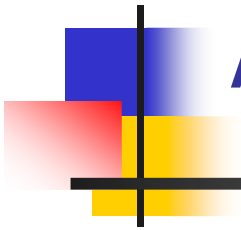


ARQUITECTURA DE COMPUTADORAS

Unidad N° 1

- Introducción
- Sistemas Numéricos

Profesor: Fabio Bruschetti

Ver 2013-01

UP Universidad de Palermo

Historia

Generación	Años	Características
0	hasta 1945	Sistemas mecánicos y electromecánicos
1	1945 – 1954	Tubos al vacío, tableros
2	1955 – 1965	Transistores y sistemas por lotes
3	1965 – 1980	Circuitos integrados
4	desde 1980	Computadores personales y supercomputadoras

UP Universidad de Palermo

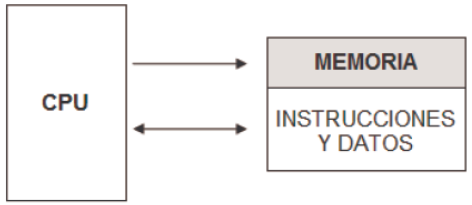
Historia – Modelo de Babbage

- Máquina diferencial (1823 – 1833)
  - Tabulación de polinomios de segundo grado por diferencias
  - Nunca se terminó de construir
- Máquina analítica (1833 – 1842)
  - Sus operaciones eran programables
  - Basada en el concepto de tarjetas perforadas de Joseph Jackard
  - Los programas podían tener bifurcaciones condicionadas (IF)
  - Tenía 3 componentes principales
    - La “fábrica” (Unidad Aritmético Lógica) – 4 operaciones
    - El “almacén” (Memoria)
    - Unidad de control y secuencia

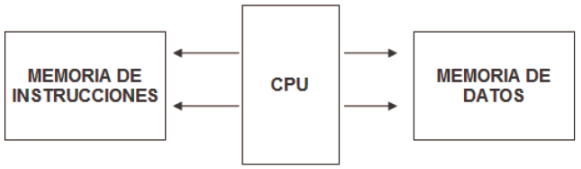
UP Universidad de Palermo

Historia – Arquitecturas

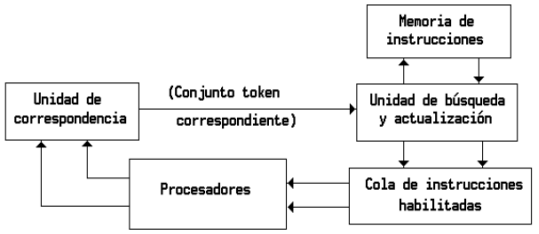
- Control Flow (secuenciamiento de las instruciones)
  - Von Newman



  - Hardvard



- Data Flow (disponibilidad de los datos)
  - Dinámica



Sistemas numéricos

- Sistema numéricos basados (base = b)
  - La base define el conjunto de símbolos
  - Un número = es una secuencia de símbolos
  - Reglas
    - Los dígitos (d) se enumeran de derecha a izquierda desde 0 (d_3 d_2 d_1 d_0)
    - La ubicación de cada dígito tiene un "peso" definido por la base
      - peso = base $^{\text{posición}}$
    - El valor del dígito depende del símbolo y de su ubicación
      - Valor = dígito * peso
    - El valor del número es la suma de los valores de sus dígitos

$$Valor = \sum_0^n d_i * b^i$$

UP Universidad de Palermo

Sistemas numéricos

- Sistema Decimal
  - Base = 10
  - Símbolos {0,1,2,3,4,5,6,7,8,9}
  - Notación: NNN_d – NNN_{10}
  - Ejemplo de 4 dígitos
    - $d_3\ d_2\ d_1\ d_0 = 1436_d$
      - $= d_3 \times 10^3 + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$
      - $= 1 \times 10^3 + 4 \times 10^2 + 3 \times 10^1 + 6 \times 10^0$
      - $= 1000 + 400 + 30 + 6$
- Nota:
  - Sumas y Restas: Pueden llevar o quitar 10's
  - Multiplicación y División por 10: Agrego o quito un cero de la derecha. Desplazo el número agregando o quitando de a un cero en la posición de menor peso d_0 ("Shift")

UP Universidad de Palermo

Números binarios enteros positivos

- Sistema Binario
  - Base = 2
  - Símbolos = {0,1}
  - Notación: NNN_b – NNN_2
  - Ejemplo de 4 dígitos
    - $d_3\ d_2\ d_1\ d_0 = 1010_b$
      - $= d_3 \times 2^3 + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$
      - $= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
      - $= 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$
      - $= 8 + 2$
      - $= 10_d$
- Nota:
  - Sumas y Restas: Pueden llevar o quitar 2's
  - Multiplicación y División por 2: "Shift" a Izquierda o Derecha

UP Universidad de Palermo

Números binarios enteros positivos

Suma

111_b + 10_b = 1001_b

1010_b + 111_b = 10001_b

Resta

101_b - 10_b = 011_b

1001_b - 111_b = 010_b

Multiplicación por potencias de 2

Por 2^1 : $100_b * 10_b = 1000_b$

Por 2^2 : $11_b * 100_b = 1100_b$

Por 2^3 : $100_b * 1000_b = 100000_b$

División por potencias de 2

Por 2^1 : $100_b / 10_b = 10_b$

Por 2^2 : $1100_b / 100_b = 11_b$

Por 2^3 : $100000_b / 1000_b = 100_b$

UP Universidad de Palermo

Números octales enteros positivos

- Sistema Octal
  - Base = 8 = 23
  - Notación: NNN_o – NNN_8
  - Símbolos = {0,1, 2, 3, 4, 5, 6, 7}
  - Ejemplo de 4 dígitos
    - $d_3\ d_2\ d_1\ d_0 = 1703_8$
      - $= d_3 \times 8^3 + d_2 \times 8^2 + d_1 \times 8^1 + d_0 \times 8^0$
      - $= 1 \times 8^3 + 7 \times 8^2 + 0 \times 8^1 + 3 \times 8^0$
      - $= 1 \times 512 + 7 \times 64 + 0 \times 8 + 3 \times 1$
      - $= 512 + 448 + 0 + 3$
      - $= 963_d$
- Nota:
  - Sumas y Restas: Pueden llevar o quitar 8's
  - Multiplicación y División por 8: "Shift" a Izquierda o Derecha

UP Universidad de Palermo

Números hexadecimales enteros positivos

- Sistema Hexadecimal
  - Base = 16 = 2^4
  - Permite manejar mejor los números binarios grandes
  - Notación: NNN_h – NNN_{16}
  - Símbolos = {0,1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
  - Ejemplo de 4 dígitos
    - $d_3\ d_2\ d_1\ d_0 = 12AF_h$
      - $= d_3 \times 16^3 + d_2 \times 16^2 + d_1 \times 16^1 + d_0 \times 16^0$
      - $= 1 \times 16^3 + 2 \times 16^2 + A \times 16^1 + F \times 16^0$
      - $= 1 \times 4096 + 2 \times 256 + 10 \times 16 + 15 \times 1$
      - $= 4096 + 512 + 160 + 15$
      - $= 4783_d$
- Nota:
  - Sumas y Restas: Pueden llevar o quitar 16's
  - Multiplicación y División por 16: "Shift" a Izquierda o Derecha

UP Universidad de Palermo

Números hexadecimales enteros positivos

Suma

181_h	1510_h
$+ \ 89_h$	$+ \ E11_h$
---	---
$20A_h$	2321_h

Resta

$1E1_h$	1001_h
$- \ 1F_h$	$- \ 111_h$
---	---
$1C2_h$	$EF0_h$

Multiplicación por potencias de 16

Por 16^1 : $20_h * 10_h = 200_h$

Por 16^2 : $3_h * 100_h = 300_h$

División por potencias de 16

Por 16^1 : $105_h / 10_h = 10_h$

Por 16^2 : $10E0_h / 100_h = 10_h$

Números binarios enteros positivos

- Dentro del computador, todos los números son representados sobre una cantidad fija de bits.
- Rango de representación
  - Con n bits se pueden formar 2^n combinaciones binarias distintas. Cada una de ellas corresponde a su respectivo número decimal
  - Si se comienza la representación en 0, entonces el número más grande representable es $2^n - 1$
  - Se pueden usar n bits para representar los números decimales
  - Ejemplo con 4 bits

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

UP Universidad de Palermo

Cambio de base

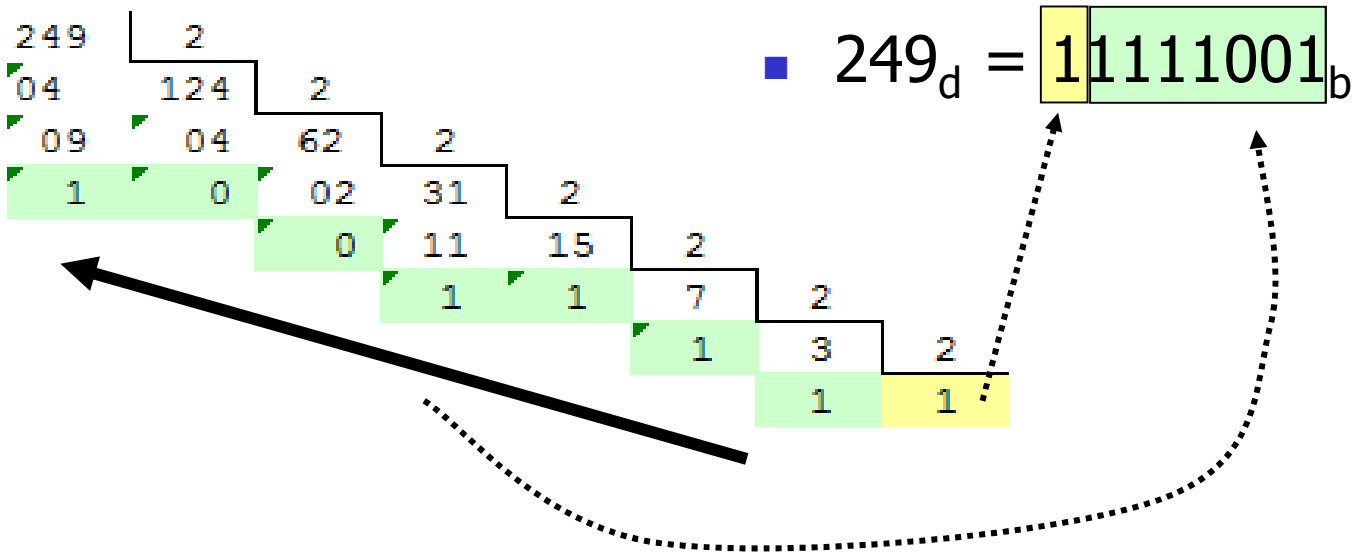
- Binario a Hexadecimal
  - 10 bits: 1001011001_2 $(= 601_{10})$
    - Agrupando en 4: 0010 0101 1001
    - Reemplazo con Hex: 2 5 9_{16}
- Hexadecimal a Binario
  - Binario: Reemplazar cada dígito hexadecimal por sus 4 dígitos binarios
    - Ejemplo: $A9F_h = 101010011111_b$

Cambio de base

- Binario a Octal
  - 16 bits: 1000101001101110_2 (= 35468_d)
    - Agrupando en 3: 001 000 101 001 101 110
    - Reemplazo con Hex: 1 0 5 1 5 6_8
- Octal a Binario
  - Binario: Reemplazar cada dígito octal por sus 3 dígitos binarios
    - Ejemplo: $134_8 = 001011100_b$

Cambio de base

- Decimal a Binario, Octal, Hexadecimal
  - Dividir el número decimal por la base a la que se lo quiere cambiar
  - Cuando se obtenga un cociente que menor a la base, formar el número partiendo desde el último cociente y, de derecha a izquierda completar sucesivamente con todos los restos



- $249_d = 11111001_b$

Preguntas

- ¿Qué significa cuando se dice "diez hex"?
- ¿Es verdad que $10100010_b = A2_h$?
- ¿Es verdad que $107_h = 407_8$
- ¿Cuántos bits tienen los siguientes números?
  - 0010010_b
  - 2_h
  - 22_h
  - 1010_h
  - 1010_b
  - 372_8

UP Universidad de Palermo

Unidades de Medidas

- Algunas abreviaturas:
  - Nibble = 4 bits
  - Byte = 8 Bits
  - Word (palabra)
    - 8 bits, 16 bits, 32 bits, 64 bits +
  - DWord (palabra doble)

Codificación de Caracteres

- Representación de caracteres mostrables:
  - Caracteres: { A, B , . . . , Y, Z }
  - 26 × 2 = 52 (mayúsculas y minúsculas)
  - Dígitos (10) decimales: {0, 1, . . . , 8, 9 }
  - Puntuación: ! ”’, . –? / : ;
  - Símbolos matemáticos: + * = (– /)
  - Paréntesis: ( ) [ ] { } < >
  - Otros: @ # $ % ^ & \| ~
  - Espacio en blanco: “ ”
  - 90+ Símbolos (??)
  - Varios esquemas de codificación han sido usados

UP Universidad de Palermo

Codificación ASCII

- ASCII = **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (7 bits)
  - 7 bits para codificar cada caracter (128 códigos)
  - Se extiende a 8 bits (byte) poniendo el bit más significativo = 0
  - 2 dígitos hexadecimales
- Ejemplo
  - "306 is FUN!" → 33_h 30_h 36_h 20_h 69_h 73_h 20_h 46_h 55_h $4E_h$ 21_h
- ASCII Estendido (8 bits)
  - Van del 128 hasta el 255
  - Incluye caracteres propios de diferentes lenguajes (á, é, ĉ) , caracteres gráficos (▒), símbolos especiales (√), etc.

Codificación ASCII (7-bits)

ASCII Code Table

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?
ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	•

Codificación EBCDIC (8-bits)

- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- Código estándar de 8 bits usado por computadoras mainframe de IBM. IBM adaptó el EBCDIC del código de tarjetas perforadas en los años 1960

	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0000 0	NUL 0	DLE 16	DS 32	48	SP 64	& 80	- 96	112	128	144	160	176	{ 192	} 208	\ 224	0 240
0001 1	SOH 1	DCI 17	SOS 33	49	65	/ 81	97	113	a 129	j 145	161	177	A 193	J 209	225	1 241
0010 2	STX 2	DC2 18	FS 34	SYN 50	66	82	98	114	b 130	k 146	s 162	178	B 194	K 210	S 226	2 242
0011 3	ETX 3	TM 19	35	51	67	83	99	115	c 131	l 147	t 163	179	C 195	L 211	T 227	3 243
0100 4	PF 4	RES 20	BYP 36	PN 52	68	84	100	116	d 132	m 148	u 164	180	D 196	M 212	U 228	4 244
0101 5	HT 5	NL 21	LF 37	RS 53	69	85	101	117	e 133	n 149	v 165	181	E 197	N 213	V 229	5 245
0110 6	LC 6	BS 22	ETB 38	UC 54	70	86	102	118	f 134	o 150	w 166	182	F 198	O 214	W 230	6 246
0111 7	DEL 7	IL 23	ESC 39	EOT 55	71	87	103	119	g 135	p 151	x 167	183	G 199	P 215	X 231	7 247
1000 8	8	CAN 24	40	56	72	88	104	120	h 136	q 152	y 168	184	H 200	Q 216	Y 232	8 248
1001 9	RLF 9	EM 25	41	57	73	89	105	\ 121	i 137	r 153	z 169	185	I 201	R 217	Z 233	9 249
1010 A	SMM 10	CC 26	SM 42	58	cent 74	! 90	| 106	: 122	138	154	170	186	202	218	234	250
1011 B	VT 11	CU1 27	CU2 43	CU3 59	. 75	$ 91	, 107	# 123	139	155	171	187	203	219	235	251
1100 C	FF 12	IFS 28	44	DC4 60	< 76	* 92	% 108	@ 124	140	156	172	188	204	220	236	252
1101 D	CR 13	IGS 29	ENQ 45	NAK 61	(77	) 93	_ 109	' 125	141	157	173	189	205	221	237	253
1110 E	SO 14	IRS 30	ACK 46	62	+ 78	; 94	> 110	= 126	142	158	174	190	206	222	238	254
1111 F	SI 15	IUS 31	BEL 47	SUB 63	| 79	¬ 95	? 111	" 127	143	159	175	191	207	223	239	255

UP Universidad de Palermo

Código BCD Natural

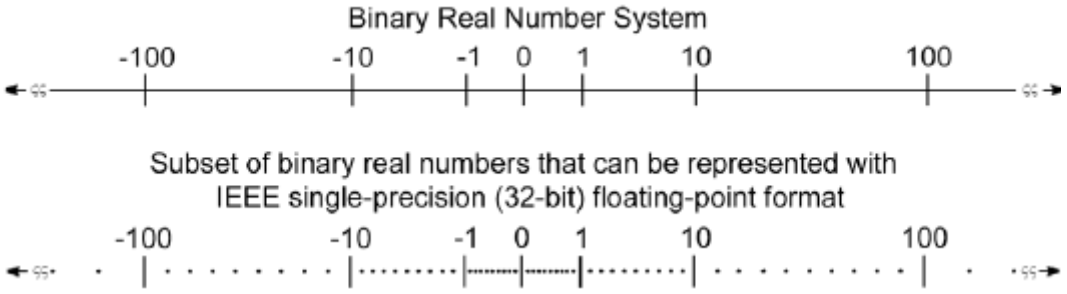
- Representa los dígitos decimales del 0 al 9 con una combinación de 4 bits para cada uno de ellos
- Ejemplo
  - 348_{10}
    - = 0011|0100|1000 en BCD
- Los números decimales no se representan por su correspondiente combinación binaria
- Ejemplo
  - 348_{10}
    - = 101011100_2

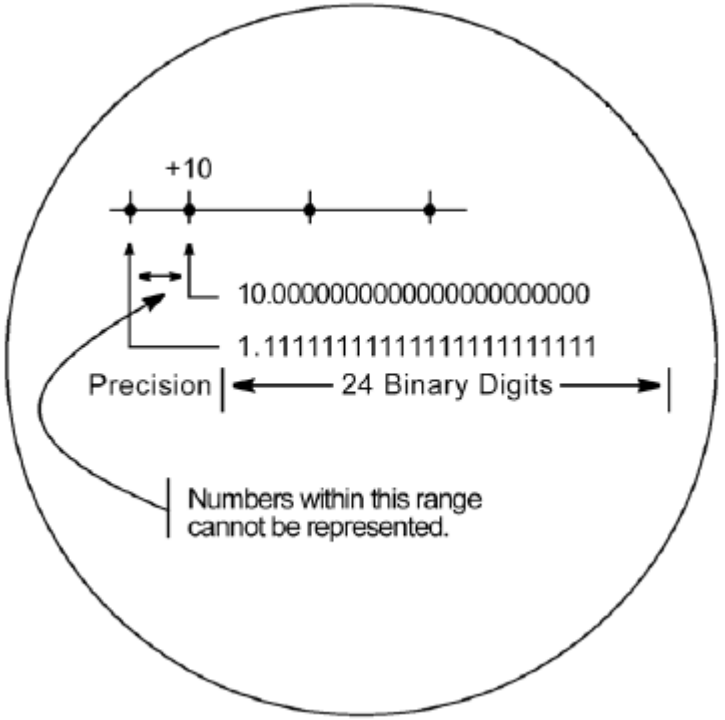
Decimal	BCD Natural			
N°	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

UP Universidad de Palermo

Números Reales

- El rango de los números reales comprende desde -∞ hasta +∞.
- Los registros de un procesador tienen resolución finita.
- Por lo tanto un computador solo puede representar un sub conjunto de R . (No es solo un tema de magnitud sino de resolución)





de Palermo

Números Reales

- En general se puede formalizar la representación de un número real expresado en los siguientes formatos:
  - Punto Fijo
    - Con Módulo y signo
    - Con complemento a 2
  - Punto Flotante
- Como convertir un número decimal a binario:
  - 0,828125 * 2 = 1,65625
  - 0,65625 * 2 = 1,3125
  - 0,3125 * 2 = 0,625
  - 0,625 * 2 = 1,25
  - 0,25 * 2 = 0,5
  - 0,5 * 2 = 1

$0{,}828125_d = 0{,}110101_b$

Números Reales – Punto Fijo

- Punto Fijo con signo
  - Se representan mediante una expresión del tipo
    - $(a_n\, a_{n-1}\, ..a_0\, .\, a_{-1}\, a_{-2}\, ..a_{-m})_2 = (-1)^s\, (a_n 2^n + .. + a_0 2^0 + a_{-1} 2^{-1} + .. + a_{-m}\, 2^{-m})$
    - *Donde: s=0 si el número es positivo o =1 si el número es negativo*
    - a_i *es un entero y* $0 \leq a_i \leq 1$, *para todo* $i = -m, ...-1, 0, 1, ...n$
  - Distancia entre dos números consecutivos es 2^{-m} (su base elevado a la –m)
  - Deja de ser un rango continuo de números y pasa a ser un rango discreto.

Números Reales – Punto Fijo

- Cuando la cantidad de dígitos disponible no alcanza para representar el número …
  - Problema: Representar un número de n dígitos decimales en un sistema con m dígitos decimales, siendo m < n
- Truncamiento:
  - Descarta los dígitos fraccionarios de orden mayor a m.
- Redondeo:
  - Descarta los dígitos fraccionarios de orden mayor a m pero se suma 1 al menos significativo en caso que el bit inmediato (m+1) descartado valga 1.

UP Universidad de Palermo

Números Reales – Punto Fijo

- Con truncamiento y redondeo
  Punto fijo en posición 4 con 11 bits:
  $31,9375_{10}$ = 0011111.1111_2
  $0,0625_{10}$ = 0000000.0001_2
  $32,0000_{10}$ = 0100000.0000_2
  - Si tenemos el siguiente número $31,906025_{10}$
    0011111.111010_2
  - Si se trunca en 4 bits
    0011111.1110_2 → $31,875_{10}$
  - Si se redondea y trunca en 4 bits
    0011111.1111_2 → $31,921875_{10}$
- Vemos que con truncamiento hay menor error

Números Reales – Punto Flotante

- Para el caso de los números reales se trabaja en notación científica.
  - $-725.832 = -7.25832 \cdot 10^2 = -725.832 \times 10^0$
  - $3.14 = 0.314 * 10^1 = 3.14 * 10^0$
  - $0.000001 = 0.1 * 10^{-5} = 1.0 * 10^{-6}$
  - $1941 = 0.1941 * 10^4 = 1.941 * 10^3$
- Para unificar la representación se recurre a la notación científica normalizada, en donde
  - $n = \pm f * 10^e$
    - $0.1 \leq f < 1$
    - e es un entero con signo
- En el sistema binario la expresión de un número en notación científica normalizada es:
  - $n = \pm f * 2^e$
    - $0.5 \leq f < 1$
    - e es un entero con signo

UP Universidad de Palermo

Números Reales – Punto Flotante

- Representación en Punto Flotante
  - Se representan con los pares de valores (m, e), denotando:
    - $(m, e) = m * b^e$
  - **m** llamado mantisa, y que representa un número fraccionario
  - **e**, llamado exponente, al cual se debe elevar la base numérica (b) de representación para obtener el valor real
  - Mantisa y exponente pueden representarse:
    - con signo
    - sin signo
    - con notación complemento
    - con notación exceso m (biased)
  - **Para que las representaciones sean únicas, la mantisa deberá estar normalizada**.
    - Cuando un número fraccionario tiene su dígito más significativo distinto de 0, se dice que está normalizado

UP Universidad de Palermo

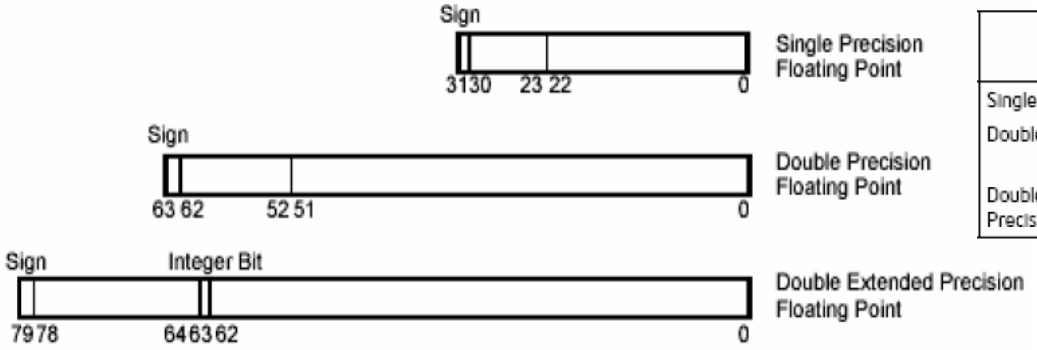
Números Reales – Punto Flotante

- Representación en Punto Flotante de 32 bits:
  - los bits 0 al 22 (b_0 a b_{22}) representa la mantisa normalizada para el sistema Módulo y Signo
  - los bits 23 al 30 (b_{23} a b_{30}) representa el exponente en exceso a 128
  - el bit 31 (b_{31}), para representar el signo de la mantisa (0 para el +)
  - la base de exponenciación es 2
  - el 0 se representa con todos los bits en 0.

- Ej: Representar el número 12 en formato de 32 bits:
  - 12 en notación normalizada de base 2 es $0{,}75 * 2^4$
  - el exponente de valor 4 en exceso a 128 es: $4 + 128 = 132_{10} = 10000100_2$
  - la mantisa 0,75 en binario es 0,11
  - de donde la representación del número 12 quedará como:
    - **0** **10000100** **11000000000000000000000**
    - signo (+) exponente 4 mantisa 0,75

Números Reales – Punto Flotante

- Punto Flotante: Formato IEEE 754 (Institute of Electrical and Electronics Engineers, Inc. Año 1985)
  - Cuatro formatos
    - Precisión simple (32 bits)
    - Precisión doble (64 bits)
    - Precisión simple extendida (≥ 43 bits), no muy usada
    - Precisión doble extendida (80 bits)
  - Sólo los valores de 32 bits son requeridos por el estándar, los otros son opcionales



Data Type	Length	Precision (Bits)	Approximate Normalized Range	
			Binary	Decimal
Single Precision	32	24	2^{-126} to 2^{127}	1.18×10^{-38} to 3.40×10^{38}
Double Precision	64	53	2^{-1022} to 2^{1023}	2.23×10^{-308} to 1.79×10^{308}
Double Extended Precision	80	64	2^{-16382} to 2^{16383}	3.37×10^{-4932} to 1.18×10^{4932}

Números Enteros Negativos

- Para números sin signo:
  - Los n bits son usados para la magnitud del número
  - Ejemplo: Número de 4-bits: 0000b 1111_b
- Para números con signo: (Negativos)
  - Un bit debe ser usado para el signo
    - Usualmente el bit mas significativo: 0= positivo 1=negativo
  - n-1 bits son usados para la magnitud del número
- Codificación de la magnitud (2 aproximaciones)
  - Codificación magnitud con signo
    - Progresión natural de los números sin signo
    - Inadecuado para las matemáticas
  - Codificación complemento a 2
    - Excelente para las matemáticas

UP Universidad de Palermo

Codificación con signo

- La magnitud se codifica en los n-1 bits restantes usando el sistema binario usado para contar
  - Ejemplo: $10000001_b = -1_d$
    - magnitud: 000 0001b = 1
    - signo: 1 o sea, número negativo
  - Problemas:
    - Dos representaciones para el 0
      - 0 positivo signo=0
      - 0 negativo signo=1
    - Problemas con la aritmética:

```
   0000 0010b        2
+  1000 0001b    + -1
-----------------  ------
   1000 0011b       -3
```

Codificación complemento a 2

- La magnitud de los números positivos se codifica normalmente como número binario
- La magnitud de los números negativos se codifica como "flip&add"
- Definiciones:
  - Complemento de un bit:
    - Complemento de 1: 0
    - Complemento de 0: 1
  - Complemento a 1 de un valor de n-bits:
    - Complemento de cada bit
  - Complemento a 2 de un valor de n-bits:
    - Complemento de cada bit y luego (add) sumar 1
    - Se ignora cualquier carry del bit mas significativo.

UP Universidad de Palermo

Codificación complemento a 2

- Ejemplo:
  - Encuentre la representación de –1 en complemento a 2 con 8 bits.

$+1_{10}$	$0000\ 0001_b$
Complemento	$1111\ 1110_b$
Sumar 1	$+\quad 1$
----------	----------
-1_{10}	$1111\ 1111_b = FF_h$

Operación de complemento a 2

  - Encuentre la representación de 1 en complemento a 2 con 8 bits
    - $+1_{10}$ $0000\ 0001_b = 01_h$
  - Encuentre el valor decimal del valor FEh en complemento a 2
    - Bit de signo = 1 $1111\ 1110_b$
    - Complemento $0000\ 0001_b$
    - Sumar 1 $+\quad 1_b$
    - ---------- ----------
    - FE_h $0000\ 0010_b$ = (bit de signo) $2_{10} = -2_{10}$

Operación de complemento a 2

UP Universidad de Palermo

Codificación complemento a 2

- Ejemplo:
  - Encuentre el valor decimal del valor 79h en complemento a 2
    - Bit de signo = 0 $0111\ 1001_b = 64+32+16+8+1= 121_d$
    - $79_h = 121_d$

- Ejercicios:
  - Encuentre el valor decimal del valor 90_h en complemento a 2
  - Encuentre el valor decimal del valor $8D_h$ en complemento a 2
  - Encuentre el valor decimal del valor $6F_h$ en complemento a 2

Codificación complemento a 2

- Rango:
  - Con n-bits se pueden representar a lo sumo 2^n números naturales diferentes pero...
  - Usando la mitad de los valores para los número negativos y el resto para los positivos y el 0 nos quedan (2^{n-1} valores positivos y 2^{n-1} valores negativos)
  - Rango para valores positivos y negativos:
    - Todos los negativos comienzan con 1
    - Rango: -2^{n-1} ..... 0 ..... $2^{n-1}-1$

Codificación complemento a 2

- Negar un número negativo se obtiene el número original [- (-x) = x]

-1_{10}		$1111\ 1111_b$
Complemento		$0000\ 0000_b$
Sumar 1	+	1_b
----------		----------
$+1_{10}$		$0000\ 0001_b = 01_h$ → Funciona !!!

- Cuantas representaciones para 0? [única]

0_{10}		$0000\ 0000_b$
Complemento		$1111\ 1111_b$
Sumar 1	+	1_b
----------		----------
0_{10}		$0000\ 0000_b$ Funciona !!!

Se ignora el carry del bit mas significativo

Codificación complemento a 2

- Como se obtiene la codificación de –128?
  - Usando matemáticas
    - -127 - 1 = -128

-127_{10}		$1000\ 0001_b$
Restar 1	-	$0000\ 0001_b$
----------		----------
-128_{10}		$1000\ 0000_b$

  - Usando Complemento a 2
    - Si se niega –128 ? [Caso especial]

-128_{10}		$1000\ 0000_b$
Complemento		$0111\ 1111_b$
Sumar 1	+	1_1
----------		----------
-128_{10}		$1000\ 0000_b$

UP Universidad de Palermo

Suma y resta aritmética

- Aritmética sin signo
  - 117_{10} = $0111\ 0101_b$
  - $+\ 99_{10}$ = $0110\ 0011_b$
  - 216_{10} = $1001\ 1000_b$
- Aritmética con signo
  - -117_{10} = $1000\ 1011_b$
  - $+\ 99_{10}$ = $0110\ 0011_b$
  - -18_{10} = $1110\ 1110_b$ $(0001\ 0001 + 1) = 12_h = 18_d$
- Restas en binario:
  - Negar y sumar: $X - Y = X + (-Y)$
  - Ejemplo: $32 - 65 = 32 + (-65)$
    - $32_{10} = 0010\ 0000_b$
    - $+\ -65_{10} = 1011\ 1111_b$ ($0100\ 0000_b + 1 = 41_h = 65_d$)
    - $-33_{10} = 1101\ 1111_b$ ($0010\ 0000_b + 1 = 21_h = 33_d$)

Aritmética y Lógica

- Multiplicación binaria
  - Procedimiento de resolución del cálculo de forma idéntica al de la multiplicación decimal

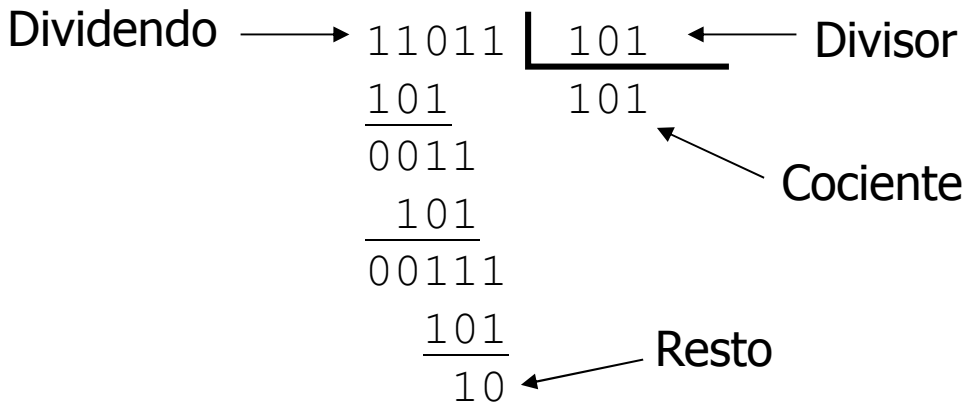
```
   1101   <-- Multiplicando
 x 101    <-- Multiplicador
  1101
+ 0000
 1101
1000001
```

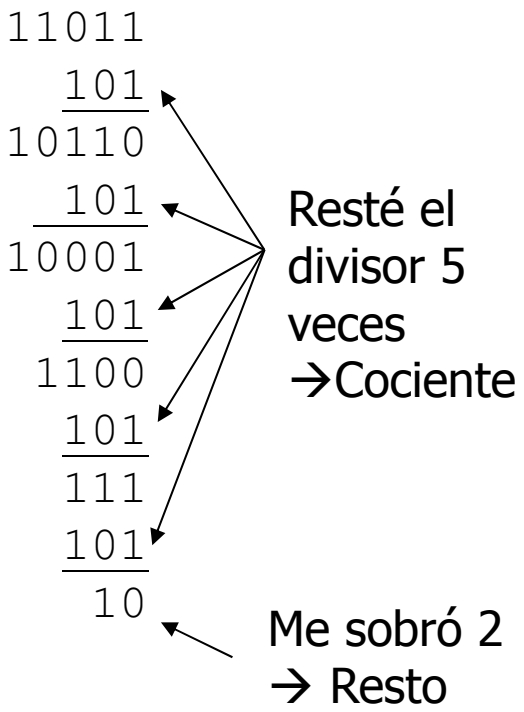
  - Por cada cifra del multiplicador, me desplazo en la suma final
  - Por cada 1 del multiplicador, repito el multiplicando en la suma final
  - La multiplicación es una "sucesión de sumas y desplazamientos"
  - Multiplicar 2 cifras binarias de n bits darán como resultado otra cifra binaria de 2.n bits! (1111_2 x 1111_2 = 11100001_2)

Aritmética y Lógica

División binaria

- Procedimiento de resolución del cálculo de forma idéntica al de la multiplicación decimal





- Ejemplo: 27 / 5 = 5 con resto 2
- La división es una "sucesión de restas y desplazamientos"

Pentium – Formatos de datos

- General → contenido binario arbitrario
- Entero → binario entero con signo representado en Ca2
- Ordinal → binario entero sin signo
- BCD → 1 ó 2 dígitos BCD por byte
- Punto flotante → precisión simple, doble y ampliada