

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 = \$0007003D A0 = \$00005000 PC = \$00006000
 D1 = \$12340004 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  #-218,d7
next1     moveq.l #1,d1
          cmpi.b  #$30,d7
          blo     next2
          moveq.l #2,d1
next2     move.l  d7,d2
          lsr.l  #8,d2
          ror.w  #4,d2
          lsl.l  #8,d2
next3     clr.l   d3
          move.l #$FFFFFFF,d0
loop3     addq.l  #1,d3
          subq.b #1,d0
          bne   loop3
next4     clr.l   d4
          move.b  #$20,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 couleur possède trois composantes (les composantes rouge, verte et bleue). Une couleur est encodée dans un mot de 32 bits de la façon suivante : $00RRGGBB_{16}$

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

Par exemple :

- Si l'encodage d'une couleur est $002B048D_{16}$, alors sa composante rouge est $2B_{16}$, sa composante verte est 04_{16} et sa composante bleue est $8D_{16}$;
- La couleur noire est encodée 00000000_{16} ;
- La couleur blanche est encodée $00FFFFFF_{16}$.

1. Réalisez le sous-programme **SplitColor** qui renvoie les trois composantes d'une couleur.

Entrée : **D0.L** contient une couleur codée sur 32 bits ($00RRGGBB_{16}$).

Sorties : **D1.B** = Composante rouge de la couleur (RR).

D2.B = Composante verte de la couleur (GG).

D3.B = Composante bleue de la couleur (BB).

Le sous-programme SplitColor est limité à 10 lignes d'instructions (RTS compris).

2. À l'aide du sous-programme **SplitColor**, réalisez le sous-programme **IsGray** qui détermine si une couleur correspond à un niveau de gris. C'est-à-dire si ses trois composantes sont identiques.

Entrée : **D0.L** contient une couleur codée sur 32 bits ($00RRGGBB_{16}$).

Sortie : **D0.L** = 0 (faux) si la couleur ne correspond pas à un niveau de gris.

D0.L = 1 (vrai) si la couleur correspond à un niveau de gris.

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

3. À l'aide du sous-programme **IsGray**, réalisez le sous-programme **IsAllGray** qui détermine si toutes les couleurs d'un tableau correspondent à un niveau de gris. Chaque élément du tableau est une couleur codée sur 32 bits ($00RRGGBB_{16}$).

Entrées : **A0.L** pointe sur le premier élément d'un tableau de couleurs.

D0.L contient le nombre d'éléments du tableau.

Sortie : **D0.L** = 0 (faux) si au moins une couleur du tableau ne correspond pas à un niveau de gris.

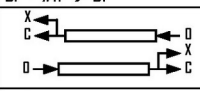
D0.L = 1 (vrai) si toutes les couleurs du tableau correspondent à un niveau de gris.

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

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	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 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 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	$1 \rightarrow \text{bit } n \text{ of } d$	Arithmetic shift Dy #n bits L/R (#n: 1 to B) 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) (B 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 (B 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 (B or 16-bit \pm offset)	
BTST	B L	Dn,d #n,d	---*--	e ¹	-	d	d	d	d	d	d	d	-	-	-	$\text{NOT}(\text{bit } n \text{ of } d) \rightarrow Z$ $\text{NOT}(\text{bit } \#n \text{ of } d) \rightarrow Z$	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 $Dn < -1$ then addr \rightarrow PC }	Test condition, decrement and branch (16-bit \pm offset to address)	
DIVS	W	s,Dn	---*00	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	---*00	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	d	-	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	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR } CCR \rightarrow CCR$	Logical exclusive OR #n to CCR	
EDRI 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	-	$\uparrow d \rightarrow$ 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	BWL	#n,Dy d		d	-	-	-	-	-	-	-	-	-	-	s	$0 \rightarrow \text{bit } n \text{ of } d$	Logical shift Dy, #n bits L/R (#n: 1 to B) 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	↑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	BWL	#n,Dy	-**0*	d	-	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, #n bits left/right (#n: 1 to 8)
RDR	W	d	-**0*	-	-	d	d	d	d	d	d	d	-	-	-	-		Rotate d l-bit left/right (W only)
RDXL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits L/R, X used then updated
RDXR	BWL	#n,Dy	***0*	d	-	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, #n bits left/right (#n: 1 to 8)
RDXR	W	d	***0*	-	-	d	d	d	d	d	d	d	-	-	-	-		Rotate destination l-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	-	-	-	-	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 (+ DR, ! NOT, ⊕ XOR: ^u Unsigned, ^a Alternate cc)					
cc	Condition	Test	cc	Condition	Test
T	true	I	VC	overflow clear	IV
F	false	O	VS	overflow set	V
HI ^a	higher than	I(C + Z)	PL	plus	!N
LS ^a	lower or same	C + Z	MI	minus	N
HS ^a , CC ^a	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LD ^a , 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 Cszasz, 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.W #0560, -6(A1)	\$005000 54 AF 02 30 E7 21 48 C0	Aucun changement
MOVE.B -(A1), -57(A2, D0.W)	\$005010 13 79 01 80 C0 1A 2D 49	A1 = \$00005007
MOVE.B -1(A1), \$4(A1, D2.L)	\$005008 C0 10 11 C8 D4 36 1F 88	Aucun changement

Exercice 2

Opération	Taille (bits)	Nombre manquant (hexadécimal)	N	Z	V	C
\$4570 + \$?	16	\$3A90	1	0	1	0
\$F431C16A + \$?	32	\$0BCE3E96	0	1	0	1

Exercice 3

Valeurs des registres après exécution du programme. Utilisez la représentation hexadécimale sur 32 bits.	
D1 = \$00000001	D3 = \$000000FF
D2 = \$FFFFFF00	D4 = \$0000FF21

Exercice 4

```
SplitColor    move.b  d0,d3
              ror.l  #8,d0
              move.b d0,d2
              ror.l  #8,d0
              move.b d0,d1
              swap   d0
              rts
```

```
IsGray       movem.l d1-d3,-(a7)
              jsr    SplitColor
              cmp.b  d1,d2
              bne   \false
              cmp.b  d1,d3
              bne   \false
\true        moveq.l #1,d0
              bra   \quit
\false       moveq.l #0,d0
\quit        movem.l (a7)+,d1-d3
              rts
```

```
IsAllGray    movem.l d7/a0,-(a7)
              move.l d0,d7
\loop        move.l (a0)+,d0
              jsr   IsGray
              tst.l d0
              beq   \quit
              subq.l #1,d7
              bne   \loop
\quit        movem.l (a7)+,d7/a0
              rts
```