

Contrôle 2

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)
2. ADDI.W #50, D2
3. MOVE.W \$6000, 14(A1)
4. MOVE.W #\$6000, -3(A4)

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)
2. MOVE.L 6(A1, D0.W), \$5002
3. MOVE.B \$5005(PC), -5(A2, D2.L)
4. MOVE.W 9(A0, D1.W), -(A2)

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 + \$?? avec N = 1, Z = 0, V = 1, C = 0
2. Addition sur 16 bits : \$3A24 + \$???? avec N = 0, Z = 1, V = 0, C = 1

Exercice 4 (5 points)

Soit les cinq programmes ci-dessous :

```
Prog1      tst.b    d6
          bmi      quit1
          moveq.l #2,d1
quit1
```

```
Prog2      tst.w    d6
          bmi      quit2
          moveq.l #2,d2
quit2
```

```
Prog3      move.l   #$FFFF05,d7
loop3     addq.l   #1,d3
          subq.b   #1,d7
          bne      loop3
quit3
```

```
Prog4      moveq.l #7,d7
loop4     addq.l   #1,d4
          dbra     d7,loop4
quit4
```

```
Prog5      rol.l    #4,d5
          rol.w    #8,d5
          ror.b    #4,d5
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** ?
2. Quelle sera la valeur du registre **D2** après l'exécution du programme **Prog2** ?
3. Quelle sera la valeur du registre **D3** après l'exécution du programme **Prog3** ?
4. Quelle sera la valeur du registre **D4** après l'exécution du programme **Prog4** ?
5. Quelle sera la valeur du registre **D5** après l'exécution du programme **Prog5** ?

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](#).

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		REGISTER		DESTINATION		MODE		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

Integer Instructions

MOVE Move Data from Source to Destination **MOVE**

(M68000 Family)

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							
16-BIT WORD DATA												EFFECTIVE ADDRESS			
												MODE	REGISTER		
32-BIT LONG DATA												8-BIT BYTE 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,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

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 v2.1www.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 - - - -	s - - - -	(An) - - - -	(An) - - - -	(i,An) - - - -	(i,An,Rn) - - - -	abs.W - - - -	abs.L - - - -	(i,PC) - - - -	(i,PC,Rn) - - - -	- - - -	Dy ₁₀ + Dx ₁₀ + X → Dx ₁₀ -(Ay) ₁₀ + -(Ax) ₁₀ + X → -(Ax) ₁₀	Z cleared if result not 0 unchanged otherwise			
ADD ⁴	BWL	s,Dn Dn,d	*****	e s s s s s s s s s s s s	d d d d d d d d d d d d	s s s s s s s s s s s s	s s s s s s s s s s s s	s s s s s s s s s s s s	s s s s s s s s s s s s	s s s s s s s s s s s s	s s s s s s s s s s s s	s ⁴ s + Dn → Dn Dn + d → d		Add binary (ADDI or ADDQ is used when source is #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			
ADDI ⁴	BWL	#n,d	*****	d - d d d d d d d d d d												s #n + d → d	Add immediate to destination		
ADDO ⁴	BWL	#n,d	*****	d d d d d d d d d d d d												s #n + d → d	Add quick immediate (#n range: I to 8)		
ADDX	BWL	Dy,Dx -(Ay).-(Ax)	*****	e - - - -	s - - - -	(An) - - - -	(An) - - - -	(i,An) - - - -	(i,An,Rn) - - - -	abs.W - - - -	abs.L - - - -	(i,PC) - - - -	(i,PC,Rn) - - - -	- - - -	Dy + Dx + X → Dx -(Ay) + -(Ax) + X → -(Ax)				
AND ⁴	BWL	s,Dn Dn,d	-**00	e - - - -	s s s s s s s s s s s s	d d d d d d d d d d d d	s s s s s s s s s s s s	s s s s s s s s s s s s	s s s s s s s s s s s s	s s s s s s s s s s s s	s s s s s s s s s s s s	s ⁴ s AND Dn → Dn Dn AND d → d		Logical AND source to destination (ANDI is used when source is #n)					
ANDI ⁴	BWL	#n,d	-**00	d - d d 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 - - - -	s - - - -	(An) - - - -	(An) - - - -	(i,An) - - - -	(i,An,Rn) - - - -	abs.W - - - -	abs.L - - - -	(i,PC) - - - -	(i,PC,Rn) - - - -	x c l r		Arithmetic shift Dy by Dx bits left/right			
ASR	BWL	#n,Dy	*****	d - - - -	d - - - -	(An) - - - -	(An) - - - -	(i,An) - - - -	(i,An,Rn) - - - -	abs.W - - - -	abs.L - - - -	(i,PC) - - - -	(i,PC,Rn) - - - -	x c l r		Arithmetic shift Dy #n bits L/R (#n: I to 8)			
ASR	W	d	-----	d - - - -	d - - - -	d - - - -	d - - - -	d - - - -	d - - - -	d - - - -	d - - - -	d - - - -	d - - - -	x c l r		Arithmetic shift ds 1 bit left/right (W only)			
Bcc	BW ³	address ⁷	-----	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- if cc true then address → PC		Branch conditionally (cc table on back) (8 or 16-bit ± offset to address)		
BCHG	B L	Dn,d #n,d	--*--	e ^l - d d d d d d d d d d d d	d ^l - d d d d d d d d d d d d										s NOT(bit number of d) → Z NOT(bit n of d) → bit n 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 ^l - d d d d d d d d d d d d	d ^l - d d d d d d d d d d d d										s NOT(bit number of d) → Z 0 → bit number of d		Set Z with state of specified bit in d then clear the bit in d		
BFCHG	5	d{0:w}	-**00	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	s NOT bit field of d		Complement the bit field at destination		
BFCLR	5	d{0:w}	-**00	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	s 0 → bit field of d		Clear the bit field at destination		
BFEEXTS	5	s{0:w},Dn	-**00	d - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s bit field of s extend 32 → Dn		Dn = bit field of s sign extended to 32 bits		
BFEEXTU	5	s{0:w},Dn	-**00	d - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	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{0:w},Dn	-**00	d - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s - s - - -	s bit number of 1st I → Dn		Dn = bit position of 1st I or offset + width		
BFINS	5	Dn,s{0:w}	-**00	s - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	s low bits Dn → bit field at d		Insert low bits of Dn to bit field at d		
BFSET	5	d{0:w}	-**00	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	s I → bit field of d		Set all bits in bit field of destination		
BFTST	5	d{0:w}	-**00	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	d - d - - -	s N = high bit of bit field, Z set if all bits 0				
BRA	BW ³	address ⁷	-----	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	s address → PC		Branch always (8 or 16-bit ± offset to addr)		
BSET	B L	Dn,d #n,d	--*--	e ^l - d d d d d d d d d d d d	d ^l - d d d d d d d d d d d d										s NOT(bit n of d) → Z I → bit n of d		Set Z with state of specified bit in d then set the bit in d		
BSR	BW ³	address ⁷	-----	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	s PC → -(SP); address → PC		Branch to subroutine (8 or 16-bit ± offset)		
BTST	B L	Dn,d #n,d	--*--	e ^l - d d d d d d d d d d d d	d ^l - d d d d d d d d d d d d										s NOT(bit Dn of d) → Z NOT(bit #n of d) → 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 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 d d d d											s 0 → d		Clear destination to zero		
CMP ⁴	BWL	s,Dn	*****	e s ⁴ s s s 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 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 d d d d											s set CCR with d - #n		Compare destination to #n		
CMPM ⁴	BWL	(Ay)+(Ax)+	*****	- - - e - - -	- - - - -	- - - - -	- - - - -	- - - - -	- - - - -	- - - - -	- - - - -	- - - - -	- - - - -	- - - - -	s set CCR with (Ax) - (Ay)		Compare (Ax) to (Ay); Increment Ax and Ay		
DBcc	W	Dn,address ⁷	-----	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	s if cc false then { Dn-I → Dn if Dn <> -I 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 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 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 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 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 - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	s register ↔ register		Exchange registers (32-bit only)		
EXT	WL	Dn	-**00	d - - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	s Dn.B → Dn.W Dn.W → Dn.L		Sign extend (change B to W or W to L)		
ILLEGAL			-----	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	s PC → -(SSP); SR → -(SSP)		Generate illegal instruction exception		
JMP	d		-----	- - d - -	- - - -	d d d d d d d d d d d d									s ↑d → PC		Jump to effective address of destination		
JSR	d		-----	- - d - -	- - - -	d d d d d d d d d d d d									s 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 s s s s s									s ↑s → An		Load effective address of s to An		
LINK		An,#n	-----	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	s An → -(SP); SP → An; SP + #n → SP		Create local workspace on stack (negative n to allocate space)		
LSL	BWL	Dx,Dy	***0*	e - - - -	d - - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	x c l r		Logical shift Dy, Dx bits left/right		
LSR	BWL	#n,Dy	***0*	d - - - -	d - - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	x c l r		Logical shift Dy, #n bits L/R (#n: I to 8)		
MOVE ⁴	BWL	s,d	-**00	e s ⁴ e e e e e e s s 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 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 s											s s → SR		Move source to Status Register (Privileged)		
MOVE	W	SR,d	-----	d - d d d d d d d d d d d											s 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 An,USP	-----	-d	-	-	-	-	-	-	-	-	-	-	-	USP → An An → USP	Move User Stack Pointer to An (Privileged) 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 (MOVEA s,An use MOVEA)
MOVEM ⁴	WL	Rn-Rn,d s,Rn-Rn	-----	-	-d	-	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 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 ₀ - 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 Dn,d	-**00	e	-	s	s	s	s	s	s	s	s	s	s ⁴	s OR Dn → Dn (OR is used when source is #n)	Logical OR
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 #n,Dy d	-**0*	e	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, Dx bits left/right (without X)
ROR	W			d	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, #n bits left/right (#n: I to 8)
ROXL	BWL	Dx,Dy #n,Dy d	***0*	e	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, Dx bits L/R, X used then updated
ROXR	W			d	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, #n bits left/right (#n: I to 8)
ROXR																	Rotate destination I-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	-	-	-	-	-	-	-	-	-	-	s	D ₁₀ - D ₁₀ - X → D ₁₀ (Ax) ₁₀ - (Ay) ₁₀ - X → -(Ax) ₁₀	BCD destination - BCD source - eXtend Z cleared if result not 0 unchanged otherwise
Scc	B	d	-----	d	-	d	d	d	d	d	d	d	d	-	s	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 Dn,d	****	e	s	s	s	s	s	s	s	s	s	s	s ⁴	Dn - s → Dn Dn - s → Dn d - Dn → d	Subtract binary (SUBI or SUBQ used when 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	d	-	s	d - #n → d	Subtract immediate from destination
SUBQ ⁴	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	d	-	s	d - #n → d	Subtract quick immediate (#n range: I to 8)
SUBX	BWL	Dy,Dx (Ay),(Ax)	*****	e	-	-	-	-	-	-	-	-	-	-	s	Dx - Dy - X → Dx (Ax) - (Ay) - X → -(Ax)	Subtract source and eXtend bit from destination
SWAP	W	Dn	-**00	d	-	-	-	-	-	-	-	-	-	-	s	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	d	-	s	test d → CCR; I → bit7 of d N and Z set to reflect d, bit7 of d set to I	N and Z set to reflect destination
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			-----	-	-	-	-	-	-	-	-	-	-	-	s	If V then TRAP #7	If overflow, execute an Overflow TRAP
TST	BWL	d	-**00	d	-	d	d	d	d	d	d	d	d	-	s	test d → CCR	N and Z set to reflect destination
UNLK		An	-----	-	d	-	-	-	-	-	-	-	-	-	s	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	cc	Condition	Test	cc	Condition	Test	cc	Condition	Test	cc	Condition	Test
T	true	I	VC	overflow clear	IV				Rn	any data or address register					D	Destination	
F	false	0	VS	overflow set	V										E	Either source or destination	
H ^u	higher than	!(C + Z)	PL	plus	IN										i	Displacement	
L ^u	lower or same	C + Z	MI	minus	N										↑	Effective address	
HS ^u , CC ^a	higher or same	!C	GE	greater or equal	(N ⊕ V)										{a:w}	offset:width of bit field	
LO ^u , CS ^a	lower than	C	LT	less than	(N ⊕ V)										SSP	Supervisor Stack Pointer (32-bit)	USP User Stack Pointer (32-bit)
NE	not equal	!Z	GT	greater than	!(N ⊕ V) + Z												
EQ	equal	Z	LE	less or equal	(N ⊕ V) + Z												

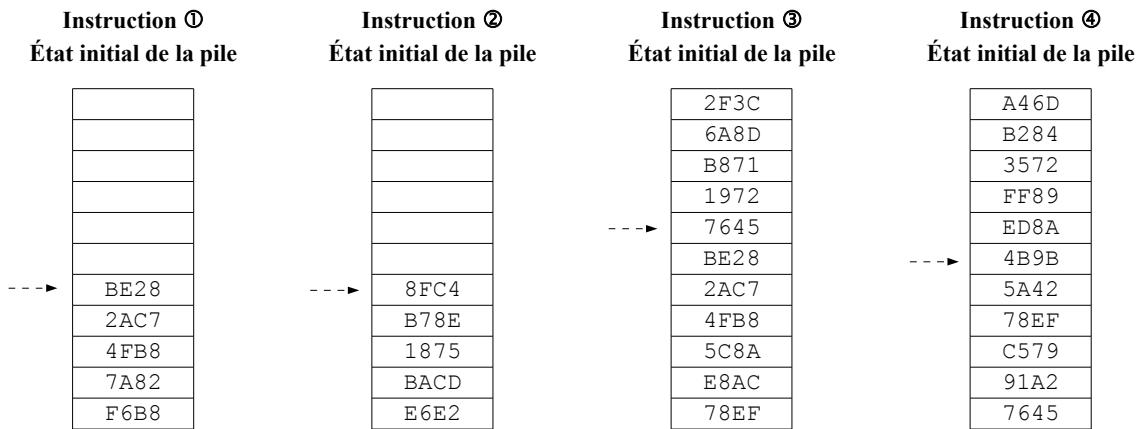
Commonly Used Simulator Input/Output Tasks TRAP #15 is used to run simulator tasks. Place the task number in register D0. See Help for a complete description of available tasks. (cstring is null terminated)

1	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=crr, \$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

Nom : Prénom : Classe :

DOCUMENT RÉPONSE À RENDRE AVEC LA COPIE

Exercice 5



D0 = \$ _____

A6 = \$ _____