

Final Exam S3

Computer Architecture

Duration: 1 hr 30 min.

Exercise 1 (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 A1 = \$1000, A2 = \$2000 and D0 = 3.	
CopyInc : (\$1000) → (\$2000)	CopyDec : (\$1002) → (\$2002)
(\$1001) → (\$2001)	(\$1001) → (\$2001)
(\$1002) → (\$2002)	(\$1000) → (\$2000)

Exercise 2 (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 = \$0004FFFF A0 = \$00005000 PC = \$00006000
 D1 = \$0001000A A1 = \$00005008
 D2 = \$FFFFFFFD 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 3 (3 points)

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

1. 8-bit addition: \$7F + \$? with N = 1, Z = 0, V = 1, C = 0
2. 16-bit addition: \$98BD + \$? with N = 0, Z = 1, V = 0, C = 1
3. 32-bit addition: \$98BD + \$? with N = 1, Z = 0, V = 0, C = 0

Exercise 4 (4 points)

Let us consider the four following programs:

```
Prog1      tst.b   d5
           beq     quit1
           moveq.l #2,d1
quit1
```

```
Prog2      tst.w   d5
           bpl     quit2
           moveq.l #2,d2
quit2
```

```
Prog3      move.w  #100,d7
loop3      addq.l  #1,d3
           dbra    d7,loop3 ; DBRA = DBF (DBcc with cc = F)
```

```
Prog4      move.l  #1000,d0
loop4      addq.l  #1,d4
           addi.l  #10,d0
           cmpi.l  #2000,d0
           bne     loop4
```

- Each program is independent.
- The initial values are identical for each program.
- Initial values:
 - D1** = \$00000001
 - D2** = \$00000001
 - D3** = \$00000000
 - D4** = \$00000000
 - D5** = \$0067A200

Answer on the answer sheet.

1. What will the value of **D1** be after the execution of **Prog1** ?
2. What will the value of **D2** be after the execution of **Prog2** ?
3. What will the value of **D3** be after the execution of **Prog3** ?
4. What will the value of **D4** be after the execution of **Prog4** ?

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)	(i,An)	(i,An,Rn)	abs,W	abs,L	(i,PC)	(i,PC,Rn)	#n		
ABCD	B	Dy,Dx -(Ay),-(Ax)	*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	BWL	s,Dn Dn,d	*****	e	s	s	s	s	s	s	s	s	s	s	s	$s + Dn \rightarrow Dn$ $Dn + d \rightarrow d$	Add binary (ADD) or ADDQ is used when source is #n. Prevent ADDQ with #n.L
ADDA	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	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	$\#n + d \rightarrow d$	Add immediate to destination
ADDQ	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	BWL	s,Dn Dn,d	---*00	e	-	s	s	s	s	s	s	s	s	s	s	$s \text{ AND } Dn \rightarrow Dn$ $Dn \text{ AND } d \rightarrow d$	Logical AND source to destination (ANDI is used when source is #n)
ANDI	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	B	#n,CCR	---*00	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ AND CCR} \rightarrow \text{CCR}$	Logical AND immediate to CCR
ANDI	W	#n,SR	---*00	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ AND SR} \rightarrow \text{SR}$	Logical AND immediate to SR (Privileged)
ASL	BWL	Dx,Dy #n,Dy	*****	d	-	-	-	-	-	-	-	-	-	-	s		Arithmetic shift Dy by Dx bits left/right
ASR	W	d	-----	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)
BCNG	B	L Dn,d #n,d	---*--	e ¹ d 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 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 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)
BIST	B	L Dn,d #n,d	---*--	e ¹ d ¹	-	d	d	d	d	d	d	d	d	d	d	NOT(bit n 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	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	BWL	s,Dn	---***	e	s	s	s	s	s	s	s	s	s	s	s	set CCR with $Dn - s$	Compare Dn to source
CMPA	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	BWL	#n,d	---***	d	-	d	d	d	d	d	d	d	-	-	-	set CCR with $d - \#n$	Compare destination to #n
CMPI	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	$32\text{bit } Dn / 16\text{bit } s \rightarrow Dn$	$Dn = (16\text{-bit remainder}, 16\text{-bit quotient})$
EOR	BWL	Dn,d	---*00	e	-	d	d	d	d	d	d	d	-	-	s	$Dn \text{ XOR } d \rightarrow d$	Logical exclusive OR Dn to destination
EORI	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	B	#n,CCR	---*00	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ XOR CCR} \rightarrow \text{CCR}$	Logical exclusive OR #n to CCR
EORI	W	#n,SR	---*00	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \text{ XOR SR} \rightarrow \text{SR}$	Logical exclusive OR #n to SR (Privileged)
EXG	L	Rx,Ry	---*00	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			---*00	-	-	-	-	-	-	-	-	-	-	-	-	PC \rightarrow -(SSP); SR \rightarrow -(SSP)	Generate illegal instruction exception
JMP		d	---*--	-	-	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	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	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 #n,Dy	---*00	e	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, Dx bits left/right
LSR	W	d	---*00	d	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, #n bits L/R (#n: 1 to 8)
MOVE	BWL	s,d	---*00	e	s	e	e	e	e	e	e	e	e	e	s	$s \rightarrow d$	Move data from source to destination
MOVE	W	s,CCR	---*00	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	---*00	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	---*00	d	-	d	d	d	d	d	d	d	-	-	-	SR \rightarrow d	Move Status Register to destination
MOVE	L	USP,An An,USP	---*00	-	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)	(iAnRn)	absW	absL	(iPC)	(iPC,Rn)	#n	
MOVEA*	WL	s,An	-----	s	B	s	s	s	s	s	s	s	s	s	s	s → An
MOVEM*	WL	Rn-Rn,d	-----	-	d	-	-	d	d	d	d	d	-	-	-	Registers → d
		s,Rn-Rn		-	s	s	-	s	s	s	s	s	s	-	-	s → Registers
MOVEP	WL	Dn,(iAn)	-----	s	-	-	-	d	-	-	-	-	-	-	-	Dn → (iAn)...(i+2An)...(i+4A
		(iAn),Dn		d	-	-	-	s	-	-	-	-	-	-	-	(iAn) → Dn...(i+2An)...(i+4A
MOVED*	L	#n,Dn	---*00	d	-	-	-	-	-	-	-	-	-	-	s	#n → Dn
MULS	W	s,Dn	---*00	e	-	s	s	s	s	s	s	s	s	s	s	±16bit s * ±16bit Dn → ±Dn
MULU	W	s,Dn	---*00	e	-	s	s	s	s	s	s	s	s	s	s	16bit s * 16bit Dn → Dn
NBCD	B	d	*U*U*	d	-	d	d	d	d	d	d	d	-	-	-	D - d ₀ - X → d
NEG	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	D - d → d
NEGX	BWL	d	*****	d	-	d	d	d	d	d	d	d	-	-	-	D - d - X → d
NOP			-----	-	-	-	-	-	-	-	-	-	-	-	-	None
NOT	BWL	d	---*00	d	-	d	d	d	d	d	d	d	-	-	-	NOT(d) → d
OR*	BWL	s,Dn	---*00	e	-	s	s	s	s	s	s	s	s	s	s	s OR Dn → Dn
		Dn,d		e	-	d	d	d	d	d	d	d	-	-	-	(Dn OR d) → d
ORI*	BWL	#n,d	---*00	d	-	d	d	d	d	d	d	d	-	-	s	#n OR d → d
ORI*	B	#n,CCR	00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	s	#n OR CCR → CCR
ORI*	W	#n,SR	00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	s	#n OR SR → SR
PEA	L	s	-----	-	s	-	-	s	s	s	s	s	s	s	s	↑s → -(SP)
RESET			-----	-	-	-	-	-	-	-	-	-	-	-	-	Assert RESET Line
ROL	BWL	Dx,Dy	---*0*	e	-	-	-	-	-	-	-	-	-	-	-	c ← Dx, Dx ← Dy, Dy ← c
ROR	W	#n,Dy		d	-	-	-	-	-	-	-	-	-	-	s	Dy ← Dy, Dy ← Dy, Dy ← c
		d		-	d	d	d	d	d	d	d	d	-	-	-	Rotate d 1-bit left/right (W only)
ROXL	BWL	Dx,Dy	---*0*	e	-	-	-	-	-	-	-	-	-	-	-	c ← Dx, Dx ← Dy, Dy ← c
ROXR	W	#n,Dy		d	-	-	-	-	-	-	-	-	-	-	s	Dy ← Dy, Dy ← Dy, Dy ← c
		d		-	d	d	d	d	d	d	d	d	-	-	-	Rotate Dy, #n bits left/right (#n: 1 to 8)
																Rotate destination 1-bit left/right (W only)
RTE			00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → SR; (SP)+ → PC
RTR			00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → CCR; (SP)+ → PC
RTS			-----	-	-	-	-	-	-	-	-	-	-	-	-	(SP)+ → PC
SBCD	B	Dy,Dx	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	Dx ₀ - Dy ₀ - X → Dx ₀
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ax) ₀ - (Ay) ₀ - X → -(Ax) ₀
SCC	B	d	-----	d	-	d	d	d	d	d	d	d	-	-	-	If cc is true then f's → d
																else 0's → d
STOP		#n	00 00 00 00	-	-	-	-	-	-	-	-	-	-	-	s	#n → SR; STOP
SUB*	BWL	s,Dn	*****	e	s	s	s	s	s	s	s	s	s	s	s	Dn - s → Dn
		Dn,d		e	d	d	d	d	d	d	d	d	-	-	-	d - Dn → d
SUBA*	WL	s,An	-----	s	e	s	s	s	s	s	s	s	s	s	s	An - s → An
SUBI*	BWL	#n,d	*****	d	-	d	d	d	d	d	d	d	-	-	s	d - #n → d
SUBQ*	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	s	d - #n → d
SUBX	BWL	Dy,Dx	*****	e	-	-	-	-	-	-	-	-	-	-	-	Dx - Dy - X → Dx
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ax) - (Ay) - X → -(Ax)
SWAP	W	Dn	---*00	d	-	-	-	-	-	-	-	-	-	-	-	bits[31:16] ↔ bits[15:0]
TAS	B	d	---*00	d	-	d	d	d	d	d	d	d	-	-	-	test d → CCR; 1 → bit7 of d
TRAP		#n	-----	-	-	-	-	-	-	-	-	-	-	-	s	PC → -(SSP); SR → -(SSP);
																(vector table entry) → PC
TRAPV			-----	-	-	-	-	-	-	-	-	-	-	-	-	If V then TRAP #7
TST	BWL	d	---*00	d	-	d	d	d	d	d	d	d	-	-	-	test d → CCR
UNLK		An	-----	-	d	-	-	-	-	-	-	-	-	-	-	An → SP; (SP)+ → An
	BWL	s,d	XNZVC	Dn	An	(An)	(An)*	-(An)	(iAn)	(iAnRn)	absW	absL	(iPC)	(iPC,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	(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, 0 cleared, 1 set, U undefined

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

Distributed under the GNU general public use license.

Last name: First name: Group:

ANSWER SHEET TO BE HANDED IN WITH THE SCRIPT**Exercise 2**

Instruction	Memory	Register
Example	\$005000 54 AF 00 40 E7 21 48 C0	A0 = \$00005004 A1 = \$0000500C
Example	\$005008 C9 10 11 C8 D4 36 FF 88	No change
MOVE.B -(A2), -(A1)		
MOVE.W \$5010, -5(A2, D2.L)		
MOVE.L #\$500E, -10(A0, D1.W)		
MOVE.B \$5007(PC), 8(A1)		

Exercise 3

Operation	Size (bits)	Missing Number (hexadecimal)	N	Z	V	C
\$7F + \$?	8		1	0	1	0
\$98BD + \$?	16		0	1	0	1
\$98BD + \$?	32		1	0	0	0

Exercise 4

Use the 32-bit hexadecimal representation.

D1 = \$

D2 = \$

D3 = \$

D4 = \$