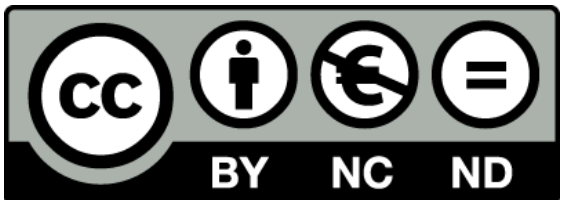
Laboratorio di Programmazione di Sistema

Interpretazione dei Dati nei Linguaggi Assembly

Luca Forlizzi, Ph.D.

Versione 23.1



Luca Forlizzi, 2023

© 2023 by Luca Forlizzi. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode.

Interpretazioni di Dato

- Come già sappiamo, negli *ASM* gli aspetti sintattici e quelli semantici dei dati non sono integrati in un unico concetto
  - I formati di dato definiscono gli aspetti sintattici
  - Le interpretazioni di dato definiscono gli aspetti semantici
- Mentre il formato può essere considerato una proprietà intrinseca di un dato (un dato è contenuto in una parola, le cui caratteristiche sono definite dal formato), l'interpretazione del dato dipende dall'istruzione che vi accede
- Ovvero le interpretazioni di dato vengono applicate dalle istruzioni ai dati su cui esse operano

Interpretazioni di Dato

- Ad esempio, un'ipotetico *ASM-PM* potrebbe avere
  - un formato **F** che definisce parole di una certa lunghezza L
  - un'istruzione **SOMMA-I** che, applicando un'interpretazione per numeri interi, calcola la somma tra i contenuti di due parole di formato **F** e la memorizza in una parola di formato **F**
  - un'istruzione **SOMMA-F** che, applicando un'interpretazione per numeri in virgola mobile, calcola la somma tra i contenuti di due parole di formato **F** e la memorizza in una parola di formato **F**
- Pur effettuando entrambe un'operazione di somma, **SOMMA-I** e **SOMMA-F**, applicate agli stessi operandi, produrrebbero in generale risultati del tutto diversi

Interpretazioni di Dato

- Un'interpretazione di dato è una funzione, calcolata da un algoritmo, che ha come dominio l'insieme delle stringhe binarie di una determinata lunghezza L e come codominio un insieme ben definito in senso matematico
- Se un insieme è codominio di una interpretazione di dato, si dice che tale insieme è *rappresentato* da quella interpretazione di dato

Interpretazioni di Dato

- Ogni istruzione di un *ASM-PM*, per ciascun operando applica una interpretazione di dato
- Concettualmente, attraverso le interpretazioni di dato, l'istruzione
  - Legge i contenuti degli operandi sorgente e li trasforma in elementi degli insiemi matematici che sono codomini delle interpretazioni dato applicate a ciascun operando
  - Esegue un'operazione matematica sugli elementi ottenuti
  - Trasforma i risultati dell'operazione in stringhe binarie e le memorizza negli operandi destinazione
- Nell'esempio precedente, **SOMMA-I** e **SOMMA-F**, applicando interpretazioni di dato diverse, otterrebbero addendi e, quindi risultati, diversi

Interpretazioni di Dato

- Tecnicamente, un'istruzione adotta, per ogni operando, un'interpretazione di dato su stringhe binarie di una determinata lunghezza L che corrisponde alla lunghezza del formato di dato dell'operando
- Quindi per ogni diverso formato di dato sono definite interpretazioni di dato diverse
- Tuttavia spesso per formati diversi sono possibili interpretazioni simili
- Ad esempio, se in un *ASM-PM* un formato F_1 di lunghezza k e un formato F_2 di lunghezza $2k$ hanno entrambi un'interpretazione come numeri interi, è possibile che le due interpretazioni differiscano solo per il fatto che quella relativa ad F_2 ha come dominio stringhe più lunghe e come codominio un insieme più ampio di numeri interi

Interpretazioni di Dato

- Le interpretazioni di dato presenti nella maggior parte degli *ASM-PM*, permettono di rappresentare
  - Numeri interi con una quantità fissata di cifre
  - Vettori di valori booleani, con cardinalità fissata
  - Indirizzi di memoria
  - Numeri floating point
- Le interpretazioni di dato per numeri floating point non sono trattate in LPS
- In questa presentazione vengono discusse interpretazioni di dato per numeri interi con una quantità fissata di cifre
- Si mostrerà come tali interpretazioni hanno effetto sulla valutazione delle condizioni e sul controllo dell'esecuzione

Interpretazioni di Dato

- Gli *ASM-PM*, tipicamente, forniscono al programmatore la possibilità di usare (almeno) due possibili interpretazioni di dato per numeri interi
  - Una *codifica* (o *rappresentazione*) *senza segno* (*unsigned*), ovvero un'interpretazione che consente di rappresentare intervalli di interi non-negativi, mediante stringhe binarie
  - Una *codifica* (o *rappresentazione*) *con segno* (*signed*), ovvero un'interpretazione che consente di rappresentare intervalli di interi sia negativi che non-negativi, mediante stringhe binarie

Codifica Naturale

- La codifica senza segno di gran lunga più utilizzata è la *codifica naturale* (o *rappresentazione naturale*), in quanto semplice e immediata
- Sia **F** un formato di lunghezza n
- Il contenuto c di ogni parola di **F** è una stringa binaria di lunghezza n, indicata con $c_{n-1}c_{n-2}\ldots c_1c_0$ dove $c_i \in \{0,1\}$ per $0 \leq i \leq n-1$
- La codifica naturale per stringhe binarie di lunghezza n, è una funzione ValNat_n che associa ad ogni stringa binaria $c_{n-1}c_{n-2}\ldots c_1c_0$ un intero non negativo definito come segue

$$\mathrm{ValNat}_n(c_{n-1}c_{n-2}\ldots c_1c_0) = \sum_{i=0}^{n-1} c_i 2^i$$

Codifica Naturale

- Le cifre binarie che hanno posizioni maggiori vengono dette più *significative* di quelle che hanno posizioni minori, proprio perché la codifica naturale attribuisce loro un valore maggiore
- La codifica naturale per stringhe di `n` cifre binarie consente di rappresentare interi compresi tra 0 e $2^n - 1$
- Sia P una parola appartenente ad un formato **F** di lunghezza n
  - Si definisce *valore in codifica naturale* di P (in un certo istante di tempo) il valore associato da ValNat_n al contenuto di P (in tale istante di tempo)
  - I bit che contengono le cifre binarie di posizioni maggiori vengono detti più *significativi* di quelli che hanno posizioni minori

Codifiche con Segno

- Sono state ideate e utilizzate diverse codifiche con segno
- C Standard, ad esempio, prescrive che i tipi di dato interi che possono assumere anche valori negativi, debbano essere rappresentati mediante una delle seguenti
  - Modulo e segno
  - Complemento a 1
  - Complemento a 2
- Ogni implementazione di C Standard deve scegliere una di tali interpretazioni, e documentare la scelta fatta (implementation-defined behavior)

Codifiche con Segno

- La codifica modulo e segno è nota per la sua semplicità concettuale, ma non è mai stata molto utilizzata
- La codifica in complemento a 1 è invece stata utilizzata in diverse architetture di supercomputer Cray, perché consentiva di ottimizzare al massimo le unità aritmetico-logiche della CPU
- Praticamente tutte le architetture attualmente in uso utilizzano la codifica in complemento a 2
- Nel resto della presentazione, pertanto ci concentriamo su *ASM-PM* che adottano il complemento a 2 come codifica con segno

Codifica in Complemento a 2

- Sia $c = c_{n-1}c_{n-2}\ldots c_1c_0$ una stringa binaria di lunghezza n, dove $c_i \in \{0,1\}$ per $0 \leq i \leq n-1$
- La *codifica in Complemento a 2* per stringhe binarie di lunghezza n, è una funzione $\mathrm{ValC2}_n$ che associa a c il valore intero definito come segue

$$\mathrm{ValC2}_n(c_{n-1}c_{n-2}\ldots c_1c_0) = -c_{n-1}2^{n-1} + \sum_{i=0}^{n-2} c_i 2^i$$

- Si noti che la cifra di posizione $n-1$, determina il segno del numero: se c_{n-1} è pari a 1, il numero è negativo, in caso contrario il numero è maggiore o uguale a 0; per tale motivo c_{n-1} viene chiamata *cifra di segno*

Codifica in Complemento a 2

- Se c è una stringa binaria di lunghezza n e $c_{n-1} = 0$, allora $\mathrm{ValNat}_n(c) = \mathrm{ValC2}_n(c)$, ovvero le stringhe binarie di lunghezza n la cui cifra più significativa è 0, vengono interpretate allo stesso modo dalla codifica naturale e dalla codifica in complemento a 2
- La codifica in complemento a 2 per stringhe binarie di lunghezza n, consente di rappresentare interi compresi tra -2^{n-1} e $2^{n-1} - 1$
- Sia P una parola appartenente ad un formato **F** di lunghezza n
  - Si definisce *valore in complemento a 2* di P (in un certo istante di tempo) il valore associato da $\mathrm{ValC2}_n$ al contenuto di P (in tale istante di tempo)
  - Il bit di P che contiene la cifra di segno viene detto *bit di segno*

Codifica in Complemento a 2

- Il complemento a 2 viene preferito ad altre codifiche con segno, in quanto ha le seguenti utili proprietà
  - Rappresentazione unica del numero 0
  - I comuni algoritmi per somma e differenze di interi in codifica naturale, possono essere usati anche per effettuare somma e differenza di due interi in complemento a 2
  - Esempio

Valore binario	Valore in codifica naturale	Valore in complemento a 2	
0101	5	5	+
1001	9	-7	=
1110	14	-2	

Codifica in Complemento a 2

- Il fatto che il comune algoritmo per la somma di interi in codifica naturale, permetta anche di calcolare la somma di due interi in complemento a 2, consente di utilizzare un'unica rete combinatoria per eseguire sia la somma tra interi in codifica naturale che quella tra interi in complemento a 2
- Ciò si riflette, ai livelli 2, 3 e 4, nel fatto che un'unica istruzione di addizione permette di eseguire sia la somma tra interi in codifica naturale che quella tra interi in complemento a 2
- Situazione analoga si verifica per la differenza

Codifica in Complemento a 2

- Gli *ASM-PM*, dunque, offrono al programmatore la possibilità di interpretare una stringa binaria sia come intero senza segno in codifica naturale, sia come intero con segno, mediante complemento a 2
- Nel seguito della presentazione, analizziamo in che modo il programmatore può scegliere l'interpretazione desiderata e come può operare con essa in modo corretto, in relazione alle principali operazioni su cui la scelta dell'interpretazione ha un impatto

Segno di un Intero

- Quasi tutti gli *ASM-PM* permettono di prendere decisioni su come proseguire l'esecuzione del programma sulla base del valore del segno di un intero
- Il segno di un valore intero coincide con il valore della cifra di segno: se essa vale 1 allora il numero è negativo, altrimenti è non-negativo
- Gli *ASM-PM* che adottano la gestione separata delle decisioni, hanno tipicamente istruzioni che registrano nei CCB il segno di un determinato operando o di un risultato di un'operazione e durante l'esecuzione del programma
  - Se l'istruzione che memorizza il segno di un operando o di un risultato eseguita più di recente, ha memorizzato segno positivo, allora la condizione *Plus* risulta vera e la condizione *Minus* risulta falsa
  - Altrimenti, *Plus* risulta falsa e *Minus* risulta vera

Addizione e Sottrazione

- Come abbiamo già sottolineato, adottando il complemento a 2 come codifica con segno, si ha l'importante vantaggio che la medesima istruzione consente di calcolare in modo corretto la somma di due dati, sia interpretandoli in codifica con segno che in codifica senza segno
- Situazione analoga per la sottrazione
- Quindi il programmatore non deve fare nulla per scegliere l'interpretazione da usare: addizione e sottrazione usano, al tempo stesso, entrambe le interpretazioni
- Tuttavia, in base alla codifica scelta, cambia la gestione delle situazioni in cui una operazione di addizione o sottrazione produce un risultato non correttamente rappresentabile

Addizione e Sottrazione

- Descriviamo cosa accade con riferimento all'addizione; per quanto riguarda la sottrazione vale un discorso simile
- L'addizione di due interi senza segno formati da `LEN` cifre può produrre un risultato di `LEN` $+ 1$ cifre
- Esempio: addizione ordinaria con `LEN` pari a 4

Valore binario	Valore in codifica naturale	
0110	6	$+$
1011	11	$=$
10001	17	

Addizione e Sottrazione

- Tipicamente, gli *ASM-PM* effettuano addizioni tra parole che hanno lo stesso formato di dato e producono un risultato in una parola che ha lo stesso formato degli operandi
- Dunque, se `LEN` è la lunghezza del formato degli operandi, può accadere che il risultato di un'addizione abbia $\texttt{LEN} + 1$ cifre, ma che la parola in cui memorizzare tale risultato abbia solo `LEN` bit: pertanto la cifra più significativa del risultato dell'addizione non viene memorizzata
- Esempio: addizione su un *ASM-PM* con `LEN` pari a 4

Valore binario	Valore in codifica naturale	
0110	6	+
1011	11	=
0001	1	

Addizione e Sottrazione

- Il caso in cui la somma tra due interi in codifica naturale produce un risultato maggiore del massimo intero in codifica naturale rappresentabile, viene tipicamente usato per illustrare il concetto di *overflow*: il risultato di un'operazione è troppo grande per essere rappresentato in modo esatto
- In molte architetture reali, tuttavia, questo caso non viene considerato una situazione di *overflow*

Addizione e Sottrazione

- Vi è infatti un'altro modo di considerare la situazione: l'operazione di addizione effettuata dalla abstract machine non è in realtà l'ordinaria addizione, in quanto l'eventuale riporto generato dalle cifre più significative dei due operandi non viene considerato parte del risultato
- Ciò vuol dire che il risultato dell'addizione è la rappresentazione delle `LEN` cifre meno significative della somma dei due operandi
- In altre parole, indicando con v_1 e v_2 i valori dei due operandi, l'addizione effettuata dalla abstract machine calcola, in modo esatto, l'espressione matematica $(v_1 + v_2) \mod 2^{\texttt{LEN}}$, ovvero l'addizione modulo $2^{\texttt{LEN}}$

Addizione e Sottrazione

- Negli *ASM-PM* che memorizzano informazioni di stato nei CCB, una delle informazioni tipicamente memorizzata è il verificarsi o meno di un riporto in uscita dalle cifre più significative degli addendi, a seguito di un'operazione di somma
- Se, nell'operazione di somma eseguita più di recente, le cifre più significative dei due addendi hanno generato un riporto (il che implica che il risultato, interpretato come numero senza segno, non può essere contenuto nell'operando destinazione), la condizione *Carry Set* risulta vera e la condizione *Carry Clear* risulta falsa
- Altrimenti, *Carry Set* risulta falsa e *Carry Clear* risulta vera

Addizione e Sottrazione

- Consideriamo ora cosa accade interpretando addendi e somma mediante la codifica in complemento a 2
- Nel caso di addizione tra valori dello stesso segno, può accadere che la `LEN`-esima cifra della somma abbia valore opposto a quello delle `LEN`-esime cifre dei due addendi
- Esempio con `LEN` pari a 4, i due addendi sono positivi ma la somma è negativa

Valore binario	Valore in complemento a 2	
0110	6	+
0100	4	=
1010	-6	

Addizione e Sottrazione

- La situazione in cui il risultato di un'addizione ha segno diverso da quello degli addendi viene considerata un *overflow* in quanto il risultato così ottenuto è scarsamente significativo
- Negli *ASM-PM* che memorizzano condizioni nei CCB, viene tipicamente memorizzato il verificarsi o meno di un overflow, a seguito di un'addizione
- Se, nell'addizione eseguita più di recente si è verificato un *overflow* (il che implica che il risultato, interpretato come numero con segno, ha un segno diverso da quello atteso), la condizione *Overflow Set* risulta vera mentre *Overflow Clear* risulta falsa
- Altrimenti, *Overflow Set* risulta falsa e *Overflow Clear* risulta vera
- Come vedremo, comunque, nel modo di gestire gli overflow vi sono differenze significative un'architettura e l'altra

Addizione e Sottrazione

- I seguenti esempi di addizioni, con operandi e risultato memorizzati in 4 bit, mostrano i valori assunti dalle condizioni *Carry Set*, *Overflow Set* in diverse situazioni
- Addendi di segno opposto

Valore binario	Valore in codifica naturale	Valore in complemento a 2	*Carry Set*	*Overflow Set*	
0110	6	6			+
1011	11	-5			=
0001	1	1	*true*	*false*	

Addizione e Sottrazione

- Addendi positivi

Valore binario	Valore in codifica naturale	Valore in complemento a 2	*Carry Set*	*Overflow Set*	
0110	6	6			+
0100	4	4			=
1010	10	-6	*false*	*true*	

- Addendi negativi

Valore binario	Valore in codifica naturale	Valore in complemento a 2	*Carry Set*	*Overflow Set*	
1100	12	-4			+
1011	11	-5			=
0111	7	7	*true*	*true*	

Confronto tra Interi

- Consideriamo i due numeri interi (di 16 cifre binarie) 0x6000 e 0x9000: quale dei due è minore dell'altro?

Confronto tra Interi

- Consideriamo i due numeri interi (di 16 cifre binarie) 0x6000 e 0x9000: quale dei due è minore dell'altro?
- Dipende dall'interpretazione di dato che si sceglie!
  - scegliendo la codifica naturale, 0x6000 è minore di 0x9000
  - scegliendo il complemento a 2, è invece il contrario, in quanto 0x6000 è positivo mentre 0x9000 è negativo

Confronto tra Interi

- Un confronto tra due interi permette di determinare informazioni relative a di due tipi di relazioni tra i due numeri
  - *Relazioni di uguaglianza*, ovvero le relazioni "uguale a" e "diverso da"
  - *Relazioni d'ordine*, ovvero le relazioni "minore di", "minore o uguale di", "maggiore di", "maggiore o uguale di"
- Negli *ASM-PM* che memorizzano informazioni nei CCB, molte istruzioni effettuano un confronto tra due numeri e memorizzano nei CCB informazioni che permettono di stabilire quali di tali relazioni sono vere, e quali false

Confronto tra Interi

- Sia in codifica naturale che in complemento a 2, due stringhe binarie rappresentano lo stesso numero se e solo se tutte le cifre dell'una sono uguali alle corrispondenti cifre dell'altra
- Pertanto le informazioni relative alle relazioni di uguaglianza sono indipendenti dalle interpretazioni di dato
- Se il confronto tra due valori eseguito più di recente ha rilevato che i due valori sono uguali, la condizione *Equal* risulta vera mentre *Not Equal* risulta falsa
- Altrimenti, *Equal* risulta falsa e *Not Equal* risulta vera

Confronto tra Interi

- Invece, come l'esempio precedente mostra, le informazioni relative alle relazioni d'ordine sono diverse per diverse interpretazioni di dato
- Dunque, negli *ASM* ci sono 8 diverse condizioni relative alle relazioni d'ordine: ciascuna condizione è il valore di una delle 4 relazioni d'ordine in base a una delle due possibili interpretazioni di dato per interi
- La seguente tabella ne descrive i nomi

	codifica con segno	**codifica senza segno**
"minore di"	*Less Than*	*Lower Than*
"minore o uguale di"	*Less Than or Equal*	*Lower Than or Same*
"maggiore di"	*Greater Than*	*Higher Than*
"maggiore o uguale di"	*Greater Than or Equal*	*Higher Than or Same*

Confronto tra Interi

- Negli *ASM-PM* che adottano la gestione separata delle decisioni, le istruzioni di memorizzazione di condizione memorizzano nei CCB informazioni sufficienti a stabilire la verità o falsità di tutte e 8 le condizioni descritte dalla precedente tabella
- Quindi, la scelta di quale istruzione di confronto utilizzare non dipende dalla interpretazione di dato che si vuole adottare
- L'esito di una istruzione condizionata, invece, è conseguenza del valore di una sola delle 8 condizioni
- Pertanto, la scelta dell'istruzione condizionata da utilizzare, dipende dall'interpretazione di dato che si vuole adottare

Confronto tra Interi

- Ad esempio, se si vogliono confrontare due interi v_1 e v_2 e si vuole effettuare un salto condizionato se $v_1 < v_2$ interpretando i due valori come numeri con segno, si utilizzerà un'istruzione di confronto seguita da un'istruzione di salto condizionata da *Less Than*
- Se si decidesse di cambiare interpretazione di dato, usando la codifica senza segno, l'istruzione di confronto rimarrebbe la stessa, mentre sarebbe necessario cambiare l'istruzione di salto condizionato con una che dipenda da *Lower Than*

Moltiplicazione e Divisione tra Interi

- Il prodotto tra due interi di `LEN` cifre può richiedere fino a $2 \cdot$ `LEN` cifre, sia interpretando mediante codifica naturale che mediante complemento a 2
- In genere, quindi, il risultato della moltiplicazione viene memorizzato usando un formato di lunghezza $2 \cdot$ `LEN` oppure 2 parole che hanno un formato di lunghezza `LEN`
- La divisione tra due interi di `LEN` cifre, può generare un quoziente ed un resto, ciascuno di al più `LEN` cifre, sia interpretando mediante codifica naturale che mediante complemento a 2
- In genere, essi vengono memorizzati usando un formato di lunghezza $2 \cdot$ `LEN` oppure 2 parole che hanno un formato di lunghezza `LEN`

Moltiplicazione e Divisione tra Interi

- Moltiplicazione e divisione tra interi con segno, richiedono algoritmi diversi da quelli per effettuare moltiplicazione e divisione tra interi senza segno
- Dunque, i linguaggi *ASM* (così come i corrispondenti *LM*) hanno istruzioni diverse per la moltiplicazione tra numeri con segno e per la moltiplicazione tra numeri senza segno
- In modo analogo, gli *ASM* hanno istruzioni diverse per la divisione tra numeri con segno e per la divisione tra numeri senza segno

Codifiche di interi in MC68000

- In MC68000, i dati possono essere interpretati come interi senza segno oppure come interi con segno codificati in complemento a 2
- Entrambe le interpretazioni di dato possono essere applicate a tutti e 3 i formati di dato generali

Istruzioni di Determinazione di Condizioni

- MC68000 adotta la gestione separata delle decisioni e dispone di numerose istruzioni di determinazione di condizioni, che memorizzano in `ccr` quali condizioni sono vere e quali false
  - *Plus* e *Minus* vengono determinate da molte istruzioni in base al segno di un risultato o di un determinato operando
  - *Carry Set*, *Overflow Set* e le loro opposte, vengono determinate dalle istruzioni aritmetiche e da altre istruzioni
  - Le condizioni relative a relazioni di uguaglianza e a relazioni d'ordine, vengono determinate da istruzioni che confrontano due operandi tra loro, oppure confrontano rispetto a 0 il valore di un determinato operando o risultato
- Tutte queste istruzioni determinano le condizioni relative sia alla codifica con segno che a quella senza segno, pertanto sono indipendenti dall'interpretazione di dato desiderata
- In **[M68000]**, viene descritto come ciascuna istruzione dell'ASM MC68000 modifica le condizioni memorizzate in `ccr`

Istruzioni Condizionate

- MC68000 dispone di due tipi di istruzioni condizionate
  - le istruzioni di salto condizionato `b`**cc**
  - le istruzioni di memorizzazione di condizione `s`**cc**
- In una precedente lezione abbiamo descritto le regole di entrambi i tipi di istruzioni
- Ora, dopo aver studiato il ruolo delle interpretazioni di dato e descritto le condizioni *Carry Set*, *Overflow Set* e le loro opposte, possiamo elencare tutti i possibili valori dei codici **cc** e i corrispondenti significati

Istruzioni Condizionate

- La tabella seguente mostra i valori di **cc** e i corrispondenti significati, per le condizioni *Plus*, *Carry Set*, *Overflow Set*, e le loro opposte

Codice	Condizione	Significato corrispondente
`mi`	*Minus*	un valore è negativo
`pl`	*Plus*	un valore è non-negativo
`cc`	*Carry Clear*	non si è generato un riporto dalle cifre più significative
`cs`	*Carry Set*	si è generato un riporto dalle cifre più significative
`vc`	*Overflow Clear*	non si è verificato un *overflow*
`vs`	*Overflow Set*	si è verificato un *overflow*

Istruzioni Condizionate

- La tabella seguente (già presentata in una precedente lezione) mostra i valori di **cc** e i corrispondenti significati, per le condizioni relative alle relazioni di uguaglianza

Codice	Condizione	Operatore C corrispondente
`eq`	*Equal*	`==`
`ne`	*Not Equal*	`!=`

Istruzioni Condizionate

- La tabella seguente (già presentata in una precedente lezione) mostra i valori di **cc** e i corrispondenti significati, per le condizioni relative alle relazioni d'ordine determinate applicando la codifica con segno

Codice	Condizione	Operatore C corrispondente
`lt`	*Less Than*	<
`le`	*Less Than or Equal*	<=
`gt`	*Greater Than*	>
`ge`	*Greater Than or Equal*	>=

Istruzioni Condizionate

- La tabella seguente mostra i valori di **cc** e i corrispondenti significati, per le condizioni relative alle relazioni d'ordine determinate applicando la codifica senza segno

Codice	Condizione	Operatore C corrispondente
`lo`	*Lower Than*	<
`ls`	*Lower Than or Same*	<=
`hi`	*Higher Than*	>
`hs`	*Higher Than or Same*	>=

Moltiplicazione e Divisione

- Le istruzioni `muls` e `divs`, già presentate in una precedente lezione, effettuano moltiplicazione e divisione interpretando i loro operandi come numeri con segno
- Le istruzioni `mulu` e `divu` effettuano le analoghe operazioni interpretando i loro operandi come numeri senza segno
- Tranne che per l'interpretazione di dato usata, l'istruzione `mulu` funziona in modo analogo a `muls` e l'istruzione `divu` in modo analogo a `divs`

Codifiche di interi in MIPS32

- In MIPS32, i dati possono essere interpretati come interi senza segno o interi con segno codificati in complemento a 2
- Entrambe le interpretazioni di dato possono essere applicate a tutti i formati di dato generali

Addizione e Sottrazione

- Nell'eseguire una somma o sottrazione, se le cifre più significative dei due numeri generano un riporto (il che implica che il risultato, interpretato come numero senza segno, non può essere contenuto nell'operando destinazione), non vi è alcuna conseguenza
- Le situazioni di *overflow* causate dalla somma o sottrazione di numeri con segno possono essere gestite in due modi diversi
  - Le istruzioni `add` e `sub`, in caso di *overflow*, avviano una *eccezione*, ossia una particolare azione che interrompe la normale esecuzione del programma, che approfondiremo in future lezioni di LPS
  - Le istruzioni `addu` e `subu`, invece, non rilevano il verificarsi di un eventuale *overflow*, e quindi producono un risultato in ciascun caso

Istruzioni Condizionate

- MIPS32-MARS dispone di due tipi di istruzioni condizionate
  - le istruzioni di salto condizionato
  - le istruzioni di memorizzazione di condizione
- In una precedente lezione abbiamo descritto alcune di queste istruzioni, ora completiamo l'elenco
- MIPS32 non ha istruzioni condizionate relative a situazioni di overflow o riporto generato dalle cifre più significative degli operandi

Istruzioni Condizionate

- Come detto in una precedente lezione, le istruzioni condizionate relative alle relazioni di uguaglianza sono
  - le istruzioni di salto condizionato a 2 operandi b**cc**z
  - le istruzioni di salto condizionato a 3 operandi b**cc**
  - le istruzioni di memorizzazione di condizione s**cc**
- I valori dei codici **cc** sono riportati nella tabella seguente, con i corrispondenti significati

Codice	Condizione	Operatore C corrispondente
`eq`	*Equal*	`==`
`ne`	*Not Equal*	`!=`

Istruzioni Condizionate

- In una precedente lezione sono state presentate le seguenti istruzioni condizionate relative a relazioni d'ordine
  - le istruzioni di salto condizionato a 2 operandi `b`**`cc`**`z`
  - le istruzioni di salto condizionato a 3 operandi `b`**`cc`**
  - le istruzioni di memorizzazione di condizione `s`**`cc`** (c'è un'irregolarità sintattica nel caso del codice `lt`, si veda la documentazione di MARS)
- Tali istruzioni adottano la codifica con segno
- I valori dei codici **cc** sono riportati nella tabella seguente, con i corrispondenti significati

Codice	Condizione	Operatore C corrispondente
`lt`	*Less Than*	<
`le`	*Less Than or Equal*	<=
`gt`	*Greater Than*	>
`ge`	*Greater Than or Equal*	>=

Istruzioni Condizionate

- MIPS32-MARS ha anche istruzioni condizionate relative a relazioni d'ordine che adottano la codifica senza segno
  - istruzioni di salto condizionato a 3 operandi `b`**`cc`**`u`
  - istruzioni di memorizzazione di condizione `s`**`cc`**`u` (c'è un'irregolarità sintattica nel caso del codice `lt`, si veda la documentazione di MARS)
- I valori dei codici **cc** sono riportati nella tabella seguente, con i corrispondenti significati

Codice	Condizione	Operatore C corrispondente
`lt`	*Less Than*	<
`le`	*Less Than or Equal*	<=
`gt`	*Greater Than*	>
`ge`	*Greater Than or Equal*	>=

Moltiplicazione e Divisione

- MIPS32-MARS ha diverse istruzioni per effettuare moltiplicazioni e divisioni
- Le istruzioni viste in precedenti lezioni interpretano i loro operandi come interi con segno
- Come nel caso di MC68000, per effettuare moltiplicazioni e divisioni tra interi senza segno si utilizzano istruzioni diverse
- Le istruzioni per effettuare moltiplicazioni e divisioni tra interi senza segno hanno regole simili a quelle che effettuano analoghe operazioni tra interi con segno; hanno anche nomi simili, che si caratterizzano per il suffisso `u`
- Inoltre, come per addizione e sottrazione, esistono istruzioni che ignorano eventuali overflow ed altre che invece, in caso di overflow, generano un'eccezione
- Per i dettagli si rimanda a **[MIPS32]**