**Multi Pass Assembler**

* Multipass assembler means more than one pass is used by assembler.
* Multipass assembler is used to eliminate forward references in sybol definition.
* it creates a number of passes that is necessary to process the definition of symbols
* â€¢Multi pass assembler:

-Does the work in two pass

-Resolves the forward references

**First pass:**

-Scans the code

-Validates the tokens

-Creates a symbol table

**Second Pass**:

-Solves forward references

-Converts the code to the machine code

If we use a two-pass assembler, the following symbol definition cannot be allowed.

ALPHA EQU BETA

BETA EQU DELTA

DELTA RESW 1

This is because ALPHA and BETA cannot be defined in pass 1. Actually, if we allow multi-pass processing, DELTA is defined in pass 1, BETA is defined in pass 2, and ALPHA is defined in pass 3, and the above definitions can be allowed.

This is the motivation for using a multi-pass assembler.

* It is unnecessary for a multi-pass assembler to make more than two passes over the entire program.
* Instead, only the parts of the program involving forward references need to be processed in multiple passes.
* The method presented here can be used to process any kind of forward references.
* Use a symbol table to store symbols that are not totally defined yet.
* For a undefined symbol, in its entry, – We store the names and the number of undefined symbols which contribute to the calculation of its value. – We also keep a list of symbols whose values depend on the defined value of this symbol.
* When a symbol becomes defined, we use its value to reevaluate the values of all of the symbols that are kept in this list.
* The above step is performed recursively.
* **Examples**

Microsoft MASM Assembler, Sun Sparc Assembler, IBM AIX Assembler

* **Microsoft MASM Assembler**
* SEGMENT - a collection segments, each segment is defined as belonging to a particular class, CODE, DATA, CONST, STACK
* registers: CS (code), SS (stack), DS (data), ES, FS, GS
* similar to program blocks in SIC l ASSUME

e. g. MOVE ES: DATASEG 2 AX, DATASEG 2 ES, AX » similar to BASE in SIC

* JUMP with forward reference
* near jump: 2 or 3 bytes
* far jump: 5 bytes
* e. g. JMP TARGET
* Warning: JMP FAR PTR TARGET
* Warning: JMP SHORT TARGET
* Pass 1: reserves 3 bytes for jump instruction phase error PUBLIC, EXTRN
* similar to EXTDEF, EXTREF in SIC 12

Symbol definition must be completed in pass 1.

* Prohibiting forward references in symbol definition is not a serious inconvenience.
* Forward references tend to create difficulty for a person reading the program.

**Advanced Assembly Process**

|  |  |
| --- | --- |
| 1. Assembling the source code into an object file 2. Linking the object file with other modules or libraries into an executable program 3. Loading the program into memory 4. Running the program   assembler translates instruction into object code | |
| * what the assembler does |  |

**3. Small Assembly Sample**

**; add\_16\_bytes.asm**

**;**

.586P

**; Flat memory model, standard calling convention:**

.MODEL FLAT, STDCALL

**;**

**; Data segment**

\_DATA SEGMENT

values db 16 DUP( 5 ) **; 16 bytes of values "5"**

\_DATA ENDS

**; Code segment**

\_TEXT SEGMENT

START:

mov eax, 0 **; clear result**

mov bl, 16 **; init loop counter**

lea esi, values **; init data pointer**

addup:

add al, [esi] **; add byte to sum**

inc esi **; increment data pointer**

dec bl **; decrement loop counter**

jnz addup **; if BL not zero, continue**

mov [esi], al **; save sum**

ret **; Exit**

\_TEXT ENDS

END START

* Listing file: add\_16\_bytes.lst

**4. MASM Command Line Interface**

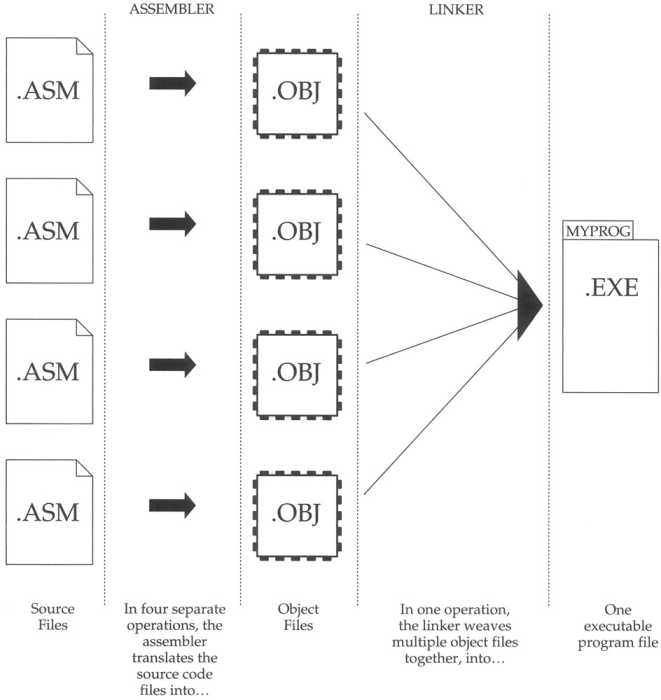
* To assemble and run an assembler program named **myprog.asm**, type the following commands:
* call "C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\vsvars32.bat"
* ML /coff /c /Fl myprog.asm
* LINK /debug /subsystem:console /entry:start /out:myprog.exe myprog.obj ..\iolib\io.obj kernel32.lib
* myprog.exe

**5. Assembling**

|  |  |
| --- | --- |
| * At assembly time, the assembler:   + Evaluates conditional-assembly directives, assembling if the conditions are true.   + Expands macros and macro functions.   + Evaluates constant expressions such as **MYFLAG AND 80H**, substituting the calculated value for the expression.   + Encodes instructions and nonaddress operands. For example, **mov cx, 13;** can be encoded at assembly time because the instruction does not access memory.   + Saves memory offsets as offsets from their segments.   + Places segments and segment attributes in the object file.   + Saves placeholders for offsets and segments (relocatable addresses).   + Outputs a listing if requested.   + Passes messages (such as INCLUDELIB) directly to the linker. | * translating a single assembly module |

**6. Linking**

|  |  |
| --- | --- |
| * Once your source code is assembled, the resulting object file is passed to the linker. At this point, the linker may combine several object files into an executable program. The linker:   + Combines segments according to the instructions in the object files, rearranging the positions of segments that share the same class or group.   + Fills in placeholders for offsets (relocatable addresses).   + Writes relocations for segments into the header of .EXE files (but not .COM files).   + Writes the result as an executable program file.   Linking multiple object files together |  |

****

**7. Loading**

* After loading the executable file into memory, the operating system:
  + Creates the program segment prefix (PSP) header in memory.
  + Allocates memory for the program, based on the values in the PSP.
  + Loads the program.
  + Calculates the correct values for absolute addresses from the relocation table.
  + Loads the segment registers SS, CS, DS, and ES with values that point to the proper areas of memory.

**8. Useful Tools and Utilities**

* DUMPBIN disassembly program
* Debuggers: OllyDbg and WinDbg
* Consol I/O: iolib.