

BASIC ASSEMBLY LANGUAGE PROGRAM STRUCTURE

By Mark E. Donaldson

SEGMENT DEFINITION & SEGMENT ORDER

1. An assembly language program must contain at least three segments:
 - **Stack Segment** containing the program's stack.
 - **Data Segment** for variables used by the program.
 - **Code Segment** containing the program's machine instructions.
2. These segments are defined using two directives:
 - **SEGMENT** directive which marks the start of the segment.
 - **ENDS** directive which marks the end of the segment.
3. Each segment must have a name, and the name must be used twice in the definition:
 - Once before the **SEGMENT** directive.
 - Again before the **ENDS** directive.
4. The stack segment must be defined with the **STACK** directive following the **SEGMENT** directive. This definition tells DOS which segment address to load into the **SS** register.
5. The size of the stack segment is dictated by the definition of some data within the segment. Enough stack space must be defined to cover any possible need the program may have, plus the needs of DOS and any interrupt service routines (including loaded TSR's) that may be active while the program is running. If enough space is not defined, there may be a stack crash, which will terminate the program and possibly crash the machine. **Decide how much stack space is realistically needed, and allocate twice as much.** Use the DB and DUP directives to allocate space.
6. The segments are order independent. They may be placed in any order without changing the way the segments work together, nor the way the assembler treats them. However, it is recommended the convention of defining the stack segment first, the data segment second, and the code segment third be used.

DATA DEFINITIONS FOR VARIABLES & STACK SPACE

1. Variables and space for the stack must be allocated during assembly. The **DB**, **DW**, **DD**, and **DUP** directives are the most common means to do this.
2. **DW** allocates word sized (16 bit) variables typically, to contain register sized values. **DD** allocates double word sized (32 bit) variables, typically for full addresses containing both segment and offset.

These definitions have a form as such:

Label DW OFFFH

3. The **DB** directive is designed to allocate byte sized (8 bit) quantities like characters and register halves. It has the special property of being able to allocate strings as well. Elements of the string may be numbers, characters, or quoted strings, separated by commas. The following are legal **DB** variable definitions:

BASIC ASSEMBLY LANGUAGE PROGRAM STRUCTURE

By Mark E. Donaldson

```
Label DB 042H
Label DB 17
Label DB Insert some text
Label DB ODH, OAH
```

4. The **DUP** directive with a question mark (?) may be used to allocate a variable or buffer without specifying any initial values. The ? Sets aside memory but stores nothing in it. The **DUP** directive can also be used to store repeated patterns into larger variables and buffers. This can make buffers and variables easy to spot when a hex dump from DEBUG is performed. The **DUP** directive will take the following forms:

```
Label DB 10 DUP (02H,04H,06H,08H)
Label DB 64 DUP (STACK)
```

SETTING UP CODE SEGMENT

1. Like any segment, the code segment must have a name, and the name must be given at the start and end of the segment definition, before the **SEGMENT** and **ENDS** directives. Although the name is not important and probably won't be reference anywhere in the code, it must be there or an assembler error will be received.
2. An **ASSUME** directive must be included in the program. Its purpose is to tell the assembler which of the defined segments is to be used for the code segment, and which segment is to be used for the data segment. Unlike the stack segment, which has the directive **STACK** to tell the assembler what sort of segment it is, nothing in the code or data segments specifies which sort of segment they are. It isn't enough that there are variables defined in the data segment or machine instructions in the code segment.
3. An assembly language program should have its machine instructions grouped together in a named procedure with the **PROC** directive. This is not strictly necessary unless the program is broken down into procedures or modules. However, *it is advised that the main program portion of any assembly language program into a procedure called **Main*** to help make the program more readable.
4. What is essential is to provide a label that marks the place where program execution is to begin. It is recommended that the label **Start:** be used as a convention, but the label can be any legal identifier. Whatever the label, mark the main program's starting point with the label and a colon. Place the same label minus the colon after the **END** directive, which marks the end of the source code file. Placing the start label after the **END** directive tells the assembler that there is no more source code, and that the label is the point at which execution is to begin.

SEGMENT REGISTER ASSUMPTIONS

Where allowed, segment assumptions can be overridden with the segment override prefixes. These are DS: SS: CS: ES:. For example:

```
mov ES:[BX],AX
```

The assumptions are these:

1. When the offset is specified in BX, SI, or DI, the assumed segment register is DS.
2. When the offset is specified in SP, the assumed segment register is SS. **CANNOT BE OVERRIDDEN.**
3. When the offset is specified in BP, the assumed segment register is SS.
4. For string instruction LODS, the assumed segment is DS and the assumed offset is DI. **CANNOT BE OVERRIDDEN.**

BASIC ASSEMBLY LANGUAGE PROGRAM STRUCTURE

By Mark E. Donaldson

5. For string instruction STOS and SCAS, the assumed segment is ES and the assumed offset is DI. **CANNOT BE OVERRIDDEN.**
6. For string instruction MOVS, the source must be pointed to be DS:SI and the destination must be pointed to by ES:DI. **CANNOT BE OVERRIDDEN.**

Rules For External Procedures Modules and Libraries.

RULE #1: Declare the code segments Public in all modules, and give them all the same name.

RULE #2: Declare the data segments Public in all modules, and give them all the same name.

RULE #3: Declare all exported procedures, entry points, and variables as Public. Place the PUBLIC directive inside the segment where the exported items are declared.

RULE #4: Declare all imported procedures, entry points, and variables as external. Put the external directive inside the segment where the imported items are to be used. Data is used in the data segment, code in the code segment.

RULE #5: Make sure that there is a common ASSUME statement in the code segment of every module associating the CS register with the shared code segment and the DS register with the shared data segment.

RULE #6: Don't forget to add the names of all external modules to the linker command line in the link setup.

DOS DEBUG COMMON COMMANDS

- D Hex Dump
- E Enter New Data (Change Bytes In Memory)
- Q Quit
- R Register (Register Dump or -R Specific Register)
- A Assemble (Assemble Directly To Memory Incrementing CS:IP)
- T Trace (Execute Machine Instructions At CS:IP)
- W Write (Save Altered Memory Image Back To Disk)
- G GO (Begins Execution At The Address CS:IP)

DOS DEBUG FLAG STATE SYMBOLS

<u>FLAG</u>	<u>Set Symbol</u>	<u>Clear Symbol</u>
OF - Overflow Flag	OV	NV
DF - Direction flag	DN	UP
IE - Interrupt enable flag	EI	DI
SF - Sign Flag	NG	PL
ZF - Zero Flag	ZR	NZ
AF - Auxiliary carry flag	AC	NA
PF - Parity flag	PE	PO
CF - Carry flag	CY	NC

BASIC ASSEMBLY LANGUAGE PROGRAM STRUCTURE

By Mark E. Donaldson

Stack Segment

```
<name>SEGMENT STACK  
    <data definition>  
<name>    ENDS
```

The size of this data definition becomes the size of the stack.

Data Segment

```
<name> SEGMENT  
<variables>  
<name> ENDS
```

Data Segment

```
<name> SEGMENT  
  
<name> PROC  
    ASSUME CS:<name>  
           DS:<name>  
  
<start label>:  
    <instructions>  
  
<name> ENDP  
<name> ENDS
```

The segments are independent and may appear in any order. This order is a suggestion.

END <start label>

BASIC ASSEMBLY LANGUAGE PROGRAM STRUCTURE

By Mark E. Donaldson

