

Architecture et Système

Stefan Schwoon

Cours L3, 2026/2027, ENS Paris-Saclay

Unité de contrôle (CU)

La CU travaille dans des phases indiquées par l'horloge :

Chaque phase réalise une opération nécessaire pour exécuter une instruction:

- obtenir et décoder l'instruction

- exécuter une opération, peut-être en plusieurs étapes

Dans chaque phase, le CU doit organiser les transferts de données:

- donner les bons bits de sélection au bus

- transférer les données entre ALU et registres

- réagir aux périphériques

Pour chaque phase d'une instruction, les signaux de contrôle sont stockés dans une ROM qu'on appelle **unité de microprogrammation**.

Phases

Phase 1 (Instruction fetch):

Laisser le bus utiliser la valeur d'IP comme sélecteur d'adresse, transférer au CPU

Phase 2 (Decode):

Consulter unité de microprogrammation pour organiser les transferts nécessaires pour l'opération

Phases 3, 4, ... (Execute):

En utilisant le MU, organiser les bons transferts et manipulations dans chaque phase.

En fonction de leur complexité, les différentes instructions nécessitent un nombre différent de phases.

Interruptions

Des périphériques (clavier, réseau) n'ont pas toujours besoin de communiquer avec le processeur, mais si besoin, il faut réagir vite.

Quand un périphérique souhaite échanger des données avec le processeur, il met son **signal d'interruption** à 1.

Quand le processeur est prêt pour la prochaine instruction, il vérifie s'il y a reçu un tel signal d'interruption et choisit celui avec la priorité la plus élevée.

Dans ce cas le processeur continue son travail à une certaine adresse en fonction du signal choisi. Cette adresse devrait contenir du code pour gérer la communication. Ensuite on continue l'exécution normale.

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)

→ circuit sous-utilisé et inefficace

Cycles

Dans un premier temps, on sous-divise le circuit en plusieurs parties indépendants qui achève des tâches bien précises et moins complexes :

- obtenir le prochain code d'instruction (IF = instruction fetch)

- obtenir les signaux de contrôle (ID = instruction decode)

- exécuter, p.ex. faire un calcul dans l'ALU (EX)

- lire/écrire dans la mémoire (MEM)

- exécution en plusieurs cycles, mais d'une profondeur réduite

- 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

→ 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

→ introduire cycles d'attente

→ gestion mémoire (plus tard)

Interruptions

Des périphériques (clavier, réseau, sous-unités pour transfert données) ont besoin de communiquer avec le processeur.

Quand un périphérique souhaite échanger des données avec le processeur, il met son **signal d'interruption** à 1.

Quand le processeur est prêt pour la prochaine instruction, il vérifie s'il y a reçu un tel signal d'interruption et choisit celui avec la priorité la plus élevée.

Dans ce cas le processeur continue son travail à une certaine adresse en fonction du signal choisi. Cette adresse devrait contenir du code pour gérer la communication. Ensuite on continue l'exécution normale.

Développement historique

Les premiers ordinateurs (années 50) :

peu d'instructions et signaux → architecture câblée

Les années 60-80 : âge de la microprogrammation

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

une même architecture peut être adapté aux besoins différents,
voir pour émuler d'autres machines

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

Pipeline / Chaîne de traitement

Si on sous-divise le calcul en plusieurs cycles, chaque instruction prend un nombre variable de cycles (IF, ID, EX1, EX2, ...)

On sous-divise alors le circuit du processeur en plusieurs unités indépendantes qui s'occupent d'une tâche précise. Ainsi, l'exécution des instructions peut se chevaucher :

Au premier cycle, on exécute la phase IF de la première instruction.

Au second cycle, la phase ID de la première et la phase ID de la seconde.

etc...

Problème: Gestion des dépendances, branchements (conditionnels), longueur d'instructions et exécutions inégales, ...

L'architecture RISC

A partir des années 80 : retour au mode câblé

Facteurs technologiques :

- miniaturisation rend possible des circuits plus complexes

- outils pour automatiquement concevoir et arranger des circuits (CAD)

 - 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 intégrée dans le processeur (ou proche)

- éliminer la phase *décodage* : codes opération toujours d'une même longueur, opérandes 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

Inconvénients de RISC

Incompatible avec des architectures existantes (notamment x86)

Plus de travail pour les compilateurs

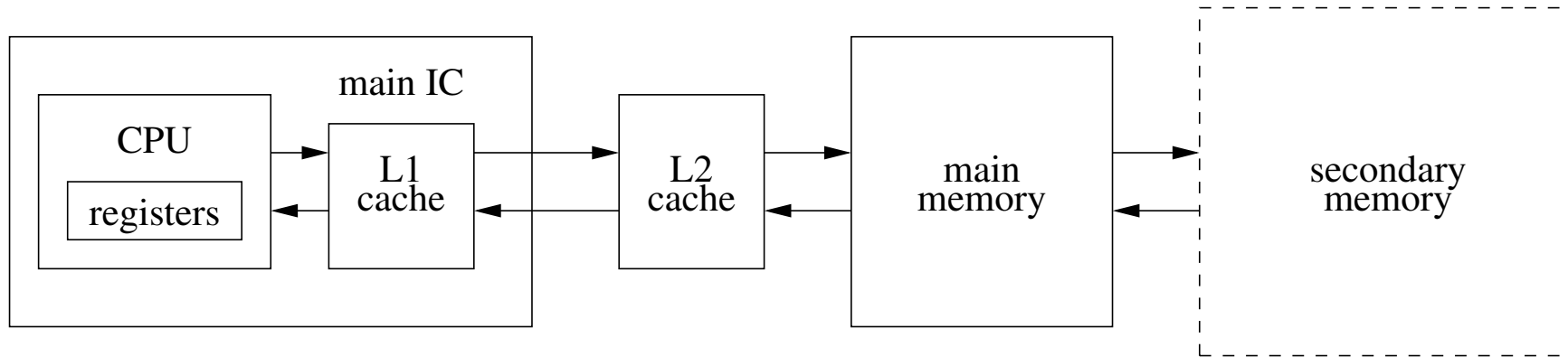
Mots d'instructions très grands, code machine peu compact

→ combinaison des deux techniques:

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

une autre couche opère sur ces instructions RISC

Les caches



Idée: garder les données utilisées souvent dans une mémoire spéciale rapide, mais de taille limitée

Facteurs limitants : coût, espace physique limité

Cache à plusieurs niveaux, les cache distants sont plus lents mais aussi plus grands

Transparent : Le programmeur accède à une adresse virtuelle, le contenu se trouve dans la mémoire principale ou dans l'un des caches.

DRAM vs SRAM

Rappel: DRAM plus compact (un seul transistor + condensateur)

lecture destructive, fuite dans les condensateurs → recharge périodique

compact mais lent, typiquement utilisé pour mémoire principale

SRAM (*static random-access memory*)

réalisation par une bascule D

moins compact mais plus rapide, utilisé pour les caches

L1 = intégré dans le CPU, L2 = dans un autre chip

Une adresse dans le cache L1 est fournie très rapidement (~ 4 cycles).

Pour récupérer une adresse dans la mémoire principale, il faut une centaine de cycles !

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 ?

Ces problèmes s'aggravent grâce aux différentes vitesses de mémoire.

Solutions :

- analyse des dépendances, exécution "out of order"

- exécution spéculative (sur un autre jeu de registres), prédiction des branchements