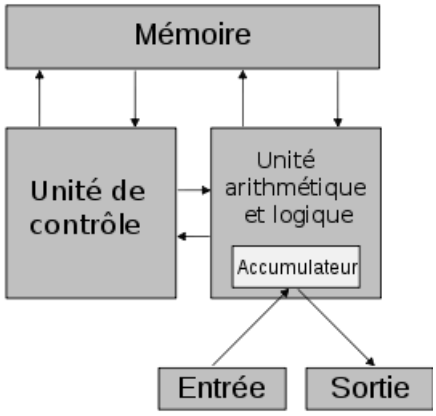
Notions de langage machine

18 décembre 2009

Rappels et introduction



Architecture de Van Neumann :
- Processeur = UC + UAL
- Mémoire interne au processeur = les registres (soit seulement l'accumulateur, soit des registres généraux, ...)

Programme
= suite d'instructions exécutées par le processeur
- Instruction = action très simple
- Ensemble des instructions possibles = langage machine associé à un processeur
- Chaque instruction est codée par un mot binaire (suite de 0 et de 1) stocké en mémoire

Mémoire et registres

Mémoire (aussi appelée mémoire centrale, mémoire vive, RAM) :
- Extérieure au processeur
- Composée d'une suite d'octets, parfois regroupés par 2 (16 bits), 4 (32 bits) ou 8 (64 bits)
- Accès assez lent

Registres :
- Internes au processeur
- Chaque registre a une capacité mémoire fixée (souvent 32 bits ou 64 bits)
- Accès rapide

Différents types de registres :
- Compteur Ordinal (CO) : contient l'adresse de la prochaine instruction à exécuter

- D'autres registres spécifiques
- Des registres généraux (une dizaine à plusieurs centaines) : pour le stockage temporaire des données nécessaires à l'exécution des instructions
- Certains des registres sont invisibles aux utilisateurs (ex : registre d'instruction, qui contient l'instruction en cours d'exécution)

Langage assembleur

- Le langage machine n'est pas compréhensible par un être humain "normal".
- Langage proche du langage machine mais compréhensible = Assembleur.
  C'est aussi le premier langage utilisé.
- Une instruction en assembleur pour chaque instruction du langage machine.
- Donc un langage assembleur pour chaque type de processeur.
- Pour rendre les instructions lisibles, on les représente en assembleur par des mnémoniques (ou mots clés) plutôt que par les codes binaires du langage machine.

Opération d'assemblage

- Traduction d'un programme en langage assembleur à une séquence binaire exécutable par le processeur.
- Cette opération de traduction est s'appelle l'assemblage.
- Elle est réalisée par un programme appelé (aussi) assembleur.

Généralités sur les instructions en langage machine

Structure des instructions

- Jeu d'instructions = Ensemble des instructions qui sont disponibles dans le processeur.
- Exemple : peu d'instructions dans les processeurs RISC, beaucoup plus dans les processeurs CISC.
- Programme = séquence d'instructions exécutées séquentiellement (a priori – voir le cas des sauts)

Une instruction contient les informations suivantes :

- le code opération (ou op code) qui représente la nature de l'opération,
- la désignation des opérandes (sources et résultat) dans les registres, en mémoire, ... (voir plus loin les modes d'adressage).
- Le plus souvent, la séquence de bits de l'instruction est divisée en champs qui correspondent à ces différentes parties

Le nombre de bits pour coder une instruction peut être identique pour toutes les instructions (ex : ARM, MIPS, PowerPC) ou différent selon la nature de l'opération, c'est-à-dire le op code (ex : x86)

Placement des opérandes

Plusieurs choix possibles, qui correspondent à différents types d'architectures (de la plus ancienne à la plus récente) :

Architectures à pile : Les opérandes sont placées au sommet de la pile : même principe que dans la forme polonaise suffixée. On accède aux opérandes par les opérations PUSH (empiler) et POP dépiler.

Exemple : Z = X + Y correspond à `PUSH X ; PUSH Y ; ADD ; POP Z`

Architectures à accumulateur : Toutes les valeurs chargées depuis la mémoire (instruction LOAD) ou copiées en mémoire (instruction STORE) le sont en passant par un registre spécifique, appelé accumulateur

Exemple : Z = X + Y correspond à `LOAD X ; ADD Y ; STORE Z`

Architectures à registres généraux :

- Plusieurs registres, qui peuvent contenir des données
- Chaque instruction fait explicitement appel à ses opérandes, qui peuvent être des registres, des emplacements mémoire, des valeurs constantes, ...
- Instructions LOAD et STORE pour le transfert de données de la mémoire vers les registres et vice versa.

Exemple : Z = X + Y peut se traduire en `LOAD R1 X ; ADD R1 Y ; STORE Z R1`, si on choisit d'utiliser le registre R1.

Modes d'adressage

Dans les architectures à registres généraux, les arguments dans une instructions du langage machine peuvent être interprétés de différentes manières (registres, emplacements mémoire, constantes, ...) : c'est cette interprétation qui est définie par le mode d'adressage.

Les principaux modes d'adressage existant sont décrit ci-dessous. Un processeur peut disposer seulement d'une partie de ces modes d'adressage, ou avoir en plus des modes d'adressage moins standard. Les notations adoptées dans la suite ne correspondent pas à un assembleur en particulier. On travaille avec un assembleur "fictif", pour présenter les concepts sans s'attacher à une architecture spécifique.

Notations :

- le symbole $\leftarrow$ représente l'affectation (chargement de la valeur à droite de $\leftarrow$ dans l'emplacement à gauche de $\leftarrow$)
- M est un tableau représentant la mémoire
- M[n] représente la case mémoire d'adresse n, pour tout entier $n \geq 0$
- M[n] représente la case mémoire dont l'adresse est la valeur contenue dans R, pour tout registre R

Mode d'adressage implicite : il n'y a pas d'argument explicite (il est implicite, et donc toujours le même pour une opération). C'est ce qui se passe avec l'accumulateur par exemple.

Mode d'adressage immédiat : l'argument est interprété comme une constante numérique.
Exemple : L'instruction `ADD R3, #6` réalise R3 ← R3+6.

Mode d'adressage registre : l'argument est désigné par un numéro de registre, et correspond à la valeur contenue dans ce registre.
Exemple : L'instruction `ADD R3, R5` ajoute (dans R3) la valeur contenue dans R3 à la valeur contenue dans R5, c'est-à-dire réalise R3 ← R3+R5.

Mode d'adressage direct : l'argument est désigné par une adresse mémoire, et correspond à la valeur contenue dans la case mémoire à cette adresse.
Exemple : L'instruction `ADD R2, 1000` réalise R2 ← R2+M[1000].

Mode d'adressage basé (ou indexé) : il combine l'adressage par registre et l'adressage immédiat. L'argument correspond au contenu d'une case mémoire dont l'adresse est donnée par une case mémoire de départ et une incrémentation de cette adresse de la valeur contenue dans un registre.
Exemple : L'instruction `ADD R2, 1200(R1)` réalise R2 ← R2+M[1200+R2].
Utilisations : parcours d'un tableau, accès à l'ensemble de la mémoire (c'est-à-dire même si le nombre de bits réservés à la désignation de la case mémoire dans l'instruction est insuffisant)

Mode d'adressage indirect par registre : l'argument est désigné par un registre (qui contient une adresse n en mémoire), et correspond à la valeur contenue dans la case mémoire n.
Exemple : L'instruction `ADD R3, (R1)` réalise R3 ← R3+M[R1].

Mode d'adressage indirect mémoire : de même qu'au-dessus, mais l'adresse n est stockée dans une case mémoire (d'adresse m)...
Exemple : L'instruction `ADD R3, @1000` réalise R3 ← R3+M[M[1000]].

Spécificités des sauts Les instructions d'un programme sont a priori exécutées de manière séquentielle. Pour rompre cet enchaînement séquentiel, il y a des modes d'adressage qui modifient la valeur du compteur ordinal pour permettre d'effectuer des sauts.
Mode d'adressage absolu : spécifie dans une instruction de saut l'adresse absolue en mémoire de la case qui contient la prochaine instruction à exécuter.
Mode d'adressage relatif : ajoute ou retranche une certaine valeur à la valeur du compteur ordinal. Le saut est donc relatif à l'instruction en cours d'exécution. Ajouter

correspond à sauter un certain nombre d'instructions, et retrancher correspond à revenir un certain nombre d'instruction en arrière.

Types d'instruction machine

Les instructions machines sont regroupées en plusieurs familles, dont les principales sont :

- les opérations arithmétiques
- les opérations logiques
- les opérations de lecture et d'écriture en mémoire
- les opérations de contrôle (branchements inconditionnels ou conditionnels)

On donne des exemples dans la suite.

Exemple de langage machine

On considère le langage machine simplifié décrit ci-dessous.

Chaque instruction est codée sur deux octets (processeur 16 bits). Le processeur dispose de huit registres. Chaque registre entre R0 et R7 est codé par les trois bits représentant son numéro.

Tableau des abréviations utilisées :

Rd ou Rdest	Registre destination (qui reçoit le résultat de l'opération)
RG	Registre qui contient l'opérande gauche
RD	Registre qui contient l'opérande droit
Rs ou Rsource	Registre qui contient l'unique opérande
M[.]	Représentation de la mémoire
adrAbs	Adresse absolue en mémoire
adrRel	Adresse en mémoire relative à la valeur du compteur ordinal
Rb ou Rbase	Registre qui contient l'adresse en mémoire par rapport à laquelle sont calculées les adresses relatives
Rtest	Registre qui contient la valeur qui est comparée à zéro dans le test

Les instructions sont données dans le tableau de la page suivante.

Remarque : Les entiers sur 5 bits permettent de couvrir l'intervalle [-16,15].

Instruction	Langage machine		Code binaire	
Arithmétique et logique				
Rd ← RG + RD	ADD	Rdest RG RD	0001	Rd RG 000 RD
Rd ← RG + n	ADD	Rdest RG #n	0001	Rd RG 1[n/5 bits]
Rd ← RG & RD	AND	Rdest RG RD	0010	Rd RG 000 RD
Rd ← RG & n	AND	Rdest RG #n	0010	Rd RG 1[n/5 bits]
Rd ← ¬ Rsource	NOT	Rdest Rsource	0011	Rd Rs 111 111
Transfert Processeur/ Mémoire				
Rd ← M[adrAbs]	LD	Rdest adrAbs	0100	Rd [adr Abs/9 bits]
R ← M[Rb+adrRel]	LOAD	Rdest Rbase adrRel	0101	Rd Rb [adrRel/6 bits]
Rs → M[adrAbs]	ST	Rsource adrAbs	0110	Rd [adrAbs/9 bits]
Rs → M[Rb+adrRel]	STORE	Rsource Rbase adrRel	0111	Rd Rb [adr rel/6 bits]
Contrôle				
Saut M[adrAbs]	JMP	adrAbs	1000	000 [adrAbs du saut]
Saut adrRel	JMPR	adrRel	1001	000 000 [adrRel du saut par rapport au compteur ordinal/6 bits]
Si test Saut adrRel sinon continue	BR	[z/n/p] Rtest adrRel/CO	1010	[znp] Rtest [adrRel de la marque par rapport au compteur ordinal/6 bits]
end	TRAP	message	1111	0000 1000 0000

Remarque : On aurait pu ajouter des opérations d'ajout en adressage direct ($01b_3b_2b_1b_0$ pour les 6 derniers bits, les 4 derniers codant une adresse mémoire), ou en adressage indirect par registre ($001b_2b_1b_0$ pour les 6 derniers bits, les 3 derniers codant le numéro d'un registre).

Les langages assembleurs contiennent aussi des directives. Dans notre langage exemple, il y a seulement deux directives `.DEBUT` et `.END` qui marquent le début et la fin d'un programme.

Exemple de programme

En code assembleur

```
        .DEBUT  x8000           // adresse de début du programme
        AND     R0, R0, #0      // initialisation de i à 0
        ADD     R2, R0, #-12
        ADD     R2, R2, #-13
        ADD     R2, R2, R2
        ADD     R2, R2, R2      // R2 <- -100
boucle  LOAD    R1, R0, #30     // R1 <- MEM[30+i]
        ADD     R1, R1, #5      // R1 <- R1 + 5
        STORE   R1, R0, #30     // MEM[30+i] <- R1
        ADD     R0, R0, #1      // i <- i + 1
```

```
        ADD     R3, R0, R2
        BRn     R3 boucle       // si i < 100 alors reprendre l'exécution à
                                //  boucle (reculer de 5 instructions)
        .END                    // arrêter
```

En langage machine

```
adresse d'octet      contenu des seize bits (2 octets mémoire)
x8000                0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
x8002                0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0
x8004                0 0 0 1 0 1 0 0 1 0 1 1 0 0 1 1
x8006                0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0
x8008                0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0
x800A                0 1 0 1 0 0 1 0 0 0 0 1 1 1 1 0
x800C                0 0 0 1 0 0 1 0 0 1 1 0 0 1 0 1
x800E                0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0
x8010                0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
x8012                0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0
x8014                1 0 1 0 1 0 0 0 1 1 1 1 1 0 1 1
x8016                1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0
```

Exercices

Exercice 0
Faire tourner à la main le programme ci-dessus. Que réalise-t-il ?

Exercice 1
Coder dans le langage assembleur précédent l'instruction conditionnelle suivante :

```
if (x>21)
  x = x+3;
```

En donner une traduction en langage machine en supposant que
- les données du programme commencent en mémoire à l'adresse hexadécimale 40000, valeur supposée contenue dans le registre R7
- l'adresse de x est 44008
- les instructions commencent à l'adresse hexadécimale 1000

Exercice 2
Coder dans le langage assembleur précédent les instructions conditionnelles suivantes :

```
Instruction 1 :
  if (x==17)
    x = x+4;
  else
    x = x-3;
```

```
Instruction 2 :
  while (x<11)
    x = x+1;
```

Exercice 3
Coder dans le langage précédent, l'itération suivante qui recherche l'indice et la valeur du premier élément de la suite de Fibonacci de valeur inférieure à une valeur a donnée :

```
// la variable a est supposée initialisée
i = 0;
x = 1;
y = 1;
while (x < a){
   z = x + y;
   z = x + y;
   x = y;
   y = z;
   i = i + 1;
}
```

Exercice 4

De Java au langage machine On suppose que la variable x est mémorisée à l'adresse hexadécimale $x20$. Traduire en langage machine les programmes ci-dessous.

```
Programme 1 :
      if (x == 17){
         x = x + 4;}
      else{
         x = x - 3;}
      x=3*x
```

```
Programme 2 :
      for (i=0 ; i<12 ; i++)
         x=x*2;
```

Représentation des programmes en machine Supposons que le premier programme débute à l'adresse héxadécimale $x4000$ et que le second débute à l'adresse héxadécimale $x5000$. Donner la représentation machine des programmes exécutables décrit par les programmes java ci-dessus.