Key to Final Exam S4
Computer Architecture

Duration: 1 hr 30 min

Write answers only on the answer sheet.

Exercise 1 (3 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 = $0007003D   A0 = $00005000   PC = $00006000
D1 = $12340004   A1 = $00005008
D2 = $FFFFFFFC   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 (2 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  #-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  #$FFFFFFFF,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
```

Exercise 4 (11 points)

All the 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 color is made up of three components (the red, green and blue components). A color is encoded in a 32-bit word as follows: $00RRGGBB_{16}$

- RR represents the red component (8-bit unsigned integer between 0_{16} and FF_{16}).
- GG represents the green component (8-bit unsigned integer between 0_{16} and FF_{16}).
- BB represents the blue component (8-bit unsigned integer between 0_{16} and FF_{16}).

For instance:

- If the encoded value of a color is $002B048D_{16}$, the red component is $2B_{16}$, the green component is 04_{16} and the blue component is $8D_{16}$.
- The encoded value for the black color is 00000000_{16}.
- The encoded value for the white color is $00FFFFFF_{16}$.

1. Write the **SplitColor** subroutine that returns the three components of a color.
   Input: **D0.L** holds a 32-bit encoded color ($00RRGGBB_{16}$).
   Outputs: **D1.B** = Red component of the color (RR).
   D2.B = Green component of the color (GG).
   D3.B = Blue component of the color (BB).

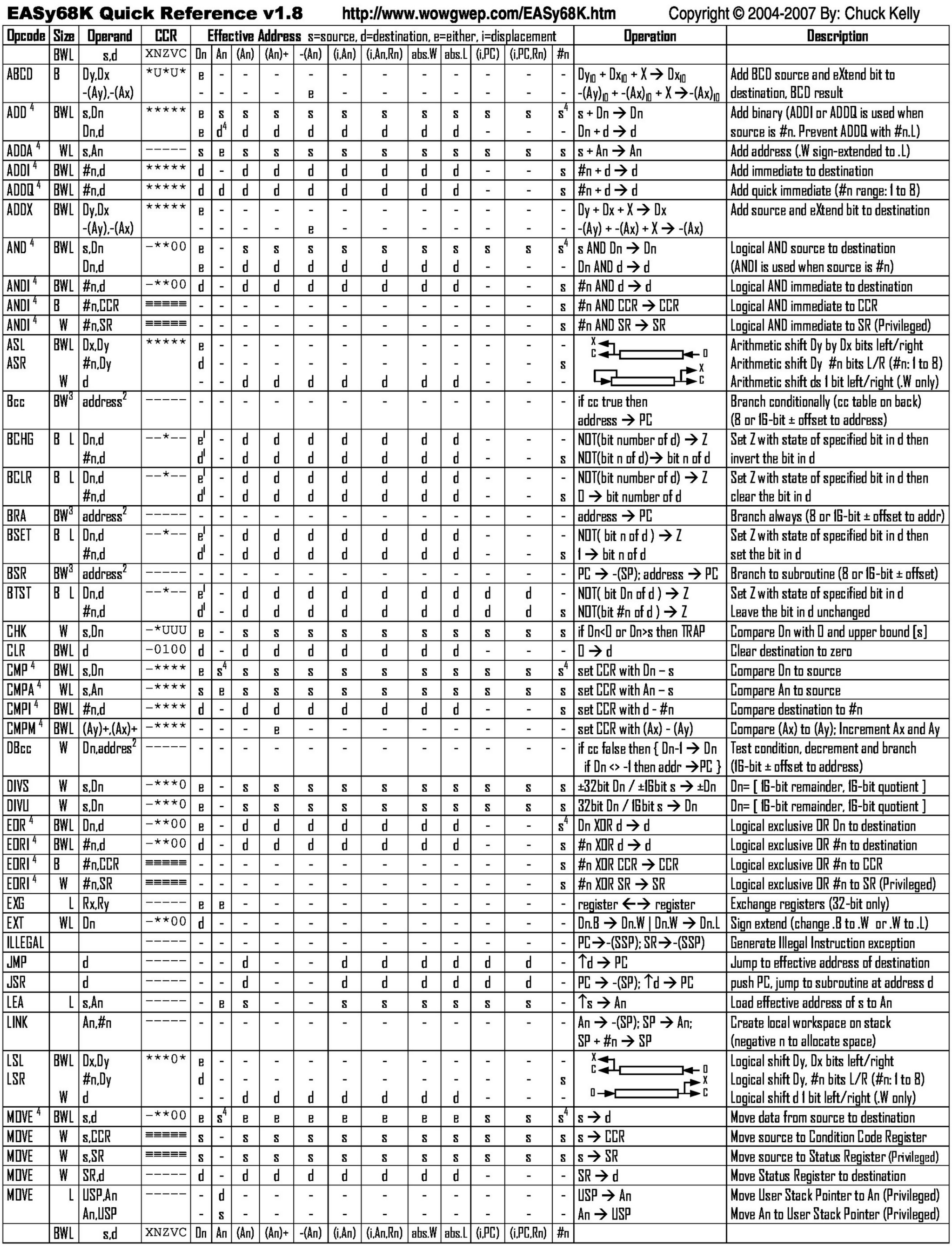
   The SplitColor subroutine must contain 10 lines of instructions at the most (RTS included).

2. By using the **SplitColor** subroutine, write the **IsGray** subroutine that determines whether a color is in grayscale. That is, if the the three components of the color are equal.
   Input: **D0.L** holds a 32-bit encoded color ($00RRGGBB_{16}$).
   Output: **D0.L** = 0 (false), if the color is not in grayscale.
   D0.L = 1 (true), if the color is in grayscale.

   The IsGray subroutine must contain 13 lines of instructions at the most (RTS included).

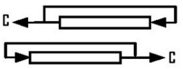
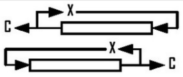
3. By using the **IsGray** subroutine, write the **IsAllGray** subroutine that determines whether all the colors in an array are in grayscale. Each element in the array is a 32-bit encoded color ($00RRGGBB_{16}$).
   Inputs: **A0.L** points to the first element of an array of colors.
   D0.L holds the number of elements in the array.
   Output: **D0.L** = 0 (false), if at least one color in the array is not in grayscale.
   D0.L = 1 (true), if all the colors in the array are in grayscale.

   The IsAllGray subroutine must contain 13 lines of instructions at the most (RTS included).



EASy68K Quick Reference v1.8 http://www.wowgwep.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	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$	Add BCD source and eXtend bit to
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	$-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	destination, BCD result
ADD[4]	BWL	s,Dn	*****	e	s	s	s	s	s	s	s	s	s	s	s[4]	s + Dn → Dn	Add binary (ADDI or ADDQ is used when
		Dn,d		e	d[4]	d	d	d	d	d	d	d	-	-	-	Dn + d → d	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	d	-	-	s	#n + d → d	Add quick immediate (#n range: 1 to 8)
ADDX	BWL	Dy,Dx	*****	e	-	-	-	-	-	-	-	-	-	-	-	Dy + Dx + X → Dx	Add source and eXtend bit to destination
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ay) + -(Ax) + X → -(Ax)	
AND[4]	BWL	s,Dn	-**00	e	-	s	s	s	s	s	s	s	s	s	s[4]	s AND Dn → Dn	Logical AND source to destination
		Dn,d		e	-	d	d	d	d	d	d	d	-	-	-	Dn AND d → d	(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		#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 ds 1 bit left/right (.W only)
Bcc	BW[3]	address[2]	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc true then	Branch conditionally (cc table on back)
																address → PC	(8 or 16-bit ± offset to address)
BCHG	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	-	-	-	NOT(bit number of d) → Z	Set Z with state of specified bit in d then
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	NOT(bit n of d)→ bit n of d	invert the bit in d
BCLR	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	-	-	-	NOT(bit number of d) → Z	Set Z with state of specified bit in d then
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	0 → bit number of d	clear the bit in d
BRA	BW[3]	address[2]	-----	-	-	-	-	-	-	-	-	-	-	-	-	address → PC	Branch always (8 or 16-bit ± offset to addr)
BSET	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	-	-	-	NOT(bit n of d) → Z	Set Z with state of specified bit in d then
		#n,d		d[1]	-	d	d	d	d	d	d	d	-	-	s	1 → bit n of d	set the bit in d
BSR	BW[3]	address[2]	-----	-	-	-	-	-	-	-	-	-	-	-	-	PC → -(SP); address → PC	Branch to subroutine (8 or 16-bit ± offset)
BTST	B L	Dn,d	--*--	e[1]	-	d	d	d	d	d	d	d	d	d	-	NOT(bit Dn of d) → Z	Set Z with state of specified bit in d
		#n,d		d[1]	-	d	d	d	d	d	d	d	d	d	s	NOT(bit #n of d) → Z	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 → 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,addres[2]	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc false then { Dn-1 → Dn	Test condition, decrement and branch
																if Dn <> -1 then addr →PC }	(16-bit ± offset to address)
DIVS	W	s,Dn	-***0	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	-***0	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	-**00	e	-	d	d	d	d	d	d	d	-	-	s[4]	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 → PC	Jump to effective address of destination
JSR		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 → An	Load effective address of s to An
LINK		An,#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	An → -(SP); SP → An;	Create local workspace on stack
																SP + #n → SP	(negative n to allocate space)
LSL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, Dx bits left/right
LSR		#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[4]	e	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	d	-	-	-	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)	(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)	(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 → An	Move source to An (MOVE s,An use MOVEA)
MOVEM[4]	WL	Rn-Rn,d s,Rn-Rn	-----	- -	- -	d s	- s	d -	d s	d s	d s	d s	- s	- s	- -	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	- -	- -	- -	- -	d s	- -	- -	- -	- -	- -	- -	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	±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	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 e	- -	s d	s d	s d	s d	s d	s d	s d	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 ROR	BWL W	Dx,Dy #n,Dy d	-**0*	e d -	- - -	- - d	- - d	- - d	- - d	- - d	- - d	- - d	- - -	- - -	- s -		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 ROXR	BWL W	Dx,Dy #n,Dy d	***0*	e d -	- - -	- - d	- - d	- - d	- - d	- - d	- - d	- - d	- - -	- - -	- s -		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)
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 -	- -	- -	- -	- 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 e	s d[4]	s d	s d	s d	s d	s d	s d	s d	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	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)	(i,An)	(i,An,Rn)	abs.W	abs.L	(i,PC)	(i,PC,Rn)	#n		

Condition Tests (+ OR, ! NOT, ⊕ XOR; [u] Unsigned, [a] Alternate cc)

cc	Condition	Test	cc	Condition	Test
T	true	1	VC	overflow clear	!V
F	false	0	VS	overflow set	V
HI[u]	higher than	!(C + Z)	PL	plus	!N
LS[u]	lower or same	C + Z	MI	minus	N
HS[u], CC[a]	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LO[u], 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

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

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

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.W #0560,-6(A1)`	`$005000  54 AF` **`02 30`** `E7 21 48 C0`	No change
`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`	No change

Exercise 2

Operation	**Size (bits)**	**Missing Number (hexadecimal)**	**N**	**Z**	**V**	**C**
`$4570 + $?`	16	**`$3A90`**	1	0	1	0
`$F431C16A + $?`	32	**`$0BCE3E96`**	0	1	0	1

Exercise 3

Values of registers after the execution of the program. **Use the 32-bit hexadecimal representation.**	
D1 = $00000001	**D3** = $000000FF
D2 = $FFFFFF00	**D4** = $0000FF21

Exercise 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
```