A summary of the 680X0's instruction types (excluding floating-point instructions).

Instruction type	*Opcode*	*Description*
Data transfer	EXG	Exchange (swap) contents of two registers
	MOVE	Move (copy) data unchanged from source to destination
	SWAP	Swap left and right halves of register
Data processing	ABCD	Add decimal (BCD) numbers with carry (extend) flag
	ADD	Add binary (two's-complement) numbers
	AND	Bitwise logical AND
	AS*x*	Arithmetic shift left (*x* = L) or right (*x* = R) with sign extensi
	CLR	Clear operand by resetting all bits to 0
	DIV	Divide binary numbers
	EOR	Bitwise logical EXCLUSIVE OR
	EXT	Extend the sign bit of subword to fill register
	LS*x*	Logical shift left (*x* = L) or right (*x* = R)
	MUL	Multiply binary numbers
	NBCD	Negate decimal number (subtract with carry from zero)
	NEG	Negate binary number (subtract from zero)
	NOT	Bitwise logical complement
	OR	Bitwise logical OR
	RO*x*	Rotate left (*x* = L) or right (*x* = R)
	SBCD	Subtract decimal (BCD) numbers
	SUB	Subtract binary (two's-complement) numbers
Program control	B*cc*	Branch relative to *PC* if specified condition *cc* is true
	BRA	Branch unconditionally, relative to *PC*
	BSR	Call (branch to) subroutine at address relative to *PC*; save *PC* st (return address) in stack
	CMP	Compare two operand values and set flags based on result
	DB*cc*	Loop instruction: Test condition *cc* and perform no operation if condition is true; otherwise, decrement specified register and branch to specified address
	JMP	Branch unconditionally to specified address
	JSR	Call (jump to) subroutine at specified address; save *PC* state (return address) in stack
	NOP	No operation, but instruction execution continues
	RTS	Return from subroutine
	S*cc*	Set operand to 1s (0s) if condition *cc* is true (false)
	TST	Test an operand by comparing it to zero and setting flags

SOME INSTRUCTIONS OF 68000

		Effect	Restrictions
	Transfer of Data		
	MOVE.size <s>,<d>	<d> ← <s>	(<d> ≠ Ai)
(Address)	MOVEA.size <s>,<d>	<d> ← <s>	(<d> = Ai) (size ≠ B)
(Quick)	MOVEQ.size <s>,<d>	<d> ← <s>	(<s> = Imm) (<d> ≠ Ai) (size ≠ B | W)
	SWAP Di	Di(31:16) ↔ Di(15:00)	
	EXG Xi, Xj	Xi(31:00) ↔ Xj(31:00)	
	Addition		
	ADD.size <s>,<d>	<d> ← <d> + <s>	(<d> ≠ Ai) (<s> | <d> = Di)
(Address)	ADDA.size <s>,<d>	<d> ← <d> + <s>	(<d> = Ai) (size ≠ B)
(Immediate)	ADDI.size <s>,<d>	<d> ← <d> + <s>	(<s> = Imm) (<d> ≠ Ai)
	Subtraction	Same as ADD, but with keyword **SUB**	
	Multiplication	(restricted size: both operands are words)	
(Unsigned)	MULU <s>, Di	Di(31:00) ← Di(15:00) * <s>	(<s> ≠ Ai)
(Signed)	MULS <s>, Di	Di(31:00) ← Di(15:00) * <s>	(<s> ≠ Ai)
	Division	(restricted size: size of <s> = W)	
(Unsigned)	DIVU <s>, Di	Di(31:00) ← Di(31:00) / <s>	(<s> ≠ Ai)
(Signed)	DIVS <s>, Di	Di(31:00) ← Di(31:00) / <s>	(<s> ≠ Ai)
		Remainder= Di(31:16) & Quotient = Di(15:00)	
	Other		
	CLR.size <d>	<d> ← 0	(<d> ≠ Ai)
	NEG.size <d>	<d> ← 2's C of <d>	(<d> ≠ Ai)
	Logical		
	AND.size <s>,<d>	<d> ← <d> ∩ <s>	(<d> ≠ Ai) (<s> | <d> = Di)
(Immediate)	ANDI.size <s>,<d>	<d> ← <d> ∩ <s>	(<s> = Imm) (<d> ≠ Ai)
	OR.size <s>,<d>	<d> ← <d> ∪ <s>	(<d> ≠ Ai) (<s> | <d> = Di)
(Immediate)	ORI.size <s>,<d>	<d> ← <d> ∪ <s>	(<s>= Imm) (<d> ≠ Ai)
	EOR.size <s>,<d>	<d> ← <d> ⊕ <s>	(<s> = Di) (<d> ≠ Ai)
(Immediate)	EORI.size <s>,<d>	<d> ← <d> ⊕ <s>	(<s>= Imm) (<d> ≠ Ai) (size≠W)
	NOT.size <d>	<d> ← <d>′	(<d> ≠ Ai)

Transfer of Control

Bcc <relative address or label>
Branch to (relative address or label) if condition (cc) is true

A) Conditional Transfer

(cc)	Name	Flag Conditions
CC	Carry clear	C = 0
CS	Carry set	C = 1
EQ	Equal to (0)	Z = 1
GE	Greater than or equal to	N ⊕ V = 0
GT	Greater than	(N ⊕ V) + Z = 0
HI	Higher than	C + Z = 0
LE	Less than or equal to	(N ⊕ V) + Z = 1
LS	Lower than or the same as	C + Z = 1
LT	Less than	N ⊕ V = 1
MI	Minus, or negative	N = 1
NE	Not equal to (0)	Z = 0
PL	Plus, or positive	N = 0
VC	No overflow	V = 0
VS	Result is too large	V = 1

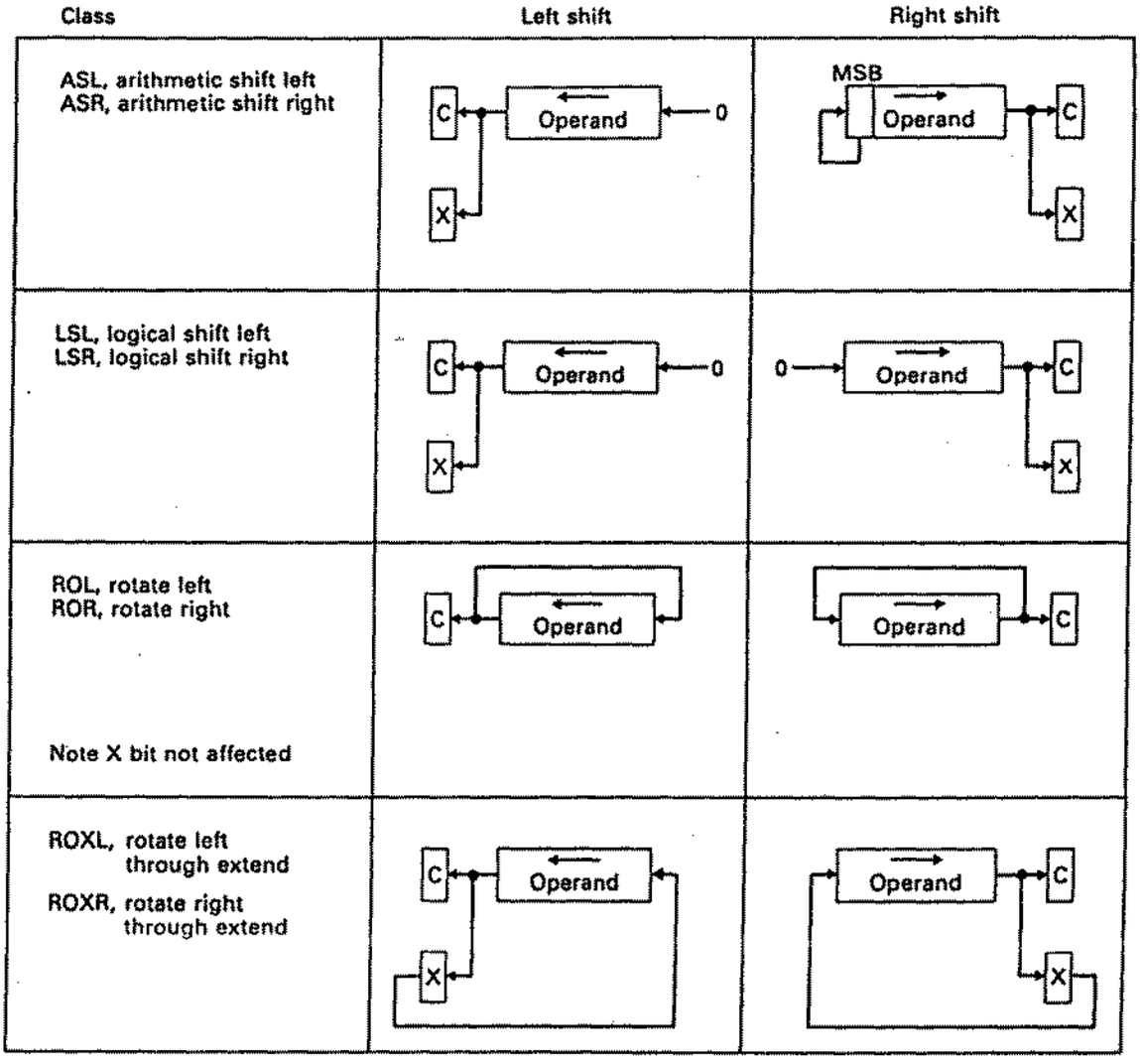
B) Unconditional Transfer

JMP <relative address or label>
BRA <relative address or label>

Branch to (relative address or label) unconditionally

	Compare	Effect	Restriction
	CMP.size <s>, <d>	? = <d> - <s>	(<d> = Di)
(Address)	CMPA.size <s>, <d>	? = <d> - <s>	(<d> = Ai) (size ≠ B)
(Immediate)	CMPI.size <s>, <d>	? = <d> - <s>	(<s> = Imm) (<d> ≠ Ai)
(Memory)	CMPM.size <s>, <d>	? = <d> - <s>	(Both <s> and <d> = (Ai)+)

68000's shift and rotate instructions



Arithmetic shifts update *all* bits of the CCR. The N and Z bits are set or cleared as we would expect. The V bit is set if the most significant bit of the operand is changed at any time during the shift operation. The C and X bits are set according to the last bit shifted out of the operand. However, if the shift count is zero, C is cleared and X is unaffected. Logical shifts and rotates clear the V bit.

Assembly Language Form of Shift Operations

All eight shift instructions are expressed in one of three ways. These are illustrated by the ASL (arithmetic shift left) instruction.

Mode 1.	ASL Dx,Dy	Shift Dy by Dx bits
Mode 2.	ASL #⟨data⟩,Dy	Shift Dy by #data bits
Mode 3.	ASL ⟨ea⟩	Shift the contents of ea by one place

A shift instruction can be applied to a byte, word, or longword operand, with the exception of mode 3 shifts, which act only on words.

In mode 1, the "source" operand, Dx, specifies how many places the destination operand, Dy, needs to be shifted. Dy may be shifted by 1 to 32 bits. In mode 2, the literal, #⟨data⟩, specifies how many places Dy needs to be shifted; this must be in the range 1 to 8. In mode 3, the memory location specified by the effective address, ⟨ea⟩, is shifted one place. Many microprocessors permit only the *static* shifts of modes 2 and 3. The 68000 permits *dynamic* shifts (i.e., mode 1) because the number of bits to be shifted is computed at run-time.

ASSEMBLER DIRECTIVES

(Allocate storage, Define constants, Link identifiers to values, control assembling)
DS DC EQU

A line starting at column 1 with an * is a comment.

Any other line is composed of three parts:

- Label (starts at column 1 with a letter and consists of at most 8 characters)
- instruction or assembler directive
- Comment

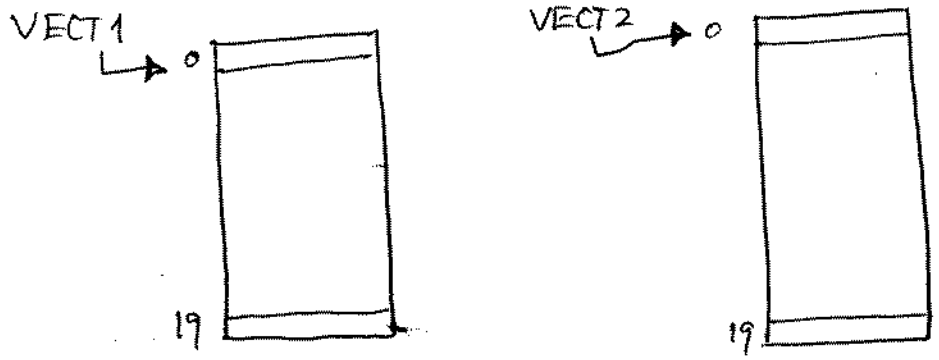
(1st and 3rd parts are optional. If label is not used then column 1 is empty)

```
            TTL                     (gives the title of the program)
identifier  EQU     value           (links an identifier to a value)
            ORG     $HHHHHH         (Origin of data)
identifier  DS.size length          (reserves memory space for variables)
identifier  DC.size value           (defines constants)
            ORG     $HHHHHH         (Origin of program)
            .....

            INSTRUCTIONS ARE PLACED HERE

            .....
            END                     (indicates the end of the program)
```

a) Given two vectors of 20 one-byte numbers each, located at addresses VECT1 and VECT2.



Add these vectors and put the resulting vector at address VECT2.

```
AddVect:
        CLR.L       D0              D0 ← 0
        MOVEA.L     VECT1, A0       A0 ← VECT1
        MOVEA.L     VECT2, A1       A1 ← VECT2

Loop:
        MOVE.B      (A0)+, D1       D1 ← M[A0] ; A0 ← A0 + 1

        ADD.B       D1, (A1)+       M[A1] ← M[A1] + D1 ; A1 ← A1 + 1

        ADDI.B      #$1, D0         D0 ← D0 + 1
        CMPI.B      #20, D0         D0 - 20 ($14) ?
        BNE         Loop            ≠ , branch to Loop.

Done:
```

Note that

```
        ADD.B       D1, (A1)+
```

can be replaced by

```
        ADD.B       D1, (A1)
        ADDA.L      #1, A1
```

b) Given a vector of 10 one-byte numbers,
located at address NUMBERS.
Find the sum (address SUM) and
the average (address AVERAGE) of these numbers.

```
Stats:
        CLR.L       D0          D0 ← 0
        MOVE.B      #10, D1     D1 ← 10
        CLR.L       D2          D2 ← 0
        MOVEA.L     NUMBERS, A0 A0 ← NUMBERS

Loop:
        MOVE.B      (A0)+, D2   D2 ← M[A0]; A0 ← A0+1

        ADD.L       D2, D0      D0 ← D0 + D2
        SUBI.B      #1, D1      D1 ← D1 - 1
        BNE         Loop        if (D1 ≠ 0) branch to Loop

        MOVE.W      D0, SUM     M[SUM] ← D0

        .
        DIVU        #$A, D0     D0.W ← D0/10
        MOVE.B      D0, AVERAGE M[AVERAGE] ← D0
Done:
```