

Partiel S4 – Corrigé

Architecture des ordinateurs

Durée : 1 h 30

Répondre exclusivement sur le document réponse.

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 = \$00000003 A0 = \$00005000 PC = \$00006000
 D1 = \$0001FFFF A1 = \$00005008
 D2 = \$FFFFFFFF 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)

Remplir le tableau présent sur le [document réponse](#). Vous devez trouver le nombre manquant (sous sa forme hexadécimale) en fonction de la taille de l'opération et de la valeur des *flags* après l'opération. **Si plusieurs solutions sont possibles, vous retiendrez uniquement la plus petite.**

Exercice 3 (4 points)

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

```

Main      move.l  #$80100200,d7
next1     moveq.l #1,d1
          tst.w   d7
          bmi   next2
          moveq.l #2,d1
next2     moveq.l #1,d2
          cmpi.w #200,d7
          blo   next3
          moveq.l #2,d2
next3     clr.l   d3
          move.w #$F0F0,d0
loop3     addq.l  #1,d3
          subq.b #1,d0
          bne   loop3
next4     clr.l   d4
          move.w  #$22,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 **IsNumber** qui détermine si une chaîne de caractères contient uniquement des chiffres.

Entrée : **A0.L** pointe sur une chaîne qui n'est pas vide.

Sortie : Si la chaîne contient uniquement des chiffres, **D0.L** renvoie 0.

Autrement, **D0.L** renvoie 1.

Le sous-programme IsNumber est limité à 15 lignes d'instructions (RTS compris).

2. Réalisez le sous-programme **GetSum** qui additionne tous les chiffres présents dans une chaîne de caractères.

Entrée : **A0.L** pointe sur une chaîne qui n'est pas vide et qui contient uniquement des chiffres.

Sortie : **D0.L** renvoie la somme de tous les chiffres de la chaîne.

Exemple :

A0 →

'7'	'0'	'4'	'8'	'9'	'4'	'2'	'0'	'3'	0
-----	-----	-----	-----	-----	-----	-----	-----	-----	---

D0 doit renvoyer la valeur 37 ($37 = 7 + 0 + 4 + 8 + 9 + 4 + 2 + 0 + 3$).

Le sous-programme GetSum est limité à 13 lignes d'instructions (RTS compris).

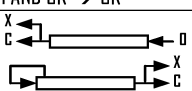
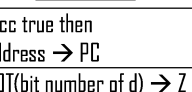
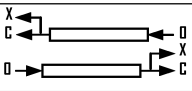
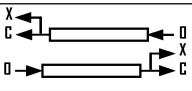
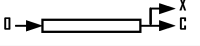
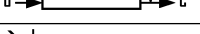
3. À l'aide des sous-programmes **IsNumber** et **GetSum**, réalisez le sous-programme **Checksum** qui renvoie la somme des chiffres d'une chaîne de caractères.

Entrée : **A0.L** pointe sur une chaîne qui n'est pas vide.

Sortie : Si la chaîne contient uniquement des chiffres : **D0.L** renvoie 0 et **D1.L** renvoie la somme.

Autrement : **D0.L** renvoie 1 et **D1.L** renvoie 0.

Le sous-programme CheckSum est limité à 12 lignes d'instructions (RTS compris).

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	-	-	-	-	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$ $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	Add BCD source and eXtend bit to destination, BCD result
ADD ⁴	BWL	s,Dn Dn,d	*****	e	s	s	s	s	s	s	s	s	s	s	s	$s + Dn \rightarrow Dn$ $Dn + d \rightarrow d$	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	-	-	-	$\#n + d \rightarrow d$	Add immediate to destination
ADDQ ⁴	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	-	$\#n + d \rightarrow d$	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	$s \text{ AND } Dn \rightarrow Dn$ $Dn \text{ AND } d \rightarrow d$	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	-	-	-	$\#n \text{ AND } d \rightarrow d$	Logical AND immediate to destination
ANDI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ AND } CCR \rightarrow CCR$	Logical AND immediate to CCR
ANDI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ AND } SR \rightarrow SR$	Logical AND immediate to SR (Privileged)
ASL	BWL	Dx,Dy	*****	e	-	-	-	-	-	-	-	-	-	-	-		Arithmetic shift Dy by Dx bits left/right
ASR	BWL	#n,Dy	*****	d	-	-	-	-	-	-	-	-	-	-	-		Arithmetic shift Dy #n bits L/R (#n: 1 to 8)
ASR	W	d	*****	-	-	d	d	d	d	d	d	d	-	-	-		Arithmetic shift ds 1 bit left/right (.W only)
Bcc	BW ³	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	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 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	#n,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 → PC	Branch always (8 or 16-bit ± 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 → -(SP); address → PC	Branch to subroutine (8 or 16-bit ± offset)
BTST	B L	Dn,d #n,d	---*---	e ¹	-	d	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	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	-	-	-	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 $Dn < -1$ 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	$\pm 32\text{bit } Dn / \pm 16\text{bit } s \rightarrow \pm Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
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 ⁴	BWL	Dn,d	-**00	e	-	d	d	d	d	d	d	d	-	-	-	$Dn \text{ XOR } d \rightarrow d$	Logical exclusive OR Dn to destination
EORI ⁴	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	-	$\#n \text{ XOR } d \rightarrow d$	Logical exclusive OR #n to destination
EORI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ XOR } CCR \rightarrow CCR$	Logical exclusive OR #n to CCR
EORI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ XOR } SR \rightarrow SR$	Logical exclusive OR #n to SR (Privileged)
EXG	L	Rx,Ry	-----	e	e	-	-	-	-	-	-	-	-	-	-	register ↔ 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 → -(SSP); SR → -(SSP)	Generate Illegal Instruction exception
JMP		d	-----	-	-	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	d	PC → -(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	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	BWL	#n,Dy	***0*	d	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, #n bits L/R (#n: 1 to 8)
LSR	W	d	***0*	-	-	d	d	d	d	d	d	d	-	-	-		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	$s \rightarrow d$	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)
MOVE	L	An,USP	-----	-	s	-	-	-	-	-	-	-	-	-	-	$An \rightarrow USP$	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		

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 20488, -(A2)	\$005008 C9 10 11 C8 C9 10 11 C8	A2 = \$0000500C
MOVE.B 5(A0), 10(A1, D1.W)	\$005000 54 AF 21 B9 E7 21 48 C0	Aucun changement
MOVE.W -4(A2), 6(A0, D2.L)	\$005000 D4 36 18 B9 E7 21 48 C0	Aucun changement

Exercice 2

Opération	Taille (bits)	Nombre manquant (hexadécimal)	N	Z	V	C
\$1E + \$?	8	\$62	1	0	1	0
\$\$FF1E + \$?	16	\$80E2	1	0	0	1

Exercice 3

Valeurs des registres après exécution du programme. Utilisez la représentation hexadécimale sur 32 bits.	
D1 = \$00000002	D3 = \$000000F0
D2 = \$00000002	D4 = \$00000023

Exercice 4

```

IsNumber      move.l  a0,-(a7)

\loop         move.b  (a0)+,d0
              beq     \number

              cmpi.b  #'0',d0
              blo     \notANumber

              cmpi.b  #'9',d0
              bls     \loop

\notANumber   moveq.l  #1,d0
              bra     \quit

\number      clr.l   d0
\quit        movea.l  (a7)+,a0
              rts

```

```

GetSum        movem.l  a0/d1,-(a7)

              clr.l   d0
              clr.l   d1

\loop         move.b  (a0)+,d1
              beq     \quit

              sub.b   #'0',d1
              add.l   d1,d0
              bra     \loop

\quit        movem.l  (a7)+,a0/d1
              rts

```

```

Checksum      jsr     IsNumber
              tst.l   d0
              bne     \notANumber

\number      jsr     GetSum
              move.l  d0,d1
              clr.l   d0
              rts

\notANumber   clr.l   d1
              rts

```