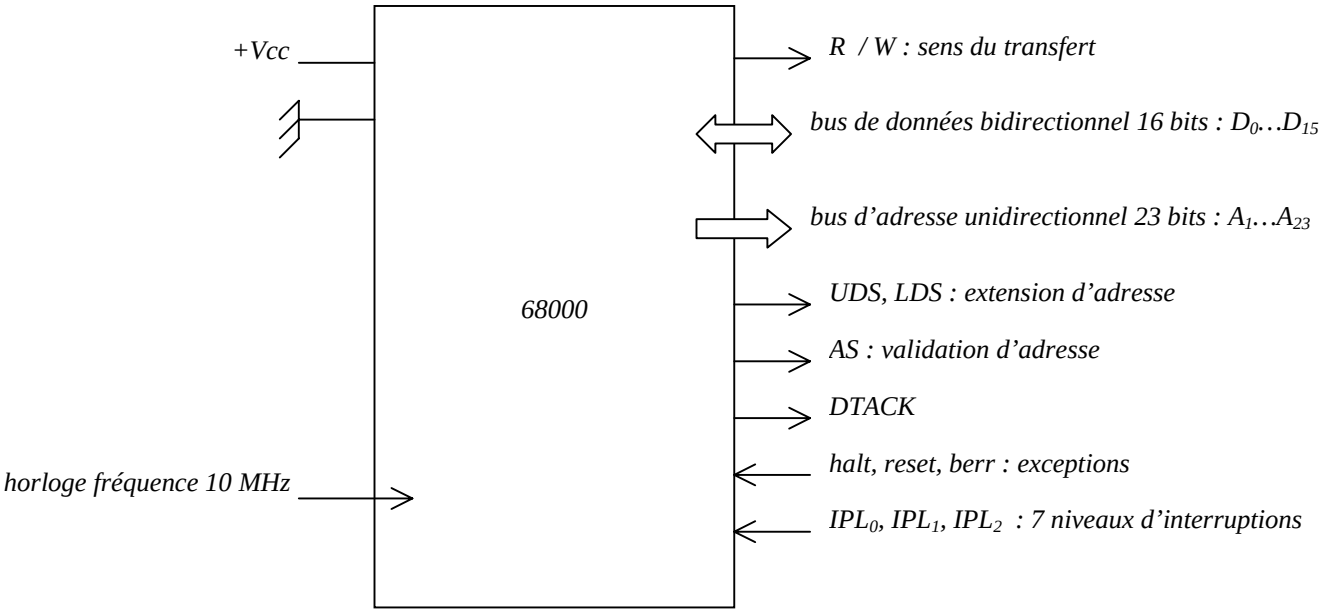
Cours de Microprocesseur

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 bis et un bus d'adressage sur 23 bits, ce qui détermine une région mémoire maximum de 8 Mega-octets[1]. 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

bits		
bytes	*8 bits*	*.B*
words	*16 bits*	*.W*
long words	*32 bits*	*.L*

Registres internes

- ***8 registres de données, 32 bits : D_0 … D_7***

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

[1] 2^{20} bits = 1 Mega-octet

- ***8 registres d'adresses[2], 32 bits : A_0 … A_7 et $A_{7\,bis}$***

  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* $A_{7\,bis}$, 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	*zero*	*zéro*
N	*negative*	*bit de poids fort ou de signe*
C	*carry*	*retenue*
X	*extend*	*retenue non signée*
V	*pverflows*	*overflow signé*

  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*[3]), 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[4]). De ce fait, les adresses de début d'instruction sont toujours paires.

Codage des instructions

…

Move

syntaxes : *MOVE.B* *source , destination*
MOVE.W *source , destination*
MOVE.L *source , destination*

[2] physiquement, 9 registres, mais toujours 8 accessibles
[3] mnémonique de la norme *Motorola*
[4] 1 mot = 16 bits

L'instruction *MOVE* transfert la *source* vers la *destination*, en écrasant le contenu de cette dernière. Par défaut, le format de l'instruction *MOVE* est *.W*.

Modes d'adressage

- ***adressage absolue***

  L'adresse effective est une constante, le plus souvent écrite en hexadécimal comme par exemple *$1000*.

  ```
  MOVE.W    $1000 ,$2000
  ```

- ***direct des registres : D_x ou A_x***

  On opère directement sur les registres soit en lecture, soit en écriture.

  ```
  MOVE.B    D0 ,$2000
  MOVE.W    A0 ,A1
  MOVE.L    $2000,D0
  MOVE.L    A1,$2000
  ```

- ***immédiat : #constante***

  La source est la valeur d'une constante immédiatement citée dans le code machine. Par défaut, les constantes sont décimales. Les constantes hexadécimales doivent être précédées du symbole $ et les constantes binaires du symbole %.

  ```
  MOVE.L    #$12345678,D0
  MOVE.B    #%11110000,D0
  ```

  Dans cette dernière instructions, les 24 derniers bits du registres ne sont pas affectés.

- ***indirect : (A_x)***

  L'adresse effective est le contenue du registre d'adresse A_x.

  ```
  MOVE.L    #$2000,A1
  MOVE.W    (A1),D0          ; MOVE.W    $2000 ,D0
  ```

  On ne passe pas l'adresse directement, mais on donne une référence à cette adresse.

- ***indirect post-incrémenté : (A_x)+***

  L'adresse effective est le contenu de l'adresse A_x ; après exécution de l'instruction utilisant ce mode d'adressage, A_x est incrémenté de 1,2 ou 4 selon le format de l'instruction *.b, .w,* ou *.l.*

  ```
  MOVE.L    #$2000,A1
  MOVE.B    (A1)+ ,D1        ; A1 ← A1 + 1
  MOVE.W    (A1)+ ,D1        ; A1 ← A1 + 2
  ```

  On rappelle que les adresses doivent toujours être paires ; attention, donc à cette instruction :

  ```
  MOVE.B    (A1)+,D1
  ```

- ***indirect pré-décrémenté : - (A_x)***

  L'adresse effective est le contenu de l'adresse A_x décrémentée de 1,2 ou 4 selon le format de l'instruction *.b, .w,* ou *.l.*

  ```
  MOVE.L    #$2000,A1
  ```

```
MOVE.W      -(A1),D1        ; A1 ← A1 - 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,A1
MOVE.L      3(A1),D0        ; AE ← $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,A1
MOVE.L      #2,D5
MOVE.W      0(A1,D5),D0     ; AE ← $2000 + 2 + 0
MOVE.W      -1(A1,D5),D1    ; AE ← $2000 + 2 - 1
```

Données mémoires

- ***DS : data store***

Réservation d'un espace mémoire.

```
étiquette   DS.W    4
```

Cette instruction réserve 4×2 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,A0
MOVE.W      étiquette,D0
```

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,A0
```

- ***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_{CC}***

  syntaxe : B_{CC} *étiquette* , avec $_{CC}$ le code d'une condition et *étiquette* une adresse symbolique.

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

On dispose encore de tests plus évolués :

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

- ***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***

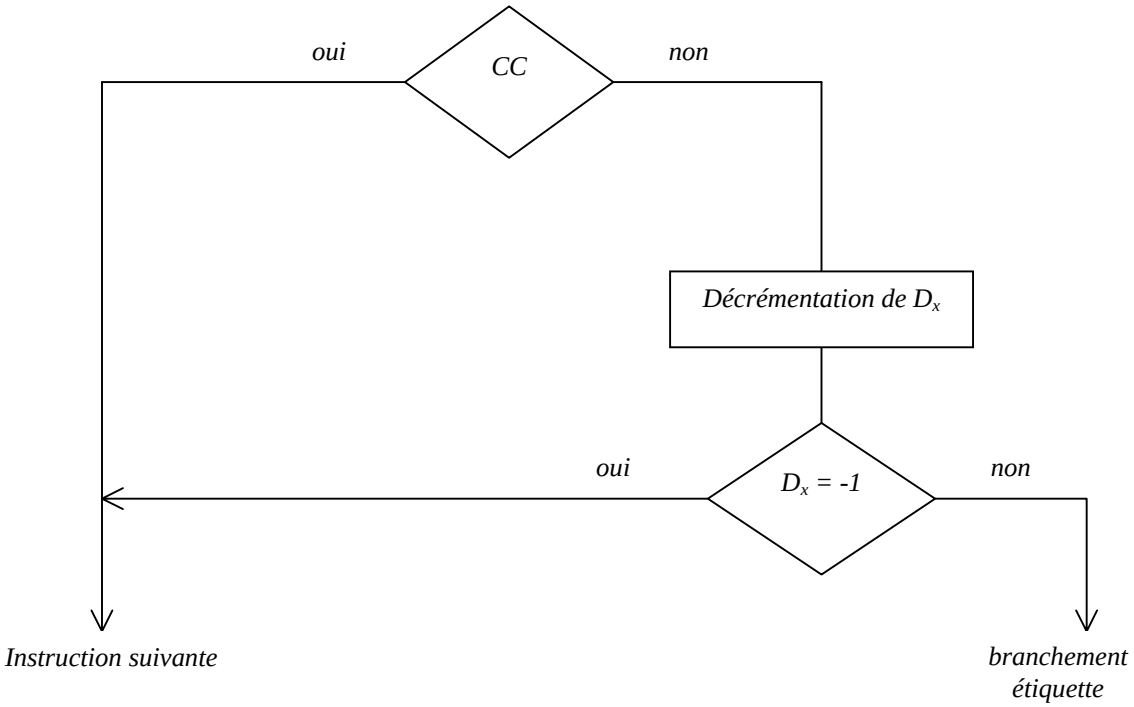
  L'instruction de comparaison positionne les *flags* relativement à la soustraction de *la destination* par *la source.*

```
CMP     D0,D1
BPL     étiquette       ; branchement si d1 - d0 > 0
```

- ***les instructions du type : DB_{CC}***

  Syntaxe : DB_{CC} D_x*,étiquette*

  Attention! Le compteur D_x est limité à 16 bits pour cette instruction.



Test, décrémentation 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
repeter BTST.B      D1,PADR
        DBEQ        D1,repeter
        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 MgHz) 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 cett 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