SYSTEM SOFTWARE ASSIGNMENT 1

TOPICS :- 1. CONTROL SECTIONS & ADVANTAGE OF USING THEM
2. MASM ASSEMBLER

REFERENCES :- Leland L. Beck, System Software : An Introduction to Systems Programming, 3/E

SUBMISSION DATE :- 12-10-2019

Done By,
Hasiksrishnan V
CS5A
Roll No: 27

CONTROL SECTIONS

A control section is a part of the program that maintains its identity after assembly; each control section can be loaded & relocated independently of the others. Different control sections are most often used for subroutines or other logical subdivisions of a program. The programmer can assemble, load & manipulate each of these control sections separately. The resulting flexibility is a major benefit of using control sections.

When control sections form logically related parts of a program, it is necessary to provide some means for linking them together. For example, instructions in 1 control section might need to refer to instructions or data located in another section. Because control sections are independently loaded & relocated, the assembler is unable to process these references in the usual way. The assembler has no idea where any other control section will be located at execution time. Such references between control sections are called external references. The assembler generates information for each external reference that will allow the loader to perform the required linking.

The assembler establishes a separate location counter (beginning at 0) for each control section, just as it does for program blocks. Control sections differ from program blocks in that they are handled separately by the assembler. Symbols that are defined in 1 control section may not be used directly by another control section; they must be identified as external references for the loader to handle. 2 assembler directives used to identify such references

are EXTDEF (external definition) & EXTREF (external reference). The EXTDEF statement in a control section names symbols, called external symbols, that are defined in this control section & may be used by other sections. Control section names don't need to be named in an EXTDEF statement because they are automatically considered to be external symbols. The EXTREF statement names symbols that are used in this control section & are defined elsewhere. The order in which symbols are listed in the EXTDEF & EXTREF statements is not significant.

The assembler must also include information in the object program that will cause the loader to insert the proper values where they are required. We need 2 new record types in the object program & a change in a previously defined record type. The 2 new record types are Define & Refer. A Define record gives information about external symbols that are defined in this control section - that is, symbols named by EXTDEF. A Refer record lists symbols that are used as external references by the control section - that is, symbols named by EXTREF. The formats of these records are as follows.

Define record:

Col. 1	D
Col. 2-7	Name of external symbol defined in this control section
Col. 8-13	Relative address of symbol within this control section (hexadecimal)
Col. 14-73	Repeat information in Col. 2-13 for other external symbols.

Refer record:

Col. 1	R
Col. 2-7	Name of external symbol referred to in this control section.
Col. 8-73	Names of other external reference symbols

The other information needed for program linking is added to the Modification record type. The new format is as follows.

Modification record (revised):

Col. 1	M
Col. 2-7	Starting address of the field to be modified, relative to the beginning of the control section (hexadecimal)
Col. 8-9	Length of the field to be modified, in half-bytes (hexadecimal)
Col. 10	Modification flag (+ or -)
Col. 11-16	External symbol whose value is to be added to or subtracted from the indicated field.

The 2 new items specify the modification to be performed: adding or subtracting the value of some external symbol. The symbol used for modification may be defined either in this control section or in another one.

There is a separate set of object program records for each control section. The records for each control section are exactly the same as they would be if the sections were assembled separately.

The Define & Refer records for each control section include the symbols named in the EXTDEF & EXTREF statements. In the case of Define, the record also indicates the relative address of each external symbol within the control section. For EXTREF symbols, no address information is available. These symbols are simply named in the Refer record.

The revised Modification record may still be used to perform program relocation. In the case of relocation, the modification required is adding the beginning address of the control section to certain fields in the object program. The symbol used as the name of the control section name has as its value the required address. Since the control section name is automatically an external symbol, it is available for use in Modification records. In this way, exactly the same mechanism can be used for program relocation & for program linking..

When an expression involves external references, the assembler cannot in general determine whether or not the expression is legal. The pairing of relative terms to test legality cannot be done without knowing which of the terms occur in the same control sections, & this is unknown at assembly time. In such a case, the assembler evaluates all of the terms it can, & combines these to form an initial expression value. It also generates Modification records so the loader can finish the evaluation. The loader can then check the expression for errors.

MASM Assembler

An MASM assembler language program is written as a collection of segments. Each segment is defined as belonging to a particular class, corresponding to its contents. Commonly used classes are CODE, DATA, CONST & STACK.

During program execution, segments are addressed via the x86 segment registers. In most cases, code segments are addressed using register CS & stack segments are addressed using register SS. These segment registers are automatically set by the system loader when a program is loaded for execution. Register CS is set to indicate the segment that contains the starting label specified in the END statement of the program. Register SS is set to indicate the last stack segment processed by the loader.

Data segments are normally addressed using DS, ES, FS or GS. The segment register to be used can be specified explicitly by the programmer. If the programmer does not specify a segment register, 1 is selected by the assembler.

By default, the assembler assumes that all references to data segments use register DS. This assumption can be changed by the assembler directive ASSUME. For example, the directive

```
ASSUME  ES: DATASEG2
```

tells the assembler to assume that register ES indicates the segment DATASEG2. Thus, any references to labels that are defined in DATASEG2 will be assembled using register ES. It is also possible to collect several segments into a group & use ASSUME to associate a segment register with the group.

Registers DS, ES, FS & GS must be loaded by the program before they can be used to address data segments.

Jump instructions are assembled in 2 different ways, depending on whether the target of the jump is in the same code segment as the jump instruction. A near jump is a jump to a target in the same code segment; a far jump is a jump to a target in a different code segment. A near jump is assembled using the current code segment register CS. A far jump must be assembled using a different segment register, which is specified in an instruction prefix. The assembled machine instruction for a near jump occupies 2 or 3 bytes. The assembled instruction for a far jump requires 5 bytes.

Forward references to labels in the source program can cause problems. For example, consider a jump instruction like

```
JMP   TARGET
```

If the definition of the label TARGET occurs in the program before the JMP instruction, the assembler can tell whether this is a near jump or a far jump. However, if this is a forward reference to TARGET, the assembler does not know how many bytes to reserve for the instruction.

By default, MASM assumes that a forward jump is a near jump. If the target of the jump is in another code segment, the programmer must warn the assembler by writing

```
JMP   FAR PTR TARGET
```

If the jump address is within 128 bytes of the current instruction, the programmer can specify the shorter (2-byte) near jump by writing

```
JMP   SHORT TARGET
```

If the JMP to TARGET is a fast jump, & the programmer does not specify FAR PTR, a problem occurs. During Pass 1, the assembler reserves 3 bytes for the jump instruction. However, the actual assembled instruction requires 5 bytes. In the earlier versions of MASM, this caused an assembly error. In later versions of MASM, the assembler can repeat Pass 1 to generate the correct location counter values.

Segments in an MASM source program can be written in more than 1 part. If a SEGMENT directive specifies the same name as a previously defined segment, it is considered to be a continuation of that segment. All of the parts of a segment are gathered together by the assembly process. Thus, segments can perform a similar function to the program blocks in SIC/XE.

References between segments that are assembled together are automatically handled by the assembler. External references between separately assembled modules must be handled by the linker. The MASM directive PUBLIC has approximately the same function as the SIC/XE directive EXTDEF. The MASM directive EXTRN has approximately the same function as EXTREF.

The object program for MASM may be in several different formats, to allow easy & efficient execution of the program in a variety of operating environments. MASM can also produce an instruction timing listing that shows the number of clock cycles required to execute each machine instruction. This allows the programmer to exercise a great deal of control in optimizing timing-critical sections of code.