

1. 1. Introduction
2. 2. Registres protégés utilisables seulement en mode kernel
3. 3. Adressage de la mémoire
4. 4. Instructions protégées
 1. E) EXCEPTIONS / INTERRUPTIONS / APPELS SYSTÈME
 1. 1) Exceptions
 2. 2) Interruptions
 3. 3) Appels système: instructions `syscall` et `break`
 4. 4) Signal RESET
 5. 5) Sortie du noyau
 6. 6) Gestion du registre d'état `c0_sr`
 7. 7) Gestion du registre de cause `c0_cause`

Documentation MIPS Architecture et assembleur (mode kernel)

1. Introduction

Ce document est la suite du document Documentation MIPS32 architecture et assembleur (mode user) (*Ce document est tiré du document initialement écrit par Alain Greiner*).

Le MIPS supporte deux modes de fonctionnement utilisateur (*user*) et système (*kernel*).

- Dans le mode *user*, certaines régions de la mémoire et certains registres du processeur sont protégés et donc inaccessibles. C'est dans ce mode que s'exécute les applications.
- Dans le mode *kernel*, toutes les ressources sont accessibles, c'est-à-dire toute la mémoire et tous les registres. Dans ce mode, toutes les instructions sont autorisées, à la fois les instructions standards (`add`, `or`, `lw`, `mul`, etc.), mais aussi des **instructions protégées** qui vont permettre de contrôler l'état de fonctionnement du processeur. C'est dans ce mode que s'exécute le noyau du système d'exploitation.

Ce document détaille les éléments de l'architecture du processeur et du langage d'assemblage spécifique au mode *kernel*.

2. Registres protégés utilisables seulement en mode kernel

En mode *kernel*, tous les registres sont accessibles, à la fois les registres non protégés et aussi des registres protégés. Pour rappel, les registres non protégés sont les GPR (\$0 à \$31), le registre PC (accessible implicitement avec les instructions de branchement) et les registres HI et LO. Les registres protégés sont destinés au calcul alors que les **registres protégés** sont destinés au contrôle de l'état du processeur.

L'architecture du MIPS32 définit 32 registres protégés, numérotés de \$0 à \$31, c'est-à-dire comme les registres GPR mais ils ont des instructions d'accès spécifiques. En effet, ces registres protégés ne sont accessibles que par des instructions protégées présentées dans la section 4.

Ces registres appartiennent au "coprocesseur système" n°0 (appelé aussi `c0` pour *Coprocessor 0*). Dans cette version du processeur MIPS32, il y en a 6. Ils sont tous utilisés par le système d'exploitation pour la gestion des interruptions, des exceptions et des appels système. Dans ce document, nous ferons précéder le numéro du registre protégé par `c0_` afin de lever l'ambiguïté.

Le 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 d'exécution du processeur: *user* ou *kernel*, ainsi que les bits de masquage des interruptions. Ce registre a le numéro \$12.

Le 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 programme en cours d'exécution est dérivé vers le noyau du système d'exploitation. Le contenu de `c0_cause` contient un code qui définit la cause d'appel du noyau. Ce registre a le numéro \$13.

Le registre `c0_epc`

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

Le registre `c0_bar`

Le registre `bar` de `c0` est le registre d'adresse illégale (*Bad Address Register*). En cas d'exception de type *adresse illégale*, il contient la valeur de l'adresse mal formée. Une adresse est illégale, par exemple, si vous tentez une lecture de mot (`lw`) à une adresse non-alignée (non multiple de 4) ou si vous tentez une lecture en dehors des segments d'adresse où se trouve de la mémoire. Ce registre a le numéro \$8.

Le registre `c0_procid`

Le registre `procid` est un registre en lecture seule contenant le numéro du processeur. Cet index « câblé » est utilisé par le noyau du système d'exploitation. Il n'a de sens que pour gérer des architectures multiprocesseurs (*multicore*). Ce registre possède le numéro \$15.

Le registre `c0_count`

Le registre `count` de `c0` est le registre en lecture seule contenant le nombre de cycles exécutés depuis l'initialisation du processeur. Ce registre possède le numéro \$16.

3. Adressage de la mémoire

L'espace d'adressage de la mémoire est découpé en 2 parties identifiées par le bit de poids fort de l'adresse (bit n°31). Quand le processeur est en mode *kernel* alors les 2 parties (protégée et non protégée) sont accessibles. Quand le processeur est en mode *user* alors seule la partie protégée est accessible.

bit n°31 de l'adresse = 0

partie non protégée utilisable dans tous les modes du processeur
destinée au programme de l'utilisateur

bit n°31 de l'adresse = 1

partie protégée utilisable seulement en mode kernel
réservée au noyau du système d'exploitation

Quand le processeur est en mode utilisateur, si une instruction essaie d'accéder à la mémoire avec une adresse de la partie système alors le processeur part en exception, c'est-à-dire que le programme fautif est dérouté vers le noyau du système d'exploitation.

4. Instructions protégées

La version du MIPS32 que nous utilisons possède une cinquantaine d'instructions, il y a les instructions standards utilisables quel que soit le mode d'exécution du processeur et il y a les instructions protégées qui ne sont utilisables qu'en mode *kernel*. Les instructions standards sont présentées dans le document sur [l'architecture et l'assembleur en mode user](#). Ce sont les instructions arithmétiques/logiques entre registres, les instructions de branchement, les instructions de lecture et écriture mémoire et l'instruction `syscall`. La présente section décrit les instructions protégées (utilisables seulement en mode *kernel*). Nous en utilisons 3 : `mtc0`, `mfc0` et `eret`.

`mtc0` et `mfc0`

signifient respectivement *Move-To-Coprocessor-0* et *Move-From-Coprocessor-0*. Comme leur nom l'indique, elles permettent de déplacer le contenu des registres entre les bancs (GPR et Copro).

instruction assembleur	comportement dans le processeur	Remarques
<code>mtc0 \$GPR, \$C0</code>	COPRO.0 (\$C0) ? GPR (\$GPR)	\$C0 = \$8, \$12, \$13, \$14, \$15 OU \$16 \$GPR = \$0 ... \$31
<code>mfc0 \$GPR, \$C0</code>	GPR (\$GPR) ? COPRO.0 (\$C0)	\$C0 = \$8, \$12, \$13, \$14, \$15 OU \$16 \$GPR = \$0 ... \$31

`eret`

signifie *Exception-RETurn*, c'est-à-dire *retour d'une exception*. Nous allons voir en détail dans la section 5. ce que cela signifie. Pour le moment, comprenez que c'est l'unique instruction permettant de sortir du mode *kernel* pour entrer dans le mode *user*.

instruction assembleur	comportement dans le processeur	Remarques
<code>eret</code>	PC ? CO_EPC CO_SR.EXL ? 0	copie le contenu du registre CO_EPC (CO_\$14) dans le registre PC et met 0 dans le bit EXL du registre CO_SR (CO_\$12)

Codage des instructions protégées

Elles utilisent toutes le format R avec le champ `OPCOD` à la valeur `COPRO` (c.-à-d. `0b010000`). L'instruction est alors codée avec les bits 25 et 23 de l'instruction (ces deux bits sont dans le champs `RS`). Remarquez que `ERET` à deux codages.

OPCOD	RS	RT	RD	SH	FUNC
-------	----	----	----	----	------

31 25 20 15 10 5 0

INS 23

INS 25 0 1

0 MFC0 MTC01 ERET

Pour les instructions `mtc0` et `mfc0`, le premier argument est mis dans le champs `RT` et le second argument est mis dans le champs `RD`.

instruction	comportement	commentaire
mtc0 RT, RD	<code>C0_RD ? RT</code>	Recopie le contenu du registre GPR n° RT dans le registre du coprocesseur 0 n° RD
mfc0 RT, RD	<code>RT ? C0_RD</code>	Recopie le contenu du registre du coprocesseur 0 n° RD dans le registre GPR n° RT

Par exemple:

`mtc0 $5, $14` l'instruction `mtc0` a un **OPCOD** à `010000`", le bit `INS 25` est à 0 et le bit `INS 23` est à 1.

? `mtc0 $5, $14` = `0b010000|0.1..|00101|01110|. ....|. ....`

? . peut être remplacé par 0 ou 1, utilisons 0

? `mtc0 $5, $14` = `0b010000|00100|00101|01110|00000|000000`

? `mtc0 $5, $14` = `0b0100 0000 1000 0101 0111 0000 0000 0000`

? `mtc0 $5, $14` = `0x40857000`

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 dérouter le programme vers un code spécial (appelée noyau du système d'exploitation) qui s'exécute en mode système et à qui il faut transmettre les informations minimales lui permettant de traiter le problème.

1) Exceptions

Les exceptions sont des évènements "anormaux" détectés au moment de l'exécution des instructions. Ils sont 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, cela signifie que l'on ne peut pas interdire leur gestion. 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 la partie système alors que le processeur est en mode utilisateur.

ADES

Adresse illégale en écriture : adresse non alignée ou accès à une donnée dans la partie 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 à une adresse qui n'a pas de case mémoire associée. On dit qu'elle n'est pas *mappée*. Cette erreur est aussi nommée erreur de segmentation ('segmentation fault' en anglais).

IBE

Instruction Bus Erreur : le système mémoire signale une erreur en activant le signal BERR à l'occasion d'une lecture instruction. C'est le même problème que pour **DBE** mais cela concerne les instructions.

OVF

Dépassement de capacité : lors de l'exécution d'une instruction arithmétique (ADD ou ADDI), le résultat ne peut être représenté sur 32 bits. Par exemple, la somme de 2 nombres positifs donne un nombre négatif.

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

Coprocasseur inaccessible : tentative d'exécution d'une instruction privilégiée (`mtc0`, `mfc0`, `eret`) alors que le processeur est en mode utilisateur.

Dans tous les cas, le processeur doit passer en mode système et se brancher au noyau du système d'exploitation implanté à l'adresse `0x80000180`. Après avoir identifié que la cause est une exception (en examinant le contenu du registre `c0_cause`), le noyau se branche alors au gestionnaire d'exception. Ici, toutes les exceptions sont fatales, il n'y a pas de reprise de l'exécution de l'application 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 `c0_epc` ;
- passer en mode système et masquer les interruptions dans `c0_sr` ;
- sauvegarder éventuellement l'adresse fautive dans `c0_bar` ;
- écrire le type de l'exception dans le registre `c0_cause` ;
- brancher à l'adresse `0x80000180`.

2) Interruptions

Les requêtes d'interruption matérielles sont des événements asynchrones provenant des contrôleurs de périphériques. Elles peuvent être masquées (ignorées) par le processeur. Le processeur MIPS32 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 notifiées dans le registre `c0_cause` et, si elles ne sont pas masquées, elles sont prises en compte à la fin de l'exécution de l'instruction en cours. Cette requête doit être maintenue active par le contrôleur de périphérique tant qu'elle n'a pas été prise en compte par le processeur.

Le processeur doit alors passer en mode système et se brancher au noyau du système d'exploitation. Après avoir identifié que la cause est une interruption (en examinant le contenu du registre `c0_cause`), le noyau se branche au gestionnaire d'interruption qui doit appeler une fonction appropriée pour le traitement de la requête. Cette fonction est appelée routine d'interruption ou ISR (pour *Interrupt Service Routine*). Comme il faut reprendre l'exécution de l'application en cours à la fin du traitement de l'interruption, il faut sauvegarder une adresse de retour.

1) Exceptions

Lorsqu'il reçoit une requête d'interruption non masquée, le matériel doit donc :

- sauvegarder PC+4 (l'adresse de retour) dans le registre `c0_epc` ;
- passer en mode système et masquer les interruptions dans `c0_sr` ;
- écrire qu'il s'agit d'une interruption dans le registre `c0_cause` ;
- brancher à l'adresse `0x80000180` program.

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 `c0_cause` 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 application de l'utilisateur de demander un service au noyau du système d'exploitation, comme par exemple effectuer une entrée-sortie. Le code définissant le type de service demandé au système, et d'éventuels paramètres 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 debuggage du logiciel): on remplace brutalement une instruction du programme à debugger par l'instruction `break`. Dans les deux cas, le processeur passe en mode système et se branche ici encore au noyau. Après avoir identifié que la cause est un appel système (en examinant le contenu du registre `c0_cause`), le noyau 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 `c0_epc` (l'adresse de retour est PC + 4) ;
- passer en mode système et masquage des interruptions dans `c0_sr` ;
- écrire la cause du déroutement dans le registre `c0_cause` ;
- 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 du code de démarrage de l'ordinateur (code de boot). Ce code, implanté à l'adresse `0xBFC00000` doit principalement charger le code du noyau du système d'exploitation dans la mémoire depuis le disque ou le réseau et se brancher à la fonction d'initialisation du noyau. Cette fonction initialise les contrôleurs de périphériques et les structures internes du noyau et, à la fin elle se branche à la première application utilisateur. Dans notre modèle d'ordinateur, le noyau est pré-chargé en mémoire et le code de boot se contente d'appeler la fonction d'initialisation.

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 système et masque les interruptions dans `SR`
- brancher à l'adresse "0xBFC00000"

5) Sortie du noyau

Avant de reprendre l'exécution de l'application qui a effectué un appel système (instructions `syscall`) ou qui a été interrompu par une interruption, il est nécessaire d'exécuter l'instruction `eret`. Cette instruction modifie le contenu du registre `c0_sr`, et effectue un branchement à l'adresse contenue dans le registre `c0_epc`.

6) Gestion du registre d'état `c0_sr`

Le registre `c0_sr` contient l'état du processeur. Cela concerne son comportement vis-à-vis des requêtes d'interruptions, c'est-à-dire les masques des interruptions matérielles et logicielles, et le mode d'exécution, mode système (*kernel*) ou en mode utilisateur (*user*).

- La figure suivante présente le contenu des 16 bits de poids faible du registre `c0_sr`. Cette version du MIPS32 n'utilise que 12 bits:

IM[7:0]	0	0	0	UM	0	ERL	EXL	IE							
15	8	7	6	5	4	3	2	1	0						

IE	Interrupt Enable	0 = toutes les interruptions sont masquées 1 = interruptions non-masquées mais elles peuvent l'être avec IM[7:0]
EXL	Exception Level	0 = aucun effet sur le processeur 1 = le processeur vient d'entrer dans le noyau et donc le processeur est en mode <i>kernel</i> et interruptions masquées
ERL	Error Level	1 = après le signal reset ou certaines erreurs de la mémoire
UM	User Mode	0 = mode d'exécution <i>kernel</i> 1 = mode d'exécution <i>user</i>
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])

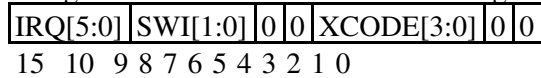
- Quelques remarques :
 - Le processeur a le droit d'accéder aux ressources protégées (registres du coprocessor 0 `c0`), et aux adresses mémoires $\geq 0x80000000$) si et seulement si le bit `UM` vaut 0, ou si l'un des deux bits `ERL` et `EXL` vaut 1.
 - Les interruptions sont autorisées si et seulement 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 noyau: (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 code de boot 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` :
 - `c0_sr` contient donc la valeur `0x0004` (`0b000000000000000100`).
 - Pour exécuter un programme utilisateur en mode protégé, avec interruptions activées, il doit contenir la valeur `0xFF11`.

- ♦ Le noyau doit écrire la valeur 0xFF13 dans `c0_sr` et l'adresse de la première fonction du programme utilisateur dans `c0_epc` avant d'appeler l'instruction `eret`.

7) Gestion du registre de cause `c0_cause`

Le registre `c0_cause` contient trois champs. Les 4 bits du champ `XCODE[3:0]` définissent la cause de l'appel du noyau. Les 6 bits du champ `IRQ[5:0]` représentent l'état des lignes d'interruption externes au moment de l'appel au noyau. 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 (<code>syscall</code>)
1001	BP	Point d'arrêt (<code>break</code>)
1010	RI	OPCOD illégal
1011	CPU	Coprocasseur inaccessible
1100	OVF	Overflow arithmétique
1101		Inutilisé
1110		Inutilisé
1111		Inutilisé