Architecture des Ordinateurs

Instructions et Adressage

Florent Hivert

Instructions - Généralités

* Instructions codées sous forme de 0 et de 1.

* Jeu d'instructions : ensemble des possibilités offertes par l'ordinateur.

  $\Rightarrow$ **ce qui n'existe pas ne peut être réalisé par le matériel. Pour le réaliser, il faut faire appel au logiciel, si c'est réalisable. (Calcul flottant sans instructions flottantes, par exemple)**

* Les instructions sont principalement produites par les compilateurs

* **registre :** emplacement mémoire privilégié,

  de longueur fixe
  ayant une appellation contrôlée
  accessible très rapidement par l'unité centrale

* Le jeu d'instructions considéré est celui d'une machine actuelle généraliste.

Contenu de l'instruction

* Type de l'opération (« operation code », « opcode »)
* Type des opérandes : qui sont-ils, source et cible
* Localisation des opérandes, type d'accès = adressage
  - **valeur explicite**
  - **emplacement mémoire**
  - **numéro d'un registre**
* L'instruction est découpée en champs ayant une signification distincte :

code opération	**type opérande**	**Opérande 1**	...

  Le découpage de l'instruction est souvent le même, quelque soit le type de l'opération (Pas obligatoire (Intel86) Seul le code opération permet d'interprêter les champs suivants. Tout en dépend.
* La longueur du code opération détermine le nombre maximum d'instructions :

$$2^n \text{ pour } n \text{ bits}$$

L'Assembleur

* Instruction $\leftrightarrow$ sigle mnémonique.

* Assembleur : traducteur (sigle $\rightarrow$ chaîne de bits)

* Spécifique à l'ordinateur, pas de langage assembleur normalisé.

* Généralement sont mentionnés : le type de l'opération, les opérandes.
  - `LOAD Reg1, adresse-mémoire`
  - `Reg1 adresse-mémoire`
  - `adresse-mémoire` $\rightarrow$ `Reg1`

* De plus, indications (instructions fictives) ne générant pas d'instructions en langage machine (chaîne de bits).

  Ne pas confondre avec les instructions :
  – **réservation de zones mémoire**
  – **initialisation de zones mémoire**

* Peu d'utilisateurs programment en assembleur.

Instructions : Longueurs

* La longueur des instructions varie selon les ordinateurs.
  Longueur variable indiquée implicitement.

* Longueurs différentes de celle des données traitées.

* Plus compactes possible, afin de ne pas prendre trop de place en mémoire.
  Plus un programme est dense, plus l'accès en est rapide.

* Longueurs des opérandes :
  Un même ordinateur peut travailler sur des opérandes de longueur différente.
  Le code de l'opération précise la longueur opérandes.
  - caractères sont toujours sur 8 bits (ASCII, EBCDIC) ;
  - entiers sur 10,32 voire 64 et 128 bits ;
  - flottants sur 32 ou 64 bits.

Contenu implicite des instructions

* Longueur de l'instruction, en cas de taille variable
* Longueur des opérandes cible et source
* Nombre des opérandes :

  3 : 2 sources et 1 cible

  2 : 1 opérande est source et cible à la fois
* Types des opérandes : entier, flottant

Instructions de transfert entre deux emplacements

* Duplication d'une valeur et de son rangement à un autre emplacement (a=b en langage C).
* Toutes les combinaisons possibles entre registres et cases mémoire.
* Seul point délicat, si deux emplacements n'ont pas la même taille/longueur, il faut préciser comment se déroule :
  - la troncation (de 32 à 16 bits par exemple)
  - le remplissage

Exemple :

```
move  Reg1 -> case mémoire ; stockage d'un registre en mémoire
move   3   -> Reg1         ; chargement d'un registre
```

Instructions de calcul arithmétique

* Opérations les plus fréquentes : addition, multiplication, pouvant porter sur des longueurs différentes de données 8, 16, 32, 64 bits
* Le calcul sur des entiers ou des flottants : d'instructions différentes, dans la plupart des architectures. Les registres ou emplacements mémoire utilisés peuvent être les mêmes. L'interprétation de leur contenu change.
* Cas particuliers : addition / soustraction de 1, Utilisé pour les compteurs de boucles.

Instructions d'opérations logiques

* Opérations travaillant sur des opérandes considérées comme des chaînes de bits, qui ne font aucun cas du type d'opérande, réel, caractères ...

* Les plus courantes :
  - complément, $0 \rightarrow 1, \quad 1 \rightarrow 0$
  - ou inclusif
  - ou exclusif
  - et

* Utilisation courante du ET : masque pour extraction d'information.

Exemple : **4 bits du milieu sur 8 bits**
masque 00111100

Instructions de décalage

* Opèrent sur une chaîne de bits contenue dans un registre

valeur originelle	`. . . .          xxx`
décalage à droite	$\rightarrow$`. . . .          x`
décalage à gauche	`. .          xxx`$\leftarrow$

* Il y a trois types de décalage :

logique	**ce qui sort du registre est perdu, les emplacements vides sont remplis par des bits à zéro**
circulaire	**ce qui sort à un bout rentre à l'autre bout**
arithmétique	**le bit du poids fort est considéré comme bit de signe ; le décalage n'en tient pas compte et le laisse inchangé**

* Instructions très utilisées pour les multiplications/divisions par puissances de 2.

Instructions de comparaison et tests

* Comparer deux opérandes entre eux, ou bien un opérande et une valeur fixe (zéro), et positionner un bit selon le résultat de la comparaison.
* Les relations de comparaison peuvent être : <, <=, =, >=, >
* Le résultat est un drapeau (« flag »), souvent rangé dans un registre particulier, dit registre drapeau.
* Les instructions de test portent sur la valeur des drapeaux, ou de bits particuliers :
  - retenue lors d'opération arithmétique
  - bit de signe

Instructions de branchement

* **Principe : les instructions sont exécutées en séquence, un branchement conduit à exécuter une instruction autre que la suivante en mémoire.**

* L'adresse de la prochaine instruction à exécuter peut être :
  - l'adresse par rapport au début du programme (« absolue ») ;
  - l'adresse par rapport à l'instruction de branchement (« relative »)
    Dans ce cas, cette adresse est souvent signée.

* L'adresse de la prochaine instruction à exécuter se trouve généralement dans un registre, compteur ordinal (IP = Instruction Pointeur). Un branchement revient à une modification du contenu de ce registre.

Instructions de branchement (suite)

Types de branchement :

- ∗ Inconditionnel : le branchement a lieu dans tous les cas ;

- ∗ Conditionnel : selon le résultat d'un test ;
  - comparaison entre deux nombres ;
  - comparaison d'un nombre à zéro.
  - selon l'état d'un drapeau (équivalent au résultat d'un test) ;
  - selon l'état d'autres valeurs : bit de retenue, bit de signe.

- ∗ Retour : après un premier branchement, retour à l'instruction suivant l'appel
  - **conservation de l'adresse de retour** ; selon les cas en mémoire (prise en compte logicielle) dans une pile dans un registre particulier ;
  - utilisé fréquemment dans les langages évolués :
    appel de sous-programme/procédure/routine
  - récursivité , « in-lining »

Instructions particulières

* Manipulation de chaînes de caractères : concaténation, etc ...
* Manipulation de pile
* Manipulation de drapeaux
* Entrées/Sorties, Traitements d'interruption
* Gestion du multitache, du multiprocesseurs : changement de contexte, communication inter-processeurs
* Il existe des instructions ayant des durées d'exécution très longues :
  - Intel i960 : reste d'une division réelle pouvant durer de 67 à 75878 périodes d'horloge
  - GE 600 : parcours d'une liste chaînée, avec test à chaque structure rencontrée
  - Texas TI TMS34010/20 : traitement de pixels depuis un tableau source vers un tableau destination ; « rasterisation », balayage d'écran afin de récupérer les valeurs des pixels sous forme numérique

Adressage : généralités

* Adressage = types d'accès aux opérandes ;
* A quoi accède-t-on ?

octets	8 bits seul terme identique pour toutes les architectures
1/2 mot	16 bits
mot	32 bits $\rightarrow$ 1/2 mot
double mot	64 bits $\rightarrow$ mot

  terminologie : mieux vaut parler de mots de xx bits

* Rappel : les octets ne sont pas rangés en mémoire dans le même ordre, selon les architectures
  - Big Endian (octets de poids forts d'abord)
  - Little Endian (octets de poids faible d'abord)
* Le mode d'adressage se trouve indiqué au sein de l'instruction dans le code opération ou dans un champ séparé

Adressage : généralités (suite)

* Instructions peuvent comprendre 0, 1, 2 ou 3 opérandes. Exemple :

  0 `exit`

  1 `incrément Reg`

  2 `move expression` $\longrightarrow$ `Reg`

  3 `addition Reg 1, Reg 2` $\longrightarrow$ `Reg 3`

* Les opérandes peuvent être :
  - des valeurs explicites ;
  - des registres, un accumulateur, des élements de pile ;
  - des emplacements (ou cases) mémoire ;

Adressage immédiat

* Il ne s'agit pas d'un adressage à proprement parler, la valeur de l'opérande étant contenue dans un champ de l'instruction réservé à l'opérande.
* Au lieu de désigner un emplacement (registre, mémoire) où se trouve une valeur, on désigne la valeur elle-même.

  Exemple : `addition 2, Reg`
  - Limitation : la valeur ne doit pas excéder la taille du champ opérande
  - Avantage : plus rapide que l'accès à la mémoire ou à un registre

Adressage direct

∗ L'adresse de l'opérande est donnée dans l'instruction, sous forme de :
- adresse mémoire
- identifiant / numéro de registre

Exemple : `move case-mémoire` $\rightarrow$ `Reg`

∗ Limitation : la valeur de l'adresse mémoire est limitée par la taille du champ de l'instruction.

Adressage indirect

* L'emplacement où se trouve l'adresse est désigné, et non plus l'adresse elle-même.
* Principe identique pour les registres et les cases mémoire. Un registre peut contenir une adresse mémoire, ou bien le numéro d'un autre registre.
  Exemple `move [case 1]` $\rightarrow$ `case 2`
* Nécessite au moins deux accès (à la mémoire, ou aux registres).
* En langage de haut niveau, cela s'appelle des pointeurs.

```
int case2;
int *case1;
case2 = *case1;
```

Adressage Indexe

* Adresse recherchée est la somme d'une adresse et d'un index.
  Exemple : `move case1[Reg1]` $\rightarrow$ `Reg2`

* Utile pour la manipulation de tableaux

Adressage relatif ou Basé

* L'adresse réelle est l'adresse indiquée dans l'instruction, ajoutée à une adresse de référence, contenue dans un registre
  – le plus souvent
  – dans une case mémoire particulière.

* Exemple : adressage basé par rapport à un registre contenant l'adresse du début du programme / de la page.

* Avantage : déplacement du programme sans changement des adresses

Adressage avec auto-incrément

* Lorsque un registre – une case mémoire – est utilisé, il est automatiquement incrémenté ou décrémenté de 1, afin de passer à l'adresse séquentielle suivante.
* Très utile pour parcourir un tableau de données au sein d'une boucle.
* Ce type d'adressage existe dans certains langages de haut niveau, C par exemple.

Combinaisons d'adressage

* Indirect indexé (parcours d'un tableau de valeurs)

  1 indirection
  2 ajout de l'index à l'adresse obtenue

* Indexé indirect (tableau de pointeurs)

  1 adresse dans l'instruction + index
  2 indirection à partir du résultat obtenu

* Indexé avec auto-incrémentation

  Une seule instruction suffit pour la manipulation d'un tableau, dans une boucle.

* Tout peut être simulé

  Rappel : on peut presque toujours faire par logiciel ce que le matériel ne fait pas.

Adressage par pile

* Instruction : `Push` changement d'une valeur dans la pile
  `Pop` extraction d'une valeur depuis la pile
* Utilisation dans les calculatrices
* Notation polonaise inverse pratique avec les piles ; inverse de notation algébrique, ou infixée

Evolution des jeux d'instructions : CISC → RISC : Les années 60

* Buts : économiser la mémoire (coût) ⇒ réduire la taille des programmes exécutables, simplifier le travail des compilateurs en mettant à leur disposition des instructions évolués.
* Caractéristiques : jeu d'instructions complexes, grand nombre d'adressages
* **CISC : Complex Instruction Set Computer**

Evolution des jeux d’instructions : CISC $\rightarrow$ RISC : les années 80

* Progrès dans la densité des circuits intégrés pour la mémoire et pour les processeurs $\Rightarrow$ réduction des coûts de l’espace mémoire.
* But : performance $\Leftrightarrow$ efficacité, simplicité
* **RISC : Reduced Instruction Set Computer** - vise l’exécution d’une instruction par cycle d’horloge.
* Caractéristiques : "petit" nombre d’instructions
* peu d’adressages possibles
* instructions de longueur fixe

L’evolution technologique guide les choix des jeux d’instructions.

Compilateurs de langages évolués

* Buts d'un compilateur :
  - générer du code instruction faisant ce qui a été demandé par le programmeur
  - générer du code s'exécutant le plus rapidement possible :

  $\Rightarrow$ **utilisation des composants de stockage accessibles le plus rapidement** :
    - un accumulateur ($\Leftrightarrow$ un registre particulier) assez ancien, micro-processeurs spécialisés pour le contrôle et les Entrées/Sorties
    - une pile, calculatrices de poche par exemple
    - des registres - mode le plus courant actuellement

  le compilateur tient compte du nombre de registres, et de leur spécificité éventuelle.

* Les types de données des langages évolués (entier, flottant, caractère) correspondent directement aux opérandes des instructions, et même à l'adressage (octet pour traitement des caractères).

Comparaison de 2 processeurs : CISC et RISC

	CISC : DEC VAX	RISC : SUN SPARC
Différentes tailles d'instructions	54	1
Intervalle de taille des instructions	54 de 16 à 456 bits	32 bits
Nombre d'instructions	244	72
Nombre de modes d'adressage	22	2
Présence d'opération combinée : Chargement/stockage mémoire et calcul arithmétique	oui	non

De l'influence des compilateurs sur le jeu d'instructions

* Années 60 / 70 :
  - Les compilateurs utilisent peu les riches potentialités des jeux d'instructions
* Années 80 :
  - L'optimisation prend une part plus importante dans l'écriture d'un compilateur (calcul scientifique)
  - Plus simple et plus efficace, s'il y a moins d'instructions
  - L'optimisation logicielle + jeu d'instructions réduit donnent d'aussi bons résultats en performance qu'un jeu d'instructions riche

    ⇒ prédominance de processeurs RISC et de compilateurs sophistiqués

Tendance actuelle : Very Long Instruction Word (VLIW) où le compilateur prend une plus grande part dans l'ordonnancement d'instructions pris en charge par le matériel dans les machines RISC.

Comparaison de compilateurs sur processeurs CISC et RISC

Nombre d'instructions utilisées par compilateur C sur machine CISC : 30%

Modes d'adressage utilisés par compilateur C sur machine CISC : 66%

Tailles relatives des codes exécutables : 1,33

Rappel : les instructions générées moins nombreuses sur un processeur CISC, mais plus longues.

Conclusion : les fonctionnalités complexes du CISC sont malheureusement rarement utilisées par les compilateurs.

Il y a deux raisons pour lesquelles des machines RISC peuvent fournir des avantages de performance sur les machines CISC :
- une technologie compilateur avancée
- des caches d'instruction pour réduire le déficit d'accès du à la taille