Number Representation

Negative numbers

Possibilities

- Unsigned only
  - Many issues simplified if no sign
  - Multi-precision arithmetic is straight forward
- Absolute value plus sign
  - Add a bit to the unsigned representation that tells whether the magnitude is positive or negative
  - Signs of both operands must be checked before adding or subtracting to determine if we add or subtract magnitudes
  - May have to recomplement answer if magnitude underflows
  - Changing sign of number easy - just change the sign bit
  - Both +0 and -0 exist in this representation
- 1's complement
  - Add a sign bit but represent negative numbers by complementing all bits including sign bit
  - Carries out of sign position must be added to least significant place
  - Changing sign of number easy - just complement all the bits
  - Both +0 and -0 exist in this representation
- 2's complement
  - Add a sign bit but just ignore carries out of end of register
  - Changing sign of number requires complementing number and adding one
  - Largest negative value is larger in magnitude than largest positive value
  - Addition and subtraction can be done serially - one pass from least significant end

Possible ways to handle overflow

- Mod range of numbers
- Make sign correct and overflow magnitude

Picture of Binary Representations

bits	unsigned	abs val + sign	1's comp	2's comp
0000	0	+0	+0	0
0001	1	+1	+1	1
0010	2	+2	+2	2
0011	3	+3	+3	3
0100	4	+4	+4	4
0101	5	+5	+5	5
0110	6	+6	+6	6
0111	7	+7	+7	7
1000	8	−0	−7	−8
1001	9	−1	−6	−7
1010	10	−2	−5	−6
1011	11	−3	−4	−5
1100	12	−4	−3	−4
1101	13	−5	−2	−3
1110	14	−6	−1	−2
1111	15	−7	−0	−1

Decimal Representations

Generally encode digits by a group of binary bits

Several possibilities

- Weight the bits 1248
  - Other weights 1242
- 2 out of 5
  - Used by telephone company in some crossbar offices
- Biquinary 1245
- Excess 3
  - 9's complement formed by complementing all bits

What about the sign? - Similar to binary

- Unsigned only
- Absolute value plus sign
- 9's complement
- 10's complement

Picture of Decimal Representations

digits	unsigned	abs val + sign	1's comp	2's comp
0000	0	+0	+0	+0
0001	1	+1	+1	+1
0002	2	+2	+2	+2
0009	9	+9	+9	+9
0010	10	+10	+10	+10
0011	11	+11	+11	+11
0998	998	+998	+998	+998
0999	999	+999	+999	+999
9000	9000	−0	−999	−1000
9001	9001	−1	−998	−999
9002	9002	−2	−997	−998
9009	9009	−9	−990	−991
9010	9010	−10	−989	−990
9011	9011	−11	−988	−989
9998	9998	−998	−1	−2
9999	9999	−999	−0	−1

Redundant Representations

Redundant digit representation

- Can eliminate carry propagation

Carry save adders

- Can represent numbers as a sum of numbers
- Delay long carry until end of computation

Implementation of the Rules of Arithmetic

Serial arithmetic vs parallel arithmetic

- No carry propagation problems
- Smaller hardware
- Less logic
- Recirculating registers for memory
- Slower

Addition

- Unsigned arithmetic
  - Start from right and add with carry
- 1's complement (or 9's complement)
  - Start from right and add with carry
  - Add carry out of most significant end to least significant end
    - and keep going until no carry
- 2's complement (or 10's complement)
  - Start from right and add with carry
  - Ignore any carry out of most significant end
- Absolute value plus sign
  - Start from right and add or subtract depending if signs are equal or different
  - If carry out of most significant end and
    - Adding - result overflowed
    - Subtracting - must recomplement answer and change sign
      - This step can add extra time for the operation when necessary

Subtraction

- Just like addition with a different table or
  - Absolute value plus sign
    - Change the sign of the subtrahend and add
  - 1's or 2's or 9's or 10's complement
    - do a 1's or 2's or 9's or 10's complement of the subtrahend and add

Notes on Addition and Subtraction

- Both absolute value plus sign and 1's complement have two zeros - plus zero and minus zero
- Must define result on overflow or underflow
  - Absolute value plus sign generally mod one more than largest value
  - 1's complement generally mod 2^n-1 for n-bit numbers (including sign)
  - 2's complement generally mod 2^n for n-bit numbers (including sign)
- To change the sign of a number
  - For 1's complement we subtract each digit from 1
  - For 9's complement we subtract each digit from 9
  - For 2's complement we form the 1's complement and add 1
  - For 10's complement we form the 9's complement and add 1
- Addition
  - For 2's or 10's complement we add normally and ignore the carry out of the most significant end
  - For 1's or 9's complement we add any carry out of the most significant end back into the least significant end
- Subtraction
  - For 1's complement we form the 1's complement of the subtrahend and do a 1's complement addition
  - For 2's complement we form the 2's complement of the subtrahend and do a 2's complement addition
  - For 9's complement we form the 9's complement of the subtrahend and do a 1's complement addition
  - For 10's complement we form the 10's complement of the subtrahend and do a 2's complement addition

Multiplication

- Can add or subtract multiplier just so final sum is correct
- We can have a multiplier that has mixed positive and negative digits
  - the number 19 [1, 9] or 10 + 9 could be represented as the digits [2,-1] or 20 - 1
- To multiply by a multiplier with mixed positive and negative digits we just add or subtract the corresponding partial products
  - Instead of multiplying by 3 we can add the multiplicand 3 times
  - We can do the subtractions by adding complements
    - We must be sure that the length of the complements is bigger than the size of the final product

Division

- Restoring division
  - Remainder is always positive
  - Keep subtracting the divisor until the quotient goes negative
  - Back up to the previous step, shift right one place and continue

  - Remainder is always positive (or zero)
- Nonrestoring division
  - Subtract the divisor until the divident changes sign (can add complement of divisor)
  - shift right one place
  - Add the divisor until the divident changes sign
  - shift right one place
  - go back to the first step (subtracting)

  - Remainder may be positive or negative
  - Quotient digits alternate positive and negative

Division example

```
       _
    __322 R 0
016/04512
    984
    029
    984
    013
    984
    9971
    0016
    9987
    0016
    00032
    99984
    00016
    99984
    00000
```

Square root

- Very similar to division
  - Note that the sum of odd numbers is a perfect square

```
 1 = 1
 4 = 1 + 3
 9 = 1 + 3 + 5
16 = 1 + 3 + 5 + 7
```

  - Odd numbers method
    - Keep subtracting the next odd number until remainder would go negative
    - When shifting (two places) for next digit
      - add one and append odd digit
  - Nonrestoring method

- Keep subtracting the next odd number until remainder goes negative
- When shifting (two places) for next digit
  - add one and subtract odd digits
- we now add instead of subtract until remainder goes positive
- When shifting (two places) for next digit
  - subtract one and add odd digits

Square Root Example (restoring)

```
  2 5 R 0
\/625
- 1
  5
- 3
  225
-  41
  184
-  43
  141
-  45
   96
-  47
   49
-  49
    0
```

Square Root Example (nonrestoring)

```
    _
  3 5 R 0
\/625
- 1
  5
- 3
  2
- 5
 9725
+  59
 9784
+  57
 9841
+  55
 9896
+  53
 9949
+  51
 0000
```

Big Square Root Example (nonrestoring)

```
      _
  7 8 5 R 0
\/390625
 - 1
  38
 - 3
  35
 - 5
  30
 - 7
  23
 - 9
  14
 -11
   3
 -13
 99006
 + 139
 99145
 + 137
 99282
 + 135
 99417
 + 133
 99550
 + 131
 99681
 + 129
 99810
 + 127
 99937
 + 125
    6225
 -  1241
    4984
 -  1243
    3741
 -  1245
    2496
 -  1247
    1249
 -  1249
```