HSBPVT’s PARIKRAMA GOI COE, KASHTI

SYSTEM PROGRAMMING & OPERATING SYSTEM [310251]
T.E. COMPUTER
(MACRO PROCESSOR)
Prof. SURYAVANSHI ARTI PRASHANT
E-mail:- artips15@gmail.com
Prof. SURYAVANSHI PRASHANT MAHARUDRA
E-mail:- sprashant1234@gmail.com

HSBPVT’S, PARIKRAMA GOI COE KASHTI.
CONTENT

- Introduction
- Basic macro processor functions
- Macro expansion
- Macro invocation
- Pass-1 macro processor Flowchart
- Pass-2 macro processor Flowchart
- Data structures for one-pass macro processor
- Algorithm of Macro Processor
- Comparison of Macro Processors Design
INTRODUCTION

- A macro instruction (macro) is a notational convenience for the programmer.
- It allows the programmer to write shorthand version of a program (module programming).
- The macro processor replaces each macro instruction with the corresponding group of source language statements (expanding).
- Normally, it performs no analysis of the text it handles.
- It does not concern the meaning of the involved statements during macro expansion.
- The design of a macro processor generally is *machine independent*!
BASIC MACRO PROCESSOR FUNCTIONS

- Two new assembler directives are used in macro definition
  - MACRO: identify the beginning of a macro definition
  - MEND: identify the end of a macro definition
  - Prototype for the macro Each parameter begins with ‘&’
    name MACRO parameters
      :
      body
      :
      MEND

Body: the statements that will be generated as the expansion of the macro.
MACRO EXPANSION

- Each macro invocation statement will be expanded into the statements that form the body of the macro.
- Arguments from the macro invocation are substituted for the parameters in the macro prototype (according to their positions).
  - In the definition of macro: Parameter
  - In the macro invocation: Argument
- Comment lines within the macro body will be deleted.
- Macro invocation statement itself has been included as a comment line.
- The label on the macro invocation statement has been retained as a label on the first statement generated in the macro expansion.
## MACRO EXPANSION

<table>
<thead>
<tr>
<th>Source</th>
<th>Expanded source</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 MACRO &amp;D1, &amp;D2</td>
<td>.</td>
</tr>
<tr>
<td>STA &amp;D1</td>
<td>.</td>
</tr>
<tr>
<td>STB &amp;D2</td>
<td>.</td>
</tr>
<tr>
<td>MEND</td>
<td>.</td>
</tr>
<tr>
<td>M1 DATA1, DATA2</td>
<td>STA DATA1</td>
</tr>
<tr>
<td>.</td>
<td>STB DATA2</td>
</tr>
<tr>
<td>M1 DATA4, DATA3</td>
<td>STA DATA4</td>
</tr>
<tr>
<td>.</td>
<td>STB DATA3</td>
</tr>
</tbody>
</table>
MACRO INVOCATION

A macro invocation statement (a macro call) gives the name of the macro instruction being invoked and the arguments to be used in expanding the macro.

```
macro_name p1, p2, ...
```

Difference between macro call and procedure call:

Macro call: statements of the macro body are expanded each time the macro is invoked.

Procedure call: statements of the subroutine appear only once, regardless of how many times the subroutine is called.
void exchange(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

main() {
    int i=1, j=3;
    printf("BEFORE - %d %d\n", i, j);
    exchange(i, j);
    printf("AFTER - %d %d\n", i, j);
}
PASS BY REFERENCE

void exchange(int *p1, int *p2) {
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

main() {
    int i=1, j=3;
    printf("BEFORE - %d %d\n", i, j);
    exchange(&i, &j);
    printf("AFTER - %d %d\n", i, j);
}
ASSEMBLY CODE

. Subroutine EXCH
EXCH LDA @P1
STA TEMP
LDA @P2
STA @P1
LDA TEMP
STA @P2
RSUB
P1 RESW 1
P2 RESW 1
TEMP RESW 1

MAIN
LDA #1
STA I
LDA #3
STA J
. Call a subroutine
LDA #I
STA P1
LDA #J
STA P2
JSUB EXCH
I RESW 1
J RESW 1
END MAIN
NO LABEL IN THE MACRO BODY

- Problem of the label in the body of macro:
  - If the same macro is expanded multiple times at different places in the program…
  - There will be *duplicate labels*, which will be treated as errors by the assembler.

- **Solutions:**
  - Do not use labels in the body of macro.
  - Explicitly use PC-relative addressing instead.
  - Ex, in RDBUFF and WRBUFF macros,
    - `JEQ *+11`
    - `JLT *-14`
  - It is inconvenient and error-prone.
ONE-PASS MACRO PROCESSOR

- A one-pass macro processor that alternate between *macro definition* and *macro expansion* in a recursive way is able to handle recursive macro definition.

- **Restriction:**
  - The definition of a macro must appear in the source program before any statements that invoke that macro.
  - This restriction does not create any real inconvenience.
FLOWCHART OF PASS-1 MACRO PROCESSOR

Pass 1

MDTC<1
MNTC<1

Read next source card

MACRO pseudo-op?

Yes

Write copy of source card

Read next source card

Enter macro name and current value of MDTC in MNT entry number MNTC

MNTC <- MNTC + 1

Prepare argument list array

Enter macro name card into MDT

MDTC <- MDTC + 1

No

End pseudo-op?

Yes

Go to Pass 2

No

END pseudo-op?

Yes

Read next source card

MDTC <- MDTC + 1

MEND pseudo-op?
TWO-PASS MACRO PROCESSOR

- You may design a two-pass macro processor

  **Pass 1:**
  - Process all macro definitions

  **Pass 2:**
  - Expand all macro invocation statements

- However, one-pass may be enough

- Because all macros would have to be defined during the first pass before any macro invocations were expanded.

  - The definition of a macro must appear before any statements that invoke that macro.

- Moreover, the body of one macro can contain definitions of other macros.
Pass 2 – processing macro calls and expansion

Read next source card (copied by pass 1)

Search MNT for match with operation code

MACRO name found?

Write into expanded source card file

MDTP <- MDT index from MNT entry

Set up argument list array

MDTP <- MDTP + 1

Get line from MDT

Substitute arguments from macro call

MEND pseudo-op?

Write expanded source card

END pseudo-op?

Supply expanded source file to assembler processing

Yes

No
**DATA STRUCTURES FOR ONE-PASS MACRO PROCESSOR**

<table>
<thead>
<tr>
<th><strong>DEFTAB (definition table)</strong></th>
<th><strong>NAMTAB</strong></th>
<th><strong>ARGTAB</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>❑ Stores the macro definition including <em>macro prototype</em> and <em>macro body</em></td>
<td>❑ Stores macro names Serves as an index to DEFTAB. ❑ The begin Pointers to begining and the end of the macro definition (DEFTAB)</td>
<td>❑ Stores the arguments of macro invocation according to their positions in the argument list ❑ As the macro is expanded, arguments from ARGTAB are substituted for the corresponding parameters in the macro body.</td>
</tr>
<tr>
<td>❑ Comment lines are omitted.</td>
<td>❑ References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.</td>
<td>❑</td>
</tr>
</tbody>
</table>
Macro Processor Data Structures

DATA STRUCTURES
ALGORITHM

**Procedure EXPAND**
- Set up the argument values in ARGTAB
- Expand a macro invocation statement (like in MAIN procedure)
- Iterations of
  - GETLINE
  - PROCESSLINE

**Procedure GETLINE**
- If EXPANDING then
  - get the next line to be processed from DEFTAB
- Else
  - read next line from input file

**Procedure PROCESSLINE**
- DEFINE
- EXPAND
- Output source line

**Procedure DEFINE**
- Make appropriate entries in DEFTAB and NAMTAB

**MAIN program**
- Iterations of
  - GETLINE
  - PROCESSTLINE
## COMPARISON OF MACRO PROCESSORS DESIGN

<table>
<thead>
<tr>
<th>One-pass algorithm</th>
<th>Two-pass algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ Every macro must be defined before it is called</td>
<td>➢ Pass1: Recognize macro definitions</td>
</tr>
<tr>
<td>➢ One-pass processor can alternate between macro definition and macro expansion</td>
<td>➢ Pass2: Recognize macro calls</td>
</tr>
<tr>
<td>➢ Nested macro definitions are allowed but nested calls are not</td>
<td>Nested macro definitions are not allowed</td>
</tr>
</tbody>
</table>
THANK YOU

Prof. SURYAVANSHI ARTI PRASHANT
E-mail:- artips15@gmail.com

Prof. SURYAVANSHI PRASHANT MAHARUDRA
E-mail:- sprashant1234@gmail.com

HSBPVT’s PARIKRAMA GOI COE KASHTI