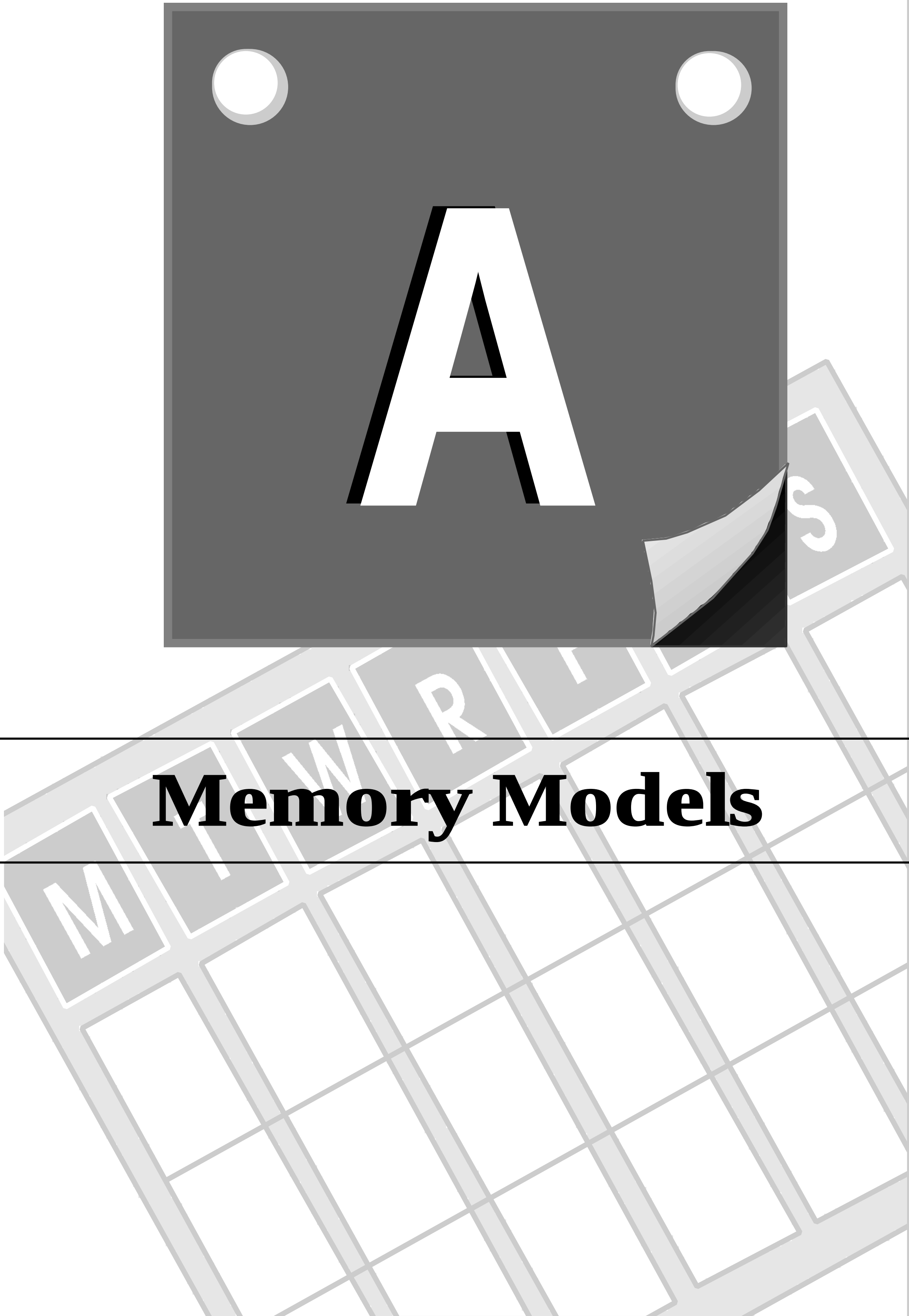

A

Memory Models

Every C programmer eventually is confronted with memory models. Beginning C programmers tend to avoid the subject. The compiler's default memory model is the one that is used. As programs grow in size, or as writing optimal programs becomes an issue, memory models may become a concern. If your compiler is defaulted to the large memory model, you may never question or worry about memory models. In fact, many programmers who develop at an advanced level don't know the difference between the memory models.

What Are the Memory Models?

There are six different memory models; not all of them are supported by every computer. The six models are:

☐ The tiny model
☐ The small model
☐ The compact model
☐ The medium model
☐ The large model
☐ The huge model

Virtually every computer supports four of these models. These are the small, compact, medium, and large models. The tiny and huge memory models are supported by a large number of compilers, but fewer than the other four.

The Difference Among Memory Models

The preceding memory models apply to the Intel 8086 and other *x*86 microprocessors. The Intel microprocessor uses a *segmented memory architecture*. The *x*86 microprocessors can only address 64K segments of memory at a time. In total, an 8086 can address only a single megabyte of memory. Higher level *x*86 processors have the capability to address more than a single megabyte; however, to be completely compatible with the 8086 microprocessor, only a megabyte of memory can be accessed at a time.

The microprocessor can keep track of more than one segment at a time. Generally, four different segments are tracked. These segments are the code segment, which tracks where the computer—or machine—instructions are, the data segment, which is where the data—or information—is stored, the stack, and an extra segment, which may be used for additional data.

There are several segment registers in the microprocessor. Four of these registers are used to track the segments. The CS register tracks the code segment, the DS register tracks the data segment, the SS register tracks the stack, and ES register tracks the extra segment. Each of these registers contains a 16-bit value. Each of these registers contains a value that is the address of the beginning of a 64K segment. A pointer can then be used to state the offset into this segment.

Note: If you wanted to know where a specific data item is, it would be determined by knowing the offset to the segment along with the offset into the segment. The actual address is determined by taking the segment address, shifting it four bits to the left, and then adding the pointer value containing the offset into the segment.

Note: In a 32-bit compiler, on a 32-bit machine, 32-bit register values may be used.

The segments that are used may or may not overlap. If all four segments completely overlapped with each other, you would only be using 64K at one time. If all four segments didn't overlap at all, then 64K multiplied by four, or 256K, of memory could be used at a time. In the second scenario, each 64K segment must be used for its individual purpose of code, data, stack, and extra.

The Memory Models

Now that the background materials have been presented, you're ready to see the differences among the six memory models. These differences revolve around how the segments are set up.

The Tiny Memory Model

With the tiny memory model, the CS, DS, SS, and ES segments all overlap. Looked at a different way, the CS, DS, SS, and ES registers all point to the same offset.

Additionally, the value in the registers can't be changed. This means only a single 64K segment is used. Figure A.1 illustrates the tiny memory model.

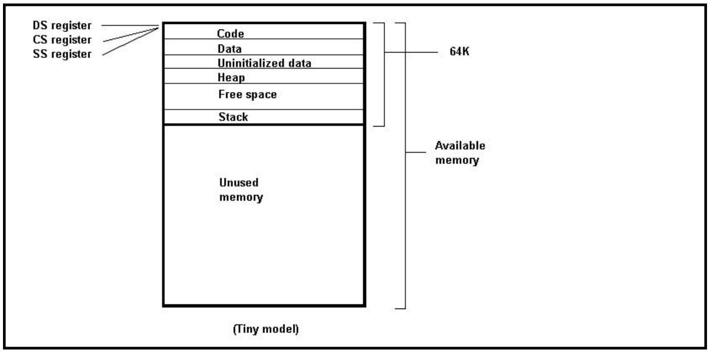

Figure A.1. *The tiny memory model.*

The Small Memory Model

The small memory model enables different values to be placed in the CS and DS registers. This means that the code segment is different than the data segment. In addition, the SS register for the stack contains the same address as the DS register. These values in the registers don't change so only two individual 64K segments are used. Because the addresses of the registers don't change, you only need to worry about the offsets into segments and not their locations. Figure A.2 illustrates the small memory model.

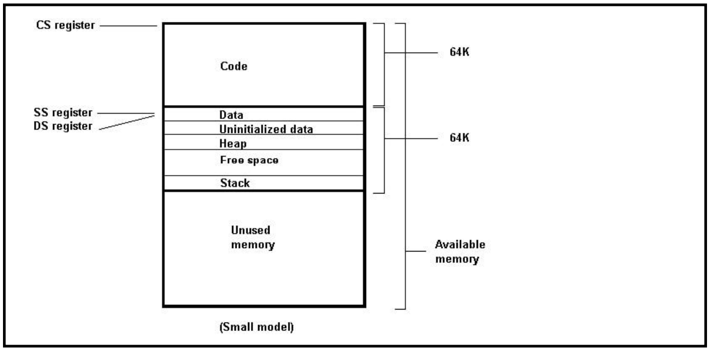

Figure A.2. *The small memory model.*

The Medium Memory Model

Like the small memory model, the medium model allows for two different 64K segments. The use of these segments is the same except the small memory model doesn't allow the address of any of the registers to be changed; the medium model does. In the medium model, the value in the CS register that points to the code segment can be changed, but the value in the DS and SS registers can't. In addition, the DS and SS registers point to the same segment.

By being able to change the CS segment, more than 64K can be used for code. The data and the stack are still limited to a total of 64K because their register values can't be changed, but the code is not—up to a megabyte of code can be used. Figure A.3 illustrates the medium memory model.

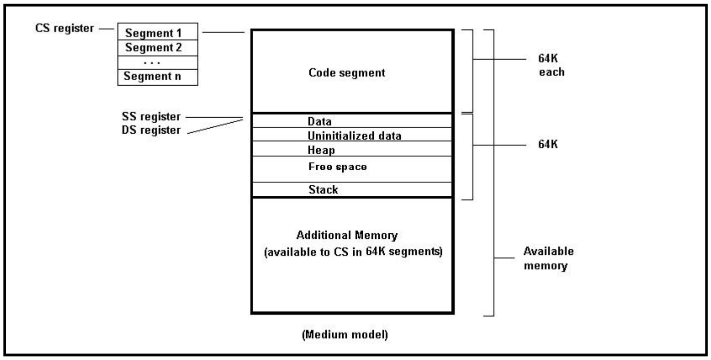

Figure A.3. *The medium memory model.*

The Compact Memory Model

The compact memory model is basically the opposite of the medium model. It enables the value in the DS register to be modified. This allows for a large amount of data within a program. The CS register's value can't be modified, thus limiting the code size.

Whereas the medium model allows for a large amount of code with a limited amount of data, the medium model allows for a maximum of 64K in code and a large amount of data. Figure A.4 illustrates the compact model.

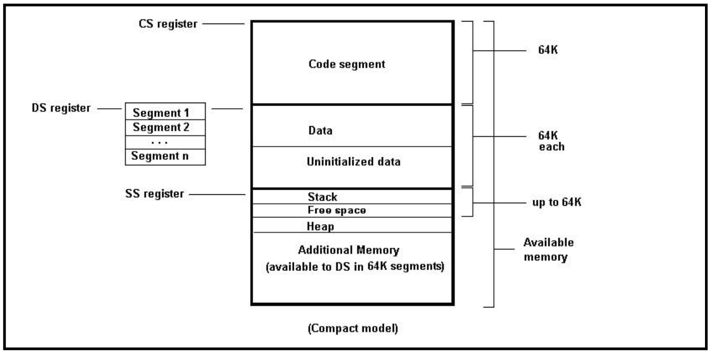

Figure A.4. *The compact memory model.*

The Large Memory Model

Many programmers work with the large memory model. With it, the values in both the DS and CS registers can be modified. This gives you the flexibility of working with up to a megabyte with either the code or the data. Figure A.5 illustrates the large memory model.

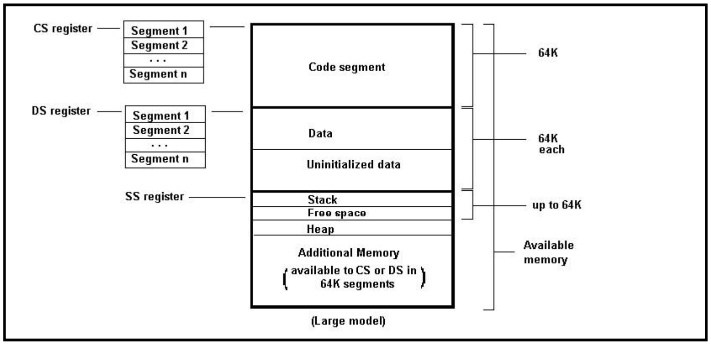

Figure A.5. *The large memory model.*

The Huge Memory Model

The final memory model is the huge model. It may not be supported by every compiler. The huge memory model differs from the large in that the 64K segment size limit on static data isn't applied. This data can occupy the rest of the free memory. Figure A.6 illustrates the huge memory model.

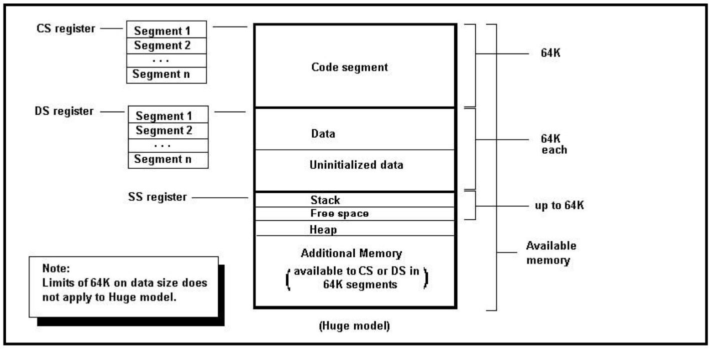

Figure A.6. *The huge memory model.*

Summarizing the Memory Models

Each of the memory models has its place. Depending on the structure of the program you are developing, one memory model may be better than another. The tiny memory model is best when you want to write an extremely small program or when memory is at a premium. The small model is good for most small applications. Most of the applications in this book will compile easily under the small memory model. The medium model is best for larger programs that don't have a lot of data that is used in memory.

Note: We are talking about data in memory, not data stored elsewhere. You can store a great deal of data on a hard drive. If you are only using a few small records at a time, you don't need to worry about a large data segment. It's only when you are loading the data into memory that it's applied to the 64K segments in the data segment.

The compact memory model is most appropriately used with small programs that use a great deal of data in memory. The large program is used for large programs that use a lot of data in memory. The huge memory model is for extremely large programs.

Expert Tip: I use the small memory model for a majority of my programming. The small memory model offers enough room to write utility programs and many other smaller programs. When the small model is no longer big enough to hold the variables or code, the compiler will provide errors. At the time I get these errors, I switch to the large model. If I know the program I am writing will be large, I use the large model.

64K Limits

Although some of the models allow data and code to be larger than 64K, there is a restriction that the code must be broken into segments that are smaller than 64K. If your code is more than 64K, you'll need to break it down into multiple source files. Each source file will need to be compiled separately. Once compiled, they can be linked together.