Prédicteur de Branchement

introduction

Tout commence à partir du *Program Counter* (PC). Il contient la prochaine adresse du programme. L'étage **Ifetch** demande au cache un paquet d'instructions. Pour ce faire, il envoie au cache une adresse alignée sur une frontière de la taille d'un bloc de ligne du Icache. Ceci permet de n'avoir besoin que d'un seul port de lecture au cache par interface avec le Icache (sinon, un paquet d'instructions peut chevaucher 2 blocs). Le paquet retourné par le cache est stocké dans une file d'attente nommée **Ifetch queue**

En paralèlle, le PC est envoyé au prédicteur de branchements. Le rôle de ce dernier est triple :

- déterminer la présence d'un branchement (le prochain paquet contient il branchement?)
- déterminer la destination du branchement
- déterminer la direction du branchement (Est ce que le mécanisme de génération de la prochaine adresse doit prendre l'adresse en séquence ou prendre l'adresse de destination du branchement)

Présence d'un branchement

Posons comme hypothèse que la section code d'un programme est en lecture seule, ceci implique que si le processeur décode un branchement à une adresse donnée, alors pour tous cycles, il y aura toujours le même branchement à cette même adresse. Ce principe est exploité dans les caches et ce base sur la localité temporelle (il y a de grande probabilité pour qu'un branchement soit réutilisé dans un avenir proche).

Ce cache ce nomme couramment **Branch Target Buffer** (BTB) (1983_lee). La partie contrôle de ce cache sert à déterminer la présence d'un branchement. Avec les hypothèses données, un hit informe la présence d'un branchement, alors qu'un miss informe soit l'absence d'un branchement, ou un branchement mais jamais rencontré ou évincé car rarement utilisé. Ceci est une prédiction de la présence d'un branchement.

Destination

Déterminer la présence la présence d'un branchement n'est utile que si nous pouvons déterminer la destination. Nous pouvons regrouper les instructions de l'OpenRisc en trois groupes :

1. **l.bf, l.bnf, l.jal** Ces instructions font un déplacement relatif par rapport au PC. Ce déplacement est contenus dans l'encodage de l'instruction. L'adresse de destination est donc fixe au cours du temps.

2. **l.jr r!=LR, l.jalr** Ces instructions réalisent un déplacement absolue. La destination est contenu dans un registre. Il est calculé pendant l'exécution du programme. Deux exécutions du programme avec un environnement différent ne donne pas les mêmes destinations de sauts.
3. **l.jr r=LR** Ceci est un cas spécial du précèdent : les instructions l.jal et l.jalr sont des appels de procédure. Elle sauvegarde dans le registre *Link Register* (LR) l'adresse de retour du branchement. Donc l'instruction l.jr avec r = LR est un retour de procédure. Pour chaque appel de procédure il y a un retour de procédure.

Pour le premier cas, l'adresse de destination étant donné par l'instruction de branchement et restera inchangée au cours du temps, nous pouvons l'enregistrer dans le BTB. Le numéro de voie fournit par la partie contrôle permet de sélectionner la bonne adresse de destination.

Pour le deuxième cas, nous allons réutilisons l'adresse stocké dans le BTB. (Il y a donc 2 ports d'écritures pour mettre à jour l'adresse : le premier au moment de l'étage décode, le second au moment à l'étage commit.

Pour le troisième cas, nous nous référons au travaux de Kaeli et Emma (1991_kaeli) qui constatent que pour chaque appel de procédure, il y a un retour de procédure associé. Pour ce faire, nous utilisons une pile qui va stocker les adresses de retour des appels de procédure. Cette pile porte en général le nom de **Return Address Stack** (RAS). A chaque appel de procédure, nous allons empiler l'adresse de retour (PC + 8 (ne pas oublier le Delayed Slot)). Si la pile est pleine, la plus ancienne sera écraser. Dans le cadre d'un retour de procédure, nous dépilons alors le sommet de pile et sera l'adresse de retour. Si la pile est vide, le prédicteur de branchement informe l'étage **Ifetch** d'attendre de ne plus être spéculatif. Ceci est réaliser en informant que le prédicteur est occupé.

Afin de sélectionner la bonne adresse de destination, nous devons connaître le type du branchement. Le type sera donc également enregistré dans la BTB.

Direction

Le troisième rôle d'un prédicteur de branchement est de déterminer si la partie génératrice d'adresse doit utiliser une adresse en séquence ou doit utiliser l'adresse fournit par le prédicteur de branchement. Pour ce faire, le type de l'instruction est importante : les sauts (l.j, l.jr, l.jal, l.jalr) vont toujours prendre alors que les branchements conditionnels (l.bf et l.bnf) vont dépendre de leur historique.

Les articles 1992_yeh, 1992_pan et 1993_yeh définissent un schma gnral des prdicteurs de branchements deux niveaux. Selon les notations de 1993_yeh, le premier niveau appelé **Branch History Table** (BHT) est une table contenant des registres à décalage. Ces derniers représentent les *c* derniers directions de branchements survenus. Le deuxième niveau est appelé **Pattern History Table** (PHT) qui est une table de compteur à saturation de *d* bits. Ce compteur permet de sauvegarder le comportement des 2^d dernières occurrences.

La figure \label{MORPHEO_component-prediction_unit-predictor} présente un prédicteur 2 niveaux généralistes. Par la suite les BHT et PHT sont considéré comme des structures à correspondance direct. Ceci permet d'économiser de la surface en ne rajoutant aucun Tag. De plus le gain est négligeable. Nous allons maintenant approfondir les deux blocs principaux du choix de la direction : la BHT et la PHT.

Branch History Table

Le premier niveau de prédiction contient 2^a registres à décalages. Ces registres contiennent les résultats des branchements de *c* dernier branchement. La table est indexée par les *a* bits les moins significatif de l'adresse du branchement. Selon la nomenclature de Yeh et Patt (1992_yeh, 1993_yeh), si $2^a=1$, alors il y a un seul historique pour tout les branchements, le BHT est dit Prédicteur Global (Global Branch History Register (GBHR)). Sinon le Prédicteur est dit Local (Per address Branch History Table (PBHT)). Il existe une troisième version, le Par ensemble (Per set Branch History Register (SBHT)) mais dont les ensembles sont déterminées par le compilateur

ou à partir de l'adresse du branchement, ou encore par un numéro de contexte.

Pattern History Table

Le second niveau du prédicteur contient 2^d compteur à saturation de e bits (sature à 0 en cas de décrémentation et à 2^{e-1} dans le cas d'une incrémentation). Le registre est incrémenter dans si le branchement est prit, et est décrémenté dans l'autre cas. Le bit le plus significatif indique la prédiction à prendre : 1 pour prendre, 0 pour continuer en séquence. Le bloc **index generator** va déterminer le compteur à choisir. Pour ce faire il prend les b bits les moins significations de l'adresse ainsi que la sortie du BHT (soit c bits) et va réaliser un ou exclusif sur l'intersection de ces deux adresses. Le tous revoie une valeur sur d bits. Si l'intersection est nulle, on dit qu'il y a une sélection d'index (Predictor with Index Selection), sinon on dit que l'index est partagée (Predictor with Index Sharing).

Avec la même nomenclature de la sous section précédente, si $2^b=1$, alors il y a un seul compteur par historique (Global Pattern History Table (GPHT)). Sinon la structure est appelé Per-address Pattern History Table (SPHT) ou Per-set Pattern History Table (PPHT) suivant qu'il y a un pattern par branchement ou par ensemble de branchement.

Il existe un cas spécial : si le BHT n'existe pas ($c = 0$), alors il y a uniquement l'adresse qui indexe le PHT. Dans ce cas, le prédicteur est alors nommé **Prédicteur bimodal**.

Intégration

Chaque prédicteur est efficace sur un type précis pattern. Par exemple les prédicteurs locaux sont performants pour les branchements qui exécute des patterns répétitifs (L'exemple le plus commun étant la boucle énumérée (for)), alors que les prédicteurs globaux sont efficaces pour par exemple des boucles imbriqués.

De cette constatation, McFarling 1993_mcfarling à conçut le prédicteur combiné (appelé également le méta prédicteur). L'objectif de ce prédicteur est de tirer partie de plusieurs prédicteurs différent. Dans son concept général (voir figure \label{MORPHEO_component-prediction_unit-meta_predictor}), il y a 3 prédicteurs. 2 qui vont prédire la direction à prendre, et le dernier qui va prédire quel prédicteur à le plus de chance d'avoir la bonne prédiction. Ce dernier prédicteur est mis à jour par rapport au succès de sa prédiction. Cette structure peut être récursive : un prédicteur appartenant au méta prédicteur peut lui-même être un méta prédicteur.

Dans la version définit dans 1993_mcfarling, il s'agit d'un prédicteur bimodal qui va sélectionner entre un prédicteur global et un prédicteur local.

Mise à jour

Pour mettre à jour les predicteurs de branchements, il faut respecter les résultats des prédictions dans l'ordre de séquence du programme. En particulier pour les registres d'historiques. Fonctionnellement, ils imposent fortement la contrainte d'insertion dans l'ordre du programme. Il est à noter également que les prédictions qui sont effectuées dans le mauvais chemin de la prédiction ne doivent pas mettre à jour les prédicteurs.

Dans un processeur à exécution dans le désordre (*Out Of Order*), le **Re Order Buffer** est la structure qui va permettre de valider les instructions dans l'ordre. Allons nous utiliser cette structure pour mettre à jour le prédicteur de branchement? Pour répondre à cette question, nous allons déterminer les informations nécessaires à sauvegarder pour la mise à jour.

La BHT et la PHT contiennent un banc de registres, dont les registres sont gérés respectivement comme des registres à décalage ou des compteurs à saturation. Si nous souhaitons minimiser le nombre de ports pour ces bancs de registres, nous devons sauvegarder le contenu du registre (soit respectivement c et e bits). De plus pour mettre à jour le bon registre, il faut posséder l'index. Ce qui augmente de respectivement a et d bits. Or pour le PHT, la

génération de l'adresse est en fonction de la sortie du BHT, donc au lieu de sauvegarder l'index complet, il suffit de sauvegarder la partie de l'adresse nécessaire dans ce calcul, soit *b* bits. De plus, *a* et *b* sont les bits les moins significatifs de l'adresses. Nous avons juste besoin de l'union de cette deux champs. Finalement pour un prédicteur à 2 niveaux, il faut $max(a,b)+c+e$ bits pour le mettre à jour.

Comme le Re Order Buffer ne contient pas que des branchements, nous avons décider d'implémenter une table ne contenant que les branchements et qui ce nomme **Branch Update Table** (BUT). Elle est gérée de la même façon que le Re Order Buffer.