CS1101: Lecture 38

Macros and Pass One of an Assembler

Dr. Barry O'Sullivan
b.osullivan@cs.ucc.ie

Course Homepage
http://www.cs.ucc.ie/~osullb/cs1101

Lecture Outline

- Macros
  - Macro Definition
  - Macro Call and Expansion
  - Macros versus Procedures
  - Macros with Parameters
- Two-Pass Assemblers
  - Forward Reference Problem
  - Pass One
  - Data used during Pass One
  - The Symbol Table
  - The Opcode Table
- **Reading**: Tanenbaum, Chapter 7, Section 2/3.

Macros

- A macro definition is a way to give a name to a piece of text.
- After a macro has been defined, the programmer can write the macro name instead of the piece of program.
- A macro is, in effect, an abbreviation for a piece of text.
- See Figure 7-4.

Macro Definition

- Although different assemblers have slightly different notations for defining macros, all require the same basic parts in a macro definition:
  1. A macro header giving the name of the macro being defined.
  2. The text comprising the body of the macro.
  3. A pseudoinstruction marking the end of the definition (e.g., ENDM).

Macro Call and Expansion

- When the assembler encounters a macro definition, it saves it definition table for subsequent use.
- From that point on, whenever the macro appears as an opcode, the assembler replaces it by the macro body.
- The use of a macro name as an opcode is known as a **macro call** and its replacement by the macro body is called **macro expansion**.

Macro Call and Expansion

- Macro expansion occurs during the assembly process and not during execution of the program.
- Both programs we have seen will produce precisely the same machine language code.
- Looking only at the machine language program, it is impossible to tell whether or not any macros were involved in its generation.
- The reason is that once macro expansion has been completed the macro definitions are discarded by the assembler.

Macros versus Procedures

- Macro calls should not be confused with procedure calls.
- The basic difference is that a macro call is an instruction to the assembler to replace the macro name with the macro body.
- A procedure call is a machine instruction that is inserted into the **object program** and that will later be executed to call the procedure.
- See Figure 7-5

Macros with Parameters

- Frequently, however, a program contains several sequences of instructions that are almost but not quite identical.
- Macro assemblers handle the case of nearly identical sequences by allowing macro definitions to provide **formal parameters** and by allowing macro calls to supply **actual parameters**.
- When a macro is expanded, each formal parameter appearing in the macro body is replaced by the corresponding actual parameter.
- The actual parameters are placed in the operand field of the macro call.
- See Figure 7-6

Two-Pass Assemblers

- An assembly language program consists of a series of one-line statements.
- Reading the program one line at a time and generating machine code for each line does not work!
- Why not?
- **Forward Reference Problem**

Forward Reference Problem

- Consider the situation where the first statement is a branchto L;
- The assembler needs to know the address of statement L before it can assemble it.
- Statement L could be anywhere in the programme.
- This is a **forward reference problem** since L can be used before it has been defined – a reference has been made to a symbol whose definition will only occur later.

Resolving Forward References

- We can resolve the problem in two ways:
- **Approach 1:** The assembler may read the source program twice – **two passes**. On **pass one**, the definitions of symbols, including statement labels, are collected and stored in a table. When **pass two** starts, the definitions of symbols are known.
- **Approach 2:** On the first reading of the assembly program convert it to an intermediate form stored in memory. The second pass is made over this intermediate form. This saves on I/O time.
- Another task of pass one is to save all macro definitions and expand the calls as they are encountered.
- Therefore, pass one performs two tasks: defining the symbols and expanding macros.

Pass One

- The principal function of pass one is to build up a table called the **symbol table**, containing the values of all symbols.
- A symbol is either a label or a value that is assigned a symbolic name, for example:

```
BUFSIZE EQU 8192
```

- During pass one the assembler “remembers” the address of each instruction as it is read.
- This is done using a variable called the **ILC (Instruction Location Counter)**.
- This variable is set to 0 at the beginning of pass one and incremented by the instruction length for each instruction processed.

Data used during Pass One

- Pass one of most assemblers uses at least three tables:
  - the symbol table
  - the pseudoinstruction table
  - the opcode table
- Some details of these follow.

The Symbol Table

- The symbol table has one entry for each symbol.
- Symbols are defined by using them as labels or through the EQU pseudo-instruction.
- Each symbol table entry contains the symbol itself, its numerical value, and sometimes additional information such as:
  - The length of the data field associated with the symbol.
  - The relocation bits (does the symbol change value if the program is loaded at a different address than the assember assumed?)
  - Whether or not the symbol is to be accessible outside the procedure.

The Opcode Table

- The opcode table contains at least one entry for each symbolic opcode in the assembly language.
- Each entry contains:
  - the symbolic opcode
  - two operands
  - the opcode's numerical value
  - the instruction length
  - a type number that separates the opcodes into groups depending on the number and kind of operands.