Contrôle S3 – Corrigé
Architecture des ordinateurs

Durée : 1 h 30

Exercice 1 (5 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 = $FFFF0005  A1 = $00005008
                      D2 = $FFFFFFFE  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 2 (4 points)

Remplissez le tableau présent sur le document réponse. Donnez le résultat des additions ainsi que le contenu des bits **N**, **Z**, **V** et **C** du registre d'état.

Exercice 3   (3 points)

Donnez quelques instructions qui modifient la valeur de **D1** afin de lui donner les valeurs présentent sur le document réponse. Pour chaque cas, la valeur initiale de **D1** est $76543210. **Utilisez uniquement les instructions ROR, ROL ou SWAP**. Répondez sur le document réponse.

Exercice 4   (2 points)

Répondez aux questions sur le document réponse.

Exercice 5   (6 points)

Soit le programme ci-dessous :

```
Main        move.l  #$23456789,d7 ; $23456789 -> D7.L

next1       moveq.l #1,d1          ; $00000001 -> D1.L
            tst.b   d7            ; Mise à jour de N et de Z en fonction de D7.B.
            bmi     next2         ; Saut si N = 1 (D7.B < 0).
            moveq.l #2,d1         ; Sinon, $00000002 -> D1.L

next2       moveq.l #1,d2          ; $00000001 -> D2.L
            tst.w   d7            ; Mise à jour de N et de Z en fonction de D7.W.
            bpl     next3         ; Saut si N = 0 (D7.W ≥ 0).
            moveq.l #2,d2         ; Sinon, $00000002 -> D2.L

next3       clr.l   d3            ; $00000000 -> D3.L
            move.w  #$4321,d0     ; $4321 -> D0.W (D0.B = $21)
loop3       addq.l  #1,d3         ; D3.L + 1 -> D3.L
            subq.b  #1,d0         ; D0.B – 1 -> D0.B ; Seul D0.B est décrémenté.
            bne     loop3         ; Saut si Z = 0 (D0.B ≠ 0)

next4       clr.l   d4            ; $00000000 -> D4.L
            move.w  #$44,d0       ; $0044 -> D0.W
loop4       addq.l  #1,d4         ; D4.L + 1 -> D4.L
            dbra    d0,loop4      ; DBRA = DBF ; D0.W – 1 -> D0.W
                                  ; Saut si D0.W ≠ -1 (D0.W ≠ $FFFF)

next5       clr.l   d5            ; $00000000 -> D5.L
            moveq.l #10,d0        ; $0000000A -> D0.L
loop5       addq.l  #1,d5         ; D5.L + 1 -> D5.L
            addq.l  #1,d0         ; D0.L + 1 -> D0.L
            cmpi.l  #30,d0        ; Compare D0.L à la valeur 30.
            bne     loop5         ; Saut si Z = 0 (D0.L ≠ 30)

next6       moveq.l #1,d6          ; $00000001 -> D6.L
            cmp.b   #$70,d7       ; Compare D7.B à la valeur $70.
            blt     quit          ; Saut si D7.B < $70 (comparaison signée).
            moveq.l #2,d6         ; Sinon, $00000002 -> D6.L

quit        illegal
```

Complétez le tableau présent sur le document réponse.

EASy68K Quick Reference v1.8   http://www.wowgwep.com/EASy68K.htm   Copyright © 2004-2007 By: Chuck Kelly

Opcode	Size	Operand	CCR	\<Dn\>	\<An\>	\<(An)\>	\<(An)+\>	\<-(An)\>	\<(i,An)\>	\<(i,An,Rn)\>	\<abs.W\>	\<abs.L\>	\<(i,PC)\>	\<(i,PC,Rn)\>	\<#n\>	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	Effective Address s=source, d=destination, e=either, i=displacement	
ABCD	B	Dy,Dx	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$	Add BCD source and eXtend bit to
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	$-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	destination, BCD result
ADD [4]	BWL	s,Dn	*****	e	s	s	s	s	s	s	s	s	s	s	s[4]	$s + Dn \rightarrow Dn$	Add binary (ADDI or ADDQ is used when
		Dn,d		e	d[4]	d	d	d	d	d	d	d	-	-	-	$Dn + d \rightarrow d$	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	$s + An \rightarrow An$	Add address (.W sign-extended to .L)
ADDI [4]	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	$\#n + d \rightarrow d$	Add immediate to destination
ADDQ [4]	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	*****	e	-	-	-	-	-	-	-	-	-	-	-	$Dy + Dx + X \rightarrow Dx$	Add source and eXtend bit to destination
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	$-(Ay) + -(Ax) + X \rightarrow -(Ax)$	
AND [4]	BWL	s,Dn	-**00	e	-	s	s	s	s	s	s	s	s	s	s[4]	$s \text{ AND } Dn \rightarrow Dn$	Logical AND source to destination
		Dn,d		e	-	d	d	d	d	d	d	d	-	-	-	$Dn \text{ AND } d \rightarrow d$	(ANDI is used when source is #n)
ANDI [4]	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 [4]	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ AND } CCR \rightarrow CCR$	Logical AND immediate to CCR
ANDI [4]	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ AND } SR \rightarrow SR$	Logical AND immediate to SR (Privileged)
ASL ASR	BWL	Dx,Dy	*****	e	-	-	-	-	-	-	-	-	-	-	-	(shift diagram)	Arithmetic shift Dy by Dx bits left/right
		#n,Dy		d	-	-	-	-	-	-	-	-	-	-	s		Arithmetic shift Dy #n bits L/R (#n: 1 to 8)
	W	d		-	-	d	d	d	d	d	d	d	-	-	-	(shift diagram)	Arithmetic shift ds 1 bit left/right (.W only)
Bcc	BW[3]	address[2]	------	-	-	-	-	-	-	-	-	-	-	-	-	if cc true then address $\rightarrow$ 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	-	-	-	$\text{NOT(bit number of d)} \rightarrow Z$	Set Z with state of specified bit in d then
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	$\text{NOT(bit n of d)} \rightarrow \text{bit n of d}$	invert the bit in d
BCLR	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	-	-	-	$\text{NOT(bit number of d)} \rightarrow Z$	Set Z with state of specified bit in d then
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	$0 \rightarrow \text{bit number of d}$	clear the bit in d
BRA	BW[3]	address[2]	------	-	-	-	-	-	-	-	-	-	-	-	-	address $\rightarrow$ PC	Branch always (8 or 16-bit ± offset to addr)
BSET	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	-	-	-	$\text{NOT(bit n of d)} \rightarrow Z$	Set Z with state of specified bit in d then
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	$1 \rightarrow \text{bit n of d}$	set the bit in d
BSR	BW[3]	address[2]	------	-	-	-	-	-	-	-	-	-	-	-	-	PC $\rightarrow$ -(SP); address $\rightarrow$ 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	-	$\text{NOT(bit Dn of d)} \rightarrow Z$	Set Z with state of specified bit in d
		#n,d		d[1]	-	d	d	d	d	d	d	d	d	d	s	$\text{NOT(bit #n of d)} \rightarrow Z$	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	-	-	-	$0 \rightarrow d$	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 $\rightarrow$ Dn if Dn <> -1 then addr $\rightarrow$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	$\pm 32\text{bit } Dn / \pm 16\text{bit } s \rightarrow \pm Dn$	Dn= [16-bit remainder, 16-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-bit remainder, 16-bit quotient]
EOR [4]	BWL	Dn,d	-**00	e	-	d	d	d	d	d	d	d	-	-	s[4]	$Dn \text{ XOR } d \rightarrow d$	Logical exclusive OR Dn to destination
EORI [4]	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 [4]	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR } CCR \rightarrow CCR$	Logical exclusive OR #n to CCR
EORI [4]	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR } SR \rightarrow SR$	Logical exclusive OR #n to SR (Privileged)
EXG	L	Rx,Ry	------	e	e	-	-	-	-	-	-	-	-	-	-	register $\leftarrow\rightarrow$ register	Exchange registers (32-bit only)
EXT	WL	Dn	-**00	d	-	-	-	-	-	-	-	-	-	-	-	Dn.B $\rightarrow$ Dn.W	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	-	$\uparrow d \rightarrow$ PC	Jump to effective address of destination
JSR		d	------	-	-	d	-	-	d	d	d	d	d	d	-	PC $\rightarrow$ -(SP); $\uparrow d \rightarrow$ 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 LSR	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-	(shift diagram)	Logical shift Dy, Dx bits left/right
		#n,Dy		d	-	-	-	-	-	-	-	-	-	-	s		Logical shift Dy, #n bits L/R (#n: 1 to 8)
	W	d		-	-	d	d	d	d	d	d	d	-	-	-	(shift diagram)	Logical shift d 1 bit left/right (.W only)
MOVE [4]	BWL	s,d	-**00	e	s[4]	e	e	e	e	e	e	s	s	s	s[4]	$s \rightarrow d$	Move data from source to destination
MOVE	W	s,CCR	=====	s	-	s	s	s	s	s	s	s	s	s	s	$s \rightarrow$ CCR	Move source to Condition Code Register
MOVE	W	s,SR	=====	s	-	s	s	s	s	s	s	s	s	s	s	$s \rightarrow$ SR	Move source to Status Register (Privileged)
MOVE	W	SR,d	------	d	-	d	d	d	d	d	d	d	-	-	-	$SR \rightarrow$ d	Move Status Register to destination
MOVE	L	USP,An	------	-	d	-	-	-	-	-	-	-	-	-	-	USP $\rightarrow$ An	Move User Stack Pointer to An (Privileged)
		An,USP		-	s	-	-	-	-	-	-	-	-	-	-	An $\rightarrow$ USP	Move An to User Stack Pointer
	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	Dn	An	(An)	(An)+	-(An)	(i,An)	(i,An,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n	Operation	Description
	BWL	s,d	XNZVC														Effective Address s=source, d=destination, e=either, i=displacement
MOVEA⁴	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)
MOVEM⁴	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)
MOVEQ⁴	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	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	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 - d₁₀ - 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)
OR⁴	BWL	s,Dn	-**00	e	-	s	s	s	s	s	s	s	s	s	s⁴	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)
ORI⁴	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	s	#n OR d → d	Logical OR #n to destination
ORI⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n OR CCR → CCR	Logical OR #n to CCR
ORI⁴	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	-	-	-	-	-	-	-	-	-	-	-	Dx₁₀ - Dy₁₀ - X → Dx₁₀	Subtract BCD source and eXtend bit from
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ax)₁₀ - (Ay)₁₀ - X → -(Ax)₁₀	destination, BCD result
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)
SUB⁴	BWL	s,Dn	*****	e	s	s	s	s	s	s	s	s	s	s	s⁴	Dn - s → Dn	Subtract binary (SUBI or SUBQ used when
		Dn,d		e	d⁴	d	d	d	d	d	d	d	-	-	-	d - Dn → d	source is #n. Prevent SUBQ with #n.L)
SUBA⁴	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)
SUBI⁴	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	d - #n → d	Subtract immediate from destination
SUBQ⁴	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
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ax) - (Ay) - X → -(Ax)	destination
SWAP	W	Dn	-**00	d	-	-	-	-	-	-	-	-	-	-	-	bits[31:16] ←→ bits[15:0]	Exchange the 16-bit halves of Dn
TAS	B	d	-**00	d	-	d	d	d	d	d	d	d	-	-	-	test d → CCR; 1 → bit7 of d	N and Z set to reflect d, bit7 of d set to 1
TRAP		#n	-----	-	-	-	-	-	-	-	-	-	-	-	s	PC → -(SSP);SR → -(SSP); (vector table entry) → PC	Push PC and SR, PC set by vector table #n (#n range: 0 to 15)
TRAPV			-----	-	-	-	-	-	-	-	-	-	-	-	-	If V then TRAP #7	If overflow, execute an Overflow TRAP
TST	BWL	d	-**00	d	-	d	d	d	d	d	d	d	-	-	-	test d → CCR	N and Z set to reflect destination
UNLK		An	-----	-	d	-	-	-	-	-	-	-	-	-	-	An → SP; (SP)+ → An	Remove local workspace from stack
	BWL	s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(i,An)	(i,An,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n		

Condition Tests (+ OR, ! NOT, ⊕ XOR; 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
HIᵘ	higher than	!(C + Z)	PL	plus	!N
LSᵘ	lower or same	C + Z	MI	minus	N
HSᵘ, CCª	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LOᵘ, CSª	lower than	C	LT	less than	(N ⊕ V)
NE	not equal	!Z	GT	greater than	![(N ⊕ V) + Z]
EQ	equal	Z	LE	less or equal	(N ⊕ V) + Z

An Address register (16/32-bit, n=0-7)
Dn Data register (8/16/32-bit, n=0-7)
Rn any data or address register
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

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

Nom : .... Prénom : .... Classe : ....

DOCUMENT RÉPONSE À RENDRE

Exercice 1

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.L (A2)+,(A0)+	$005000 **13 79 01 80** E7 21 48 C0	A0 = $00005004 A2 = $00005014
MOVE.L 4(A2),4(A0)	$005000 54 AF 18 B9 **42 1A 2D 49**	Aucun changement
MOVE.B $500A,-1(A1,D0.W)	$005000 54 AF 18 B9 E7 21 **11** C0	Aucun changement
MOVE.L #$500A,-5(A1,D1.W)	$005008 **00 00 50 0A** D4 36 1F 88	Aucun changement
MOVE.W $500A,-(A1)	$005000 54 AF 18 B9 E7 21 **11 C8**	A1 = $00005006

Exercice 2

Opération	Taille (bits)	Résultat (hexadécimal)	N	Z	V	C
$F0 + $11	8	$01	0	0	0	1
$F0 + $11	16	$0101	0	0	0	0
$8000 + $8000	16	$0000	0	1	1	1
$40000000 + $80000000	32	$C0000000	1	0	0	0

Exercice 3

Valeur finale de **D1** : **$76542301**. Utilisez au maximum quatre lignes d'instructions.

```
                              ; D1 = $ 7654 3210
        ror.b   #4,d1         ; D1 = $ 7654 3201
        ror.w   #8,d1         ; D1 = $ 7654 0132
        ror.b   #4,d1         ; D1 = $ 7654 0123
        ror.w   #8,d1         ; D1 = $ 7654 2301
```

Valeur finale de **D1** : **$54231067**. Utilisez au maximum quatre lignes d'instructions.

```
                              ; D1 = $ 7654 3210
        ror.l   #8,d1         ; D1 = $ 1076 5432
        ror.b   #4,d1         ; D1 = $ 1076 5423
        swap    d1            ; D1 = $ 5423 1076
        ror.b   #4,d1         ; D1 = $ 5423 1067
```

Exercice 4

Question	Réponse
Donnez deux directives d'assemblage.	ORG, DC
Combien de registres d'état possède le 68000 ?	1 seul
Quelle est la taille du registre CCR ?	8 bits
Quel mode du 68000 a des privilèges limités ?	Le mode utilisateur

Exercice 5

Valeurs des registres après exécution du programme. **Utilisez la représentation hexadécimale sur 32 bits.**	
D1 = $00000001	**D4** = $00000045
D2 = $00000001	**D5** = $00000014
D3 = $00000021	**D6** = $00000001