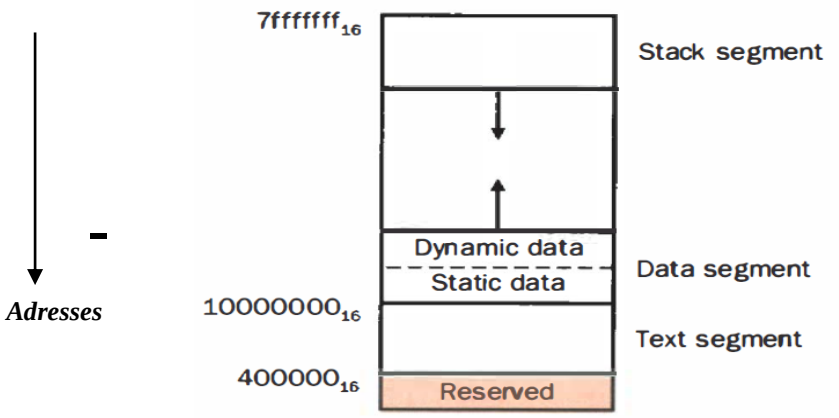


**Figure 1.2. Architecture du processeur MIPS R3000**

La mémoire centrale dans les machines MIPS R3000 est subdivisée en deux parties (voir la figure 1.2) :

1) Partie réservée au système d'exploitation.

2) Partie réservée au programme utilisateur. Cette dernière partie est à son tour subdivisée en trois parties :

- La 1ère partie (à partir de l'adresse $400000_{16}$) : représente le segment de texte ou code qui contient les instructions du programme.

- La 2ème partie (à partir de l'adresse $10000000_{16}$) : représente le segment de données qui inclut les données globales du programme. Cette partie inclut les données statiques (ayant une taille connu et fixe) et les données dynamiques (ayant une taille changeante ou dynamique durant l'exécution).

- La 3ème partie (à partir de l'adresse $7fffffff_{16}$) : représente le segment de pile (pile du programme). Comme pour le cas du segment de données, le segment de pile à une taille maximale inconnu. L'espace alloué à cette pile est étendu vers le segment de données chaque fois que des valeurs sont déposées à sont niveau (voir la figure 1.2).

## 6. Format général d'un programme en assembleur

Le programme assembleur contient 2 segments :

- Le segment de données, débuté par le mot clé « **.data** » .
- Le segment de code, débuté par le mot clé «**.txt** ». Le segment de code contient à sont tour la procédure principale nommé « **main** ». Cette procédure **main** doit clôturer l'exécution par un retour au système d'exploitation (voir la figure 1.3).

Dans le squelette du programme, schématisé par la figure 1.3, ***les lignes en rouge sont obligatoires*** et doivent apparaitre dans tous les programmes abordés durant le TP.

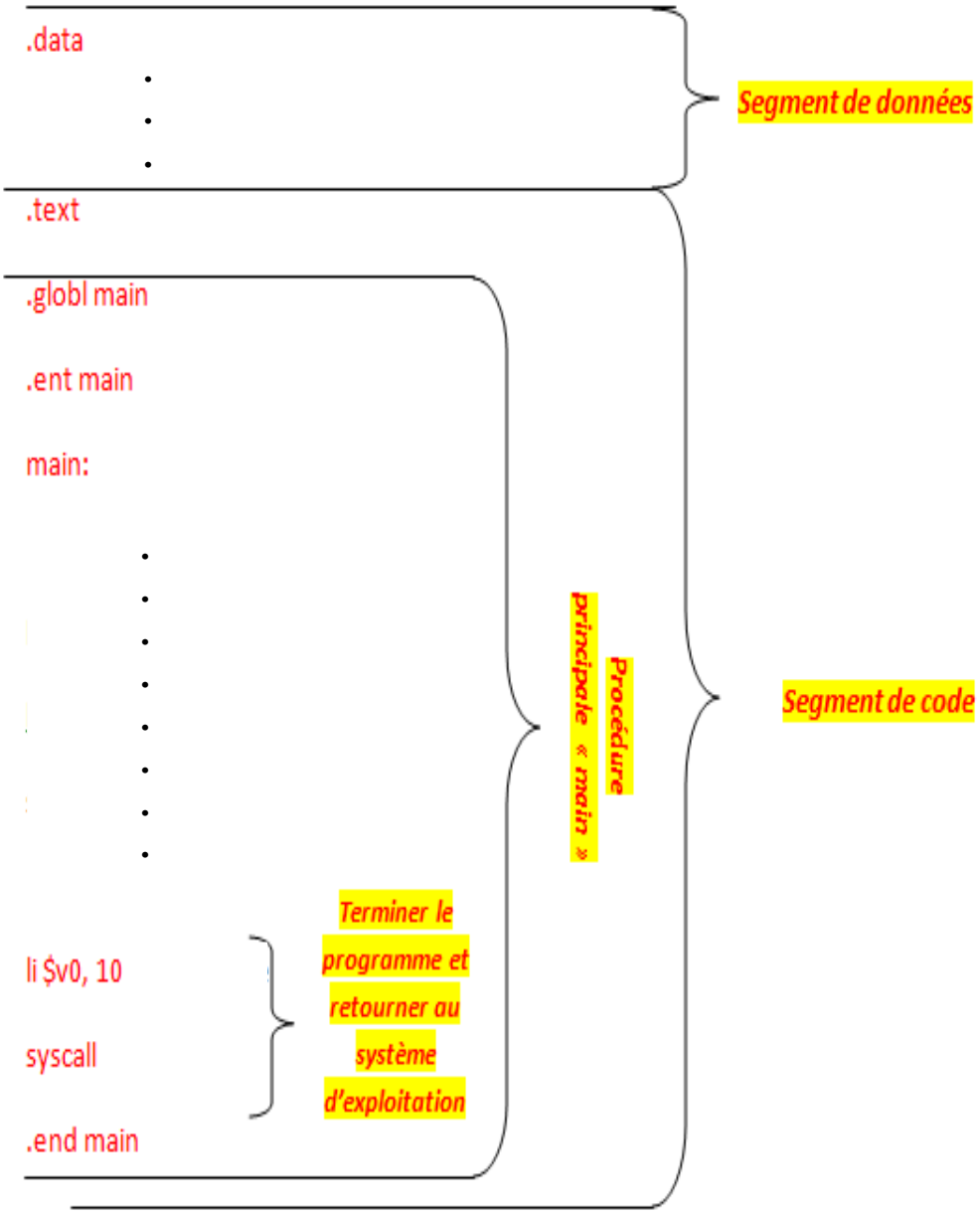


**Figure 1.3. Format général d'un programme assembleur .**

Il est à noter que le programmeur en assembleur MIPS R3000, ne déclare que le segment de code (partie "txt") et le segment des données (partie "data"). Le segment de pile (pile du programme) est créé et ajouté au fichier exécutable par l'outil assembleur (QtSpim).