The Cinematronics CPU Programmer's Reference Guide

Version 1.0

Written by Zonn Moore

1. PREFACE 5

2. THE CINEMATRONICS CPU HARDWARE LAYOUT 5

2.1 MEMORY LAYOUT 5
2.1.1 Program Memory (ROM). *5*
2.1.2 Working Storage Memory (RAM). *6*
2.2 REGISTER OVERVIEW 6
2.2.1 User Accessible Registers *6*
2.2.2 Non-accessible Registers. *6*
2.3 REGISTERS 6
2.3.1 The 'A' and 'B' Accumulators *6*
2.3.2 Accessing the 'B' Register *7*
2.3.3 The 'P' Register *7*
2.3.4 The 'I' Register *7*
2.3.5 The 'J' register *8*
2.4 FLAGS 9
2.4.1 The 'EI' Flag *9*
2.4.2 The 'MI' Flag *9*
2.4.3 The 'DR' Flag *9*
2.4.4 The 'LT' Flag *9*
2.4.5 The 'EQ' Flag *9*
2.4.6 The 'NC' Flag *10*
2.4.7 The 'A0' Flag *10*

3. CINEMATRONICS INSTRUCTION SET OVERVIEW 10

3.1 ADDRESSING MODES 10
3.1.1 Implied *10*
3.1.2 Immediate *10*
3.1.3 Direct Page *10*
3.1.4 Indirect *11*
3.1.5 Table Lookup *11*
3.1.6 Input / Output *11*
3.2 FLAG MODES 11
3.2.1 Flags Common to All Instructions *11*
3.2.2 Flag Mode 1 ..... *12*
3.2.3 Flag Mode 2 ..... *12*
3.2.4 Flag Mode 3 ..... *12*
3.2.5 Flag Mode 4 ..... *13*
3.2.6 Flag Mode 5 ..... *13*
3.2.7 Flag Mode 6 ..... *13*
3.2.8 Flag Mode 7 ..... *13*
3.2.9 Flag Mode 8 ..... *14*
3.2.10 Flag Mode 9 ..... *14*
3.2.11 Flag Mode 10 ..... *14*
3.2.12 Flag Mode 11 ..... *15*
3.2.13 Flag Mode 12 ..... *15*
3.2.14 Flag Mode 13 ..... *15*
3.2.15 Flag Mode 14 ..... *16*
3.2.16 Flag Mode 15 ..... *16*
3.2.17 Flag Mode 16 ..... *16*
3.2.18 Flag Mode 17 ..... *16*
3.3 INSTRUCTION SET SUMMARY 17
3.3.1 Cinematronics Instruction Set *18*

4. ON BOARD I/O 21

4.1 INPUTS 21
4.2 OUTPUTS 21

5. THE CINEMATRONICS VECTOR GENERATOR 22

5.1 COLORS AND INTENSITIES 23
5.1.1 The Bi-Level display 23
5.1.2 Multiple Intensity (Gray Scale) Displays and Color Displays 23
5.1.2.1 Using the 'X' Register to set Colors / Intensities 23
5.1.2.2 The 16 Level Gray Scale Display 24
5.1.2.3 The 64 Level Gray Scale Display 24
5.1.2.4 The Color Display 25
5.2 VECTORS 25
5.2.1 Screen Resolution 25
5.2.2 Drawing a Vector 25

1. Preface

This information was gathered solely through the use of the program 'CINESIM.EXE' and few digital data books, (and well of course, a schematic of the Cinematronics CPU.)

My hope is this document helps in Cinematronics CPU repair / program hacking / diagnostic programs, etc.

The instruction set vaguely reminds me of the AB = D register of the 6803, and 6809 CPUs. And it seemed that a Motorola type assembler would be easiest to implement on this processor, so a Motorola type syntax is what I used to describe the instructions.

For those not familiar with this syntax:

Hexadecimal values are prefixed with a '$'.

All immediate values are prefixed with a '#', otherwise Direct Page is assumed.

2. The Cinematronics CPU Hardware Layout

The Cinematronics CPU is a Harvard Architecture design. It runs on a four phase 5 mhz clock, doing 99% of it's clocking on phase 3. It consists of two 12 bit accumulators, 24 input lines, 8 output lines, and 24 bits of vector position outputs. Also included is a frame rate timer / watchdog timer.

The standard unmodified (Rev. K) board uses 8k of ROM and 256 12-bit words of RAM.

ROM memory cycles are limited to a minimum of three cycles per access, allowing for use of 600ns (worst case) memory. This allowed the use of 450 to 500ns memories that were readily available at the time of the CPU's design.

While the ROMs are accessed 16 bits at a time, the ROM (or instruction) data bus is only 8 bits wide. 8 bits of the ROM is latched for later use in a 1 level deep pipeline, this allows for better throughput, while still maintaining a 600ns memory cycle time.

The RAM data bus is 12 bits wide.

The Cinematronics CPU consists of five software accessible registers, and three non-accessible registers.

2.1 Memory Layout

2.1.1 Program Memory (ROM).

The program (ROM) consists of 2 to 8 4096 byte banks, depending upon board configurations.

All banks start at address 000h and run through address FFFh.

ROM Banking is accomplished by loading the 'P' register with the new bank address and then executing a 'JPP' instruction. Program control will be transferred to the bank given in the 'P' register at the address given in the 'J' register.

In a 8k boards (Star Castle) the 'P' register values are:

01 = Bank 0
02 = Bank 1

For 16k boards (Solar Quest) the 'P' register values are:

00 = Bank 0
01 = Bank 1
02 = Bank 2
03 = Bank 3

For 32k boards (Boxing Bugs) the 'P' register value are:

00 = Bank 0

01 = Bank 1
02 = Bank 2
03 = Bank 3
04 = Bank 4
05 = Bank 5
06 = Bank 6
07 = Bank 7

The mapping of ROM chips to memory is a bit unusual, and also changes per board configurations.

The 'normal' EPROM configurations is that even address are all contained in sockets T7 and U7, and odd addresses are in sockets P7 and R7. Bank 0 usually starts in sockets T7 and P7, then continues through sockets U7 and R7.

Because PROMs can have their address/selection lines inverted from the normal EPROM configuration, these can have their start address in either the T7/P7 or U7/R7 pair. I've seen both.

2.1.2 Working Storage Memory (RAM).

The CPU's RAM is laid out in 16 pages of 16 registers, for 256 12 bit RAM addresses.

Page 0 contains RAM addresses 0-15, page 1 contains RAM addresses 16-31, …, page 15 contains RAM addresses 240-255.

When using 'DIR' addressing, the upper 'PAGE' value is read from the 'P' register, while the lower 4 bits are given as part of the opcode. To access absolute RAM location $34, which is RAM location 4 on page 3, one would:

```
ldp       #3        ; Load 'P' register to point to page 3
lda       $4        ; Load accumulator 'A' with the value at RAM location $34
```

When using the 'IND' addressing mode, then 'P' is not used. The value read is the value pointed at by the 8 bits in the 'I' register.

2.2 Register Overview

2.2.1 User Accessible Registers

A = Primary accumulator. (12 bits.)
B = Secondary accumulator. (12 bits.)
P = Page register. Holds RAM page, and 'Jump to new ROM bank' value. (4 bits.)
I = Indirection register. Points to last accessed RAM location. (8 bits.)
J = Jump register. Holds the destination of a 'JMP' instruction. (12 bits.)

2.2.2 Non-accessible Registers

There are two 12 bit line draw registers and a vector timer register that are not directly accessible, though can be set through the use of the vector instructions.

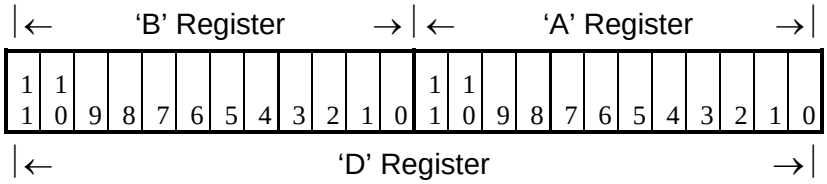
X = X position of either the start or end of a vector. (12 bits.)
Y = Y position of either the start or end of a vector. (12 bits.)
T = Timer used to time the length of a vector draw. (12 bits.)

2.3 Registers

2.3.1 The 'A' and 'B' Accumulators

The CPU has two 12 bit accumulators, referred to as registers or accumulators 'A' and 'B'.

A 'D' register is also referenced by the instruction set. The 'D' register is not a separate register, but the concatenation of the 'A' and 'B' registers. They are linked to form the 'D' register. The 'B' register becomes the high order 12 bits, and the 'A' register becomes the low order 12 bits.



The linkage is not perfect. The low order bit of the 'B' register can be shifted into the high order bit of the 'A' register using an 'ASRD' (Arithmetic Shift Right 'D' register) instruction. On the other hand the high order bit of the 'A' register can not be shifted into the low order bit of the 'B' register – there is no "LSLD" (Logical Shift Left 'D' register) instruction. There is however a 'SLAB' (Shift Left 'A' and 'B' registers) in which both the 'A' and 'B' registers are left shifted independent of one another.

There are other idiosyncrasies in the register hardware.

The 'B' register cannot be logically shifted right. When the 'B' register is shifted right (using either the 'ASR' or 'LSR' instruction), the sign bit will always be extended into the shifted register.

The 'A' register can be logically or arithmetically shifted. An 'ASR' (Arithmetic Shift Right) instruction when performed on the 'A' register, will do a sign extended right shift. A 'LSR' (Logical Shift Right) instruction will cause the 'A' register to be shifted right with a zero being shifted into bit-11.

The B and A registers combined hold the results of the 'MUL' instruction. The 'MUL' instruction performs one shift/add cycle of a full binary multiply algorithm.

2.3.2 Accessing the 'B' Register

The 'B' accumulator is accessed through a 'prefix' opcode. To access the 'B' register, a prefix instruction 'SSA' (Set Secondary Accumulator) must be executed and followed by any accumulator related instruction. The 'SSA' prefix has no effect on non-accumulator instructions.

Any instruction executed after an 'SSA' instruction (including another 'SSA' instruction) causes the CPU to default back to the primary 'A' register.

The 'SSA' instruction is the standard way to access the 'B' register. However any jump instruction with its bit-3 cleared will cause the next instruction to use 'B' accumulator.

NOTE: The 'JPP' instruction has its bit-3 clear, and as such *will* cause the next instruction to use the 'B' accumulator. Unless this is desired by the programmer the 'JPP' instruction should normally jump to a 'NOP' instruction which will cause the CPU to default back to the 'A' accumulator.

2.3.3 The 'P' Register

The 'P' register is used to hold the current RAM memory page, and the ROM banked to be jumped to via the 'JPP' instruction. (See *Memory layout – Program Memory*.)

The 'P' can only be loaded using the load immediate instruction 'LDP #x'. (See RAM memory layout for the values the 'P' register can hold.)

2.3.4 The 'I' Register

The 'I' register is a special register that can only be used as an indirect address.

It is always loaded by the use of any DIR address mode instructions, and continues to point to the last RAM location addressed until a new 'DIR' address mode instruction is executed. The 'I' register is an 8 bit register and holds both the 4 bit page address and the 4 bit RAM address of the last RAM location accessed. For example:

```
ldp       #$2       ; Point to page 2 of RAM
lda       $7        ; Read RAM location $27 into 'A', loads 'I' with $27
add       #1        ; Increment the 'A' register
sta       [i]       ; Write new value back to address $27
ldp       #$3       ; Point to a new page
add       #1        ; Increment the 'A' register again
sta       [i]       ; Write back address is still $27
```

The 'I' register can also be loaded from RAM, by reading the value pointed to by the 'I' register, back into the 'I' register allowing for the indirect addressing of RAM. For example:

```
; Routine for storing the 'B' register at the address in RAM location $F5

ldp       #$F       ; Point to page $F of RAM
ldi       #$5       ; Set 'I' register to point to $F5
ldi       [i]       ; Read value at $F5 into 'I' register
ssa                 ; Use the secondary accumulator ('B' register).
sta       [i]       ; Store accumulator 'B' at RAM pointed to by RAM
                    ; location $F5
```

2.3.5 The 'J' register

The CPU's jump opcodes do not, in themselves, contain the destination addresses of the jumps. Instead all jumps jump to the address contained in the 'J' register, which must be loaded previous to the execution of a jump instruction. For example:

```
; Jump to location $125

ldj       #$125     ; Load 'J' register with destination address
jmp                 ; Jump to address $125
```

Conditional jumps either jump to the value in 'J' or skip to the next instruction, depending upon the value of the flag being tested. For example:

```
; Jump if the 'ADD' instruction does not overflow...

ldj       #$372     ; point to location $372
ldp       #0        ; Point to page 0
lda       $3        ; Get value at $03
add       #$1A      ; Add $1A to value
jnc                 ; If no carry, jump to $372

; Else fall through to next instruction...
```

The CPU hardware does not incorporate a stack, and there is no 'CALL' instruction, yet an effective subroutine can still be written, if the proper use of RAM is enforced. This is possible using the 'J' register, which can be loaded from RAM indirectly through the 'I' register. Assuming RAM location $00 is used to store the "return" address, a subroutine that adds 5 to RAM location $03 can be written as:

```
addfive:  clr                 ; Clear the 'A' register
          add       #5        ; Load 'A' register with 5
          ldp       #0        ; Point to page 0
          add       $3        ; Add value at location $3 to the accumulator
          sta       [i]       ; Write new value back to location $3
          ldi       #$0       ; Load the 'I' register with address $00
          ldj       [i]       ; Load the return address into the 'J' register
          jmp                 ; Return to calling routine
```

And a call to the above routine could be written as:

```
          lda       #retadr & $F00      ; Load 'A' reg with high nibble of 'retadr'
          add       #retadr & $0FF      ; Load 'A' low byte if 'retadr'
          ldp       #0                  ; Point to page 0
          sta       $0                  ; Set $00 to return address
          ldj       #addfive            ; Point to subroutine
          jmp                           ; Jump to the subroutine
retadr:   ...                           ; Subroutine will return here
```

The 'P' can be combined with the 'J' to create an address that extends beyond the 12-bits used in one ROM bank. Once program is running in a new bank, all jumps become local to that bank. For example:

```
; Jump to address $450 on ROM bank 2 (on a Solar Quest board)
ldp       #2        ; Point to Bank 2
ldj       #$450     ; Point to address $450
jpp                 ; Jump using 'P' as bank extension to $2450
```

2.4 Flags

There are six software accessible flags:

EI = External Input (If the 'JMI' is *not* installed on the CPU board.)
MI = Minus flag. (If 'JMI' jumper *is* installed on the CPU board.)
DR = Drawing flag. Set if a vector is currently being drawn.
LT = Less than flag. Compared value was less than the Accumulator.
EQ = Equal to flag. Compared value was equal to the Accumulator.
NC = Not Carry flag. Set if last arithmetic function did *not* generate a carry
A0 = Bit-0 of the 'A' register was a '1' before last access of either accumulator.

2.4.1 The 'EI' Flag

If 'JMI' is *not* installed, then the EI flag is set depending upon the External Input on pin 10 of J4. If this pin is High, the jump will be taken. If Low, program execution continues with the next instruction following the 'JEI'.

2.4.2 The 'MI' Flag

The original CPU hardware did not have a 'MI' flag. This flag becomes available when a jumper is installed from bit-11 of the working accumulator, to the External Input bit. (This jumper is labeled 'JMI' on Rev. K boards). Since the external input is read through a latch, there is a delay of one instruction before the newly named 'MI' flag can be tested. For example:

```
ldj       #$125     ; Point to location $125
lda       #$800     ; Set 'A' register to $800
sub       #1        ; subtract 1
nop                 ; At this point the MI flag is set from the 'LDA'
jmi                 ; The MI flag is reset from the 'SUB #1' instruction
                    ; at this point, and this jmp is not taken.
```

2.4.3 The 'DR' Flag

This flag indicates a vector is currently being drawn. This flag should be checked before each 'VIN' instruction to verify the previous vector is finished being drawn. This flag is not set until 11 cycles after the 'VDR' instruction is executed, therefore an immediate test of this flag would falsely show that a vector is not being drawn.

2.4.4 The 'LT' Flag

This flag is set/reset every time an accumulator instruction is executed. For many of the instructions this flag is rather meaningless. When checked after a 'CMP', it indicates the value compared with the accumulator is less than value of the accumulator. An unsigned comparison is performed.

2.4.5 The 'EQ' Flag

This flag is set/reset every time an accumulator instruction is executed. For many of the instructions this flag is rather meaningless. When checked after a 'CMP', it indicates the value compared with the accumulator is equal to value of the accumulator.

2.4.6 The 'NC' Flag

This flag is set/reset every time an accumulator instruction is executed. For many of the instructions this flag is rather meaningless. When checked after any accumulator arithmetic instruction, it indicates the arithmetic instruction did not generate a Carry for an 'ADD' instruction or a borrow for a 'SUB' instruction.

Subtract instructions are performed using an 1's compliment addition, plus 1. This might not always set the C-flag as one would expect. For example:

```
ldj       #$125     ; Point to address $125
clr                 ; Zero the 'A' register
sub       #1        ; Subtract 1 from 'A' register
jnc                 ; Program jumps to address $125 even though
                    ; logically, a borrow should have occurred
```

The value in the 'A' register will be $FFF as expected, yet the carry flag (indicating a borrow in this case) will not be set.

What was actually performed by the above code is: The value $FFE (one's compliment of 1) is added to 0. Then to create a two's compliment value, the value is incrementing by one, resulting in $FFF, since no overflow is generated by the addition, the carry flag is not set.

This flag is strictly a Carry/Borrow flag and is *never* set by the shifting out of a registers bit during any shift instruction. You cannot, for instance, shift left the 'A' register and use the 'NC' flag to check for any shifted out bits.

2.4.7 The 'A0' Flag

This flag is set/reset every time a accumulator instruction is executed. For many of the instructions this flag is rather meaningless. When checked after any accumulator instruction it reflects the value of the 'A' accumulator's bit-0 prior to the execution of the instruction. Bit-0 of the 'A' register is always the bit tested, regardless of whether the 'A' or 'B' register was accessed.

3. Cinematronics Instruction Set Overview

3.1 Addressing Modes

The CPU has its instruction divided into six addressing modes:

IMP = Implied.
IMM = Immediate.
DIR = Direct Page, using the 'P' register.
IND = Indirect through the 'I' register.
LKP = Lookup table data in ROM.
I/O = Read or set an I/O bit.

3.1.1 Implied

An action performed on an accumulator that needs no other data, for example the shift instructions.

3.1.2 Immediate

A 4 bit value as part of the instructions opcode or an 8 bit value following the instruction can be loaded, added to, or subtracted from the accumulators.

3.1.3 Direct Page

A 4 bit value is given as part of the instruction's opcode, this is used as the lower 4 bits of the memory's address. The upper 4 bits are taken from the 'P' register. Together they make up the 8 bit address of a RAM location. The contents of this RAM location can be written to, or read. Or the contents of RAM can added to, or subtracted from, the selected accumulator.

3.1.4 Indirect

The 'I' register is an 8 bit register that holds the 8 bit address of the last RAM location accessed. Memory can be read/written using the value in the 'I' register as a pointer to RAM.

3.1.5 Table Lookup

The Table Lookup instructions consists of only the 'LKP' instruction. The value in the accumulator is used as a 12 bit address into the current ROM bank. The ROM value at the given address is then loaded into the accumulator.

3.1.6 Input / Output

The 'INP' instruction is used to read the control panel switches. A single input bit is pointed to by the lower 4 bits of the 'INP' opcode. The bit is read into the accumulator's bit 0. All other bits of the accumulator are set to zero.

There are also 8 configuration switches. These are read by using the secondary 'B' accumulator. The indicated switch will be read into bit-0 of the 'B' registers, all other bits of the 'B' register will be set to zero.

All configuration switches and control panel switches are normally high, and go low when the configuration switch is switched on, or the control panel switch is pressed.

The 'OUT' instruction sets the output line referenced by the lower 4 bits of the 'OUT' opcode, to the *inverted* value of bit-0 of the accumulator. Accumulator 'A' or 'B' may be used.

3.2 Flag Modes

The following are definitions are used to describe the actions of the flags:

'n' = A 12 bit immediate value that is created by using the lower 4 bits of the current opcode as the lower 4 bits of the value, and zeroing the upper 8 bits.
'n00' = A 12 bit immediate value that is created by using the lower 4 bits of the current opcode as the upper 4 bits of the value and zeroing the lower 8 bits.
'mm' = A 12 bit immediate value that is created by using the byte following the current opcode as the lower 8 bits of the value and zeroing the upper 4 bits.
'Pn' = The 12 bit value accessed when using the 'DIR' addressing mode. (See the 'DIR' addressing mode.)
'[i]' = The 12 bit value accessed when using the 'IND' addressing mode. (See the 'IND' addressing mode.)
'bab' = A 12 bit value created by using the current opcode where 'a' is the upper 4 bits of the opcode, and 'b' is the lower 4 bits. For example the opcode $E5 would create the 'bab' value of: $5E5.
'ttt' = a 12 bit value created by using the 8 bit value stored at the ROM address given by the selected Acc, as the lower 8 bits of the value and zeroing the upper 4 bits. (See the 'LKP' addressing mode.)
'b' = The 12 bit value read as the result of an 'INP' instruction. Bit-0 is set to the input line's current value, the upper 11 bits are zeroed.
~ = Ones compliment. The inversion of all bits, the same as exclusive or'ing with all ones.
Acc = Either the 'A' or 'B' accumulator.

The expression 'original value' refers to the value of an accumulator *before* the opcode was executed.

Every instruction will fit into one of the following categories as indicated in the *"Cinematronics Instruction Set Summary"* table under the *'Flags'* column.

3.2.1 Flags Common to All Instructions

All opcodes treat these flags in an identical manner. After any opcode execution, the following flags will always be set as:

EI ← The current state of the External Input. (Assumes 'JMI' jumper is *not* present.)
MI ← Bit-11 of the selected acc. (Assumes 'JMI' jumper *is* present. Flag will be delayed by one instruction.)
DR ← The 'drawing' state of the vector generator. (Set if vector being drawn.)

The 'EI' flag is set to the current state of the external input latch assuming the 'JMI' jumper is *not* installed.

The 'MI' flag assumes the 'JMI' jumper *is* installed. The 'MI' flag is set to the selected accumulator's bit-11 through a latch and will be readable only *after* the next instruction is executed. A 'NOP' instruction is usually placed before any 'JMI' instruction to allow the 'JMI' instruction to properly test the 'MI' flag. In reality, any instruction can be placed between the 'JMI' instruction and the instruction being tested, and still allow the 'MI' flag to be properly tested, a 'NOP' simply leads to less confusion. If the instruction setting the 'MI' flag accesses the 'B' register (the instruction was proceeded with a 'SSA' instruction) , then the 'B' registers bit-11 will placed into the 'MI' flag. Otherwise the 'A' register's bit-11 will be used as the next 'MI' flag.

The 'DR' flag is set 11 cycles after the execution of the 'VDR' instruction. It will remain set until the vector is finished being drawn.

3.2.2 Flag Mode 1

The flags are set as follows:

NC ← Unchanged
A0 ← Unchanged
LT ← Unchanged
EQ ← Unchanged

The flags 'NC', 'A0', 'LT', and 'EQ' are unchanged by this instruction.

3.2.3 Flag Mode 2

The flags are set as follows:

NC ← Result of ('n' + Acc) addition.
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('n' < Acc) test.
EQ ← Result of ('n' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and the immediate value 'n'. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'n' and the selected accumulator. The value 'n' is compared to the value in the selected accumulator. If the value 'n' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.4 Flag Mode 3

The flags are set as follows:

NC ← Result of ('mm' + Acc) addition.
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('mm' < Acc) test.
EQ ← Result of ('mm' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and the immediate value 'mm'. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'mm' and the selected accumulator. The value 'mm' is compared to the value in the selected accumulator. If the value 'mm' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.5 Flag Mode 4

The flags are set as follows:

NC ← Result of ('Pn' + Acc) addition.
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('Pn' < Acc) test.
EQ ← Result of ('Pn' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and 'Pn'. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'Pn' and the selected accumulator. The value 'Pn' is compared to the value in the selected accumulator. If the value 'Pn' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.6 Flag Mode 5

The flags are set as follows:

NC ← Result of ('[i] + Acc) addition.
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('[i]' < Acc) test.
EQ ← Result of ('[i]' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and '[i]'. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between '[i]' and the selected accumulator. The value '[i]' is compared to the value in the selected accumulator. If the value '[i]' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.7 Flag Mode 6

The flags are set as follows:

NC ← Result of ('bab' + Acc) addition.
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('bab' < Acc) test.
EQ ← Result of ('bab' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and 'bab'. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'bab' and the selected accumulator. The value 'bab' is compared to the value in the selected accumulator. If the value 'bab' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.8 Flag Mode 7

The flags are set as follows:

NC ← Result of (~'n' + Acc + 1) addition.

A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('n' < Acc) test.
EQ ← Result of ('n' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and the one's compliment of the immediate value 'n', plus one. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'n' and the selected accumulator. The value 'n' is compared to the value in the selected accumulator. If the value 'n' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.9 Flag Mode 8

The flags are set as follows:

NC ← Result of (~'mm' + Acc + 1) addition.
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('mm' < Acc) test.
EQ ← Result of ('mm' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and the ones compliment of the immediate value 'mm', plus one. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'mm' and the selected accumulator. The value 'mm' is compared to the value in the selected accumulator. If the value 'mm' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.10 Flag Mode 9

The flags are set as follows:

NC ← Result of (~'Pn' + Acc + 1) addition.
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('Pn' < Acc) test.
EQ ← Result of ('Pn' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and the ones compliment of 'Pn', plus one. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between the selected accumulator and 'Pn'. The value 'Pn' is compared to the value in the selected accumulator. If the value in the selected accumulator is less than 'Pn', the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.11 Flag Mode 10

The flags are set as follows:

NC ← Result of (~'[i]' + Acc + 1) addition.
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('[i]' < Acc) test.
EQ ← Result of ('[i]' = Acc) test.

The 'NC' flag is set as the result of the addition of the selected accumulator's original value and the ones compliment of '[i]', plus one. The flag will be *set* if no overflow occurred during the addition.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between '[i]' and the selected accumulator. The value '[i]' is compared to the value in the selected accumulator. If the value '[i]' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.12 Flag Mode 11

The flags are set as follows:

NC ← 1
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('n00' < Acc) test.
EQ ← Result of ('n00' = Acc) test.

The 'NC' flag is always set by this instruction.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'n00' and the selected accumulator. The value 'n00' is compared to the value in the selected accumulator. If the value 'n00' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.13 Flag Mode 12

The flags are set as follows:

NC ← 1
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('Pn' < Acc) test.
EQ ← Result of ('Pn' = Acc) test.

The 'NC' flag is always set by this instruction.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'Pn' and the selected accumulator. The value 'Pn' is compared to the value in the selected accumulator. If the value 'Pn' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.14 Flag Mode 13

The flags are set as follows:

NC ← 1
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('[i]' < Acc) test.
EQ ← Result of ('[i]' = Acc) test.

The 'NC' flag is always set by this instruction.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between '[i]' and the selected accumulator. The value '[i]' is compared to the value in the selected accumulator. If the value '[i]' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.15 Flag Mode 14

The flags are set as follows:

NC ← 1
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('bab' < Acc) test.
EQ ← Result of ('bab' = Acc) test.

The 'NC' flag is always set by this instruction.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'bab' and the selected accumulator. The value 'bab' is compared to the value in the selected accumulator. If the value 'bab' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.16 Flag Mode 15

The flags are set as follows:

NC ← 1
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('ttt' < Acc) test.
EQ ← Result of ('ttt' = Acc) test.

The 'NC' flag is always set by this instruction.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'ttt' and the selected accumulator. The value 'ttt' is compared to the value in the selected accumulator. If the value 'ttt' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.17 Flag Mode 16

The flags are set as follows:

NC ← 1
A0 ← Bit-0 of the original value of the 'A' register.
LT ← Result of ('b' < Acc) test.
EQ ← Result of ('b' = Acc) test.

The 'NC' flag is always set by this instruction.

The 'A0' flag is set to the value of the 'A' register's bit-0 *before* the instruction is executed. The 'A' register's bit-0 is used, regardless of which accumulator is being accessed by this instruction.

The 'LT' and 'EQ' flags are set as the result of an unsigned comparison between 'b' and the selected accumulator. The value 'b' is compared to the value in the selected accumulator. If the value 'b' is less than the value in the selected accumulator, the 'LT' flag will be set. If the values are equal, the 'EQ' flag will be set.

3.2.18 Flag Mode 17

The flags are set as follows:

NC ← Unchanged
A0 ← Unchanged
LT ← Unchanged
EQ ← Unchanged

DR ← Set 11 cycles after execution of 'VDR' instruction.

The flags 'NC', 'A0', 'LT', and 'EQ' are unchanged by this instruction.

The 'DR' flag will be set 11 cycles after the execution of the 'VDR' instruction. It will remain set until the vector is finished being drawn. An attempt to branch on this flag before the 11 cycles will result in a flag reading false even though the vector is being drawn. This flag should be tested just before issuing a 'VIN' instruction to insure the vector generator is not actively drawing a vector.

3.3 Instruction Set Summary

The following is a short description of the instructions available on the Cinematronics CPU.

For each instruction the following are listed:

Opcode = The object code of the instruction.
Mnemonic = The assembly language mnemonic of the instruction.
Mode = The addressing mode of the instruction.
Flag = The flag mode of the instruction.
Cycles = The number of 5 mhz cycles needed to execute the instruction.
Notes = Possible reference notes. A list of notes follow the summary table.

Many of the opcodes between the E0-FF range are repeated. Both instructions perform the same function.

A wait state is added to an instruction fetch cycle anytime a ROM would be accessed faster than 600ns. Therefore two sequential single cycle instructions will cause a waitstate to be added before the fetch of the next instruction.

Each instruction will set flags according to its flag mode. Refer to *"Flag Modes"* for further information on flag settings.

3.3.1 Cinematronics Instruction Set

Opcode	Mnemonic	Mode	Flags	Cycles	Description	Notes
00	CLR	IMP	11	1~	Clear accumulator.	15
0n	LDA #n00	IMM	11	1~	Load immediate value 'n00' into accumulator.	1
1b	INP b	I/O	16	1~	Read input bit 'b' into bit-0 of accumulator.	
20 mm	ADD #mm	IMM	3	3~	Add immediate value 'mm' to the accumulator.	
2n	ADD #n	IMM	2	1~	Add immediate value 'n' to the accumulator.	13
30 mm	SUB #mm	IMM	8	3~	Subtract immediate value 'mm' from accumulator.	
3n	SUB #n	IMM	7	1~	Subtract immediate value 'n' from accumulator.	13
4z yx	LDJ #xyz	IMM	1	3~	Load immediate value 'xyz' into the 'J' register.	
50	JPP	IMP	1	4~	Jump using 'P' reg as bank and 'J' reg as destination.	2,12
51	JMIB	IMP	1	2~,4~	Jump if 'MI' (minus) flag is set.	2,12
51	JEIB	IMP	1	2~,4~	Jump if 'EI' (external input) flag is set.	2,12
52	JDRB	IMP	1	2~,4~	Jump if 'DR' (drawing vector) flag is set.	2,12
53	JLTB	IMP	1	2~,4~	Jump if 'LT' (less than) flag is set.	2,12
54	JEQB	IMP	1	2~,4~	Jump if 'EQ' (equal) flag is set.	2,12
55	JNCB	IMP	1	2~,4~	Jump if 'NC' (not carry) flag is set.	2,12
56	JA0B	IMP	1	2~,4~	Jump if 'A0' ('A' register, Bit-0) flag is set.	2,12
57	SSA	IMP	1	2~	Select secondary accumulator.	
58	JMP	IMP	1	4~	Jump using the 'J' register as the destination.	12
59	JMI	IMP	1	2~,4~	Jump if 'MI' (minus) flag is set.	12
59	JEI	IMP	1	2~,4~	Jump if 'EI' (external input) flag is set.	12
5A	JDR	IMP	1	2~,4~	Jump if 'DR' (drawing vector) flag is set.	12
5B	JLT	IMP	1	2~,4~	Jump if 'LT' (less than) flag is set.	12
5C	JEQ	IMP	1	2~,4~	Jump if 'EQ' (equal) flag is set.	12
5D	JNC	IMP	1	2~,4~	Jump if 'NC' (not carry) flag is set.	12
5E	JA0	IMP	1	2~,4~	Jump if 'A0' ('A' register, Bit-0) flag is set.	12
5F	NOP	IMP	1	2~	No operation.	
6n	ADD n	DIR	4	3~	Add RAM at 'Pn' to accumulator.	3
7n	SUB n	DIR	9	3~	Subtract RAM at 'Pn' from accumulator.	3
8n	LDP #n	IMM	1	1~	Load immediate value 'n' into the 'P' register.	3
9n	OUT b	I/O	1	1~	Set output pin 'b' to inversion of acc's bit-0.	
An	LDA n	DIR	12	3~	Load RAM at 'Pn' into accumulator.	3
Bn	CMP n	DIR	9	3~	Compare accumulator with RAM at 'Pn'.	3
Bn	LDI #n	IMM	9	3~	Load 'I' register with value 'Pn'.	11
Cn	LDI n	DIR	1	3~	Load RAM at 'Pn' into the 'I' register.	3
Dn	STA n	DIR	1	2~	Store accumulator into RAM at 'Pn'.	3
E0	VDR	IMP	17	1~	Set ending positions and draw vector.	4
E1	LDJ [I]	IND	1	2~	Load RAM at 'I' reg into the 'J' reg.	
E2 xx	LKP	LKP	15	7~	Load ROM data pointed to by acc. into the acc.	9
E3	MUL	IMP	5	2	'LSRD', then if 'A0' set, add RAM at 'I' reg to acc.	14
E4	NRM	IMP	1	n~	Load line length timer, normalize vector.	5
E5	FRM	IMP	1	n~	Wait for next frame.	6
E6	STA [I]	IND	1	2~	Store accumulator into RAM at 'I' register.	
E7	ADD [I]	IND	5	2~	Add RAM at 'I' to the accumulator.	
E8	SUB [I]	IND	10	2~	Subtract RAM at 'I' from the accumulator.	
E9	AND [I]	IND	13	2~	Logical 'AND' RAM at 'I' into the accumulator.	

EA	LDA [I]	IND	13	2~	Load RAM at 'I' into the accumulator.	
EB	LSR	IMP	6	1~	Do a logical shift right of the accumulator.	7
EC	LSL	IMP	6	1~	Do a logical shift left of the accumulator.	
ED	ASR	IMP	14	1~	Do an arithmetic shift right of the acc.	
EE	ASRD	IMP	6	1~	Do an arithmetic shift right of the 'D' reg.	
EF	LSLAB	IMP	6	1~	Do a logical shift left of the 'A' and 'B' regs.	8
F0	VIN	IMP	1	1~	Set the initial vector positions.	4
F1	LDJ [I]	DIR	1	2~	Load RAM at 'I' reg into the 'J' reg.	
F2 xx	LKP	LKP	15	7~	Load ROM data pointed to by acc. into the acc.	9
F3	MUL	IMP	5	2	'LSRD', if 'A0' set, add RAM at 'I' reg to acc.	
F4	NRM	IMP	1	n~	Load line length timer, normalize vector.	5
F5	FRM	IMP	1	n~	Wait for next frame.	6
F6	STA [I]	DIR	1	2~	Store accumulator into RAM at 'I' register.	
F7	WDG	IMP	5	2~	Reset the watchdog timer.	10
F8	SUB [I]	DIR	10	2~	Subtract RAM at 'I' from the accumulator.	
F9	AND [I]	DIR	13	2~	Logical 'AND' RAM at 'I' into the accumulator.	
FA	LDA [I]	DIR	13	2~	Load RAM at 'I' into the accumulator.	
FB	LSR	IMP	6	1~	Do a logical shift right of the accumulator.	7
FC	LSL	IMP	6	1~	Do a logical shift left of the accumulator.	
FD	ASR	IMP	14	1~	Do an arithmetic shift right of the acc.	
FE	ASRD	IMP	6	1~	Do an arithmetic shift right of the 'D' reg.	
FF	LSLAB	IMP	6	1~	Do a logical shift left of the 'A' and 'B' regs.	8

Note 1: The 4 bits of 'n' is loaded into the upper nibble of the accumulator. The 8 lower bits of the accumulator are always set to zero.

Note 2: This instruction will also select the secondary accumulator. If it is desired to use the 'A' register after this instruction, it is recommended to make the destination instruction a 'NOP' followed by the desired accumulator instruction.

Note 3: The address of the RAM location is formed by using 'n' as the lower 4 bits of the RAM address, and the 'P' register as the upper 4 bits.

Note 4: The 'A' register is loaded into the 'X' register, and the 'B' register is loaded into the 'Y' register.

Note 5: The number of cycles needed to execute this instruction is dependent upon the values loaded into the 'A' and 'B' registers. See instruction details for more information. If both the 'A' and 'B' registers contains all zeros, then the CPU will lockup and eventually reset when the watchdog timer goes off.

Note 6: The number of cycles needed to execute this instruction is variable. The CPU stops and waits for the next frame time. It is a requirement that a 'WDG' instruction be executed after a 'FRM' instruction is executed to keep the watchdog timer from timing out and causing a system reset.

Note 7: Do to hardware idiosyncrasies of the CPU, the 'B' register cannot be logically shifted right. The 'B' register will always be sign extended during both an 'ASR' and an 'LSR' instruction.

Note 8: The 'D' register cannot be shifted left as a whole unit. The 'LSLAB' instructions logically shifts the 'A' register and the 'B' register left as separate registers.

Note 9: Do to hardware idiosyncrasies of the CPU, the byte following the 'LKP' instruction is lost in the instruction fetch pipeline. Any byte following the 'LKP' instruction will always be ignored. It is therefore recommend that the 'LKP' instruction be followed by a 'NOP'.

Note 10: The 'WDG' instruction, along with resetting the watchdog timer, behaves exactly like the 'ADD [I]' instruction. Care should be taken to note that the accumulator will be modified by the execution of the 'WDG' instruction.

Note 11: The 'LDI #n' instruction is identical to the 'CMP n' instruction. The upper 4 bits of the 'I' register will loaded with the value of the 'P' register, and the lower 4 bits will be set to 'n'. The flags will also be set as if a 'CMP n' instruction were executed. Proper used of the 'LDI #n' instruction can help in the readability of source code.

Note 12: The number of cycles executed by this instruction depends on whether or not the branch is taken. The first cycle count is the number of cycles executed if the branch is not taken. The second cycle count is the number of cycles executed when the branch is taken.

Note 13: The operand value of 'n' must be a value between 1 and 15. A value of zero is not allowed.

Note 14: The 'MUL' instruction is only one shift/add instruction in the standard multiply algorithm. To do a full 12 x 12 bit multiply, twelve consecutive 'MUL's must be executed.

Note 15: The 'CLR' instruction is identical to the 'LDA #0' instruction, and can be used for clarity.

4. On Board I/O

4.1 Inputs

There are twenty-four software readable inputs on the CPU board. Twenty-one are routed to the control panel, and seven are routed to the on board configuration switches, and one is used to read the coin detect latch. Five of the inputs are shared between the configuration switches and the control panel. There is also the possibility of an 'External Input' pin, depending upon the 'JMI' jumper setting on the CPU board.

The switches are mapped as:

I0 = Control Panel (J3-11)
I1 = Control Panel (J3-10)
I2 = Control Panel (J3-9)
I3 = Control Panel (J3-8)
I4 = Control Panel (J3-15)
I5 = Control Panel (J3-14)
I6 = Control Panel (J3-13)
I7 = Control Panel (J3-12)

I8 = Control Panel (J3-19)
I9 = Control Panel (J3-18)
I10 = Control Panel (J3-17)
I11 = Control Panel (J3-16)
I12 = Control Panel (J3-20)
I13 = Control Panel (J3-21)
I14 = Control Panel (J3-22)
I15 = Control Panel (J3-23)

S0 = Configuration Switch 0, Control Panel (J3-3)
S1 = Configuration Switch 1, Control Panel (J3-6)
S2 = Configuration Switch 2, Control Panel (J3-5)
S3 = Configuration Switch 3, Control Panel (J3-4)
S4 = Configuration Switch 4
S5 = Configuration Switch 5
S6 = Configuration Switch 6, Control Panel (J3-7)
CD = Coin detect latch

EI = External Input (J4-10)

Inputs I0 through I15 are read using the 'INP b' instruction where 'b' is the input bit to be read. The value of the input will be returned in the 'A' register's bit-0. All other bits of the 'A' register will be set to zeros.

To read S0 through S6 the secondary accumulator must be selected before issuing the 'INP b' instruction. The value of the input will be returned in the 'B' registers bit-0. All other bits of the 'B' register will be set to zeros.

CD is read the same way as S0 through S6 are. A high value indicates no coin has been detected, and a low value indicates a dropped coin has triggered the coin detect latch. Once set low by the dropping of a coin into the coin chute, the CD input will remain low until the software releases the coin detect latch by toggling output bit O5. (See *Outputs.)*

The EI is tested directly by the 'JEI' instruction. Upon the execution of a 'JEI' instruction, a jump will be taken if the EI input is high.

4.2 Outputs

There are eight directly addressable outputs on the CPU board. They are mapped as:

O0 = Sound board (J4-11)
O1 = Sound board (J4-12)

O2 = Sound board (J4-13)
O3 = Sound board (J4-14)
O4 = Sound board (J4-15)
O5 = Coin detect reset logic
O6 = Vector generator logic (bright/dim or color/intensity latch)
O7 = Sound board (J4-16)

The Outputs are set/reset using the 'OUT b' instruction where 'b' is the output bit to be set. The value of the output bit will be set to the *inverted* value of bit-0 of the 'A' register. The 'B' register may also be selected in which case the output bit will be set to the *inverted* value of bit-0 of the 'B' register. All other register bits will be ignored.

O0 through O4 and O7 are all used to control the game's sound board. The definition of these pins are highly dependent upon the sound board logic and varies from game to game.

O5 is used to reset the coin detect latch. A coin being dropped into the coin chute triggers the coin detect latch which can be read as the CD input. (See *Inputs.*) An example of a program to read, and then reset, the coin detect latch is:

```
            ldj     #noCoin
            ssa             ; coin detect must be read into 'B' register
            inp     7       ; read coin detect latch
            ssa
            sub     #1      ; set to 0, or FFF
            ssa
            add     #1      ; set to 1, or 0, also set C-flg
            jnc             ; jump to 'noCoin' if no coin detected
            lda     $0      ; else, get coin counter
            add     #1      ; increment coin counter
            sta     [i]     ; save new coin count
            clr             ; reset bit-0
            add     #1      ; set bit-0
            out     5       ; reset coin latch by clearing output 5
            clr             ; reset bit-0
            out     5       ; setup to read next coin

noCoin:     ...
```

This routine counts coins and saves the count in RAM location $0 on the current RAM page.

O6 is used to control the intensity or color of a drawn vector and is defined in the *"Cinematronics Vector Generator"* section.

5. The Cinematronics Vector Generator

The Cinematronics Vector Generator's hardware is spread between the CPU board and the Vectorbeam monitor. The position registers are located on the CPU board, and are connected to the positions DACs, located on the Vectorbeam monitor through a ribbon cable.

A (very) quick Cinematronics Vector Generator theory of operation is: Two DACs are set to an initial X/Y position value, they are then used to quickly charge two capacitors to the value of the DACs. One capacitor is used to hold the X position of the CRT beam's location, and the other is used to hold the Y position. A new ending X/Y position is then loaded into the DACs, but instead of quickly charging the capacitors to this new value, the capacitor are allowed to slowly (relatively speaking) charge to the ending position through a couple of resistors. The line seen on the CRT is created by turning on the trace during this transition from the initial position values to the ending position values.

That's it in it's simplest form, with one caveat. The trace that would be drawn by simply allowing a capacitor to charge through a resistor, from one value to another, would not be a straight line, but would instead be curved. A capacitor starts charging quickly, but its charge rate will slow as the voltage approaches the ending voltage. This would draw a curved line, whose curve would be more pronounced the closer it came to the ending position. To compensate for this the ending position of the vector is place well off the screen. When the vector is to be drawn,

the capacitors are connect through resistors to the DACs. The CRT trace is then turn on and the line length timer is started, when the line length timer times out the CRT trace is turned off and the DACs are disconnected from the capacitors. Since only the first part of a very long line is displayed, the curve of the line is very small (it's unnoticeable on the Vectorbeam monitor).

5.1 Colors and Intensities

5.1.1 The Bi-Level display

As mentioned earlier the brightness of the Bi-Level display is set by using the OUT instruction to turn on and off bit-6 (O6). An 'OUT 6' instruction with bit-0 of the 'A' register set to 1, will cause all following vectors to be drawn "bright". If bit-0 if the 'A' register was set to 0, then all following vectors will be drawn "normal" intensity.

The Bi-Level display is used on all Cinematronics vector games except Sundance, Solar Quest, Boxing Bugs, the color version of War of the Worlds and Cosmic Chasm. Cosmic Chasm used a completely redesigned vector game system based on the 68000 CPU and is not covered by this document.

5.1.2 Multiple Intensity (Gray Scale) Displays and Color Displays

The X register (the register used to hold the X position of a vector) is used as a temporary register to hold the color/intensity of a vector. The color/intensity register is loaded by placing a value in the 'A' register and executing a 'VIN' instruction. This sets the color value into the X register. Output Bit-6 is now toggled using the 'OUT' instruction. This latches the color/intensity into the color/intensity register on the CRT or Color Conversion board. Output bit-6 is always left in the SET state.

There are three different types of color / intensity hardware used by Cinematronics games. A 16 level display, a 64 level display, and a color display. All use the 'X' register as a temporary storage of the color / intensity register.

5.1.2.1 Using the 'X' Register to set Colors / Intensities

A color or a gray scale intensity is set by loading the 'X' position register to the desired color / intensity, and strobing the O6 output. This is done by loading the 'A' register with the value intended for the 'X' register and then executing a 'VIN' instruction to load the 'A' register into the 'X' register. And then strobing the O6 output to load the 'X' register into the color / intensity register.

CAUTION! The 'VIN' instruction always attempts to move the CRT trace to the X and Y positions given in the 'A' and 'B' registers. The trace will stay at that position until it is moved elsewhere, or another vector is drawn. If the trace is moved too far from center for too long a time, damage to the monitor can result!

It is *very* important to set the Y value pointing to a valid position on the screen before executing the 'VIN' instruction (when using the instruction to set the color/intensity of a vector) and to reset the X and Y position to a valid position immediately after the color/intensity has been set! The Y value may be set to the next Y value to be used when drawing a vector. This would allow a sort of "head start" to the Y position. The X and Y position should be immediately set to the next X/Y position, or the center of the screen, after setting the color or intensity level. Leaving the X register loaded with the color value can cause the CRT beam to be moved off screen and can cause the circuit breaker to blow, and also cause undo stress on the Vectorbeam monitor.

The following code will safely set a value into the color/intensity register without damage to the CRT deflection circuits. It assumes the position of the CRT beam will be set to the center of the screen and the constant 'color' has been previously set to the desired color/intensity being set:

```
lda       #color & $F00       ; get the upper nibble of accumulator
add       #color & $0FF       ; get the lower byte of color/intensity value

; Be sure 'B' register points to the center of the screen

ssa                 ; use the B accumulator
lda     #$300       ; load maximum Y-value (768)
ssa                 ; use the B accumulator
lsr                 ; get Y-value's center of screen (768/2)
vin                 ; set color into CRT's X-reg
```

```
; Now latch color/intensity into color/intensity register

clr
add       #1        ; load $001 into acc
out       6         ; RESET OUT bit-6
clr       #1        ; load $000 into acc
out       6         ; SET OUT bit-6

; Set monitor's deflection back to the center of the screen

lda       #$400     ; load maximum X-value (1024)
ssa                 ; use the B accumulator
lda       #$300     ; load maximum Y-value (768)
asrd                ; set to center of screen (1024/2,768/2)
vin                 ; set CRT beam to the center of screen
```

5.1.2.2 The 16 Level Gray Scale Display

The format of the 'X' register for setting an intensity on a 16 gray scale monitor is:

'X' register value

x	x	x	x	x	x	x	x	i	i	i	i

Where:

x = Don't care.
i i i i = Intensity level.
0000 = Lowest intensity.
0001 = 2nd lowest intensity.
1111 = Highest intensity.

Setting the 'X' register to an intensity value, then strobing the O6 line, causes all subsequent vectors to be drawn in that intensity. A four bit value of '1111' ($F) is the brightest intensity, while '0000' ($0) is the dimmest.

The most significant bit of the intensity level is bit-3 of the 'X' register.

The 16 level gray scale monitor is used only in the game Sundance.

5.1.2.3 The 64 Level Gray Scale Display

The format of the 'X' register for setting an intensity on a 64 gray scale monitor is:

'X' register value

x	x	x	x	c	c	c	c	c	c	x	x

Where:

x = Don't care.
cccc cc = Intensity level.
1111 11 = Lowest intensity.
1111 10 = 2nd lowest intensity.
0000 00 = Highest intensity.

Setting the 'X' register to an intensity value, then strobing the O6 line, causes all subsequent vectors to be drawn in that intensity. The intensity bits are inverted, and a six bit value of '000000' ($00) is the brightest intensity, while '111111' ($3F) is the dimmest.

The most significant bit of the intensity level is bit-7 of the 'X' register.

The 64 level gray scale monitor is used only in the game Solar Quest.

5.1.2.4 The Color Display

The format of the 'X' register for setting 1 of 4096 colors on a color monitor is:

'X' register value

b	b	b	b	g	g	g	g	r	r	r	r

Where:

bbbb = Blue intensity level.
1111 = No blue.
1110 = Dimmest blue.
0000 = Maximum blue level.

gggg = Green intensity level.
1111 = No green.
1110 = Dimmest green.
0000 = Maximum green level.

r r r r = Red intensity level.
1111 = No red.
1110 = Dimmest red.
0000 = Maximum red level.

1111 1111 1111 = Black.
1110 1111 1111 = Lowest intensity blue.
1111 1110 1111 = Lowest intensity green.
1111 1111 1110 = Lowest intensity red.
0000 0000 0000 = Brightest white.

Setting the 'X' register to a color value, then strobing the O6 line, causes all subsequent vectors to be drawn in that color. The color bits are inverted for each color, and setting a color to the four bit value of '0000' ($00), sets that color to its brightest intensity, while a setting of '1111' ($F) sets the color to its dimmest intensity.

The most significant bits (in the 'X' register) for the different color's intensity levels are: Bit-11 for blue, bit-7 for green and bit-3 for red.

The color monitor is not a Vectorbeam monitor. It instead uses a Cinematronics to Wells Gardner conversion board to draw color vectors. The conversion board looks to the CPU like a Vectorbeam monitor and outputs signals that are compatible the Wells Gardner series of vector monitors. The Color Display monitors are used only in Boxing Bugs, and color versions of War of the Worlds.

5.2 Vectors

The responsibility of refreshing the vector screen lies completely on the CPU, hence the programmer. There is no "vector refresh engine". Instead all refreshing is the responsibility of the software and all vector refreshes must be intermingled with game logic.

5.2.1 Screen Resolution

The resolution of an X / Y monitor is not measured in pixels, but in starting and ending positions. The vector drawn between two points on a Cinematronics Vectorbeam monitor is an analog trace, and as such there is no resolution, or you can say it has infinite resolution since there is no stair-stepping effect one sees when a line is drawn on a raster display.

There is, however, a finite set of starting and ending points. On the Vectorbeam monitor a vector can start and end anywhere on a 1024(horizontal) by 768 (vertical) display. The lower left corner represents 0,0 and the upper right corner represents 1023,767. The center of the screen is 512,384.

5.2.2 Drawing a Vector

Drawing a vector can be broken down into 5 steps:

Step 1 - Verify that the previous vector is finished being drawn. This is done using the 'JDR' instruction. Usually the 'JDR' loops back on itself until the current vector is finished being drawn. Though this can be a good time to steal some CPU cycles if the vector being drawn is known to be a long one.

Step 2 - Set the intensity bit accessed by using the 'OUT 6' instruction. To set a vector to BRIGHT, the output bit 6 must be at logic zero. Since the 'OUT' instruction uses the inversion of bit 0 of the accumulator, to set vector draws to BRIGHT, load the accumulator with 1, and execute an 'OUT 6' instruction. For example:

```
clr                 ; Reset bit zero of 'A' register
out       6         ; Set vector draws to LOW intensity
```

or,

```
clr                 ; Clear 'A' register
add       #1        ; Set 'A' register to 1
out       6         ; Set vector draws to BRIGHT (high) intensity
```

(For multi-intensity monitors and Color monitors, see *"Colors and Intensities"*.)

Step 3 - Load the X and Y starting positions. This is done by loading the X position into the 'A' register and the Y position into the 'B' register. And then executing a 'VIN' instruction. This initializes the line draw sequence by loading the DACs on the monitor with the starting X and Y values and charging the capacitors to this starting value.

Step 4 - Normalize the vectors and load the length of the vector into the Line Length Timer. This is done by subtracting the Start positions of the vectors from the End positions, to calculate vector length, and then executing a 'NRM' instruction.

The 'NRM' instruction shifts the A and B registers left until bit-11 of the 'A' register does not match bit-9 of the 'A' register, or bit-11 of the 'B' register does not match bit-9 of the 'B' register. Each shift also clocks a new value into the Line Length register. The proper values for the Line Length register have been stored in PROM E8 on the CPU.

Note 1: Before executing the 'NRM' instruction some time must be wasted to allow the hardware to charge the capacitors to their initial values and stabilize after execution of the 'VIN' instruction. This is done with a small software timing loop.

Note 2: If both the 'A' register and 'B' register contain 0 at the execution of the 'NRM' instruction the CPU will hang, and eventually reset do to the timing out of the watch dog timer.

Note 3: When drawing a point 'Step 4' will lead to the 'A' and 'B' registers both containing zero. To draw a point place a 1 into the 'A' register, clear the 'B' register, and execute the 'NRM'. Then use the same ending positions as the starting positions when executing the 'VDR'. Placing a 2, 4, 8 etc. in the 'A' register allows the CRT beam to be held longer in one spot, which in turn creates a brighter dot. This allows for more than just two intensity levels when a point is being drawn.

Step 5 - Set ending points and draw vector. This is done by adding the starting values to the values that resulted from the 'NRM' instruction and placing the ending X position into the 'A' register and the ending Y position into the 'B' register and executing a 'VDR' instruction. The drawing of the line begins 11 cycles after the execution of the 'VDR' instruction, and before this time a 'JDR' instruction would indicate the previous line is finished being drawn.

A routine for drawing a vector is:

```
; Routine for drawing a vector.
;
; $4 = Start X position of vector to be drawn
; $5 = End X position of vector to be drawn
; $6 = Start Y position of vector to be drawn
; $7 = End Y position of vector to be drawn
;
; $F = Return address
;
; All RAM addresses are located in the current page
```

```
; and the 'P' register is left unchanged.
;
; It is assumed the intensity of the line has already
; been set using the 'OUT 6' instruction.

vectDr:   lda       $4        ; get Start X in 'A' register
          ssa                 ; point to B accumulator
          lda       $6        ; get start Y in 'B' register

; Wait for any previous vector to finish drawing.

          ldj     #loop1      ; point to next instruction
loop1:    jdr                 ; loop until previous draw is done

          vin                 ; load starting X/Y positions

; Wait for position capacitors to stabilize.

          lda       #$800     ; setup 'A' register for timing loop
          add       #$41      ; $841 = Loop $41 times
          ldj       #loop2    ; setup loop address for 'jmi'
loop2:    sub       #1        ; decrement timing loop counter
          nop                 ; MI flag not set until next instruction
          jmi                 ; if 'A' register still negative, loop

          lda       $5        ; get X end position in 'A' register
          sub       $4        ; subtract start to get length
          ssa
          lda       $7        ; get Y end position in 'B' register
          ssa
          sub       $6        ; subtract start to get length
          nrm                 ; load the Line Length Timer
          add       $4        ; add X start to 'A' reg to get new 'X' end
          ssa
          add       $6        ; add Y start to 'B' reg to get new 'Y' end
          vdr                 ; start vector drawing

; Retrieve return address and return to caller

          ldi       #$F       ; set I-reg to point to return address RAM
          ldj       [i]       ; load J-reg with return address in RAM
          jmp                 ; return to caller
```

And a routine for drawing a single point is:

```
; Routine for drawing a single point.
;
; $4 = X position of point to be drawn
; $6 = Y position of point to be drawn
;
; $F = Return address
;
; All RAM addresses are located in the current page
; and the 'P' register is left unchanged.

pointDr:  lda       $4        ; get Start X in 'A' register
          ssa                 ; point to B accumulator
          lda       $6        ; get start Y in 'B' register

; Wait for any previous vector to finish drawing.

          ldj     #loop1      ; point to next instruction
```

```
loop1:    jdr                ; loop until previous draw is done

          vin                ; load starting X/Y positions

; Wait for position capacitors to stabilize.

          lda       #$800    ; setup 'A' register for timing loop
          add       #$41     ; $841 = Loop $41 times
          ldj       #loop2   ; setup loop address for 'jmi'
loop2:    sub       #1       ; decrement timing loop counter
          nop                ; MI flag not set until next instruction
          jmi                ; if 'A' register still negative, loop

          ssa                ; select the 'B' register
          clr                ; clear the 'B' register

; At this point the value of the 'A' register will determine the
; brightness of the point.  The larger the value the brighter the point.
; The values must be powers of two. The acceptable values are:
;
;    1,2,4,8,16,32,64,128,256,512,1024
;
; allowing for 11 levels of intensities when drawing a point.

          clr                ; clear the 'A' register
          add       #32      ; set the 'A' reg to a 'medium' intensity
          lda       $5       ; get X end position in 'A' register
          nrm                ; load the Line Length Timer

          lda       $4       ; re-load X start position to use as end position
          ssa
          add       $6       ; re-load Y start position to use as end position
          vdr                ; start vector drawing

; Retrieve return address and return to caller

          ldi       #$F      ; set I-reg to point to return address RAM
          ldj       [i]      ; load J-reg with return address in RAM
          jmp                ; return to caller
```