

Partiel S3

Architecture des ordinateurs

Durée : 1 h 30

Exercice 1 (9 points)

Dans cet exercice, vous devrez réaliser trois sous-programmes qui copient des octets situés à un emplacement mémoire vers un autre emplacement mémoire. Aucun registre ne devra être modifié en sortie de vos sous-programmes. Chacun de ces trois sous-programmes possède les entrées suivantes :

Entrées : **A1.L** pointe sur l'emplacement source des octets à copier.

A2.L pointe sur l'emplacement destination.

D0.L contient le nombre d'octets à copier (entier non signée).

La conception de chaque sous-programme est indépendante.

1. Réalisez le sous-programme **CopyInc** qui copie les données en commençant par le premier octet et qui incrémente les adresses (*cf. exemple ci-dessous*). On suppose que lors d'un appel à **CopyInc** :
 - le registre **D0** n'est jamais nul ;
 - les registres **A1** et **A2** ne sont jamais égaux.
2. Réalisez le sous-programme **CopyDec** qui copie les données en commençant par le dernier octet et qui décrémente les adresses (*cf. exemple ci-dessous*). On suppose que lors d'un appel à **CopyDec** :
 - le registre **D0** n'est jamais nul ;
 - les registres **A1** et **A2** ne sont jamais égaux.
3. Réalisez le sous-programme **Copy** qui appelle **CopyInc** si l'adresse de l'emplacement destination est inférieure stricte à l'adresse de l'emplacement source, ou qui appelle **CopyDec** si l'adresse de l'emplacement destination est supérieure stricte à l'adresse de l'emplacement source. On suppose que lors d'un appel à **Copy** :
 - le registre **D0** peut être nul : dans ce cas, aucun octet ne doit être copié ;
 - les registres **A1** et **A2** peuvent être égaux : dans ce cas, aucun octet ne doit être copié.

Exemple pour A1 = \$1000, A2 = \$2000 et D0 = 3.	
CopyInc : (\$1000) → (\$2000)	CopyDec : (\$1002) → (\$2002)
(\$1001) → (\$2001)	(\$1001) → (\$2001)
(\$1002) → (\$2002)	(\$1000) → (\$2000)

Exercice 2 (4 points)

Remplir le tableau présent sur le document réponse. Donnez le nouveau contenu des registres (sauf le PC) et/ou de la mémoire modifiés par les instructions. **Vous utiliserez la représentation hexadécimale. La mémoire et les registres sont réinitialisés à chaque nouvelle instruction.**

Valeurs initiales : D0 = \$0004FFFF A0 = \$00005000 PC = \$00006000
 D1 = \$0001000A A1 = \$00005008
 D2 = \$FFFFFFFD A2 = \$00005010

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

Exercice 3 (3 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. Remplir le tableau présent sur le document réponse.

1. Addition sur 8 bits : \$7F + \$? avec N = 1, Z = 0, V = 1, C = 0
2. Addition sur 16 bits : \$98BD + \$? avec N = 0, Z = 1, V = 0, C = 1
3. Addition sur 32 bits : \$98BD + \$? avec N = 1, Z = 0, V = 0, C = 0

Exercice 4 (4 points)

Soit les quatre programmes ci-dessous :

```
Prog1      tst.b   d5
           beq     quit1
           moveq.l #2,d1
quit1
```

```
Prog2      tst.w   d5
           bpl     quit2
           moveq.l #2,d2
quit2
```

```
Prog3      move.w  #100,d7
loop3      addq.l  #1,d3
           dbra    d7,loop3      ; DBRA = DBF (DBcc avec cc = F)
```

```
Prog4      move.l  #1000,d0
loop4      addq.l  #1,d4
           addi.l  #10,d0
           cmpi.l  #2000,d0
           bne     loop4
```

- 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** = \$0067A200


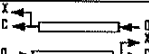

Répondre sur le document réponse.

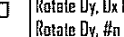
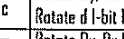
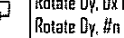
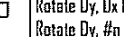
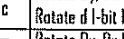
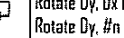
1. Quelle sera la valeur du registre **D1** après l'exécution du programme **Prog1** ?
2. Quelle sera la valeur du registre **D2** après l'exécution du programme **Prog2** ?
3. Quelle sera la valeur du registre **D3** après l'exécution du programme **Prog3** ?
4. Quelle sera la valeur du registre **D4** après l'exécution du programme **Prog4** ?

EASy68K Quick Reference v1.8

<http://www.wowgweb.com/EASy68K.htm>

Copyright © 2004-2007 By: Chuck Kelly

Opcode	Size	Operand	CCR	Effective Address s=source, d=destination, e=either, i=displacement												Operation	Description
		s,d	XNZVC	Un	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	-	-	-	-	-	-	-	-	-	-	-	$Dy + Dx + X \rightarrow D_{x0}$ $-(Ay)_0 + -(Ax)_0 + X \rightarrow -(Ax)_0$	Add BCD source and eXtend bit to destination, BCD result
ADD	BWL	s,Dn Dn,d	*****	e	s	s	s	s	s	s	s	s	s	s	s	$s + Dn \rightarrow Dn$ $Dn + d \rightarrow d$	Add binary (ADD) or ADDQ is used when source is #n. Prevent ADDQ with #n.L
ADDA	WL	s,An	-----	s	e	s	s	s	s	s	s	s	s	s	s	$s + An \rightarrow An$	Add address (W sign-extended to .L)
ADDI	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	$\#n + d \rightarrow d$	Add immediate to destination
ADDQ	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	s	$\#n + d \rightarrow d$	Add quick immediate (#n range: 1 to 8)
ADDX	BWL	Dy,Dx -(Ay),-(Ax)	*****	e	-	-	-	-	-	-	-	-	-	-	-	$Dy + Dx + X \rightarrow Dx$ $-(Ay) + -(Ax) + X \rightarrow -(Ax)$	Add source and eXtend bit to destination
AND	BWL	s,Dn Dn,d	-**00	e	-	s	s	s	s	s	s	s	s	s	s	$s \text{ AND } Dn \rightarrow Dn$ $Dn \text{ AND } d \rightarrow d$	Logical AND source to destination (ANDI is used when source is #n)
ANDI	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	s	$\#n \text{ AND } d \rightarrow d$	Logical AND immediate to destination
ANDI	B	#n,CCR	000 000 000 000	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ AND CCR} \rightarrow \text{CCR}$	Logical AND immediate to CCR
ANDI	W	#n,SR	000 000 000 000	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ AND SR} \rightarrow \text{SR}$	Logical AND immediate to SR (Privileged)
ASL	BWL	Dx,Dy	*****	e	-	-	-	-	-	-	-	-	-	-	-		Arithmetic shift Dy by Dx bits left/right
ASR	W	#n,Dy		d	-	-	-	-	-	-	-	-	-	-	s		Arithmetic shift Dy #n bits L/R (#n: 1 to 8)
Bcc	BW	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc true then address \rightarrow PC	Branch conditionally (cc table on back) (8 or 16-bit \pm offset to address)
BCHG	B L	Dn,d #n,d	-**---	e ¹ d ¹	-	d	d	d	d	d	d	d	-	-	-	NOT(bit number of d) \rightarrow Z NOT(bit n of d) \rightarrow 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 ¹ d ¹	-	d	d	d	d	d	d	d	-	-	-	NOT(bit number of d) \rightarrow Z 0 \rightarrow bit number of d	Set Z with state of specified bit in d then clear the bit in d
BRA	BW	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	address \rightarrow PC	Branch always (8 or 16-bit \pm offset to addr)
BSET	B L	Dn,d #n,d	-**---	e ¹ d ¹	-	d	d	d	d	d	d	d	-	-	-	NOT(bit n of d) \rightarrow Z 1 \rightarrow bit n of d	Set Z with state of specified bit in d then set the bit in d
BSR	BW	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	PC \rightarrow (SP); address \rightarrow PC	Branch to subroutine (8 or 16-bit \pm offset)
BTST	B L	Dn,d #n,d	-**---	e ¹ d ¹	-	d	d	d	d	d	d	d	-	-	-	NOT(bit 0n of d) \rightarrow Z NOT(bit #n of d) \rightarrow Z	Set Z with state of specified bit in d Leave the bit in d unchanged
CHK	W	s,Dn	-**UUU	e	-	s	s	s	s	s	s	s	s	s	s	if $Dn < 0$ or $Dn > s$ then TRAP	Compare Dn with D and upper bound (s)
CLR	BWL	d	-0100	d	-	d	d	d	d	d	d	d	-	-	-	0 \rightarrow d	Clear destination to zero
CMP	BWL	s,Dn	-****	e	s	s	s	s	s	s	s	s	s	s	s	set CCR with $Dn - s$	Compare Dn to source
CMPA	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	BWL	#n,d	-****	d	-	d	d	d	d	d	d	d	-	-	s	set CCR with $d - \#n$	Compare destination to #n
CMPI	BWL	(Ay),(Ax)	-****	-	-	e	-	-	-	-	-	-	-	-	-	set CCR with $(Ax) - (Ay)$	Compare (Ax) to (Ay); Increment Ax and Ay
DBcc	W	Dn,address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc false then { $Dn - 1 \rightarrow Dn$ if $Dn < -1$ then addr \rightarrow PC }	Test condition, decrement and branch (16-bit \pm offset to address)
DIVS	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	$\pm 32\text{bit } Dn / \pm 16\text{bit } s \rightarrow \pm Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
DIVU	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	$32\text{bit } Dn / 16\text{bit } s \rightarrow Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
EOR	BWL	Dn,d	-**00	e	-	d	d	d	d	d	d	d	-	-	s	$Dn \text{ XOR } d \rightarrow d$	Logical exclusive OR Dn to destination
EORI	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	s	$\#n \text{ XOR } d \rightarrow d$	Logical exclusive OR #n to destination
EORI	B	#n,CCR	000 000 000 000	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR CCR} \rightarrow \text{CCR}$	Logical exclusive OR #n to CCR
EORI	W	#n,SR	000 000 000 000	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR SR} \rightarrow \text{SR}$	Logical exclusive OR #n to SR (Privileged)
EXG	L	Rx,Ry	-----	e	e	-	-	-	-	-	-	-	-	-	-	register \leftrightarrow register	Exchange registers (32-bit only)
EXT	WL	Dn	-**00	d	-	-	-	-	-	-	-	-	-	-	-	$Dn.B \rightarrow Dn.W \mid Dn.W \rightarrow Dn.L$	Sign extend (change .B to .W or .W to .L)
ILLEGAL				-	-	-	-	-	-	-	-	-	-	-	-	PC \rightarrow (SSP); SR \rightarrow (SSP)	Generate illegal instruction exception
JMP		d	-----	-	-	d	-	-	d	d	d	d	d	d	-	$\uparrow d \rightarrow \text{PC}$	Jump to effective address of destination
JSR		d	-----	-	-	d	-	-	d	d	d	d	d	d	-	PC \rightarrow (SP); $\uparrow d \rightarrow \text{PC}$	push PC, jump to subroutine at address d
LEA	L	s,An	-----	-	e	s	-	-	s	s	s	s	s	s	-	$\uparrow s \rightarrow An$	Load effective address of s to An
LINK		An,#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	$An \rightarrow$ (SP); $SP \rightarrow An$; $SP + \#n \rightarrow SP$	Create local workspace on stack (negative n to allocate space)
LSL	BWL	Dx,Dy #n,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, Dx bits left/right
LSR	W	d		d	-	-	-	-	-	-	-	-	-	-	s		Logical shift Dy, #n bits L/R (#n: 1 to 8)
MOVE	BWL	s,d	-**00	e	s	e	e	e	e	e	e	e	s	s	s	$s \rightarrow d$	Move data from source to destination
MOVE	W	s,CCR	000 000 000 000	s	-	s	s	s	s	s	s	s	s	s	s	$s \rightarrow \text{CCR}$	Move source to Condition Code Register
MOVE	W	s,SR	000 000 000 000	s	-	s	s	s	s	s	s	s	s	s	s	$s \rightarrow \text{SR}$	Move source to Status Register (Privileged)
MOVE	W	SR,d	-----	d	-	d	d	d	d	d	d	d	-	-	-	$\text{SR} \rightarrow d$	Move Status Register to destination
MOVE	L	USP,An An,USP	-----	-	d	-	-	-	-	-	-	-	-	-	-	$\text{USP} \rightarrow An$ $An \rightarrow \text{USP}$	Move User Stack Pointer to An (Privileged) Move An to User Stack Pointer (Privileged)
	BWL	s,d	XNZVC	Un	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	On	An	(An)	(An)+	-(An)	(iAn)	(iAn.Rn)	abs.W	abs.L	(i.PC)	(i.PC.Rn)	#n				
MOVEA*	WL	s,An	-----	s	e	s	s	s	s	s	s	s	s	s	s	s	s → An	Move source to An (MOVE s,An use MOVEA)	
MOVM*	WL	Rn-Rn,d	-----	-	-	d	-	-	d	d	d	d	-	-	-	-	Registers → d	Move specified registers to/from memory (W source is sign-extended to .L for Rn)	
		s,Rn-Rn		-	-	s	s	-	-	s	s	s	s	s	-	-	s → Registers		
MOVEP	WL	On,(iAn)	-----	s	d	-	-	-	d	-	-	-	-	-	-	-	On → (iAn)...(i+2,An)...(i+4,An)	Move On to/from alternate memory bytes	
		(iAn),On		d	-	-	-	-	s	-	-	-	-	-	-	-	(i,An) → On...(i+2,An)...(i+4,An)	(Access only even or odd addresses)	
MOVED*	L	#n,On	---*00	d	-	-	-	-	-	-	-	-	-	-	-	-	#n → On	Move sign extended 8-bit #n to On	
MULS	W	s,On	---*00	e	-	s	s	s	s	s	s	s	s	s	s	s	±16bit s * ±16bit On → ±On	Multiply signed 16-bit; result: signed 32-bit	
MULU	W	s,On	---*00	e	-	s	s	s	s	s	s	s	s	s	s	s	16bit s * 16bit On → On	Multiply unsig'd 16-bit; result: unsig'd 32-bit	
NBCD	B	d	*U*U*	d	-	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	d	-	-	-	0 - d → d	Negate destination (2's complement)	
NEGX	BWL	d	*****	d	-	d	d	d	d	d	d	d	d	-	-	-	0 - d - X → d	Negate destination with eXtend	
NDP			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	None	No operation occurs	
NOT	BWL	d	---*00	d	-	d	d	d	d	d	d	d	d	-	-	-	NOT(d) → d	Logical NOT destination (1's complement)	
OR*	BWL	s,On	---*00	e	-	s	s	s	s	s	s	s	s	s	s	s	s OR On → On	Logical OR	
		On,d		e	-	d	d	d	d	d	d	d	d	-	-	-	On OR d → d	(ORI is used when source is #n)	
ORI*	BWL	#n,d	---*00	d	-	d	d	d	d	d	d	d	d	-	-	-	#n OR d → d	Logical OR #n to destination	
ORI*	B	#n,CCR	00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	-	-	#n OR CCR → CCR	Logical OR #n to CCR	
ORI*	W	#n,SR	00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	-	-	#n OR SR → SR	Logical OR #n to SR (Privileged)	
PEA	L	s	-----	-	-	s	-	-	-	s	s	s	s	s	s	s	↑s → -(SP)	Push effective address of s onto stack	
RESET			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Assert RESET Line	
ROL	BWL	0x,0y	---*0*	e	-	-	-	-	-	-	-	-	-	-	-	-			Rotate 0y, 0x bits left/right (without X)
ROR	W	#n,0y		d	-	-	d	d	d	-	d	d	d	-	-	-			Rotate 0y, #n bits left/right (#n: 1 to 8)
		d		-	-	d	d	d	d	d	d	d	d	-	-	-			Rotate d 1-bit left/right (W only)
RDXL	BWL	0x,0y	---*0*	e	-	-	-	-	-	-	-	-	-	-	-	-			Rotate 0y, 0x bits L/R, X used then updated
RDXR	W	#n,0y		d	-	-	d	d	d	d	d	d	d	-	-	-			Rotate 0y, #n bits left/right (#n: 1 to 8)
		d		-	-	d	d	d	d	d	d	d	d	-	-	-			Rotate destination 1-bit left/right (W only)
RTE			00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → SR; (SP)+ → PC	Return from exception (Privileged)	
RTR			00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → CCR; (SP)+ → PC	Return from subroutine and restore CCR	
RTS			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → PC	Return from subroutine	
SBCD	B	0y,0x -(Ay),-(Ax)	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	-	0x ₀ - 0y ₀ - X → 0x ₀	Subtract BCD source and eXtend bit from destination, BCD result	
				-	-	-	-	-	e	-	-	-	-	-	-	-	-(Ax) ₀ - -(Ay) ₀ - X → -(Ax) ₀		
Sec	B	d	-----	d	-	d	d	d	d	d	d	d	d	-	-	-	If cc true then d.B = 11111111 else d.B = 00000000	If cc true then d.B = 11111111 else d.B = 00000000	
STOP		#n	00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	-	-	#n → SR; STOP	Move #n to SR, stop processor (Privileged)	
SUB*	BWL	s,On	*****	e	s	s	s	s	s	s	s	s	s	s	s	s	On - s → On	Subtract binary (SUB) or SUBQ used when source is #n. Prevent SUBQ with #n.L	
		On,d		e	d ¹	d	d	d	d	d	d	d	d	-	-	-	d - On → d		
SUBA*	WL	s,An	-----	s	e	s	s	s	s	s	s	s	s	s	s	s	An - s → An	Subtract address (W sign-extended to .L)	
SUBI*	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	d	-	-	-	d - #n → d	Subtract immediate from destination	
SUBQ*	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	d	-	-	-	d - #n → d	Subtract quick immediate (#n range: 1 to 8)	
SUBX	BWL	0y,0x -(Ay),-(Ax)	*****	e	-	-	-	-	-	-	-	-	-	-	-	-	0x - 0y - X → 0x	Subtract source and eXtend bit from destination	
				-	-	-	-	e	-	-	-	-	-	-	-	-	-(Ax) - -(Ay) - X → -(Ax)		
SWAP	W	On	---*00	d	-	-	-	-	-	-	-	-	-	-	-	-	bits(31:16) ↔ bits(15:0)	Exchange the 16-bit halves of On	
TAS	B	d	---*00	d	-	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	-----	-	-	-	-	-	-	-	-	-	-	-	-	-	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	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	On	An	(An)	(An)+	-(An)	(iAn)	(iAn.Rn)	abs.W	abs.L	(i.PC)	(i.PC.Rn)	#n				

Condition Tests (* OR, ! NOT, * Unsigned, * Alternate cc)					
cc	Condition	Test	cc	Condition	Test
T	true	I	VC	overflow clear	IV
F	false	O	VS	overflow set	V
H*	higher than	I(C + Z)	PL	plus	IN
LS*	lower or same	C + Z	MI	minus	N
HS*, CC*	higher or same	IC	GE	greater or equal	I(N ⊕ V)
LO*, CS*	lower than	C	LT	less than	(N ⊕ V)
NE	not equal	IZ	GT	greater than	I((N ⊕ V) + Z)
EQ	equal	Z	LE	less or equal	(N ⊕ V) + Z

Revised by Peter Csaszar, Lawrence Tech University – 2004-2006

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

s Source, d Destination

e Either source or destination

#n Immediate data, i Displacement

BCD Binary Coded Decimal

↑ Effective address

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

SSP Supervisor Stack Pointer (32-bit)

USP User Stack Pointer (32-bit)

SP Active Stack Pointer (same as A7)

PC Program Counter (24-bit)

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 operation's result, = set directly

- not affected, 0 cleared, 1 set, U undefined

Distributed under the GNU general public use license.

Nom : Prénom : Classe :

DOCUMENT RÉPONSE À RENDRE AVEC LA COPIE

Exercice 2

Instruction	Mémoire	Registre
Exemple	\$005000 54 AF 00 40 E7 21 48 C0	A0 = \$00005004 A1 = \$0000500C
Exemple	\$005008 C9 10 11 C8 D4 36 FF 88	Aucun changement
MOVE.B -(A2), -(A1)		
MOVE.W \$5010, -5(A2,D2.L)		
MOVE.L #\$500E, -10(A0,D1.W)		
MOVE.B \$5007(PC), 8(A1)		

Exercice 3

Opération	Taille (bits)	Nombre manquant (hexadécimal)	N	Z	V	C
\$7F + \$?	8		1	0	1	0
\$98BD + \$?	16		0	1	0	1
\$98BD + \$?	32		1	0	0	0

Exercice 4

Utilisez la représentation hexadécimale sur 32 bits.

D1 = \$

D2 = \$

D3 = \$

D4 = \$