INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

Macro Processors

Ref: L. L. Beck Book Chapter 4

Position-Independent Code (PIC)

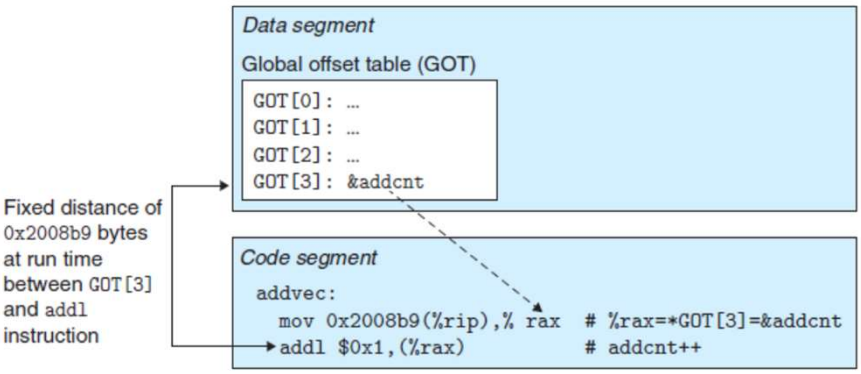
- Code segments of the shared modules are compiled so that they can be loaded anywhere in memory without having to be modified by the linker => No relocations are needed
- Compile using –fpic option when using gcc
- Compilers that generate PIC references to global variables use a table called the *global offset table (GOT)*
- This table is generally at the beginning of the data segment
- The **GOT contains an entry for each global data object** (procedure or global variable) that is referenced by the object module.
- The **compiler also generates a relocation record for each entry in the GOT.**

IIT ROORKEE

code/link/ab

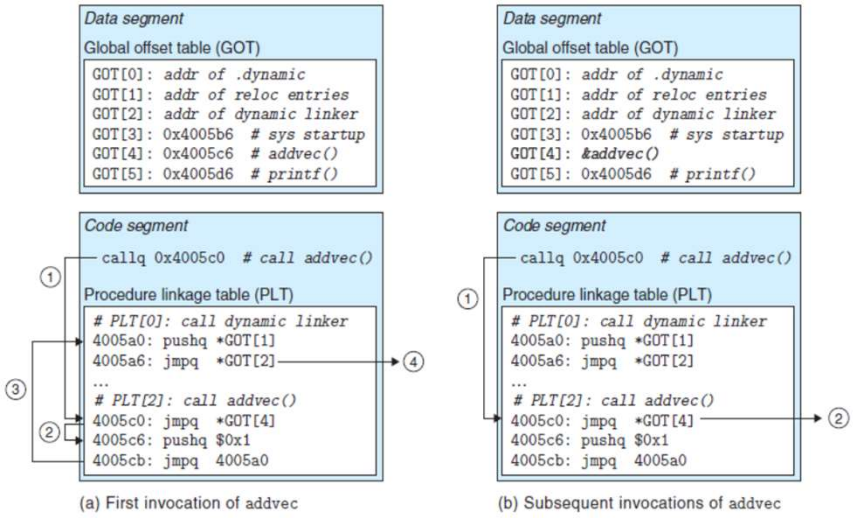
```
int addcnt = 0;

void addvec(int *x, int *y,
            int *z, int n)
{
    int i;

    addcnt++;

    for (i = 0; i < n; i++)
        z[i] = x[i] + y[i];
}
```


Using the GOT to reference a global variable



- At load time, the dynamic linker relocates each GOT entry so that it contains the absolute address of the object.
- Each object module that references global objects has its own GOT.

Using the PLT and GOT to call external functions

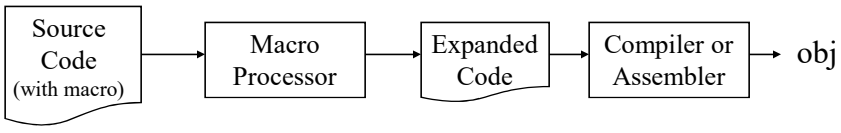


Lazy Binding

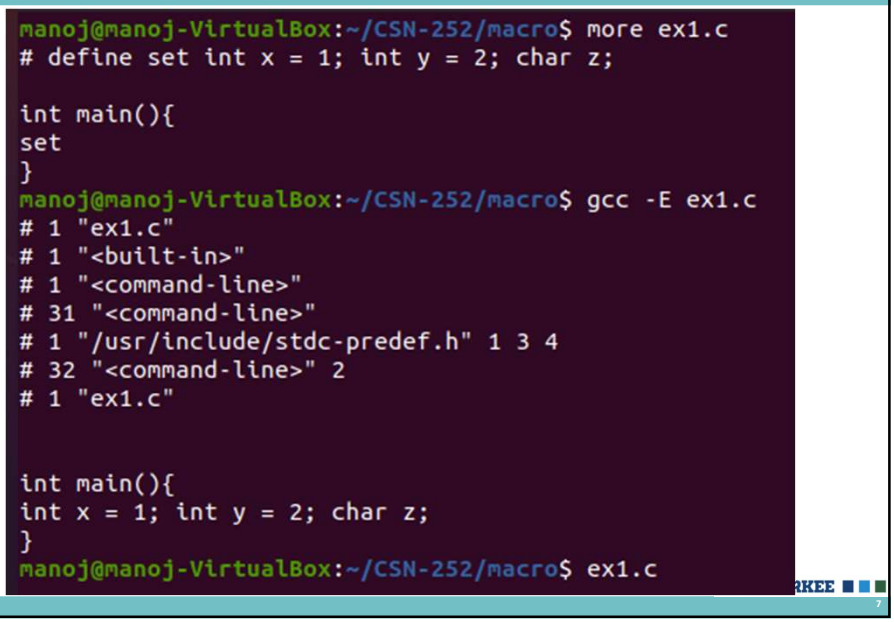
IIT ROORKEE

Introduction

- A macro instruction is a notational convenience for the programmer
- It allows the programmer to write **shorthand version** of a set of instructions
- The macro processor replaces each **macro invocation** with the corresponding sequence of statements (*expanding*)



Example from C preprocessor



```
manoj@manoj-VirtualBox:~/CSN-252/macro$ more ex1.c
# define set int x = 1; int y = 2; char z;

int main(){
set
}
manoj@manoj-VirtualBox:~/CSN-252/macro$ gcc -E ex1.c
# 1 "ex1.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "ex1.c"


int main(){
int x = 1; int y = 2; char z;
}
manoj@manoj-VirtualBox:~/CSN-252/macro$ ex1.c
```

Macro Processor

- Function
  - Substitute one group of characters or lines for another
  - No analysis of the text that it handles
  - Design is not directly related to the architecture of the computer
  - Can be used with assembler language / high-level programming language

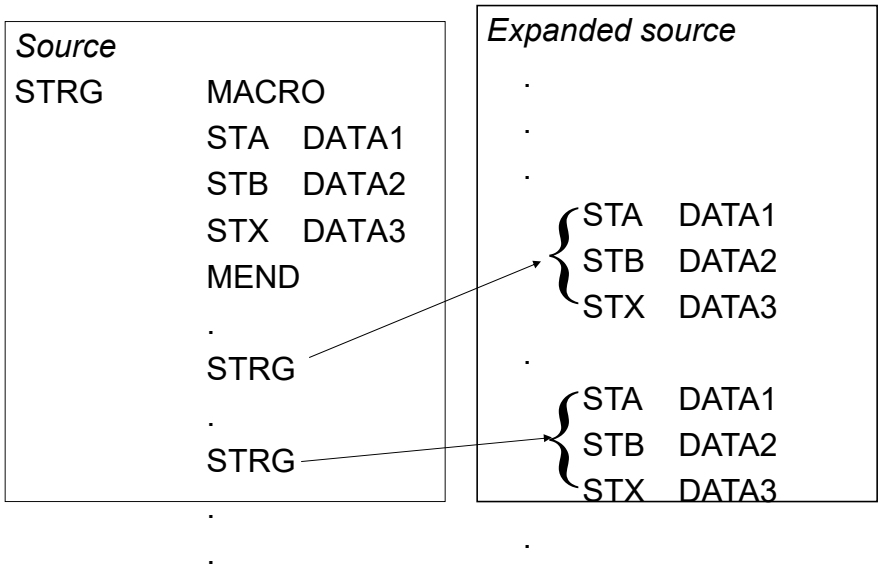
Macro Processor

- Recognize macro definitions
  - Save the macro definition
- Recognize macro calls
  - Expand macro calls

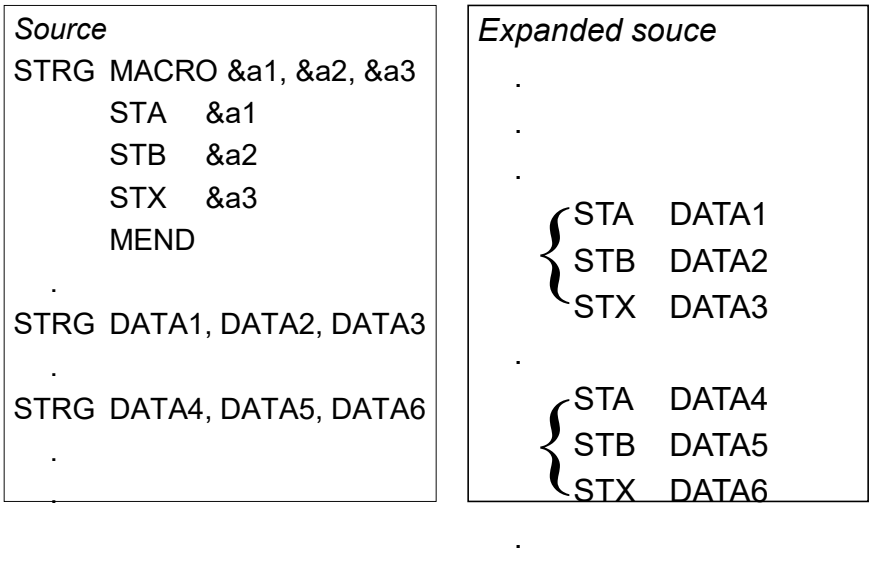
Macro Definition

- copy code
- parameter substitution
- conditional macro expansion
- macro instruction defining macros

Copy code -- Example



Parameter Substitution -- Example



Macro Definition and Expansion

Macro name

Macro parameters

Macro invocation / call

arguments

```
1   copy    start   0
2   rdrec   macro   &in, &buf, & recl
3           clear   x
4           clear   a
5           clear   s
6           +ldt    #4096
7           td      =x'&in'
8           jeq     *-3
9           rd      =x'&in'
10          compr   a, s
11          jeq     *+11
12          stch    &buf,x
13          tixr    t
14          jlt     *-19
15          stx     &recl
16          mend
17  first   stl     retadr
18  cloop   rdrec   f1, buf, length
19          lda     length
20          comp    #0
21          jeq     endfil
22          j       cloop
23  endfil  J       @retadr
24  eof     byte    c'eof'
25  three   word    3
26  retadr  resw    1
27  length  resw    1
28  buffer  resb    4096
29          end     first
```

- Two new assembler directives
- Body of the macro contains no labels
- *-3 – previous instruction

Macro Definition and Expansion

```
1   copy    start   0
2   rdrec   macro   &in, &buf, & recl
3           clear   x
4           clear   a
5           clear   s
6           +ldt    #4096
7           td      =x'&in'
8           jeq     *-3
9           rd      =x'&in'
10          compr   a, s
11          jeq     *+11
12          stch    &buf,x
13          tixr    t
14          jlt     *-19
15          stx     &recl
16          mend
17. first   stl     retadr
18. cloop   rdrec   f1, buf, length
```

Code after the expansion of macro

```
1   copy    start   0
2   first   stl     retadr
3   .cloop  rdrec   f1, buf, length
4   cloop   clear   x
2           clear   a
5           clear   s
6           +ldt    #4096
7           td      =x'f1'
8           jeq     *-3
9           rd      =x'f1'
10          compr   a, s
11          jeq     *+11
12          stch    buf,x
13          tixr    t
14          jlt     *-19
15          stx     length
```

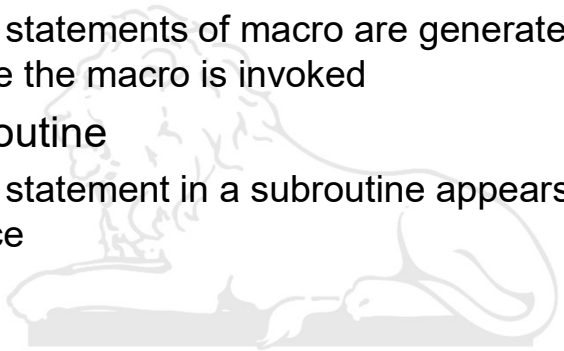
Macro Definition and Expansion

Output of macro processor

```
1     copy      start   0
2     first     stl     retadr
3     .cloop    rdrec   f1, buf, length
4     cloop     clear   x
2               clear   a
5               clear   s
6               +ldt    #4096
7               td      =x'f1'
8               jeq     *-3
9               rd      =x'f1'
10              compr   a, s
11              jeq     *+11
12              stch    buf,x
13              tixr    t
14              jlt     *-19
15              stx     length
19              lda     length
20              comp    #0
21              jeq     endfil
22              j       cloop
23    endfil    J       @retadr
24    eof       byte    c'eof'
25    three     word    3
26    retadr    resw    1
27    length    resw    1
28    buffer    resb    4096
29              end     first
```

Macro vs. Subroutine

- Macro
  - the statements of macro are generated each time the macro is invoked
- Subroutine
  - the statement in a subroutine appears only once



Parameter Substitution

- Dummy arguments
  - Positional argument

```
RDBUF  MACRO  &IN, &BUFADR, &RECLTH, &EOR, &MAX
RDBUF  F3, BUF, RECL, 04, 2048
RDBUF  0E, BUFFER, LENGTH,,80

GENER  MACRO  &A, &B, &C, &D, &E, &F, &G, &H, &I
GENER    ,,DIRECT,,,,,,3
```

Parameter Substitution

- Dummy arguments
  - Keyword argument

```
RDBUF  MACRO    &IN, &BUFADR, &RECLTH, &EOR, &MAX
RDBUF       BUFADR=BUFFER, RECLETH=LENGTH


GENER  MACRO  &A, &B, &C, &D, &E, &F, &G, &H, &I
GENER       C=DIRECT, I=3
```

Concatenation of Macro Parameters

Most macro processors allow parameters to be concatenated with other character strings.

```
LDA     XA1
ADD     XA2
ADD     XA3
STA     XAS
```

```
LDA     XBETA1
ADD     XBETA2
ADD     XBETA3
STA     XBETAS
```

Concatenation of Macro Parameters

- Pre-concatenation
  - LDA X&ID1
- Post-concatenation
  - LDA X&ID→1
- Example:

```
1  SUM     MACRO   &ID
2          LDA     X&ID→1
3          ADD     X&ID→2
4          ADD     X&ID→3
5          STA     X&ID→S
6          MEND
```

(a)

```
SUM     A
```

↓

```
LDA     XA1
ADD     XA2
ADD     XA3
STA     XAS
```

(b)

```
SUM     BETA
```

↓

```
LDA     XBETA1
ADD     XBETA2
ADD     XBETA3
STA     XBETAS
```

(c)