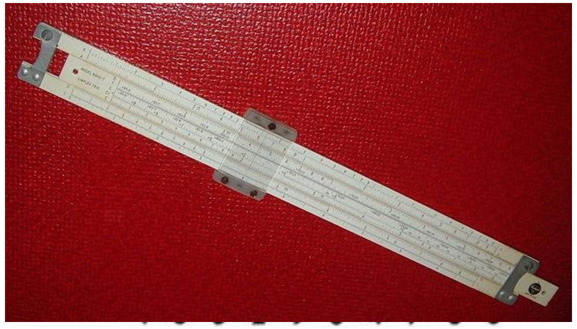
UNIVERSIDAD POLITÉCNICA
TAIWAN – PARAGUAY
臺灣-巴拉圭科技大學

Digital Logic Design

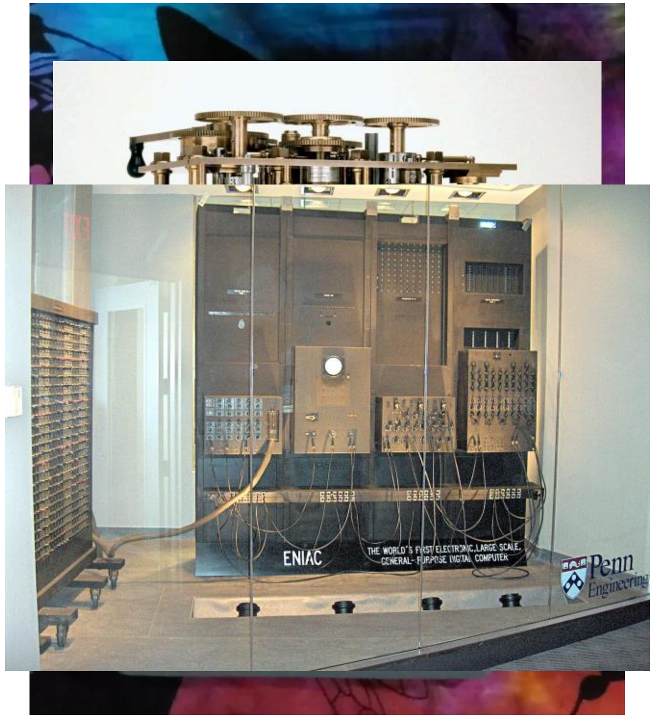
http://168.90.177.204/moodle-UPTP/moodle-app/

UNIVERSIDAD POLITÉCNICA
TAIWAN – PARAGUAY
臺灣-巴拉圭科技大學

History of Computing - Early Computers



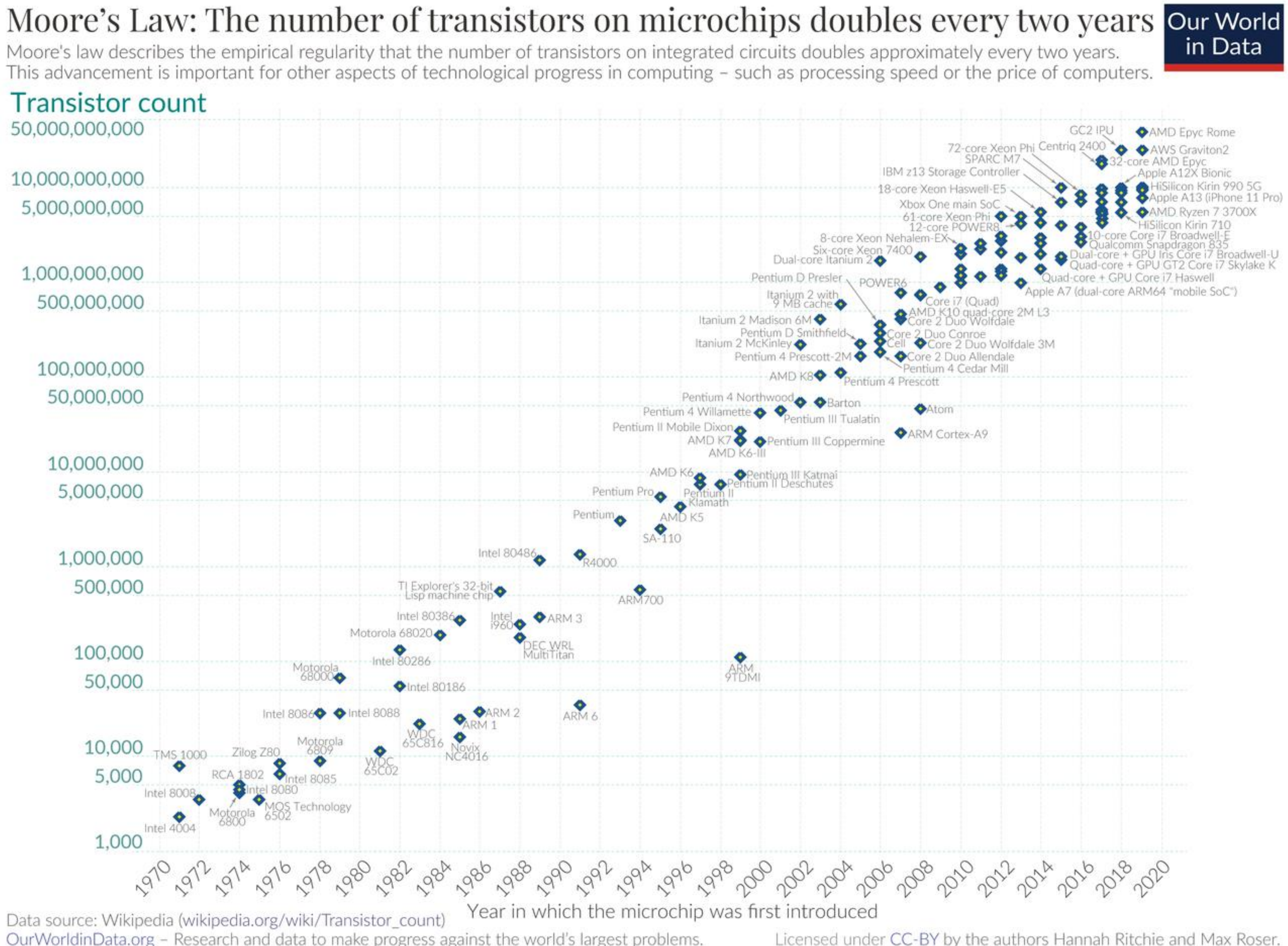
- Abacus (ancient orient, still in use)
- Slide rule (17C, John Napier)
- Adding machine with geared wheels (17C, B. Pascal)
- Difference Engine (19C, C. Babbage): First device using the principles of modern computer.
- ENIAC (1945, John Mauchly and J. Presper Eckert, Jr.)
  - Vacuum tube computer (18,000 electron tubes)
- Three important inventions 20C
  - Stored program concept (John von Neumann)
  - Transistor (J. Bardeen, W.H. Brattain, W. Shockley)
  - Magnetic core memory (J.W. Forrester and colleagues in MIT)



History of Computing - First Four Generations

- First generation: Vacuum tube computers (1940s - 1950s)
- Second generation (1950s): Transistors
- Third generation (1960s and 1970s): Integrated circuits
- Fourth generation (late 1970s through present): LSI and VLSI
  - Personal computers, computer networks, WWW, etc.
- Actual generation:
  - New user interfaces (voice activation, etc.)
  - New computational paradigm (parallel processing, neural network, etc.)
  - Parallel processing, artificial intelligence, optical processing, visual programming, gigabit networks, etc.

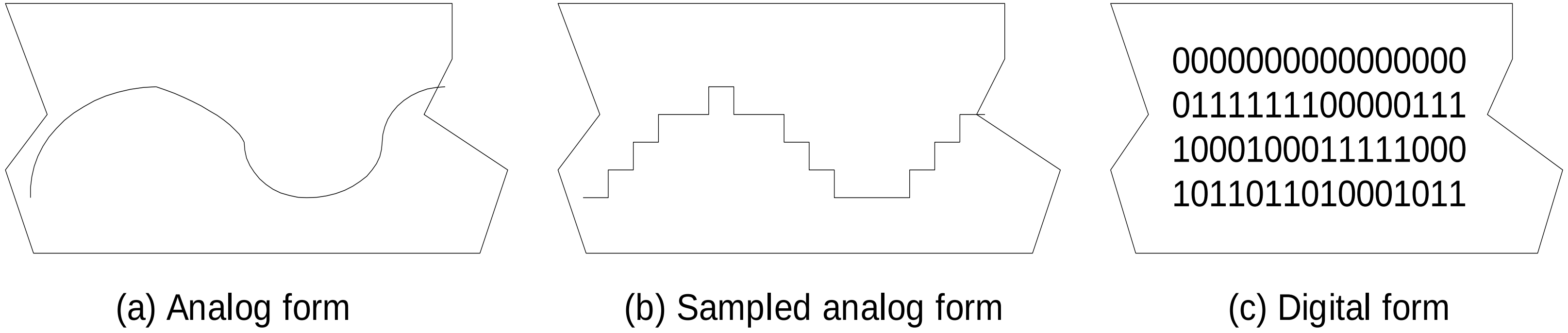
History of Computing - Evolution of processors



UNIVERSIDA
TAIWAN ·
臺灣－巴拉

Digital Systems - Analog vs. Digital

- Analog vs. Digital: Continuous vs. discrete.

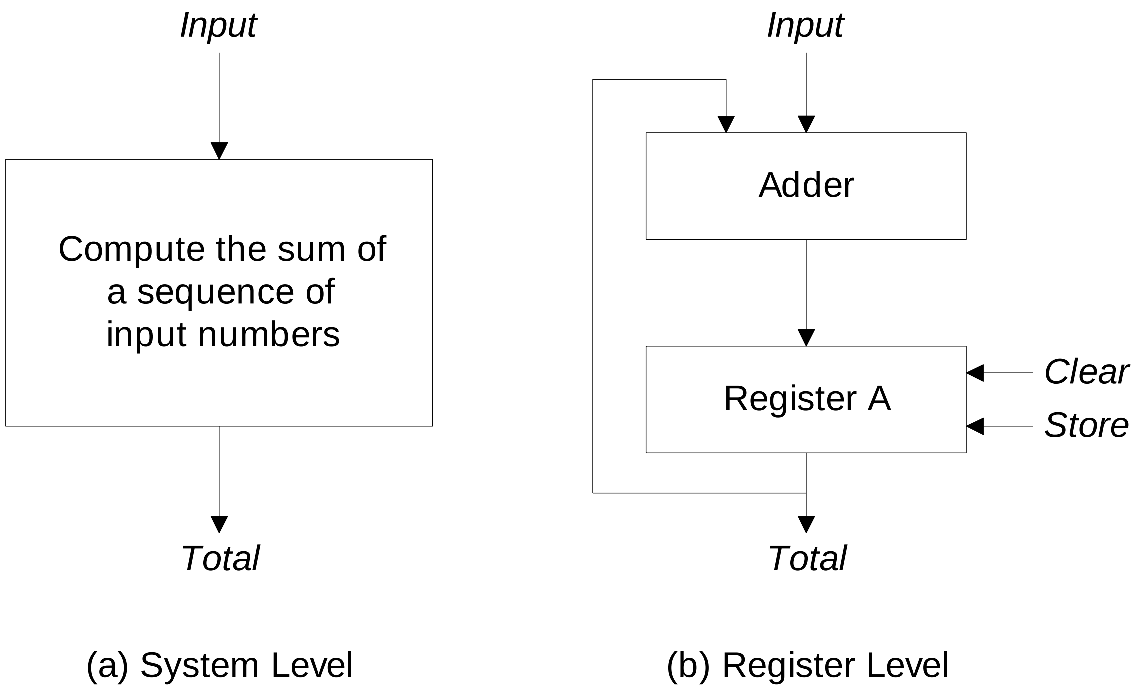


Magnetic tape contaning analog and digital forms of a signal.

- Digital computers replaced analog computers:
    - More flexible (easy to program), faster, more precise.
    - Storage devices are easier to implement.
    - Built-in error detection and correction.
    - Easier to minimize.

Digital Systems - Design Hierarchy (1)

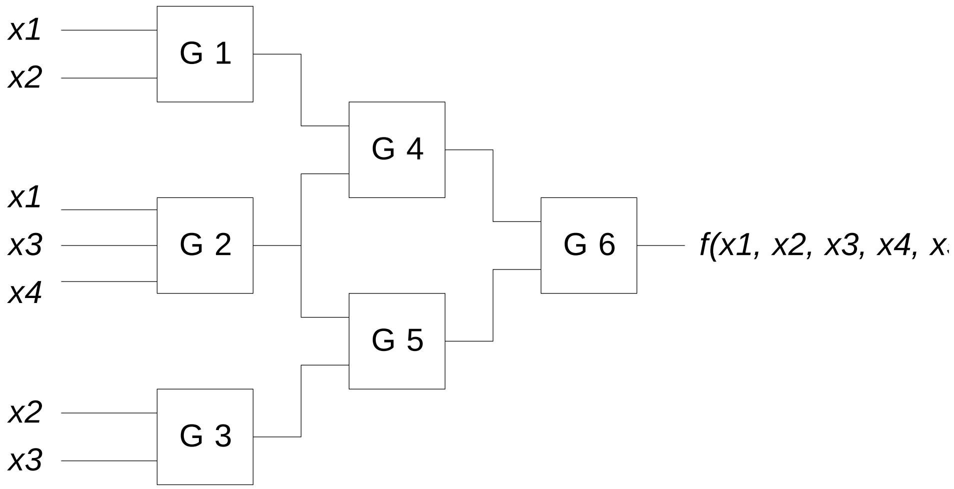
- System level: Black box specification.
- Register level: Collection of registers.



Models of a digital system that adds lists of numbers.

Digital Systems - Design Hierarchy (2)

- Gate level: Collection of logic gates.



A combinational logic circuit with six gates.

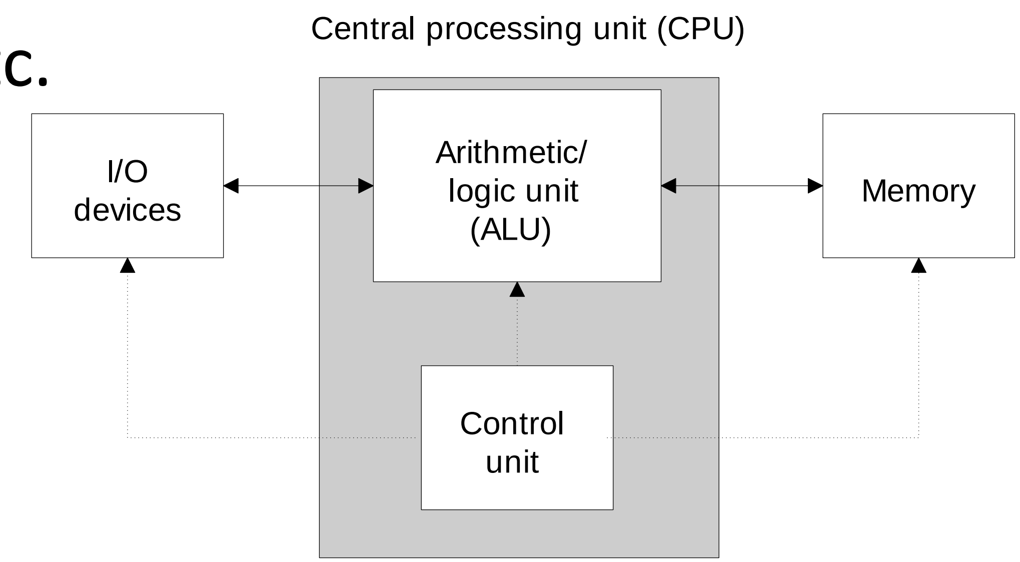
Digital Systems - Design Hierarchy (3)

- Transistor and physical design level: Each logic gate is implemented by a lower-level transistor circuit.
- Electronic Technologies:

Technology (Device Type)	Power Consumption	Speed	Packaging
RTL (Bipolar junction)	High	Low	Discrete
DTL (Bipolar junction)	High	Low	Discrete, SSI
TTL (Bipolar junction)	Medium	Medium	SSI, MSI
ECL (Bipolar junction)	High	High	SSI, MSI, LSI
pMOS (MOSFET)	Medium	Low	MSI, LSI
nMOS (MOSFET)	Medium	Medium	MSI, LSI, VLSI
CMOS (MOSFET)	Low	Medium	SSI, MSI, LSI, VLSI
GaAs (MOSFET)	High	High	SSI, MSI, LSI

Organization of a Digital Computer - Four Major Components

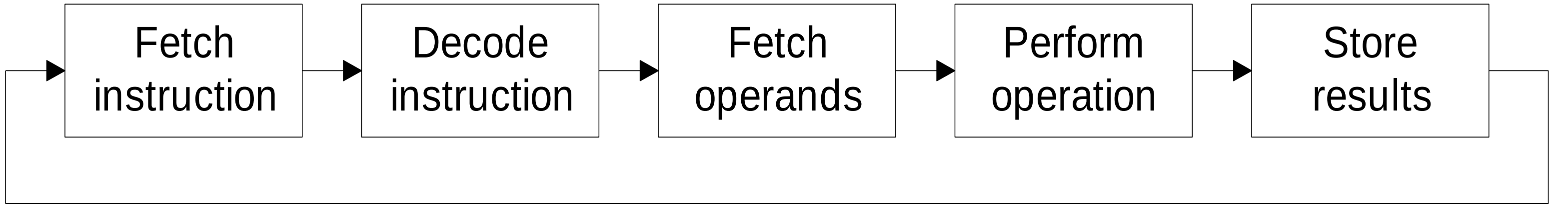
- Control unit: Follows the stored list of instructions and supervises the flow of information among other components.
- Arithmetic/logic unit (ALU): Performs various operations.
- Memory unit: Stores programs, input, output, and intermediate data.
- I/O devices: Printers, monitors, keyboard, etc.



High-level organization of a digital computer.

Organization of a Digital Computer - Instruction Cycle

- Fetch the next instruction into the control unit.
- Decode the instruction.
- Fetch the operands from memory or input devices.
- Perform the operation.
- Store the results in the memory (or send the results to an output device).



Instruction cycle of a stored program computer.

Organization of a Digital Computer - Computer Instructions

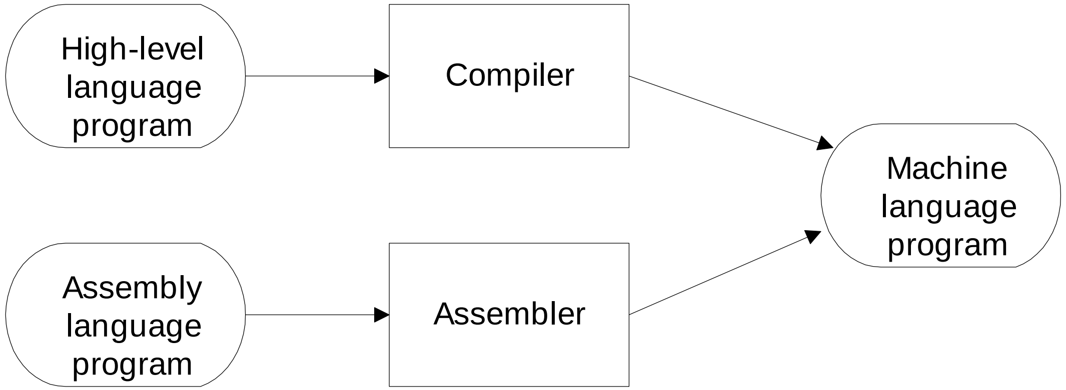
- Arithmetic instructions.
- Test and compare instructions.
- Branch or skip instructions.
- Input and output commands.
- Logical and shift operations.

Organization of a Digital Computer - Information Representation

- Numeric data: Binary number system.
- Numeric (Input/Output) codes: ASCII.
- Instruction codes: Operation code and memory addresses of operands and result.

Organization of a Digital Computer - Software

- Programming: The process of designing a list of instructions.
- Application programs: Word processor, spreadsheet, drawing programs, inventory management programs, accounting programs, etc.
- System programs: Operating systems, language translation programs, utility programs, performance monitoring programs, etc.



Translation of computer programs into machine language

Number systems and codes

Number Systems (1)

- ***Positional Notation***

  $N = (a_{n-1}a_{n-2} \ldots a_1a_0 \,.\, a_{-1}a_{-2} \ldots a_{-m})_r$ (1.1)

  where . = radix point (decimal point)

  r = radix or base

  n = number of integer digits to the left of the radix point

  m = number of fractional digits to the right of the radix point

  a_{n-1} = most significant digit (MSD)

  a_{-m} = least significant digit (LSD)

- ***Polynomial Notation*** (Series Representation)

  $N = a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \ldots + a_0 \times r^0 + a_{-1} \times r^{-1} \ldots + a_{-m} \times r^{-m}$

  $= \sum_{i=-m}^{n-1} a_i r^i$ (1.2)

- $N = (251.41)_{10} = 2 \times 10^2 + 5 \times 10^1 + 1 \times 10^0 + 4 \times 10^{-1} + 1 \times 10^{-2}$

Number Systems (2)

- ***Binary*** numbers
  - Digits = {0, 1}
  - $(11010.11)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$
    $= (26.75)_{10}$

- ***Octal*** numbers
  - Digits = {0, 1, 2, 3, 4, 5, 6, 7}
  - $(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$

- ***Hexadecimal*** numbers
  - Digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
  - $(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46{,}687)_{10}$

Number Systems (2)

- ***Special power of 2***

  1 K (kilo) $= 2^{10} = 1{,}024$,

  1M (mega) $= 2^{20} = 1{,}048{,}576$,

  1G (giga) $= 2^{30} = 1{,}073{,}741{,}824$

  1T (tera) $= 2^{40} = 1{,}099{,}511{,}627{,}776$

Number Systems (3)

- ***Important Number Systems***

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	1	F
16	10000	20	10

Arithmetic - *Binary Arithmetic*

- ***Addition***

```
  111011   Carries
   101011   Augend
+   11001   Addend
 -------
  1000100
```

- ***Subtraction***

```
 0   1  10   0  10       Borrows
  1   0   0   1   0   1   Minuend
-     1   1   0   1   1   Subtrahend
-----------------------
          1   0   1   0
```

Arithmetic - *Binary Arithmetic*

- ***Multiplication***

```
        1 1 0 1 0    Multiplicand
      x   1 0 1 0    Multiplier
      -----------
        0 0 0 0 0
      1 1 0 1 0
    0 0 0 0 0
  1 1 0 1 0
-----------------
1 0 0 0 0 0 1 0 0    Product
```

Division

```
                    1 1 0    Quotient
Divider  1 0 0 1 )1 1 1 1 0 1    Dividend
                  1 0 0 1
                  -------
                    1 1 0 0
                    1 0 0 1
                    -------
                      1 1 1    Remainder
```

Arithmetic - *Octal Arithmetic*

- ***Addition***

```
1 1 1        Carries
  5 4 7 1    Augend
+ 3 7 5 4    Addend
-----------------
1 1 4 4 5    Sum
```

- ***Subtraction***

```
   6 10   4 10   Borrows
   7   4   5   1   Minuend
-  5   6   4   3   Subtrahend
--------------------------------
   1   6   0   6   Difference
```

OCTAL ADDITION TABLE

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Arithmetic - *Octal Arithmetic*

- ***Multiplication***

```
     326   Multiplicand
 x    67   Multiplier
    2732   Partial products
   2404
   26772   Product
```

OCTAL MULTIPLICATION TABLE

X	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Division

```
            114     Quotient
Divider  63)7514    Dividend
            63
            114
             63
             364
             314
              50    Remainder
```

UNIVERSIDAD POLITÉCNICA TAIWAN – PARAGUAY
臺灣-巴拉圭科技大學

Arithmetic - *Hexadecimal Arithmetic*

- ***Addition***

```
  1 0 1 1     Carries
    5 B A 9   Augend
+   D 0 5 8   Addend
  1 2 C 0 1   Sum
```

- ***Subtraction***

```
    9 10  A 10   Borrows
    A  5  B  9   Minuend
+   5  8  0  D   Subtrahend
    4  D  A  C   Difference
```

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Arithmetic – *Hexadecimal Arithmetic*

- ***Multiplication***

```
   B9A5   Multiplicand
 x  D50   Multiplier
 ------
 3A0390   Partial products
96D61
-------
9A76490   Product
```

×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

- ***Division***

```
              79B    Quotient
Divider  B9 ) 57F6D  Dividend
              50F
              ----
               706
               681
              ----
               85D
               7F3
              ----
                6A   Remainder
```

UNIVERSIDAD POLITÉCNICA TAIWAN – PARAGUAY
臺灣–巴拉圭科技大學

Base Conversion (1)

- ***Series Substitution Method***
  - Expanded form of polynomial representation:
    $N = a_{n-1}r^{n-1} + \ldots + a_0r^0 + a_{-1}r^{-1} + \ldots + a_{-m}r^{-m}$ (1.3)
  - Conversation Procedure (base *A* to base *B*)
    - Represent the number in base *A* in the format of Eq. 1.3.
    - Evaluate the series using base *B* arithmetic.
  - ***Examples***:
    - $(11010)_2 \rightarrow (\ ?\ )_{10}$
      $N = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$
      $= (16)_{10} + (8)_{10} + 0 + (2)_{10} + 0$
      $= (26)_{10}$
    - $(627)_8 \rightarrow (\ ?\ )_{10}$
      $N = 6\times8^2 + 2\times8^1 + 7\times8^0$
      $= (384)_{10} + (16)_{10} + (7)_{10}$
      $= (407)_{10}$

Base Conversion (2)

- ***Radix Divide Method***
  - Used to convert the integer in base A to the equivalent base B integer.
  - Underlying theory:
    - $(N_I)_A = b_{n-1}B^{n-1} + \ldots + b_0B^0$ (1.4)

      Here, b_i's represents the digits of $(N_I)_B$ in base A.
    - $N_I / B = (b_{n-1}B^{n-1} + \ldots + b_1B^1 + b_0B^0) / B$

      $= (\text{Quotient } Q_1\text{: } b_{n-1}B^{n-2} + \ldots + b_1B^0) + (\text{Remainder } R_0\text{: } b_0)$
    - In general, $(b_i)_A$ is the remainder R_i when Q_i is divided by $(B)_A$.

  - ***Conversion Procedure***
    1. Divide $(N_I)_B$ by $(B)_A$, producing Q_1 *and* R_0. R_0 is the least significant digit, d_0, of the result.
    2. Compute d_i, for $i = 1 \ldots n - 1$, by dividing Q_i by $(B)_A$, producing Q_{i+1} and R_i, which represents d_i.
    3. Stop when $Q_{i+1} = 0$.

Base Conversion (3)

- ***Examples***
  - $(315)_{10} = (473)_8$

$(123)_{10} = (111\ 1011)_2$

  - $(315)_{10} = (13B)_{16}$

Base Conversion (4)

- ***Radix Multiply Method***
  - Used to convert fractions.
  - Underlying theory:
    - $(N_F)_A = b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-m}B^{-m}$ (1.5)
      Here, $(N_F)_A$ is a fraction in base A and b_i's are the digits of $(N_F)_B$ in base A.
    - $B \times N_F = B \times (b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-m}B^{-m})$
      $= (\text{Integer } I_{-1}\text{: } b_{-1}) + (\text{Fraction } F_{-2}\text{: } b_{-2}B^{-1} + \ldots + b_{-m}B^{-(m-1)})$
    - In general, $(b_i)_A$ is the integer part I_{-i}, of the product of $F_{-(i+1)} \times (B_A)$.
  - ***Conversion Procedure***
    1. Let $F_{-1} = (N_F)_A$.
    2. Compute digits $(b_{-i})_A$, for $i = 1 \ldots m$, by multiplying F_i by $(B)_A$, producing integer I_{-i}, which represents $(b_{-i})_A$, and fraction $F_{-(i+1)}$.
    3. Convert each digits $(b_{-i})_A$ to base B.

Base Conversion (5)

- ***Examples***
  - $(0.479)_{10} = (0.3651\ldots)_8$

    $$
    \begin{array}{ll}
    \text{MSD} & 3.832 \leftarrow 0.479 \times 8 \\
     & 6.656 \leftarrow 0.832 \times 8 \\
     & 5.248 \leftarrow 0.656 \times 8 \\
    \text{LSD} & 1.984 \leftarrow 0.248 \times 8 \\
     & \ldots
    \end{array}
    $$

  - $(0.479)_{10} = (0.0111\ldots)_2$

    $$
    \begin{array}{ll}
    \text{MSD} & 0.9580 \leftarrow 0.479 \times 2 \\
     & 1.9160 \leftarrow 0.9580 \times 2 \\
     & 1.8320 \leftarrow 0.9160 \times 2 \\
    \text{LSD} & 1.6640 \leftarrow 0.8320 \times 2 \\
     & \ldots
    \end{array}
    $$

Base Conversion (6)

- ***General Conversion Algorithm***
- ***Algorithm 1.1***

  To convert a number *N* from base *A* to base *B*, use

  (a) the series substitution method with base *B* arithmetic, or

  (b) the radix divide or multiply method with base *A* arithmetic.

- ***Algorithm 1.2***

  To convert a number *N* from base *A* to base *B*, use

  (a) the series substitution method with base 10 arithmetic to convert *N* from base *A* to base 10, and

  (b) the radix divide or multiply method with decimal arithmetic to convert *N* from base 10 to base *B*.

- Algorithm 1.2 is longer, but easier and less error prone.

Base Conversion (7)

- ***Example***

  $(18.6)_9 = (\ ?\ )_{11}$

  (a) Convert to base 10 using series substitution method:

  $$N_{10} = 1 \times 9^1 + 8 \times 9^0 + 6 \times 9^{-1}$$
  $$= 9 + 8 + 0.666\ldots$$
  $$= (17.666\ldots)_{10}$$

  (b) Convert from base 10 to base 11 using radix divide and multiply method:

  17 | 11
  (6) 1 | 11
  (1) 0

  $$7.326 \leftarrow 0.666 \times 11$$
  $$3.586 \leftarrow 0.326 \times 11$$
  $$6.446 \leftarrow 0.586 \times 11$$

  $N_{11} = (16.736\ \ldots)_{11}$

Base Conversion (8)

- When $B = A^k$
- ***Algorithm 1.3***

  (a) To convert a number *N* from base *A* to base *B* when $B = A^k$ and *k* is a positive integer, group the digits of *N* in groups of *k* digits in both directions from the radix point and then replace each group with the equivalent digit in base *B*

  (b) To convert a number *N* from base *B* to base *A* when $B = A^k$ and *k* is a positive integer, replace each base *B* digit in *N* with the equivalent *k* digits in base *A*.

- ***Examples***
  - $(001\ 010\ 111.\ 100)_2 = (127.4)_8$ (group bits by 3)
  - $(1011\ 0110\ 0101\ 1111)_2 = (B65F)_{16}$ (group bits by 4)

Signed Number Representation

- ***Signed Magnitude Method***
  - $N = \pm(a_{n-1} \ldots a_0.a_{-1} \ldots a_{-m})_r$ is represented as
    $N = (sa_{n-1} \ldots a_0.a_{-1} \ldots a_{-m})_{rsm}$, (1.6)
    where $s = 0$ if N is positive and $s = r - 1$ otherwise.
  - $N = -(15)_{10}$
  - In binary: $N = -(15)_{10} = -(1111)_2 = (1, 1111)_{2sm}$
  - In decimal: $N = -(15)_{10} = (9, 15)_{10sm}$

- ***Complementary Number Systems***
  - ***Radix complements*** (r's complements)
    $[N]_r = r^n - (N)_r$ (1.7)
    where n is the number of digits in $(N)_r$.
  - *Positive full scale*: $r^{n-1} - 1$
  - *Negative full scale*: $-r^{n-1}$
  - ***Diminished radix complements*** (r-1's complements)

    $[N]_{r-1} = r_n - (N)_r - 1$

Radix Complement Number Systems (1)

- Two's complement of $(N)_2 = (101001)_2$
  $[N]_2 = 2^6 - (101001)_2 = (1000000)_2 - (101001)_2 = (010111)_2$

- $(N)_2 + [N]_2 = (101001)_2 + (010111)_2 = (1000000)_2$
  If we discard the carry, $(N)_2 + [N]_2 = 0$.
  Hence, $[N]_2$ can be used to represent $-(N)_2$.
- $[\,[N]_2\,]_2 = [(010111)_2]_2 = (1000000)_2 - (010111)_2 = (101001)_2 = (N)_2$.

- Two's complement of $(N)_2 = (1010)_2$ for $n = 6$
  $[N]_2 = (1000000)_2 - (1010)_2 = (110110)_2$.

- Ten's complement of $(N)_{10} = (72092)_{10}$
  $[N]_{10} = (100000)_{10} - (72092)_{10} = (27908)_{10}$.

Radix Complement Number Systems (2)

- ***Algorithm 1.4* Find $[N]_r$ given $(N)_r$.**
  - Copy the digits of N, beginning with the LSD and proceeding toward the MSD until the first nonzero digit, a_i, has been reached
  - Replace a_i with $r - a_i$.
  - Replace each remaining digit a_j , of N by $(r - 1) - a_j$ until the MSD has been replaced.

- ***Example***: 10's complement of $(56700)_{10}$ is $(43300)_{10}$
- ***Example***: 2's complement of $(10100)_2$ is $(01100)_2$.
- ***Example***: 2's complement of $N = (10110)_2$ for $n = 8$.
  - Put three zeros in the MSB position and apply algorithm 1.4
  - $N = 00010110$
  - $[N]_2 = (11101010)_2$

- The same rule applies to the case when N contains a radix point.

Radix Complement Number Systems (3)

- ***Algorithm 1.5* Find $[N]_r$ given $(N)_r$.**
  - First replace each digit, a_k , of $(N)_r$ by $(r - 1) - a_k$ and then add 1 to the resultant.
- For binary numbers (r = 2), complement each digit and add 1 to the result.

- ***Example***: Find 2's complement of $N = (01100101)_2$.

  N = 01100101
  10011010 Complement the bits
  +1 Add 1
  $[N]_2 = (10011011)_{10}$

- ***Example***: Find 10's complement of $N = (40960)_{10}$

  N = 40960
  59039 Complement the bits
  +1 Add 1
  $[N]_2 = (59040)_{10}$

Radix Complement Number Systems (4)

- ***Two's complement number***:
  - Positive number :
    - $N = +(a_{n-2}, \ldots, a_0)_2 = (0, a_{n-2}, \ldots, a_0)_{2cns}$,
      where $0 \le N \le 2^{n-1} - 1$.
  - Negative number:
    - $N = (a_{n-1}, a_{n-2}, \ldots, a_0)_2$
    - $-N = [a_{n-1}, a_{n-2}, \ldots, a_0]_2$ (two's complement of N),
      where $-1 \ge N \ge -2^{n-1}$.

  - ***Example***: Two's complement number system representation of $\pm (N)_2$

    when $(N)_2 = (1011001)_2$ for $n = 8$:
    - $+(N)_2 = (0, 1011001)_{2cns}$
    - $-(N)_2 = [+(N)_2]_2 = [0, 1011001]_2 = (1, 0100111)_{2cns}$

Signed Decimal	Sign Magnitude Binary	Two's Complement System	One's Complement System
+15	0,1111	0,1111	0,1111
+14	0,1110	0,1110	0,1110
+13	0,1101	0,1101	0,1101
+12	0,1100	0,1100	0,1100
+11	0,1011	0,1011	0,1011
+10	0,1010	0,1010	0,1010
+9	0,1001	0,1001	0,1001
+8	0,1000	0,1000	0,1000
+7	0,0111	0,0111	0,0111
+6	0,0110	0,0110	0,0110
+5	0,0101	0,0101	0,0101
+4	0,0100	0,0100	0,0100
+3	0,0011	0,0011	0,0011
+2	0,0010	0,0010	0,0010
+1	0,0001	0,0001	0,0001
0	0,0000	0,0000	0,0000
	(1,0000)		(1,1111)
-1	1,0001	1,1111	1,1110
-2	1,0010	1,1110	1,1101
-3	1,0011	1,1101	1,1100
-4	1,0100	1,1100	1,1011
-5	1,0101	1,1011	1,1010
-6	1,0110	1,1010	1,1001
-7	1,0111	1,1001	1,1000
-8	1,1000	1,1000	1,0111
-9	1,1001	1,0111	1,0110
-10	1,1010	1,0110	1,0101
-11	1,1011	1,0101	1,0100
-12	1,1100	1,0100	1,0011
-13	1,1101	1,0011	1,0010
-14	1,1110	1,0010	1,0001
-15	1,1111	1,0001	1,0000
-16	—	1,0000	—

Radix Complement Number Systems (5)

- ***Example***: Two's complement number system representation of $-(18)_{10}$, n = 8:
  - $+(18)_{10} = (0, 0010010)_{2cns}$
  - $-(18)_{10} = [0, 0010010]_2 = (1, 1101110)_{2cns}$

- ***Example***: Decimal representation of $N = (1, 1101000)_{2cns}$
  - $N = (1, 1101000)_{2cns} = -[1, 1101000]_2 = -(0, 0011000)_{2cns} = -(24)_2$.

Radix Complement Arithmetic (1)

- Radix complement number systems are used to convert subtraction to addition, which reduces hardware requirements (only adders are needed).
- *A* - *B* = *A* + (-*B*) (add *r*'s complement of *B* to *A*)

- Range of numbers in two's complement number system:

$-2^{n-1} \leq N \leq 2^{n-1} - 1$, where *n* is the number of bits.

- $2^{n-1} - 1 = (0, 11 \ldots 1)_{2cns}$ and $-2^{n-1} = (1, 00 \ldots 0)_{2cns}$
- If the result of an operation falls outside the range, an ***overflow condition*** is said to occur and the result is not valid.

- Consider three cases:
  - *A* = *B* + *C*,
  - *A* = *B* - *C*,
  - *A* = - *B* - *C*,

    (where $B \geq 0$ and $C \geq 0$.)

UNIVERSIDAD POLITÉCNICA TAIWAN – PARAGUAY
臺灣–巴拉圭科技大學

Radix Complement Arithmetic (2)

- ***Case 1***: $A = B + C$
  - $(A)_2 = (B)_2 + (C)_2$
  - If $A > 2^{n-1} - 1$ (***overflow***), it is detected by the n^{th} bit, which is set to 1.

  - ***Example***: $(7)_{10} + (4)_{10} = ?$ using 5-bit two's complement arithmetic.
    - $+ (7)_{10} = +(0111)_2 = (0, 0111)_{2cns}$
    - $+ (4)_{10} = +(0100)_2 = (0, 0100)_{2cns}$
    - $(0, 0111)_{2cns} + (0, 0100)_{2cns} = (0, 1011)_{2cns} = +(1011)_2 = +(11)_{10}$
    - No overflow.
  - ***Example***: $(9)_{10} + (8)_{10} = ?$
    - $+ (9)_{10} = +(1001)_2 = (0, 1001)_{2cns}$
    - $+ (8)_{10} = +(1000)_2 = (0, 1000)_{2cns}$
    - $(0, 1001)_{2cns} + (0, 1000)_{2cns} = (\boldsymbol{1}, 0001)_{2cns}$ (***overflow***)

Radix Complement Arithmetic (3)

- ***Case 2***: $A = B - C$
  - $A = (B)_2 + (-(C)_2) = (B)_2 + [C]_2 = (B)_2 + 2^n - (C)_2 = 2^n + (B - C)_2$
  - If $B \geq C$, then $A \geq 2^n$ and the carry is discarded.

    So, $(A)_2 = (B)_2 + [C]|_{\text{carry discarded}}$
  - If $B < C$, then $A = 2^n - (C - B)_2 = [C - B]_2$ or $A = -(C - B)_2$ (no carry in this case).

    No overflow for Case 2.

  - ***Example***: $(14)_{10} - (9)_{10} = ?$
    - Perform $(14)_{10} + (-(9)_{10})$

$(14)_{10} = +(1110)_2 = (0, 1110)_{2cns}$

$-(9)_{10} = -(1001)_2 = (1, 0111)_{2cns}$

$(14)_{10} - (9)_{10} = (0, 1110)_{2cns} + (1, 0111)_{2cns} = (0, 0101)_{2cns} + carry$

$= +(0101)_2 = +(5)_{10}$

Radix Complement Arithmetic (4)

- ***Example***: $(9)_{10}$ - $(14)_{10}$ = ?
  - Perform $(9)_{10} + (-(14)_{10})$

    $(9)_{10} = +(1001)_2 = (0, 1001)_{2cns}$

    $-(14)_{10} = -(1110)_2 = (1, 0010)_{2cns}$

    $(9)_{10} - (14)_{10} = (0, 1001)_{2cns} + (1, 0010)_{2cns} = (1, 1011)_{2cns}$

    $= -(0101)_2 = -(5)_{10}$

- ***Example***: $(0, 0100)_{2cns} - (1, 0110)_{2cns}$ = ?
  - Perform $(0, 0100)_{2cns} + (-(1, 0110)_{2cns})$

    $-(1, 0110)_{2cns}$ = two's complement of $(1,0110)_{2cns}$

    $= (0, 1010)_{2cns}$

    $(0, 0100)_{2cns} - (1, 0110)_{2cns} = (0, 0100)_{2cns} + (0, 1010)_{2cns}$

    $= (0, 1110)_{2cns} = +(1110)_2 = +(14)_{10}$

    $+(4)_{10} - (-(10)_{10}) = +(14)_{10}$

Radix Complement Arithmetic (5)

- ***Case 3***: $A = -B - C$
  - $A = [B]_2 + [C]_2 = 2^n - (B)_2 + 2^n - (C)_2 = 2^n + 2^n - (B + C)_2 = 2^n + [B + C]_2$
  - The carry bit (2^n) is discarded.
  - An overflow can occur, in which case the sign bit is 0.

  - ***Example***: $-(7)_{10} - (8)_{10} = ?$
    - Perform $(-(7)_{10}) + (-(8)_{10})$

      $-(7)_{10} = -(0111)_2 = (1, 1001)_{2cns}$, $-(8)_{10} = -(1000)_2 = (1, 1000)_{2cns}$

      $-(7)_{10} - (8)_{10} = (1, 1001)_{2cns} + (1, 1000)_{2cns} = (1, 0001)_{2cns} + carry$

      $= -(1111)_2 = -(15)_{10}$
  - ***Example***: $-(12)_{10} - (5)_{10} = ?$
    - Perform $(-(12)_{10}) + (-(5)_{10})$

      $-(12)_{10} = -(1100)_2 = (1, 0100)_{2cns}$, $-(5)_{10} = -(0101)_2 = (1, 1011)_{2cns}$

      $-(7)_{10} - (8)_{10} = (1, 0100)_{2cns} + (1, 1011)_{2cns} = (0, 1111)_{2cns} + carry$

      Overflow, because the sign bit is 0.

UNIVERSIDAD POLITÉCNICA TAIWAN – PARAGUAY 臺灣–巴拉圭科技大學

Radix Complement Arithmetic (6)

- ***Example***: $A = (25)_{10}$ and $B = -(46)_{10}$
  - $A = +(25)_{10} = (0, 0011001)_{2cns}$, $-A = (1, 1100111)_{2cns}$
  - $B = -(46)_{10} = -(0, 0101110)_2 = (1, 1010010)_{2cns}$, $-B = (0, 0101110)_{2cns}$

  - $A + B = (0, 0011001)_{2cns} + (1, 1010010)_{2cns} = (1, 1101011)_{2cns} = -(21)_{10}$
  - $A - B = A + (-B) = (0, 0011001)_{2cns} + (0, 0101110)_{2cns}$
    $= (0, 1000111)_{2cns} = +(71)_{10}$
  - $B - A = B + (-A) = (1, 1010010)_{2cns} + (1, 1100111)_{2cns}$
    $= (1, 0111001)_{2cns} + carry = -(0, 1000111)_{2cns} = -(71)_{10}$
  - $-A - B = (-A) + (-B) = (1, 1100111)_{2cns} + (0, 0101110)_{2cns}$
    $= (0, 0010101)_{2cns} + carry = +(21)_{10}$
  - Note: Carry bit is discarded.

Radix Complement Arithmetic (7)

- Summary

Case	Carry	Sign Bit	Condition	Overflow ?
B + C	0 0	0 1	$B + C \leq 2^{n-1} - 1$ $B + C > 2^{n-1} - 1$	No Yes
B - C	1 0	0 1	$B \leq C$ $B > C$	No No
-B - C	1 1	1 0	$-(B + C) \geq -2^{n-1}$ $-(B + C) < -2^{n-1}$	No Yes

- When numbers are represented using two's complement number system:
  - Addition: Add two numbers.
  - Subtraction: Add two's complement of the subtrahend to the minuend.
  - Carry bit is discarded, and overflow is detected as shown above.

  - Radix complement arithmetic can be used for any radix.

Diminished Radix Complement Number systems (1)

- ***Diminished radix complement*** $[N]_{r-1}$ of a number $(N)_r$ is:

  $[N]_{r-1} = r^n - (N)_r - 1$ (1.10)

- ***One's complement*** (r = 2):

  $[N]_{2-1} = 2^n - (N)_2 - 1$ (1.11)

- ***Example***: One's complement of $(01100101)_2$

  $[N]_{2-1} = 2^8 - (01100101)_2 - 1$

  $= (100000000)_2 - (01100101)_2 - (00000001)_2$

  $= (10011011)_2 - (00000001)_2$

  $= (10011010)_2$

Diminished Radix Complement Number systems (2)

- ***Example***: Nine's complement of (40960)

  $[N]_{2\text{-}1} = 10^5 - (40960)_{10} - 1$
  $= (100000)_{10} - (40960)_{10} - (00001)_{10}$
  $= (59040)_{10} - (00001)_{10}$
  $= (59039)_{10}$

- ***Algorithm 1.6* Find $[N]_{r-1}$ given $(N)_r$.**

  Replace each digit a_i of $(N)_r$ by $r - 1 - a$. Note that when $r = 2$, this simplifies to complementing each individual bit of $(N)_r$.

- Radix complement and diminished radix complement of a number (N):

  $[N]_r = [N]_{r-1} + 1$ (1.12)

Diminished Radix Complement Arithmetic (1)

- Operands are represented using diminished radix complement number system.
- The carry, if any, is added to the result (***end-around carry***).

- ***Example***: Add $+(1001)_2$ and $-(0100)_2$.
  One's complement of +(1001) = 01001
  One's complement of -(0100) = 11011
  01001 + 11011 = 100100 (carry)
  Add the carry to the result: correct result is 00101.

- ***Example***: Add $+(1001)_2$ and $-(1111)_2$.
  One's complement of +(1001) = 01001
  One's complement of -(1111) = 10000
  01001 + 10000 = 11001 (no carry, so this is the correct result).

Diminished Radix Complement Arithmetic (2)

- ***Example***: Add $-(1001)_2$ and $-(0011)_2$.
  One's complement of the operands are: 10110 and 11100
  10110 + 11100 = 110010 (carry)
  Correct result is 10010 + 1 = 10011.

- ***Example***: Add $+(75)_{10}$ and $-(21)_{10}$.
  Nine's complements of the operands are: 075 and 978
  075 + 978 = 1053 (carry)
  Correct result is 053 + 1 = 054

- ***Example***: Add $+(21)_{10}$ and $-(75)_{10}$.
  Nine's complements of the operands are: 021 and 924
  021 + 924 = 945 (no carry, so this is the correct result).

Exercise

Consider A=5, B=6. Use n=4 bits for representations of numbers

Calculate (using 2 complement number system and 1 complement number system):

A+B

A-B

B-A

-A-B

Computer Codes (1)

- ***Code*** is a systematic use of a given set of symbols for representing information.

  Example: Traffic light (Red: stop, Yellow: caution, Blue: go).

- ***Numeric Codes***
  - To represent numbers.
  - Fixed-point and floating-point number.

- ***Fixed-point Numbers***
  - Used for signed integers or integer fractions.
  - Sign magnitude, two's complement, or one's complement systems are used.
  - Integer: (Sign bit) + (Magnitude) + (Implied radix point)
  - Fraction: (Sign bit) + (Implied radix point) + (Magnitude)

Computer Codes (2)

- ***Excess or Biased Representation***
  - An excess -*K* representation of a code *C*: Add *K* to each code word *C*.
  - Frequently used for the exponents of floating-point numbers.
  - Excess-8 representation of 4-bit two's complement code

Decimal	Two's Complement	Excess-8
+7	0111	1111
+6	0110	1110
+5	0101	1101
+4	0100	1100
+3	0011	1011
+2	0010	1010
+1	0001	1001
0	0000	1000
-1	1111	0111
-2	1110	0110
-3	1101	0101
-4	1100	0100
-5	1011	0011
-6	1010	0010
-7	1001	0001
-8	1000	0000

Characters and Other Codes (1)

To represent information as strings of alpha-numeric characters.

- ***Binary Coded Decimal (BCD)***
  - Used to represent the decimal digits 0 - 9.
  - 4 bits are used.

  Used
    - to encode numbers for output to numerical displays
    - Used in processors that perform decimal arithmetic.

<u>Example</u>: $(9750)_{10} = (1001011101010000)_{BCD}$

Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Characters and Other Codes (2)

- ***ASCII*** (American Standard Code for Information Interchange)
  - Most widely used character code.
  - The eighth bit is often used for error detection (parity bit)
  - ***Example***: ASCII code representation of the word ***Digital***

Character	Binary Code	Hexadecimal Code
D	1000100	44
i	1101001	69
g	1100111	67
i	1101001	69
t	1110100	74
a	1100001	61
l	1101100	6C

American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	∧	n	~
1111	SI	US	/	?	O	–	o	DEL

Characters and Other Codes (3)

- ***Gray Code***

  Two consecutive code words differ in only 1 bit

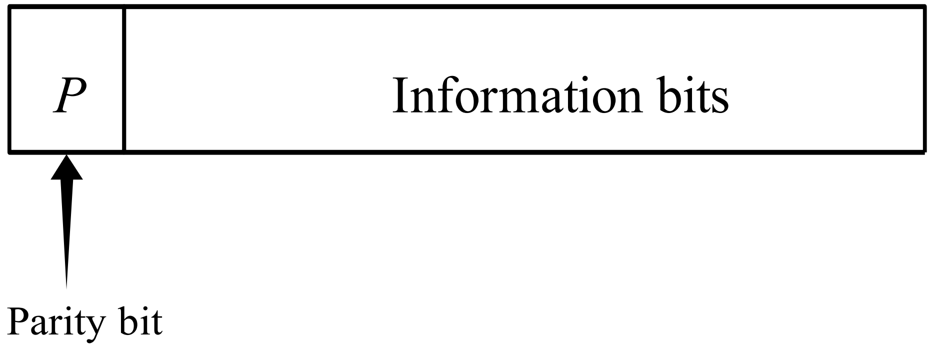
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

Error Detection Codes

- ***Simple Parity Code***
  - Concatenate (|) a *parity bit*, *P*, to each code word of *C*.

A parity bit is an extra bit included with a
message to make the total number of 1's either even or odd



Error Detection Codes

Character	ASCII Code	Odd-parity Code
0	0110000	10110000
X	1011000	01011000
=	0111100	1111100
BEL	0000111	00000111

This method detects one, three, or any odd combination of errors in each character that is transmitted. An even combination of errors, however, goes undetected, and additional error detection codes may be needed to take care of that possibility.

Floating Point Numbers (1)

- $N = M \times r^E$, where
  - M (mantissa or significand) is a significant digits of N
  - E (exponent or characteristic) is an integer exponent.
- In general, $N = \pm (a_{n-1} \ldots a_0 . a_{-1} \ldots a_{-m})_r$ is represented by
  - $N = \pm (.a_{n-1} \ldots a_{-m})_r \times r^n$
- M is usually represented in sign magnitude:
  - $M = (S_M . a_{n-1} \ldots a_{-m})_{rsm}$, where (1.14)

    $(.a_{n-1} \ldots a_{-m})_r$ represents the magnitude

    $M = (-1)^{S_M} \times (.a_{n-1} \ldots a_{-m})_r$ (0: positive, 1: negative) (1.15)

UNIVERSIDAD POLITÉCNICA TAIWAN – PARAGUAY
臺灣-巴拉圭科技大學

Floating Point Numbers (2) - IEEE 754 binary floating point

- *E* is coded in excess -*K*.
- *K* is called a bias and IEEE 754 selected to be 2^{e-1} -1 (*e* is the number of bits).
- So, biased *E* is:
  - $-2^{e-1} + 1 \leq \quad E \quad \leq 2^{e-1} - 1$

- The number 0 is represented by an all-zero word.

Floating Point Numbers (3)

- Multiple representations of a given number:

$$N = M \times r^{E} \quad (1.19)$$
$$= (M \div r) \times r^{E+1} \quad (1.20)$$
$$= (M \times r) \times r^{E-1} \quad (1.21)$$

- ***Example***: $M = +(1101.0101)_2$

$$M = +(1101.0101)_2$$
$$= (1.1010101)_2 \times 2^3 \quad (1.22)$$
$$= (0.011010101)_2 \times 2^5 \quad (1.23)$$
$$= (0.0011010101)_2 \times 2^6 \quad (1.24)$$
$$\ldots$$

- ***Normalization*** is used for a unique representation: mantissa has a nonzero value in its MSD position.
- Eq. 1.22 gives the normalization representation of M.

Floating Point Numbers (4)

- ***Floating-point Number Formats***
  - Typical single-precision format

S_M	Exponent E	Mantissa M

Sign of mantissa

code + bias

  - Typical extended-precision format

S_M	Exponent E	Mantissa M (most significant part)

Mantissa M (least significant part)

Convert a base 10 decimal number to 32 bit single precision (IEEE 754 binary floating point)

1. If the number to be converted is negative, start with its the positive version.
2. Convert from base 10 to binary number.
3. Normalize the binary representation of the number, by shifting the decimal point "*n*" positions either to the left or to the right, so that only one non zero digit remains to the left of the decimal point.
4. Normalize mantissa, remove the leading (leftmost) bit, since it's always '1' (and the decimal sign if the case) and adjust its length to 23 bits, either by removing the excess bits from the right (losing precision...) or by adding extra '0' bits to the right.
5. Adjust the exponent in 8 bit excess/bias notation and then convert it from decimal (base 10) to 8 bit binary:

   Exponent (adjusted) = Exponent (unadjusted) + $[2^{(8-1)} – 1]$

6. Sign (it takes 1 bit) is either 1 for a negative or 0 for a positive number.

Floating Point Numbers - IEEE 754 binary floating point

- ***Example***:

convert 325.25_{10} to IEEE 754 32bits floating point

$325_{10} = 1\ 0100\ 0101_2$

$0.25_{10} = 0.01_2$

$325.25_{10} = 1\ 0100\ 0101.01_2$

$$325.25_{10} = \underbrace{+}_{\text{sign}}1.\underbrace{0100\ 010101}_{\text{mantissa}}{}_2 \times 2^{\overset{\text{exponent}}{8}}$$

Mantissa= 010 0010 1010 0000 0000 0000

Exponent = $8 + (2^8 - 1) = 135_{10} = 1000\ 0111$

Sign= 0

number: $0\ 1000\ 0111\ 010\ 0010\ 1010\ 0000\ 0000\ 0000_2$

$43a2a000_{16}$

Floating Point Numbers - IEEE 754 binary floating point

- ***Example***:

convert -325.25_{10} to IEEE 754 32bits floating point

$325_{10} = 1\ 0100\ 0101_2$

$0.25_{10} = 0.01_2$

$325.25_{10} = 1\ 0100\ 0101.01_2$

$$325.25_{10} = \underbrace{+}_{\text{sign}}1.\underbrace{0100\ 010101}_{\text{mantissa}}{}_2 \times 2^{\overset{\text{exponent}}{8}}$$

Mantissa= 010 0010 1010 0000 0000 0000

Exponent = $8 + (2^8 - 1) = 135_{10} = 1000\ 0111$

Sign= 1

number: $1\ 1000\ 0111\ 010\ 0010\ 1010\ 0000\ 0000\ 0000_2$

$c3a2a000_{16}$

Floating Point Numbers - IEEE 754 binary floating point

- ***Example***:

convert 325.40_{10} to IEEE 754 32bits floating point

$325_{10} = 1\ 0100\ 0101_2$

$0.4_{10} = 0.011001100110\ldots_2$

Mantissa= 01000101011001100110011

Exponent = $8 + (2^8 - 1) = 135_{10} = 1000\ 0111$

Sign= 0

number: $0\ 1000\ 0111\ 010\ 0010\ 1011\ 0011\ 0011\ 0011\ _2$

$43a2b333_{16}$

$325.25_{10} = 1\ 0100\ 0101.0110\ 0110\ 0110\ 0110_2$

$325.25_{10} = \underset{\text{sign}}{+}1.\underbrace{01000101011001100110011}_{\text{Mantissa -- 23 bits}}{}_2 \times 2^{\overset{\text{exponent}}{8}}$

Convert a 32 bit single precision (IEEE 754 binary floating point) number to base 10

1. Sign (it takes 1 bit) is either 1 for a negative or 0 for a positive number.
2. Add the leading (leftmost) bit, since it's always '1'
3. Adjust the exponent in 8 bit excess/bias notation and then convert it from 8 bit binary to decimal (base 10):

Exponent (unadjusted) = Exponent (adjusted) - [$2^{(8-1)}$ – 1]

4. Convert from binary number to base 10.
5. Add sign

Floating Point Numbers - IEEE 754 binary floating point

- ***Example***:

convert $c5aa9000_{16\ \text{floating point}}$ to decimal

1- 10001011-01010101001000000000000

Mantissa= 1.01010101001000000000000

Exponent (adjusted) = $10001011_2 = 139_{10}$
Exponent (unadjusted) = 139 – 127 = 12

Number: $1.01010101001000000000000_2 \times 2^{12}$
$= 1010101010010.00000000000_2$
$= 1010101010010_2 = 5458_{10}$

UNIVERSIDAD POLITÉCNICA TAIWAN – PARAGUAY
臺灣–巴拉圭科技大學

Convert to floating point

a) 4518712.375

b) -12.37505

c) 124.203125

d) -124203125

Convert from single precision floating point to decimal

a) $C4025000_{16}$

b) $4c026800_{16}$