| ***NAME: TWINKAL PARMAR***  ***ID:2016UCP1700***  ***BATCH: (3,4)***  ***SUB:SP LAB***  TOPIC :MACROS |  |  |  |
| --- | --- | --- | --- |

MACROS:

It is just a block of codes that can be used with input parameter anywhere in the program with name.

WHY DO WE NEED THIS?

* Reusability with input parameter.
* Reduce complexity.
* Do not need to write same instruction again and again in the code.

SYNTAX:

1. MACRO DEFINITION:

MACRO->[name\_of\_the\_macro]

Number\_of\_argument

~arg1,~arg2,~arg3…………

instruction1

instruction2 /c/ any explaination

…..

….

…

[name\_of\_the\_macro]<-MACRO

1. CALLING:

Name\_of\_the\_macro ARG (number\_of\_argument):a1,a2….

1. COMMENT:

🡪/c/:single line comment

🡪Text after this(/c/) will not appear in the expanded part.

RULES :

1. Every macro should define at the start of the program.

2.We must follow the sequence in which the above macro is defined.

3.Number of argument should be>=0.Means if there is no arguments in the macro definition is required than we have to explicitly write 0.

3.“~” symbol is compulsory to add with the argument (Only the time of definition).

4.Number of arguments (At the time of call) is already defined in definition, so we cannot change it. It must be same as in the macro definition.

5.We cannot leave number of argument part (In calling) blank. (if argument>0)

6. If there is no argument in the macro than we can simply call the macro just by name. Do not need to specify “ARG (0):” (only at the time of calling).

7. Default value is NULL.

8. Arguments are separated by comma.

9.If the actual arguments are lesser than the formal argument then it will take the default value for the rest of the argument (At the time of expanding).

10.We must follow the syntax of the macro, otherwise we may get wrong result.

11. No restriction for the name of arguments (It must not be the name of any instruction).

12.Expanded code does not have any macro comment.

13.Same macro definition and calling can be used for both NASM and GAS.

14. Name of the macro should not be substring of any instruction.

HOW IT WORKS?

Code with macro Expanded code

|  |  |
| --- | --- |
| EXAMPLE 0:  MACRO->[abc]  1  ~data  mov eax,data /c/ comment  mov ebx,data  [abc]->MACRO  ..  ..  abc(1) ARG:1  ..  ..  ..  ..  ..  abc(1) ARG:alpha  ..  ..  .. | EXAMPLE 0:  .  .  .  .  .  mov eax,1  mov ebx,1  ..  ..  ..  ..  ..  mov eax,alpha  mov ebx,alpha  ..  ..  .. |

Note that: In the above example, expanded code did not have the macro comment and also formal argument is replaced by actual argument. It means it does not copy the definition part as it is.

FEATURES:

1. PARAMETER SUBSITUTION:

🡪This feature replaces formal arguments with actual arguments.

🡪If no argument, then it expands the macro same as the definition.

🡪Substitutes the “NULL” value whenever required (Default value).

Code with macro Expanded code

|  |  |
| --- | --- |
| EXAMPLE 1: (No argument)  MACRO->[into]  0  push ebp  mov ebp,esp  [into]<-MACRO  MACRO->[out]  0  mov esp,ebp  pop ebp  [out]<-MACRO  SECTION .data  var db 'this is my first nasm program',0,10  SECTION .text  extern printf  global main  main:  into  push var  call printf  out  ret | EXAMPLE 1: (No argument)  SECTION .data  var db 'this is my first nasm program',0,10  SECTION .text  extern printf  global main  main:  push ebp  mov ebp,esp  push var  call printf  mov esp,ebp  pop ebp  ret |
| Example2: (Parametric)  MACRO->[to\_print]  1  ~arg1  push ~arg1  call printf  [to\_print]<-MACRO  SECTION .data  var db 'this is my first nasm program',0,10  SECTION .text  extern printf  global main  main:  push ebp  mov ebp,esp  to\_print ARG(1):var  mov esp,ebp  pop ebp  ret | Example2: (Parametric)  SECTION .data  var db 'this is my first nasm program',0,10  SECTION .text  extern printf  global main  main:  push ebp  mov ebp,esp  push var  call printf  mov esp,ebp  pop ebp  ret |
| EXAMPLE3: (Positioned)  MACRO->[repeat]  4  ~arg1,~arg2,~arg3,~arg4  mov eax, ~arg1  mov ebx, ~arg2  mov ecx, ~arg3  mov edx, ~arg4  int 80h  [repeat]<-MACRO  section .data  userMsg db 'Please enter a number: '  lenUserMsg equ $-userMsg  dispMsg db 'You have entered: '  lenDispMsg equ $-dispMsg  section .bss  num resb 5  section .text  global main  main:  repeat ARG(4):4,1,userMsg,lenUserMsg  repeat ARG(4):3,2,num,5  repeat ARG(4):4,1,dispMsg,lenDispMsg  repeat ARG(4):4,1,num,5  mov eax, 1  mov ebx, 0  int 80h | EXAMPLE3: (Positioned)  section .data  userMsg db 'Please enter a number: '  lenUserMsg equ $-userMsg  dispMsg db 'You have entered: '  lenDispMsg equ $-dispMsg  section .bss  num resb 5  section .text  global main  main:  mov eax, 4  mov ebx, 1  mov ecx, userMsg  mov edx, lenUserMsg  int 80h  mov eax, 3  mov ebx, 2  mov ecx, num  mov edx, 5  int 80h  mov eax, 4  mov ebx, 1  mov ecx, dispMsg  mov edx, lenDispMsg  int 80h    mov eax, 4  mov ebx, 1  mov ecx, num  mov edx, 5  int 80h  mov eax, 1  mov ebx, 0  int 80h |
| EXAMPLE4: ( Default value)  MACRO->[camp]  4  ~arg1,~arg2,~arg3,~arg4  cmp esi,ebx  JNS ~arg1  push ~arg2  push ~arg3  call printf  add esp,8  JMP ~arg4  [camp]<-MACRO  SECTION .data  msg1 db "1st number=%d",0,10  msg2 db "2nd number=%d",0,10  msg3 db "largest=%d",0,10  msg4 db "smallest=%d",0,10  SECTION .text  extern printf  extern atoi  global main  main:  push ebp  mov ebp,esp  mov ebx,DWORD[esp+12]  mov esi,[ebx+4];ebx=1st number  mov ebx,[ebx+8]  push esi  call atoi  mov esi,eax  add esp,4  push esi  push msg1  call printf  add esp,8  push ebx  call atoi  mov ebx,eax  add esp,4  push ebx  push msg2  call printf  add esp,8  camp ARG(4):L1,ebx,msg3,small  L1:  push esi  push msg3  call printf  add esp,8  small:  camp ARG(4):L2,esi,msg4  L2:  push ebx  push msg4  call printf  add esp,8  end:  mov esp,ebp  pop ebp  ret | EXAMPLE4: (Default value)  SECTION .data  msg1 db "1st number=%d",0,10  msg2 db "2nd number=%d",0,10  msg3 db "largest=%d",0,10  msg4 db "smallest=%d",0,10  SECTION .text  extern printf  extern atoi  global main  main:  push ebp  mov ebp,esp  mov ebx,DWORD[esp+12]  mov esi,[ebx+4];ebx=1st number  mov ebx,[ebx+8]  push esi  call atoi  mov esi,eax  add esp,4  push esi  push msg1  call printf  add esp,8  push ebx  call atoi  mov ebx,eax  add esp,4  push ebx  push msg2  call printf  add esp,8  cmp esi,ebx  JNS L1  push ebx  push msg3  call printf  add esp,8  JMP small  L1:  push esi  push msg3  call printf  add esp,8  small:  cmp esi,ebx  JNS L2;means esi>ebx  push esi;means esi<ebx  push msg4  call printf  add esp,8  JMP NULL  L2:  push ebx  push msg4  call printf  add esp,8  end:  mov esp,ebp  pop ebp  ret |

1. NESTED MACRO DEFINITION:

🡪We can define macro within a macro definition.

🡪At the time of expand it will take the formal arguments as actual arguments (for the macro which is defined inside the other macro).

Code with macro Expanded code

|  |  |
| --- | --- |
| EXAMPLE5:  MACRO->[nested1]  1  ~arg1  push ~arg1  call atoi;  add esp,4  MACRO->[nested2]  1  ~arg1  mov ~arg1,eax  push ~arg1  push msg1  call printf  add esp,8  [nested2]<-MACRO  [nested1]<-MACRO    SECTION .data  msg1 db "you entered=%d",0,10  msg2 db "you entered=%d",0,10  SECTION .text  extern printf  extern atoi  global main  main:  push ebp  mov ebp,esp  mov ebx,DWORD[esp+12]  mov esi,[ebx+4]  mov ebx,[ebx+8]  nested1 ARG(1):esi  nested1 ARG(1):ebx  mov esp,ebp  pop ebp  ret | EXAMPLE5:  SECTION .data  msg1 db "you entered=%d",0,10  msg2 db "you entered=%d",0,10  SECTION .text  extern printf  extern atoi  global main  main:  push ebp  mov ebp,esp  mov ebx,DWORD[esp+12]  mov esi,[ebx+4]  mov ebx,[ebx+8]  push esi  call atoi;  add esp,4  mov esi,eax  push esi  push msg1  call printf  add esp,8  push ebx  call atoi;  add esp,4  mov ebx,eax  push ebx  push msg1  call printf  add esp,8  mov esp,ebp  pop ebp  ret |

1. Comments:

🡪The scope of the macro comments is only its definition part.

🡪Expanded code does not have macro comments.

🡪It is helpful for the user to understand the macro definition without difficulty.

Code with macro Expanded code

|  |  |
| --- | --- |
| EXAMPLE6:  MACRO->[str\_to\_int]  1  ~arg1  push ~arg1  call atoi /c/ this is c function which convert string to int  add esp,4 /c/ ans will store in the eax register  mov ~arg1,eax /c/ so we have to move ans to the required register  [str\_to\_int]<-MACRO  MACRO->[to\_print]  2  ~arg1,~arg2  push ~arg1  push ~arg2  call printf /c/ this is c function which will print on the screen  add esp,8  [to\_print]<-MACRO  SECTION .data  msg1 db "You entered=%d",0,10  msg2 db "Factorail=%d",0,10  SECTION .text  extern printf  extern atoi  global main  main:  push ebp  mov ebp,esp  mov ebx,DWORD[esp+12]  mov ebx,[ebx+4]  str\_to\_int ARG(1):ebx  to\_print ARG(2):ebx,msg1  mov eax,1  L1:  MUL ebx  dec ebx  JNZ L1  To\_print ARG(2):eax,msg2  mov esp,ebp  pop ebp  ret | EXAMPLE6:  SECTION .data  msg1 db "You entered=%d",0,10  msg2 db "Factorail=%d",0,10  SECTION .text  extern printf  extern atoi  global main  main:  push ebp  mov ebp,esp  mov ebx,DWORD[esp+12]  mov ebx,[ebx+4]  push ebx  call atoi  add esp,4  mov ebx,eax  push ebx  push msg1  call printf  add esp,8  mov eax,1  L1:  MUL ebx  dec ebx  JNZ L1  push eax  push msg2  call printf  add esp,8  mov esp,ebp  pop ebp  ret |

1. Single line macro:

Syntax:

🡪[DEFINE] macro\_name: instruction

or

🡪[DEFINE] macro\_name(~arg1,~arg2,~arg3---):instruction

🡪Must define in the section. data.

|  |  |
| --- | --- |
| EXAMPLE7: (in gas)  [DEFINE] abc(~arg1):$~arg1,%rax  [DEFINE] x:$13  .global \_start          .text \_start:         mov     abc(1)                  mov     $1, %rdi                         mov     $message, %rsi                   mov     x, %rdx                       syscall                                  mov   abc(60)                      xor     %rdi, %rdi                      syscall                          message:         .ascii  "Hello, world\n" | EXAMPLE7:  .global \_start          .text \_start:         mov     $1, %rax                        mov     $1, %rdi                         mov     $message, %rsi                   mov     $13, %rdx                       syscall                                  mov     $60, %rax                      xor     %rdi, %rdi                      syscall                          message:         .ascii  "Hello, world\n" |
| EXAMPLE8: (in nasm)  [DEFINE] twinkal:0x80  [DEFINE] length(~arg1):equ $-~arg1  SECTION .data  msg: db "Hello World",10  len: length(msg)    SECTION .text  global main  main:    mov edx,len  mov ecx,msg  mov ebx,1  mov eax,4  int twinkal  mov ebx,0  mov eax,1  int twinkal | EXAMPLE8:  SECTION .data  msg: db "Hello World",10  len: equ $-msg    SECTION .text  global main  main:    mov edx,len  mov ecx,msg  mov ebx,1  mov eax,4  int 0x80  mov ebx,0  mov eax,1  int 0x80 |

e.)conditional macro:

MACRO-PREPROSSEOR DIRECTIVE:

🡪IF

🡪END\_IF

🡪ELSE

🡪END\_ELSE

SYNTAX:

MACRO->[macro\_name]

number\_of\_arg

~arg1,~arg2,~arg3-----

Instructions

IF{condition}

..

..

END\_IF

IF{condition}

..

..

..

END\_IF

ELSE

..

..

..

END\_ELSE

..

..

[macro\_name]<-MACRO

Note: In macro definition, if there is at least one IF statement than there must be ELSE in the macro definition.

|  |  |
| --- | --- |
| EXAMPLE9:  MACRO->[conditional]  1  ~arg1  mov ebx, ~arg1  IF{~arg1==1}  mov eax,4  END\_IF  IF{~arg1==0}  mov eax,1  END\_IF  int 0x80  ELSE  END\_ELSE  [conditional]<-MACRO  Section .data  msg1 db “hellow”,10  len equ $-msg  section .text  global main  main:  mov edx,len  mov ecx,msg  conditional(1):1  conditional(1):0 | EXAMPLE9:  Section .data  msg1 db “hellow”,10  len equ $-msg  section .text  global main  main:  mov edx,len  mov ecx,msg  mov ebx,1  mov eax,4  int 0x80  mov ebx,0  mov eax,1  int 0x80 |
| EXAMPLE10:  ..  ..  MACRO->[check]  4  ~arg1,~arg2,~arg3,~arg4  call atoi  IF{~arg1==0}  mov eax ,4  mov ebx,1  mov ecx,~arg3  mov edx,~arg4  END\_IF  IF{~arg1==1}  mov eax ,3  mov ebx,0  mov ecx,~arg2  mov edx,~arg4  END\_IF  ELSE  mov eax ,0  END\_ELSE  ..  ..  [check]<-MACRO  ..  ..  ..  check(0,1,msg,len)  check(1,msg2,msg,len)  check(4,4,msg,len)  .  .. | EXAMPLE10:  ..  ..  ..  call atoi  mov eax ,4  mov ebx,1  mov ecx,msg  mov edx, len  .  ..  ..  call atoi  mov eax ,3  mov ebx,0  mov ecx,msg2  mov edx,len  call atoi  mov eax ,0 |