

# Partiel 2 – Corrigé

## Architecture des ordinateurs

Durée : 1 h 30

**Exercice 1 (9 points)**

Toutes les questions de cet exercice sont indépendantes. À l'exception d'éventuels registres de sortie, aucun registre ne devra être modifié en sortie d'un sous-programme.

- Réalisez le sous-programme **IsLower** qui détermine si le code ASCII d'un caractère correspond à une minuscule.

Entrée : **D1.B** contient le code ASCII d'un caractère.

Sortie : **D0.L** renvoie 1 si le caractère est une minuscule.

**D0.L** renvoie 0 si le caractère n'est pas une minuscule.

**Indications :**

- Si le code ASCII d'un caractère est inférieur au code ASCII du caractère 'a', alors le caractère n'est pas une minuscule.
- Si le code ASCII d'un caractère est supérieur au code ASCII du caractère 'z', alors le caractère n'est pas une minuscule.

```
IsLower      ; Si le caractère est inférieur au code ASCII de 'a',
              ; le caractère n'est pas une minuscule.
              cmp.b    #'a',d1
              blo.s    \false

              ; Si le caractère est supérieur au code ASCII de 'z',
              ; le caractère n'est pas une minuscule.
              cmp.b    #'z',d1
              bhi.s    \false

\true        ; Le caractère est une minuscule, on renvoie 1 dans D0.
              moveq.l  #1,d0
              rts

\false       ; Le caractère n'est pas une minuscule, on renvoie 0 dans D0.
              moveq.l  #0,d0
              rts
```

- À l'aide du sous-programme **IsLower**, réalisez le sous-programme **ToUpperChar** qui convertit un caractère minuscule en un caractère majuscule.

Entrée : **D1.B** contient le code ASCII d'un caractère.

Sortie : Si **D1.B** contient une minuscule, alors **D1.B** est converti en majuscule.

Si **D1.B** ne contient pas une minuscule, alors **D1.B** reste inchangé.

**Indication :**

- La conversion d'une minuscule en majuscule s'obtient en soustrayant la valeur hexadécimale \$20 du code ASCII d'une minuscule (par exemple : 'a' = \$61 et 'A' = \$41).

```

ToUpperChar    ; Sauvegarde D0 dans la pile.
               move.l    d0,-(a7)

               ; Si le caractère est une minuscule, on quitte sans modifier D1.
               jsr      IsLower
               tst.l     d0
               beq.s     \quit

               ; Sinon, on convertit D1 en majuscule.
               subi.b    #$20,d1

\quit          ; Restaure D0 puis sortie.
               move.l    (a7)+,d0
               rts

```

3. À l'aide du sous-programme **ToUpperChar**, réalisez le sous-programme **ToUpperString** qui convertit toutes les minuscules d'une chaîne de caractères en majuscules.

Entrée : **A0.L** pointe sur une chaîne de caractères terminée par un caractère nul.

Exemple d'utilisation de **ToUpperString** :

```

Main          lea      sTest,a0
               jsr      ToUpperString
               illegal

sTest         dc.b      "Ceci est une chaine a convertir.",0

; En sortie du sous-programme, la chaîne contiendra les caractères suivants :
; "CECI EST UNE CHAINE A CONVERTIR."

```

```

ToUpperString ; Sauvegarde D1 et A0 dans la pile.
               movem.l   d1/a0,-(a7)

\loop         ; Copie un caractère de la chaîne dans D1.
               move.b    (a0),d1

               ; Convertit le caractère en majuscule.
               jsr      ToUpperChar

               ; Le caractère converti est copié dans la chaîne.
               ; On fait pointer A0 sur le caractère suivant.
               move.b     d1,(a0)+

               ; On reboucle tant que le caractère qui vient d'être copié
               ; dans la chaîne n'est pas nul.
               ; (L'instruction précédente "move.b d1,(a0)+" à modifié
               ; le flag Z en fonction de la valeur de D1.)
               bne.s     \loop

\quit         ; Restaure D1 et A0 puis sortie.
               movem.l   (a7)+,d1/a0
               rts

```

**Exercice 2 (2 points)**

Codez les instructions ci-dessous en langage machine 68000. **Vous détaillerez les différents champs** puis **vous exprimerez le résultat final sous forme hexadécimale** en précisant **la taille des mots supplémentaires** lorsque le cas se présente.

1. `MOVE.W (A7)+, $3000`

**MOVE** (cf. [documentation ci-annexée](#))

0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		
0	0	1	1	0	0	1	1	1	1	0	1	1	1	1	1
<b>MOVE</b>		<b>.W</b>		<b>(xxx.L)</b>						<b>(A7)+</b>					

- Mot d'extension à ajouter pour la destination : **(xxx).L = \$00003000**

Un adressage absolu long représente une adresse sur 32 bits non signés.

Code machine complet en représentation hexadécimale : **33DF 00003000**

2. `MOVE.L #$3000, -4(A3)`

**MOVE** (cf. [documentation ci-annexée](#))

0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		
0	0	1	0	0	1	1	1	0	1	1	1	1	0	0	0
<b>MOVE</b>		<b>.L</b>		<b>d16(A3)</b>						<b>#&lt;data&gt;</b>					

- Mot d'extension à ajouter pour la source : **#<data> = #\$00003000**

La taille de la donnée du mode d'adressage immédiat correspond à la taille de l'instruction. L'instruction possède ici l'extension .L. La taille de la donnée est donc de 32 bits.

- Mot d'extension à ajouter pour la destination : **d16 = -4 = \$FFFC**

d16 représente un déplacement sur 16 bits signés. Il faut donc convertir la valeur -4 en représentation hexadécimale sur 16 bits signés.

Code machine complet en représentation hexadécimale : **277C 00003000 FFFC**

**Exercice 3 (2 points)**

Vous indiquerez après chaque instruction, le nouveau contenu des registres (sauf le **PC**) et/ou de la mémoire qui viennent d'être modifiés. **Vous utiliserez la représentation hexadécimale. La mémoire et les registres sont réinitialisés à chaque nouvelle instruction.**

Valeurs initiales :    D0 = \$0000FFFF    A0 = \$00005000    PC = \$00006000  
                              D1 = \$FFFFFF0004    A1 = \$00005008  
                              D2 = \$FFFFFFFFFE    A2 = \$00005010

\$005000    54 AF 18 B9 E7 21 48 C0  
 \$005008    C9 10 11 C8 D4 36 1F 88  
 \$005010    13 79 01 80 42 1A 2D 49

1. MOVE.W    12(A0), -\$(A2, D2.L)

Source	Destination
12(A0)	-\$A(A2, D2.L)
(A0 + 12)	(A2 + D2 - \$A)
(\$5000 + 12)	(\$5010 - 2 - \$A)
(\$500C)	<b>(\$5004)</b>
<b>#\$D436</b>	

\$005000    54 AF 18 B9 **D4 36** 48 C0

2. MOVE.B    3(A2, D1.W), -(A1)

Source	Destination
3(A2, D1.W)	(A1)
(A2 + D1.W + 3)	(\$5008 - 1)
(\$5010 + 4 + 3)	<b>(\$5007)</b>
(\$5017)	
<b>#\$49</b>	

\$005000    54 AF 18 B9 E7 21 48 **49**        **A1 = \$00005007**

**Exercice 4 (4 points)**

Soit le sous-programme **Select** qui utilise comme registre d'entrée **D1.B** et comme registre de sortie **D0.L**. Inscrivez sur le [document réponse](#) la valeur que prendra le registre **D0.L** après l'exécution du sous-programme **Select** pour les différentes valeurs du registre **D1.B**.

Select	tst.b	d1	; Mise à jour de N et de Z en fonction de D1.B.
	bne.s	\next1	; Si Z = 0 (D1.B ≠ 0), saut à \next1.
	move.l	#100,d0	; Sinon (D1.B = 0), charge 100 dans D0.L.
	rts		; Sortie.
\next1	bmi.s	\next2	; Si N = 1 (D1.B < 0), saut à \next2.
	cmp.b	#\$40,d1	; Sinon D1.B est comparé à \$40 (\$40 = 64).
	bgt.s	\next3	; Si D1.B > \$40, saut à \next3.
	move.l	#200,d0	; Sinon D1.B ≤ \$40, charge 200 dans D0.L.
	rts		; Sortie.
\next2	move.l	#300,d0	; D1.B < 0, charge 300 dans D0.L
	rts		; Sortie.
\next3	move.l	#400,d0	; D1.B > \$40, charge 400 dans D0.L.
	rts		; Sortie.

### **Exercice 5 (3 points)**

Question de cours. Pas de point négatif. Vous inscrirez vos réponses sur le [document réponse](#).

Q1. Choisir l'affirmation correcte :

- (a) **Il existe des exceptions programmées.**
- (b) Une exception s'exécute en mode utilisateur.
- (c) À l'issue du traitement d'une exception, on ne peut pas reprendre l'exécution de programme en cours au moment où elle s'est produite.
- (d) Une demande issue d'un périphérique n'est pas une exception.

Q2. Une interruption est :

- (a) Une exception programmée.
- (b) Une exception d'origine interne uniquement.
- (c) **Une exception d'origine externe.**
- (d) Une anomalie d'exécution.

Q3. Le 68000 permet l'association directe de 199 traitements distincts aux demandes d'interruption, selon :

- (a) **2 modes qui dépendent du périphérique d'où émane la demande.**
- (b) 1 seul mode.
- (c) 199 modes différents (1 mode par demande).
- (d) Aucune de ces réponses.

Q4. Quelle est la différence entre un sous-programme et une exception de type TRAP ?

- (a) Il n'y a pas de différence.
- (b) Le sous-programme s'exécute toujours en mode superviseur.
- (c) **L'exception de type TRAP s'exécute toujours en mode superviseur.**
- (d) Aucune de ces réponses.

Q5. Quelle instruction peut-on utiliser pour revenir d'un TRAP ?

- (a) RETURN
- (b) **RTE**
- (c) BSR
- (d) RTS

Q6. L'*Erreur Bus* est :

- (a) Une exception d'origine interne.
- (b) Une interruption.
- (c) **Une exception d'origine externe.**
- (d) Aucune de ces réponses.

## Integer Instructions

**MOVE****Move Data from Source to Destination**  
**(M68000 Family)****MOVE****Operation:** Source → Destination**Assembler****Syntax:** MOVE < ea > , < ea >**Attributes:** Size = (Byte, Word, Long)**Description:** Moves the data at the source to the destination location and sets the condition codes according to the data. The size of the operation may be specified as byte, word, or long. Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

**Instruction Format:**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		

**Instruction Fields:**

Size field—Specifies the size of the operand to be moved.

01 — Byte operation

11 — Word operation

10 — Long operation

**MOVE****Move Data from Source to Destination  
(M68000 Family)****MOVE**

Destination Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d <sub>16</sub> ,An)	101	reg. number:An
(d <sub>8</sub> ,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d <sub>16</sub> ,PC)	—	—
(d <sub>8</sub> ,PC,Xn)	—	—

**MC68020, MC68030, and MC68040 only**

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

\*Can be used with CPU32.

Source Effective Address field—Specifies the source operand. All addressing modes can be used as listed in the following tables:

Addressing Mode	Mode	Register
Dn	000	reg. number:Dn
An	001	reg. number:An
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (An)	100	reg. number:An
(d <sub>16</sub> ,An)	101	reg. number:An
(d <sub>8</sub> ,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d <sub>16</sub> ,PC)	111	010
(d <sub>8</sub> ,PC,Xn)	111	011

**MC68020, MC68030, and MC68040 only**

(bd,An,Xn)**	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)**	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xn,od)	111	011

\*For byte size operation, address register direct is not allowed.

\*\*Can be used with CPU32.

**NOTE**

Most assemblers use MOVEA when the destination is an address register.

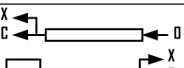
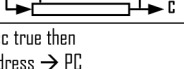
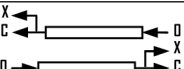
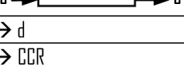
MOVEQ can be used to move an immediate 8-bit value to a data register.

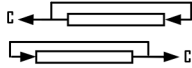
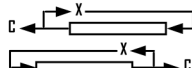


**EASy68K Quick Reference v2.1**

www.easy68k.com

Copyright © 2004-2009 By: Chuck Kelly

Opcode	Size	Operand	CCR	Effective Address s=source, d=destination, e=either, i=displacement												Operation	Description
	BWL	s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(i.An)	(i.An,Rn)	abs.W	abs.L	(i.PC)	(i.PC,Rn)	#n		
ABCD	B	Dy,Dx -(Ay),-(Ax)	*U*U*	e	-	-	-	e	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$ $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	BCD destination + BCD source + eXtend Z cleared if result not 0 unchanged otherwise
ADD <sup>4</sup>	BWL	s,Dn Dn,d	*****	e	s	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	$s + Dn \rightarrow Dn$ $Dn + d \rightarrow d$	Add binary (ADDI or ADDQ is used when source is #n. Prevent ADDQ with #n.L)
ADDA <sup>4</sup>	WL	s,An	-----	s	e	s	s	s	s	s	s	s	s	s	s	$s + An \rightarrow An$	Add address (W sign-extended to .L)
ADDI <sup>4</sup>	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	$\#n + d \rightarrow d$	Add immediate to destination
ADDQ <sup>4</sup>	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	s	$\#n + d \rightarrow d$	Add quick immediate (#n range: 1 to 8)
ADDX	BWL	Dy,Dx -(Ay),-(Ax)	*****	e	-	-	-	e	-	-	-	-	-	-	-	$Dy + Dx + X \rightarrow Dx$ $-(Ay) + -(Ax) + X \rightarrow -(Ax)$	Add source and eXtend bit to destination
AND <sup>4</sup>	BWL	s,Dn Dn,d	-**00	e	-	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	$s \text{ AND } Dn \rightarrow Dn$ $Dn \text{ AND } d \rightarrow d$	Logical AND source to destination (ANDI is used when source is #n)
ANDI <sup>4</sup>	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	s	$\#n \text{ AND } d \rightarrow d$	Logical AND immediate to destination
ANDI <sup>4</sup>	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ AND CCR} \rightarrow \text{CCR}$	Logical AND immediate to CCR
ANDI <sup>4</sup>	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ AND SR} \rightarrow \text{SR}$	Logical AND immediate to SR (Privileged)
ASL	BWL	Dx,Dy	*****	e	-	-	-	-	-	-	-	-	-	-	-		Arithmetic shift Dy by Dx bits left/right
ASR	W	#n,Dy d		d	-	-	-	-	-	-	-	-	-	-	s		Arithmetic shift Dy #n bits L/R (#n: 1 to 8) Arithmetic shift ds 1 bit left/right (W only)
Bcc	BW <sup>3</sup>	address <sup>2</sup>	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc true then address $\rightarrow$ PC	Branch conditionally (cc table on back) (8 or 16-bit $\pm$ offset to address)
BCHG	B L	Dn,d #n,d	--*--	e <sup>1</sup>	-	d	d	d	d	d	d	d	-	-	-	$\text{NOT}(\text{bit number of } d) \rightarrow Z$ $\text{NOT}(\text{bit } n \text{ of } d) \rightarrow \text{bit } n \text{ of } d$	Set Z with state of specified bit in d then invert the bit in d
BCLR	B L	Dn,d #n,d	--*--	e <sup>1</sup>	-	d	d	d	d	d	d	d	-	-	-	$\text{NOT}(\text{bit number of } d) \rightarrow Z$ $0 \rightarrow \text{bit number of } d$	Set Z with state of specified bit in d then clear the bit in d
BFCBG	<sup>5</sup>	d{o:w}	-**00	d	-	d	-	-	d	d	d	d	-	-	-	$0 \rightarrow \text{bit field of } d$	Complement the bit field at destination
BFCBL	<sup>5</sup>	d{o:w}	-**00	d	-	d	-	-	d	d	d	d	-	-	-	$0 \rightarrow \text{bit field of } d$	Clear the bit field at destination
BFEFXTS	<sup>5</sup>	s{o:w},Dn	-**00	d	-	s	-	-	s	s	s	s	s	s	-	bit field of s sign extended $32 \rightarrow Dn$	Dn = bit field of s sign extended to 32 bits
BFEFTU	<sup>5</sup>	s{o:w},Dn	-**00	d	-	s	-	-	s	s	s	s	s	s	-	bit field of s unsigned $\rightarrow Dn$	Dn = bit field of s zero extended to 32 bits
BFFFD	<sup>5</sup>	s{o:w},Dn	-**00	d	-	s	-	-	s	s	s	s	s	s	-	bit number of 1 <sup>st</sup> 1 $\rightarrow Dn$	Dn = bit position of 1 <sup>st</sup> 1 or offset + width
BFINS	<sup>5</sup>	Dn,s{o:w}	-**00	s	-	d	-	-	d	d	d	d	-	-	-	low bits Dn $\rightarrow$ bit field at d	Insert low bits of Dn to bit field at d
BFSET	<sup>5</sup>	d{o:w}	-**00	d	-	d	-	-	d	d	d	d	-	-	-	$1 \rightarrow \text{bit field of } d$	Set all bits in bit field of destination
BFTST	<sup>5</sup>	d{o:w}	-**00	d	-	d	-	-	d	d	d	d	d	d	-	set CCR with bit field of d	N = high bit of bit field, Z set if all bits 0
BRA	BW <sup>3</sup>	address <sup>2</sup>	-----	-	-	-	-	-	-	-	-	-	-	-	-	address $\rightarrow$ PC	Branch always (8 or 16-bit $\pm$ offset to addr)
BSET	B L	Dn,d #n,d	--*--	e <sup>1</sup>	-	d	d	d	d	d	d	d	-	-	-	$\text{NOT}(\text{bit } n \text{ of } d) \rightarrow Z$ $1 \rightarrow \text{bit } n \text{ of } d$	Set Z with state of specified bit in d then set the bit in d
BSR	BW <sup>3</sup>	address <sup>2</sup>	-----	-	-	-	-	-	-	-	-	-	-	-	-	PC $\rightarrow$ -(SP); address $\rightarrow$ PC	Branch to subroutine (8 or 16-bit $\pm$ offset)
BTST	B L	Dn,d #n,d	--*--	e <sup>1</sup>	-	d	d	d	d	d	d	d	d	d	s	$\text{NOT}(\text{bit } Dn \text{ of } d) \rightarrow Z$ $\text{NOT}(\text{bit } \#n \text{ of } d) \rightarrow Z$	Set Z with state of specified bit in d Leave the bit in d unchanged
CHK	W	s,Dn	-*UUU	e	-	s	s	s	s	s	s	s	s	s	s	if $Dn < 0$ or $Dn > s$ then TRAP	Compare Dn with 0 and upper bound [s]
CLR	BWL	d	-0100	d	-	d	d	d	d	d	d	d	-	-	-	$0 \rightarrow d$	Clear destination to zero
CMP <sup>4</sup>	BWL	s,Dn	-****	e	s <sup>4</sup>	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	set CCR with $Dn - s$	Compare Dn to source
CMPA <sup>4</sup>	WL	s,An	-****	s	e	s	s	s	s	s	s	s	s	s	s	set CCR with $An - s$	Compare An to source
CMPI <sup>4</sup>	BWL	#n,d	-****	d	-	d	d	d	d	d	d	d	-	-	s	set CCR with $d - \#n$	Compare destination to #n
CMPPM <sup>4</sup>	BWL	(Ay)+,(Ax)+	-****	-	-	-	e	-	-	-	-	-	-	-	-	set CCR with $(Ax) - (Ay)$	Compare (Ax) to (Ay); Increment Ax and Ay
DBcc	W	Dn,address <sup>2</sup>	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc false then { $Dn - 1 \rightarrow Dn$ if $Dn < -1$ then addr $\rightarrow$ PC }	Test condition, decrement and branch (16-bit $\pm$ offset to address)
DIVS	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	$\pm 32\text{bit } Dn / \pm 16\text{bit } s \rightarrow \pm Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
DIVU	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	$32\text{bit } Dn / 16\text{bit } s \rightarrow Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
EOR <sup>4</sup>	BWL	Dn,d	-**00	e	-	d	d	d	d	d	d	d	-	-	s <sup>4</sup>	$Dn \text{ XOR } d \rightarrow d$	Logical exclusive OR Dn to destination
EORI <sup>4</sup>	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	s	$\#n \text{ XOR } d \rightarrow d$	Logical exclusive OR #n to destination
EORI <sup>4</sup>	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR CCR} \rightarrow \text{CCR}$	Logical exclusive OR #n to CCR
EORI <sup>4</sup>	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR SR} \rightarrow \text{SR}$	Logical exclusive OR #n to SR (Privileged)
EXG	L	Rx,Ry	-----	e	e	-	-	-	-	-	-	-	-	-	-	register $\leftrightarrow$ register	Exchange registers (32-bit only)
EXT	WL	Dn	-**00	d	-	-	-	-	-	-	-	-	-	-	-	$Dn.B \rightarrow Dn.W \mid Dn.W \rightarrow Dn.L$	Sign extend (change .B to .W or .W to .L)
ILLEGAL			-----	-	-	-	-	-	-	-	-	-	-	-	-	PC $\rightarrow$ -(SSP); SR $\rightarrow$ -(SSP)	Generate Illegal Instruction exception
JMP		d	-----	-	-	d	-	-	d	d	d	d	d	d	-	$\uparrow d \rightarrow \text{PC}$	Jump to effective address of destination
JSR		d	-----	-	-	d	-	-	d	d	d	d	d	d	-	PC $\rightarrow$ -(SP); $\uparrow d \rightarrow \text{PC}$	push PC, jump to subroutine at address d
LEA	L	s,An	-----	-	e	s	-	-	s	s	s	s	s	s	-	$\uparrow s \rightarrow An$	Load effective address of s to An
LINK		An,#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	$An \rightarrow \text{-(SP)}; SP \rightarrow An;$ $SP + \#n \rightarrow SP$	Create local workspace on stack (negative n to allocate space)
LSL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, Dx bits left/right
LSR	W	#n,Dy d		d	-	-	-	-	-	-	-	-	-	-	s		Logical shift Dy, #n bits L/R (#n: 1 to 8) Logical shift d 1 bit left/right (W only)
MOVE <sup>4</sup>	BWL	s,d	-**00	e	s <sup>4</sup>	e	e	e	e	e	e	e	s	s	s <sup>4</sup>	$s \rightarrow d$	Move data from source to destination
MOVE	W	s,CCR	=====	s	-	s	s	s	s	s	s	s	s	s	s	$s \rightarrow \text{CCR}$	Move source to Condition Code Register
MOVE	W	s,SR	=====	s	-	s	s	s	s	s	s	s	s	s	s	$s \rightarrow \text{SR}$	Move source to Status Register (Privileged)
MOVE	W	SR,d	-----	d	-	d	d	d	d	d	d	d	-	-	-	$\text{SR} \rightarrow d$	Move Status Register to destination
	BWL	s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(i.An)	(i.An,Rn)	abs.W	abs.L	(i.PC)	(i.PC,Rn)	#n		

Opcode	Size	Operand	CCR	Effective Address s=source, d=destination, e=either, i=displacement											Operation		Description
	BWL	s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(i,An)	(i,An,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n		
MOVE	L	USP,An An,USP	-----	-	d	-	-	-	-	-	-	-	-	-	-	USP → An An → USP	Move User Stack Pointer to An (Privileged) Move An to User Stack Pointer (Privileged)
MOVEA <sup>4</sup>	WL	s,An	-----	s	e	s	s	s	s	s	s	s	s	s	s	s → An	Move source to An (MOVE s,An use MOVEA)
MOVEM <sup>4</sup>	WL	Rn-Rn,d s,Rn-Rn	-----	-	-	d	-	d	d	d	d	d	-	-	-	Registers → d s → Registers	Move specified registers to/from memory (.W source is sign-extended to .L for Rn)
MOVEP	WL	Dn,(i,An) (i,An),Dn	-----	s	-	-	-	-	d	-	-	-	-	-	-	Dn → (i,An)...(i+2,An)...(i+4,A. (i,An) → Dn...(i+2,An)...(i+4,A.	Move Dn to/from alternate memory bytes (Access only even or odd addresses)
MOVEQ <sup>4</sup>	L	#n,Dn	---*00	d	-	-	-	-	-	-	-	-	-	-	s	#n → Dn	Move sign extended 8-bit #n to Dn
MULS	W	s,Dn	---*00	e	-	s	s	s	s	s	s	s	s	s	s	±16bit s * ±16bit Dn → ±Dn	Multiply signed 16-bit; result: signed 32-bit
MULU	W	s,Dn	---*00	e	-	s	s	s	s	s	s	s	s	s	s	16bit s * 16bit Dn → Dn	Multiply unsig'd 16-bit; result: unsig'd 32-bit
NBCD	B	d	*U*U*	d	-	d	d	d	d	d	d	d	-	-	-	0 - d <sub>10</sub> - X → d	Negate BCD with eXtend, BCD result
NEG	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	0 - d → d	Negate destination (2's complement)
NEGX	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	0 - d - X → d	Negate destination with eXtend
NOP			-----	-	-	-	-	-	-	-	-	-	-	-	-	None	No operation occurs
NOT	BWL	d	---*00	d	-	d	d	d	d	d	d	d	-	-	-	NOT( d ) → d	Logical NOT destination (1's complement)
OR <sup>4</sup>	BWL	s,Dn Dn,d	---*00	e	-	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	s OR Dn → Dn Dn OR d → d	Logical OR (ORI is used when source is #n)
ORI <sup>4</sup>	BWL	#n,d	---*00	d	-	d	d	d	d	d	d	d	-	-	s	#n OR d → d	Logical OR #n to destination
ORI <sup>4</sup>	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n OR CCR → CCR	Logical OR #n to CCR
ORI <sup>4</sup>	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n OR SR → SR	Logical OR #n to SR (Privileged)
PEA	L	s	-----	-	-	s	-	-	s	s	s	s	s	s	-	↑s → -(SP)	Push effective address of s onto stack
RESET			-----	-	-	-	-	-	-	-	-	-	-	-	-	Assert RESET Line	Issue a hardware RESET (Privileged)
ROL	BWL	Dx,Dy	---*0*	e	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits left/right (without X)
ROR		#n,Dy		d	-	-	-	-	-	-	-	-	-	-	-		s
	W	d		-	-	d	d	d	d	d	d	d	-	-	-		Rotate d l-bit left/right (.W only)
ROXL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits L/R. X used then updated
RORX		#n,Dy		d	-	-	-	-	-	-	-	-	-	-	-		s
	W	d		-	-	d	d	d	d	d	d	d	-	-	-		Rotate destination l-bit left/right (.W only)
RTE			=====	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → SR; (SP)+ → PC	Return from exception (Privileged)
RTR			=====	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → CCR; (SP)+ → PC	Return from subroutine and restore CCR
RTS			-----	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → PC	Return from subroutine
SBCD	B	Dy,Dx -(Ay),-(Ax)	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	Dx <sub>10</sub> - Dy <sub>10</sub> - X → Dx <sub>10</sub> -(Ax) <sub>10</sub> - -(Ay) <sub>10</sub> - X → -(Ax) <sub>10</sub>	BCD destination - BCD source - eXtend Z cleared if result not 0 unchanged otherwise
SCC	B	d	-----	d	-	d	d	d	d	d	d	d	-	-	-	If cc is true then !s → d else D's → d	If cc true then d.B = 11111111 else d.B = 00000000
STOP		#n	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n → SR; STOP	Move #n to SR, stop processor (Privileged)
SUB <sup>4</sup>	BWL	s,Dn Dn,d	*****	e	s	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	Dn - s → Dn d - Dn → d	Subtract binary (SUB) or SUBQ used when source is #n. Prevent SUBQ with #n.L
SUBA <sup>4</sup>	WL	s,An	-----	s	e	s	s	s	s	s	s	s	s	s	s	An - s → An	Subtract address (.W sign-extended to .L)
SUBI <sup>4</sup>	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	d - #n → d	Subtract immediate from destination
SUBQ <sup>4</sup>	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	s	d - #n → d	Subtract quick immediate (#n range: 1 to 8)
SUBX	BWL	Dy,Dx -(Ay),-(Ax)	*****	e	-	-	-	-	-	-	-	-	-	-	-	Dx - Dy - X → Dx -(Ax) - -(Ay) - X → -(Ax)	Subtract source and eXtend bit from destination
SWAP	W	Dn	---*00	d	-	-	-	-	-	-	-	-	-	-	-	bits[31:16] ↔ bits[15:0]	Exchange the 16-bit halves of Dn
TAS	B	d	---*00	d	-	d	d	d	d	d	d	d	-	-	-	test d → CCR; 1 → bit7 of d	N and Z set to reflect d, bit7 of d set to 1
TRAP		#n	-----	-	-	-	-	-	-	-	-	-	-	-	s	PC → -(SSP);SR → -(SSP); (vector table entry) → PC	Push PC and SR, PC set by vector table #n (#n range: 0 to 15)
TRAPV			-----	-	-	-	-	-	-	-	-	-	-	-	-	If V then TRAP #7	If overflow, execute an Overflow TRAP
TST	BWL	d	---*00	d	-	d	d	d	d	d	d	d	-	-	-	test d → CCR	N and Z set to reflect destination
UNLK		An	-----	-	d	-	-	-	-	-	-	-	-	-	-	An → SP; (SP)+ → An	Remove local workspace from stack
	BWL	s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(i,An)	(i,An,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n		

Condition Tests (+ OR, ! NOT, ⊕ XOR, * Unsigned, * Alternate cc)					
cc	Condition	Test	cc	Condition	Test
T	true	I	VC	overflow clear	IV
F	false	O	VS	overflow set	V
HI <sup>u</sup>	higher than	!(C + Z)	PL	plus	!N
LS <sup>u</sup>	lower or same	C + Z	MI	minus	N
HS <sup>u</sup> , CC <sup>a</sup>	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LO <sup>u</sup> , CS <sup>a</sup>	lower than	C	LT	less than	(N ⊕ V)
NE	not equal	!Z	GT	greater than	![(N ⊕ V) + Z]
EQ	equal	Z	LE	less or equal	(N ⊕ V) + Z

**An** Address register (16/32-bit, n=0-7)  
**Dn** Data register (8/16/32-bit, n=0-7)  
**Rn** any data or address register  
**BCD** Binary Coded Decimal  
**PC** Program Counter (24-bit)  
**↑** Effective address  
**#n** Immediate data  
**SP** Active Stack Pointer (same as A7)  
<sup>1</sup> Long only; all others are byte only  
<sup>2</sup> Assembler calculates offset  
<sup>3</sup> Branch sizes: **B** or **S** -128 to +127 bytes, **W** or **L** -32768 to +32767 bytes  
<sup>4</sup> Assembler automatically uses A, I, Q or M form if possible. Use #n.L to prevent Quick optimization  
<sup>5</sup> Bit field determines size. Not supported by 68000. EASY68K hybrid form of 68020 instruction

**SR** Status Register (16-bit)  
**CCR** Condition Code Register (lower 8-bits of SR)  
**N** negative, **Z** zero, **V** overflow, **C** carry, **X** extend  
 \* set by operation's result, = set directly  
 - not affected, **O** cleared, **I** set, **U** undefined

**USP** User Stack Pointer (32-bit)

Distributed under GNU general public use license

Commonly Used Simulator Input/Output Tasks				TRAP #15 is used to run simulator tasks. Place the task number in register D0. See Help for a complete description of available tasks. (cstring is null terminated)	
<b>0</b>	Display n characters of string at (AI), n=D1.W (stops on NULL or max 255) with CR,LF	<b>1</b>	Display n characters of string at (AI), n=D1.W (stops on NULL or max 255) without CR,LF	<b>2</b>	Read characters from keyboard. Store at (AI). Null terminated. D1.W = length (max 80)
<b>4</b>	Read number from keyboard into D1.L	<b>5</b>	Read single character from keyboard in D1.B	<b>6</b>	Display D1.B as ASCII character
<b>8</b>	time in 1/100 second since midnight → D1.L	<b>9</b>	Terminate the program. (Halts the simulator)	<b>10</b>	Print cstring at (AI) on default printer.
<b>13</b>	Display cstring at (AI) with CR,LF	<b>14</b>	Display cstring at (AI) without CR,LF	<b>15</b>	Display unsigned number in D1.L in D2.B base
<b>18</b>	Display cstring at (AI), read number into D1.L	<b>19</b>	Return state of keys or scan code. See help	<b>20</b>	Display ± number in D1.L, field D2.B columns wide
<b>3</b>	Display D1.L as signed decimal number	<b>7</b>	Set D1.B to 1 if keyboard input pending else set to 0	<b>11</b>	Position cursor at row,col D1.W=ccrr, \$FFF0 clears
<b>17</b>	Display cstring at (AI) , then display number in D1.L	<b>21</b>	Set font properties. See help for details		

**DOCUMENT RÉPONSE À RENDRE AVEC LA COPIE**

**Exercice 4**

<b>Valeur d'entrée (D1.B)</b>	<b>-25</b>	<b>45</b>	<b>0</b>	<b>64</b>
<b>Valeur de sortie (D0.L)</b>	300	200	100	200

**Exercice 5**

<b>Question</b>	<b>Q1</b>	<b>Q2</b>	<b>Q3</b>	<b>Q4</b>	<b>Q5</b>	<b>Q6</b>
<b>Réponse</b>	a	c	a	c	b	c