

Contrôle S3

Architecture des ordinateurs

Durée : 1 h 30

Répondre exclusivement sur le document réponse.

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 = \$FFFF0005 A0 = \$00005000 PC = \$00006000
 D1 = \$00000008 A1 = \$00005008
 D2 = \$0000FFFA 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)

Réalisez le sous-programme **AlphaCount** qui renvoie le nombre de caractères alphanumériques dans une chaîne de caractères. Une chaîne de caractères se termine par un caractère nul (la valeur zéro). À l'exception des registres de sortie, aucun registre de donnée ou d'adresse ne devra être modifié en sortie de ce sous-programme.

Entrée : **A0.L** pointe sur le premier caractère d'une chaîne de caractères.

Sortie : **D0.L** renvoie le nombre de caractères alphanumériques de la chaîne.

Indications :

- Un caractère alphanumérique est une lettre (minuscule ou majuscule) ou un chiffre (de 0 à 9).
- On considère que les trois sous-programmes ci-dessous sont déjà écrits et que vous pouvez les appeler (ils ne modifient que **D0**) :
 - **LowerCount** renvoie dans **D0** le nombre de minuscules dans une chaîne pointée par **A0**.
 - **UpperCount** renvoie dans **D0** le nombre de majuscules dans une chaîne pointée par **A0**.
 - **DigitCount** renvoie dans **D0** le nombre de chiffres dans une chaîne pointée par **A0**.

Attention ! le sous-programme AlphaCount est limité à 10 lignes d'instructions.

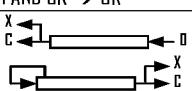
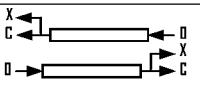
Exercice 4 (2 points)

Répondez aux questions sur le [document réponse](#).

Exercice 5 (6 points)

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

Main	<code>move.l #520037f0,d7</code>
next1	<code>moveq.l #1,d1</code> <code>tst.w d7</code> <code>beq next2</code> <code>moveq.l #2,d1</code>
next2	<code>moveq.l #1,d2</code> <code>cmpi.l #ffffff,d7</code> <code>blo next3</code> <code>moveq.l #2,d2</code>
next3	<code>clr.l d3</code> <code>move.l #66666666,d0</code>
loop3	<code>addq.l #1,d3</code> <code>subq.b #2,d0</code> <code>bne loop3</code>
next4	<code>clr.l d4</code> <code>move.l #66666666,d0</code>
loop4	<code>addq.l #1,d4</code> <code>dbra d0,loop4 ; DBRA = DBF</code>
next5	<code>move.l d7,d5</code> <code>move.w #ffff,d5</code> <code>swap d5</code> <code>tst.b d5</code> <code>beq next6</code> <code>swap d5</code>
next6	<code>move.l d7,d6</code> <code>ror.b #4,d6</code> <code>ror.w #4,d6</code> <code>swap d6</code> <code>rol.l #8,d6</code> <code>rol.l #4,d6</code> <code>ror.w #8,d6</code>
quit	<code>illegal</code>

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 ⁴	s	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	$s + An \rightarrow An$	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 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	-	-	-	-	$#n \text{ AND } d \rightarrow d$	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	-	-	-	-	-	-	-	-	-	-	-	-	s	Arithmetic shift Dy #n bits L/R (#n: 1 to 8)
	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 (8 or 16-bit ± offset to address)
BCHG	B L	Dn,d #n,d	---*---	e	-	d	d	d	d	d	d	d	-	-	-	-	-	NOT(bit number of d) → Z NOT(bit n of d) → bit n of d
BCLR	B L	Dn,d #n,d	---*---	e	-	d	d	d	d	d	d	d	-	-	-	-	-	NOT(bit number of d) → Z 0 → bit number of d
BRA	BW ³	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	address → PC
BSET	B L	Dn,d #n,d	---*---	e	-	d	d	d	d	d	d	d	-	-	-	-	-	NOT(bit n of d) → Z 1 → bit n of d
BSR	BW ³	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	PC → -(SP); address → PC
BTST	B L	Dn,d #n,d	---*---	e	-	d	d	d	d	d	d	d	d	d	d	d	-	NOT(bit Dn of d) → Z NOT(bit #n of d) → Z
CHK	W	s,Dn	-*UUU	e	-	s	s	s	s	s	s	s	s	s	s	s	s	if Dn<0 or Dn>s then TRAP
CLR	BWL	d	-0100	d	-	d	d	d	d	d	d	d	-	-	-	-	-	0 → d
CMP ⁴	BWL	s,Dn	-*****	e	s ⁴	s	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	s	s	set CCR with An - s
CMPI ⁴	BWL	#n,d	-*****	d	-	d	d	d	d	d	d	d	-	-	-	-	-	set CCR with d - #n
CMPP ⁴	BWL	(Ay)+,(Ax)+	-*****	-	-	-	e	-	-	-	-	-	-	-	-	-	-	set CCR with (Ax) - (Ay)
DBcc	W	Dn,address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	if cc false then { Dn-1 → Dn if Dn < -1 then addr → PC }
DIVS	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	s	s	±32bit Dn / ±16bit s → ±Dn
DIVU	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	s	s	32bit Dn / 16bit s → Dn
EOR ⁴	BWL	Dn,d	-**00	e	-	d	d	d	d	d	d	d	-	-	-	-	-	Dn XOR d → d
EORI ⁴	BWL	#n,d	-**00	d	-	d	d	d	d	d	d	d	-	-	-	-	-	#n XOR d → d
EORI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	-	-	-	#n XOR CCR → CCR
EORI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	-	-	-	#n XOR SR → SR
EXG	L	Rx,Ry	-----	e	e	-	-	-	-	-	-	-	-	-	-	-	-	register ↔ register
EXT	WL	Dn	-**00	d	-	-	-	-	-	-	-	-	-	-	-	-	-	Dn.B → Dn.W Dn.W → Dn.L
ILLEGAL			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	PC → -(SSP); SR → -(SSP)
JMP		d	-----	-	-	d	-	-	d	d	d	d	d	d	d	d	d	↑d → PC
JSR		d	-----	-	-	d	-	-	d	d	d	d	d	d	d	d	d	PC → -(SP); ↑d → PC
LEA	L	s,An	-----	-	e	s	-	-	s	s	s	s	s	s	s	s	s	↑s → An
LINK		An,#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	An → -(SP); SP → An; SP + #n → SP
LSL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-	-	-	
LSR	BWL	#n,Dy	***0*	d	-	-	-	-	-	-	-	-	-	-	-	-	-	Logical shift Dy, #n bits L/R (#n: 1 to 8)
	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 ⁴	s → d	Move data from source to destination
MOVE	W	s,CCR	=====	s	-	s	s	s	s	s	s	s	s	s	s	s	s	s → CCR
MOVE	W	s,SR	=====	s	-	s	s	s	s	s	s	s	s	s	s	s	s	s → SR
MOVE	W	SR,d	-----	d	-	d	d	d	d	d	d	d	-	-	-	-	-	SR → d
MOVE	L	USP,An	-----	-	d	-	-	-	-	-	-	-	-	-	-	-	-	USP → An
		An,USP	-----	-	s	-	-	-	-	-	-	-	-	-	-	-	-	An → USP

Architecture des ordinateurs – EPITA – S3 – 2023/2024

Opcode	Size	Operand	CCR	Effective Address												Operation	Description				
				s=source, d=destination, e=either, i=displacement	Dn	An	(An)	(An)+	-(An)	(i,An)	(i,An,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	s	s	$s \rightarrow An$	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 \rightarrow d s \rightarrow 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 \rightarrow (i,An)...(i+2,An)...(i+4,A. (i,An) \rightarrow 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 \rightarrow 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	s	s	s	$\pm 16\text{bit } s * \pm 16\text{bit } Dn \rightarrow \pm Dn$	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	s	s	s	16bit s * 16bit Dn \rightarrow Dn	Multiply unsig'd 16-bit; result: unsig'd 32-bit
NBCD	B	d	*0*0*	d	-	d	d	d	d	d	d	d	-	-	-	-	-	-	-	D - d ₁₀ - X \rightarrow d	Negate BCD with eXtend, BCD result
NEG	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	-	-	-	-	D - d \rightarrow d	Negate destination (2's complement)
NEGX	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	-	-	-	-	D - d - X \rightarrow d	Negate destination with eXtend
NDP			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	None	No operation occurs
NOT	BWL	d	---*00	d	-	d	d	d	d	d	d	d	-	-	-	-	-	-	-	NOT(d) \rightarrow 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	s	s	s	s OR Dn \rightarrow Dn Dn OR d \rightarrow 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 \rightarrow d	Logical OR #n to destination	
ORI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	-	-	-	s	#n OR CCR \rightarrow CCR	Logical OR #n to CCR	
ORI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	-	-	-	s	#n OR SR \rightarrow SR	Logical OR #n to SR (Privileged)	
PEA	L	s	-----	-	-	s	-	-	s	s	s	s	s	s	s	s	s	s	s	$\uparrow_s \rightarrow$ -(SP)	Push effective address of s onto stack
RESET			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Assert RESET Line	Issue a hardware RESET (Privileged)
ROL	BWL	Dx,Dy #n,Dy	---*0*	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Rotate Dy, Dx bits left/right (without X)	
ROR	W	d	---*0*	d	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Rotate Dy, #n bits left/right (#n: 1 to 8)	
ROXL	BWL	Dx,Dy #n,Dy	---*0*	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Rotate Dy, Dx bits L/R, X used then updated	
ROXR	W	d	---*0*	d	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Rotate Dy, #n bits left/right (#n: 1 to 8)	
RTE			=====	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ \rightarrow SR; (SP)+ \rightarrow PC	Return from exception (Privileged)
RTR			=====	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ \rightarrow CCR; (SP)+ \rightarrow PC	Return from subroutine and restore CCR
RTS			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ \rightarrow PC	Return from subroutine
SBCD	B	Dy,Dx -(Ay)- (Ax)	*0*0*	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Dx ₁₀ - Dy ₁₀ - X \rightarrow Dx ₁₀ -(Ax) ₁₀ - (Ay) ₁₀ - X \rightarrow -(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 true then 1's \rightarrow d else 0's \rightarrow d	If cc true then d.B = 11111111 else d.B = 00000000
STOP		#n	=====	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	s	#n \rightarrow 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	s	s	s	Dn - s \rightarrow Dn d - Dn \rightarrow 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	s	s	s	An - s \rightarrow An	Subtract address (.W sign-extended to .L)
SUBI ⁴	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	-	-	-	-	-	d - #n \rightarrow d	Subtract immediate from destination
SUBQ ⁴	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	-	-	-	-	-	d - #n \rightarrow d	Subtract quick immediate (#n range: 1 to 8)
SUBX	BWL	Dy,Dx -(Ay)- (Ax)	*****	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Dx - Dy - X \rightarrow Dx -(Ax) - (Ay) - X \rightarrow -(Ax)	Subtract source and eXtend bit from destination
SWAP	W	Dn	---*00	d	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	bits[31:16] \leftrightarrow bits[15:0]	Exchange the 16-bit halves of Dn
TAS	B	d	---*00	d	-	d	d	d	d	d	d	d	-	-	-	-	-	-	-	test d \rightarrow CCR; 1 \rightarrow bit7 of d	N and Z set to reflect d, bit7 of d set to 1
TRAP		#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	s	PC \rightarrow -(SSP); SR \rightarrow -(SSP); (vector table entry) \rightarrow 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 \rightarrow CCR	N and Z set to reflect destination
UNLK		An	-----	-	d	-	-	-	-	-	-	-	-	-	-	-	-	-	-	An \rightarrow SP; (SP)+ \rightarrow 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
HI ^u	higher than	!(C + Z)	PL	plus	!N
LS ^u	lower or same	C + Z	MI	minus	N
HS ^u , CC ^a	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LD ^u , CS ^a	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

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.B #18, -(A1)		
MOVE.W \$500E, (A2)+		
MOVE.W #\$500E, 2(A1)		
MOVE.B 7(A0), 7(A1, D2.W)		
MOVE.L -4(A2), -6(A1, D1.L)		

Exercice 2

Opération	Taille (bits)	Résultat (hexadécimal)	N	Z	V	C
\$59 + \$A7	8					
\$FFFF + \$AAAA	16					
\$FFFF + \$0100	16					
\$76543210 + \$12345678	32					

Exercice 3

AlphaCount

Exercice 4

Question	Réponse
Est-ce que le mode d'adressage Dn spécifie un emplacement mémoire ?	
Quels sont les noms des pointeurs de pile superviseur et utilisateur ?	
Si A7 = \$5004 juste avant un RTS, quelle est sa nouvelle valeur juste après le RTS ?	
Dans quel registre se trouvent les drapeaux (<i>flags</i>) ?	

Exercice 5

Valeurs des registres après exécution du programme. Utilisez la représentation hexadécimale sur 32 bits.		
D1 = \$	D3 = \$	D5 = \$
D2 = \$	D4 = \$	D6 = \$