

Partiel S3 – Corrigé

Architecture des ordinateurs

Durée : 1 h 30

Répondre exclusivement sur le document réponse.

Ne pas écrire à l'encre rouge ni au crayon à papier.

Exercice 1 (3 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 = \$FFFF0005 A0 = \$00005000 PC = \$00006000
 D1 = \$FFFFFFE0 A1 = \$00005008
 D2 = \$AAAA0018 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 (2 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 (4 points)

Soit le programme ci-dessous. Complétez le tableau présent sur le [document réponse](#).

```

Main      move.l  #$ff,d7
next1     moveq.l #1,d1
          cmpi.w  #$fe,d7
          ble     next2
          moveq.l #2,d1
next2     moveq.l #1,d2
          cmpi.b  #$fe,d7
          ble     next3
          moveq.l #2,d2
next3     clr.l   d3
          move.l  #518,d0
loop3     addq.l  #1,d3
          subq.b  #2,d0
          bne     loop3
next4     clr.l   d4
          clr.l   d0
loop4     addq.l  #1,d4
          dbra    d0,loop4      ; DBRA = DBF
  
```

Exercice 4 (11 points)

Toutes les questions de cet exercice sont indépendantes. **À l'exception des registres utilisés pour renvoyer une valeur de sortie, aucun registre de donnée ou d'adresse ne devra être modifié en sortie de vos sous-programmes.** Une chaîne de caractères se termine toujours par un caractère nul (la valeur zéro). On suppose pour tout l'exercice que les chaînes ne sont jamais vides (elles possèdent au moins un caractère non nul).

1. Réalisez le sous-programme **GetStart** qui renvoie l'adresse de la première occurrence d'un caractère dans une chaîne.

Entrée : **A0.L** pointe sur une chaîne de caractères.

D0.B contient le code ASCII d'un caractère. On appelle C ce caractère et l'on suppose qu'il est présent dans la chaîne pointée par **A0.L**.

Sortie : **A0.L** pointe sur la première occurrence de C dans la chaîne.

Attention ! le sous-programme GetStart est limité à 4 lignes d'instructions.

2. Réalisez le sous-programme **GetEnd** qui renvoie l'adresse située juste après le dernier caractère d'une suite de caractères identiques. On considère qu'une suite de caractères identiques peut être constituée d'un seul caractère ou de plusieurs caractères identiques.

Entrée : **A0.L** pointe sur un caractère non nul dans une chaîne. On appelle C ce caractère.

Sortie :

- Si le caractère qui suit C est différent de C, alors **A0.L** pointera sur le caractère qui suit C.
- Si le caractère C est répété plusieurs fois à la suite, alors **A0.L** pointera sur le caractère qui suit le dernier C.

Par exemple, considérons la chaîne suivante : « Heeeellooooo Woorld »

- Si **A0.L** pointe sur le « H », alors l'adresse renvoyée sera celle du premier « e ».
- Si **A0.L** pointe sur le premier « e », alors l'adresse renvoyée sera celle du premier « l ».
- Si **A0.L** pointe sur le premier « l », alors l'adresse renvoyée sera celle du premier « o ».
- Si **A0.L** pointe sur le premier « o », alors l'adresse renvoyée sera celle de l'espace.
- Si **A0.L** pointe sur le « r », alors l'adresse renvoyée sera celle du dernier « l ».
- Si **A0.L** pointe sur le « d », alors l'adresse renvoyée sera celle du caractère nul.

Attention ! le sous-programme GetEnd est limité à 12 lignes d'instructions.

3. À l'aide des sous-programmes **GetStart** et **GetEnd**, réalisez le sous-programme **SuccessiveCount** qui compte le nombre de caractères dans une suite de caractères identiques. Cette suite de caractères se trouve dans une chaîne. Si plusieurs suites d'un même caractère se trouvent dans la chaîne, alors seule la première suite de caractères sera prise en compte.

Entrée : **A0.L** pointe sur une chaîne de caractères.

D0.B contient le code ASCII d'un caractère. On appelle C ce caractère et l'on suppose qu'il est présent dans la chaîne pointée par **A0.L**.

Sortie : **D0.L** contient le nombre de caractères C successifs à partir du premier C.

Par exemple, considérons que **A0.L** pointe sur la chaîne suivante : « Heeeellooooo Woorld »


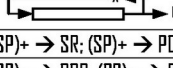
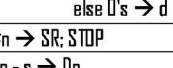
- Si **D0.B** contient « H », alors la valeur renvoyée sera 1.
- Si **D0.B** contient « e », alors la valeur renvoyée sera 4.
- Si **D0.B** contient « l », alors la valeur renvoyée sera 2.
- Si **D0.B** contient « o », alors la valeur renvoyée sera 5.
- Si **D0.B** contient « W », alors la valeur renvoyée sera 1.
- Si **D0.B** contient « d », alors la valeur renvoyée sera 1.

Attention ! le sous-programme SuccessiveCount est limité à 12 lignes d'instructions.

EASy68K Quick Reference v1.8<http://www.wowgwp.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
	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	-	-	-	-	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow D_{x10}$ $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	Add BCD source and eXtend bit to destination, BCD result	
ADD ⁴	BWL	s,Dn Dn,d	*****	e	s ⁴	s	s	s	s	s	s	s	s	s	s ⁴	$s + Dn \rightarrow Dn$ $Dn + d \rightarrow d$	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	$s + An \rightarrow An$	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 -(Ay),-(Ax)	*****	e	-	-	-	-	-	-	-	-	-	-	-	$Dy + Dx + X \rightarrow Dx$ $-(Ay) + -(Ax) + X \rightarrow -(Ax)$	Add source and eXtend bit to destination	
AND ⁴	BWL	s,Dn Dn,d	-**00	e	-	s	s	s	s	s	s	s	s	s	s ⁴	$s \text{ AND } Dn \rightarrow Dn$ $Dn \text{ AND } d \rightarrow d$	Logical AND source to destination (ANDI is used when source is #n)	
ANDI 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	BWL	Dx,Dy	*****	e	-	-	-	-	-	-	-	-	-	-	-		Arithmetic shift Dy by Dx bits left/right	
ASR	W	#n,Dy d		d	-	-	-	-	-	-	-	-	-	-	s		Arithmetic shift Dy #n bits L/R (#n: 1 to 8) Arithmetic shift d 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 \pm offset to address)	
BCHG	B L	Dn,d #n,d	---*---	e ¹	-	d	d	d	d	d	d	d	-	-	-	$\text{NOT}(\text{bit number of } d) \rightarrow Z$ $\text{NOT}(\text{bit } n \text{ of } d) \rightarrow \text{bit } n \text{ of } d$	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	d	d	d	d	d	d	-	-	-	$\text{NOT}(\text{bit number of } d) \rightarrow Z$ $0 \rightarrow \text{bit number of } d$	Set Z with state of specified bit in d then clear the bit in d	
BRA	BW ³	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	d	d	d	d	d	d	-	-	-	$\text{NOT}(\text{bit } n \text{ of } d) \rightarrow Z$ $1 \rightarrow \text{bit } n \text{ of } d$	Set Z with state of specified bit in d then set the bit in d	
BSR	BW ³	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	d	d	d	d	d	d	d	d	-	$\text{NOT}(\text{bit } Dn \text{ of } d) \rightarrow Z$ $\text{NOT}(\text{bit } \#n \text{ of } d) \rightarrow Z$	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 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,address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc false then { $Dn-1 \rightarrow Dn$ if $Dn < 0$ 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	$32\text{bit } Dn / 16\text{bit } s \rightarrow Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$	
EDR 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	
EDRI 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	
EDRI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ XOR } CCR \rightarrow CCR$	Logical exclusive OR #n to CCR	
EDRI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ XOR } SR \rightarrow SR$	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	-	$\uparrow d \rightarrow$ 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	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, Dx bits left/right	
LSR	W	#n,Dy d		d	-	-	-	-	-	-	-	-	-	-	s		Logical shift Dy, #n bits L/R (#n: 1 to 8) Logical shift d 1 bit left/right (.W only)	
MOVE ⁴	BWL	s,d	-**00	e	s ⁴	e	e	e	e	e	e	e	s	s	s ⁴	$s \rightarrow d$	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 An,USP	-----	-	d	-	-	-	-	-	-	-	-	-	-	USP \rightarrow An An \rightarrow USP	Move User Stack Pointer to An (Privileged) Move An to User Stack Pointer (Privileged)	
	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)	(iAn)	(iAn,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n			
MOVEA ⁴	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)
MOVEM ⁴	WL	Rn-Rn,d s,Rn-Rn	-----	-	-	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	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	-	-	-	-	D - d ₁₀ - X → d	Negate BCD with eXtend, BCD result
NEG	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	-	D - d → d	Negate destination (2's complement)
NEGX	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	-	D - 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	s ⁴	s OR Dn → Dn Dn OR d → d	Logical OR (ORI is used when source is #n)
ORI ⁴	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	-	s	#n OR d → d	Logical OR #n to destination
ORI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	-	s	#n OR CCR → CCR	Logical OR #n to CCR
ORI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	-	s	#n OR SR → SR	Logical OR #n to SR (Privileged)
PEA	L	s	-----	-	-	s	-	-	s	s	s	s	s	s	s	-	↑s → -(SP)	Push effective address of s onto stack
RESET			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	Assert RESET Line	Issue a hardware RESET (Privileged)
RDL	BWL	Dx,Dy	-**0*	e	-	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits left/right (without X)
RDR		#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)
RDXL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits L/R, X used then updated
RDXR		#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 -(Ay),-(Ax)	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	-	Dx ₁₀ - Dy ₁₀ - X → Dx ₁₀ -(Ax) ₁₀ - -(Ay) ₁₀ - X → -(Ax) ₁₀	Subtract BCD source and eXtend bit from 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)
SUB ⁴	BWL	s,Dn Dn,d	*****	e	s	s	s	s	s	s	s	s	s	s	s	s ⁴	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	s	An - s → An	Subtract address (.W sign-extended to .L)
SUBI ⁴	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	-	s	d - #n → d	Subtract immediate from destination
SUBQ ⁴	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	-	s	d - #n → d	Subtract quick immediate (#n range: 1 to 8)
SUBX	BWL	Dy,Dx -(Ay),-(Ax)	*****	e	-	-	-	-	-	-	-	-	-	-	-	-	Dx - Dy - X → Dx -(Ax) - -(Ay) - X → -(Ax)	Subtract source and eXtend bit from 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)	(iAn)	(iAn,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
T	true	I	VC	overflow clear	IV
F	false	O	VS	overflow set	V
HI*	higher than	I(C + Z)	PL	plus	!N
LS*	lower or same	C + Z	MI	minus	N
HS*, CC*	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LD*, CS*	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, **O** cleared, **I** set, **U** undefined

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

Distributed under the GNU general public use license.

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 #2943,4(A0)	\$005000 54 AF 18 B9 00 00 0B 7F	Aucun changement
MOVE.B \$5011,34(A2,D1.L)	\$005010 13 79 79 80 42 1A 2D 49	Aucun changement
MOVE.W 18(A0), -24(A0,D2.W)	\$005000 01 80 18 B9 E7 21 48 C0	Aucun changement

Exercice 2

Opération	Taille (bits)	Résultat (hexadécimal)	N	Z	V	C
\$5D + \$6F	8	\$CC	1	0	1	0
\$87654321 + \$ABCDEF00	32	\$33333221	0	0	1	1

Exercice 3

Valeurs des registres après exécution du programme. Utilisez la représentation hexadécimale sur 32 bits.	
D1 = \$00000002	D3 = \$00000003
D2 = \$00000002	D4 = \$00000001

Exercise 4

```
GetStart      cmp.b    (a0)+,d0
              bne      GetStart

              subq.l   #1,a0
              rts
```

```
GetEnd        move.l   d0,-(a7)

              move.b   (a0)+,d0

\loop         cmp.b    (a0)+,d0
              beq      \loop

              subq.l   #1,a0
              move.l   (a7)+,d0
              rts
```

```
SuccessiveCount move.l   a0,-(a7)

              jsr      GetStart
              move.l   a0,d0

              jsr      GetEnd
              suba.l   d0,a0
              move.l   a0,d0

              move.l   (a7)+,a0
              rts
```