

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 = \$FFFF0010 A0 = \$00005000 PC = \$00006000
 D1 = \$0000FFEE A1 = \$00005008
 D2 = \$FFFFFFF9 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  #$ff,d7
next1     moveq.l #1,d1
          cmpi.l  #$01,d7
          bgt   next2
          moveq.l #2,d1
next2     clr.l   d2
          move.l #$11112222,d0
loop2    addq.l  #1,d2
          subq.w #2,d0
          bne   loop2
next3     clr.l   d3
loop3    addq.l  #1,d3
          dbra  d0,loop3      ; DBRA = DBF
next4     clr.l   d4
loop4    move.l  #$12345678,d0
          addq.l  #1,d4
          dbra  d0,loop4      ; DBRA = DBF

```

Exercise 4 (9 points)

All questions in this exercise are independent. **Except for the output registers, none of the data or address registers must be modified when the subroutine returns.** A string of characters always ends with a null character (the value zero). A blank character is either a space character or a tab character.

1. Write the **IsBlank** subroutine that determines if a character is blank (i.e. if it is a space or a tab character).

Input : **D1.B** holds the ASCII code of the character to test.

Output : If the character is blank, **D0.L** returns 0.

If the character is not blank, **D0.L** returns 1.

Tip: The ASCII code of the tab character is 9.

2. Write the **BlankCount** subroutine that returns the number of blank characters in a string. To know if a character is blank, use the **IsBlank** subroutine.

Input : **A0.L** points to a string of character.

Output : **D0.L** returns the number of blank characters in the string.

Tips:

- Use **D2** as a blank-character counter (because **D0** is used by **IsBlank**).
- Then, copy **D2** into **D0** before returning from the subroutine.

3. Write the **BlankToUnderscore** subroutine that converts the blank characters in a string into underscore characters. To know if a character is blank, use the **IsBlank** subroutine.

Input : **A0.L** points to a string of characters.

Output : The blank characters of the string are replaced by the « _ » character.

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	s	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	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	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	s	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	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)	
	W	d		-	-	d	d	d	d	d	d	d	-	-	-	-		Arithmetic shift d 1 bit left/right (.W only)	
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	d	d	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	d	d	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	d	d	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	d	d	d	d	d	d	d	d	d	d	d	d	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	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 → d	Clear destination to zero	
CMP ⁴	BWL	s,Dn	-----	e	s	s	s	s	s	s	s	s	s	s	s	s	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 → Dn if Dn < -1 then addr → PC }	Test condition, decrement and branch (16-bit ± offset to address)	
DIVS	W	s,Dn	---*00	e	-	s	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	---*00	e	-	s	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	---*00	e	-	d	d	d	d	d	d	d	-	-	-	s	Dn XOR d → 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 → 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	L	Rx,Ry	-----	e	e	-	-	-	-	-	-	-	-	-	-	-	register ↔ register	Exchange registers (32-bit only)	
EXT	WL	Dn	---*00	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	d	-	↑d → PC	Jump to effective address of destination	
JSR		d	-----	-	-	d	-	-	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	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)	
	W	d		-	-	d	d	d	d	d	d	d	-	-	-	-		Logical shift d 1 bit left/right (.W only)	
MOVE ⁴	BWL	s,d	---*00	e	s	d	-	-	-	-	-	-	-	-	-	s	s → d	Move data from source to destination	
MOVE	W	s,CCR	=====	s	-	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	s → SR	Move source to Status Register (Privileged)	
MOVE	W	SR,d	-----	d	-	d	d	d	d	d	d	d	-	-	-	s	SR → d	Move Status Register to destination	
MOVE	L	USP,An An,USP	-----	-	d	-	-	-	-	-	-	-	-	-	-	-	USP → An An → USP	Move User Stack Pointer to An (Privileged) 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)	(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 → 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	-	-	-	-	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 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 → -(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)	
ROR	W	d		d	-	-	-	-	-	-	-	-	-	-	-	-	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	
ROXR	W	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	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 ^o	higher than	I(C + Z)	PL	plus	IN
LS ^o	lower or same	C + Z	MI	minus	N
HS ^o , CC ^o	higher or same	IC	GE	greater or equal	!(N ⊕ V)
LO ^o , CS ^o	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.

Last name: First name: Group:

ANSWER SHEET TO BE HANDED IN

Exercise 1

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.L #1024, -4(A1)		
MOVE.B \$5008, -10(A0, D0.W)		
MOVE.L 2(A2), 4(A2, D1.W)		
MOVE.B -1(A2), \$E(A0, D2.L)		

Exercise 2

Operation	Size (bits)	Missing Number (hexadecimal)	N	Z	V	C
\$1A + \$?	8		0	0	0	1
\$7FFF + \$?	16		0	0	0	0
\$7FFFFFFF + \$?	32		1	0	0	0

Exercise 3

Values of registers after the execution of the program. Use the 32-bit hexadecimal representation.	
D1 = \$	D3 = \$
D2 = \$	D4 = \$

Exercise 4

IsBlank

BlankCount

BlankToUnderscore