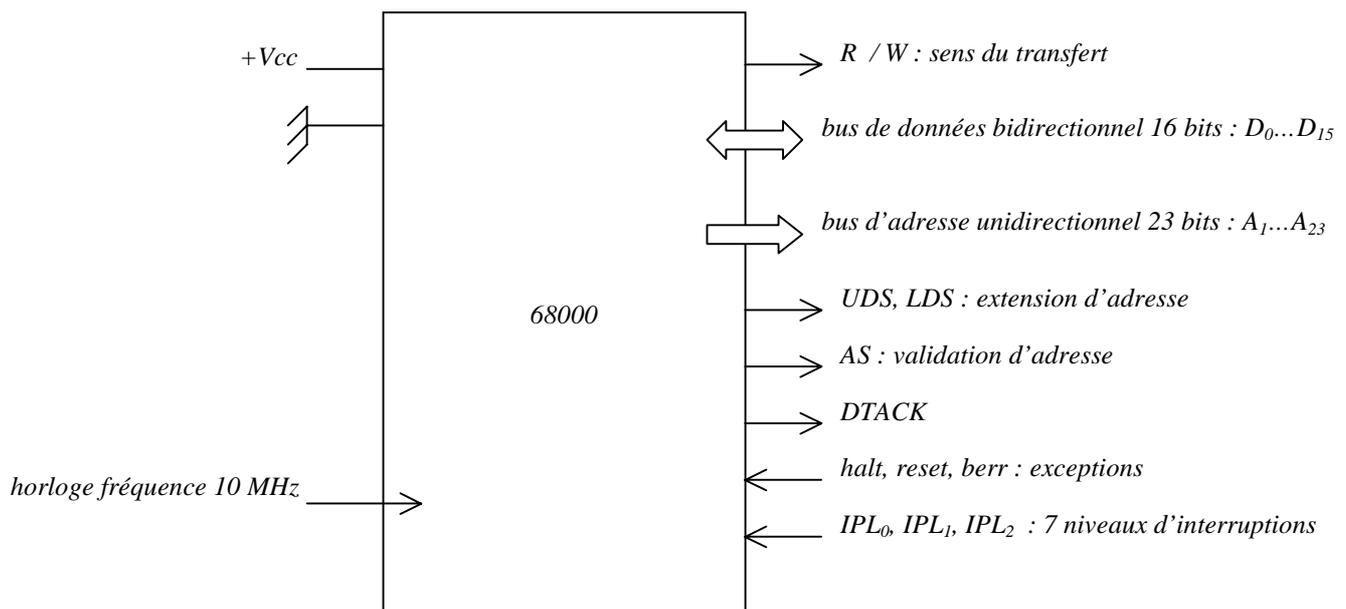


## Cours de Microprocesseur

### Présentation du 68000 de Motorola

Le microprocesseur que nous étudions est le 68000 de *Motorola*. Bien que désuet aujourd'hui, l'architecture de ce composant et sa programmation reste une référence dans l'étude des microprocesseurs.

Le 68000 est un composant électronique de 64 broches, qui possède un bus de données sur 16 bits et un bus d'adressage sur 23 bits, ce qui détermine une région mémoire maximum de 8 Mega-octets<sup>1</sup>. La fréquence de l'horloge est de 10 Mega-hertz, ce qui correspond à un cycle d'horloge de 100 ns. Notons que la plus petite opération nécessite 4 cycles d'horloge.



L'état du processeur est caractérisé par les fonctions codes  $FC_0$ ,  $FC_1$ ,  $FC_2$ . Par ailleurs, on distinguera le mode utilisateur et superviseur.

#### Agencement de la mémoire

...

#### Types de données

<i>bits</i>		
<i>bytes</i>	8 bits	.B
<i>words</i>	16 bits	.W
<i>long words</i>	32 bits	.L

#### Registres internes

- 8 registres de données, 32 bits :  $D_0 \dots D_7$

Ces registres peuvent être manipulés soit comme des *bytes*, soit comme des *word*, soit comme des *long*.

<sup>1</sup>  $2^{20}$  bits = 1 Mega-octet

- **8 registres d'adresses<sup>2</sup>, 32 bits :  $A_0 \dots A_7$  et  $A_7 \text{ bis}$**

On peut manipuler *des adresses longues (long) ou des adresses courtes (word)*. Notons que les adresses réelles du 68000 sont codés sur 24 bits, par conséquent les 8 derniers bits d'une adresse longue ne sont pas significatifs !

Notons également le rôle particulier des deux registres d'adresse  $A_7$  et  $A_7 \text{ bis}$ , encore appelé *USP (user stack pointer) et SSP (supervisor stack pointer)*. **SP** ou  $A_7$  contient l'adresse du **pointeur de pile** relatif au mode courant, c'est-à-dire *utilisateur* ou *superviseur*.

- **PC ou program counter**

Il contient l'adresse de la prochaine instruction à exécuter.

- **SR ou status register, et CCR ou condition code register**

Registre 16 bits : *SR au format word (16 bits) ou CCR au format byte (8 bits)*

C'est le registre d'état, qui comporte des bits indicateurs ou *flags*.

T	-	S	-	-	$I_2$	$I_1$	$I_0$	-	-	-	X	N	Z	V	C
---	---	---	---	---	-------	-------	-------	---	---	---	---	---	---	---	---

*Bits systèmes:*

- $T$  : mode trace
- $S$  : mode superviseur
- $I_2, I_1, I_0$  : masque d'interruptions

*Bits utilisateurs (CCR):*

- $X, N, Z, V, C$  : indicateurs arithmétiques qui peuvent avoir des sens différents selon les opérations...

$Z$	<i>zero</i>	<i>zéro</i>
$N$	<i>negative</i>	<i>bit de poids fort ou de signe</i>
$C$	<i>carry</i>	<i>retenue</i>
$X$	<i>extend</i>	<i>retenue non signée</i>
$V$	<i>overflows</i>	<i>overflow signé</i>

Pour un *move*, seul  $N$  et  $Z$  sont affectés mais pas  $C$  et  $V$ . Pour une opération arithmétique  $N, Z, C, V$  sont affectés.

## Présentation du jeu d'instruction

On distingue des instructions de trois sortes : les transferts de données (ex. : *move*<sup>3</sup>), les opérations arithmétiques (ex. : *add*), et les tests / ruptures de séquence (ex. : *beq*).

Un programme est une suite de code machine représentant des instructions (selon un codage bien défini), et placé en mémoire. Les instructions peuvent avoir des longueurs variables (de un à plusieurs mots<sup>4</sup>). De ce fait, les adresses de début d'instruction sont toujours paires.

### Codage des instructions

...

#### Move

*syntaxes :*

<i>MOVE.B</i>	<i>source , destination</i>
<i>MOVE.W</i>	<i>source , destination</i>
<i>MOVE.L</i>	<i>source , destination</i>

<sup>2</sup> physiquement, 9 registres, mais toujours 8 accessibles

<sup>3</sup> mnémorique de la norme *Motorola*

<sup>4</sup> 1 mot = 16 bits



*MOVE.W*  $-(A_1), D_1$  ;  $A_1 \leftarrow A_1 - 2$

Ces deux derniers modes d'adressage sont utiles pour le parcours des tableaux dans un sens ou dans l'autre.

- **indirect avec déplacement :  $d_{16}(A_x)$**

A la différence du mode d'adressage indirect post-incrémenté ou pré-décrémenté, l'adressage indirect avec déplacement ne modifie pas l'adresse de base  $A_x$ . L'adresse effective est la somme de  $(A_c)$  et de  $d_{16}$ , un déplacement signée sur 16 bits compris entre -17784 et +17783.

*MOVE.L*  $\#\$2000, A_1$   
*MOVE.L*  $3(A_1), D_0$  ;  $AE \leftarrow \$2000 + 3$

- **indirect indexé avec déplacement :  $d_8(A_x, X)$**

L'adresse effective est calculée à partir de l'adresse de base  $A_x$  ajouté d'un déplacement signé sur 8 bits et de la valeur de l'*index*  $X$  (un registre de donnée, un registre d'adresse ou le PC). L'adresse de base n'est pas modifié par ce mode d'adressage.

*MOVE.L*  $\#\$2000, A_1$   
*MOVE.L*  $\#2, D_5$   
*MOVE.W*  $0(A_1, D_5), D_0$  ;  $AE \leftarrow \$2000 + 2 + 0$   
*MOVE.W*  $-1(A_1, D_5), D_1$  ;  $AE \leftarrow \$2000 + 2 - 1$

### Données mémoires

- **DS : data store**

Réservation d'un espace mémoire.

*étiquette* *DS.W* 4

Cette instruction réserve  $4 \times 2$  octets, auquel on pourra faire référence avec *étiquette*.

On rappelle que *#étiquette* renvoie l'adresse de l'étiquette, tandis que *étiquette* désigne son contenu. Ainsi on aura les deux exemples suivants:

*MOVE.L*  $\#étiquette, A_0$   
*MOVE.W*  $étiquette, D_0$

Il existe toutefois une exception à cette règle pour l'instruction *LEA* (*load effective address*) qui charge l'adresse d'une étiquette immédiatement, comme suit :

*LEA.L*  $étiquette, A_0$

- **DC : define constant**

Définition d'une constante au format *byte*, *word*, ou *long*.

*étiquette* *DC.W* 1000

### Instructions de branchement et de branchement conditionnel

- **JMP : jump**

*syntaxe* : *JMP AE*, avec *AE* l'adresse effective en mode relatif, absolu, ou indirect.

*JMP*  $\$1000$

- **BRA : branch always**

syntaxes : *BRA AE*, avec *AE* l'adresse effective en mode indirect.  
*BRA étiquette*, avec *étiquette* qui désigne une adresse symbolique.

début ...  
 ...  
 ...  
*BRA*                    *début*

- **les instructions du type : B<sub>CC</sub>**

syntaxe : *B<sub>CC</sub> étiquette*, avec *CC* le code d'une condition et *étiquette* une adresse symbolique.

test de V	<i>V = 0</i>	<i>BVC</i>	<i>overflow clear</i>	
	<i>V = 1</i>	<i>BVS</i>	<i>overflow set</i>	
test de C	<i>C = 0</i>	<i>BCC</i>	<i>carry clear</i>	
	<i>C = 1</i>	<i>BCS</i>	<i>carry set</i>	
test de Z	<i>Z = 0</i>	<i>BEQ</i>	<i>not equal</i>	<i>≠</i>
	<i>Z = 1</i>	<i>BNE</i>	<i>equal</i>	<i>=</i>
test de N	<i>N = 0</i>	<i>BPL</i>	<i>plus</i>	<i>&gt; 0</i>
	<i>N = 1</i>	<i>BMI</i>	<i>minus</i>	<i>&lt; 0</i>

On dispose encore de tests plus évolués :

	<i>BF</i>	<i>never true</i>	<i>NOP</i>
	<i>BT</i>	<i>always true</i>	<i>BRA</i>
	<i>BGE</i>	<i>greater or equal</i>	
	<i>BGT</i>	<i>greater than</i>	
	<i>BHI</i>	<i>high</i>	
	<i>BLE</i>	<i>less or equal</i>	
	<i>BLS</i>	<i>low or same</i>	
	<i>BLT</i>	<i>less than</i>	

- **TST**

Cette instruction positionne les indicateurs.

*TST.W*                    *\$2000*  
*BEQ*

Les branchements conditionnels nécessitent un positionnement préalable des *flags*, ce qui peut être réalisé automatiquement par certaines instructions. Ex. : *ADD, SUB...*

- **CMP**

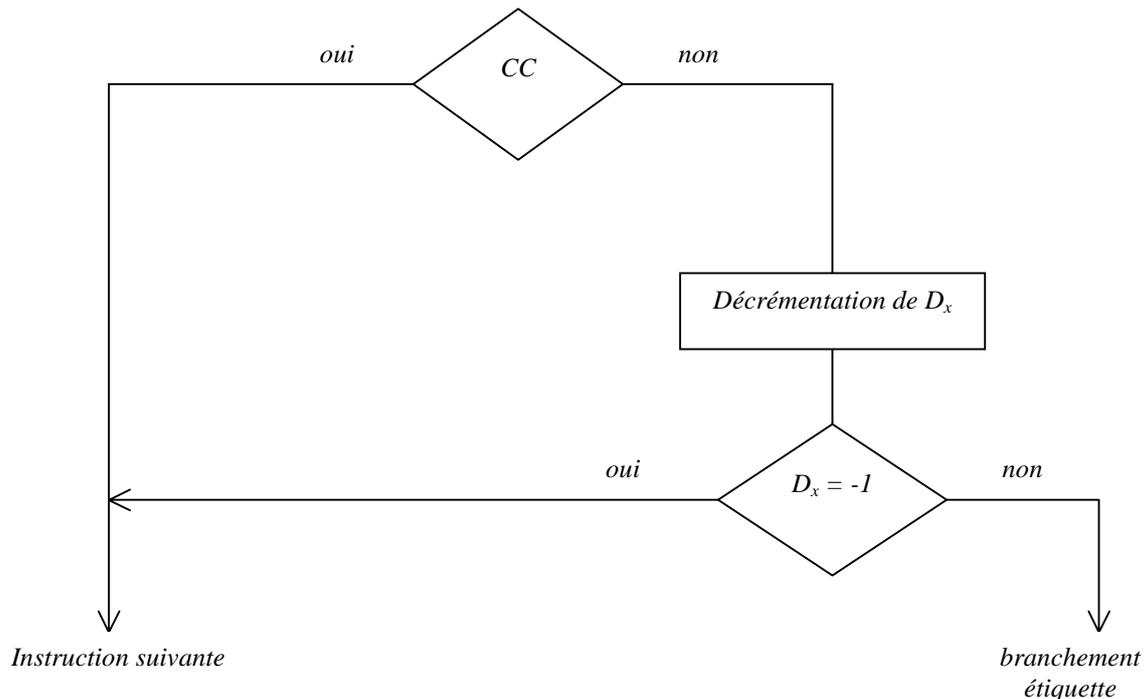
L'instruction de comparaison positionne les *flags* relativement à la soustraction de *la destination* par *la source*.

*CMP*                    *D<sub>0</sub>D<sub>1</sub>*  
*BPL*                    *étiquette*                    ; *branchement si d<sub>1</sub> - d<sub>0</sub> > 0*

- **les instructions du type : DB<sub>CC</sub>**

Syntaxe : *DB<sub>CC</sub>*                    *D<sub>x</sub>étiquette*

Attention! Le compteur *D<sub>x</sub>* est limité à 16 bits pour cette instruction.



*Test, décrément et branchement* : On sort en passant à l'instruction suivante si le compteur se termine ou si la condition *cc* est vraie, sinon on effectue le branchement sur l'étiquette.

Cette instruction est particulièrement adaptée pour réaliser des boucles utilisant un compteur. On donne l'exemple suivant du test des 4 premiers bits du *PADR*. On sort si un bit est à zéro, ou si aucun n'est à zéro.

```

      MOVE.W      #3,D1
repet  BTST.B     D1,PADR
      DBEQ       D1,repet
      CMP.W      #-1,D1      ; test pour savoir comment on est sortit
      BEQ        compteur_épuisée
  
```

## Assembleur

On distingue *l'assembleur* et *le cross-assembleur*. L'assembleur opère directement sur la machine (sur le 68000), tandis que le cross-assembleur opère sur une autre machine. Nous utiliserons un cross-assembleur, en composant nos programmes sur les TX, ce qui a l'avantage d'être plus souple.

### Description du kit

Le kit se compose d'une carte CPU 68010 (10 MHz) et de la carte manip. comprenant 8 interrupteurs + 8 diodes, 8 afficheurs 7 segments, un clavier hexadécimal, un CAN. En outre, on dispose d'une RAM de  $8 \times 128k$ , et d'une ROM de  $8 \times 32k$  associé au moniteur et permettant d'effectuer des opérations de debugage. On dispose également de l'interface // 68230 assurant la liaison avec la carte manip. et réalisant la fonction *timer*, d'une interface série 68564 assurant la liaison avec le TX, et de l'horloge temps réel RTC.

### Utilisation du kit

Ouvrir un *shell*. Commencer par se reconnecter sur le serveur *matho* et lancer le moniteur au moyen de la commande *tkit &*.

A partir d'un éditeur, on compose des fichiers assembleur (extension *.s*) et un fichier de commande (extension *.cmd*) dirigeant l'édition de lien.

Pour la compilation, on commence par générer les fichiers objets (extension *.o*) à partir des fichiers assembleur grâce à la commande : *asm68k fichier -l > fichier.l*.

Puis on effectue l'édition de lien à partir du fichier de commande *main.cmd* par exemple: *lnk68k -F S -c main.cmd -m > main.m*.

Si tout se passe bien cette dernière opération fournit le fichier *main.x* qui est un exécutable au format *Motorola*. On le charge simplement dans la mémoire du kit au moyen de la commande *lkit main.x*.

L'exécution s'effectue par la commande *GO starting\_adress* sous le moniteur.

### Assembleur ligne

- **instructions**

- *HE* : help
- *DU 1000 1030* : visualiser la zone mémoire entre \$1000 et \$1030 (dump)
- *FI 1000 1030 41* : remplissage de la zone mémoire avec 41(fill)
- *TS* : time set
- *DT* : display time
- *OP 1000* : lecture, écriture (sous-fonction : = ..., -, ↵ et Q)
- *USP* : user status pointer
- *SSP* : supervisor status pointer
- *SR* : status register
- *PC = 1000* : modification directe des registres
- *AS 1000* : assembleur en ligne (sous-fonction : ↵ et Q)
- *DI 1000 1030* : désassemble le code machine à partir de l'adresse \$1000 jusqu'à \$1030

- **pas à pas**

- *GO 1000* : exécution du programme à partir de l'adresse \$1000
- *ST* : pas à pas (step)
- *RE* : affiche tous les registres
- *DB* : display all break points
- *CB* : clear all break points
- *B0 = \$1000* : création d'un point d'arrêt à l'adresse \$1000
- *B0* : efface le point d'arrêt B0

- **plantage**

Le *reset* ne réinitialise jamais la mémoire ! Il permet tout au plus d'arrêter le programme en cours. Dans le cas d'une boucle infinie, on peut reprendre la main, au moyen de la commande *C-C*, puis *MCR>ABO*.

### Programme d'assemblage

#### Les sous-programmes

exemple de la tempo...

### **Carte manip. PIT 68230**

#### Présentation du PIT 68230

schéma...

#### Led

...

#### Afficheurs 7 segments

...

#### Clavier

### **Exceptions & Interruptions**