**9** Structs

Creating and Accessing Structs

struct Point

{

int x;

Semicolon required

int y;

};

struct Point a;

Point a; // illegal in C, but not in C++.

a

x

y

a.x = 1;

a.y = 2;

struct Point // create new type

{

int x;

int y;

} a; // also declare a

struct Point \*p;

p = &a;

We then get the following configuration:

p a

1 x

2 y

p->y = 3; // assign 3 to the y field of the struct that p points to

(\*p).y = 3;

1 ; ex0901.a Structs

2 startup: bl main

3 halt

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

5 ; #include <stdio.h>

6 ; struct Point

7 ; {

8 ; int x;

9 ; int y;

10 ; };

11

12 a: .zero 2 ; struct Point a;

13 p: .word 0 ; struct Point \*p;

14

15 main: push lr ; int main()

16 push fp ; {

17 mov fp, sp

18

19 mov r0, 1 ; a.x = 1;

20 st r0, a

21

22 mov r0, 2 ; a.y = 2;

23 st r0, a+1

24

25 lea r0, a ; p = &a;

26 st r0, p

27

28 ld r0, a+1 ; printf("%d\n", a.y);

29 dout r0

30 nl

31

32 ld r0, p ; printf("%d\n", p->y);

33 ldr r0, r0, 1

34 dout r0

35 nl

36

37 ld r0, p ; printf("%d\n", (\*p).y);

38 ldr r0, r0, 1

39 dout r0

40 nl

41

42 mov r0, 0 ; return 0;

43 mov sp, fp

44 pop fp

45 pop lr

46 ret

47 ; }

Dynamically Allocating Structs

1 // ex0902.c Dynamically allocating structs

2 #include <stdio.h>

3 #include <stdlib.h> // required my malloc

4 struct Point

5 {

6 int x;

7 int y;

8 };

9 struct Point \*p;

10 //==================================

11 int main()

12 {

13 p = (struct Point \*)malloc(sizeof(struct Point));

14 p->y = 5;

15 printf("%d\n", p->y);

16 return 0;

17 }

1 ; ex0902.a Dynamically allocating structs

2 startup: bl main

3 halt

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

5 ; #include <stdio.h>

6 ; #include <stdlib.h>

7 ; struct Point

8 ; {

9 ; int x;

10 ; int y;

11 ; };

12

13 p: .word 0 ; struct Point \*p;

14

15 main: push lr ; int main()

16 push fp ; {

17 mov fp, sp

18

19 mov r1, 2 ; p = (struct Point \*)malloc(sizeof(struct Point));

20 bl malloc

21 st r0, p

22

23 mov r0, 5 ; p -> y = 5;

24 ld r1, p

25 str r0, r1, 1

26

27 ld r0, p ; printf("%d\n", p -> y);

28 ldr r0, r0, 1

29 dout r0

30 nl

31

32 mov r0, 0 ; return 0;

33 mov sp, fp

34 pop fp

35 pop lr

36 ret

37 ; }

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

39 malloc: ld r0, @avail ; get address of next free block

40 add r1, r0, r1 ; r1 holds size of allocation

41 st r1, @avail ; update @avail

42 ret ; return address of allocated block

43 @avail: .word \*+1

Passing Structs

1 // ex0903.c Passing structs

2 #include <stdio.h>

3 struct Point

4 {

5 int x;

6 int y;

7 };

8 struct Point a;

9 //===============================

10 void f(struct Point s)

11 {

12 printf("%d %d\n", s.x, s.y);

13 }

14 //===============================

15 void g(struct Point \*p)

16 {

17 printf("%d %d\n", p->x, p->y);

18 }

19 //===============================

20 int main()

21 {

22 a.x = 1;

23 a.y = 2;

24 f(a); // pass by value

25 g(&a); // pass by address

26 }

fp ~ ~

saved fp

saved lr

1 x

s parameter

2 y

~ ~

In f, the fields of the s parameter are accessed with a single instruction. For example, the y field is loaded into r0 with

ldr r0, fp, 3

The call of g on line 25 passes a by address. The calling sequence for g pushes the address of a onto the stack thereby creating the parameter p in g. Here is a picture of the stack when g is executing:

fp ~ ~

saved fp

saved lr

addr of a p

~ ~

To access the fields of a via the parameter p in g requires two instructions: one to get the address of a and a second to access the desired field. For example, the y field is loaded into r0 with

ldr r1, fp, 2 ; get address of a

ldr r0, r1, 1 ; get y field

*Rule*: Use pass by address to pass a struct or any array.

1 ; ex0903.a Passing structs

2 startup: bl main

3 halt

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

5 ; #include <stdio.h>

6 ; struct Point

7 ; {

8 ; int x;

9 ; int y;

10 ; };

11 a: .word 0 ; struct Point a;

12 .word 0

13 ;==============================================================

14 f: push lr ; void f(struct Point s)

15 push fp ; {

16 mov fp, sp

17

18 ldr r0, fp, 2 ; printf("%d %d\n", s.x, s.y);

19 dout r0

20 mov r0, ' '

21 aout

22 ldr r0, fp, 3

23 dout r0

24 nl

25

26 mov sp, fp ; }

27 pop fp

28 pop lr

29 ret

30 ;==============================================================

31 g: push lr ; void g(struct Point \*p)

32 push fp ; {

33 mov fp, sp

34

35 ldr r1, fp, 2 ; printf("%d %d\n", p->x, p->y);

36 ldr r0, r1, 0

37 dout r0

38 mov r0, ' '

Only one instruction needed to access y

39 aout

40 ldr r0, r1, 1

41 dout r0

42 nl

43

44 mov sp, fp ; }

45 pop fp

46 pop lr

47 ret

48 ;==============================================================

49 main: push lr ; int main()

50 push fp ; {

51 mov fp, sp

52

53 mov r0 1 ; a.x = 1;

54 st r0, a

55

56 mov r0, 2 ; a.y = 2;

57 st r0, a+1

58

59 ld r0, a+1 ; f(a);

60 push r0

61 ld r0, a

62 push r0

63 bl f

64 add sp, sp, 2

65

66 lea r0, a ; g(&a);

67 push r0

68 bl g

69 add sp, sp, 1

70

71 mov r0, 0 ; return 0;

72 mov sp, fp

73 pop fp

74 pop lr

75 ret

76

77 ; }