```
Carry Bit     Set unless instruction is a Jump

Subroutine    jsb n [label]           stack[0] -> stack[1]                nn nnnn nn01
                                      PC       -> stack[0]
              PC = PC and $1F00 + (Code and $FF)
              PC = PC and $10FF + Delayed ROM

Jump
              go to n                 Carry bit must be clear or no jump
              then go to n            n = address or [label] in same ROM page
              if no carry go to n

              The jump executes if the carry flag is clear
              If previous instruction was an [if] type then
                PC = (PC and $1C00) + 10 bit code                         nn nnnn nnnn
              else
                PC = PC and $1F00 + (Code and $FF)                        nn nnnn nn11
                PC = PC and $10FF + DelayedROM
              If (PC > $FFF) and (PC < $1400) then PC = PC and $FFF

Arithmetic

  Registers are 56 bits wide, split into 14 x 4 bit nibbles
  Nibbles are modified as determined by the Data Select [fs] bits in the instruction
                                                                          fff
    [p]   -  determined by P register                                     000
    [wp]  -  word up to and including P register                          001
             Eg P=3, nibbles 0,1,2,3
    [xs]  -  exponent and sign                                            010
    [x]   -  exponent                                                     011
    [s]   -  mantissa sign                                                100
    [m]   -  mantissa                                                     101
    [w]   -  word (entire register)                                       110
    [ms]  -  mantissa and sign                                            111

    Example:      0 -> a[m]    A=91234560000912 -> A=90000000000912

  Clear
              0 -> a[fs]              Clear A                             00000fff10
              0 -> b[fs]              Clear B                             00001fff10
              0 -> c[fs]              Clear C                             01000fff10

  Transfer
              a -> b[fs]              Transfer A into B                   00011fff10
              b -> c[fs]              Transfer B into C                   00110fff10
              c -> a[fs]              Transfer C into A                   00101fff10

  Exchange
              a exchange b[fs]                                            00010fff10
              b exchange c[fs]                                            00111fff10
              a exchange c[fs]                                            00100fff10

  Add         a + c -> c[fs]                                              01100fff10
              a + b -> a[fs]                                              01001fff10
              a + c -> a[fs]                                              01010fff10
              c + c -> c[fs]                                              01011fff10

  Subtract
              a - c -> c[fs]                                              10001fff10
              a - b -> a[fs]                                              10000fff10
              a - c -> a[fs]                                              11100fff10

  Compare     if b[fs] = 0                                                10110fff10
              if c[fs] = 0                                                10111fff10
              if a[fs] # 0  (not = 0)
              if c[fs] # 0  (not = 0)                                     11011fff10
              if a >= b[fs]                                               11001fff10
              if a >= c[fs]                                               11000fff10

  Complement
              0 - c -> c[fs]          Tens Complement   0-C->C            10100fff10
              0 - c - 1 -> c[fs]      Nines Complement  0-C-1->C          10101fff10

  Increment
              a + 1 -> a[fs]          Increment A                         01101fff10
              c + 1 -> c[fs]          Increment C                         01111fff10

  Decrement
              a - 1 -> a[fs]          Decrement A                         10010fff10
              c - 1 -> c[fs]          Decrement C                         10101fff10
```

Shift			
	shift right a[fs]		11101fff10
	shift right b[fs]		11110fff10
	shift right c[fs]		11111fff10
	shift left a[fs]		01110fff10
Status Register Operations		(16 bit register -> n = 0 - 15)	
	1 -> s(Stat Bit)	S[n] = 1	ssss000100
	0 -> s(Stat Bit)	S[n] = 0	ssss001100
	clear status	All bits cleared except 1, 2 5, 15	0001001000
	if s(n) # 1	if S[n] not equal to 1 - carry = 0	ssss011100
	if s(n) = 1	if S[n] equals 1 - carry = 0	ssss010100
P Operations		(4 bit register -> n = 0 - 13)	
	0 -> p		1100111100
	1 -> p		1000111100
	2 -> p		0100111100
	3 -> p		1001111100
	4 -> p		0001111100
	5 -> p		1110111100
	6 -> p		1011111100
	7 -> p		0010111100
	8 -> p		0011111100
	9 -> p		1101111100
	10 -> p		0110111100
	11 -> p		0100111100
	12 -> p		0111111100
	13 -> p		1010111100
	p + 1 -> p	Increment p (if > 15 = 0)	0111010000
	p - 1 -> p	Decrement p (if < 0 = 15)	0110010000
	if p # n	If P not equal to n -> carry - 0	
	if p # 0		1011101100
	if p # 1		0101101100
	if p # 2		0011101100
	if p # 3		0111101100
	if p # 4		0000101100
	if p # 5		1010101100
	if p # 6		0110101100
	if p # 7		1110101100
	if p # 8		0001101100
	if p # 9		0100101100
	if p # 10		1101101100
	if p # 11		1100101100
	if p # 12		0010101100
	if p # 13		1001101100
	if p = n	If P equal to n -> carry = 0	
	if p = 0		1011100100
	if p = 1		0101100100
	if p = 2		0011100100
	if p = 3		0111100100
	if p = 4		0000100100
	if p = 5		1010100100
	if p = 6		0110100100
	if p = 7		1110100100
	if p = 8		0001100100
	if p = 9		0100100100
	if p = 10		1101100100
	if p = 11		1100100100
	if p = 12		0010100100
	if p = 13		1001100100
Data Entry			
	load constant n	n -> C[p], p - 1 -> p (n = 0 - 15)	nnnn011000
	m1 -> c		0101001000
	m2 -> c		0111001000
	m1 exchange c		0100001000
	m2 exchange c		0110001000
	c -> stack	Push C onto stack C->C->D->E->F	1011001000
	stack -> A	Pop stack into A F->F->E->D->A	1000001000
	down rotate	Rotate stack down C->F->E->D->C	1001001000
	c -> data address	Data Address = (C[1] * 16) + C[0]	1001110000
	data -> c	RAM[RAMadd] into C	0000111000
	c -> data	C -> RAM[RAMadd]	1011110000
	c -> data register n	C -> RAM[RAMadd and $30 + n]	nnnn101000
	data register n -> c	RAM[RAMaddr and $30 + n] -> C	nnnn111000
	clear registers	Clear registers A B C D E F	0000001000
	clear data registers	Clear RAM[RAMaddr and $30 + n0-n15]	1010110000
	f exchange a	Swap Freg (4 bit) and [A][0]	1111001000
	f -> a	Freg (4 bit) -> [A][0]	1110001000

Rom Select (n = 0 - 15)

select rom n	PC = PC and $1F00 + (Code div 4)	nnnn100000
delayed select rom n	DelayedROM = (Code and $001) x $100	nnnn110100

General

display off	Turn display off	0011001000
display toggle	Toggle display on/off	0010001000
return	Subroutine return Stack[0] -> PC Stack[1] -> Stack[0] DelayedROM = 0	1000010000
a -> rom address	PC = (PC and $1F00) + ([A][2] * 16) + [A][1]	0010010000
y -> a	[d] -> [a]	
no operation	Does nothing	0000000000
binary	Set binary math mode - base 16 Nibble Example: 9 + 3 = C	0100010000
decimal	Set decimal math mode - base 10 Nibble example: 9 + 3 = 2	1100001000
bank switch	Toggle ROM [0 -> 7] or [8 -> F]	1000110000
display reset kmf	???	0011010000
hi i'm woodstock	???	

Card Reader Commands

CRC 060	Set display digits flag		0000110000
CRC 100	Test CRC ready	True -> S3 = True	0001000000
CRC 160	Test display digits flag	True -> S3 = True Flag = False	0001110000
CRC 260	Card Motor = On		0010110000
CRC 300	Test RUN/PGM switch	PGM -> S3 = True	0011000000
CRC 360	Card Motor = Off		
CRC 560	Test if card inserted	True -> S3 = True	0101110000
CRC 660	Set card write mode		0110110000
CRC 760	Set card read mode		0111110000
CRC 1000	Set default function flag		1000000000
CRC 1100	Test default function flag	True -> S3 = True Flag = False	1001000000
CRC 1200	Set merge flag		1010000000
CRC 1300	Test merge flag	True -> S3 = True Merge flag = False	1011000000
CRC 1400	Set pause flag		1100000000
CRC 1500	Test pause 2 flag	True -> S3 = True Flag = False	1101000000
CRC 1700	Read/Write data to/from card		1111000000

CRC_Flag1 -> S[3]		???
CRC_Flag2 -> S[3]		???
CRC_Flag3 -> S[3]		???
CRC_Flag4 -> S[3]		???

Printer Interface Keyboard Commands

PIK 1120	Print 1 -> Test carriage home True -> S3 = True	1001010000
PIK 1220	Print 2 -> Test out of paper switch True -> S3 = False	1010010000
PIK 1320	Print 3 -> Test if key press available True -> S3 = True	1011010000
PIK 1660	Print 6 -> Output alpha data to printer [C] -> PIK buffer - 6 bit	1110110000
PIK 1720	Print 0 -> Output numeric data to buffer [C] -> PIK buffer - 4 bit	1111010000