UNIVERSITE DE GABES
École Nationale d'Ingénieurs de Gabès
Département GCR

ENIG

Année: 2019/2020
Niveau : 1ème année
Ens. Res. HAMDI Ridha

TD

Architecture des Ordinateurs et Microprocesseurs

Exercice 1 :

Donnez la taille (nb d'octets) de la représentation-mémoire de ces instructions

MOVE.W $80004,D3

MOVEQ.B $04,D0

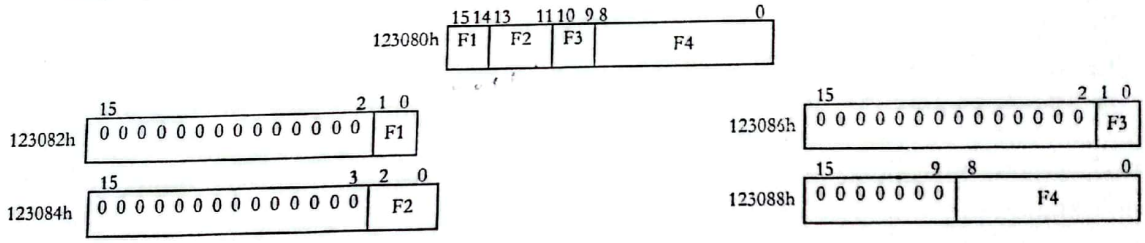
MOVE.L (A0)+,D0

Exercice 2 :

Proposez un sous programme qui découpe le mot mémoire d'adresse 123080h comme suit :

- les bits 14-15 sont rangés dans le mot d'adresse 123082h,
- les bits 11-13 sont rangés dans le mot d'adresse 123084h,
- les bits 9-10 sont rangés dans le mot d'adresse 123086h,
- les bits 0-8 sont rangés dans le mot d'adresse 123088h,
- les bits restants des quatre mots de destination sont à 0.

Ce sous programme réside dans la mémoire à partir de l'adresse 120100h.



Exercice 3 :

1- Le tableau suivant donne le continu d'un bloc de la mémoire RAM. Il s'agit de la représentation-mémoire d'un petit programme assembleur 68000. Déduisez les instructions de ce programme (Désassemblage).

.....		
.....		
.....		
.....		
.....		
.....		
.....		

$ 400400	$ 34	$ 00
$ 400402	$ 34	$ 70
$ 400404	$ 00	$ 70
$ 400406	$ 34	$ 00
$ 400408	$ E9	$ 18
$ 40040A	$ 55	$ 42

00 1 10 100 | 0 111 1100

Move.W #$007C, A2

TD:

Ex01

Donnez la taille de Representation memoire des instructions suivantes :

. Move.w $800 c4, D_3 (6 octets)

. MoveQ #b #$04, D_0

. Move.L #A_0)+, D_0 2 octet

Representation memoire = opcode + l'extension

Ex 02

Ecrire un Sous prog [illegible] BTS

15 14	13 11	10 9	
F1	F2	F3	F4

$123080

$123082 $123084 $123088

Decou

org $120100

ce sous Programme reside dans la mémoire à partir de l'adresse 120100h

Décompo Move $123080, D_0

Move D_0, D1

AND #$C000, D1

ROL.W #2, D1

Move D1, $123082

Move D_0, D1

AND #$3800, D1

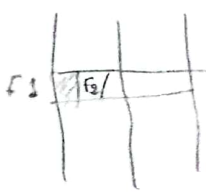
ROL #5, D1

Move D1, $123084

⋮

RTS

Rol #2



AND
MOVP
POC

15 14
1100 _0_ _0_ -
C

Indirect
Suite
d'élément

D_1.W

EX3 :

move.W, D_2 1+070

00	11	0 100	0 000	000	0003
	Size	X_n	M	M	X_n
		Dest			Source

3 → Move.W

Size est 111

Move.W #$ D_2

① MoveQ # $70, D_0

0000 0000 0111 0 000

ORI.W

0011 0100

EX5 :

```
Init  Move.L #$800001,A0
      Move.b #$80,$0C(A0)
      Move.b #$80,$0E(A0)
      Move.b #$FF,$04(A0)
      Move.b #$C0,$06(A0)
      RTS
```

PIT

- 1 octet | PGCR
- 1 octet | PACR/PBCR
- 1 octet | PADDR/PBDDR/PCDDR
- 1 octet | PADR/PBDR/PCDR

Mode 1X — Direction

registre de données du Port A

```
③ Display  Move.b D0,$10(A0)
           Move.b D0,D3
           ROL.b D1,D3
           OR.b D1,D4
           Move.b D4,$10(A0)
           BSR Delay
           RTS
```

D0 Dizaine
D1 unités

Mode 0
sous Mode 1X

PA →
PB ⇆

. interface entre de registre configurable

registre de contrôle → Mode

PA et PB (Mode, Sous Mode

pas reg de contrôle p88

sens unité (par défaut : en entrée

unidirectionnel sortie

bidirectionnel → en cas presque sortie entrée

programme d'initialisation des registres de et B de PI/T. Donnez ce sous-programme qui ableau suivant.

Registre	Abréviation	Adresse	Offset
Registre de contrôle général du port	PGCR	$800001	$00
Registre de direction du port A	PADDR	$800005	$04
Registre de direction du port B	PBDDR	$800007	$06
Registre de contrôle du port A	PACR	$80000D	$0C
Registre de contrôle du port B	PBCR	$80000F	$0E
Registre de données du port A	PADR	$800011	$10
Registre de données du port B	PBDR	$800013	$12
Registre alterné du port A	PAAR	$800015	$14
Registre alterné du port B	PBAR	$800017	$16

3) Dans tous ce qui suit, on suppose que les registres de données D0 et D1 reflètent respectivement les valeurs affichées ou qui seront affichées par l'afficheur des unités et celui des dizaines.

c-à-d :

Si D0 contient $06 et D1 contient $04, la minuterie affiche la valeur suivante

Proposez un sous-programme qui débute à l'adresse *'Display'* permettant d'afficher sur la minuterie les valeurs des unités et des dizaines lus respectivement à partir de D0 et D1. Ce sous-programme fait appel à un autre sous-programme de temporisation d'une seconde qui débute à l'adresse *'Delay'* pour pouvoir visualiser chaque changement. Donnez aussi ce sous-programme de temporisation.

```
Display...MOVE.b....D0,..D3.....
.........MOVE.b.....D1,..D4.....
.........AND.b...#$F,..D4.....
.........MULU.b...D3,..D4.....
.........MOVE.b...D4,....$10(A0)
.........BSR.....Delay.....
.........RTS.....
```

```
Delay.....
.....MOVE...L..#$5.1800,D7
...next...SUBQ..L..#1,D7
.....BNE.....next.....
.....RTS.....
.....
.....
```

4) Complétez le programme principal, suivant, qui débute à l'adresse $400400, en tenant compte des indications suivantes :

* Vous pouvez éventuellement changer la partie du programme donnée si elle ne correspond pas à votre propre choix des registres.
* Si l'utilisateur appuie sur le bouton '*+1s*' et que l'afficheur des unités affiche la valeur 9, celui ci est mise à zéro tandis que l'afficheur des dizaines est incrémenté de 1 tant que ce dernier est inférieure à 9. Si la minuterie contient la valeur 99, tout appuis sur le bouton '*+1s*' sera ignoré par le microprocesseur.
* Si l'utilisateur appuie sur le bouton '*+10s*' et que l'afficheur des dizaines affiche la valeur 9, cet appuie sera ignoré par le microprocesseur.

2) Le programme principal appelle un sous-programme d'initialisation des registres de direction et de contrôle de deux ports A et B de PI/T. Donnez ce sous-programme qui débute à l'adresse *'Init'* en se referant au tableau suivant.

```
Init  move.l  #$1000,A0
      .....
      move.b  #$80,$0C(A0)
      move.b  #$00,$0E(A0)
      move.b  #$FF,$04(A0)
      move.b  #$00,$06(A0)
      .....
      .....
```

Registre	Abréviation	Adresse	Offset
Registre de contrôle général du port	PGCR	$800001	$00
Registre de direction du port A	PADDR	$800005	$04
Registre de direction du port B	PBDDR	$800007	$06
Registre de contrôle du port A	PACR	$80000D	$0C
Registre de contrôle du port B	PBCR	$80000F	$0E
Registre de données du port A	PADR	$800011	$10
Registre de données du port B	PBDR	$800013	$12
Registre alterné du port A	PAAR	$800015	$14
Registre alterné du port B	PBAR	$800017	$16

3) Dans tous ce qui suit, on suppose que les registres de données D0 et D1 reflètent respectivement les valeurs affichées ou qui seront affichées par l'afficheur des unités et celui des dizaines.

c-à-d :
Si D0 contient $06 et D1 contient $04, la minuterie affiche la valeur suivante

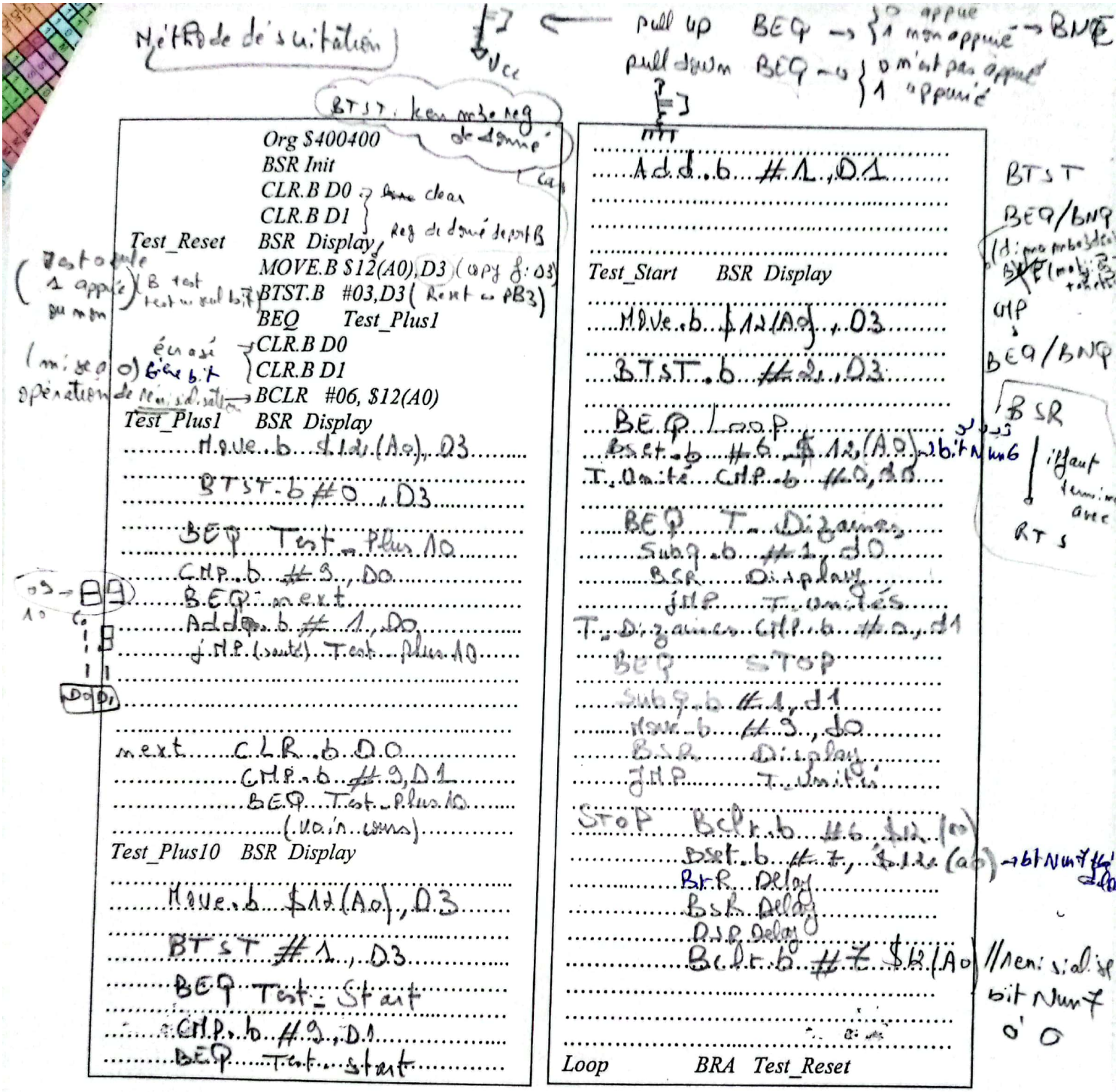
Proposez un sous-programme qui débute à l'adresse *'Display'* permettant d'afficher sur la minuterie les valeurs des unités et des dizaines lus respectivement à partir de D0 et D1. Ce sous-programme fait appel à un autre sous-programme de temporisation d'une seconde qui débute à l'adresse *'Delay'* pour pouvoir visualiser chaque changement. Donnez aussi ce sous-programme de temporisation.

```
Display  MOVE.b  D0,D3
         MOVE.b  D1,D4
         ADD.b   #$0,D4
         LSL.b   #4,D4
         OR.b    D3,D4
         MOVE.b  D4,$10(A0)
         BSR     Delay
         RTS
```

```
Delay    MOVE.L  #$5F5E100,D7
         SUBQ.L  #1,D7
         BNE     Next
         RTS
```

4) Complétez le programme principal, suivant, qui débute à l'adresse $400400, en tenant compte des indications suivantes :

* Vous pouvez éventuellement changer la partie du programme donnée si elle ne correspond pas à votre propre choix des registres.
* Si l'utilisateur appuie sur le bouton *'+1s'* et que l'afficheur des unités affiche la valeur 9, celui ci est mise à zéro tandis que l'afficheur des dizaines est incrémenté de 1 tant que ce dernier est inférieure à 9. Si la minuterie contient la valeur 99, tout appuis sur le bouton *'+1s'* sera ignoré par le microprocesseur.
* Si l'utilisateur appuie sur le bouton *'+10s'* et que l'afficheur des dizaines affiche la valeur 9, cet appuie sera ignoré par le microprocesseur.



Exercice 6 :

Une entreprise industrielle spécialisée dans la production des câbles en plastique pour automobiles est dotée d'une machine 'ULMER' de découpage des gaines en plastique. Cette machine est contrôlée par un système à base du microprocesseur 68000. Sa fonction est de couper une gaine en plastique en morceaux de longueur prédéfinie.

. move # $FFFF, $3025 ⚠ impo (3085 impair) → on ne peut pas écrire
16 ¢ sur 2 octets dans le @ impaire

. move # $22 A0, $ FFA 320 → @ d'une case mémoire
16 ¢

. move # $403C . $B320 ← absolute short (court)
16 ¢
FF FFB320
si je veut l'@ 00B320
il faut écrire
$ 00 B 320
absolute long (large)

$ 00 0000 … $ 007FFE	absolute short
(dans ce deux zone on peut écrire l'@ sur 2 octets seulement)	(absolute long)
FF 8000 … FFFFFE	absolute short

move # $ 403C , $ ~~X~~ 7A20
Architecture signe = plus rapide (très compliqué)

move # $3C , $ 7840
move.L # 30 , $ A0 B~~7~~ 6
(en hexa 1E)

	0	0
	00	1E

move # $28, $ FF 403C

@ move.b # $7 . $12(A0) ($800007) #indirecte
+12 ⚠ pointeur
→ move.b #$7. $800013

move.L # $FF. $06(A4) ← En dehors de l'@
∟ case mémoire qui se trouve dans A4

| 400 400 | A4

$ 400406 :

move # $FF, A4 | 00 00 00 FF | A4
> 2oct : absolut long
≤ 2oct : absolut short

directe . move.b # 00 . $400 400
⋮
move.b # 00 . $4007E7

Indirect → plus rapide + moindre espace mémoire

@ move.L #$ 400400, A5

move.b # 00, -(A5)+ ← post incrémenté
↑ prédécrimenté
il faut incrémenter avec
- un seule octet (.b)
- 2 octets (.w)
- 4 octets (.L)

CMP.L # $4007E8, A5 (comparaison)
l'@ et l'@ qui se trouve en A5

BNE LOOP
(Branch if not equal)

dc : define constant

BLT : Branch if ~~not~~ less than

trap #15) pour arrêter le prog
dc . W

faire un boucle pour s'arrêter
Loop Move.b # 00, (A3)+
cmp.L #$4007E8, A3
∟ compare // l'@ sur 4 octet
BNE Loop (if not equal continue)

move # 00 , 500 (A3)
(500₁₆ → ( ) ₁₆ puis on l'ajoute à A3)

org $400500
dc.b " A 68000 is 16 bit complex instruction"
∟ define ct

Arch.

move : par défaut elle prend .W : 2 octets

⚠ moveQ : tjr (.L) par défaut

⊙ move # $304C, D1 (move.W)
- # : immédiat
- $304C : Hexa
- D1 : registre de donnée

opcode (16 bits / 2 octets)

3		A		3		C
00	11	101	000	111	100	= 3A3C
move	.W	X_d	M	M	X_s	

* l'opcode ne tient pas compte de la constante → il faut écrire dans la RAM → l'opcode est tjr sur 2 octets ⚠

RAM

3A	3C
30	4C

(2 octets)

Nc
3A3C

Fetch → decode → execute → writeBACK

⊙ move.L #$304C, D5

00	00	30	4C	D5

car (.L) (4 octets)

opcode :

0 010 1 010 0 0111 1100

2 A 3 C

2A	3C
00	00
30	4C

: 2 octets

⊙ move.b #$4C, D5

00	FF	80	4C	D5

déjà trouvé — va écraser le premier case et mettre 4C dedans

opcode

00 01 101 000 111 100

1 A 3 C

: 004C — il va mettre LSB dans D5

1A	3C
00	4C

car l'opération d'écriture se fait sur 2 octets

move.b # $4C, D5
⚠ erreur car 4C s'écrit pas dans la Base décimale

move.b # 40, D5 conversion en Hexa

			28

⊙ moveQ.L # $28, D5 →

00	00	00	28	D5

(.L par défaut)

- -128 ≤ cte ≤ +127
  (0) (255)

7A	28	opcode

éliminer l'extension (move quickly) ✱

opcode

0111 1010 0010 1000

7 A 2 8

⊙ moveQ #(-1), D5 ; (-1) ≠

00	00	00	FF

nbre (+1) ... 255

⇓

(-1) =

FF	FF	FF	FF

0111 1010 1

moveQ # $99, D5 (Hexa)

=

moveQ # -67, D5 (Déc)

FF	FF	FF	99

1001 1001 : bit de signe (1) → il faut mettre 1 pour tout (extension de bit de signe) → pour les Reg de donnée seulement avec MoveQ

move # $908A, A3

FF	FF	90	8A	A3

bit n°15 = 1 ; 0 → 1001

=

⚠ ⊙ MoveA (.W) .L
✱ pas de .b en moveA
↑ extension de bit de signe = de poids le plus fort

move.b # 3A, A2 (faux) → move.b ?? ⚠

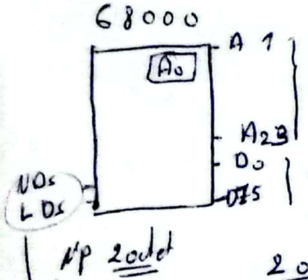
move.b # 3A, A2 ... cte ne peut pas être dans la destination

Note Arch

type de Bus { de données (RAM) / de controle (lecture dans RAM | Ecrire | Activation) (UDs, LDs) / d'adressage (2) : à l'exterieur de µP

- Microprocesseur : unité de traitement et execution de programme
- Memori management unit (MMU) — refraichement de mémoire
  ↳ copie intégrale de programme qui se trouve dans la mémoire

- on a 3 type de Registre → Data (D0 – D7) (4 octet) (16 b)
  ↳ @ : registre ... à l'interieur de µp (A0 – A7)
  ↳ controle



bus d'@ — 8 bait donné — 2 octet

µp 2 octet — cad 1@ pair ou impaire (à effet sur D0)

- taille totale = nbre de cases memoire × nbre de octet données par le bus d'@

$2^{23} = 2^3 \times 2^{20}$

$= 8\,MO \times 2$

$= 8M \times 2 = 16\,MO$

- Motorola : (Big endian)

$ 3C A0 2847 → MSB

@ pair	@ impair
3C	A0
28	47

← LSB

Bus de control { UDs : Upper Data search / LDs : Low Data search

A0 … A2 — 4 octet

pas .b dans le R. d'adresse
↳ .w ou .L

- operation

Move D0, D2 — src / dest

deplacer (copy)

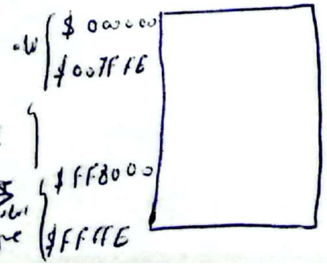
registre d'@ sur 3 octet ⚠

move.W # $ C340, A5

~~00~~	FF	C3	40

A5 — non utilisé

adresse A5 — extension de bit de signe



move.W # $ C340, A5 → Ecrire le @ dans un registre d'@

FF	C3	40

A5 — ↳ extension de bit de signe ⚠ regist. d'adresse

move.L # $ C340 , A5

00	C3	40

A5

ALU : Arithmetique Unite

Tjr F D E W — write back "optionnel" : expl si on change le adresse dans la RAM

fetch " lecture — Decode — execute

move — opcode (2)

- .W
- FF = -1
- FE = -2

- Langage Assembleur :

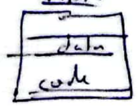
- BRA : branch Always — Bosch White ... opcode (7 … …)

opcode tjr 2 octets — extension x2 octets

- Move q # $FF , D3 → moveq .W (par defaut) (extension de bit de signe just avec moveq) ↳ copy un immediate vers une R. de données — taille de l'opcode = 2 octets tjr dans la RAM — cste [-128 . 127] : .W

- Addq #5 , D3 ((W) par defaut) — ctre de dépasse pas 8 [1 .. 8]

Absolute long > 2 octets



(An)+ : post incrémentation

-(An) : pré-decrémentation

$72(A0) = $800073

$800001

BNE : Branch not equal

BNE Loop ; etiquet : label

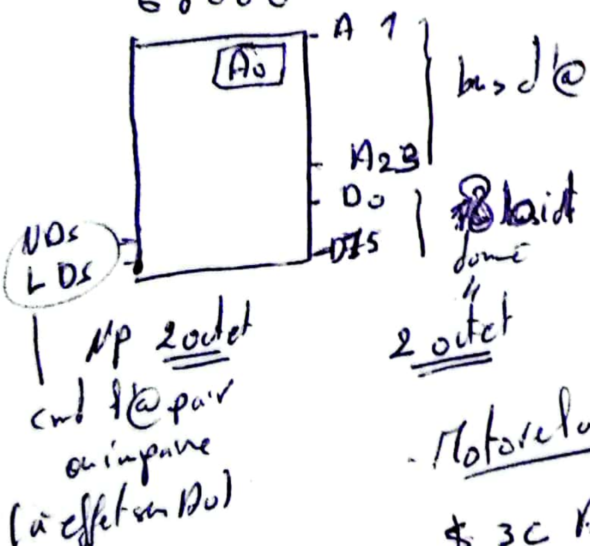
CMP # $4006CB , A : condition d'arret

Note Arch

type de Bus
- de donnés (RAM)
- de controle (lecture dans RAM / Ecrire / Activation) (UDS, LDS)
- d'adressage (2) : à l'exterieur de µp

- Microprocesseur : unité de traitement et execution de programme
- Memori managment unit (MMU) — refraichement de mémoire
  - copie integrale de programme qui se trouve dans la mémoire flash
- on a 3 type de Registre → Data (D0 – D7) (4 octet) (16 b)
  - @ : registre à l'interieur de µp (A0 – A7)
  - controle

68000



- taille totale = nbre de cases memoire × nbre de octet données par le bus d'@

$2^{23} = 2^3 \times 2^{20}$ × 2
= 8 MO
= 8 M × 2 = 16 MO

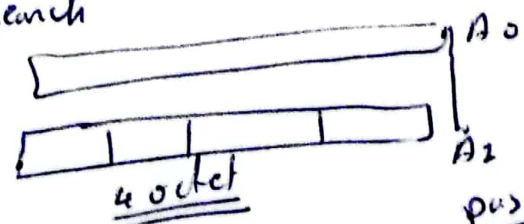
- Motorola : (Big endian)

$ 3C A0 28 47 → MSB

@ pair	@ impair
3C	A0
28	47

← LSB

Bus de controle
- UDS : Uper Data search
- LDS : Low Data search



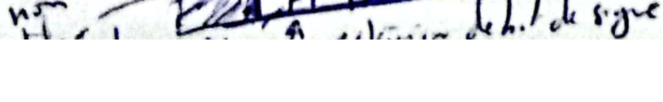
- operation

Move D0, D2 (src, dest) — deplacer (copy)

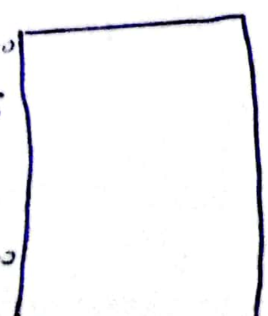
registre d'@ sur 3 octet ⚠

pas .b dans le R. d'adress → .w ou .L

move.W # $ C340, A5



.W { $ 000000 … $ 007FFE }
.L { $ FF8000 … $ FFFFE }



Ecrire le @ dans le registre d'@

move.W #$C340,A5 → | FF | C3 | 40 | A5

↳ extension de bit de signe ⚠ regist. d'adresse

move.L #$C340,A5 | 00 | C3 | 40 | A5

ALU : Arithmétique Unit

Tjr F D E W → write back "optionnel" : expl si on change l'adresse dans la RAM

fetch " lecture / Decode / execute

move | 2 | — opcod

.w
FF = "-1"
FE = "-2"
⋮

- Langage Assembleur :

• BRA : branch Always
Branch While true :
opcod | 7 — | — |

• Moveq #$FF,D3 → moveq .W (par defaut) | extension de bit de signe just avec moveQ)
↳ copy un immédiat vers un R. de données
cste
[-128, 127] : .W

Taille de l'opcode = 2 octets tjr dans la RAM

opcode tjr 2 octets
extension x2 octets

• Addq #5,D3 (W par defaut)
cte ne dépasse pas 8
[1 .. 8]

1	001
⋮	⋮
7	111
8	000

Absolute long → 2 octets

RAM
| data |
| code |

(An)+ : post incrémentation
-(An) : pré-décrémentation
$72(A0) = $800073
$800001

BNE : Branch not equal

BNE Loop : etiquette : label

CMP #$4006C8,A : condition d'arrêt