**6** Decisions, Loops, and Recursion

Decisions

if statement.

1. code to evaluate the true/false expression within parentheses in the C code and set the n, z, c and v flags accordingly
2. a conditional branch instruction that branches over the body of the if statement (to the label in item 4 below) if the expression within parentheses is false
3. the body of the if statement
4. a label

Simple if Statement

Can use brne here

ld r0, x ; if (x == 5)

cmp r0, 5 ; {

brnz @L0

Branch if x == 5 is false

*code for body* ; *body of if statement*

; }

@L0: *statement following if statement*

if-else Statement

ld r0, x ; if (x == 5)

cmp r0, 5 ; {

brnz @L0

Branch on false to else part

Unconditional branch

over else part

*code for if part* ; *if part*

br @L1 ; }

@L0: ; else

; {

*code for else part* ; *else part*

; }

@L1: *statement following* if-else *statement*

Loops

while statement

Branch on false

initial label

@L0: ld r0, x ; while (x == 5)

cmp r0, 5 ; {

brnz @L1

Can use

brne here

*code for body* ; *body of while statement*

br @L0 ; }

Unconditional branch back to initial label

@L1: *statement following while statement*

Loop That Displays Numbers Down To 1

1 ; ex0601.a while statement in a non-recursive function

2 startup: bl main

3 halt

4 ;==============================================================

5 nrf: push lr ; #include <stdio.h>

6 push fp ; void nrf(int x)

7 mov fp, sp ; {

8

9 @L0: ldr r0, fp, 2 ; while (x != 0)

10 cmp r0, 0 ; {

11 brz @L1

Can use bre here

Sets flags

12

13 ldr r0, fp, 2 ; printf("%d\n", x);

14 dout r0

15 nl

16

17 ldr r0, fp, 2 ; x = x - 1;

18 sub r0, r0, 1

19 str r0 fp, 2

20

21 br @L0 ; }

Return instructions even though no C return statement

22 @L1:

23 mov sp, fp ; }

24 pop fp

25 pop lr

26 ret

27 ;==============================================================

28 main: push lr ; int main()

29 push fp ; {

30 mov fp, sp

31

32 mov r0, 2 ; nrf(2);

33 push r0

34 bl nrf

35 add sp, sp, 1

36

37 mov r0, 0 ; return 0;

38 mov sp, fp

39 pop fp

40 pop lr

41 ret

42 ; }

Parameter on Stack Accessed by Called Function

sp (r6)

x decremented on each iteration of the loop

fp (r5) ~ ~

saved fp

saved lr

~~2~~ ~~1~~ 0 x

~ ~

Question:

Why not create dynamic local variables with .fill directives, like global and static local variables?

Recursion

Tail recursive

1 // ex0602.c Recursion example 1 (tail recursion)

2 #include <stdio.h>

3 void rf1(int x)

4 {

5 if (x != 0)

6 {

7 printf("%d\n", x);

8 rf1(x - 1); // recursive call

9 }

10 }

11 // =====================

12 int main()

13 {

14 rf1(2);

15 return 0;

16 }

main

a rf1(2);

return 0; k

if (x != 0) b if (x != 0) e if (x != 0) h  
 { { {

c printf("%d\n", x); f printf("%d\n", x); printf("%d\n", x);

d rf1(x - 1); g rf1(x - 1); rf1(x - 1);

} j } i }

first x second x third x

2 1 0

; ex0602.a Recursion example 1 (tail recursive)

0000 4814 startup: bl main

0001 f000 halt

;==============================================================

; #include <stdio.h>

0002 ae01 rf1: push lr ; void rf1(int x)

0003 aa01 push fp ; {

0004 1ba0 mov fp, sp

0005 f00d s ; debugging instruction that displays stack

0006 6142 ldr r0, fp, 2 ; if (x != 0)

0007 8020 cmp r0, 0 ; {

0008 0008 brz @L0

Can use bre here

0009 6142 ldr r0, fp, 2 ; printf("%d\n", x);

000a f027 dout r0

000b f001 nl

000c 6142 ldr r0, fp, 2 ; rf1(x - 1);

000d b021 sub r0, r0, 1

000e a001 push r0

000f 4ff2 bl rf1

0010 1da1 add sp, sp, 1

@L0: ; }

0011 1d60 mov sp, fp ; }

0012 aa02 pop fp

0013 ae02 pop lr

0014 c1c0 ret

;==============================================================

0015 ae01 main: push lr ; int main()

0016 aa01 push fp ; {

0017 1ba0 mov fp, sp

0018 d002 mov r0, 2 ; rf1(2);

0019 a001 push r0

001a 4fe7 bl rf1

001b 1da1 add sp, sp, 1

001c d000 mov r0, 0 ; return 0;

001d 1d60 mov sp, fp

001e aa02 pop fp

001f ae02 pop lr

0020 c1c0 ret; ; }

Stack:

fffb: fffe saved fp for main <--- fp

fffc: 001b address in main to return to

fffd: 0002 first x stack at point b in Fig. 6.1

fffe: 0000 saved fp for startup code

ffff: 0001 address in startup to return to

2

Stack:

fff8: fffb saved fp for rf1 (1st call) <--- fp

fff9: 0010 address in rf1 to return to

fffa: 0001 second x

fffb: fffe saved fp for main stack at point e in Fig. 6.1

fffc: 001b address in main to return to

fffd: 0002 first x

fffe: 0000 saved fp for startup code

ffff: 0001 address in startup to return to

1

Stack:

fff5: fff8 saved fp for rf1 (2nd call) <--- fp

fff6: 0010 address in rf1 to return to

fff7: 0000 third x

fff8: fffb saved fp for rf1 (1st call)

fff9: 0010 address in rf1 to return to

fffa: 0001 second x stack at point h in Fig. 6.1

fffb: fffe saved fp for main

fffc: 001b address in main to return to

fffd: 0002 first x

fffe: 0000 saved fp for startup code

ffff: 0001 address in startup to return to

ex0601.a (loop) ex0602.a (recursion)

Instructions executed 44 60

Program size 31 32

Maximum stack size 5 11

The difference between ex0601.a and ex0602.a becomes more dramatic if main passes 100 instead of 2:

ex0601.a (loop) ex0602.a (recursion)

Instructions executed 1024 1824

Program size 31 32

Maximum stack size 5 305

Not Tail Recursive

1 ; ex0603.a Recursion example 2 (not tail recursive)

2 startup: bl main

3 halt

4 ;==============================================================

5 ; #include <stdio.h>

6 rf2: push lr ; void rf2(int x)

7 push fp ; {

8 mov fp, sp

9

10 ldr r0, fp, 2 ; if (x == 0)

11 cmp r0, 0

12 brnz @L0

Can use brne here

13

14 lea r0, @m0 ; printf("bottom\n");

15 sout r0

16

17 br @L1 ; else

18 @L0: ; {

19

20 lea r0, @m1 ; printf("down\n");

21 sout r0

22

rf2 does this after the recursive call so rf2 is not tail recursive

23 ldr r0, fp, 2 ; rf2(x - 1);

24 sub r0, r0, 1

25 push r0

26 bl rf2

27 add sp, sp, 1

28

29 lea r0, @m2 ; printf("up\n");

30 sout r0

31

32 @L1: ; }

33

34 mov sp, fp ; }

35 pop fp

36 pop lr

37 ret

38 ;==============================================================

39 main: push lr ; int main()

40 push fp ; {

41 mov fp, sp

42

43 mov r0, 2 ; rf2(2);

44 push r0

45 bl rf2

46 add sp, sp, 1

47

48 mov r0, 0 ; return 0;

49 mov sp, fp

50 pop fp

51 pop lr

52 ret

53 ; }

54 @m0: .stringz "bottom\n"

55 @m1: .stringz "down\n"

56 @m2: .stringz "up\n"

down

down

bottom

up

up