# Processeur MIPS32

# Architecture externe

Version 2.4

(Octobre 2009)

Alain Greiner

## A) INTRODUCTION

Ce document présente une version légèrement 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 programmeur souhaitant programmer en assembleur, ou la personne souhaitant écrire un compilateur pour ce processeur:

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

Le processeur **MIPS32** est un processeur 32 bits industriel conçu dans les années 80. Son jeu d'instructions est de type RISC. Il existe plusieurs réalisations industrielles de cette architecture (SIEMENS, NEC, LSI LOGIC, SILICON GRAPHICS, 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 multi-tâches tel qu'UNIX, puisqu'il supporte deux modes de fonctionnement : dans le mode **utilisateur,** certaines zones de la mémoire et certains registres du processeur réservés au système d'exploitation sont protégés et donc inaccessibles; dans le mode **superviseur,** toutes les ressources sont accessibles.

L'architecture interne dépend des choix de réalisation matérielle. Plusieurs implantations matérielles de cette architecture ont été réalisées à l'Université Pierre et Marie Curie dans un but d'enseignement et de recherche : une version microprogrammée, simple mais peu performante (4 cycles par instruction), une version pipe-line plus performante (une instruction par cycle), mais plus complexe, une version superscalaire, encore plus performante (2 instructions par cycle) et 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 multi-tâches, 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 standard (c'est à dire les instructions qui peuvent s'exécuter aussi bien en mode utilisateur qu'en mode superviseur).

- **Ri** ($0 \leq i \leq 31$) 32 registres généraux
  Ces registres sont directement adressés par les instructions, et permettent de stocker des résultats de calculs intermédiaires.
  Le registre **R0** est un registre particulier:
  - la lecture fournit la valeur constante "0x00000000"
  - l'écriture ne modifie pas son contenu.
  Le registre **R31** est utilisé par les instructions d'appel de procédures (instructions **BGEZAL**, **BLTZAL**, **JAL** et **JALR**) pour sauvegarder l'adresse de retour.

- **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.

- **HI et LO** Registres pour la multiplication ou la division
  Ces deux registres 32 bits sont utilisés pour stocker le résultat d'une multiplication ou d'une division, qui est un mot de 64 bits.

Contrairement à d'autres processeurs plus anciens, le processeur MIP32 ne possède pas de registres particuliers pour stocker les "codes conditions". Des instructions de comparaison permettent de calculer un booléen qui est stocké dans l'un quelconque des registres généraux. La valeur de ce booléen peut ultérieurement être testée par les instructions de branchement conditionnel.

## 2) Registres protégés

L'architecture MIPS32 définit 32 registres (numérotés de 0 à 31), qui ne sont accessibles, en lecture comme en écriture, que par les instructions privilégiées MTC0 et MFC0 (ces instructions ne peuvent être exécutées qu'en mode superviseur). On dit qu'ils appartiennent au "coprocesseur système" (appélé aussi CP0). 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 systèmes (voir chapitre E).

- **SR** Registre d'état (Status Register).
  Il contient en particulier le bit qui définit le mode : superviseur ou utilisateur, ainsi que les bits de masquage des interruptions.
  (Ce registre possède le numéro 12)

- **CR** Registre de cause (Cause Register).
  En cas d'interruption ou d'exception, son contenu définit la cause pour laquelle on fait appel au programme de traitement des interruptions et des exceptions.
  (Ce registre possède le numéro 13)

- **EPC** Registre d'exception (Exception Program Counter).
  Il contient l'adresse de retour (PC + 4) en cas d'interruption. Il contient l'adresse de l'instruction fautive en cas d'exception (PC).
  (Ce registre possède le numéro 14)

- **BAR** Registre 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 possède le numéro 8)

- **PROCID** Registre en lecture seulement contenant le numéro du processeur.
  Cet index « cablé » est utilisé par le système d'exploitation pour gérer des architectures multi-cores.
  (Ce registre possède le numéro 15)

- **CYCLECOUNT** 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) Adresses octet

Toutes les adresses émises par le processeur sont des adresses 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 consecutifs), 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 logiques 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).
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.

## 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.

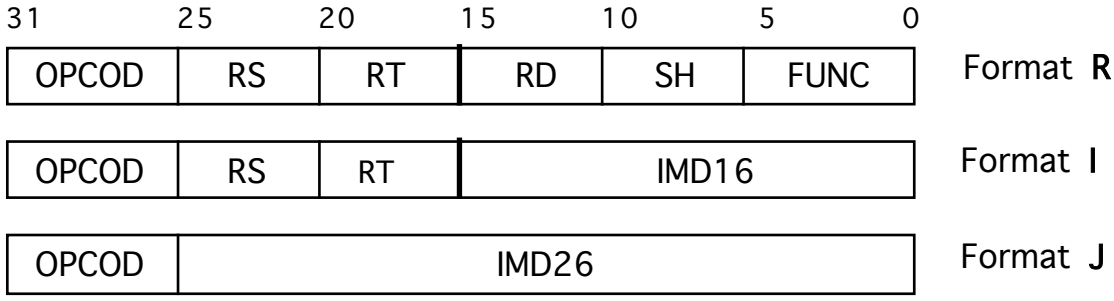
## D) JEU D'INSTRUCTIONS

### 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 systèmes

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




Le format **J** n'est utilisé que pour les branchements à longue distance (inconditionnels).
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 **R** est utilisé par les instructions nécessitant 2 registres sources (désignés par RS et RT) et un registre résultat désigné par RD.

## 2) Codage des instructions

Le codage des instructions est principalement défini par les 6 bits du champs **code opération** 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")

DECODAGE OPCOD

| INS 31 : 29 \ INS 28 : 26 | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 000 | SPECIAL | BCOND | J | JAL | BEQ | BNE | BLEZ | BGTZ |
| 001 | ADDI | ADDIU | SLTI | SLTIU | ANDI | ORI | XORI | LUI |
| 010 | COPRO | | | | | | | |
| 011 | | | | | | | | |
| 100 | LB | LH | | LW | LBU | LHU | | |
| 101 | SB | SH | | SW | | | | |
| 110 | | | | | | | | |
| 111 | | | | | | | | |

Ce tableau exprime que l'instruction LHU (par exemple) possède le code opération "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**):

**OPCOD = SPECIAL**

| INS 5:3 \ INS 2:0 | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 000 | SLL | | SRL | SRA | SLLV | | SRLV | SRAV |
| 001 | JR | JALR | | | SYSCALL | BREAK | | |
| 010 | MFHI | MTHI | MFLO | MTLO | | | | |
| 011 | MULT | MULTU | DIV | DIVU | | | | |
| 100 | ADD | ADDU | SUB | SUBU | AND | OR | XOR | NOR |
| 101 | | | SLT | SLTU | | | | |
| 110 | | | | | | | | |
| 111 | | | | | | | | |

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

**OPCOD = BCOND**

| INS 20 \ INS 16 | 0 | 1 |
|---|---|---|
| 0 | BLTZ | BGEZ |
| 1 | BLTZAL | BGEZAL |

**OPCOD = COPRO**

| INS 25 \ INS 23 | 0 | 1 |
|---|---|---|
| 0 | MFC0 | MTC0 |
| 1 | RFE | |

## 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:

| | | | | | | |
|---|---|---|---|---|---|---|
| ADD | : | R(rd) | <--- | R(rs) | op R(rt) | format R |
| ADDI | : | R(rt) | <--- | R(rs) | op IMD | 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 R31. 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 SYSTEME

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 empèche 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 Codop illégal : le codop 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**, E**RET**) 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 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 ligne 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 l'adresse de retour (PC + 4) 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 de l'adresse de retour (PC + 4) dans le registre **EPC**
- 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 **boot-loader**. 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:

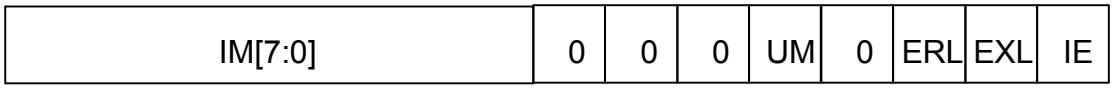
- passer en mode superviseur et masque les interruptions dans **SR**
- brancher à l'adresse "0xBFC00000"

## 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 **bootloade**r, 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

La figure suivante présente le contenu des 16 bits de poids faible du registre **SR**:




Cette version du processeur MIP32 n'utilise que 12 bts du registre SR :

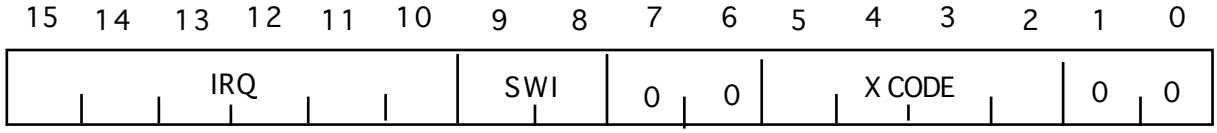
- **IE : Interrupt Enable**
- **EXL : Exception Level**
- **ERL : Reset Level**
- **UM : User Mode**
- **IM[7:0] : Masques individuels pour les six ligne d'nterruption matérielles (bits IM[7:2]) et pour les 2 interruptions logicielles (bits IM[1:0])**

- 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'événements 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 champs **XCODE(3:0)** définissent la cause de l'appel au **GIET.** Les 6 bits du champs **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 champs 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** | **Codop illégal** |
| **1011** | **CPU** | **Coprocesseur inaccessible** |
| **1100** | **OVF** | **Overflow arithmétique** |
| **1101** | | **Inutilisé** |
| **1110** | | **Inutilisé** |
| **1111** | | **Inutilisé** |