Résumé Assembleur

Labo 2

06.11.2020

Owen Gombas
ISC1c

bro Entry

Constante d'assemblage EQU

permet de substituer un nom par une valeur. c'est lors de l'assemblage que ce nom est remplacé par sa valeur dans le code.

```
LABEL:    EQU    VALUE
```

utilisation :

```
COUNT:    EQU    5

          LDAA   #COUNT
```

à ne pas oublier pour charger la valeur et non l'adresse

Après assemblage c'est comme si on avait écrit :

```
          LDAA   #5
```

Champ opérande (les expressions)

Une opérande peut être

- Un symbole
  C'est une adresse remplacée par un nom
- Une constante
- Un symbole constant défini par EQU
- Une expression
- Il ne peut que avoir qu'un opérande représentant une adresse

Une expression c'est :

- Une combinaison de symboles et/ou de constantes combinées avec des opérateurs algébriques, logiques ou rationnels
- Elle est évaluée par l'assembleur lors de l'assemblage et non pas à l'exécution.
  Pour cette raison, les expressions sont uniquement utilisables comme constante.

Opérateurs algébriques

Opérateurs algébriques	Description
+	Addition
*	Multiplication
/	Division produces truncated result
-	Subtraction
%	Modulo division

```
CST1: EQU 50          ; Constante(de l'assembleur)
CST2: EQU CST1*2      ; Constante(de l'assembleur)

LDAA #CST1+1          ; Additionne 1 à CST1

LDAA #CST1/2          ; Divise CST1 par 2

LDAA #CST2*2          ; Multiplie CST2 par 2
```

Opérateurs logiques

Opérateurs logiques	Description
>>	Déplacement bit à bit vers la droite
<<	Déplacement bit à bit vers la gauche
&	ET logique bit à bit
|	OU logique bit à bit
^	OU exclusive logique bit à bit
~	Inversion logique bit à bit

```
VAR1: EQU 100         ; Constante (de l'assembleur)
VAR2: EQU $FF         ; Constante (de l'assembleur)

LDAA #VAR1>>1         ; Déplace chaque bit de VAR1 de 1 pas vers la droite puis
                      ; charge A avec cette valeur
LDD #VAR1<<2          ; Déplace chaque bit de VAR1 de 2 pas vers la gauche puis
                      ; charge A avec cette valeur
LDAA #VAR2&$1         ; ET logique entre $FF et 1
```

Opérateurs rationnels

Opérateurs relationnels	Description
!	Logical NOT
!= or <>	Not equal
<=	Less than or equal
>=	Greater than or equal
= or = =	Equal
<	Less than
>	Greater than

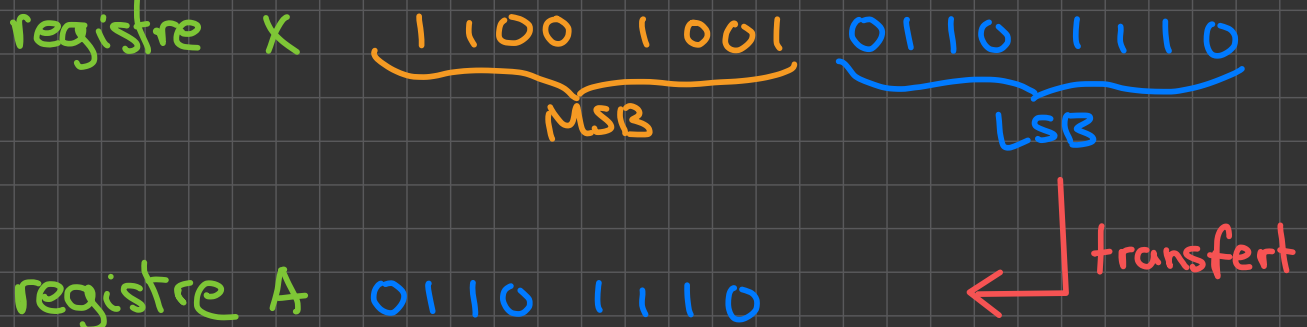
```
VAR1: EQU 100                ; Constante(de l'assembleur)
VAR2: EQU $FF                ; Constante(de l'assembleur)

LDAA #((VAR1>>1)>=10)        ; A = 1 si VAR1>>1 est plus grand ou égal à 10, sinon 0

LDAA #(VAR1!=100)            ; A = 0 si VAR1 est égal à 100, sinon 1
```

Instruction de transfert

- Permet de transferer une données d'un registre à l'autre
- Lorsqu'on transfert depuis un registre 16 bits vers un registres 8 bits, seul le LSB est transfereée

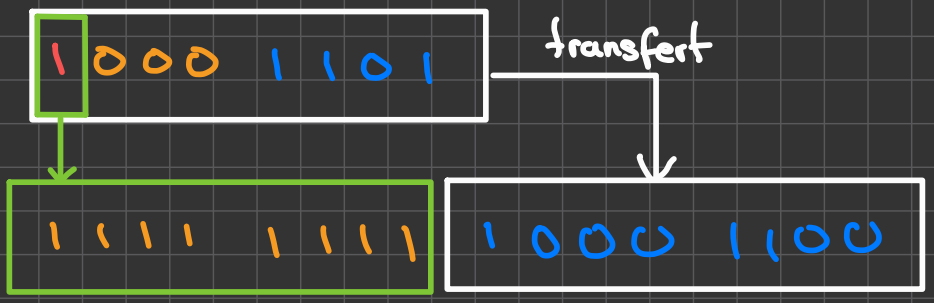


- Lorsqu'on transfert depuis un registre 8 bits vers un registres 16 bits, il y'a une extension de signe.

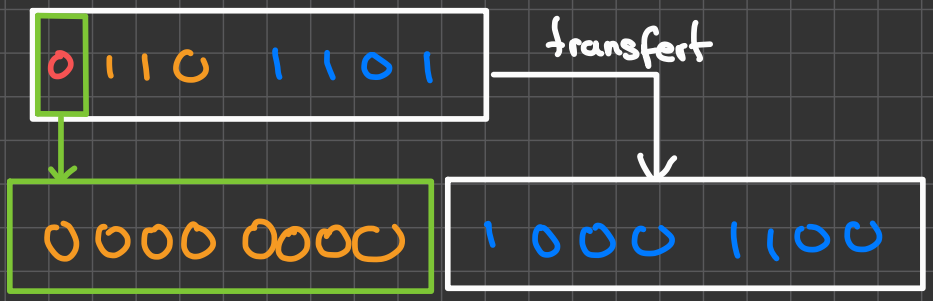
① Si le bit de poid fort de la donnée 8bits est à 1 :
- le MSB prend la valeur $ff

② Sinon
- le MSB prend la valeur $00

①



②



Utilisation :

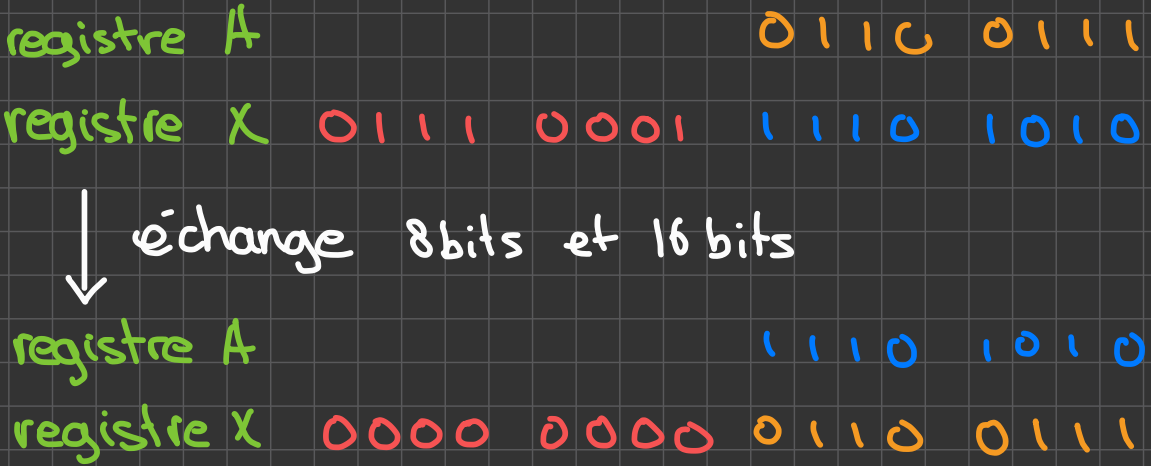
```
TFR      source, destination
```

Exemples :

```
TFR      A, B
TFR      X, B
TFR      A, D
```

Instruction d'échange de deux registres

- Permet de faire un échange non signé entre deux registres (A, B, D, X, Y, SP, CCR)
- Lors de l'échange d'un registre 8 bits et 16 bits, le LSB du registre 16 bits est toujours copié dans le registre 8 bits. Le MSB du registre 16 bits est à 0



Utilisation

```
EXG   r1, r2
```

Exemples :

```
exg   X, Y
exg   A, B
exg   X, A
```

Transfert de mémoire à mémoire

- MOVB

  transfert 1 byte d'une case mémoire à une autre

- MOVW

  transfert un word (2 bytes)

Utilisation

```
MOVB    addr1, addr2
```

```
MOVB    #value, addr2
```

Exemples:

```
movb  #32, $811
movw  ABC, DEF
```

adresses mémoires