Architecture des ordinateurs
Contrôle 2 – Corrigé

Exercice 1 (4 points)

Codez les instructions suivantes 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 D4,(A6)`

MOVE (*cf.* documentation ci-annexée)

0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		
0	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
MOVE		**.B**		**(A6)**						**D4**					

Code machine complet en représentation hexadécimale : **1C84**

2. `SUBX.L -(A3),-(A2)`

SUBX (*cf.* documentation ci-annexée)

1	0	0	1	REGISTER Dy/Ay			1	SIZE		0	0	R/M	REGISTER Dx/Ax		
1	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**
SUBX				**y = 2**				**.L**		**-(Ax),-(Ay)**			**x = 3**		

Code machine complet en représentation hexadécimale : **958B**

3. `MOVE.L #$19,$12(A4)`

MOVE (*cf.* documentation ci-annexée)

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

Information à ajouter pour la source : **#<data>** = #$19 = **#$00000019**
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.
Information à ajouter pour la destination : **d16 = $0012**

Code machine complet en représentation hexadécimale : **297C 00000019 0012**

4. MOVE.W #$19,$12(A4,D5.W)

MOVE (*cf.* documentation ci-annexée)

0	0	SIZE		DESTINATION						SOURCE					
				REGISTER			MODE			MODE			REGISTER		
0	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
MOVE		**.W**		**d8(A4,Xn)**						**#<data>**					

Information à ajouter pour la source : **#<data>** = #$19 = **#$0019**
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.

Il y a deux informations à ajouter pour la destination : la valeur de **d8** et la valeur de **Xn**. Ces deux informations doivent être contenues dans ce qui s'appelle le **mot d'extension**.

Mot d'extension du 68000 (*cf.* documentation ci-annexée)

D/A	REGISTER			W/L	0	0	0	DISPLACEMENT INTEGER							
0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**
D5				**.W**				**d8 = $12**							

Les 5 bits de poids fort du mot d'extension servent à identifier le registre Xn et les 8 bits de poids faible contiennent la valeur de d8. d8 est un déplacement codé sur 8 bits signés.
La représentation hexadécimale du mot d'extension est : **5012**

Code machine complet en représentation hexadécimale : **39BC 0019 5012**

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**.

Attention : <u>La mémoire et les registres sont réinitialisés à chaque nouvelle instruction</u>.

<u>Valeurs initiales</u> :
```
D0 = $FFFFFFFF  A0 = $00005000  PC = $00006000
D1 = $FFFF0003  A1 = $00005008
D2 = $FFFFE000  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.L  #33,(A1)+`

Source	Destination
#33 **#$00000021**	(A1) **($5008)**

`$005008` **`00 00 00 21`** `D4 36 1F 88` **A1 = $0000500C**

2. `MOVE.B  D2,-1(A1,D0.L)`

Source	Destination
D2.B **#$00**	-1(A1,D0.L) (A1 + D0 - 1) ($5008 - 1 - 1) **($5006)**

`$005000  54 AF 18 B9 E7 21` **`00`** `C0`

3. `MOVE.L  $500A(PC),4(A2)`

Source	Destination
$500A(PC) ($500A) **#$11C8D436**	4(A2) (A2 + 4) ($5010 + 4) **($5014)**

`$005010  13 79 01 80` **`11 C8 D4 36`**

4. `MOVE.W  -4(A2),5(A0,D1.W)`

Source	Destination
-4(A2)	5(A0,D1.W)
(A2 - 4)	(A0 + D1.W$_{(\rightarrow 32)}$ + 5)
($5010 - 4)	($5000 + 3 + 5)
($500C)	**($5008)**
#$D436	

$005008 **D4 36** 11 C8 D4 36 1F 88

Exercice 3 (3 points)

Donnez le résultat des additions hexadécimales suivantes, ainsi que le contenu des bits **N**, **Z**, **V** et **C** du registre d'état.

1. $7A + $86 **opération en .B**
   = $1**00**
   avec **N = 0, Z = 1, V = 0 et C = 1**

2. $FFFF + $FFFF **opération en .W**
   = $1**FFFE**
   avec **N = 1, Z = 0, V = 0 et C = 1**

Exercice 4 (3 points)

Donnez, **en représentation décimale**, les valeurs que prendront les registres **D1** et **D2** en sortie des deux boucles suivantes :

```
                    clr.l   d1
                    move.l  #$8000 0007,d0  ; D0.L = $8000 0007 (D0.W = $0007 = 7).
loop1               addq.l  #1,d1
                    subq.w  #1,d0           ; Attention ! seul D0.W est décrémenté !
                    bne     loop1           ; Saut tant que Z = 0 (D0.W ≠ 0).
                                            ; D1 = 7

                    clr.l   d2
                    moveq.l #125,d0         ; Initialisation de D0 à 125.
loop2               addq.l  #1,d2
                    dbra    d0,loop2        ; DBRA = DBF
                                            ; Décrémente D0.W de 1.
                                            ; Saut tant que D0.W ≠ -1.
                                            ; D2 = 126
```

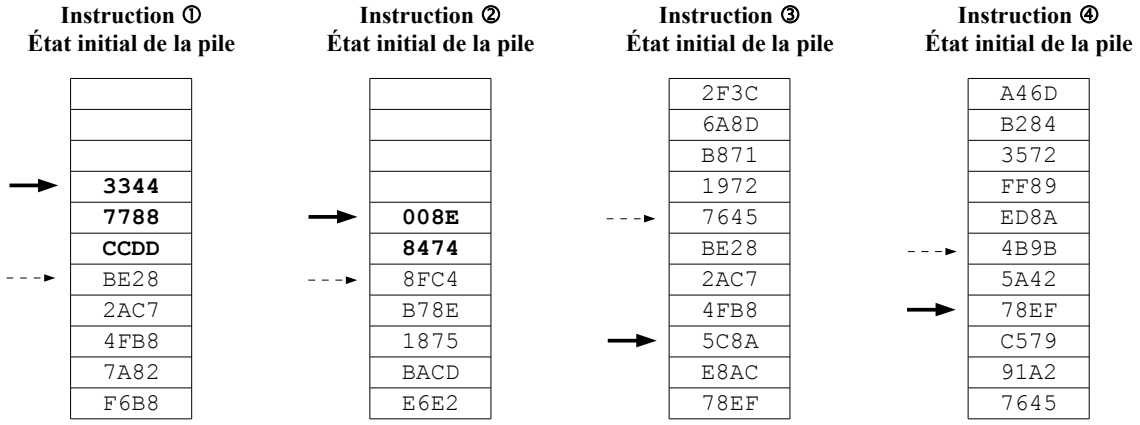
Exercice 5 (5 points)

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. À partir de cet état initial, vous en déterminerez l'état final (juste après l'exécution de l'instruction). Donner l'état final revient à remplir 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**.

   Les registres seront modifiés de la façon suivante :
   - **D0 = $7645BE28**
   - **A6 = $2AC74FB8**

Exercice 6 (1 point)

1. Quelle est la principale différence entre les deux instructions suivantes :
   `MOVE.B -(A0),D0` et `MOVE.B -1(A0),D0`

   - `MOVE.B -(A0),D0`
     Cette instruction modifie le registre d'adresse **A0** (il est décrémenté de 1).

   - `MOVE.B -1(A0),D0`
     Cette instruction ne modifie pas le registre d'adresse **A0**.

2. Quelle opération arithmétique réalise le décalage logique suivant : `LSL.L #5,D0`

   Un décalage logique vers la gauche de **n** bits équivaut à une multiplication par 2^n. **L'instruction `LSL.L #5,D0` multiplie donc le registre D0 par 32** ($2^5 = 32$).

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.

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

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.

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.

MOVEM — Move Multiple Registers (M68000 Family) — MOVEM

Condition Codes:

Not affected.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	SIZE	EFFECTIVE ADDRESS MODE			REGISTER		
REGISTER LIST MASK															

Instruction Fields:

dr field—Specifies the direction of the transfer.
0 — Register to memory.
1 — Memory to register.

Size field—Specifies the size of the registers being transferred.
0 — Word transfer
1 — Long transfer

Effective Address field—Specifies the memory address for the operation. For register-to-memory transfers, only control alterable addressing modes or the predecrement addressing mode can be used as listed in the following tables:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(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.

MOVEM Move Multiple Registers (M68000 Family) MOVEM

For memory-to-register transfers, only control addressing modes or the postincrement addressing mode can be used as listed in the following tables:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
– (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)	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

*Can be used with CPU32.

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

SUBX — Subtract with Extend (M68000 Family) — SUBX

Operation: Destination – Source – X → Destination

Assembler Syntax: SUBX Dx,Dy
SUBX – (Ax), – (Ay)

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand and the extend bit from the destination operand and stores the result in the destination

location. The instruction has two modes:

1. Data register to data register—the data registers specified in the instruction contain the operands.
2. Memory to memory—the address registers specified in the instruction access the operands from memory using the predecrement addressing mode.

The size of the operand is specified as byte, word, or long.

Condition Codes:

X	N	Z	V	C
*	*	*	*	*

X — Set to the value of the carry bit.
N — Set if the result is negative; cleared otherwise.
Z — Cleared if the result is nonzero; unchanged otherwise.
V — Set if an overflow occurs; cleared otherwise.
C — Set if a borrow occurs; cleared otherwise.

NOTE

Normally the Z condition code bit is set via programming before the start of an operation. This allows successful tests for zero results upon completion of multiple-precision operations.

SUBX — Subtract with Extend (M68000 Family) — SUBX

Instruction Format:

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	0	0	1	REGISTER Dy/Ay	1	SIZE	0	0	R/M	REGISTER Dx/Ax

Instruction Fields:

Register Dy/Ay field—Specifies the destination register.
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

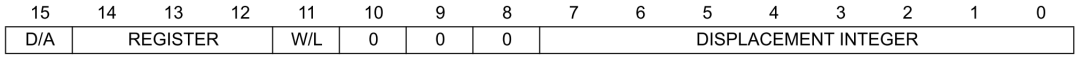
Size field—Specifies the size of the operation.
00 — Byte operation
01 — Word operation
10 — Long operation

R/M field—Specifies the operand addressing mode.
0 — The operation is data register to data register.
1 — The operation is memory to memory.

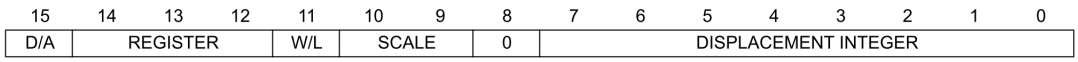
Register Dx/Ax field—Specifies the source register:
If R/M = 0, specifies a data register.
If R/M = 1, specifies an address register for the predecrement addressing mode.

2.4 BRIEF EXTENSION WORD FORMAT COMPATIBILITY

Programs can be easily transported from one member of the M68000 family to another in an upward-compatible fashion. The user object code of each early member of the family, which is upward compatible with newer members, can be executed on the newer microprocessor without change. Brief extension word formats are encoded with information that allows the CPU32, MC68020, MC68030, and MC68040 to distinguish the basic M68000 family architecture's new address extensions. Figure 2-3 illustrates these brief extension word formats. The encoding for SCALE used by the CPU32, MC68020, MC68030, and MC68040 is a compatible extension of the M68000 family architecture. A value of zero for SCALE is the same encoding for both extension words. Software that uses this encoding is compatible with all processors in the M68000 family. Both brief extension word formats do not contain the other values of SCALE. Software can be easily migrated in an upward-compatible direction, with downward support only for nonscaled addressing. If the MC68000 were to execute an instruction that encoded a scaling factor, the scaling factor would be ignored and would not access the desired memory address. The earlier microprocessors do not recognize the brief extension word formats implemented by newer processors. Although they can detect illegal instructions, they do not decode invalid encodings of the brief extension word formats as exceptions.



(a) MC68000, MC68008, and MC68010



(b) CPU32, MC68020, MC68030, and MC68040

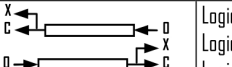
Figure 2-3. M68000 Family Brief Extension Word Formats

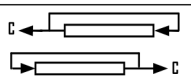
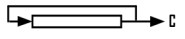
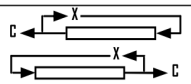
Table 2-1. Instruction Word Format Field Definitions

Field	Definition
Instruction	
Mode	Addressing Mode
Register	General Register Number
Extensions	
D/A	Index Register Type 0 = Dn 1 = An
W/L	Word/Long-Word Index Size 0 = Sign-Extended Word 1 = Long Word
Scale	Scale Factor 00 = 1 01 = 2 10 = 4 11 = 8
BS	Base Register Suppress 0 = Base Register Added 1 = Base Register Suppressed
IS	Index Suppress 0 = Evaluate and Add Index Operand 1 = Suppress Index Operand
BD SIZE	Base Displacement Size 00 = Reserved 01 = Null Displacement 10 = Word Displacement 11 = Long Displacement
I/IS	Index/Indirect Selection Indirect and Indexing Operand Determined in Conjunction with Bit 6, Index Suppress

For effective addresses that use a full extension word format, the index suppress (IS) bit and the index/indirect selection (I/IS) field determine the type of indexing and indirect action. Table 2-2 lists the index and indirect operations corresponding to all combinations of IS and I/IS values.

68000 Quick Reference

Opcode	Size	Operand	CCR	Effective Address												Operation	Description
	BWL	sr,ds	XNZVC	Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Rn)	abs.W	abs.L	(d,PC)	(d,PC,Rn)	#n		
ABCD	B	Dy,Dx -(Ay),-(Ax)	*U*U*	-	-	-	-	-	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$ $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	Add BCD with Extend
ADD	BWL	sr,Dn Dn,ds	*****	sr ds	sr -	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr -	sr -	sr -	sr + Dn → Dn Dn + ds → ds	Add binary (ADDI or ADDQ is used when source is #n)
ADDA [2]	WL	sr,An	-----	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr + An → An	Add address (.W sign-extended to .L)
ADDI [2]	BWL	#n,ds	*****	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	#n + ds → ds	Add immediate
ADDQ [2]	BWL	#n,ds	*****	ds	ds	ds	ds	ds	ds	ds	ds	ds	-	-	-	#n + ds → ds	Add quick immediate (#n range: 1 to 8)
ADDX	BWL	Dy,Dx -(Ay),-(Ax)	*****	-	-	-	-	-	-	-	-	-	-	-	-	Dy + Dx + X → Dx -(Ay) + -(Ax) + X → -(Ax)	Add with Extend
AND	BWL	sr,Dn Dn,ds	-**00	sr ds	- -	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr -	sr -	sr -	sr & Dn → Dn Dn & ds → ds	Logical AND (ANDI is used when source is #n)
ANDI [2]	BWL	#n,ds	-**00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	#n & ds → ds	Logical AND immediate
ANDI [2]	B	#n,CCR	*****	-	-	-	-	-	-	-	-	-	-	-	-	#n & CCR → CCR	Logical AND immediate to CCR
ANDI [2]	W	#n,SR	*****	-	-	-	-	-	-	-	-	-	-	-	-	#n & SR → SR	Logical AND immediate to SR (Privileged)
ASL ASR	BWL W	Dx,Dy #n,Dy ds	*****	- - -	- - -	- - ds	- - ds	- - ds	- - ds	- - ds	- - ds	- - ds	- - -	- - -	- - -		Arithmetic shift Dy Dx bits left/right Arithmetic shift Dy #n bits L/R (#n: 1 to 8) Arithmetic shift ds 1 bit left/right (.W only)
Bcc	BW [3]	label	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc true then PC + d → PC	Branch conditionally (cc: See Table next pg) (d: 8/16-bit signed integer)
BCHG	B L	Dn,ds #n,ds	--*--	ds [1] ds [1]	- -	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	- -	- -	- -	NOT(bit number of ds) → Z NOT(bit n of ds)→ bit n of ds	Set Z with state of specified bit in ds then invert the bit in ds
BCLR	B L	Dn,ds #n,ds	--*--	ds [1] ds [1]	- -	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	- -	- -	- -	NOT(bit number of ds) → Z 0 → bit number of ds	Set Z with state of specified bit in ds then clear the bit in ds
BRA	BW [3]	label	-----	-	-	-	-	-	-	-	-	-	-	-	-	PC + d → PC	Branch always (d: 8/16-bit signed integer)
BSET	B L	Dn,ds #n,ds	--*--	ds [1] ds [1]	- -	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	- -	- -	- -	NOT(bit n of ds) → Z 1 → bit n of ds	Set Z with state of specified bit in ds then set the bit in ds
BSR	BW [3]	label	-----	-	-	-	-	-	-	-	-	-	-	-	-	PC → -(SP); PC + d → PC	Branch to subroutine (d: 8/16-bit sign-int)
BTST	B L	Dn,ds #n,ds	--*--	ds [1] ds [1]	- -	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	- -	NOT(bit Dn of ds) → Z NOT(bit #n of ds) → Z	Set Z with state of specified bit in ds Leave the bit in ds unchanged
CHK	W	sr,Dn	-*UUU	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	if Dn<0 or Dn>sr then TRAP	Compare Dn with 0 and upper bound [sr]
CLR	BWL	ds	-0100	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	0 → ds	Clear destination to zero
CMP	BWL	sr,Dn	-****	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	set CCR with Dn – sr	Compare Dn to source
CMPA [2]	WL	sr,An	-****	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	set CCR with An – sr	Compare An to source
CMPI [2]	BWL	#n,ds	-****	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	set CCR with ds - #n	Compare destination to #n
CMPM [2]	BWL	(Ay)+,(Ax)+	-****	-	-	-	ea	-	-	-	-	-	-	-	-	set CCR with (Ax) - (Ay)	Compare (Ax) to (Ay); Increment Ax & Ay
DBcc	W	Dn,label	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc false then { Dn-1 → Dn if Dn <> -1 then PC+d→PC }	Test condition, decrement & branch (d: 16-bit signed integer)
DIVS	W	sr,Dn	-***0	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	±32bit Dn / ±16bit sr → ±Dn	Dn= [16-bit remainder, 16-bit quotient]
DIVU	W	sr,Dn	-***0	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	32bit Dn / 16bit sr → Dn	Dn= [16-bit remainder, 16-bit quotient]
EOR	BWL	Dn,ds	-**00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	Dn XOR ds → ds	Logical exclusive OR Dn to ds
EORI [2]	BWL	#n,ds	-**00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	#n XOR ds → ds	Logical exclusive OR #n to ds
EORI [2]	B	#n,CCR	*****	-	-	-	-	-	-	-	-	-	-	-	-	#n XOR CCR → CCR	Logical exclusive OR #n to CCR
EORI [2]	W	#n,SR	*****	-	-	-	-	-	-	-	-	-	-	-	-	#n XOR SR → SR	Logical exclusive OR #n to SR (Privileged)
EXG	L	Rx,Ry	-----	ea	ea	-	-	-	-	-	-	-	-	-	-	register ←→ register	Exchange registers (32-bit only)
EXT	WL	Dn	-**00	-	-	-	-	-	-	-	-	-	-	-	-	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		ds	-----	-	-	ds	-	-	ds	ds	ds	ds	ds	ds	-	ds → PC	Jump to address specified by ds
JSR		ds	-----	-	-	ds	-	-	ds	ds	ds	ds	ds	ds	-	PC → -(SP); ds → PC	push PC, jump to subroutine at address ds
LEA	L	sr,An	-----	-	-	sr	-	-	sr	sr	sr	sr	sr	sr	-	sr → An	Load effective address of sr to An
LINK		An,#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	An → -(SP); SP → An; SP + #n → SP	Create local workspace on stack (n must be negative to allocate!)
LSL LSR	BWL W	Dx,Dy #n,Dy ds	***0*	- - -	- - -	- - ds	- - ds	- - ds	- - ds	- - ds	- - ds	- - ds	- - -	- - -	- - -		Logical shift Dy, Dx bits left/right Logical shift Dy, #n bits L/R (#n: 1 to 8) Logical shift ds 1 bit left/right (.W only)
MOVE	BWL	ea,ea	-**00	ea	sr	ea	ea	ea	ea	ea	ea	ea	sr	sr	sr	sr → ds	Move data from source to destination
MOVE	W	sr,CCR	*****	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr → CCR	Move source to Condition Code Register
MOVE	W	sr,SR	*****	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr → SR	Move source to Status Register (Privileged)
MOVE	W	SR,ds	-----	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	SR → ds	Move Status Register to destination
MOVE	L	USP,An An,USP	-----	-	-	-	-	-	-	-	-	-	-	-	-	USP → An An → USP	Move User Stack Pointer to An (Privileged) Move An to User Stack Pointer (Privileged)

Opcode	Size	Operand	CCR	Effective Address												Operation	Description
	BWL	sr,ds	XNZVC	Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Rn)	abs.W	abs.L	(d,PC)	(d,PC,Rn)	#n		
MOVEA[2]	WL	sr,An	-----	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr → An	Move source to An (MOVE sr,An use MOVEA)
MOVEM[2]	WL	Rn-Rn,ds sr,Rn-Rn	-----	- -	- -	ds sr	- sr	ds -	ds sr	ds sr	ds sr	ds sr	- sr	- sr	- -	Registers → ds sr → Registers	Move specified registers to/from memory (.W source is sign-extended to .L for Rn)
MOVEP	WL	Dn,d(An) d(An),Dn	-----	-	-	-	-	-	-	-	-	-	-	-	-	Dn → d(An)...d+2(An)...d+4(A d(An) → Dn; d+2(An)...d+4(A	Move Dn to/from alternate memory bytes (Access only even or odd addresses)
MOVEQ[2]	L	#n,Dn	-**00	-	-	-	-	-	-	-	-	-	-	-	-	#n → Dn	Move sign extended 8-bit #n to Dn
MULS	W	sr,Dn	-**00	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	±16bit sr * ±16bit Dn → ±Dn	Multiply signed 16-bit; result: signed 32-bit
MULU	W	sr,Dn	-**00	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	16bit sr * 16bit Dn → Dn	Multiply unsig'd 16-bit; result: unsig'd 32-bit
NBCD	B	ds	*U*U*	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	0 - ds_{10} - X → ds	Negate BCD with Extend
NEG	BWL	ds	*****	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	0 - ds → ds	Negate ds
NEGX	BWL	ds	*****	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	0 - ds - X → ds	Negate ds with Extend
NOP			-----	-	-	-	-	-	-	-	-	-	-	-	-	None	No operation occurs
NOT	BWL	ds	-**00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	NOT(ds) → ds	Logical NOT (ones complement of ds)
OR	BWL	sr,Dn Dn,ds	-**00	sr -	- -	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr -	sr -	sr -	sr OR Dn → Dn Dn OR ds → ds	Logical OR (ORI is used when source is #n)
ORI[2]	BWL	#n,ds	-**00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	#n OR ds → ds	Logical OR #n to ds
ORI[2]	B	#n,CCR	*****	-	-	-	-	-	-	-	-	-	-	-	-	#n OR CCR → CCR	Logical OR #n to CCR
ORI[2]	W	#n,SR	*****	-	-	-	-	-	-	-	-	-	-	-	-	#n OR SR → SR	Logical OR #n to SR (Privileged)
PEA	L	ds	-----	-	-	ds	-	-	ds	ds	ds	ds	ds	ds	-	ds → -(SP)	Push effective address of ds onto stack
RESET			-----	-	-	-	-	-	-	-	-	-	-	-	-	Assert RESET Line	Issue a hardware RESET (Privileged)
ROL ROR ROR	BWL W	Dx,Dy #n,Dy ds	-**0*	- - -	- - -	- - ds	- - ds	- - ds	- - ds	- - ds	- - ds	- - ds	- - -	- - -	- - -	 	Rotate Dy, Dx bits left/right (without X) Rotate Dy, #n bits left/right (#n: 1 to 8) Rotate ds 1-bit left/right (.W only)
ROXL ROXR ROXR	BWL W	Dx,Dy #n,Dy ds	***0*	- - -	- - -	- - ds	- - ds	- - ds	- - ds	- - ds	- - ds	- - ds	- - -	- - -	- - -		Rotate Dy, Dx bits L/R (X used then updated) Rotate Dy, #n bits left/right (#n: 1 to 8) Rotate ds 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 -(Ay),-(Ax)	*U*U*	-	-	-	-	-	-	-	-	-	-	-	-	Dx_{10} - Dy_{10} - X → Dx_{10} -$(Ax)_{10}$ - -$(Ay)_{10}$ - X → -$(Ax)_{10}$	Subtract BCD with Extend
Scc	B	ds	-----	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	If cc is true then 1's → ds else 0's → ds	If cc true then ds.B = 11111111 else ds.B = 00000000
STOP		#n	*****	-	-	-	-	-	-	-	-	-	-	-	-	#n → SR; STOP	Move #n to SR, stop processor (Privileged)
SUB	BWL	sr,Dn Dn,ds	*****	sr -	sr -	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr -	sr -	sr -	Dn - sr → Dn ds - Dn → ds	Subtract binary (SUBI or SUBQ is used when source is #n)
SUBA[2]	WL	sr,An	-----	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	An - sr → An	Subtract address (.W sign-extended to .L)
SUBI[2]	BWL	#n,ds	*****	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	ds - #n → ds	Subtract immediate
SUBQ[2]	BWL	#n,ds	*****	ds	ds	ds	ds	ds	ds	ds	ds	ds	-	-	-	ds - #n → ds	Subtract quick immediate (#n range: 1 to 8)
SUBX	BWL	Dy,Dx -(Ay),-(Ax)	*****	-	-	-	-	-	-	-	-	-	-	-	-	Dx - Dy - X → Dx -(Ax) - -(Ay) - X → -(Ax)	Subtract with Extend
SWAP	W	Dn	-**00	-	-	-	-	-	-	-	-	-	-	-	-	bits[31:16]←→bits[15:0]	Exchange the 16-bit halves of Dn
TAS	B	ds	-**00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	test ds→CCR; 1 →bit7 of ds	N and Z set to reflect ds, bit7 of ds set to 1
TRAP		#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	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	ds	-**00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	test ds → CCR	N and Z set to reflect ds
UNLK		An	-----	-	-	-	-	-	-	-	-	-	-	-	-	An → SP; (SP)+ → An	Remove local workspace from stack

Condition Tests (**&** logical AND, **+** logical OR, **!** logical NOT, **u** Unsigned)					
cc	**Condition**	**Test**	**cc**	**Condition**	**Test**
T	true	1	VC	overflow clear	**!**V
F	false	0	VS	overflow set	V
HI[u]	high	**!**C & **!**Z	PL	plus	**!**N
LS[u]	low or same	C + Z	MI	minus	N
CC, HS[u]	carry clear	**!**C	GE	greater or equal	N & V + **!**N & **!**V
CS, LO[u]	carry set	C	LT	less than	N & **!**V + **!**N & V
NE	not equal	**!**Z	GT	greater than	N&V& **!**Z + **!**N& **!**V& **!**Z
EQ	equal	Z	LE	less or equal	Z + N & **!**V + **!**N & V

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
PC Program Counter (24-bit)
sr Source **ds** Destination
#n Immediate data **d** Displacement
ea Effective Address (source or destination)
BCD Binary Coded Decimal

SSP Supervisor Stack Pointer (32-bit)
USP User Stack Pointer (32-bit)
SP Active Stack Pointer (same as A7)
label Destination of Branch (Assembler calculates displacement value)
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 according to result of operation
- not affected, **0** cleared, **1** set, **U** undefined

1 Long only; all others are byte only
2 Assembler selects appropriate opcode
3 Branch sizes: **.B** or **.S** -128 to +127 bytes, **.W** or **.L** -32768 to +32767 bytes