

# Key to Final Exam S3

## Computer Architecture

Duration: 1 hr 30 min

Write answers only on the answer sheet.

**Exercise 1 (4 points)**

Complete the table shown on the [answer sheet](#). Write down the new values of the registers (except the PC) and memory that are modified by the instructions. **Use the hexadecimal representation. Memory and registers are reset to their initial values for each instruction.**

Initial values: D0 = \$12340007 A0 = \$00005000 PC = \$00006000  
 D1 = \$FFFFFFEF A1 = \$00005008  
 D2 = \$0000FFFD 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

**Exercise 2 (3 points)**

Complete the table shown on the [answer sheet](#). Determine the missing number for each addition in order to match the given flags (use the hexadecimal representation). **If multiple answers are possible, choose the smallest one.**

**Exercise 3 (4 points)**

Let us consider the following program. Complete the table shown on the [answer sheet](#).

```

Main      move.l  #$8421,d7
next1     moveq.l #1,d1
          cmpi.l  #$525,d7
          bgt     next2
          moveq.l #2,d1
next2     clr.l   d2
          move.l  #$11112222,d0
loop2     addq.l  #1,d2
          subq.b  #1,d0
          bne     loop2
next3     clr.l   d3
          move.b  #$05,d0
loop3     addq.l  #1,d3
          dbra    d0,loop3      ; DBRA = DBF
next4     clr.l   d4
          move.w  #10,d0
loop4     addq.l  #1,d4
          dbra    d0,loop4      ; DBRA = DBF
  
```

**Exercise 4 (9 points)**

In this exercise, you should write three subroutines that copy some bytes from a memory location to another memory location. **None of the data and address registers should be modified when the subroutine returns.** Each of the subroutines has the following inputs:

**Inputs:** **A1.L** points to the source memory location.  
**A2.L** points to the destination memory location.  
**D0.L** holds the number of bytes to copy (unsigned integer).

**Each subroutine can be written independently.**

- Write the **CopyInc** subroutine that copies data by starting with the first byte and that increments the addresses (see the [example below](#)). We assume that when **CopyInc** is called:
  - The **D0** register is not null.
  - The **A1** and **A2** registers are not equal.
- Write the **CopyDec** subroutine that copies data by starting with the last byte and that decrements the addresses (see [example below](#)). We assume that when **CopyDec** is called:
  - The **D0** register is not null.
  - The **A1** and **A2** registers are not equal.
- Write the **Copy** subroutine that calls **CopyInc** if the destination address is smaller than the source address or that calls **CopyDec** if the destination address is greater than the source address. We assume that when **Copy** is called:
  - The **D0** register can be null. If so, no bytes are copied.
  - The **A1** and **A2** registers can be equal. If so, no bytes are copied.

Example for <b>A1</b> = \$1000, <b>A2</b> = \$2000 and <b>D0</b> = 3.	
<b>CopyInc</b> : (\$1000) → (\$2000)	<b>CopyDec</b> : (\$1002) → (\$2002)
(\$1001) → (\$2001)	(\$1001) → (\$2001)
(\$1002) → (\$2002)	(\$1000) → (\$2000)

**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
		s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(iAn)	(iAn,Rn)	abs.W	abs.L	(iPC)	(iPC,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 <sup>4</sup>	BWL	s,Dn Dn,d	*****	e	s	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	$s + Dn \rightarrow Dn$ $Dn + d \rightarrow d$	Add binary (ADDI or ADDQ is used when source is #n. Prevent ADDQ with #n.L)
ADDA <sup>4</sup>	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 <sup>4</sup>	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	$\#n + d \rightarrow d$	Add immediate to destination
ADDQ <sup>4</sup>	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 <sup>4</sup>	BWL	s,Dn Dn,d	-**00	e	-	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	$s \text{ AND } Dn \rightarrow Dn$ $Dn \text{ AND } d \rightarrow d$	Logical AND source to destination (ANDI is used when source is #n)
ANDI <sup>4</sup>	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 <sup>4</sup>	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ AND CCR} \rightarrow \text{CCR}$	Logical AND immediate to CCR
ANDI <sup>4</sup>	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ AND SR} \rightarrow \text{SR}$	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 <sup>4</sup>	address <sup>2</sup>	-----	-	-	-	-	-	-	-	-	-	-	-	-	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 <sup>1</sup>	-	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 <sup>1</sup>	-	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 <sup>4</sup>	address <sup>2</sup>	-----	-	-	-	-	-	-	-	-	-	-	-	-	address $\rightarrow$ PC	Branch always (8 or 16-bit $\pm$ offset to addr)
BSET	B L	Dn,d #n,d	---*---	e <sup>1</sup>	-	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 <sup>4</sup>	address <sup>2</sup>	-----	-	-	-	-	-	-	-	-	-	-	-	-	PC $\rightarrow$ -(SP); address $\rightarrow$ PC	Branch to subroutine (8 or 16-bit $\pm$ offset)
BTST	B L	Dn,d #n,d	---*---	e <sup>1</sup>	-	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	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 <sup>4</sup>	BWL	s,Dn	-*****	e	s <sup>4</sup>	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	set CCR with $Dn - s$	Compare Dn to source
CMPA <sup>4</sup>	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 <sup>4</sup>	BWL	#n,d	-*****	d	-	d	d	d	d	d	d	d	-	-	s	set CCR with $d - \#n$	Compare destination to #n
CMPP <sup>4</sup>	BWL	(Ay),-(Ax)+	-*****	-	-	-	e	-	-	-	-	-	-	-	-	set CCR with $(Ax) - (Ay)$	Compare (Ax) to (Ay); Increment Ax and Ay
DBcc	W	Dn,address <sup>2</sup>	-----	-	-	-	-	-	-	-	-	-	-	-	-	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	$\pm 32\text{bit } Dn / \pm 16\text{bit } s \rightarrow \pm Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
DIVU	W	s,Dn	-***0	e	-	s	s	s	s	s	s	s	s	s	s	$32\text{bit } Dn / 16\text{bit } s \rightarrow Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
EOR <sup>4</sup>	BWL	Dn,d	-**00	e	-	d	d	d	d	d	d	d	-	-	s <sup>4</sup>	$Dn \text{ XOR } d \rightarrow d$	Logical exclusive OR Dn to destination
EORI <sup>4</sup>	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 <sup>4</sup>	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR CCR} \rightarrow \text{CCR}$	Logical exclusive OR #n to CCR
EORI <sup>4</sup>	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	s	$\#n \text{ XOR SR} \rightarrow \text{SR}$	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	-	$\uparrow d \rightarrow \text{PC}$	Jump to effective address of destination
JSR		d	-----	-	-	d	-	-	d	d	d	d	d	d	-	PC $\rightarrow$ -(SP); $\uparrow d \rightarrow \text{PC}$	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	-----	-	-	-	-	-	-	-	-	-	-	-	-	$An \rightarrow \text{-(SP)}; SP \rightarrow An;$ $SP + \#n \rightarrow SP$	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	-	-	-	-	-	-	-	-	-	-	s		Logical shift Dy, #n bits L/R (#n: 1 to 8)
MOVE <sup>4</sup>	BWL	s,d	-**00	e	s <sup>4</sup>	e	e	e	e	e	e	e	s	s	s <sup>4</sup>	$s \rightarrow d$	Move data from source to destination
MOVE	W	s,CCR	=====	s	-	s	s	s	s	s	s	s	s	s	s	$s \rightarrow \text{CCR}$	Move source to Condition Code Register
MOVE	W	s,SR	=====	s	-	s	s	s	s	s	s	s	s	s	s	$s \rightarrow \text{SR}$	Move source to Status Register (Privileged)
MOVE	W	SR,d	-----	d	-	d	d	d	d	d	d	d	-	-	-	$\text{SR} \rightarrow d$	Move Status Register to destination
MOVE	L	USP,An	-----	-	d	-	-	-	-	-	-	-	-	-	-	$\text{USP} \rightarrow An$	Move User Stack Pointer to An (Privileged)
		An,USP	-----	-	s	-	-	-	-	-	-	-	-	-	-	$An \rightarrow \text{USP}$	Move An to User Stack Pointer (Privileged)
	BWL	s,d	XNZVC	Dn	An	(An)	(An)+	-(An)	(iAn)	(iAn,Rn)	abs.W	abs.L	(iPC)	(iPC,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)	(i,An)	(i,An,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n			
MOVEA <sup>4</sup>	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 <sup>3</sup>	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 <sup>4</sup>	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 unisig'd 16-bit; result: unisig'd 32-bit
NBCD	B	d	*U*U*	d	-	d	d	d	d	d	d	d	-	-	-	-	0 - d <sub>10</sub> - 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 <sup>4</sup>	BWL	s,Dn Dn,d	-***00	e	-	s	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	s OR Dn → Dn Dn OR d → d	Logical OR (ORI is used when source is #n)
ORI <sup>4</sup>	BWL	#n,d	-***00	d	-	d	d	d	d	d	d	d	-	-	-	s	#n OR d → d	Logical OR #n to destination
ORI <sup>4</sup>	B	#n,CCR	=====	-	-	-	-	-	-	-	-	-	-	-	-	s	#n OR CCR → CCR	Logical OR #n to CCR
ORI <sup>4</sup>	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 → -(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) Rotate Dy, #n bits left/right (#n: 1 to 8) Rotate d 1-bit left/right (W only)
ROXL	BWL	Dx,Dy #n,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-	-		Rotate Dy, Dx bits L/R, X used then updated Rotate Dy, #n bits left/right (#n: 1 to 8) Rotate destination 1-bit left/right (W only)
ROXR	W	d		-	-	d	d	d	d	d	d	d	-	-	-	-		
RTE			=====	-	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → SR; (SP)+ → PC	Return from subroutine and restore CCR
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 <sub>10</sub> - Dy <sub>10</sub> - X → Dx <sub>10</sub> -(Ax) <sub>10</sub> - (Ay) <sub>10</sub> - X → -(Ax) <sub>10</sub>	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 <sup>4</sup>	BWL	s,Dn Dn,d	*****	e	s	s	s	s	s	s	s	s	s	s	s	s <sup>4</sup>	Dn - s → Dn d - Dn → d	Subtract binary (SUBI or SUBQ used when source is #n. Prevent SUBQ with #n.L)
SUBA <sup>4</sup>	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 <sup>4</sup>	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	-	s	d - #n → d	Subtract immediate from destination
SUBQ <sup>4</sup>	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 <sup>u</sup>	higher than	I(C + Z)	PL	plus	IN
LS <sup>u</sup>	lower or same	C + Z	MI	minus	N
HS <sup>u</sup> , CC <sup>a</sup>	higher or same	IC	GE	greater or equal	!(N ⊕ V)
LO <sup>u</sup> , CS <sup>a</sup>	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

Revised by Peter Csaszar, Lawrence Tech University – 2004-2006

Distributed under the GNU general public use license.

**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  
<sup>1</sup> Long only; all others are byte only  
<sup>2</sup> Assembler calculates offset  
<sup>3</sup> Branch sizes: **.B** or **.S** -128 to +127 bytes, **.W** or **.L** -32768 to +32767 bytes  
<sup>4</sup> 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

Last name: ..... First name: ..... Group: .....

**ANSWER SHEET TO BE HANDED IN****Exercise 1**

Instruction	Memory	Register
Example	\$005000 54 AF <b>00 40</b> E7 21 48 C0	A0 = \$00005004 A1 = \$0000500C
Example	\$005008 C9 10 11 C8 D4 36 <b>FF</b> 88	No change
MOVE.W #18, -6(A2)	\$005008 C9 10 <b>00 12</b> D4 36 1F 88	No change
MOVE.W \$5004, 3(A0, D0.W)	\$005008 C9 10 <b>E7 21</b> D4 36 1F 88	No change
MOVE.B 5(A1), \$18(A1, D1.L)	\$005008 C9 10 11 C8 D4 36 1F <b>36</b>	No change
MOVE.L -\$8(A1), -1(A2, D2.W)	\$005008 C9 10 11 C8 <b>54 AF 18 B9</b>	No change

**Exercise 2**

Operation	Size (bits)	Missing Number (hexadecimal)	N	Z	V	C
\$7F + \$?	8	<b>\$01</b>	1	0	1	0
\$98BD + \$?	16	<b>\$6743</b>	0	1	0	1
\$98BD + \$?	32	<b>\$80000000</b>	1	0	0	0

**Exercise 3**

Values of registers after the execution of the program. Use the 32-bit hexadecimal representation.	
<b>D1</b> = \$00000001	<b>D3</b> = \$00002206
<b>D2</b> = \$00000022	<b>D4</b> = \$0000000B

**Exercise 4**

```

CopyInc      movem.l  d0/a1/a2,-(a7)

\loop        move.b   (a1)+,(a2)+
              subq.l   #1,d0
              bne      \loop

              movem.l  (a7)+,d0/a1/a2
              rts

```

```

CopyDec      move.l   d0,-(a7)

              adda.l   d0,a1
              adda.l   d0,a2

\loop        move.b   -(a1),-(a2)
              subq.l   #1,d0
              bne      \loop

              move.l   (a7)+,d0
              rts

```

```

Copy         tst.l     d0
              beq       \quit

              cmpa.l    a1,a2
              beq       \quit
              blo       \inc

\dec         jsr        CopyDec
              rts

\inc
\quit        jsr        CopyInc
              rts

```