Week 6 Lecture 3

Decimal to Binary Conversion

Additions to Calculator

- Decimal to Binary Conversion
  - Requires integer input

Fixed point number

- Binary number
- Sign bit
  - Two's compliment
- Limited numbers
  - [-214748646, 214748647]

Binary Numbers

- Decimal: 11 = $1 * 10^1 + 1 * 10^0 = 1 * 10 + 1 * 1$
- Binary: 11 = $1 * 2^1 + 1 * 2^0 = 1 * 2 + 1 * 1$
  - Decimal 3
- Binary
  - Addition Table

+	0	1
0	0	1
1	1	10

  - Multiplication Table

*	0	1
0	0	0
1	0	1

Reasoning

- Each position represents a power of two:
  - 1, 2, 4, 8, 16, 32, 64, 218
  - Each digit represents whether that element belongs in the representation
- E.g
  - 1111 = 8 + 4 + 2 + 1 = 15
  - 1010 = 8 + 0 + 2 + 0 = 10
  - 1001 = 8 + 0 + 0 + 1 = 9

Reasoning 2

- Let the number we are trying to write be x
  - If it x even, we write 0 in the one's place, otherwise we write 1
  - If x is evenly divisible by 4 we write 0 in the two's place, otherwise we write 1
  - …
  - If x is evenly divisible by 2^n, we write 0 in the n's place, otherwise we write 1

Reasoning 3

- If x is evenly divisible by 2^n, we write 0 in the n's place, otherwise we write 1.
- If 2^n is larger than x, we stop.

Iterative Algorithm

- To write x in binary
  - Let i be 1
  - While ($2^i < x$)
    - If (x mod $2^{(i-1)}$) ≠ 0
      - Write 1 in the i's place
      - Set x to x - 2^i
    - Else
      - Write 0 in the i's place

Iterative Algorithm

- To write x in binary
  - Let i be the lowest power of 2 less than x
  - While (i > 0)
    - If (x div i) = 1
      - Write 1 in the i's place
      - Set x to x - i
    - Else (x div i) != 1
      - Write 0 in the i's place
    - Set x to x - i

Implement in C (Try 1)

```
int binary_iterative1(int x)
{
  for (int i = 1; x > 0 ; i *= 2) {
    if ((x % (i * 2)) != 0) {
      printf("1");
      x = x - i;
    } else {
      printf("0");
    }
  }
}
```

```
0 =
1 = 1
2 = 01
3 = 11
4 = 001
5 = 101
6 = 011
7 = 111
8 = 0001
9 = 1001
10 = 0101
11 = 1101
12 = 0011
13 = 1011
14 = 0111
15 = 1111
```

Digits are reversed!!!

Why are the digits reversed

- We print the lowest order digit first followed by the next lowest …
- We need to print the highest order digit first and then each of the lower order ones.
  - But we don't know what the higher order digits are until we calculate the lower order digits

Analysis

- $x = d_1 2^0 + d_2 2^1 + d_2 2^2 + \ldots + d_n 2^{n-1}$
- $x = \sum_{i=0}^{n} d_i \cdot 2^{i-1}$
- $x = d_1 2^0 + \sum_{i=2}^{n} d_i 2^{i-2}$
- $x = d_1 + (x \operatorname{div} 2)$
  - $\sum_{i=2}^{n} d_i 2^{i-2} = d_2 2^0 + d_3 2^1 \cdots d_n 2^{n-1}$
- $binary(x) = binary(x \operatorname{div} 2) \circ digit_1$
  - $digit_0 = x \, mod \, 2$

Recursive Algorithm

- Binary(x)
  - If x > 1 Binary(x div 2)
  - Print (x mod 2)

Implementation

```
int binary_recursive2(int x)
{
  if (x > 1) binary_recursive2(x/2);
  printf("%d", x % 2);
}
```

```
0 = 0
1 = 1
2 = 10
3 = 11
4 = 100
5 = 101
6 = 110
7 = 111
8 = 1000
9 = 1001
10 = 1010
11 = 1011
12 = 1100
13 = 1101
14 = 1110
15 = 1111
```

Why does it work

- Each time we divide by two we shift the binary digit left (E.g. binary: 1010 / 10 = 101, just as it would be in decimal)
- The recursive call with the parameter divided by two can then print out the next highest order digit if it is the last one.
- Or call again with a new parameter divided by two.

More detail

```
int binary_recursive3(int x)
{
  printf("binary_recursive2(%d) \
        %d mod 2 = %d\n",
        x, x, x % 2);
  if (x > 1) binary_recursive3(x/2);
  printf("%d", x % 2);
}
```

```
0 = binary_recursive2(0)         0 mod 2 = 0
0
1 = binary_recursive2(1)         1 mod 2 = 1
1
2 = binary_recursive2(2)         2 mod 2 = 0
binary_recursive2(1)     1 mod 2 = 1
10
3 = binary_recursive2(3)         3 mod 2 = 1
binary_recursive2(1)     1 mod 2 = 1
11
4 = binary_recursive2(4)         4 mod 2 = 0
binary_recursive2(2)     2 mod 2 = 0
binary_recursive2(1)     1 mod 2 = 1
100
5 = binary_recursive2(5)         5 mod 2 = 1
binary_recursive2(2)     2 mod 2 = 0
binary_recursive2(1)     1 mod 2 = 1
101
6 = binary_recursive2(6)         6 mod 2 = 0
binary_recursive2(3)     3 mod 2 = 1
binary_recursive2(1)     1 mod 2 = 1
110
7 = binary_recursive2(7)         7 mod 2 = 1
binary_recursive2(3)     3 mod 2 = 1
binary_recursive2(1)     1 mod 2 = 1
111
```

```
8 = binary_recursive2(8)         8 mod 2 = 0
binary_recursive2(4)     4 mod 2 = 0
binary_recursive2(2)     2 mod 2 = 0
binary_recursive2(1)     1 mod 2 = 1
1000
9 = binary_recursive2(9)         9 mod 2 = 1
binary_recursive2(4)     4 mod 2 = 0
binary_recursive2(2)     2 mod 2 = 0
binary_recursive2(1)     1 mod 2 = 1
1001
10 = binary_recursive2(10)       10 mod 2 = 0
binary_recursive2(5)     5 mod 2 = 1
binary_recursive2(2)     2 mod 2 = 0
binary_recursive2(1)     1 mod 2 = 1
1010
11 = binary_recursive2(11)       11 mod 2 = 1
binary_recursive2(5)     5 mod 2 = 1
binary_recursive2(2)     2 mod 2 = 0
binary_recursive2(1)     1 mod 2 = 1
1011
12 = binary_recursive2(12)       12 mod 2 = 0
binary_recursive2(6)     6 mod 2 = 0
binary_recursive2(3)     3 mod 2 = 1
binary_recursive2(1)     1 mod 2 = 1
1100
```

Example

- Binary(10)
  - Binary(10/2 == 5)
    - Binary(5/2 == 2)
      - Binary(2/2 == 1)
      - Print(1 % 2) 1
    - Print(2 % 2) 0
  - Print(5 % 2) 1
- Print(10 % 2) 0

Example 2

- Binary(11)
  - Binary(11/2 == 5)
    - Binary(5/2 == 2)
      - Binary(2/2 == 1)
      - Print(1 % 2) 1
    - Print(2 % 2) 0
  - Print(5 % 2) 1
- Print(11 % 2) 1