Contrôle 2 – Corrigé
Architecture des ordinateurs

Durée : 1 h 30

Exercice 1 (4 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.B (A2)+,(A0)`

MOVE (*cf.* documentation ci-annexée)

0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
MOVE		**.B**		**(A0)**						**(A2)+**					

Code machine complet en représentation hexadécimale : **`109A`**

2. `ADDI.W #50,D2`

ADDI (*cf.* documentation ci-annexée)

0	0	0	0	0	1	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		
0	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**
ADDI #<data>								**.W**		**D2**					

- Mot d'extension à ajouter pour la source : **`#<data>`** = `#50` = **`#$0032`**
  La taille de la donnée du mode d'adressage immédiat correspond à la taille de l'instruction. L'instruction possède ici l'extension `.W`. La taille de la donnée est donc de 16 bits.

Code machine complet en représentation hexadécimale : **`0642 0032`**

3. `MOVE.W $6000,14(A1)`

MOVE (*cf.* documentation ci-annexée)

0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		
0	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**
MOVE		**.W**		**d16(A1)**						**(xxx).L**					

- Mot d'extension à ajouter pour la source : **`(xxx).L = $00006000`**
  Un adressage absolu long représente une adresse sur 32 bits non signés.

- Mot d'extension à ajouter pour la destination : **`d16`** = `14` = **`$000E`**
  `d16` représente un déplacement sur 16 bits signés. Il faut donc convertir la valeur `14` en représentation hexadécimale sur 16 bits signés.

Code machine complet en représentation hexadécimale : **`3379 00006000 000E`**

4. `MOVE.W  #$6000,-3(A4)`

MOVE (*cf.* documentation ci-annexée)

0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		
0	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
MOVE		**.W**		**d16(A4)**						**#<data>**					

- Mot d'extension à ajouter pour la source : **`#<data>`** = **`#$6000`**
  La taille de la donnée du mode d'adressage immédiat correspond à la taille de l'instruction. L'instruction possède ici l'extension `.W`. La taille de la donnée est donc de 16 bits.

- Mot d'extension à ajouter pour la destination : **`d16`** = `-3` = **`$FFFD`**
  `d16` représente un déplacement sur 16 bits signés. Il faut donc convertir la valeur `-3` en représentation hexadécimale sur 16 bits signés.

Code machine complet en représentation hexadécimale : **`397C 6000 FFFD`**

Exercice 2 (4 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 = $0000FFFC   A0 = $00005000   PC = $00006000
D1 = $0008000B   A1 = $00005008
D2 = $00000004   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 48
```

1. `MOVE.W  #28,-4(A1)`

Source	Destination
#28 **#$001C**	-4(A1) (A1 - 4) ($5008 - 4) **($5004)**

`$005000  54 AF 18 B9 **00 1C** 48 C0`

2. `MOVE.L  6(A1,D0.W),$5002`

Source	Destination
6(A1,D0.W) (A1 + D0.W + 6) ($5008 - 4 + 6) ($500A) **#$11C8D436**	**($5002)**

`$005000  54 AF **11 C8 D4 36** 48 C0`

3. `MOVE.B  $5005(PC),-5(A2,D2.L)`

Source	Destination
$5005(PC) ($5005) **#$21**	-5(A2,D2.L) (A2 + D2 - 5) ($5010 + 4 - 5) **($500F)**

`$005008  C9 10 11 C8 D4 36 1F **21**`

4. `MOVE.W  9(A0,D1.W),-(A2)`

Source	Destination
`9(A0,D1.W)`	`(A2)`
`(A0 + D1.W + 9)`	`($5010 - 2)`
`($5000 + $B + 9)`	**`($500E)`**
`($5014)`	
`#$421A`	

`$005008  C9 10 11 C8 D4 36` **`42 1A`** **`A2 = $0000500E`**

Exercice 3 (2 points)

Trouvez le nombre manquant pour chaque addition ci-dessous afin d'obtenir la bonne combinaison de *flags* (vous utiliserez la représentation hexadécimale). Si plusieurs solutions sont possibles, vous retiendrez uniquement la plus petite.

1. Addition sur 8 bits : `$60 + $`**`20`** avec **N** = 1, **Z** = 0, **V** = 1, **C** = 0
2. Addition sur 16 bits : `$3A24 + $`**`C5DC`** avec **N** = 0, **Z** = 1, **V** = 0, **C** = 1

- Le *flag* **N** est le bit de signe du résultat. Il prend donc la valeur du bit de poids fort du résultat.
- Le *flag* **Z** est à 1 si le résultat est nul.
- Le *flag* **C** est à 1 s'il y a une retenue.
- Le *flag* **V** se détermine en faisant la **supposition** que les nombres à additionner sont signés. Il est à 1 uniquement si l'une des deux conditions suivantes est vraie :
  - On additionne deux nombres positifs et le résultat est négatif ;
  - On additionne deux nombres négatifs et le résultat est positif.

Exercice 4 (5 points)

Soit les cinq programmes ci-dessous :

```
Prog1       tst.b   d6          ; Mise à jour de N et de Z en fonction de D6.B.
            bmi     quit1       ; Si N = 1 (D6.B < 0), saut à quit1.
            moveq.l #2,d1       ; Sinon, 2 -> D1.
quit1
```

```
Prog2       tst.w   d6          ; Mise à jour de N et de Z en fonction de D6.W.
            bmi     quit2       ; Si N = 1 (D6.W < 0), saut à quit2.
            moveq.l #2,d2       ; Sinon, 2 -> D2.
quit2
```

```
Prog3       move.l  #$FFFF05,d7 ; $FFFF05 -> D7.L (D7.B = $05 = 5)
loop3       addq.l  #1,d3       ; D3 + 1 -> D3
            subq.b  #1,d7       ; D7.B - 1 -> D7.B ; Seul D7.B est décrémenté.
            bne     loop3       ; Saut tant que Z = 0 (D7.B ≠ 0)
quit3
```

```
Prog4       moveq.l #7,d7       ; 7 -> D7
loop4       addq.l  #1,d4       ; D4 + 1 -> D4
            dbra    d7,loop4    ; Décrémente D7.W de 1. Saut tant que D7.W ≠ -1.
quit4
```

```
                                ; D5 = $76543210
Prog5       rol.l   #4,d5       ; D5 = $65432107
            rol.w   #8,d5       ; D5 = $65430721
            ror.b   #4,d5       ; D5 = $65430712
quit5
```

- Chaque programme est indépendant.
- Les valeurs initiales des registres sont identiques pour chaque programme.
- Valeurs initiales des registres :

  D1 = $00000001

  D2 = $00000001

  D3 = $00000000

  D4 = $00000000

  D5 = $76543210

  D6 = $0000C421

1. Quelle sera la valeur du registre **D1** après l'exécution du programme **`Prog1`** ?

   Après l'exécution du programme, **D1** = **`$00000002`**.

2. Quelle sera la valeur du registre **D2** après l'exécution du programme **`Prog2`** ?

   Après l'exécution du programme, **D2** = **`$00000001`**.

3. Quelle sera la valeur du registre **D3** après l'exécution du programme **`Prog3`** ?

   Après l'exécution du programme, **D3** = **`$00000005`**.

4. Quelle sera la valeur du registre **D4** après l'exécution du programme **`Prog4`** ?

   Après l'exécution du programme, **D4** = **`$00000008`**.

5. Quelle sera la valeur du registre **D5** après l'exécution du programme **`Prog5`** ?

   Après l'exécution du programme, **D5** = **`$65430712`**.

Exercice 5 (5 points)

1. Soit les quatre instructions ci-dessous **totalement indépendantes**. Pour chacune d'entre elles, vous prendrez l'état initial de la pile qui vous est proposé sur le document réponse. À partir de cet état initial, vous déterminerez l'état final (juste après l'exécution de l'instruction). Déterminer l'état final revient à remplir, sur le document réponse, les cases laissées vides et/ou à préciser la nouvelle position du pointeur de pile (la position initiale étant représentée par une flèche en pointillés). Le `BSR` est codé sur deux mots de 16 bits et l'état final de la pile devra être celui qui est présent juste avant l'exécution de la première instruction du sous-programme **`PRINT`**.

```
① 00A000 MOVEM.W D5/A3/D1,-(A7)
② 8E8470 BSR     PRINT
③ 012A80 MOVEM.L (A7)+,A6/D0
④ 00B410 RTS
```

Valeurs initiales des registres :

```
D1 = $11223344
D5 = $55667788
A3 = $AABBCCDD
```

2. Après l'exécution de l'instruction ③, quelles valeurs prendront les registres **D0** et **A6** ? Vous inscrirez vos réponses sur le document réponse.

   La première valeur qui sera dépilée sur 32 bits sera chargée dans le registre **D0**. La seconde valeur sera chargée dans le registre **A6**.

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 REGISTER	DESTINATION MODE	SOURCE MODE	SOURCE 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_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,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_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,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.

ADDI — Add Immediate (M68000 Family) — ADDI

Operation: Immediate Data + Destination → Destination

Assembler Syntax: ADDI # < data > , < ea >

Attributes: Size = (Byte, Word, Long)

Description: Adds the immediate data to the destination operand and stores the result in the destination location. The size of the operation may be specified as byte, word, or long. The size of the immediate data matches the operation size.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set the same as the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	SIZE		EFFECTIVE ADDRESS					
										MODE			REGISTER		
16-BIT WORD DATA										8-BIT BYTE DATA					
32-BIT LONG DATA															

Instruction Fields:

Size field—Specifies the size of the operation.
00 — Byte operation
01 — Word operation
10 — Long operation

Effective Address field—Specifies the destination operand. 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_{16},An)	101	reg. number:An
(d_8,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xn)	—	—

MC68020, MC68030, and MC68040 only

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

*Can be used with CPU32

Immediate field—Data immediately following the instruction.
If size = 00, the data is the low-order byte of the immediate word.
If size = 01, the data is the entire immediate word.
If size = 10, the data is the next two immediate words.

Integer Instructions

MOVEM — Move Multiple Registers (M68000 Family) — MOVEM

Operation: Registers → Destination; Source → Registers

Assembler Syntax: MOVEM < list > , < ea >
MOVEM < ea > , < list >

Attributes: Size = (Word, Long)

Description: Moves the contents of selected registers to or from consecutive memory locations starting at the location specified by the effective address. A register is selected if the bit in the mask field corresponding to that register is set. The instruction size determines whether 16 or 32 bits of each register are transferred. In the case of a word transfer to either address or data registers, each word is sign-extended to 32 bits, and the resulting long word is loaded into the associated register.

Selecting the addressing mode also selects the mode of operation of the MOVEM instruction, and only the control modes, the predecrement mode, and the postincrement mode are valid. If the effective address is specified by one of the control modes, the registers are transferred starting at the specified address, and the address is incremented by the operand length (2 or 4) following each transfer. The order of the registers is from D0 to D7, then from A0 to A7.

If the effective address is specified by the predecrement mode, only a register-to-memory operation is allowed. The registers are stored starting at the specified address minus the operand length (2 or 4), and the address is decremented by the operand length following each transfer. The order of storing is from A7 to A0, then from D7 to D0. When the instruction has completed, the decremented address register contains the address of the last operand stored. For the MC68020, MC68030, MC68040, and CPU32, if the addressing register is also moved to memory, the value written is the initial register value decremented by the size of the operation. The MC68000 and MC68010 write the initial register value (not decremented).

If the effective address is specified by the postincrement mode, only a memory-to-register operation is allowed. The registers are loaded starting at the specified address; the address is incremented by the operand length (2 or 4) following each transfer. The order of loading is the same as that of control mode addressing. When the instruction has completed, the incremented address register contains the address of the last operand loaded plus the operand length. If the addressing register is also loaded from memory, the memory value is ignored and the register is written with the postincremented effective address.

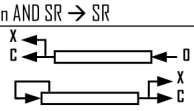
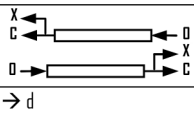
Register List Mask field—Specifies the registers to be transferred. The low-order bit corresponds to the first register to be transferred; the high-order bit corresponds to the last register to be transferred. Thus, for both control modes and postincrement mode addresses, the mask correspondence is:

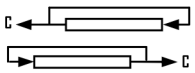
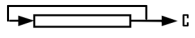
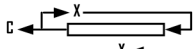
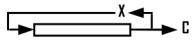
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

For the predecrement mode addresses, the mask correspondence is reversed:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

EASy68K Quick Reference v2.1 **www.easy68k.com**

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	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$	BCD destination + BCD source + eXtend
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	$-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	Z cleared if result not 0 unchanged otherwise
ADD [4]	BWL	s,Dn	*****	e	s	s	s	s	s	s	s	s	s	s	s[4]	s + Dn → Dn	Add binary (ADDI or ADDQ is used when source is
		Dn,d		e	d[4]	d	d	d	d	d	d	d	-	-	-	Dn + d → d	#n. Prevent ADDQ with #n.L)
ADDA [4]	WL	s,An	-----	s	e	s	s	s	s	s	s	s	s	s	s	s + An → An	Add address (.W sign-extended to .L)
ADDI [4]	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	#n + d → d	Add immediate to destination
ADDQ [4]	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	s	#n + d → d	Add quick immediate (#n range: 1 to 8)
ADDX	BWL	Dy,Dx	*****	e	-	-	-	-	-	-	-	-	-	-	-	Dy + Dx + X → Dx	Add source and eXtend bit to destination
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ay) + -(Ax) + X → -(Ax)	
AND [4]	BWL	s,Dn	-**00	e	-	s	s	s	s	s	s	s	s	s	s[4]	s AND Dn → Dn	Logical AND source to destination
		Dn,d		e	-	d	d	d	d	d	d	d	-	-	-	Dn AND d → d	(ANDI is used when source is #n)
ANDI [4]	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	s	#n AND d → d	Logical AND immediate to destination
ANDI [4]	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n AND CCR → CCR	Logical AND immediate to CCR
ANDI [4]	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n AND SR → SR	Logical AND immediate to SR (Privileged)
ASL	BWL	Dx,Dy	*****	e	-	-	-	-	-	-	-	-	-	-	-		Arithmetic shift Dy by Dx bits left/right
ASR		#n,Dy		d	-	-	-	-	-	-	-	-	-	-	s		Arithmetic shift Dy #n bits L/R (#n: 1 to 8)
	W	d		-	-	d	d	d	d	d	d	d	-	-	-		Arithmetic shift ds 1 bit left/right (.W only)
Bcc	BW[3]	address[2]	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc true then address → PC	Branch conditionally (cc table on back) (8 or 16-bit ± offset to address)
BCHG	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	-	-	-	NOT(bit number of d) → Z	Set Z with state of specified bit in d then invert
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	NOT(bit n of d)→ bit n of d	the bit in d
BCLR	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	-	-	-	NOT(bit number of d) → Z	Set Z with state of specified bit in d then clear
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	0 → bit number of d	the bit in d
BFCHG	[5]	d{o:w}	-**00	d	-	d	-	-	d	d	d	d	-	-	-	NOT bit field of d	Complement the bit field at destination
BFCLR	[5]	d{o:w}	-**00	d	-	d	-	-	d	d	d	d	-	-	-	0 → bit field of d	Clear the bit field at destination
BFEXTS	[5]	s{o:w},Dn	-**00	d	-	s	-	-	s	s	s	s	s	s	-	bit field of s extend 32 → Dn	Dn = bit field of s sign extended to 32 bits
BFEXTU	[5]	s{o:w},Dn	-**00	d	-	s	-	-	s	s	s	s	s	s	-	bit field of s unsigned → Dn	Dn = bit field of s zero extended to 32 bits
BFFFO	[5]	s{o:w},Dn	-**00	d	-	s	-	-	s	s	s	s	s	s	-	bit number of 1st 1 → Dn	Dn = bit position of 1st 1 or offset + width
BFINS	[5]	Dn,s{o:w}	-**00	s	-	d	-	-	d	d	d	d	-	-	-	low bits Dn → bit field at d	Insert low bits of Dn to bit field at d
BFSET	[5]	d{o:w}	-**00	d	-	d	-	-	d	d	d	d	-	-	-	1 → bit field of d	Set all bits in bit field of destination
BFTST	[5]	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[3]	address[2]	-----	-	-	-	-	-	-	-	-	-	-	-	-	address → PC	Branch always (8 or 16-bit ± offset to addr)
BSET	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	-	-	-	NOT(bit n of d) → Z	Set Z with state of specified bit in d then
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	1 → bit n of d	set the bit in d
BSR	BW[3]	address[2]	-----	-	-	-	-	-	-	-	-	-	-	-	-	PC → -(SP); address → PC	Branch to subroutine (8 or 16-bit ± offset)
BTST	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	d	d	-	NOT(bit Dn of d) → Z	Set Z with state of specified bit in d
		#n,d		d[1]	-	d	d	d	d	d	d	d	d	d	s	NOT(bit #n of d) → Z	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 → d	Clear destination to zero
CMP [4]	BWL	s,Dn	-****	e	s[4]	s	s	s	s	s	s	s	s	s	s[4]	set CCR with Dn – s	Compare Dn to source
CMPA [4]	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 [4]	BWL	#n,d	-****	d	-	d	d	d	d	d	d	d	-	-	s	set CCR with d - #n	Compare destination to #n
CMPM [4]	BWL	(Ay)+,(Ax)+	-****	-	-	-	e	-	-	-	-	-	-	-	-	set CCR with (Ax) - (Ay)	Compare (Ax) to (Ay); Increment Ax and Ay
DBcc	W	Dn,addres[2]	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc false then { Dn-1 → Dn if Dn <> -1 then addr →PC }	Test condition, decrement and branch (16-bit ± offset to address)
DIVS	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	±32bit Dn / ±16bit s → ±Dn	Dn= [16-bit remainder, 16-bit quotient]
DIVU	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	32bit Dn / 16bit s → Dn	Dn= [16-bit remainder, 16-bit quotient]
EOR [4]	BWL	Dn,d	-**00	e	-	d	d	d	d	d	d	d	-	-	s[4]	Dn XOR d → d	Logical exclusive OR Dn to destination
EORI [4]	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	s	#n XOR d → d	Logical exclusive OR #n to destination
EORI [4]	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n XOR CCR → CCR	Logical exclusive OR #n to CCR
EORI [4]	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n XOR SR → SR	Logical exclusive OR #n to SR (Privileged)
EXG	L	Rx,Ry	-----	e	e	-	-	-	-	-	-	-	-	-	-	register ←→ register	Exchange registers (32-bit only)
EXT	WL	Dn	-**00	d	-	-	-	-	-	-	-	-	-	-	-	Dn.B → Dn.W | Dn.W → Dn.L	Sign extend (change .B to .W or .W to .L)
ILLEGAL			-----	-	-	-	-	-	-	-	-	-	-	-	-	PC→-(SSP); SR→-(SSP)	Generate Illegal Instruction exception
JMP		d	-----	-	-	d	-	-	d	d	d	d	d	d	-	↑d → PC	Jump to effective address of destination
JSR		d	-----	-	-	d	-	-	d	d	d	d	d	d	-	PC → -(SP); ↑d → PC	push PC, jump to subroutine at address d
LEA	L	s,An	-----	-	e	s	-	-	s	s	s	s	s	s	-	↑s → An	Load effective address of s to An
LINK		An,#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	An → -(SP); SP → An; SP + #n → 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		#n,Dy		d	-	-	-	-	-	-	-	-	-	-	s		Logical shift Dy, #n bits L/R (#n: 1 to 8)
	W	d		-	-	d	d	d	d	d	d	d	-	-	-		Logical shift d 1 bit left/right (.W only)
MOVE [4]	BWL	s,d	-**00	e	s[4]	e	e	e	e	e	e	e	s	s	s[4]	s → d	Move data from source to destination
MOVE	W	s,CCR	=====	s	-	s	s	s	s	s	s	s	s	s	s	s → CCR	Move source to Condition Code Register
MOVE	W	s,SR	=====	s	-	s	s	s	s	s	s	s	s	s	s	s → SR	Move source to Status Register (Privileged)
MOVE	W	SR,d	-----	d	-	d	d	d	d	d	d	d	-	-	-	SR → 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	-----	-	d	-	-	-	-	-	-	-	-	-	-	USP → An	Move User Stack Pointer to An (Privileged)
		An,USP		-	s	-	-	-	-	-	-	-	-	-	-	An → USP	Move An to User Stack Pointer (Privileged)
MOVEA[4]	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[4]	WL	Rn-Rn,d	-----	-	-	d	-	d	d	d	d	d	-	-	-	Registers → d	Move specified registers to/from memory
		s,Rn-Rn		-	-	s	s	-	s	s	s	s	s	s	-	s → Registers	(.W source is sign-extended to .L for Rn)
MOVEP	WL	Dn,(i,An)	-----	s	-	-	-	-	d	-	-	-	-	-	-	Dn → (i,An)...(i+2,An)...(i+4,A.	Move Dn to/from alternate memory bytes
		(i,An),Dn		d	-	-	-	-	s	-	-	-	-	-	-	(i,An) → Dn...(i+2,An)...(i+4,A.	(Access only even or odd addresses)
MOVEQ[4]	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_{10} - 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[4]	BWL	s,Dn	-**00	e	-	s	s	s	s	s	s	s	s	s	s[4]	s OR Dn → Dn	Logical OR
		Dn,d		e	-	d	d	d	d	d	d	d	-	-	-	Dn OR d → d	(ORI is used when source is #n)
ORI[4]	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	s	#n OR d → d	Logical OR #n to destination
ORI[4]	B	#n,CCR	≡≡≡≡≡	-	-	-	-	-	-	-	-	-	-	-	s	#n OR CCR → CCR	Logical OR #n to CCR
ORI[4]	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		Rotate Dy, #n bits left/right (#n: 1 to 8)
	W	d		-	-	d	d	d	d	d	d	d	-	-	-		Rotate d 1-bit left/right (.W only)
ROXL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits L/R, X used then updated
ROXR		#n,Dy		d	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, #n bits left/right (#n: 1 to 8)
	W	d		-	-	d	d	d	d	d	d	d	-	-	-		Rotate destination 1-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	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	Dx_{10} - Dy_{10} - X → Dx_{10}	BCD destination – BCD source – eXtend
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ax)$_{10}$ - -(Ay)$_{10}$ – X →-(Ax)$_{10}$	Z cleared if result not 0 unchanged otherwise
Scc	B	d	-----	d	-	d	d	d	d	d	d	d	-	-	-	If cc is true then 1's → d else 0'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[4]	BWL	s,Dn	*****	e	s	s	s	s	s	s	s	s	s	s	s[4]	Dn - s → Dn	Subtract binary (SUBI or SUBQ used when
		Dn,d		e	d[4]	d	d	d	d	d	d	d	-	-	-	d - Dn → d	source is #n. Prevent SUBQ with #n.L)
SUBA[4]	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[4]	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	d - #n → d	Subtract immediate from destination
SUBQ[4]	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	*****	e	-	-	-	-	-	-	-	-	-	-	-	Dx - Dy - X → Dx	Subtract source and eXtend bit from destination
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ax) - -(Ay) – X → -(Ax)	
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; [u] Unsigned, [a] Alternate cc)					
cc	**Condition**	**Test**	**cc**	**Condition**	**Test**
T	true	1	VC	overflow clear	!V
F	false	0	VS	overflow set	V
HI[u]	higher than	!(C + Z)	PL	plus	!N
LS[u]	lower or same	C + Z	MI	minus	N
HS[u], CC[a]	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LO[u], CS[a]	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) **s** Source,
Dn Data register (8/16/32-bit, n=0-7) **d** Destination
Rn any data or address register **e** Either source or destination
BCD Binary Coded Decimal **i** Displacement
PC Program Counter (24-bit) ↑ Effective address
#n Immediate data **{a:w}** offset:width of bit field
SP Active Stack Pointer (same as A7) **SSP** Supervisor Stack Pointer (32-bit) **USP** User Stack Pointer (32-bit)
[1] Long only; all others are byte only [2] Assembler calculates offset
[3] Branch sizes: **.B** or **.S** -128 to +127 bytes, **.W** or **.L** -32768 to +32767 bytes
[4] Assembler automatically uses A, I, Q or M form if possible. Use #n.L to prevent Quick optimization
[5] 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, **0** cleared, **1** set, **U** undefined

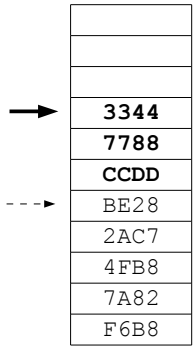
Commonly Used Simulator Input/Output Tasks TRAP #15 is used to run simulator tasks. Place the task number in register D0. See Help for a complete description of available tasks. (cstring is null terminated)

0	Display n characters of string at (A1), n=D1.W (stops on NULL or max 255) with CR,LF	**1**	Display n characters of string at (A1), n=D1.W (stops on NULL or max 255) without CR,LF	**2**	Read characters from keyboard. Store at (A1). Null terminated. D1.W = length (max 80)	**3**	Display D1.L as signed decimal number
4	Read number from keyboard into D1.L	**5**	Read single character from keyboard in D1.B	**6**	Display D1.B as ASCII character	**7**	Set D1.B to 1 if keyboard input pending else set to 0
8	time in 1/100 second since midnight →D1.L	**9**	Terminate the program. (Halts the simulator)	**10**	Print cstring at (A1) on default printer.	**11**	Position cursor at row,col D1.W=ccrr, $FF00 clears
13	Display cstring at (A1) with CR,LF	**14**	Display cstring at (A1) without CR,LF	**15**	Display unsigned number in D1.L in D2.B base	**17**	Display cstring at (A1) , then display number in D1.L
18	Display cstring at (A1), read number into D1.L	**19**	Return state of keys or scan code. See help	**20**	Display ± number in D1.L, field D2.B columns wide	**21**	Set font properties. See help for details

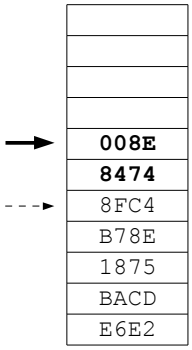
DOCUMENT RÉPONSE À RENDRE AVEC LA COPIE

Exercice 5

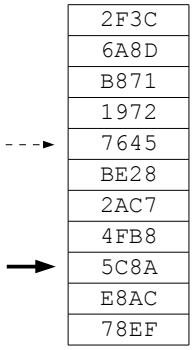
Instruction ①
État initial de la pile



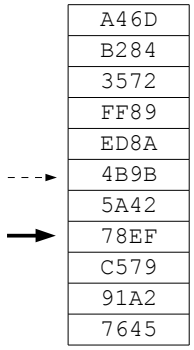
Instruction ②
État initial de la pile



Instruction ③
État initial de la pile



Instruction ④
État initial de la pile



D0 = $7645BE28
A6 = $2AC74FB8