Architecture et Système

Stefan Schwoon

Cours L3, 2020/21, ENS Cachan

Architecture: remarques

Notre petit ordinateur (voir documentation) comporte quelques simplifications et faiblesses importantes qui empêcheraient sa mise en pratique dans un ordinateur réel.

Notamment, la profondeur est trop importante :

obtenir les signaux de contrôle (17)

obtenir la valeur sur le bus (+42), dont +33 pour la RAM

calcul de l'ALU (+29)

$\rightarrow$ circuit sous-utilisé et inefficace

Cycles

Solution : sous-diviser le calcul en plusieurs cycles tel que chaque cycle achève une tâche moins complexe, p.ex. :

obtenir le prochain code d'instruction

obtenir les signaux de contrôle

fournir une adresse à AR (qui devient un registre interne)

faire un calcul dans l'ALU

$\rightarrow$ plusieurs cycles, mais d'une profondeur réduite

$\rightarrow$ permet d'adapter le temps d'exécution à la complexité de l'instruction

D’autres faiblesses

Dans notre architecture primitive, tout transfert de données passe par le CPU.

CPU surchargé de travail

la vitesse de l’ordinateur dépend du composant le plus lent

$\rightarrow$ diviser le travail, introduire des sous-composants chargés d’organiser le transfert de données à une vitesse propre

Notre architecture traite registres et mémoire avec la même vitesse.

pas réaliste, un accès mémoire peut être beaucoup plus lent qu’un accès registre

$\rightarrow$ introduire cycles d’attente

$\rightarrow$ gestion mémoire (plus tard)

Développement historique

Les premiers ordinateurs (années 50) :

peu d'instructions et signaux $\rightarrow$ architecture câblée

Les années 60-80 : âge de la microprogramation

jeux d'instructions et signaux toujours plus complexe :
microprogrammation plus facile à construire et gerer

une même architecture peut être adapté aux besoins différents

microprogrammation utilisateur sur certaines machines

Exemple: Jeu d'instructions Intel x86

Jeu d'instructions complexe, avec quelques instructions assez puissants (boucles pour traiter des blocs de mémoire)

Opérandes de 8/16/32 bits (pour compatibilité en arrière)

Instructions peuvent utiliser un registre partiel ou complet (AL/AH, AX, EAX)

Longueur d'instructions variable (1 à 7 octets), décodage compliqué

L'architecture RISC

A partir des annes 80 : retour au mode câblé

Facteurs technologiques :

miniaturisation rend possible des circuits plus complexes

outils pour automatiquement concevoir et arranger des circuits (CAD)
$\rightarrow$ construction des circuits complexes devient plus facile

apparition de l'architecture RISC qui permet des optimisations

RISC = reduced instruction set computing

Architecture RISC

Idée en général : uniformiser les instructions afin de les exécuter plus efficacement.

Caracteristiques typiques :

éliminer des opérations complexes, juste load/store/op.arithmétiques

mémoire rapide integrée dans le processeur (ou proche)

éliminer la phase *décodage* : codes opération toujours d'une même longeur, opérands toujours dans la même place de l'opcode.

exécution parallèle : exécuter plusieurs instructions à la fois : une instruction en phase "instruction fetch", une autre en "exécution"

plus de registres pour minimiser les transferts vers la mémoire

Inconveniences de RISC

Incompatible avec des architectures existantes (notamment x86)

Plus de travail pour les compilateurs

Mots d'instructions très grands, code machine peu compact

$\rightarrow$ combinaison des deux techniques:

(pré-)processeur traduit les opérations complexes vers (plusieurs) instructions RISC

une autre couche opère sur ces instructions RISC

Parallelisme dans les architectures modernes

Le processeur exécute plusieurs instructions en parallèle, en profitant des différentes phases d'exécution (pipelining).

Plusieurs unités d'exécution travaillant en parallèle (superscalaire).

En principe, cela permet d'exécuter plus d'instructions dans une même temps. Mais il y a des problèmes :

- dépendances : le résultat d'une instruction est nécessaire pour le prochain
- branchements : quelle instruction sera la prochaine ?

Solutions :

- analyse des dépendences, exécution "out of order"
- exécution spéculative (sur un autre jeu de registres), prédiction des branchements