

Contrôle S4

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 = \$FFFF0002 A0 = \$00005000 PC = \$00006000
 D1 = \$12340004 A1 = \$00005008
 D2 = \$FFFFFFF2 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	<code>move.l #\$87654321,d7</code>
next1	<code>moveq.l #1,d1 cmpl.w #0xff,d7 bgt next2 moveq.l #2,d1</code>
next2	<code>move.l d7,d2 lsl.l #4,d2 ror.w #4,d2 swap d2 rol.l #8,d2</code>
next3	<code>clr.l d3 move.l d7,d0</code>
loop3	<code>addq.l #1,d3 subl.b #11,d0 bne loop3</code>
next4	<code>clr.l d4 move.l d7,d0</code>
loop4	<code>addq.l #1,d4 dbra d0,loop4 ; DBRA = DBF</code>

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.**

L'objectif de cet exercice est de réaliser le fondu de fermeture d'une couleur d'arrière-plan. C'est-à-dire de faire tendre progressivement la couleur d'arrière-plan vers la couleur noire.

Une couleur possède trois composantes :

- Une composante rouge ;
- Une composante verte ;
- Une composante bleue.

Les trois composantes sont encodées dans un mot de 32 bits : $00RRGGBB_{16}$

- RR représente la valeur de la composante rouge (entier sur 8 bits non signés compris entre 0_{16} et FF_{16}) ;
- GG représente la valeur de la composante verte (entier sur 8 bits non signés compris entre 0_{16} et FF_{16}) ;
- BB représente la valeur de la composante bleue (entier sur 8 bits non signés compris entre 0_{16} et FF_{16}).

Par exemple :

- Si la couleur d'arrière-plan vaut $002B048D_{16}$, alors sa composante rouge sera $2B_{16}$, sa composante verte 04_{16} et sa composante bleue $8D_{16}$;
- La couleur noire sera encodée 00000000_{16} ;
- La couleur blanche sera encodée $00FFFFFF_{16}$.

1. Pour commencer, réalisez le sous-programme **Decrement** qui décrémente un entier codé sur 8 bits non signés en limitant sa valeur minimale à 0.

Entrées : **D0.B** contient un entier codé sur 8 bits non signés.

D1.B contient un entier codé sur 8 bits non signés.

Sortie : **D0.B** = **D0.B** – **D1.B** si le résultat n'est pas négatif.

D0.B = 0 si **D0.B** – **D1.B** est négatif.

Attention ! le sous-programme Decrement est limité à 4 lignes d'instructions (RTS compris).

2. À l'aide du sous-programme **Decrement**, réalisez le sous-programme **Darker** qui décrémente les trois composantes (rouge, verte et bleue) d'une couleur et qui limite chaque composante à 0.

Entrées : **D0.L** contient une couleur codée sur 32 bits (00RRGGBB₁₆).

D1.B contient un entier codé sur 8 bits non signés.

Sortie : **D0.L** renvoie la nouvelle couleur dont chaque composante a été décrémentée de **D1.B**.
Lorsqu'une composante a atteint 0, elle reste à 0.

Par exemple :

```
Main      move.l  #$00c0306,d0    ; D0.L = $000C0306
           move.b  #4,d1      ; D1.B = $04
           jsr    Darker      ; D0.L = $00080002
           jsr    Darker      ; D0.L = $00040000
           jsr    Darker      ; D0.L = $00000000
           jsr    Darker      ; D0.L = $00000000
```

Attention ! le sous-programme Darker est limité à 7 lignes d'instructions au maximum et vous devez utiliser uniquement les instructions JSR, ROR, SWAP et RTS.

3. La carte graphique utilise la valeur encodée sur 32 bits contenue dans l'adresse mémoire BackgroundColor. Dès que cette valeur est changée, la couleur d'arrière-plan sur l'écran est modifiée. Nous souhaitons faire tendre graduellement cette couleur vers le noir.

À l'aide du sous-programme **Darker**, réalisez le sous-programme **FadeOut** qui décrémente graduellement les trois composantes (rouge, verte et bleue) d'une couleur jusqu'à atteindre la couleur noire.

Entrée : **A0.L** pointe sur l'adresse contenant la couleur codée sur 32 bits à modifier.

Sortie : La couleur présente dans la case mémoire pointée par **A0.L** est modifiée.

La couleur est codée sur 32 bits et chaque composante est décrémentée de un en un.

Prenons par exemple le programme principal suivant :

```
Main      lea    BackgroundColor,a0
           jsr    FadeOut

           ; ...
           ; ...

BackgroundColor  dc.l  $0043021B
```

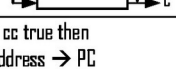
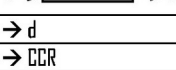
Il aura pour effet de modifier le contenu de BackgroundColor conformément au tableau ci-après. Chaque ligne du tableau correspond à un tour de boucle.

BackgroundColor	
\$0043021B	← Couleur initiale
\$0042011A	
\$00410019	
\$00400018	
:	
\$002A0002	
\$00290001	
\$00280000	
\$00270000	
:	
\$00020000	
\$00010000	
\$00000000	← Couleur noire

Remarque :

Le temps d'exécution d'une itération n'est pas à prendre en compte pour l'exercice (si l'effet de fondu est trop rapide, il sera facile de le ralentir).

Attention ! le sous-programme FadeOut est limité à 8 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 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	-	-	-	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 B)
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	-	-	-	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	-	-	-	-	-	-	-	-	-	-	-	s		Arithmetic shift Dy #n bits L/R (#n: 1 to B)
Bcc	BW ³	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	-	if cc true then address \rightarrow PC	Branch conditionally (cc table on back) (B or 16-bit \pm offset to address)
BCHG	B L	Dn,d #n,d	---*--	e ^l	-	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 ^l	-	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 (B or 16-bit \pm offset to addr)
BSET	B L	Dn,d #n,d	---*--	e ^l	-	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 (B or 16-bit \pm offset)
BTST	B L	Dn,d #n,d	---*--	e ^l	-	d	d	d	d	d	d	d	d	d	d	s	$\text{NOT}(\text{bit } Dn \text{ of } d) \rightarrow Z$ $\text{NOT}(\text{bit } \#n \text{ of } d) \rightarrow Z$	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	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	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	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 $Dn < -1$ 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	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	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	-	-	-	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 \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	d	-	$\uparrow d \rightarrow PC$	Jump to effective address of destination
JSR		d	-----	-	-	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	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	-	-	-	-	-	-	-	-	-	-	-	s		Logical shift Dy, #n bits L/R (#n: 1 to B)
MOVE ⁴	BWL	s,d	-**00	e	s ⁴	e	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	$s \rightarrow CCR$	Move source to Condition Code Register
MOVE	W	s,SR	=====	s	-	s	s	s	s	s	s	s	s	s	s	s	$s \rightarrow SR$	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)
	BWL	s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(i,An)	(i,An,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n		An \rightarrow USP	Move An to User Stack Pointer (Privileged)

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 20498, -(A2)		
MOVE.W -6(A1), -18(A2,D0.W)		
MOVE.B 5(A2), \$14(A0,D2.L)		

Exercice 2

Opération	Taille (bits)	Nombre manquant (hexadécimal)	N	Z	V	C
\$70 + \$?	8		1	0	1	0
\$70000000 + \$?	32		1	0	0	0

Exercice 3

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

Exercice 4

Decrement

Darker

FadeOut