Module 4: Langage assembleur

1 Introduction

Le langage assembleur est de plus en plus rarement utilisé par les programmeurs. Toutefois, il est important de comprendre le fonctionnement interne du processeur et de comprendre également son langage de commande.

Dans ce module, nous allons présenter le langage assembleur et comment les fonctions à savoir les instructions classiques, le transfert des données, les opérations arithmétiques et logiques, les ruptures de séquences, etc. peuvent être implémentées en langage assembleur et transformées en code machine.

Pour faciliter l'apprentissage de ce module, nous allons utiliser un type particulier de processeur, soit le Intel 8086 qui est largement utilisé dans les programmes académiques, plus particulièrement dans les cours d'architecture des ordinateurs vu sa simplicité.

Le processeur Intel 8086 possède les caractéristiques suivantes :

- Le 8086 est un processeur 16 bits, c'est-à-dire qu'il traite des données codées sur 16 bits.
- Le 8086 possède 14 registres de 16 bits qui sont :
  - ax (accumulateur) registre d'usage général contenant des données. Les 8 bits de poids faible se nomment al et les 8 bits de poids fort se nomment ah.
  - bx (base) registre d'usage général contenant des données. Comme ax, bx se décompose en bl et bh.
  - cx (compteur) registre d'usage général contenant des données. Comme ax, cx se décompose en cl et ch.
  - dx (accumulateur double mot) registre d'usage général contenant des données. Comme ax, dx se décompose en dl et dh.
  - si registre d'usage général contenant généralement le déplacement dans un segment d'une donnée (Source Index). On a aussi le ESI pour le "Extended Source Index".
  - di registre d'usage général contenant généralement le déplacement dans un segment d'une donnée (Destination Index). On a aussi le EDI pour le "Extended Destination Index".
  - bp registre utilisé pour adresser des données dans la pile.
  - sp registre pointeur de pile.
  - ip registre pointeur d'instruction (compteur ordinal). Ce registre indique la prochaine instruction à exécuter.

- flags registre d'indicateurs de l'état du processeur. Certains bits de ce registre portent des noms. Ce sont tous des indicateurs binaires comme nous l'avons vu dans le module 3.

— Le 8086 comporte également des registres 16 bits pour contenir des numéros de segment :
  - cs code segment : segment contenant le programme en cours d'exécution.
  - ds data segment : segment contenant les données.
  - es registre segment auxiliaire pour adresser des données.
  - ss stack segment : segment contenant la pile.
— Dans les versions des processeurs qui sont venus après le 8086, nous avions des registres de 32 bits comme le cas du X86. Dans ce cas, les registres généraux s'appellent EAX, EBX, ECX, et EDX. La partie basse des registres, par exemple EAX s'appelle AX (16 bits). De la même façon, la partie basse du registre AX s'appelle AL et la partie haute s'appelle AH, et pareil pour le reste des registres.

Quelques définitions utiles :
- *un bit est une valeur binaire qui, par convention, peut prendre la valeur 0 ou 1 ;*
- *un octet est une donnée codée sur 8 bits ;*
- *un mot est une donnée codée sur 16 bits ;*
- *un Koctet est un ensemble de 1024 (2^{10}) octets.*

2 Caractéristiques d'un programme assembleur

Un programme écrit en assembleur possède une forme bien particulière. En effet, chaque ligne dans le programme comporte une seule instruction. Chaque ligne est composée de plusieurs champs décrits, de gauche à droite, comme suit :
- le champ étiquette, qui peut être vide. Une étiquette est un identificateur composé de lettres, chiffres et de caractères $, %, _ et ?. Quelques exemples d'étiquettes valides : boucle, fin_de_tant_que, etc.
- le champ mnémonique (le nom de l'instruction) : Un mnémonique est, généralement, composé uniquement de lettres. Quelques mnémoniques que nous retrouverons souvent : MOV, CMP, LOOP, ADD, SUB, MUL, OR, XOR, etc. On pourra indifféremment écrire les mnémoniques avec des lettres minuscules ou majuscules.
- le champ opérande (les arguments de l'instruction), qui peut être vide. Les opérandes d'une instruction indiquent les données à traiter, soit sous la forme de valeurs constantes, soit en spécifiant l'adresse mémoire (l'emplacement en mémoire) où se trouve la donnée, soit le nom d'un registre contenant la donnée ou contenant l'adresse mémoire de la donnée.
- le champ commentaire, qui peut être vide. Un commentaire commence par un caractère **;** et se termine en fin de ligne.

Donc, un programme assembleur aura généralement la structure suivante : Le début d'un programme assembleur, par convention, est un bloc de commentaires qui décrivent entre autres l'auteur, le nom du programme assembleur, la date de réalisation, ainsi que son objectif et utilisation.

```
1  ;  Exemple de structure générale de programme assembleur ↩
       8086
2  ;  Nom du fichier :     exemple_cours_INF1427.asm
3  ;  Date           : 06 Juin 2018
4  ;  Objet          : exemple illustratif dâ un programme
5  ;                   assembleur.
6
7  .MODEL       small   ; définit le modèle mémoire.
8  .STACK       4096    ; définit la taille de la pile
9
10 .DATA                ; définit le segment de données
11 ...
12 ...
13 .CODE                ; définit les instructions composant le ↩
       programme
14 ...
15 ...
16 MOV      ax, [bx]
17 ADD      ax, 10
18 ...
19 ...
20 END                  ; fin du programme
```

Les instructions commençant par un point (.) dans le code précédent représentent des directives au compilateur et non pas des instructions à traduire en langage machine. Ces instructions sont appelées ***pseudo-instructions****.*

Les différentes pseudo-instructions sont définies dans les points suivants :

1. .MODEL : permet de définir le modèle de mémoire à utiliser. Cela va naturellement dépondre de la taille du code et des données utilisées dans le programme. Nous pouvons avoir des modèles mémoires comme TINY, SMALL, COMPACT, MEDIUM, LARGE, HUGE, ou FLAT.
2. .STACK : cette directive spécifie le nombre d'octets à réserver pour la taille de la pile. Le choix de la taille de la pile dépend également du programme. Par exemple, si le programme traite des gros tableaux, à ce moment là, la taille de la pile devrait, de préférence, être augmentée. Notez que cette valeur est optionnelle, si la taille de la pile n'est pas spécifiée, la valeur par défaut est 1024 (1k).
3. .DATA : cette directive définit le segment des données. C'est là que les variables sont stockées. Nous pouvons définir dans ce segment les types

de variables par exemple à utiliser. Les types de variables peuvent être définis de la façon suivante :

```
1    .DATA
2    v1: .byte  â  aâ   ; octet
3    v2: .halfword 26 ; 2 octets
4    v3: .word 353 ; 4 octets
```

Les noms v1, v2 et v3 sont appelés des étiquettes. Tout étiquette doit être suivi par (:). Le (.byte) représente le type de variable, dans ce cas un octet, et 'a' représente la valeur de la variable. Nous pouvons avoir plusieurs type de variables à savoir .int pour les entiers, .float pour les flottants (32 bits), .double (64 bits), etc.

4. .CODE : c'est là que nous devons spécifier les instructions du programme à savoir MOV, ADD, MUL, SUB, etc. Nous allons voir dans les points suivants, les différents types d'instructions et leur utilisation en langage assembleur.

Pour pouvoir développer des programmes en langage assembleur, il faut tout d'abord comprendre les instructions liées aux différentes fonctionnalités d'un processeur. Ces fonctionnalités sont présentées dans les sections suivantes.

3 Instructions

Dans cette section, nous allons présenter les différentes fonctionnalités du processeur à savoir le transfert des données, les opérations arithmétiques et logiques, les ruptures de séquences, etc. et comment elles sont traduites en langage assembleur.

3.1 Instructions de transfert de données

Nous avons vu précédemment, que les données peuvent être stockées en mémoire ou dans les registres. Il faut donc prévoir un transfert entre ces différents emplacements. Pour assurer une meilleur performance, le processeur interdit les transferts directs de données entre mémoire et mémoire, et se contente des transferts plutôt mémoire-registre ou entre registres. Les instructions de transfert des données sont résumées dans les points suivants [1] :

— MOV AX, BX : le registre AX reçoit la donnée contenue dans le registre BX.
— MVI AX, #v : le registre AX reçoit la donnée immédiate v indiquée dans l'instruction (MVI : MoVe Immediate).
— LDB AX, (BX) : le registre AX reçoit l'octet (8 bits) mémoire dont l'adresse est indiquée dans le registre BX (LDB : LoaD Byte).
— LDH AX, (BX) : le regsitre AX reçoit les deux octets (16 bits) mémoire dont l'adresse est indiquée dans le registre BX (LDH : LoaD Half word).

1. Instructions tirées du livre d'architceture de l'ordinateur d'Emmanuel Lazard.

— LDW AX, (BX) : le regsitre AX reçoit les 4 octets (32 bits) mémoire dont l'adresse est indiquée dans le registre BX (LDW : LoaD Word).
— STB (AX), BX : l'octet de poids faible de BX est stocké en mémoire à l'adresse indiquée dans le registre AX (STB : STore Byte).
— STH (AX), BX : les deux octets de poids faible de BX sont stockés en mémoire à l'adresse indiquée dans le registre AX (STH : STore Half word).
— STW (AX), BX : les 4 octets de poids faible de BX sont stockés en mémoire à l'adresse indiquée dans le registre AX (STW : STore Word).
— PUSH AX : cette instruction permet de ranger sur le haut de la pile le contenu du registre AX. Ranger un mot par exemple dans la pile permet de décrémenter le pointeur de la pile (SP) de -2 unités Comme illustrée dans la figure 1.
— POP AX : cette instruction permet de récupérer le contenu du registre AX. Récupérer un mot par exemple de la pile permet d'incrémenter le pointeur de la pile (SP) de +2 unités Comme illustrée dans la figure 1.
— XCHG AX, BX : cette instruction permet d'échanger le contenu de 2 registres dans ce cas AX et BX ou bien d'un registre avec une case mémoire. Nous pouvons écrire dans ce cas : XCHG CASE, BX.
— BSWAP registre32bits : cette instruction permet d'inverser l'ordre des 4 octets d'un registre de taille de 32 bits. L'octet de poids le plus fort devient celui de poids le plus faible et on affecte les 2 autres octets de la même manière en les inversant eux aussi. Par exemple : BSWAP 12345678 donne 78563412.

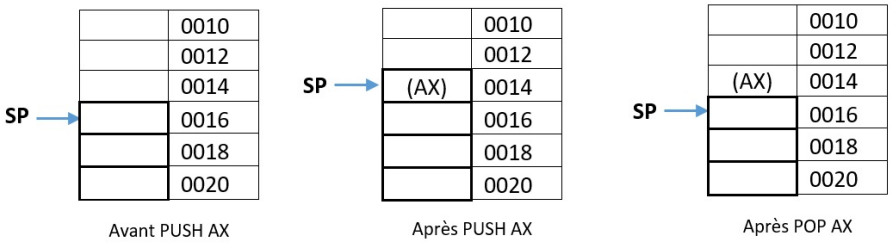

FIGURE 1 – Exemple d'exécution des instructions PUSH et POP et mise à jour du pointeur de la pile (SP).

Comme vous pouvez le constater dans les instructions de transfert de données. Les instructions MOV, et LoaD chargent des données dans les registres, tans disque les instructions de stockage (STore) chargent des données en mémoire à partir des registres.

Toutes les instructions de transfert de données que nous avons vu modifient les bits du registre d'état : les bits Z et N sont positionnés en fonction des résultats obtenus, tant disque les bits C et V sont mis à 0 (pas de retenue ou de débordement).

Maintenant, vous avez appris comment charger et transférer les données, vous pouvez maintenant effectuer des opérations sur les données. C'est ce que nous allons voir dans la section suivante :

3.2 Opérations arithmétiques et logiques

La réalisation des calculs représente la tâche la plus importante du processeur. Pour cela, chaque processeur est doté de plusieurs instructions arithmétiques et logiques qui opèrent sur les registres pour effectuer des calculs. Ces instructions sont présentées dans les points suivants :

- ADD AX, BX, CX : le registre AX reçoit la somme des valeurs contenues dans les deux registres BX et CX (AX = BX + CX).
- ADD AX, BX, #v : le registre AX reçoit la somme de la valeur contenue dans le registre BX et la valeur v (AX = BX + v).
- SUB AX, BX, CX : le registre AX reçoit la différence des valeurs contenues dans les registres BX et CX (AX = BX - CX).
- MUL AX, BX, CX : le registre AX reçoit le produit des 16 bits (2 octets) de poids faible des registres BX et CX. Le résultat du produit sera codé sur 32 bits.
- DIV AX, BX, CX : le registre AX reçoit le quotient de la division entière des registres BX et CX. ce des valeurs contenues dans les registres BX et CX (AX = BX/CX). Le résultat sera codé sur 32 bits.
- AND AX, BX, CX : le registre AX reçoit le résultat du ET LOGIQUE des valeurs contenues dans les registres BX et CX (AX = BX ET CX). Le résultat sera codé sur 32 bits.
- OR AX, BX, CX : le registre AX reçoit le résultat du OU LOGIQUE des valeurs contenues dans les registres BX et CX (AX = BX OU CX). Le résultat sera codé sur 32 bits.
- XOR AX, BX, CX : le registre AX reçoit le résultat du OU EXCLUSIF des valeurs contenues dans les registres BX et CX (AX = BX $\oplus$ CX). Le résultat sera codé sur 32 bits.
- NOT AX, BX : le registre AX reçoit le complémentaire de la valeur contenue dans le registre BX. Le résultat sera codé sur 32 bits.
- NEG AX, BX : le registre AX reçoit l'opposé arithmétique de la valeur contenue dans le registre BX. Le résultat sera codé sur 32 bits (AX = -BX).

```
1 ;  Exemple de code assembleur pour le calcul du reste de la ↩
     division entière\footnote{Exemple enrichi de celui tiré ↩
     du livre de l'architecture de l'ordinateur.}.
```

```
MOV AX, #8
MOV BX, #3
DIV CX, AX, BX       ; CX = AX / BX
MUL DX, CX, BX       ; DX = CX * BX
SUB CX, AX, DX       ; CX = AX - DX (l'ancienne valeur du CX ↩
    sera écrasée)
```

Toutes les instructions relatives aux opérations arithmétiques et logiques modifient les bits du registre d'état. Les opérations comme l'addition, la soustraction et la multiplication affectent directement les bits C et V (retenue et débordement).