

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 = \$FFFF0011 A0 = \$00005000 PC = \$00006000
 D1 = \$00000004 A1 = \$00005008
 D2 = \$FFFFFFF1 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)

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 (4 points)

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

Main	<code>move.w #-256,d7</code>
next1	<code>moveq.l #1,d1 tst.b d7 bpl next2 moveq.l #2,d1</code>
next2	<code>moveq.l #1,d2 cmp.w #-5,d7 ble next3 moveq.l #2,d2</code>
next3	<code>clr.l d3 move.w #25A,d0</code>
loop3	<code>addq.l #1,d3 subq.b #1,d0 bne loop3</code>
next4	<code>clr.l d4 move.w #3,d0</code>
loop4	<code>addq.l #1,d4 dbra d0,loop4 ; DBRA = DBF</code>
quit	<code>illegal</code>

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. Une chaîne de caractères se termine toujours par un caractère nul (la valeur zéro). Pour tout l'exercice, on suppose que les chaînes ne sont jamais vides (elles possèdent au moins un caractère non nul) et qu'elles contiennent uniquement des chiffres ou des minuscules sans accent.

1. Réalisez le sous-programme **StrRev** qui inverse une chaîne de caractères.

Entrées : **A0.L** pointe sur une chaîne de caractères à inverser (chaîne source).

A1.L pointe sur un emplacement mémoire où écrire la chaîne inversée (chaîne destination).

Sortie : La chaîne destination contient la chaîne source inversée.

(La chaîne source n'est pas modifiée.)

Par exemple :

- Si chaîne source = « hello »
- Alors chaîne destination = « olleh »

2. Réalisez le sous-programme **IsPal** qui détermine si une chaîne de caractères est un palindrome. Nous dirons qu'une chaîne est un palindrome si elle contient des caractères qui peuvent être lus indifféremment de droite à gauche ou de gauche à droite. Ce sous-programme ne doit pas appeler **StrRev**.

Entrée : **A0.L** pointe sur une chaîne à tester.

Sortie : **D0.L** renvoie 1 (*true*) si la chaîne un palindrome.

D0.L renvoie 0 (*false*) si la chaîne n'est pas un palindrome.

Par exemple :

- « a », « kayak », « radar », « 36544563 » sont des palindromes.
- « ab », « hello », « 123 » ne sont pas des palindromes.

3. À l'aide des sous-programmes **StrRev** et **IsPal**, réalisez le sous-programme **RevIfNotPal** qui inverse une chaîne de caractères si cette dernière n'est pas un palindrome.

Entrées : **A0.L** pointe sur une chaîne de caractères à inverser (chaîne source).

A1.L pointe sur un emplacement où écrire la chaîne inversée (chaîne destination).

Sorties : Si la chaîne source n'est pas un palindrome :

La chaîne destination contient la chaîne source inversée.

D0.L renvoie 0.

Si la chaîne source est un palindrome :

La chaîne destination n'est pas modifiée (**A1.L** est ignoré).

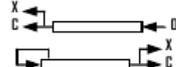
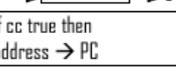
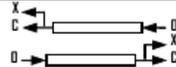
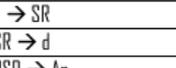
D0.L renvoie 1.

(La chaîne source n'est jamais modifiée.)

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		
				Dn	An	(An)	(An)+	-(An)	(iAn)	(iAn,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 → Dx ₁₀ -(Ay) ₁₀ + -(Ax) ₁₀ + X → -(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 → Dn Dn + d → 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 → An	Add address (.W sign-extended to .L)	
ADDI ⁴	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	#n + d → d	Add immediate to destination	
ADDQ ⁴	BWL	#n,d	*****	d	d	d	d	d	d	d	d	-	-	s	#n + d → d	Add quick immediate (#n range: 1 to 8)		
ADDX	BWL	Dy,Dx -(Ay),-(Ax)	*****	e	-	-	-	-	-	-	-	-	-	-	-	Dy + Dx + X → Dx -(Ay) + -(Ax) + X → -(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 AND Dn → Dn Dn AND d → 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	-	-	s	#n AND d → d	Logical AND immediate to destination		
ANDI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	s	#n AND CCR → CCR	Logical AND immediate to CCR		
ANDI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	s	#n AND SR → 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 8)		
Bcc	BW ⁴	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 ^l	-	d	d	d	d	d	d	-	-	-	-	NOT(bit number of d) → Z NOT(bit n of d) → 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 ^l	-	d	d	d	d	d	d	-	-	-	-	NOT(bit number of d) → Z 0 → bit number of d	Set Z with state of specified bit in d then clear the bit in d	
BRA	BW ⁴	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	address → PC	Branch always (8 or 16-bit ± offset to addr)	
BSET	B L	Dn,d #n,d	---*---	e ^l	-	d	d	d	d	d	d	-	-	-	-	NOT(bit n of d) → Z 1 → bit n of d	Set Z with state of specified bit in d then set the bit in d	
BSR	BW ⁴	address ²	-----	-	-	-	-	-	-	-	-	-	-	-	-	PC → -(SP); address → PC	Branch to subroutine (8 or 16-bit ± offset)	
BTST	B L	Dn,d #n,d	---*---	e ^l	-	d	d	d	d	d	d	d	d	d	s	NOT(bit Dn of d) → Z NOT(bit #n of d) → 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	-	-	-	-	0 → 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	-	-	s	set CCR with d - #n	Compare destination to #n		
CMPP 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 → Dn if Dn < -1 then addr → PC }	Test condition, decrement and branch (16-bit ± offset to address)	
DIVS	W	s,Dn	---*000	e	-	s	s	s	s	s	s	s	s	s	s	±32bit Dn / ±16bit s → ±Dn	Dn = [16-bit remainder, 16-bit quotient]	
DIVU	W	s,Dn	---*000	e	-	s	s	s	s	s	s	s	s	s	s	32bit Dn / 16bit s → Dn	Dn = [16-bit remainder, 16-bit quotient]	
EOR ⁴	BWL	Dn,d	---*000	e	-	d	d	d	d	d	d	-	-	s ⁴	Dn XOR d → d	Logical exclusive OR Dn to destination		
EORI ⁴	BWL	#n,d	---*000	d	-	d	d	d	d	d	d	-	-	s	#n XOR d → d	Logical exclusive OR #n to destination		
EORI ⁴	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	s	#n XOR CCR → CCR	Logical exclusive OR #n to CCR		
EORI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	s	#n XOR SR → SR	Logical exclusive OR #n to SR (Privileged)		
EXG	WL	Rx,Ry	-----	e	e	-	-	-	-	-	-	-	-	-	-	register ↔ register	Exchange registers (32-bit only)	
EXT	WL	Dn	---*000	d	-	-	-	-	-	-	-	-	-	-	-	Dn.B → Dn.W Dn.W → 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 → PC	Jump to effective address of destination	
JSR		d	-----	-	-	d	-	-	d	d	d	d	d	-	-	PC → -(SP); ↑d → PC	push PC, jump to subroutine at address d	
LEA	L	s,An	-----	-	e	s	-	-	s	s	s	s	s	-	-	↑s → An	Load effective address of s to An	
LINK		An,#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	An → -(SP); SP → An; SP + #n → 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 8)		
		d		-	-	d	d	d	d	d	d	-	-	-	-		Logical shift d 1 bit left/right (.W only)	
MOVE ⁴	BWL	s,d	---*000	e	s ⁴	e	e	e	e	e	e	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 → CCR	Move source to Condition Code Register	
MOVE	W	s,SR	=====	s	-	s	s	s	s	s	s	s	s	s	s	s → SR	Move source to Status Register (Privileged)	
MOVE	W	SR,d	-----	d	-	d	d	d	d	d	d	-	-	s	-	SR → d	Move Status Register to destination	
MOVE	L	USP,An	-----	-	d	-	-	-	-	-	-	-	-	-	-	USP → An	Move User Stack Pointer to An (Privileged)	
		An,USP	-----	-	s	-	-	-	-	-	-	-	-	-	-	An → USP	Move An to User Stack Pointer (Privileged)	
	BWL	s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(iAn)	(iAn,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 ³	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	-	-	-	-	-	-	-	-	-	-	-	-	#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	-	-	-	-	0 - d ₁₀ - X → d	Negate BCD with eXtend, BCD result
NEG	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	-	0 - d → d	Negate destination (2's complement)
NEGX	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	-	0 - 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 → -(SP)	Push effective address of s onto stack
RESET			-----	-	-	-	-	-	-	-	-	-	-	-	-	-	Assert RESET Line	Issue a hardware RESET (Privileged)
ROL	BWL	Dx,Dy	-***0*	e	-	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits left/right (without X)
ROR	W	#n,Dy		d	-	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, #n bits left/right (#n: 1 to 8)
	W	d		-	-	d	d	d	d	d	d	d	-	-	-	-		Rotate d 1-bit left/right (.W only)
ROXL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits L/R, X used then updated
ROXR	W	#n,Dy		d	-	-	-	-	-	-	-	-	-	-	-	s		Rotate Dy, #n bits left/right (#n: 1 to 8)
	W	d		-	-	d	d	d	d	d	d	d	-	-	-	-		Rotate destination 1-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	-	-	-	-	-	-	-	-	-	-	-	-	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)	(i,An)	(i,An,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n			

Condition Tests (+ OR, ! NOT, ⊕ XOR; ° Unsigned, ° Alternate cc)					
cc	Condition	Test	cc	Condition	Test
T	true	I	VC	overflow clear	IV
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	IC	GE	greater or equal	!(N ⊕ V)
LO [°] , CS [°]	lower than	C	LT	less than	(N ⊕ V)
NE	not equal	IZ	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 \$5006, (A2)+	\$005010 48 C0 C9 10 42 1A 2D 49	A2 = \$00005014
MOVE.L #50, 10(A1)	\$005010 13 79 00 00 00 32 2D 49	Aucun changement
MOVE.B 12(A1, D2.L), 7(A2)	\$005010 13 79 01 80 42 1A 2D 21	Aucun changement
MOVE.L -2(A1), -17(A2, D0.W)	\$005010 48 C0 C9 10 42 1A 2D 49	Aucun changement

Exercice 2

Opération	Taille (bits)	Résultat (hexadécimal)	N	Z	V	C
\$7F + \$7F	8	\$FE	1	0	1	0
\$7F + \$80	8	\$FF	1	0	0	0
\$7F + \$81	8	\$00	0	1	0	1

Exercice 3

Valeurs des registres après exécution du programme. Utilisez la représentation hexadécimale sur 32 bits.	
D1 = \$00000001	D3 = \$0000005A
D2 = \$00000001	D4 = \$00000004

Exercice 4

```

StrRev          movem.l  a1/a2, -(a7)

                movea.l  a0, a2
\loop1          tst.b    (a2)+
                bne     \loop1
                subq.l  #1, a2

\loop2          move.b   -(a2), (a1)+
                cmpa.l  a0, a2
                bne     \loop2

                clr.b   (a1)

                movem.l (a7)+, a1/a2
                rts

```

```

IsPal           movem.l  a0/a1, -(a7)

                movea.l  a0, a1
\loop1          tst.b    (a1)+
                bne     \loop1
                subq.l  #1, a1

\loop2          move.b   (a0)+, d0
                cmp.b   -(a1), d0
                bne     \false

                cmpa.l  a0, a1
                bhi     \loop2

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

\false          moveq.l  #0, d0

\quit          movem.l  (a7)+, a0/a1
                rts

```

```

RevIfNotPal     jsr      IsPal
                tst.l   d0
                bne     \quit

                jsr      StrRev

\quit          rts

```