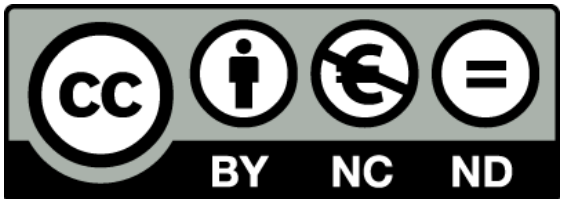
Laboratorio di Programmazione di Sistema

Organizzazione dei Dati in Memoria

Luca Forlizzi, Ph.D.

Versione 23.2



Memoria Principale

- Come sappiamo, la abstract machine di un *ASM-PM* memorizza dati in *dispositivi di memorizzazione*
- I dispositivi di memorizzazione più semplici, chiamati *bit* sono in grado di memorizzare una cifra binaria
- I bit sono raggruppati in dispositivi più complessi
  - i *registri*, che contengono di solito da qualche decina a qualche centinaio di bit
  - la *memoria*, che contiene da migliaia a miliardi di bit
- Ogni dato (ovvero ogni stringa binaria) che un programma utilizza durante l'esecuzione deve essere necessariamente memorizzato in un insieme di bit

Memoria Principale

- Un insieme di bit a cui la abstract machine di un *ASM-PM* può accedere mediante una singola operazione (in lettura o in scrittura), è detto *parola*
- In relazione al dispositivo che le contiene, distinguiamo
  - *parole di registro*, formate da bit contenuti in uno o più registri della abstract machine
  - *parole di memoria*, formate da bit della memoria

Memoria Principale

- Le *parole* sono quindi specifici insiemi di bit, che vengono definiti da un *ASM-PM*
- La *lunghezza* di una parola è il numero di bit che la formano
- Un *formato di dato* è un'insieme di parole che condividono alcune caratteristiche, tra le quali uno specifico valore di lunghezza
- L'*identificativo* di una parola contenuta in un formato **F**, è una stringa alfanumerica che consente di identificare la parola tra tutte quelle contenute in **F**

Memoria Principale

- Utilizzi della memoria
  - Dati Memorizzati
  - Programma Memorizzato
    - Le istruzioni del programma sono sequenze di stringhe binarie e come tali vengono memorizzate in parole
    - Le istruzioni sono memorizzate nella memoria principale, ovvero in dispositivi di memorizzazione usati anche per memorizzare i dati
  - Memory-Mapped Input/Output (MMIO)
    - Le *parole magiche* sono connesse a dispositivi di I/O
    - Gli accessi alle parole magiche producono eventi e azioni esterni alla abstract machine

Memoria Principale

- Tipologie di accesso permesse
  - Parole *read-write*: si può sia leggere che scrivere il contenuto
  - Parole *read-only*: si può leggere il contenuto ma non scriverlo
  - Parole *write-only*: si può scrivere il contenuto ma non leggerlo
- Tentare di scrivere in una parola read-only o di leggere da una parola write-only, in alcuni *ASM-PM* non produce alcun effetto, mentre in altri può produrre malfunzionamenti o comportamenti non definiti
- Semantica delle parole read-write
  - Un'operazione di lettura su una parola ordinaria (ovvero non magica) P ottiene, come contenuto di P, la stringa binaria che è stata memorizzata dall'operazione di scrittura su P più recentemente eseguita
  - Gli effetti prodotti dalle operazioni read-write su parole magiche dipendono da caso a caso

Memoria Principale

- In questa presentazione studiamo da vicino la struttura e l'organizzazione dei dati in memoria, e in particolare delle parole di memoria
- Faremo riferimento all'utilizzo della memoria per la memorizzazione dei dati, ma vedremo, in presentazioni successive, che i concetti esposti sono validi anche in riferimento agli altri utilizzi
- Come già noto, per i formati di dato più utilizzati (tra cui quelli generali), gli identificativi delle parole di memoria sono stringhe binarie dette *indirizzi di memoria* (o semplicemente *indirizzi* quando non vi sono ambiguità possibili), di cui ora parliamo più approfonditamente

Indirizzi di Memoria

- In ogni *ASM-PM* è definito un parametro chiamato *lunghezza degli indirizzi di memoria*, che indichiamo con $\mathrm{LEN_A}$
- Tipicamente $\mathrm{LEN_A}$ è una potenza di 2 e, nella maggioranza architetture in uso oggi, $\mathrm{LEN_A}$ vale 32 o 64
- Gli *indirizzi di memoria* di un *ASM-PM* sono le stringhe binarie di lunghezza pari a $\mathrm{LEN_A}$
- L'insieme di tutti gli indirizzi di memoria è chiamato *spazio di indirizzamento* ed ha cardinalità pari a $2^{\mathrm{LEN_A}}$

Indirizzi di Memoria

- Un formato di dato è detto *Memory Address Identifiable (MAI)* se
  - definisce parole di memoria
  - usa indirizzi di memoria come identificativi per le parle di memoria
- In un formato di dato MAI **F**
  - Se una parola di memoria p appartiene al formato **F**, allora un indirizzo di memoria di p consente di identificare univocamente p tra tutte le parole di memoria di **F**
  - Ovvero ogni parola di memoria q di **F** diversa da p, ha indirizzo di memoria diverso dall'indirizzo di memoria di p
  - Due parole di memoria p e q appartenenti a due formati diversi, possono avere lo stesso indirizzo di memoria
  - Due diversi indirizzi di memoria possono identificare, in un formato **F**, la stessa parola (in questo caso si dice che i due indirizzi di memoria sono *alias* l'uno dell'altro)

Indirizzi di Memoria

- Dunque, in un formato di dato MAI **F** ogni parola di memoria ha almeno un indirizzo di memoria
- Tuttavia, può accadere che vi sia un indirizzo di memoria che non identifica una parola di memoria
- Gli indirizzi di memoria che non identificano parole di **F**, vengono detti indirizzi di memoria *non validi* o *illegali* per **F**

Indirizzi di Memoria

- Gli indirizzi di memoria definiti in un *ASM-PM*, vengono a volte chiamati *indirizzi di memoria logici* in quanto alcuni computer hanno una caratteristica chiamata *memoria virtuale* che prevede una differenziazione tra gli indirizzi di memoria utilizzati dai programmi *ASM* e *LM* per identificare le parole (detti logici) e quelli utilizzati dai programmi dei linguaggi di livello 1 per effettuare l'accesso ai bit (detti fisici)
- Nei computer privi di memoria virtuale, non vi è differenza tra indirizzi di memoria logici e indirizzi di memoria fisici

Indirizzi di Memoria

- La memoria virtuale consente di utilizzare la memoria secondaria per dotare le abstract machine di livello 3 e 4 di una quantità di memoria principale superiore a quella effettivamente disponibile alla abstract machine di livello 2
- Si tratta di una caratteristica realizzata dai sistemi operativi, utilizzando a volte apposite caratteristiche di alcune *ISA*
- La memoria virtuale viene studiata approfonditamente nel corso di Sistemi Operativi
- Se non diversamente specificato, in LPS con indirizzo di memoria si intende un indirizzo di memoria logico

Indirizzi di Memoria

- Gli indirizzi di memoria possono essere suddivisi in parti, dette *componenti*, ciascuna delle quali viene utilizzata dall'*ASM-PM* come una stringa binaria separata dalle altre
  - La maggior parte degli *ASM-PM*, adottano il *flat memory model*, nel quale un indirizzo di memoria ha una sola componente
  - Alcuni *ASM-PM*, tra cui quelli della famiglia IA-32, adottano (a volte opzionalmente) il *segmented memory model*, nel quale ogni indirizzo di memoria ha 2 componenti

Indirizzi di Memoria

- Nel *flat memory model* ogni indirizzo di memoria ha una sola componente, ovvero viene gestito come un'unica stringa binaria di $\mathrm{LEN_A}$ cifre, e quindi può essere interpretato come un singolo numero intero in codifica naturale
- Lo spazio di indirizzamento è quindi l'insieme di tutti gli interi positivi compresi tra 0 e $2^{\mathrm{LEN_A}} - 1$
- Risulta quindi definito un *ordine tra gli indirizzi di memoria*, che coincide con l'ordine tra gli interi positivi

Indirizzi di Memoria

- Nel *segmented memory model* ogni indirizzo di memoria i ha 2 componenti, e viene quindi indicato mediante una coppia di stringhe binarie (s, o)
  - La componente s, detta *selettore di segmento*, identifica il segmento cui appartiene l'indirizzo di memoria
  - La componente o, detta *offset*, identifica l'indirizzo di memoria all'interno del segmento cui appartiene
- Sia i selettori di segmento che gli offset possono essere interpretati come interi in codifica naturale
- Da ciò segue un *ordine tra gli indirizzi di memoria* che appartengono allo stesso segmento

Indirizzi di Memoria

- Dall'*ordine tra gli indirizzi di memoria*, sia nel *flat* che nel *segmented memory model* derivano i seguenti concetti
  - *Predecessore* di un indirizzo di memoria
    - Nel flat memory model, se i è un indirizzo di memoria compreso tra 1 e $2^{\mathrm{LEN_A}} - 1$, $i - 1$ è il predecessore di i
    - Nel segmented memory model, se $i = (s, o)$ è un indirizzo di memoria tale che $o > 0$, l'indirizzo $(s, o - 1)$ è il predecessore di i
  - *Successore* di un indirizzo di memoria
  - Relazioni *minore di* e *maggiore di* tra indirizzi di memoria
  - *Minimo* e *massimo* in un insieme di indirizzi di memoria
  - *Distanza* tra indirizzi di memoria, definita come il valore assoluto della differenza tra
    - due indirizzi di memoria, nel flat memory model
    - due offset di indirizzi di memoria che appartengono allo stesso segmento, nel segmented memory model
  - *Contiguità* tra indirizzi di memoria, proprietà vera per due indirizzi di memoria la cui distanza vale 1

Indirizzi di Memoria in MC68000

- Di norma MC68000, come tutte le versioni di M68000, utilizza il flat memory model
- Il segmented memory model può essere utilizzato grazie ad un'estensione, opzionale, chiamata Memory Management Unit (MMU)
- Le implementazioni di MC68000 non hanno MMU, ma possono essere connesse a un coprocessore che fornisce la MMU
- In tutte le versioni di M68000, gli indirizzi di memoria sono formati da 32 cifre binarie, ma in alcune, due indirizzi che differiscono solo per le cifre di posizioni maggiori, sono per definizione dell'architettura, alias l'uno dell'altro
- Ad esempio, in MC68000, due indirizzi di memoria diversi ma che hanno uguali le cifre di posizione compresa tra 0 e 23, sono, per definizione, alias l'uno dell'altro

Indirizzi di Memoria in MIPS32

- Di norma MIPS32 utilizza il flat memory model
- Gli indirizzi di memoria sono formati da 32 cifre binarie

Formato Byte

- In ogni *ASM-PM*, vi sono uno o più formati di dato MAI
- Tra di essi, quello che ha lunghezza minima, viene definito come *formato byte*
- Le parole di memoria che appartengono al formato byte sono chiamate *byte* di memoria
- La lunghezza dei byte di memoria viene indicata con $\mathrm{LEN}_{\mathrm{byte}}$, e per definizione è minore di quella di ogni altra parola di memoria che appartiene ad un formato MAI

Formato Byte

- In molti *ASM-PM*, tra cui quelli della famiglia M68000, il formato byte contiene anche delle parole di registro chiamate *byte* di registro, che naturalmente hanno lunghezza pari a $\mathrm{LEN}_{\mathrm{byte}}$
- Ciò consente di copiare il contenuto di un byte di memoria in un registro, e viceversa, in modo efficiente
- In questa presentazione, poiché ci concentriamo sulle parole di memoria, il termine byte, non ulteriormente specificato, indica un byte di memoria

Formato Byte

- Nella quasi totalità dei computer in uso oggi, $\mathrm{LEN}_{\mathrm{byte}}$ è pari ad 8, ma non è sempre stato così
- Ad esempio nel PDP-7, computer per il quale Ritchie ha iniziato lo sviluppo del C, $\mathrm{LEN}_{\mathrm{byte}}$ è pari a 18
- In una implementazione C, le variabili di tipo `char` hanno un numero di bit pari al valore di $\mathrm{LEN}_{\mathrm{byte}}$ dell'*ASM-PM* usato dall'implementazione, il che consente di rappresentare una variabile di tipo `char` mediante un byte (di memoria o di registro)
- La macro `CHAR_BIT` definita nell'header `limits.h` ha valore pari al numero di bit di una variabile di tipo `char`, ovvero pari a $\mathrm{LEN}_{\mathrm{byte}}$
- C Standard richiede che il valore di `CHAR_BIT` sia maggiore o uguale ad 8

Formato Byte

- È interessante osservare che molto spesso i dispositivi che costituiscono una CPU (i quali sono implementazioni di abstract machine di livello 1) svolgono molte delle loro operazioni con insiemi di bit di lunghezza $\mathrm{LEN}_{\mathrm{byte}}$, ovvero con byte
- Tipicamente, un byte è l'insieme di bit di lunghezza minima che può essere trasferito tra memoria e CPU mediante una singola operazione
- Dunque il byte è un concetto presente anche al livello di astrazione 1
- Questa osservazione non ha comunque effetto sul livello 2 e sui livelli superiori

Formato Byte

- Dati due indirizzi di memoria i_1 e i_2, validi per il formato byte e tali che i_2 è l'indirizzo successore di i_1, diciamo che
  - il byte che ha indirizzo di memoria i_1 è il *predecessore* del byte che ha indirizzo di memoria i_2
  - il byte che ha indirizzo di memoria i_2 è il *successore* del byte che ha indirizzo di memoria i_1
  - il byte che ha indirizzo di memoria i_1 e il byte che ha indirizzo di memoria i_2 sono *contigui*

Formato Byte

- In un *ASM-PM* che adotta il segmented memory model, si dice che un byte b che ha indirizzo di memoria (s, o) *appartiene* al segmento s
- Quindi un byte b_1, che ha indirizzo di memoria (s_1, o_1), e un byte b_2, che ha indirizzo di memoria (s_2, o_2), appartengono allo stesso segmento se e solo se $s_1 = s_2$

Organizzazione dei Bit in un Byte

- Come sappiamo, le posizioni dei bit all'interno di una parola sono molto importanti perché sono alla base delle semantiche fornite attraverso le interpretazioni di dato
- Il formato byte definisce le posizioni dei bit che formano ciascuno dei byte B
- A ciascuno dei bit b contenuti in B, viene assegnato un numero intero compreso tra 0 e $\mathrm{LEN}_{\mathrm{byte}}$-1, detto *posizione* e indicato come $\mathrm{pos\_bit}_B(b)$
- L'assegnazione fa in modo che bit diversi abbiano posizioni diverse
- Se un bit b ha posizione pari a k nel byte B, diciamo che b *è il bit k di B*

Byte e Organizzazione dei Dati in Memoria

- In quasi tutti gli *ASM-PM*, il formato byte ha un ruolo chiave nell'organizzazione dei dati in memoria
- In primo luogo, il formato byte gode di una importante proprietà: è una partizione dell'insieme di tutti i bit della memoria, ovvero ogni bit della memoria appartiene ad uno ed un solo byte di memoria
- Inoltre, ogni altro formato di dato MAI **F**
  - ha lunghezza che non solo (per definizione) è maggiore di $\mathrm{LEN}_{\mathrm{byte}}$, ma è anche un multiplo di $\mathrm{LEN}_{\mathrm{byte}}$
  - ciascuna parola che appartiene al formato **F** è formata da tutti i bit contenuti in un insieme di byte
- Per spiegare in dettaglio come è organizzata la memoria e il ruolo dei byte, è utile definire alcuni concetti relativi a insiemi di bit di memoria che non sono necessariamente parole

Aggregazione di Byte

- Un insieme G di bit di memoria è un'*aggregazione di byte* se esiste un insieme di byte $\mathrm{ByteBase}_G = \{B_0, B_1, \ldots, B_k\}$ tale che
  - Ogni bit di G, è contenuto in un byte dell'insieme $\mathrm{ByteBase}_G$
  - Ogni bit che è contenuto in un byte dell'insieme $\mathrm{ByteBase}_G$, è contenuto in G
  - Se l'*ASM-PM* adotta il segmented memory model, tutti i byte dell'insieme $\mathrm{ByteBase}_G$ appartengono allo stesso segmento
- L'insieme $\mathrm{ByteBase}_G$ viene detto *base di byte di G*
- Si osservi che dati un byte B e un'aggregazione di byte G, si ha che $B \in \mathrm{ByteBase}_G \Rightarrow B \subseteq G$

Aggregazione di Byte

- Data un'aggregazione di byte G, definiamo
  - *dimensione* di G (indicata come DIM_G) la cardinalità di $\mathrm{ByteBase}_G$
  - *indirizzo* di G (indicato come I_G) l'indirizzo di memoria minimo tra quelli dei byte di $\mathrm{ByteBase}_G$
  - *lunghezza* di G (indicata come LEN_G) il numero di bit contenuti in G, pari a $\mathrm{DIM}_G \cdot \mathrm{LEN}_{\mathrm{byte}}$
- Si noti, quindi, che la lunghezza di un'aggregazione di byte è un multiplo di $\mathrm{LEN}_{\mathrm{byte}}$
- Un singolo byte è un'aggregazione di byte, di dimensione pari a 1

Aggregazione di Byte

- Esistono quindi moltissime aggregazioni di byte, una per ciascun diverso insieme di byte
- La maggior parte delle aggregazioni di byte non è ovviamente una parola
- Invece, nella maggior parte degli *ASM-PM*, per ogni formato di dato MAI **F**, le parole di memoria contenute in **F** sono aggregazioni di byte

Aggregazione di Byte

- Esempi di insiemi di bit in MC68000 ($\mathrm{LEN}_{\mathrm{byte}}$ vale 8)
  - G_1: bit 1, 2, 3, 4, 5, 6, 7 del byte di indirizzo \$2000 e bit 4, 2, 0, 6, 1, 3, 5, 7 del byte di indirizzo \$200C
  - G_2: bit 1, 2, 3, 4, 5, 6, 7, 0 del byte di indirizzo \$2000 e bit 4, 2, 0, 6, 1, 3, 5, 7 del byte di indirizzo \$200C
  - G_3: bit 1, 2, 3, 4, 5, 6, 7, 0 del byte di indirizzo \$2000 e bit 4, 2, 0, 6, 1, 3, 5, 7 del byte di indirizzo \$2002
  - G_4: bit 1, 2, 3, 4, 5, 6, 7, 0 del byte di indirizzo \$2000 e bit 4, 2, 0, 6, 1, 3, 5, 7 del byte di indirizzo \$2001
- G_1 non è un'aggregazione di byte, mentre G_2, G_3, G_4 lo sono
- G_2 non è una parola
- G_3 è una parola del formato `word-p`, come vedremo a breve
- G_4 è una parola del formato `word`, come vedremo a breve

Area di Memoria

- Definiamo *area di memoria* (semplicemente *area* quando non vi sono ambiguità) un insieme di bit G con le seguenti proprietà:
  - G è un'aggregazione di byte
  - Se $i_{\min}$ e $i_{\max}$ sono, rispettivamente, l'indirizzo di memoria minimo e l'indirizzo di memoria massimo tra quelli dei byte contenuti in $\mathrm{ByteBase}_G$, allora per ogni indirizzo di memoria i tale che $i_{\min} \leq i \leq i_{\max}$, i è valido e il byte che ha indirizzo di memoria i appartiene a $\mathrm{ByteBase}_G$

Area di Memoria

- In parole povere, un'area di memoria G è un insieme di bit che non ha "buchi"
- Nel flat memory model
  - l'insieme degli indirizzi di memoria di tutti byte di $\mathrm{ByteBase}_G$, interpretati come interi positivi, forma un intervallo
  - se $i_{\max}$ è l'indirizzo di memoria massimo tra quelli dei byte di $\mathrm{ByteBase}_G$, si ha $i_{\max} = \mathrm{I}_G + \mathrm{DIM}_G - 1$
- Nel segmented memory model
  - l'insieme degli offset degli indirizzi di memoria di tutti byte di $\mathrm{ByteBase}_G$, interpretati come interi positivi, forma un intervallo
  - se $o_{\min}$ è l'offset di I_G e $o_{\max}$ è l'offset dell'indirizzo di memoria massimo tra quelli dei byte di $\mathrm{ByteBase}_G$, si ha $o_{\max} = o_{\min} + \mathrm{DIM}_G - 1$

Area di Memoria

- Poiché un'area di memoria G è un caso particolare di aggregazione di byte, "eredita" da esso le definizioni di *dimensione*, *indirizzo* e *lunghezza*
- Inoltre, se $i_{\max}$ è l'indirizzo di memoria massimo tra quelli dei byte di $\mathrm{ByteBase}_G$, il *byte successore di* G, se esiste, è il byte successore del byte (di $\mathrm{ByteBase}_G$) che ha indirizzo di memoria $i_{\max}$

Area di Memoria

- Esempi di insiemi di bit in MC68000 ($\mathrm{LEN}_{\mathrm{byte}}$ vale 8)
  - G_5: tutti i bit dei byte di indirizzi \$3000 e \$300C
  - G_6: tutti i bit dei byte di indirizzi \$3000 e \$3002
  - G_7: tutti i bit dei byte di indirizzi \$3000, \$3001, \$3002
  - G_8: tutti i bit dei byte di indirizzi \$3000, \$3001, \$3002, \$3003
- G_5, G_6, G_7, G_8 sono tutte aggregazioni di byte
- G_5 non è un'area di memoria, né è una parola
- G_6 non è un'area di memoria ed è una parola del formato `word-p`, come vedremo a breve
- G_7 è un'area di memoria e non è una parola
- G_8 è un'area di memoria ed è una parola del formato `long`, come vedremo a breve

Formati Standard

- La maggior parte degli *ASM-PM* ha dei formati di dato che definiscono parole di memoria con caratteristiche legate a quelle dei byte
- Un *formato standard* **F** è un formato di dati che
  - ha lunghezza pari a $\mathrm{DIM}_F \cdot \mathrm{LEN}_{\mathrm{byte}}$, dove DIM_F è un valore intero detto *dimensione* di **F**
  - è un formato MAI (quindi definisce parole di memoria che hanno come identificativo un indirizzo di memoria)
  - ciascuna parola di memoria P di **F** è un'area di memoria
- Si osservi che anche il formato byte soddisfa la definizione di formato standard, con dimensione pari a 1

Formati Standard

- Le parole di memoria definite da un formato standard, vengono dette *parole standard*, e sono per definizione aree di memoria di dimensione pari a DIM_F
- Tutti gli *ASM-PM*, quindi, hanno almeno un formato standard, ovvero il formato byte
- Spesso vengono definiti altri formati standard, con dimensione pari a una potenza di 2
- Valori tipici per la dimensione DIM_F di un formato standard F, sono 1, 2, 4, 8, ma a volte si incontrano anche valori superiori (ad esempio MC68040 e MC68060 hanno un formato standard di dimensione 16)

Formati Standard

- Spesso un formato standard definisce anche delle parole di registro
- Ovviamente tali parole hanno la medesima lunghezza delle parole di memoria del formato, ciò consente trasferimenti di dato efficienti tra memoria e registri
- Tali parole di registro sono anch'esse chiamate *parole standard*; in questa presentazione, se non diversamente specificato, con il termine parola standard ci riferiamo ad una parola standard di memoria

Formati Standard

- Poiché una parola standard P è un caso particolare di area di memoria
  - per P sono definite dimensione, indirizzo (che come detto è l'identificativo di P nel formato), lunghezza e byte successore
  - se B è un byte e $B \in \mathrm{ByteBase}_P$ allora $B \subseteq P$
- Se P è una parola standard, $\mathrm{ByteBase}_P$ contiene DIM_P byte
  - nel flat memory model, se I_P è l'indirizzo di P, allora gli indirizzi dei byte contenuti in $\mathrm{ByteBase}_P$ sono $\mathrm{I}_P, \mathrm{I}_P + 1, \ldots, \mathrm{I}_P + \mathrm{DIM}_P - 1$
  - nel segmented memory model, se (s, o) è l'indirizzo di P, allora gli indirizzi dei byte contenuti in $\mathrm{ByteBase}_P$ sono $(s, o), (s, o + 1), \ldots, (s, o + \mathrm{DIM}_P - 1)$

Formati Standard

- È importante notare che mentre l'insieme di tutti i byte costituisce sempre una partizione della memoria, lo stesso non sempre accade per l'insieme di tutte le parole standard che hanno un determinato formato **F**
- Infatti può accadere che due diverse parole di **F**, contengano uno stesso byte o anche uno stesso insieme di byte
- Ad esempio, un *ASM-PM* che adotta il flat memory model potrebbe avere un formato **F** che definisce come parole standard di dimensione 2, tutte le aree di memoria formate da una coppia di byte contigui; allora, se i, $i+1$ e $i+2$ sono indirizzi validi, si ha che
  - i byte di indirizzo i e $i+1$ formano la parola di indirizzo i
  - i byte di indirizzo $i+1$ e $i+2$ formano la parola di indirizzo $i+1$
- Com'è evidente, il byte di indirizzo $i+1$ sarebbe contenuto sia nella parola di indirizzo i che in quella di indirizzo $i+1$

Formati Standard

- Un caso tipico di formato standard è il formato `word` in MC68000
  - Come sappiamo, la lunghezza di `word` è 16 e per MC68000 $\mathrm{LEN}_{\mathrm{byte}}$ vale 8
  - Il formato `word` definisce parole di memoria standard
  - Quindi le parole di memoria di `word` sono aree di memoria di dimensione 2
  - L'identificativo di una parola di memoria P di `word` è l'indirizzo di P, ovvero l'indirizzo di memoria minimo tra quelli dei due byte che appartengono a $\mathrm{ByteBase}_P$
  - Ad esempio l'insieme di bit G_9 costituito da tutti i bit dei byte di indirizzi \$3000 e \$3001, è un'area di memoria ed ha indirizzo \$3000
  - G_9 è una parola standard appartenente a `word` e il suo identificativo è il suo indirizzo, ovvero \$3000

Allineamento nei Formati Standard

- Alcuni formati standard, diversi dal formato byte, definiscono una restrizione agli indirizzi validi per le parole standard di tale formato, chiamata *vincolo di allineamento*
- Per un formato **F**, un vincolo di allineamento con *fattore di allineamento* h, dove $h > 0$ è un intero, prescrive che
  - nel *flat memory model*, ogni indirizzo i di una parola di **F**, interpretato come numero in codifica naturale, sia multiplo di h
  - nel *segmented memory model*, per ogni indirizzo (s, o) di una parola di **F**, l'offset o, interpretato come numero in codifica naturale, sia multiplo di h
- Se un formato standard **F** stabilisce un vincolo di allineamento con fattore di allineamento h, diciamo che le parole definite da **F** *hanno allineamento h*

Allineamento nei Formati Standard

- Spesso, un formato standard con dimensione D ha fattore di allineamento pari a D
- Questa condizione garantisce che l'insieme delle parole definite da tale formato costituisca una partizione della memoria
- Ad esempio, in MC68000, che adotta il flat memory model, il formato `word` definisce parole standard di dimensione pari a 2, con fattore di allineamento 2:
  - La parola di `word` che ha indirizzo i, contiene i byte di indirizzi i, $i + 1$
  - Poiché i è multiplo 2 e il fattore di allineamento è 2, $i - 1$ e $i + 1$ non sono indirizzi di parole di `word`
  - Quindi nessun altra parola di `word` contiene i byte che hanno indirizzi i e $i + 1$

Organizzazione di Bit e Byte nei Formati Standard

- Ciascuna parola di memoria di un formato standard è un'area di memoria, quindi è possibile accedere individualmente ai byte che la compongono
- Ad esempio, è possibile modificare il contenuto di una parola standard P, anche attraverso la modifica del contenuto di uno dei byte di $\mathrm{ByteBase}_P$
- In molte applicazioni ciò è utile e/o necessario
- È quindi indispensabile capire come, in un formato standard, i bit di una parola P siano organizzati in relazione all'organizzazione dei byte di $\mathrm{ByteBase}_P$

Organizzazione di Bit e Byte nei Formati Standard

- Sia **F** un formato che contiene parole standard di dimensione D, con $D > 1$ e di lunghezza pari a $L = D \cdot \mathrm{LEN}_{\mathrm{byte}}$
- Per ogni parola $P \in \mathbf{F}$, a ciascuno dei byte B che appartiene a $\mathrm{ByteBase}_P$, identificato dal suo indirizzo di memoria, viene assegnata una *posizione* $\mathrm{pos\_byte}_P(B)$, compresa tra 0 e $D - 1$
- La relazione tra gli indirizzi di memoria e le posizioni dei byte di $\mathrm{ByteBase}_P$ viene detta *endianness* e, in linea di principio, può essere del tutto arbitraria

Organizzazione di Bit e Byte nei Formati Standard

- Nel seguito, illustreremo come *ASM-PM* diversi definiscano in modo diverso la endianness ed è proprio da queste differenze che scaturisce l'origine di questo termine
- Data una parola $P \in \mathbf{F}$, le posizioni assegnate ai byte di $\mathrm{ByteBase}_P$ vengono utilizzate per assegnare a ciascun bit b di P, una *posizione* in P, mediante una regola che invece è uguale per ciascun *ASM-PM*

Organizzazione di Bit e Byte nei Formati Standard

- A ciascun bit b di una parola P, contenuto nel byte $B \in \mathrm{ByteBase}_P$, viene assegnata da **F** una posizione in P in base a
  - la posizione in P del byte B, indicata come $\mathrm{pos\_byte}_P(B)$
  - la posizione di b in B, indicata come $\mathrm{pos\_bit}_B(b)$
- La posizione di b in P, indicata come $\mathrm{pos\_bit}_P(b)$, è

$$\mathrm{pos\_bit}_P(b) = \mathrm{pos\_byte}_P(B) \cdot \mathrm{LEN}_{\mathrm{byte}} + \mathrm{pos\_bit}_B(b)$$

- Si osservi che $\mathrm{pos\_bit}_P(b)$ è compresa tra 0 e $L - 1$

Organizzazione di Bit e Byte nei Formati Standard

- Capita sovente, nella programmazione, di dover fare il calcolo inverso, ovvero data la posizione $\mathrm{pos\_bit}_P(b)$ di un bit b all'interno di una parola P di un formato standard **F**, trovare
  - La posizione $\mathrm{pos\_bit}_B(b)$ di b all'interno del byte B in cui è contenuto
  - La posizione $\mathrm{pos\_byte}_P(B)$ di B all'interno di P
- Indicando con $\div$ e $\mod$, rispettivamente, gli operatori che calcolano il quoziente e il resto della divisione intera, si ha:
  - $\mathrm{pos\_bit}_B(b) = \mathrm{pos\_bit}_P(b) \mod \mathrm{LEN}_{\mathrm{byte}}$
  - $\mathrm{pos\_byte}_P(B) = \mathrm{pos\_bit}_P(b) \div \mathrm{LEN}_{\mathrm{byte}}$

Endianness dei Formati Standard

- Come abbiamo detto, in molte applicazioni è importante accedere ai bit di una parola P che sono contenuti in uno specifico byte di $\mathrm{ByteBase}_P$
- Se b è un bit di P contenuto in $B \in \mathrm{ByteBase}_P$, per poter accedere al byte B è necessario calcolare l'indirizzo di memoria di B

Endianness dei Formati Standard

- In particolare, si supponga di conoscere l'indirizzo di memoria di P e la posizione $\mathrm{pos\_bit}_P(b)$ di b in P
- Da tali dati, in base alle relazioni precedenti, possiamo determinare
  - Quali sono gli indirizzi di memoria di tutti i byte di $\mathrm{ByteBase}_P$
  - La posizione in P dello specifico byte B, ovvero $\mathrm{pos\_byte}_P(B)$
- Per poter individuare, tra gli indirizzi di memoria di tutti i byte di $\mathrm{ByteBase}_P$, lo specifico indirizzo di B, è necessario conoscere la *endianness*, ovvero la relazione che c'è tra gli indirizzi dei byte di $\mathrm{ByteBase}_P$ e le posizioni di tali byte in P

Endianness dei Formati Standard

- Come abbiamo già detto, *ASM-PM* diversi definiscono in modi diversi la endianness
- I due modi più comuni, ma non gli unici, sono chiamati *little-endian* e *big-endian*
- Per capire tali modi di organizzare i byte, immaginiamo di osservare una parola standard un byte per volta, a partire dal byte che ha indirizzo minimo e procedendo sempre per indirizzi crescenti

Endianness dei Formati Standard

- Nel modo *little-endian*, "l'estremità piccola" di una parola P è quella che si osserva per prima
  - il byte di $\mathrm{ByteBase}_P$ che ha indirizzo minimo, è il byte che ha posizione minima
  - i byte di $\mathrm{ByteBase}_P$ che hanno indirizzi crescenti, hanno posizioni crescenti all'interno della parola
- Più precisamente, se la memoria è organizzata in modo *little-endian*, P è una parola standard e B è un byte contenuto in P
  - nel flat memory model, se i è l'indirizzo di P, l'indirizzo di B è $i + \mathrm{pos\_byte}_P(B)$
  - nel segmented memory model, se (s, o) è l'indirizzo di P, l'indirizzo di B è $(s, o + \mathrm{pos\_byte}_P(B))$

Endianness dei Formati Standard

- Nel modo *big-endian*, invece, "l'estremità grande" di una parola P è quella che si osserva per prima
  - il byte di $\mathrm{ByteBase}_P$ che ha indirizzo minimo, è il byte che ha posizione massima
  - i byte di $\mathrm{ByteBase}_P$ che hanno indirizzi crescenti, hanno posizioni decrescenti all'interno della parola
- Più precisamente, se la memoria è organizzata in modo *big-endian*, P è una parola standard di dimensione D e B è un byte contenuto in P
  - nel flat memory model, se i è l'indirizzo di P, l'indirizzo di B è $i + (D - 1) - \mathrm{pos\_byte}_P(B)$
  - nel segmented memory model, se (s, o) è l'indirizzo di P, l'indirizzo di B è $(s, o + (D - 1) - \mathrm{pos\_byte}_P(B))$

Endianness dei Formati Standard

- I termini *little-endian* e *big-endian* sono tratti dal romanzo "*I viaggi di Gulliver*", di Jonathan Swift, in cui gli abitanti delle isole di *Lilliput* e *Blefuscu* sono in guerra a causa del diverso modo di aprire le uova: dalla parte più piccola (*little-endian*) o dalla più grande (*big-endian*)
- L'uso di tali parole nacque per sottolineare la futilità delle frequenti dispute tra i "tifosi" dell'una o dell'altra organizzazione, in quanto in realtà nessuna delle due offre vantaggi davvero significativi e generalizzabili rispetto all'altra

Endianness dei Formati Standard

- Con pochissime eccezioni, gli *ASM-PM* adottano o l'organizzazione *little-endian* o quella *big-endian*
- *ASM-PM* che usano il modo *little-endian*: IA-32, 6502, Z80, Alpha, PDP-11 (per la maggior parte dei formati), VAX
- *ASM-PM* che usano il modo *big-endian*: IBM System/360, IBM POWER, SPARC (fino alla versione 9 della *ISA*)
- *ASM-PM* che possono essere configurati per usare uno dei due modi: ARM, PowerPC, SPARC (dalla versione 9 della *ISA*), PA-RISC, IA-64
- PDP-11 ha un formato di parole di lunghezza 32 in cui i bit sono organizzati in modo diverso sia dal *little-endian* che dal *big-endian*

Parole di Memoria Non-standard

- Le parole di memoria di più comune utilizzo nella programmazione *ASM*, sono parole standard
- Tuttavia alcuni *ASM-PM* hanno anche formati di dato che definiscono parole di memoria ma non soddisfano la definizione di formato standard
- Le parole definite da formati non-standard sono chiamate *parole non-standard*

Parole di Memoria Non-standard

- Parole di memoria non-standard possono essere
  - aggregazioni di byte che non formano un'area di memoria
  - singoli bit
  - bit-field, ovvero insiemi di bit che non sono aggregazioni di byte
- Si noti che una parola non-standard che è un bit field, potrebbe avere una lunghezza minore rispetto a quella di un byte
- Ovvero esistono, in alcuni *ASM-PM*, formati di dato non-standard che definiscono parole di memoria che hanno una lunghezza minore di $\mathrm{LEN}_{\mathrm{byte}}$
- Ciò non viola la definizione di formato byte, in quanto nei formati di dato non-standard di lunghezza minore di $\mathrm{LEN}_{\mathrm{byte}}$, gli identificativi delle parole di memoria non sono indirizzi di memoria

Formati di Dato in MC68000

- MC68000 ha 3 formati di dato standard, che già conosciamo
- Tutti e 3 sono formati generali
- Le caratteristiche dei 3 formati, in relazione alle parole di memoria, sono le seguenti
  - `byte`: lunghezza 8
  - `word`: lunghezza 16, fattore di allineamento 2 (quindi l'insieme delle `word` è una partizione della memoria), organizzazione *big-endian*
  - `long`: lunghezza 32, fattore di allineamento 2 (quindi l'insieme delle `long` non è una partizione della memoria), organizzazione *big-endian*

Formati di Dato in MC68000

- MC68000 ha anche formati non-standard, usati da particolari istruzioni, che definiscono parole di memoria; ad esempio
  - *bit-field* di lunghezza minore di 8, usati come operandi immediati dalle istruzioni di shift
  - il formato `bit`, usato dalle istruzioni su singolo bit introdotte in una precedente presentazione, definisce anche singoli bit contenuti in memoria
  - il formato `word-p`, usato dall'istruzione `movep`, definisce parole costituite da 2 byte i cui indirizzi hanno distanza 2, e che quindi sono aggregazioni di byte ma non aree di memoria
- Versioni successive di M68000 hanno ulteriori formati non-standard, tra cui, in MC68020 e versioni successive, *bit-field* di lunghezza compresa tra 1 e 32 bit

Formati di Dato in MIPS32

- MIPS32 ha 3 formati standard
  - `byte`: lunghezza 8
  - `half`: lunghezza 16, fattore di allineamento 2 (quindi l'insieme delle `half` è una partizione della memoria), organizzazione configurabile come *little-endian* o *big-endian*
  - `word`: lunghezza 32, fattore di allineamento 4 (quindi l'insieme delle `word` è una partizione della memoria), organizzazione configurabile come *little-endian* o *big-endian*
- In MIPS32-MARS l'organizzazione sia del formato `half` che del formato `word` è configurata come *little-endian*

Formati di Dato in MIPS32

- In MIPS32, come sappiamo, il formato `byte` non definisce parole di registro, ma solo di memoria
- Inoltre il formato `byte`, pur essendo ovviamente un formato standard, non è un formato generale in quanto è usato solo da poche istruzioni che effettuano trasferimenti dati da byte a registri e viceversa
- MIPS32 ha anche formati non-standard, che definiscono parole di memoria usate come operandi immediati da particolari istruzioni
- Versioni successive di MIPS32 hanno ulteriori formati non-standard, tra cui *bit-field* di registro

Parole di Memoria Come Operandi

- Utilizzando parole di memoria come operandi, la abstract machine di un *ASM-PM* vede aumentare notevolmente le proprie capacità
  1. In primo luogo l'utilizzo della memoria permette di superare il principale limite dei registri, ovvero la loro scarsità
  2. Non meno importante è il fatto che le parole di memoria sono molto versatili e possono essere usate non solo come alternativa ai registri ma anche in modi diversi e più sofisticati, che verranno illustrati in successive presentazioni

Parole di Memoria Come Operandi

- La notevole flessibilità delle parole di memoria è dovuta principalmente al fatto che gli identificativi usati per la maggior parte delle parole di memoria, ovvero gli indirizzi di memoria, non sono semplici stringhe alfanumeriche, date e immutabili, ma stringhe binarie
- Essendo stringhe binarie, gli indirizzi di memoria possono essere gestiti come dati, ovvero
  - possono essere memorizzati in altre parole
  - possono essere utilizzati e calcolati da operazioni aritmetico-logiche

Parole di Memoria Come Operandi

- Dunque le parole di memoria sono dispositivi
  - in grado di contenere una stringa binaria (il *contenuto* della parola)
  - identificate mediante una stringa binaria (l'*indirizzo* della parola)
- Avere chiara la distinzione tra contenuto e indirizzo di una parola è fondamentale per programmare in *ASM* e in *LM*
- Ciò emergerà con più chiarezza nelle prossime presentazioni

Parole di Memoria Come Operandi

- La possibilità di utilizzare in svariati modi le parole di memoria, passa attraverso il concetto di *modo di indirizzamento*, che è stato introdotto in LPS sin dall'inizio dello studio dell'*ASM*
- Si ricorda che un *modo di indirizzamento* (*addressing mode*) è una regola che, a partire da alcune informazioni, permette di indirizzare una parola
- Ogni *ASM-PM*, definisce una certa varietà di modi di indirizzamento
- Le regole semantiche di ciascuna istruzione indicano, per ciascuno degli operandi, quali sono i modi di indirizzamento che è possibile utilizzare per tale operando

Parole di Memoria Come Operandi

- Nella maggior parte degli *ASM-PM*, per le parole di registro esiste un solo modo di indirizzamento, ovvero l'indirizzamento diretto-registro, più l'indirizzamento implicito per alcuni registri speciali
- Per le parole di memoria la situazione è ben diversa
- Conosciamo già un modo di indirizzamento che consente di accedere a parole di memoria: l'indirizzamento immediato
- Tale modo di indirizzamento ha caratteristiche ben precise
  - per una data istruzione I, consente di accedere ad un'unica, specifica, parola di memoria associata ad I, alla quale nessuna altra istruzione può accedere mediante indirizzamento immediato
  - l'indirizzamento immediato non consente di modificare la parola indirizzata

Parole di Memoria Come Operandi

- È chiaro, quindi, che una parola indirizzata mediante indirizzamento immediato ha un utilizzo molto specifico, ovvero l'equivalente di un valore costante in un *HLL*
- Per poter utilizzare una parola di memoria come variabile, modificabile ed accessibile da più di una istruzione, è necessario indirizzarla in modo diverso
- Come vedremo in questa e in successive presentazioni, oltre all'indirizzamento immediato esistono diversi altri modi di indirizzamento per parole di memoria che permettono di implementare in maniera più efficiente determinate tecniche di programmazione e ne rendono possibili di nuove

Parole di Memoria Come Operandi

- Un operando che viene indirizzato con un modo di indirizzamento per parole di memoria diverso dall'indirizzamento immediato, viene chiamato *operando memoria*
- Ci sarà utile quindi classificare gli operandi in 4 categorie, rispetto al modo di indirizzamento usato per indirizzarli
  1. operandi impliciti: indirizzati mediante indirizzamento implicito
  2. operandi registro: indirizzati mediante indirizzamento diretto-registro
  3. operandi immediati: indirizzati mediante indirizzamento immediato
  4. operandi memoria: indirizzati mediante un modo di indirizzamento per parole di memoria diverso dall'immediato

Parole di Memoria Come Operandi

- Le possibilità di utilizzare parole di memoria come operandi variano sensibilmente tra diversi *ASM-PM*, in particolare tra architetture CISC e RISC
- Tipicamente, nelle architetture CISC
  - ci sono molti modi di indirizzamento per parole di memoria
  - molte istruzioni, in particolare aritmetico-logiche, possono usare operandi memoria
  - le parole di memoria dei formati di dato standard possono essere usate come operandi memoria per molte istruzioni
  - sono disponibili parole di memoria non-standard per istruzioni particolari
- Invece, nelle architetture RISC
  - ci sono pochi modi di indirizzamento per parole di memoria
  - le uniche istruzioni che usano operandi memoria sono istruzioni di trasferimento da memoria a registro o vicenversa
  - non sono disponibili parole di memoria non-standard

Parole di Memoria Come Operandi

- Si possono classificare gli *ASM-PM* in modo ancora più preciso, in relazione alla possibilità di usare parole di memoria come operandi
  - **Registro-Registro** (RISC): le istruzioni aritmetico-logiche non hanno operandi memoria; le uniche istruzioni che usano operandi memoria sono istruzioni di trasferimento dati che hanno un operando memoria e un operando registro o implicito
  - **Registro-Memoria** (CISC): le istruzioni aritmetico-logiche possono avere al più un operando memoria
  - **Memoria-Memoria** (CISC): le istruzioni aritmetico-logiche possono avere più di un operando memoria

Indirizzamento Diretto-Memoria

- In questa presentazione introduciamo il più semplice modo di indirizzamento per operandi memoria, chiamato *indirizzamento diretto-memoria*, altri verranno trattati in future presentazioni
- Il modo di indirizzamento diretto-memoria prevede che l'identificativo della parola di memoria che costituisce l'operando, venga indicato in modo diretto, esplicito e completo nello specificatore di operando
- Ricordiamo che per le parole standard, gli identificativi sono gli indirizzi di memoria
- L'indirizzamento diretto-memoria permette di indirizzare parole di memoria con modalità analoga a quella con cui, mediante l'indirizzamento diretto-registro, si indirizzano parole di registro

Indirizzamento Diretto-Memoria

- Ci sono diverse possibilità per indicare un indirizzo di memoria in modo diretto nello specificatore di operando
  - rappresentazioni numeriche (di solito espressioni con numeri decimali e/o esadecimali) dei valori ottenuti interpretando le componenti dell'indirizzo di memoria come interi in codifica naturale
  - rappresentazioni simboliche di espressioni o valori numerici ottenuti interpretando le componenti dell'indirizzo di memoria come interi in codifica naturale
  - *label* che sono *legate* all'indirizzo di memoria
- Nel seguito di questa presentazione, mostreremo esempi delle prime due modalità
- La terza modalità, molto comoda, richiede l'utilizzo dell'allocazione statica della memoria che verrà illustrata in una successiva presentazione

Introduzione all'Accesso a Dati in Memoria in MIPS32

- MIPS32 è un *ASM-PM* di tipo Registro-Registro
- Le uniche istruzioni che hanno operandi memoria, sono istruzioni trasferimento dati, da memoria a registro o viceversa
- Poiché l'unico formato di dato per registri è `word`, le istruzioni che accedono a parole di memoria in formato `byte` o `half` effettuano una conversione di dato

Introduzione all'Accesso a Dati in Memoria in MIPS32

- MIPS32-MARS dispone di diversi modi di indirizzamento per operandi memoria, tra cui l'indirizzamento diretto-memoria
- Principali istruzioni di trasferimento da registro a memoria
  - `sw`: copia il contenuto di un registro in una word di memoria
  - `sh`: legge il contenuto di un registro, lo converte ad half mediante **modulo-narrowing** e scrive il dato convertito in una half di memoria
  - `sb`: legge il contenuto di un registro, lo converte a byte mediante **modulo-narrowing** e scrive il dato convertito in un byte di memoria

Introduzione all'Accesso a Dati in Memoria in MIPS32

- Principali istruzioni di trasferimento da memoria a registro
  - `lw`: copia il contenuto di una `word` di memoria in un registro
  - `lh`: legge il contenuto di una `half` di memoria, lo converte a `word` mediante **sign-extension** e scrive il dato convertito in un registro
  - `lhu`: legge il contenuto di una `half` di memoria, lo converte a `word` mediante **zero-extension** e scrive il dato convertito in un registro
  - `lb`: legge il contenuto di un `byte` di memoria, lo converte a `word` mediante **sign-extension** e scrive il dato convertito in un registro
  - `lbu`: legge il contenuto di un `byte` di memoria, lo converte a `word` mediante **zero-extension** e scrive il dato convertito in un registro

Introduzione all'Accesso a Dati in Memoria in MC68000

- Escludendo alcune istruzioni particolari, MC68000 è un *ASM-PM* di tipo Registro-Memoria
- La possibilità di effettuare operazioni aritmetico-logiche con operandi memoria è molto utile in considerazione del numero di registri limitato
- MC68000 dispone di numerosi modi di indirizzamento per operandi memoria, tra cui l'indirizzamento diretto-memoria

Introduzione all'Accesso a Dati in Memoria in MC68000

- Le istruzioni `add`, `sub` e `cmp` ammettono che uno dei loro operandi sia una parola di memoria di uno qualunque dei 3 formati standard
- Le istruzioni di moltiplicazione e divisione ammettono che il loro primo operando sia una `word` di memoria
- MC68000 consente di trasferire dati da memoria a memoria con una singola istruzione: ad esempio l'istruzione `move` ammette che entrambi i suoi operandi siano parole di memoria, di uno qualunque dei 3 formati standard
- In tutte queste istruzioni, per gli operandi memoria sono possibili numerosi modi di indirizzamento, tra cui il diretto-memoria

Esempi di Accesso a Dati in Memoria

- Nei seguenti esempi, si accede a dati in memoria attraverso l'indirizzamento diretto-memoria
- Gli indirizzi di memoria delle parole che contengono i dati, vengono indicati in modo esplicito attraverso rappresentazioni numeriche o simboliche
- In MC68000, la possibilità di usare operandi memoria per molte istruzioni, permette di scrivere codice più conciso ed efficiente
- In MIPS32 si è invece sempre obbligati a "far transitare" i dati provenienti dalla memoria in registri, prima di utilizzarli in operazioni

Esempi di Accesso a Dati in Memoria

- Code1_m68k e Code1_mips mostrano come sommare il contenuto del byte di memoria il cui indirizzo è rappresentato dal numero 1000, a quello di un registro
- Si ricorda che in MIPS32 l'unico formato valido per registri è `word`, quindi l'istruzione che legge il byte di memoria, converte il dato letto al formato `word`; in questo esempio si interpreta il byte letto mediante codifica con segno

Code1_m68k

```
* somma di byte:
; da indirizzo 1000 a d2
 add.b  1000,d2
```

Code1_mips

```
# somma di byte:
# da indirizzo 1000 a s2
 lb     $t0,1000
 add    $s2,$s2,$t0
```

Esempi di Accesso a Dati in Memoria

- Code2_m68k e Code2_mips mostrano come sommare i contenuti di due parole di memoria di lunghezza 16 bit
- In MIPS32 è necessario trasferire entrambe le parole di memoria in registri, e in questo caso i due dati vengono interpretati mediante codifica senza segno
- Gli indirizzi delle due parole di memoria sono rappresentati dai numeri 4096 e 4098 in codifica esadecimale

Code2_m68k

```
* somma di word
; da $1000 a $1002
 move.w $1000,d0
 add.w  d0,$1002
```

Code2_mips

```
# somma di half:
# da 0x1000 a 0x1002
 lhu    $t0,0x1000
 lhu    $t1,0x1002
 add    $t1,$t1,$t0
 sh     $t1,0x1002
```

Esempi di Accesso a Dati in Memoria

- Code3_m68k e Code3_mips sono esempi di trasferimento dati tra due parole di memoria di 32 bit
- Gli indirizzi delle due parole di memoria sono rappresentati dai nomi simbolici *sorgente* e *destinazione*, equivalenti, rispettivamente, ai numeri 41120 e 36620 in codifica esadecimale

Code3_m68k

```
* copia di long:
; da $a0a0 a $8f0c
sorgente        equ $a0a0
destinazione    equ $8f0c
 move.l sorgente,destinazione
```

Code3_mips

```
# copia di word:
# da 0xa0a0 a 0x8f0c
 .eqv sorgente 0xa0a0
 .eqv destinazione 0x8f0c
 lw     $t0,sorgente
 sw     $t0,destinazione
```