

Laboratorio di Architettura degli Elaboratori

Organizzazione dei Dati in Memoria 1

Luca Forlizzi, Ph.D.

Versione 19.1



Luca Forlizzi, 2019

© 2019 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>.

- 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

Dati in Memoria

- I dati che un programma deve poter variare durante l'esecuzione devono essere necessariamente memorizzati e vengono quindi detti *memorizzati*
- Tali dati sono memorizzati in gruppi di bit contenuti in registri o memoria, chiamati in generale *parole*
- L'accesso alle parole contenute in un registro, dette *parole di registro*, avviene attraverso il *nome* del registro
- L'accesso alle parole contenute in memoria, dette *parole di memoria*, avviene attraverso un *indirizzo*

- I registri si differenziano dalle variabili di un *HLL*, in quanto
 - Esiste una quantità fissata di registri, ciascuno con uno o più nomi predefiniti
 - Pertanto i registri non devono essere “dichiarati” in un programma *ASM*: sono automaticamente pronti all’uso
 - Tuttavia i registri non sono inizializzati automaticamente
- Le parole di memoria sono più simili alle variabili di un *HLL*
 - Non esiste una quantità fissata di parole di memoria
 - Esistono costrutti *ASM* analoghi alle dichiarazioni di variabile, per indicare il nome ed il contenuto iniziale di una parola di memoria
- In questa lezione introduciamo l’utilizzo di parole di memoria in *ASM*

Interpretazioni di Dato per Indirizzi

- Ogni parola di memoria di una abstract machine di livello 4 (ma un discorso analogo vale ai livelli 2 o 3) viene identificata mediante un *indirizzo*
- Gli indirizzi utilizzati da un *ASM-PM* vengono a volte chiamati *indirizzi logici* in quanto alcuni computer hanno una caratteristica chiamata *memoria virtuale* che prevede una differenziazione tra gli indirizzi logici utilizzati dai programmi per identificare le parole e gli *indirizzi fisici* utilizzati al livello 1 per effettuare l'accesso ai bit

Interpretazioni di Dato per Indirizzi

- 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 al 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 LAE con indirizzo si intende un indirizzo logico

Interpretazioni di Dato per Indirizzi

- Un indirizzo è un insieme ordinato di A_LEN cifre binarie, dove A_LEN è una costante positiva specifica per ogni computer
- Di solito A_LEN è una potenza di 2
- Gli indirizzi possono a loro volta essere memorizzati in dispositivi di memorizzazione, divenendo quindi essi stessi dati
- In generale, un indirizzo può essere suddiviso in parti, dette *componenti*, ciascuna delle quali viene considerata come stringa binaria

Interpretazioni di Dato per Indirizzi

- La maggior parte degli *ASM-PM*, adottano il *flat memory model*, nel quale un indirizzo ha una sola componente
- Ovvero, nel *flat memory model* ogni indirizzo è una singola stringa binaria di *A_LEN* cifre, e quindi può essere interpretato come un numero in codifica naturale
- I possibili indirizzi sono tutti gli interi positivi compresi tra 0 e $2^{A_LEN} - 1$
- Interpretando gli indirizzi come numeri in codifica naturale, risulta definito un *ordine tra gli indirizzi*

Interpretazioni di Dato per Indirizzi

- Alcuni *ASM-PM*, tra cui Intel386, adottano (a volte opzionalmente) il *segmented memory model*, nel quale ogni indirizzo i ha 2 componenti, e viene quindi indicato mediante una coppia di stringhe binarie (s, o)
- Nel *segmented memory model* l'insieme di tutti gli indirizzi è suddiviso in sottoinsiemi chiamati *segmenti*
 - La prima componente di un indirizzo, detta *selettore di segmento*, identifica il segmento cui appartiene l'indirizzo
 - La seconda componente, detta *offset*, identifica l'indirizzo all'interno del segmento cui appartiene
- Sia i selettori di segmento che gli offset possono essere interpretati come interi positivi
- Interpretando gli offset come interi positivi, risulta definito un *ordine tra gli indirizzi* che appartengono allo stesso segmento

Interpretazioni di Dato per Indirizzi

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

Interpretazioni di Dato per Indirizzi in MC68000

- Di norma M68000 utilizza il flat memory model
- Il segmented memory model può essere utilizzato grazie ad un'estensione, opzionale, chiamata Memory Management Unit (MMU)
- Gli indirizzi sono formati da 32 cifre binarie
- In alcune versioni di M68000, due indirizzi che differiscono solo per le cifre di posizioni maggiori identificano le stesse parole di memoria
- Ad esempio, in MC68000 due indirizzi diversi, ma che hanno uguali le cifre di posizione compresa tra 0 e 23, identificano la stessa parola di memoria

Interpretazioni di Dato per Indirizzi in MIPS32

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

Formati di Dato per Memoria

- Come sappiamo, ogni parola di memoria è identificata da un indirizzo
- Una abstract machine che ha indirizzi formati da A_LEN cifre binarie, ha 2^{A_LEN} diversi indirizzi
- Tuttavia, non tutti gli indirizzi identificano una parola di memoria
- Gli indirizzi che *non* identificano una parola di memoria vengono detti indirizzi *non validi* o *illegali*
- La struttura e le caratteristiche delle parole di memoria, tra cui il loro rapporto con gli indirizzi, sono descritti da uno o più formati di dato

Formati di Dato per Memoria

- Ogni *ASM-PM* ha un formato intero per memoria, chiamato *locazione di memoria* o *byte*, dotato delle seguenti caratteristiche
 - È un formato generale, ovvero usato da molte istruzioni
 - È il formato generale di lunghezza più piccola
 - È un formato presente anche nei livelli di astrazione inferiori: livello 1 e a volte anche al livello 0
 - Al livello 1, questo formato descrive il più piccolo gruppo di bit che può essere trasferito tra memoria e CPU mediante una singola operazione
 - L'insieme di tutti i byte forma una partizione dell'insieme di tutti i bit della memoria

Formati di Dato per Memoria

- Nella quasi totalità dei computer in uso oggi, la lunghezza di un byte è pari ad 8, ma non è sempre stato così
- Ad esempio nel PDP-7, per il quale Ritchie ha iniziato lo sviluppo del C, la lunghezza è pari a 18
- In una implementazione C, le variabili di tipo `char` hanno un numero di bit pari a quello della lunghezza dei byte del computer usato dall'implementazione
- In C Standard tale numero vale almeno 8 , ed è pari al valore della macro `CHAR_BIT` definita nell'header `limits.h`

Formati di Dato per Memoria

- Siano i_1 un indirizzo e i_2 l'indirizzo successore di i_1
- Se i_1 e i_2 sono entrambi indirizzi validi, diciamo che
 - il byte di indirizzo i_1 è il *predecessore* del byte di indirizzo i_2
 - il byte di indirizzo i_2 è il *successore* del byte di indirizzo i_1
 - il byte di indirizzo i_1 e il byte di indirizzo i_2 sono *contigui*

Formati di Dato per Memoria

- Definiamo *area di memoria* (semplicemente *area* quando non vi sono ambiguità) un gruppo G di byte con la seguente proprietà:
 - Nel caso di *segmented memory model* tutti i byte di G appartengono allo stesso segmento
 - Inoltre, se i_{\min} è l'indirizzo minimo tra quelli dei byte in G e i_{\max} è l'indirizzo massimo tra quelli dei byte in G , allora per ogni indirizzo i tale che $i_{\min} \leq i \leq i_{\max}$, i è un indirizzo valido e il byte di indirizzo i appartiene a G
- In altre parole un'area di memoria è un gruppo di byte che non ha "buchi"
- Nel *flat memory model*, l'insieme degli indirizzi di tutti byte di un'area, interpretati come interi positivi, forma un intervallo
- Nel *segmented memory model*, l'insieme degli offset degli indirizzi di tutti byte di un'area, interpretati come interi positivi, forma un intervallo

Formati di Dato per Memoria

- Definiamo
 - *dimensione* di un'area di memoria G , il numero di byte che la compongono
 - *indirizzo* di un'area di memoria G , l'indirizzo minimo tra quelli dei byte in G
 - Se i_{\max} è l'indirizzo massimo tra quelli dei byte di un'area G , il *byte successore di G* , se esiste, è il byte successore del byte (di G) che ha indirizzo i_{\max}
- Di norma, l'insieme di tutti i byte di una abstract machine è costituito da una o più aree di memoria, ciascuna avente dimensione pari ad una potenza di 2

Formati di Dato per Memoria

- Oltre al byte, spesso gli *ASM-PM* hanno altri formati di dato per memoria
- I formati di dato per memoria *generali*, nella maggior parte degli *ASM-PM*, hanno le seguenti caratteristiche
 - la lunghezza del formato è pari a quella di un byte moltiplicata per una potenza di 2, compresa tra 2 e 32
 - le parole di memoria definite dal formato sono aree di memoria
 - la dimensione delle parole di memoria definite dal formato è pari a una potenza di 2, compresa tra 2 e 256
- Nel seguito indichiamo con **F** un formato di dato per memoria generale, con L la sua lunghezza e con D la sua dimensione
- Si ha $D = L \div M$, dove M è la lunghezza di un byte e l'operatore \div calcola il quoziente della divisione intera

Formati di Dato per Memoria

- Dunque se un formato di dato per memoria generale definisce parole di dimensione pari a 4
 - l'indirizzo di una parola è l'indirizzo minimo tra quelli dei byte della parola
 - nel *flat memory model*, una parola p di indirizzo i è formata dai byte che hanno indirizzo i , $i + 1$, $i + 2$, $i + 3$
 - nel *segmented memory model*, una parola p di indirizzo (s, o) è formata dai byte che hanno indirizzo (s, o) , $(s, o + 1)$, $(s, o + 2)$, $(s, o + 3)$

Formati di Dato per Memoria

- In generale, l'insieme di tutte le parole di un formato diverso dal byte non costituisce una partizione della memoria, in quanto un byte può appartenere a 2 o più parole distinte
- Ad esempio, in un *ASM-PM* che adotta il *flat memory model*, un formato di dato per memoria generale potrebbe definire parole di dimensione 2 tali che sia i che $i + 1$ sono indirizzi validi: allora il byte di indirizzo $i + 1$ sarebbe contenuto sia nella parola di indirizzo i che in quella di indirizzo $i + 1$

Formati di Dato per Memoria

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

Formati di Dato per Memoria

- Spesso un vincolo di allineamento per un formato che ha lunghezza L , ha fattore di allineamento pari a $L \div M$, dove M è la lunghezza del formato byte
- Questa condizione garantisce che l'insieme delle parole definite da tale formato costituisca una partizione della memoria
- Ad esempio, in un *ASM-PM* che adotta il *flat memory model* ed ha byte di 8 bit e parole di 16 bit con fattore di allineamento 2:
 - Una parola che ha indirizzo valido i contiene i byte di indirizzi $i, i + 1$
 - Poiché i è valido e il fattore di allineamento è 2, $i - 1$ e $i + 1$ non sono indirizzi di parola validi
 - Quindi nessun'altra parola di 16 bit contiene i byte di indirizzi $i, i + 1$

Formati di Dato per Memoria

- I formati di dato per memoria diversi dal byte definiscono le posizioni dei bit che formano le parole in relazione alle posizioni dei bit all'interno di un byte
- Sia \mathbf{F} un formato di lunghezza L e dimensione $D = L \div M$
- A ciascuno dei byte che formano una parola $P \in \mathbf{F}$ viene assegnata una *posizione* B compresa tra 0 e $D - 1$
- Al bit che ha posizione p_B all'interno del byte di posizione B della parola P , viene assegnata una posizione all'interno di P pari a

$$p_P = B * M + p_B$$

Formati di Dato per Memoria

- Capita sovente, nella programmazione, di dover fare il calcolo inverso, ovvero data la posizione p_P di un bit all'interno di una parola P di un formato \mathbf{F} di lunghezza L , si devono trovare
 - La posizione p_B del bit all'interno del byte in cui è contenuto
 - La posizione B all'interno di P , del byte che contiene il bit di posizione p_P
- Indicando con \div e mod , rispettivamente, gli operatori che calcolano il quoziente e il resto della divisione intera, si ha:
 - $p_B = p_P \text{ mod } M$
 - $B = p_P \div M$

Formati di Dato per Memoria

- Se si vuole accedere ad uno dei byte che formano una parola P , è necessario determinarne l'indirizzo, che naturalmente in relazione con la posizione B del byte all'interno di P
- La relazione tra posizione di un byte all'interno di una parola e l'indirizzo del byte può essere definita in modi diversi in diversi *ASM-PM*
- I due modi più comuni sono chiamati *little-endian* e *big-endian*

Formati di Dato per Memoria

- Nel modo *little-endian*, “l'estremità piccola” di una parola viene prima osservando la memoria per indirizzi crescenti
 - il byte di posizione minima è il byte di indirizzo minimo tra quelli che formano la parola
 - i byte di posizioni crescenti all'interno della parola, hanno indirizzi crescenti
- In dettaglio
 - nel *flat memory model*, se i è l'indirizzo della parola P , l'indirizzo del byte che ha posizione B all'interno di P è

$$i + B$$

- nel *segmented memory model*, se (s, o) è l'indirizzo della parola P , l'indirizzo del byte che ha posizione B all'interno di P è

$$(s, o + B)$$

Formati di Dato per Memoria

- Nel modo *big-endian*, “l'estremità grande” di una parola viene prima osservando la memoria per indirizzi crescenti
 - il byte di posizione minima è il byte di indirizzo massimo tra quelli che formano la parola
 - i byte di posizioni crescenti all'interno della parola, hanno indirizzi decrescenti
- In dettaglio, indicando con $D = L \div M$ il numero di byte che formano una parola P
 - nel *flat memory model*, se i è l'indirizzo di P , l'indirizzo del byte che ha posizione B all'interno di P è

$$i + (D - 1) - B$$

- nel *segmented memory model*, se (s, o) è l'indirizzo della di P , l'indirizzo del byte che ha posizione B all'interno di P è

$$(s, o + (D - 1) - B)$$

Formati di Dato per Memoria

- I termini *little-endian* e *big-endian* derivano, 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

Formati di Dato per Memoria

- La maggior parte (ma non tutti) gli *ASM-PM* organizzano i bit nelle parole o nel modo *little-endian* o in quello *big-endian*
- Esempi di *ASM-PM* che usano il modo *little-endian*: IA-32 (in particolare Intel386), 6502, Z80, Alpha, PDP-11 (per la maggior parte dei formati), VAX
- Esempi di *ASM-PM* che usano il modo *big-endian*: IBM System/360 (e successive architetture), IBM POWER, SPARC (fino alla versione 9 della ISA)
- Esempi di *ASM-PM* che possono essere configurati per usare uno dei due modi: ARM, PowerPC, SPARC (da versione 9 in avanti), PA-RISC, IA-64
- PDP-11 fornisce un esempio di formato di parole di lunghezza 32 in cui i bit sono organizzati in modo diverso sia dal *little-endian* che dal *big-endian*

Formati di Dato per Memoria

- Alcuni *ASM-PM* hanno anche formati di dato per memoria *speciali*, ovvero usati solo da poche istruzioni
- Tali formati possono non essere formati da aree di memoria e/o da byte
- Alcuni formati di dato per memoria speciali possono avere addirittura lunghezza minore di quella del byte
- Tuttavia non godono delle altre caratteristiche del byte, in particolare non sono presenti anche al livello 1
- Presenteremo nelle prossime lezioni alcuni esempi di formati di dato per memoria speciali

Formati di Dato per Memoria in MC68000

- I formati di dato per memoria *generali* in MC68000 sono
 - 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*
- M68000 definisce anche alcuni formati di dato per memoria *speciali*, tra cui i *bit-field*, definiti da MC68020 e versioni successive, che sono sequenze di bit di lunghezza compresa tra 1 e 32 bit, quindi possono essere anche più corti di un byte

Formati di Dato per Memoria in MIPS32

- I formati di dato per memoria in MIPS32 sono
 - `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 MARS l'organizzazione sia del formato `half` che del formato `word` è configurata come *little-endian*

Primo Sguardo all'Accesso alla Memoria

- Tra i diversi *ASM-PM*, si possono distinguere diverse tipologie in relazione alla possibilità di usare parole di memoria come operandi di istruzioni aritmetico-logiche
 - **Registro-Registro**: le parole di memoria non possono essere operandi di istruzioni aritmetico-logiche (che quindi hanno come operandi in prevalenza registri); solo alcune istruzioni di trasferimento dati accedono alla memoria
 - **Registro-Memoria**: le istruzioni aritmetico-logiche possono avere al più una parole di memoria come operando, e quindi è possibile effettuare operazioni tra una parola di registro ed una di memoria
 - **Memoria-Memoria**: tutti gli operandi delle istruzioni aritmetico-logiche possono essere parole di memoria, e quindi è possibile effettuare operazioni tra due o più parole di memoria

Primo Sguardo all'Accesso alla Memoria

- Le istruzioni *ASM* possono accedere ad una parola di memoria specificandone l'indirizzo
- Gli *ASM-PM* permettono di specificare l'indirizzo di una parola di memoria che è operando di una istruzione, attraverso diversi tipi di meccanismo (detti, come vedremo, *modi di indirizzamento*)
- Il più semplice prevede di indicare direttamente nello specificatore di operando, l'indirizzo di tale parola
- Nello specificatore di operando, l'indirizzo è indicato mediante
 - rappresentazioni numeriche (di solito espressioni con numeri decimali e/o esadecimali) dei valori ottenuti interpretando le componenti dell'indirizzo come interi in codifica naturale
 - *label legate* all'indirizzo, come vedremo in seguito
- Presenteremo in seguito altri modi di indirizzamento

Primo Sguardo all'accesso alla Memoria in MIPS32

- MIPS32, come sappiamo, è un *ASM-PM* di tipo Registro-Registro
- Le uniche istruzioni che accedono a parole di memoria, sono istruzioni trasferimento dati da memoria e 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

Primo Sguardo all'accesso alla Memoria in MIPS32

- Principali istruzioni di trasferimento da registro a memoria
 - `sw`: legge il contenuto di un registro e lo scrive 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

Primo Sguardo all'accesso alla Memoria in MIPS32

- Principali istruzioni di trasferimento da memoria a registro
 - `lw`: legge il contenuto di una `word` di memoria e lo scrive 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

Primo Sguardo all'accesso alla Memoria in MC68000

- MC68000, come sappiamo, è un *ASM-PM* di tipo Registro-Memoria
- La possibilità di effettuare operazioni aritmetico-logiche con operandi in memoria è molto utile in considerazione del numero di registri limitato
- `add` e `sub` ammettono che uno dei suoi operandi sia una parola di memoria di qualunque formato
- Le istruzioni di moltiplicazione e divisione ammettono che il loro primo operando sia una `word` di memoria
- MC68000 consente, tramite `move` di effettuare trasferimenti di dato di qualunque formato da memoria a memoria