ESET 349 – Microcontroller Architecture

Number Systems

Dr. Muhammad Faeyz Karim

Kinds Of Data

Numbers

- Integers
  - Unsigned
  - Signed

- Reals
  - Fixed-Point
  - Floating-Point

- Binary-Coded Decimal

Numbers Are Different!

- Computers use binary (not decimal) numbers (0's and 1's).
  - Requires more digits to represent the same magnitude.
- Computers store and process numbers using a fixed number of digits (“fixed-precision”).
- Computers represent signed numbers using 2's complement instead of the more natural (for humans) “sign-plus-magnitude” representation.

Polynomial Evaluation

Whole Numbers (Radix = 10):

$$1234_{10} = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

With Fractional Part (Radix = 10):

$$36.72_{10} = 3 \times 10^1 + 6 \times 10^0 + 7 \times 10^{-1} + 2 \times 10^{-2}$$

General Case (Radix = R):

$$(S_1 S_0 . S_{-1} S_{-2})_R =$$

$$S_1 \times R^1 + S_0 \times R^0 + S_{-1} \times R^{-1} + S_{-2} \times R^{-2}$$

Converting Radix R *to Decimal*

$$36.72_8 = 3 \times 8^1 + 6 \times 8^0 + 7 \times 8^{-1} + 2 \times 8^{-2}$$

$$= 24 + 6 + 0.875 + 0.03125$$

$$= 30.90625_{10}$$

Binary to Decimal Conversion

Converting *to decimal*, so we can use polynomial evaluation:

10110101_2

$$= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 128 + 32 + 16 + 4 + 1$$

$$= 181_{10}$$

Decimal to Binary Conversion (Fractional Part: Repeated Multiplication)

- Multiply by target radix (2 in this case)
- Whole part of product becomes digit in the new representation (0 <= digit < R)
- Digits produced in left to right order.
- Fractional part of product is used as next multiplicand.
- Stop when the fractional part becomes zero (sometimes it won't).

Decimal to Binary Conversion (Fractional Part: Repeated Multiplication)

$.1 \times 2 \rightarrow 0.2$ (fractional part = .2, whole part = 0)

$.2 \times 2 \rightarrow 0.4$ (fractional part = .4, whole part = 0)

$.4 \times 2 \rightarrow 0.8$ (fractional part = .8, whole part = 0)

$.8 \times 2 \rightarrow 1.6$ (fractional part = .6, whole part = 1)

$.6 \times 2 \rightarrow 1.2$ (fractional part = .2, whole part = 1)

Result = $.00011001100110011_2$.....

(How much should we keep?)

Counting in Binary

Dec	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Note the pattern!

- LSB (bit 0) toggles on *every* count.
- Bit 1 toggles on every *other* count.
- Bit 2 toggles on every *fourth* count.
- Etc....

Memorize This!

Hex	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Hex	Binary
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Binary/Hex Conversions

- Hex digits are in one-to-one correspondence with groups of four binary digits:
- 0011 1010 0101 0110 . 1110 0010 1111 1000

  3 A 5 6 . E 2 F 8

- Conversion is a simple table lookup!
- Zero-fill on left and right ends to complete the groups!
- Works because $16 = 2^4$ (power relationship)

Rollover in Unsigned Binary

Consider an 8-bit byte used to represent an unsigned integer:

- Range: 00000000 ➔ 11111111 (0 ➔ 255_{10})
- Incrementing a value of 255 should yield 256, but this exceeds the range.
- Decrementing a value of 0 should yield –1, but this exceeds the range.
- <u>Exceeding the range</u> is known as *overflow*.

Two Interpretations

unsigned signed

167_{10} ← 10100111_2 → -89_{10}

- Signed vs. unsigned is a matter of interpretation; thus <u>a single bit pattern can represent two different values.</u>
- Allowing both interpretations is useful:

Some data (e.g., count, age) can never be negative, and having a greater range is useful.

Why Not Sign+Magnitude?

+3	0011
+2	0010
+1	0001
+0	0000
-0	1000
-1	1001
-2	1010
-3	1011

- Complicates addition :
  - To add, first check the signs. If they agree, then add the magnitudes and use the same sign; else subtract the *smaller* from the *larger* and use the sign of the larger.
  - How do you determine which is smaller/larger?
- Complicates comparators:
  - Two zeroes!

Why 2's Complement?

+3	0011
+2	0010
+1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100

1. Just as easy to determine sign as in sign+magnitude.
2. Almost as easy to change the sign of a number.
3. Addition can proceed w/out worrying about which operand is larger.
4. A single zero!
5. One hardware adder works for both signed and unsigned operands.

Changing the Sign

Sign+Magnitude:

+5 = 0101

Change 1 bit

-5 = 1101

2's Complement:

+5 = 0101

Invert

1010

+1 *Increment*

-5 = 1011

Easier Hand Method

Step 2: Copy the inverse of the remaining bits.

+4 = 0100

-4 = 1100

Step 1: Copy the bits from right to left, through and including the first 1.

Representation Width

Be Careful! You must be sure to pad the original value out to the full representation width before applying the algorithm!

Apply algorithm

Expand to 8-bits

Wrong: +25 = 11001 → 00111 → 00000111 = +7

Right: +25 = 11001 → 00011001 → 11100111 = -25

If positive: Add leading 0's
If negative: Add leading 1's

Apply algorithm

Unsigned Range

- Byte (8 bits) 0 to 255
- Halfword (16 bits) 0 to 65535
- Word (32 bits) 0 to 4,294,967,295
- Double Word 0 to $2^{64}-1$

Range of Signed Integers

- Half of the 2^n patterns will be used for positive values, and half for negative.

- Half is 2^{n-1}.

- Positive Range: 0 to $2^{n-1}-1$ (2^{n-1} patterns)
- Negative Range: -2^{n-1} to -1 (2^{n-1} patterns)

- 8-Bits (n = 8): -2^7 (-128) to $+2^7-1$ (+127)

2's Complement Integer Range

- Byte (8 bits) − 128 to 127
- Halfword (16 bits) − 32,768 to 32,767
- Word (32 bits) − 2,147,483,648 to 2,147,483,647
- Double Word − 2^{63} to $2^{63} - 1$

Floating Point (Real Nos)

- Floating point numbers are also know as real numbers (i.e. 3.141)
- Floating point extends the range of numbers that can be stored by a computer, and the accuracy is independent of the number magnitude.
- Generally floating point numbers store a sign (***S***), exponent (***E***) *and mantissa* (***F***) and are of predetermined number base (***B***). ***B*** is usually set to base 2.
- Floating point value = $(-1)^S \times F \times B^E$
- 32-bit Single precision format

content	Sign	Exponent	Mantissa
Bit	31	23 to 30	0 to 22

IEEE standard 754

- Used in almost all modern FPUs.
- IEEE754 has 1 sign **(S), 8 exponent (E) and 23 mantissa bits (F)** for single precision and for 1 (S), 11 (E) and 52 bits (F) for double precision.
- The exponent is stored in excess 127 for single precision (and excess 1023 for double precision). Will focus on single precision for now.
- The mantissa is slightly unusual; for "normal" numbers, the 23 bits mantissa is a fraction and it is assumed that 1 must be added to the mantissa, but the 1 is not stored. In other words, the mantissa is a fractional value ranging from 1 (mantissa bits 0) to slightly below 2 (all mantissa bits 1).
- Not "normal" numbers will be discussed later.
- As the exponent is a power of 2, the mantissa never needs to be greater than 2 (you would just add 1 to the exponent instead).

5 different number types

Zero	±	0	0
Infinity	±	111...111	0
NaN	±	111...111	non 0
Denormalised	±	0	any non-zero sequence
Normalised	±	E != 0	anything

Exponent

- IEEE 754's exponent is the actual exponent + 127

Actual Exponent	IEEE 754 Exponent
NaN or infinity!	255
…	…
3	130
2	129
1	128
0	127
-1	126
…	…
Subnormal number !	0

"Normal"Single-precision Floating-point Representation

```
       S  Exp+127            Mantissa

 2.000 0 10000000 (1).00000000000000000000000
 1.000 0 01111111 (1).00000000000000000000000
 0.750 0 01111110 (1).10000000000000000000000
 0.500 0 01111110 (1).00000000000000000000000
 0.000 0 00000000 (0).00000000000000000000000
-0.500 1 01111110 (1).00000000000000000000000
-0.750 1 01111110 (1).10000000000000000000000
-1.000 1 01111111 (1).00000000000000000000000
-2.000 1 10000000 (1).00000000000000000000000
```

"Normal" Floating point examples

No   Binary Representation

1   0011 1111 1000 0000 0000 0000 0000 0000

sign (1 bit)   exponent (8 bits)   mantissa (23 bits)

- This is positive, has exponent of 127−127= 0 (remember it's in excess 127 format), and mantissa of 1 so 1×2^0 = 1.0
- The most significant mantissa bit has the value of 0.5 or ½ , the next bit has the value of 0.25 or ¼, and so on.

"Normal" Floating point examples

No Binary Representation

1.5 0011 1111 1100 0000 0000 0000 0000 0000

sign (1 bit) exponent (8 bits) mantissa (23 bits)

- This is positive, has exponent of 127−127= 0 (remember it's in excess 127 format), and mantissa of 1.5 so 1.5×2^0 = 1.5

“Normal” Floating point examples

No Binary Representation

−40 1 100 0010 0 010 0000 0000 0000 0000 0000

sign (1 bit) exponent (8 bits) mantissa (23 bits)

- This is negative, has exponent of 132−127= 5 (remember it's in excess 127 format), and mantissa of 1.25
- so $(-1) \times 1.25 \times 2^5 = -40$

Alternative Method (Calculator required)

- Convert – 3.14159 into IEEE 754 format
- Step 1 Determine the sign bit

– 3.14159 is negative, hence sign bit, S, is 1

- Step 2 Determine the exponent
  - Use floor($\log_2$ 3.14159) = floor (1.65) = 1
  - Exponent = 1
  - Exponent in IEEE 754 = 127 + 1 = 128
  - 128 = 10000000B

Mantissa

- Step 3: Compute Mantissa
- Formula - Number x 2^{-E}
  - $3.14159 \times 2^{-1} = 1.570795$ (between 1 and 2)
  - Convert 0.570795 (subtract 1) into 23 bit binary fraction
  - $\mathrm{round}(0.570795 \times 2^{23}) = 4788176$
  - Convert 4788176 into hexadecimal first
  - 4788176 = 0x490FD0 =
  - 100 1001 0000 1111 1101 0000B (23 bits)

Final Step

Step 4 Put everything together

S	E	E	E	E	E	E	E	E	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
1	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0
C				0				4				9				0				F				D				0			

Counter check your answer using the following website
http://www.h-schmidt.net/FloatConverter/

Or download the IEEE 754 app from Google play store for Android machines

Smallest magnitude"Normal" number

No	Binary Representation
?	0000 0000 1000 0000 0000 0000 0000 0000

sign (1 bit) exponent (8 bits) mantissa (23 bits)

- Smallest exponent is 1 for "normal" floating point number, if exponent is 0, it is a "denormalised" or "subnormal"number.
- This is negative, has exponent of $1-127=-126$ (remember it's in excess 127 format), and mantissa of 1.00
- so $(1) \times 1.00 \times 2^{-126} = 2^{-126} \approx 1.1755 \times 10^{-38}$

“Denormalised” or “Subnormal”

- Denormalised floating point numbers are also known as Subnormal floating point numbers.
- They are smaller than “normal” numbers.
- The exponent field is all zeros.
- The mantissa field do not have a 1 added, it is just a straightforward binary fraction.
- The binary fraction has to be multiplied by the smallest magnitude “normal” number.

"Denormalised" Floating point

No	Binary Representation
?	0000 0000 0101 0000 0000 0000 0000 0000

sign (1 bit) exponent (8 bits) mantissa (23 bits)

- Since exponent is 0, it is a "denormalised" number.
- Mantissa is 0.5+ 0.125 = 0.625 (Note: no need to add 1)
- so (1) $\times$ 0.625 $\times$ $2^{-126} \approx 7.347 \times 10^{-39}$

Range of IEEE 754 "normal" numbers

- Single precision range is :

  $\pm(1.2\times10^{-38}$ to $3.4\times10^{38})$
- Double precision range is :

  $\pm(2.2\times10^{-308}$ to $1.8\times10^{308})$
- What are the range of the denormalised numbers?

Summary

- Data are stored in computer as bits.
- Number base conversion
- 2's complement representation for negative numbers.
- IEEE 754 floating point representation
- Character and string representations