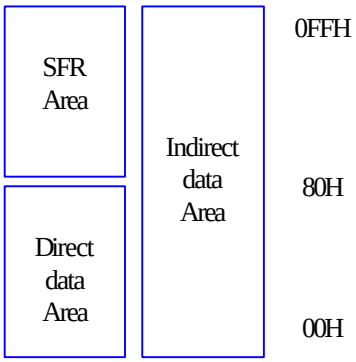
OVERVIEW

The 8051 processor uses a Harvard memory architecture. This means that each memory area is separate and distinct from the other areas. Code space is physically different from RAM space.

DATA AND IDATA MEMORY

There are two internal ram areas. One is accessed directly, and the other indirectly through R0 or R1. It is important to remember that the lower 128 bytes of these areas overlap. A picture looks like this:



Remember that if you use a 256 byte derivative it is possible to get an "Out of Data Space" error from the linker even when you have used only 80H bytes of **data**. You still have access to the rest of memory (80H - 0FFH), but you must specify **idata** for variables that can be located in this range.

CODE MEMORY

C51 uses the ROM directive (page 49 in C51 manual) to help optimize call and jump instructions. If you use one of Philips reduced instruction set processors such as the 80C751 or 80C752, you must use the ROM (SMALL) directive so that C51 generates only **ACALL** and **AJMP** instructions. With other processors we recommend that you use ROM(LARGE) which is the default. This generates **LCALL** and **LJMP** instructions for the full 64KB range.

You may optimize the code after your project is fully debugged and if you determine that further optimization is required. You may then use ROM (COMPACT) which generates **AJMP** instructions within a function and **LCALL** instructions for function calls. This saves 1 byte per jump. There is not a great savings to be gained from this step, but if your code is just a few bytes over what your hardware allows it may help. Remember that when using ROM (COMPACT) no function may be longer than 2KB.

MEMORY MODELS

When designing your 8051 project there are a few considerations that should be taken into account to optimize the use of the internal RAM area. Since the directly accessed internal **data** area is an extremely limited commodity it must be used wisely. However, there are few, if any good reasons to use any memory model other than SMALL. SMALL memory model is the default. It generates the tightest code with the fastest execution speed. To use this limited resource most effectively, there are a few guidelines you should follow.

VARIABLES

Global variables should be located in RAM areas other than the data area through the use of one of the memory type specifiers **idata**, **pdata**, or **xdata** as your application permits. The goal here is to give the linker the freedom to use the data area for the DATA_GROUP local variable segment. Its size and location can be found by looking in the M51 file generated by BL51.

If you are using data types that are **larger than integers** they should also be located in a memory area other than data. This includes longs, floats, arrays, structures, or unions larger than an integer.

INTERRUPT SERVICE ROUTINES

Interrupt routines require special consideration as well. All local variables declared within an interrupt are static in nature and are given unique data locations. For this reason, an ISR's local variables should also be located as if they were variables larger than an integer. Doing so slows down the ISR and is discussed under exceptions.

EXCEPTIONS

The exceptions to these guidelines are determined by the variable's access. If speed of access is critical, as in a fast responding interrupt or a small ring buffer used for high speed serial communications, then the default creates the fastest access. Even then, **idata** is usually adequate. If, after your project is debugged, you find that internal data space is available, there is a significant gain to be realized by moving floats and longs into the data area. Try these first, then the other variables. Begin optimizing variable locations only after your project is debugged and you are looking for specific optimizations.

CONCLUSION

It is important to watch how your directly accessed data space is used. Use it wisely and your applications can be nice and clean with tight, fast code. Use it unwisely and you will be forced to use the LARGE memory model which will make your code much less efficient. A few steps taken at the beginning of your design can make all the difference in the world.

In the USA:
Keil Software, Inc.
16990 Dallas Parkway, Suite 120
Dallas, TX 75248-1903
USA

Sales: 800-348-8051
Phone: 972-735-8052
FAX: 972-735-8055

E-mail: sales.us@keil.com
support.us@keil.com

Internet: http://www.keil.com/

In Europe:
Keil Elektronik GmbH
Bretonischer Ring 15
D-85630 Grasbrunn b. Munchen
Germany

Phone: (49) (089) 45 60 40 - 0
FAX: (49) (089) 46 81 62

E-mail: sales.intl@keil.com
support.intl@keil.com