Corso di Laurea in Informatica
Architettura degli elaboratori
a.a. 2020-2021

Architettura degli Elaboratori 2020-2021

Rappresentazione numeri interi con segno

Prof. Elisabetta Fersini
elisabetta.fersini@unimib.it

Sistema binario: operazioni aritmetiche

- Somma: si esegue la somma tra i bit di pari ordine
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = 0 con riporto 1 sul bit di ordine superiore
  - 1 + 1 + 1 = 1 con riporto 1 sul bit di ordine superiore

- La somma è definita su 3 elementi:
  - due addendi
  - il riporto (*carry*)

- La somma di 2 unità (valore 1) di un dato ordine, creano 1 unità dell'ordine immediatamente superiore (carry).

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la somma tra 010011 e 010001 (che indicano, rispettivamente, i numeri decimali 19 e 17)

RIPORTO						
PRIMO ADDENDO	0	1	0	0	1	1
SECONDO ADDENDO	0	1	0	0	0	1
SOMMA						

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la somma tra 010011 e 010001 (che indicano, rispettivamente, i numeri decimali 19 e 17)

RIPORTO					1	
PRIMO ADDENDO	0	1	0	0	1	1
SECONDO ADDENDO	0	1	0	0	0	1
SOMMA						0

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la somma tra 010011 e 010001 (che indicano, rispettivamente, i numeri decimali 19 e 17)

RIPORTO				1	1	
PRIMO ADDENDO	0	1	0	0	1	1
SECONDO ADDENDO	0	1	0	0	0	1
SOMMA					0	0

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la somma tra 010011 e 010001 (che indicano, rispettivamente, i numeri decimali 19 e 17)

RIPORTO			0	1	1	
PRIMO ADDENDO	0	1	0	0	1	1
SECONDO ADDENDO	0	1	0	0	0	1
SOMMA				1	0	0

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la somma tra 010011 e 010001 (che indicano, rispettivamente, i numeri decimali 19 e 17)

RIPORTO		0	0	1	1	
PRIMO ADDENDO	0	1	0	0	1	1
SECONDO ADDENDO	0	1	0	0	0	1
SOMMA			0	1	0	0

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la somma tra 010011 e 010001 (che indicano, rispettivamente, i numeri decimali 19 e 17)

RIPORTO	1	0	0	1	1	
PRIMO ADDENDO	0	1	0	0	1	1
SECONDO ADDENDO	0	1	0	0	0	1
SOMMA	1	0	0	1	0	0

36 in decimale

Sistema binario: operazioni aritmetiche

- Sottrazione: si esegue la differenza tra i bit di pari ordine
  - 0 – 0 = 0
  - 1 – 0 = 1
  - 1 – 1 = 0
  - 0 – 1 = 1 con prestito dal bit di ordine immediatamente superiore

- Anche la sottrazione opera su gruppi di 3 bit:
  - minuendo e sottraendo
  - prestito (borrow) proveniente dalla cifra di ordine immediatamente superiore

- Ogni volta che si deve sottrarre dalla cifra 0 la cifra 1, occorre chiedere in prestito una unità alla cifra di ordine immediatamente superiore che vale due unità della cifra di ordine inferiore.

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

PRESTITO						
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA						

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

PRESTITO						
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA						1

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

PRESTITO					2	
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA					1	1

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

Prestito di una unità -> il minuendo sarà 0 e non 1

PRESTITO					2	
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA					1	1

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

Prestito di una unità -> il minuendo sarà 0 e non 1

PRESTITO				2	2	
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA				1	1	1

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

Prestito di una unità -> il minuendo sarà 0 e non 1

PRESTITO				2	2	
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA				1	1	1

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

Prestito di una unità -> il minuendo sarà 0 e non 1

PRESTITO			2	2	2	
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA			1	1	1	1

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

Prestito di una unità -> il minuendo sarà 0 e non 1

PRESTITO			2	2	2	
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA			1	1	1	1

Sistema binario: operazioni aritmetiche

- *Esempio*: si esegua la sottrazione tra 11101 e 01110 (che indicano, rispettivamente, i numeri decimali 29 e 14)

Prestito di una unità -> il minuendo sarà 0 e non 1

PRESTITO			2	2	2	
MINUENDO		1	1	1	0	1
SOTTRAENDO		0	1	1	1	0
DIFFERENZA		0	1	1	1	1

15 in decimale

Sistema binario: operazioni aritmetiche

- Supponendo di avere a disposizione 6 bit, si calcoli la somma e la sottrazione in binario dei seguenti numeri:

010101 e 110100

- Effettuando la somma:

RIPORTO						
PRIMO ADDENDO	0	1	0	1	0	1
SECONDO ADDENDO	1	1	0	1	0	0
SOMMA						

Sistema binario: operazioni aritmetiche

- Supponendo di avere a disposizione 6 bit, si calcoli la somma e la sottrazione in binario dei seguenti numeri:

010101 e 110100

- Effettuando la somma:

RIPORTO	1		1			
PRIMO ADDENDO	0	1	0	1	0	1
SECONDO ADDENDO	1	1	0	1	0	0
SOMMA	0	0	1	0	0	1

il numero ottenuto non è rappresentabile in quanto il risultato è su 7 bit!

Sistema binario: operazioni aritmetiche

- Supponendo di avere a disposizione 6 bit, si calcoli la somma e la sottrazione in binario dei seguenti numeri:

010101 e 110100

- Effettuando la somma: **OVERFLOW!**

RIPORTO	1		1			
PRIMO ADDENDO	0	1	0	1	0	1
SECONDO ADDENDO	1	1	0	1	0	0
SOMMA	0	0	1	0	0	1

il numero ottenuto non è rappresentabile in quanto il risultato è su 7 bit!

Sistema binario: operazioni aritmetiche

- Supponendo di avere a disposizione 6 bit, si calcoli la somma e la sottrazione in binario dei seguenti numeri:

010101 e 110100

- Effettuando la differenza:

PRESTITO						
MINUENDO	0	1	0	1	0	1
SOTTRAENDO	1	1	0	1	0	0
DIFFERENZA						

Sistema binario: operazioni aritmetiche

- Supponendo di avere a disposizione 6 bit, si calcoli la somma e la sottrazione in binario dei seguenti numeri:

010101 e 110100

- Effettuando la differenza:

PRESTITO						
MINUENDO	0	1	0	1	0	1
SOTTRAENDO	1	1	0	1	0	0
DIFFERENZA	???	0	0	0	0	1

Non è possibile eseguire l'operazione

Sistema binario: operazioni aritmetiche

- Perché non è possibile effettuare l'operazione?
  ① Per poter eseguire l'operazione avrei bisogno di chiedere in prestito un un bit da un ordine superiore, ma **avrei bisogno di 7 bit** e non 6 bit.
  ② L'operazione $(010101 \quad -110100)_2$ sarebbe $(21 - 52)_{10} = -31_{10}$. Un numero **negativo non è rappresentabile** nel sistema binario puro.

PRESTITO						
MINUENDO	0	1	0	1	0	1
SOTTRAENDO	1	1	0	1	0	0
DIFFERENZA	???	0	0	0	0	1

Non è possibile eseguire l'operazione

Rappresentazione numeri negativi

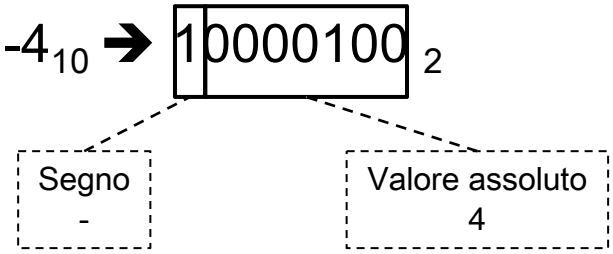
- Come visto in precedenza, con la modalità di rappresentazione semplice non è possibile rappresentare **numeri negativi**.
- Per ovviare a questo problema è stato definito un metodo di rappresentazione dal nome di **Modulo e Segno (MS)**.
- Vedremo anche altre rappresentazioni:
  - Complemento a 1 (CA1)
  - Complemento a 2 (CA2)
  - Eccesso 128

Modulo e Segno

- Supponiamo di avere a disposizione 1 Byte (8bit) per rappresentare numeri sia positivi che negativi.

- Ricorrendo al metodo del **Modulo e Segno** (MS) utilizzeremo:
  - i primi 7 bit da destra per il valore assoluto del numero
  - il bit più a sinistra (MSB) per indicare il segno
    - 1 se il numero è negativo, 0 se è positivo

- *Esempio*: supponiamo di voler rappresentare -4_{10} con 8bit



Modulo e Segno

- Con ***n*** **bit** totali, si possono rappresentare i numeri interi nell'**intervallo**

$$[\,-(2^{n-1}-1)\;,\;+(2^{n-1}-1)\,]_{10}$$

- Quali sono i **problemi** della rappresentazione MS?
  - Esistono **due diverse rappresentazioni dello 0**. Presi ad esempio 4 bit totali:

$$0000_2 = +0_{10} \qquad\qquad 1000_2 = -0_{10}$$

  - Un bit tra tutti i bit disponibili **viene "speso" per il segno.**

Complemento a 1 (CA1)

- Oltre al metodo MS esiste un altro modo di rappresentare i numeri interi (positivi e negativi), ovvero il **Complemento a 1 (CA1)**.
- Come indica il nome stesso, questo metodo si basa sull'**operazione di complemento**.
- Con **complemento** si intende l'operazione che associa ad un bit (o ad ogni sequenza di bit) il suo opposto, cioè il valore ottenuto sostituendo tutti gli 1 con 0, e tutti gli 0 con 1.
- *Esempio*: il complemento di 1001 è 0110.

Complemento a 1 (CA1)

- Il metodo CA1 è molto semplice e diretto:

  ① Se il numero da codificare è **positivo** lo si converte in binario con il metodo tradizionale.

  ② Se il numero è **negativo** basta convertire in binario il suo modulo e quindi eseguire l'operazione di complemento sulla codifica binaria effettuata.

- *Esempi*:
  - supponiamo di avere 4 bit e di voler rappresentare in CA1 il valore 3_{10}

$$3_{10} = 0011_2$$

  - supponiamo di avere 4 bit e di voler rappresentare in CA1 il valore -3_{10}

$$-3_{10} = \overline{0011_2} = 1100_2$$

Complemento a 1 (CA1)

- Anche in questo caso sussiste il problema delle due diverse rappresentazioni dello 0.

0000 0000 (+0)

1111 1111 (-0)

- È stato quindi introdotto un ulteriore metodo di codifica, ovvero il Complemento a 2.

Complemento a 2 (CA2)

- Basato CA1, il **Complemento a 2 (CA2)** è un altro metodo di codifica usato per rappresentare i numeri interi sia positivi che e negativi.

- Il metodo CA2 opera come segue sul valore **X** da codificare:

  ① Se il numero **X** è **positivo** esso rimane invariato.

  ② Se il numero **X** è **negativo**

  - Si effettua il complemento a 1 (CA1) sul valore da codificare
  - Si somma +1 al risultato ottenuto con CA1

Complemento a 2 (CA2)

- Il metodo CA2 **supera** il principale **difetto di CA1**, ossia la presenza di una doppia codifica per lo 0. In CA2 lo 0 ha un'unica rappresentazione.

- In CA2 i valori **negativi** hanno **MSB = 1**

- Dati **n bit**, si possono rappresentare i numeri interi nell'intervallo

$$[-(2^{n-1}) \ , \ +(2^{n-1}-1)]_{10}$$

- Esempio: dati 5 bit si ha

$$-16_{10} \leq X \leq +15_{10}$$

Complemento a 2 (CA2)

- Esempi di CA2

Bit	Valore Assoluto	Complemento a 2
0111 1111	127	127
0111 1110	126	126
0000 0010	2	2
0000 0001	1	1
0000 0000	0	0
1111 1111	255	−1
1111 1110	254	−2
1000 0010	130	−126
1000 0001	129	−127
1000 0000	128	−128

Complemento a 2 (CA2)

- Esistono 3 metodi per il calcolo di CA2 di un numero X:

① Definizione di complemento alla base

$$\mathrm{CA2}(X) = 2^n - X$$

② Per calcolare CA2 si calcola CA1 e si somma 1

- Partendo dalla definizione di complemento alla base – 1

$$\mathrm{CA1}(X) = (2^n - 1) - X$$

possiamo definire CA2 in funzione di CA1

$$\mathrm{CA2}(X) = \mathrm{CA1}(X) + 1$$

Complemento a 2 (CA2)

- Esistono 3 metodi per il **calcolo di CA2** di un numero X:

  ③ Regola pratica:
  - si parte **da destra**, si trascrivono tutti gli 0 fino ad incontrare il primo 1 e si trascrive anch'esso
  - si complementano a 1 ($0 \rightarrow 1$ e $1 \rightarrow 0$) tutti i bit restanti

- Nota bene:
  - Il CA2 di un numero negativo dà il corrispondente valore positivo
  - $CA2(CA2(X)) = X$

Complemento a 2 (CA2)

- Esempio: Rappresentare in CA2 su 4 bit il numero –7

$$7_{10} = 111_2$$

① Applicando la definizione:

$$2^4 - 7 = 10000 - 111 = 1001_{CA2}$$

① Passando dal CA1:

$$\overbrace{111_2}^{7} \xrightarrow{4bit} \overbrace{0111_{CA1}}^{+7} \xrightarrow{CA1} \overbrace{1000_{CA1}}^{-7} \xrightarrow{+1} \overbrace{1001_{CA2}}^{-7}$$

① Con la regola pratica

$$\overbrace{111_2}^{7} \xrightarrow{4bit} \overbrace{0111_{CA2}}^{+7} \xrightarrow{CA2} \overbrace{1001_{CA2}}^{-7}$$

Rappresentazione CA2 e operazione CA2

- Distinzione tra rappresentazione in CA2 e operazione di CA2
  - La rappresentazione - come sono organizzati i bit
  - Il calcolo - procedura di trasformazione dei bit

- Per rappresentare un numero positivo in CA2 non serve applicare l'operazione di CA2.

- Per rappresentare un numero negativo in CA2 è necessario applicare l'operazione di CA2 alla rappresentazione del corrispondente valore positivo.

Conversione da CA2 a decimale

- Se il numero è **positivo** (MSB = 0), si converte in base decimale usando il numero binario puro.

- Se il numero è **negativo** (MSB = 1), si applica l'operazione di CA2 a questo valore ottenendo la rappresentazione del corrispondente positivo, si converte il risultato come numero in binario puro e si aggiunge il segno meno.

Operazioni aritmetiche - MS

- Come si esegue la **somma** di due valori in Modulo e Segno?
- Confronto i **bit di segno** dei due numeri:

a) Se i bit di segno sono **uguali**:

- Il bit di segno risultante sarà il bit di segno dei due addendi
- Eseguo la somma bit a bit (a meno di overflow)

b) Se i bit di segno sono **diversi**:

- Confronto i valori assoluti dei due addendi
- Il bit di segno risultante sarà il bit di segno dell'addendo con valore assoluto maggiore
- Eseguo la differenza bit a bit

Operazioni aritmetiche - MS

- Come si esegue la **sottrazione** di due valori in Modulo e Segno?
- Confronto i **bit di segno** dei due numeri:

  a) Se i bit di segno sono **uguali**:

  - Il bit di segno risultante sarà uguale al bit di segno dell'operando a modulo maggiore
  - Il risultato avrà modulo pari al modulo della differenza dei moduli degli operandi

  b) Se i bit di segno sono **diversi**:

  - Il bit di segno risultante sarà uguale al bit di segno del minuendo
  - Il risultato avrà modulo pari alla somma dei moduli dei due operandi

- Osservazione: $A - B = A + (-B)$

Operazioni aritmetiche - MS

- Osservazioni:
  - Si può avere **overflow** solo quando:
    - si sommano due operandi con segno concorde
    - si sottraggono due operandi con segno discorde
  - L'overflow si verifica quando c'è un riporto dalla cifra più significativa del modulo, cioè non si è nella condizione di rappresentare il risultato ottenuto.
  - Nelle operazioni tra valori rappresentati in MS, gli operandi **devono** essere rappresentati con lo stesso numero di cifre (si aggiungono gli zeri necessari a sinistra del modulo, prima del bit di segno).

Operazioni aritmetiche – CA2

- Come si esegue la **somma** di due valori in CA2?

  ① Si esegue la somma su tutti i bit degli addendi, **segno compreso**

  ② Un eventuale **riporto** (*carry*) **oltre il bit di segno** (MSB) **viene scartato**

  ③ Nel caso gli operandi siano di **segno concorde** (entrambi positivi o entrambi negativi) occorre verificare la presenza o meno di overflow (il segno del risultato non è concorde con quello dei due addendi)

  - L'**overflow** non si presenta mai quando si sommano operandi di segno opposto.
  - L'**overflow** si presenta se:

    $(+A)+(+B)=-C$ oppure $(-A) + (-B) = +C$

  - Un altro modo per vedere se c'è overflow è guardare i riporti nelle ultime due posizioni più significative: se sono diversi c'è overflow.

Operazioni aritmetiche – CA2

- *Esempi di somma in CA2*:

$3 + (-8)$

```
 (+3)   0000 0011
+(-8)   1111 1000
-----------------
 (-5)   1111 1011
```

$-2 + (-5)$

```
 (-2)   1111 1110
+(-5)   1111 1011
-----------------
 (-7) 1 1111 1001 : scarto il carry
```

Operazioni aritmetiche – CA2

- La **sottrazione** tra due numeri in CA2 viene trasformata in somma applicando la regola

$$A - B = A + (-B)$$

ovvero

$$A - B = A + \mathrm{CA2}(B)$$

- Per assicurarsi della correttezza del risultato di un'operazione di somma in CA2 bisogna verificare l'assenza di overflow:
  - non si ha overflow se gli operandi hanno segno discorde
  - si ha overflow se gli operandi hanno segno concorde e il segno del risultato è discorde con essi

Operazioni aritmetiche – CA2

- *Esempio di sottrazione in CA2*:

```
 (+8) 0000 1000                    0000 1000
-(+5) 0000 0101 -> Complementa -> +1111 1011
-----                              -----------
 (+3)                             1 0000 0011 : scarto il carry
```

Operazioni aritmetiche – CA2

- Osservazioni:
  - Gli operandi devono essere rappresentati con lo stesso numero di bit
  - Nell'ipotesi di avere un valore X in CA2 su n bit (segno incluso) e di volerne ricavare la rappresentazione, sempre in CA2, su m bit ($m > n$), si attua l'*estensione del segno*:

    si replica l'MSB negli ($m - n$) bit più a sinistra

Operazioni aritmetiche – CA2

- *Esempio*:
  - Per i numeri positivi si aggiungono 0 nella parte più significativa

    +18 = 00010010

    +18 = 00000000 00010010

  - Per i numeri negativi si aggiungono 1 nella parte più significativa

    -18 = 101110

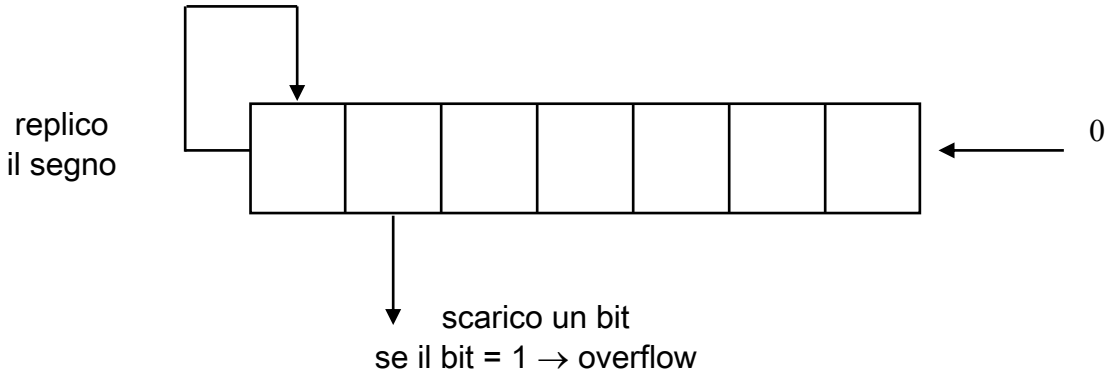
    -18 = 1111 101110

Operazioni aritmetiche

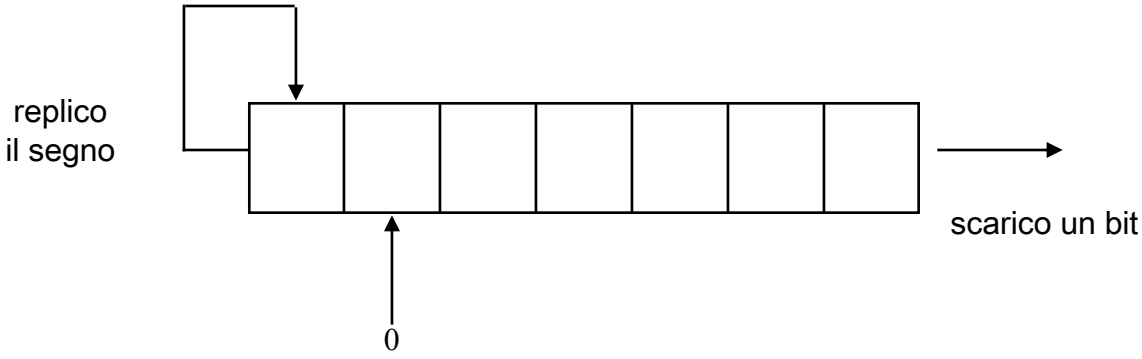
- Esiste un'ulteriore operazione detta shift:
  - Consiste nello spostare (shift) verso destra (right) o verso sinistra (left) la posizione delle cifre di un numero, espresso in una base qualsiasi, inserendo uno zero nelle posizioni lasciate libere.
  - Left : equivale a moltiplicare il numero per la base
  - Right : equivale a dividere il numero per la base

Shift - MS

- Left: equivale a moltiplicare il numero per la base

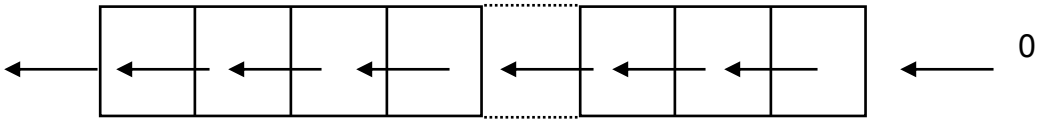


- Right: equivale a dividere il numero per la base



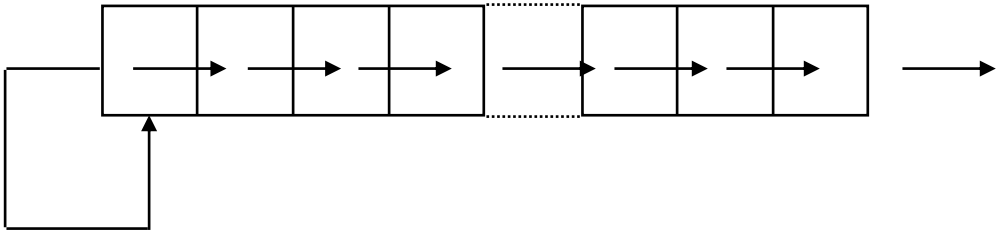
Shift – CA2

- Left: equivale a moltiplicare il numero per la base



se il nuovo MSB è diverso
dal precedente c'è overflow

- Right: equivale a dividere il numero per la base



Esercizio

0100 0010 (= +66) Arithmetic shift left by 1 (i.e. x2)

Rappresentazione Eccesso 2^{n-1}

- Nella rappresentazione **Eccesso 2^{n-1}** un numero X è rappresentato come segue:

$$X + 2^{n-1}$$

- Con n bit si rappresenta l'eccesso 2^{n-1}
- L'intervallo è asimmetrico (come CA2: $[- 2^{n-1} ,+ 2^{n-1} -1]$) e semplice rappresentazione dello zero!
- Regola pratica: I numeri in eccesso 2^{n-1} si ottengono da quelli in CA2 complementando il bit più significativo

Rappresentazione Eccesso 128

- Nella rappresentazione **Eccesso 128** (2^7) un numero X è rappresentato come segue:

  X + 128

- Esempio:

X = 5	5+128=133	$1000\ 0101_2$
X = -3	-3+128=125	$0111\ 1101_2$

- Passando da CA2:

$$-3_{10} \xrightarrow{3} 00000011_2 \xrightarrow{+3} 11111100_{CA2} \xrightarrow{CA2} 11111101_{CA2} \xrightarrow[\text{più significativo}]{\text{complemento il bit}} 01111101_{Eccesso128} \xleftarrow{-3}$$

- Osservazioni:
  - In Eccesso 128 i numeri da [-128, 127] si mappano su [0, 255]
  - In pratica si usa l'eccesso 127 (esponenti dei numeri in virgola mobile)

Da leggere

- Rappresentazione dei numeri relativi, appendice A del libro di A. S. Tanenbaum, "Structured computer Organization", 5th ed.