CS 271 TUES LEC

NOTE: using
8 bit binary numbers
2's complement form of negatives

Binary Addition, Subtraction, Mult.

- Table

	0	1
0	0	1
1	1	10

- Bin 1 + Bin 1 = 0 with 1 carry
- Overflow occurs when a result is larger than available bits. This sets status register.
- Binary Subtraction

```
   0 110 10  10
0 0 1 0 1 1 0 1
0 0 0 1 1 1 1 0
0 0 0 0 1 1 1 1
```

Note: borrowing:

10 → *0 10

100 → 0 1 10

↳ or add 2's complement of second method ~~to the c~~ then add.

- Multiplication can happen as well. Just like decimal mult.
  ↳ or adding repeatedly.
- Division is/can be just repeated subtraction or long division.
  ↳ can also right-shift

$45 \div 8$: shift 3 as $8 = 2^{3}$

$00101101 \rightarrow \underbrace{00000101}_{\text{quotient}} \vdots \underbrace{101}_{R}$

= 5R5.

NOTE:
Add is just electrical add
Sub is complement, add
mul is repeated add
Div is repeated sub
} all with add and complement!

Endian-ness (Big, Little)

DONT CHANGE BITS!

- Big Endian: most popular: Bytes ordered left to right. MSB on "left" at lowest address
- Little endian: right to left, used by Intel, MSB on right.
- for -1234 binary: $\underbrace{FF}_{\text{Byte}}\ \underbrace{FF}_{"}\ \underbrace{FB}_{"}\ \underbrace{2E}_{"}$

↳ Big Endian:

FF	FF	FB	2E
0xn	0xn+1	0xn+2	0xn+3

Little endian

2E	FB	FF	FF
0xn	0xn+1	0xn+2	0xn+3

↳ "Remember Big Endian is what you'd expect". Big to small, Normal

- Network communications are always Big Endian form.

Floating Point Numbers

- float, real, having a decimal point.
- the "decimal point" (".") = radix point
-

... 2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4} ...
32	16	8	4	2	1	0.5	0.25	0.125	0.0625

- therefore, ~~binary~~ dec 4.5 → 100.1 binary
- some rational decimal numbers are irrational in binary...
- IEEE754 Standard

"Name"	Bits	Sign bits	Exponent	Mantissa
Single Precision	32	1	8	23
Double Precision	64	1	11	52
extended	80	1	15	64

size, bits

AKA Double extended

NEXT PAGE

Forming IEEE754 FP NUM's

Ex: 6.25 to single precision FP

1] $6.25_{DEC} \rightarrow 110.01_{BIN}$

2] move radix point to left until a single "1" appears to its left, then multiply by the corresponding power of 2:

$$1.\overset{2\;1}{1001} \cdot 2^{2}_{DEC}$$

3] Determine sign bit, in this case it is 0_{BIN}, "positive".

4] Biased Exponent is $2 + 127 = 129$ $= 10000001_{BIN}$.

5] Normalized Mantissa is formed by dropping the "1" to the left of the radix $(1.1001 \rightarrow 1001)$ and zero fill to mantissa size for specific precision:

$$\underbrace{1001\,0000000000000000000}_{LEN = 23}$$

6] concatenate:

0100 0000 1100 1000 0000
0000 0000 0000 BIN

= 0x40C80000

NOTE that if the number has a repeating binary ending / right of radix, it will fill the mantissa with that repetition...

UNDO FP NUM
- just undo all the steps above. easy!.
↳ make sure to not forget the sign!!

CS 271 THURS LEC

Parity

- either "even" or "odd" parity
- Parity is the number of one ("1") bits in a binary code
- Parity Bit Example:

  P.B. in Blue

  Even Parity : 111010110

  Odd Parity : 011010110

- Error Checking Feature:
  - for an even-parity system:

    101010101 ≡ OK

    100101010 ≡ ERROR

Hamming (Error Correcting) Codes

- for an n-bit code, $n = m + r$
  - m = data bits, r = parity bits
- There are ~~approximately~~ Exactly 2^n possible combinations of bits (different numbers), but only 2^m <u>correct</u> numbers.
- Parity is the sum of one check bit and its selected data bits.
- Number of parity bits depends on word size. $r = \log_2 m + 1$
  - guarentees Hamming Distance of 2
- Using this scheme, if one bit is changed (an "error"), we can find out what bit that is and correct it.
- There are 2^r invalid codes (error indicators)

Hamming Code Example for 8-bits

- $\log_2 8 = 3 + 1 = 4$ parity bits $= r$
- Note: ~~Ex~~ Using Left to Right representation which may not be <u>accurate</u>

Total size $m+r$

bit #	1	2	3	4	5	6	7	8	9	10	11	12
Type	P	P	D	P	D	D	D	P	D	D	D	D

Parity bits on powers of 2.

|| NOTE: Assuming Even Parity Machine

bit #	1	2	3	4	5	6	7	8	9	10	11	12
type	P	P	D	P	D	D	D	P	D	D	D	D
	1	0	0	0	0	1	0	1	1	1	0	1

Data is 45 DEC = 00101101 BIN

1] Parity bit #1 "controls" all odd spaces, so we set it to <u>1</u> so all odd bits have even parity. (1, 3, 5 .. 11).
   all have 1 in the 1's place!

2] Parity Bit #2 "controls" all spaces with a BIN 1 in the "2's" place. This includes: (3, 6, 7, 10, 11). To get even parity, this should be set to <u>0</u>

3] Parity Bit #4 controls all spaces with a 1 in the 4's place, so (6, 7, 12) this means to make even parity, we need to set this to <u>0</u>

4] Parity Bit #8 controls all with a 1 in the 8's place. (9, 10, 11, 12). Therefore this must be set to <u>1</u>.

Therefore, the 8-bit Decimal 00101101 becomes the 12-bit, even parity, hamming code 100001011101.

NEXT PAGE

Correcting / Checking Errors

Ex. <u>odd parity</u>, 12-bit:

bit #	1	2	3	4	5	6	7	8	9	10	11	12
type	P	P	D	P	D	D	D	P	D	D	D	D
Value	1	0	0	1	1	1	1	1	0	1	~~1~~0	1

Data Bits mean that currently, we have:
BIN 0111 0111 = DEC 119.

1] Parity Bit #1 Parity is ERROR

2] Parity Bit #2 Parity is ERROR

3] Parity Bit #4 Parity is <u>OK</u>

4] Parity Bit #8 Parity is ERROR

∴ Error must be at bit (1+2+8) = 11

∴ Error fixed by swapping bit 11. Val = <u>117</u>