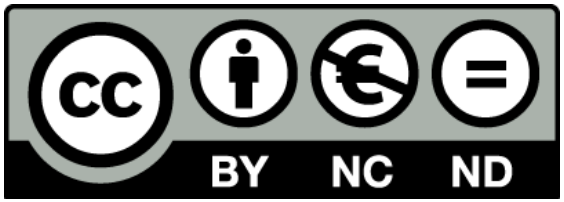
Laboratorio di Programmazione di Sistema

Organizzazione dei Dati in Memoria 1

Luca Forlizzi, Ph.D.

Versione 20.2



Dati in Memoria

- 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

- Ogni dato (ovvero ogni stringa binaria) che un programma utilizza durante l'esecuzione deve essere necessariamente memorizzato in un gruppo di bit
- Affinché la abstract machine possa leggere, scrivere o modificare un dato mediante una singola operazione, il gruppo di bit in cui tale dato è memorizzato deve essere una *parola*
- Le *parole* sono quindi specifici gruppi di bit, che vengono definiti da un *ASM-PM*
- La *lunghezza* di una parola è il numero di bit che la formano

Dati in Memoria

- 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
- In questa presentazione studiamo da vicino la struttura e l'organizzazione dei dati in memoria, e in particolare delle parole di memoria

Indirizzi di Memoria

- Sia $\mathrm{LEN_A}$ un intero positivo, caratteristico di un *ASM-PM*
- Un *indirizzo di memoria* (o semplicemente *indirizzo*) è una stringa binaria di lunghezza $\mathrm{LEN_A}$, potenzialmente associabile ad una parola di memoria
- In alcuni *ASM-PM*, una parola di memoria può avere più di un indirizzo
- Le istruzioni *ASM*, possono usare un indirizzo per identificare una specifica parola all'interno della memoria, come discuteremo più dettagliatamente in seguito

Indirizzi di Memoria

- 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
- Nei computer privi di memoria virtuale, non vi è differenza tra indirizzi logici e indirizzi 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 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 LPS con indirizzo si intende un indirizzo logico

Byte

- Ogni *ASM-PM* ha un insieme di parole di memoria, chiamate *byte*, con le seguenti caratteristiche:
  - Tutti i byte hanno la stessa lunghezza $\mathrm{LEN}_{\mathrm{byte}}$
  - L'insieme di tutti i byte, forma una partizione dell'insieme di tutti i bit della memoria
  - A ciascun byte è associato almeno un indirizzo
  - Ogni indirizzo è associato ad un solo byte

Byte

- Molti *ASM-PM*, tra cui quelli della famiglia M68000, definiscono anche delle parole di registro chiamate *byte* di lunghezza pari a $\mathrm{LEN}_{\mathrm{byte}}$
- Ciò viene fatto, in modo che il contenuto di un byte di memoria possa essere copiato 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

Byte

- Ogni *ASM-PM* ha un formato di dato chiamato *formato byte* che descrive le caratteristiche dell'insieme dei byte di memoria e, se esistono, di registro
- 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, un byte ha lunghezza 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`

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 gruppi di bit di lunghezza $\mathrm{LEN}_{\mathrm{byte}}$, ovvero con byte
- Tipicamente, un byte è il gruppo 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

Interpretazioni di Dato per Indirizzi

- Un indirizzo è una stringa di $\mathrm{LEN_A}$ cifre binarie, dove $\mathrm{LEN_A}$ è una costante positiva specifica per ogni computer
- Di solito $\mathrm{LEN_A}$ è una potenza di 2
- Gli indirizzi possono a loro volta essere memorizzati in parole, divenendo quindi essi stessi dati
- In generale, un indirizzo può essere suddiviso in parti, dette *componenti*, ciascuna delle quali viene considerata come una stringa binaria a se stante

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 $\mathrm{LEN_A}$ cifre, e quindi può essere interpretato come un numero in codifica naturale
- I possibili indirizzi sono tutti gli interi positivi compresi tra 0 e $2^{\mathrm{LEN_A}} - 1$
- 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 quelli della famiglia IA-32, 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 $2^{\mathrm{LEN_A}} - 1$, $i - 1$ 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 tra
    - due indirizzi, nel flat memory model
    - due offset di indirizzi che appartengono allo stesso segmento, nel segmented memory model
  - *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)
- Le implementazioni di MC68000 non hanno MMU, ma possono essere connesse a un coprocessore che fornisce la MMU
- In M68000, gli indirizzi sono formati da 32 cifre binarie
- In alcune versioni di M68000, due indirizzi che differiscono solo per le cifre di posizioni maggiori sono associati allo stesso byte
- Ad esempio, in MC68000, due indirizzi diversi ma che hanno uguali le cifre di posizione compresa tra 0 e 23, sono associati allo stesso byte

Interpretazioni di Dato per Indirizzi in MIPS32

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

Indirizzi e Byte

- Una abstract machine ha indirizzi formati da $\mathrm{LEN_A}$ cifre binarie, e quindi ha $2^{\mathrm{LEN_A}}$ diversi indirizzi
- Tuttavia, non tutti gli indirizzi sono associati ad un byte
- Gli indirizzi *non* associati ad un byte vengono detti indirizzi *non validi* o *illegali*

Indirizzi e Byte

- In un *ASM-PM* che adotta il segmented memory model, diciamo che un byte b che ha indirizzo (s, o) *appartiene* al segmento s
- Quindi un byte b_1, che ha indirizzo (s_1, o_1), e un byte b_2, che ha indirizzo (s_2, o_2), appartengono allo stesso segmento se e solo se $s_1 = s_2$
- Dati due indirizzi validi i_1 e i_2 tali che i_2 è l'indirizzo successore di i_1, 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*

Byte e Altre Parole di Memoria

- Molti *ASM-PM* hanno, oltre ai byte, altre parole di memoria
- Molte parole di memoria sono "costruite" mediante byte
- Per spiegare con precisione la struttura e le caratteristiche di tali parole, è utile definire alcuni concetti relativi a gruppi di bit di memoria che non sono necessariamente parole

Aggregazione di Byte

- Un gruppo G di bit di memoria è un'*aggregazione di byte* se esiste un insieme di byte $\mathrm{B}_G = \{B_0, B_1, \ldots, B_k\}$ tale che
  - Ogni bit di G, è contenuto in un byte dell'insieme B_G
  - Ogni bit che è contenuto in un byte dell'insieme B_G, è contenuto in G
  - Se l'*ASM-PM* adotta il segmented memory model, tutti i byte dell'insieme B_G appartengono allo stesso segmento
- L'insieme B_G viene detto *insieme di byte di G*
- Data un'aggregazione di byte G, definiamo
  - *dimensione* di G (indicata come DIM_G) la cardinalità di B_G
  - *indirizzo* di G (indicato come I_G) l'indirizzo minimo tra quelli dei byte di B_G
  - *lunghezza* di G (indicata come LEN_G) il numero di bit contenuti in G, pari a $\mathrm{DIM}_G \cdot \mathrm{LEN}_{\mathrm{byte}}$

Area di Memoria

- Definiamo *area di memoria* (semplicemente *area* quando non vi sono ambiguità) un gruppo di bit G con le seguenti proprietà:
  - G è un'aggregazione di byte
  - Se $i_{\min}$ e $i_{\max}$ sono, rispettivamente, l'indirizzo minimo e l'indirizzo massimo tra quelli dei byte contenuti in B_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 B_G

Area di Memoria

- In parole povere, un'area di memoria G è un gruppo di bit che non ha "buchi"
- Nel flat memory model
  - l'insieme degli indirizzi di tutti byte di B_G, interpretati come interi positivi, forma un intervallo
  - se $i_{\max}$ è l'indirizzo massimo tra quelli dei byte di B_G, si ha $i_{\max} = \mathrm{I}_G + \mathrm{DIM}_G - 1$
- Nel segmented memory model
  - l'insieme degli offset degli indirizzi di tutti byte di B_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 massimo tra quelli dei byte di B_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 massimo tra quelli dei byte di B_G, il *byte successore di* G, se esiste, è il byte successore del byte (di B_G) che ha indirizzo $i_{\max}$
- Di norma, l'insieme di tutti i bit di una abstract machine è costituito da una o più aree di memoria, ciascuna avente dimensione pari ad una potenza di 2

Parole Standard

- La maggior parte degli *ASM-PM* ha dei formati di dato che definiscono parole di memoria che hanno caratteristiche legate a quelle dei byte
- Una *parole standard* è un gruppo di bit P che ha le seguenti caratteristiche
  - P è una parola (ovvero la abstract machine è in grado di accedere a P mediante un singola operazione)
  - P è un'area di memoria
  - DIM_P (ovvero la quantità di byte che formano P) è pari a una potenza di 2
- Si osservi che anche il byte soddisfa banalmente la definizione di parola standard

Parole Standard

- Molti *ASM-PM*, quindi, hanno dei formati di dato, detti *formati standard*, ciascuno dei quali definisce le parole standard aventi una determinata dimensione
- Il formato byte è uno dei formati standard di un *ASM-PM*
- Valori tipici per la dimensione DIM_P di una parola standard P, sono 1, 2, 4, 8, ma a volte si incontrano anche valori superiori (ad esempio MC68040 e MC68060 hanno parole standard di dimensione 16)

Parole Standard

- Spesso un formato standard definisce anche delle parole di registro aventi medesima lunghezza delle parole di memoria del formato, per consentire 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

Parole Standard

- Una parola standard è un caso particolare di area di memoria, pertanto "eredita" le definizioni di indirizzo, dimensione e byte successore
- Una parola standard P è quindi un'aggregazione di byte: diciamo pertanto che ciascuno dei byte di B_G *appartiene a* o *è contenuto in* P
- L'indirizzo di una parola standard è in relazione con gli indirizzi dei byte in essa contenuti
- Ad esempio, se un formato standard definisce parole standard con dimensione pari a 4 e se P è una di tali parole, si ha che
  - nel flat memory model, se i è l'indirizzo di P, allora P è formata dai byte che hanno indirizzo i, $i+1$, $i+2$, $i+3$
  - nel segmented memory model, se (s, o) è l'indirizzo di P, allora P è formata dai byte che hanno indirizzo (s, o), $(s, o+1)$, $(s, o+2)$, $(s, o+3)$

Parole 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 $i+1$

Allineamento delle Parole 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*
- 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 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 delle Parole 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 un *ASM-PM* che adotta il flat memory model, si consideri un formato **F** che definisce parole standard di dimensione pari a 2, con fattore di allineamento 2:
  - La parola standard 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 **F**
  - Quindi nessun altra parola di **F** contiene i byte di indirizzi i e $i + 1$

Posizioni dei Bit

- 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 i byte
- Ciascun bit b ha posizione $\mathrm{pos\_bit}_B(b)$, all'interno del byte B che lo contiene, compresa tra 0 e $\mathrm{LEN}_{\mathrm{byte}}$-1
- I formati standard diversi dal formato byte, definiscono le posizioni dei bit che formano le loro parole, in relazione alle posizioni dei bit all'interno di un `byte`

Posizioni dei Bit

- Sia **F** un formato che contiene parole standard di dimensione D, con $D > 1$ e di lunghezza pari a $L = D \cdot \mathrm{LEN}_{\mathrm{byte}}$
- A ciascuno dei byte B che appartiene a una parola $P \in \mathbf{F}$ viene assegnata una *posizione* $\mathrm{pos\_byte}_P(B)$ compresa tra 0 e $D - 1$
- Al bit b, che, all'interno del byte B che appartiene a P, ha posizione $\mathrm{pos\_bit}_B(b)$, viene assegnata una posizione all'interno di P come segue

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

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

Posizioni dei Bit

- 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**, si devono 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:
  - $\mathrm{pos\_bit}_B(b) = \mathrm{pos\_bit}_P(b) \mod \mathrm{LEN}_{\mathrm{byte}}$
  - $\mathrm{pos\_byte}_P(B) = \mathrm{pos\_bit}_P(b) \div \mathrm{LEN}_{\mathrm{byte}}$

Posizioni dei Bit

- In molte applicazioni è importante accedere individualmente ai bit di una parola P che sono contenuti in uno specifico byte
- Sia b un bit di P e sia B il byte, contenuto in P, che contiene b: è opportuno poter calcolare l'indirizzo di B a partire da $\mathrm{pos\_bit}_P(b)$ e dall'indirizzo di P
  - Le formule precedenti mettono in relazione $\mathrm{pos\_bit}_P(b)$ con $\mathrm{pos\_byte}_P(B)$
  - Resta quindi da conoscere la relazione tra $\mathrm{pos\_byte}_P(B)$ e l'indirizzo di B

Endianness delle Parole Standard

- La relazione tra la posizione di un byte all'interno di una parola e l'indirizzo del byte viene definita in modi diversi, da *ASM-PM* diversi
- I due modi più comuni sono chiamati *little-endian* e *big-endian*
- Per capire i due 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 delle Parole Standard

- Nel modo *little-endian*, "l'estremità piccola" di una parola è quella che si osserva per prima
  - il byte di indirizzo minimo tra quelli che formano la parola è il byte che ha posizione minima
  - i byte di 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 delle Parole Standard

- Nel modo *big-endian*, invece, "l'estremità grande" di una parola è quella che si osserva per prima
  - il byte di indirizzo minimo tra quelli che formano la parola è il byte che ha posizione massima
  - i byte di 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 delle Parole 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 delle Parole Standard

- La maggior parte (ma non tutti) gli *ASM-PM* organizzano i bit nelle parole standard, o nel modo *little-endian* o in quello *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 Speciali

- Le parole standard sono di gran lunga le parole di memoria più utilizzate nella programmazione *ASM* e infatti, in diversi *ASM-PM*, tutte le parole di memoria sono parole standard
- Tuttavia alcuni *ASM-PM* definiscono anche parole di memoria che non soddisfano la definizione di parola standard
- Chiamiamo *parola di memoria speciale* una parola di memoria che non è una parola standard
- In modo simile, alcuni *ASM-PM* definiscono anche parole di registro diverse da quelle contenute in un formato standard, chiamate *parole di registro speciali*

Parole Speciali

- Chiamiamo *formato speciale* un formato che definisce un insieme di parole (di registro o di memoria) speciali
- Alcuni formati speciali definiscono parole speciali sia di registro che di memoria, tutte della stessa lunghezza, al fine di rendere possibili trasferimenti efficienti del contenuto di tali parole da registri a memoria e viceversa

Parole Speciali

- Parole speciali possono essere costituite da aggregazioni di byte che non formano un'area di memoria
- Un altro esempio particolarmente significativo è quello dei *bit-fields*, che sono gruppi di bit che non formano un'aggregazione di byte
  - I bit-fields di memoria, non sono identificati univocamente da un indirizzo, proprio perché non formano aggregazioni di byte; per identificarli sono necessarie, oltre ad un indirizzo, informazioni aggiuntive
  - Alcuni bit-fields possono avere lunghezza inferiore a quella dei byte: sono le uniche parole ad avere questa proprietà

Formati Standard e Speciali

- I formati di dato di un *ASM-PM* che sono usati da molte istruzioni, vengono detti *generali*
- Tipicamente, i formati generali di un *ASM-PM* sono tutti formati standard; ovvero i formati speciali sono utilizzati solo da poche istruzioni, che eseguono operazioni particolari, le quali necessitano appunto di operare su parole definite in modo diverso rispetto alle parole standard
- Tuttavia, non tutti i formati standard sono generali: alcuni *ASM-PM* hanno formati standard che sono usati solo da poche istruzioni
- Ad esempio, in MIPS32, il formato byte non è un formato generale

Formati di Dato in MC68000

- MC68000 ha 3 formati standard, che già conosciamo, tutti 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 alcuni formati speciali, usati da particolari istruzioni; ne incontreremo alcuni in future presentazioni
- Versioni successive di M68000 hanno ulteriori formati speciali, 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 alcuni formati speciali, usati da particolari istruzioni; ne incontreremo alcuni in future presentazioni
- Versioni successive di MIPS32 hanno ulteriori formati speciali, tra cui *bit-field*

Introduzione all'Accesso a Dati in Memoria

- In relazione alla possibilità di usare parole di memoria ordinarie come operandi di istruzioni aritmetico-logiche, si possono classificare gli *ASM-PM* in diverse tipologie
  - **Registro-Registro**: le parole di memoria ordinarie non possono essere operandi di istruzioni aritmetico-logiche (che quindi hanno, in prevalenza, registri come operandi); solo alcune istruzioni di trasferimento dati accedono alla memoria
  - **Registro-Memoria**: le istruzioni aritmetico-logiche possono avere al più una parola di memoria ordinaria come operando
  - **Memoria-Memoria**: tutti gli operandi delle istruzioni aritmetico-logiche possono essere parole di memoria ordinarie

Introduzione all'Accesso a Dati in Memoria

- Le istruzioni *ASM* possono accedere ad una parola di memoria ordinarie specificandone l'indirizzo
- La specifica dell'indirizzo è l'unico modo di accedere a parole di memoria ordinarie
- Gli *ASM-PM* permettono di specificare l'indirizzo di una parola di memoria, attraverso diversi tipi di meccanismo chiamati *modi di indirizzamento*
- In questa presentazione introduciamo il più semplice dei modi di indirizzamento per operandi in memoria, chiamato *indirizzamento diretto-memoria*
- Altri modi di indirizzamento verranno trattati in future presentazioni

Introduzione all'Accesso a Dati in Memoria

- Il modo di indirizzamento diretto-memoria prevede di indicare l'indirizzo di una parola in modo esplicito nello specificatore di operando
- Ci sono due diverse possibilità per indicare l'indirizzo in modo esplicito nello specificatore di operando
  - 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* che sono *legate* all'indirizzo, come descriveremo in una prossima presentazione

Introduzione all'Accesso a Dati in Memoria in MIPS32

- MIPS32 è un *ASM-PM* di tipo Registro-Registro
- Le uniche istruzioni che accedono a parole di memoria ordinarie, 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 ordinarie in formato `byte` o `half` effettuano una conversione di dato

Introduzione all'Accesso a Dati in 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

Introduzione all'Accesso a Dati in 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

Introduzione all'Accesso a Dati in Memoria in MC68000

- MC68000 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 loro operandi sia una parola di memoria ordinaria 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 ordinaria, di uno qualunque dei 3 formati standard

Esempi di Accesso a Dati in Memoria

- Presentiamo alcuni esempi di accesso a dati in memoria, in cui gli indirizzi delle parole di memoria usate come operandi vengono indicati in modo esplicito attraverso rappresentazioni numeriche
- Nella prossima presentazione, verrà illustrato come usare, invece, delle label per indicare in modo esplicito indirizzi di memoria
- In MC68000 il fatto che molte istruzioni possano usare parole di memoria come operandi, 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, ad un registro, il contenuto del byte di memoria il cui indirizzo è indicato dal numero 1000, mediante rappresentazione decimale
- 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 una parola di memoria ad un'altra (entrambe formate da 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 indicati dai numeri 4096 e 4098, mediante rappresentazione 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    $t1,0x1000
 add    $t1,$t1,$t0
 sh     $t1,0x1000
```

Esempi di Accesso a Dati in Memoria

- Code3_m68k e Code3_mips mostrano come copiare il contenuto di una parola di memoria in un'altra (entrambe formate da 32 bit)
- Gli indirizzi delle due parole di memoria sono indicati dai numeri 41120 e 36620, mediante rappresentazione esadecimale

Code3_m68k

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

Code3_mips

```
# copia di word:
# da 0xa0a0 a 0x8f0c
 lw     $at,0xa0a0
 sw     $at,0x8f0c
```

Sessione di Esercizi: Organizzazione dei Dati in Memoria

- Svolgere il gruppo di esercizi **Organizzazione dei Dati in Memoria** su *Edu99*
- Per ulteriori spiegazioni sui contenuti degli esercizi, si vedano
  - esempio *Absolute Beginners* su *Edu99*
  - **[M68000]**
  - **[MIPS32]**