

Un microprocesseur RISC 16 bits

Présentation générale

On choisit un format de 16 bits pour la simplicité du dessin final. Pour respecter la philosophie RISC, adresses et données sont donc codées sur 16 bits. Toutes les instructions machines sont également codées sur 16 bits, *opérandes compris*.

La première partie consiste à définir le jeu d'instructions, le format du mot d'instructions, et à l'essayer sur quelques exemples. La seconde partie construit le processeur lui-même.

Quel est l'espace d'adressage de ce processeur ? Quelle est la taille mémoire qu'il peut adresser ? Combien d'instructions aura-t-il au maximum ?

Dessiner la boîte noire de ce processeur, comportant tous les signaux d'interface avec la mémoire (on ignore les questions d'*interruptions* des processeurs réels).

1 Le jeu d'instructions

On va découper le mot d'instruction en différents *champs* codant (entre autre) l'instruction à effectuer, ses différents opérandes, etc. On s'attachera à respecter le principe d'*orthogonalité* de la philosophie RISC, qui dit que ce découpage doit être constant, même pour des instructions très différentes.

Justifier le principe d'orthogonalité.

1.1 Choix du nombre de registres

Il y a trois grandes architectures possibles pour la partie calcul d'un processeur :

- *Machine à trois adresses* : implémente des instructions de type $Rd \leftarrow Rx \text{ op } Ry$, où Rd est le registre destination, Rx et Ry sont les registres opérandes, et op est une opération arithmétique ou logique.
- *Machine à deux adresses* : le registre destination est obligatoirement un des registres opérandes, donc les instructions sont du type $Rx \leftarrow Rx \text{ op } Ry$.
- *Machine à une adresse, ou machine à accumulateur* : toute opération met en jeu un registre spécial, l'accumulateur, qui à la fois destination et l'un des opérandes, soit $Acc = Acc \text{ op } Ry$.

Question (comptée 0.25 point) de culture peu générale : donner des exemples de processeurs (réels) qui sont des machines à trois, deux ou une adresse.

Donner le nombre de bits que va nécessiter le codage des opérandes et de la destination dans notre mot d'instruction, en fonction du choix d'architecture, et du nombre de registres du processeur. Discuter. Quels vous paraissent les meilleurs choix si l'on se limite à 8 bits pour les champs correspondants ?

Dans la suite on choisit de construire une machine à deux adresses. La forme de base pour le mnémonique d'une instruction sera donc de type OPM R_x, R_y , où OPM est le mnémonique de l'opération (on choisit de l'écrire en trois lettres).

1.2 Opérations mémoire

La philosophie RISC distingue bien les opérations de calcul, dont opérandes et destination ne sont que des registres, et les opérations d'accès à la mémoire, qui n'effectuent aucun calcul.

Discuter les avantages et inconvénients de cette approche.

On définit les deux opérations mémoire LDR R_x, R_a et STR R_x, R_a qui chargent ou déchargent le contenu du registre R_x à l'adresse R_a .

Il y a plusieurs moyens d'implémenter une instruction JMP (saut), chacun avec une architecture assez différente. Lesquels ? Choisissez.

1.3 Choix des instructions arithmétiques et logiques

Nous allons maintenant définir les quelques bits de notre mot d'instruction codant l'instruction (le champ instruction).

En plus des deux opérations sur la mémoire précédentes, on aura des opérations de calcul entier, de calcul binaire, une instruction de copie de registres, des décalages, etc.

Définir un jeu d'instructions minimal (ou en tout cas très *reduced*), mais permettant par exemple de programmer en assembleur multiplication et division (même si cet exercice n'est pas complètement possible à ce stade).

Si le nombre d'instructions obtenu est différent de 8, 16 ou 32, ajouter ou on retirer des instructions jusqu'à bien remplir le champ instruction.

Discuter les différentes possibilités pour faire des opérations par des constantes.

Récapituler précisément le jeu d'instructions et discuter son codage le plus simple possible dans le champ instruction.

1.4 Contrôle d'exécution

Il doit normalement nous rester quelques bits libres parmi les 16 de notre mot d'instruction. Ils vont servir à contrôler l'exécution par des *drapeaux*, produits par l'UAL et testables par les instructions processeur.

Par souci d'orthogonalité, toutes les instructions seront (optionnellement) conditionnelles : un champ condition, dans notre mot d'instruction, contiendra un codage de la condition sous laquelle l'instruction sera effectuée. Le mnémonique d'une instruction conditionnelle sera donc de type OPM R_x, R_y IFcond.

Les trois drapeaux de base de toute UAL sont :

- *Z*, qui vaut 1 si le résultat d'une opération est nul, et 0 sinon,
- *P*, qui vaut 1 si le résultat est strictement positif, et 0 sinon,
- *C*, qui vaut 1 s'il y a dépassement de capacité, et 0 sinon.

Exprimer les conditions suivantes en fonction des drapeaux : résultat positif ou nul, négatif ou nul, strictement positif, strictement négatif, dépassement de capacité, pas de dépassement de capacité, etc.

Combien de bits faut-il pour coder un ensemble minimal de conditions (ne pas oublier le "sans condition") ?

Définir précisément le champ condition, et les mnémoniques correspondants.

1.5 Récapitulation et bouche-trous

Récapituler le mot d'instruction jusqu'ici. Reste-t-il des bits inutilisés ? Si oui, voici des suggestions d'utilisation :

- On peut définir un bit qui dit si une instruction met à jour les drapeaux, ou pas. Ceci sera utile pour des séquences de code machine du type *si alors sinon*.
- On peut définir un bit qui dit si le résultat d'une opération est écrit dans le registre résultat, ou pas. Ceci permet de transformer toute instruction arithmétique ou logique en une instruction de test (par exemple il transforme SUB en CMP (compare)).
- On peut faire jouer son imagination.

Compléter le jeu d'instruction, vérifier qu'il n'y a pas (trop) de redondance dans les instructions, récapituler.

1.6 Test : programmation en assembleur

Pour se convaincre de la qualité de notre jeu d'instruction, écrire un programme réalisant la multiplication, un réalisant la division, puis un *pacman*. Corriger les questions précédentes en fonction des oublis.

2 Construction du processeur

Les paragraphes 2.1 à 2.3 doivent être étudiés plus ou moins en parallèle et itérés jusqu'à obtention d'un résultat satisfaisant.

2.1 Architecture générale

Placer sur une grande feuille les différents composants de notre processeur :

- les registres (adresse et données) d'interface à la mémoire externe,
- le banc de registres du processeur (qui sera une mémoire à deux ports),
- la boîte noire de l'UAL, avec ses signaux de commande,
- le registre des drapeaux, et la boîte noire du contrôle d'exécution,
- le registre qui contiendra le mot d'instruction.

Attention, ne pas oublier les drapeaux et autres bits pour la boîte noire de l'UAL. Ne pas essayer de connecter les différents blocs à ce stade.

2.2 Le cycle d'instruction

Décrire le cycle de von Neumann de ce processeur, et vérifier que tous les blocs nécessaires sont présents. Discuter, en fonction de l'occupation des différents blocs et des différents fils qui vont les relier, des questions du type : quand le PC est-il incrémenté ? Attention, les choses à faire dans un cycle diffèrent selon que l'opération est une opération arithmétique ou logique ou bien un accès mémoire.

2.3 Définition des signaux de commande

Tirer des fils entre ces différents blocs, et placer lorsque nécessaire des multiplexeurs commandés par des signaux binaires. Donner des numéros au hasard à chacun de ces signaux de commande. Donner également des numéros aux différents signaux de commande des différents

blocs (signaux *read* et *write* a destination de la mémoire externe, bits de l'UAL, etc. Si l'on en oublie on pourra toujours les rajouter par la suite).

S'extasier devant les bienfaits du principe d'orthogonalité.

Le reste de cette partie consiste en la définition de l'automate qui va activer ces différents signaux dans le bon ordre pour assurer le fonctionnement du processeur.

2.4 Synthèse de l'automate de commande

Définir très précisément un cycle d'instruction (en moins de 10 étapes, entre deux et six paraît un bon nombre). On pourra considérer que le délai de la mémoire est équivalent à celui de l'UAL, et toute autre hypothèse simplificatrice raisonnable. Attention, ce qui est fait à chaque étape dépend de l'instruction.

Construire l'automate très simple qui, à partir d'une horloge unique, produit n signaux binaires ϕ_i , chacun actif au cours d'une étape.

Écrire un tableau donnant, en fonction de l'instruction, ceux des signaux de commande numérotés qui sont actifs à chaque étape.

Implémenter cette table comme un bloc combinatoire prenant en entrée les ϕ_i et les autres signaux utiles, et produisant les signaux de commande numérotés.

Placer ce bloc sur le schéma (il n'est pas utile de *dessiner* tous les fils des signaux de commande).

2.5 Remplissage de boîtes noires

Construire l'intérieur de la boîte noire de l'UAL.

Construire l'intérieur de la boîte noire de contrôle d'exécution.

Construire l'intérieur des autres boîtes noires.

3 C'est fini

Estimer très grossièrement le nombre de portes.