Initiation à l'assembleur

Le langage assembleur est, en programmation informatique, le langage de plus **bas niveau** que peut comprendre la machine.

Les combinaisons de bits du langage machine sont représentées par des symboles dits « **mnémoniques** »

Une instruction machine est une chaîne binaire composée principalement de 2 parties :

instruction machine

Le champ "code opération" qui indique au processeur le type de traitement à réaliser. Par exemple le code "00100110" donne l'ordre au CPU d'effectuer une multiplication.

le champ "opérandes" indique la nature des données sur lesquelles l'opération désignée par le "code opération" doit être effectuée.

Les instructions

Les instructions machines sont relativement basiques (on parle d'instructions de bas niveau), voici quelques exemples :

→ les instructions arithmétiques (addition, soustraction, multiplication...). Par exemple, on peut avoir une instruction qui ressemble à "additionne la valeur contenue dans le registre R1 et le nombre 789 et range le résultat dans le registre R0" (l'adresse mémoire est donnée en base 10 pour souci de simplicité, n'oubliez pas qu'en interne elle est codée en binaire)

→ les instructions de transfert de données qui permettent de transférer une donnée d'un registre du CPU vers la mémoire vive et vice versa.

Par exemple, on peut avoir une instruction qui ressemble à "prendre la valeur située à l'adresse mémoire 487 et la placer dans la registre R2" ou encore "prendre la valeur située dans le registre R1 et la placer à l'adresse mémoire 512"

→ les instructions de rupture de séquence : les instructions machines sont situées en mémoire vive, si, par exemple :

l'instruction n°1 est située à l'adresse mémoire 343,
l'instruction n°2 sera située à l'adresse mémoire 344,
l'instruction n°3 sera située à l'adresse mémoire 345...

Au cours de l'exécution d'un programme, le CPU passe d'une instruction à une autre en passant d'une adresse mémoire à l'adresse mémoire immédiatement supérieure : après avoir exécuté l'instruction n°2 (situé à l'adresse mémoire 344), le CPU "va chercher" l'instruction suivante à l'adresse mémoire 344+1=345.

→ Les instructions de rupture de séquence d'exécution encore appelées instructions de saut ou de branchement permettent d'interrompre l'ordre initial sous certaines conditions en passant à une instruction située une adresse mémoire donnée.

Par exemple :

à l'adresse mémoire 354 nous avons l'instruction :

"Si la valeur contenue dans le registre R1 est strictement supérieure à 0 alors la prochaine instruction à exécuter est l'adresse mémoire 4521, dans le contraire, la prochaine instruction à exécuter est à l'adresse mémoire 355" .

Les opérandes

Les opérandes désignent les données sur lesquelles le code opération de l'instruction doit être réalisée. Un opérande peut être de 3 natures différentes :

→ l'opérande est une **valeur immédiate** : l'opération est effectuée directement sur la valeur donnée dans l'opérande

→ l'opérande est un **registre du CPU** : l'opération est effectuée sur la valeur située dans un des registres (R0,R1, R2,...), l'opérande indique de quel registre il s'agit.

→ l'opérande est **une donnée située en mémoire vive** : l'opération est effectuée sur la valeur située en mémoire vive à l'adresse XXXXX. Cette adresse est indiquée dans l'opérande.

Exemples :

"additionne le nombre 125 et la valeur située dans le registre R2 , range le résultat dans le registre R1",
nous avons 2 valeurs : le "nombre 125" (qui est une valeur immédiate, nous sommes dans le cas n°1) et "la valeur située dans le registre R2" (nous sommes dans le cas n°2)

Quand on considère l'instruction machine : "prendre la valeur située dans le registre R1 et la placer à l'adresse mémoire 512", nous avons 2 valeurs : "à l'adresse mémoire 512" (nous sommes dans le cas n°3) et "la valeur située dans le registre R1" (nous sommes toujours dans le cas n°2)

Le microprocesseur est incapable d'interpréter la phrase "additionne le nombre 125 et la valeur située dans le registre R2 , range le résultat dans le registre R1" tout cela doit être codé sous forme binaire.

Un programme en langage machine est donc une suite très très longue de "1" et de "0".

ce qui vous en conviendrez est quelque peu rébarbatif à programmer : sur les dizaines de milliers de "1" et de "0" qui composent un programme en langage machine de taille modeste, une seule erreur, et votre programme ne fonctionne pas...imaginer la difficulté pour retrouver l'erreur ! Bref programmer en langage machine est extrêmement difficile, pour pallier cette difficulté, les informaticiens ont remplacé les codes binaires abscons par des symboles mnémoniques (plus facile à retenir qu'une suite de "1" et de "0").

Nous avons toujours des instructions machines du genre "additionne le nombre **125** et la valeur située dans le registre **R2** , range le résultat dans le registre **R1**", mais au lieu d'écrire "11100010100000100001000001111101",

nous pourrons écrire

"ADD R1,R2,#125"

Dans les 2 cas, la signification est identique : "additionne le nombre 125 et la valeur située dans le registre R2 , range le résultat dans le registre R1".

Le processeur est uniquement capable d'interpréter le langage machine, un programme appelé "assembleur" assure donc le passage de "ADD R1,R2,#125" à "11100010100000100001000001111101".

Par extension, on dit que l'on programme en assembleur quand on écrit des programmes avec ces symboles mnémoniques à la place de suites de "0" et de "1". Aujourd'hui plus personne n'écrit de programme directement en langage machine, en revanche l'écriture de programme en assembleur est encore chose relativement courante.

Exemples d'instructions en assembleur Motorola 6800

Exemple 1 :

```
LDR R1,78
```

LOAD REGISTER - Place la valeur stockée à l'adresse mémoire 78 dans le registre R1 (par souci de simplification, nous continuons à utiliser des adresses mémoire codées en base 10)

Exemple 2 :

```
STR R3,125
```

STORE - Place la valeur stockée dans le registre R3 en mémoire vive à l'adresse 125

Exemple 3 :

```
ADD R1,R0,#128
```

Additionne le nombre 128 (une valeur immédiate est identifiée grâce au symbole #) et la valeur stockée dans le registre R0, place le résultat dans le registre R1

Exemple 4 :

```
ADD R0,R1,R2
```

Additionne la valeur stockée dans le registre R1 et la valeur stockée dans le registre R2, place le résultat dans le registre R0

Exemple 5 :

```
SUB R1,R0,#128
```

Soustrait le nombre 128 de la valeur stockée dans le registre R0, place le résultat dans le registre R1

Exemple 6 :

```
SUB R0,R1,R2
```

Soustrait la valeur stockée dans le registre R2 de la valeur stockée dans le registre R1, place le résultat dans le registre R0

Exemple 7 :

```
MOV R1, #23
```

MOVE - Place le nombre 23 dans le registre R1

Exemple 8 :

```
MOV R0, R3
```

Place la valeur stockée dans le registre R3 dans le registre R0

Exemple 9 :

```
B 45
```

BRANCH - Nous avons une structure de rupture de séquence, la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 45

Exemple 10 :

```
CMP R0, #23
```

Compare la valeur stockée dans le registre R0 et le nombre 23. Cette instruction CMP doit précéder une instruction de branchement conditionnel BEQ, BNE, BGT, BLT.
Branch if Equal, Branch if Not Equal, Branch if Greater, Branch if Less Than

Exemple 11 :

```
CMP R0, R1
```

Compare la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1.

Exemple 12 :

```
1. CMP R0, #23
2. BEQ 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est égale à 23

Exemple 13 :

```
3. CMP R0, #23
4. BNE 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 n'est pas égale à 23.

Exemple 14 :

```
5. CMP R0, #23
6. BGT 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est plus grand que 23

Exemple 15 :

```
7. CMP R0, #23
8. BLT 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est plus petit que 23

Exemple 16 :

```
HALT
```

Arrête l'exécution du programme

Sources https://pixees.fr/informatiquelycee/