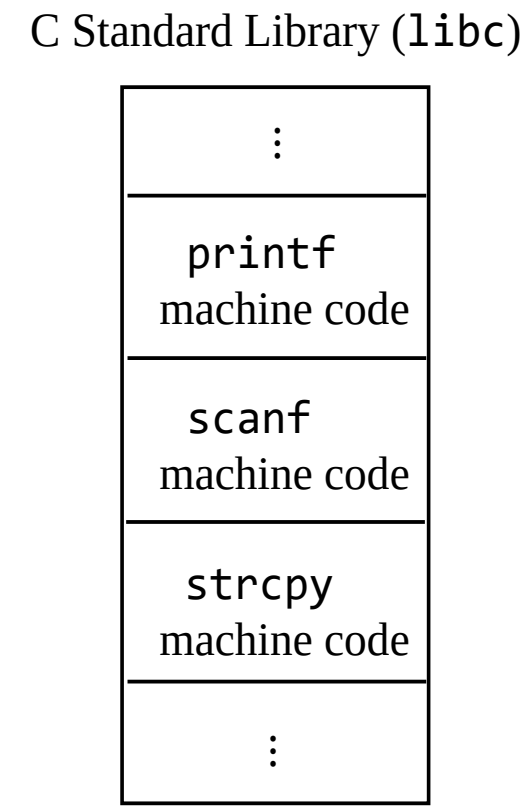
12 Calling C Functions from Assembler Code

Introduction



Converting to and from Binary

01000001 00110001 00110010 'A12'

00110001 00110010 '12'

00000000000000000000000000001100 12

Structure of a C program in Executable Form

```
1 // display.c
2 #include <stdio.h>
3 int x = 1;
4 int main(void)
5 {
6    printf("x = %d\n", x);
7    return 0;           // return 0 return code
8 }
```

Figure 12.2

```
gcc display.c -o display
```

`gcc` will then output the executable program to the `display` file. Fig. 12.3 shows the structure of the executable program in `display` (on Windows, use the `bcc32c` compiler in place of the `gcc` compiler).

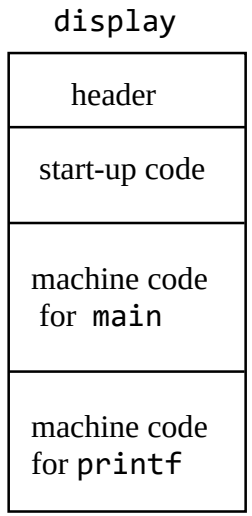
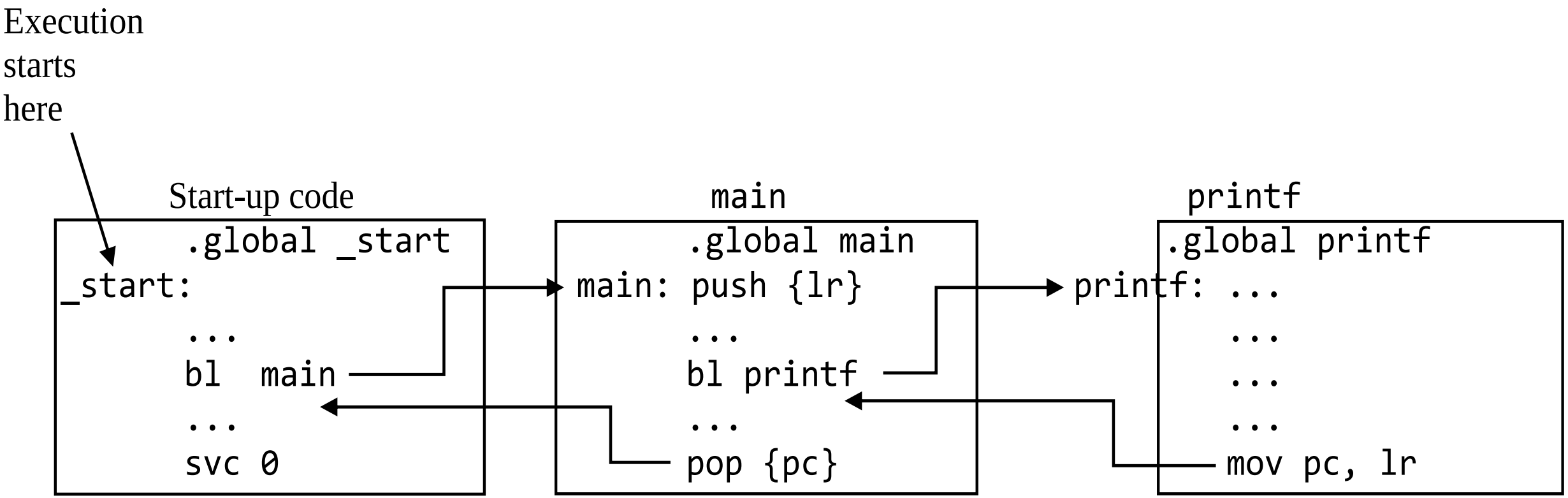


Figure 12.3



Showing assembler code but machine code really is in these modules.

Figure 12.4

Calling printf from an Assembly Language Program

```
 1                                  @ display.s
 2            .global main          @ printf assumed global
 3            .text                 @ start of read-only segment
 4 main:      push {lr}             @ save lr by pushing onto stack
 5
 6            ldr r0, =.fmt0        @ get address of string
 7            ldr r1, =x            @ get address of x
 8            ldr r1, [r1]          @ get value of x
 9            bl printf             @ call printf
10
11            mov r0, #0            @ 0 return code
12            pop {pc}              @ pop saved lr into pc
13            .data                 @ start of read/write segment
14 .fmt0:     .asciz "%d\n"         @ null-terminated ASCII string
15 x:         .word 27              @ value to be displayed
```

Figure 12.5

```
gcc display.s -o display
display     (or ./display)

rpi display.s



as display.s -o display.o          (works okay)
ld display.o -o display            (warning and error message)

warning: cannot find entry symbol _start

undefined reference to 'printf'
```

Calling scanf from an Assembly Language Program

```
1 // keyin.c
2 #include <stdio.h>
3 int x;
4 int main(void)
5 {
6     scanf("%d", &x);          // read int from keyboard into x
7     printf("x = %d\n", x);    // display value of x
8     return 0;
9 }
```

Figure 12.6

```
 1                              @ keyin.s
 2          .global main        @ printf scanf assumed global
 3          .text               @ start of read-only segment
 4 main:    push {lr}           @ save lr by pushing onto stack
 5
 6          ldr r0, =.fmt0      @ get address of string
 7          ldr r1, =x          @ get address of x
 8          bl scanf            @ call scanf
 9
10          ldr r0, =.fmt1      @ get address of string
11          ldr r1, =x          @ get address of x
12          ldr r1, [r1]        @ get value of x
13          bl printf           @ call printf
14
15          mov r0, #0          @ 0 return code
16          pop {pc}            @ return address popped into pc
17          .data               @ start of read/write segment
18 .fmt0:   .asciz "%d"         @ null-terminated ASCII string
19 .fmt1:   .asciz "x = %d\n"   @ null-terminated ASCII string
20 x:       .word 0
```

Figure 12.7