I2 - Partie 1 : Architecture matérielle Langage machine

N. Prcovic

04.91.28.89.51

nicolas.prcovic@univ-cezanne.fr

L'ordinateur en tant que machine numérique

- **Rappel** : Un μP ne connaît qu'un seul langage, appelé *langage machine*
  (qui diffère selon les types de μP)
- **Objectif** : expliquer comment un programme écrit en C peut avoir une équivalence en langage machine.
- **Moyen** : Nous définissons un μP simplifié (inspiré du MC68000 de Motorola) que nous appellerons **MP0**.
  Il saura exécuter des programmes dans un langage appelé **LM0**.

Les registres de MP0

- MP0 ne possèdent que des registres 8 bits (c'est bien trop peu pour un vrai μP).
- Il ne peut donc que
  - manipuler des octets
  - et accéder à des adresses de 0 à 255.

Les registres de MP0

- Un *compteur ordinal* **CO** [PC] : contient l'adresse de la prochaine instruction à exécuter.
- Un *registre d'état* [SR] qui contient les bits **Z** et **N**.
  - Quand Z $=$ 1, cela signifie que la dernière instruction a produit un résultat égal à zéro.
  - Quand N $=$ 1, cela signifie que la dernière instruction a produit un résultat négatif.
- Deux *registres de données* **D0** et **D1** qui stockent chacun un octet à traiter.
- Deux *registres d'adresse* **A0** et **A1** qui stockent chacun une adresse pour accéder à des octets de la RAM.
- Un *pointeur de pile* **PP** [SP] : registre d'adresse réservé pour la pile.

Formats numérique des instructions

- Chaque instruction de LM0 est codée sur 2 octets :
  - 1^{er} octet : code le type de l'opération et le type de son ou ses opérandes.
  - 2^{e} octet : peut être utilisé pour coder une valeur associée à une opérande.
- Ex : "mettre la valeur 23 dans D0"
  - 1^{er} octet : 121 ("mettre la valeur [octet suivant] dans D0")
  - 2^{e} octet : 23
- Si pas de valeur à associer à une opérande : le 2^{e} octet est 0.
- Ex : "mettre la valeur de D0 dans D1"
  - 1^{er} octet : 248 ("mettre la valeur de D0 dans D1")
  - 2^{e} octet : 0

Formats mnémonique des instructions

- Comme il n'est pas pratique de désigner une instruction du langage machine par son numéro, on lui attribuera un **nom** et une **syntaxe.**
- On écrira chaque instruction sous forme de *mnémonique* (= nom rappelant l'opération).
  - MOVE (transfert),
  - ADD (addition), SUB (soustraction),
  - JMP (saut), etc.
- On ajoutera l'opérande après la mnémonique

JMP #120

et s'il y en a deux, on les séparera avec une virgule

MOVE #100, D0

Adressage

- Les octets que manipulent les langages machines sont stockés
  - soit dans des registres de donnée ou d'adresse
  - soit dans la mémoire principale :
    - dans l'instruction (2^e octet)
    - à l'adresse donnée dans l'instruction (2^e octet)
    - à l'adresse dans un registre d'adresse
- **Remarque** : Les instructions de LM0 ne permettent de désigner qu'une **seule** opérande au maximum grâce à un numéro, l'autre opérande devant se référer à un registre.

Adressage immédiat

- Si on connait la valeur que l'on veut traiter, on peut la spécifier **explicitement.**
- La valeur désignée est stockée dans le 2^e octet de l'instruction
- **Syntaxe** : on préfixe la valeur avec le caractère '#'.
- Ex : `MOVE #3,D0` place la **valeur 3** dans le registre D0.
  Code : 1^{er} octet : `MOVE #?,D0`, 2^e octet : `3`

Adressage direct

- *Adressage direct* : accès à une valeur située à une adresse dont on spécifie le numéro.
- **Syntaxe** : on écrit simplement adresse (et rien d'autre).
- Ex : `MOVE 100, D0` place la **valeur située à l'adresse 100** dans le registre D0.
  Code : 1^{er} octet : `MOVE ?,D0`, 2^{e} octet : `100`

Adressage indirect

- *Adressage indirect* : accès à une valeur située à une adresse mémorisée dans un registre d'adresse.
- **Syntaxe** : le registre d'adresse entre parenthèses.
- Ex : `MOVE (A1),D1` place la valeur située à l'adresse contenue dans le registre d'adresse A1 dans le registre D1.
  Code : 1^{er} octet : `MOVE (A1),D1`, 2^{e} octet : `0`

Les instructions de LM0

On peut regrouper l'ensemble des instructions en 3 groupes :

- les instructions de transfert (MOVE, PSH, POP),
- les opérations arithmétiques et logiques (ADD, SUB, MUL, DIV, NOT, AND, OR, ...)
- et les instructions de saut (JMP, JEQ, JSR, ...)

Instructions de transfert

- L'instruction `MOVE` permet de transférer un octet d'un emplacement (registre ou case mémoire) à un autre.
- **Syntaxe** : `MOVE` *source, destination*.
- Ex : `MOVE #10,A1`
  `MOVE D0,123`
  `MOVE (A0),(A1).`
- De plus :
  - si la valeur transférée est nulle, le bit Z du registre d'état passe à 1.
  - si la valeur transférée est négative, le bit N du registre d'état passe à 1.

Instructions de transfert

- Les instructions `PSH` et `POP` respectivement **empile** et **dépile** un octet grâce au registre PP.
- L'instruction `PSH` décrémente PP et place la valeur de l'opérande à l'adresse pointée par PP.

`PSH` *source* $\equiv$ `MOVE` *source*`, (PP)`

puis `SUB #1, PP`

- L'instruction `POP` place la valeur pointée par PP dans l'opérande et incrémente PP.

`POP` *dest* $\equiv$ `ADD #1, PP`

puis `MOVE (PP),` *dest*

- Au démarrage de la machine, le registre PP contient l'adresse 255.

Opérations arithmétiques et logiques

- Les opérations arithmétiques et logiques
  - modifient la valeur de l'(unique) opérande
  - ou calculent une valeur à partir de deux opérandes.
- En LM0, le résultat du calcul remplace la valeur du **deuxième** opérande.
- Ex : `ADD D0, D1` fait la somme des valeurs de `D0` et `D1` et place le résultat dans `D1`.
- Le résultat d'une opération affecte les bits Z et N du registre d'état.

Opérations arithmétiques

- **L'addition** : `ADD` *source*, *destination*
- **La soustraction** : `SUB` *source*, *destination*
  (on soustrait *source* à *destination*).
- **La multiplication** : `MUL` *source*, *destination*.
- **La division (entière)** : `DIV` *source*, *destination*
  (on divise *destination* par *source*).
- Exemples :
  - `SUB #3, D0` : si D0 contenait 5, D0 passe à 2.
  - `DIV #5, D0` : si D0 contenait 18, D0 passe à 3.

Opérations logiques

- **La négation** : `NOT` *destination*
  - `NOT D0` : si D0 contenait 55, D0 passe à 200
    00110111 (55)
    ————————- NOT
    11001000 (200)
- **Le ET** : `AND` *source*, *destination*
  - `AND #179, D0` : si D0 contenait 241, D0 passe à 177.
    10110011 (179)
    11110001 (241)
    ————————- AND
    10110001 (177)
- **Le OU** : `OR` *source*, *destination*
  - `OR #179, D0` : si D0 contenait 241, D0 passe à 243.
    10110011 (179)
    11110001 (241)
    ————————- OR
    11110011 (243)

Opération de comparaison (CMP)

- Une instruction de comparaison :

  `CMP` *source*, *destination*

  fait la même chose que `SUB` mais ne modifie **pas** le deuxième opérande.
- Par contre, elle **modifie** les bits Z et N du registre d'état.
- Ex : `CMP #3, D0`
  - Z passe à 1 si D0 contient 3 (et Z passe à 1 sinon)
  - N passe à 1 si D0 contient une valeur $<$ 3 (et N passe à 0 sinon)
- Cette instruction s'utilise en préparation d'une instruction de saut conditionnel
(dont l'exécution dépend de Z et de N).

Instructions de saut

- Les instructions de saut modifient la valeur de `CO` afin d'exécuter une autre série d'instructions que celle qui suivent l'instruction courante.
- Trois types de sauts :
  - Saut inconditionnel : `JMP` *destination*
  - Sauts conditionnels : `JEQ/JNE/JGT/...` *destination*
  - Saut à un sous-programme : `JSR` *destination* et retour de sous-programme : `RTS`.

Saut inconditionnel

- `JMP` *destination* $\rightarrow$ la prochaine instruction à exécuter est désignée par *destination*.
- `JMP` *destination* $\equiv$ "`MOVE` *destination*, `CO`"
- Exemples :
  - JMP # 210 $\rightarrow$ la prochaine instruction à exécuter est à l'adresse 210.
  - JMP 10 $\rightarrow$ la prochaine instruction à exécuter est à l'adresse mémorisée à l'adresse 10.
  - JMP A0 $\rightarrow$ la prochaine instruction à exécuter est à l'adresse mémorisée dans le registre A0.

Sauts conditionnels

Les *saut conditionnels* ne sont effectués que si des conditions sur les bits Z et N du registre d'état sont réalisés.

Instruction	Sens	Conditions
JEQ	**J**ump if **EQ**ual	$Z = 1$
JNE	**J**ump if **N**ot **E**qual	$Z = 0$
JLT	**J**ump if **L**ess **T**han	$N = 0$
JLE	**J**ump if **L**ess or **E**qual	$N = 0$ ou $Z = 1$
JGT	**J**ump if **G**reater **T**han	$N = 1$
JGE	**J**ump if **G**reater or **E**qual	$N = 1$ ou $Z = 1$

Sauts conditionnels

- Un saut conditionnel se trouve généralement après une instruction `CMP`.
- Interprétation globale (sur les deux instructions) :
  - `CMP #5, D0` **suivi de** `JEQ #100` :
    saut à l'adresse 100 si le contenu de D0 égale 5.
  - `CMP D0, D1` **suivi de** `JLT #100` :
    saut à l'adresse 100 si le contenu de D0 est $<$ au contenu de D1
  - `CMP 88, D0` **suivi de** `JGE #100` :
    saut à l'adresse 100 si la valeur à l'adresse 88 est $\geq$ au contenu de D0.

Saut à un sous-programme (et retour)

- Saut (inconditionnel) à un sous-programme `JSR` *destination* :
  - empile l'adresse de l'instruction suivante (celle contenue dans CO) avant de sauter à l'adresse désignée par l'opérande.
  - `JSR` *destination* $\equiv$ "`ADD #2, CO`" **puis** "`PSH CO`" **puis** `JMP` *destination*.
- Retour de sous-programme `RTS` :
  - dépile une adresse (normalement celle empilée par un JSR) et saute à cette adresse.
  - `RTS` $\equiv$ "`POP CO`"

Saut à un sous-programme (et retour)

```
110 :   JSR #150
112 : ...
...
150 : ... (début du sous-programme)
...
162 :   RTS (fin du sous-programme)
```

Ordre d'exécution des instructions : 110, 150 $\rightarrow$ 162, 112.

Equivalence d'un programme en LM0 et en C

- On peut faire en LM0 les mêmes choses que l'on fait en C.
- Il suffit de montrer qu'on peut simuler
  - l'affection
  - l'instruction conditionnelle `if`
  - l'instruction de boucle `while`
  - l'appel d'une fonction

L'affectation de variable

- Le nom d'une variable en C correspond à une adresse.
- Affecter une valeur à une variable correspond à faire un transfert de valeur grâce à une (ou plusieurs) instruction(s) `MOVE`.
- Ex : quand en C, on a l'instruction `x = 1;`,
  en LM0, on fait l'équivalent grâce aux deux instructions
  `MOVE #1,D0` **puis** `MOVE D0, 110`
  (en associant x et l'adresse 110).

L'affectation de variable à un résulat d'expression

```
x = 3 * y + 5;
```

se simule en LM0 ainsi :
(x : adresse 100, y : adresse 101)

```
MOVE 101, D0
MUL #3, D0
ADD #5, D0
MOVE D0, 100
```

L'instruction conditionnelle `if`

L'instruction `if(...) {...} else {...}` se simule en LM0 ainsi :

En C	En LM0
if(*test*)	... : **CMP** *test*
[*Instructions si vrai*]	... : **J***test siFaux*
else	... : [*Instructions si vrai*]
[*Instructions si faux*]	... : **JMP** *après*
[...]	*siFaux* : [*Instructions si faux*]
	après : [...]

L'instruction conditionnelle `if`

Exemple : (on utilise le registre D0 pour mémoriser x)

En C	En LM0
if($x > 3$)	100 : CMP #3,D0
$x = x - 5$;	102 : JGE #108
else	104 : SUB #5,D0
$x = x + 10$;	106 : JMP #110
[...]	108 : ADD #10,D0
	110 : [...]

La boucle `while`

L'instruction `while(...){...}` se simule en LM0 ainsi :

En C	En LM0	
while(*test*)	*boucle*	: **CMP** *test*
[*Instructions si vrai*]	...	: **J***test siFaux*
[...]	...	: [*Instructions si vrai*]
	...	: **JMP** *boucle*
	siFaux	: [...]

La boucle `while`

Exemple :

En C	En LM0
while($x > 10$) $x = x - 1$; [...]	104 : CMP #10,D0 106 : JGE #112 108 : SUB #1,D0 110 : JMP #104 112 : [...]

Les fonctions

- Une fonction C se simule en LM0 grâce à un appel à un sous-programme (se terminant par l'instruction `RTS`) par l'instruction `JSR`.
- Le passage de paramètre peut se simuler
  - en les empilant avant le saut au sous-programme
  - en les dépilant au début du sous-programme (attention à l'adresse de retour située au sommet de la pile)

Les fonctions

Schéma général :

En C	En LM0	
int **f**(*paramètres*)	...	: *empilement des paramètres*
{	...	: **JSR** *f*
corps	...	: *depile la valeur de retour*
return *valeur*;	...	: [...]
}	*f*	: *depile l'adresse de retour*
	...	: *depile les paramètres*
	...	: *corps*
[...]	...	: *empile la valeur de retour*
x = f(*paramètres*);	...	: *empile l'adresse de retour*
[...]	...	: **RTS**

Les fonctions

Exemple de fonction : `int min(int x, int y)`
En LM0 :

```
120 : POP A0      (depile l'adresse de retour)
122 : POP D0      (depile les paramètres)
124 : POP D1
126 : CMP D0, D1
128 : JLT #136
130 : PSH D1      (empile la valeur de retour)
132 : PSH A0      (empile l'adresse de retour)
134 : RTS
136 : PSH D0      (empile la valeur de retour)
138 : PSH A0      (empile l'adresse de retour)
140 : RTS
```

Exemple d'appel de la fonction :

```
160 : PSH 102     (empilement des paramètres)
162 : PSH 103
164 : JSR #120     (appel fonction min)
166 : POP 104     (récupération du résultat)
```

Exemple de fonction récursive

En C	En LM0	
int factorielle(int n)	200 : POP A0	(dépile adresse retour)
{	202 : POP D0	(récupère n)
if($n < 2$)	204 : CMP #2,D0	(compare n et 2)
return 1;	206 : JLE #212	(saute au cas général si $2 \leq n$)
else	208 : PSH #1	(cas de base : on retourne 1)
return n$*$fact($n-1$);	210 : JMP A0	(retour)
}	212 : PSH A0	(on réempile A0)
	214 : PSH D0	(on met la valeur n de côté)
[...]	216 : SUB #1,D0	
y = factorielle(x);	218 : PSH D0	(on empile n-1 pour l'appel récursif)
[...]	220 : JSR #200	(on fait l'appel récursif)
	222 : POP D1	(on récupère factorielle(n-1)
	224 : POP D0	(on récupère la valeur n)
	226 : MUL D0, D1	(on multiplie les deux)
	228 : POP A0	(on récupère l'adresse de retour)
	230 : PSH D1	(on retourne le résultat)
	232 : JMP A0	(retour)

Les tableaux à une dimension

En C	En LM0
char t[10];	10 octets réservés à partir de l'adresse 100
int i;	i associé à l'adresse 110
i = 0;	200 : MOVE #0, D0
	202 : MOVE D0, 110
while(i < 10)	204 : MOVE 110, D0
{	206 : CMP #10, D0
	208 : JLE #224
t[i] = 'a';	210 : MOVE #100, A0
	212 : ADD 110, A0
	214 : MOVE #97, (A0)
i = i + 1;	216 : MOVE 110, D0
	218 : ADD #1, D0
	220 : MOVE D0, 110
}	222 : JMP #204
[...]	224 : [...]

Exemples

- Une chaîne de caractères est codée en C grâce à un tableau de caractères dont le dernier caractère de la chaîne est suivi par un 0.
- Ex : "abc" est stocké dans un tableau déclaré par `char t[10];` alors
  t[0] contient 'a' (97), t[1] contient 'b' (98),
  t[2] contient 'c' (99) et **t[3] contient 0.**
- Nous allons écrire les instructions en LM0 qui permettent de :
  - récupérer la longueur d'une chaîne.
  - concaténer deux chaînes.
  - vérifier si deux chaînes sont identiques.
  - inverser le contenu d'une chaîne.

Exemple 1 : longueur d'une chaîne

```
100 : MOVE #0, D0      (D0 est le compteur de caractères de la chaîne)
102 : MOVE #200, A0    (la chaîne débute à l'adresse 200)
104 : CMP #0, (A0)     (le caractère est-il nul (fin de chaîne) ?)
106 : JEQ #114
108 : ADD #1, D0
110 : ADD #1, A0
112 : JMP #104
114 : ... (D0 contient la longueur de la chaîne)
```

Exemple 2 : concaténation de 2 chaînes

```
100 : MOVE #150, A0    (la 1ère chaîne débute à l'adresse 150)
102 : MOVE #200, A1    (la chaîne résultat débute à l'adresse 200)

104 : CMP #0, (A0)     (le caractère est-il nul (fin de chaîne) ?)
106 : JEQ #116
108 : MOVE (A0), (A1)
110 : ADD #1, A0
112 : ADD #1, A1
114 : JMP #104

116 : MOVE #175, A0    (la 2ème chaîne débute à l'adresse 175)

118 : CMP #0, (A0)     (le caractère est-il nul (fin de chaîne) ?)
120 : JEQ #130
122 : MOVE (A0), (A1)
124 : ADD #1, A0
126 : ADD #1, A1
128 : JMP #118
130 : MOVE #0, (A1)    (on termine la chaîne résultat)
```

Exemple 3 : identité de 2 chaînes

```
100 : MOVE #150, A0    (la 1ère chaîne débute à l'adresse 150)
102 : MOVE #200, A1    (la 2ème chaîne débute à l'adresse 200)
104 : CMP #0, (A0)     (est-on au bout de la 1ère chaîne ?)
106 : JEQ #122         (oui : il faut tester si on est au bout de la 2ème
108 : CMP #0, (A1)     (est-on au bout de la 2ème chaîne ?)
110 : JEQ #130          (oui : les 2 chaînes diffèrent)
112 : CMP (A0), (A1)   (les caractères sont-ils identiques ?)
114 : JNE #130          (non : les 2 chaînes diffèrent)
116 : ADD #1, A0
118 : ADD #1, A1
120 : JMP #104         (on passe à la position suivante)

122 : CMP #0, (A1)     (est-on au bout de la 2ème chaîne ?)
124 : JEQ #130          (non : les 2 chaînes diffèrent)
126 : MOVE #1, D0      (on indique que les chaînes sont identiques)
128 : JMP #132

130 : MOVE #0, D0      (on indique que les chaînes diffèrent)
132 : ... (D0 contient 1 si les chaînes sont identiques et 0 sinon)
```

Exemple 4 : inversion d'une chaîne

```
100 : MOVE #200, A0    (la chaîne débute à l'adresse 200)

102 : CMP #0, (A0)     (le caractère est-il nul (fin de chaîne) ?)
104 : JEQ #110
106 : ADD #1, A0
108 : JMP #102

110 : SUB #1, A0       (on se positionne sur le dernier caractère)
112 : MOVE #200, A1    (on se positionne sur le premier caractère)

114 : CMP A1, A0
116 : JGE #130
118 : MOVE (A1), D0
120 : MOVE (A0), (A1)
122 : MOVE D0, (A0)
124 : SUB #1, A0
126 : ADD #1, A1
128 : JMP #114
130 : ... (la chaîne est inversée)
```