
Premiers pas en assembleur

BARBIER J.M. - LGT Dumezil - NSI

22 septembre 2025

1 Le jeu d'instructions

1.1 Définition

1.1.1 Jeu d'instructions

Le **jeu d'instructions** d'un processeur précise

- les registres du processeur
- les instructions qu'il peut exécuter
- la manière dont ces instructions sont représentées en mémoire

1.1.2 Éléments communs

On trouve dans à peu près tous les processeurs :

- des registres
 - R0, R1, .. = registres de données; le nombre de bits qui peut être manipulé dans un registre donne le nombre de bits du processeur
 - PC = compteur de programme, contient l'adresse de la prochaine instruction à exécuter
 - autres registres spécialisés
- des "drapeaux" (flags), qui sont levés ou non après des opérations; par exemple :
 - V = oVerflow (le résultat a dépassé la capacité du registre)
 - N = Negative (le résultat est négatif)
 - Z = Zero (le résultat est nul)
 - C = Carry (une retenue n'a pas pu être comptée dans le résultat)
- des bus d'entrée et de sortie, pour communiquer avec l'extérieur (mémoire,...)
- une ALU : unité d'arithmétique et logique
- une unité de contrôle : reçoit et décode les instructions

1.1.3 Exemples : RISC

RISC = Reduced Instruction Set Computing

Dans le simulateur <https://peterhigginson.co.uk/RISC/V2.php>

- identifier chacun de ces éléments
- en vous plaçant en mode "affichage hexadécimal" (OPTION > hex), déterminez le nombre de bits de ce processeur.

1.2 Instruction machine

1.2.1 Définition

Une **instruction machine** contient

- un code permettant de préciser quelle est l'instruction (déplacer, ajouter,...)

- des opérandes (si nécessaire) : registres, adresses mémoire, ...

Cette instruction est représentée par une séquence de bits, contenant une information sur l'instruction à faire (**opcode**) et des opérandes

Selon **l'architecture** du processeur, les instructions sont plus ou moins nombreuses et ont des fonctions plus ou moins étendues.

Exemples :

- jeu d'instruction x86
- jeu d'instruction ARM
- ...

Pour plus d'infos : fr.wikipedia.org/wiki/Jeu_d'instructions

1.2.2 Le jeu d'instructions x86

Historiquement :

- jeu du processeur 8086 (16 bits) (1978)
- étendu depuis
 - 80186 (16 bits) : 1982
 - 80286 (16 bits) : 1982
 - 80386 (32 bits) : 1986
 - 80486 (32 bits) : 1989
 - Pentium, pentium MMX (32 bits) : 1993
 - PPro, PII, PIII, PIV, Athlon (32 bits) : 1995-2000
 - Athlon 64, Opteron, P4, Intel Core 2 (64 bits) : 2003-2006
 - ... Phenom, Core i7, Bulldozer, Haswell (64 bits) : 2007-...

Détails : fr.wikipedia.org/wiki/X86

et évolution du jeu d'instruction : fr.wikipedia.org/wiki/Jeu_d'instructions_x86

1.2.3 Exemple

Dans le jeu d'instruction RISC16,

0010101011111111 = placer la valeur 0xFF dans le registre R2

- 00101 placer la valeur xx dans le registre yy
- 010 registre 2
- 11111111 valeur FF

Pas très lisible....

1.3 Programme en langage machine (assembleur)

1.3.1 Principe

Les instructions sont écrites dans un langage “humain”, et traduites ensuite en code binaire par le compilateur.

Exemple 1 :

```
00101  010  11111111
MOV     R2,  #255
```

Exemple 2 :

```
// Addition de deux nombres
    INP R0,2
    INP R1,2
    ADD R2,R1,R0
    OUT R2,4
    HLT
```

traduit par

```
0111000100000010
0111000100010010
0110000010001000
0111000110100100
0000000000000000
```

1.3.2 Jeu d'instruction RISC16

Jeu d'instruction correspondant au simulateur :

https://www.peterhigginson.co.uk/RISC/instruction_set_V2.pdf

- déterminer le nombre de bits correspondant à une instruction complète
- déterminez l'instruction (en binaire) correspondant à l'instruction `ADD R3, #15`

1.3.3 Types d'instructions

Uniquement celles que nous allons utiliser :

- Mettre des valeurs dans un registre : `MOV`
- Faire des calculs : `ADD, SUB, MUL, DIV, MOD`
- Faire des opérations binaires : `LSR, LSL, AND, ORR, XOR`
- Lire ou écrire dans la mémoire : `LDR / STR`
- Comparer des valeurs : `CMP`
- Sauter à une autre instruction : `Bxx`

1.3.4 Adresses mémoire, labels

Les accès en mémoire se font en utilisant une adresse; cette adresse peut être exprimée

- directement (en donnant l'adresse)
- “directement” (en donnant un label)
- relativement (via un registre qui contient l'adresse et éventuellement un décalage - offset) - CF plus loin

1.4 Exercices

1.4.1 Programme 1

```
INP R0
INP R1
ADD R2,R1,R0
OUT R2
HLT
```

1.4.2 Programme 2

```
        LDR R0, .dn
        LDR R1, .da
.l1      ADD R1, R1
        SUB R0, #1
        BPL .l1
        STR R1, .dres
        OUT R1
        HLT
.dn      DAT 5
.da      DAT 4
.dres    DAT
```

-
- DAT : directive pour réserver une case mémoire en mettant éventuellement une valeur initiale
 - INP R0 : lit une valeur sur le bus d'entrée et la place dans le registre R0
 - OUT R1 : envoie la valeur contenue dans le registre R1 sur le bus de sortie
 - HLT : arrête le programme
 - LDR R0, .dn : charge dans R0 la valeur contenue à l'adresse marquée par le label .dn
 - ADD R1, R1 : ajoute R1 à lui-même (double la valeur de R1), met le résultat dans R1, c'est un raccourci pour ADD R1, R1, R1 BPL : branch if plus (zéro est considéré comme positif)

1.4.3 Programme 3

```
MOV R1, #5
MOV R2, #65
LDR R3, .data

.loop   STR R2, 0(R3)
        ADD R2, #1
        ADD R3, #1
        SUB R1, #1
        BPL .loop
```

```

        HLT
.data   DAT .data

```

- DAT .data : réserve une case mémoire, initialisée avec l'adresse de cette case (utile pour faire des accès en mémoire)
- BPL : branch if plus (zéro est considéré comme positif)
- STR R2, 0(R3) : enregistre le contenu de R2 dans la case mémoire dont l'adresse est contenue dans R3, plus zéro.

1.4.4 Exercice 1

```

        LDR R0, .di
        MOV R1, #42
        LDR R2, .ds
.l1     ADD R1, R0
        STR R1, 2(R2)
        ADD R0, #1
        STR R0, .di
        ADD R2, #1
        CMP R0, #6
        BLE .l1
        HLT
.di     DAT 0
.ds     DAT .ds

```

BLE : branche if lower or equal du CMP du dessus (si $R0 \leq 6$)

1.4.5 Exercice 2

```

        INP R0, 2
        INP R1, 2
        LDR R2, .dres
.l1     ADD R0, R0
        STR R0, 1(R2)
        ADD R2, #1
        SUB R1, #1
        BPL .l1
        STR R0, .dres
        OUT R0, 4
        HLT
.dres   DAT

```

1.4.6 Exercice 3

```
// Tester avec différentes valeurs
// en utilisant INP R0, 2
    MOV R0, #129
    STR R0, .dfnd
    LDR R1, .dlft
    LDR R2, .drgt
.centr ADD R3, R1, R2
    LSR R3, #1
    CMP R3, R0
    BEQ .equal
    BHI .great
.lower MOV R1, R3
    BRA .centr
.great MOV R2, R3
    BRA .centr
.equal LDR R3, .dres
    HLT
.dlft  DAT 0
.drgt  DAT 1000
.dcnt  DAT
.dfnd  DAT
.dres  DAT
```