**7** Pointers

Pointers in C

int p; // p has type int

int \*p; // p has type int pointer

p x

7

p x

7

3000

p = &x; // &x is the address of x

q = p;

p x

7

q

y = \*p; // assigns y the value that p points to

\*p = 10; // assigns 10 to the location that p points to

y = x; // directly accessing x

y = \*p; // accessing x via a pointer

p = p + 1; // cannot do this in Java

Pointers to Global Variables

x: .word 0

p: .word 0

p = &x;

is  
 lea r0, x ; get address of x

st r0, p ; store address of x in p

y = \*p

ld r0, p ; load r0 with address in p

ldr r0, r0, 0 ; load from address given by r0 +

st r0, y

\*p = 10;

mov r0, 10

ld r1, p ; load r1 with address in p   
 str r0, r1, 0 ; store 10 at address given by r1 + 0

Example with Pointers to Global Variables

1 ; ex0701.a Pointers to global variables

2 startup: bl main

3 halt

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

5 ; #include <stdio.h>

6 p: .word 0 ; int \*p, x = 7;

7 x: .word 7

8

9 main: push lr ; int main()

10 push fp ; {

11 mov fp, sp

12

13 lea r0, x ; p = &x;

14 st r0, p

15

16 ld r0, p ; printf("%d\n", \*p);

17 ldr r0, r0, 0

18 dout r0

19 nl

20

21 mov r0, 8 ; \*p = 8;

22 ld r1, p

23 str r0, r1, 0

24

25 ld r0, x ; printf("%d\n", x);

26 dout r0

27 nl

28

29 mov r0, 0 ; return 0;

30 mov sp, fp

31 pop fp

32 pop lr

33 ret

34 ; }

Pointers to Dynamic Local Variables

ldr r0, fp, -1 // loads r0 from local var at offset -1

If, instead, we want to load r0 with the *address* of the stack item at offset -1, we use

add r0, fp, -1 // loads r0 with addr of local var at offset -1

Dereferencing

ldr r0, fp, -1 ; get local pointer into r0

We then load r0 from the address that r0 points to:

ldr r0, r0, 0 ; load r0 from address given by r0 + 0

To store a value in a location that a local variable points to,

mov r0, 10 ; get the value to be stored

ldr r1, fp, -1 ; get the pointer into r1

str r0, r1, 0 ; store value in r0 at address given by r1 + 0

Example with Pointers to Local Variables

1 ; ex0702.a Pointers to local variables

2 startup: bl main

3 halt

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

5 ; #include <stdio.h>

6 ; int main()

7 main: push lr ; {

8 push fp

9 mov fp, sp

10

11 sub sp, sp, 1 ; int \*p, x = 7;

12 mov r0, 7

Get address of local var x

13 push r0

14

15 add r0, fp, -2 ; p = &x;

16 str r0, fp, -1

17

18 ldr r0, fp, -1 ; printf("%d\n", \*p);

19 ldr r0, r0, 0

Dereference p

20 dout r0

21 nl

22

23 mov r0, 8 ; \*p = 8;

24 ldr r1, fp, -1

25 str r0, r1, 0

Dereference p

26

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

28 dout r0

29 nl

30

31 mov r0, 0 ; return 0;

32 mov sp, fp

33 pop fp

34 pop lr

35 ret

36 ; }

Pointers to Functions

void f(int x, int y)

f(1, 2); // calls f function

p = f; // does not call f

p f function

f

f(1, 2);

p(1, 2);

p = g; // p now points to the g function

But assigning g to f is not legal because f is a constant pointer:

f = g; // illegal because f always points to the f function

Declaring p:

void (\*p)(int, int);

Incorrect!!!

void \*p(int, int); // parens high precedence that \*

p would then be a function (because the parentheses indicate p is a function) that returns

void \*

Example of Pointers to Functions

1 ; ex0703.a Pointers to functions

2 startup: bl main

3 halt

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

5 ; #include <stdio.h>

6 sum: .word 0 ; int sum;

7 p: .word 0 ; int (\*p)(int, int);

8 ;==============================================================

9 f: push lr ; int f(int x, int y)

10 push fp ; {

11 mov fp, sp

12

13 ldr r0, fp, 2 ; return x+y;

14 ldr r1, fp, 3

15 add r0, r0, r1

16 mov sp, fp

17 pop fp

18 pop lr

19 ret

20 ; }

21 ;==============================================================

22 main: push lr ; int main()

23 push fp ; {

24 mov fp, sp

25

26 mov r0, 2 ; sum = f(1, 2);

27 push r0

28 mov r0, 1

29 push r0

30 bl f

31 add sp, sp, 2

32 st r0, sum

33

34

35 ld r0, sum ; printf("%d\n", sum);

36 dout r0

37 nl

38

39 lea r0, f ; p = f;

40 st r0, p

41

42 mov r0, 2 ; sum = p(1, 2);

43 push r0

44 mov r0, 1

45 push r0

46 ld r0, p

47 blr r0

48 add sp, sp, 2

49 st r0, sum

50

51 ld r0, sum ; printf("%d\n", sum);

52 dout r0

53 nl

54

55 mov r0, 0 ; return 0;

56 mov sp, fp

57 pop fp

58 pop lr

59 ret

60 ; }