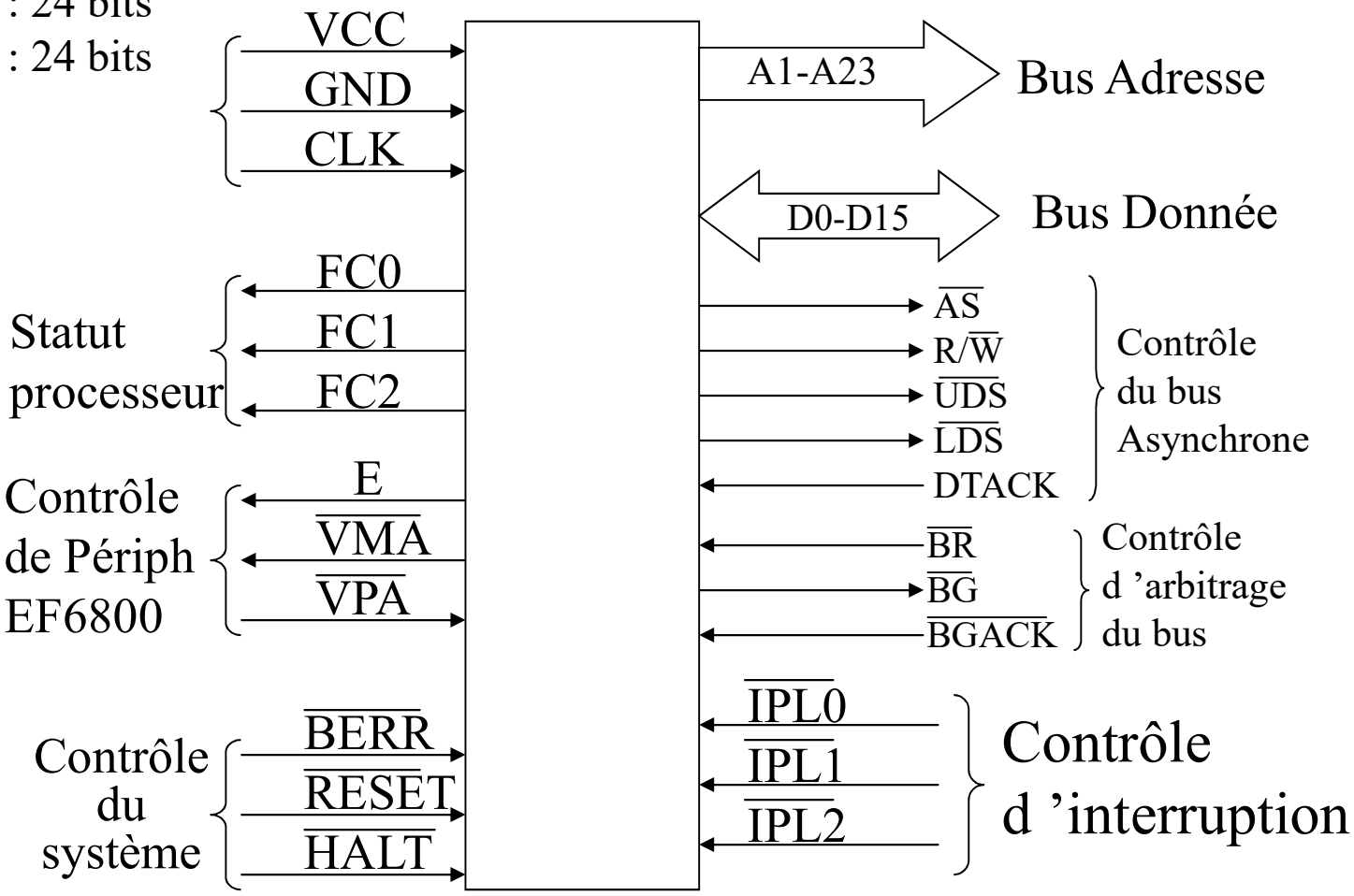
ASSEMBLEUR 68000

ENSIAS

- I - Organisation interne du MC68000
- II - Mémoire et périphériques
- III - Organisation des données dans la mémoire
- IV - Mode d'adressage
- V - Jeux d'instruction du MC68000
- VI - Sous programme et Manipulation de la pile
- VIII - ASMC68
- IX - Routines Systèmes (BIOS/RDOS)

SYSTEME MC 68000 : HARDWARE

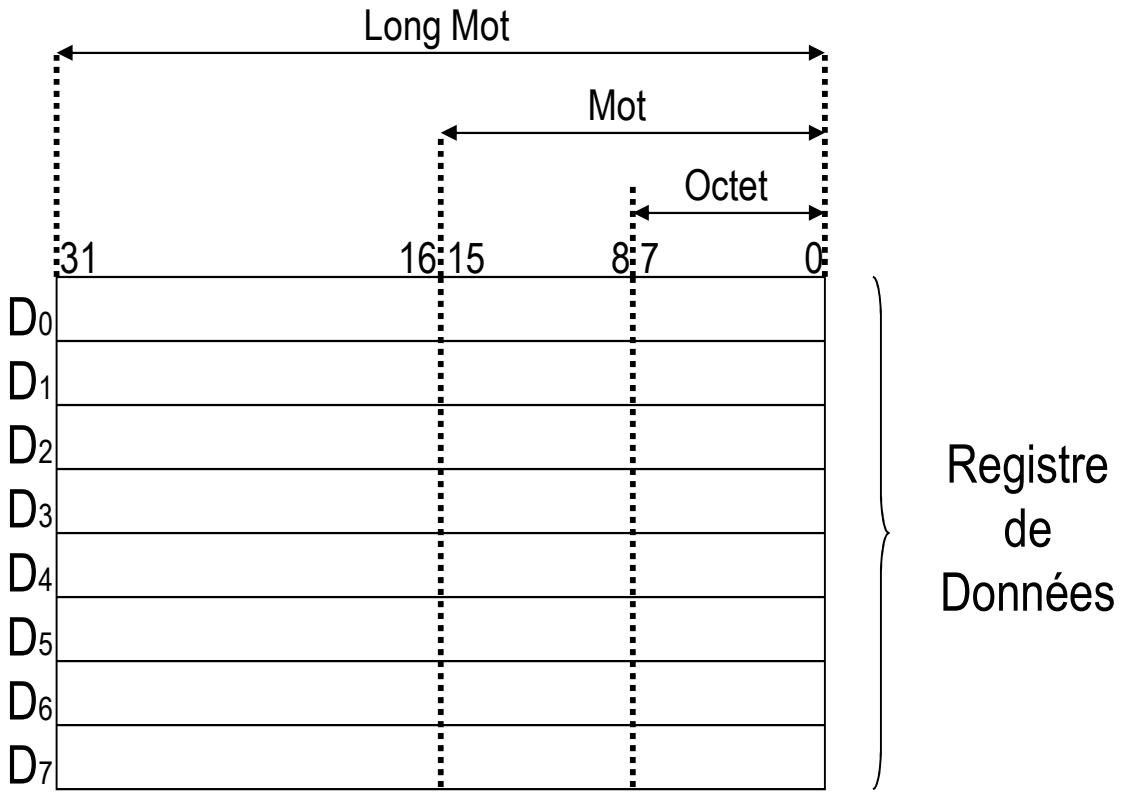
Bus de données : 16 bits
Bus d 'adresses : 24 bits
Bus de contrôle : 24 bits
Total de 64 Pins



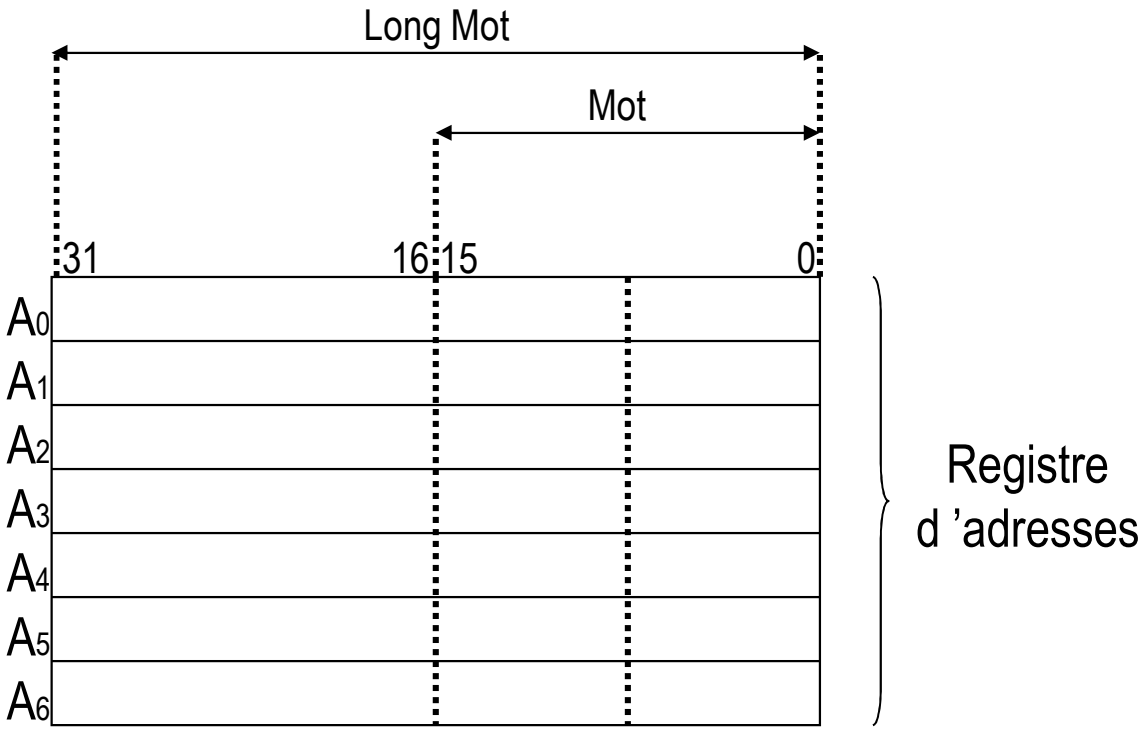
Organisation interne du 68000

- Structure des registres
  - 8 registres de données : D_0-D_7
  - 7 registres d ’adresses : A_0-A_6
  - pointeur de pile : SP
  - Compteur de Programme : PC
  - Registre d ’état : SR

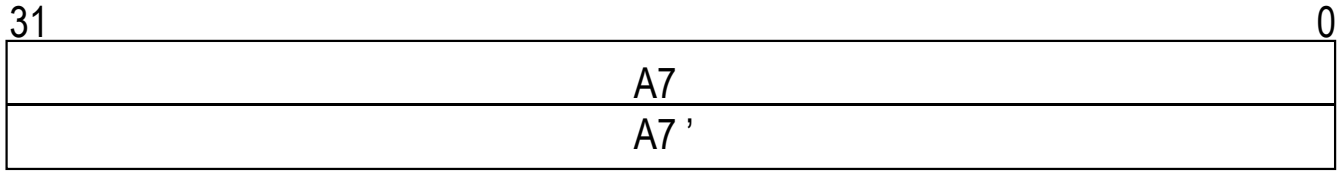
Registres de données



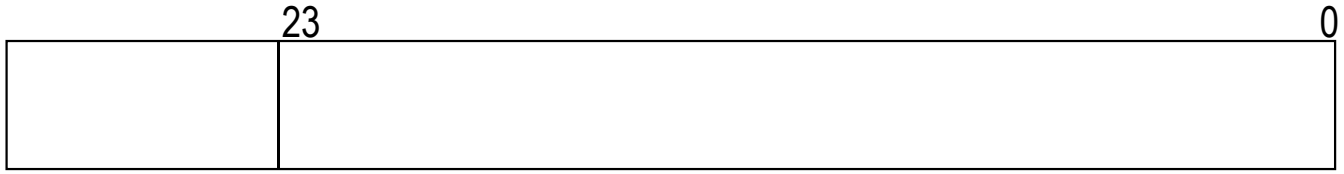
Registres d 'adresses



Les autres Registres



Pointeur de Pile : A7



Compteur de Programme :PC

T	0	S	0	0	I_2	I_1	I_0	0	0	0	X	N	Z	V	C

Registre d'état du 68000

ASSEMBLEUR 68000

ENSIAS

- I - Organisation interne du MC68000
- II - Mémoire et périphériques
- III - Organisation des données dans la mémoire
- IV - Mode d'adressage
- V - Jeux d'instruction du MC68000
- VI - Sous programme et Manipulation de la pile
- VIII - ASMC68
- IX - Routine Systèmes (BIOS/RDOS)

Mémoires et périphériques

- Le 68000 gère un bus d ’adresse de 24 bits
- Donc l ’espace adressable est de 16Mbyte.

ROM1	000000-00FFFF
ROM2	010000-01FFFF
Libre	020000-03FFFF
Libre VPA	040000-043FFF
PIA	044000-044FFF
Libre VPA	045000-04FFFF
DUART U	050000-05001F
DUART S	050020-05003F
MFP 68901	060000-06FFFF
PIT 68230	070000-07FFFF
Libre	080000-DFFFFF
RAM Moniteur	E00000-E0FFFF
Libre	E10000-FDFFFF
RAM1	FE0000-FEFFFF
RAM2	FF0000-FFFFFF

ASSEMBLEUR 68000

ENSIAS

- I - Organisation interne du MC68000
- II - Mémoire et périphériques
- III - Organisation des données dans la mémoire
- IV - Mode d'adressage
- V - Jeux d'instruction du MC68000
- VI - Sous programme et Manipulation de la pile
- VIII - ASMC68
- IX - Routine Systèmes (BIOS/RDOS)

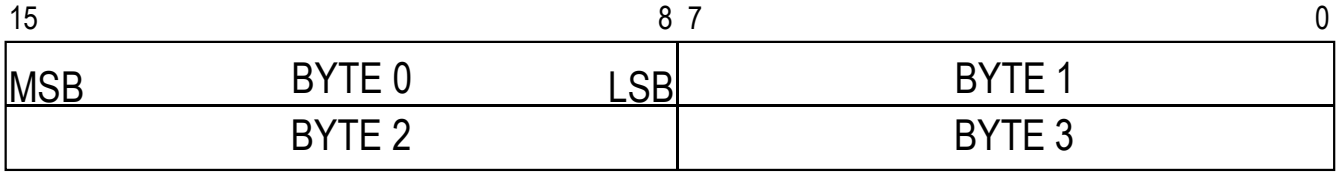
Organisation des données dans la mémoire

ENSIAS

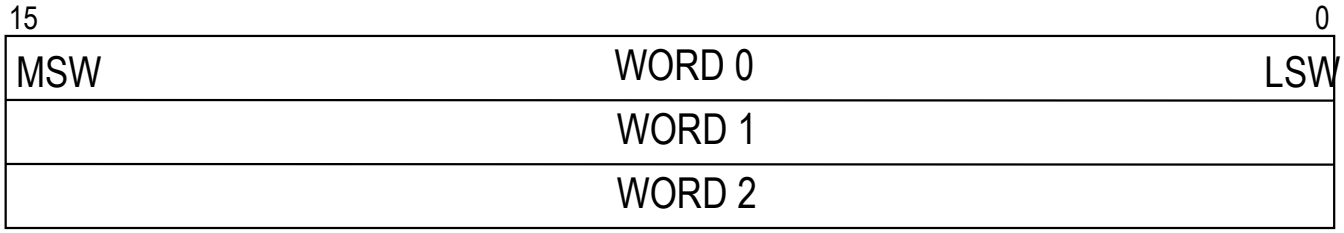
- Les données manipulées par le 68000 sont de 5 types :
  - le BIT
  - le DIGIT BCD (4 bits)
  - le BYTE (octet 8 bits)
  - le WORD (mot 16 bits)
  - le LONG WORD (long mot 32 bits)

Données :

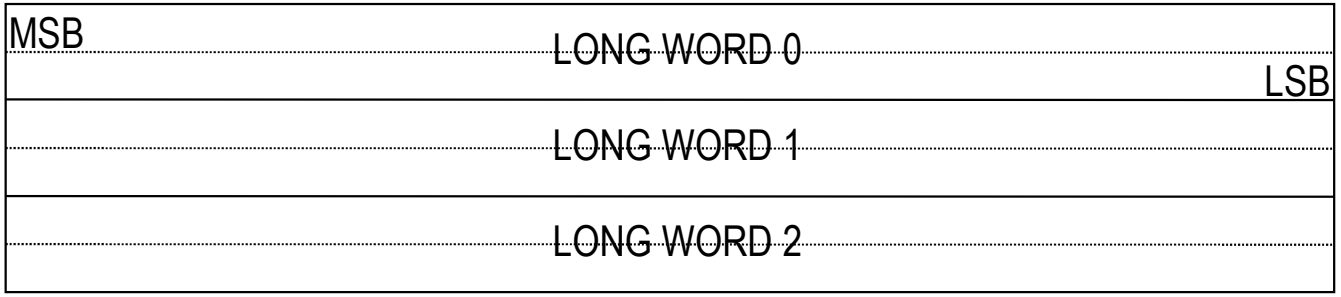
BYTE = 8 Bits



WORD = 16 Bits



LONG WORD
32 Bits



BCD 0	BCD 1	BCD 2	BCD 3
BCD 4	BCD 5	BCD 6	BCD 7

Instructions :

- les instructions 68000 opèrent sur le BYTE, le WORD et/ou le LONG WORD.
- Cette taille est précisée par les suffixes : B , W , L

- MOVE.B Src,Dst
  MOVE.W Src,Dst
  MOVE Src,Dst
  MOVE.L Src,Dst
  ADD Src,Dst

- Le code opération de toutes les instructions 68000 est fixé à 16 bits : **1 WORD**
- Des extensions sont exigées quand le mode D'ADRESSAGE spécifié utilise des :
  - constantes
  - adresses en absolues
  - Déplacements
  - Registres indexes
- $\Rightarrow$ 1 instruction 68000 $\Rightarrow$ 5 WORD (Maxi)

ASSEMBLEUR 68000

- I - Organisation interne du MC68000
- II - Mémoire et périphériques
- III - Organisation des données dans la mémoire
- IV - Mode d'adressage
- V - Jeux d'instruction du MC68000
- VI - Sous programme et Manipulation de la pile
- VIII - ASMC68
- IX - Routine Systèmes (BIOS/RDOS)

Mode d 'adressage

Définition

- On appel « mode d 'adressage » les différents moyens que possède une instruction pour accéder à une donnée afin d 'y effectuer un traitement.
- Taille d 'une instruction : 1 à 5 Words
  2 à 10 Octet
- Instruction à : 0,1 ou 2 opérandes

Mode d 'adressage

Le Microprocesseur 68000 possède 14 modes d 'adressage regroupés en six catégories :

- Adressage registre direct
- Adressage registre indirect
- Adressage Absolu
- Adressage immédiat
- Adressage Relatif/compteur de programme
- Adressage Implicite

Notations :

- $: donnée en Hexa
- # : donnée constante immédiate
- EA : Adresse effective

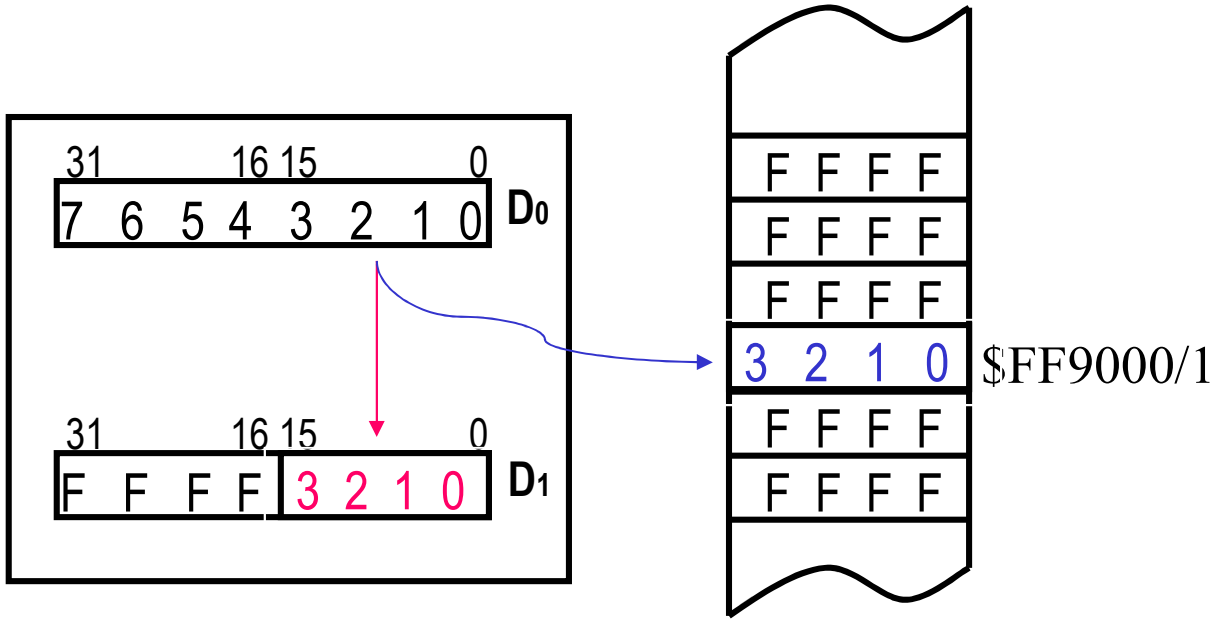
Adressage registre direct : 2 types

1- Adressage direct registre de données AE=Dn

MOVE.W D0,D1

MOVE.W D0,$0FF9000

Taille : B,W,L



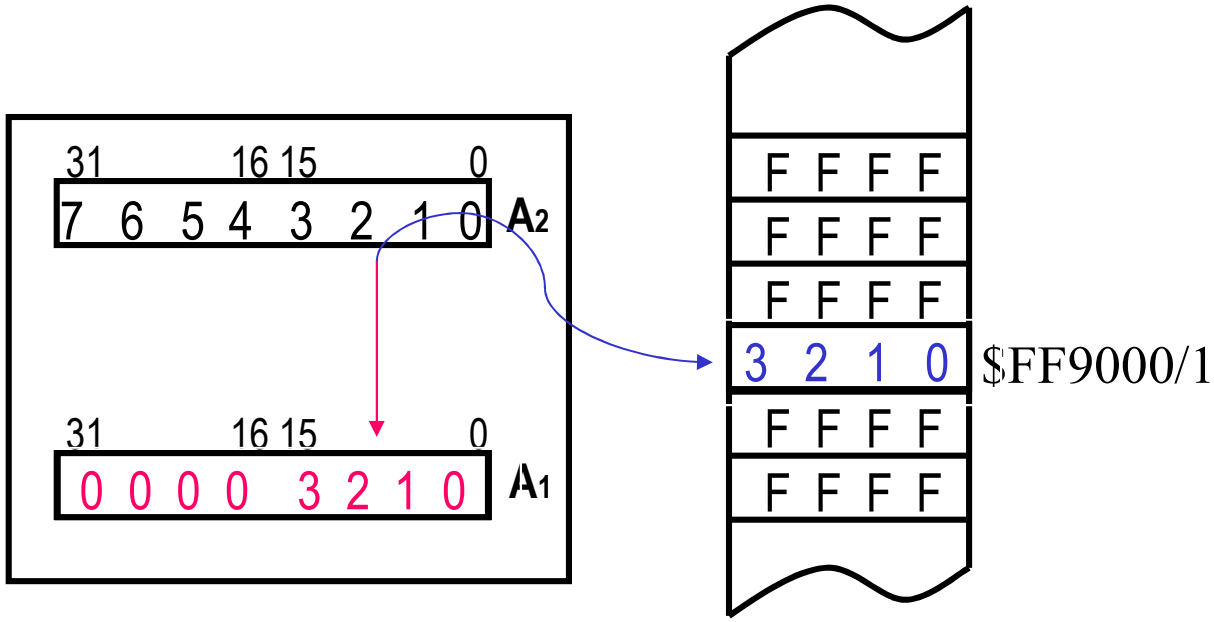
Adressage registre direct

2- Adressage direct registre d'adresses AE=An

MOVE.W A2,A1

MOVE A2,$0FF9000

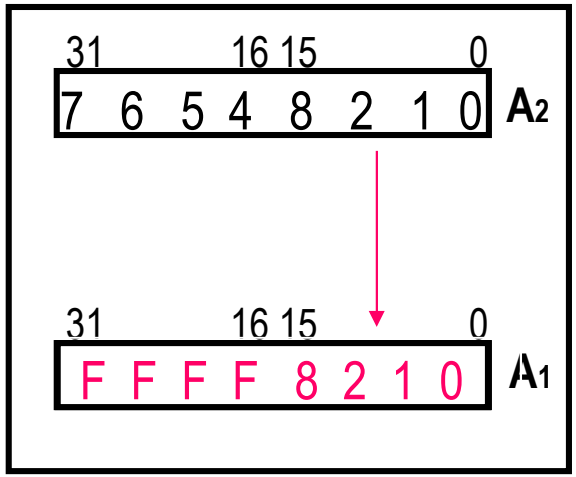
Taille : W,L



Extension du signe

2- Adressage direct registre d'adresses AE=An

MOVE.W A_2,A_1



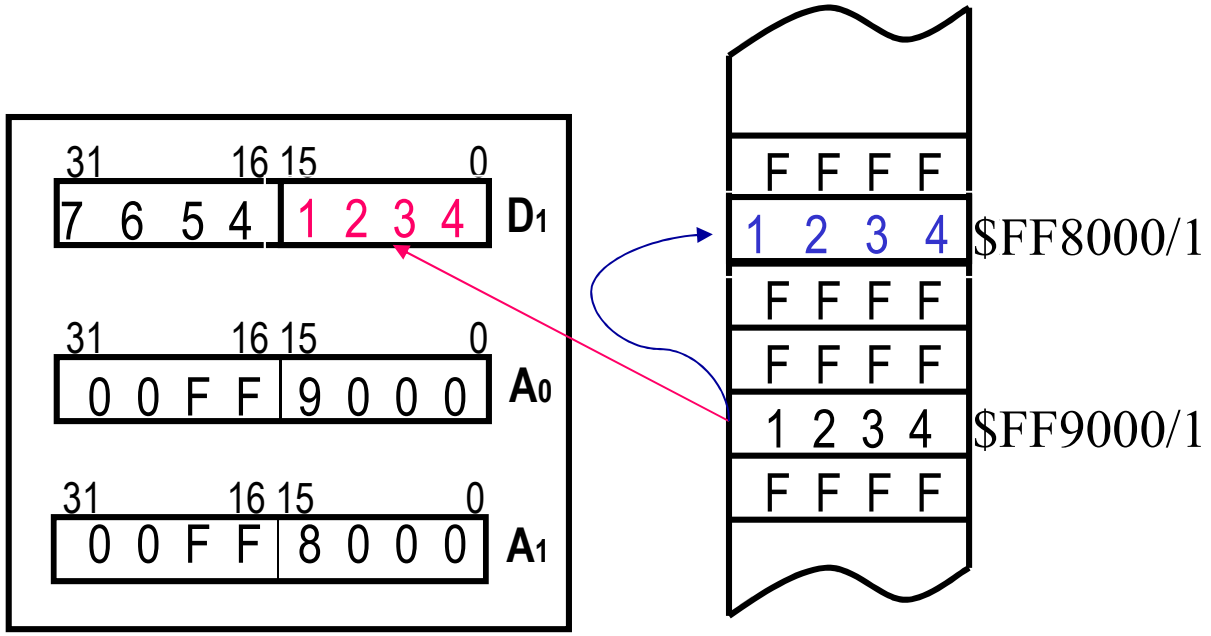
Adressage indirect : 6 types

1- Adressage registre indirect : AE = (An)

MOVE (A0),D1

MOVE.W (A0),(A1)

Taille : B,W,L



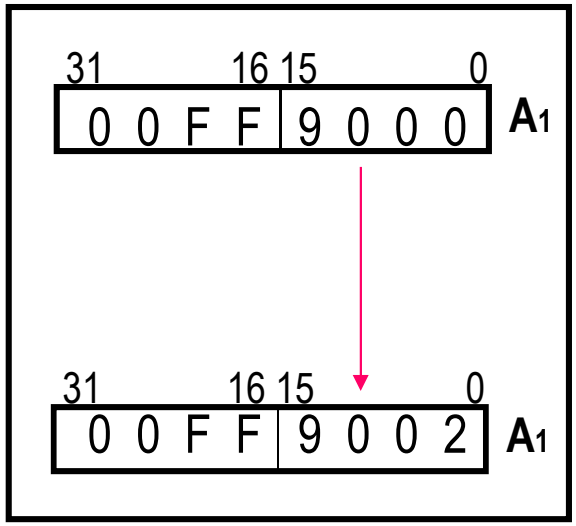
Adressage indirect : 6 types

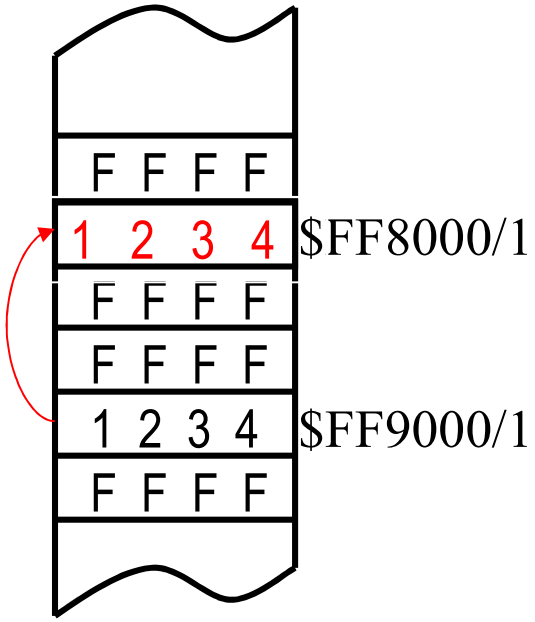
2- Adressage registre indirect postincrémentation :

AE = (An)+

MOVE (A1)+,$0FF8000

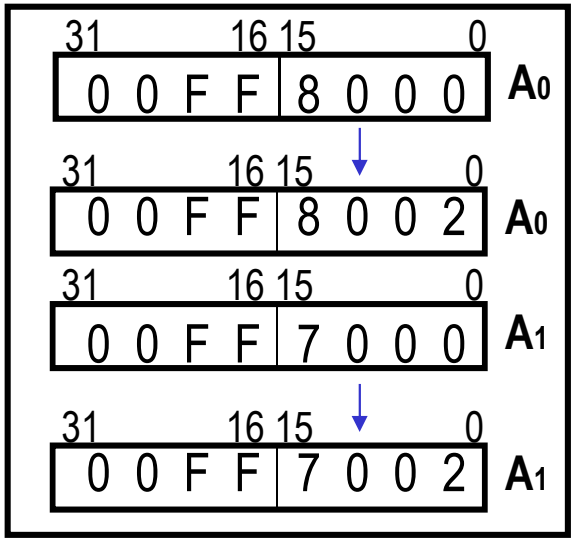
An=An+N
N = 1 si .B
= 2 si .W
= 4 si .L

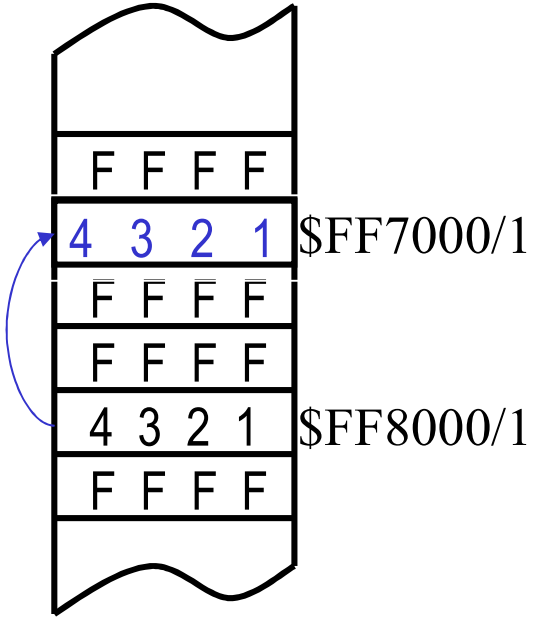




Exemple 2 postincrémentation

- MOVE.W (A0)+,(A1)+





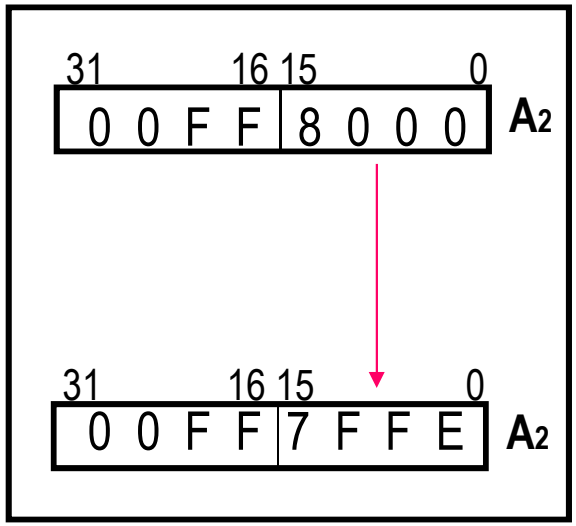
Adressage indirect : 6 types

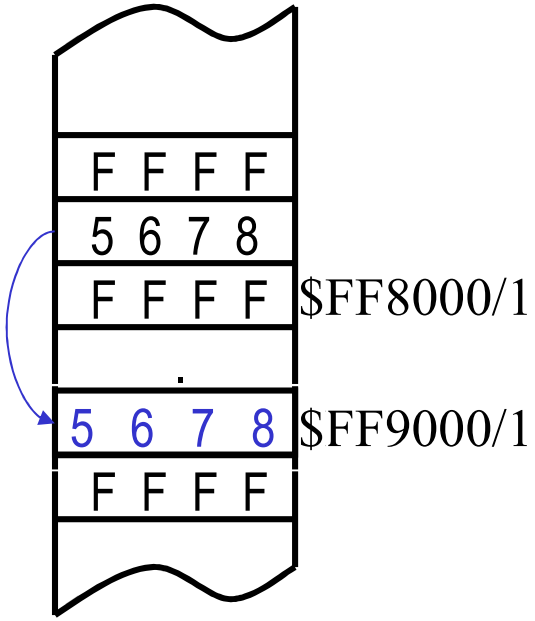
3- Adressage registre indirect prédecrémentation :

AE = -(An)

MOVE -(A2),$0FF9000

An=An-N
N = 1 si .B
= 2 si .W
= 4 si .L





Adressage indirect : 6 types

4- Adressage indirect avec déplacement :

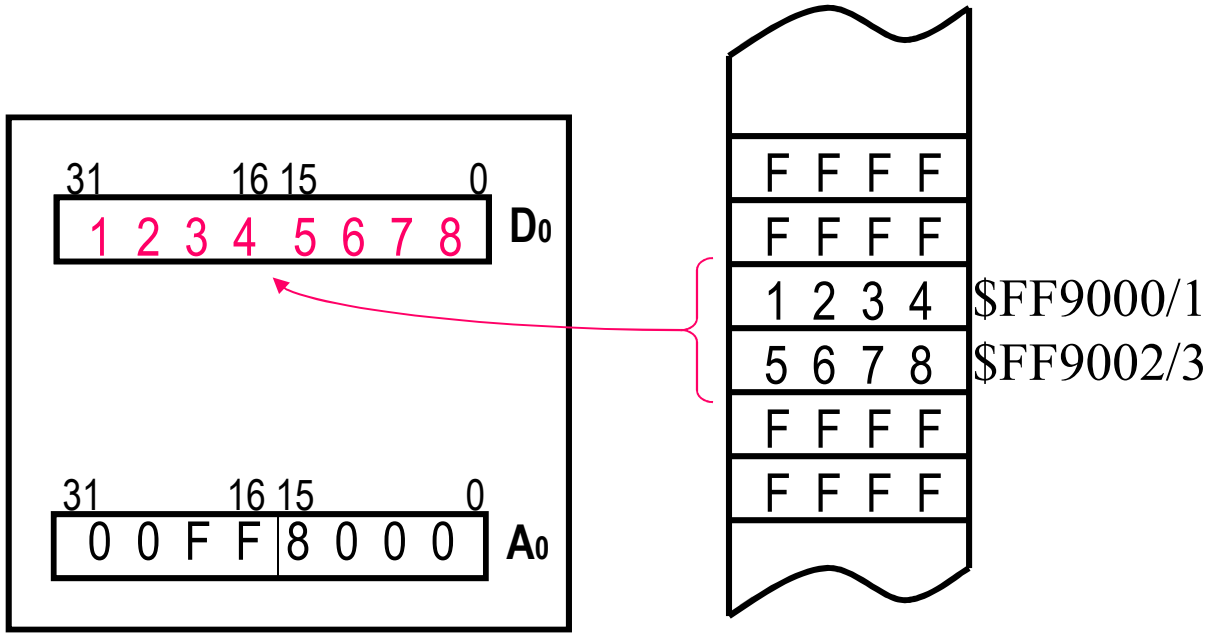
AE = (An)+d16

MOVE.L $1000(A0),D0

00FF8000
00001000

00FF9000

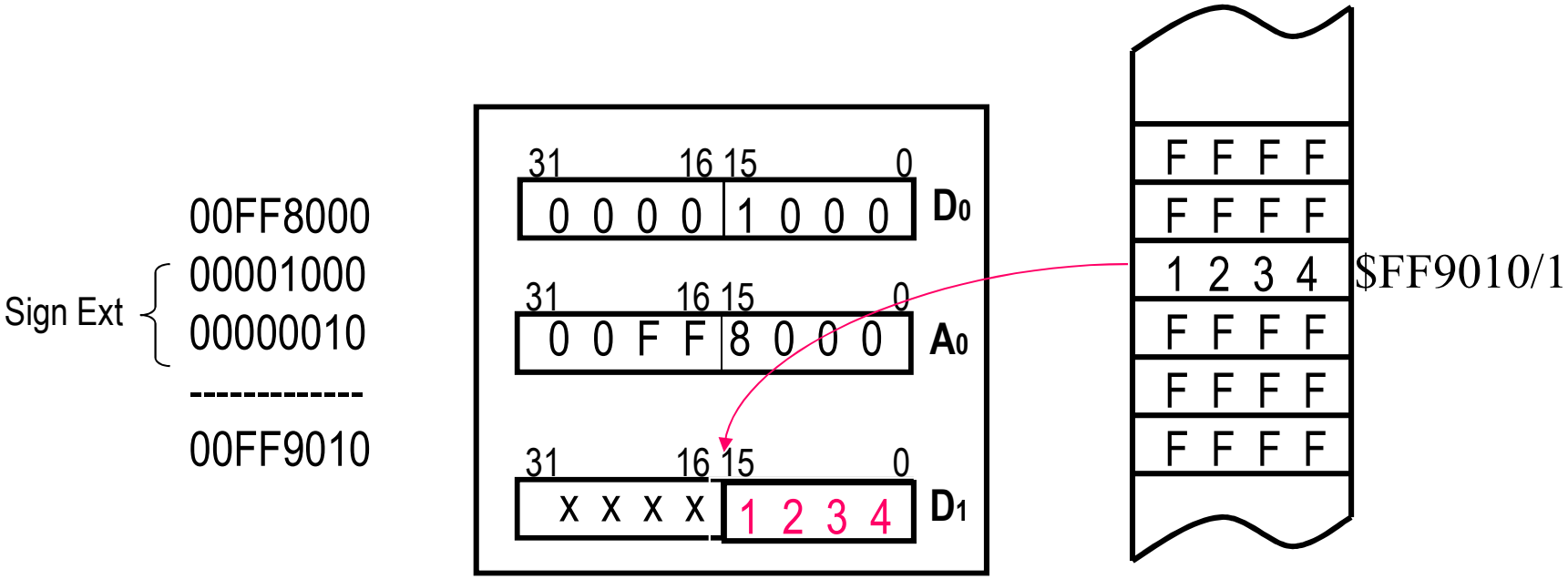
Taille : B,W,L



Adressage indirect : 6 types

5- Adressage indirect avec déplacement sur 8bits signés et indexe court : AE = (An)+(Xi.W)+d8

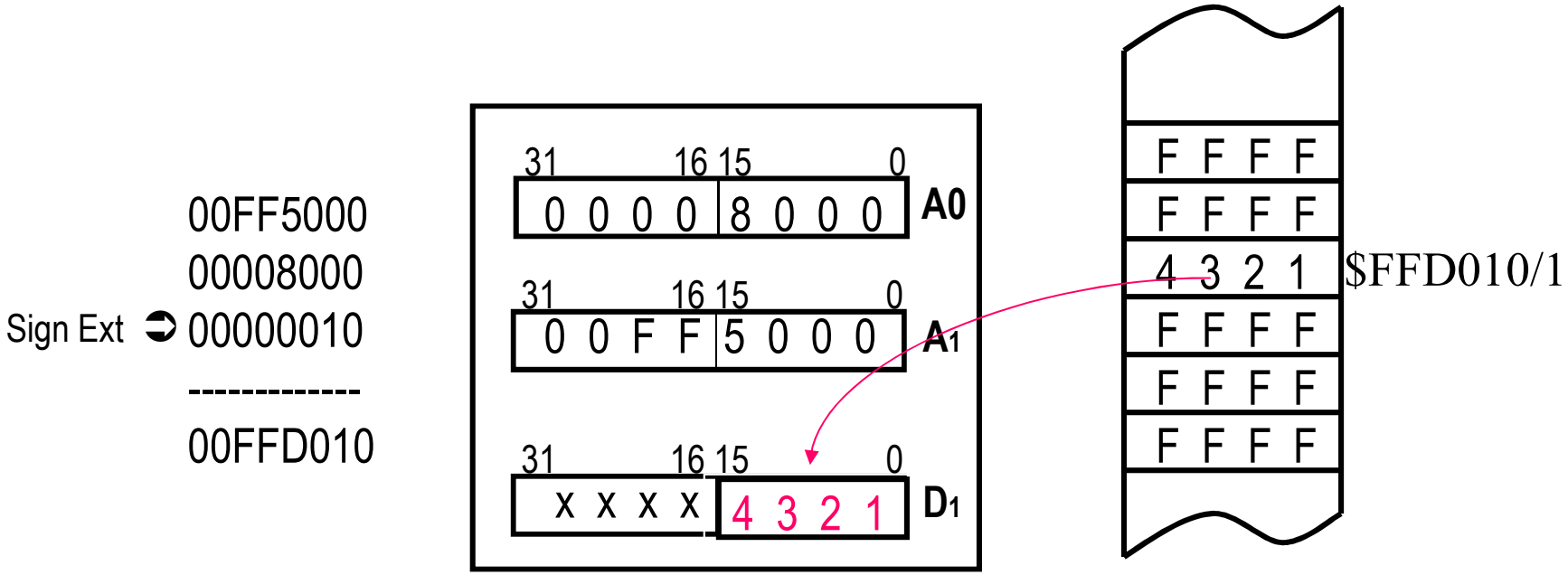
MOVE.W $10($A_0$,$D_0$.W),$D_1$



Adressage indirect : 6 types

6- Adressage indirect avec déplacement sur 8bits signés et indexe long : AE = (An)+(Xi.L)+d8

MOVE.W $10($A_1$,$A_0$.L),$D_1$

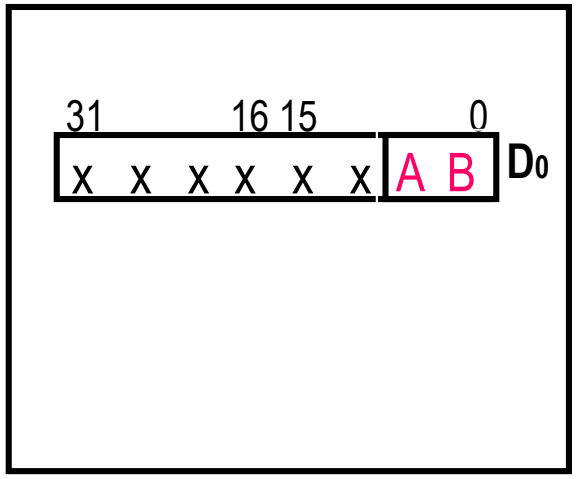


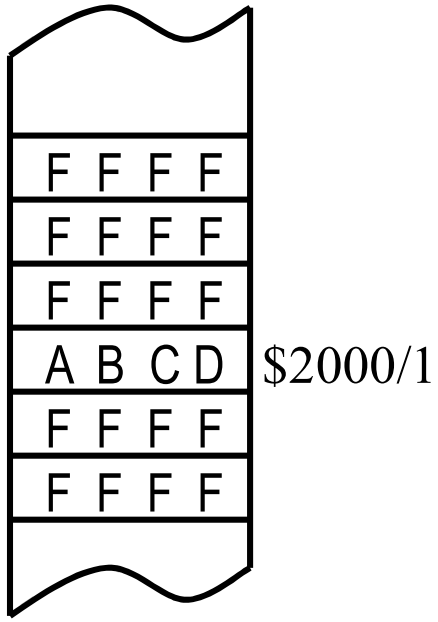
Adressage Absolu : 2 types

1- Adressage Absolu court : AE=Adresse sur 16bits

MOVE.B $2000,D0

Taille : B,W,L

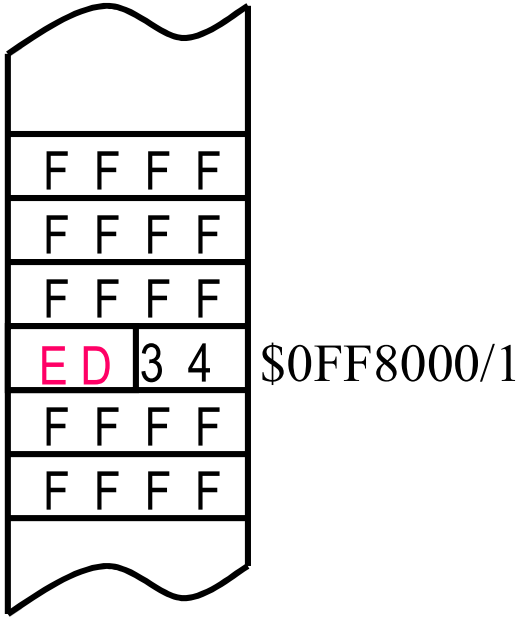




Adressage Absolu : 2 types

2- Adressage Absolu long : AE = Adresse sur 24bits

NOT.B $0FF8000



Adressage Immédiat

Trois destinations :

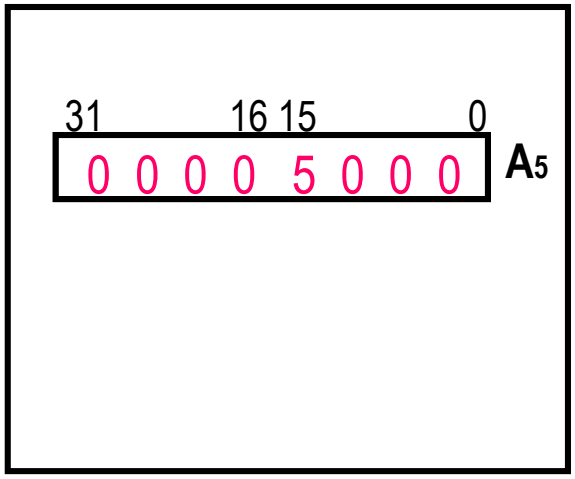
- registre adresse
- registre donnée
- mémoire

Adressage Immédiat

1-Destination Registre adresse

MOVEA.W #$5000,A5

Taille : W,L

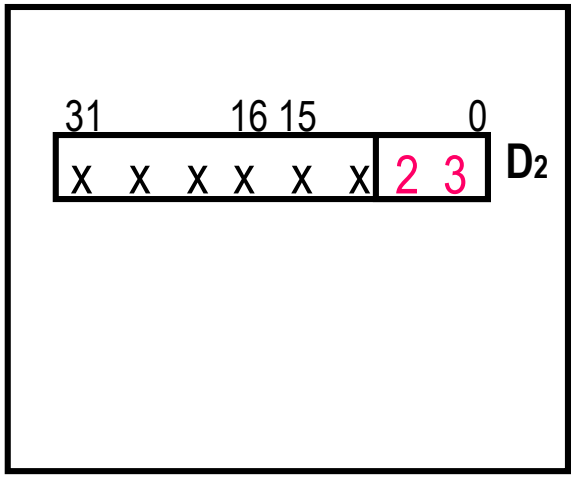


Adressage Immédiat

2- Destination Registre Données

MOVE.B #$23,D2

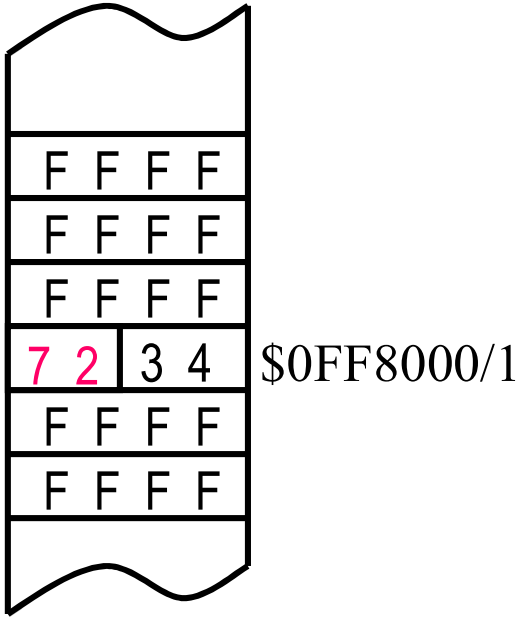
Taille : B,W,L



Adressage Immédiat

3-Destination Mémoire

ADDI.B #$60,$0FF8000



Adressage Immédiat : cas particulier

ENSIAS

- Adressage immédiat rapide

  MOVEQ, ADDQ, SUBQ
- Conditions :
  - destination toujours un Dn
  - taille toujours .B
  - extension signe sur 32 Bits
  - Pour ADDQ&SUBQ : cte codée sur 3 bits $1 \subseteq$ Cte$\subseteq 8$

```
MOVEQ     #$58,D1
```

Adressage relatif/PC : 2 types

1- Adressage/PC avec déplacement :

AE=(PC)+d16

se calcul par rapport au PC+2

LEA Table(PC),A_0

2- Adressage/PC avec indexe:

AE=(PC)+(Xn)+d8

LEA D8(PC,Xi.W),A_0

LEA D8(PC,Xi.L),A_0

Adressage Implicite

- Ce mode correspond aux instructions qui font « implicitement » référence aux :
  - Compteur de Programme (JMP, BRA, JSR, BSR)
  - Pointeur de Pile (MOVE USP, PEA, JSR, BSR)
  - Pointeur de Pile Superviseur (TRAP)
  - Registre d ’état (RTR, RTE)

ASSEMBLEUR 68000

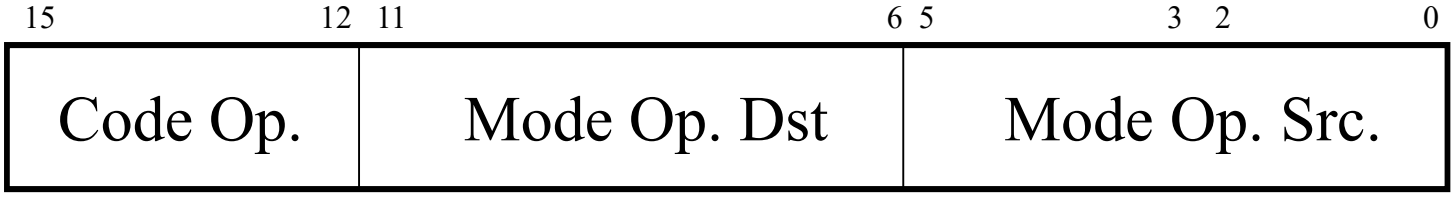
ENSIAS

- I - Organisation interne du MC68000
- II - Mémoire et périphériques
- III - Organisation des données dans la mémoire
- IV - Mode d'adressage
- V - Jeux d'instruction du MC68000
- VI - Sous programme et Manipulation de la pile
- VIII - ASMC68
- IX - Routine Systèmes (BIOS/RDOS)

JEU D INSTRUCTIONS DU 68000

ENSIAS

- Une instruction comporte de 1 à 5 mots .
- La majorité des instructions possède le format suivant pour le 1er mot :
  - Code Opération, Opérande Destination, Opérande Source



JEU D INSTRUCTIONS DU 68000

Code opération

- 0000- instruction Manip de bit
- 0001- instruction transfert .B
- 0010- instruction transfert .L
- 0011- instruction transfert.W
- 0100- divers
- 0101-ADDQ/SUBQ/DBcc
- 0110-Bcc/BSR
- 0111-MOVEQ
- 1000-OR/DIV/SBCD
- 1001-SUB/SUBX
- 1010-reserve
- 1011-CMP/EOR
- 1100-AND/MUL/ABCD/EXG
- 1101-ADD/ADDX
- 1110-Décalages/Rotations
- 1111-reserve

- ⇒ **5 classes d'instructions :**
  - instructions de transfert de données
  - instructions arithmétiques et logiques
  - instructions de décalage et de rotation
  - instructions de manipulation de bits
  - instructions de contrôle de programme

Instructions de transfert de données:

- - MOVE
  - MOVEQ
  - MOVEM
  - MOVEP
  - EXG
  - LEA
  - PEA
  - SWAP
  - LINK
  - UNLK

- MOVE (Src) → (Dst)

  Syntaxe : MOVE <AE >, <AE>

  Transfert de :

  – registre à registre
  – registre à Mémoire
  – Mémoire à registre
  – Mémoire à mémoire

Instructions de transfert de données:

- Mode d 'adressage :

  SRC → tous les modes
  DST → tous les modes
  sauf ceux relatifs
  au PC

- Formes particulières du MOVE :

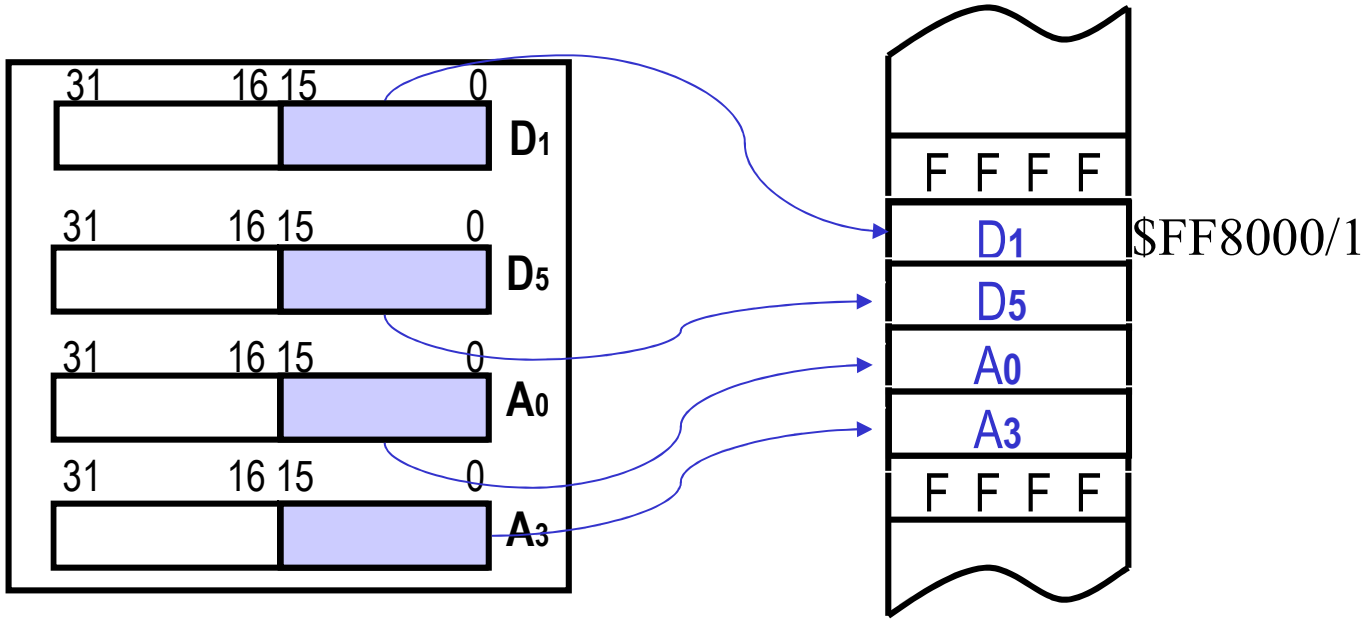
- MOVEA	src,An	(.W.L)
- MOVEQ	src,Dn	.B
- MOVEM		(.W.L)
- MOVEP		(.W.L)

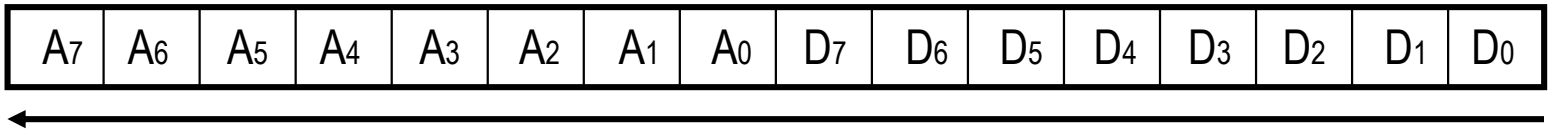
Exemple : Instruction MOVEM

- Transfert dans un ordre prédéterminé d'une liste de registre d'adresse et/ou de donnée vers un bloc mémoire.
- MOVEM <liste de registre>,<AE>
- MOVEM <AE>, <liste de registre>
- Taille : W, L

MOVEM : exemple 1

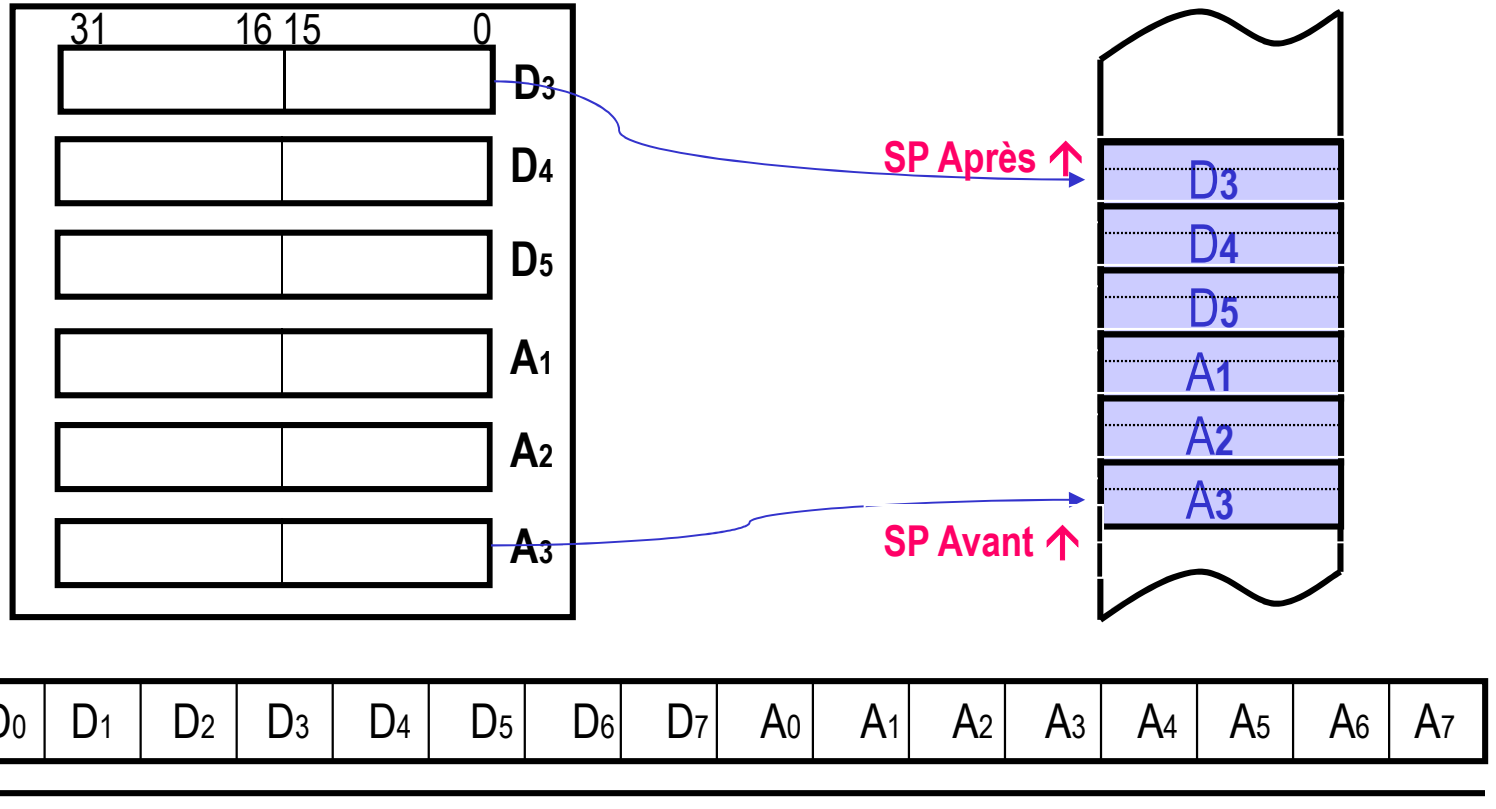
MOVEM.W A0/A3/D1/D5,$0FF8000





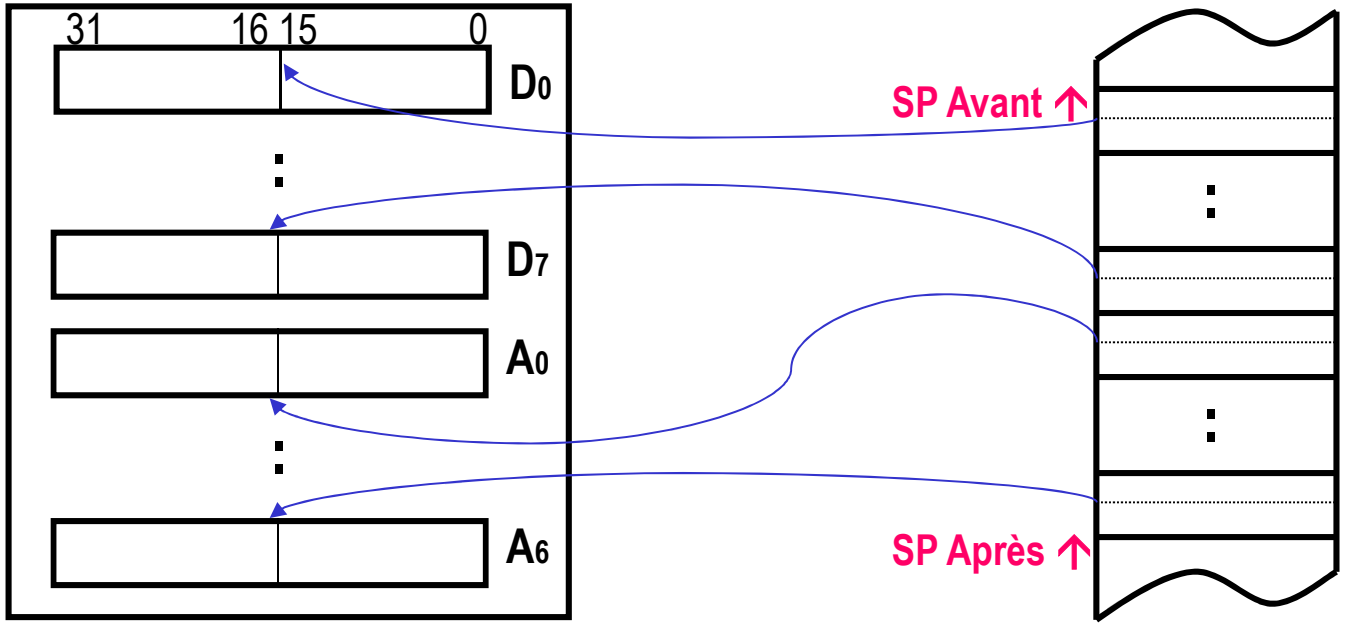
MOVEM : exemple 2

MOVEM.L A1-A3/D3-D5,-(A7)



MOVEM : exemple 3

MOVEM.L (SP)+,D0-D7/A0-A6



A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

Autres exemples :

- A1-A4/D3-D6 Signifie : A1,A2,A3,A4,D3,D4,D5,D6
- A2/A5/D1/D4 Signifie : A2,A5,D1,D4

- MOVEM.L D1/D4/A1-A3,TAB(A0,D6)
- MOVEM.L TAB(A0,D6),A1/A2/D1/D4/A3

Instructions arithmétiques et logiques

ENSIAS

- **Addition binaire**

ADD	(dst)+(src) ➔ dst	
ADD	< AE >,Dn	taille
	Dn,< AE >	(B ,W ,L)
ADDA	< AE >,An	(W,L)
ADDI	# cte,< AE >	(B ,W,L)
ADDQ	# cte,Dn	.B

cte codée sur 3 bits dans l'instruction elle même $1 \subseteq Cte \subseteq 8$

Attention : ADDQ #3,D2 ≠ ADDI.L #3,D2

ADDX	(dst)+(src)+X ➔ (dst)	
ADDX	Dx,Dy	(B ,W,L)
ADDX	-(Ax),-(Ay)	(W,L)

Instructions arithmétiques et logiques

ENSIAS

- **Soustraction binaire**

SUB	(dst)-(src) ➔ dst	
SUB	< AE >,Dn	taille
	Dn ,< AE >	(B ,W ,L)
SUBA	< AE >, An	(W,L)
SUBI	# cte ,< AE >	(B ,W,L)
SUBQ	# cte ,Dn	.B

cte codée sur 3 bits dans l'instruction elle même $1 \subseteq Cte \subseteq 8$

Attention : SUBQ #3,D2 ≠ SUBI.L #3,D2

SUBX	(dst)-(src)-X ➔ (dst)	
SUBX	Dx,Dy	(B ,W,L)
SUBX	-(Ax),-(Ay)	(W,L)

Instructions arithmétiques et logiques

Multiplication

- MULU < AE >,Dn
- MULS < AE >,Dn

Dn*< AE > ➔ Dn

16*16 ➔ 32

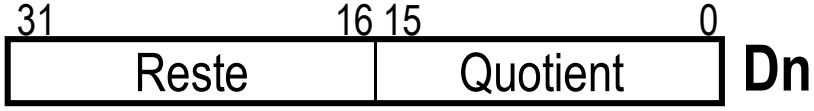
Division

- DIVU < AE >,Dn
- DIVS < AE >,Dn

Dn / < AE > ➔ Dn

32 / 16 ➔ 32

Tel que :



Instructions arithmétiques et logiques

- **<u>Addition Décimale avec retenue</u>**

  ABCD Dx,Dy

  ABCD -(Ax),-(Ay)

  $(dst)_{10}+(src)_{10}+X$ ➔ dst

  Taille .B

- **<u>Soustraction Décimale avec retenue</u>**

  SBCD Dx,Dy

  SBCD -(Ax),-(Ay)

  $(dst)_{10}-(src)_{10}-X$ ➔ dst

  Taille .B

Instructions arithmétiques et logiques

ENSIAS

- **Comparaison**

CMP <AE>,Dn Taille : B,W,L

(dst)-(src) : les indicateurs du registre d ’état traduisent le résultat de la soustraction,

X	N	Z	V	C
	*	*	*	*

Formes Particulières :

- CMPA <AE>,An (W,L)
- CMPI #Cte,<AE> (B,W,L)
- CMPM (Ax)+,(Ay)+(B,W,L)

Instructions arithmétiques et logiques

- CLR <AE> remise à zéro (B,W,L)
- NOT <AE> Complément à 1 (B,W,L)
- NEG <AE> Négation (complément à 2)
  0-(dst) ➔ (dst)(B,W,L)
  NEGX : 0-(dst)-X ➔ (dst)
- NBCD <AE> Complément à 10 (B)
- TAS <AE> (B) section critique semaphore
  - Multiprocesseur
  - Mémoire commune
  - Périphérique commun à plusieurs processeurs

Instructions arithmétiques et logiques

- AND <AE>,Dn (src).ET.(dst) ➔ (dst)
  Dn,<AE> (B,W,L)
  Forme particulière : ANDI #Cte,<AE> (B,W,L)
- OR <AE>,Dn (src).OU.(dst) ➔ (dst)
  Dn,<AE> (B,W,L)
  Forme particulière : ORI #Cte,<AE> (B,W,L)
- EOR <AE>,Dn (src).⊕.(dst) ➔ (dst)
  Dn,<AE> (B,W,L)
  Forme particulière : EORI #Cte,<AE> (B,W,L)

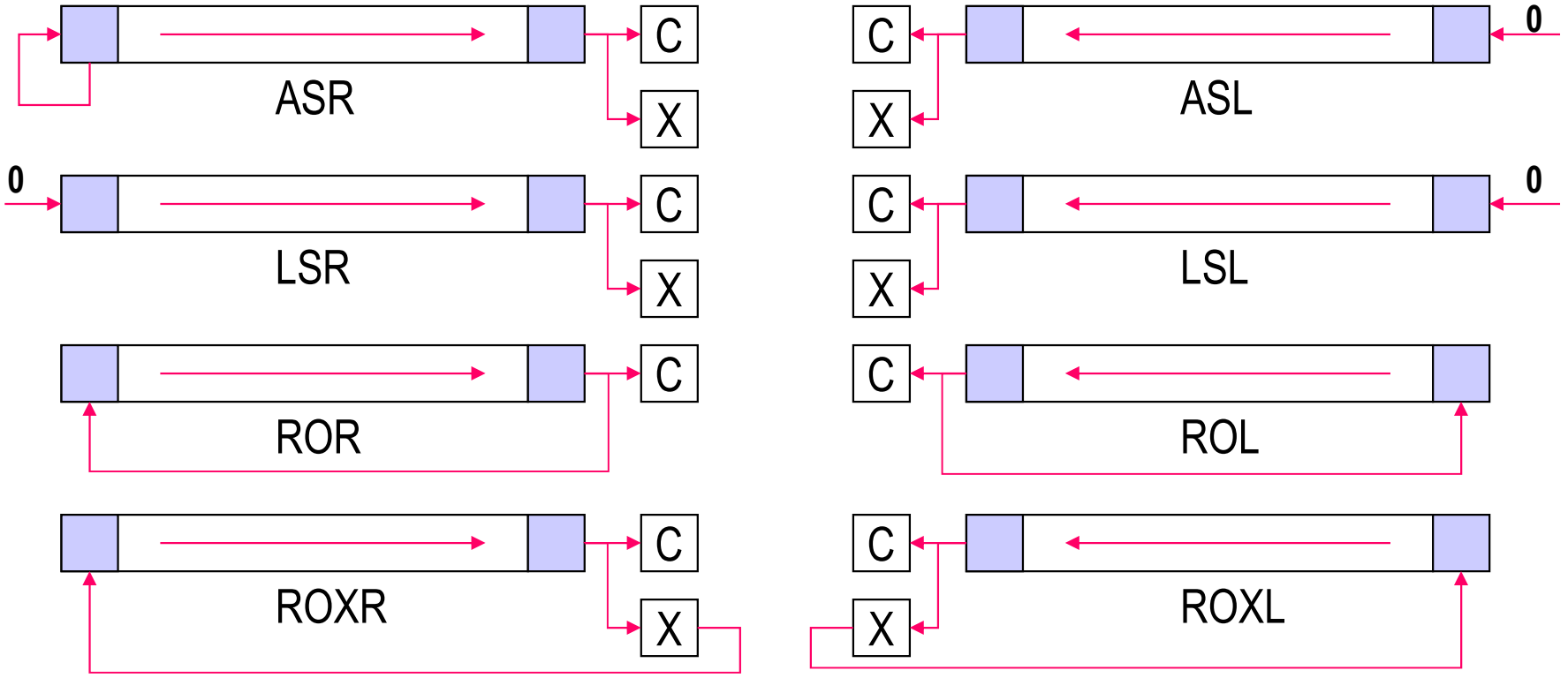
Applications :

- ANDI Permet de forcer un bit à ZERO
- ORI Permet de forcer un bit à UN
- EORI Permet de complémenter un BIT

Instructions arithmétiques et logiques

ENSIAS

- Décalages et rotation



Instructions arithmétiques et logiques

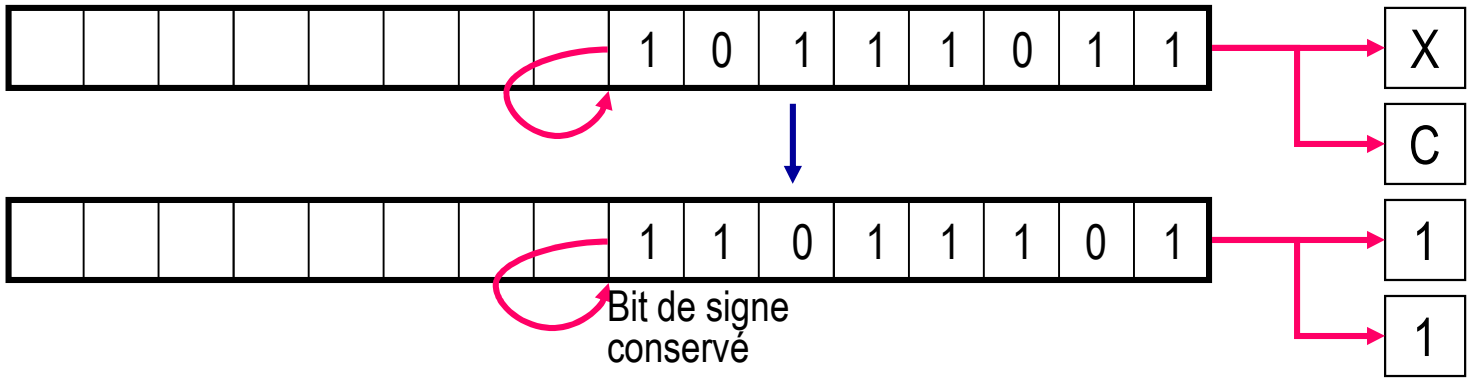
ENSIAS

- Décalages et rotation :
  - Statique,
  - Dynamique
  - Memoire
- Statique

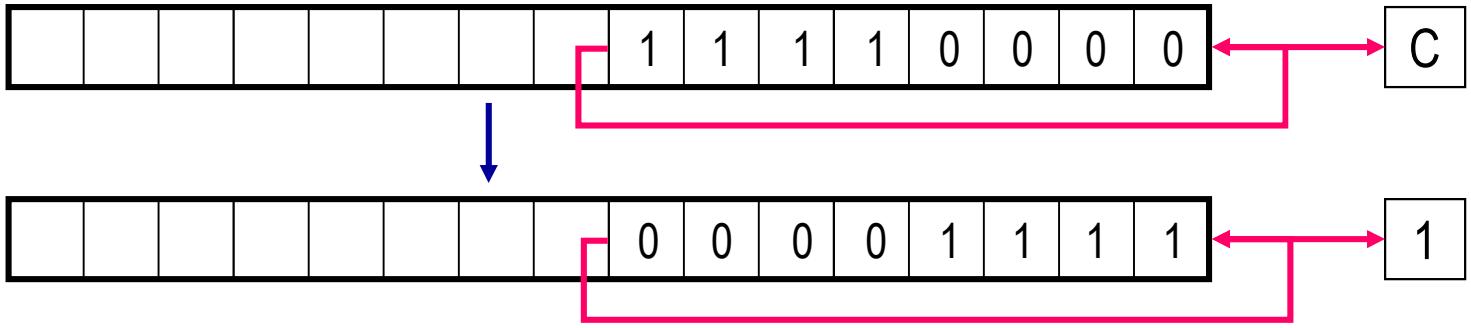
  OP.x #Cte,Dn
  - Cte : nombre de décalage (1à 8 au maximum)
  - Dn : registre de destination (donnée à décaler)
  - x : taille, B,W,L

Décalage statique

Ex 1: ASR.B #1,D0



Ex 2 : ROL.B #4,D0



Instructions arithmétiques et logiques

ENSIAS

- Décalages et rotation dynamique

  OP.x Dn,Dm

  Le total de rotation ou de décalage du contenu de Dm est spécifié par les 6 bits de poids faible du registre Dn.

  – Dm : registre a décaler
  – Dn : contient le nombre de décalage
  – x : taille, B,W,L

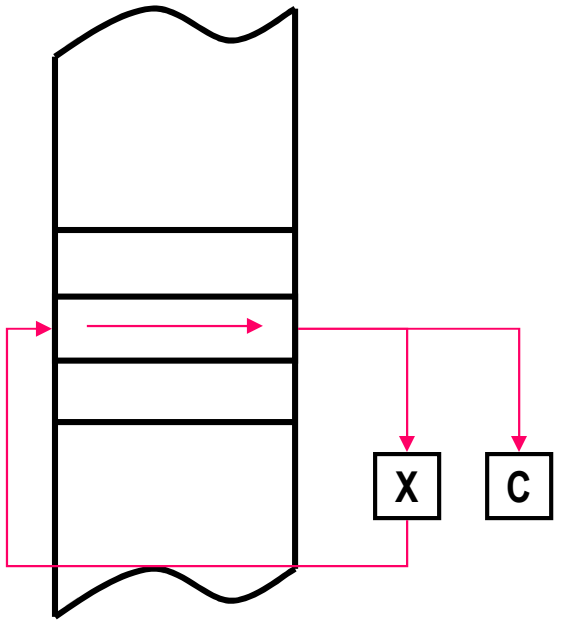
Instructions arithmétiques et logiques

- Décalages et rotation : position mémoire

OP.W <AE>

- Le nombre de traitement est toujours égale à 1

Ex : ROXR.W MEM



Instruction de Manipulation de bits

- BTST Si bit=0 Z=1, si bit=1 Z=0
- BSET positionne un bit à 1
- BCLR positionne un bit à 0
- BCHG Complémente un bit

  OP #Cte,<AE> Cte : N° du bit

  OP Dn,<AE> AE : An exclut

Instruction de contrôle de programme

- Deux types d'instructions :
- Conditionnelles:
  - Bcc
  - DBcc
- Inconditionnelles :
  - BRA
  - BSR
  - JMP
  - JSR
  - RTS

Branchements Conditionnels

- Les branchements conditionnels se font selon le résultat des conditions sur les bits du CCR
- Ils sont utilisées après une instruction de comparaison ou de test.

  Bcc <Etiquette>

  cc : correspond a 14 conditions

Branchements Conditionnels

BCC :	Carry Clear (C=0)	
BCS :	Carry Set (C=1)	
BEQ :	Equal (Z=1)	
BNE :	Not Equal (Z=0)	
BPL :	Positif (N=0)	
BMI :	Négatif(N=1)	
BVC :	Overflow Clear (V=0)	
BVS :	Overflow Set (V=1)	
BGT :	Greater Than > (Z+(N⊕V)=0)	(nombre signé)
BHI :	Higher > (C+Z)=0	(nombre non signé)
BGE :	Greater or Equal ⊇ N⊕V=0	(nombre signé)
BLT :	Less Than < N⊕V=1	(nombre signé)
BLE :	Less or Equal ⊆ (Z+(N⊕V)=1)	(nombre signé)
BLS :	Lower or Equal ⊆ (C+Z)=1	(nombre non signé)

Comment le µP calcule l '@ de brt ?

- Branchement Relatif
- Le déplacement Dp :
  - Si .s -128 ⊆ DP ⊆ 127
  - Si .l -32Ko ⊆ DP ⊆ +32Ko-1

$0FF7000 BRA LABEL

$0FF8000LABEL

- Le déplacement est calculé :

  $0FF8000-$0FF7000-2

Impact du signe sur le branchement

```
LABEL   .
        .
        CMP.B       #128,D0
        BGT         LABEL
```

- si $D_0 = 50$

Instruction DBcc

DBcc Dn,<LABEL>

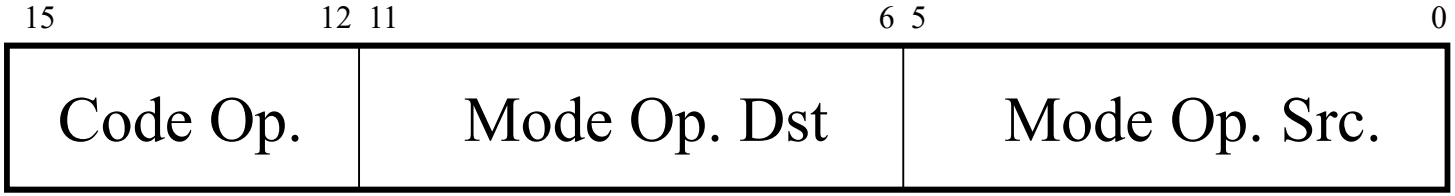
- Intérêt :
  - Condition d ’arrêt cc (même que Bcc)
  - Compteur 16 bits Dn (.W implicitement)
  - Déplacement <Label>, relatif +/-32Ko
- L’opération est la suivante :
  - Si condition cc est verifie : PC=PC+2
  - Si Condition cc non vérifié alors Dn=Dn-1
    - Si Dn ≠ -1 alors PC=PC+DP
    - Si Dn = -1 alors PC=PC+2

Exercices : 1ère série

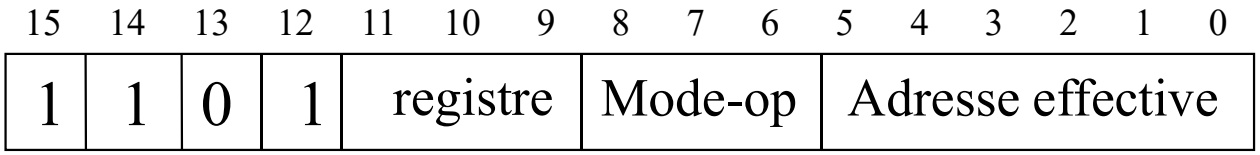
- Addition de 2 chiffres de 16 bits
- Addition 64 bits
- Recherche du nombre 20 dans un buffer de 30 chiffres
- Addition en BCD

Codage des Instructions

OP.x SRC,DST



Format de l 'instruction ADD



Champs de l'instruction

Champ registre -Spécifie un des huit registres de donnée.

Champ mode-op-

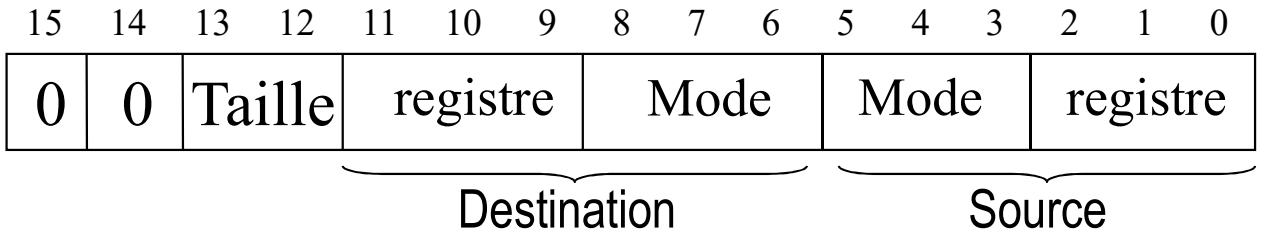
Octet	Mot	Long Mot	opération
000	001	010	(< Dn >) + (< ae >) ➔ (< Dn >)
100	101	110	(< ae >) + (< Dn >) ➔ (< ae >)

Champ adresse effective -Détermine le mode d'adressage :

a- Tous les modes d'adressage sont permis, lorsque l'emplacement spécifié est un opérande source.

b- Si l'emplacement spécifié est un opérande destination, seul l'adressage de mémoires modifiables est autorisé.

Format de l 'instruction MOVE



Champs de l'instruction

Champ Taille - spécifie la taille de l 'opérande devant être transféré :

01 - Opération sur Octet

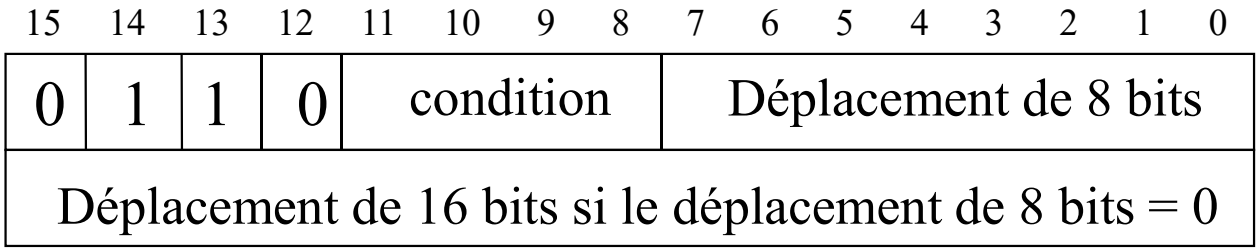
11 - Opération sur Mot

10 - Opération sur long mot

Champ Adresse Effective Destination : spécifie l 'emplacement de destination. Seules les adresses de données modifiables sont permis.

Champ Adresse Effective Source : spécifie l 'opérande source. Tous les modes d 'adressage sont permis. Si la taille de l 'opérande est l'Octet, l 'adressage direct des registres adresse n 'est pas permis.

Format de l 'instruction Bcc



Champs de l'instruction

Champ condition - spécifie une des quatorze conditions

Champ déplacement de 8 bits : entier spécifiant la distance relative entre l 'instruction de branchement et l 'instruction qui doit être exécutée si la condition est remplie.

Champ déplacement de 16 bits : permet un déplacement supérieur à 8 bits. Utilisé seulement si le déplacement de 8 bits et à 0

Extensions aux instructions

Format de l'adressage absolu

- L'instruction est codé sur 1 à 5 Words

- Exemples
  - MOVE D0,$2000 **31C0 2000**
  - MOVE D0,$FF8000 **33C0 00FF 8000**
  - MOVE $1000,$2000 **31F8 1000 2000**

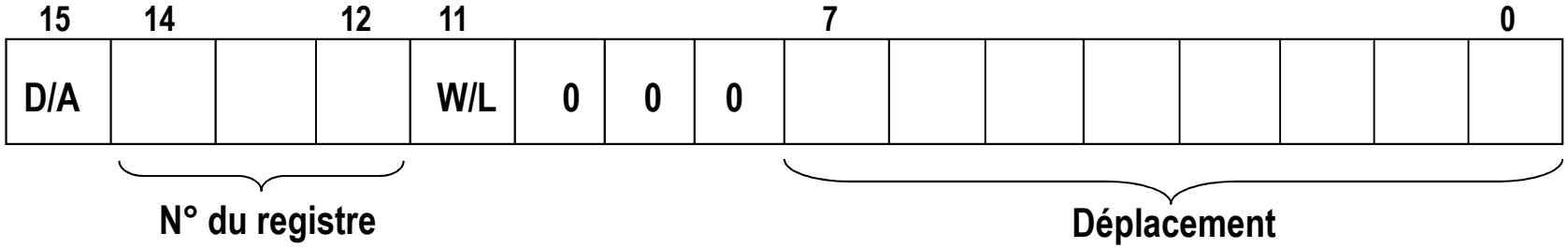
Format de l'adressage immédiat

- L'instruction est codé sur 1 à 5 Words

- Exemplesbb
  - MOVE #2,D0 **303C 0002**
  - MOVE.B #2,D0 **103C 0002**
  - MOVE #$05,$FF8000

    33FC 0005 00FF 8000

Format de l'adressage indexé



- Bit 15 :
  - D/A = 0 le registre d'indexe et un registre de donnée
  - D/A = 1 le registre d'indexe et un registre d'adresse
- Bit 11 :
  - W/L = 0 le registre d'indexe contient un mot dont le signe doit être étendu à 32 bits
  - W/L = 1 le registre d'indexe contient un long mot

Exemples adressage indexé

- MOVE #$03,(A0,D0) **31BC 0003 0000**

- ADD 5(A1,D1),D0 **D071 1005**

Format de l'adressage avec déplacement

- Indirect avec déplacement
  - MOVE $10(A0),$0FF8000

33E8 0010 00FF 8000

- Indirect indexé avec déplacement
  - CMP #$20,2(A0,D0)

0C70 0020 0002

Format de l'adressage relatif

- 3 cas de figures :
  - Branchement relatif
  - Adressage relatif au PC
  - Adressage indexé relatif au PC

Instruction de branchement relatif

- L'instruction est codé sur 1 à 2 Words
- Exemples

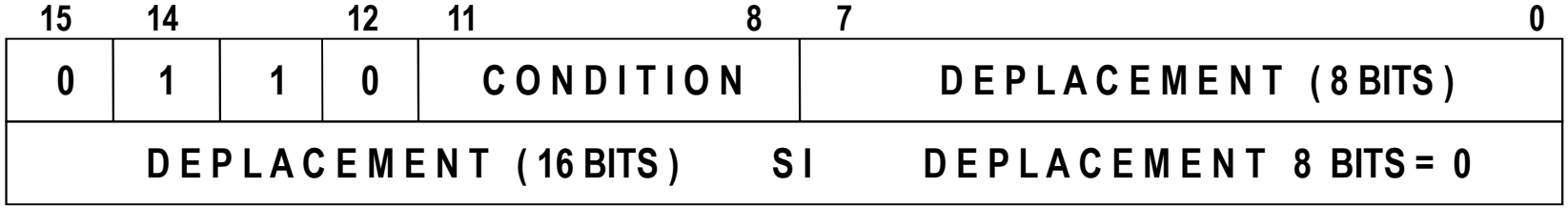
  $0FF7000 BRA LABEL

  :

  $0FF8000LABEL

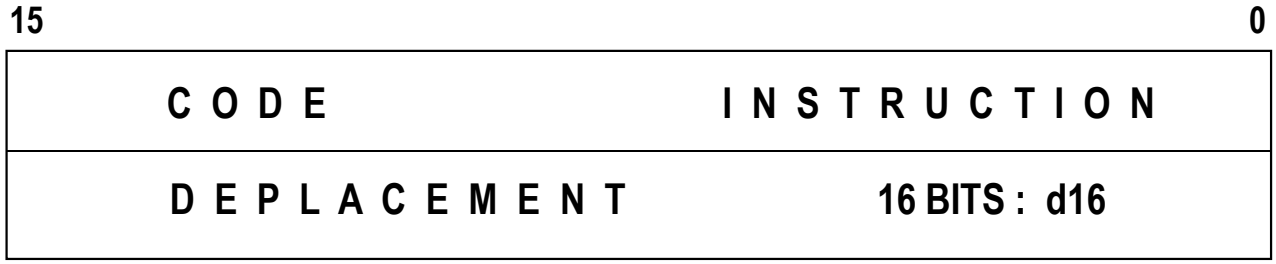
- Le déplacement est calculé :

  $0FF8000-$0FF7000-2 = $0FFE



6000 0FFE

Adressage relatif au PC & d16




```
$FF8000     MOVE            TAB(PC),D1
                            NOP
                            NOP
$FF8008                     NOP
$FF800A     TAB             DS          1
```

3 2 3 A 0 0 0 8

Adressage indexé relatif au PC & d8


```
$FF8000        LEA      TAB(PC,A0),A1
$FF8004        NOP
$FF8006        NOP
$FF8008        NOP
$FF800A        NOP
$FF800CTAB     DS       1

               43FB     800A
```

Format de l 'instruction MOVEM

Temps d'exécution des instructions

- Voir support

ASSEMBLEUR 68000

ENSIAS

- I - Organisation interne du MC68000
- II - Mémoire et périphériques
- III - Organisation des données dans la mémoire
- IV - Mode d'adressage
- V - Jeux d'instruction du MC68000
- ⇨ VI - Sous programme et Manipulation de la pile
- VIII - ASMC68
- IX - Routine Systèmes (BIOS/RDOS)

Sous programme

Définition :

un sous programme correspond a une partie du traitement qui se répètent fréquemment

Un sous programme (Subroutine) est un petit programme il peut être utilisé dans un ou plusieurs programmes, et peut être appelé à partir d ’un ou plusieurs points d ’un même programme.

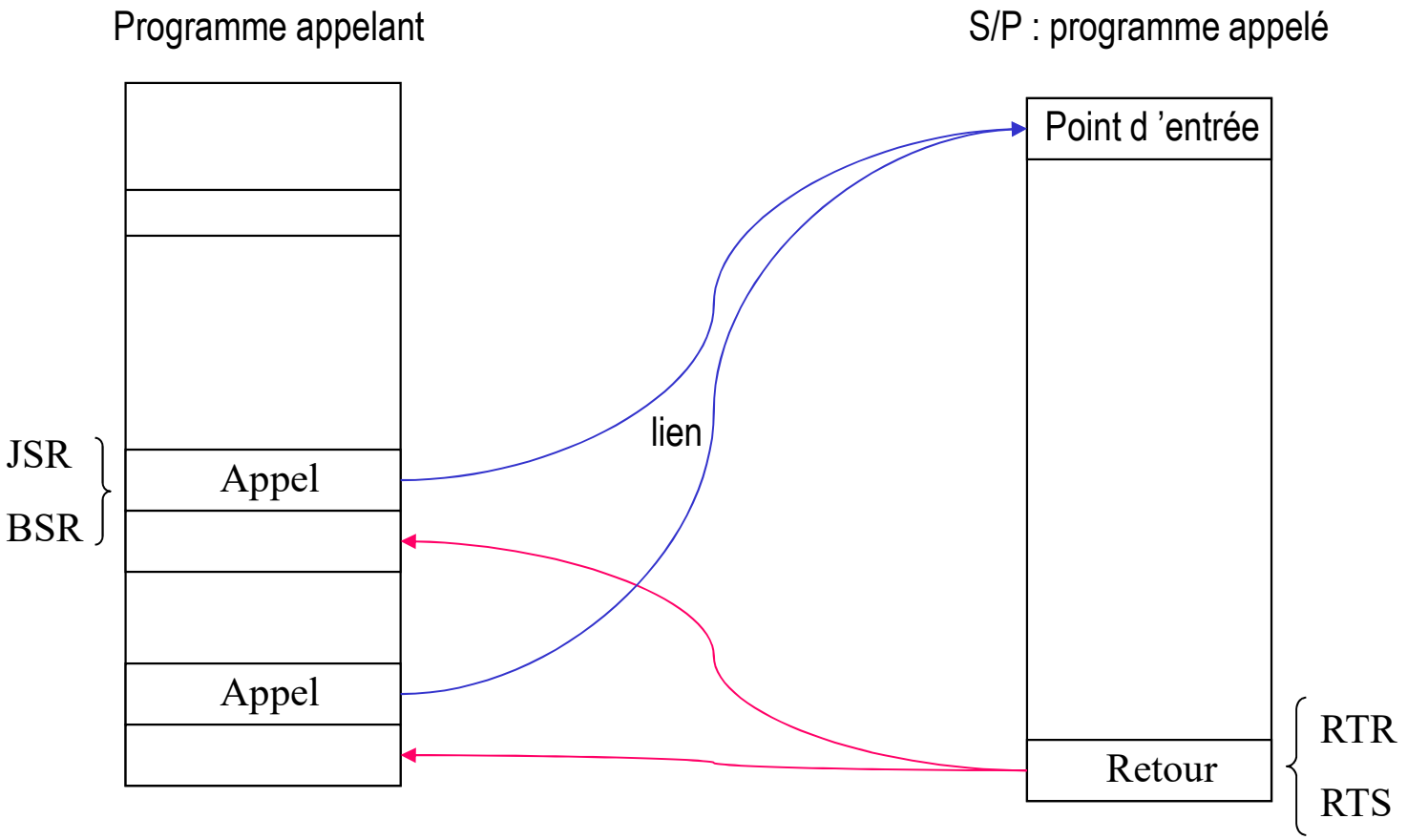
Sous programme

On appel lien la partie du programme qui permet la transmission des paramètres entre le programme appelant et le programme appelé.

L 'instruction d 'appel doit permettre la sauvegarde de l 'adresse de retour (écrite dans le programme appelant)

L 'instruction de retour (écrite dans le S/P), permet de restituer l 'adresse de retour et la mise à jour du compteur de programme

Sous programme



Manipulation de la pile

La pile est une zone mémoire de la MC permet :

- la transmission des paramètres entre Programme Principal et les sous programmes.
- Sauvegarde du contexte,
- Sauvegarder l 'adresse de retour des subroutines et exceptions.
- Permet la reantrance et la récursivité

A7 est le pointeur de Pile

```
MOVE        X,-(SP)   ou MOVE  X,-(A7)
PEA         MESSAGE(PC)
```

Initialisation pile :

```
LEA         PILE(PC),A7  ou MOVE.L      #PILE,A7
```

ASSEMBLEUR 68000

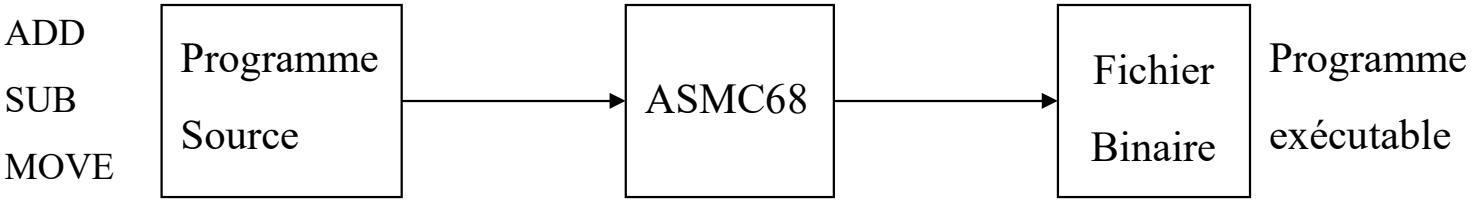
ENSIAS

- I - Organisation interne du MC68000
- II - Mémoire et périphériques
- III - Organisation des données dans la mémoire
- IV - Mode d'adressage
- V - Jeux d'instruction du MC68000
- VI - Sous programme et Manipulation de la pile
- ⇒ VIII - ASMC68
- IX - Routine Systèmes (BIOS/RDOS)

Assembleur ASMC68

- **ASSEMBLEUR : ASMC68**

L 'assembleur ASMC68 reçoit en entrée un fichier source contenant les lignes d 'instructions du programme à assembler. En sortie, le résultat de l 'assemblage est un fichier exécutable contenant le code binaire du programme.



Fichier source : est constitué de :

- une ligne blanche
- une ligne de commentaire
- une ligne de directive de l 'assembleur
- une ligne d 'instruction exécutable

Les logiciels d 'exploitation

ENSIAS

- **Moniteur**

  Il gère l'ensemble de la configuration physique du système, assure l'interface avec l'utilisateur, les programmes utilitaires et les applications.

- **Utilitaires** :
  - éditeur de texte (EDIT)
  - assembleur de fichier source (ASMC68)
  - desassemblage du binaire (ADSM68)
  - Programme debugger (DBUG)

Commandes du moniteur

ASM	: assembleur de fichier source
DBUG	: debugger du fichier Assemblé
DUMP	: impression du contenu binaire d'un fichier
FREC	: réception d'un fichier d'un PC
FSEND	: Transfert d'un fichier vers un PC
LIST	: impression ASCII d 'un fichier
MEM	: examen modification de mémoire
RUN	: exécution d'un fichier programme

Assembleur : ASMC68

ASM : Assembleur de fichier

Syntaxe : > A <fic-t> [,<fic-b>[,-p1[p2]]]

Description :

Assemblage d'un fichier source et, suivants les options p1,p2 si elles sont données, génération d'un fichier binaire, d'un listing sur l'écran, sur l'imprimante ou dans un fichier texte.

Les options p1 et p2 peuvent prendre comme valeur une des lettres suivantes :

- N : le fichier binaire ne sera pas généré, même si l 'option <fic-b> est donnée.
- V : le listing est généré sur l'écran.
- P : le listing est généré sur l 'imprimante
- L : le listing est généré dans fichier texte
- X : le fichier binaire est généré avec la table des symboles.

Remarques

- Les options V, P et L sont incompatibles deux à deux.

- Par défaut, lorsque aucune option n ’est donnée, il y a génération du fichier binaire et pas de génération de listing.

Lorsque le paramètre <fic-b> est omis, le fichier binaire, s ’il est généré, a pour nom, le nom du fichier source avec le type ‘ BIN ’.

Lorsque le listing est généré dans un fichier, ce fichier a pour nom, le nom du fichier source avec le type ‘ LST ’.

Même s ’il est demandé, le fichier binaire n ’est pas généré lorsque l ’assembleur rencontre une erreur d’assemblage ou une erreur système; et un éventuel fichier binaire ancien se trouve supprimé.

Exemples

1- > A TOTO.ASM

Génération du binaire dans ‘ TOTO.BIN ’

pas de listing

2- > A PROG.SRC -NV

Pas de fichier binaire - option N

Listing visualisé à l ’écran. - option V

3- > A TOTO.ASM , DUMY.PRG   -L

Génération du binaire dans ‘ DUMY.PRG ’

Génération du listing dans ‘ TOTO.LST ’-option L)

Remarques

1- Lorsque l ’assembleur rencontre une ou plusieurs erreurs, un message donnant le nombre d ’erreurs est affiché à l ’écran.

- Ex : ‘’fin de l ’assemblage : 04 erreurs ’’

  ‘’pas de fichier binaire ’’

- Lorsque l ’option -L n ’est pas demandée, toute ligne contenant une erreur d ’assemblage est visualisée à l ’écran.

2- Lorsque l ’assemblage se termine sans erreur, un message donnant la taille du binaire en octets est affiché à l ’écran.

Ex : ‘’fin de l ’assemblage : 0 erreur ; $004D octets ’’

Ce nombre d ’octets est la taille du code binaire

3- Avec l ’option ‘ N ’, un éventuel fichier binaire de même non existant déjà dans le catalogue ne sera pas détruit en cas d ’erreur d ’assemblage.

STRUCTURE DE ASMC68

L 'assembleur ASMC68 est un assembleur de fichier à deux passes.

L 'assembleur ASMC68 accepte comme paramètre d 'entrée un fichier source contenant les lignes d 'instructions du programme a assembler.

En sortie, le résultat de l 'assemblage est un fichier exécutable contenant le code binaire du programme.

FICHIER SOURCE

Le fichier source est un ensemble de lignes et chaque ligne peut être :

- une ligne blanche
- une ligne de commentaires
- une ligne de directive de l 'assembleur
- une ligne d 'instruction exécutable.

Ligne commentaire

*- Ligne de commentaire :

Un commentaire peut être introduit dans le programme par :

- un point-virgule (;) ou une étoile (*) en première colonne de la ligne. Dans ce cas toute la ligne est considéré comme un commentaire et n 'est pas interprétée par l 'assembleur.

- un point-virgule (;) après la fin d 'une instruction ou après un label sans instruction.

Ex : LAB.T ; label sans instruction.
CMP.L A0, A1; tester la fin du tableau

- un espace ( ) après une instruction complète.

*- Ligne blanche :

Une ligne blanche est considérée comme une ligne de commentaire.

Ligne instruction

- [<label>] <op.code> [<opérande1>[,<opérande2>]] [commentaire]

- Champs <label> :
  - Lorsqu'il est présent commence a la 1ère colonne
  - un symbole utilisé comme label n 'apparaît q 'une seule fois
- Champs <op.code> : mnemonique d 'une instruction du MC68000
  - taille .B, .W, .L (par défaut .W)
  - instruction branchement : .S, .L
  - variante des instructions : ADD, ADDA, ADDI
- Champs <opérande> : voir jeu d 'instruction
- Champs [commentaire] : tous caractère après l espace est considéré comme un commentaire et n 'est pas interprété par l 'assembleur

Ligne directive

Elle a le format suivant :

[<label>] <directive> [<opérande>[,<opérande>…] [commentaire]

Les directives de l ’assembleur sont :

- ORG : initialisation du compteur de programme
- EQU : définition de symbole
- DS : réservation de mémoire
- DC : réservation de mémoire initialisée
- END : fin du programme source
- LIS : autorise la génération des lignes listing

Directives de l 'assembleur

Appelé aussi pseudo-instructions

ORG	: initialisation du compteur de programme
EQU	: définition de symbole
DS	: réservation de mémoire
DC	: réservation du bloc de mémoire initialisée
END	: fin du programme source

EQU : Définition de symbole

Syntaxe : **<label> EQU <expression>**

<label> est le nom du symbole.

<expression> est une expression absolue sera la valeur du symbole.

Description :

Le paramètre <expression> ne doit pas comporter de symbole indéfini

Le paramètre <expression> doit pouvoir être calculé dés la première passe de l'assembleur.

Le paramètre <expression> est une valeur calculée signée sur 32 bits lorsque le symbole <label> est utilisé dans une expression arithmétique ou logique, il est remplacé par sa valeur arithmétique sur 32 bits.

Exemples

```
AA      EQU   $B0       ====> AA = $FFFFFFB0
BB      EQU   10+AA           définition correcte
AB      EQU   $100
XZ      EQU   AB+ZZ     ====> erreur 'ZZ' n'est pas encore définit

ZZ      …
TAB     BRA   FIN             Cette instruction génère du code
                                donc TAB est un symbole relatif
TOTO    EQU   TAB+10    ====>  erreur l 'expression TAB+10
                               n'est pas absolue.
```

DS : Réservation d 'une zone mémoire

Syntaxe : label DS.x <expression>

DESCRIPTION :

Cette directive permet de réserver à l 'assemblage une zone mémoire.

'x' représente l 'unité de réservation.

Pour 'x'=B, l'assembleur réserve n octets, pour 'x' = w, n mots et pour ' x ' = L, n longs mots où n représente la valeur du paramètre <expression>.

Les zones réservées par la directive DS, ont une valeur indéfinie au début de l 'exécution du programme

REMARQUE : Après réservation d'un nombre impair d'octets, si l 'instruction ou la directive suivante nécessite un alignement sur une adresse paire, l'assembleur le fait automatiquement.

Exemple

	ORG	$0100	
100	DS	10	; Réservation de 10 mots (20 octets).
114	DS.B	5	; Réservation de 5 octets.
119 TAB	DS.B	6	; il n 'y a pas d 'alignement, car la réservation porte sur des octets. La prochaine adresse du PC est toujours impaire.
11F			
120 BUF	DS.L	4	; réservation de longs mots==> il faut une valeur paire au symbole BUF. Donc le symbole BUF aura la valeur $0120 au lieu $011F

DC : Initialisation et réservation de zone mémoire


```
                ORG     $FF0200
FF0200          DC      1,2,3,$0D0A     ; réservation de 4 mots initialisés par
                                        les valeurs $0001, $0002, $0003 et
                                        $0d0A par défaut  x=W.
FF0208          DC.B    $A,$5,$0        ; réservation de 3 octets initialisés
                                        par les valeurs $FA, $05 et $00
FF020B                                  ; alignement
FF020C  TAB     DS      32
FF024C  ADR     DC.W    TAB             ; réservation d 'un mot initialisé par la
                                        valeur $020C.
                DC.L    TAB             ; réservation d 'un long-mot
                                        de valeur $FFFF020C
        BUF     DC.W    « ABCDEFGH »    ; erreur; plus de 4 caractères
        BUFA    DC.B    'ABCDE ',$00    ; réservation de 6 octets de valeur
                                        $41, $42, $43, $44, $45 et $00.
```

Exemples de Manipulation du signe

ADD.B D0,D1

- D0=$40,D1=$40
- D0=$80,D1=$80 (-128)
- -5+15
- -5+(-2)
- 5+(-15)

Fonction débordement :

D = Sx\.Sy\.Ss+Sx.Sy.Ss\

Exercices : 2ème série

Ecrire les programmes, coder les instructions, donner les adresses d'implantation des instructions et calculer le temps d'exécution de chaque programme

- Somme d 'une suite de 16 Mots résultat dans un Long Mot
- Multiplication de deux chiffres de 16 bits le résultat sur 32 bits (non signé) sans l'instruction MULU comparer le temps d 'exécution a l 'instruction MULU
- Tri des données présente dans le tableau Tab par valeur croissante dans le tableau Dest

ASSEMBLEUR 68000

ENSIAS

- I - Organisation interne du MC68000
- II - Mémoire et périphériques
- III - Organisation des données dans la mémoire
- IV - Mode d'adressage
- V - Jeux d'instruction du MC68000
- VI - Sous programme et Manipulation de la pile
- VIII - ASMC68
- ⇨ IX - Routine Systèmes (BIOS/RDOS)

Routines systèmes

Se sont des procédures utilitaires qui peuvent être appelés à partir d'un programme d'application. Ces procédures se répartissent en deux groupes :

1) Routines BIOS (Basic INPUT/OUTPUT System)

gestion des entrées/sorties physique du système.

C 'est l 'interface entre les programmes d 'application et les périphériques du système :

Lecture d'un canal/sortie sur un canal

l 'appel se fait par l 'instruction TRAP #13 ($0D)

Routines systèmes (suite)

2°) Routine RDOS (RAM-DISK OPERATING SYSTEME)

Ce sont les routines de gestion des fonctions logiques du système. C 'est l 'interface entre les programmes d 'application et le moniteur du système :

- Gestion des entrées/sorties logiques : lecture/Ecriture, Création/suppression de fichier, lecture, clavier, écriture écran ou Imprimante

l 'appel à ces routines se fait par TRAP #01.

Routines système

Procédure d 'appel :

```
MOVE.x      <pn>, - (A7)
MOVE.x      <pn-1>, - (A7)      Empiler les n paramètres de la routine
   :
   :
   :
MOVE.x      <P2>, -(A7)
MOVE.x      <P1>,-(A7)
MOVE.w      #<fonct>, -(A7)     pousser le n° de fonction de la routine

TRAP        # <num>
ADDL        # <taille>, A7      Restaurer la pile
```

Routines système

- taille = est la somme des tailles des paramètres + 2
  (restauration de la pile)
- Pi = représente une adresse, on peut remplacer
  MOVE.L <Pi>,-(A7) par PEA <Pi> ou LEA <Pi>(PC),A7
- le registre D_0 est le seul registre modifié après l'appel d'une routine. Au retour, son contenu représente le compte rendu de l'exécution de la routine sauf indication contraire :

  D0 = 0 ➔ fonction bien terminée

  D0 = 1 ➔ le numéro de la fonction donnée est inconnu

  D0 = 2 ➔ un paramètre Pi est incorrect

  D0 < -2 ➔ erreur durant l 'exécution de la fonction.

RDOS $01:Lecture du clavier avec Echo

```
MOVE        #$01,-(A7)
TRAP        #1
ADDQ.L      #2,A7
```

- pas de paramètre
- le caractère saisie est sur 8 bits de poids faible de D_0 avec tous les autre bits à 0.
- $D_0.w = \$00H_1H_2$ H_1H_2 ➔ code ASCII du caractère

RDOS $02 : Sortie d'un caractère à l 'écran

```
MOVE.W    <P1>,-(A7)
MOVE.W    #$02,-(A7)
TRAP      #1
ADDQ.L    #4,A7
```

- Paramètres : P1 (W) caractère a sortir (les 8 bits du poids faible du Mot) en ASCII
- D0=0 opérande bien terminée

Exemples RDOS $02

```
MOVE.W    #$0042,-(A7)
MOVE.W    #$02,-(A7)
TRAP      #1
ADDQ.L    # 4,(A7)
```

$42='B' en ASCII
sera affiché à l 'écran

```
MOVE.W    D1,-(A7)
MOVE.W    #$02,-(A7)
TRAP      #1
ADDQ.L    # 4, (A7)
```

Si D1 = $00FF4530
le caractère ASCII '0' de code
$30 sera affiché à l'écran

RDOS $05:

- **Sortie d 'un caractère sur l 'imprimante**

```
MOVE.W      <p1>, -(A7)
MOVE.W      #$05, -(A7)
TRAP        #1
ADDQ.L      #4,A7
```

- Le caractère codé sur les 8 bits de poids faibles du mot de 16 bits passé comme paramètre est transmis vers l'imprimante.
- D0 = 0 Opération bien terminée
- D0 = -13; le caractère n'a pu être transmis l'imprimante n'est pas prête

Exemple : RDOS $05

```
            MOVEQ       #$0A,D0
              .
              .
              .
            MOVE        D0,-(A7)
            MOVE        #5,-(A7)
            TRAP        #1
            ADDW.L      #4,A7
            TST         D0
            BNE         ERREUR
              .
              .
ERREUR      CMP         #-13, D0
                        Traitement d 'erreur de transmission
```

Le caractère ASCII LF est transmis à l 'imprimante.

RDOS $06:

Entrée d ’un caractère au clavier sans attente

```
MOVE    #$06,-(A7)
TRAP    #1
ADDQ.L  #2, A7
```

Pas de paramètre

D0 = -1; $FFFF aucun caractère n’est disponible en entrée

D0 > 0; alors les 8 bits de poids faible représentent le caractère saisi

Exemple : RDOS $06

```
        MOVE        #$06, -(A7)
        TRAP        #1
        ADDQ.L      #2,A7
        TST         D0
        BMI         N0_CAR          ; il n'y a pas eu de saisie
        :
        :
        MOVE        D0, D1          le caractère saisie est
                                    transféré dans D1
N0_CAR
```

RDOS $07:

Entrée d ’un caractère au clavier sans Echo

```
MOVE        # $07,-(A7)
TRAP        #1
ADDQ.L      # 2, A7
```

Equivalente à RDOS $01 mais sans Echo

le caractère saisie est transmis au programme appelant par D_0 (8 bits poids faible)

Exemple RDOS $07

Exemple : Saisie d 'un mot de passe sur 4 lettres.

```
                MOVE.Q      #3,D7     Nbr de lettre -1.
                CLR.L       D1
BCLE            MOVE.       #07,-(A7)
                TRAP        #1
                ADDQ.L      #2,A7
                LSL.L       #8,D1
                MOVE.B      D0, D1
                DBRA        D7, BCLE
                CMP.L       MOT.PA, D1
                .
                .
MOT.PA          DC.B        ' xxxx ',$00
```

RDOS $09:

Sortie d ’une chaîne de caractère sur l’ecran

```
MOVE.L      <P1>,- (A7)
MOVE.W      #$09,- (A7)
TRAP        #1
ADDQ.L      #6,A7
```

P1 est une Chaîne de caractère de longueur illimitée

Affichage à partir de la position courante du curseur

D_0=0 opération bien terminée

Exemple RDOS $09

```
        MOVE.L      #TAB,- (A7) ou PEA     <P1> ou LEA
        MOVE.W      #$09,- (A7)
        TRAP        #1
        ADDQ.L      #6,A7
TAB     DC.B        10,13, 'Ceci est une chaîne ',10,13,00
```

Le MOVE.L pourra être remplacé par : LEA TAB(PC),A7 ou PEA TAB(PC)

RDOS $0A:

Entrée d ’une chaîne de caractère au clavier

```
MOVE.L      <P1>,-(A7)    ou PEA <P1> ou LEA
MOVE.W      #$0A,-(A7)
TRAP  #1
ADDQ.L      #6,A7
```

Descriptif

Cette fonction attend la saisie au clavier d 'une chaîne de caractères. P1 est l 'adresse d 'un buffer qui indique le nombre maximum à saisir et reçoit les caractères saisis.

L 'octet contenu à l 'adresse <P1> a pour valeur le nombre maximum de caractères à saisir.

Au retour au programme appelant, l 'octet à l 'adresse <p1>+1 indique le nombre de caractère effectivement saisis. Les caractères saisis sont rangés les uns à la suite des autres à partir de l 'adresse <p1>+2.

La saisie des caractères se termine lorsque la touche <CR> (Carriage Return = retour du chariot) est tapé, ou lorsque le nombre maximum imposé par le paramètre d 'appel est atteint.

Le caractère <CR> n 'est pas compté dans le nombre de caractères saisis.

Exemple RDOS $0A

```
        MOVE.L    #BUFF,-(A7)
        MOVE.W    #$0A,-(A7)
        TRAP  #1
        ADDQ.L    #6,A7
BUFF    DC.B      70      (70 caractères Maxi)
        DS.B      1       nbr de caractères saisie
        DS.B      70      buffer caractères saisie
```

MOVE.L #BUFF,A7 pourra être remplacé par :

- PEA BUFF(PC) ou
- LEA BUFF(PC),A7