**10** Arrays

Declaring Arrays

int a[3];

a index

0

1

2

a[2] = 30;

int a[3] = {10, 20, 30};

a index

10 0

20 1

30 2

int a[] = {10, 20, 30}; // 3 slots by default

sub sp, sp, 3 ; reserve three slots on the stack

The code to create the same array but initialized with 10, 20, and 30 is

mov r0, 30

push r0

mov r0, 20

push r0

mov r0, 10

push r0

Global Array

a .zero 3

This .zero directive both reserves three words on memory and initializes them to 0. The code to create a global array a with three slots with the initial values 10, 20, and 30 is

a .word 10

.word 20

.word 30

Indexing

y = a[2];

is

ld r0, a+2 ; indexing at assembly time

st r0, y

y = a[x]; // indexing at run time

The corresponding code is

lea r0, a ; get address of a

ld r1, x ; get x

add r0, r0, r1 ; get address of a[x]

ldr r0, r0, 0 ; get a[x]

st r0, y ; store in y

1 ; ex1001.a Accessing arrays

2 startup bl main

3 halt

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

5 ; #include <stdio.h>

6 ga .zero 10 ; int ga[10], x = 3;

7 x .word 3

8

9

10 main push lr ; int main()

11 push fp ; {

12 mov fp, sp

13

14 sub sp, sp, 10 ; int la[10];

15

16 mov r0, 10 ; ga[2] = 10;

17 st r0 ga+2

18

19 mov r0, 11 ; ga[x] = 11;

20 lea r1, ga

Offset of la[0] is -10. Thus, offset of la[2] = -10+2 = -8

21 ld r2, x

22 add r1, r1, r2

23 str r0, r1, 0

24

25 mov r0, 12 ; la[2] = 12;

26 str r0, fp, -8

27

28 mov r0, 13 ; la[x] = 13;

29 add r1, fp, -10

Get address of la[0]. Then add x to get the address of la[x]

30 ld r2, x

31 add r1, r1, r2

32 str r0, r1, 0

33

34 ld r0, ga+2 ; printf("%d\n", ga[2]);

35 dout r0

36 nl

37

38 ld r0, ga+3 ; printf("%d\n", ga[3]);

39 dout r0

40 nl

41

42 ldr r0, fp, -8 ; printf("%d\n", la[2]);

43 dout r0

44 nl

45

46 ldr r0, fp, -7 ; printf("%d\n", la[3]);

47 dout r0

48 nl

49

50 mov r0, 0 ; return 0;

51 mov sp, fp

52 pop fp

53 pop lr

54 ret

55 ; }

Name of an Array is a Pointer

int a[3];

a index

0

1

2

int \*p;

p = a; ; equivalent to p = &a[0];

We get

a index

0

1

2

p

a[2] = 10;

\*(a+2) = 10;

p[2] = 10;

\*(p+2) = 10;

Important point

Because a pointer to the first slot of an array and the name of the array have the same type and the same value, a pointer to the first slot of an *array can be used as if it is the name of the array*. That is, it can be used with square brackets enclosing an index. For example, if p is pointing to the first slot of the a array, then the statement

p[2] = 10;

is legal, and it has the same effect as

a[2] = 10;

Create an array dynamically

p = (int \*)malloc(100\*sizeof(int));

p[3] = 10;

which is equivalent to

\*(p+3) = 10;

Passing an Array in a Function Call

int a[3];

f(a);

void f(int \*p) // parameter should be int pointer  
 {  
 ...  
 }

p a index

0

1

2

void f(int \*p)  
 {  
 p[2] = 99; // use p as the name of an array

}

void f(int \*p)  
 {  
 \*(p+2) = 99; // use p as a pointer

}

void f(int p[])  
 {  
 p[2] = 99;

An alternative way of declaring p as an int pointer

}

void f(int p[3])  
 {  
 p[2] = 99;

Does not make sense to put 3 here

}

1 ; ex1002.a Passing arrays

2 startup bl main

3 halt

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

5 ; #include <stdio.h>

6 a .zero 2 ; int a[2];

7

8

9 f1 push lr ; void f1(int z[])

10 push fp ; {

11 mov fp, sp

12

13 ldr r0, fp, 2 ; printf("%d\n", z[1]);

14 ldr r0, r0, 1

15 dout r0

16 nl

17

18 mov sp, fp ; }

19 pop fp

20 pop lr

21 ret

22 ;==============================================================

23 f2 push lr ; void f2(int \*z)

24 push fp ; {

25 mov fp, sp

26

27 ldr r0, fp, 2 ; printf("%d\n", \*(z+1));

28 ldr r0, r0, 1

29 dout r0

30 nl

31

32 mov sp, fp ; }

33 pop fp

34 pop lr

35 ret

36 ;==============================================================

37 f3 push lr ; void f3(int z[])

38 push fp ; {

39 mov fp, sp

40

41 ldr r0, fp, 2 ; printf("%d\n", \*(z+1));

42 ldr r0, r0, 1

43 dout r0

44 nl

45

46 mov sp, fp ; }

47 pop fp

48 pop lr

49 ret

50 ;==============================================================

51 f4 push lr ; void f4(int \*z)

52 push fp ; {

53 mov fp, sp

54

55 ldr r0, fp, 2 ; printf("%d\n", z[1]);

56 ldr r0, r0, 1

57 dout r0

58 nl

59

60 mov sp, fp ; }

61 pop fp

62 pop lr

63 ret

64 ;==============================================================

65 main push lr ; int main()

66 push fp ; {

67 mov fp, sp

68

69 mov r0, 99 ; a[1] = 99;

70 st r0, a+1

71

72 lea r0, a ; f1(a);

73 push r0

74 bl f1

75 add sp, sp, 1

76

77 lea r0, a ; f2(a);

78 push r0

79 bl f2

80 add sp, sp, 1

81

82 lea r0, a ; f3(a);

83 push r0

84 bl f3

85 add sp, sp, 1

86

87 lea r0, a ; f4(a);

88 push r0

89 bl f4

90 add sp, sp, 1

91

92 mov r0, 0 ; return 0;

93 mov sp, fp

94 pop fp

95 pop lr

96 ret

97 ; }

Strings

char sa[3] = {'A', 'B', '\0'};

We get in sa the null-terminated string “AB”:

sa index

'A' 0

'B' 1

'\0' 2

sa index

0000000001000001 0

0000000001000010 1

ASCII code for 'A'extended to 16 bits

0000000000000000 2

char sa[] = "AB"; // initialized with 'A', 'B', '\0'

sa .string "AB" ; global

@s0\_sa .string "AB" ; static local

If sa is a dynamic local array,

mov r0, 0 ; push null character  
 push r0

mov r0, 'B'  
 push r0

mov r0, 'A'  
 push r0

String Assignment

p = "Hello"; // assign address of ‘H’

Because p is assigned a char pointer in this statement, p should be declared as a char pointer:

char \*p;

Suppose p is a global variable. Then the assembler code for the assignment statement above is

lea r0, @m0 ; get address of 1st char in string

st r0, p

where @m0 is a label on the string constant:

@m0 .string "Hello"

printf("%c\n", \*(p+1));

printf("%c\n", p[1]);

char b[10];

b = "Bye"; // illegal

strcpy(b, "Bye"); // passing strcpy two addresses

*Rule*: The name of an array without square brackets should not appear on the left side of an assignment statement.

printf("%s\n", b); // passing printf an address (of b)

printf("%s\n", "Bye"); // passing printf an address of ‘B’

q = p; // does not copy string

1 ; ex1003.a Strings

2 startup bl main

3 halt

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

5 ; include <stdio.h>

6 g .string "AX" ; char g[] = "AX";

7

8 p .word @m0 ; char \*p = "BX";

9 @m0 .string "BX"

10 ;==============================================================

11 mystrcpy push lr ; void mystrcpy(char \*p, char \*q)

12 push fp

13 mov fp, sp ; {

14 @L0 ; while (1)

15 ; {

16

17 ldr r0, fp, 3 ; \*p = \*q;

18 ldr r0, r0, 0

19 ldr r1, fp, 2

20 str r0, r1, 0

21

22 ldr r0, fp, 3 ; if (\*q == 0)

23 ldr r0, r0, 0

24 cmp r0, 0

25

26 brz @L1 ; break;

27

28 ldr r0, fp, 2 ; p++;

29 add r0, r0, 1

30 str r0, fp, 2

31

32 ldr r0, fp, 3 ; q++;

33 add r0, r0, 1

34 str r0, fp, 3

35

36 br @L0 ; }

37 @L1 ; }

38 mov sp, fp

39 pop fp

40 pop lr

41 ret

42 ;==============================================================

43 main push lr ; int main()

44 push fp ; {

45 mov fp, sp

46

47 mov r0, 0 ; char c[] = "CX";

48 push r0

49 mov r0, 'X'

50 push r0

51 mov r0, 'C'

52 push r0

53

54 lea r0, @m1 ; char \*q = "DX";

55 push r0

56

57 sub sp, sp, 1 ; char \*r;

58

59 lea r0, @m2 ; r = "EX";

60 str r0, fp, -5

61

62 lea r0, g ; printf("%s\n", g);

63 sout

64 nl

65

66 ld r0, p ; printf("%s\n", p);

67 sout

68 nl

69

70 add r0, fp, -3 ; printf("%s\n", c);

71 sout

72 nl

73

74 ldr r0, fp, -4 ; printf("%s\n", q);

75 sout

76 nl

77

78 ldr r0, fp, -5 ; printf("%s\n", r);

79 sout

80 nl

81

82 lea r0, @m3 ; mystrcpy(g, "FX");

83 push r0

84 lea r0, g

85 push r0

86 bl mystrcpy

87 add sp, sp, 2

88

89 lea r0, g ; printf("%s\n", g)

90 sout

91 nl

92

93 mov r0, 0 ; return 0;

94 mov sp, fp

95 pop fp

96 pop lr

97 ret

98

99 @m1 .string "DX" ; }

100 @m2 .string "EX"

101 @m3 .string "FX"