Chapitre 2. Architecture d'un microprocesseur 16 bits (MC68000) (5 semaines)
Architecture interne, Brochage, Registres spéciaux, Modes d'adressages, Jeux d'instructions, Différentes architectures : Harvard,pipeline

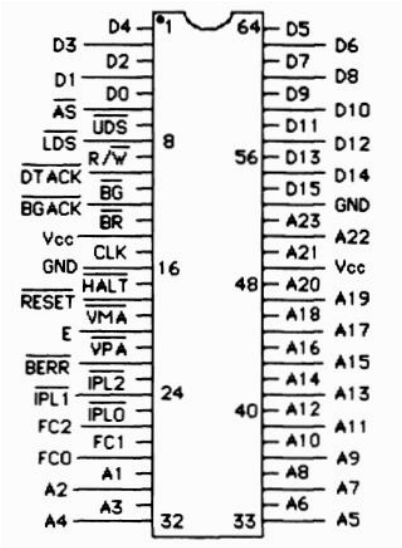
Présentation du microprocesseur (µP)MC 68000 de Motorola

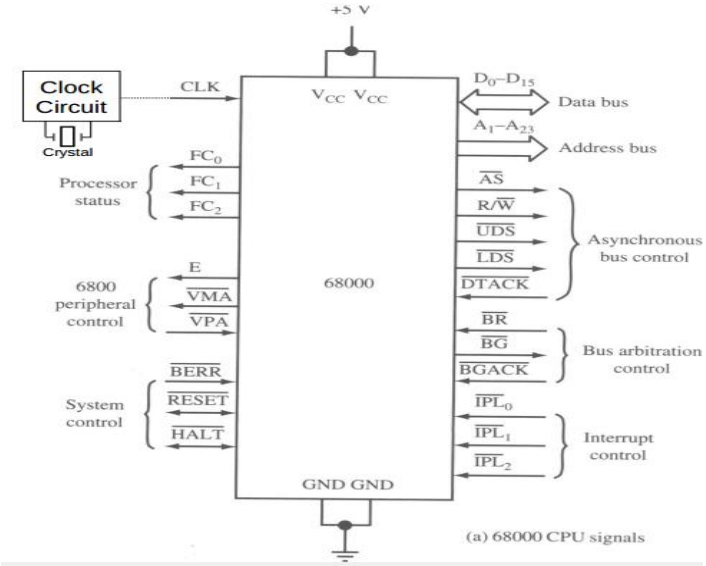
Le µP 68000 a été introduit en Septembre 1979 et comporte 68000 transistors à l'intérieur du circuit intégré. Ces transistors forment tous ses circuits logiques formant une architecture de type Von Neuman et CISC. Le MC 68000 est dit à 16 ou 32 bits (16/32), ceci résume la façon dont le circuit manipule les données. Le bus de données interne est sur 32 bits mais le processeur communique avec l'extérieur (Entrées/sorties) sur un bus de données externe de 16 bits.
Tous les registres du 68000 sont sur 32 bits excepte le registre d'état CCR 16 bits.
les formats de données manipulées : Octet = 8 bits Mot / Word = 16 bits Long mot / Long Word = 32 bits

Le brochage du MC68000

Le 68000 est un composant électronique de 64 broches, qui possède un bus de données sur 16 bis et un bus d'adressage sur 23 bits en externe, ce qui détermine une région mémoire maximum de 8 Mega-mots avec le mot qui est de 16 bits = 2 octets donc 16 Mega octets de mémoire adressable totale.
Note: 2^{20} x 8bits = 1 Mega-octet

Ses Niveaux de tension : Vdd-Vss compris entre 3V et 7V. Compatible TTL (Vdd=Vcc=5volts et Gnd=VSS=0V)
La Fréquence d'utilisation est dans la gamme : 4Mhz à 12.5Mhz selon la version.
Pour une fréquence de l'horloge f=10 Mega-hertz, correspond un cycle d'horloge de T=1/f =100 nanosecondes. Notons que la plus petite opération nécessite 4 cycles d'horloge donc 400nS.

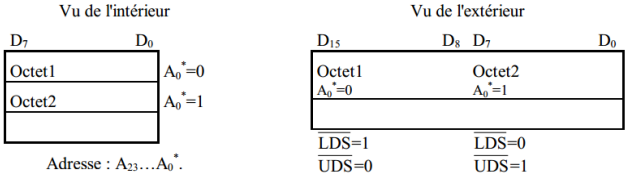




Le 68000 a deux modes de fonctionnement : utilisateur (normal; mode du programmateur) et superviseur (mode privilégié accès au ressource système).
L'état du processeur est caractérisé par les fonctions codes FC0, FC1, FC2 (processor status).
Sur la figure on a 16 lignes de données (bidirectionnelles; entrées/sorties) et 23 lignes d'adresse unidirectionnelles (sorties) donc il reste 64-16-23=25 lignes (pins) sur le circuit intégré du processeur. Ces lignes sont celles du bus de contrôle, l'alimentation et l'horloge. Chaque ligne a une fonction bien précise telle que R/W pour lire ou écrire à la mémoire

Le problème du bus d'adresse

Le 68000 est un processeur 16 bits, mais les circuits mémoires sont des circuits 8 bits. Donc, le 68000 gère la mémoire comme une mémoire à 8 bits mais il adresse des mots de 16 bits. Il n'y a donc pas de bit A0 sur le bus d'adresses externe. Il est remplacé par les signaux UDS et LDS.



Bus d'adresses : A1…A23 (en externe).
Lecture octet par octet avec LDS et UDS .
Noter que la majorité des signaux de contrôle sont actifs bas ceci veut dire que c'est le 0 qui les valident et le 1 représente l'état du repos (logique inverse)
Lecture d'un mot avec LDS=0 et UDS =0.

Registres internes

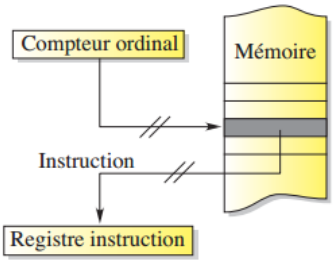
• 8 registres de données, 32 bits : D0… D7 Ces registres peuvent être manipulés soit comme des bytes B, soit comme des word W, soit comme des long L.

• 8 registres d'adresse, 32 bits : A0… A7

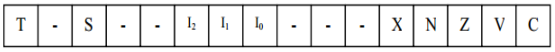
A7 est aussi appelé SP (Stack pointer) qui représente le pointeur de pile. En interne, il existe en fait deux pointeurs différents l'USP (User Stack Pointer) en mode utilisateur et le SSP (Supervisor Stack Pointer) en mode superviseur. Il s'agit toujours du registre A7, le 68000 utilisant implicitement l'un ou l'autre en fonction de son mode de fonctionnement.

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

• Le compteur ordinal, PC (Program Counter), contient l'adresse de la prochaine instruction à exécuter, c'est un compteur programmable . Ceci indique qu'il est possible de modifier à tout moment son contenu en ne respectant pas la séquence de comptage (cas de branchements, appels de sous-programmes). L'instruction dont l'adresse est fournie par le contenu du compteur ordinal est déposée dans le registre instruction pour décodage (figure L'accès à une instruction).



• SR ou status register et CCR ou condition code register



Registre à 16 bits : C'est le registre d'état, qui comporte des bits indicateurs ou flags. On distingue les bits manipulés directement par l'utilisateur via son programme on les appelle les indicateurs et les bits manipulés par le système (mode superviseur), les autres bits marqués par – sont indéfinis.

Bits systèmes:

- T: mode trace (mode débogage)
- S: mode superviseur (S=0 utilisateur, S=1 superviseur)
- I2, I1, I0: masque d'interruptions

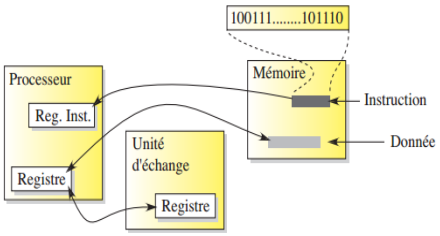
Bits utilisateurs (CCR):

- X, N, Z, V, C: indicateurs arithmétiques Z: zéro, N: négative bit de poids fort ou de signe, C: carry retenue, X: extend retenue non signée, V: overflow signé (débordement)

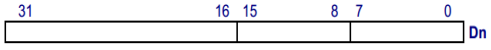
Voici quelques exemples du positionnement des flags C, V, Z et N pour les instructions d'additions :
Addition sur 8 bits (.B) $7A + $86 = $100 (le résultat sur 8 bits est $00) C= 1, V= 0, Z= 1, N= 0
Addition sur 16 bits(.W) $9F00 + $8E00 = $12D00 (le résultat sur 16 bits est $2D00) C=1,V=1,Z=0,N=0
Addition sur 32 bits (.L) $70000000 + $10000000 = $80000000 C= 0, V=1, Z= 0, N= 1

Format des instructions

Le format représente la structure et la taille des instructions qui doivent suivre une représentation spécifique et un arrangement de bits bien défini imposé par le type du processeur utilisé.
Dans le cas du 68000, les instructions contiennent les informations relatives au travail à entreprendre, soit :
1. le type de l'opération ou code opération (en anglais opcode, contraction de opération code) : opérations de transferts, lecture, écriture ; – opérations arithmétiques, additions, soustractions. .
- opérations logiques : ET, OU. . . – opérations de branchement, etc.
2. des informations définissant le type des opérandes. Un opérande peut être : – une valeur, – le numéro d'une cellule mémoire (l'adresse), – ou le numéro d'un registre.
3. un spécificateur de mode d'accès aux opérandes indiquant comment il faut interpréter l'opérande qui a été codée dans l'instruction.
Chacune de ces informations est codée dans l'instruction par un ensemble de bits, ou champ. l'instruction présente en mémoire contient ces 3 informations dans des mots consécutifs.
Le nombre d'instructions (jeu d'instructions) est directement lié au format du code opération.



• la taille minimale d'une instruction en mémoire est d'un mot de 16 bits (2 octets) ;
• la taille maximale d'une instruction en mémoire est de cinq mots de 16 bits (10 octets).
• Chaque instruction est un multiple de mots de 16 bits
• Le premier mot de 16 bit contient le code opération qui indique implicitement la taille de l'instruction
• Le Compteur de programme correctement initialisé assure la bonne exécution
La forme générale de l'instruction: operation.<format> <source>, <destination>
Les instructions peuvent porter sur le format de données suivant:
Un octet (.B), les bits concernés sont alors de 0 à 7, les autres bits du registre ne sont pas influencés. - un mot (.W), bits de 0 à 15, les autres bits du registre ne sont pas influencés. - un mot long (.L), bits de 0 à 31



Les modes d'adressage

La façon d'accéder aux données dans une instruction est désignée par mode d'adressage.

L'adresse mémoire effective est l'adresse qui est effectivement utilisée au moment de l'exécution de l'instruction. Quelquefois cette adresse est la même que celle qui apparaît dans l'instruction, mais ce n'est généralement pas le cas. Cette adresse peut être modifiée de plusieurs façons au moment de l'exécution selon les divers modes d'adressage utilisés qui sont présentés ci-dessous.

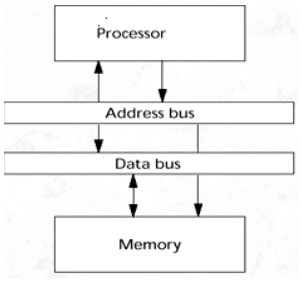
Sur le MC68000 , Il y a sept modes d'adressage principaux offrant 14 modes au total:
•1 **absolue**
L'adresse effective est une constante, le plus souvent écrite en hexadécimal comme par exemple $1000, l'adresse de la donnée est fournie dans le code instruction. MOVE.W $1000 ,$2000
• 2 **direct** des registres : Dx ou Ax
On opère directement sur les registres soit en lecture, soit en écriture. Exemple: MOVE.B D0 ,$2000
MOVE.W A0 ,A1
MOVE.L $2000,D0 MOVE.L A1,$2000
• 3 **immédiat** : écrite avec le signe **#** suivie d'une constante
La source est la valeur d'une constante immédiatement citée dans l'instruction. Par défaut, les constantes sont décimales. Les constantes hexadécimales doivent être précédées du symbole **$** et les constantes binaires du symbole **%**.
MOVE.L #$12345678,D0 (déplace la valeur 12345678 en hexa dans D0) MOVE.B #%11110000,D0
Dans cette dernière instructions, les 24 derniers bits du registre D0 ne sont pas affectés.

• 4 **indirect** : (Ax) On ne passe pas l'adresse directement, mais on donne une référence à cette adresse. L'adresse effective est le contenue du registre d'adresse Ax. Ou obtenue par l'addition du contenu d'un registre d'adresse et d'une constante (déplacement ou offset) et/ou du contenu d'un registre de donnée ou d'adresse désigné alors par le terme d'index.
MOVE.L #$2000,A1 MOVE.W (A1),D0 ce qui est équivalent à MOVE.W $2000 ,D0
•5 **indirect post-incrémenté** : noté (Ax)+
L'adresse effective est le contenu de l'adresse Ax; après exécution de l'instruction utilisant ce mode d'adressage, Ax est incrémenté de 1,2 ou 4 selon le format de l'instruction .B, .W, ou .L.
MOVE.L #$2000,A1 MOVE.B (A1)+ ,D1 ; A1 = A1+ 1
MOVE.W (A1)+ ,D1 ; A1 = A1+ 2
Les adresses doivent toujours être paires pour manipulation du mot (.W) et du long mot (.L):
•6 **indirect pré-décrémenté** : noté - (Ax)
L'adresse effective est le contenu de l'adresse Ax décrémentée de 1,2 ou 4 selon le format de l'instruction .B, .W, ou .L respectivement.
MOVE.L #$2000,-(A2)
•7 **relatif** au compteur ordinal : l'adresse est calculée à partir de l'addition du compteur ordinal et d'un déplacement et/ou d'un index.
An et Dn désignent respectivement les registres d'adresse et de donnée. Aven n=0,1,….,7
La table précise 12 modes d'adressage.

Mode	Code	Champ registre	Syntaxe
Direct	000	Num. reg.	Dn
Direct	000	Num. reg.	An
Indirect	010	Num. reg.	(An)
Indirect	011	Num. reg.	(An)+
Indirect	100	Num. reg.	-(An)
Indirect	101	Num. reg.	d(An)
Indirect	110	Num. reg.	d(An,Rm)
Absolu	111	000	xxxx
Absolu	111	001	xxxxxxxx
Relatif	111	010	Rel. CO
Relatif	111	011	Rel. CO+Rm
Immédiat	111	100	#xxxx

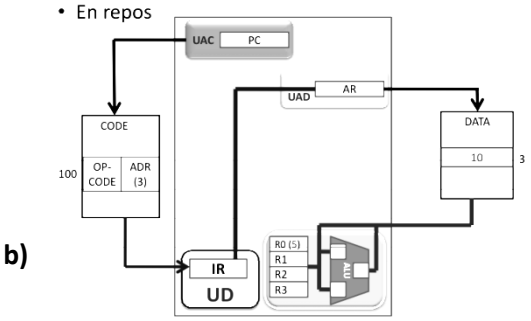
Organisation de la mémoire
Pour Le MC68000, Les mots sont stockés avec les 8 bits **inférieurs** (end) (moins significatifs) dans le plus **élevé** des deux emplacements de stockage (big) (adresse impaire) c'est l'ordre tde ype Big-endian Contrairement aux processeurs dit little-endian, comme la famille Intel 80x86
Exemple: MOVE # $3210, 0
Déplacement du mot 3210 dans l'adresse 0 en fait les deux octets du mot à savoir: 32 (MSB) et 10 (LSB: moins significatif) se déplacent respectivement à l'dresse 0 (paire: end) et l'adresse 1 (impaire)
(32 dans 0 et 10 dans 1)

Architecture de base des microprocessurs
Les deux célèbres architectures des µP sont :
–L'architecture de Von-Neumann:se base sur le partage de buses entre les différentes zones mémoires (CODE, DATA et I/O).
–L'architecture de Harvard se base sur la séparation de buses entre les différentes zones mémoires.



a)

a) Von-Neumann
b) Harvard: Deux bus de données et deux bus d'adresse



b)

Technique de pipeline
Le µP exécute un traitement qui existe dans une mémoire. Le traitement est un ensemble d'instructions l'une après l'autre (séquence), dont le µP prend un temps pour exécuter. On peut accélérer ce traitement par la notion de pipeline dont l'Idée de principe est inspirée de l'organisation du travail à la chaîne. L'exécution d'une instruction peut être décomposée en plusieurs phases qui

s'exécutent indépendamment les unes des autres si l'on dispose d'unités fonctionnelles (du matériel) le permettant. La séquence de ces phases d'une instruction est exécutée comme si celle-ci était dans une chaîne de montage.

Chaque instruction prend une phase ou plusieurs pour exécuter; en général, ces phases sont (FETCH,DECODE,LECTURE DES OPERANDES, EXECUTION, ECRITURE).

– fetch: phase de recherche de l'instruction dans la mémoire et l'apporter au niveau du µP.

--décode décoder l'instruction pour sélectionner l'opération à exécuter

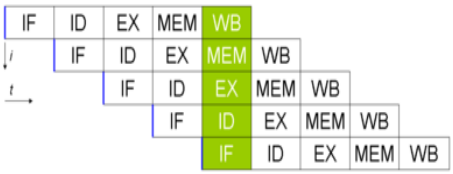
– lecture des opérandes : phase optionnelle, si l'instruction a besoin des opérandes, le µP charge ces opérandes à partir de la mémoire ou un port (exemple : adresse d'une case mémoire, constante…).

– exécution: phase de réalisation de l'opération demander dans l'instruction (exemple : addition, soustraction…).

– écriture: phase optionnelle, si l'instruction a besoin d'écrire le résultat –dans une case mémoire ou un port.

On peut chevaucher ces 5 phases appartenant à 5 instructions différentes de telle sorte que ces phases s'exécute simultanément délivrant ainsi l'exécution d'une instruction pendant un cycle d'horloge au lieu de 5 cycles si on n'applique pas le pipeline et donc on accélère ainsi le traitement

Note: Le pipeline n'est pas utilisé dans le MC6800



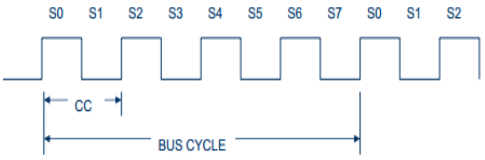
Définitions

Cycle horloge(cc) →1 période du signal horloge

Etat du bus(Sn) →½ période du signal horloge

Cycle de bus →temps pour accomplir une lecture ou écriture

Cycle d'instruction →temps pour lire, décoder et exécuter une instruction



L'assembleur

Lorsqu'un constructeur conçoit un processeur, il détermine aussi son assembleur. C'est pour cela que l'assembleur n'est pas universel car il est différent sur chaque processeur , même si une instruction est identique entre 2 CPU différentes, elle ne sera pas compatible sur chaque processeur car le code binaire correspondant sera différent. Ceci est l'un des plus gros inconvénients de la programmation en assembleur car il faut alors reprogrammer tout, si on veut le porter sur un autre processeur.

Chaque processeur comporte un nombre plus ou moins important d'instructions, qu'on appelle le jeu d'instruction. Le Jeu d'instruction du 6800 comporte 56 instructions.

Ces instructions sont formées à l'aide du code binaire dans le programme. Il est beaucoup plus facile d'utiliser des symboles à la place de ces codes binaires. C'est pour cela que l'on fait appel au traducteur qui permet de transformer un programme assembleur fait avec des mots clés compréhensibles pour nous (mais incompréhensible pour la machine), en un programme exécutable compréhensible par le processeur. Ces mots clés, ou mnémoniques, sont souvent la compression d'un mot ou d'une expression en anglais présentant l'action de l'instruction.

Les instructions sont souvent suivies d'opérandes permettant d'indiquer sur quel(s) registre(s) on veut effectuer le traitement et quelle valeur on veut utiliser.

Exemple sur 68000: ADD.W D1,D0

Cette instruction correspond a l'opération D0=D0+D1

Le langage d'assemblage est une représentation symbolique du codage binaire des instructions machine, ce dernier codage étant la seule forme d'information que peut traiter le processeur. L'assembleur est chargé, non seulement de la traduction en codage binaire, mais aussi d'affecter une adresse à chaque symbole (noms de variable, étiquettes) utilise par le programme.
Le langage d'assemblage propose aussi des directives et des pseudoinstructions qui servent a réserver des cellules mémoire, les initialiser et faciliter l'écriture des programmes.

Assembleur est un langage de programmation donc il est régi par des règles sémantiques et par syntaxes. Un code source contient une instruction par ligne. Une instruction est divisée en quatre champs distincts; L'écriture d'une instruction en assembleur MC68000 a la forme suivante:

Étiquette Mnémonique.format source, destination ; commentaires

Étiquette : Facultative, permet de repérer une instruction dans un programme (une adresse littérale) c'est l'adresse de l'instruction en mémoire (obtenue après assemblage en langage machine).
Mnémonique : Nom de l ' instruction
Format : Taille des données manipulées: .B ,.W ou .L
Source : Donnée de départ (source)
Destination : Endroit d'arrivée (destination)
La source et la destination peuvent être confondues
Exemple :

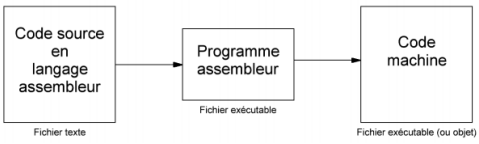
```
DEB MOVE.B D0,D1 ; Transfert l'octet de poids faible du registre D0 dans le registre D1.
 NOT.B D1; Complément à 1 de l'octet de poids faible de D1.
 BRA DEB ; Saute à l'étiquette DEB
```

Les pseudo-instructions
Les pseudo-instructions permettent de gérer l'allocation de la mémoire pour les variables : en allouant la taille, alignement et type.

Les directives
Les directives sont des indications données à l'assembleur et à l'´editeur de liens : elles fournissent des informations pour la construction du fichier objet. Elles ne donnent lieu ni a une traduction ni à une réservation de cellules mémoire. Une directive ne fait pas partie du jeu d'instructions d'un microprocesseur. Elle ne peut pas être traduite en langage machine.

Les macros
Une suite d'instructions peut apparaitre plusieurs fois dans un programme et utiliser des opérandes différents à chacune de ses occurrences. Une macro-instruction, ou macro, permet de gérer une telle suite d'instructions sans avoir a la réécrire a chaque occurrence. Une macro contient les éléments suivants :
Un nom, un corps, un début et une fin. Pour définir une macro, on dispose de pseudo-instructions qui en définissent les éléments. Une macro s'utilise en donnant son nom suivi de paramètres, comme si elle était une simple instruction.

Le programme assembleur traducteur
Un programme assembleur est un programme informatique qui traduit un programme écrit en code assembleur en code machine. Ce processus de traduction s'appelle l'assemblage. Il peut exister plusieurs programmes assembleurs différents pour un même langage assembleur (tout comme il existe plusieurs compilateurs C différents pour le langage C).



Une fois l'assemblage terminé, le code machine généré peut être chargé en mémoire et exécuté par un microprocesseur.
Les instructions se retrouvent en mémoire sous la forme de mots de 16 bits. Parce que certaines instructions prennent Jusqu'à 5 mots d'espace dans la mémoire, le compteur PC sera incrémenté par 2 jusqu'à 10 avec un pas de 2.

Le champ étiquette
Le champ étiquette est le premier champ d'une instruction en langage assembleur.
Une étiquette est facultative. Si la ligne contient une étiquette, cette dernière doit débuter sur le premier caractère de la ligne. Si la ligne ne contient pas d'étiquette, le premier caractère de la ligne doit être un es pace ou une tabulation.
L'assembleur attribue à une étiquette la valeur de l'adresse à laquelle sera placée l'instruction qui suit l'étiquette dans le code source.
Le champ mnémonique
Le champ mnémonique contient le nom d'une instruction ou d'une directive d'assemblage.
Une instruction peut être traduite en langage machine. Elle appartient au jeu d'instructions d'un microprocesseur. Elle permet, comme son nom l'indique, de donner une directive au programme assembleur au moment de l'assemblage.
Le code opératoire est décodé par des circuits de décodage contenus dans le microprocesseur.
Des signaux de commande pour l'UAL sont produits en fonction de l'opération demandée qui est alors exécutée.
Remarque : pour exécuter une instruction, l'UAL utilise des registres de travail.
Instructions de branchement
Deux manières d'effectuer un branchement
Inconditionnel: avec JMP et BRA
Conditionnel avec Bcc, (DBcc)
BRA étiquette Déroutement relatif PC = PC + étiquette
JMP étiquette Déroutement absolu. PC = étiquette (Adresse de branchement).
On parle parfois de Déplacement définit comme étant le nombre d'instructions entre l'endroit du branchement et l'endroit où le programme doit se brancher.

Les instructions de branchement conditionnel
Dérouter un programme en fonction d'une condition .
Elles permettent d'effectuer un branchement en fonction du registre de conditions.
La condition d'un branchement conditionnel se fait sur un flag ou une combinaison de flags.
Syntaxe : Bcc etiquette (cc : code condition)
Remarque : il s'agit d'un branchement relatif .
Les différentes conditions sont résumées dans le tableau suivant

CC	retenue à 0	$\overline{C}$	
CS	retenue à 1	C	
EQ	égal	Z	
NE	non égal	$\overline{Z}$	
GE	supérieur ou égal	$N.V + \overline{N}.\overline{V}$	Arithmétique signée
GT	supérieur	$N.V.\overline{Z} + \overline{N}.\overline{V}.\overline{Z}$	
HI	plus grand	$\overline{C}.\overline{Z}$	Arithmétique non signée
LE	inférieur ou égal	$Z + N.\overline{V} + \overline{N}.V$	Arithmétique signée
LT	inférieur	$N.\overline{V} + \overline{N}.V$	
LS	plus petit ou égal	$C + Z$	Arithmétique non signée
MI	négatif	N	
PL	positif	$\overline{N}$	
VC	pas de dépassement	$\overline{V}$	
VS	dépassement	V	

Pour la soustraction, le bit C (carry) est le complément de celui trouvé dans la soustraction avec la convention signée. Ce choix est cohérent avec la retenue utilisée pendant l'opération de soustraction.

```
MOVE.B  #$0A,D0   ; X
MOVE.B  #$0B,D1   ;Y
CMP.B D0,D1   ;  Y-X = #$1 N=0, Z=0, V=0, C=0
BGT vrai          ; Y>X
```

L'instruction CMP

Elle permet de comparer 2 données. Elle positionne les bits du registre code condition.
Syntaxe : CMP.<format> <source>, <destination>
L'opération de comparaison fait la soustraction <destination> - <source>, mais ne modifie pas la destination.

Principe d'extension du signe

L'adresse effective(AE) est l'emplacement d'un opérande est calculée sur 24 bits (A23…A0).
Pb : Si l'adresse n'est pas sur 24 bits, elle est complétée en extension de signe.
Exemple: NOT.L $F000 AE : FFF000 NOT.L $7000 AE : 007000

Les principales directives d'assemblage pour le MC68000

1. La directive ORG

La directive ORG(Origin) sert à préciser à l'assembleur l'adresse à partir de laquelle seront assemblées les instructions en mémoire.
Un programme peut posséder plusieurs ORG (il faut faire attention au recouvrement des adresses).
Exemple: ORG $1000 le programme qui suit aura son adresse d'origine (début) à 1000 en hexa

2.La directive EQU

La directive EQU(Equate) permet d'attribuer explicitement une valeur à une étiquette. L'assembleur remplacera ensuite l'étiquette par la valeur qui lui a été attribuée.

3.La directive DC

La directive DC(Define Constant) permet de placer des données en mémoire. Ces données peuvent être des nombres en représentation décimale, hexadécimale ou binaire, mais aussi des chaînes de caractères. Dans ce dernier cas, ce sont les codes ASCII des caractères qui sont placés en mémoire.
Exemples :

```
ORG  $1000
DC.B  10,5,7,$7a,255,%11111001
DC.B  "Hello World",13,10,0
DC.W  5,6
DC.L  5,6
```

Le contenu de la mémoire à partir de l'adresse $1000 (chaque octet dans une adresse) sera donc le suivant :

```
1000   0A 05 07 7A FF F9                        DC.B  10,5,7,$7a,255,%11111001
1006   48 65 6C 6C 6F 20 57 6F 72 6C 64 0D 0A 00  DC.B  "Hello World",13,10,0
1014   00 05 00 06                              DC.W  5,6
1018   00 00 00 05 00 00 00 06                  DC.L  5,6
```

4. La directive DS

La directive DS(Define Storage) permet de réserver un espace de stockage qui pourra être utilisé lors de l'exécution d'un programme. Ceci permet d'éviter au programme d'écrire dans des emplacements qui ne lui seraient pas réservés et d'effacer des données ou du code qui ne doivent pas l'être.
Exemples :

```
ORG  $5000
TAB1  DS.B  4  ; Réserve 4 octets en mémoire (4 x 8 bits) à partir de  TAB1 = $5000
TAB2  DS.W  3  ; Réserve 3 mots en mémoire (3 x 16 bits) à partir de l'adresse suivante TAB2 = $5004
TAB3  DS.L  1  ; Réserve 1 mot long en mémoire (1 x 32 bits) à partir de l'adresse TAB3 = $500A
NEXT  ; NEXT = $500E  (l'adresse suivante qui suit 500A , 500B,500C, 500D)
```

Exemples de boucle

1-La structure de boucle suivante est très utilisée en assembleur 68000

```
        MOVE.W  #n,D7       ; n → D7
                            ; (D7.W = compteur de boucle)
LOOP    ; ...               ; Corps de la boucle
        ; ...               ; exécuté n fois
        SUBQ.W  #1,D7       ; D7.W - 1 → D7.W (si D7.W est nul, Z passe à 1)
        BNE     LOOP        ; Branchement si D7.W ≠ 0 (si Z = 0)
```

2-L'instruction CMP s'utilise principalement avec des branchements conditionnels selon le modèle suivant :

CMP source,destination

Bcc <étiquette> ; Branchement si destination <condition> source

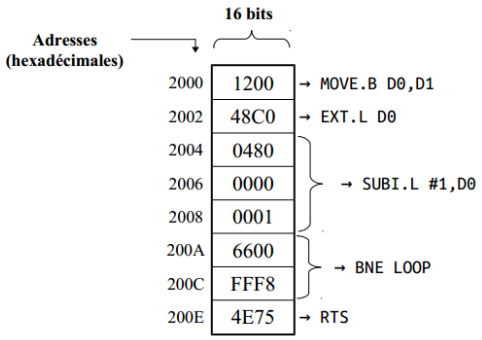
Exemple: CMP D0,D1

BNE LOOP Branchement si D1 n'est pas égale à D0

Exemple de code

```
ORG $2000               ; Place le programme à l'adresse $2000.
START   MOVE.B  D0,D1   ; D0 → D1.
        EXT.L   D0      ; Extension de signe.
LOOP    SUBI.L  #1,D0   ; D0 - 1 → D0.
        BNE     LOOP    ; Saut à LOOP tant que D0 n'est pas nul.
        RTS             ; Retour de sous-programme.
```

Une fois assemblé et chargé en mémoire, voici le code machine obtenu :



Avec le microprocesseur 68000, l'écriture ou la lecture d'un mot (ou d'un mot long) à une adresse impaire n'est pas admise.

Lorsque le transfert concerne un octet, celui-ci se présente sur les deux parties du bus de données, les deux même cases dans chacune des mémoires sont sélectionnées mais si l'adresse est impaire le 68000 active LDS (**low** data strobe) et le transfert se fait de ou dans la mémoire validée si l'adresse est paire c'est UDS (**upper** data strobe) qui est actif.

Il est facile de voir que la différence entre Adresse paire et impaire qui se fait par le bit A0 à l'intérieur du microprocesseur, se fait à l'extérieur avec LDS et UDS.

parité de l'adresse Æ	A0 = 0		A0 = 1	
Ø taille du transfert	LDS	UDS	LDS	UDS
OCTET	1	0	0	1
MOT	0	0		
MOT LONG	0	0		

Structure d'un programme pour l'assembleur MC68000:

Le programme placé dans un fichier text avec extension .asm (exemple pgr1.asm) contient deux parties la partie données et la partie code voir exemple

```
LABEL   OPCODE    OPERAND(S)        COMMENTS
        ORG       $1000             ;start of PROGRAM area
        MOVE.L    #$12,d0
        CLR.L     d1
        MOVE.B    data,d1
        ADD.B     d0,d1
        MOVE.B    d1,result
        RTS                         ;return

        ORG       $2000             ;start of DATA area
data    DC.B      $24
result  DS.B      1                 ;reserve a byte for result
        END       $1000             ;end of program and entry point
```

PROGRAM AREA

DATA AREA

La traduction de ce code source est faite par un assembleur (dans notre cas easy68k).

Le programmeur doit connaitre le jeu d'instructions qu'il peut employer

les différents types d'instruction

leur effet sur les registres accessibles

les registres auquel il a accès

le compteur de programme (adresse de la prochaine instruction)

les registres de travail,

les indicateurs de condition

Le jeu d'instruction du MC68000 comporte les instructions de déplacement (les move), les instructions arithmétiques, les instructions logiques, de décalage, de manipulation de bit, de branchement et les instructions systèmes (mode superviseur):

Data transfer group	
EXG	Exchange registers
LEA	Load effective address
LINK	Link and allocate
MOVE	Move data
MOVEA	Move address
MOVEM	Move multiple registers
MOVEP	Move peripheral data
MOVEQ	Move quick
PEA	Push effective address
SWAP	Swap register halves
UNLK	Unlink

Arithmetic group	
ADD	Add binary
ADDA	Add address
ADDI	Add immediate
ADDQ	Add quick
CLR	Clear operand
CMP	Compare
CMPA	Compare address
CMPI	Compare immediate
CMPM	Compare memory
DIVS	Divide signed numbers
DIVU	Divide unsigned numbers
EXT	Extend sign
MULS	Multiply signed numbers
MULU	Multiply unsigned numbers
NEG	Negate
NEGX	Negate with extend
SUB	Subtract binary
SUBA	Subtract address
SUBI	Subtract immediate
SUBQ	Subtract quick
SUBX	Subtract with extend
TAS	Test and set
TST	Test

Logical group	
AND	Logical AND
ANDI	AND immediate
OR	Logical OR
ORI	OR immediate
EOR	Exclusive OR
EORI	Exclusive OR immediate
NOT	Logical complement

Shift and rotate group	
ASL	Arithmetic shift left
ASR	Arithmetic shift right
LSL	Logical shift left
LSR	Logical shift right
ROL	Rotate left
ROR	Rotate right
ROXL	Rotate left with extend
ROXR	Rotate right with extend

Bit manipulation group	
BCHG	Bit change
BCLR	Bit clear
BSET	Bit set
BTST	Bit test

Binary coded decimal group	
ABCD	Add BCD
NBCD	Negate BCD
SBCD	Subtract BCD

Program control group	
B*cc**	Conditional branch
DB*cc**	Decrement and branch
S*cc**	Conditional set
BRA	Branch always
BSR	Branch to subroutine
JMP	Jump
JSR	Jump to subroutine
RTR	Return and restore
RTS	Return from subroutine

System control group	
ANDI SR	AND immediate to SR
EORI SR	EOR immediate to SR
MOVE SR	Move to/from SR
MOVE USP	Move to/from USP
ORI SR	OR immediate to SR
RESET	Reset processor
RTE	Return from exception
STOP	Stop processor
CHK	Check register
ILLEGAL	Illegal instruction
TRAP	Trap call
TRAPV	Trap on overflow
ANDI CCR	AND immediate to CCR
ORI CCR	OR immediate to CCR
EORI CCR	EOR immediate to CCR
MOVE CCR	Move to/from CCR
NOP	No operation

*Note: *cc* stands for condition code