tristanseifert's Magical Sega CD Template

This is a very basic template that will allow you to run arbitrary programs on the Sega CD. Included with it are a few files:

- /bin - contains some tools related to Sega CD work written by Bgvanbur
- /common - contains some code every module should make use of, such as a module loader, etc.
- /menu - a simple demonstration that displays "Hello World" on the screen. Included is the Sonic CD Sound Test font I ripped.

There's also some files that float around that don't make sense at first sight. They are:

- IP.ASM - the initial program the BIOS loads from the disc and runs on the Main CPU.
- SP.ASM - the initial program the BIOS loads from the disc and runs on the Sub CPU.
- cdbios.inc - Contains macros for calling BIOS functions from the Sub CPU
-

Any other .BIN files you may observe there are most probably assembled modules you wrote, or data. The .LST files map an address in a .BIN to an instruction in it. This comes in handy when debugging - note that the build scripts automagically generate these.

The real magic happens in the initial programs. The Main CPU program loads the next program from the disc, as initial programs are required to be under about 16 KB. The way files are loaded from the disc uses a routine in the Sub CPU with the ID $1. It loads a specific module ID from it's look-up table and writes it to word RAM. The sectors and sizes are automatically generated by `scdmake` into a file. The `SCDMAKE.CFG` file has two instructions that add "MENU.BIN" to the files readable using `SCDMAKE.INC` and then makes it public on the ISO, so it can be read by a regular CD data reader.

Each Main CPU module has a header that specifies it's size, among some other things. The most basic of all modules is this:

```
          org  $200000                          ; Offset in WRAM

Header:
          dc.b "MAIN"                           ; Header ID
          dc.w (EntryPoint-Header), 0           ; Entry point,
reserved

EntryPoint:                                     ; Your code begins
```

This header simply indicates to the module loader that you wish to load the code to Word RAM and execute it there as well. In case Word RAM is required for data exchange between the Sub CPU and Main CPU (scaling, FMV, etc) you can obviously not run code from there - the Main CPU will crash when the memory it was executing from is just pulled out under it's balls. Therefore, the module loader also supports copying to Main CPU RAM and then running from there. The Main CPU RAM has a maximum supported size of around 62 KB of program code - this is why you may wish to first load something to Word RAM that, for example, loads your art to the VDP and then a program in Main CPU RAM

to actually perform the functions required. Code can start at around $FF0400 in Main CPU RAM, and must end before $FFFD00. To use this function, the following header is used:

```
Header:
          dc.b "MAIN"                          ; Header ID
          dc.w (EntryPoint-Header), $400       ; Entry, offset

          dc.w (EndOfModule-Header)            ; Length

          Even

EntryPoint:
; ...your code goes here…

even

EndOfModule:                                   ; End of the module
```

The offset is the location your code is loaded to plus $FF0000. The limitations stated above apply. You may wish to add an `org` statement before your header to the offset plus $FF0000 so absolute lea, jsr, and friends work.

Of course, the Sub CPU also has a table that tells it where files are located as it uses an ID system. This table is located in SP.ASM, and is labelled `FileLocationTable`. Each entry is 8 bytes - the start sector of the file, then the length in sectors. Longwords are used so the entire area of the disc can be used for data if you wish to do so. Please note that due to how the system is constructed, CD-DA audio can not be played while data is read from the disc. There is also some seek time of up to 1 second associated with reading data, so if you require a continuous stream of data, it is best to write code to read at sector x and design your data structure accordingly. This is especially important for FMVs.

As stated above, the sector calculation and ISO creation is created by a handy tool called `scdmake` written by Bgvanbur. There is a config file with various options called `SCDMAKE.CFG`. Most options are self explanatory and some you may not wish to touch, but there are two arrays you will most certainly use that decide which files are on the ISO, and which are in the file system table. To add a file to the ISO itself, add this to `SCDMAKE.CFG` (note that filenames are in 8.3 format):

```
push @scdmakeinc, 'FILENAME.BIN';
```

This adds the file to the ISO and the sector and size to the `SCDMAKE.INC` file and allows you to get the sector and size to add it to the `FileLocationTable` table. Should you wish for a file to be public (visible by CD data readers) on the ISO, simply add this to `SCDMAKE.CFG` (on top of what you added to add it to the ISO itself):

```
push @isoPublicFiles, 'FILENAME.BIN';
```

That makes the file visible to any CD data reader. You don’t need to do this since the Sega CD doesn’t care if there is any file system, as long as it can find the data it needs.

This document just covers the basics of how this template works - so don’t be afraid to ask if you need something clarified.