

Contrôle S4 – Corrigé

Architecture des ordinateurs

Durée : 1 h 30

Répondre exclusivement sur le document réponse.

Exercice 1 (4 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 = \$FFFF0015 A0 = \$00005000 PC = \$00006000
 D1 = \$12340004 A1 = \$00005008
 D2 = \$FFFFFFE1 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 (3 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 #85A51000,d7
next1	moveq.l #1,d1 cmpl.w #80,d7 blt next2 moveq.l #2,d1
next2	move.l d7,d2 ror.l #4,d2 swap d2 rol.w #8,d2 rol.b #4,d2
next3	clr.l d3 move.l d7,d0
loop3	addq.l #1,d3 subq.w #2,d0 bne loop3
next4	clr.l d4 move.l d7,d0
loop4	addq.l #1,d4 dbra d0,loop4 ; DBRA = DBF

Exercice 4 (9 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. Attention ! Tous les sous-programmes sont limités à 15 lignes d'instructions au maximum.**

Structure d'un bitmap :

Champ	Taille (bits)	Codage	Description
WIDTH	16	Entier non signé	Largeur du bitmap en pixels
HEIGHT	16	Entier non signé	Hauteur du bitmap en pixels
MATRIX	Variable	Bitmap	Matrice de points du bitmap. Si un bit est à 0, le pixel affiché est noir. Si un bit est à 1, le pixel affiché est blanc.

Structure d'un sprite :

Champ	Taille (bits)	Codage	Description
STATE	16	Entier non signé	État d'affichage du sprite. Seulement deux valeurs possibles : HIDE = 0 ou SHOW = 1
X	16	Entier signé	Abscisse du sprite
Y	16	Entier signé	Ordonnée du sprite
BITMAP1	32	Entier non signé	Adresse du premier bitmap
BITMAP2	32	Entier non signé	Adresse du second bitmap

On suppose que la taille du bitmap 1 est toujours égale à celle du bitmap 2.

Constantes déjà définies :

VIDEO_START	equ	\$ffb500	; Adresse de départ de la mémoire vidéo
VIDEO_SIZE	equ	(480*320/8)	; Taille en octets de la mémoire vidéo
WIDTH	equ	0	
HEIGHT	equ	2	
MATRIX	equ	4	
STATE	equ	0	
X	equ	2	
Y	equ	4	
BITMAP1	equ	6	
BITMAP2	equ	10	
HIDE	equ	0	
SHOW	equ	1	

1. Réalisez le sous-programme **FillScreen** qui remplit la mémoire vidéo d'une valeur numérique. Le remplissage se fera par mot de 32 bits.

Entrée : **D0.L** = Valeur numérique sur 32 bits avec laquelle sera remplie la mémoire vidéo.

2. Réalisez le sous-programme **GetRectangle** qui renvoie les coordonnées du rectangle qui délimite un sprite.

Entrée : **A0.L** = Adresse d'un sprite.

Sorties : **D1.W** = Abscisse du point supérieur gauche du sprite.

D2.W = Ordonnée du point supérieur gauche du sprite.

D3.W = Abscisse du point inférieur droit du sprite.

D4.W = Ordonnée du point inférieur droit du sprite.

3. Réalisez le sous-programme **MoveSprite** qui déplace un sprite. Le déplacement se fera de façon relative. Si la nouvelle position du sprite fait sortir le sprite de l'écran, alors la position du sprite restera inchangée (la nouvelle position ne sera pas prise en compte).

Entrées : **A1.L** = Adresse du sprite.

D1.W = Mouvement relatif horizontal en pixels (entier signé sur 16 bits).

D2.W = Mouvement relatif vertical en pixels (entier signé sur 16 bits).

Sorties : **D0.L** renvoie *false* (0) si le sprite n'a pas été déplacé (car cela le faisait sortir de l'écran).

D0.L renvoie *true* (1) si le sprite a été déplacé.

Pour savoir si un sprite sort de l'écran, vous pouvez effectuer un appel au sous-programme **IsOutOfScreen**. On supposera que ce sous-programme existe déjà (vous n'avez pas besoin de l'écrire).

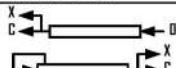
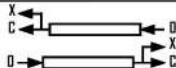
Entrées : **A0.L** = Adresse du bitmap.

D1.W = Abscisse en pixels du bitmap (entier signé sur 16 bits).

D2.W = Ordonnée en pixels du bitmap (entier signé sur 16 bits).

Sorties : **Z** renvoie *false* (0) si le bitmap ne sort pas de l'écran.

Z renvoie *true* (1) si le bitmap sort de l'écran.

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	-	-	-	-	-	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$ $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	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 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	-	-	-	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	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 \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	-	-	-	-	NOT(bit number of d) \rightarrow Z NOT(bit n of d) \rightarrow bit n 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	-	-	-	-	NOT(bit number of d) \rightarrow Z 0 \rightarrow 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	-	-	-	-	NOT(bit n of d) \rightarrow Z 1 \rightarrow bit n 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	-	-	-	-	NOT(bit Dn of d) \rightarrow Z NOT(bit #n 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-bit remainder, 16-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-bit remainder, 16-bit quotient]
EOR ⁴	BWL	Dn,d	-**00	e	-	d	d	d	d	d	d	d	-	-	-	s ⁴	Dn 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 XOR d \rightarrow d	Logical exclusive OR #n to destination
EORI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	-	s	#n XOR CCR \rightarrow CCR	Logical exclusive OR #n to CCR
EORI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	-	s	#n 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 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	BWL	#n,Dy	***0*	d	-	-	-	-	-	-	-	-	-	-	-	s	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 \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	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			

Architecture des ordinateurs – EPITA – S4 – 2021/2022

Opcode	Size	Operand	CC	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				
MOVEA ⁴	WL	s,An	-----	s e s s s s s s s s s s s s	s → An	Move source to An (MOVE s,An use MOVEA)	
MOVE ⁴	WL	Rn-Rn,d s,Rn-Rn	-----	- - d - d d d 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 - - - - 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 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 s s s	16bit s * 16bit Dn → Dn	Multiply unsg'd 16-bit; result: unsg'd 32-bit	
NBCD	B	d	*U*U*	d - d d 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 - - - -	D - d → d	Negate destination (2's complement)	
NEGX	BWL	d	*****	d - d 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 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 s s s ⁴ e - d d d d d d d d d d - - - -	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 d d d - - - -	#n OR d → d	Logical OR #n to destination	
ORI ⁴	B	#n,CCR	=====	- - - - - - - - - - - - - - - -	#n OR CCR → CCR	Logical OR #n to CCR	
ORI ⁴	W	#n,SR	=====	- - - - - - - - - - - - - - - -	#n OR SR → SR	Logical OR #n to SR (Privileged)	
PEA	L	s	-----	- - 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	W	#n,Dy d		d - - d d d d d d d d d d - - - -		Rotate Dy, #n bits left/right (#n: 1 to 8)	
ROXL	BWL	Dx,Dy	***0*	e - - - - - - - - - - - - - - - -		Rotate Dy, Dx bits L/R, X used then updated	
ROXR	W	#n,Dy d		d - - d d d d d d d d d d - - - -		Rotate Dy, #n bits left/right (#n: 1 to 8)	
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 d d d - - - -	If cc is true then f's → d else D's → d	If cc true then d.B = 11111111 else d.B = 00000000	
STOP		#n	=====	- - - - - - - - - - - - - - - -	#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 s s ⁴ e d ⁴ d d d d d d d d d d - - - -	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 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 d d d - - - -	d - #n → d	Subtract immediate from destination	
SUBQ ⁴	BWL	#n,d	*****	d d d d d d d d d d d d - - - -	d - #n → d	Subtract quick immediate (#n range: 1 to 8)	
SUBX	BWL	Dy,Dx -(Ay),-(Ax)	*****	e - - - - - 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 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	-----	- - - - - - - - - - - - - - - -	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 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 (← DR, 1 NOT, ⊕ XOR; * Unsigned, # Alternate cc)					
cc	Condition	Test	cc	Condition	Test
T	true	I	VC	overflow clear	!V
F	false	O	VS	overflow set	V
HI [#]	higher than	I(C + Z)	PL	plus	IN
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 #4507, -(A1)	\$005000 54 AF 18 B9 00 00 11 9B	A1 = \$00005004
MOVE.B \$5009, -6(A1)	\$005000 54 AF 10 B9 E7 21 48 C0	Aucun changement
MOVE.W 8(A1), -37(A2, D0.W)	\$005000 13 79 18 B9 E7 21 48 C0	Aucun changement
MOVE.L -4(A2), \$21(A0, D2.L)	\$005000 54 AF D4 36 1F 88 48 C0	Aucun changement

Exercice 2

Opération	Taille (bits)	Nombre manquant (hexadécimal)	N	Z	V	C
\$80 + \$?	8	\$00	1	0	0	0
\$8000 + \$?	16	\$8000	0	1	1	1
\$80000000 + \$?	32	\$80000001	0	0	1	1

Exercice 3

Valeurs des registres après exécution du programme. Utilisez la représentation hexadécimale sur 32 bits.	
D1 = \$00000002	D3 = \$00000800
D2 = \$51005A80	D4 = \$00001001

Exercice 4

```

FillScreen      movem.l  d7/a0, -(a7)

                lea     VIDEO_START, a0
                move.w  #VIDEO_SIZE/4-1, d7

\loop          move.l  d0, (a0)+
                dbra   d7, \loop

                movem.l (a7)+, d7/a0
                rts

```

```

GetRectangle    move.l  a0, -(a7)

                move.w  X(a0), d1
                move.w  Y(a0), d2

                movea.l BITMAP1(a0), a0

                move.w  WIDTH(a0), d3
                add.w   d1, d3
                subq.w  #1, d3

                move.w  HEIGHT(a0), d4
                add.w   d2, d4
                subq.w  #1, d4

                movea.l (a7)+, a0
                rts

```

```

MoveSprite     movem.l  d1/d2/a0, -(a7)

                add.w   X(a1), d1
                add.w   Y(a1), d2

                movea.l BITMAP1(a1), a0

                jsr     IsOutOfScreen
                beq     \false

                move.w  d1, X(a1)
                move.w  d2, Y(a1)

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

\false        moveq.l  #0, d0
\quit        movem.l  (a7)+, d1/d2/a0
                rts

```