***Malaviya National Institute Of Technology***

***Jaipur(Rajasthan)***

******

**Systems Programming Lab**

**Project Documentation**

**Multi-purpose MACRO Pre-Processor**

Personal details-

* Name-Ashish Goyal
* College Id-2016ucp1100
* Phone Number-9024686835
* Email id-2016ucp1100@mnit.ac.in
* A1 Batch

Professor in-charge: Dr. Arka Prokash Mazumdar

Part1-Design Goal:

a. Parameter Substitution (positional and parametric, default value)

b. Nested MACRO definitions

c. Comments

**CONTENTS:**

1. MACROS

2. Macro Pre- processor

3. Syntax of the designed macro.

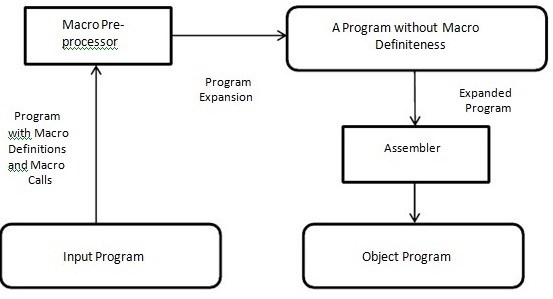
4. Macro examples to demonstrate the specific features.

1. MACROS:

* A macro is a unit of specification for program generation through expansion.
* A Macro consists of a name, a set of formal parameters and a body of code.
* The use of Macro name with a set of actual parameters is replaced by code generated from its body.
* Macro is used when same thing is to be done in a program a number of times.

**2.Macro Pre- processor :**

* Macro preprocessors are vital for processing all programs that contain macro definitions and/or calls.
* Language translators such as assemblers and compilers cannot directly generate the target code from the programs containing definitions and calls for macros.
* Therefore, most language processing activities by assemblers and compilers preprocess these programs through macro processors.
* A macro preprocessor essentially accepts an assembly program with macro definitions and calls as its input and processes it into an equivalent expanded assembly program with no macro definitions and calls.
* The macro preprocessor output program is then passed over to an assembler to generate the target object program.



3. Syntax of the designed macro:

Macro definition(multiline):

**@@BEGIN** ...macro\_name... (parameters)

Inner code with comment that will expand later

**@@END**

* **Macro Start with @@BEGIN**
* **Macro end with @@END**

Macro call:

* ...macro\_name... (para1,para2,…)
* para1,para2:: can be anything string,integer etc

Macro\_name **:**

* Any name without fixed keyword
* Start and end with …(triple dot)

**Parameters :**

(&d1\_=1,&d2\_=1,&d3\_=1,&d4\_=1,…….)

* Parameter name starts with ‘&’ and end with ‘\_’
* All parameters are given default value in input file.

Here default value for example, for &d1\_ is 1.

* Order can be anything of the arguments, value assign according to position of the parameter .

i.e (&d8\_=1,&d2\_=1,&d32\_=1,&d4\_=1,…….)

Syntax for Comments:

* Comment for macro start with “<#” and end with “#>”
* Example: <# you are in macro comment #>

Nested macro:

**@@BEGIN** ...PRINT... (parameter)

Inner code that will expand later

**@@BEGIN** ...QUIT... (parameter)

Inner code that will expand later

**@@END**

**@@END**

**Note:**

* **Macro inside another macro can’t be called before expanding**

**the outer macro first.**

* **After their definations stored in deftab and macro\_name store in nametab we can call nested macro independently.**

4. Macro examples to demonstrate the specific features:

**Ex1. Comments:**

@@BEGIN ...SINGLE... (&d1\_=1,&d2\_=1,&d3\_=1,&d4\_=1,&d5\_=1)

<#YOU ARE IN MACRO SINGLE#>

integerr:db '%d',&d1\_,&d2\_

@@END

extern printf

extern atoi

SECTION .data

...SINGLE... (20,10)

SECTION .text

global main

main:

push ebp

mov ebp,esp

int 80h

EXPANDED CODE:

extern printf

extern atoi

SECTION .data

integerr:db '%d',20,10

SECTION .text

global main

main:

push ebp

mov ebp,esp

int 80h

**Ex2. Parameter Substitution :**

@@BEGIN ...SINGLE... (&d1\_=1,&d2\_=1,&d3\_=1,&d4\_=1,&d5\_=1,&d6\_=1,&d7\_=1,&d8\_=1,&d8\_=1,&d9=1,&d10\_=1)

<#YOU ARE IN MACRO SINGLE#>

integerr:db '%d',&d1\_,&d2\_

@@END

@@BEGIN ...MULTI... (&d1\_=1,&d2\_=1,&d3\_=1,&d4\_=1,&d5\_=1,&d6\_=1,&d7\_=1,&d8\_=1,&d8\_=1,&d9=1,&d10\_=1)

i RESB &d1\_ <# hELLO THIS IS i #>

n RESB &d2\_

Sum RESB &d3\_

@@END

extern printf

extern atoi

SECTION .data

...SINGLE... (20,10)

SECTION .bss

...MULTI... (64,32,32)

SECTION .text

global main

main:

push ebp

mov ebp,esp

mov esp,ebp

pop ebp

int 80h

**EXPANDED CODE:**

extern printf

extern atoi

SECTION .data

integerr:db '%d',20,10

SECTION .bss

i RESB 64

n RESB 32

Sum RESB 32

SECTION .text

global main

main:

push ebp

mov ebp,esp

mov esp,ebp

pop ebp

int 80h

**Ex3. Nested MACRO definitions**: (the original code )

@@BEGIN ...SINGLE... (&d1\_=1,&d2\_=1,&d3\_=1,&d4\_=1,&d5\_=1,&d6\_=1,&d7\_=1,&d8\_=1,&d8\_=1,&d9=1,&d10\_=1)

<#YOU ARE IN MACRO SINGLE#>

integerr:db '%d',&d1\_,&d2\_

@@END

@@BEGIN ...MULTI... (&d1\_=1,&d2\_=1,&d3\_=1,&d4\_=1,&d5\_=1,&d6\_=1,&d7\_=1,&d8\_=1,&d8\_=1,&d9=1,&d10\_=1)

i RESB &d1\_ <# hELLO THIS IS i #>

n RESB &d2\_

Sum RESB &d3\_

@@END

@@BEGIN ...PRINT... (&d10\_=1,&d2\_=1,&d3\_=1,&d4\_=1,&d5\_=1,&d6\_=1,&d7\_=1,&d8\_=1,&d8\_=1,&d9=1,&d1\_=1)

mov eax,[Sum]

push &d10\_

push integerr

call printf

@@BEGIN ...QUIT... (&d1\_=1,&d2\_=1,&d3\_=1,&d4\_=1,&d5\_=1,&d6\_=1,&d7\_=1,&d8\_=1,&d8\_=1,&d9=1,&d10\_=1)

mov esp,ebp

pop ebp

ret

@@BEGIN ...FINISH... (&d1\_=80h,&d2\_=1,&d3\_=1,&d4\_=1,&d5\_=1,&d6\_=1,&d7\_=1,&d8\_=1,&d8\_=1,&d9=1,&d10\_=1)

<# BYE BYE#>int &d1\_

@@END

@@END

@@END

extern printf

extern atoi

SECTION .data

...SINGLE... (20,10)

SECTION .bss

...MULTI... (64,32,32)

SECTION .text

global main

main:

push ebp

mov ebp,esp

mov ebx,dword[esp+12]

mov ecx,[ebx+4]

push ecx

call atoi

mov [i],eax

mov edx,[ebx+8]

push edx

call atoi

mov [n],eax

mov eax,1

mov [Sum],eax

loop:

mov eax,[n]

inc eax

cmp eax,[i]

jz end

mov eax,[i]

mov edx,[i]

mul edx

mov edx,[Sum]

mul edx

mov [Sum],eax

mov eax,[i]

inc eax

mov [i],eax

jmp loop

end:

...PRINT... (eax)

...FINISH... (90h)

**EXPANDED CODE:**

extern printf

extern atoi

SECTION .data

integerr:db '%d',20,10

SECTION .bss

i RESB 64

n RESB 32

Sum RESB 32

SECTION .text

global main

main:

push ebp

mov ebp,esp

mov ebx,dword[esp+12]

mov ecx,[ebx+4]

push ecx

call atoi

mov [i],eax

mov edx,[ebx+8]

push edx

call atoi

mov [n],eax

mov eax,1

mov [Sum],eax

loop:

mov eax,[n]

inc eax

cmp eax,[i]

jz end

mov eax,[i]

mov edx,[i]

mul edx

mov edx,[Sum]

mul edx

mov [Sum],eax

mov eax,[i]

inc eax

mov [i],eax

jmp loop

end:

mov eax,[Sum]

push eax

push integerr

call printf

mov esp,ebp

pop ebp

ret

int 80h

int 90h

PART-2-Design Goal:

a. Single-line/multi-line definitions

b. Conditional Macros.

**CONTENTS:**

1. Syntax of the designed macro.

2. Macro examples to demonstrate the specific features.

1.Syntax of the designed macro:

Single line Macro definition:

* **SYNTAX:**

**@@BEGINN** ...macro\_name... (parameters)Inner code **@@ENDD**

* For single line macro definition macro starts with **@@BEGINN** and ends it with **@@ENDD** in the same line.
* Macro naming, parameters, comments and macro calling are same as that for multiline definitions.(see part1)

Conditional macros:

1)if-else:

Syntax:

@@BEGIN ...macro\_name... (arguments)

Iff(condition)

inner code

elsee

inner code

endif

@@END

* If block starts with **iff**
* Else block starts with **elsee**
* If block end with **endif**
* For checking conditions instead of symbols “<,>,=”etc, use GREATER,LESS,EQUAL etc

2)while loop:

Syntax:

@@BEGIN ...macro\_name... (arguments)

WHILEE (condition)

Inner code

ENDWHIL

@@END

* while block starts with **WHILEE**
* while block ends with **ENDWHIL**
* For checking conditions instead of symbols “<,>,=”etc,use GREATER,LESS,EQUAL etc
* For increment a variable- INRR
* For decrement a variable- DCRR

2. Macro examples to demonstrate the specific features:

EX1:(single line macro defination)

@@BEGINN ...SINGLE... (&d1\_=1,&d2\_=1,&d3\_=1,&d4\_=1) integerr:db '%d',&d1\_,&d2\_ @@ENDD

extern printf

extern atoi

SECTION .data

...SINGLE... (20,10)

**EXPANDED CODE:**

extern printf

extern atoi

SECTION .data

integerr:db '%d',20,10

EX2:(both conditional [if-else and while])

@@BEGIN ...MULTI... (&d1\_=1,&d2\_=2,&d3\_=1,&d4\_=1,&d5\_=1,&d6\_=1,&d7\_=1,&d8\_=1,&d8\_=1,&d9=1,&d10\_=16)

iff(&d1\_ EQUAL &d2\_)

i RESB &d1\_

elsee

j RESB &d2\_

endif

WHILEE (&d4\_ LESS &d5\_)

INRR &d4\_ <#use INRR and DCRR according to the condition in WHILEE such that it will remain valid#>

n RESB &d3\_

ENDWHIL

k RESB &d10\_

@@END

extern printf

extern atoi

SECTION .data

SECTION .bss

...MULTI... (62,64,32,1,5)

SECTION .text

global main

main:

push ebp

mov ebp,esp

mov ebx,dword[esp+12]

mov ecx,[ebx+4]

push ecx

call atoi

mov [i],eax

mov edx,[ebx+8]

push edx

call atoi

mov [n],eax

mov eax,1

mov [Sum],eax

loop:

mov eax,[n]

inc eax

cmp eax,[i]

jz end

mov eax,[i]

mov edx,[i]

mul edx

mov edx,[Sum]

mul edx

mov [Sum],eax

mov eax,[i]

inc eax

mov [i],eax

jmp loop

int 80h

Expanded code:

extern printf

extern atoi

SECTION .data

SECTION .bss

j RESB 64

n RESB 32

n RESB 32

n RESB 32

n RESB 32

k RESB 16

SECTION .text

global main

main:

push ebp

mov ebp,esp

mov ebx,dword[esp+12]

mov ecx,[ebx+4]

push ecx

call atoi

mov [i],eax

mov edx,[ebx+8]

push edx

call atoi

mov [n],eax

mov eax,1

mov [Sum],eax

loop:

mov eax,[n]

inc eax

cmp eax,[i]

jz end

mov eax,[i]

mov edx,[i]

mul edx

mov edx,[Sum]

mul edx

mov [Sum],eax

mov eax,[i]

inc eax

mov [i],eax

jmp loop

int 80h