# Architecture externe du Processeur MIPS32

? Version 3.0
? Septembre 2020
? Alain Greiner (auteur initial)

# A) INTRODUCTION

Ce document présente une version simplifiée de l'architecture externe du processeur MIPS32. Pour des raisons de simplicité, tous les mécanismes matériels de gestion de la mémoire virtuelle ont été délibérément supprimés.

L'architecture externe représente ce que doit connaître un utilisateur souhaitant programmer en assembleur, ou souhaitant écrire un compilateur. Elle définit:

- Les registres visibles du logiciel.
- L'adressage de la mémoire.
- Le jeu d'instructions.
- Les mécanismes de traitement des interruptions, des exceptions et appels système.

Le processeur MIPS32 est un processeur 32 bits conçu dans les années 1980. Son jeu d'instructions est de type RISC. Il existe de nombreuses réalisations industrielles de cette architecture (SIEMENS, NEC, LSI LOGIC, SILICON GRAPHICS, MICROCHIP, etc.)

Cette architecture est suffisamment simple pour présenter les principes de base de l'architecture des processeurs, et

suffisamment puissante pour supporter un système d'exploitation multitâches tel qu'UNIX, puisqu?il supporte deux modes de fonctionnement utilisateur (*user*) et superviseur (*kernel*).

- Dans le mode *user*, certaines régions de la mémoire et certains registres du processeur sont protégés et donc inaccessibles.
- Dans le mode *kernel*, toutes les ressources sont accessibles, c'est-à-dire toutes la mémoire et tous les registres.

L'architecture interne (qui n'est pas présentée ici) dépend des choix de réalisation matérielle. Plusieurs implantations matérielles de cette architecture ont été réalisées à Sorbonne Université dans un but d'enseignement et de recherche : une version microprogrammée, simple mais peu performante (1 instruction s'exécute en 4 cycles d'horloge), une version pipeline plus performante (1 instruction par cycle) mais plus complexe, une version superscalaire, encore plus performante (2 instructions par cycle) mais beaucoup plus complexe.

La spécification du langage d'assemblage, des conventions d?utilisation des registres, ainsi que des conventions d?utilisation de la pile fait l'objet d'un document séparé.

# B) REGISTRES VISIBLES DU LOGICIEL

Tous les registres visibles du logiciel, c'est-à-dire ceux dont la valeur peut être lue ou modifiée par les instructions, sont des registres 32 bits.

Afin de mettre en oeuvre les mécanismes de protection nécessaires pour un système d'exploitation multitâche, le processeur possède deux modes de fonctionnement : utilisateur/superviseur. Ces deux modes de fonctionnement imposent d'avoir deux catégories de registres.

## 1) Registres non protégés

Le processeur possède 35 registres manipulés par les instructions standards (c'est-à-dire les instructions qui peuvent s'exécuter aussi bien en mode *user* qu'en mode *kernel*).

Registres GPR `$0` à `$31`

32 registres généraux (GPR : *General Purpose Register*)
Ces registres sont directement adressés par les instructions, et permettent de stocker des résultats de calculs intermédiaires.
Le registre `$0` est particulier:
? la lecture fournit la valeur constante `0x00000000`
? l?écriture ne modifie pas son contenu.
Le registre `$31` est utilisé par les instructions d'appel de fonctions pour sauvegarder l'adresse de retour.
? Les instructions d'appel de fonctions sont : `bgezal`, `bltzal`, `jal`et `jalr`

Registre `PC`

Registre compteur de programme (*Program Counter*).
Ce registre contient l'adresse de l'instruction en cours d'exécution. Sa valeur est modifiée par toutes les instructions.

Registres `HI` et `LO`

Registres pour la multiplication entière ou la division Euclidienne.
Ces deux registres 32 bits sont utilisés pour stocker le résultat d'une multiplication ou d'une division.

La multiplication de deux nombre de 32 bits est un mot de 64 bits.
La division Euclidienne de deux nombres 32 bits produit un quotient sur 32 bits et un reste sur 32 bits

## 2) Registres protégés

L'architecture du MIPS32 définit 32 registres protégés (numérotés de 0 à 31). Ces registres protégés ne sont accessibles, en lecture comme en écriture, que par les instructions privilégiées `mtc0` et `mfc0` ne pouvant être exécutées qu'en mode *kernel*. `mtc0` et `mfc0` signifient respectivement *Move-To-Coprocessor-0* et *Move-From-Coprocessor-0*. Ces registres appartiennent au "coprocesseur système" (appélé aussi `c0` pour *Coprocessor 0*). En pratique, cette version du processeur MIPS32 en définit 6. Ils sont utilisés par le système d?exploitation pour la gestion des interruptions, des exceptions et des appels système (voir chapitre E).

Registre `c0_sr`
: Le registre `sr` de `c0` est le registre d'état (*Status Register*). Il contient en particulier le bit qui définit le mode : *user* ou *kernel*, ainsi que les bits de masquage des interruptions.
Ce registre a le numéro `$12`.

Registre `c0_cause`
: Le registre `cause` de `c0` est le registre de cause (*Cause Register*). En cas d'interruption, d'exception ou d'appel système, le code en cours d'exécution par le processeur est dérouté vers le noyau du système d'exploitation. Le contenu de `c0_cause` définit la cause d'appel du noyau.
Ce registre a le numéro `$13`.

Registre `c0_epc`
: Le registre `epc` de `c0 est le registre d'exception (*Exception Program Counter*). Il contient soit l'adresse de retour (PC + 4) en cas d'interruption, soit l'adresse de l'instruction courante (`pc`) en cas d'exception ou d'appel système.
Ce registre a le numéro `$14`.

Registre `c0_bar`
: Le registre `bar` de `c0` est legistre d'adresse illégale (*Bad Address Register*). En cas d'exception de type "adresse illégale", il contient la valeur de l'adresse mal formée.
Ce registre a le numéro `$8`.

Registre `c0_procid`
: Le registre `procid` est le registre en lecture seulement contenant le numéro du processeur. Cet index « cablé » est utilisé par le noyau du système d?exploitation pour gérer des architectures multiprocesseurs.
Ce registre possède le numéro `$15`.

Registre `c0_count`
: Le registre `count` de `c0` est le registre en lecture seulement contenant le nombre de cycles exécutés depuis l?initialisation du processeur.
Ce registre possède le numéro `$16`.

# C) ADRESSAGE MÉMOIRE

## 1) Espace d'adressage : adressage par octet

L'ensemble des adresses que peut former le processeur définit son espace d'adressage. Toutes les adresses formées sont des adresses d'octets, ce qui signifie que la mémoire est vue comme un tableau d'octets, qui contient aussi bien

les données que les instructions.

Les adresses sont codées sur 32 bits. Les instructions sont codées sur 32 bits. Les échanges de données avec la mémoire se font par mot (4 octets consécutifs), demi-mot (2 octets consécutifs), ou par octet. Pour les transferts de mots et de demi-mots, le processeur respecte la convention "little endian".

L'adresse d'un mot de donnée ou d'une instruction doit être multiple de 4. L'adresse d'un demi-mot doit être multiple de 2. (on dit que les adresses doivent être "alignées"). Le processeur part en exception si une instruction calcule une adresse qui ne respecte pas cette contrainte.

## 2) Calcul d'adresse

Il existe un seul mode d?adressage, consistant à effectuer la somme entre le contenu d'un registre général Ri, défini dans l'instruction, et d'un déplacement qui est une valeur immédiate signée, sur 16 bits, contenue également dans l'instruction:

```
adresse = Ri + Déplacement
```

## 3) Mémoire virtuelle

Pour des raisons de simplicité, cette version du processeur MIPS32 ne possède pas de mémoire virtuelle, c'est à dire que le processeur ne contient aucun mécanisme matériel de traduction des adresses virtuelles en adresses physiques. Les adresses calculées par le logiciel sont donc transmises au système mémoire sans modifications.

On suppose que la mémoire répond en un cycle. Un signal permet au système mémoire de "geler" le processeur s'il n'est pas capable de répondre en un cycle (ce mécanisme peut être utilisé pour gérer les MISS du ou des caches).

## 4) Protection mémoire

En l?absence de mémoire virtuelle, l?espace mémoire est simplement découpé en 2 segments identifiés par le bit de poids fort de l?adresse :

```
adr 31 = 0      ==>     segment utilisateur
adr 31 = 1      ==>     segment système
```

Quand le processeur est en mode superviseur, les 2 segments sont accessibles. Quand le processeur est en mode utilisateur, seul le segment utilisateur est accessible. Le processeur part en exception si une instruction essaie d'accéder à la mémoire avec une adresse correspondant au segment système alors que le processeur est en mode utilisateur.

Si une anomalie est détectée au cours du transfert entre le processeur et la mémoire, le système mémoire peut le signaler au moyen d'un signal d'erreur, qui déclenche un départ en exception.

# D) JEU D'INSTRUCTIONS

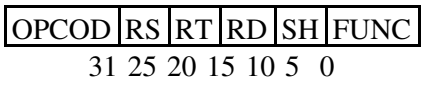
## 1) Généralités

Le processeur possède 57 instructions qui se répartissent en 4 classes :

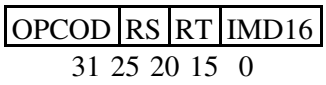
- 33 instructions arithmétiques/logiques entre registres
- 12 instructions de branchement
- 7 instructions de lecture/écriture mémoire
- 5 instructions système

Toutes les instructions ont une longueur de 32 bits et possèdent un des trois formats suivants :

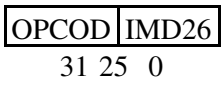
- Le format R est utilisé par les instructions nécessitant 2 registres sources (désignés par RS et RT) et un registre résultat désigné par RD.




- Le format I est utilisé par les instructions de lecture/écriture mémoire, par les instructions utilisant un opérande immédiat, ainsi que par les branchements courte distance (conditionnels).




- Le format J n?est utilisé que pour les branchements inconditionnels lointains




## 2) Codage des instructions

- Le codage des instructions est principalement défini par les 6 bits du champ code opération (appelé OPCOD) de l'instruction (INS 31:26). Cependant, trois valeurs particulières de ce champ définissent en fait une famille d'instructions : il faut alors analyser d'autres bits de l'instruction pour décoder l'instruction. Ces codes particuliers sont : SPECIAL (valeur "000000"), BCOND (valeur "000001") et COPRO (valeur "010000")

| | **INS 28:26** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **INS 31:29** | **000** | **001** | **010** | **011** | **100** | **101** | **110** | **111** |

Par exemple, ce tableau indique que l'instruction LHU a un OPCOD à "100101".

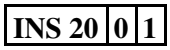
- Lorsque le code opération a la valeur SPECIAL ("000000"), il faut analyser les 6 bits de poids faible de l'instruction (INS 5:0)

| | **INS 2:0** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **INS 5:3** | **000** | **001** | **010** | **011** | **100** | **101** | **110** | **111** |

Par exemple, l'instruction MTLO a un OPCOD à "000000" et le champ FUNC est à "010011".

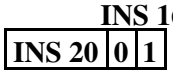
- Lorsque le code opération a la valeur BCOND, il faut analyser les bits 20 et 16 de l'instruction.

**INS 16**




Par exemple, l'instruction BTLZAL a un OPCOD à "000001",
le bit IN20 est à "1" et le bit INS16 est à "0".

- Lorsque le code opération a la valeur COPRO, il faut analyser les bits 25 et 23 de l'instruction. Les trois instructions de cette famille COPRO sont des instructions privilégiées. Remarquez que ERET à deux codages.




Par exemple, l'instruction MTC0 a un OPCOD à "010000",
le bit IN20 est à "0" et le bit INS16 est à "0".

## 3) Jeu d'instructions

Le jeu d'instructions est "orienté registres". Cela signifie que les instructions arithmétiques et logiques prennent leurs opérandes dans des registres et rangent le résultat dans un registre. Les seules instructions permettant de lire ou d'écrire des données en mémoire effectuent un simple transfert entre un registre général et la mémoire, sans aucun traitement arithmétique ou logique.

La plupart des instructions arithmétiques et logiques se présentent sous les 2 formes registre-registre et registre-immédiat:

| instruction assembleur | comportement dans le processeur | format d'instruction |
|---|---|---|
| **add rd, rs, rt** | R(rd) <--- R(rs) op R(rt) | *format R* |
| **addi rd, rs, imm** | R(rd) <--- R(rs) op IMD16 | *format I* |

L?opérande immédiat 16 bits est signé pour les opérations arithmétiques et non signé pour les opérations logiques.

Le déplacement est de 16 bits pour les instructions de branchement conditionnelles (Bxxx) et de 26 bits pour les instructions de saut inconditionnelles (Jxxx). De plus les instructions JAL, JALR, BGEZAL, et BLTZAL sauvegardent une adresse de retour dans le registre $31. Ces instructions sont utilisées pour les appels de sous-programme.

Toutes les instructions de branchement conditionnel sont relatives au compteur ordinal pour que le code soit translatable. L'adresse de saut est le résultat d'une addition entre la valeur du compteur ordinal et un déplacement signé.

Les instructions MTC0 et MFC0 permettent de transférer le contenu des registres SR, CR, EPC et BAR vers un registre général et inversement. Ces 2 instructions ne peuvent être exécutées qu?en mode superviseur, de même que l'instruction ERET qui permet de restaurer l'état antérieur du registre d'état avant de sortir du gestionnaire d'exceptions.

# E) EXCEPTIONS / INTERRUPTIONS / APPELS SYSTÈME

Il existe quatre types d'évènements qui peuvent interrompre l'exécution "normale" d'un programme:

- les exceptions
- les interruptions
- les appels système (instructions SYSCALL et BREAK)
- le signal RESET

Dans tous ces cas, le principe général consiste à passer la main à une procédure logicielle spécialisée (appelée Gestionnaire d?Interrutions, Exceptions et Trappes) qui s'exécute en mode superviseur, à qui il faut transmettre les informations minimales lui permettant de traiter le problème.

## 1) Exceptions

Les exceptions sont des évènements "anormaux", le plus souvent liés à une erreur de programmation, qui empêche l'exécution correcte de l'instruction en cours. La détection d'une exception entraîne l'arrêt immédiat de l'exécution de l'instruction fautive. Ainsi, on assure que l'instruction fautive ne modifie pas la valeur d'un registre visible ou de la mémoire. Les exceptions ne sont évidemment pas masquables. Il y a 7 types d'exception dans cette version du processeur MIPS32 :

ADEL
: Adresse illégale en lecture : adresse non alignée ou se trouvant dans le segment système alors que le processeur est en mode utilisateur.

ADES
: Adresse illégale en écriture : adresse non alignée ou accès à une donnée dans le segment système alors que le processeur est en mode utilisateur.

DBE
: Data bus erreur : le système mémoire signale une erreur en activant le signal BERR à la suite d'un accès de donnée.

IBE
: Instruction bus erreur : le système mémoire signale une erreur en activant le signal BERR à l'occasion d'une lecture instruction.

OVF
: Dépassement de capacité : lors de l'exécution d'une instruction arithmétique (ADD, ADDI ou SUB), le résultat ne peut être représenté sur 32 bits.

RI
: OPCOD illégal : l'OPCOD ne correspond à aucune instruction connue (il s'agit probablement d'un branchement dans une zone mémoire ne contenant pas du code exécutable.

CPU
: Coprocesseur inaccessible : tentative d'exécution d'une instruction privilégiée (MTC0, MFC0, ERET) alors que le processeur est en mode utilisateur.

Le processeur doit alors passer en mode superviseur, et se brancher au Gestionnaire d'Interruptions, Exceptions et Trappes (GIET), implanté conventionnellement à l'adresse "0x80000180". Après avoir identifié que la cause est une exception (en examinant le contenu du registre CR), le GIET se branche alors au gestionnaire d?exception. Toutes les exceptions étant fatales il n'est pas nécessaire de sauvegarder une adresse de retour, car il n'y a pas de reprise de l'exécution du programme contenant l'instruction fautive. Le processeur doit cependant transmettre au gestionnaire d'exceptions l'adresse de l'instruction fautive et indiquer dans le registre de cause le type d'exception détectée.

Lorsqu?il détecte une exception, le matériel doit donc:

- sauvegarder PC (l'adresse de l'instruction fautive) dans le registre EPC
- passer en mode superviseur et masque les interruptions dans SR
- sauvegarder éventuellement l?adresse fautive dans BAR
- écrire le type de l'exception dans le registre CR
- brancher à l'adresse "0x80000180".

## 2) Interruptions

Les requêtes d'interruption matérielles sont des évènements asynchrones provenant généralement de périphériques externes. Elles peuvent être masquées. Le processeur possède 6 lignes d'interruptions externes qui peuvent être masquées globalement ou individuellement. L'activation d'une de ces lignes est une requête d'interruption. Elles sont écrites dans le registre CR, et elles sont prises en compte à la fin de l'exécution de l'instruction en cours si elles ne sont pas masquées. Cette requête doit être maintenue active par le périphérique tant qu'elle n'a pas été prise en compte par le processeur.

Le processeur doit alors passer alors en mode superviseur et se brancher au GIET. Après avoir identifié que la cause est une interruption (en examinant le contenu du registre CR), le GIET se branche au gestionnaire d?interruption, qui doit appeler la routine d?interruption (ISR) appropriée. Comme il faut reprendre l'exécution du programme en cours à la fin du traitement de l'interruption, il faut sauvegarder une adresse de retour. Lorsqu?il reçoit une requête d?interruption non masquée, le matériel doit donc :

- sauvegarder PC+4 (l'adresse de retour) dans le registre EPC
- passer en mode superviseur et masque les interruptions dans SR
- écrire qu'il s'agit d'une interruption dans le registre CR
- brancher à l'adresse "0x80000180".

En plus des 6 lignes d'interruption matérielles, le processeur MIPS32 possède un mécanisme d'interruption logicielle: Il existe 2 bits dans le registre de cause CR qui peuvent être écrits par le logiciel au moyen de l'instruction privilégiée MTC0. La mise à 1 de ces bits déclenche le même traitement que les requêtes d'interruptions externes, s'ils ne sont pas masqués.

## 3) Appels système: instructions SYSCALL et BREAK

L'instruction SYSCALL permet à une tâche (utilisateur ou système) de demander un service au système d'exploitation, comme par exemple effectuer une entrée-sortie. Le code définissant le type de service demandé au système, et un éventuel paramètre doivent avoir été préalablement rangés dans des registres généraux. L'instruction BREAK est utilisée plus spécifiquement pour poser un point d'arrêt (dans un but de déverminage du logiciel): on remplace brutalement une instruction du programme à déverminer par l'instruction BREAK. Dans les deux cas, le processeur passe en mode superviseur et se branche ici encore au GIET. Après avoir identifié que la cause est un appel système (en examinant le contenu du registre CR), le GIET se branche au gestionnaire d?appels système. Lorsqu?il rencontre une des deux instructions SYSCALL ou BREAK, le matériel effectue les opérations suivantes :

- sauvegarder PC (l'adresse de l'instruction) dans le registre EPC (l?adresse de retour est PC + 4)
- passer en mode superviseur et masquage des interruptions dans SR
- écrire la cause du déroutement dans le registre CR
- brancher à l'adresse "0x80000180".

## 4) Signal RESET

Le processeur possède également une entrée RESET dont l'activation, pendant au moins un cycle, entraîne le branchement inconditionnel au logiciel bootloader. Ce logiciel, implanté conventionnellement à l?adresse 0xBFC00000 doit principalement charger le code du système d?exploitation dans la mémoire et initialiser les périphériques. Cette requête est très semblable à une septième ligne d'interruption externe avec les différences importantes suivantes :

- elle n'est pas masquable.
- il n'est pas nécessaire de sauvegarder une adresse de retour.
- le gestionnaire de reset est implanté à l'adresse "0xBFC00000".

Dans ce cas, le processeur doit :

- passer en mode superviseur et masque les interruptions dans SR
- brancher à l'adresse "0xBFC00000"

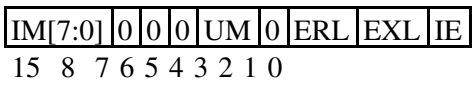
## 5) Sortie du GIET ou du bootloader

Avant de reprendre l'exécution d'un programme qui a effectué un appel système (instructions SYSCALL ou BREAK) ou qui a été interrompu par une interruption, ou pour sortir du bootloader, il est nécessaire d'exécuter l'instruction ERET. Cette instruction modifie le contenu du registre SR, et effectue un branchement à l?adresse contenue dans le registre EPC.

## 6) Gestion du registre d'état SR

Le registre SR contient le mode d'exécution du processeur. Cela concerne son comportement vis-à-vis des signauc d'interruptions, c'est-à-dire les masques, et s'il est en mode *kernel* ou en mode *user*

- La figure suivante présente le contenu des 16 bits de poids faible du registre SR. Cette version du MIP32 n?utilise que 12 bits:




| | |
|---|---|
| **IE** | Interrupt Enable |
| **EXL** | Exception Level |
| **ERL** | Reset Level |
| **UM** | User Mode |
| **IM[7:0]** | Masques individuels pour les six lignes d?interruption matérielles (bits IM[7:2]) et pour les 2 interruptions logicielles (bits IM[1:0]) |

- Le processeur a le droit d?accéder aux ressources protégées (registres du CP0, et adresses mémoires > 0x7FFFFFFF) si et seulement si le bit UM vaut 0, ou si l?un des deux bits ERL et EXL vaut 1.
- Les interruptions sont autorisées si et seulement si le bit IE vaut 1, et si les deux bits ERL et EXL valent 00, et si le bit correspondant de IM vaut 1.
- Les trois types d?évènements qui déclenchent le branchement au GIET (interruptions, exceptions et appels système) forcent le bit EXL à 1, ce qui masque les interruptions et autorise l?accès aux ressources

protégées.
- L?activation du signal RESET qui force le branchement au Boot-Loader force le bit ERL à 1, ce qui masque les interruptions et autorise l?accès aux ressources protégées.
- L?instruction ERET force le bit EXL à 0.

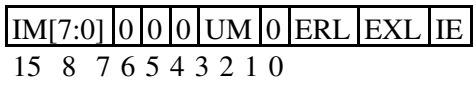
Lors de l?activation du RESET :

- SR contient donc la valeur 0x0004.
- Pour exécuter un programme utilisateur en mode protégé, avec interruptions activées, il doit contenir la valeur 0xFF11.
- Le code de boot doit écrire la valeur 0xFF13 dans SR et l?adresse du programme utilisateur dans EPC avant d?appeler l?instruction ERET.

## 7) Gestion du registre de cause CR

Le registre CR contient trois champs. Les 4 bits du champ XCODE(3:0) définissent la cause de l'appel au GIET. Les 6 bits du champ IRQ(5:0) représentent l'état des lignes d'interruption externes au moment de l'appel au GIET. Les 2 bits SWI(1:0) représentent les requêtes d'interruption logicielle.

- La figure suivante montre le format du registre de cause CR :




- Les valeurs possibles du champ XCODE sont les suivantes :

| | | |
|---|---|---|
| 0000 | **INT** | Interruption |
| 0001 | | Inutilisé |
| 0010 | | Inutilisé |
| 0011 | | Inutilisé |
| 0100 | **ADEL** | Adresse illégale en lecture |
| 0101 | **ADES** | Adresse illégale en écriture |
| 0110 | **IBE** | Bus erreur sur accès instruction |
| 0111 | **DBE** | Bus erreur sur accès donnée |
| 1000 | **SYS** | Appel système (SYSCALL) |
| 1001 | **BP** | Point d'arrêt (BREAK) |
| 1010 | **RI** | OPCOD illégal |
| 1011 | **CPU** | Coprocesseur inaccessible |
| 1100 | **OVF** | Overflow arithmétique |
| 1101 | | Inutilisé |
| 1110 | | Inutilisé |
| 1111 | | Inutilisé |