Numbe3rs

Embedded Systems Interfacing

1

Overview

- Integer Data Types
- Machine Code
- Floating-Point Data Types
- Math Library
- Performance

2

Integer Data Types

Match ALU

Type	Bits	Min	Max
char,signed char	8	-128	127
unsigned char	8	0	255
short, signed short	16	-32768	32767
unigned short	16	0	65535
int, signed int	16	-32768	32767
unsigned int	16	0	65535
long, signed long	32	-2^{31}	$2^{31}-1$
unsigned long	32	0	$2^{32}-1$
long long **, signed long long	64	-2^{63}	$2^{63}-1$
unsigned long long **	64	0	$2^{64}-1$

$2^{10} \approx 10^3$

10^9 = 1 Billion

10^{18} = 1 Quintillion or "1 Trillion" using long scale

** ANSI C89 Extension

3

Limits.h

```
/* char properties */
#define CHAR_BIT   8
#define SCHAR_MAX  0x7f
#define SCHAR_MIN  (-SCHAR_MAX -1)
#define UCHAR_MAX  0xff
#define CHAR_MIN   SCHAR_MIN
#define CHAR_MAX   SCHAR_MAX

/* int properties */
#define SHRT_MAX   0x7fff
#define SHRT_MIN   (-SHRT_MAX -1)
#define USHRT_MAX  0xffff
#define INT_MAX    SHRT_MAX
#define INT_MIN    SHRT_MIN
#define UINT_MAX   USHRT_MAX
```

4

Limits.h

```
/* long properties */
#define LONG_MAX   0x7fffffff
#define LONG_MIN   (-LONG_MAX -1)
#define ULONG_MAX  0xffffffffU

/* long long properties */
#define LLONG_MAX  0x7fffffffffffffffLL
#define LLONG_MIN  (-LLONG_MAX -1)
#define ULLONG_MAX 0xffffffffffffffffUL
```

5

Char Data Type

- 8-Bits or 1-Byte
- Unsigned
  - $0000\ 0000_2$ (Smallest)
  - $1111\ 1111_2$ (Largest)
- Signed
  - 0111 1111 (Most positive)
  - 1000 0000 (Most negative)
  - 1111 1111 (-1)
- Similar for short, int, long and long long

6

Floating-Point Data Type

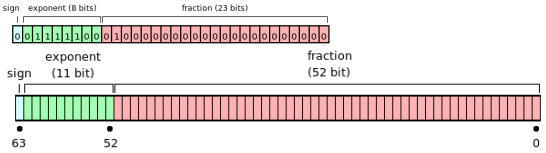
Type	Bits	E Min	E Max	N Min	N Max
float	32	-126	127	2^{-126}	2^{128}
double *	32	-126	127	2^{-126}	2^{128}
long double	64	-1022	1023	2^{-1022}	2^{1024}

E = exponent, N = Normalized (approximate)
* Double is equivalent to long double if –fno-short-double is used

$$value = (-1)^{sign} \bullet 2^{(exponient - 2^{e-1} - 1)} \bullet 0.1 fraction$$

7

IEEE 754 Floating-Point Spec.[1]



- The exponent is biased by $2^{e-1}-1$,
- *fraction"* is the significand minus the *most significant 1 bit*,
- and sign = 0 for positive (sign = 1 for negative) number.

[1] http://en.wikipedia.org/wiki/IEEE_754

8

Special Cases

Type	Exponent	Fraction
Zero	0	0
Normalized[1] Number	1 to 2^e-2	Any
Denormalized[2] Number	0	Non-zero
Infinities	2^e-1	0
Not A Number (NAN)	2^e-1	Non-zero

[1] Significant of form 0.1• • • [2] Soft Underflow

9

C-30 Single & Double Precision

Data Type	default	-fno-short-double
float	single	single
double	single	double
long double	double	double

10

8-Bit char Type Performance

```
        9:
         00282  B3C0C0      mov.b #0xc,0x0000
         00284  986F40      mov.b 0x0000,[0x001c+44]
        10:
         00286  B3C220      mov.b #0x22,0x0000
         00288  986F30      mov.b 0x0000,[0x001c+43]
        11:
         0028A  9068CE      mov.b [0x001c+44],0x0002
         0028C  90683E      mov.b [0x001c+43],0x0000
         0028E  40C000      add.b 0x0002,0x0000,0x0000
         00290  986F20      mov.b 0x0000,[0x001c+42]
```

6 Cycles

11

16-Bit int Type Performance

```
        13:
         00292  204D20      mov.w #0x4d2,0x0000
         00294  981740      mov.w 0x0000,[0x001c+40]
        14:
         00296  2162E0      mov.w #0x162e,0x0000
         00298  981730      mov.w 0x0000,[0x001c+38]
        15:
         0029A  9010CE      mov.w [0x001c+40],0x0002
         0029C  90103E      mov.w [0x001c+38],0x0000
         0029E  408000      add.w 0x0002,0x0000,0x0000
         002A0  981720      mov.w 0x0000,[0x001c+36]
```

4 Cycles

12

32-Bit long Type Performance

```
17:
 002A2  2614E0     mov.w #0x614e,0x0000
 002A4  200BC1     mov.w #0xbc,0x0002
 002A6  981700     mov.w 0x0000,[0x001c+32]
 002A8  981711     mov.w 0x0002,[0x001c+34]
18:
 002AA  268B10     mov.w #0x68b1,0x0000
 002AC  23ADE1     mov.w #0x3ade,0x0002
 002AE  980F60     mov.w 0x0000,[0x001c+28]
 002B0  980F71     mov.w 0x0002,[0x001c+30]
19:
 002B2  90110E     mov.w [0x001c+32],0x0004
 002B4  90119E     mov.w [0x001c+34],0x0006
 002B6  90086E     mov.w [0x001c+28],0x0000
 002B8  9008FE     mov.w [0x001c+30],0x0002
 002BA  400002     add.w 0x0000,0x0004,0x0000
 002BC  488083     addc.w 0x0002,0x0006,0x0002
 002BE  980F40     mov.w 0x0000,[0x001c+24]
 002C0  980F51     mov.w 0x0002,[0x001c+26]
```

8 Cycles

13

64-bit long long Type Performance

```
21:
 002C2  21C3D0     mov.w #0x1c3d,0x0000
 002C4  2DFDC1     mov.w #0xdfdc,0x0002
 002C6  200022     mov.w #0x2,0x0004
 002C8  200003     mov.w #0x0,0x0006
 002CA  980F00     mov.w 0x0000,[0x001c+16]
 002CC  980F11     mov.w 0x0002,[0x001c+18]
 002CE  980F22     mov.w 0x0004,[0x001c+20]
 002D0  980F33     mov.w 0x0006,[0x001c+22]
22:
 002D2  216EA0     mov.w #0x16ea,0x0000
 002D4  24CB01     mov.w #0x4cb0,0x0002
 002D6  200022     mov.w #0x2,0x0004
 002D8  200003     mov.w #0x0,0x0006
 002DA  980740     mov.w 0x0000,[0x001c+8]
 002DC  980751     mov.w 0x0002,[0x001c+10]
 002DE  980762     mov.w 0x0004,[0x001c+12]
 002E0  980773     mov.w 0x0006,[0x001c+14]
```

14

64-bit long long Type Performance

```
23:
 002E2  900A0E     mov.w [0x001c+16],0x0008
 002E4  900A9E     mov.w [0x001c+18],0x000a
 002E6  900B2E     mov.w [0x001c+20],0x000c
 002E8  900BBE     mov.w [0x001c+22],0x000e
 002EA  90004E     mov.w [0x001c+8],0x0000
 002EC  9000DE     mov.w [0x001c+10],0x0002
 002EE  90016E     mov.w [0x001c+12],0x0004
 002F0  9001FE     mov.w [0x001c+14],0x0006
 002F2  400004     add.w 0x0000,0x0008,0x0000
 002F4  488085     addc.w 0x0002,0x000a,0x0002
 002F6  490106     addc.w 0x0004,0x000c,0x0004
 002F8  498187     addc.w 0x0006,0x000e,0x0006
 002FA  BE9F00     mov.d 0x0000,[0x001c++]
 002FC  BE9702     mov.d 0x0004,[0x001c--]
```

21 Cycles

15

Performance

Type	+	-	*	/	%	Units
char	6	6	6		25	cycles
int	4	4	4		23	cycles
long	8	8	13		111	cycles
long long	21	21	110		146	cycles
float	130	126	121		80	cycles
long double	142	140	141		153	cycles

Data Dependent

math.h

16

Math.h

```
double ldexp(double X, int Y);      // X*2^Y
double log(double);                 // natural logarithm or ln
double log10(double);               // common logarithm
double pow(double X, double Y);     // X^Y
double sqrt(double);                // square root
double ceil(double X);              //ceiling of X
double fabs(double);                //absolute value
double floor(double X);             //floor of X
double fmod(double X, double Y);//X mod Y
```

2.000 ceiling

1.756

1.000 floor

-1.000 ceiling

-1.756

-2.000 floor

18

Math.h

```
double acos(double);                // arc cosine
double asin(double);                //arc cosine
double atan(double);                // arc tangent
double atan2(double, double);       //arc tangent of (Y/X)
double cos(double);                 //cosine
double sin(double);                 //sine
double tan(double);                 //tangent
double cosh(double);                //hyperbolic cosine
double sinh(double);                //hyperbolic sine
double tanh(double);                //hyperbolic tangent
double exp(double);                 //exponential function
```

Angle in radians

19

Math Functions

- Floating-point calculations done to double precision
- To calculate to single precision append 'f'
  - acos( ) -- double precision
  - acosf( ) -- single precision

20

Complex Numbers

- Append __complex__ (double underscore as prefix and suffix)
  - __complex__ short int x;
  - __complex__ float y;
- For imaginary part append 'i' or 'j'
  - y = -6.7f+2.5fj

21

Complex Numbers

- To extract real part use __real__
  - r=__real y;
- To extract imaginary part use __imag__
  - i=__imag y;

22

Homework

- Chapter 4
  - 2 Use simulator with stopwatch to measure number of cycles.
  - 3 (Sin, Cos and two argument Atan) Use simulator with stopwatch to measure number of cycles.

25