Document No : UniKL MFI_SD_AC41
Revision No: 02
Effective Date: 01 December 2008

CONFIDENTIAL

SET A

UNIVERSITI KUALA LUMPUR
Malaysia France Institute

FINAL EXAMINATION

JANUARY 2014 SESSION

SUBJECT CODE	:	**FSD23102**
SUBJECT TITLE	:	**MICROPROCESSOR**
LEVEL	:	**DIPLOMA**
TIME / DURATION	:	**(2 HOURS)**
DATE	:	

INSTRUCTIONS TO CANDIDATES

1. **Please read the instructions given in the question paper CAREFULLY.**
2. **This question paper is printed on both sides of the paper.**
3. **Please write your answers on the answer booklet provided.**
4. **Answer should be written in blue or black ink except for sketching, graphic and illustration.**
5. **This question paper consists of TWO (2) sections. Section A and B. Answer all questions in Section A. For Section B, answer two (2) questions only.**
6. **Answer all questions in English.**

THERE ARE 8 PAGES OF QUESTIONS AND 2 PAGES OF APPENDICES, EXCLUDING THIS PAGE.

SECTION A (Total: 60 marks)

INSTRUCTION: Answer ALL questions.
Please use the answer booklet provided.

Question 1

(a) State the function of Microprocessor.

(2 marks)

(b) Calculator is a one of Microprocessor System. Draw a block diagram to represent main components for calculator system.

(5 marks)

(c) CPU is the “master” component in Microprocessor System. It consist of ALU,CU and Registers. Briefly explain on the functions of these three (3) items; ALU,CU and Registers.

(6 marks)

(d) Describe the function of System Bus in Microprocessor Based System and state TWO (2) system buses in M68000 microprocessor.

(3 marks)

(e) Define the function of timing circuit in microprocessor interfaces and discuss on the clock signal vs processing speed.

(4 marks)

Question 2

(a) Fill in the blanks with correct answers for the following questions:

i. The size of Data Register are _____ and the function of this register is to _____.

(2 marks)

ii. Address Register consist of 32 bits but only use for _____ with A7 is reserved for _____.

(2 marks)

iii. The size of Status Register is _____. It consist of _____, _____, _____ and Condition Code Register Flag (CCR).

(4 marks)

(b) Figure 1 shows the Pin Assignment for M68000 microprocessor. Based on the figure, answer the following questions:

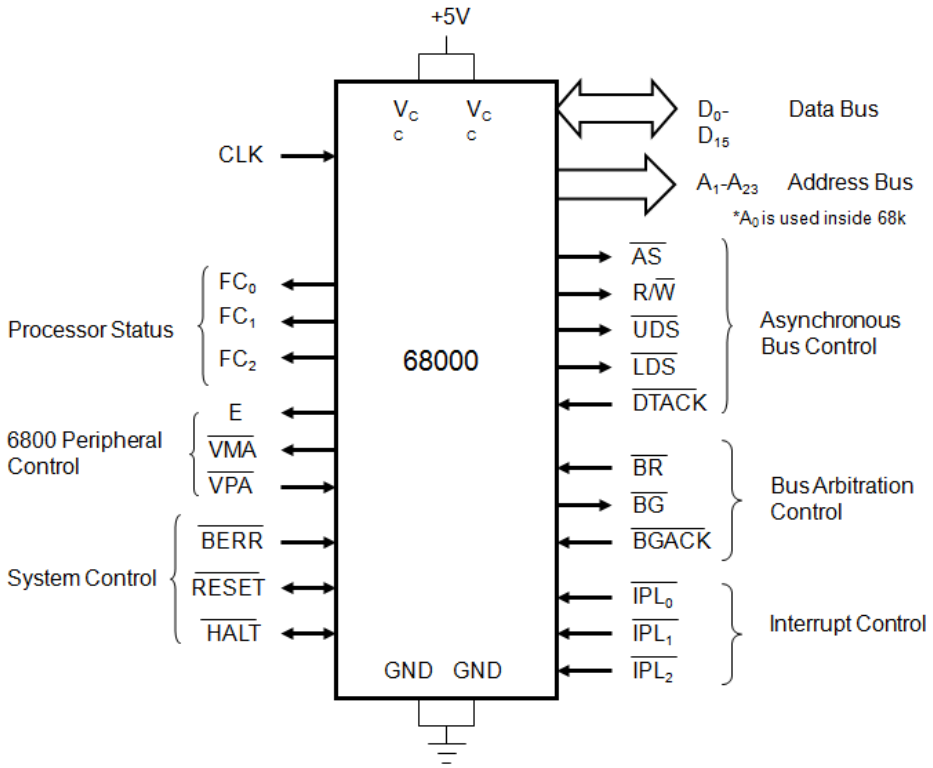


Figure 1: Pin Assignment for M68K microprocessor

i. Describe on the function of Processor Status pin and determine the status of AS for the valid output.

(4 marks)

ii. Briefly explained on how M68000 interfacing with devices using older processors (M6800) via 6800 Peripheral Control pins.

(4 marks)

(c) Consider the following instruction code:

```
MOVE.L   #$12345678,$1000
```

Sketch on how data is usually stored in pairs of chips controlled by UDS and LDS pins in M68000.

(4 marks)

Question 3

Convert and perform the arithmetic operation below and show the conversion procedure algorithmically.

(a) Convert **445** to hexadecimal form.

(3 marks)

(b) Convert **$B4A** to decimal form.

(3 marks)

(c) Convert **%110110100011100** to hexadecimal form.

(2 marks)

(d) Convert signed number **$F4** to decimal form.

(3 marks)

(e) By using *two's complement* binary arithmetic, compute the following operation.

Note: *Your calculations should be in 8-bit format for integer numbers.*

$9C - 87

(6 marks)

(f) If the Status Register contains of **$870A**, determine the state of Trace, Extend and Negative flag.

(3 marks)

SECTION B (Total: 40 marks)

INSTRUCTION: Answer TWO (2) questions only

Please use the answer booklet provided.

Question 4

(a) Figure 2 shows the initial values of Address Registers, Data Registers and memory locations in M68K microprocessor.

Initial Values for Address & Data Registers

A1 = $400401
A2 = $400405
A3 = $600600

D0 = $00000000
D1 = $1234ABCD
D2 = $FFFF1111
D3 = $00000002

Initial Memory

$400400	$78
$400401	$43
$400402	$66
$400403	$AA
$400404	$DE
$400405	$27
$400406	$12
……..	……
……..	……
$600600	$FF
$600601	$FF

Figure 2: Initial values for Address Registers, Data Registers and memory

Explain the contents of the affected registers or memory locations when each of the following instructions are executed. Each instruction is executed independently. The initial values of the registers and memory are the same before each instruction is executed.

i. `SUB.W D2 , D1`

(2 marks)

ii. `MOVE.B $03(A1,D3.W), D1`

(2 marks)

iii. `MOVE. L (A1)+, D0`

(2 marks)

iv. `MOVE. W D2, (A3)`

(2 marks)

(b) Consider the assembly language programs below: .

```
START ORG        $400700
      MOVE.W     #$89C3, D0
      LEA        $400980, A0
      MOVE.W      D0, (A0)
      MOVE.B     (A0), D2
      MOVE.W     (A0), D5
      END   START
```

i. Write comments for the whole programs above.

(6 marks)

ii. Show the contents of D2 and D5 after all the programs above is executed. Assume D2 and D5 contents are $00000000 before execution.

(4 marks)

iii. State the Addressing Mode of the below instruction.

```
MOVE.W     #$89C3, D0
```

(2 marks)

Question 5

Figure 3 shows the assembly language programs, memory address and machine code after the instructions source has been assembled.

INSTRUCTION LINE NO:

```
00100200                          1          ORG     $100200
00100200                          2  START:
00100200                          3
00100200  4240                    4  TOTAL   CLR.W   D0
00100202  123C 0005               5          MOVE.B  #5, D1
00100206  207C 0010021C           6          MOVEA.L #DATA, A0
0010020C  D018                    7  LOOP    ADD.B   (A0)+, D0
0010020E  5301                    8          SUBI.B  #1, D1
00100210  66FA                    9          BNE     LOOP
00100212  227C 0010021A          10          MOVEA.L #SUM, A1
00100218  3280                   11          MOVE.W  D0, (A1)
0010021A= 0000                   12  SUM     DC.W            0
0010021C= 0A 11 05 1B 25         13  DATA    DC.B    $0A,$11,$05,$1B,$25
00100221                         14          END     START
```

Figure 3 : Assembly language programs, memory address and machine code

Based on the Figure 3, answer the following questions:

(a) List the Program Counter value before start the execution.

(1 mark)

(b) State the Conditional Branching instruction that has been used in the Figure 3 and justify the fucntion.

(2 marks)

(c) Based on the answer in Question 5(b), calculate the range of maximum forward and maximum backward of that conditional branching limitation.

(4 marks)

(d) Show the contents of memory address from $10021C until $100220 after execution of the programs above.

(5 marks)

(e) Refer instruction below (line number 7 in Figure 3) and answer the questions given:

```
LOOP        ADD.B (A0)+, D0
```

i. Identify the target, action and label name for the instruction above.

(3 marks)

ii. State the Addressing Mode and briefly explain on the instruction above.

(3 marks)

iii. Show the contents of A0 and D0 after this instructions was executed. Consider the initial value of A0 is $0010021C and D0 is $00000000.

(2 marks)

Question 6

(a) Design a flowchart to inspect the contents D2 and, if the contents are greater than $55, divide the value with $02 and store the result in memory location $100000. Otherwise, multiply it with $04 and store the result in memory location $200000.

(4 marks)

(b) Write an assembly language program to represent your algorithm written in Question 6 (a). Your program also must store value $60 in D2 as initial data to compare.

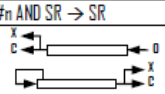
(12 marks)

(c) Briefly explain on the Conditional Branching and give TWO (2) example of instruction in this type.

(4 marks)

END OF QUESTION

APPENDIX 1: M68K Datasheet

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		

APPENDIX 1: M68K Datasheet (continue)

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	-----	-	-	d	-	d	d	d	d	d	-	-	-	Registers → d	Move specified registers to/from memory
		s,Rn-Rn		-	-	s	s	-	s	s	s	s	s	s	-	s → Registers	(.W source is sign-extended to .L for Rn)
MOVEP	WL	Dn,(i,An)	-----	s	-	-	-	-	d	-	-	-	-	-	-	Dn → (i,An)...(i+2,An)...(i+4,A.	Move Dn to/from alternate memory bytes
		(i,An),Dn		d	-	-	-	-	s	-	-	-	-	-	-	(i,An) → Dn...(i+2,An)...(i+4,A.	(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	-**00	e	-	s	s	s	s	s	s	s	s	s	s[4]	s OR Dn → Dn	Logical OR
		Dn,d		e	-	d	d	d	d	d	d	d	-	-	-	Dn OR d → d	(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		#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		#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	*U*U*	e	-	-	-	-	-	-	-	-	-	-	-	$Dx_{10} - Dy_{10} - X \rightarrow Dx_{10}$	Subtract BCD source and eXtend bit from
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	$-(Ax)_{10} - -(Ay)_{10} - X \rightarrow -(Ax)_{10}$	destination, BCD result
Scc	B	d	-----	d	-	d	d	d	d	d	d	d	-	-	-	If cc is true then 1's → d	If cc true then d.B = 11111111
																else 0's → d	else d.B = 00000000
STOP		#n	=====	-	-	-	-	-	-	-	-	-	-	-	s	#n → SR; STOP	Move #n to SR, stop processor (Privileged)
SUB [4]	BWL	s,Dn	*****	e	s	s	s	s	s	s	s	s	s	s	s[4]	Dn - s → Dn	Subtract binary (SUBI or SUBQ used when
		Dn,d		e	d[4]	d	d	d	d	d	d	d	-	-	-	d - Dn → d	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	*****	e	-	-	-	-	-	-	-	-	-	-	-	Dx - Dy - X → Dx	Subtract source and eXtend bit from
		-(Ay),-(Ax)		-	-	-	-	e	-	-	-	-	-	-	-	-(Ax) - -(Ay) – X → -(Ax)	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);	Push PC and SR, PC set by vector table #n
																(vector table entry) → PC	(#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, ² Alternate cc)					
cc	Condition	Test	cc	Condition	Test
T	true	1	VC	overflow clear	!V
F	false	0	VS	overflow set	V
HI*	higher than	!(C + Z)	PL	plus	!N
LS*	lower or same	C + Z	MI	minus	N
HS*, CC²	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LO*, CS²	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