Processeur MIPS R3000.

Architecture externe

Version 2.2

(Septembre 1999)

A) INTRODUCTION

Ce document présente une version légèrement simplifiée de l'architecture externe du processeur **MIPS R3000** (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 connaitre un programmeur souhaitant programmer en assembleur, ou la personne souhaitant écrire un compilateur pour ce processeur:

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

Le processeur **MIPS R3000** 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.

L'architecture interne dépend des choix de réalisation matérielle. Deux implantations matérielles de cette architecture ont été réalisées à l'Université Pierre et Marie Curie (jusqu'au silicium) dans un but d'enseignement et de recherche : la version microprogrammée R3000_M, simple mais peu performante, et la version pipe-line R3000_P, plus performante mais plus complexe. La description de l'architecture interne de ces deux micro-processeurs fait l'objet de deux documents séparés.

Un simulateur de cette architecture, a été également développé à l'Université Pierre et Marie Curie. Ce simulateur accepte en entrée des programmes décrits en langage d'assemblage, et permet de visualiser le comportement du processeur, instruction par instruction. La spécification du langage d'assemblage fait elle aussi 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 R3000 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 **MIPS** définit 32 registres (numérotés de 0 à 31), qui ne sont accessibles, en lecture comme en écriture, que par les instructions privilégiées (c'est à dire les instructions qui ne peuvent être exécutées qu'en mode superviseur). On dit qu'ils appartiennent au "coprocesseur système". En pratique, cette version du processeur **MIPS R3000** en utilise 4 pour la gestion des interruptions et des exceptions (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)

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 R3000 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) Segmentation

L’espace mémoire est 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.

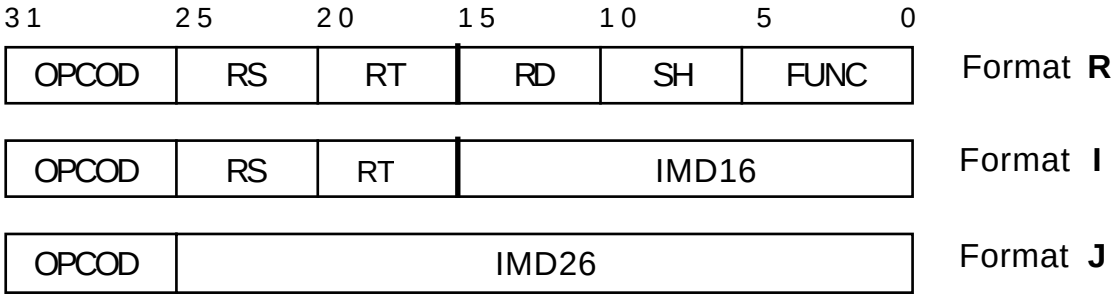
D) JEU D'INSTRUCTIONS

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 RFE qui permet de restaurer l'état antérieur du registre d'état avant de sortir du gestionnaire d'exceptions.

Instructions Arithmétiques/Logiques entre registres

Assembleur	Opération		Format
Add Rd, Rs, Rt	Add overflow detection	Rd <- Rs + Rt	R
Sub Rd, Rs, Rt	Subtract overflow detection	Rd <- Rs - Rt	R
Addu Rd, Rs, Rt	Add no overflow	Rd <- Rs + Rt	R
Subu Rd, Rs, Rt	Subtract no overflow	Rd <- Rs - Rt	R
Addi Rt, Rs,I	Add Immediate overflow detection	Rt <- Rs + I	I
Addiu Rt, Rs,I	Add Immediate no overflow	Rt <- Rs + I	I
Or Rd, Rs, Rt	Logical Or	Rd <- Rs or Rt	R
And Rd, Rs, Rt	Logical And	Rd <- Rs and Rt	R
Xor Rd, Rs Rt	Logical Exclusive-Or	Rd <- Rs xor Rt	R
Nor Rd, Rs, Rt	Logical Not Or	Rd <- Rs nor Rt	R
Ori Rt, Rs, I	Or Immediate unsigned immediate	Rt <- Rs or I	I
Andi Rt, Rs, I	And Immediate unsigned immediate	Rt <- Rs and I	I
Xori Rt, Rs, I	Exclusive-Or Immediate unsigned immediate	Rt <- Rs xor I	I
Sllv Rd, Rt, Rs	Shift Left Logical Variable 5 lsb of Rs is significant	Rd <- Rt << Rs	R
Srlv Rd, Rt, Rs	Shift Right Logical Variable 5 lsb of Rs is significant	Rd <- Rt >> Rs	R
Srav Rd, Rt, Rs	Shift Right Arithmetical Variable 5 lsb of Rs is significant	Rd <- Rt >>* Rs	R
Sll Rd, Rt, sh	Shift Left Logical	Rd <- Rt << sh	R
Srl Rd, Rt, sh	Shift Right Logical	Rd <- Rt >> sh	R
Sra Rd, Rt, sh	Shift Right Arithmetical * : with sign extension	Rd <- Rt >>* sh	R
Lui Rt, I	Load Upper Immediate 16 lower bits of Rt are set to zero	Rt <- I || "0000"	I

Instructions Arithmétiques/Logiques (suite)

Assembleur	Opération		Format
Slt Rd, Rs, Rt	Set if Less Than	Rd <- 1 if Rs<Rt else 0	R
Sltu Rd, Rs, Rt	Set if Less Than Unsigned	Rd <- 1 if Rs<Rt else 0	R
Slti Rt, Rs, I	Set if Less Than Immediate sign extended Immediate	Rt <- 1 if Rs < I else 0	I
Sltiu Rt, Rs, I	Set if Less Than Immediate unsigned immediate	Rt <- 1 if Rs < I else 0	I
Mult Rs, Rt	Multiply	Rs * Rt LO <- 32 low significant bits HI <- 32 high significant bits	R
Multu Rs, Rt	Multiply Unsigned	Rs * Rt LO <- 32 low significant bits HI <- 32 high significant bits	R
Div Rs, Rt	Divide	Rs / Rt LO <- Quotient HI <- Remainder	R
Divu Rs, Rt	Divide Unsigned	Rs / Rt LO <- Quotient HI <- Remainder	R
Mfhi Rd	Move From HI	Rd <- HI	R
Mflo Rd	Move From LO	Rd <- LO	R
Mthi Rs	Move To HI	HI <- Rs	R
Mtlo Rs	Move To LO	LO <- Rs	R

Instructions de Branchement

Assembleur	Opération		Format
Beq Rs, Rt, Label	Branch if Equal	PC <- PC+4+(I*4) if Rs = Rt PC <- PC+4 if Rs ≠ Rt	I
Bne Rs, Rt, Label	Branch if Not Equal	PC <- PC+4+(I*4) if Rs ≠ Rt PC <- PC+4 if Rs = Rt	I
Bgez Rs, Label	Branch if Greater or Equal Zero	PC <- PC+4+(I*4) if Rs ≥ 0 PC <- PC+4 if Rs < 0	I
Bgtz Rs, Label	Branch if Greater Than Zero	PC <- PC+4+(I*4) if Rs > 0 PC <- PC+4 if Rs ≤ 0	I
Blez Rs, Label	Branch if Less or Equal Zero	PC <- PC+4+(I*4) if Rs ≤ 0 PC <- PC + 4 if Rs > 0	I
Bltz Rs, Label	Branch if Less Than Zero	PC <- PC+4+(I*4) if Rs < 0 PC <- PC+4 if Rs ≥ 0	I
Bgezal Rs, Label	Branch if Greater or Equal Zero and link	PC <- PC+4+(I*4) if Rs ≥ 0 PC <- PC+4 if Rs < 0 R31 <- PC+4 in both cases	I
Bltzal Rs, Label	Branch if Less Than Zero and link	PC <- PC+4+(I*4) if Rs < 0 PC <- PC+4 if Rs ≥ 0 R31 <- PC+4 in both cases	I
J Label	Jump	PC <- PC 31:28 || I*4	J
Jal Label	Jump and Link	R31 <- PC+4 PC <- PC 31:28 || I*4	J
Jr Rs	Jump Register	PC <- Rs	R
Jalr Rs	Jump and Link Register	R31 <- PC+4 PC <- Rs	R
Jalr Rd, Rs	Jump and Link Register	Rd <- PC+4 PC <- Rs	R

Instructions de lecture/écriture mémoire

Assembleur	Opération	Format
Lw Rt, I (Rs)	Load Word Rt <- M (Rs + I) sign extended Immediate	I
Sw Rt, I (Rs)	Store Word M (Rs + I) <- Rt sign extended Immediate	I
Lh Rt, I (Rs)	Load Half Word Rt <- M (Rs + I) sign extended Immediate. Two bytes from storage is loaded into the 2 less significant bytes of Rt. The sign of these 2 bytes is extended on the 2 most signficant bytes.	I
Lhu Rt, I (Rs)	Load Half Word Unsigned Rt <- M (Rs + I) sign extended Immediate. Two bytes from storage is loaded into the the 2 less significant bytes of Rt, other bytes are set to zero	I
Sh Rt, I (Rs)	Store Half Word M (Rs + I) <- Rt sign extended Immediate. The Two less significant bytes of Rt are stored into storage	I
Lb Rt, I (Rs)	Load Byte Rt <- M (Rs + I) sign extended Immediate. One byte from storage is loaded into the less significant byte of Rt. The sign of this byte is extended on the 3 most signficant bytes.	I
Lbu Rt, I (Rs)	Load Byte Unsigned Rt <- M (Rs + I) sign extended Immediate. One byte from storage is loaded into the less significant byte of Rt, other bytes are set to zero	I
Sb Rt, I (Rs)	Store Byte M (Rs + I) <- Rt sign extended Immediate. The less significant byte of Rt is stored into storage	I

Instructions Systèmes		
Assembleur	Opération	Format
Rfe	Restore From Exception SR <- SR 31:4 II SR 5:2 Privileged instruction. Restore the prevous IT mask and mode	R
Break n	Breakpoint Trap SR <- SR 31:6 II SR 3:0 II "00" Branch to exception handler. PC <- "8000 0080" n defines the breakpoint number CR <- cause	R
Syscall	System Call Trap SR <- SR 31:6 II SR 3:0 II "00" Branch to exception handler PC <- "8000 0080" CR <- cause	R
Mfc0 Rt, Rd	Move From Control Coprocessor Rt <- Rd Privileged Instruction . The register Rd of the Control Coprocessor is moved into the integer register Rt	R
Mtc0 Rt, Rd	Move To Control Coprocessor Rd <- Rt Privileged Instruction . The integer register Rt is moved into the register Rd of the Control Coprocessor	R

E) EXCEPTIONS ET INTERRUPTIONS

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 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 empèchent 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 R3000 :

- 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**, **RFE**) alors que le processeur est en mode utilisateur.

Le processeur doit alors passer en mode superviseur, et se brancher au **gestionnaire d'exceptions** qui est une routine logicielle implantée conventionnellement à l'adresse "0x80000080". Toutes les exceptions étant fatales dans cette version du processeur R3000, 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'une exception est détectée, le processeur :

- sauvegarde l'adresse de l'instruction fautive dans le registre **EPC**
- sauvegarde l'ancienne valeur du registre d'état **SR**
- passe en mode superviseur et masque les interruptions dans **SR**
- écrit le type de l'exception dans le registre **CR**
- branche à l'adresse "0x80000080".

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 inconditionnellement 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 passe alors en mode superviseur et se branche ici encore au **gestionnaire d'exceptions**. 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'une requête d'interruption non-masquée est détectée, le processeur :

- sauvegarde l'adresse de retour (PC + 4) dans le registre **EPC**
- sauvegarde l'ancienne valeur du registre d'état **SR**
- passe en mode superviseur et masque les interruptions dans **SR**
- écrit qu'il s'agit d'une interruption dans le registre **CR**
- branche à l'adresse "0x80000080".

En plus des 6 lignes d'interruption matérielles, le processeur R3000 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 au **gestionnaire d'exceptions.** Ces deux instructions sont exécutables en mode utilisateur. Elles effectuent les opérations suivantes :

- sauvegarde de l'adresse de retour (PC + 4) dans le registre **EPC**
- sauvegarde de l'ancienne valeur du registre d'état **SR**
- passage en mode superviseur et masquage des interruptions dans **SR**
- écriture de la cause du déroutement dans le registre **CR**
- branchement à l'adresse "0x80000080".

4) Signal **RESET**

Le processeur possède également une ligne **RESET** dont l'activation, pendant au moins un cycle, entraîne le branchement inconditionnel au logiciel 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 :

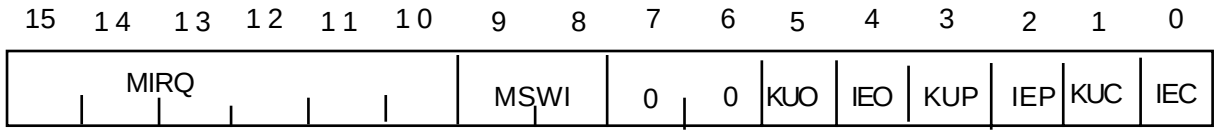
- passe en mode superviseur et masque les interruptions dans **SR**
- branche à l'adresse "0xBFC00000".

5) Retour d'interruption

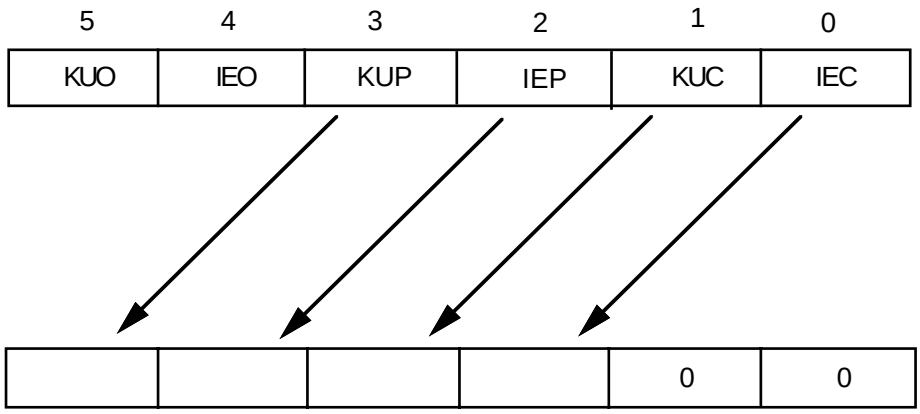
Avant de reprendre l'exécution d'un programme qui a effectué un appel système (instructions **SYSCALL** ou **BREAK**) ou qui a été interrompu, il est nécessaire d'exécuter l'instruction **RFE**.
Cette instruction effectue la restitution de l'état précédent dans le registre **SR**

6) Gestion du registre d'état **SR**

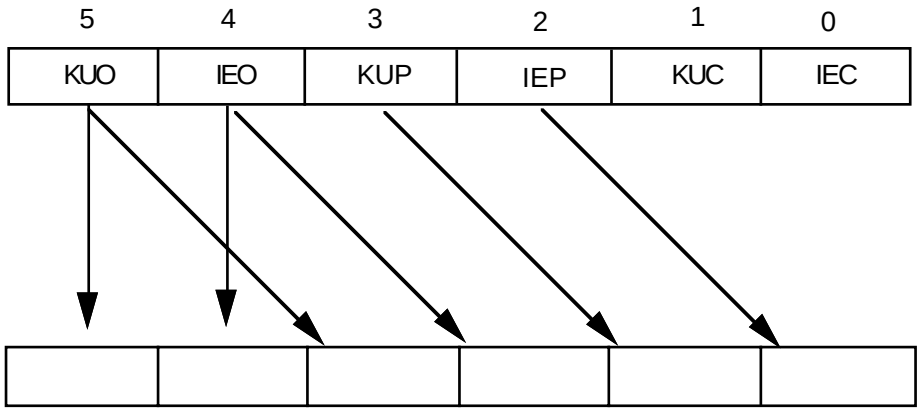
Le registre d'état contient l'état courant du processeur, défini par le bit de mode **KUC** et par le bit de masque global des interruptions **IEC**. La valeur 00 pour les bits **KUC** et **IEC** correspond à l'état superviseur et interruptions masquées. Le registre **SR** contient aussi l'état précédent (bits **KUP** et **IEP**) et l'état antérieur (bits **KUO** et **IEO**). Il constitue donc une petite pile matérielle capable d'empiler 3 états successifs du processeur. Le registre **SR** contient par ailleurs 6 bits **MIRQ(5:0)** permettant de masquer individuellement les 6 interruptions externes et 2 bits **MSWI(1:0)** permettant de masquer les deux interruptions logicielles. Les 16 bits de poids fort et les deux bits 6 et 7 du registre **SR** ne sont pas utilisés: On récupère la valeur 0 en lecture et ils ne sont pas modifiés par les écritures.
La figure suivante montre le format du registre d'état **SR**:



Lors d'un appel au **gestionnaire d'exception**, la sauvegarde de l'ancien état et le passage dans l'état "superviseur avec interruptions masquées" peuvent s'effectuer en une seule opération par décalage de deux bits vers la gauche des six bits de poids faible du registre **SR** (les 8 bits **MIRQ** et **MSWI** ne sont pas modifiés) :

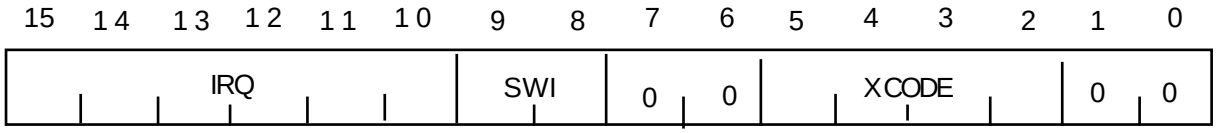


La restitution de l'ancien état, réalisée par l'instruction **RFE**, est obtenue par un simple décalage de deux bits vers la droite des six bits de poids faible du registre **SR** (les 8 bits **MIRQ** et **MSWI** ne sont pas modifiés) :



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 **gestionnaire d'exceptions.** Les 6 bits du champs **IRQ(5:0)** représentent l'état des lignes d'interruption externes au moment de l'appel au **gestionnaire d'exception.** Les 2 bits **SWI(1:0)** représentent les requêtes d'interruption logicielle.
La figure suivante montre le format du registre de cause **CR** :



Les codes d'exceptions sont les suivants :

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**