IntyBASIC 1.2.5 Quick Reference

16-bit variables and arrays must start with #
$0000 = %00000000 = " " = 0

REM *This is a comment*
' *This is a comment too*

CONST name = value
SIGNED var_or_array
UNSIGNED var_or_array
VARPTR var_or_array
DIM array(size)
label: **PROCEDURE** ' *One line*
    … **END**

GOTO label
GOSUB procedure
RETURN

ON expr GOTO label[, label…]   ' *IntyBASIC's switch statement*
ON expr GOSUB label[,, label…] ' *Can omit labels to skip for certain values*
ON FRAME GOSUB label   ' *Can only appear once*
STACK_CHECK   ' *Checks for stack overflows in VBLANK routine*
WAIT   ' *For VBLANK - 60hz NTSC, 50hz PAL*

POKE address, value
RESTORE label ' *Sets read point*
READ var[, var…]
var = labelname(index) ' *Equivalent*
DATA const[, const…]
DATA string

FOR i = value TO value [STEP value]
NEXT [i]

WHILE expr … WEND
DO WHILE expr … LOOP
DO UNTIL expr … LOOP
DO … LOOP WHILE expr
DO … LOOP UNTIL expr
EXIT (FOR/WHILE/DO)

IF expr GOTO label
IF expr [ELSEIF expr THEN …]
    [ELSE label] END IF

INCLUDE "filename.bas"
ASM INCLUDE "filename.asm"
ASM assembler_instruction
CALL asm_function([argument, …])
var = USR asm_function([argument, …])
DEF FN function([var, …]) = expr

DEFINE [ALTERNATE] card_num, total, label
DEFINE [ALTERNATE] card_num, total, VARPTR label(expr)

'*Secondary PSG (requires ECS) identical except SOUND 5 through 9*
SOUND 0, sound_12bit, vol_4bit   ' *Channel A*
SOUND 1, sound_12bit, vol_4bit   ' *Channel B*
SOUND 2, sound_12bit, vol_4bit   ' *Channel C*
SOUND 3, sound_16bit, type_4bit   ' *Volume envelope (frequency/shape)*
SOUND 4, noise_5bit, mix_reg ' *Noise and mix register ($38 value by default)*
SOUND (0…2), sound_12bit, 48 ' *Magic volume number, means use envelope on 3*

SPRITE index, x_coord, y_coord, cardinfo   ' Index = sprite number

x_coord bits:
7-0: position
  8: interaction
  9: visibility
 10: double width
	y_coord bits:
	6-0: position
	  7: 16 line sprite
	9-8: Scale: 00,01,10,11
	    (0.5x, 1x, 2x, 4x)
	 10: flip X
	 11: flip Y
		cardinfo bits:
		 2-0: lower bits/color
		11-3: card number
		 12: upper bit/color
		 13: mob behind bg

CLS ' Clears screen, sets cursor to upper left

PRINT [**AT** expr] [**COLOR** expr][**,**] string[, string…] ' *XOR with color present*

Foreground/Background **COLOR** bits:
  2-0: Foreground color (0-7)
  8-9: Background bits 0-1
   12: Background bit <u>3</u> (not 2!)
   13: Background bit <u>2</u>
	Color Stack **COLOR** bits:
	  2-0: Low 3 bits of FG color
	   12: High bit of FG color
	    (Must be 0 for GROM cards)
	   13: Change color stack

SCROLL offset_x, offset_y, move_screen
' *move_screen will move screen if offset_x or offset_y exceed 7*
' *0 = no move, 1 = move left, 2 = move right, 3 = move up, 4 = move down*

BORDER color, mask   ' *color = 0 through 15*
' *mask: 0 = mask none, 1 = mask left column, 2 = mask top row, 3 = mask both*

SCREEN label[, source_offset, target_offset, cols, rows[, source_width]]
' *Label can also be #array() for dynamically-drawn elements*
' *source_offset = distance from label, target_offset = distance/screen pos. 0*

BITMAP "___XX___"   ' *Anything not "0", "_", ".", or space = 1.*
' *Should be paired. Stored as 16-bit DECLEs (high bits = row 2, low = row 1)*

PLAY SIMPLE [**NO DRUMS**] ' *Simple means sound channel 2 is available*
PLAY FULL [**NO DRUMS**]   ' *NO DRUMS means sound channel 4 is available*
PLAY VOLUME expr   ' *0 = silent, 15 = max*
PLAY NONE
' *Turn off sound with SOUND 4,0,$38 after using NO DRUMS*
' *Turn off sound with SOUND (1…3),1,0 and SOUND 4,0,$38 after NONE*

PLAY label
' *Label can also be #array() for dynamically generated music*

label: **DATA** tempo   ' *Ticks per note, 50 ticks per second NTSC/PAL both*
  MUSIC note1, note2, note3[, note4]

Mandatory parts of note: (1-3 only)
  Note + Octave (C2 through C8)
  Sharp notes: D4#
Drums: (note4 only)
  M1 = strong, M2 = tap, M3 = roll
	Can also add instrument after note:
	  W = piano, X = clarinet,
	  Y = flute, Z = bass (C4#W, etc.)
	- means silence (no note for beat)
	S means sustain previous note

VOICE INIT   ' *Must come before any voice commands*
VOICE PLAY label   ' *Play sound information at label (or #array())*
VOICE WAIT   ' *Halt execution until voice queue is clear*
VOICE PLAY WAIT label ' *Play voice and halt execution until sound is done*
VOICE NUMBER expr   ' *Say number out loud ("twenty seven thousand")*

label: **VOICE** phoneme_or_phrase[, phoneme_or_phrase…], 0

Phrases:
  MATTEL, ZERO, ONE, TWO, THREE,
  FOUR, FIVE, SIX, SEVEN, EIGHT,
  NINE, TEN, ELEVEN, TWELVE,
  THIRTEEN, FOURTEEN, FIFTEEN,
  SIXTEEN, SEVENTEEN, EIGHTEEN,
  NINETEEN, TWENTY, THIRTY, FO<u>U</u>RTY,
  FIFTY, SIXTY, SEVENTY, EIGHTY,
  NINETY, HUNDRED, THOUSAND, TEEN,
  TY, PRESS, ENTER, OR, AND
	Phonemes:
	  PA5, PA4, PA3, PA2, PA1 (pauses),
	  AA, AE1, AO, AR, AW, AX, AY, BB1,
	  BB2, CH, DD1, DD2, DH1, DH2, EH,
	  EL, ER1, ER2, EY, FF, GG1, GG2,
	  GG3, HH1, HH2, IH, IY, JH, KK1,
	  KK2, KK3, LL, MM, NG1, NN1, NN2,
	  OR2, OW, OY, PP, RR1, RR2, SH, SS,
	  TH, TT1, TT2, UH, UW1, UW2, VV,
	  WH, WW, XR2, YR, YY1, YY2, ZH, ZZ

FLASH INIT   ' *Put at start of program, compile --jlp*
FLASH ERASE row   ' *row goes from FLASH.FIRST to FLASH.LAST*
FLASH READ row, **VARPTR** #array() ' *#array() must hold exactly 96 elements*
FLASH WRITE row, **VARPTR** #array() ' *Flash ops stop execution for a moment*

Number of 8-bits variables allowed: <u>228</u>
       Subtract 3 if you use SCROLL   *Subtract 3 if you use VOICE*
       Subtract 6 if you use the keypad   *Subtract 26 if you use PLAY*

Number of 16-bits variables allowed: <u>47</u>
       7962 if using --jlp or --cc3 switch)
       Subtract 20 if you use SCROLL   *Subtract 30 if you use VOICE*

Controller variables (CONT is all controllers, CONT1 and CONT2 are specific):

CONT	CONT1	CONT2	
Contains bitmask from $01ff (left/1), $01fe (right/2), or both together			
CONT.UP	CONT1.UP	CONT2.UP	*Non-zero if UP pressed*
CONT.DOWN	CONT1.DOWN	CONT2.DOWN	*Non-zero if DOWN pressed*
CONT.LEFT	CONT1.LEFT	CONT2.LEFT	*Non-zero if LEFT pressed*
CONT.RIGHT	CONT1.RIGHT	CONT2.RIGHT	*Non-zero if RIGHT pressed*
CONT.BUTTON	CONT1.BUTTON	CONT2.BUTTON	*Non-zero if any button pressed*
CONT.B0	CONT1.B0	CONT2.B0	*Non-zero if top buttons pressed*
CONT.B1	CONT1.B1	CONT2.B1	*Non-zero if left button pressed*
CONT.B2	CONT1.B2	CONT2.B2	*Non-zero if right button pressed*
CONT.KEY	CONT1.KEY	CONT2.KEY	
Current pressed key (0-9, 10=clear, 11=enter, 12=not pressed) *Because movements can be read as keys, wait for 12 before waiting for key*			

COL0	COL1	COL2	COL3	COL4	COL5	COL6	COL7
Collision between sprites for frame. Bit 0-7 = collision with that sprite Bit 8 = collision against background pixel, Bit 9 = against borders							

RAND   ' *Pseudo-random value between 0 and 255, updated each frame*
RAND(max**)**   ' *Same as RAND but for 0 to max. Powers of 2 are optimized.*
RANDOM(max**)**   ' *Same as RAND(max) but doesn't need frame wait. Slower.*

LEN(string**)**   ' *Gives length of string. Useful in macros.*
POS(expr**)**   ' *Gives current cursor position. Expression evaluated, not used*

FRAME   ' *Current frame number (0-65535, cycles over itself)*
NTSC   ' *1 if Intellivision is NTSC, 0 otherwise*
#MOBSHADOW()   ' *Alias for locations 0-23 of STIC (the MOB buffer)*
#BACKTAB()   ' *Alias for screen buffer ($0200-$02EF)*
FLASH.FIRST, **FLASH.LAST**   ' *First and last rows of JLP flash memory*

MUSIC.PLAYING ' *1 if music is playing, 0 otherwise*