Polycopié de Cours :
Le Microprocesseur 16 bits "MC68000"

Lakhdari Fethi

Département d'Electronique
Université des Sciences et de la Technologie d'Oran
(USTO)

Table des matières

Introduction Générale

Ce document est destiné aux étudiants de la graduation du département d'électronique pour être un support de cours du module de microinformatique consacré aux systèmes à microprocesseur 16 bits. Le document traite d'une manière plus ou moins détaillée le cas du MC68000 de Motorola.

Après un chapitre introductif, consacré aux notions de base relative à la représentation numérique ainsi que la terminologie de base, le document est reparti en cinq chapitres distincts :

Le premier chapitre donne une description du coté hardware, allant de la description du brochage du microprocesseur, au principales caractéristiques et fonctionnalités, ainsi que le la présentation des cycles relatifs aux opérations de lecture et d'écriture de base.

Dans le second chapitre une description assez détaillée du software du microprocesseur est donnée, où une grande partie du jeu d'instructions est décrite avec un ensemble d'exemples et d'exercices.

Les interruptions et les techniques de décodage d'adresse dans les systèmes à base du MC68000 sont décrites respectivement dans les chapitres trois et quatre.

Des exemples de circuits périphériques d'entrées/sorties numériques sont donnés dans un dernier chapitre pour servir en particulier comme exemple d'implémentation de différents types d'applications.

Le document est doté d'annexes regroupant des résumés d'informations relatives en particulier au jeu d'instructions du microprocesseur, et à certains détails techniques des circuits intégrés utilisés et décrits dans ce documents.

Enfin je tiens à remercier vivement mes collègues et chers enseignants, Monsieur Bouchiba Bahari, et Monsieur Sahari Mohamed qui ont accepté de faire une première lecture et correction de ce document, Qu'ils trouvent ici l'expression de ma profonde gratitude.

Chapitre 1

Introduction et Rappel

Ce chapitre introduit en bref les notions de base relatives à la représentation numérique, ainsi qu'une certaine terminologie couramment utilisées en micro-informatique.

L'objectif est de faciliter l'assimilation et l'utilisation des chapitres suivants de ce document.

1.1 La représentation numérique

Dans un système à base de microprocesseur ou microcontrôleur tel que les cartes d'acquisition et/ou de commande, le traitement d'un signal donné nécessite impérativement sa quantification numérique.

En considérant une représentation pondérée à base 'b', un nombre N peut être écrit sous la forme :

$$N = \sum_{i=0}^{n-1} a_i b^i = a_0 + a_1 b + ... + a_{n-1} b^{n-1} \quad , \qquad N\,[a_{n-1}, ..., a_0] \tag{1.1}$$

où les paramètres de la représentation vérifient la condition .

Remarque

En microinformatique la base 2 est utilisée, car elle permet l'utilisation de deux nombres {0,1}, qui sont pratiquement implémentés via deux niveaux électriques (exemple 0 et 5v en technologie TTL).

Pour une compression importante et une meilleure lisibilité de l'information numérique, La base 16 est très souvent utilisée.

Dans ce cas l'ensemble de la représentation est : {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}.

1.2 La représentation des entiers

1.2.1 Les entiers positifs

Une division euclidienne successive par 2 permet la conversion d'un entier positif N de la base 10 à la base 2.

Une représentation sur n bits (base 2) couvre un interval de nombre N donné par :

$0 \leq N \leq b^n - 1$

Exemple

sur 8 bits :

$0 \leq N \leq 2^8 - 1 = (255)_{10} = (FF)_{16}$

$(a_7, ..., a_0)_8$	$(h_1 h_0)_{16}$	$(N)_{10}$
00000000	00	00
00000001	01	01
00000010	02	02
00000011	03	03
...	...	...
10000000	80	128
10000001	81	129
...	...	...
11111111	FF	255

sur 16 bits :

$0 \leq N \leq 2^{16} - 1 = (65535)_{10} = (FFFF)_{16}$

Remarque

Dans la représentation numérique ,généralement la terminologie anglo-saxonne est la plus couramment utilisée :

- **Bit** : est la compression de l'expression "**bi**nary dig**it**".

- **MSB** (**M**ost **S**ignificant Bit) : est le bit de poids fort d'une séquence binaire donnée

- **LSB**(Least Significant **B**it) : est le bit de poids faible d'une séquence binaire donnée

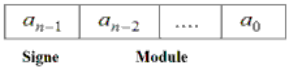
- **Byte** (ou **octet** en français) représente 8 bits.

- **Word** (ou **mot** en français) représente16 bits.

1.2.2 Les entiers signés

La représentation module-signe

il s'agit de la méthode la plus simple pour la représentation des entiers signés. Elle consiste à utiliser le MSB d'un nombre donné comme signe, et le reste des bits comme module.



Un MSB= 1 implique que le nombre est négatif,
sinon (MSB= 0) le nombre est positif
sur 8 bits :
$(11111111)_2 \leq N \leq (01111111)_2$
$(FF)_{16} \leq N \leq (7F)_{16}$
$(-127)_{10} \leq N \leq (+127)_{10}$

$(a_7, ..., a_0)_8$	$(h_1 h_0)_{16}$	$(N)_{10}$
00000000	00	00
00000001	01	01
00000010	02	02
00000011	03	03
...	...	...
01111111	7F	127
10000000	80	00
10000001	81	-01
10000010	82	-02
...	...	...
11111111	FF	-127

L'inconvénient majeur du codage 'module-signe' est le fait que le zéro possède deux représentations différentes (un zéro positif est un autre négatif), chose qui complique l'utilisation de ce codage, en particulier son implémentation matérielle.

La représentation complément à 2

Dans ce cas un entier N est représenté avec une pondération base 2, où le MSB possède une contribution négative :

$$N = -a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + ... + a_1 b + a_0 = -a_{n-1}b^{n-1} + \sum_{i=0}^{n-2} a_i b^i \quad (1.2)$$

Si dans ce cas le zéro possède une seule représentation, le nombre le plus négatif n'a pas de complément.

sur 8 bits :

$(10000000)_2 \leq N \leq (01111111)_2$

$(80)_{16} \leq N \leq (7F)_{16}$

$(-128)_{10} \leq N \leq (+127)_{10}$

$(a_7, ..., a_0)_8$	$(h_1 h_0)_{16}$	$(N)_{10}$
00000000	00	00
00000001	01	01
00000010	02	02
00000011	03	03
...	...	...
01111111	7F	127
10000000	80	-128
10000001	81	-127
10000010	82	-126
...	...	...
11111111	FF	-01

sur 16 bits

$(1000...0000)_2 \leq N \leq (0111...1111)_2$

$(8000)_{16} \leq N \leq (7FFF)_{16}$

$(-32768)_{10} \leq N \leq (+32767)_{10}$

Remarque

- De même que la représentation module-signe, le MSB indique le signe :

si le MSB=1 l'entier est négatif,

sinon (MSB=0) l'entier est positif

- L'avantage majeur de ce codage (comparé au module-signe), est le fait que matériellement l'implémentation d'un circuit soustracteur peut être réduite a l'exploitation du circuit additionneur.

car :

$N - M = N + (-M)$

$= N + M_{c2} = N + (1 + M_{c1})$

Où

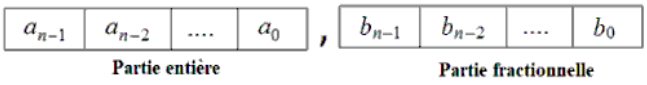
M_{c2} et M_{c1} sont respéctivement le complément à 2 et à 1 de M.

1.3 La représentation des réels

Dans le cas des nombres réels deux représentations sont utilisées avec différentes variantes liées en particulier à la taille des intervalles couverts et à la précision désirée ou obtenue..

1.3.1 La représentation virgule fixe

Dans ce cas le nombre réel est représenté avec deux entités ; où la première indique sa partie entière, et la seconde sa partie fractionnelle.



La partie entière peut être donnée en complément à 2 sur n bits $(-a_{n-1}b^{n-1} + \sum_{i=0}^{n-2} a_i b^i)$,et

la partie fractionnelle sur m bits $(\sum_{i=1}^{m} a_i b^{-i})$

1.3.2 La représentation virgule flottante

Dans ce cas un nombre réel NR s'écrit sous la forme :

$$NR = \pm M.b^E \tag{1.3}$$

Où :

M est la mantisse sur n bits

b : la base

E : l'exposant

Afin d'assurer une portabilité des produits, une normalisation du codage à été nécessaire, et une première norme (IEEE754) est proposée depuis 1985, où quatre formats sont proposés pour la base 2.

D'autres révisions de la norme originale sont proposées, dont la dernière est celle de 2008.

La version d'origine de la norme IEEE 754, datant de 1985, définit quatre formats pour représenter les nombres à virgule flottante en base 2 :

Nom	**Taille**	**Signe**	**Exposant**	**Mantisse**	**Observation**
Simple précision	32 bits	1bit	8 bits (-126 à 127)	24 bits	-
Double précision	64 bits	1bit	11 bits (-1022 à+1023)	53 bits	-
Double précision étendue	79 bits	1bit	15 bits (-16382 à 16383)	64 bits	mis en œuvre avec 80 bits

1.4 Le code ASCII

Dans le cas général, une séquence numérique n'est pas toujours un nombre, et peut être une information non-numérique.

Le code ASCII (American Standard Code for Information Interchange) proposé depuis le début des années soixante est pratiquement la norme de codage de l'information non-numérique.

Ce code a été étendu à plusieurs reprises (principalement par IBM en 1981) pour comprendre outre les lettres, le symbole, les différentes commandes.

En annexe A.1, est donnée la table du code ASCII 8 bits contenant en tout 256 caractères.

"A" $= (41)_{16} = (65)_{10}$, "B" $= (42)_{16} = (66)_{10}$
"a" $= (61)_{16} = (97)_{10}$, "b" $= (62)_{16} = (98)_{10}$
"1" $= (31)_{16} = (49)_{10}$, "2" $= (32)_{16} = (50)_{10}$
"stx" $= (02)_{16} = (02)_{10}$ $(start\ \ of\ \ text)$
"etx" $= (03)_{16} = (03)_{10}$ $(end\ \ of\ \ text)$
"ack" $= (06)_{16} = (02)_{10}$ $(Acknowledge\ \ -accusé\ \ de\ \ réception-)$

1.5 Les circuits mémoire

Dans les systèmes microinformatiques, l'information numérique est stockée sous forme binaire dans des circuits mémoire, de différents types et de technologies de conception.

L'objectif de cette partie est d'introduire la terminologie de base relative à ces circuits, avec une description succincte des pins d'un boitier mémoire de base.

Une première classification des mémoires peut être faite relativement à la volatilité de l'information stockée, et au processus de reprogrammation :

- Les mémoires RAM (Random Access Memory) : sont des mémoires à lecture/écriture appelées improprement ainsi.

Il s'agit d'un type volatile de mémoire, dont le contenu est perdu dès qu'elles ne sont plus alimentées

- Les mémoires ROM : sont des mémoires à lecture seule **(Read Only Memory)**, dont le contenu est fixé lors de la fabrication en usine. Son contenu est non volatile, et reste même sans alimentation.

- Les mémoires EPROM est un type de mémoire intermédiaire entre les deux types précédents, vu qu'il permet une non-volatilité des données, et une possibilité d'effacer le contenu par une exposition du circuit aux rayons ultra-violets.

- Les mémoires EEPROM : sont des EPROM reprogrammables et effaçables électriquement.

- Les mémoires FLASH : sont une classe d' EEPROM, dont le processus de reprogrammation est très rapide.

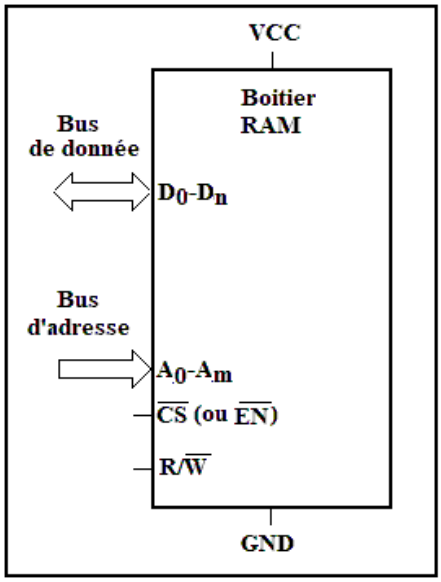


Fig1.1 : Boitier de base d'un circuit RAM

Dans sa forme standard un boitier mémoire présente les pins suivantes :

- les pins d'alimentation VCC (où Vdd) et GND
- une pin de sélection du boitier CS (chip select) où EN (enable)
-des pins fixant le sens du transfert R/$\overline{W}$ (Read/$\overline{Write}$)
- Le bus de donnée : qui regroupe les lignes véhiculant la donnée (généralement 8 bits)
- Le bus d'adresse : qui fixe l'adresse de la donnée concernée par le transfert en cours.

La taille du bus d'adresse fixe en fin de compte la taille du circuit mémoire.
Par exemple :
- pour 12 lignes, la taille de la mémoire=2^{12}=4.2^{10} = $4K$·octets
- pour 23 lignes, la taille de la mémoire= 2^{23}=8.2^{20} = $8M$·octets

Remarque

$1K$=2^{10}=1024
$1M$=2^{20}=1048576

1.6 Exercices

Exercice 1 :

1) Convertir en base 2 puis en base 16 les nombres décimaux suivants :
1789 , 2048 , 0.375 , 17.150
2) Convertir en base 10 les nombres binaires suivants :
1011 , 10110 , 101.1

Exercice 2 :

1) Convertir le nombre π en binaire avec une précision de 10^{-4}
2) Comment faut il arrondir le résultat pour obtenir :
a) l'arrondi par défaut
b) l'arrondi par excès

Exercice 3 :

Soit un nombre décimal ayant D chiffres à droite de la virgule .
Combien faut il de chiffres dans la partie fractionnaire pour écrire ce nombre avec la même précision en base 2,en base 8,et en base 16.

Exercice 4 :

Convertir les nombres :
19 ,-52 , +127 , -127 , -128
1) Suivant la représentation module et signe sur un format de 8 bits (signe compris).
2) Suivant la représentation complément à 2 sur un format de 8 bits (signe compris).

Exercice 5 :

Effectuer les opérations suivantes , sachant que les nombres sont représentés en complément à 2 (format 8 bits) :
A+B, A=10110110, B=01001010
A+B, A=10000010, B=11111111

A-B, A=10110110, B=01001010
A+A, A=11111111
A+A, A=01000000

Exercice 6 :

On considère la représentation suivante des nombres en virgule flottante :
N = S . 0,M.2^E
Où S est le signe du nombre , M est sa mantisse exprimée à l'aide de 15 bits et E l'exposant .
Si l'exposant comporte 8 bits en représentation module et signe ,
1) Quel est le nombre le plus grand représentable (en représentation normalisée) ?
2) Quel est le nombre positif minimum représentable (en représentation normalisée) ?
3) Quel est le plus petit écart entre deux nombres différents ?
4) Exprimer dans ce format le nombre π (arrondi au plus proche).
5) Exprimer dans ce format le nombre 1515.

Chapitre 2

Description du microprocesseur MC68000

2.1 Introduction

Un microprocesseur peut être défini simplement comme étant un processeur intégré dans une puce électronique, où on retrouve les éléments de la logique combinatoire et séquentielle (portes, bascules, registres,...).

L'objectif de cette intégration qui dépasse actuellement facilement les centaines de millions de transistors, est de permettre la réalisation des différentes opérations arithmétiques et logiques, ainsi que les opérations de base de transfert et de branchement.

Depuis le premier microprocesseur d'Intel réalisé en 1971, la finalité était de remplacer la logique câblée par un système permettant l'exécution d'un ensemble d'instructions stockées en mémoire, chose qui facilite l'implémentation d'algorithmes plus ou moins complexes.

Apres une brève introduction sur les systèmes à microprocesseurs de base, ce chapitre étalera sur une description relativement détaillée d'un microprocesseur 16 bits (le MC68000 de Motorola), qui présente des modes de fonctionnement, et des fonctionnalités de base qu'on retrouve pratiquement dans la majorité des microprocesseurs. La première partie de la description concernera le hardware du circuit, suivi d'une seconde sur le software.

2.2 Système à microprocesseur de base

Un système à microprocesseur de base comprend outre l'unité de traitement qui est le microprocesseur un ensemble de circuits et d'étages dont certains sont impératifs pour son fonctionnement tels que le circuit mémoire, et d'autres

dépendant des exigences de l'application elle même telles que les circuits d'interfaces.

Un microprocesseur est un circuit intégré numérique pouvant exécuter une suite d'instructions (un programme) stockées en mémoire. Ces instructions comptent parmi d'autres des opérations de transfert, des instructions arithmétiques/logiques, et des instructions de branchement.

Une instruction peut être un simple cycle d'écriture ou lecture en mémoire, ou une opération plus complexe constituée de plusieurs cycles élémentaires. Dans tous les cas de figure, un signal d'horloge est nécessaire pour cadencer l'accomplissement de l'opération.

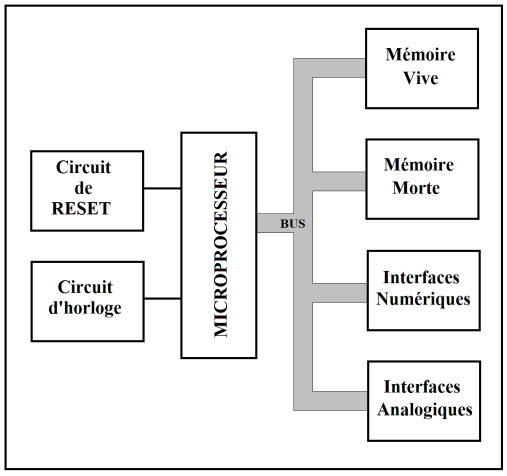


Fig2.1 : Schéma bloc de base d'un système à microprocesseur

Par conséquent, un système à microprocesseur de base comprend pratiquement les blocs suivants :

Une mémoire morte :

Dont le rôle est généralement le stockage du programme (ainsi que les données) relatives à la mise en fonctionnement de l'ensemble du système.

Une mémoire vive :

Destinée au stockage des données (volatiles) issues de l'exécution du programme du système au cours de son exploitation.

Circuit d'horloge :

Chaque instruction est exécutée par le microprocesseur dans un nombre précis de cycles d'horloge. Ce signal d'horloge est généré par un étage oscillateur,

dont la fréquence est définie par le concepteur du système, tout en respectant les limites imposées par le constructeur du circuit intégré.

Circuit de reset (ou de réinitialisation) :

il s'agit généralement d'un étage simple permettant le redémarrage du système.

Pour un microprocesseur donné une telle opération peut être une simple mise à zéro d'une pin particulière du circuit intégré, et la principale conséquence sera le chargement du compteur programme avec une adresse donnée.

Circuits d'interface :

Il s'agit d'un ensemble de circuits spécialisés destinés en particulier à assurer des taches pouvant conduire à une exploitation optimale du microprocesseur.Ces interfaces peuvent être regroupées en deux grandes classes :

- Les interfaces numériques : telles que les circuits destinés à la gestion de la communication série, ou parallèle.

- Les interfaces analogiques : qui sont principalement les convertisseurs numérique- analogique (DAC), et analogique-numérique (ADC).

2.3 Description du MC68000

Le MC68000 de Motorola est l'un des premiers microprocesseurs 16 bits, il est même considéré comme un faux 32 bits vu que son bus interne permet des transferts sur 32 bits.

Parmi caractéristiques que présente ce microprocesseur, et qui seront détaillées par la suite :

- Une fréquence d'horloge qui allant à 16 Mhz

- Un espace mémoire direct de 16 M octets

- Deux modes de fonctionnement (Superviseur, Utilisateur)

- Sept (7) niveaux de priorité d'interruptions, qui peuvent être logicielles, outre les interruptions matérielles.

- Une possibilité de fonctionnement en mode multi-processing

- Un jeu d'instructions plus ou moins complet sur trois tailles mémoire différentes (8, 16, et 32 bits)

Le circuit a été commercialisé sous des formes de boitiers différentes telles

que DIP (DIL), ou PLCC (voire annexe A.2)

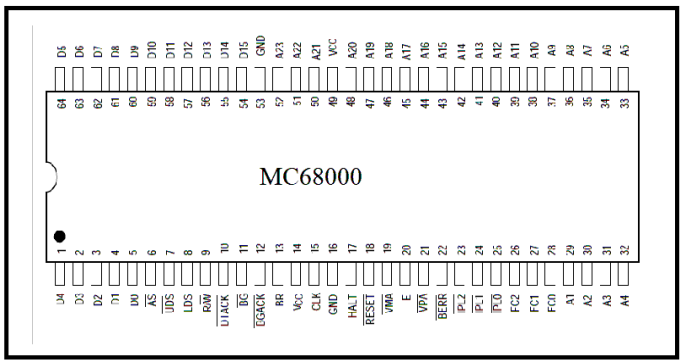


Fig2.2: Le MC68000 (boitierDIL64)

2.3.1 Architecture interne

L'architecture interne du MC68000, est pratiquement une architecture 32 bits, ou même les registres d'adresse sont des 32 bits malgré que le bus d'adresse externe soit un 24 bits.

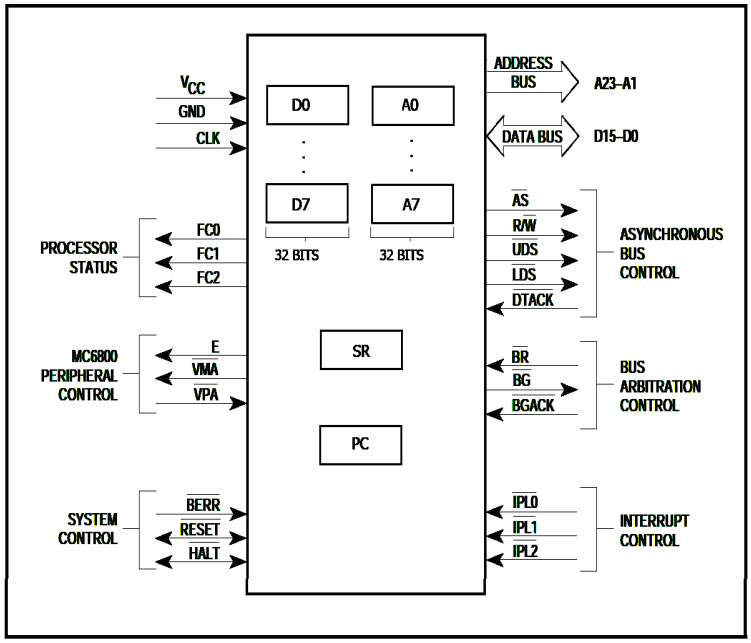


Fig2.3 : Les registres internes,et regroupement fonctionnel des pins du MC68000

Registres de donnée

Le MC68000 dispose de 8 registres de donnée (D0,..., D7) utilisés comme accumulateurs de données de 8, 16, ou 32 bits.

La taille de l'opérande (Byte, Word, Long word) est spécifiée par l'ajout d'un suffixe (B, W, L) à la mnémonique de l'instruction

Registres d'adresse

Le processeur dispose de 8 registres d'adresse, dont 7 registres (A0,.., A6) sont des registres à usage général, et un registre A7 utilisé comme pointeur de pile

Compteur programme PC

Ce registre appelé aussi compteur ordinal est aussi un registre 32 bits, dont le rôle est de contenir l'adresse de l'instruction en cours d'exécution, est qui est automatiquement mis à jour après chaque instruction.

Registres d'état (SR)

Le rôle de ce registre est d'indiquer l'état du processeur au cours de son fonctionnement. Dans le cas du MC68000, il est implémenté en deux octets distincts :

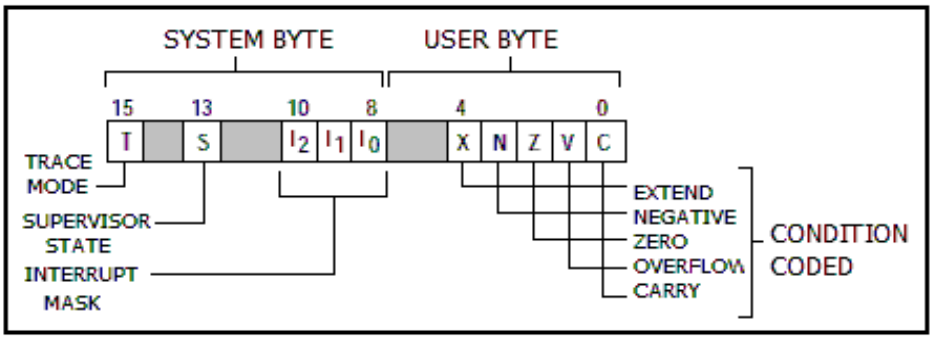


Fig2.4 : Le registre d'état (SR) du MC68000

- L'octet système

Dont les différents indicateurs (flags) sont plutôt liés aux événements systèmes tels que :

- le bit ***S*** *:*

Qui définit le mode de fonctionnement :

Si S=1 : le microprocesseur est en mode superviseur, où tout l'espace mémoire est pratiquement accessible, ainsi que tout l'ensemble du jeu d'instruction, y compris celle dites de privilège.

Si S=0 le mode est UTILISATEUR, où lES instructionS de privilège (par exemple l'instruction reset) ne sont plus possibles, avec une possibilité de limiter l'espace mémoire accessible

- *les bits* $\mathbf{I_0}$ -$\mathbf{I_1}$ -$\mathbf{I_2}$ *:*

Qui définissent le masque d'interruption, où toute demande d'interruption d'un niveau inferieur à la valeur du masque sera ignorée.

I_0	I_1	I_2	Niveau dinterruption
1	1	1	7 (le plus élevé)
1	1	0	6
.	.	.	.
.	.	.	.
0	0	1	1 (le plus bas)
0	0	0	pas d'interruption

Tab2.1 :Le masque d'interruption du registre d'etat

- *le bit* ***T*** *:*

Si le flag du mode Trace est activé (T=1),

le microprocesseur est dérouté après chaque instruction vers un programme d'interruption, qui peut par exemple être la visualisation des contenus des registres internes.

Un tel mode permet l'implémentation facile d'une exécution pas à pas des programmes, chose très utile dans les systèmes de développement.

- L'octet utilisateur

Cette partie du registre d'état nommée aussi CCR (condition code register) contient les cinq (5) flags indicateurs des conséquences de chaque instruction exécutée par le microprocesseur

- *le bit* ***C*** *(carry) :*

Mis à 1, si une retenu est générée après une instruction

- *le bit* ***Z*** *(zero) :*

Mis à 1, s i le résultat est nul.

- *le bit* ***N*** *(negative) :*

Mis à 1, si le résultat est négatif

- *le bit* ***V*** *(overflow) :*

Mis à 1, si un dépassement de taille se produit.

- *le bit* ***X*** *(extend) :*

Ce flag est le même que le bit de retenue C, mais n'est activé que dans le cas des opérations arithmétiques.

2.3.2 Architecture externe

Le bus de donnée et d'adresse

Le MC 68000 effectue les échanges de donnée avec les circuits et les périphériques d'interfaces sur un bus de donnée de 16 bits (D_0-D_{15})

IL possède aussi 24 lignes d'adresse, dont 23 lignes sont explicites (A_1-A_{23}), et une ligne A_0 implicite remplacée par deux lignes : $\overline{UDS}$, et $\overline{LDS}$.

Les deux lignes $\overline{UDS}$ (upper data strobe), et LDS (lower data strobe) indiquent sur quelle octet du bus de donnée est deposé la donnée concernée par le transfert en cours :

- si $\overline{UDS} = 0$, et ,$\overline{LDS} = 1$ le transfert s'effectue sur les 8 bits de poids fort (D8-D15)

- si $\overline{UDS} = 1$, et , $\overline{LDS} = 0$ le transfert s'effectue sur les 8 bits de poids faible (D0-D7)

- si $\overline{UDS} = 0$, et , $\overline{LDS} = 0$ le transfert s'effectue sur les 16 bits (D0-D15)

Dans le modèle (Big Indian) adopté par Motorola l'octet de poids fort est placé avant l'octet de poids faible.

Exemple

- MOVE.B #$44, $4000

implique que la donnée (44)16 sera déposée sur (D8-D15) vu que l'adresse est paire et le transfert sur un seul octet.

- MOVE.B #$72, $4001

implique que la donnée (72)16 sera déposée sur (D0-D7) vu que l'adresse est impaire et le transfert sur un seul octet.

- MOVE.W #$4472, $4000

peut remplacer les deux instructions précédentes avec un dépôt de la donnée (4472)16 sur (D0-D15) vu qu'il s'agit d'un transfert de deux octets (dans ce cas l'adresse est impérativement paire).

Les signaux d'état du processeur

Les pins $\overline{FC_0}$-$\overline{FC_2}$(Function Code) indiquent l'état interne du microprocesseur, et la nature du cycle en cours.

$\overline{FC_0}$	$\overline{FC_1}$	$\overline{FC_2}$	Type de cycle
0	0	0	Réservé
0	0	1	Donnée utilisateur
0	1	0	Programme utilisateur
0	1	1	Réservé
1	0	0	Réservé
1	0	1	Donnée superviseur
1	1	0	Programme superviseur
1	1	1	interruption

Tab2.2 : Les bits de fonctions du registre d'état

L'utilisation de ces signaux permet d'une part la séparation physique entre les

zones mémoire données et programme, ainsi que l'espace mémoire superviseur et utilisateur, augmentant ainsi la sécurité de fonctionnement.

Par conséquent l'espace mémoire adressable du MC68000 (limité à 16Mo par le nombre de lignes d'adresse), peut être ainsi augmenté par un facteur de 4.

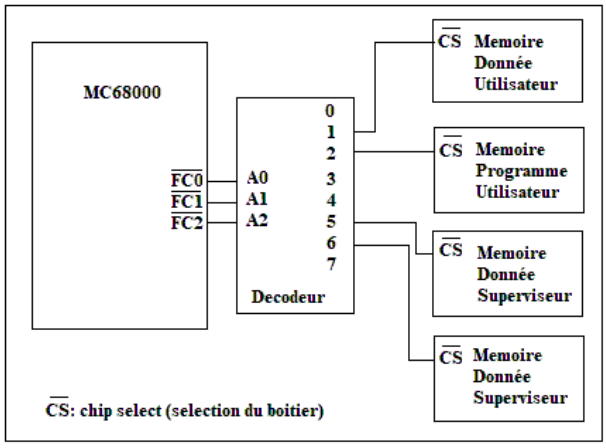


Fig2.5 : Schéma de principe de la séparation des zones mémoire du MC68000

Les signaux de transfert Asynchrone

En plus du bus de données et d'adresses, la gestion temporelle des opérations de transfert entre le microprocesseur et les circuits périphériques nécessite un ensemble de lignes de contrôle :

- Le passage de la pin $\overline{AS}$(Adress Srobe) à l'état bas indique qu'une adresse valide est déposée sur le bus.

- La pin $R/\overline{W}$(Read/Write) indique le sens de transfert.

si $R/\overline{W} = 0$, il s'agit d'une opération de lecture (un transfert du périphérique vers le microprocesseur)

si $R/\overline{W} = 1$, le microprocesseur effectue une opération d'écriture (un transfert vers le périphérique).

- la pin $\overline{DTACK}$ joue le rôle de l'accusé de transfert au cours d'un cycle de transfert asynchrone.

- les pins $\overline{UDS}$,et $\overline{LDS}$ sont pratiquement aussi des lignes de contrôle, puisqu'elles spécifient sur quelle partie du bus est déposée la donnée.

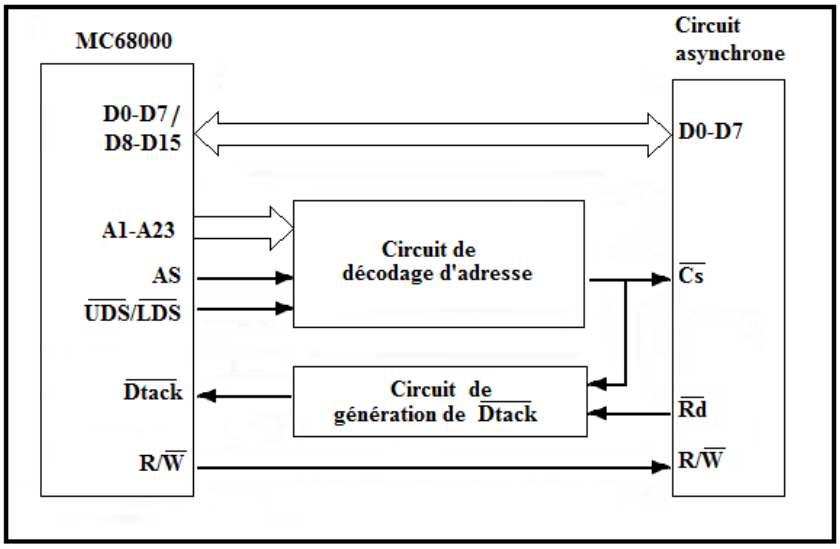


Fig2.6 : Schéma de base d'un interfacage asynchrone du MC68000

Les signaux de transfert synchrone

En plus des lignes de contrôle de transfert asynchrone, le MC68000 de Motorola est aussi muni des lignes de contrôle de transfert dit synchrone.

La finalité était de garantir une compatibilité entre cette génération de microprocesseur 16 bits elle celle des 8 bits (MC68xx) en particuliers avec les différents circuits d'interface déjà réalisés par le constructeur.

Dans une opération de lecture/écriture synchrone, le transfert dure un nombre précis de cycles, et il est cadencé avec une horloge de fréquence égale au dixième de la fréquence d'horloge du microprocesseur.

Les lignes de transfert synchrone sont :

- $\overline{VMA}$: une ligne activée par le périphérique, pour signaler qu'il s'agit d'un transfert synchrone.

- $\overline{VPA}$: une sortie activée par le microprocesseur, comme accusé de réception en réponse à $\overline{VMA}$.

- E : signal d'horloge synchronisant le transfert (E=CLK/10)

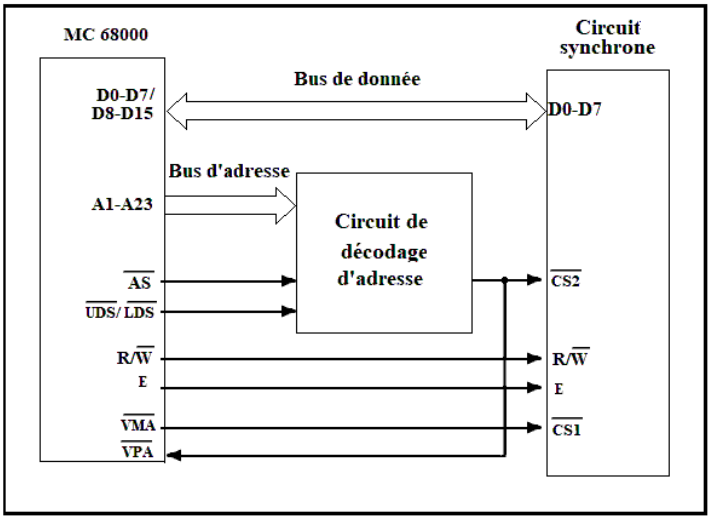


Fig2.7 : Schéma de base d'un interfacage synchrone du MC68000

Les pins de requête d'interruption

Les pins $\overline{IPL_0}$-$\overline{IPL_2}$sont simultanément des lignes de requête d'interruption et de gestion de matérielle de la priorité

Un niveau haut sur les 3 lignes (valeur$\neq(111)_2$) implique l'absence de demande d'interruption, et le microprocesseur continue l'exécution du programme de la tache en cours.

Le cas inverse implique la présence d'une interruption, qui ne sera prise en considération que sauf si le niveau de priorité est supérieur à celui du programme en cours d'execution.

$\overline{IPL_2}$	$\overline{IPL_1}$	$\overline{IPL_0}$	Niveau d'interruption
1	1	1	Aucune requête
1	1	0	1 (le plus bas)
.	.	.	
.	.	.	
0	0	1	6
0	0	0	7 (le plus élevé)

Tab2.3 : Etat des pins de requête d'interruption

Les pins d'arbitrage de bus

Le MC 68000 est conçu pour une possible utilisation dans un système multiprocesseurs, d'où la nécessité de signaux de gestion des requêtes de contrôle du bus. Tout un protocole est à suivre en exploitant les trois signaux :

- $\overline{BR}$: Une transition du niveau haut au niveau bas, implique qu'un autre système demande le contrôle du bus

- $\overline{BG}$: un niveau bas implique que le microprocesseur libérera le bus à la fin du cycle

- $\overline{BGACK}$: un passage au niveau bas implique que le contrôle du bus est pris par un autre système (le demandeur)

Les pins de contrôle du système

Les broches ($\overline{RESET}$,$\overline{HLT}$,$\overline{BERR}$) sont utilisées pour générer des interruptions liés à des problèmes tels que les erreurs d'adressage, ou à la réinitialisation du système.

- Si un cycle de transfert n'est pas achevé correctement, une circuiterie extérieure conçue pour la détection du problème produira le signal d'interruption (sur la pin $\overline{BERR}$), qui déroutera le microprocesseur vers

l'exécution du programme relatif aux erreurs bus.

Dans le cas ou $\overline{BERR}$ est activée simultanément avec la ligne $\overline{HLT}$, alors le microprocesseur exécute une nouvelle fois le cycle bus

- Les pins $\overline{RESET}$, et $\overline{HLT}$, sont des lignes bidirectionnelles à collecteur ouvert.

Pour une réinitialisation (le reset) du MC68000, le constructeur impose la mise à zéro des lignes $\overline{RESET}$, et $\overline{HLT}$ pendant au moins 100ms. Cette opération est impérative à la mise sous tension, mais aussi en cas de blocage du système au cours de fonctionnement.

Un simple circuit monostable, peut assurer le reset matériel du microprocesseur, mais aussi le reset manuel (par bouton poussoir).

En réponse à l'instruction 'RESET ', le MC68000 active la pin $\overline{RESET}$ pour réinitialiser les circuits périphériques sans être affecté.

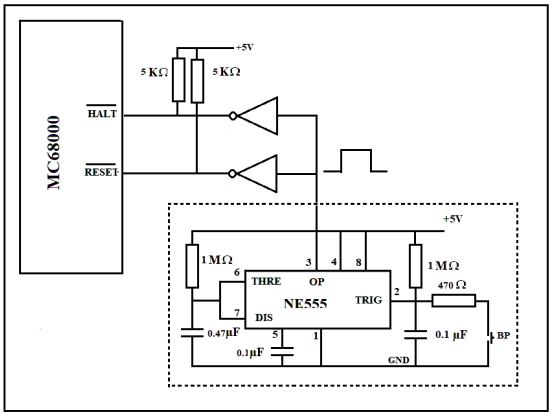


Fig2.8 : Exemple d'un circuit de reset duMC68000

2.4 Les cycles de lecture/écriture de base

L'accomplissement d'une instruction par un microprocesseur revient à exécuter une séquence de cycles bus. Ces cycles durent un nombre de périodes d'horloge durant lesquelles des opérations fondamentales (telles que l'écriture, ou la lecture) sont effectuées. Les deux modes de transfert que permet le MC68000 sont déjà introduits dans la section (2.2.1) et (2.2.2) lors de la description des pins du microprocesseur.

Dans les deux cas de figure le MC68000 exploitent des lignes de contrôle commune telles que $\overline{AS}$ (pour la validation d'adresse sur le bus), ou $R/\overline{W}$ (pour designer le sens du transfert).

Dans le cas d'un transfert synchrone , $\overline{VMA}$ et $\overline{VPA}$ sont utilisées pour confirmer de la part du microprocesseur et du périphérique qu'il s'agit d'un transfert synchrone.

La cadence du transfert est synchronisée avec le signal de la pin E (égale à clk/10).

Dans le cas d'un transfert asynchrone, outre les signaux communs ($\overline{AS}$, $R/\overline{W}$, $\overline{UDS}$, $\overline{LDS}$), le $\overline{DTACK}$ (Data Transfer Acknowledge) est le signal de poignée de main (handshaking) généré par le circuit adressé, et via lequel il confirme, soit que la donnée sur le bus est mémorisé (cycle d'écriture), ou soit que la donnée est sur le bus (cycle de lecture).

Remarque

Par la suite, la description des chronogrammes des opérations de transfert, la demi-période d'horloge (state ou état) est utilisée comme unité de base de temps.

2.4.1 Le transfert asynchrone

Le cycle d'écriture

Au cours d'un transfert asynchrone (Fig 2.6), le chronogramme d'un cycle d'écriture est illustré sur la figure 2.9, pour les trois cas possibles (16 bits, 8 bits adresse paire, et 8 bits adresse impaire).

Dans le cas 16 bits, l'adresse est impérativement paire et $\overline{UDS}$, $\overline{LDS}$ sont simultanément activés (mis à zero).

La séquence du transfert (16 bit) peut être décomposée comme suit :

Etat S0 :Le microprocesseur dépose le code fonction sur les pins FC_0-FC2, et le bus d'adresse est initialement à l'état haute impédance.

Etat S1 :Le microprocesseur dépose l'adresse sur le bus d'adresse.

Etat S2 : Sur le flanc montant de S2, la ligne $\overline{AS}$ est mise à zéro pour indiquer qu'il y a une adresse valide sur le bus.

La ligne $R/\overline{W}$ passe aussi à zéro pour indiquer qu'il s'agit d'une opération d'écriture (un transfert vers le circuit périphérique)

Etat S3 : Le microprocesseur dépose la donnée sur les pins D_0-D_{15}.

Etat S4 : Activation des pins $\overline{UDS}$,et $\overline{LDS}$.

Si le microprocesseur reçoit le signal $\overline{DTACK}$ d'acquittement (sinon $\overline{BERR}$, ou $\overline{VPA}$) avant l'état S7, il le synchronise à l'état S7, sinon il génère des états d'attente Sw.

Etat S5- S6 : Période inactive.

Etat S7 : synchronisation du signal $\overline{DTACK}$.

Le microprocesseur désactive les signaux $\overline{AS}$, $\overline{UDS}$,et $\overline{LDS}$, et le périphérique désactive le signal $\overline{DTACK}$.

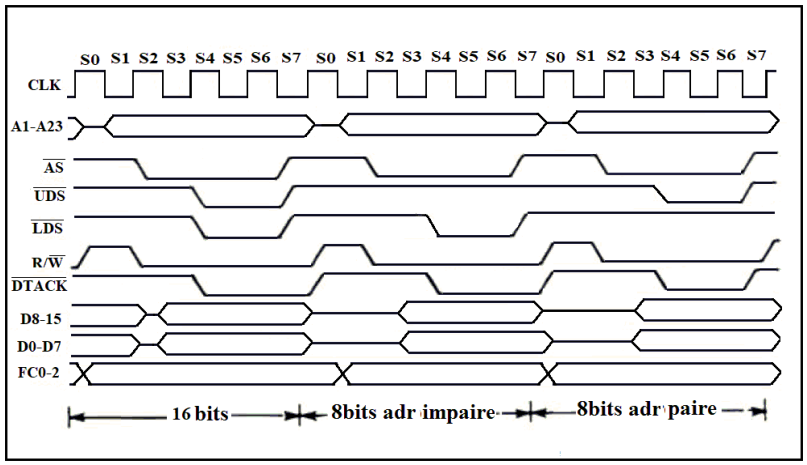


Fig2.9 : Chronogramme d'un cycle d'écriture asynchrone (16, et 8bits)

Les bus d'adresse et de donnée sont remis à l'état haute impédance, et la ligne $R/\overline{W}$ retourne à l'état haut.Dans le cas d'un transfert 8 bits le chronogramme

est pratiquement le même, sauf que si l'adresse est paire, la donnée est déposée sur D_8-D_{15} est seulement le signal $\overline{UDS}$ est activé.

Par contre si l'adresse est impaire, la donnée est déposée sur D_0-D_7 est c'est le signal $\overline{LDS}$ qui est activé.

Le cycle de lecture

La figure 2.10 illustre le chronogramme d'un cycle de lecture asynchrone (16 bits, 8 bits adresse paire, et 8 bits adresse impaire).

La séquence des transfert peut être décomposée comme suit :

Etat S0 :Le microprocesseur dépose le code fonction sur les pins FC_0-FC_2, et La ligne $R/\overline{W}$ est a l'état haut (lecture : un transfert de donnée vers le microprocesseur).

Etat S1 :Le microprocesseur dépose l'adresse sur le bus d'adresse

Etat S2 : Sur le flanc montant de S2, la ligne $\overline{AS}$ est mise à zéro pour indiquer qu'il y a une adresse valide sur le bus, avec une activation des pins $\overline{UDS}$,et/ou $\overline{LDS}$.

Etat S3-S4 :

Le microprocesseur est en attente du signal d'acquittement $\overline{DTACK}$ (sinon $\overline{BERR}$, ou $\overline{VPA}$).

Si rien n'est reçu, alors des états d'attente **Sw** sont générés.

Etat S5 : Période inactive.

Etat S6 : La donnée est prélevée sur le bus (8 ou 16 bits)

Etat S7 : Le microprocesseur désactive les signaux $\overline{AS}$, $\overline{UDS}$,et $\overline{LDS}$, et par la suite le périphérique désactive le $\overline{DTACK}$.

Les bus d'adresse et de donnée sont remis à l'état haute impédance au flanc montant de S7, et la ligne $R/\overline{W}$ reste à l'état haut.

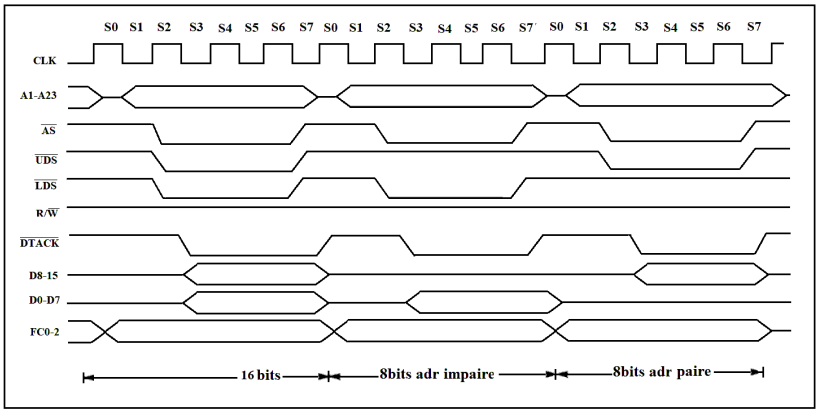


Fig2.10 : Chronogramme d'un cycle de lecture asynchrone (16,et8 bits)

2.4.2 Le transfert synchrone

Dans le cas d'un schéma d'interfaçage synchrone (figure 2.7), les chronogrammes de lecture et d'écriture sont donnés sur la figure 2.11.

L'état des lignes $\overline{LDS}$,et $\overline{UDS}$ marquera la différence entre un transfert 16 bits (remise à zéro simultanée), et les transferts 8 bits (remise à zéro d'une seule ligne)

Comparé à un transfert Asynchrone, les chronogrammes débutent de la même façon.

Si le microprocesseur reçoit un niveau bas sur $\overline{VPA}$ au lieu de $\overline{DTACK}$ à l'état S4, alors le transfert est cadencé avec le signal d'horloge E.

Le microprocesseur confirme que le transfert est synchrone en activant $\overline{VMA}$, pour soit déposer ou récupérer la donnée sur le bus.

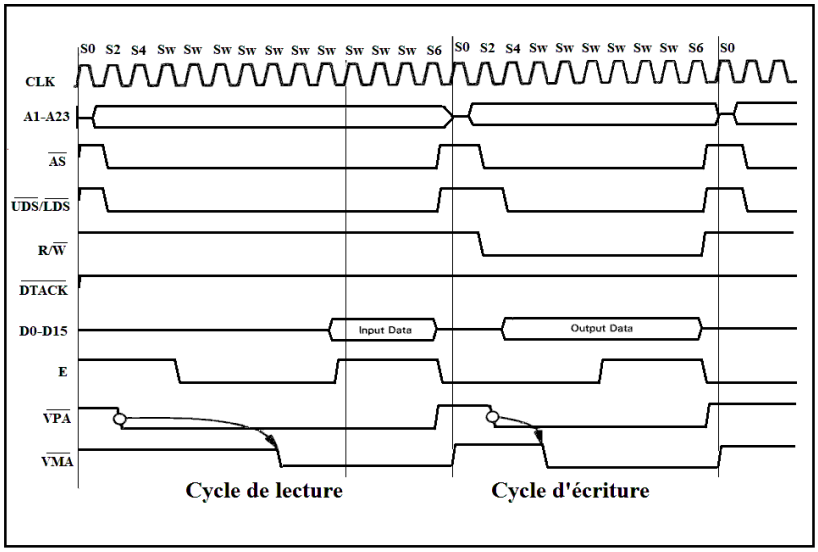


Fig2.11 : Chronogramme d'un cycle de lecture/écriture synchrone (32its)

Chapitre 3

L'assembleur du MC68000

3.1 Introduction

Les instructions exécutées par un microprocesseur sont en réalité un code machine sous forme binaire. Une telle représentation est très difficile à utiliser et à manipuler.

Le langage assembleur est en fin de compte une représentation lisible du code machine sous forme de mnémoniques et symboles, offrant ainsi à l'utilisateur une grande facilité et simplicité de programmation.

Un logiciel (dit généralement aussi assembleur) permet la conversion du programme rédigé en assembleur en code machine.

Dans ce chapitre on décrira les différents modes d'adressage que permet le MC68000, ainsi qu'une description plus ou moins détaillée du jeu d'instructions. Ce dernier est donné sous forme de tableaux dans l'annexe A.3.

Language assembler	Code machine
MOVE.B #$FF,D0	$(103C\ 00FF)_{16}$
MOVE.B #$44,D0	$(103C0044)_{16}$

Le MC 68000 possède un jeu d'instructions dit CISC (Complex Instruction Set Computer), avec un ensemble de 56 mnémoniques (ADD, SUB, MOVE...), 14 modes d'adressage différents, et une possibilité d'exécution des instructions sur différentes tailles (8, 16, 32 bits).

Dans sa forme générale une instruction peut avoir un ou deux opérandes et s'écrit :

*L***abel <code instruction>.T <Opérande source> < Opérande destination>**

Label : est un mot alphanumérique (une étiquette) qui représente l'adresse de l'instruction.

T : est la taille sur laquelle est exécutée l'instruction.

$$\begin{cases} \text{B : 8 bits} \\ \text{W : 16 bits} \\ \text{L : 32 bits} \end{cases}$$

Exemple

Syntaxe générale d'un programme en assembleur 68000

```
adr EQU $ 3000

        ORG $1000
debut   MOVE.B  adr, D0
        ADD.B  val, DO
        MOVE.B  DO,rs1
loop    ADDI.B #1, DO
        CMPI.B #200, D0
        BLT  loop
        BRA suite
val     DC.B 44
rs1     DS.B
suite    ...
        END
```

En plus des instructions, un programme comprend aussi des directives, qui sont l'ensemble des ordres destinés au logiciel assembleur :

ORG : indique au logiciel à partir de quelle adresse placer le code machine.

EQU : est une directive d'équivalence.

END : indique la fin du fichier à assembler.

DC :(define constant) directive d'initialisation de la mémoire avec des constante pendant la phase d'assemblage (sur 8,16, ou 32 bits).

DS : (define space) directive de réservation de zone mémoire pendant l'assemblage (sur 8,16, ou 32 bits)

3.2 Les modes d'adressage

Le mode d'adressage définit la façon avec laquelle est exprimée l'opérande dans une instruction donnée.

Le MC68000 permet 14 modes d'adressage, dont les principaux sont définis dans cette section avec des exemples illustratifs.

La majorité des instructions utilisées dans les exemples se limitent à des opérations simples, de transfert (MOVE) , de mise à zéro (CLR), ou d'opérations arithmétiques (ADD, SUB,...).

3.2.1 L'adressage registre

L'opérande dans ce cas est un registre d'adresse ou de donnée.

Exemple

CLR.B DO	Remise à zéro (8 bits) du registre D0
ADD.W D1, DO	Addition 16 bits de D1, et D0 (résultat dans D0)
MOVEA.L A7, A2	Transfert 32 bits du contenu de A7 vers A2

3.2.2 L'adressage absolu

Dans ce cas l'opérande est spécifié dans l'instruction par son adresse mémoire.

Exemple

$CLR.B\$5000$	Remise à zéro (8 bits) de la case mémoire$(5000)_{16}$
$ADD.WD1, \$4000$	Addition 16 bits de D1,et des cases mémoire $(4000\text{-}4001)_{16}$
$MOVE.L\$2000, \$2600$	Transfert 32bits des cases $(2000\text{-}2003)_{16}$vers $(2600\text{-}2603)_{16}$

3.2.3 L'adressage immédiat

Dans ce cas la valeur est spécifiée directement dans l'instruction précédée par le symbole #.

Exemple

MOVE.B #100,D0	D0←$(100)_{10}$
MOVE.B #$64,D0	D0←$(100)_{10}$
MOVE.B #%01100100,D0	D0←$(100)_{10}$

Le préfixe de chaque donnée spécifie au compilateur assembleur dans quelle base est exprimée la valeur :

- #% ⇒ base 2
- #@ ⇒ base 8
- # ⇒ base 10
- #$ ⇒ base 16

Le mode immédiat accepte aussi le code ascii

Exemple

MOVE.L #'face',D1	D1← $(66616365)_{16}$
MOVE.L #'FACE',D1	D1←$(46414345)_{16}$

L'adressage immédiat court

Certaines instructions de l'assembleur du MC68000 possèdent des variantes plus rapideS, mais avec des limites concernant la taille des opérandes :

- Transfert rapide : MOVEQ #data,Dn
l'opérande source est une donnée 8 bits,
la destination est impérativement un registre de donnée
le résultat dans le registre subit une extension de signe sur 32 bits

- Addition rapide : ADDQ #data,Dn
l'opérande source est une donnée entre 1 et 8,
la destination est un registre de donnée

- Soustraction rapide : SUBQ #data,Dn
l'opérande source est une donnée entre 1 et 8,
la destination est un registre de donnée.

Exemple

MOVEQ #$72,D2	Résultat dans D2← $(00000072)_{16}$
MOVEQ #$8B,D3	Résultat dans D3← $(FFFFFF8B)_{16}$
ADDQ #2,D1	Incrémentation par 2 du contenu de D1
SUBQ #5,D7	Décrémentation par 5 du contenu de D7

3.2.4 L'adressage indirect simple

Dans ce cas la l'opérande est spécifié par son adresse mémoire contenu dans un registre d'adresse.

Une mise entre parenthèses du registre d'adresse différencie ce mode du mode registre.

l'Adresse Effective = (An).

Exemple

```
adr EQU $ 3000

ORG $1000
LEA  $2000, A0        ; A0 ←$2000
LEA  adr, A2          ; A2 ← $3000
CLR.W (A0)            ; Remise à zéro des cases
                        mémoire  $2000 et $2001
MOVE.B #$FF, (A2)  ; $3000 ←(FF)16
...
END
```

3.2.5 L'adressage indirect post-incrémenté

Il s'agit d'un mode d'adressage indirect, où le registre pointeur An est incrémenté (apres l'exécution de l'instruction) par un nombre n.

- n=1 si l'opérande est un 8 bits.

- n=2 si l'opérande est un 16 bits.

- n=4 si l'opérande est un 32 bits.

Exemple

```
ORG $1000
LEA  $2000, A0          ; A0 ← (2000)16
MOVE.W #$54FF,D0        ; D0 ← (54FF )16
MOVE.W D0, (A0)+        ; Les cases mémoire$2000 et $2001 ← (54FF )16
                          Incrémentation du pointeur (A0← $2002).
MOVE.L #00, (A0)+       ; Remise à zéro de quatre octets de $2002 à $2005
                          Incrémentation du pointeur A0← $2006.
MOVE.B #$FF, (A0)+      ; La case mémoire $2006←(FF)16,
                          incrémentation du pointeur A0←$2007
...
END
```

3.2.6 L'adressage indirect pré-décrémenté

Dans ce cas l'adresse de l'opérande est déterminée après une décrémentation du registre d'adresse pointeur avec une valeur n (qui dépend de la taille de l'opération) :

- n=1, dans le cas 8 bits.

- n=2, dans le cas 16 bits.

- n=4, dans le cas 32 bits.

Exemple

```
ORG $1000
LEA  $2008, A0              ; A0 ← (2008)16
 MOVE.W #$54FF,D0           ; D0  ← (54FF )16
MOVE.W D0, -(A0)            ; Décrémentation du pointeur (A0 ←(2008-2=2006)16 ,
                              les cases mémoire $2006 et  $2007←(54FF )16 ,
MOVE.L #00, -(A0)           ; Décrémentation du pointeur   A0 ←(2006-4=$2002)16
                              remise à zéro de quatre octets(de $2002 à $2005 ),
MOVE.B #$FF, -(A0)          ; Décrémentation du pointeur A0 ←(2002-1=2001) 16,
...                           la cases mémoire  $2001  ← ($FF) 16
END
```

3.2.7 L'adressage indirect avec déplacement

L'adresse de l'opérande est la somme du contenu d'un registre d'adresse An et d'un déplacement d signé codé sur 16 bits (-32768≤d≤+32767).

l'opérande est exprimé : d(An).

Exemple

```
ORG      $1000
LEA      $2000,A0        ; A0←($2000)16
MOVE.B #$55,+3(A2)   ; l'adresse $2003←(55)16
...
END
```

3.2.8 L'adressage relatif PC

Dans ce type d'adressage l'adresse effective est obtenue en ajoutant un offset (un déplacement) à la valeur en cours du compteur programme PC.

L'offset est calculé par le logiciel assembleur pendant la compilation du programme.

Exemple

Dans l'exemple suivant deux constantes 32 bits sont définies à la fin du programme avec la directive DC.L, et une zone mémoire 32 bit est réservée avec la directive DS.L L'accès aux trois zone mémoire est réalisé relativement au contenu du PC.

```
ORG $1000
MOVE.L adr1(PC), D0
ADD.L adr2(PC), D0
LEA  result(PC),A2
MOVE.L D0, (A2)
....
....
adr1    DC.L   12345678
adr2    DC.L   98765432
result  DS.L   1
END
```

3.3 Le jeu d'instructions

Les paragraphes suivants présentent une description minimale des instructions de base de l'assembleur du MC68000.

Des détails supplémentaires sont tabulés et résumés dans l'annexe A.3.

3.3.1 Les instructions de transfert

* MOVE. $\begin{pmatrix} B \\ W \\ L \end{pmatrix}$ *source*, *destination*

La destination appriori ne peut pas être un registre d'adresse.

Dans le cas ou la source est un registre d'adresse, la taille se limite à W et L.

* MOVEA. $\begin{pmatrix} W \\ L \end{pmatrix}$ *source*, *An*

Transfert vers registre d'adresse.

Dans le cas où la source est un mot, le résultat est étendu à 32 bits par une extension de signe

* MOVEQ *#data*, *Dn*

La destination est un registre de donnée, et l'opérande source est une donnée 8 bits, qui subit une extension de signe sur 32 bits.

* CLR. $\begin{pmatrix} B \\ W \\ L \end{pmatrix}$ *destination*

Remise à zéro de la destination.

* LEA $< AE >$, *An*

Chargement d'une adresse effective dans un registre d'adresse.

Exemple

```
ORG $1000
LEA  $2000, A0              ; A0←(2000)16
MOVE.B  #$D2, A0            ; A0← (xxxxxxD2)16
MOVE.W  #$8000, D5          ; D5← (xxxx8000)16
MOVE.L  #$78123456, D0      ; D0← (78123456)16
MOVEA.W #$8000, A0          ; A0← (FFFF8000)16
MOVEA.L  #$8000, A0         ; A0← (00008000)16
MOVEQ #$8B,D3               ;D3← (FFFFFF8B)16
...
END
```

* EXG $R1$, $R2$

Permutation du contenu de deux registres (de donnée ou d'adresse)

* SWAP Dn

Permutation du mot de poids fort et de poids faible d'un registre de donnée.

Exemple

```
ORG $1000
LEA  $2000, A0              ; A0←(2000)16
LEA  $6000, A5              ; A5←(6000)16
MOVE.L  #$FFFF, D0          ; D0← (0000FFFF)16
EXG   A0, A5                ; A0←(6000)16 ,A5←(2000)16
SWAP D0                     ; D0← (FFFF0000)16
...
END
```

3.3.2 Les instructions arithmétiques

L'assembleur du MC68000 permet la réalisation des opérations arithmétiques de base (+ , - , * , /, comparaison, calcul d'opposé. . .) .

Des opérations telles que l'addition, la soustraction, ou la comparaison s'effectuent indifféremment sur des registres de donnée ou d'adresse.

- Les instructions d'addition

L'addition avec un registre

$$\begin{cases} ADD.\begin{pmatrix} B \\ W \\ L \end{pmatrix} < AE >,\ Dn & ; Dn \leftarrow [Dn] + (AE) \\ ADD.\begin{pmatrix} B \\ W \\ L \end{pmatrix} Dn,\ < AE > & ; AE \leftarrow [Dn] + (AE) \\ ADDA.\begin{pmatrix} W \\ L \end{pmatrix} < AE >,\ An & ; An \leftarrow [An] + (AE) \end{cases}$$

L'addition avec un adressage immédiat

$$ADDI.\begin{pmatrix} B \\ W \\ L \end{pmatrix} \#data, < AE > \quad ; AE \leftarrow data + (AE)$$

L'addition rapide

$$ADDQ.\begin{pmatrix} B \\ W \\ L \end{pmatrix} \#data, < AE > \quad ; AE \leftarrow data + (AE)\ (1 < data < 8)$$

L'addition avec prise en compte du bit d'extension x

$$ADDX.\begin{pmatrix} B \\ W \\ L \end{pmatrix} Dx,\ Dy \quad ; Dy \leftarrow [Dx] + [Dy] + X$$

$$ADDX.\begin{pmatrix} B \\ W \\ L \end{pmatrix} -(Ax), -(Ay) \quad ; \begin{cases} Ax \leftarrow Ax - n \\ Ay \leftarrow Ay - n \\ (Ax) \leftarrow (Ax) + (Ay) + X \end{cases}$$

Les opérations d'addition affectent les indicateurs (X,N,Z,V,C) du registre CCR à L'exception de de l'instruction ADDA vu qu'elle manipule des adresses.

Exemple

Addition 64 bits de deux données stockées respectivement aux adresses adr1, et adr2, avec un stockage du résultat à partir de l'adresse adr3.

```
adr1 EQU $ 2000
adr2 EQU $ 3000
adr3 EQU $ 4000

ORG $1000
LEA  adr1, A1        ; A1 ←adr1
LEA  adr2, A2        ; A2 ← adr2
LEA  adr3, A3        ; A3 ←adr3

MOVE.L  ( A1)+, D0   ; D0 ←( A1) (32 bits de poids forts)
MOVE.L  ( A1), D1    ; D1 ←( A1) (32 bits de poids faibles)
MOVE.L  ( A2)+, D2   ; D2 ←( A2) (32 bits de poids forts)
MOVE.L  ( A2), D3    ; D3 ←( A2) (32 bits de poids faibles)

ADD.L   D3, D1       ; D1← D3 + D1 (32 bits de poids faibles)
ADDX.L  D2, D0       ; D0 ← D2 + D0 +X (32 bits de poids forts)
MOVE.L  D0, (A3)+    ; Sauvegarde des 32 bits de poids forts
MOVE.L  D1, (A3)     ; Sauvegarde des 32 bits de poids faibles
...
END
```

Les instructions de soustraction

Les instructions de soustraction du MC6800 sont pratiquement similaires à celles d'addition :

$$\begin{cases} SUB.\begin{pmatrix} B \\ W \\ L \end{pmatrix} < AE >,\ Dn\ ; Dn \leftarrow [Dn]\text{-}(AE) \\ SUB.\begin{pmatrix} B \\ W \\ L \end{pmatrix} Dn,\ < AE >\ \ ; AE \leftarrow (AE)\text{-}[Dn] \\ SUBA.\begin{pmatrix} W \\ L \end{pmatrix} < AE >,\ An\ \ ; An \leftarrow [An]\text{-}(AE) \\ SUBI.\begin{pmatrix} B \\ W \\ L \end{pmatrix} \#data, < AE >\ \ ; AE \leftarrow (AE)\text{-}data \\ SUBQ.\begin{pmatrix} B \\ W \\ L \end{pmatrix} \#data, < AE >\ \ ; AE \leftarrow (AE)\text{-}data\ \ (1 < data < 8) \\ SUBX.\begin{pmatrix} B \\ W \\ L \end{pmatrix} Dx,\ Dy\ \ ; Dy \leftarrow [Dy]\text{-}[Dx]\text{-}X \end{cases}$$

Les instructions de multiplication et de division

Dans le cas des opérations de multiplication et de division la taille des opérandes est fixe.

La multiplication signée

$MULS < AE >,\ Dn\ \ ; Dn\_{32} \leftarrow (Dn)_{16} * (AE)_{16}$

La multiplication non signée

$MULU < AE >,\ Dn\ \ ; Dn\_{32} \leftarrow (Dn)_{16} * (AE)_{16}$

Dans les deux cas, les opérandes source et destination sont codés sur 16 bits (poids faibles de l'opérande), et le résultat tient sur 32 bits (d'un registre de donnée).

Dans le cas où les opérandes sont codés sur 8 bits, il est nécessaire de les représenter sur 16 bits avant l'opération de multiplication :

- par une extension de signe dans le cas signé.

- par une remise a zéro de l'octet de poids fort dans le cas non-signé.

Exemple

```
ORG $1000
MOVE.W  #$ FFFF, D0        ; D0 ←( FFFF)16
MOVE.W  #$ FFFE, D1        ; D1 ←( FFFE)16
MOVE.W  #$ 0002, D2        ; D2 ←( 0002)16
MOVE.W  #$ 0002, D3        ; D3 ←( 0002)16
; Multiplication signée 16bits * 16bits
MULS  D0, D1          ;D1 ←( FFFF* FFFE)16 =( 00000002)16 =( 2)10
MULS  D0, D3          ;D1 ←( FFFF* 00002)16 =( FFFFFFFE)16 =( -2)10
; Multiplication non signée 16bits * 16bits
MULU  D0, D2          ;D2←( FFFF* 00002)16 =( 000001FFFE)16=( 131070)10
; Multiplication signée 8bits * 8bits
MOVE.B  #$ FF, D4          ; D4 ←(FF)16
MOVE.B  #$ FE, D5          ; D5 ←(FE)16
EXT.W    D4                ; D4 ←(FFFF)16 =(-1)10
EXT.W   D5                 ; D5 ←(FFFE)16 =(-2)10
MULS    D4, D5             ;D1 ←( FFFF* FFFE)16 =( 00000002)16 =( 2)10
; Multiplication non signée 8bits * 8bits
CLR.W D6
CLR.W D7
MOVE.B  #$ 02, D6          ; D0 ←(0002)16 =(2)10
MOVE.B  #$ FE, D7          ; D1 ←(00FE)16 =(254)10
MULU    #$ D6, D7          ;D7 ←( 00002* 00FE)16 =( 00001FC)16 =(508)10
...
END
```

Exemple

Calcul de l'expression $S = A^2 + B^2 + C^2$, où A,B, et C sont des entiers signés codés sur 16 bits

```
ORG $1000
MOVE  #$0003, D0       ; D0 ←(0003)16 =(3)10
MOVE  #- 2, D1          ; D1 ←(FFFE)16 =(- 2 )10
MOVE.W  #$ F03, D2     ; D2 ←(FF03)16=(-253)10
MULS  D0, D0          ;D0 ←( 00000009)16 =( 9)10
MULS  D1, D1          ;D1 ←( 00000004)16 =( 4)10
MULS  D2, D2          ;D2 ←(FF03* FF03)16
                            =( 0000FA09)16 =(64009)10
ADD.L  D1, D0          ;D0 ←( 0000000D)16 = (13 )10
ADDX.L  D2, D0         ;D0 ←( 0000FA16)16 = (64022)10
...
END
```

La division signée
$DIVS < AE >,\ Dn \quad ; Dn\_32 \leftarrow (Dn)_{32}/(AE)_{16}$

La division non signée

$DIVU < AE >,\ Dn \quad ; Dn\_{32} \leftarrow (Dn)_{32}/(AE)_{16}$

Dans les deux cas l'opérande source est sur 16 bits, et l'opérande destination sur 32 bits.

Le résultat dans le registre de donnée est réparti en deux mots :

- Les 16 bits de poids faibles : le quotient .
- Les 16 bits de poids forts : le reste de la division entière.

Exemple

Calcul de l'expression $S = (1 + A.B)/C$, où A,B, et C sont des entiers signés codés sur 8 bits

```
ORG $1000
MOVE.B  #-2, D0      ; D0 ←(FE)16  =(-2)10
MOVE.B #-8, D1       ; D1 ←(F8)16 =(-8)10
MOVE.B  #5, D2       ; D2 ←(5)16=(5)10
EXT.W    D0          ; D0 ←(FFFE)16 =(-2)10
EXT.W   D1           ; D1 ←(FFF8)16 =(-8)10
EXT.W   D1           ; D1 ←(0005)16 =(05)10
MULS  D0, D1         ;D1 ←( 00000010)16 =( 16)10
ADDI.L  #1, D1       ;D0 ←( 00000011)16 =( 17)10
DIVS  D3, D1         ;D1 ←(00020003)16 ,
                            quotient=03, reste=02
...
END
```

3.3.3 Les instructions logiques

L'assembleur du MC68000 permet la réalisation des opérations logiques de base (ET , OU , XOR , Complément).

Ces instructions logiques ne manipulent jamais des registres d'adresse.

Dans le cas du 'ET' logique les instructions disponibles sont :

$$\begin{cases} AND.\begin{pmatrix} B \\ W \\ L \end{pmatrix} < AE >, Dn & ; Dn \leftarrow (AE) \text{ AND } Dn \\ AND.\begin{pmatrix} B \\ W \\ L \end{pmatrix} Dn, < AE > & ; AE \leftarrow (AE) \text{ AND } Dn \\ ANDI.\begin{pmatrix} B \\ W \\ L \end{pmatrix} \#data, Dn & ; Dn \leftarrow Dn \text{ AND } data \end{cases}$$

- Des instructions de formes similaires réalisent le 'OU' logique, et le OU exclusif : $OR, ORI, EOR, EORI$

- $NOT < AE >$: réalise le complément à 1 du contenu de l'adresse effective.

Exemple

```
ORI  #$20, D0          ; Mise à 1 du bit 5 de D0
ANDI  #$F7, D1         ; Mise à 0 du bit 3 de D1
NOT.L  D2              ; Complément à 1 de D2
AND.B  #$EF, CCR       ; Mise à 0 du bit X du registre CCR
```

3.3.4 Les instructions de comparaison

Les instructions de comparaison sont identiques à celle de soustraction, sauf que le résultat n'est pas affecté à l'opérande destination, mais uniquement les flags (C, N,Z, et V) du registre CCR qui sont modifiés

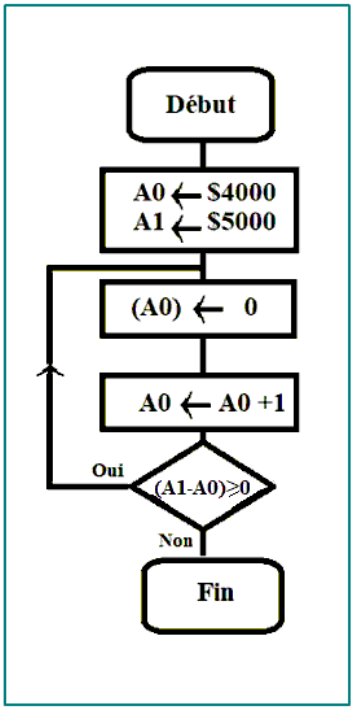
Les instructions de comparaison du MC68000 permettent des opérations sur des registres de donnée et d'adresse, des valeurs immédiates, et des données en mémoire :

$$\begin{cases} CMP.\begin{pmatrix} B \\ W \\ L \end{pmatrix} < AE >,\ Dn & ;(Dn\text{-}(AE))\ affecte\ le\ CCR \\ CMPA.\begin{pmatrix} B \\ W \\ L \end{pmatrix} < AE >, An & ;(An\text{-}(AE))\ affecte\ le\ CCR \\ CMPI.\begin{pmatrix} B \\ W \\ L \end{pmatrix} \#data, < AE > & ;((AE)\text{-}data)\ affecte\ le\ CCR \\ CMPM.\begin{pmatrix} B \\ W \\ L \end{pmatrix} (Ax)+, (\ Ay)+ \begin{cases} ((Ay)\text{-}(Ax))\ affecte\ le\ CCR \\ Ax \leftarrow Ax + n \\ Ay \leftarrow Ay + n \end{cases} \end{cases}$$

Exemple

Mise à zéro d'une zone mémoire délimitée par deux pointeurs A0, et A1.

```
        ORG $1000
        LEA  $5000, A0
        LEA  $5200, A1
boucle  MOVE.B  #0, D0
        MOVE.B  D0, (A0)+
        CMPA A0,A1        ;(A1-A0)
        BPL boucle        affecte le CCR
        END
```



L'instruction BPL (Branch if plus) provoque un branchement vers l'étiquette , si le bit N =0 (après le calcul de A1-A0)

Remarque

Dans le programme la remise à zeo de la case mémoire pointée par par A0 est effectuée en deux étapes, avec une incrémentation du pointeur dans la deuxieme instruction.

3.3.5 Les instructions de branchement

Le jeu d'instructions du MC68000 dispose de deux types de branchement :

Le branchement inconditionnel

BRA *etiquette* ; Saut à l'adresse pointée par l'étiquette.
JMP *AE* ; Saut à une adresse effective.

l'instruction **JMP** est plus générale que **BRA** car elle permet plusieurs modes d'adressage.

Le branchement conditionnel

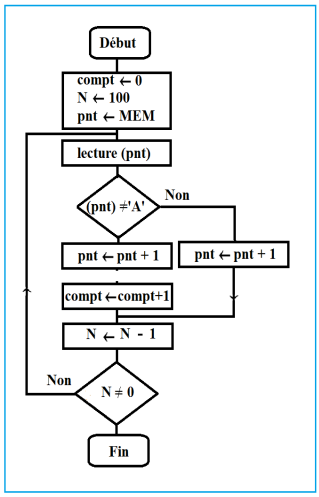
La forme **:B**$_{CC}$ *etiquette* ;(*CC* : *Condition Code*)
La condition du branchement depend des indicateurs du registre CCR.
par exemple :
B$_{CC}$: (branch if carry clear) branchement si la retenu est zéro (bit c=0)
B$_{CS}$: (branch if carry set) branchement si la retenu est 1 (bit c=1)
B$_{EQ}$: (branch if equal) branchement en cas d'égalité (bit z=1)
B$_{NE}$: (branch if not equal) branchement en cas d'inégalité (bit z=0)
L'utilisation des instructions de branchement dépend aussi de la nature des données traitées (annexe A.3) :

	Le cas signé	Le cas non signé
$>$	**B**$_{GT}$	**B**$_{HI}$
$\leq$	**B**$_{LS}$	**B**$_{LE}$
$<$	**B**$_{LT}$	**B**$_{CS}$

Exemple

Détermination du nombre de caractère 'A' dans une chaine de 100 caractères ascii situés en mémoire à partir de l'adresse MEM.

```
        MEM  EQU $ 2000
        ORG $1000
        CLR.W D7              ; compt=0
        MOVE.B #100, D1       ; N=100
        LEA  MEM, A0          ; pnt=MEM
loop    CMPI #'A',(A0)+
        BNE suite
        ADDQ.W  #1, D7
suite   SUBQ.W  #1, D1
        BNE loop
        ...
        END
```



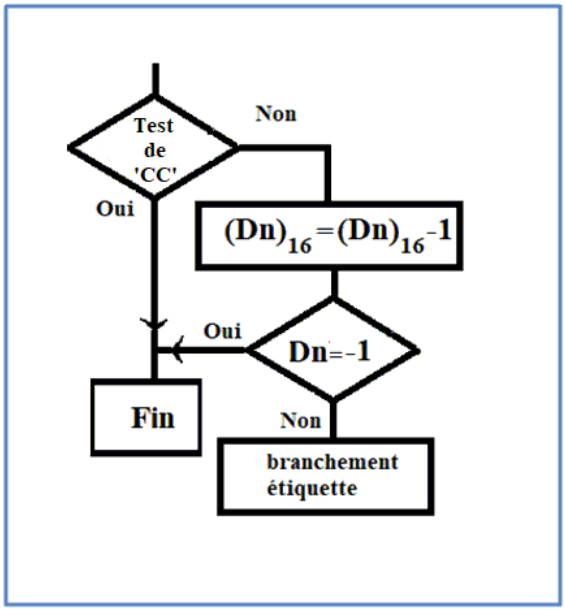
Le branchement conditionnel

La forme :DB_{CC} $Dn, etiquette$;$(CC : Condition\ \ Code)$

Tout d'abord la condition 'cc' est testée, puis le registre est décrémenté par 1.

La sortie de la boucle se produit si la condition est vérifié ou si le registre atteint -1.

Ainsi pour une valeur initiale N, la boucle peut être effectuée (N+1) fois (si la condition 'cc' n'est jamais vérifiée).



- Généralement les instructions de la forme sont placées en fin de la séquence à répéter

- L'instruction est un cas particulier de cette forme de branchement (condition cc= false), car la sortie de la boucle est contrôlée par la valeur du registre Dn qui doit atteindre -1.

Par conséquent le nombre d'itération=N+1 (où N est la valeur initiale affectée à Dn).

Exemple

Initialisation à zéro d'une zone mémoire de 256 octets pointée par le registre A0

```
        MEM  EQU $ 3000

        ORG $1000
        LEA  MEM, A0          ; pointeur=MEM
        MOVE.W  #255, D1
loop    CLR.B (A0)+
        DBF  D1, loop
;Si l'adresse pointé est paire
        LEA  MEM, A0          ; pointeur=MEM
        MOVE.W  #127, D1
loop2   CLR.W (A0)+
        DBF  D1, loop2
;ou encore
        LEA  MEM, A0          ; pointeur=MEM
        MOVE.W  #63, D1
loop2   CLR.L (A0)+
        DBF  D1, loop2
        ...
        END
```

3.3.6 Les instructions de décalage

Les instructions de décalage sont largement utilisées en arithmétique pour remplacer les instructions de multiplication et division par des puissance de 2, vu qu'elles offrent un temps d'exécution plus rapide que les instructions (MULU, MULS, DIVU, DIVS)

Rappel :

-Dans le cas des entiers non signés :

* la multiplication par 2 équivaut à un décalage gauche par un bit, en remplacant le bit manquant par un zéro.

* la division par 2 équivaut à un décalage droite par un bit, en remplaçant le bit manquant par un zéro.

-Dans le cas des entiers signés (complément à 2) :

* La multiplication par 2 équivaux un décalage gauche par un bit, en remplaçant le bit manquant par un zéro.

* La division par 2 équivaux un décalage à droit par un bit, en remplaçant le bit manquant par le bit de signe (MSB).

L'assembleur du MC68000 dispose de deux types d'instructions de décalage :

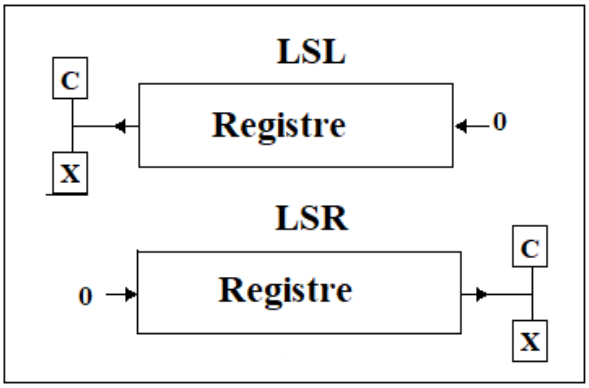
- Le décalage logique (logical shift) : compatible à la division et la multiplication non signées par 2 :

$LSR.\begin{pmatrix} B \\ W \\ L \end{pmatrix}\ Dx,\ Dy$; décalage droite de Dy

$LSRI.\begin{pmatrix} B \\ W \\ L \end{pmatrix}\ \#N, Dy$;décalage droite de N bits du registre Dy $(1 \leq N \leq 8)$

$LSL.\begin{pmatrix} B \\ W \\ L \end{pmatrix}\ Dx,\ Dy$; décalage gauche de Dy

$LSLI.\begin{pmatrix} B \\ W \\ L \end{pmatrix}\ \#N, Dy$;décalage gauche de N bits du registre Dy $(1 \leq N \leq 8)$



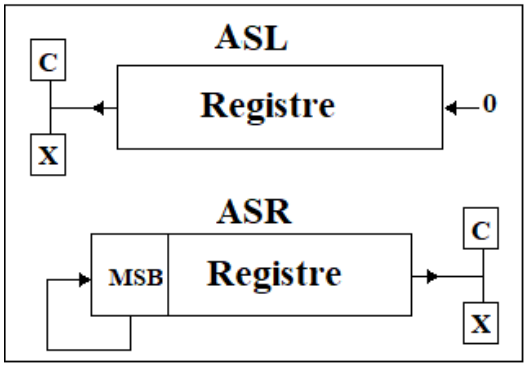
- Le décalage arithmétique : compatible à la division et la multiplication signées par 2 :

$ASR.\begin{pmatrix} B \\ W \\ L \end{pmatrix}\ Dx,\ Dy$; décalage droite de Dy

$ASRI.\begin{pmatrix} B \\ W \\ L \end{pmatrix}\ \#N, Dy$;décalage droite de N bits du registre Dy $(1 \leq N \leq 8)$

$ASL.\begin{pmatrix} B \\ W \\ L \end{pmatrix}\ Dx,\ Dy$; décalage gauche de Dy

$ASLI.\begin{pmatrix} B \\ W \\ L \end{pmatrix}\ \#N, Dy$;décalage gauche de N bits du registre Dy $(1 \leq N \leq 8)$



Exemple

```
ORG $1000
; Nombre signés (en complément à 2)
MOVE.B  #$FE, D0        ; D0 ←(FE)16  =(-2)10
ASL.B   #$1, D0         ; D0 ←(FC)16  =(-4)10 , X=C=1
MOVE.B  #$80, D1        ; D1 ←(80)16  =(-128)10
ASR.B   #1, D1          ; D1 ←(C0)16  =(-64)10, X=C=0
; Nombre non signés
MOVE.B  #$05, D1        ; D1 ←(05)16  =(5)10
LSL.B   #1, D1          ; D1 ←(0A)16  =(10)10, X=C=0
MOVE.B  #$81, D0        ; D0 ←(81)16  =(129)10
LSR.B   #$1, D0         ; D0 ←(40)16  =(64)10 , X=C=1
...
END
```

9

Remarque

La forme des instructions de rotation est similaire à celles des instructions de décalage, et sont présentées dans l'annexe 1.

3.4 Exercices

Exercice 1 :

Préciser pourquoi les instructions suivantes sont illicites :
MOVE.L $FF001,D6 MOVE.B $FF,A1
CLR.B A1 MOVEA.L A0,D0
MOVEQ.W #01,S$4045 MOVEQ #$2C,D5
MOVE.B A0,A1 MOVE.B CCR

Exercice 2 :

Sachant que la mémoire est initialisée pendant la phase d'assemblage par :
ORG $1F264
DC.W $AB04,$5266,$FF11
Quel est le résultat de chacune des instructions suivantes :
MOVE.B $1F264,D0 MOVEQ #$AB,D1
MOVE.L $1F2C,D2 MOVE.W #$A164,A0
MOVE.L $0001f264,A1 MOVE.L (A1),D3
MOVE.W (A1)+,D4 MOVE.W -(A1),D4

Exercice 3 :

Quel est le contenu des registres après l'exécution des instructions suivantes :
MOVEQ #$1B,D1 MOVEQ #$A4,D0
MOVEA.W #$EA10,A0 MOVEA.W #$2A01,A2
Donner l'équivalent de ces instructions en utilisant « MOVE »

Exercice 4 :

Ecrire en langage assembleur les programmes suivants :

- un programme qui initialise les cases mémoire de $12002 ā $12002 par la valeur $1c, et les 5 octets ā partir de $12003 par la valeur $ab
- un programme qui charge la mémoire par le message 'ELECTRONIQUE-USTO' ā partir de l'adresse $3000 (utiliser ensuite une directive pour réaliser la même opération)
- un programme réalisant la permutation des données de la zone mémoire $12000- $12002 avec ceux de la zone $12001-$120003

Exercice 5 :

Donner l'état des flags du registre CCR après chaque instruction :
MOVE.L #$F001,D6
MOVE.W #$F001,$14560
CLR.L D2
MOVEQ #129,D1

```
MOVE .L #-2,D3
MOVE #$6F ,CCR
```

Exercice 6 :

Ecrire un programme qui additionne deux nombres non signés codés sur 16 bits, et rangés respectivement aux adresses mémoire adr1, adr2.

Le résultat de l'addition est sauvegardé à l'adresse mémoire adrs.

Exercice 7 :

A, B, deux nombres non signés codés sur 8 bits et rangés en mémoire respectivement aux adresses adr1, adr2.

1) Développer un programme qui calcule l'expression :

S=(A/8)+B*2

(Avec le transfert du résultat dans le registre D2)

- en utilisant les instructions de multiplication et de division
- en utilisant les instructions de décalage

2) Refaire la question (a) dans le cas ou a et b sont signés codés sur 8 bits

Exercice 8 :

Ecrire un programme qui calcule le produit de deux nombres 'a', et 'b' non signés rangés respectivement aux adresses mémoire adr1, et adr2,

dans les deux cas suivants :

- multiplication 16bits x 16 bits, résultat sur 32 bits
- multiplication 32bits x 32 bits, résultat sur 64 bits

Le résultat de la multiplication est sauvegardé à partir de l'adresse mémoire adrs.

Exercice 9 :

Soit à permuter deux zones mémoire de 1 K Octet l'une pointée par le registre A1 , l'autre par A2.

Ecrire un programme assembleur qui effectue cette opération.

Exercice 10 :

On désire calculer la somme et la moyenne de 100 nombres signés stockés en mémoire à partir de l'adresse $2000 .

1) Quelle est la taille envisageable pour les résultats (Somme ,Moyenne)

2) Développer un programme assembleur qui permet de faire ce calcul .

Exercice 11 :

Un ensemble de 1000 entiers signés se trouve en mémoire à partir de l'adresse pointée par A2 . Développer un programme assembleur permettant de :

1. Compter le nombre d'éléments positifs (résultat dans D0) .
2. Compter le nombre d'éléments négatifs (résultat dans D1) .
3. Compter le nombre d'éléments nuls (résultat dans D2)

Chapitre 4

La pile et les exceptions

4.1 Introduction

Pratiquement le MC68000 peut être soit en état de fonctionnement normal équivalent à l'exécution d'un programme donné, ou en état d'arrêt (Halt) qui peut être par exemple la conséquence d'une erreur bus (double), ou finalement dans un état de traitement d'une exception.

Du point de vue définition, l'exception est une généralisation de la notion classique d'interruption pour inclure en plus des requêtes matérielles, les requêtes de nature logicielle.

Avant de détailler le mécanisme d'interruption et son implémentation dans le cas du MC68000, ce chapitre introduira tout d'abord la notion de mémoire pile, ainsi que le processus d'appel de sous programme.

Des exemples d'applications seront exposés et traités dans le chapitre consacré aux entrées-sorties numériques.

4.2 La notion de pile

Une pile est une zone mémoire de type LIFO (Last In First Out) pointée avec un registre d'adresse (appelé pointeur de pile) dont le contenu est l'adresse de la dernière donnée introduite.

Une opération d'écriture est précédée par une décrémentation du pointeur pour assurer un empilement dans la mémoire.

En contre partie, une opération de lecture correspond à une incrémentation du pointeur assurant ainsi un dépilement de la pile.

Dans le cas du MC68000 le pointeur de pile est le registre A7.

4.2.1 L'appel et le retour d'un sous programme

Deux instructions peuvent être utilisées pour un appel de sous programme :

$$\begin{cases} BSR\ etiquette & ;identique\ à\ BRA \\ JSR\ <AE> & ;identique\ à\ JMP \end{cases}$$

Dans les deux cas un branchement vers un sous-programme, implique la séquence d'opération suivantes :

$$\begin{cases} A7 \leftarrow [A7]\text{-}4 & ;\text{décrementation du pointeur pile} \\ A7 \leftarrow PC & ;\text{sauvegarde de l'adresse de retour} \\ PC \leftarrow \text{adresse du sous-prog} \end{cases}$$

Le A7 est décrémenté de 4 octets pour l'empilement de l'adresse de retour dans la pile, et l'exécution du sous programme commence après le chargement de son adresse de début dans le registre PC.

L'instruction RTS à la fin d'un sous-programme permet une récupération de l'adresse de retour, ainsi qu'une incrémentation du pointeur A7 :

$$\begin{cases} PC \leftarrow (A7) & ;\text{réstauration de l'adresse de retour} \\ A7 \leftarrow A7 + 4 & ;\text{incrementation du pointeur pile} \end{cases}$$

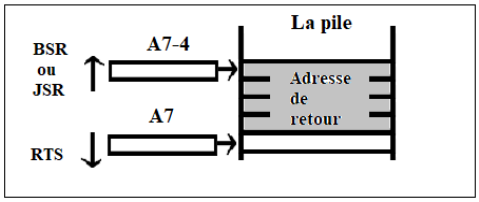


Fig4.1 :La pile et le pointeur A7

Exemple

Un sous-programme qui calcule le carré d'un entier avec un passage d'arguments et du résultat dans des registres de donnée (D0, D1)

.

Adresse		
$1000		**ORG** $1000
$1004		LEA $5008,A7
$1008		MOVE.W #03, D0
$100C		**BSR carre**
.....		ADD.L #1,D1
$1012		.....
$1014	**carre**	MOVE D0,D1
$1016		MULU D0,D1
.....		**RTS**
		.....
		END

Apres l'instruction BSR :

- A7←$5008-4=$5004 (décrémentation du pointeur pile)

- L'adresse de retour (PC+4=$100C), qui représente l'adresse de l'instruction suivante est sauvegardée dans la pile

- Le PC est chargé par l'adresse de début du sous-programme (PC←$1012)

Apres l'instruction RTS :

- L'adresse que pointe le A7 est transférée au PC←$100C (l'adresse de retour)

- A7←$5004+4=$5008 (incrémentation du pointeur pile)

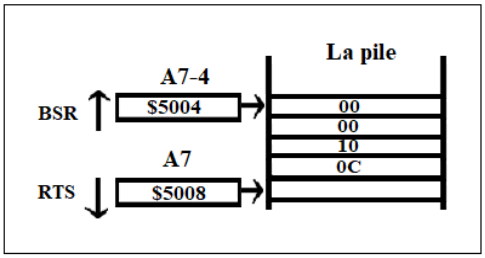


Fig4.2 :Contenu de la pile et du pointeur A7

Passage d'arguments

Le passage d'arguments entre un programme et un sous-programme peut etre effectuée soit par valeurs ou adresses, en utilisant soit des registres, ou une partie de la mémoire pile.

Exemple

Le sous-programme dans l'exemple suivant calcule l'expression $S + x^2$-5, où la pile est utilisée pour une transmission de l'argument 'x' par valeur, et un retour du résultat 'S'par adresse.

```
        x     EQU $ 05
        adrS EQU $2000
        ORG $1000
        LEA $5020,A7          ; pointeur A7←$5020
        MOVE.W #x, -(A7)      ; valeur dans la pile, et A7←$501E
        PEA adrS              ; adresse dans la pile, et A7←$501A
        BSR CALC              ; A7←$5016
        ADDA #6,A7            ; restauration du pointeur A7←$5020
        ......
CALC    MOVE +8(A7),D0        ; D0←x=05
        MOVEA +4(A7),A5       ; A5←adrS= $2000
        MULS D0,D0            ; D0←x.x=25
        SUBI #5,D0;           ; D0←D0-5=20
        MOVE.L D0,(A5)        ; résultat affecté à l'adresse $2000
        RTS
        ......
        END
```

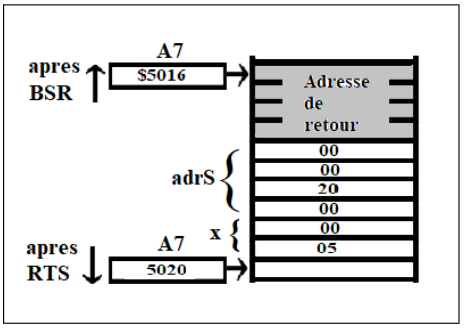


Fig4.3 :Contenu de la pile et du pointeur A7

Remarque

L'instruction PEA (push effective address) permet une sauvegarde directe d'une adresse dans la pile, avec une pré-décrémentation du pointeur A7. elle remplace pratiquement l'instruction :

MOVE.l #adrS,-(A7)

Sauvegarde de registres dans la pile

Lors de l'appel d'un sous programme une sauvegarde temporaire du contenu des registres (utilisés par ce S/P) dans la mémoire pile est possible.

L'instruction MOVEM est généralement utilisée vu quelle permet un transfert multiple de registres avec différent mode d'adressage :

$$MOVEM.\begin{pmatrix} W \\ L \end{pmatrix} \ liste\ de\ \ registres, AE$$

$$MOVEM.\begin{pmatrix} W \\ L \end{pmatrix} \ < AE >, liste\ de\ \ registres$$

Le mode indirect pré-décrémenté :

Ce mode est utilisé pour la sauvegarde multiple des registres dans la memoire.

par défaut l'ordre de sauvegarde des registre commence par les registres d'adresse puis ceux de donnée (A7, A6, A5,....,A0, D7, D6,....,D0).

Le mode indirect post-incrémenté :

Ce mode est limité aux transferts de la mémoire vers les registres, et l'ordre de restitution est :

(D0, D1,....,D7, A0, A1, A3,....,A7,).

Exemple

$MOVEM.W \quad D0, D4, /A1\text{-}A3,\text{-}(A0)$, Les registres sauvegardés en mémoire (pointée par A0) : A1,A2,A3,D0,D4

$MOVEM.W \quad (A5)+, A2\text{-}A4/D0, D7$, Les registres restitués de la mémoire (pointée par A5) : A2,A3,A4,D0,D7

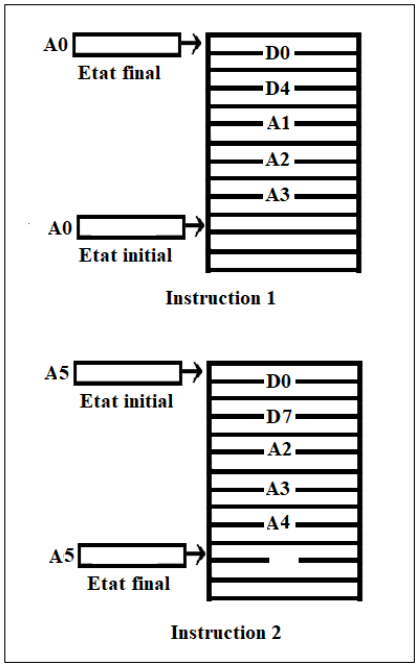


Fig4.4 :Etat initial et final de la pile et du pointeur A7

Dans le cas des autres modes d'adressage, le transfert se fait à partir de l'adresse spécifiée vers les adresses mémoire croissantes.

Exemple

On considère l'exemple précédent (calcul de l'expression) où la pile est utilisée pour :

- Une transmission de l'argument 'x' par valeur.

- Un retour du résultat 'S'par adresse.
- Une sauvegarde, et une restitution du contenu initial du registre de donnée (D0).

```
        x     EQU $ 05
        adrS EQU $2000
         ORG $1000
         LEA $5020,A7          ; pointeur A7←$5020
         MOVE.W  #x, -(A7)     ; valeur  dans la pile, et A7←$501E
         PEA adrS              ; adresse dans la pile, et A7←$501A
         BSR  CALC             ; A7←$5016
         ADDA #6,A7            ; restauration du pointeur A7←$5020
         ......
CALC     MOVEM.L D0,-(A7)      ; sauvegarde de D0 dans la pile
         MOVE +12(A7),D0       ; D0←x=05
         MOVEA +4(A7),A5       ; A5←adrS= $2000
         MULS D0,D0            ; D0←x.x=25
         SUBI #5,D0;           ; D0←D0-5=20
         MOVE.L D0,(A5)        ; résultat affecté à l'adresse $2000
         MOVEM.L +(A7)+,D0     ; restitution de D0
         RTS
         ......
         END
```

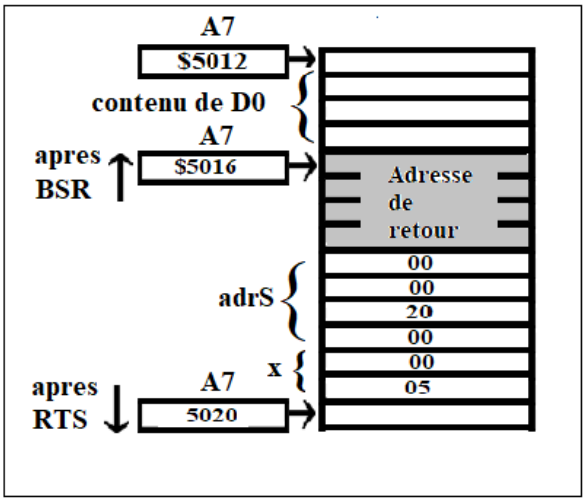


Fig4.5 :Contenu de la pile et du pointeur A7

4.3 La notion d'interruption

L'interruption est une procédure qui répond à un signal matériel externe (IRQ : Interrupt Request), et qui permet d'interrompre le programme en cours au profit d'un autre.

En d'autres termes, une interruption provoque le déroutement du processeur de la tache en cours vers une autre qui peut être une tache d'urgence.

Dans un système à microprocesseur, l'implémentation matérielle d'interruption est équivalente à la mise en place d'une communication par poigné de main

(handshaking). Car si le périphérique demandeur d'interruption active la ligne de demande d'interruption, le microprocesseur doit signaler que la requête est acceptée en activant une ligne de reconnaissance de la demande.

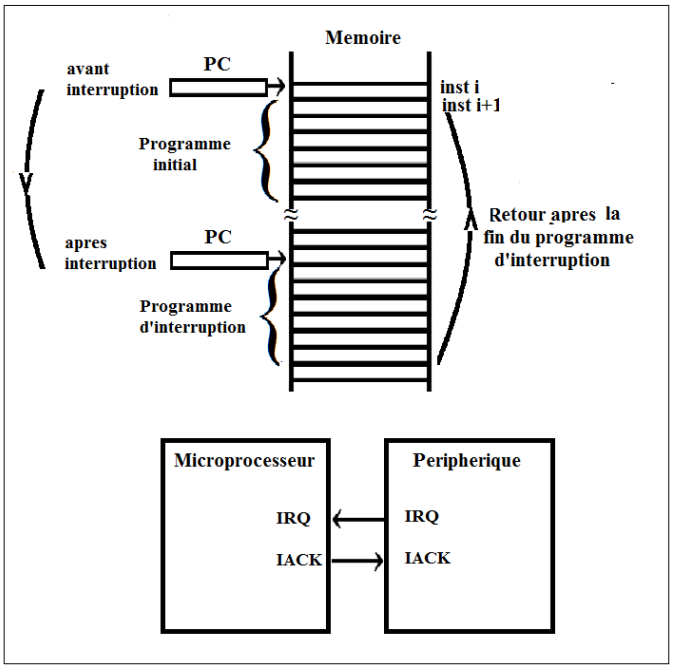


Fig4..6 : Schéma simplifié du mécanisme de base d'une interruption

Si l'interruption est acceptée, le microprocesseur doit impérativement :

- Terminer l'instruction en cours
- Sauvegarder l'adresse de retour (l'instruction suivante) ainsi que l'état des différents indicateurs (flags) afin de pouvoir continuer correctement la tache une fois le programme relatif à l'interruption terminé.
- Déterminer et charger dans le registre PC (program counter) l'adresse du sous programme d'interruption.

Ainsi définie la gestion des interruptions pose un ensemble de problèmes, dont le premier est la détermination de l'adresse du sous programme d'interruption, et le second est relatif à la gestion de la priorité dans le cas où plusieurs sources d'interruption sont possibles.

4.3.1 Les interruptions du MC68000

Dans le cas du MC68000, et tel qu'il a été décrit dans le premier chapitre, la ligne de requête d'interruption est remplacé par trois pins ($\overline{IPL_0}$-$\overline{IPL_2}$)

qui servent en outre à la gestion matérielle de la priorité entre les différents demandeurs.

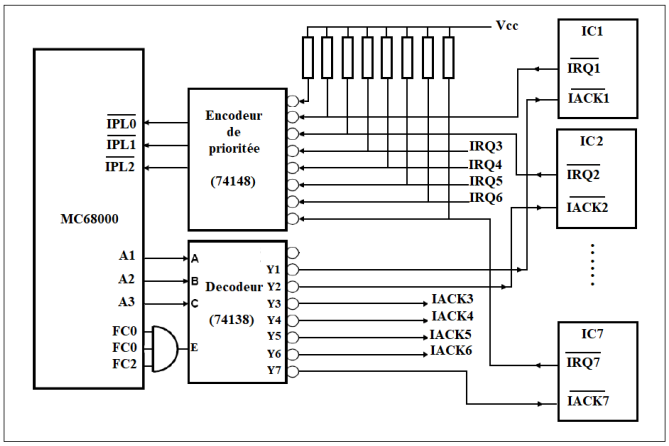


Fig4.7 **: Exemple d'un circuit de gestion des interruptions du MC68000**

Pratiquement dans un système à base du MC68000 une demande d'interruption suit un ensemble d'étapes :

Etape 1 :

Le périphérique demandeur d'interruption signale sa requête par le dépôt de son niveau de priorité (complément à 1) sur les trois pins ($\overline{IPL_0}$-$\overline{IPL_2}$). Par conséquent, un circuit logique encodeur de priorité est suffisant pour assurer à la fois la prise en compte de la demande d'interruption ainsi que la gestion matérielle de la priorité entre les différents circuits périphériques.

Etape 2 :

La requête d'interruption est prise en considération uniquement si le niveau de priorité déposé sur ($\overline{IPL_0}$-$\overline{IPL_2}$) est supérieur a celui de la tache en cours (les bit du masque d'interruption (I0-I1-I2) du registre d'état SR).

Si la condition est vérifiée, le microprocesseur dépose le niveau de priorité sur la ligne d'adresse A1-A2-A3), et passe au mode superviseur (bit S=1, et FC_0-FC_1-FC_2=111). La combinaison de ces signaux permet la génération des signaux de validation des requêtes d'interruption (IACK : Interrupt acknowledge).

La valeur du masque d'interruption (I_0-I_1-I_2) est modifiée suivant le niveau de priorité de la nouvelle tache validée.

Etape 3 :

Sauvegarde du contenu du registre PC (32 bits), et du registre d'état SR (16 bits) dans la pile.

Etape 4 :

Détermination de l'adresse du sous programme d'interruption relatif à la requête validée.

L'ensemble des adresses de sous-programmes d'interruption sont stockés en mémoire dans une table nommé la table des vecteurs d'exceptions.

L'instruction RTE à la fin du sous-programme d'interruption permet une récupération des valeurs du PC, et du SR stockées dans la pile.

Cette restitution permet une continuité normale de l'exécution du programme qui était déjà en cours avant l'interruption.

4.3.2 La table des vecteurs d'exceptions

Dans un système à base du MC68000 la table des vecteurs d'exceptions est une zone mémoire ou sont stockées les adresses de début des différents sous-programmes d'interruption.

Cette zone mémoire occupe le premier K octets de l'espace mémoire superviseur.

Pour déterminer l'adresse de la tache à exécuter, le microprocesseur a besoin uniquement du numéro de vecteur pour identifier l'adresse de début du sous programme.

On distingue différents types de sources d'interruptions :

- **Les interruptions système** : telles que le RESET, l'erreur bus, ou la violation de privilège.

- **Les interruptions logicielles** (TRAP #n) : dont les numéros de vecteur varient de 32 à 47.

- **Les interruptions auto-vectorisées** :

Dans ce cas le microprocesseur calcule le numéro de vecteur à partir du niveau de priorité du périphérique (numéro de vecteur= niveau de priorité+24), et par conséquent ce type d'interruption occupe les numéros de vecteur de 25 à 31.

Cette approche est adoptée par Motorola pour permettre un interfaçage facile du MC68000 avec les circuits périphériques (auto-vectorisés) conçus initialement pour les microprocesseurs 8 bits du type : 68xx.

Une interruption est considérée auto-vectorisée si le périphérique accompagne sa requête par activation du signal $\overline{VPA}$.

Vector Numbers	Address Dec	Address Hex	Space[2]	Assignment
0	0	000	SP	Reset: Initial SSP[3]
	4	004	SP	Reset: Initial PC[3]
2	8	008	SD	Bus error
3	12	00C	SD	Address error
4	16	010	SD	Illegal instruction
5	20	014	SD	Zero divide
6	24	018	SD	CHK instruction
7	28	01C	SD	TRAPV instruction
8	32	020	SD	Privilege violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 emulator
11	44	02C	SD	Line 1111 emulator
12[1]	48	030	SD	(Unassigned, reserved)
13[1]	52	034	SD	(Unassigned, reserved)
14	56	038	SD	Format error[4]
15	60	03C	SD	Uninitialized interrupt vector
16–23[1]	64	040	SD	(Unassigned, reserved)
	92	05C		—
24	96	060	SD	Spurious interrupt[5]
25	100	064	SD	Level 1 interrupt autovector
26	104	068	SD	Level 2 interrupt autovector
27	108	06C	SD	Level 3 interrupt autovector
28	112	070	SD	Level 4 interrupt autovector
29	116	074	SD	Level 5 interrupt autovector
30	120	078	SD	Level 6 interrupt autovector
31	124	07C	SD	Level 7 interrupt autovector
32–47	128	080	SD	TRAP instruction vectors[6]
	188	0BC		—
48–63[1]	192	0C0	SD	(Unassigned, reserved)
	255	0FF		—
64–255	256	100	SD	User interrupt vectors
	1020	3FC		—

[1]Vector numbers 12, 13, 16 through 23, and 49 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.
[2]SP denotes supervisor program space, and SD denotes supervisor data space.
[3]Reset vector (0) requires four words, unlike the other vectors, which only require two words, and is located in the supervisor program space.
[4]MC68010 only. This vector is unassigned, reserved on the MC68000 and MC68008.
[5]The spurious interrupt vector is taken when there is a bus error indication during interrupt processing.
[6]TRAP #n uses vector number 32 + n.

Tab4.1 :La table du vecteur d'exceptions du MC68000

- **Les interruptions utilisateur** :

Si lors de la requête le signal $\overline{VPA}$ n'est pas activé, alors l'interruption est non-auto-vectorisée, et par conséquent le périphérique dépose un numéro de vecteur sur le bus de donnée (D0-D7) et active le signal $\overline{DTACK}$.

Les interruptions utilisateurs occupent les numéros de vecteur de 64 à 255.

Multiplié par 4, un numéro de vecteur donne l'adresse où est stockée l'adresse de début du sous-programme d'interruption.

- Si une interruption de niveau trois 3 est reçue avec une activation du signal $\overline{VPA}$, alors l'interruption est auto-vectorisée (numéro de vecteur=3+24=27), et par conséquent le PC reçoit les 32 bits à partir de l'adresse mémoire (27x4=108=$06C).
- Si l'interruption est de niveau trois 3 (sans l'activation de $\overline{VPA}$) avec un dépôt de la valeur 65 sur les lignes de donnée D0-D7, alors l'interruption est

vectorisée (numéro de vecteur =65), et par conséquent le PC est chargé par le contenu mémoire à partir de l'adresse (65x4=260=$104).

- Si au cours de son fonctionnement normal, le microprocesseur exécute l'instruction TRAP#02, alors une interruption logicielle se produit avec le branchement vers une adresse (le nouveau contenu du PC) qui est la valeur stockée à partir de $084

4.3.3 Le traitement du RESET

Le RESET (la réinitialisation) est l'exception du niveau le plus élevée (7), est par conséquent elle présente une interruption non-masquable.

Pratiquement le reset du MC68000 est réalisé par une mise à zéro simultanée des pins ($\overline{RESET}$, $\overline{HLT}$) pendant au moins 100ms (le processus est déjà expliqué dans le chapitre 2 -Fig 2.7).

En plus du passage au mode superviseur (bit S=1), cette interruption particulière provoque :

- Le chargement du pointeur pile A7 avec les 32 bits de l'adresse mémoire $0000 (le numéro de vecteur 0).

- Le chargement du compteur programme PC avec les 32 bits de l'adresse mémoire $0004 (le numéro de vecteur 1), qui présente l'adresse de début du programme.

Remarque

La pin $RESET$ du MC68000 est une ligne bidirectionnelle (à collecteur ouvert), et l'instruction 'RESET ' provoque sa remise à zéro pendant 124 periodes d'horloge, réinitialisant ainsi les circuits périphériques sans que le microprocesseur ne soit affecté.

4.4 Exercices

Exercice 1 :

Ecrire un programme qui calcule l'expression :

$S=A^2+B^2+C^2$

Sachant que A, B, et C sont trois nombres signés codés sur 16 bits et rangés en mémoire respectivement aux adresses adra, adrb, et adrc

Le résultat S est sauvegardé dans le registre D5

Exercice 2 :

Développer un programme qui calcule l'expression :

$S=A^3+B^3+C^3$

Sachant que A, B, et C sont trois nombres signés codés sur 8 bits et rangés en mémoire respectivement aux adresses adra, adrb, et adrc

Le résultat S est sauvegardé à l'adresse adrs

Chapitre 5

Le décodage d'adresses

5.1 Introduction

Dans un système à microprocesseurs, au cours d'un cycle de lecture ou d'écriture, il est nécessaire d'assurer l'activation du circuit concerné par le transfert, tout en désactivant le reste des circuits périphériques connectés à ce microprocesseur.

Un circuit logique exploitant une partie du bus d'adresse et des lignes de contrôle peut être conçu de différente façon pour assurer le décodage d'adresse dans une carte à microprocesseur.

Dans le cas du MC68000, si le périphérique est un 8 bits, le choix de la partie du bus de donnée du microprocesseur à connecté avec le circuit impose implicitement une adresse impaire (si D0-D7 est utilisée) ou une adresse paire (si D8-D15 est connectée)

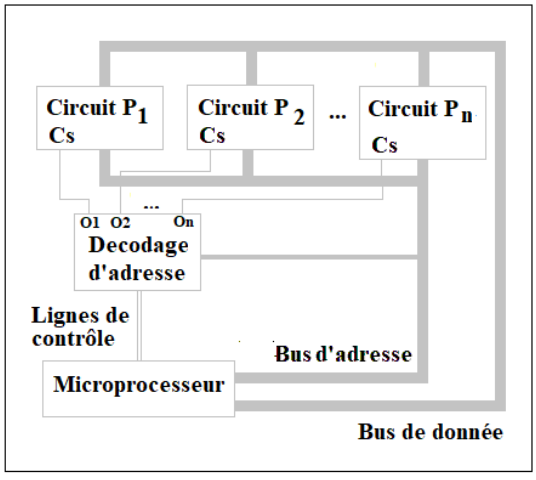


Fig 5.1 :Schéma de principe du décodage d'adresses dans un système à microprocesseur

Il est important de noter, que la conception d'un circuit de décodage d'adresse doit respecter outre, les chronogrammes de lecture/écriture des constructeurs, certaines contraintes propres à chaque microprocesseur.

Par exemple dans le cas du MC68000, le premier K octets de l'espace mémoire adressable est réservé à la table du vecteur d'exceptions. Par conséquent cette zone (ou une partie d'elle) doit être impérativement une mémoire non-volatile. Une telle contrainte n'est pas respectée dans l'exemple introductif suivant.

D'autre part, les circuits asynchrones -pouvant provoquer une interruption- doivent déposer un numéro de vecteur sur le bus de donnée D0-D7, ce qui implique que l'adresse est obligatoirement impaire.

5.2 Exemple introductif

Dans ce cas une seule ligne d'adresse A23 est utilisé pour différencier entre les deux circuits périphériques C1 (un registre 8 bits) et C2 (un registre 16 bits).

C1 est connecté à la partie de poids faibles du bus de donnée (D0-D7), est par conséquent son adresse est impaire (A0=1), d'ou la nécessité de conditionner l'activation du circuit par le passage à zéro du signal $\overline{LDS}$.

C2 est connecté au 16 bits du bus de donnée (D0-D15), est son adresse est par conséquent paire (A0=0).

D'après les chronogrammes de base de lecture/écriture (chapitre 2), l'activation d'un circuit donné doit être conditionnée par le passage à zero de la ligne $\overline{AS}$, qui indique qu'il y a effectivement une adresse valide sur le bus.

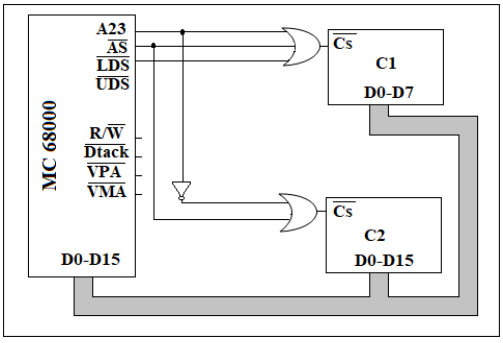


Fig 5.2 : Exemple simple d'un schéma de décodage (utilisation d'une seule ligne d'dresse)

D'après le schéma proposé les adresses possibles de C1 et C2 sont :

C1 :

A23 A22 A21... A1 A0 $=000....00)_2=(000000)_{16}$

$=(000....10)_2=(000002)_{16}$

$=(000...100)_2=(000004)_{16}$

$=(000...110)_2=(000006)_{16}$

$=(000..1000)_2=(000008)_{16}$
$=(0xx.....x0)_2=(xxxxx0)_{16}$
$=(011.....10)_2=(7FFFFE)_{16}$

C2 :

A23 A22 A21... A1 A0 $=(100....00)_2=(800000)_{16}$
$=(100....10)_2=(800002)_{16}$
$=(100...100)_2=(800004)_{16}$
$=100...110)_2=(800006)_{16}$
$=100..1000)_2=(800008)_{16}$
$=(1xx.....x0)_2$
$=111.....10)_2=(FFFFFE)_{16}$

L'utilisation d'une seule ligne d'adresse A23 a permis une division en deux de l'espace mémoire adressable par le microprocesseur ($2^{24}/2=2^{23}$=8 M).

Les adresses possibles pour le circuit 8 bits C1, sont uniquement les adresses impaires de la première zone vu qu'il est connecté aux lignes de données D0-D7, et conditionné par la ligne $\overline{LDS}$.

Par conséquent C1 possède ($2^{23}/2=2^{22}$=4 M) adresses possibles.

Remarque :

Le circuit complet d'interfaçage d'un circuit périphérique à un microprocesseur doit inclure aussi la ligne indiquant le sens de transfert R/$\overline{W}$, si les deux opérations (lecture/écriture) sont possibles.

Dans le cas du MC68000, une génération d'un signal accusé de réception est aussi nécessaire telle qu'il est imposé par le constructeur dans les chronogrammes de transferts.

Si le périphérique est asynchrone le transfert par poignée de main est complété par la génération du signal $\overline{DTACK}$, ou par le couplé $\overline{VPA}/\overline{VMA}$ dans le cas d'un circuit synchrone.

5.3 Décodage avec un circuit décodeur

Si N lignes d'adresses sont utilisées pour la répartition de l'espace mémoire adressable du MC68000 (16 Mo) on obtient des zones dont la taille de chacune est égale à $16/2^N$ (M octets) .

Dans l'exemple suivant un décodeur 1/8 permet la répartition de l'espace adressable en 8 zones de 2Mo chacune ($2^{24}/2^3$).

Les circuits périphériques sont considérés asynchrones, et les détails du circuit de génération du signal $\overline{DTACK}$ ne sont donnés.

Dans le schéma proposé, il est supposé que les circuits mémoires possèdent deux pins de sélection ($\overline{Cs}0$-$\overline{Cs}1$), et une seule pin ($\overline{Cs}$) pour IC1 et IC2

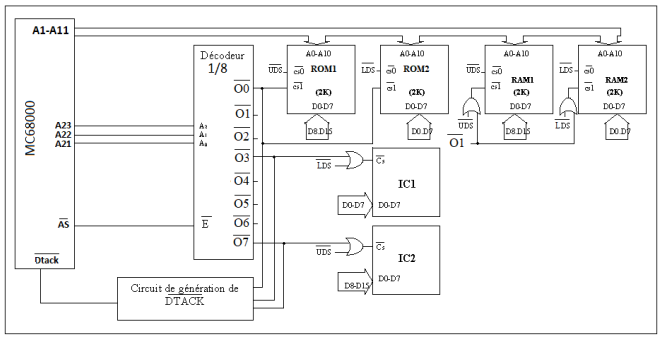


Fig 5.3 : Exemple de décodage d'adresses avec un circuit décodeur 1/8

La mémoire morte :

La mémoire morte dans cet exemple occupe les premiers K octets de l'espace mémoire adressable.

Par conséquent elle est supposée contenir les adresses des tables du vecteur d'exceptions, ainsi que le code machine du programme.

De ce fait, l'accès à la ROM s'effectue en mode 16 bits, et au cours de chaque cycle de lecture les deux circuits sont activés simultanément.

- Les circuits ROM (ROM1+ ROM 2), dont la taille totale est 4Ko, occupent la première zone mémoire (la sortie $\overline{O0}$, équivalente à (A23 A22 A21=000).

Cette zone délimite l'intervalle : $00 00 00-$1F FF FF

- La plage d'adresses de base des deux circuits ROM est :$00 00 00-00 0FFF.

- Les 9 lignes d'adresses libres (A12-A20) fixent le nombre de reflets ou d'images de cette zone mémoire ROM : 2^9 =512 images.

Circuit	Adresse en Bin A23 A22 A21 A20 ..... A12 A11 A10 ...A1 A0	Adresses (Hex)
ROM1	**0** **0** **0** 0 ..... 0 0 0 ... 0 0	00 00 00
ROM2	**0** **0** **0** 0 ..... 0 0 0 ... 0 1	00 00 01
ROM1	**0** **0** **0** 0 ..... 0 0 0 ... 1 0	00 00 02
ROM2	**0** **0** **0** 0 ..... 0 0 0 ... 1 1	00 00 03
...	...	...
ROM1	**0** **0** **0** 0 ..... 0 1 ... 1 ... 0 0	00 0FFC
ROM2	**0** **0** **0** 0 ..... 0 1 ... 1 ... 0 1	00 0FFD
ROM1	**0** **0** **0** 0 ..... 0 1 ... 1 ... 1 0	00 0FFE
ROM2	**0** **0** **0** 0 ..... 0 1 ... 1 ... 1 1	00 0FFF

Tab 5.1 : La première image d'adresses occupée par les circuits ROM

La mémoire vive :

Dans le schéma proposé le 'ET' logique entre le signal de sélection issu du décodeur, et le signal $\overline{UDS}/\overline{LDS}$, offre à la fois une possibilité d'écriture/ lecture en mode 8 bits (une seule RAM) et en mode 16 bits (RAM1+RAM2).

- Les circuits RAM (RAM1+ RAM 2) présentent aussi 4Ko de mémoire, et occupent la deuxième zone mémoire (la sortie $\overline{O1}$, équivalente à A23 A22 A21=001).

Cette zone délimite l'intervalle : $20\ 00\ 00-$3F\ FF\ FF

- La plage d'adresses de base est :$20 00 00-20 0FFF

- Le nombre de reflets ou d'images de cette zone mémoire est aussi : 2^9 =512 images.

Circuit	Adresses en (Bin) A23 A22 A21 A20 ..... A12 A11 A10 ... A1 A0	Adresses (Hex)
RAM1	**0 0 1** 0 ..... 0 0 0 ... 0 0	20 00 00
RAM2	**0 0 1** 0 ..... 0 0 0 ... 0 1	20 00 01
RAM1	**0 0 1** 0 ..... 0 0 0 ... 1 0	20 00 02
RAM2	**0 0 1** 0 ..... 0 0 0 ... 1 1	20 00 03
...	...	...
RAM1	**0 0 1** 0 ..... 0 1 ... 1 ... 0 0	20 0FFC
RAM2	**0 0 1** 0 ..... 0 1 ... 1 ... 0 1	20 0FFD
RAM1	**0 0 1** 0 ..... 0 1 ... 1 ... 1 0	20 0FFE
RAM2	**0 0 1** 0 ..... 0 1 ... 1 ... 1 1	20 0FFF

Tab 5.2 : La première image d'adresses occupée par les circuits RAM

Le circuit IC1 :

- IC1 occupe la quatrième zone mémoire (la sortie $\overline{O3}$, équivalente à A23 A22 A21=011).

Cette zone délimite l'intervalle : $60\ 00\ 00-$7F\ FF\ FF

Ce circuit ne présente qu'une seule case mémoire, et puisqu'il est connecté à la partie basse du bus de donnée (D0-D7), il n'aura comme adresses possibles que les impaires de la zone occupée.

- Les adresses possibles de IC1 : $\begin{cases} \$600001 \text{ l'adresse debase} \\ \$600003 \\ ... \\ \$7FFFFF \\ \$7FFFFD \end{cases}$

- Le nombre de reflets d'adresses de ce circuit est 2^{21} /2=1M images (adresses impaires).

Le circuit IC2 :

- IC2 occupe la dernière zone mémoire (la sortie $\overline{O7}$, équivalente à A23 A22 A21=111).

Cette zone délimite l'intervalle : $E0 00 00-$FF FF FF

Ce circuit présente aussi une seule case mémoire, connectée à la partie supérieur du bus de données (D8-D15), et par conséquent n'aura comme adresses possibles que les paires de la zone occupée.

- Les adresses possibles de IC2 : $\begin{cases} \$E00000 & \text{l'adresse debase} \\ \$E00002 & \\ ... & \\ \$FFFFFC & \\ \$FFFFFE & \end{cases}$

- Le nombre de reflets d'adresses de ce circuit est 2^{21} /2=1M images (adresses paires).

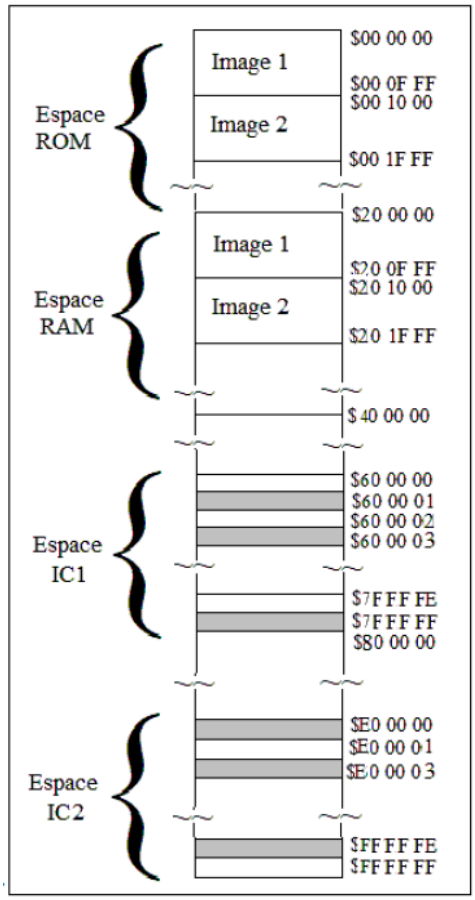


Fig 5.4 : Répartition de l'espace mémoire résultant du schéma de la figure 4.3

5.4 La génération du signal $\overline{DTACK}$ /($\overline{VPA}$-$\overline{VMA}$)

Les circuits périphériques conçus par Motorola pour le MC68000 (et ses successeurs) tels que le MFP68901 sont des circuits asynchrones possédant une sortie $\overline{DTACK}$ permettant un interfaçage direct avec le microprocesseur.

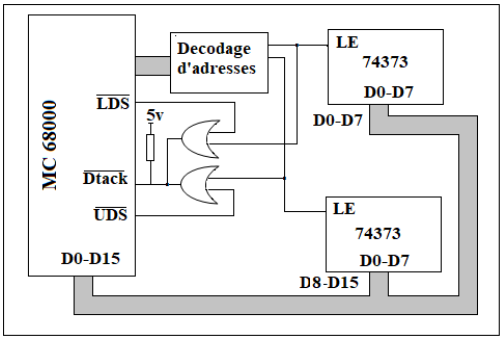
L'exploitation des signaux générés par le microprocesseur au cours des cycles de lecture/écriture ($\overline{AS}$,$\overline{UDS}$,$\overline{LDS}$...), ainsi que les signaux du périphérique lui même, permet pratiquement un interfaçage adéquat même avec des circuits réalisés par d'autres constructeurs.

Dans la figure 5.5, deux circuits logiques 8 bits (74373) sont respectivement connectés à une partie du bus de données du MC68000.

La génération du signal $\overline{DTACK}$ résulte d'un 'ou' logique entre $\overline{UDS}$ (ou $\overline{LDS}$) et le signal de sélection du boitier issu du circuit de décodage d'adresses.

La solution proposée, exploite d'une part le fait que le comportement des signaux $\overline{UDS}$/ $\overline{LDS}$) est relativement similaire à celui du $\overline{DTACK}$, et d'autre part le fait que le temps de réponse des circuits 74373 est assez rapide.

D'après les chronogrammes de transferts asynchrones (présentés au chapitre 2), le signal $\overline{UDS}$ (et/ou $\overline{LDS}$) passe à zéro automatiquement après le dépôt d'une adresse valide sur le bus ($\overline{AS}$=0), pour revenir à l'état haut en fin de cycle.



Fign 5.5 : Exemple de génération de $\overline{\text{DTACK}}$ pour deu xcircuits buffers 74373

Dans le cas des circuits périphériques synchrones, le problème d'interfaçage est solutionné, vu que les lignes $\overline{VPA}$/$\overline{VMA}$ du MC68000 le rendent compatible à ce type de circuits conçus initialement pour les microprocesseurs 8 bits (68xx).

La figure 5.6 illustre l'exemple du circuit PIA6821, qui possède trois pins de sélection CS0, et CS1, et $\overline{CS2}$.

Si au cours d'un cycle donné, le circuit est sollicité, le 'ou' logique entre $\overline{AS}$ et le signal de sélection du boitier (issu du circuit de décodage d'adresses) produira le signal $\overline{VPA}$ (voir le chronogramme dans le chapitre 2).

En réponse le microprocesseur générera le signal $\overline{VMA}$, qui représentera (son complément) le deuxième signal de sélection du PIA.

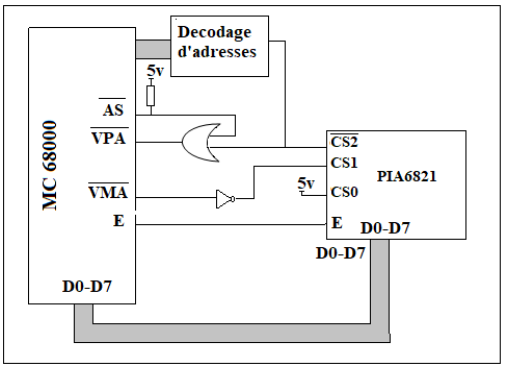


Fig 5.6 : Exemple d'un interfaçage synchrone (PIA6821)

Chapitre 6

Les Interfaces d'entrée-sortie numériques

6.1 Introduction

Ce chapitre expose une description succincte de quelques circuits d'entrée-sortie numérique.

L'objectif principal est de présenter des exemples et des exercices d'applications de ces circuits dans un système à base du MC68000, soit en mode scrutation, ou en mode interruption.

6.2 Le DUART 68681

Le DUART est un circuit d'interface pour communication série, comportant deux ports de transmission/réception asynchrone (UART).

Les deux ports sont indépendemment programmables avec la possibilité d'une transmission/réception simultanée.

Le circuit comporte aussi 6 entrées, et 8 sorties parallèles, pouvant être utilisées dans le cas d'une communication par poignée de main (handshacking).

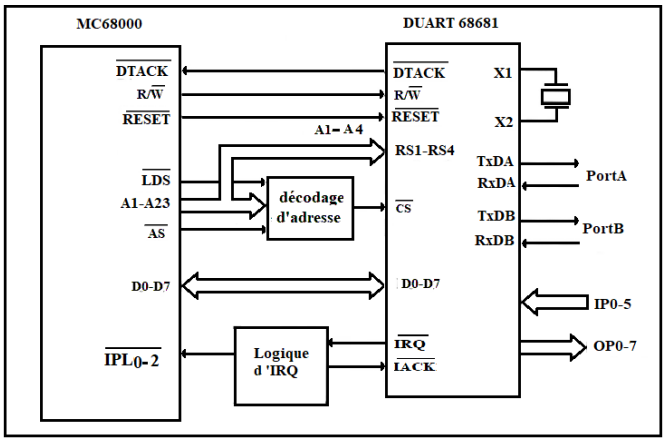


Fig6.1 : Exemple d'interfaçage du DUART68230 au MC68000

6.3 L'interface parallel-Timer PI/T 68230

Ce circuit comporte :

. Trois ports parallèles bidirectionnels (A, B, C), ou les ports A, et B peuvent être utilisés comme un port 16 bits.

. Un timer de 24 bits, exploitable soit en mode décompteur, où en générateur de signal carré.

. Cinq lignes (H1-H5) permettent un transfert par poigné de main (hand-shacking) pour les ports A, B.

. 23 registres internes permettent la configuration des différentes fonctions possibles

. Le circuit est interfaçable directement au MC68000 via une liaison asynchrone, et peut être exploité en mode interruption.

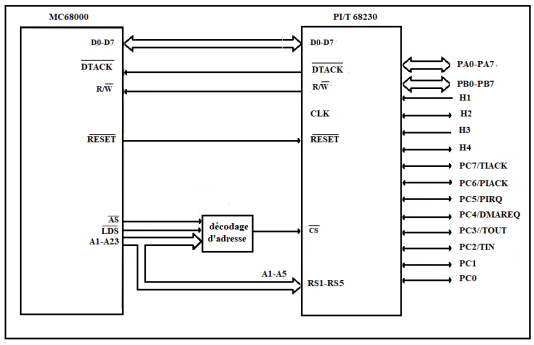


Fig6.2 : Exemple d'interfaçage du PI/T68230 auMC68000

6.4 Exercices

Exercice 1 :

Développer un programme assembleur permettant la réception série (sur le canal RxA d'un DUART) d'une chaîne de caractères se terminant par le caractère @, sous le format : 8 bits + parité paire + 1 bit stop, et avec une fréquence de 1200 bauds.

La chaîne reçue doit être sauvegardée en mémoire à partir de l'adresse $3000.

Exercice 2 :

On dispose d'un terminal connecté au canal A (émetteur / récepteur) d'un DUART 68681.On désire générer une interruption à chaque fois qu'un caractère est validé par le clavier du terminal. Cette interruption a pour tâche : la sauvegarde du caractère à l'adresse $4000, et l'affichage du caractère reçu sur l'écran du terminal (en utilisant le bloc émetteur A)

1) Donner la séquence d'initialisation du DUART sachant que le n° de vecteur est 65

2) Donner le sous programme d'interruption.

NB : format de donnée : 8 bits + parité paire + 1 bit stop.

- fréquence = 9600 bauds.

Exercice 3

Soit le montage de la figure 1 .

- En utilisant une temporisation logicielle de 1s (sous programme 'time').Développer un programme assembleur permettant l'allumage séquentiel des leds connectés au port A .

- Ce même programme doit permettre d'arrêter l'allumage après l'appui du bouton poussoir « BP ».

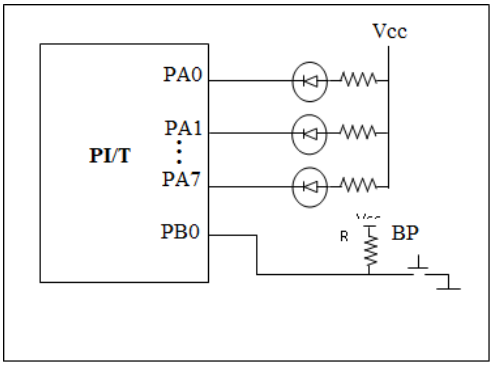


Figure 1

Exercice 4 :

Soit le schéma de la figure 2.

L'ADC utilisé commence une opération de conversion après un niveau bas sur sa ligne Start , et indique la fin de conversion avec un niveau haut sur la ligne EOC .

1- Développer un programme qui permet de prélever séquentiellement 100 mesures du signal d'entrée (Les données prélevées sont stockées à partir de l'adresse adrs).

2- Refaire la question 1, en fixant une fréquence d'échantillonnage de 1khz sachant que l'horloge du microprocesseur est de 4Mhz.

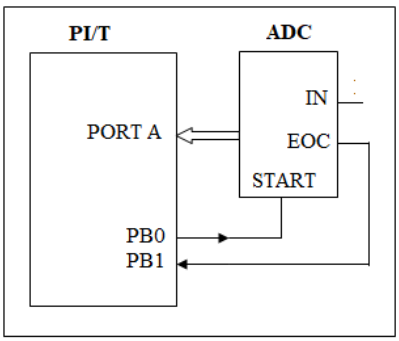


Figure 2

Exercice 5 :

Soit le schéma de la figure 3.

On désire tracer une courbe y=f(x) ou y représente un vecteur de 1 K octets stocké à partir de l'adresse adry, et x un autre vecteur de même dimension stocké à partir de l'adresse adrx .

Développer un programme assembleur qui permet de tracer :

- L'axe des x et des y .
- La courbe y =f(x).

En supposant que les valeurs de x sont croissantes, et que les deux DAC utilisés sont rapides.

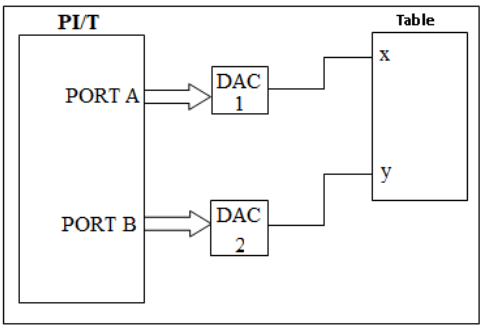


Figure 3

Exercice 6 :

Sachant que la fréquence du signal imposé à la pin TIN est de 50khz, et celle du microprocesseur est de 8Mhz,

-développer un programme permettant la génération d'un signal carré d'une fréquence de (2khz, 30khz, 200khz) au niveau de la pin TOUT.

-développer un autre programme permettant la génération d'un signal carré et son complément au niveau des pins PA0 ,PA1 d'une fréquence de 25khz.

Chapitre 7

Bibliographie

Bibliographie

[1] M68000 8-/16-/32- Bit Microprocessors User's Manual, Motorola inc., 1993.

[2] M.Aumiaux, Microprocesseur 16 bits, édition Masson, 1985.

[3] P.Jaulent, Circuits périphériques de la famille 68000, Eyrolles, 1985.

[4] T.P.Skinner, Assembly language programming for MC68000 family, John Wiley & Sons,1988.

[5] MC68681,Dual Asynchronous Receiver/Transmitter (DUART) datasheet, Motorola inc., 1983.

[6] MC68230 Parallel Interface/Timer (PI/T) Datasheet, Motorola inc., 1985.

Annexe A

Annexe

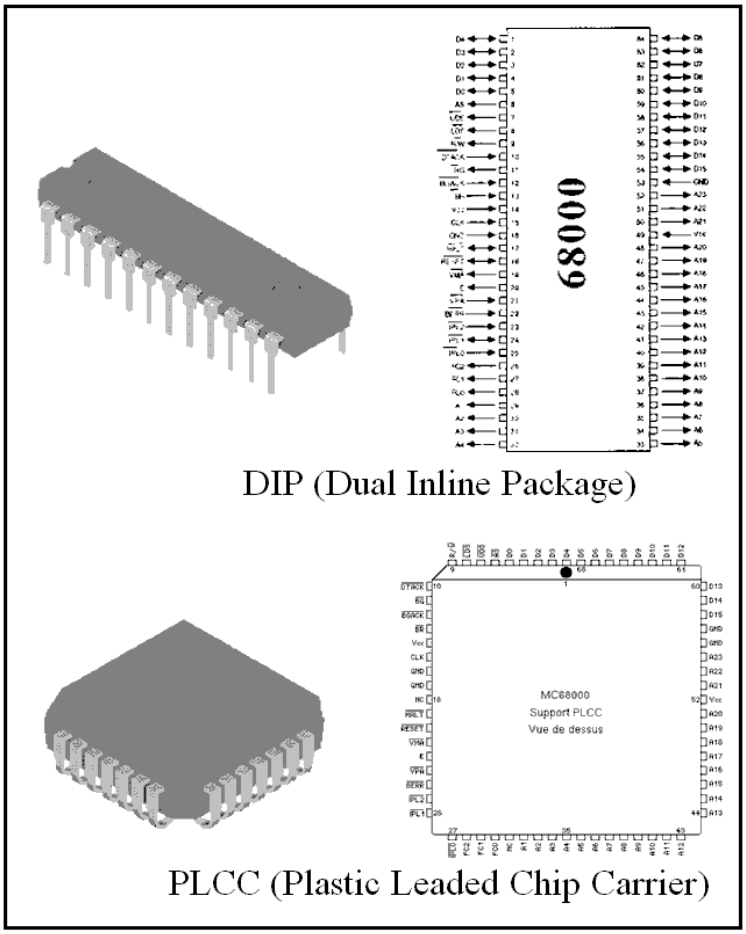
L'ensemble d'informations résumées dans cet annexe est principalement issu des datasheets fournis par le constructeur.

A.1 Code ASCII

Table des codes ASCII étendus OEM et ANSI

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR OEM	CHAR ANSI	DEC	HEX	CHAR OEM	CHAR ANSI	DEC	HEX	CHAR OEM	CHAR ANSI	DEC	HEX	CHAR OEM	CHAR ANSI
0	00		32	20		64	40	@	96	60	`	128	80	Ç	€	160	A0	á		192	C0	└	À	224	E0	α	à
1	01	☺	33	21	!	65	41	A	97	61	a	129	81	ü		161	A1	í	¡	193	C1	┴	Á	225	E1	ß	á
2	02	☻	34	22	"	66	42	B	98	62	b	130	82	é	‚	162	A2	ó	¢	194	C2	┬	Â	226	E2	Γ	â
3	03	♥	35	23	#	67	43	C	99	63	c	131	83	â	ƒ	163	A3	ú	£	195	C3	├	Ã	227	E3	∏	ã
4	04	♦	36	24	$	68	44	D	100	64	d	132	84	ä	„	164	A4	ñ	¤	196	C4	─	Ä	228	E4	∑	ä
5	05	♣	37	25	%	69	45	E	101	65	e	133	85	à	…	165	A5	Ñ	¥	197	C5	┼	Å	229	E5	σ	å
6	06	♠	38	26	&	70	46	F	102	66	f	134	86	å	†	166	A6	ª	¦	198	C6	╞	Æ	230	E6	µ	æ
7	07	•	39	27	'	71	47	G	103	67	g	135	87	ç	‡	167	A7	º	§	199	C7	╟	Ç	231	E7	τ	ç
8	08	◘	40	28	(	72	48	H	104	68	h	136	88	ê	ˆ	168	A8	¿	¨	200	C8	╚	È	232	E8	Φ	è
9	09	○	41	29	)	73	49	I	105	69	i	137	89	ë	‰	169	A9	⌐	©	201	C9	╔	É	233	E9	θ	é
10	0A	◙	42	2A	*	74	4A	J	106	6A	j	138	8A	è	Š	170	AA	¬	ª	202	CA	╩	Ê	234	EA	Ω	ê
11	0B	♂	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	‹	171	AB	½	«	203	CB	╦	Ë	235	EB	δ	ë
12	0C	♀	44	2C	,	76	4C	L	108	6C	l	140	8C	î	Œ	172	AC	¼	¬	204	CC	╠	Ì	236	EC	∞	ì
13	0D	♪	45	2D	-	77	4D	M	109	6D	m	141	8D	ì		173	AD	¡	-	205	CD	═	Í	237	ED	Ø	í
14	0E	♫	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	Ž	174	AE	«	®	206	CE	╬	Î	238	EE	∈	î
15	0F	☼	47	2F	/	79	4F	O	111	6F	o	143	8F	Å		175	AF	»	¯	207	CF	╧	Ï	239	EF	∩	ï
16	10	►	48	30	0	80	50	P	112	70	p	144	90	É		176	B0	░	°	208	D0	╨	Đ	240	F0	≡	ð
17	11	◄	49	31	1	81	51	Q	113	71	q	145	91	æ	'	177	B1	▒	±	209	D1	╤	Ñ	241	F1	±	ñ
18	12	↕	50	32	2	82	52	R	114	72	r	146	92	Æ	'	178	B2	▓	²	210	D2	╥	Ò	242	F2	≥	ò
19	13	‼	51	33	3	83	53	S	115	73	s	147	93	ô	"	179	B3	│	³	211	D3	╙	Ó	243	F3	≤	ó
20	14	¶	52	34	4	84	54	T	116	74	t	148	94	ö	"	180	B4	┤	´	212	D4	╘	Ô	244	F4	⌠	ô
21	15	§	53	35	5	85	55	U	117	75	u	149	95	ò	•	181	B5	╡	µ	213	D5	╒	Õ	245	F5	⌡	õ
22	16	▬	54	36	6	86	56	V	118	76	v	150	96	û	–	182	B6	╢	¶	214	D6	╓	Ö	246	F6	÷	ö
23	17	↨	55	37	7	87	57	W	119	77	w	151	97	ù	—	183	B7	╖	·	215	D7	╫	×	247	F7	≈	÷
24	18	↑	56	38	8	88	58	X	120	78	x	152	98	ÿ	˜	184	B8	╕	¸	216	D8	╪	Ø	248	F8	°	ø
25	19	↓	57	39	9	89	59	Y	121	79	y	153	99	Ö	™	185	B9	╣	¹	217	D9	┘	Ù	249	F9	¨	ù
26	1A	→	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	š	186	BA	║	º	218	DA	┌	Ú	250	FA	·	ú
27	1B	←	59	3B	;	91	5B	[	123	7B	{	155	9B	ø	›	187	BB	╗	»	219	DB	█	Û	251	FB	√	û
28	1C	∟	60	3C	<	92	5C	\	124	7C	|	156	9C	£	œ	188	BC	╝	¼	220	DC	▄	Ü	252	FC	³	ü
29	1D	↔	61	3D	=	93	5D	]	125	7D	}	157	9D	Ø		189	BD	╜	½	221	DD	▌	Ý	253	FD	²	ý
30	1E	▲	62	3E	>	94	5E	^	126	7E	~	158	9E	×	ž	190	BE	╛	¾	222	DE	▐	Þ	254	FE	■	þ
31	1F	▼	63	3F	?	95	5F	_	127	7F	⌂	159	9F	ƒ	Ÿ	191	BF	┐	¿	223	DF	▀	ß	255	FF		ÿ

A.2 Boitiers du MC68000



A.3 Jeu d'instructions du MC68000

Motorola 68000 Instruction Set (1/2)				
Instruction	**Description**	**Assembler Syntax**	**Data Size**	**X N Z V C**
ABCD	Add BCD with extend	Dx,Dy	B--	* U * U *
		-(Ax),-(Ay)		
ADD	ADD binary	Dn,<ea>	BWL	* * * * *
		<ea>,Dn		
ADDA	ADD binary to An	<ea>,An	-WL	- - - - -
ADDI	ADD Immediate	#x,<ea>	BWL	* * * * *
ADDQ	ADD 3-bit immediate	#<1-8>,<ea>	BWL	* * * * *
ADDX	ADD eXtended	Dy,Dx	BWL	* * * * *
		-(Ay),-(Ax)		
AND	Bit-wise AND	<ea>,Dn	BWL	- * * 0 0
		Dn,<ea>		
ANDI	Bit-wise AND with Immediate	#<data>,<ea>	BWL	- * * 0 0
ASL	Arithmetic Shift Left	#<1-8>,Dy	BWL	* * * * *
		Dx,Dy		
		<ea>		
ASR	Arithmetic Shift Right	...	BWL	* * * * *
Bcc	Conditional Branch	Bcc.S <label>	BW-	- - - - -
		Bcc.W <label>		
BCHG	Test a Bit and CHanGe	Dn,<ea>	B-L	- - * - -
		#<data>,<ea>		
BCLR	Test a Bit and CLeaR	...	B-L	- - * - -
BSET	Test a Bit and SET	...	B-L	- - * - -
BSR	Branch to SubRoutine	BSR.S <label>	BW-	- - - - -
		BSR.W <label>		
BTST	Bit TeST	Dn,<ea>	B-L	- - * - -
		#<data>,<ea>		
CHK	CHecK Dn Against Bounds	<ea>,Dn	-W-	- * U U U
CLR	CLeaR	<ea>	BWL	- 0 1 0 0
CMP	CoMPare	<ea>,Dn	BWL	- * * * *
CMPA	CoMPare Address	<ea>,An	-WL	- * * * *
CMPI	CoMPare Immediate	#<data>,<ea>	BWL	- * * * *
CMPM	CoMPare Memory	(Ay)+,(Ax)+	BWL	- * * * *
DBcc	Looping Instruction	DBcc Dn,<label>	-W-	- - - - -
DIVS	DIVide Signed	<ea>,Dn	-W-	- * * * 0
DIVU	DIVide Unsigned	<ea>,Dn	-W-	- * * * 0
EOR	Exclusive OR	Dn,<ea>	BWL	- * * 0 0
EORI	Exclusive OR Immediate	#<data>,<ea>	BWL	- * * 0 0
EXG	Exchange any two registers	Rx,Ry	--L	- - - - -
EXT	Sign EXTend	Dn	-WL	- * * 0 0
ILLEGAL	ILLEGAL-Instruction Exception	ILLEGAL		- - - - -
JMP	JuMP to Affective Address	<ea>		- - - - -
JSR	Jump to SubRoutine	<ea>		- - - - -
LEA	Load Effective Address	<ea>,An	--L	- - - - -
LINK	Allocate Stack Frame	An,#<displacement>		- - - - -
LSL	Logical Shift Left	Dx,Dy	BWL	* * * 0 *
		#<1-8>,Dy		
		<ea>		
LSR	Logical Shift Right	...	BWL	* * * 0 *
MOVE	Between Effective Addresses	<ea>,<ea>	BWL	- * * 0 0

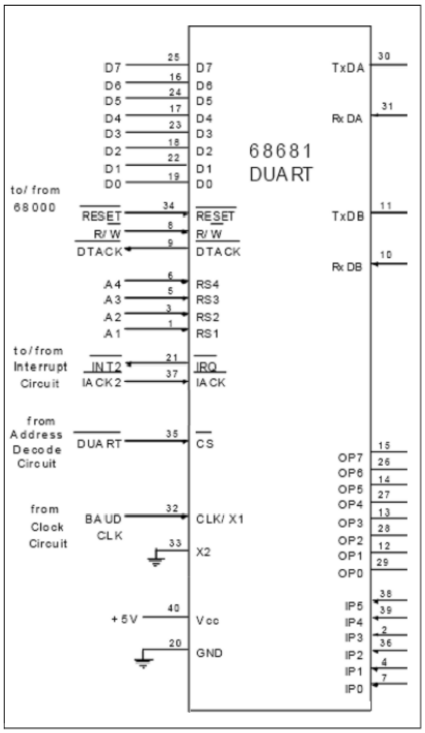
Motorola 68000 Instruction Set (2/2)				
Instruction	**Description**	**Assembler Syntax**	**Data Size**	**X N Z V C**
MOVE	To CCR	<ea>,CCR	-W-	I I I I I
MOVE	To SR	<ea>,SR	-W-	I I I I I
MOVE	From SR	SR,<ea>	-W-	- - - - -
MOVE	USP to/from Address Register	USP,An	--L	- - - - -
		An,USP		
MOVEA	MOVE Address	<ea>,An	-WL	- - - - -
MOVEM	MOVE Multiple	<register list>,<ea>		- - - - -
		-WL		
		<ea>,<register list>		
MOVEP	MOVE Peripheral	Dn,x(An)	-WL	- - - - -
		x(An),Dn		
MOVEQ	MOVE 8-bit immediate	#<-128.+127>,Dn	--L	- * * 0 0
MULS	MULtiply Signed	<ea>,Dn	-W-	- * * 0 0
MULU	MULtiply Unsigned	<ea>,Dn	-W-	- * * 0 0
NBCD	Negate BCD	<ea>	B--	* U * U *
NEG	NEGate	<ea>	BWL	* * * * *
NEGX	NEGate with eXtend	<ea>	BWL	* * * * *
NOP	No OPeration	NOP		- - - - -
NOT	Form one's complement	<ea>	BWL	- * * 0 0
OR	Bit-wise OR	<ea>,Dn	BWL	- * * 0 0
		Dn,<ea>		
ORI	Bit-wise OR with Immediate	#<data>,<ea>	BWL	- * * 0 0
PEA	Push Effective Address	<ea>	--L	- - - - -
RESET	RESET all external devices	RESET		- - - - -
ROL	ROtate Left	#<1-8>,Dy	BWL	- * * 0 *
		Dx,Dy		
		<ea>		
ROR	ROtate Right	...	BWL	- * * 0 *
ROXL	ROtate Left with eXtend	...	BWL	* * * 0 *
ROXR	ROtate Right with eXtend	...	BWL	* * * 0 *
RTE	ReTurn from Exception	RTE		I I I I I
RTR	ReTurn and Restore	RTR		I I I I I
RTS	ReTurn from Subroutine	RTS		- - - - -
SBCD	Subtract BCD with eXtend	Dx,Dy	B--	* U * U *
		-(Ax),-(Ay)		
Scc	Set to -1 if True, 0 if False	<ea>	B--	- - - - -
STOP	Enable & wait for interrupts	#<data>		I I I I I
SUB	SUBtract binary	Dn,<ea>	BWL	* * * * *
		<ea>,Dn		
SUBA	SUBtract binary from An	<ea>,An	-WL	- - - - -
SUBI	SUBtract Immediate	#x,<ea>	BWL	* * * * *
SUBQ	SUBtract 3-bit immediate	#<data>,<ea>	BWL	* * * * *
SUBX	SUBtract eXtended	Dy,Dx	BWL	* * * * *
		-(Ay),-(Ax)		
SWAP	SWAP words of Dn	Dn	-W-	- * * 0 0
TAS	Test & Set MSB & Set N/Z-bits	<ea>	B--	- * * 0 0
TRAP	Execute TRAP Exception	#<vector>		- - - - -
TRAPV	TRAPV Exception if V-bit Set	TRAPV		- - - - -
TST	TeST for negative or zero	<ea>	BWL	- * * 0 0
UNLK	Deallocate Stack Frame	An		- - - - -

Symbol	Meaning
*	Set according to result of operation
-	Not affected
0	Cleared
1	Set
U	Outcome (state after operation) undefined
I	Set by immediate data
<ea>	Effective Address Operand
<data>	Immediate data
<label>	Assembler label
<vector>	TRAP instruction Exception vector (0-15)
<rg.lst>	MOVEM instruction register specification list
<displ.>	LINK instruction negative displacement
...	Same as previous instruction

Legend	
Dn	Data Register (n is 0-7)
An	Address Register (n is 0-7)
b	08-bit constant
w	16-bit constant
l	32-bit constant
x	8-, 16-, 32-bit constant
Rx	Index Register Specification, one of:
Dn.W	Low 16 bits of Data Register
Dn.L	All 32 bits of Data Register
An.W	Low 16 bits of Address Register
An.L	All 32 bits of Address Register

Condition Codes for Bcc Instructions. Condition Codes set after CMP D0,D1 Instruction		
Relationship	**Unsigned**	**Signed**
D1 < D0	CS – Carry Bit Set	LT - Less Than
D1 <= D0	LS – Lower or Same	LE - Less than or Equal
D1 = D0	EQ – Equal (Z-bit Set)	EQ - Equal (Z-bit Set)
D1 != D0	NE - Not Equal (Z-bit Clear)	NE - Not Equal (Z-bit Clear)
D1 > D0	HI - HIgher than	GT - Greater Than
D1 >= D0	CC – Carry Bit Clear	GE - Greater than or Equal
	PL - PLus (N-bit Clear)	
	MI – Minus (N-bit Set)	
	VC - V-bit Clear (No Overflow)	
	VS - V-bit Set (Overflow)	

A.4 DUART 68681



A4	A3	A2	A1	READ (R/WN = 1)	WRITE (R/WN = 0)
0	0	0	0	Mode Register A (MR1A, MR2A)	Mode Register A (MR1A, MR2A)
0	0	0	1	Status Register A (SRA)	Clock Select Register A (CSRA)
0	0	1	0	BRG Test	Command Register A (CRA)
0	0	1	1	Rx Holding Register A (RHRA)	Tx Holding Register A (THRA)
0	1	0	0	Input Port Change Register (IPCR)	Aux. Control Register (ACR)
0	1	0	1	Interrupt Status Register (ISR)	Interrupt Mask Register (IMR)
0	1	1	0	Counter/Timer Upper Value (CTU)	C/T Upper Preset Value (CRUR)
0	1	1	1	Counter/Timer Lower Value (CTL)	C/T Lower Preset Value (CTLR)
1	0	0	0	Mode Register B (MR1B, MR2B)	Mode Register B (MR1B, MR2B)
1	0	0	1	Status Register B (SRB)	Clock Select Register B (CSRB)
1	0	1	0	1X/16X Test	Command Register B (CRB)
1	0	1	1	Rx Holding Register B (RHRB)	Tx Holding Register B (THRB)
1	1	0	0	Interrupt Vector Register (IVR)	Interrupt Vector Register (IVR)
1	1	0	1	Input Ports IP0 to IP6	Output Port Conf. Register (OPCR)
1	1	1	0	Start Counter Command	Set Output Port Bits Command
1	1	1	1	Stop Counter Command	Reset Output Port Bits Command

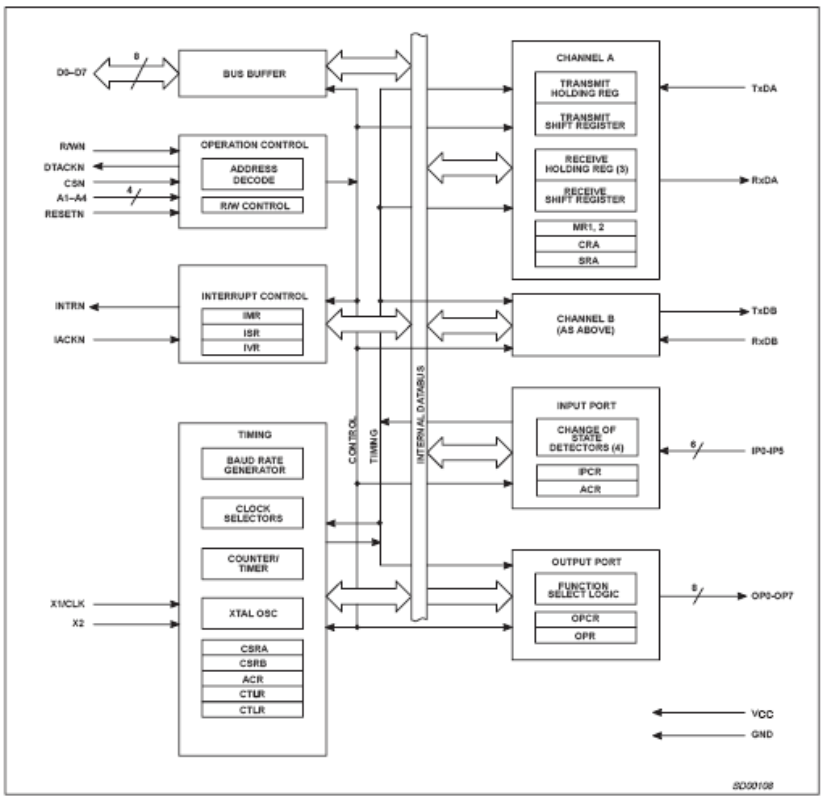


Figure 2. Block Diagram

	BIT 7	BIT 6	BIT 5	BIT 4 BIT 3	BIT 2	BIT 1 BIT 0
MR1A MR1B	RxRTS CONTROL	RxINT SELECT	ERROR MODE*	PARITY MODE	PARITY TYPE	BITS PER CHARACTER
	0 = No 1 = Yes	0 = RxRDY 1 = FFULL	0 = Char 1 = Block	00 = With Parity 01 = Force Parity 10 = No Parity 11 = Multidrop Mode	0 = Even 1 = Odd	00 = 5 01 = 6 10 = 7 11 = 8

NOTE:
*In block error mode, block error conditions must be cleared by using the error reset command (command 4x) or a receiver reset.

	BIT 7 BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
MR2A MR2B	CHANNEL MODE	TxRTS CONTROL	CTS ENABLE Tx	STOP BIT LENGTH*			
	00 = Normal 01 = Auto-Echo 10 = Local loop 11 = Remote loop	0 = No 1 = Yes	0 = No 1 = Yes	0 = 0.563 1 = 0.625 2 = 0.688 3 = 0.750	4 = 0.813 5 = 0.875 6 = 0.938 7 = 1.000	8 = 1.563 9 = 1.625 A = 1.688 B = 1.750	C = 1.813 D = 1.875 E = 1.938 F = 2.000

NOTE:
*Add 0.5 to values shown for 0 - 7 if channel is programmed for 5 bits/char.

	BIT 7 BIT 6 BIT 5 BIT 4	BIT 3 BIT 2 BIT 1 BIT 0
CSRA CSRB	RECEIVER CLOCK SELECT	TRANSMITTER CLOCK SELECT
	See Text	See Text

NOTE:
* See Table 6 for BRG Test frequencies in this data sheet, and "Extended baud rates for SCN2681, SCN68681, SCC2691, SCC2692, SCC68681 and SCC2698B" in application notes elsewhere in this publication

	BIT 7	BIT 6 BIT 5 BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CRA CRB		MISCELLANEOUS COMMANDS	DISABLE Tx	ENABLE Tx	DISABLE Rx	ENABLE Rx
	Not used – should be 0	See Text	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes

NOTE:
*Access to the upper four bits of the command register should be separated by three (3) edges of the X1 clock. A disabled transmitter cannot be loaded.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
SRA SRB	RECEIVED BREAK*	FRAMING ERROR*	PARITY ERROR*	OVERRUN ERROR	TxEMT	TxRDY	FFULL	RxRDY
	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes

NOTE:
*These status bits are appended to the corresponding data character in the receive FIFO. A read of the status provides these bits (7:5) from the top of the FIFO together with bits (4:0). These bits are cleared by a "reset error status" command. In character mode they are discarded when the corresponding data character is read from the FIFO. In block error mode, block error conditions must be cleared by using the error reset command (command 4x) or a receiver reset.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3 BIT 2	BIT 1 BIT 0
OPCR	OP7	OP6	OP5	OP4	OP3	OP2
	0 = OPR[7] 1 = TxRDYB	0 = OPR[6] 1 = TxRDYA	0 = OPR[5] 1 = RxRDY/ FFULLB	0 = OPR[4] 1 = RxRDY/ FFULLA	00 = OPR[3] 01 = C/T OUTPUT 10 = TxCB(1x) 11 = RxCB(1x)	00 = OPR[2] 01 = TxCA(16x) 10 = TxCA(1x) 11 = RxCA(1x)

OPR	BIT 7		BIT 6		BIT 5		BIT 4		BIT 3		BIT 2		BIT 1		BIT 0	
OPR bit	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
OP pin	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

NOTE:
The level at the OP pin is the inverse of the bit in the OPR register.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
ACR	BRG SET SELECT	COUNTER/TIMER MODE AND SOURCE			DELTA IP3 INT	DELTA IP2 INT	DELTA IP1 INT	DELTA IP0 INT
	0 = set 1 1 = set 2	See Table 4			0 = Off 1 = On	0 = Off 1 = On	0 = Off 1 = On	0 = Off 1 = On

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
IPCR	DELTA IP3	DELTA IP2	DELTA IP1	DELTA IP0	IP3	IP2	IP1	IP0
	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = Low 1 = High	0 = Low 1 = High	0 = Low 1 = High	0 = Low 1 = High

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
ISR	INPUT PORT CHANGE	DELTA BREAK B	RxRDY/ FFULLB	TxRDYB	COUNTER READY	DELTA BREAK A	RxRDY/ FFULLA	TxRDYA
	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
IMR	IN. PORT CHANGE INT	DELTA BREAK B INT	RxRDY/ FFULLB INT	TxRDYB INT	COUNTER READY INT	DELTA BREAK A INT	RxRDY/ FFULLA INT	TxRDYA INT
	0 = Off 1 = On	0 = Off 1 = On	0 = Off 1 = On	0 = Off 1 = On	0 = Off 1 = On	0 = Off 1 = On	0 = Off 1 = On	0 = Off 1 = On

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CTUR	C/T[15]	C/T[14]	C/T[13]	C/T[12]	C/T[11]	C/T[10]	C/T[9]	C/T[8]

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CTLR	C/T[7]	C/T[6]	C/T[5]	C/T[4]	C/T[3]	C/T[2]	C/T[1]	C/T[0]

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
IVR	IVR[7]	IVR[6]	IVR[5]	IVR[4]	IVR[3]	IVR[2]	IVR[1]	IVR[0]

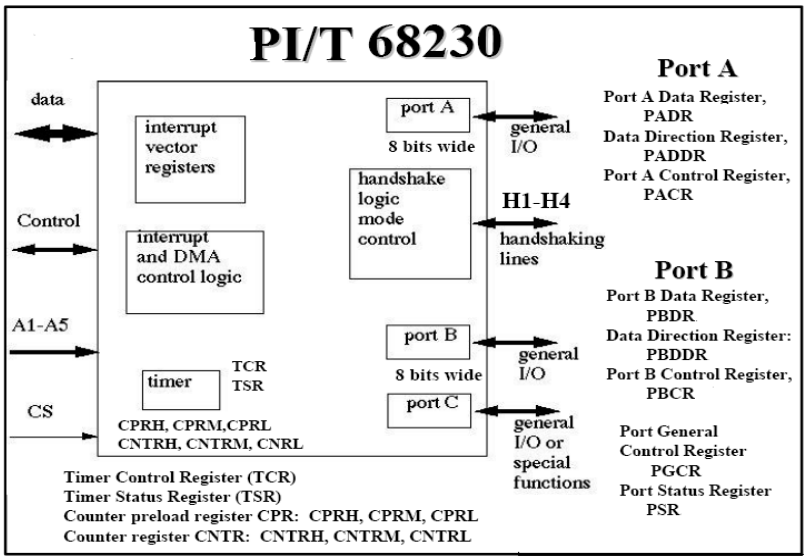
CSRA[3:0]	ACR[7] = 0	Baud Rate ACR[7] = 1
1110	IP3-16X	IP3-16X
1111	IP3-1X	IP3-1X

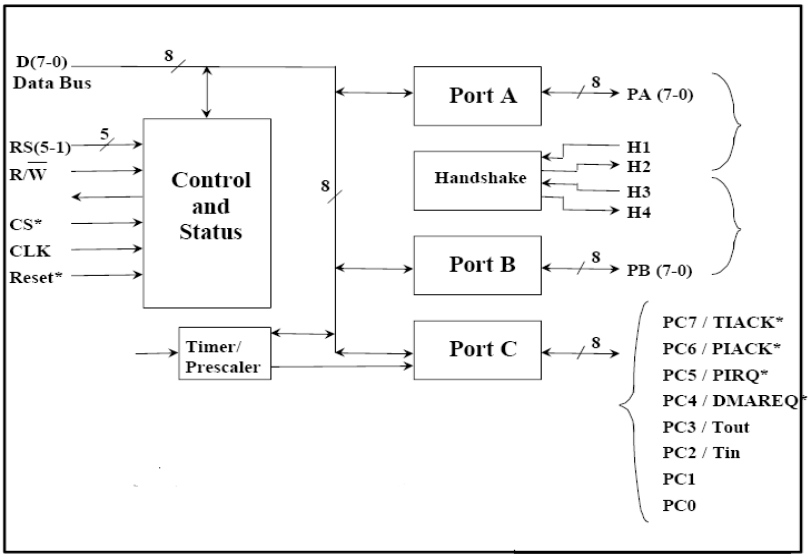
CSRA[7:4]	ACR[7] = 0	Baud Rate ACR[7] = 1
0000	50	75
0001	110	110
0010	134.5	134.5
0011	200	150
0100	300	300
0101	600	600
0110	1,200	1,200
0111	1,050	2,000
1000	2,400	2,400
1001	4,800	4,800
1010	7,200	1,800
1011	9,600	9,600
1100	38.4k	19.2k
1101	Timer	Timer
1110	IP4-16X	IP4-16X
1111	IP4-1X	IP4-1X

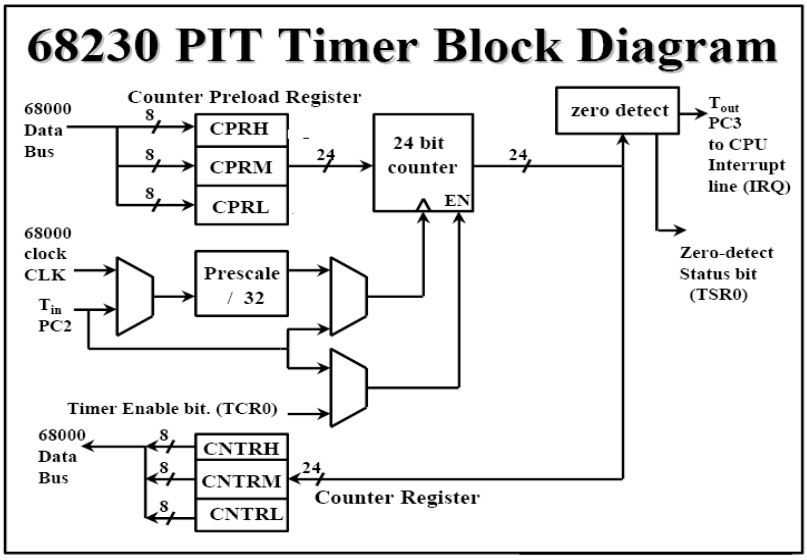
ACR [6:4]	MODE	CLOCK SOURCE
000	Counter	External (IP2)*
001	Counter	TxCA – 1x clock of Channel A transmitter
010	Counter	TxCB – 1x clock of Channel B transmitter
011	Counter	Crystal or external clock (x1/CLK) divided by 16
100	Timer (square wave)	External (IP2)*
101	Timer (square wave)	External (IP2) divided by 16*
110	Timer (square wave)	Crystal or external clock (X1/CLK)
111	Timer (square wave)	Crystal or external clock (X1/CLK) divided by 16

NOTE:
* In these modes, the Channel B receiver clock should normally be generated from the baud rate generator. Timer mode generates squarewave.

A.5 PI/T 68230


PI/T 68230
data
Control
A1-A5
CS
interrupt vector registers
interrupt and DMA control logic
timer
TCR
TSR
CPRH, CPRM,CPRL
CNTRH, CNTRM, CNRL
port A
8 bits wide
handshake logic mode control
port B
8 bits wide
port C
general I/O
H1-H4
handshaking lines
general I/O
general I/O or special functions
Port A
Port A Data Register, PADR
Data Direction Register, PADDR
Port A Control Register, PACR
Port B
Port B Data Register, PBDR
Data Direction Register: PBDDR
Port B Control Register, PBCR
Port General Control Register PGCR
Port Status Register PSR
Timer Control Register (TCR)
Timer Status Register (TSR)
Counter preload register CPR: CPRH, CPRM, CPRL
Counter register CNTR: CNTRH, CNTRM, CNTRL


D(7-0)
Data Bus
8
RS(5-1)
R/W
CS*
CLK
Reset*
5
Control and Status
8
Timer/ Prescaler
Port A
8
PA (7-0)
Handshake
H1
H2
H3
H4
Port B
8
PB (7-0)
Port C
8
PC7 / TIACK*
PC6 / PIACK*
PC5 / PIRQ*
PC4 / DMAREQ*
PC3 / Tout
PC2 / Tin
PC1
PC0



Format of Timer Control Register (TCR)

TCR7 TCR6 TCR5	TCR4	TCR3	TCR2 TCR1	TCR0
Tout/TIACK* Control	Zero-detect Control	None	Clock Control	Timer Enable
Set function of port C lines including interrupt usage (PC3/Tout)	0 load from CPR 1 roll over on zero detect	Still none ;-)	Set source of timer clock Scaled by 32 or not	Bit 0 Disable 1 Enable

TCR Format for Different Timer Modes

Mode	TCR7	TCR6	TCR5	TCR4	TCR3	TCR2	TCR1	TCR0
1	1	X	1	0	0	00 or	1X	1
2	0	1	X	0	0	00 or	1X	1
3	1	X	1	1	0	00 or	1X	1
4	0	0	X	1	0	0	0	1
5	0	0	X	1	0	0	X	1

Mode 1 = Real-time clock
Mode 2 = Square wave generator
Mode 3 = Interrupt after timeout
Mode 4 = Elapsed time measurement
Mode 5 = Pulse counter
Mode 6 = Period measurement

Format of Port General Control Register PGCR

PGCR7 PGCR6	PGCR5 PGCR4	PGCR3 PGCR2 PGCR1 PGCR0
Port Mode Control 00 Mode 0 01 Mode 1 10 Mode 2 11 Mode 3	H34 Enable H12 Enable 0 Disable 1 Enable	H4 sense H3 sense H2 sense H1 sense 0 Active low 1 Active high

PACR Format of Port A Control Register in Mode 0

PACR7 PACR6	PACR5 PACR4 PACR3	PACR2	PACR1 PACR0
Submode: 00 submode 0 01 submode 1 10 submode 1x	H2 Control 0xx Edge-sensitive input 100 output- negated 101 output - asserted 110 output - interlocked Handshake 111 Output - pulsed handshake	H2 Interrupt 0 Disabled 1 Enabled	H1 Control 0X H1 interrupt disabled 10 H1 interrupt enabled XX

PBCR Format of Port B Control Register in Mode 0

PBCR7 PBCR6	PBCR5 PBCR4 PBCR3	PBCR2	PBCR1 PBCR0
Submode: 00 submode 0 01 submode 1 10 submode 1x	H4 Control 0xx Edge-sensitive input 100 output- negated 101 output - asserted 110 output - interlocked Handshake 111 Output - pulsed handshake	H4 Interrupt 0 Disabled 1 Enabled	H3 Control 0X H3 interrupt disabled 10 H3 interrupt enabled XX

Port Status Register, PSR

PSR7	PSR6	PSR5	PSR4	PSR3	PSR2	PSR1	PSR0
H4 Level	H3 Level	H2 Level	H1 Level	H4S	H3S	H2S	H1S