Faculté des Sciences et Ingénierie
Master Informatique
Systèmes Électroniques, Systèmes Informatiques

MOCCA

Méthodes et outils pour conception des circuits numériques

Adrien Bourmault
(adrien.bourmault@etu.upmc.fr)

Enseignant : plusieurs

Table des matières

Chapitre 1

Compléments d'architecture RISC

On va notamment s'intéresser au traitement des interruptions et exceptions. Nous l'étudierons à travers MIPS32.

1.1 Rappels

On a vu en M1 les registres visibles par le logiciel et les mécanismes d'adressage mémoire.

Ce qui avait été ignoré est le mécanisme d'interruptions et exceptions ainsi que le jeu d'instructions système.

1.2 Instructions systèmes

1.2.1 Syscall

Cette instruction permet d'appeler un service du système d'exploitation. C'est une instruction de format R.

En effet, le système d'exploitation est là pour gérer le matériel et en distribuer les ressources au logiciel. Pourquoi ça existe ? Parceque gérer mal le matériel peut le détruire.

Comment le système sait quel service est demandé ? Il y a l'ABI : une convention d'appel pour passer des paramètres à l'instruction `syscall`. En MIPS, on met le numéro identifiant le service demandé dans le registre `r2`.

1.2.2 Break

C'est basiquement un `syscall` spécialisé pour les logiciels de débogage (comme `gdb`).

Note : en RISC on peut insérer un break à la place de n'importe quelle instruction sans détruire le programme car toutes les instructions font la même taille. Pour que cette instruction soit quand même exécutée, c'est le débogueur qui va l'exécuter séparément du programme.

L'instruction est de format R.

1.2.3 Eret

C'est l'instruction **privilégiée** qui permet de quitter le vecteur d'interruption/exception et on ne peut exécuter cette instruction qu'en mode système. C'est une instruction de format R*, c'est à dire un format spécial.

1.2.4 Le registre STATUS

C'est le registre de contrôle du processeur. Il y a différentes sections et notamment `mode` qui indique le mode (S/U) et la raison du passage en S s'il y a lieu.

1.2.5 Mfc0

C'est l'instruction **privilégiée** qui permet de récupérer le contenu d'un registre du coprocesseur 0 vers un registre général. C'est une instruction de format I*.

1.2.6 Wait

C'est l'instruction **privilégiée** qui permet de stopper le pipeline jusqu'à ce qu'une interruption soit reçue. Cette instruction permet d'éviter de consommer de l'énergie (excepté pour l'horloge qui consomme toujours un petit peu).

La formule de la consommation est $E = \frac{1}{2}CV^2\alpha$ avec C la capacité, V TODO et α l'activité du signal.

1.2.7 Teqi, Tlge, Tgeiu, etc

Ces instructions **privilégiées** sont des appels système conditionnels. C'est une instruction de format I*.

1.2.8 Concernant la mémoire

Il y a aussi énormément d'instructions mémoires que nous n'avions pas étudiées. Elles seront détaillées dans les diapos fournies en annexe.

1.3 Exceptions, interruptions et reset

Ces mécanismes ne sont pas liés à l'exécution des instructions mais au contrôle général du processeur.

1.3.1 Reset

Il s'agit de réinitialiser tous les composants de la machine. Cela a pour effet :
— d'annuler l'exécution du programme courant ;
— effacer la mémoire totalement ;
— tous les programmes courants sont perdus (y compris l'OS).

Pour ce faire, cette instruction consiste en un saut vers un programme spécifique en mémoire nommé *Reset Handler*. L'algorithme de ce programme est le suivant :
— initialiser l'adresse de la prochaine instruction à `0xBFC0 0000` ;
— initialiser le registre STATUS à la valeur `0x0040 0004` i.e avec le champ *mode* à `00 10` ;
— initialiser le registre CAUSE à la valeur `0x0000 0000` ;
— sauvegarde de l'adresse de retour dans EEPC (pour le reset à chaud) ;
— initialiser le registre EBASE (Exception Base register) à `0x8000 0000`.

A l'adresse `0xBFC0 0000` se trouve un système d'exploitation simple permettant de traiter les interruptions urgentes.

1.3.2 Interruptions et exceptions

La différence entre les deux est importante. L'interruption est un évènement qui nécessite que le processeur effectue une opération. Une exception est un évènement qui indique un dysfonctionnement d'un programme ou de la machine et qui requiert l'intervention du processeur.

Ainsi, les interruptions sont des évènements normaux alors que les exceptions ne devraient pas arriver... ;)

1.3.2.1 Interruptions

L'instruction `interrupt` déclenche une interruption. Cela se fait en sautant au vecteur d'interruption à l'adresse `EXH-ADR` puis en retournant au code courant pointé par le registre `EPC`. Ce mécanisme est détaillé dans les diapos en annexe.

Pendant les sauvegardes et les restaurations de contexte, aucune interruption n'est traitée (en utilisant les masques).

Pour revenir d'une interruption qui se produit pendant un branchement, on est obligé de ré-exécuter le branchement (sinon le *delayed slot* n'est pas exécuté correctement). On initialise donc dans le registre CAUSE le bit adéquat pour le signaler (Branch Delayed Slot).

1.3.2.2 Exceptions

Le principe est que lorsqu'une erreur se produit dans l'exécution d'un programme, une exception se produit et l'*Exception Handler* décide soit de tuer le programme, soit de revenir au programme.

Pour préserver l'intégrité du système, l'instruction fautive ne doit pas être exécutée. Mais comment alors savoir qu'elle est illégale ?

Le mécanisme est détaillé dans les diapos en annexe. Il y a beaucoup de causes d'exception, détaillées également au même endroit.

Fun-fact : l'instruction `syscall` déclenche une exception, ce qui veut dire que `syscall` est une instruction du jeu d'instruction que le processeur n'a pas le droit d'exécuter ?! En fait dès que le mécanisme est en place, exécuter l'instruction n'a pas de sens (elle ne fait rien que déclencher l'exception). C'est la même chose pour les `trap` et `break`.