GBDK

Lars 'Gluemaster' Malmborg

COLLABORATORS			
	TITLE : GBDK		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Lars 'Gluemaster' Malmborg	July 26, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

GBDK

1.1 GameBoy Developer's Kit

```
*************************************************************
*                            \|/                            *
*                            @ @                            *
*------------------------oooO-(_)-Ooo-----------------------*
*                                                           *
*                            GBDK                           *
*                                                           *
*-----------------------------------------------------------*
*                                                           *
*    GameBoy Developer's Kit  © 1996 by Pascal Felber       *
*          Hardware info  © 1996 by Pan/Anthrox             *
*     GBDK.guide  © 1996 by Lars 'Gluemaster' Malmborg      *
*                                                           *
*                       Public Domain                       *
*                                                           *
*************************************************************
```

Authors

1.2 GBDK

```
What is the GBDK?
The name may sound pretentious. But with the GBDK, you can develop your own
programs for the GB system, either in C or in assembly. The GBDK includes
a set of libraries for the most common requirements and generates image
files for use with a real GB or with VGB.

Features
  * A full featured C compiler (with the only limitation that a floating
    point library has yet to be written)
  * An assembler that generates relocatable code
  * A linker that produces GB image files
  * A set of basic libraries, with source code
  * Some example programs in assembly and in C
  * An imaging tool that generates GB source code for including image
    files to GB programs

Well, the programs are not written from scratch.

The compiler is based on lcc, a retargetable compiler for
ANSI/ISO C. The orininal version generates code for the SPARC, MIPS R3000,
and Intel 386 and its successors. lcc is in production use at
Princeton University and AT&T Bell Laboratories. The man page gives usage
details.

The assembler and the linker are based on public domain programs developed
by Alan R. Baldwin.

The image tool in the UNIX distribution is based on Xloadimage,
a utility which will view many different types of images under X11, load
images onto the root window, or dump processed images into one of several
image file formats. One format it does not support is IFF ILBM, which is the
most common Amiga file format. Therefor it is not very useful on the Amiga,
and that is the reason you'll find ILBMtoGB in the Amiga port instead. It will
generate about the same output as the Xloadimage, but it will only read IFF
ILBM. Keep in mind it is not a port, but a native Amiga application (written by
Lars Malmborg), hence it uses Amiga argument parsing.

For the time, only Unix is supported apart from this Amiga port.
The Unix version of GBDK has been tested on Sun Solaris 2.4 and Linux.

Limitations
  * The C compiler is missing support for floats and doubles (the compiler
    supports them, but libraries are missing. If someone is interested in writing
    them...)
  * The linker generates only 32kb images for the time. Generating 64kb
    images is not a problem, but bigger images require bank switching.
  * Do not use -0x8000 (minimum 16-bit signed integer) in divisions. -0x7FFF
    is the limit.

Sites with info on the GB
  * Nintendo GameBoy Homepage
```

```
      (http://www.freeflight.com/fms/GameBoy/)
  * Pascal Felber's GameBoy Developer's Kit
      (http://lsewww.epfl.ch/~felber/GBDK/)
  * Jeff Frohwein's Technical Information Page
      (http://fly.hiwaay.net/~jfrohwei/gameboy/home.html)

How to run the programs developed
The programs developed can be run with either a real GameBoy somehow connected
to a hosting Amiga or with a GameBoy emulator. I have ported Virtual GameBoy,
AmigaVGB (Aminet:misc/emu/AmigaVGB.lha), and all testing of GBDK has been done
with it.
```

1.3 Installing

```
Installing the GBDK should be easy. Run InstallGBDK from Workbench.
It will create a directory GBDK where it installs all the files.
It will also insert an assign to GBDK: in the User-Startup.
```

1.4 lcc

```
The driver
lcc [ option | file ]...
        except for -l, options are processed left-to-right before files
        unrecognized options are taken to be linker options
-A      warn about non-ANSI usage; 2nd -A warns more
-b      emit expression-level profiling code; see bprint(1)
-Bdir/  use the compiler named 'dir/rcc'
-c      compile only
-C      prevent the preprocessor from stripping comments
-dn     set switch statement density to 'n'
-Dname -Dname=def       define the preprocessor symbol 'name'
-E      run only the preprocessor on the named C programs and unsuffixed files
-g      produce symbol table information for debuggers
-help   print this message
-Idir   add 'dir' to the beginning of the list of #include directories
-lx     search library 'x'
-N      do not search the standard directories for #include files
-n      emit code to check for dereferencing zero pointers
-O      is ignored
-o file leave the output in 'file'
-P      print ANSI-style declarations for globals
-p -pg  emit profiling code; see prof(1) and gprof(1)
-S      compile to assembly language
-t -tname       emit function tracing calls to printf or to 'name'
-Uname  undefine the preprocessor symbol 'name'
-v      show commands as they are executed; 2nd -v suppresses execution
-w      suppress warnings
-W[pfal]arg     pass 'arg' to the preprocessor, compiler, assembler, or linker

All of these options are described in more detail in the man pages for the
original UNIX distribution of lcc.
```

1.5 lcc man pages

```
man lcc
Arguments whose names end with '.c' are taken to be C source programs; they
are preprocessed, compiled, and each object program is left on the file whose
name is that of the source with '.o' substituted for '.c'.
Arguments whose names end with '.i' are treated similarly, except they are not
preprocessed.
In the same way, arguments whose names end with '.s' are taken to be assembler
source programs and are assembled, producing a '.o' file.
If there are no arguments, lcc prints a message summarizing its
options on the standard error.
lcc deletes a '.o' file if and only if exactly one source file ('.c',
'.s', or '.i' file) is mentioned and no other file (source, object, library) or
-l option is mentioned.
lcc uses ANSI standard header files in preference to the 'old-style'
header files normally found in include.
Include files not found in the ANSI header files are taken from the normal
default include areas, which usually includes include:.
lcc interprets the following options; unrecognized options are taken as
loader options unless -c, -S or -E precedes them.
Except for -l, all options are processed before any of the files and apply to
all of the files.
Applicable options are passed to each compilation phase in the order given.
-c
Suppress the loading phase of the compilation, and force
an object file to be produced even if only one program is compiled.

-g
Produce additional symbol table information for the local debuggers.
lcc warns when -g is unsupported.

-w
Suppress warning diagnostics, such as those
announcing unreferenced statics, locals, and parameters.
The line #pragma ref id simulates a reference to the variable id.

-dn
Generate jump tables for switches whose density is at least n,
a floating point constant between zero and one. The default is 0.5.

-A
Warns about declarations and casts of function types without prototypes,
missing return values in returns from int functions, assignments between
pointers to ints and pointers to enums, and conversions from pointers to
smaller integral types.
A second -A warns about unrecognized control lines, non-ANSI language
extensions and source characters in literals, unreferenced variables and
static functions, declaring arrays of incomplete types, and exceeding
some ANSI environmental limits, like more than 257 cases in switches.
It also arranges for duplicate global definitions in separately compiled
files to cause loader errors.

-P
Writes declarations for all defined globals on standard error. Function
declarations include prototypes; editing this output can simplify conversion
```

to ANSI C. This output may not correspond to the input when there are several typedef's for the same type.

-n
Arrange for the compiler to produce code that tests for dereferencing zero pointers. The code reports the offending file and line number and calls abort.

-O
is ignored.

-S
Compile the named C programs, and leave the assembler-language output on corresponding files suffixed '.s'.

-E
Run only the preprocessor on the named C programs and unsuffixed file arguments, and send the result to the standard output.

-o output
Name the output file output. If -c or -S is specified and there is exactly one source file, this option names the object or assembly file, respectively. Otherwise, this option names the final executable file generated by the loader, and 'a.gb' is left undisturbed.
lcc warns if -o and -c or -S are given with more than one source file and ignores the -o option.

-D name=def
-D name
Define the name to the preprocessor, as if by '#define'. If no definition is given, the name is defined as "1".

-U name
Remove any initial definition of name.

-I dir
'#include' files whose names do not begin with '/' are always sought first in the directory of the file arguments, then in directories named in -I options, then in directories on a standard list.

-N
Do not search any of the standard directories for '#include' files. Only those directories specified by explicit -I options will be searched, in the order given.

-B str
Use the compiler strrcc instead of the default version. Note that str often requires a trailing slash.

-v
Print commands as they are executed; some of the executed programs are directed to print their version numbers. More than one occurrence of -v causes the commands to be printed, but not executed.

-help
Print a message summarizing lcc's options on the standard error.

-b
Produce code that counts the number of times each expression is executed. If loading takes place, replace the standard exit function by one that writes a prof.out file when the object program terminates.
A listing annotated with execution counts can then be generated with bprint. lcc warns when -b is unsupported. -Wf-C is similar, but counts only the number of function calls.

-p
Produce code that counts the number of times each function is called. If loading takes place, replace the standard startup function by one that automatically calls monitor at the start and arranges to write a mon.out file when the object program terminates normally. An execution profile can then be generated with prof. lcc warns when -p is unsupported.

-pg
Causes the compiler to produce counting code like -p, but invokes a run-time recording mechanism that keeps more extensive statistics and produces a gmon.out file at normal termination.
Also, a profiling library is searched, in lieu of the standard C library. An execution profile can then be generated with gprof. lcc warns when -pg is unsupported.

-t name
-t
Produce code to print the name of the function, an activation number, and the name and value of each argument at function entry. At function exit, produce code to print the name of the function, the activation number, and the return value. By default, printf does the printing; if name appears, it does. For null char* values, "(null)" is printed.

-W xarg
Pass argument arg to the program indicated by x; x can be one of p, f, a or l, which refer, respectively, to the preprocessor, the compiler proper, the assembler, and the loader. arg is passed as given; if a - is expected, it must be given explicitly. -Woarg specifies a system-specific option, arg.

-pipe
Forces lcc to pipe the preprocessor output directly to the compiler instead of using temporary files.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier lcc run, or perhaps libraries of C-compatible routines. Duplicate '.o' files are ignored. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name a.gb.

lcc assigns the most frequently referenced scalar parameters and locals to registers whenever possible. For each block, explicit register declarations are obeyed first; remaining registers are assigned to automatic locals if they are 'referenced' at least 3 times. Each top-level occurrence of an identifier counts as 1 reference. Occurrences in a loop, either of the then/else arms of an if statement, or a case in a switch statement each count, respectively, as 10, 1/2, or 1/10 references. These values are increased accordingly for nested control structures. -Wf-a causes lcc to read a

```
prof.out file from a previous execution and to use the data therein
to compute reference counts (see -b).

lcc is a cross compiler; -Wf-target= target-os causes lcc to
generate code for target running the operating system denoted by os.
The supported target-os combinations may include
  mips-irix     big-endian MIPS, IRIX 4.0
  mips-ultrix   little-endian MIPS, ULTRIX 4.3
  sparc-sun     SPARC, SunOS 4.1
  sparc-solaris SPARC, Solaris 2.3
  x86-dos       [345]86, DOS 6.0
  symbolic      textual rendition of the generated code
  null          no output

The -v option lists the target-os combinations supported by specific
installations of lcc.

LIMITATIONS
lcc accepts the C programming language as described in the ANSI
standard and in the second edition of Kernighan and Ritchie. lcc is
intended to be used with the GNU C preprocessor, which supports the
preprocessing features introduced by the ANSI standard. The -Wp-trigraphs
option is required to enable trigraph sequences.
Wide-character literals are accepted but are treated as plain char literals.
Plain chars are signed chars, ints and long ints are the same size as are
doubles and long doubles, and plain int bit fields are signed. Bit fields are
aligned like unsigned integers but are otherwise laid out as if by the standard
C compiler, cc. Other compilers, such as the GNU C compiler, gcc,
may choose other, incompatible layouts.
Likewise, calling conventions are intended to be compatible with cc,
except possibly for passing and returning structures. Specifically, lcc
passes structures like cc on all targets, but returns structures like cc
on only the MIPS. Consequently, calls to/from such functions compiled with cc
or other C compilers may not work. Calling a function that returns a structure
without declaring it as such violates the ANSI standard and may cause a core
dump.

FILES
The file names listed below are typical, but vary among installations;
installation-dependent variants can be displayed by running lcc with the -v
option.
  file.c      input file
  file.o      object file
  a.gb        loaded output
  T:lcc*      temporaries
  bin/cpp     preprocessor
  bin/rcc     compiler
  lib/crt0.o  runtime startup
  include     headers

lcc predefines the macro '__LCC__' on all systems and the macros 'unix' on
UNIX systems. It may also predefine some installation-dependent symbols; option
-v exposes them.

SEE ALSO
B. W. Kernighan and D. M. Ritchie,
  The C Programming Language,
```

```
  Prentice-Hall, 2nd Ed., 1988.

American National Standard for Information Systems, Programming Language C,
  ANSI X3.159-1989, American National Standards Institute, Inc., New York, 1990.
.PP
C. W. Fraser and D. R. Hanson,
  A Retargetable C Compiler: Design and Implementation,
  Benjamin Cummings, 1995. ISBN 0-8053-1670-1.

The Wide World Web page at URL http://www.cs.princeton.edu/software/lcc.
```


1.6 Compiler

```
The compiler
Pascal Felber has written a code generator for lcc that generates
code for the Z80. It does not produce optimal code, but it is usable. It took
him a long time to debug, but is now quite stable (according to himself!).
Note than due to the limitations of the Z80, sizeof(int) = sizeof(long) = 2.

For more information, read the docs included with the lcc distribution.

The following flags allow to pass options to the assembler and to the linker:
    -Wa
    -Wl
If the assembler generates an error message, you can produce an assembly
listing .lst to see where the error occurs using the flag:
    -Wa-l
If you want to see the memory mop of the image file (where the functions
are located in ROM), you can produce a .map file using:
    -Wl-m
```

1.7 Assembler

```
The assembler
The assembler accepts the following flags:
    Usage: [-vdqxcgalosf] [-n filename] file1 [file2 file3 ...]
      v    verbose
      d    decimal  listing
      q    octal    listing
      x    hex      listing (default)
      k    case sensitive
      g    undefined symbols made global
      a    all user symbols made global
      l    create list   output file[LST]
      o    create object output file[O]
      s    create symbol output file[SYM]
      f    flag relocatable references by  `   in listing file
     ff    flag relocatable references by mode in listing file
      n    name of output files (for following input file)

For more information, read the asmlnk.doc file.
```

Also check out the instruction set and the custom registers of GameBoy.

1.8 Linker

```
The linker
The linker accepts the following flags:
    Usage: [-options] -o outfile [file.o ... | @file.lst]
      @file.lst      file with list of files to link, separated by newlines
      -c             case sensitive
      -v             verbose
    Relocation:
      -b             area base address = expression
      -g             global symbol = expression
    Map format:
      -m             map output generated as file[MAP]
      -x             hexidecimal (default)
      -d             decimal
      -q             octal
    Output:
      -i             Intel hex as file[IHX]
      -s             Motorola s19 as file[S19]
      -z             Gameboy image as file[GB]

For more information, read the asmlnk.doc file.
```

1.9 Libraries

The libraries
Three libraries are included in the GBDK. Their functions are described in details below.

crt0.o
Basic C runtime, with GB initialization routines, C support (mul, div, mod) and other essential things. This library is automatically linked with every program.

stdlib.o
Standard functions to interface the hardware in the GameBoy to C.

stdio.o, terminal.o
Libraries for basic text input/output. Implements standard functions from stdio, ctype and string.

drawing.o
Very primitive graphic library that allows to draw points to the screen, and to display images. The drawing area is limited because of the way the GB handles display.

1.10 stdlib.o

```
Library
  stdlib.o

Include files
  stdlib.h

Source files
  crt0.s

Functions
  void mode(int m);
    Change current working mode (M_DRAWING or M_TEXT).
    This is normally implicitely done when using library functions.

  void delay(int d);
    Small pause.

  void pause(int p);
    Longer pause.

  int joypad();
    Read the joypad status. Joypad keys are J_START, J_SELECT, J_B, J_A,
    J_DOWN, J_UP, J_LEFT and J_RIGHT.

  int waitpad(int mask);
    Wait for one of the specified joypad keys to be pressed.

  void waitpadup();
    Wait for the joypad to be released.

  void enable_interrupts();
  void disable_interrupts();
    Enable or Disable interrupts (must be enabled for displaying sprites).

  void display_on();
  void display_off();
    Switch screen on or off.

  void show_bkg();
  void hide_bkg();
    Show or hide the background display.

  void set_bkg_data(int first_tile, int nb_tiles, unsigned char *data);
    Set the data of part of the background tiles.
      -128 <= first_tile <= 127
      -128 <= first_tile+nb_tiles <= 127
      nb_tiles >= 1

  void set_bkg_tiles(int x, int y, int w, int h, unsigned char *tilelist);
    Set the tile number of part of the background.
      0 <= x <= 31
      0 <= y <= 31
      1 <= w <= 32-x
      1 <= h <= 32-y

  void scroll_bkg(int x, int y);
```

```
    Scroll the background.

  void show_window();
  void hide_window();
    Show or hide the window display.

  void show_sprites();
  void hide_sprites();
    Show or hide the sprites display.

  void sprites8x8();
  void sprites8x16();
    Set the size of all sprites.

  void set_sprite_data(int first_tile, int nb_tiles, unsigned char *data);
    Set the data of part of the sprite tiles.
      0 <= first_tile <= 255
      0 <= first_tile+nb_tiles <= 255
      nb_tiles >= 1

  void set_sprite_tile(int nb, int tile);
    Set the tile number of a sprite.
      0 <= nb <= 39
      0 <= tile <= 255

  void set_sprite_prop(int nb, int prop);
    Set the properties of a sprite. Sprite properties bits are S_PALETTE,
    S_FLIPX, S_FLIPY and S_PRIORITY.
      0 <= nb <= 39

  void move_sprite(int nb, int x, int y);
    Change the position of a sprite.
      0 <= nb <= 39
      0 <= x <= 255
      0 <= y <= 255
```

1.11 stdio.o, terminal.o

```
Library
  stdio.o, terminal.o

Include files
  stdio.h

Source files
  stdio.c
  terminal.s

Functions
  int atoi(char *s);
    Return the integer value of a numeric string.

  int abs(int num);
    Return the absolute value of an integer.
```

```
int isalpha(char c);
int isupper(char c);
int islower(char c);
int isdigit(char c);
int isspace(char c);
  Functions that classify character-coded integer values.

int toupper(char c);
int tolower(char c);
  Change character case.

int index(char *s, char *t);
  Find index of string t in s.

char *itoa(int n, char *s);
  Transform an integer in its ascii representation.

void printn(int number, int radix);
  Print a number in any radix.

char *reverse(char *s);
  Reverse a character string.

char *strcat(char *s1, char *s2);
  Concatenate s2 on the end of s1. s1 must be large enough. Return s1.

int strcmp(char *s1, char *s2);
  Compare strings:
    s1>s2: >0
    s1==s2: 0
    s1<s2: <0

char *strcpy(char *s1, char *s2);
  Copy string s2 to s1. s1 must be large enough. Return s1.

int strlen(char *s);
  Return length of string.

char *strncat(char *s1, char *s2, int n);
  Concatenate s2 on the end of s1. s1 must be large enough. At most n
  characters are moved. Return s1.

int strncmp(char *s1, char *s2, int n);
  Compare strings (at most n bytes):
    s1>s2: >0
    s1==s2: 0
    s1<s2: <0

char *strncpy(char *s1, char *s2, int n);
  Copy s2 to s1, truncating or null-padding to always copy n bytes. Return s1.

void puts(char *str);
  Print a string with a carriage return.

void print(char *str);
  Print a string without carriage return.
```

```
void printf(char *fmt, ...);
int scanf(char *fmt, ...);
  Print a formatted string. printf and scanf support the following types:
    %c char
    %d decimal int
    %o octal int
    %p pointer
    %s string
    %x hexadecimal int
  When waiting for a user input, a kind of keyboard appears at the bottom
  of the screen, which allows to enter characters. The following buttons are
  used:
    Arrow keys: Move the cursor
    A: Enter a character
    B: Delete a character
    START: End of line (carriage return)
    SELECT: Temporarily hide the keyboard

void putchar(char c);
  Print a character.

char getchar();
  Read a character.

char *gets(char *s);
  Read a string.

void gotoxy(int x, int y);
  Move the cursor to a specific position

int posx();
int posy();
  Return the current cursor position

void setchar(char c);
  Set the character at cursor position, without character interpretation
  ('\n' does not move to the next line) and without moving the cursor.
```

1.12 drawing.o

```
Library
  drawing.o

Include files
  graphics.h

Source files
  drawing.s

Functions
  void plot(int x, int y, int color, int mode);
    Draw a pixel on screen with specific color and mode. Colors are WHITE,
    LTGREY, DKGREY and BLACK. Modes are AND, OR, XOR and SOLID.

  void draw_image(unsigned char *data);
```

```
    Draws a complete image to screen. Image size must be 0x80 * 0x78 pixels.
```

1.13 Test programs

```
The test programs

Test programs in the examples directory:
space.s
  Assembly program that demonstrates the use of sprites, window, background,
  fixed-point values and more. The following keys are used:
    Arrow   keys: Change the speed (and direction) of the sprite
    Arrow   keys+A: Change the speed (and direction) of the window
    Arrow   keys+B: Change the speed (and direction) of the background
    START:  Open/close the door
    SELECT: Basic fading effect

sound.c
  Program for experimenting with the soung generator of the GB (to use on
  a real GB). The four different sound modes of the GB are available. It also
  demonstrates the use of bit fields in C (it's a quick hack, so don't expect too
  much from the code). The following keys are used:
    UP/DOWN:       Move the cursor
    RIGHT/LEFT:    Increment/decrement the value
    RIGHT/LEFT+A:  Increment/decrement the value by 10
    RIGHT/LEFT+B:  Set the value to maximum/minimum
    START:         Play the current mode's sound (or all modes if in control screen ↩
       )
    START+A:       Play a little music with the current mode's sound
    SELECT:        Change the sound mode (1, 2, 3, 4 and control)
    SELECT+A:      Dump the sound registers to the screen

sprite.c
  Program that demonstrates the use of sprite form C.

rpn.c
  Basic RPN calculator. Try entering expressions like 12 134*
  and then 1789+.

Test programs in the tst directory (from the lcc distribution.)
8q.c
  The classic 8 queens problem (place 8 queens on a chessboard so that
  none of them threaten the others).

array.c
  Test program with arrays.

init.c
  Test program with variable initializations.

sort.c
  Sorting algorithm that uses arrays and pointers.

struct.c
  Test program with structures.
```

```
test.c
  Test program for terminal and drawing libraries.
```

1.14 C & Assembly

```
Mixing C and assembly
For mixing C and assembly, you must use different files (you cannot embed
C code with assembly) and link them together. Here are the things you must
know:
  * A C identifier i will be called _i in assembly
  * Results are always returned into the HL register
  * Parameters are always passed on the stack (starting at SP+2 because the
    return address is also saved on the stack)
  * Assembly identifiers are exported using the .globl directive
  * Registers must be preserved across function calls (you must store them at
    function begin, and restore them at the end), except HL.

Example of how to mix assembly with C:
main.c
    main()
    {
      int i;
      int add(int, int);

      i = add(1, 3);
    }

add.s
    .globl _add
    _add:           ; int add(int a, int b)
      PUSH BC       ; Save used registers (except HL)
      PUSH DE
      LDA  HL,2(SP)
      LD   C,(HL)   ; Get a
      INC  HL
      LD   B,(HL)
      INC  HL
      LD   E,(HL)   ; Get b
      INC  HL
      LD   D,(HL)
      LD   H,D      ; Move DE into HL
      LD   L,E
      ADD  HL,BC    ; Add BC to HL
      POP  DE       ; Restore registers
      POP  BC
      RET            ; Return result into HL
```

1.15 ILBMtoGB

```
ILBMtoGB
ILBMtoGB allows conversion from a 4-color ILBM image into assembly or C code to
be included into a GB program. The image will be analysed and tiles that appear
```

```
more than once will be generated only once. The dump extension generates both
data for the tiles and a table for the mapping of tiles in the image.
This program is only present in this Amiga port. The UNIX version has a
modified version of Xloadimage instead. Since Xloadimage doesn't support IFF
ILBM, there was no point in porting it, so I sat down and wrote a native Amiga
application, hence it uses Amiga argument parsing. It will generate about the
same output as the xloadimage, but it will only read IFF ILBM.

You can specify the starting tile to use for the image. This allows to generate
data for different images that will use different sets of tiles.

An option allows to store the four common tiles (all black, all dark grey,
all light grey and all white) in tiles 0xFC to 0xFF, which is a strategic
location since it can be accessed with the same number (signed or unsigned)
for the window and the background. This is especially useful when you have
more that one image that uses these tiles.

Arguments
From          - The ILBM to convert. Must be 128 x 120 x 2!
To            - Destination file for the tiles.
                (Defaults to input file name plus extension ".c" or ".asm".)
Assembler     - Generate assembler source instead of C.
FirstTile     - The first tile number to use.
                (Defaults to 0.)
StandardTiles - Use standard tiles in 0xFC to 0xFF.
DataName      - Part of the label names in the generated data.
                (Defaults to "image".)
Flat          - Generate an image to use with draw_image() in drawing.o.
                Input must be 128 x 120 x 2!
Verbose       - Write verbose information while generating source code.

Most options have an abbreviation. By typing 'ILBMtoGB ?' will display the
argument string. If you don't enter any options, a short summary of the
available arguments will be displayed.
Examples
    ILBMtoGB sky.iff FirstTile 16 StandardTiles To sky.c DataName sky

    ILBMtoGB From foo FT 64 STD DN foo V
```

1.16 Gremlins

```
Errors
Messages of the type:
    u 0226
    a 0329
    u 0333
are error messages from the assembler. To see where these errors occur, you
should produce an assembly listing using the -Wa-l flag of lcc. An object file
is generated, but must be corrupted.
For more information on the different types of errors, read the asmlnk.doc file.

Messages of the type:
    ?ASlink-W-Undefined Global     .count referenced by module     Demo
are error messages from the linker. You probably forgot a library when linking,
An image file is generated, but must be corrupted.
```

Warnings
Do not declare initialized variables at the file level, except when they are read-only, because they will be located in ROM, e.g.

```
    int i1;          /* OK    : will be located in RAM */
    char *s1;        /* OK    : will be located in RAM */
    int i2 = 0;      /* Error : will be located in ROM */
    char *s2 = &quot;Hi&quot;; /* Error : will be located in ROM */

    void main() { ... }</CODE> </PRE>
```

Both terminal.o and drawing.o libraries use a lot of tiles and sprites from the GB. You should not use your own tiles or sprites with these libraries.

If you use both libraries in a same program, keep in mind that there will be a "mode switch" when using a function from a library after one of the other and all your work will be lost (if in drawing mode you use a terminal function, your drawing will be lost).

1.17 General GameBoy

CPU
The Game Boy uses a custom/updated/or modified Z80 processor. Comparing the Game Boy's Z80 instruction set with a book on the Z80 (circa 1982) shows that the GB Z80 has a few different instructions.

Screen
Physical screen: 160*144 VRAM screen image: 256*256
Screen scrolling is wrap around type; when a part of the image is off the screen it will be shown on the opposite side of the screen.

Although the screen can contain 1024 tiles, only 256 of them may be UNIQUE.
Each tile may have up to 4 colors. You may change the color of the pixel value. There are 4 shades of gray. You can select which shade you want for that pixel value. However, when you change the color for that pixel value EVERY tile that has a pixel with the same value will also be affected.
This is good for a routine which fades out the screen or performs a GLOWING effect of some kind.
The tile graphics are 8*8 pixels, each pixel contains 2 bits of data to create 4 numbers. Each number is the color value for that pixel.
The graphics are stored as interleaved bitmapped tiles.

A tile for an 'A' of color 1 with the background of color 0.

```
 .11111..   <- first plane
 ........   <- second plane
 11...11.
 ........
 11...11.
 ........
 1111111.   <- first plane
 ........   <- second plane
 11...11.
 ........
 11...11.
```

```
 ........
 11...11.
 ........
 ........
 ........

Graphics VRAM location for OBJ and BG tiles start at $8000 and end at $97FF

Sprites
40 Sprites! They may be 8*8 or 8*16.
Each sprite has up to 4 colors. There are 2 palettes to chose from
The sprites can be flipped on the X and/or Y axis
Sprite OAM ram is localted at $FE00 to $FE9F
Each sprite data contains 4 bytes of info. They are:
  Byte 1: Y screen position; 8 bits
  Byte 2: X screen position; 8 bits
  Byte 3: Character code; tile number $00-$FF
  Byte 4: Palette, X, Y, Priority; Most Significant 4 bits.
          First 4 bits are NOT USED!

          Bit 7 - Priority
          Bit 6 - Y flip
          Bit 5 - X flip
          Bit 4 - Palette number; 0,1
          Bit 3-0 - NOT USED!

Sound
There are only 2 channels; left and right.
But there are 4 different ways to produce sound:
  Sound 1: produces quadrangular wave patterns with sweep and envelope
           functions
  Sound 2: produces quadrangular wave patterns with envelope functions
  Sound 3: produces a voluntary wave pattern (samples can be possible if done
           right)
  Sound 4: produces white noise

You tell the channel which sound number you want to use and it will produce the
sound when you've set the according data.

ROM & RAM
Display RAM size: 64k bit
Work RAM size: 64k bit

$FFFF +----------------------------------+
      | ???                              |
$FFFE +----------------------------------+
      | Work and stack area (127 bytes)  |
$FF80 +----------------------------------+
      | Sound control registers          |
      +----------------------------------+
      | LCDC control registers           |
      +----------------------------------+
      | port/mode registers              |
$FF00 +----------------------------------+
      | OAM RAM (40*4 bytes)             |
$FE00 +----------------------------------+
      | ???                              |
```

```
$F000 +-----------------------------------+
      | ???                               |
$E000 +-----------------------------------+
      | Work area (8 kbyte RAM)           |
$C000 +-----------------------------------+
      | Expanded work area (8 kbyte RAM) |
$A000 +-----------------------------------+
      | Background display data (2)       |
$9C00 +-----------------------------------+
      | Background display data (1)       |
$9800 +-----------------------------------+
      | Character data                    |
$8000 +-----------------------------------+
      | User program area (32 kbyte ROM) |
$0000 +-----------------------------------+

There are 2 Memory Bank Controllers (MBC) that can be used. MBC1 is the
standard that is used on most cartridges.
MBC2 is used with cartridges which need Save-RAM.
It controls extended Save-RAM banks.

Extended RAM may go up to 256k bit.

MBC1 - When controlling ROM only you may read up to 16 megabits! (2 MBYTES)
       When controlling RAM only you may read up to 4 megabits (512 kbytes)
                                  and read up to 256kbit RAM

MBC2 - Controls Back-Up RAM (Save-RAM) (512 * 4 bit) which can be extended
       to 2 megabits (16 kbyte * 16)  256k byte

$FFFF +---------------------+
      | Internal RAM        |
$C000 +---------------------+
      | Expanded banked RAM |
$A000 +---------------------+
      | Display RAM         |
$8000 +---------------------+
      | Banked ROM          |
$4000 +---------------------+
      | Home ROM            |
$0000 +---------------------+

Writing @$01 - #$0F in CPU address $2000 - $3FFF will select ROM bank.
Writing @$00 - #$03 in CPU address $4000 - $5FFF will select RAM bank.

Bank switching
The Z80 can only work with 16 bit addresses $0-$FFFF, so to access the other
data you must trick the machine into pointing to another piece of memory.

ROM is located from $0000 - $7FFF, RAM is from $8000-$FFFF

All game programs are ROM so we know it is from $0000-$7FFF
But the Game Boy has a fixed memory area from $0000-$3FFF; when you
access it, it will always be BANK 0. It is called the FIXED HOME ADDRESS.

That means the only other ROM addresses available are $4000-$7FFF.
```

```
  Bank 0 is read by the CPU as being at $0000-$3FFF
  Bank 1 is read by the CPU as being at $4000-$7FFF
  Bank 2 is read by the CPU as being at $4000-$7FFF
  Bank 3 is read by the CPU as being at $4000-$7FFF

See the pattern? Only the FIXED HOME ADDRESS has it’s own special location.
Banks and addresses starting at $4000 is called the CPU address.
  CPU Address $014000 is actually Bank #$01 address $4000
  CPU Address $014000 is equal to ROM address (offset) $004000
  CPU Address $024000 is equal to ROM address (offset) $008000
  CPU Address $044000 is equal to ROM address (offset) $010000

The CPU uses the CPU ADDRESS.

Switching banks
Using MBC1 (Memory Bank Controller 1):
Writing to ROM Address (CPU FIXED HOME ADDRESS) $2000-$3FFF the ROM bank can be
selected. The values are from #$01-#$0F
  LD A,#$01
  LD ($2000),A     <- this selects ROM BANK #$01


Writing to ROM Address (CPU FIXED HOME ADDRESS) $4000-$5FFF the RAM bank can be
selected. The values are from #$00-#$03
  LD A,#$03
  LD ($4000),A     <- this select RAM BANK #$03

Using MBC2 (Memory Bank Controller 2):
Writing to ROM Address (CPU FIXED HOME ADDRESS) $2100-$21FF the ROM bank can be
select. The values are from #$01-#$0F

GameBoy cartridge information
The Internal Info block begins at $100 and it’s format is as follows:
  $100-$101 - 00 C3  (2 bytes)
  $102-$102 - Lo Hi  (Start Address for Game, usually $150 it would be written
                      as 50 01)
  $100-$133 - Nintendo Character Area, if this does not exist the game
              will not run!
              000100: 00 C3 50 01 CE ED 66 66 CC 0D 00 0B 03 73 00 83
              000110: 00 0C 00 0D 00 08 11 1F 88 89 00 0E DC CC 6E E6
              000120: DD DD D9 99 BB BB 67 63 6E 0E EC CC DD DC 99 9F
              000130: BB B9 33 3E
  $134-$143 - Title Registration Area (title of the game in ASCII)
  $144-$146 - NOT USED
  $147 - CARTRIDGE TYPE
         0 - ROM ONLY
         1 - ROM+MBC1
         2 - ROM+MBC1+RAM
         3 - ROM+MBC1+RAM+BATTERY
         5 - ROM+MBC2
         6 - ROM+MBC2+BATTERY
  $148 - ROM SIZE
         0 - 256kbit
         1 - 512kbit
         2 - 1M-Bit
         3 - 2M-Bit
         4 - 4M-Bit
```

```
$149 - RAM SIZE
       0 - NONE
       1 - 16kbit
       2 - 64kbit
       3 - 256kbit

$14A-$14B - Maker Code - 2 bytes
$14C - Version Number
$14D - Complement Check
$14E-$14F - Checksum HI-LO (2 bytes in Big Endian format, high byte first)
```

1.18 Instruction set

```
GameBoy Instruction set summary

The GB processor is very similar to the Z80, although some of the instructions
are missing and some ther have been added. Also, the second set of registers
(BC', DE', HL', AF') and the index registers (IX, IY) are missing and
consequently, there are no DD and FD opcode tables. Finally, I/O ports are gone
and so are all IN/OUT opcodes.

The internal 8-bit registers are A, B, C, D, E, F, H & L. Theses registers may
be used in pairs for 16-bit operations as AF, BC, DE & HL. The two remaining
16-bit registers are the program counter (PC) and the stack pointer (SP).
The F register holds the cpu flags. The operation of these flags is identical
to their Z80 relative. The lower four bits of this register always read zero
even if written with a one.

+---------------------------------------+
|             Flag Register             |
+----+----+----+----+----+----+----+----+
| 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
+----+----+----+----+----+----+----+----+
| Z  | N  | HC | CY | 0  | 0  | 0  | 0  |
+----+----+----+----+----+----+----+----+

The GameBoy CPU is based on a subset of the Z80 microprocessor. A summary of
these commands is given below.
(This information is incomplete and have serious errors and flaws!)

+--------------+-------------------------+----------------+----------------------+
| Mnemonic     | Symbolic Operation      | Comments       | CPU Clocks           |
+--------------+-------------------------+----------------+----------------------+


8-Bit Loads
+--------------+-------------------------+----------------+----------------------+
| LD r,s       | r <- s                  | s=r,n,(HL)     | r=4, n=8, (HL)=8     |
+--------------+-------------------------+----------------+----------------------+
| LD d,r       | d <- r                  | d=r,(HL)       | r=4, (HL)=8          |
+--------------+-------------------------+                +----------------------+
| LD d,n       | d <- n                  |                | r=8, (HL)=12         |
+--------------+-------------------------+----------------+----------------------+
| LD A,(ss)    | A <- (ss)               | ss=BC,DE,HL,nn | [BC,DE,HL]=8,nn=16   |
```

LD (dd),A	(dd) <- A	dd=BC,DE,HL,nn	
LD HL,(SP+e)	HL <- (SP+e)		12
LD A,(HL-)	A <- (HL), HL <- HL - 1		8
LD (HL-),A	(HL) <- A, HL <- HL - 1		8
LD A,(HL+)	A <- (HL), HL <- HL + 1		8
LD (HL+),A	(HL) <- A, HL <- HL + 1		8
LDH (n),A	($FF00+n) <- A		12
LDH A,(n)	A <- ($FF00+n)		12
LDH (C),A	($FF00+C) <- A		12
LDH A,(C)	A <- ($FF00+C)		12
LD (nn),A	(nn) <- A		?
LD (nn),SP	(nn) <- (SP)		?

16-Bit Loads

LD dd,nn	dd <- nn	dd=BC,DE,HL,SP	12
LD (nn),SP	(nn) <- SP		20
LD SP,HL	SP <- HL		8
LDA SP,n(SP)	SP <- SP + n		?
LDA HL,n(SP)	HL <- SP + n		?
PUSH ss	(SP-1) <- ssh, (SP-2) <- ssl, SP <- SP-2	ss=BC,DE,HL,AF	16
POP dd	ddl <- (SP), ddh <- (SP+1), SP <- SP+2	dd=BC,DE,HL,AF	12

8-Bit ALU

ADD A,s	A <- A + s	s=r,n,(HL)	r=4, n=8, (HL)=8
ADC A,s	A <- A + s + CY		
SUB A,s	A <- A - s		

```
| SBC A,s       | A <- A - s - CY          |                |                      |
+---------------+--------------------------+                |                      |
| AND A,s       | A <- A AND s             |                |                      |
+---------------+--------------------------+                |                      |
| OR A,s        | A <- A OR s              |                |                      |
+---------------+--------------------------+                |                      |
| XOR A,s       | A <- A XOR s             |                |                      |
+---------------+--------------------------+                |                      |
| CP A,s        | A - s                    |                |                      |
+---------------+--------------------------+----------------+----------------------+
| INC s         | s <- s + 1               | s=r,(HL)       | r=4, (HL)=12         |
+---------------+--------------------------+                |                      |
| DEC s         | s <- s - 1               |                |                      |
+---------------+--------------------------+----------------+----------------------+


16-Bit Arithmetic
+---------------+--------------------------+----------------+----------------------+
| ADD HL,ss     | HL <- HL + ss            | ss=BC,DE,HL,SP | 8                    |
+---------------+--------------------------+                +----------------------+
| ADC HL,ss     | HL <- HL + ss            |                | 8                    |
+---------------+--------------------------+                +----------------------+
| INC ss        | ss <- ss + 1             |                | 8                    |
+---------------+--------------------------+                +----------------------+
| DEC ss        | ss <- ss - 1             |                | 8                    |
+---------------+--------------------------+----------------+----------------------+


Miscellaneous
+---------------+--------------------------+----------------+----------------------+
| SWAP A,s      |                          |  s=r,(HL)      | r=8, (HL)=16         |
+---------------+--------------------------+----------------+----------------------+
| DAA           | Convert A to packed BCD  |                | 4                    |
+---------------+--------------------------+                +----------------------+
| CPL           | A <- /A                  |                | 4                    |
+---------------+--------------------------+                +----------------------+
| CCF           | CY <- /CY                |                | 4                    |
+---------------+--------------------------+                +----------------------+
| SCF           | CY <- 1                  |                | 4                    |
+---------------+--------------------------+                +----------------------+
| NOP           | No operation             |                | 4                    |
+---------------+--------------------------+                +----------------------+
| HALT          | Halt CPU                 |                |                      |
+---------------+--------------------------+                +----------------------+
| STOP          | Halt CPU                 |                |                      |
+---------------+--------------------------+                +----------------------+
| DI            | Disable Interrupts       |                | 4                    |
+---------------+--------------------------+                +----------------------+
| EI            | Enable Interrupts        |                | 4                    |
+---------------+--------------------------+                +----------------------+
| RETI          | Return and enable int.   |                | ?                    |
+---------------+--------------------------+----------------+----------------------+


Rotates & Shifts
+---------------+--------------------------+----------------+----------------------+
| RLC A,s       | Rotate left              | s=A,r,(HL)     | A=4, r=8, (HL)=16    |
```

```
+---------------+-------------------------+                |                      |
| RL A,s        | Rotate left thru CY     |                |                      |
+---------------+-------------------------+                |                      |
| RRC A,s       | Rotate right            |                |                      |
+---------------+-------------------------+                |                      |
| RR A,s        | Rotate right thru CY    |                |                      |
+---------------+-------------------------+----------------+----------------------+
| SLA A,s       | Shift left aritmetic    | s=r,(HL)       | r=8, (HL)=16         |
+---------------+-------------------------+                |                      |
| SRA A,s       | Shift right aritmetic   |                |                      |
+---------------+-------------------------+                |                      |
| SRL A,s       | Shift left logical      |                |                      |
+---------------+-------------------------+----------------+----------------------+
```

```
Bit Opcodes
+---------------+-------------------------+----------------+----------------------+
| BIT b,s       | Z <- /sb                | Z is zero flag | s=r,(HL)r=8, (HL)=16 |
+---------------+-------------------------+                |                      |
| SET b,s       | sb <- 1                 |                |                      |
+---------------+-------------------------+                |                      |
| RES b,s       | sb <- 0                 |                |                      |
+---------------+-------------------------+----------------+----------------------+
```

```
Jumps
+---------------+-------------------------+----------------+----------------------+
| JP nn         | PC <- nn                |                | 16                   |
+---------------+-------------------------+                +----------------------+
| JP cc,nn      | If cc True, PC <- nn    |                | If cc True, 16       |
|               | Else Continue           |                | Else 12              |
+---------------+-------------------------+                +----------------------+
| JP (HL)       | PC <- HL                |                | 4                    |
+---------------+-------------------------+                +----------------------+
| JR e          | PC <- PC + e            |                | 12                   |
+---------------+-------------------------+                +----------------------+
| JR cc,e       | If cc True, PC <- PC + e|                | If cc True, 12       |
|               | Else continue           |                | Else 8               |
+---------------+-------------------------+----------------+----------------------+
```

```
Calls
+---------------+-------------------------+----------------+----------------------+
| CALL nn       | (SP-1) <- PCh,          |                | 40                   |
|               | (SP-2) <- PCl,          |                |                      |
|               | PC <- nn, SP <- SP-2    |                |                      |
+---------------+-------------------------+----------------+----------------------+
| CALL cc,nn    | If cc True, CALL nn     |                | If cc True, 40       |
|               | Else Continue           |                | Else 12              |
+---------------+-------------------------+----------------+----------------------+
```

```
Restarts
+---------------+-------------------------+----------------+----------------------+
| RST f         | (SP-1) <- PCh,          |                | 32                   |
|               | (SP-2) <- PCl,          |                |                      |
|               | PCh <- 0,               |                |                      |
```

	PCl <- f, SP <- SP-2		

Returns

RET	PCl <- (SP), PCh <- (SP+1), SP <- SP+2		16
RET cc	If cc True, RET Else continue		If cc True, 16 Else 8
RETI	Return from interrupt		16

Terminology

b	A bit number in any 8-bit register or memory location.
CY	Carry flag.
cc	Flag condition code: C, NC, Z or NZ.
d	Any 8-bit destination register or memory location.
dd	Any 16-bit destination register or memory location.
e	8-bit signed 2's complement displacement.
f	8 special call locations in page zero.
HC	Half-carry flag.
N	Subtraction flag.
NC	Not carry flag.
NZ	Not zero flag.
n	Any 8-bit binary number.
nn	Any 16-bit binary number.
r	Any 8-bit register. (A, B, C, D, E, H or L.)
s	Any 8-bit source register or memory location.
sb	A bit in a specific 8-bit register or memory location.
ss	Any 16-bit source register or memory location.
Z	Zero Flag.

1.19 Custom Registers

```
GameBoy Custom Registers

-------------------------------------------------------------------------------
   Address  - $FF00
   Name     - P1
   Contents - Register for reading joy pad info.    (R/W)

           Bit 7 - Not used
           Bit 6 - Not used
           Bit 5 - P15 out port
           Bit 4 - P14 out port
           Bit 3 - P13 in port
           Bit 2 - P12 in port
           Bit 1 - P11 in port
           Bit 0 - P10 in port

         This is a very strange way of reading joypad info.
         There are only 8 possible button/switches on the Game Boy.
         A, B, Select, Start, Up, Down, Left, Right.
         Why they made their joypad registers in this way I'll never know.
         They could have used all 8 bits and you just read which one is on.

         This is the matrix layout for register $FF00:


           P14                P15
            |                  |
--P10-------O-Right------------O-A---------
            |                  |
--P11-------O-Left-------------O-B---------
            |                  |
--P12-------O-Up---------------O-Select----
            |                  |
--P13-------O-Down-------------O-Start-----
            |                  |


          This is the logic in reading joy pad data:

          Turn on P15 (bit 5) in $ff00
          Wait a few clock cycles
          read $ff00 into A
          invert A   - same as EOR #$FF - just reverse all bits
                       apparently the joy pad info returned is like the C64
                       info. 0 means on, 1 means off. But logic tells us
                       that it should be the other way around. So to make it
                       less confusing we just flip the bits!

          AND A with #$0F - get only the first four bits
                            By turning on P15 we are trying to read column
                            P15 in the matrix layout. It contains A,B,SEL,STRT

          SWAP A - #$3f becomes #$f3, it swaps hi<->lo nibbles
```

```
          store A in B for backup


          Turn on P14 (bit 4) in $ff00
          Wait a few more clock cycles
          read $ff00 into A
          invert A - just as above
          AND A with #$0F - get first 4 bits
                          - By turning on P14 we get the data for column P14
                            in the matrix layout. It contains U,D,L,R

          OR A with B - put the two values together.

          turn on P14 and P15 in $ff00 to reset.

          The button values using the above method are such:
          $80 - Start            $8 - Down
          $40 - Select           $4 - Up
          $20 - B                $2 - Left
          $10 - A                $1 - Right

          Let's say we held down A, Start, and Up.
          The value returned in accumulator A would be $94


          Let's see this method in action!
          Game: Ms. Pacman
          Address: $3b1

0003B1: 0003B1: 3E 20          LD A,#$20       <- bit 5 = $20
0003B3: 0003B3: EA 00 FF       LD ($FF00),A    <- turn on P15
0003B6: 0003B6: FA 00 FF       LD A,($FF00)
0003B9: 0003B9: FA 00 FF       LD A,($FF00)    <- wait a few cycles
0003BC: 0003BC: 2F             CPL             <- complement (invert) EOR #$ff
0003BD: 0003BD: E6 0F          AND #$0F        <- get only first 4 bits
0003BF: 0003BF: CB 37          SWAP A          <- swap it
0003C1: 0003C1: 47             LD B,A          <- store A in B
0003C2: 0003C2: 3E 10          LD A,#$10       <- bit 4 = $10
0003C4: 0003C4: EA 00 FF       LD ($FF00),A    <- turn on P14
0003C7: 0003C7: FA 00 FF       LD A,($FF00)
0003CA: 0003CA: FA 00 FF       LD A,($FF00)
0003CD: 0003CD: FA 00 FF       LD A,($FF00)
0003D0: 0003D0: FA 00 FF       LD A,($FF00)
0003D3: 0003D3: FA 00 FF       LD A,($FF00)
0003D6: 0003D6: FA 00 FF       LD A,($FF00)    <- Wait a few MORE cycles
0003D9: 0003D9: 2F             CPL             <- complement (invert)
0003DA: 0003DA: E6 0F          AND #$0F        <- get first 4 bits
0003DC: 0003DC: B0             OR B            <- put A and B together


  The following routine is common on SNES as well. It clarifies that you've
  only pressed the specified button(s) once every other frame. That way the
  Joypad is less sensitive to wrong/bad/false movements.

0003DD: 0003DD: 57             LD D,A          <- store A in D
0003DE: 0003DE: FA 8B FF       LD A,($FF8B)    <- read old joy data from ram
0003E1: 0003E1: AA             XOR D           <- toggle w/current button bit
```

```
0003E2: 0003E2: A2             AND D           <- get current button bit back
0003E3: 0003E3: EA 8C FF       LD ($FF8C),A    <- save in new Joydata storage
0003E6: 0003E6: 7A             LD A,D          <- put original value in A
0003E7: 0003E7: EA 8B FF       LD ($FF8B),A    <- store it as old joy data


0003EA: 0003EA: 3E 30          LD A,#$30       <- turn on P14 and P15
0003EC: 0003EC: EA 00 FF       LD ($FF00),A    <- RESET Joypad?!
0003EF: 0003EF: C9             RET             <- Return from Subroutine

------------------------------------------------------------------------------

   Address  - $FF01
   Name     - SB
   Contents - Serial transfer data (R/W)

              8 Bits of data to be read/written

   Address  - $FF02
   Name     - SC
   Contents - SIO control  (R/W)

              Bit 7 - Transfer start flag
                      0: Non transfer
                      1: Start transfer

              Bit 0 - Shift Clock
                      0: External Clock
                      1: Internal Clock

------------------------------------------------------------------------------

   Address  - $FF04
   Name     - DIV
   Contents - Divider Register (R/W)

------------------------------------------------------------------------------

   Address  - $FF05
   Name     - TIMA
   Contents - Timer counter (R/W)

              The timer generates an interrupt when it overflows.

   Address  - $FF06
   Name     - TMA
   Contents - Timer Modulo (R/W)

              When the TIMA overflows, this data will be loaded.

   Address  - $FF07
   Name     - TAC
   Contents - Timer Control

              Bit 2 - Timer Stop
                      0: Stop Timer
```

```
                    1: Start Timer

            Bits 1+0 - Input Clock Select
                       00: 4.096 khz
                       01: 262.144 khz
                       10: 65.536 khz
                       11: 16.384 khz

------------------------------------------------------------------------------

   Address  - $FF0F
   Name     - IF
   Contents - Interrupt Flag (R/W)

            Bit 4: Transition from High to Low of Pin number P10-P13
            Bit 3: Serial I/O transfer end
            Bit 2: Timer Overflow
            Bit 1: LCDC (see STAT)
            Bit 0: V-Blank

   Address  - $FFFF
   Name     - IE
   Contents - Interrupt Enable (R/W)

            Bit 4: Transition from High to Low of Pin number P10-P13
            Bit 3: Serial I/O transfer end
            Bit 2: Timer Overflow
            Bit 1: LCDC (see STAT)
            Bit 0: V-Blank

            0: disable
            1: enable

   Address  - XXXX (CPU instruction command)
   Name     - IME
   Content  - Interrupt Master Enable

            To prohibit ALL interrupts use CPU instruction DI
            To acknowledge interrupt settings use CPU instruction EI
            DI - Disable Interrupts
            EI - Enable Interrupts

   The priority and jump address for the above 5 interrupts are:

    Interrupt        Priority        Start Address

    V-Blank             1              $0040
    LCDC Status         2              $0048 - Modes 0, 01, 10
                                               LYC=LY coincide (selectable)

    Timer Overflow      3              $0050
    Serial Transfer     4              $0058 - when transfer is complete
    Hi-Lo Of Pin        5              $0060

    * When more than 1 interrupts occur at the same time ONLY the interrupt
      with the highest priority can be acknowledged.
      When an interrupt is used a '0' should be stored in the IF register
```

```
      before the IE register is set.

------------------------------------------------------------------------------

   Address  - $FF40
   Name     - LCDC
   Contents - LCD Control (R/W)

              Bit 7 - LCD Control Operation
                      0: Stop completely (no picture on screen)
                      1: operation

              Bit 6 - Window Screen Display Data Select
                      0: $9800-$9BFF
                      1: $9C00-$9FFF

              Bit 5 - Window Display
                      0: off
                      1: on

              Bit 4 - BG Character Data Select
                      0: $8800-$97FF
                      1: $8000-$8FFF <- Same area as OBJ

              Bit 3 - BG Screen Display Data Select
                      0: $9800-$9BFF
                      1: $9C00-$9FFF

              Bit 2 - OBJ Construction
                      0: 8*8
                      1: 8*16

              Bit 1 - OBJ Display
                      0: off
                      1: on

              Bit 0 - BG Display
                      0: off
                      1: on


   Address  - $FF41
   Name     - STAT
   Contents - LCDC Status   (R/W)

              Bits 6-3 - Interrupt Selection By LCDC Status

              Bit 6 - LYC=LY Coincidence (Selectable)
              Bit 5 - Mode 10
              Bit 4 - Mode 01
              Bit 3 - Mode 00
                      0: Non Selection
                      1: Selection

              Bit 2 - Coincidence Flag
                      0: LYC not equal to LCDC LY
```

```
                1: LYC = LCDC LY

        Bit 1-0 - Mode Flag
                  00: Entire Display Ram can be accessed
                  01: During V-Blank
                  10: During Searching OAM-RAM
                  11: During Transfering Data to LCD Driver


   STAT shows the current status of the LCD controller.
   Mode 00: When the flag is 00 it is the H-Blank period and the CPU can
            access the display RAM ($8000-$9FFF)
            When it is not equal the display ram is being used by the
            LCD controller

   Mode 01: When the flag is 01 it is the V-Blank period and the CPU can
            access the display RAM ($800-$9FFF)

   Mode 10: When the flag is 10 then the OAM is being used ($FE00-$FE90)
            The CPU cannot access the OAM during this period

   Mode 11: When the flag is 11 both the OAM and CPU are being used.
            The CPU cannot access either during this period

-----------------------------------------------------------------------------

  Address  - $FF42
  Name     - SCY
  Contents - Scroll Y   (R/W)

            8 Bit value $00-$FF to scroll BG Y screen position

  Address  - $FF43
  Name     - SCX
  Contents - Scroll X   (R/W)

            8 Bit value $00-$FF to scroll BG X screen position

  Address  - $FF44
  Name     - LY
  Contents - LCDC Y-Coordinate (R)

          The LY indicates the vertical line to which the present data
          is transferred to the LCD Driver
          The LY can take on any value between 0 through 153. The values
          between 144 and 153 indicate the V-Blank period. Writing will
          reset the counter.

          This is just a RASTER register. The current line is thrown
          into here. But since there are no RASTERS on an LCD display.....
          it's called the LCDC Y-Coordinate.

  Address  - $FF45
  Name     - LYC
  Contents - LY Compare  (R/W)

          The LYC compares itself with the LY. If the values are the same
```

```
         it causes the STAT to set the coincident flag.

Address  - $FF47
Name     - BGP
Contents - BG Palette Data  (W)

           Bit 7-6 - Data for Dot Data 11
           Bit 5-4 - Data for Dot Data 10
           Bit 3-2 - Data for Dot Data 01
           Bit 1-0 - Data for Dot Data 00

           This selects the shade of gray you what for your BG pixel.
           Since each pixel uses 2 bits, the corresponding shade will
           be selected from here. The Background Color (00) lies at
           Bits 1-0, just put a value from 0-$3 to change the color.

Address  - $FF48
Name     - OBP0
Contents - Object Palette 0 Data (W)

           This selects the colors for sprite palette 0.
           It works exactly as BGP ($FF47).
           See BGP for details.

Address  - $FF49
Name     - OBP1
Contents - Object Palette 1 Data (W)

           This Selects the colors for sprite palette 1.
           It works exactly as BGP ($FF47).
           See BGP for details.


Address  - $FF4A
Name     - WY
Contents - Window Y Position  (R/W)

           0 <= WY <= 143

           WY must be greater than or equal to 0 and must be less than
           or equal to 143.

Address  - $FF4B
Name     - WX
Contents - Window X Position  (R/W)

           7 <= WX <= 166

           WX must be greater than or equal to 7 and must be less than
           or equal to 166.


           Lets say WY = 80 and WX = 80.
           The window would be positioned as so:

            0                  80                           159
            _________________________________________________
```

```
         0 |                   |                              |
           |                   |                              |
           |                   |                              |
           |                   |                              |
           |                   |                              |
           |                   |                              |
           |                   |                              |
           |                   |                              |
           |                   |                              |
           |                   |80                            |
        80 |-------------------+------------------------------|
           |                80 |                              |
           |                   |                              |
           |                   |      Window Display          |
           |                   |                              |
           |                   |                              |
           |                   |           Here               |
           |                   |                              |
           |                   |                              |
           |                   |                              |
       143 |___________________|______________________________|


       OBJ Characters (Sprites) can still enter the window
       So can BG characters
```

```
------------------------------------------------------------------------------

   Address  - $FF46
   Name     - DMA
   Contents - DMA Transfer and Start Address (W)

   The DMA Transfer (40*28 bit) from internal ROM or RAM ($0000-$F19F)
   to the OAM (address $FE00-$FE9F) can be performed. It takes 160 nano-seconds
   for the transfer.

   40*28 bit = #140  or #$8C.  As you can see, it only transfers $8C bytes
   of data. OAM data is $A0 bytes long, from $0-$9F.

   But if you examine the OAM data you see that 4 bits are not in use.

   40*32 bit = #$A0, but since 4 bits for each OAM is not used it's
   40*28 bit.

   It transfers all the OAM data to OAM RAM.

   The DMA transfer start address can be designated every $100 from address
   $0000-$F100.   That means $0000, $0100, $0200, $0300....

   Example program:
      DI              <- Disable Interrupt
      LD A,#$04       <- transfer data from $0400
      LD ($FF46),A    <- put A into DMA registers
      LD A,#40        <- #40 is the value to wait for. we need to wait 160
Wait:                 <- nano seconds
      DEC A           <- decrease A by 1
      JR NZ,Wait      <- branch if Not Zero to Wait
```

```
      EI              <- Enable Interrupt
      RET             <- RETurn from sub-routine

-------------------------------------------------------------------------------

   Address  - $FF10
   Name     - NR 10
   Contents - Sound Mode 1 register, Sweep register (R/W)

              Bit 6-4 - Sweep Time
              Bit 3 - Sweep Increase/Decrease
                      0: Addition    (frequency increases)
                      1: Subtraction (frequency increases)
              Bit 2-0 - Number of sweep shift (# 0-7)

              Sweep Time:

              000: sweep off
              001: 7.8 ms
              010: 15.6 ms
              011: 23.4 ms
              100: 31.3 ms
              101: 39.1 ms
              110: 46.9 ms
              111: 54.7 ms


   Address  - $FF11
   Name     - NR 11
   Contents - Sound Mode 1 register, Sound length/Wave pattern duty (R/W)

              Only Bits 7-6 can be read.

              Bit 7-6 - Wave Pattern Duty
              Bit 5-0 - Sound length data (# 0-63)

              Wave Duty:

              00: 12.5%
              01: 25%
              10: 50%
              11: 75%

   Address  - $FF12
   Name     - NR 12
   Contents - Sound Mode 1 register, Envelope (R/W)

              Bit 7-4 - Initial value of envelope
              Bit 3 - Envelope UP/DOWN
                      0: Decrease
                      1: Range of increase
              Bit 2-0 - Number of envelope sweep (# 0-7)

              Initial value of envelope is from %0000 to %1111

   Address  - $FF13
   Name     - NR 13
```

```
Contents - Sound Mode 1 register, Frequency lo (W)

           lower 8 bits of 11 bit frequency.
           Next 3 bit or in NR 14 ($FF14)

Address  - $FF14
Name     - NR 14
Contents - Sound Mode 1 register, Frequency hi (R/W)

           Only Bit 6 can be read.

           Bit 7 - Initial (when set, sound restarts)
           Bit 6 - Counter/consecutive selection
           Bit 2-0 - Frequency's higher 3 bits

Address  - $FF16
Name     - NR 21
Contents - Sound Mode 2 register, Sound Length; Wave Pattern Duty (R/W)

           Only bits 7-6 can be read.

           Bit 7-6 - Wave pattern duty
           Bit 5-0 - Sound length (# 0-63)

Address  - $FF17
Name     - NR 22
Contents - Sound Mode 2 register, envelope (R/W)

           Bit 7-4 - Initial envelope value
           Bit 3 - Envelope UP/DOWN
                   0: decrease
                   1: range of increase
           Bit 2-0 - Number of envelope step (# 0-7)


Address  - $FF18
Name     - NR 23
Contents - Sound Mode 2 register, frequency lo data (W)

           Frequency's lower 8 bits of 11 bit data
           Next 3 bits are in NR 14 ($FF19)

Address  - $FF19
Name     - NR 24
Contents - Sound Mode 2 register, frequency hi data (R/W)

           Only bit 6 can be read.

           Bit 7 - Initial
           Bit 6 - Counter/consecutive selection
           Bit 2-0 - Frequency's higher 3 bits

Address  - $FF1A
Name     - NR 30
Contents - Sound Mode 3 register, Sound on/off (R/W)

           Only bit 7 can be read
```

```
            Bit 7 - Sound OFF
                    0: Sound 3 output stop
                    1: Sound 3 output OK

Address  - $FF1B
Name     - NR 31
Contents - Sound Mode 3 register, sound length (R/W)

           Bit 7-0 - Sound length

Address  - $FF1C
Name     - NR 32
Contents - Sound Mode 3 register, Select output level

           Only bits 6-5 can be read

           Bit 6-5 - Select output level
                     00: Mute
                     01: Produce Wave Pattern RAM Data as it is
                         (4 bit length)
                     10: Produce Wave Pattern RAM data shifted once to the
                         RIGHT (1/2)  (4 bit length)
                     11: Produce Wave Pattern RAM data shifted twice to the
                         RIGHt (1/4)  (4 bit length)

    * - Wave Pattern RAM is located from $FF30-$FF3f

Address  - $FF1D
Name     - NR 33
Contents - Sound Mode 3 register, frequency's lower data (W)

           Lower 8 bits of an 11 bit frequency

Address  - $FF1E
Name     - NR 34
Contents - Sound Mode 3 register, frequency's higher data (R/W)

           Only bit 6 can be read.

           Bit 7 - Initial flag
           Bit 6 - Counter/consecutive flag
           Bit 2-0 - Frequency's higher 3 bits



Address  - $FF20
Name     - NR 41
Contents - Sound Mode 4 register, sound length (R/W)

           Bit 5-0 - Sound length data (# 0-63)


Address  - $FF21
Name     - NR 42
Contents - Sound Mode 4 register, envelope (R/W)
```

```
            Bit 7-4 - Initial value of envelope
            Bit 3 - Envelope UP/DOWN
                    0: decrease
                    1: range of increase
            Bit 2-0 - number of envelope step (# 0-7)

   Address  - $FF22
   Name     - NR 43
   Contents - Sound Mode 4 register, polynomial counter (R/W)

            Bit 7-4 - Selection of the shift clock frequency of the
                      polynomial counter
            Bit 3 - Selection of the polynomial counter's step
            Bit 2-0 - Selection of the dividing ratio of frequencies

            Selection of the dividing ratio of frequencies:
            000: f * 1/2^3 * 2
            001: f * 1/2^3 * 1
            010: f * 1/2^3 * 1/2
            011: f * 1/2^3 * 1/3
            100: f * 1/2^3 * 1/4
            101: f * 1/2^3 * 1/5
            110: f * 1/2^3 * 1/6
            111: f * 1/2^3 * 1/7           f = 4.194304 Mhz

            Selection of the polynomial counter step:
            0: 15 steps
            1: 7 steps

            Selection of the shift clock frequency of the polynomial
            counter:

            0000: dividing ratio of frequencies * 1/2
            0001: dividing ratio of frequencies * 1/2^2
            0010: dividing ratio of frequencies * 1/2^3
            0011: dividing ratio of frequencies * 1/2^4
                  :                           :
                  :                           :
                  :                           :
            0101: dividing ratio of frequencies * 1/2^14
            1110: prohibited code
            1111: prohibited code

   Address  - $FF30
   Name     - NR 30
   Contents - Sound Mode 4 register, counter/consecutive; inital (R/W)

            Only bit 6 can be read.

            Bit 7 - Inital
            Bit 6 - Counter/consecutive selection


   Address  - $FF24
   Name     - NR 50
   Contents - Channel control / ON-OFF / Volume (R/W)
```

```
            Bit 7 - Vin->SO2 ON/OFF
            Bit 6-4 - SO2 output level (volume) (# 0-7)
            Bit 3 - Vin->SO1 ON/OFF
            Bit 2-0 - SO1 output level (volume) (# 0-7)

            Vin->SO1 (Vin->SO2)

            By synthesizing the sound from sound 1 through 4, the voice
            input from Vin terminal is put out.
            0: no output
            1: output OK

    Address  - $FF25
    Name     - NR 51
    Contents - Selection of Sound output terminal (R/W)

             Bit 7 - Output sound 4 to SO2 terminal
             Bit 6 - Output sound 3 to SO2 terminal
             Bit 5 - Output sound 2 to SO2 terminal
             Bit 4 - Output sound 1 to SO2 terminal
             Bit 3 - Output sound 4 to SO1 terminal
             Bit 2 - Output sound 3 to SO1 terminal
             Bit 1 - Output sound 2 to SO1 terminal
             Bit 0 - Output sound 0 to SO1 terminal

    Address  - $FF26
    Name     - NR 52
    Contents - Sound on/off (R/W)

             Only Bit 7, 3-0 can be read.

             Bit 7 - All sound on/off
                     0: stop all sound circuits
                     1: operate all sound circuits
             Bit 3 - Sound 4 ON flag
             Bit 2 - Sound 3 ON flag
             Bit 1 - Sound 2 ON flag
             Bit 0 - Sound 1 ON flag
```

1.20 Disclaimer

```
Disclaimer
Everything in this document is a big hoax!
Any resemblance with anything in real life is coincidental.
So to speak, the options presented for the programs do only sometimes
have the effect described, and when they do, the side effects are probably
noticed clearly as the programs also trashes your hard disks and blows your
monitor into pieces. (It has also been reported that the programs will install
viruses in you power supply unit, and from there infect your shaving machine,
turning it into a vicious murderer...)
The information about the Z80 look-alike is included to make this file bigger,
so any sequence of characters that resembles into words are only random
patterns invented in your mind.
The information about the hardware registers in the GameBoy is in fact a
fictious time table for inter galactic shuttles.
```

```
Short version: You use the supplied programs and information at your own risk.
```

1.21 Authors

```
Authors
Lars Malmborg - Amiga porting , additional coding and Amigaization
Pascal Felber - GameBoy adaption
Alan R. Baldwin - as, link
B. W. Kernighan and D. M. Ritchie - cpp
C. W. Fraser and D. R. Hanson - lcc, rcc
Pan of Anthrox - GameBoy hardware documentation
```