Chapitre 5 : Instructions spéciales

1- Introduction
2- Notions sur les interruptions,
3- Les entrées-sorties et les instructions systèmes (cas du MIPS R3000)
4- Conclusion

1. Introduction

Les interruptions et les exceptions sont deux types de rupture de séquence exceptionnelle. Les interruptions surviennent de façon asynchrone. Elles sont commandées par le matériel (interruption externe) ou par le système (interruption système).
Les interruptions n'ont aucune relation avec les instructions en cours d'exécution.
Les exceptions sont déclenchées par des accidents dans l'exécution du programme : débordement arithmétique, erreur de bus, tentative d'utilisation d'instructions réservées, erreur d'adressage, défaut de cache, défaut de page...). Les exceptions provoquent un traitement spécifique : appel à une routine spécialisée. On parle de gestion précise des exceptions si l'état de la machine qui résulterait de l'exécution séquentielle de toutes les instructions antérieures à l'instruction provoquant l'exception peut être reconstruit dans tous les cas. Il est alors possible après le traitement de l'exception de reprendre l'exécution à l'instruction même ayant provoqué l'exception. Dans le cas contraire, les exceptions sont gérées de façon imprécise.

2. Notions sur les interruptions

- Certains processeurs possèdent un registre d'état, dont la valeur est mise à jour après l'exécution de chaque instruction, par exemple pour indiquer un fonctionnement normal ou un débordement.
- Le MIPS ne possède pas de registre d'état, mais utilise un système d'exceptions qui, selon les cas, peuvent amener à une interruption de l'exécution du programme.
- La partie contrôle contient neuf bascules 1 bit correspondant aux sept types d'exception définis dans l'architecture du processeur MIPS R3000 (ADEL, ADES, DBE, IBE, OVF, RI, CPU) et aux deux appels systèmes correspondant aux instructions SYSCALL et BREAK (SYS, BRK).
- Ces bascules sont mises à un en cas d'exception lorsque la détection de l'exception correspondante est autorisée par le champ EXCP de la micro-instruction.
- Elles sont toutes remises à zéro si le champ EXCP contient la commande E_CLR, ce qui est normalement fait par la première micro-instruction du microprogramme de branchement au gestionnaire d'exceptions.
- En cas d'exception, le microprogramme de l'instruction fautive se déroule "normalement", jusqu'à la dernière micro-instruction. Cependant, toutes les écritures dans les registres visibles du logiciel et les accès à la mémoire sont inhibées dès qu'une bascule d'exception est à 1.

- La dernière micro-instruction d'une instruction teste s'il n'y a pas une interruption ou une exception pendante.
- Une interruption est pendante si l'une des lignes IT5 à IT0 est active.
- Une exception est pendante si une des neuf bascules d'exception est à 1. Si c'est le cas, au lieu de démarrer l'exécution de l'instruction suivante, l'automate exécute la séquence de micro-instructions permettant de se brancher au gestionnaire d'exceptions.
- Ce microprogramme comporte quatre micro-instructions distinctes et effectue les actions suivantes :
  - passage en mode superviseur et masquage des interruptions dans SR.
  - écriture de la cause du déroutement dans CR et CLR des bascules d'exception.
  - sauvegarde du PC dans EPC.
  - chargement de l'adresse 80000080 dans PC.
  - sauvegarde de AD dans BAR.
  - chargement du registre instruction.
- Dans le cas où une exception et une interruption sont pendantes simultanément, l'exception est traitée en premier.
- Le RESET est traité de la même manière, mais le microprogramme de branchement au gestionnaire d'initialisation est plus simple (trois micro-instructions).
- L'automate complet comporte 106 états.
- Plusieurs exceptions peuvent survenir lors de l'exécution d'un programme : débordements, mémoire non allouée, etc. Par exemple la somme de deux valeurs de 32 bits ne tient pas forcément sur 32 bits. La valeur contenue dans le registre de destination n'est pas alors pas correcte :

Exemple :

```
LI $8, 0xA1234567
LI $9, 0xA1234567
ADD $10, $8, $9      # 0xA1234567 + 0xA1234567 = 0x142468ACE
                     # mais $10 ne peut contenir que 0x42468ACE (32 bits)
                     # débordement => la valeur du registre est fausse!
```

3. Les entrées/sorties

Mode appel système

Le mécanisme d'entrée/sortie de données, par exemple dans la console ou sur un périphérique externe, consiste à interrompre l'exécution du programme pour faire exécuter un service par le système.

Le principe général est le suivant :

1. Placer un code de service dans le registre **$v0** ;
2. Appeler l'instruction **syscall** : l'exécution du programme est interrompue et le système réalise la tâche dont le code est présent dans **$v0**. Une fois le service exécuté, l'exécution reprend.

Les paramètres des services sont lus dans les registres **$a0** à **$a3** (par exemple les valeurs à afficher), et les résultats des services sont placés dans **$v0** et **$v1** à la fin de l'exécution (par exemple les valeurs lues). Les services les plus courants, notamment d'entrée/sortie, sont les suivants :

- Pour demander un service, on charge le code du service (voir tableau ci-dessous) dans le registre **$v0** et ses arguments dans les registres **$a0,...,$a3** (ou **$f12** pour les valeurs flottantes).
- Les appels système qui retournent des valeurs placent leurs résultats dans le registre **$v0** (ou **$f0** pour les résultats flottants).

Le tableau suivant illustre les différents codes de **syscall**.

Service	Code	Arguments	Résultat
print_int	1	$a0 = entier	
print_float	2	$f12 = flottant simple précision	
print_double	3	$f12 = flottant double précision	
print_string	4	$a0 = chaine de caractères	
read_int	5		Un entier dans $v0
read_float	6		Un flottant simple dans $v0
read_double	7		Un flottant double dans $v0
read_string	8	$a0 = tampon, $a1 = longeur	
sbrk	9	$a0 = quantité	Une adresse dans $v0
exit	10		

Exemple 1 : print_int (Appel système code $v0=1)

Le code suivant imprime « **La réponse = 5** » dans la console.

```
.data
Str : .asciiz   "La reponse = "
.text
main:
la $a0, Str     # Mettre l'adresse de la chaine de caractère Str dans le registre $a0
li $v0, 4       # Charger le registre $v0 avec le code 4 pour affichage caractère (print_string)
syscall         # appel système pour affichage caractère
li $a0, 5       # Mettre la valeur à afficher dans le registre $a0
li $v0, 1       # Charger le registre $v0 avec le code 1 pour affichage entier (print_int)
syscall         # appel system pour l'opération affichage entier
# Fin du Programme
li $v0, 10
syscall
```

Exemple2 : print_string (Appel système code $v0 =4)

Le code suivant affiche le message « **Hello Word** » dans la console.

```
.data
Hello: .asciiz "Hello Word\n" # mettre la chaine de caractère « Hello Word » en mémoire
.text
main:           # section <code
la $a0, Hello   # charger le registre $a0 avec l'adresse de la chaine de caractère
li $v0, 4       # Charger le registre $v0 avec le code 4 pour affichage caractère
                #(print_string).
syscall         # appel system pour l'opération d'affichage
# Fin du Programme
li $v0,10
syscall
```

Exemple3 : read_int (Appel système code $v0=5)

Le code suivant affiche sur l'écran un entier saisi au le clavier.

```
.data
Str : .asciiz " Saisir un entier "
.text
main:
la $a0,Str  # Mettre l'adresse de la chaine de caractère Str dans le registre $a0
li $v0,4    # Charger le registre $v0 avec le code 4 pour affichage caractère (print_string)
syscall     # Appel système pour affichage caractère
li $v0,5    # Charger le registre $v0 avec le code 5 pour saisir un entier à partir du clavier
syscall     # Appel system pour l'opération lire entier (read_int)
move $a0,$v0 # déplacer la valeur saisie dans le register $a0
li $v0,1    # Charger le registre $v0 avec le code 1 pour affichage entier (print_int)
syscall     # Appel system pour l'opération affichage entier
# Fin du Programme
li $v0,10
syscall
```

Exemple4 : read_string (Appel système code $v0=8)

Le code suivant affiche le message écrit par l'utilisateur.

```
.data
message: .asciiz "Vous avez écrit "
UserInput: .space 20
.text
main:
# Entrer un text de 20 caractères
li $v0, 8
la $a0, UserInput
li $a1, 20
syscall
# Afficher le message « Vous avez écrit »
li $v0,4
la $a0,message
syscall
# Afficher le message
li $v0,4
la $a0,UserInput
syscall
# Fin du programme
li $v0,10
syscall
```

Mode interruption

Lorsque le périphérique est prêt à effectuer un échange élémentaire, il envoie un signal au CPU pour "interrompre" l'exécution en cours. Le contrôle du CPU est renvoyé à un programme de traitement de l'évènement stocke en mémoire à partir d'une adresse liée à l'interruption.

- Le CPU n'attends pas ;
- Temps perdu à échanger les programmes et leur contexte.
- Un dispositif, incorpore au niveau du séquenceur, qui enregistre et traite les signaux d'interruption envoyés au CPU

Le Cycle d'interruption peut être défini comme suit :

1) Arrêter le programme en cours ;
2) Sauvegarder l'état de la machine ;
3) Exécuter le programme de service de l'interruption ;
4) Rétablir l'état de la machine ;
5) Reprendre l'exécution du programme interrompu.

4. Conclusion

L'objectif de ce chapitre était de présenter quelques instructions spéciales permettant de gérer les exceptions et les interruptions, ainsi que celles permettant la gestion des entrées-sorties et les instructions systèmes dans un processeur MIPS R3000.

Nous avons essayé de décrire ces différentes instructions à l'aide d'exemples qui ont été traités lors des séances de travaux pratiques et ce afin d'expliquer leurs rôles et de faciliter leur compréhensions.