AmigaFlight System Control Instructions

Andrew Duffy Morris

COLLABORATORS

	TITLE : AmigaFlight System Control Instructions		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Andrew Duffy Morris	July 20, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 AmigaFlight System Control Instructions 1

1.1 AmigaFlight® Help: System Control Instructions 1

1.2 Check register against bounds 2

1.3 AmigaFlight® Help: Illegal 3

1.4 MOVE to Status Register 3

1.5 MOVE User Stack Pointer 5

1.6 Reset External Devices 5

1.7 Return from Exception 6

1.8 Load Status Register and Stop 7

1.9 Trap 8

1.10 Trap on Overflow 8

1.11 AmigaFlight® Help: No operation 9

Chapter 1

AmigaFlight System Control Instructions

1.1 AmigaFlight® Help: System Control Instructions

```
System Control Instructions
===========================

System   control   operations   are   accomplished  by   using   privileged
instructions,  trap generating instructions,   and instructions that use or
modify the status register.


  Privileged instructions
  -----------------------
  ANDI #d,SR        AND Immediate with Status Register
  EORI #d,SR        Exclusive or Immediate with Status Register
  MOVE <ea>,SR      Move to Status Register
  MOVE USP          Move User Stack Pointer
  ORI #d,SR         Logical Or Immediate with Status Register
  RESET             Reset External Devices
  RTE               Return from Exception
  STOP              Load Status Register and Stop

  Trap generating instructions
  ----------------------------
  CHK <ea>,Dn       Check register against bounds
  TRAP #n           Trap
  TRAPV             Trap on Overflow

  Non privileged status register operations
  -----------------------------------------
  ANDI #d,CCR       AND Immediate with Condition Codes
  EORI #d,CCR       Exclusive or Immediate with Condition Codes
  MOVE <ea>,CCR     Move to Condition Codes
  MOVE SR,<ea>      Move from Status Register
  ORI #d,CCR        Logical Or Immediate with Condition Codes

  Miscellaneous operations
  ------------------------
  NOP               No Operation
  ILLEGAL           Illegal Operation
```

1.2 Check register against bounds

```
CHK Check register against bounds
=====================================

  The contents of the  specified data  register  are compared to the
  upper  bound  effective  address and 0. If  the value of the  data
  register  is not  between 0  and the  upper bounds,  the processor
  initiates exception processing. The CHK instruction vector is used
  as the address to continue processing.

  If Dn < 0 or Dn > (<ea>) then TRAP


Assembler Syntax
----------------
  CHK{.W}   <ea>,Dn

  <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


Status Flags
------------
  N  Set if Dn < 0, clear if Dn > (<ea>) else undefined
  Z  Undefined
  V  Undefined
  C  Undefined
  X  Not affected


Instruction Size and Cycles to Execute
--------------------------------------
      Trap    No Trap
  <ea>     #    p    #    p
  Dn    2   <40   2   10
  (An)    2   <44   2   14
  (An)+   2   <44   2   14
  -(An)   2   <46   2   16
  d16(An)   4   <48   4   18
  d8(An,Ri) 4   <50   4   20
  Abs short 4   <48   4   18
  Abs long  6   <52   6   22
  d16(PC)   4   <48   4   18
  d8(PC,Ri) 4   <50   4   20
  Immediate 4   <44   4   14

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

1.3 AmigaFlight® Help: Illegal

```
ILLEGAL Illegal
===============

  This instruction will  always  generate  an  illegal   instruction
  exception.


Assembler Syntax
----------------
  ILLEGAL


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


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

1.4 MOVE to Status Register

```
MOVE_SR   MOVE to Status Register
=======================================

  Copy  the  source  operand  to  the  Status Register.

  Source -> SR
```

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

  <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


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

  This is a privileged instruction


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
```

1.5 MOVE User Stack Pointer

```
MOVE_USP  MOVE User Stack Pointer
=======================================

  Copy  the  User Stack Pointer to the destination operand, or  copy
  from the source operand to the User Stack Pointer.

  USP -> An
  An  -> USP


Assembler Syntax
----------------
  MOVE{.L}  USP,An
  MOVE{.L}  An,USP


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
  Word    2 4

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

  This is a privileged instruction
```

1.6 Reset External Devices

```
RESET Reset External Devices
==============================

  The reset line on the processor is  asserted, causing all external
  devices  to  be  reset.  This  instruction  does  not  affect  the
  processor  state  other  than  to  update  the  program counter to
  continue execution at the next instruction.


Assembler Syntax
----------------
  RESET


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   2 132

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

  This is a privileged instruction
```

1.7 Return from Exception

```
RTE Return from Exception
=============================

  Load the exception state  information  from  the  top of stack and
  continue  with  execution. This  instruction  reloads  the  status
  register  stack  pointer  and  program counter  in the appropriate
  manner for  the chip, and  continues  execution at the old program
  counter address.

  SP@+ -> SR : SP@+ -> PC


Assembler Syntax
----------------
  RTE


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

```
Status Flags
------------
  Set according to word on stack


Instruction Size nnd Cycles to Execute
--------------------------------------
      # p
  Unsized   2 20

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

  This is a privileged instruction
```

1.8 Load Status Register and Stop

```
STOP  Load Status Register and Stop
=====================================

  Load  the  immediate  data  into the  status register, advance the
  program   counter  to  the  next   instruction,   and   make   the
  microprocessor pause. The processor resumes executing instructions
  when a trace, interrupt request  or  reset execption is initiated.
  If an interrupt  request arrives whose priority is higher than the
  current   processor   priority,  an  interrupt  exception  occurs;
  otherwise the interrupt request has no effect.

  Immediate operand -> SR

  Wait for trace, interrupt or reset exception to occur


Assembler Syntax
----------------
  STOP  #<data16>


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


Status Flags
------------
  Set according to immediate operand


Instruction Size and Cycles to Execute
--------------------------------------
      # p
  Unsized   4 4

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

  This is a privileged instruction
```

1.9 Trap

```
TRAP  Trap
============

  Initiates exception processing. The program counter is incremented
  to the next instruction, then saved on  the system stack, followed
  by the current contents of the status register. Program  execution
  then continues at  an address obtained from  the exception  vector
  table.

  PC -> SSP@- : SR -> SSP@- :

  (Vector) -> PC


Assembler Syntax
----------------
  TRAP  #<vector>

  where <vector> is a 4 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
--------------------------------------
      # p
  Unsized   2 34

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

1.10 Trap on Overflow

```
TRAPV Trap on Overflow
========================

  This instruction will initiate exception  processing if the V flag
  is set when it executed.

  If V = 1 then TRAP


Assembler Syntax
----------------
  TRAPV


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
  Trap Taken  2 34
  Trap Not Taken  2 4

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

1.11 AmigaFlight® Help: No operation

```
NOP No operation
====================

  This instruction does not affect the processor state other than to
  update the program counter  to  continue  execution  at  the  next
  instruction.
  It  can be used to replace instruction  that are no longer needed,
  without  having  to recompute  displacements, to produce a precise
  time delay, or to temporarily replace instructions you do not want
  to  execute  when  debugging.  It  is  rarely  found  in  finished
  programs.


Assembler Syntax
----------------
```

```
  NOP


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 4

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