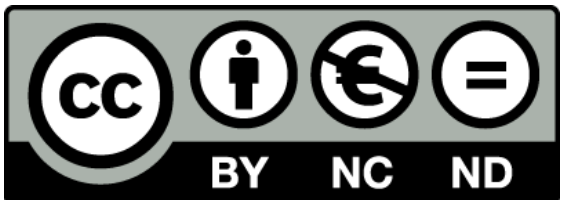
Laboratorio di Programmazione di Sistema

Operazioni su Bit

Luca Forlizzi, Ph.D.

Versione 23.1



Introduzione

- Le parole di un *ASM-PM*, sono gli insiemi di bit a cui la abstract machine può accedere mediante una singola operazione
- Nella maggior parte delle istruzioni *ASM*, il contenuto di una parola viene considerato come un singolo dato
- In alcuni casi, tuttavia, può essere vantaggioso immagazzinare un dato solo in alcuni dei bit di una parola, nonché utilizzare i bit di una singola parola per memorizzare più di un dato
- Ciò è reso possibile da operazioni, dette *operazioni su bit*, che agiscono sulle cifre binarie del contenuto di una parola considerandole dati indipendenti gli uni dagli altri

Introduzione

- Le istruzioni *ASM* che effettuano operazioni su bit, vengono chiamate *istruzioni su bit*
- Anche molti *HLL*, tra cui C Standard, permettono di eseguire tali operazioni
- In questa presentazione descriviamo le più comuni operazioni su bit, le loro principali applicazioni e introduciamo anche gli operatori di C Standard e le istruzioni di MIPS32 e MC68000 che permettono di eseguirle
- Per ulteriori informazioni ed esempi si rimanda a
  - sezione 20.1 di **[Ki]**
  - **[C99]**
  - **[MIPS32]**
  - **[M68000]**

Relazioni tra Bit

- Ricordiamo che ciascun formato di dato **F** assegna una posizione a ciascuno dei bit che formano una parola definita da **F**
- Ovvero a ciascun bit viene assegnato un numero compreso tra 0 e $L - 1$, dove L è la lunghezza di **F**
- Se si interpreta in codifica naturale il contenuto di una parola, il bit in posizione i memorizza la $i + 1$-esima cifra di un numero intero e quindi ha valore pari a 2^i
- Per questo motivo, sono detti più *significativi* i bit che hanno posizioni maggiori

Operazioni su Bit

- Le più comuni operazioni su bit, per le quali quasi tutti gli *ASM-PM* forniscono istruzioni, sono
  - Operazioni di spostamento di bit
    - Scorrimento aritmetico
    - Scorrimento logico
    - Rotazione
  - Operazioni logiche bit-a-bit
    - **not**
    - **and**
    - **or**
    - **xor**
- Alcuni *ASM-PM*, dispongono anche di istruzioni per effettuare operazioni su singoli bit di una parola
- Attraverso l'uso combinato di tali operazioni è anche possibile memorizzare e gestire dati in insiemi di bit che non sono parole dell'*ASM-PM*, chiamati *bit-field*

Scorrimento

- Un'operazione di *scorrimento* (*shift*) si applica ad una parola di L bit e consiste nel modificare le posizioni delle cifre binarie del contenuto della parola, ovvero di spostare le cifre dai bit in cui sono contenute in altri bit, come se fossero degli oggetti che scorrono all'interno di un tubo
- Le operazioni di *scorrimento aritmetico* effettuano lo scorrimento interpretando il contenuto di una parola come numero in complemento a 2
- Le operazioni di *scorrimento logico* invece modificano il contenuto di una parola senza adottare una particolare interpretazione di dato
- Un'operazione di *rotazione* (*rotate*) è simile ad un'operazione di scorrimento, ma sposta le cifre binarie come se esse fossero degli oggetti in un contenitore a forma di anello

Scorrimento Logico a Sinistra

- *Scorrimento logico a sinistra di* 1:
  - Per $0 < p \leq L - 1$, il bit di posizione p memorizza la cifra che, in precedenza, era memorizzata nel bit di posizione $p - 1$
  - Il bit di posizione 0 memorizza 0
- Si noti che la cifra memorizzata nel bit di posizione $L - 1$ prima dell'operazione viene perduta
- *Scorrimento logico a sinistra di* k: ripetere k volte uno scorrimento logico a sinistra di 1

Scorrimento Aritmetico a Sinistra

- *Scorrimento aritmetico a sinistra di* 1:
  - Per $0 < p \leq L - 1$, il bit di posizione p memorizza la cifra che, in precedenza, era memorizzata nel bit di posizione $p - 1$
  - Se la cifra in posizione $L - 1$ cambia, viene segnalato un *overflow*
  - Il bit di posizione 0 memorizza 0
- Si noti che la cifra memorizzata nel bit di posizione $L - 1$ prima dell'operazione viene perduta
- *Scorrimento aritmetico a sinistra di* k: ripetere k volte uno scorrimento aritmetico a sinistra di 1

Scorrimento Logico a Destra

- *Scorrimento logico a destra di* 1:
  - Per $0 \leq p < L - 1$, il bit di posizione p memorizza la cifra che, in precedenza, era memorizzata nel bit di posizione $p + 1$
  - Il bit di posizione $L - 1$ memorizza 0
- Si noti che la cifra memorizzata nel bit di posizione 0 prima dell'operazione viene perduta
- *Scorrimento logico a destra di* k: ripetere k volte uno scorrimento logico a destra di 1

Scorrimento Aritmetico a Destra

- *Scorrimento aritmetico a destra di* 1:
  - Per $0 \leq p < L - 1$, il bit di posizione p memorizza la cifra che, in precedenza, era memorizzata nel bit di posizione $p + 1$
  - Il bit di posizione $L - 1$ non viene modificato
- *Scorrimento aritmetico a destra di* k: ripetere k volte uno scorrimento aritmetico a destra di 1

Scorrimento e Aritmetica

- Una delle applicazioni principali delle operazioni di scorrimento è aritmetica
- Data una stringa di L bit, interpretata come numero senza segno N
  - effettuare lo scorrimento logico a sinistra di k sulla stringa, equivale a calcolare $(N \cdot 2^k) \mod 2^L$
  - effettuare lo scorrimento logico a destra di k sulla stringa, equivale a calcolare $N \div 2^k$

Scorrimento e Aritmetica

- Data una stringa di L bit, interpretata come numero con segno N
  - effettuare lo scorrimento aritmetico a sinistra di k sulla stringa, equivale a calcolare $N \cdot 2^k$
  - effettuare lo scorrimento aritmetico a destra di k sulla stringa, equivale a calcolare $N \div 2^k$
- Le operazioni di scorrimento hanno implementazioni hardware molto più efficienti delle operazioni di moltiplicazione e divisione

Principali Istruzioni/Operatori di Scorrimento

- In C
  - << scorrimento a sinistra, aritmetico o logico in base al tipo dell'operando sinistro (ma se l'operando sinistro ha tipo con segno il risultato è definito solo per valori non-negativi)
  - >> scorrimento logico a destra se il tipo dell'operando sinistro è senza segno, altrimenti il risultato è implementation defined
- In MIPS32
  - `sll` e `sllv` scorrimento logico a sinistra
  - `srl` e `srlv` scorrimento logico a destra
  - `sra` e `srav` scorrimento aritmetico a destra
- In MC68000
  - `asl` scorrimento aritmetico a sinistra
  - `lsl` scorrimento logico a sinistra
  - `asr` scorrimento aritmetico a destra
  - `lsr` scorrimento logico a destra

Rotazione a Sinistra

- *Rotazione a sinistra di* 1:
  - Per $0 < p \leq L - 1$, il bit di posizione p memorizza la cifra che, in precedenza, era memorizzata nel bit di posizione $p - 1$
  - Il bit in posizione 0 memorizza la cifra che, in precedenza, era memorizzata nel bit di posizione $L - 1$
- *Rotazione a sinistra di* k: ripetere k volte una rotazione a sinistra di 1
- Istruzioni di rotazione a sinistra in MC68000-ASM1: `rol`, `roxl`
- Istruzione di rotazione a sinistra in MIPS32-MARS: `rol`
- In C le rotazioni a sinistra vengono effettuate combinando operazioni logiche bit-a-bit e di *shift*

Rotazione a Destra

- *Rotazione a destra di* 1:
  - Per $0 \leq p < L - 1$, il bit di posizione p memorizza la cifra che, in precedenza, era memorizzata nel bit di posizione $p + 1$
  - Il bit in posizione $L - 1$ memorizza la cifra che, in precedenza, era memorizzata nel bit di posizione 0
- *Rotazione a destra di k*: ripetere k volte una rotazione a destra di 1
- Istruzioni di rotazione a destra in MC68000-ASM1: `ror`, `roxr`
- Istruzione di rotazione a destra in MIPS32-MARS: `ror`
- In C le rotazioni a destra vengono effettuate combinando operazioni logiche bit-a-bit e di *shift*

Operazioni Logiche *bit-a-bit*

- Come ovvio, una cifra binaria può essere interpretata come un valore booleano
- Convenzionalmente, il valore 1 viene interpretato come *vero* e il valore 0 come *falso*
- Una stringa binaria di n cifre può quindi essere interpretata come un insieme di n valori Booleani
- Con una tale interpretazione, ciascuna cifra di una stringa binaria è un valore Booleano del tutto indipendente da quello delle altre cifre
- Quindi, più in generale possiamo interpretare una stringa binaria di n cifre come un vettore di n valori Booleani

Operazioni Logiche *bit-a-bit*

- Gli *ASM-PM*, tipicamente, forniscono al programmatore la possibilità di interpretare il contenuto di una parola di N bit come un vettore di N valori Booleani, attraverso le operazioni *logiche bit-a-bit* (*bitwise operations*)
- Le operazioni logiche bit-a-bit effettuano una computazione indipendente su ciascuno dei bit dei loro operandi
  - Tutti i bit di un operando vengono coinvolti, ma l'esito della computazione su ogni singolo bit di un operando, non dipende dal contenuto degli altri bit dello stesso operando
  - Per tutti i bit di un operando viene effettuata la stessa computazione
  - Le computazioni effettuate sui diversi bit di un operando, essendo indipendenti le une dalle altre, sono svolte in parallelo
- Le più comuni operazioni logiche bit-a-bit effettuano operazioni Booleane

Operazioni Booleane

- Ricordiamo la definizione, tramite tabelle della verità, delle più comuni operazioni Booleane

X	*Not* X
0	1
1	0

X	Y	X *Or* Y	X *And* Y	X *Or-esclusivo* Y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Complemento

- La più semplice operazione bit-a-bit è il *Complemento*, detto anche *Not*
- Il *Not* ha un singolo operando O e produce un risultato R che ha lo stesso numero di bit di O
- Ciascun bit di R ha valore inverso di quello del bit che ha la stessa posizione nell'operando O
- Esempi operazione *Not*

O	Not O
0x00	0xFF
0x01	0xFE
0xF0	0x0F
0x5C	0xA3
0xFF	0x00

And

- L'operazione *And* bit-a-bit si applica a due operandi O_1 ed O_2 che hanno lo stesso numero di bit N e produce un risultato R formato da N bit
- Il bit in posizione k del risultato ha valore pari al risultato dell'operazione Booleana di *And* tra il bit in posizione k di O_1 e il bit in posizione k di O_2
- Esempi operazione *And*

O_1	O_2	O_1 And O_2
0x00	0xFF	0x00
0x59	0xAD	0x09

Or

- L'operazione *Or* bit-a-bit si applica a due operandi O_1 ed O_2 che hanno lo stesso numero di bit N e produce un risultato R formato da N bit
- Il bit in posizione k del risultato ha valore pari al risultato dell'operazione Booleana di *Or* tra il bit in posizione k di O_1 e il bit in posizione k di O_2
- Esempi operazione *Or*

O_1	O_2	O_1 Or O_2
0x0F	0xF0	0xFF
0x59	0xAD	0xFD

Or-esclusivo

- L'operazione *Or-esclusivo* bit-a-bit, detta brevemente *Xor* o anche *Eor*, si applica a due operandi O_1 ed O_2 che hanno lo stesso numero di bit N e produce un risultato R formato da N bit
- Il bit in posizione k del risultato ha valore pari al risultato dell'operazione Booleana di *Or-esclusivo* tra il bit in posizione k di O_1 e il bit in posizione k di O_2
- Esempi operazione *Or-esclusivo*

O_1	O_2	O_1 Xor O_2
0x3C	0x0F	0x33
0x59	0xAD	0xF4

Controllare e Modificare Bit con Operazioni Logiche

- Le operazioni logiche bit-a-bit permettono di controllare e di modificare individualmente i bit di una parola
- Data una cifra binaria x contenuta in un bit, l'operazione Booleana di *Or* ha la seguente proprietà
  - x *Or* $0 = x$
  - x *Or* $1 = 1$
- Quindi, data una parola D di N bit, è possibile porre a 1 determinati bit senza modificare gli altri, effettuando *Or* bit-a-bit tra il contenuto di D e una stringa di N bit, chiamata *maschera*, che ha:
  - cifre di valore 1, in corrispondenza delle posizioni dei bit di D che si vuole porre a 1
  - cifre di valore 0, in corrispondenza delle posizioni dei bit di D che non si vuole modificare

Controllare e Modificare Bit con Operazioni Logiche

- Esempi: maschere per porre a 1 alcuni bit di parole di 8 bit

Posizioni	Maschera
0	0x01
1	0x02
4	0x10
7	0x80
1 e 4	0x12
0, 1 e 4	0x13
1, 3, 6, 7	0xCA

Controllare e Modificare Bit con Operazioni Logiche

- Data una cifra binaria x contenuta in un bit, l'operazione Booleana di *And* ha la seguente proprietà
  - x *And* $0 = 0$
  - x *And* $1 = x$
- Data una parola D di N bit, è possibile controllare se il bit di posizione k memorizza 0 oppure 1, attraverso il risultato di *And* bit-a-bit tra D e una stringa di N bit, chiamata *maschera*, che ha:
  - la cifra in posizione k al valore 1
  - cifre di valore 0 nelle posizioni diverse da k

Controllare e Modificare Bit con Operazioni Logiche

- Data una parola D di N bit, è possibile porre a 0 determinati bit senza modificare gli altri, effettuando *And* bit-a-bit tra D e una stringa di N bit, chiamata *maschera*, che ha:
  - cifre di valore 0, in corrispondenza delle posizioni dei bit di D che si vuole porre a 0
  - cifre di valore 1, in corrispondenza delle posizioni dei bit di D che non si vuole modificare
- Esempi: maschere per porre a 0 alcuni bit di parole di 8 bit

Posizioni	Maschera
0	0xFE
1	0xFD
4	0xEF
7	0x7F
1 e 4	0xED
1, 3, 6, 7	0x35

Controllare e Modificare Bit con Operazioni Logiche

- Data una cifra binaria x contenuta in un bit, l'operazione Booleana di *Or-esclusivo* ha la seguente proprietà
  - x *Xor* $0 = x$
  - x *Xor* $1 =$ inverso di x
- Quindi, data una parola D di N bit, è possibile invertire determinati bit senza modificare gli altri, effettuando *Xor* bit-a-bit tra D e una stringa di N bit, chiamata *maschera*, che ha:
  - cifre di valore 1, in corrispondenza delle posizioni dei bit di D che si vuole invertire
  - cifre di valore 0, in corrispondenza delle posizioni dei bit di D che non si vuole modificare

Controllare e Modificare Bit con Operazioni Logiche

- Esempi: maschere per invertire alcuni bit di parole di 8 bit

Posizioni	Maschera
0	0x01
1	0x02
4	0x10
7	0x80
1 e 4	0x12
0, 1 e 4	0x13
1, 3, 6, 7	0xCA

Principali Istruzioni/Operatori bit-a-bit

- La tabella seguente elenca gli operatori e le istruzioni che effettuano operazioni logiche bit-a-bit in C, MIPS32 e MC68000

Operazione	C	MIPS32	MC68000
Not	`~`	`not`	`not`
Or	`|`	`or`	`or`
And	`&`	`and`	`and`
Xor	`^`	`xor`	`eor`

Or-esclusivo e Crittografia

- L'operazione di *Or-esclusivo* bit-a-bit trova interessanti applicazioni nel campo della crittografia
- Esse si basano sulla seguente proprietà dell'*Or-esclusivo*: date due parole X e M formate da un ugual numero di bit, si ha che $(X\ Xor\ M)\ Xor\ M = X$
- Uno dei modi più semplici per crittografare un messaggio, è quello di fare *Or esclusivo* tra ciascun carattere del messaggio e un valore segreto detto *chiave*
- La decrittazione avviene effettuando un *Or-esclusivo* tra ciascun carattere del messaggio crittato e la *chiave*
- La sezione 20.1 di **[Ki]** mostra un esempio completo
- Anche le tecniche crittografiche *One-time pad* utilizzano spesso l'*Or-esclusivo*

Operazioni su Singolo Bit

- Come abbiamo visto, è possibile leggere o modificare il contenuto di un singolo bit mediante operazioni logiche bit-a-bit
- Alcuni *ASM-PM* dispongono di istruzioni dedicate per controllare o per manipolare il contenuto di un singolo bit
- Per operare su un singolo bit, tali istruzioni possono essere più efficienti e/o più comode da usare rispetto a istruzioni bit-a-bit

Operazioni su Singolo Bit

- MC68000 dispone di quattro *istruzioni su singolo bit*, che agiscono sul bit b di posizione k di una parola P
  - `btst` interpreta il contenuto di b come valore Booleano e pone la condizione *Equal* a tale valore
  - `bclr` memorizza 0 in b
  - `bchg` inverte il contenuto di b
  - `bset` memorizza 1 in b
- Tali istruzioni hanno 2 operandi
  - Il primo operando contiene k ed ammette l'indirizzamento immediato o diretto-registro ad un registro dati
  - Il secondo operando è la parola P, può essere un registro dati oppure una parola di memoria

Operazioni su Singolo Bit

- L'esistenza delle istruzioni su singolo bit implica che, in MC68000, ogni singolo bit è una parola e quindi che MC68000 ha un formato di dato di lunghezza 1, chiamato `bit`
- Il formato `bit`
  - contiene sia parole di registro, che sono i singoli bit di tutti i registri dati, sia parole di memoria,
  - non è un formato generale, in quanto è utilizzato solo dalle 4 istruzioni su singolo bit
- Si noti che mentre gli identificativi delle parole di registro dei formati generali sono i nomi dei registri, gli identificativi delle parole di registro di `bit` sono stringhe formate dal nome di un registro *e* dalla posizione del bit all'interno del registro
- Questa osservazione è significativa dal punto di vista teorico, perché si riflette nella importante definizione di *Formato Byte*, che introdurremo in future presentazioni

Bit-field

- Come abbiamo anticipato nell'introduzione, in alcune applicazioni può essere vantaggioso immagazzinare un dato solo in alcuni dei bit di una parola e utilizzare i bit di una singola parola per memorizzare più di un dato
- Un insieme di bit che non forma necessariamente una parola ma viene usato per memorizzare un unico dato, viene chiamato *bit-field*
- Non c'è una definizione generale più precisa di bit-field: in contesti diversi i bit-field vengono definiti in modi diversi
- Tipicamente i bit-field sono formati da bit che appartengono ad una stessa parola e che hanno posizioni "vicine", oppure a parole di memoria che sono tra loro "vicine"

Bit-field

- Per poter effettuare operazioni aritmetiche o logiche con dati contenuti in un bit-field, è di solito necessario che tali dati vengano copiati in registri o parole di memoria accessibili dalle istruzioni aritmetico-logiche
- In alcuni *ASM-PM* vi sono formati di dato le cui parole sono bit-field
  - In tali *ASM-PM*, ciò che distingue i bit-field dalle altre parole è il fatto che ai bit-field si può accedere solo con istruzioni dedicate che si limitano a operazioni di trasferimento dati
  - Dunque i formati di dato bit-field non sono formati generali

Bit-field

- Di solito, quindi, su un bit-field vengono fatte solo due tipi di operazioni
  - *estrazione*: copia del contenuto di un bit-field in una parola
  - *memorizzazione*: copia del contenuto di una parola in un bit-field
- Negli *ASM-PM* che dispongono di formati di dato per i bit-field ci sono, in genere, istruzioni che eseguono tali operazioni
- Negli altri *ASM-PM*, estrazione e memorizzazione di un bit-field vengono effettuate mediante operazioni su bit

Bit-field

- In questa presentazione ci restringiamo al caso, piuttosto comune e semplice, di bit-field formati da bit che hanno posizioni consecutive in una stessa parola di registro in *ASM-PM* che non dispongono di formati di dato bit-field
- Mostriamo quindi come realizzare, mediante operazioni su bit, estrazione e memorizzazione di bit-field contenuti in parole di registro
- La sezione 20.1 di **[Ki]** fornisce ulteriori dettagli ed esempi
- In future presentazioni, dedicate all'organizzazione dei dati in memoria, faremo ulteriori considerazioni sui formati di dato bit-field presenti in alcuni *ASM-PM*

Estrazione da Bit-field

- L'estrazione da un bit-field è particolarmente semplice quando esso è costituito dai bit di posizione compresa tra 0 e un certo valore k minore della lunghezza della parola di registro che lo contiene
- In questi casi, per estrarre il valore dal bit-field è sufficiente azzerare i bit della parola che non fanno parte del bit-field
- Nel caso di un bit-field formato dai bit di una parola di registro che hanno posizioni comprese tra k_1 e k_2, con $k_1 > 0$, si può effettuare un'operazione di scorrimento che trasferisce il contenuto del bit-field nei bit che hanno posizioni comprese tra 0 e $k_2 - k_1$, e poi procedere come nel caso precedente

Estrazione da Bit-field

- Esempio: estrazione del bit-field costituito dai bit di posizione compresa tra 0 e 3 della parola S (di 8 bit), nella parola D (di 8 bit)
  - In C (`S` e `D` sono variabili di un tipo intero senza segno):

    ```
    D = S & 0x0F;
    ```

  - In *ASM* MIPS32 (S è il registro `$t0`, D è `$t1`):

    ```
    and     $t1,$t0,0x0F
    ```

  - In *ASM* MC68000 (S è il registro `d0`, D è `d1`):

    ```
    move.b  d0,d1
    and.b   #$0F,d1
    ```

Estrazione da Bit-field

- Esempio: estrazione del bit-field costituito dai bit di posizione compresa tra 2 e 6 della parola S (di 8 bit), nella parola D (di 8 bit)
  - In C (`S` e `D` sono variabili di un tipo intero senza segno):

    ```
    D = ( S >> 2 ) & 0x1F;
    ```

  - In *ASM* MIPS32 (S è `$t0`, D è `$t1`):

    ```
    srl     $t1,$t0,2
    and     $t1,$t1,0x1F
    ```

  - In *ASM* MC68000 (S è `d0`, D è `d1`):

    ```
    move.b  d0,d1
    lsr.b   #2,d1
    and.b   #$1F,d1
    ```

Memorizzazione in Bit-field

- Per memorizzare un valore in un bit-field, tipicamente si procede prima azzerando tutti i bit che compongono il bit-field e poi ponendo a 1 i soli bit necessari
- L'azzeramento viene effettuato tramite *And* bit-a-bit, e la scrittura dei valori 1 tramite *Or* bit-a-bit
- Quando il valore da memorizzare è una variabile, le maschere devono essere generate dinamicamente

Memorizzazione in Bit-field

- Esempio: memorizzazione del valore binario 101 nel bit-field costituito dai bit di posizione compresa tra 11 e 13 della parola D (di 16 bit)
  - In C (`D` è una variabile di un tipo intero senza segno):

```
D = D & ~0x3800 | 0x2800;
```

  - In *ASM* MIPS32 (D è `$t1`):

```
and     $t1,$t1,0xC7FF
or      $t1,$t1,0x2800
```

  - In *ASM* MC68000 (D è `d1`):

```
and.w   #$C7FF,d1
or.w    #$2800,d1
```

Memorizzazione in Bit-field

- Esempio: memorizzazione del contenuto della parola S (di 8 bit) nel bit-field costituito dai bit di posizione compresa tra 3 e 10 della parola D (di 16 bit)
  - In C (`S` e `D` sono variabili di un tipo intero senza segno):

    ```
    D = D & ~0x07F8 | ( S << 3 );
    ```

  - In *ASM* MIPS32 (S è `$t0`, D è `$t1`):

    ```
    and     $t1,$t1,0xF807
    sll     $t2,$t0,3
    or      $t1,$t2,$t1
    ```

  - In *ASM* MC68000 (S è `d0`, D è `d1`):

    ```
    and.w   #$F807,d1
    clr.w   d2
    move.b  d0,d2
    lsl.w   #3,d2
    or.w    d2,d1
    ```