01001110010000110101001101010101

4e435355

Number Systems

1,313,035,093

ECGR2181

Lecture Notes 1A

1.525×2^{29}

These are all different interpretations of the same bit string.

Unsigned Integers

Non-positional notation

- could represent a number (“5”) with a string of ones (“11111”)
- problems?

Weighted positional notation

- like decimal numbers: “329”
- “3” is worth 300, because of its position, while “9” is only worth 9

329

10^2 10^1 10^0

3x100 + 2x10 + 9x1 = 329

most significant → 101 ← *least significant*

2^2 2^1 2^0

1x4 + 0x2 + 1x1 = 5

Problems with non-positional: (1) large numbers require lots of bits, (2) arithmetic is not easy.

Positional: compact, simple arithmetic.

Unsigned Binary Arithmetic

Base-2 addition – just like base-10!

- add from right to left, propagating carry

carry

```
   10010        10010        1111
+   1001     +   1011     +     1
   11011        11101       10000
```

Unsigned Binary Arithmetic Practice

Base-2 addition – just like base-10!

– add from right to left, propagating carry

```
  10111          10111
+  1111        -  1111
-------        -------
```

Subtraction, multiplication, division,...

100110

Odometer numbers

Consider an odometer of a car at a location on a street:

| 0 | 0 | 1 | 3 | **Go 3 miles in reverse and it reads:** | 0 | 0 | 1 | 0 |

- **Same as “subtracting 3” or “adding -3”**
- **What happens when...**

| 0 | 0 | 0 | 0 | **Go 3 miles in reverse and it reads:** | 9 | 9 | 9 | 7 |

- **As far as the odometer is concerned, 9997 = -3**
- **<u>Note that fixed-width binary is very similar to odometer numbers in its limitations</u>**
- **Can the same representation be used?**
  - **0000_2-0001_2=“1111_2” or -1**
  - **0000_2-0011_2=“1101_2” or -3**
- **This is called “2’s complement”**

Two's Complement

Two's complement representation developed to make circuits easy for arithmetic.

– for each positive number (X), assign value to its negative (-X), such that X + (-X) = 0 with "normal" addition, ignoring carry out

```
  00101  (5)        01001  (9)
+ 11011  (-5)     +        (-9)
  -----            ------
  00000  (0)        00000  (0)
```


To add sign-magnitude numbers:

(1) if signs are the same, just add magnitudes and preserve sign (ignoring overflow for now)

(2) if signs are different, subtract smaller magnitude from larger and set sign according to larger

To add one's complement:

Add normally, then increment by carry-out.

Two's Complement Representation

If number is positive or zero,

– normal binary representation, zeroes in upper bit(s)

If number is negative,

– start with positive number
– flip every bit (i.e., take the one's complement)
– then add one

```
    00101  (5)            01001  (9)
    11010  (1's comp)            (1's comp)
  +     1               +     1
  -------               -------
    11011  (-5)                  (-9)
```


Common mistake: I say, "What is the two's complement representation of +5?"
Student takes the two's complement of +5 (00101) and tells me "11011".

Two's Complement Signed Integers

MS bit is sign bit – it has weight ***-2^{n-1}***.

Range of an n-bit number: -2^{n-1} through $2^{n-1} - 1$.

- The most negative number (-2^{n-1}) has no positive counterpart.

-2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

-2^3	2^2	2^1	2^0	
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Two's Complement Practice

Show the two's complement representation of the decimal number -6.

Converting Binary (2's C) to Decimal

1. If leading bit is one, take two's complement to get a positive number.
2. Add powers of 2 that have "1" in the corresponding bit positions.
3. If original number was negative, add a minus sign.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

$$X = 01101000_{two}$$
$$= 2^6+2^5+2^3 = 64+32+8$$
$$= 104_{ten}$$

Assuming 8-bit 2's complement numbers.

Memorize this table!

More Examples

$X = 00100111_{two}$
$= 2^5+2^2+2^1+2^0 = 32+4+2+1$
$= 39_{ten}$

$X = 11100110_{two}$
$-X = 00011010$
$= 2^4+2^3+2^1 = 16+8+2$
$= 26_{ten}$
$X = -26_{ten}$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

Assuming 8-bit 2's complement numbers.

Converting Binary (2's C) to Decimal Practice

Convert binary 00011111 to decimal:

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

Converting Decimal to Binary (2's C)

First Method: ***Division***

1. Divide by two – remainder is least significant bit.
2. Keep dividing by two until answer is zero, writing remainders from right to left.
3. Append a zero as the MS bit; if original number negative, take two's complement.

$X = \mathbf{104}_{ten}$

104/2	**=**	**52**	**r0**	*bit 0*
52/2	**=**	**26**	**r0**	*bit 1*
26/2	**=**	**13**	**r0**	*bit 2*
13/2	**=**	**6**	**r1**	*bit 3*
6/2	**=**	**3**	**r0**	*bit 4*
3/2	**=**	**1**	**r1**	*bit 5*
1/2	**=**	**0**	**r1**	*bit 6*

$X = \mathbf{01101000}_{two}$

Converting Decimal to Binary (2's C)

Second Method: ***Subtract Powers of Two***

1. Change to positive decimal number.
2. Subtract largest power of two less than or equal to number.
3. Put a one in the corresponding bit position.
4. Keep subtracting until result is zero.
5. Append a zero as MS bit; if original was negative, take two's complement.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

$X = 104_{ten}$

```
104 - 64 = 40   bit 6
 40 - 32 = 8    bit 5
  8 - 8  = 0    bit 3
```

$X = 01101000_{two}$

Converting Decimal to Binary Practice

Convert decimal 270 to binary using both methods described above:

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

More Converting Decimal to Binary Practice

Convert decimal 255 to binary using both methods described above:

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

Operations: Arithmetic

Recall: a <u>data type</u> includes *representation* and *operations*.

We now have a good representation for signed integers, so let's look at some ***arithmetic*** operations:

- Addition
- Subtraction
- Sign Extension

Addition

As we've discussed, 2's comp. addition is just binary addition.

- assume all integers have the same number of bits
- ignore carry out
- for now, assume that sum fits in n-bit 2's comp. representation

```
    01101000  (104)        11110110  (-10)
+   11110000  (-16)     +  ________  (-9)
    --------
    01011000  (88)                   (-19)
```

Assuming 8-bit 2's complement numbers.

Subtraction

Negate subtrahend (2nd no.) and add.

- assume all integers have the same number of bits
- ignore carry out
- for now, assume that difference fits in n-bit 2's comp. representation

	01101000	(104)		11110110	(-10)
–	00010000	(16)	–	________	(-9)

is just

	01101000	(104)		11110110	(-10)
+	11110000	(-16)	+	________	(9)
	01011000	(88)			(-1)

Assuming 8-bit 2's complement numbers.

Could also subtract, with borrows, from left to right.

This way, they only have to learn addition and they're prepared for LC-2, which doesn't have a subtract instruction.

Practice

Perform the Two's Complement operation to the following decimal numbers: - 56 - 14

11001000

11110010 = -14

Sign Extension

To add two numbers, we must represent them with the same number of bits.

If we just pad with zeroes on the left:

4-bit		8-bit	
0100	(4)	**00000100**	(still 4)
1100	(-4)	**00001100**	(12, not -4)

Instead, replicate the most significant bit -- the sign bit:

4-bit		8-bit	
0100	(4)	**00000100**	(still 4)
1100	(-4)	**11111100**	(still -4)

Overflow

If operands are too big,
then sum cannot be represented as an *n*-bit 2's comp number.

```
  01000  (8)          11000  (-8)
+ 01001  (9)        + 10111  (-9)
  -----               -----
  10001  (-15)        01111  (+15)
```

We have overflow if:

- signs of both operands are the same, and
- sign of sum is different.

Another test -- easy for hardware:

- carry into MS bit does not equal carry out

Hexadecimal Notation

It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead.

- fewer digits -- four bits per hex digit
- less error prone -- easy to corrupt long string of 1's and 0's

Binary	Hex	Decimal	Binary	Hex	Decimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

Memorize this table!!!!

Memorize this table!

Converting from Binary ← → Hexadecimal

Every four bits is a hex digit.

– start grouping from right-hand side

```
 011 1010 1000 1111 0100 1101 0111
  ⇓    ⇓    ⇓    ⇓    ⇓    ⇓    ⇓
  3    A    8    F    4    D    7
```

Every hex digit is represented by 4-bits.

– start with 1st hex digit from right-hand side

```
    1    F    8    C    5    3    E
    ↓    ↓    ↓    ↓    ↓    ↓    ↓
    1 1111 1000 1100 0101 0011 1100
```

This is not a new machine representation, just a convenient way to write the number.

Converting from Hexadecimal to Decimal

Every hex digit position has a base value

– multiply the value at the position by the base value

8 4 D 7

$$8 \times 16^3 + 4 \times 16^2 + 13 \times 16^1 + 7 \times 16^0 =$$
$$8 \times 4096 + 4 \times 256 + 13 \times 16 + 7 \times 1 =$$
$$32768 + 1024 + 208 + 7 = \boxed{34007}$$

Another method is to convert to binary first (easy) then convert to decimal:

– 84D7h = $1000\ 0100\ 1101\ 0111_2$ = 1+2+4+16+64+128+1024+32768 = **34007**

Practice Converting from Hex to Decimal

6 F 6 A