AmigaFlight Data Movement Instructions

Andrew Duffy Morris

COLLABORATORS			
	TITLE : AmigaFlight Data Movement Instructions		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Andrew Duffy Morris	July 20, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AmigaFlight Data Movement Instructions

1.1 AmigaFlight® Help: Data Movement Instructions

```
Data Movement Instructions
==========================

The basic method  of data aquisition (transfer and storage) is provided by
the move instruction.  The move instruction  and the  effective addressing
modes allow both  address and  data manipulation.  Data move  instructions
allow byte,  word,  and long word  operand  transfers and ensure that only
legal address manipulations are executed.  In addition to the general move
instruction, there are several special data movement instructions.


  Move
  ----
  MOVE <ea>,<ea>    Move Data from Source to Destination
  MOVE SR,<ea>      Move from Status Register
  MOVE <ea>,SR      Move to Status Register (Privileged)
  MOVE <ea>,CCR     Move to Condition Codes

  Move Multiple
  -------------
  MOVEM             Move Multiple Registers

  Move Address
  ------------
  MOVEA <ea>,An     Move Address

  Load Effective Address
  ----------------------
  LEA               Load Effective Address

  Push Effective Address
  ----------------------
  PEA <ea>          Push Effective Address

  Move Peripheral Data
  --------------------
  MOVEP             Move Peripheral Data
```

```
  Move Quick
  ----------
  MOVEQ #d,Dn       Move Quick

  Register Swap and Exchange
  --------------------------
  SWAP Dn           Swap Data Register Halves
  EXG               Exchange Registers

  Link and Unlink
  ---------------
  LINK An,#<dis>    Link and Allocate
  UNLK An           Unlink
```

1.2 AmigaFlight® Help: Exchange Registers

```
EXG Exchange Registers
==========================

  Exchange the contents of the source and destination registers. All
  32-bits are always exchanged. Any two registers may be specified.

  Rx <-> Ry


Assembler Syntax
----------------
  EXG{.L}   Rx,Ry


Addressing Modes
----------------
  Mode             Source  Destination

  Data Register Direct      * *
  Address Register Direct     * *
  Address Register Indirect   - -
  Postincrement Register Indirect   - -
  Predecrement Register Indirect    - -
  Register Indirect with Offset   - -
  Register Indirect with Index    - -
  Absolute Short        - -
  Absolute Long       - -
  P.C. Relative with Offset   - -
  P.C. Relative with Index    - -
  Immediate       - -


Data Size
---------
  Long


Status Flags
------------
```

```
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
      # p
  Dx,Dy   2 6
  Dx,Ay   2 6
  Ax,Ay   2 6

  # = no. of program bytes
  p = no. of instruction clock periods
```

1.3 AmigaFlight® Help: Load Effective Address

```
LEA Load Effective Address
==========================

  Load  the  calculated  (effective)  address  into  the destination
  address register.

  Destn -> An


Assembler Syntax
----------------
  LEA{.L}   <ea>,An

  <ea> - control


Addressing Modes
----------------
  Mode             Source  Destination

  Data Register Direct      - -
  Address Register Direct     - *
  Address Register Indirect   * -
  Postincrement Register Indirect   - -
  Predecrement Register Indirect    * -
  Register Indirect with Offset   * -
  Register Indirect with Index    * -
  Absolute Short        * -
  Absolute Long       * -
  P.C. Relative with Offset   * -
  P.C. Relative with Index    * -
  Immediate       - -


Data Size
---------
```

```
  Long


Status Flags
------------
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
  <ea>     # p
  (An)     2  4
  d16(An)  4  8
  d8(An,Ri) 4 12
  Abs short 4  8
  Abs long  6 12
  d16(PC)  4  8
  d8(PC,Ri) 4 12

  # = no. of program bytes
  p = no. of instruction clock periods
```

1.4 AmigaFlight® Help: Link and Allocate

```
LINK  Link and Allocate
=======================

  Push the current contents of the destination address register onto
  the stack. Load the contents of the stack pointer into the
  destination address register. Add the immediate value to the stack
  pointer.
  This instruction  is commonly used at subroutine entry to allocate
  a new frame pointer and  local temporary storage. This is normally
  done with a negative displacement.

  An -> SP@- : SP -> An : SP + d -> SP


Assembler Syntax
----------------
  LINK{.W}  An,#<displacement>

  where the displacement is a sign extended 16 bit value


Data Size
---------
  Unsized


Status Flags
```

```
------------
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
  <ea>    # p
      4 16

  # = no. of program bytes
  p = no. of instruction clock periods
```

1.5 AmigaFlight® Help: Move Data from Source to Destination

```
MOVE  Move Data from Source to Destination
==========================================

  Copy the source operand to the destination operand. The upper byte
  of  data is  ignored  when  moving  data  to  the  condition  code
  register. The move instructions that load and store the user stack
  pointer from and to an address register may only be executed while
  in supervisor mode.

  Source -> Destn


Assembler Syntax
----------------
  MOVE{.[B/W/L]}  <ea>,<ea>
  MOVE{.[W/L]}  <ea>,An
  MOVE{.W}  <ea>,CCR
  MOVE{.W}  <ea>,SR
  MOVE{.W}  SR,<ea>
  MOVE{.L}  An,USP
  MOVE{.L}  USP,An

  Source <ea> - all modes
  Destination <ea> -data alterable only
  USP - User Stack Pointer
  CCR - Condition Code Register


Addressing Modes
----------------
  Mode             Source  Destination

  Data Register Direct      * *
  Address Register Direct     * -
  Address Register Indirect   * *
  Postincrement Register Indirect   * *
  Predecrement Register Indirect    * *
```

```
  Register Indirect with Offset   * *
  Register Indirect with Index    * *
  Absolute Short        * *
  Absolute Long       * *
  P.C. Relative with Offset   - -
  P.C. Relative with Index    - -
  Immediate       * -
```

Data Size

Byte, Word, Long

Status Flags

N	Set if negative
Z	Set if zero
V	Always cleared
C	Always cleared
X	Not affected

Instruction Size and Cycles to Execute

BYTE/WORD	D E S T I N A T I O N							
Source	Dn		(An)		(An)+		-(An)	
<ea>	#	p	#	p	#	p	#	p
Dn	2	4	2	8	2	8	2	8
An (word)	2	4	2	8	2	8	2	8
(An)	2	8	2	12	2	12	2	12
(An)+	2	8	2	12	2	12	2	12
-(An)	2	10	2	14	2	14	2	14
d16(An)	4	12	4	16	4	16	4	16
d8(An,Ri)	4	14	4	18	4	18	4	18
Abs short	4	12	4	16	4	16	4	16
Abs long	6	16	6	20	6	20	6	20
d16(PC)	4	12	4	16	4	16	4	16
d8(PC,Ri)	4	14	4	18	4	18	4	18
Immediate	4	8	4	12	4	12	4	12

\# = no. of instruction bytes
p = no. of instruction clock periods

BYTE/WORD	D E S T I N A T I O N							
Source	d(An)		d(An,Ri)		AbsW		AbsL	
<ea>	#	p	#	p	#	p	#	p
Dn	4	12	4	14	4	12	6	16
An (word)	4	12	4	14	4	12	6	16
(An)	4	16	4	18	4	16	6	20
(An)+	4	16	4	18	4	16	6	20
-(An)	4	18	4	20	4	18	6	22
d16(An)	6	20	6	22	6	20	8	24
d8(An,Ri)	6	22	6	24	6	22	8	26
Abs short	6	20	6	22	6	20	8	24
Abs long	8	24	8	26	8	24	10	28

```
  d16(PC)    6  20      6  22      6  20      8  24
  d8(PC,Ri)  6  22      6  24      6  22      8  26
  Immediate  6  16      6  18      6  16      8  20

  # = no. of instruction bytes
  p = no. of instruction clock periods

  LONG       D E S T I N A T I O N
  Source      Dn         (An)      (An)+      -(An)
  <ea>     #   p      #   p      #   p      #   p
  Dn     2   4      2  12      2  12      2  12
  An (word)  2   4      2  18      2  12      2  12
  (An)       2  12      2  20      2  20      2  20
  (An)+      2  12      2  20      2  20      2  20
  -(An)      2  14      2  22      2  22      2  22
  d16(An)    4  16      4  24      4  24      4  24
  d8(An,Ri)  4  18      4  26      4  26      4  26
  Abs short  4  16      4  24      4  24      4  24
  Abs long   6  20      6  28      6  28      6  28
  d16(PC)    4  16      4  24      4  24      4  24
  d8(PC,Ri)  4  18      4  26      4  26      4  26
  Immediate  4  12      4  20      4  20      4  20

  # = no. of instruction bytes
  p = no. of instruction clock periods

  LONG       D E S T I N A T I O N
  Source     d(An)     d(An,Ri)    AbsW       AbsL
  <ea>     #   p      #   p      #   p      #   p
  Dn     4  16      4  18      4  16      6  20
  An (word)  4  16      4  18      4  16      6  20
  (An)       4  24      4  26      4  24      6  28
  (An)+      4  24      4  26      4  24      6  28
  -(An)      4  26      4  28      4  26      6  30
  d16(An)    6  28      6  30      6  28      8  32
  d8(An,Ri)  6  30      6  32      6  30      8  34
  Abs short  6  28      6  30      6  28      8  32
  Abs long   8  32      8  34      8  32     10  36
  d16(PC)    6  28      6  30      6  28      8  32
  d8(PC,Ri)  6  30      6  32      6  30      8  34
  Immediate  6  24      6  26      6  24      8  28

  # = no. of instruction bytes
  p = no. of instruction clock periods
```

1.6 AmigaFlight® Help: MOVE to Condition Codes

```
MOVE_CCR  MOVE to Condition Codes
=====================================

  Copy the source operand to the  Condition Code Register. The upper
  byte of data is ignored when moving  data to  the  condition  code
  register.

  Source -> CCR
```

Assembler Syntax

```
  MOVE{.W}  <ea>,CCR

  <ea> - data only
```

Addressing Modes

Mode	Source	Destination
Data Register Direct	*	-
Address Register Direct	-	-
Address Register Indirect	*	-
Postincrement Register Indirect	*	-
Predecrement Register Indirect	*	-
Register Indirect with Offset	*	-
Register Indirect with Index	*	-
Absolute Short	*	-
Absolute Long	*	-
P.C. Relative with Offset	*	-
P.C. Relative with Index	*	-
Immediate	*	-

Data Size

Word (although only low byte is used to update CCR)

Status Flags

N	Set according to source operand
Z	Set according to source operand
V	Set according to source operand
C	Set according to source operand
X	Set according to source operand

Instruction Size and Cycles to Execute

<ea>	#	p
Dn	2	12
(An)	2	16
(An)+	2	16
-(An)	2	18
d16(An)	4	20
d8(An,Ri)	4	22
Abs short	4	20
Abs long	6	24
d16(PC)	4	20
d8(PC,Ri)	4	22
Immediate	4	16

= no. of instruction bytes

```
  p = no. of instruction clock periods
```

1.7 AmigaFlight® Help: MOVE from Status Register

```
MOVE_SR   MOVE from Status Register
=========================================

  Copy the Status Register to the destination operand.

  SR -> Destination


Assembler Syntax
----------------
  MOVE{.W}  SR,<ea>

  <ea> - data alterable


Addressing Modes
----------------
  Mode             Source  Destination

  Data Register Direct      - *
  Address Register Direct     - -
  Address Register Indirect   - *
  Postincrement Register Indirect   - *
  Predecrement Register Indirect    - *
  Register Indirect with Offset   - *
  Register Indirect with Index    - *
  Absolute Short        - *
  Absolute Long       - *
  P.C. Relative with Offset   - -
  P.C. Relative with Index    - -
  Immediate       - -


Data Size
---------
  Word


Status Flags
------------
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
  <ea>    # p
  Dn    2 6
```

```
  (An)     2 12
  (An)+    2 12
  -(An)    2 14
  d16(An)    4 16
  d8(An,Ri)  4 18
  Abs short  4 16
  Abs long   6 20

  # = no. of instruction bytes
  p = no. of instruction clock periods
```

1.8 AmigaFlight® Help: Move Address

```
MOVEA Move Address
==================

  Copy the source operand to the destination operand. This opcode is
  a subset of the MOVE opcode, and requires that  the destination be
  an  address  register. If  the  value  is  loaded as a 16-bit word
  value, this value is automatically sign-extended.

  Source -> Address register


Assembler Syntax
----------------
  MOVE{.[W/L]}  <ea>,An

  <ea> - all modes


Addressing Modes
----------------
  Mode              Source  Destination

  Data Register Direct      * -
  Address Register Direct     * *
  Address Register Indirect   * -
  Postincrement Register Indirect   * -
  Predecrement Register Indirect    * -
  Register Indirect with Offset   * -
  Register Indirect with Index    * -
  Absolute Short        * -
  Absolute Long       * -
  P.C. Relative with Offset   * -
  P.C. Relative with Index    * -
  Immediate       * -


Data Size
---------
  Word, Long


Status Flags
```

```
------------
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
  <ea>     Word    Long
      #    p    #    p
  Dn    2    4    2    4
  An    2    4    2    4
  (An)    2    8    2   12
  (An)+   2    8    2   12
  -(An)   2   10    2   14
  d16(An)   4   12    4   16
  d8(An,Ri) 4   14    4   18
  Abs short 4   12    4   16
  Abs long  6   16    6   20
  d16(PC)   4   12    4   16
  d8(PC,Ri) 4   14    4   18
  Immediate 4    8    6   12

  # = no. of instruction bytes
  p = no. of instruction clock periods
```

1.9 AmigaFlight® Help: Move Multiple Registers

```
MOVEM Move Multiple Registers
=============================

  Transfer  the selected registers from the register list to or from
  the  consecutive  memory locations starting at the memory location
  specified by the effective address. The register list is evaluated
  to  a  mask  that  specifies  the  list  of  the  registers  to be
  transferred.

  Registers -> Destination
  Source -> Registers

  Selected registers  are transferred  to or from consecutive memory
  starting at <ea>


Assembler Syntax
----------------
  MOVEM{.[W/L]} <register list>,<ea>
  MOVEM{.[W/L]} <ea>,<register list>

  <ea> destination - control alterable and predecrement

  <ea> source      - control alterable and postincrement
```

```
  register list example: D3-D7/A1/A6/D1


Addressing Modes
----------------
  Mode              Source  Destination

  Data Register Direct      - -
  Address Register Direct     - -
  Address Register Indirect   - *
  Postincrement Register Indirect   - *
  Predecrement Register Indirect    - *
  Register Indirect with Offset   - *
  Register Indirect with Index    - *
  Absolute Short        - *
  Absolute Long       - *
  P.C. Relative with Offset   - -
  P.C. Relative with Index    - -
  Immediate       - -


Data Size
---------
  Word, Long


Status Flags
------------
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
  WORD    <list>,<ea> <ea>,<list>
      #     p   #     p
  (An)    4    8+4n 4   12+4n
  (An)+   invalid   4   12+4n
  -(An)   4    8+4n invalid
  d16(An)   6   12+4n 6   16+4n
  d8(An,Ri) 6   14+4n 6   18+4n
  Abs short 6   12+4n 6   16+4n
  Abs long  8   16+4n 8   20+4n
  d16(PC)   invalid   6   16+4n
  d8(PC,Ri) invalid   6   18+4n

  # = no. of instruction bytes
  p = no. of instruction clock periods
  n = no. of registers in list

  LONG    <list>,<ea> <ea>,<list>
      #     p   #     p
  (An)    4    8+8n 4   12+8n
  (An)+   invalid   4   12+8n
```

```
  -(An)    4    8+8n invalid
  d16(An)   6   12+8n 6   16+8n
  d8(An,Ri) 6   14+8n 6   18+8n
  Abs short 6   12+8n 6   16+8n
  Abs long  8   16+8n 8   20+8n
  d16(PC)   invalid   6   16+8n
  d8(PC,Ri) invalid   6   18+8n

  # = no. of instruction bytes
  p = no. of instruction clock periods
  n = no. of registers in list
```

1.10 AmigaFlight® Help: Move Peripheral Data

```
MOVEP Move Peripheral Data
==========================

  Copy  the  source  operand  to  the  destination  operand.  This
  instruction transfers data in  alternate  bytes to or from memory.
  The  starting  address  is specified  by the  displacement  of the
  specified  address  register,  and  the  remaining  addresses  are
  specified  by  incrementing  the  transfer location  by  two. This
  instruction is designed to facilitate the transfer of data between
  8-bit devices and the 16-bit data bus.

  Source -> Destination

  Transfer bytes of data register (high order byte first) to or from
  alternate  bytes of  memory starting at d(Ay)  and incrementing by
  two.


Assembler Syntax
----------------
  MOVEP{.[W/L]} Dx,d(Ay)
  MOVEP{.[W/L]} d(Ay),Dx


Addressing Modes
----------------
  Mode             Source  Destination

  Data Register Direct      - -
  Address Register Direct     - -
  Address Register Indirect   - -
  Postincrement Register Indirect   - -
  Predecrement Register Indirect    - -
  Register Indirect with Offset   - -
  Register Indirect with Index    - -
  Absolute Short        - -
  Absolute Long       - -
  P.C. Relative with Offset   - -
  P.C. Relative with Index    - -
  Immediate       - -
```

```
Data Size
---------
  Word, Long


Status Flags
------------
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
      # p
  Word    4 16
  Long    4 24

  # = no. of instruction bytes
  p = no. of instruction clock periods
```

1.11 AmigaFlight® Help: Move Quick

```
MOVEQ Move Quick
================

  Copy the source operand  to the  destination operand. This  opcode
  requires that the source be an 8-bit immediate value. The value is
  sign-extended  before  loading  it  as  a  32-bit  number into the
  specified data register.

  Immediate Data -> Destination


Assembler Syntax
----------------
  MOVEQ{.L} #<data>,Dn


Addressing Modes
----------------
  Mode              Source  Destination

  Data Register Direct      - -
  Address Register Direct     - -
  Address Register Indirect   - -
  Postincrement Register Indirect   - -
  Predecrement Register Indirect    - -
  Register Indirect with Offset   - -
  Register Indirect with Index    - -
  Absolute Short        - -
  Absolute Long       - -
```

```
  P.C. Relative with Offset   - -
  P.C. Relative with Index    - -
  Immediate       - -


Data Size
---------
  Long


Status Flags
------------
  N  Set if negative
  Z  Set if zero
  V  Always cleared
  C  Always cleared
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
      # p
  Long    2 4

  # = no. of instruction bytes
  p = no. of instruction clock periods
```

1.12 AmigaFlight® Help: Push Effective Address

```
PEA Push Effective Address
==========================

  Push the calculated (effective) address onto the stack.

  Destination  -> SP@-


Assembler Syntax
----------------
  PEA{.L}   <ea>

  <ea> - control


Addressing Modes
----------------
  Mode             Source  Destination

  Data Register Direct      - -
  Address Register Direct     - -
  Address Register Indirect   * -
  Postincrement Register Indirect   - -
  Predecrement Register Indirect    - -
  Register Indirect with Offset   * -
  Register Indirect with Index    * -
```

```
  Absolute Short        * -
  Absolute Long       * -
  P.C. Relative with Offset   - -
  P.C. Relative with Index    - -
  Immediate       - -


Data Size
---------
  Long


Status Flags
------------
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
  <ea>    # p
  (An)    2 14
  d16(An)   4 18
  d8(An,Ri) 4 22
  Abs short 4 18
  Abs long  6 22
  d16(PC)   4 18
  d8(PC,Ri) 4 22

  # = no. of program bytes
  p = no. of instruction clock periods
```

1.13 AmigaFlight® Help: Swap Data Register Halves

```
SWAP  Swap Data Register Halves
===============================

  Exchange the upper 16 bits  of the  destination data register with
  the lower 16 bits of the same register. Store  the result  in  the
  destination register.

  Register b31..16 <-> Register b15..b0


Assembler Syntax
----------------
  SWAP{.W}  Dn


Data Size
---------
  Word
```

```
Status Flags
------------
  N  Set if bit 31 of result is set, else cleared
  Z  Set if result = 0, else cleared
  V  Always cleared
  C  Always cleared
  X  Not affected

  No.of program bytes: 2
  No. of instruction clock periods: 4
```

1.14 AmigaFlight® Help: Unlink

```
UNLK  Unlink
==============

  Load the stack pointer from the destination address register, then
  pop the long value  from the new top of  the stack and place it in
  the destination  register. This  instruction is  commonly  used at
  subroutine  exit to  restore an old frame  pointer and free up any
  local temporary storage.

  An -> SP : SP@+ -> An


Assembler Syntax
----------------
  UNLK  <ea>


Data Size
---------
  Unsized


Status Flags
------------
  N  Not affected
  Z  Not affected
  V  Not affected
  C  Not affected
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
       # p
  Unsized   2 34

  # = no. of program bytes
  p = no. of instruction clock periods
```