Lecture 1

Introduction - 1/4

- A **macro** is a unit of specification for program generation through expansion

- It consists of a **name**, a set **of formal parameters** and a **body of code**

- **Macro expansion** - Use of macro name with a set of **actual parameters** is replaced by some code generated from its body

Introduction - 2/4

- Two kinds of expansion
  - Lexical expansion
    - Replacement of character string by another character string during program generation
    - Employed to replace occurrences of formal parameters by corresponding actual parameters

Introduction - 3/4

- Semantic expansion
  - Generation of instructions tailored to requirements of a specific usage
  - Different uses of macro can lead to codes which differ in number, sequence and opcodes of instructions

Introduction - 4/4

- Use of macro name in mnemonic field of assembly statement leads to its expansion

- Use of a subroutine name in a call instruction leads to its execution

- Macros can be said to trade program size for execution efficiency of a program

Macro Definition – 1/3

- A macro definition is enclosed between a **macro header** statement and a **macro end** statement
- Located at start of program
- Consists of
  - Macro prototype statement
  - One or more model statements
  - Macro preprocessor statements

Macro Definition – 2/3

- **Macro prototype** statement declares the name of a macro and the names and *kinds* of its parameters

- **Model statement** is a statement from which assembly language statement may be generated during macro expansion

- A **preprocessor statement** is used to perform auxiliary functions during macro expansion

Macro Definition – 3/3

- Macro prototype statement syntax

  <macro name> [<formal parameter spec.>[,..]]

<macro name> appears in mnemonic field

<formal parameter spec.> is of the form

  & <parameter name> [<parameter kind>]

Macro call

- A macro is called by writing the **macro name in the mnemonic field** of assembly statement

- Syntax

  <macro name> [<actual parameter spec.>[,..]]

Macro expansion – 1/2

- **Macro call leads to macro expansion**
- During macro expansion, macro call statement is replaced by a sequence of assembly statements
- To differentiate between original statements and statements resulting from macro expansion, each expanded statement is marked with a ‘+’ preceding its label field

Macro expansion – 2/2

- Two key notions
  - **Expansion time control flow** :- determines the order in which model statements are visited during macro expansion
  - **Lexical substitution** :- generate assembly statement from model statement

Macro expansion algorithm

1. MEC :=stmt no of first stmt following prototype stmt
2. While stmt pointed by MEC is not MEND stmt
   a) If model stmt then
      i. Expand stmt
      ii. MEC := MEC + 1
   b) Else (i.e. preprocessor stmt)
      i. MEC := new value specified in stmt
3. Exit from macro expansion

Expansion time control flow – 1/2

- Default flow is **sequential**

- In absence of preprocessor statements, model statements of a macro are visited sequentially starting with statement following the macro prototype statement and ending with statement preceding with **MEND** statement

Expansion time control flow – 2/2

- Preprocessor statement can alter the flow of control during expansion s.t. some model statements are either
  - Never visited during expansion (**conditional expansion**) or
  - Repeatedly visited during expansion (**expansion time loops**)

- Flow of control during expansion is implemented using a **<u>macro expansion counter (MEC)</u>**

Lexical substitution – 1/2

- Model statement consists of **three types** of strings
  1. Ordinary string, which stands for itself
  2. Name of formal parameter which is preceded by character &
  3. Name of preprocessor variable which is also preceded by character &

Lexical substitution – 2/2

- During lexical expansion, strings of type 1 are retained w/o substitution
- Strings of type 2 and 3 are replaced by values of formal parameters or preprocessor variables

Types of parameters

- Positional
- Keyword
- Default
- Mixed

Positional parameters – 1/2

- Written as **&<parameter name>**

  Example:

  &SAMPLE

SAMPLE is name of parameter

<parameter kind> is omitted

<actual parameter spec> in a call on a macro using positional parameters is simply <ordinary string>

Positional parameters – 2/2

- Value of positional formal parameter XYZ is determined by rule of positional association
  - Find the ordinal position of XYZ in the list in formal parameters in macro prototype stmt
  - Find actual parameter specification occupying the same ordinal position in the list of actual parameters in macro call stmt. Let this list be ordinary string ABC. Then, value of formal parameter XYZ is ABC

Keyword parameters – 1/2

- <parameter name> is ordinary string
- <parameter kind> is the string **"=“**
- <actual parameter spec> is written as

  <formal parameter name>=<ordinary string>

- Rule of keyword association
  - Find the actual parameter specification which has the form XYZ=<ordinary string>
  - Let <ordinary string> be ABC. Then the value of formal parameter XYZ is ABC

Keyword parameters – 2/2

- The ordinal position of the specification XYZ=ABC in list of actual parameters is immaterial
- Useful in situations where long lists of parameters have to be used

Default specification of parameters

- A default is a standard assumption in the **absence of an explicit specification** by the programmer
- Useful in situations where parameter has the same value in most calls
- Specification **overrides** the default value of the parameter for the duration of the call
- Default specification of keyword parameters can be incorporated by extending the syntax

&<parameter name>[<parameter kind>[<default value>]]

Macros with mixed parameter lists

- Macro can use both positional and keyword parameters
- **Positional parameters must precede all keyword parameters**

```
SUMUP              A, B, G=20, H=X
```

- A and B are positional parameters
- G and H are keyword parameters

Other uses of parameters

- Use of parameters is not restricted only to operand fields
- Formal parameters can also appear in the label and opcode fields of model statements