1 2 3 Idee per il tuo futuro

Paolo Ollari

Corso di sistemi e reti

per Informatica

Architetture e network

TECNOLOGIA ZANICHELLI

Paolo Ollari

Corso di sistemi e reti

per Informatica

Architetture e network

Realizzazione editoriale:
– Coordinamento editoriale: Matteo Fornesi
– Collaborazione redazionale, impaginazione e illustrazioni: Conedit Libri, Cormano (MI)
– Progetto grafico: Editta Gelsomini
– Segreteria di redazione: Deborah Lorenzini

Copertina:
– Progetto grafico: Miguel Sal & C., Bologna
– Realizzazione: Roberto Marchetti
– Immagine di copertina: Artwork Miguel Sal, Bologna

Prima edizione: gennaio 2013

L'impegno a mantenere invariato il contenuto di questo volume per un quinquennio (art. 5 legge n. 169/2008) è comunicato nel catalogo Zanichelli, disponibile anche online sul sito www.zanichelli.it, ai sensi del DM 41 dell'8 aprile 2009, All. 1/B.

Zanichelli garantisce che le risorse digitali di questo volume sotto il suo controllo saranno accessibili, a partire dall'acquisto dell'esemplare nuovo, per tutta la durata della normale utilizzazione didattica dell'opera. Passato questo periodo, alcune o tutte le risorse potrebbero non essere più accessibili o disponibili: per maggiori informazioni, leggi my.zanichelli.it/fuoricatalogo

File per diversamente abili
L'editore mette a disposizione degli studenti non vedenti, ipovedenti, disabili motori o con disturbi specifici di apprendimento i file pdf in cui sono memorizzate le pagine di questo libro. Il formato del file permette l'ingrandimento dei caratteri del testo e la lettura mediante software screen reader.
Le informazioni su come ottenere i file sono sul sito www.zanichelli.it/diversamenteabili

Suggerimenti e segnalazione degli errori
Realizzare un libro è un'operazione complessa, che richiede numerosi controlli: sul testo, sulle immagini e sulle relazioni che si stabiliscono tra essi. L'esperienza suggerisce che è praticamente impossibile pubblicare un libro privo di errori. Saremo quindi grati ai lettori che vorranno segnalarceli.
Per segnalazioni o suggerimenti relativi a questo libro scrivere al seguente indirizzo:

lineazeta@zanichelli.it

Le correzioni di eventuali errori presenti nel testo sono pubblicate nel sito www.zanichelli.it/aggiornamenti

Zanichelli editore S.p.A. opera con sistema qualità
certificato CertiCarGraf n. 477
secondo la norma UNI EN ISO 9001:2008

Paolo Ollari

Corso di sistemi e reti

per Informatica

Architetture e network

TECNOLOGIA ZANICHELLI

Indice

SEZIONE A

Struttura, architettura e componenti dei sistemi di elaborazione

A1 Architettura di von Neumann

A2 Architetture Intel

A3 Assembly x-86

A4 Assembly con Debug.exe

A5 Editare un programma

SEZIONE B

Reti. Standard di riferimento, organizzazione in livelli, reti locali e geografiche, protocolli

B1 Reti di calcolatori

B2 Modelli per le reti di calcolatori

A

Struttura, architettura e componenti dei sistemi di elaborazione

La storia del calcolatore ha inizio in epoca moderna, già a partire dalla metà del 1600, in Francia, con **Blaise Pascal** (1623-1662), che ideò il primo dispositivo manuale di calcolo aritmetico. Il primo tentativo di calcolatore programmabile fu invece ad opera del britannico **Charles Babbage** (1791-1871), i cui programmi furono scritti dalla programmatrice **Ada Lovelace** (1815-1852). Si deve attendere però il '900 per avere il primo effettivo calcolatore programmabile a relè, probabilmente lo Z1 del costruttore tedesco **Konrad Zuse** (1938). Il più significativo calcolatore elettronico invece fu l'Eniac degli statunitensi **Eckert-Mauchley** (1944), da cui si sviluppò la moderna tecnologia dei calcolatori elettronici, dopo la svolta dell'Ias di **John von Neumann** (1952) che introdusse, tra le altre innovazioni, l'uso della numerazione binaria. In seguito, un impulso fondamentale alla tecnologia costruttiva fu l'invenzione del circuito integrato in silicio da parte del premio Nobel statunitense **Robert Noyce** (1958, poi fondatore di Intel Corporation), che aprì la strada ai calcolatori di terza generazione, dopo quella delle valvole e quella dei transistor.

La struttura tipica di un calcolatore elettronico assume quindi la forma attuale in base ad almeno due svolte tecnologiche fondamentali, un modello costruttivo storico riconducibile allo scienziato ungherese naturalizzato statunitense John von Neumann (**architettura di von Neumann**) risalente agli anni 1940-50, e l'invenzione del microprocessore da parte del fisico italiano **Federico Faggin** (microprocessore Intel 4004), risalente al 1971.

Attraverso lo schema dell'architettura di von Neumann vedremo di chiarire i componenti base di un calcolatore elettronico, le loro interazioni e quindi i principi di funzionamento.

A1 Architettura di von Neumann

Il primo programma

Il primo programma per un calcolatore (1842) fu scritto dalla matematica britannica Augusta Ada Byron, nota come Ada Lovelace e figlia del poeta Lord Byron.

Destinato alla macchina analitica progettata da Charles Babbage, Ada Lovelace ideò un algoritmo per il calcolo dei numeri di Bernoulli. Implementò il programma attraverso le sole 4 istruzioni aritmetiche che costituivano il set delle istruzioni della macchina analitica di Charles Babbage, utilizzando come unità di input delle rudimentali schede perforate in legno.

Il programma compare nell'articolo "Sketch of The Analytical Engine Invented by Charles Babbage By L. F. MENABREA of Turin, Officer of the Military Engineers from the Bibliothèque Universelle de Genève, October, 1842, No. 82, With notes upon the Memoir by the Translator ADA AUGUSTA, COUNTESS OF LOVELACE".

In suo onore il Dipartimento della Difesa degli Stati Uniti ha chiamato **Ada** l'omonimo linguaggio di programmazione (1980).

Tra le varie classificazioni che nel tempo sono state formulate circa la fisionomia dei calcolatori, una è sopravvissuta con successo, data la sua semplicità e sinteticità (**tassonomia di Flynn**, 1972). Essa si basa su due concetti chiave: sequenze di istruzioni e sequenze di dati, a seconda che il calcolatore sia in grado di gestire solo una sequenza o più sequenze nello stesso tempo.

Si hanno quindi le seguenti tipologie di calcolatori, in base alle combinazioni:

- **SISD**: Single Instruction, Single Data, il modello originale, che equivale alla macchina di von Neumann.
- **SIMD**: Single Instruction, Multiple Data, modello in cui alcune singole istruzioni possono computare più sequenze di dati.
- **MISD**: Multiple Instruction, Single Data, modello senza alcuna implementazione reale
- **MIMD**: Multiple Instruction, Multiple Data, modello in cui più unità di calcolo agiscono su più sequenze di dati nello stesso tempo.

In effetti, per capire la classificazione, bisognerebbe considerare un istante di tempo come unità, e quindi verificare se, in quella unità di tempo «congelata», operano una o più istruzioni su una o più sequenze di dati.

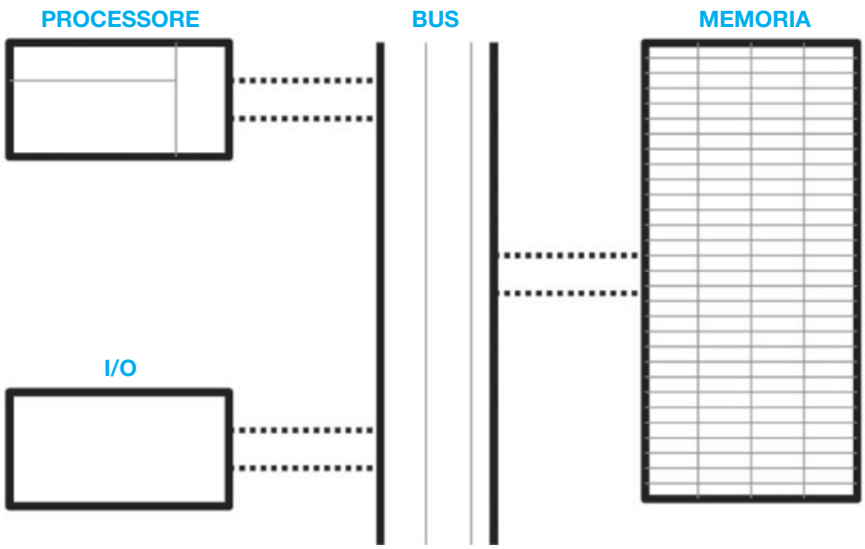
Il modello di von Neumann mette in chiaro che il funzionamento della macchina deve basarsi sul concetto di programma da eseguire, ovvero stabilisce con chiarezza le funzioni tipiche del software (cioè i programmi intesi come sequenze di istruzioni per ottenere risultati) e dell'hardware (cioè elementi elettronici intesi come unità materiali che eseguono le istruzioni del software). Questo concetto fu mutuato dagli studi teorici del britannico Alan Turing (**macchina di Turing**, 1936).

Per avvicinare la parte software a quella hardware, von Neumann stabilisce anche che il sistema di codifica dell'informazione da usare deve essere quello su base binaria, cioè il software deve essere rappresentato con **il sistema numerico in base due**, cosicché per l'hardware sia più immediato decodificare i programmi, basandosi sulle differenze di potenziale elettrico a due soli livelli.

Da un punto di vista funzionale, infine, von Neumann propone una struttura di tipo modulare per le unità hardware, e non strettamente interconnessa come in passato: diverse unità con compiti specifici devono interagire tra loro in modo sincronizzato tramite un modulo di collegamento, il **bus.**

Il modello di von Neumann prevede le seguenti unità funzionali:

- **Processore**, cioè l'unità di calcolo che esegue le istruzioni del sw;
- **Memoria**, cioè il contenitore del software e dei dati;
- **Input/Output**, cioè i moduli attraverso i quali fornire sw e dati (input) o raccogliere i risultati (output);
- **Bus**, cioè l'elemento di interconnessione comune delle suddette unità funzionali.



Bisogna ricordare che tale modello è sopravvissuto praticamente intatto fino alla realizzazione dei moderni calcolatori. L'ultimo calcolatore che ricorda il modello di von Neumann in modo ancora significativo è l'Intel 80486 (1989). In questi casi la macchina di von Neumann è una macchina puramente SISD (a eccezione dei moduli di calcolo in virgola mobile o **FPU**, da tempo in dotazione ai processori moderni).

In seguito il modello fu aggiornato, ma solo relativamente alla scomposizione dei due flussi di programma e dati, che furono gestiti in modo separato dal modello successivo denominato **Architettura Harvard**. Inoltre i moderni calcolatori contengono istruzioni e unità di calcolo che fanno pendere il modello SISD di von Neumann verso modelli misti SISD-SIMD, come si vedrà più oltre.

1 Memoria

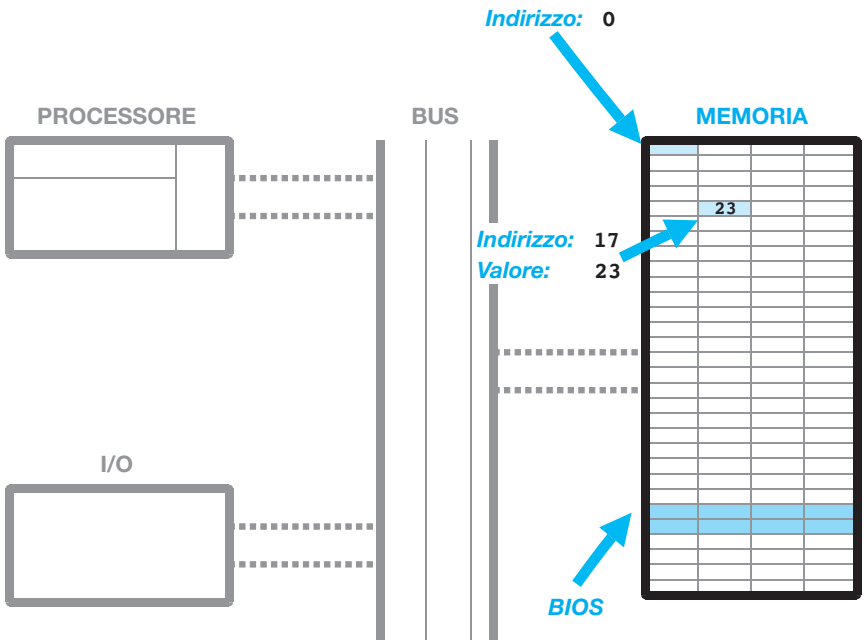
La memoria, detta normalmente **memoria principale** (per distinguerla da altri tipi di memorie dette secondarie), è un contenitore di celle ordinato. Nelle celle di memoria vengono immesse o prelevate le istruzioni del software e i dati di input e di output. Ogni cella è ampia un byte e ogni cella possiede un indirizzo (**address**).

Gli indirizzi delle celle partono da zero e l'indirizzo dell'ultima cella coincide con il numero totale di celle della memoria (meno uno, dato che gli indirizzi partono da zero); l'insieme di tutte le celle di una memoria è detto **spazio degli indirizzi** o **spazio di indirizzamento** della memoria.

L'ampiezza dello spazio di indirizzamento di una memoria principale è

determinata dall'ampiezza del bus indirizzi: se il numero delle linee del bus indirizzi vale n, lo spazio di indirizzamento di quella memoria vale 2^n, e gli indirizzi di quella memoria vanno da 0 a $2^n - 1$.

Il valore nella cella di memoria viene conservato fintanto che la memoria è alimentata, cosicché la memoria è un tipo di **memoria volatile**, cioè perde i suoi contenuti tutte le volte che la macchina viene spenta.



La memoria è realizzata in gran parte in tecnologia **RAM** (*Random Access Memory*) che, per non perdere il contenuto, deve essere sempre «rinfrescata» con un segnale elettrico a frequenza costante per tutto il tempo in cui il sistema è alimentato (refresh della memoria).

I bit nelle celle di RAM, infatti, sono dei microcondensatori ed è per ciò che questo tipo di RAM è nota anche come **DRAM** (*Dynamic Ram*).

I contenuti delle celle di **DRAM** vengono modificati continuamente durante il funzionamento del calcolatore (con nuovi programmi e dati). Il processo di lettura e scrittura delle celle di **DRAM** non è immediato ma necessita di un tempo tecnico detto **tempo di accesso** alla memoria.

Il tempo di accesso alla memoria RAM è decisamente alto rispetto ai tempi presenti nelle attività di un calcolatore (esempio: il tempo dell'esecuzione delle istruzioni nel processore). Questa situazione è detta **collo di bottiglia di von Neumann**.

All'interno della memoria di sistema deve però trovare posto una speciale area che non perde i valori dopo lo spegnimento. Infatti il sistema, per potersi avviare (fase di **bootstrap**), deve immettere sul bus le istruzioni iniziali per configurare i dispositivi di base come video e tastiera (fase di **POST**, *Power On Self Test*) e caricare i programmi del sistema operativo da una memoria secondaria, come per esempio un disco.

Quest'area è riservata all'interno dello spazio di indirizzamento, pur conservando la struttura tipica di indirizzo e cella, ed è denominata, genericamente, **BIOS** (*Basic Input/Output System*).

Unità di misura dell'informazione: byte, kB, MB, GB, TB

I multipli del byte sono comunemente utilizzati in informatica come unità di misura dell'informazione.

Bisogna ricordare che la definizione standard del primo multiplo del byte, ovvero il **kilobyte** è 1 kilobyte = 10^3 byte = 1000 byte, così come dettato dal SI (Sistema Internazionale), dall'IEEE (Institute of Electrical and Electronics Engineers) e dall'ISO (International Organization for Standardization).

Inoltre la stessa convenzione vuole che il simbolo del kilobyte sia **kB**, con la k minuscola (indica kilo, ovvero 1000) e la B maiuscola (indica byte, mentre la b minuscola indica bit).

La definizione non standard e vanamente sconsigliata di 1 kB = 2^{10} byte = 1024 byte, dovrebbe essere indicata con la sigla **KiB** (ovvero **kibibyte**).

Analogamente i successivi multipli del kilobyte sono:

megabyte (**MB**) = 10^6 byte (1 milione di byte)
gigabyte (**GB**) = 10^9 byte (1 miliardo di byte)
terabyte (**TB**) = 10^{12} byte (1 bilione di byte)

I relativi multipli non standard:

mebibyte (**MiB**) = 2^{20} byte
gibibyte (**GiB**) = 2^{30} byte
tebibyte (**TiB**) = 2^{40} byte

Per contrastare il ritardo strutturale dell'accesso alla memoria DRAM, si può realizzare la RAM con tecnologia statica o **SRAM** (*Static Ram*), dove ai microcondensatori si sostituiscono micro flip-flop. La SRAM però è più costosa e occupa troppo spazio a parità di cella, per cui il suo uso è limitato a speciali memorie di transito – tra memoria e processore – denominate **memorie Cache**.

Le memorie DRAM vengono prodotte in banchi di dimensione fissa e montati sulla piastra madre del calcolatore su schede **DIMM** (*Dual In-line Memory Module*) a due facce. La tecnologia più diffusa (2011) è la cosiddetta **DDR3 SDRAM** (*Triple Data Rate Synchronous Dynamic Ram*).

Le regioni di memoria che contengono il BIOS sono realizzate in tecnologia **ROM** (*Read Only Memory*, come per esempio memorie **Flash ROM**), che consente alle celle di mantenere il contenuto anche in assenza di alimentazione, cioè a sistema spento. Il codice e i programmi contenuti in maniera non volatile nella memoria centrale sono detti **firmware** (**Fw**).

Buona parte della memoria ROM è realizzata, in realtà, con tecnologia **EPROM** o **EEPROM**, cioè memorie che, pur mantenendo i dati nelle celle quando manca l'alimentazione, possono essere riprogrammate, cioè modificate con nuovi valori attraverso procedure speciali.

In questo modo le caratteristiche di avvio e alcune caratteristiche funzionali del sistema possono essere modificate e/o aggiornate tramite la modifica dei parametri tramite il programma di **setup del BIOS** o la sostituzione effettiva di tutto il BIOS tramite speciali procedure di configurazione.

2 Bus

Il bus è l'unità di interconnessione tra i moduli del modello di von Neumann. Esso si presenta come un fascio ordinato di linee, ognuna delle quali può assumere il significato di un bit, cioè di un valore binario. Si dice che i moduli processore, memoria e input/output si «affacciano» sul bus, ovvero essendovi collegati, possono impostare, prelevare o modificare i valori presenti sulle linee che lo compongono.

Gran parte dell'attività di un calcolatore elettronico si riduce, infatti, a trasferimenti di bit tra i moduli del modello di von Neumann: trasferimenti che vedono il processore come soggetto (**Master**), memoria e I/O come oggetti (**Slave**) e il bus come veicolo: da processore a memoria, da memoria a processore, da processore a I/O e da I/O a processore (i casi di trasferimento tra I/O e memoria sono visti come un caso speciale, per ora). Considerando come soggetto master il processore, un'operazione che trasporta un dato dal processore alla memoria (o all'I/O) è detta operazione di **scrittura** (**Write**), mentre se il verso è opposto, da memoria o I/O verso il processore, l'operazione è detta di **lettura** (**Read**).

Per gestire correttamente i trasferimenti, il bus è scomponibile in tre sottoinsiemi ordinati di linee, denominati **Address bus** (**ABus**) o bus indirizzi, **Data bus** (**DBus**) o bus dei dati e **Control bus** (**CBus**) o bus di controllo.

L'ordine e il riempimento dei bit

L'**ordine dei bit** è un concetto informatico che si ritrova in almeno due contesti: la trasmissione di bit lungo una linea seriale o la rappresentazione binaria di un valore numerico.

In ogni caso, data una sequenza di bit, si usano le sigle **MSB** (*Most Significant Bit*, bit con peso maggiore) e **LSB** (*Least Significant Bit*, bit con peso minore).

L'LSB di un gruppo di bit è il sottoinsieme ordinato dei bit che hanno peso inferiore (peso minimo 0 significa $2^0 = 1$), ovvero i bit che vengono ordinatamente trasmessi per primi su una linea seriale. Viceversa l'MSB.

Dato un valore numerico, per esempio 173, la cui rappresentazione binaria vale 10101101, si dice che il gruppo di 4 bit 1101 è LSB, mentre il gruppo di 4 bit 1010 è MSB.

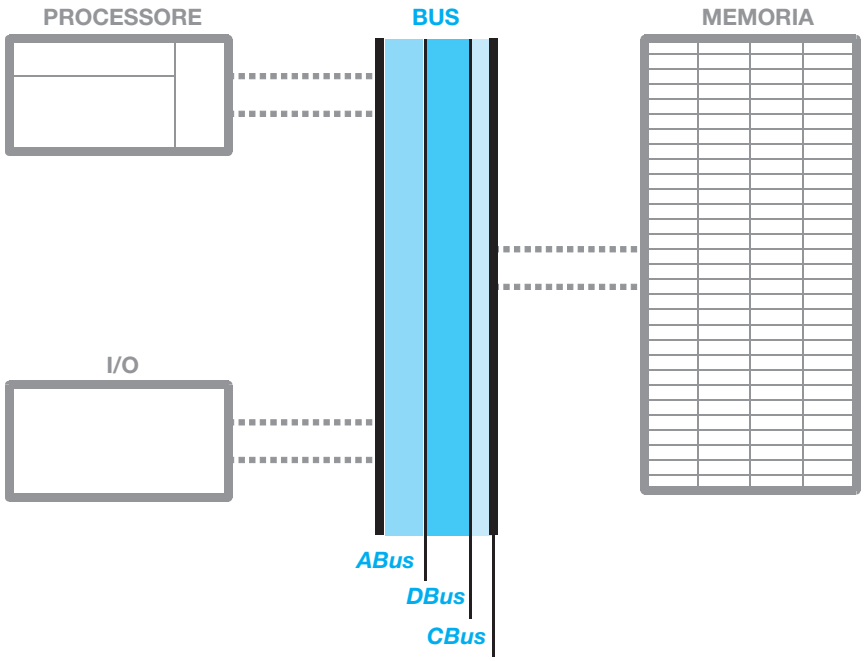
Infatti i pesi dei bit, da sinistra verso destra, sono 7,6,5,4,3,2,1,0.

Se però lo stesso valore fosse trasmesso su una linea seguendo l'ordine della rappresentazione da sinistra a destra, in questo caso LSB e MSB si invertono.

Quando invece viene dato un contenitore di bit (per esempio **nibble** 4 bit, **byte** 8 bit, **word** a 16 bit, ecc.), un valore che deve essere rappresentato in quel contenitore deve essere **riempito a sinistra** (padding) di tanti bit a zero quanti ne servono per esaurire il contenitore.

Dato un byte e una word, il valore 24, con rappresentazione binaria 11000, è sottoposto ai seguenti riempimenti:

byte: **000**11000
word: **0000000000**11000



In questo modo, se si predispongono alcune linee del CBus opportunamente, per esempio prevedendo una linea che specifica la direzione del trasferimento (memoria-processore o I/O-processore), una che specifica il verso del trasferimento (lettura o scrittura) e una che indica se il trasferimento è completato, il trasferimento di una certa quantità di informazione (sul DBus) può essere diretto nel posto giusto (all'indirizzo sull'ABus) in modo completamente sincronizzato (tramite le linee sul CBus).

Considerando che il bus possiede un proprio «orologio» che ne scadenza in modo costante le operazioni nel tempo (in MHz, detto **clock di bus**), si usano le seguenti linee di controllo (sul CBus) per gestire i trasferimenti: **I/O-$\overline{\text{Mem}}$**, **R/$\overline{\text{W}}$**, **$\overline{\text{Wait}}$** (la convenzione vuole che la sigla sopralineata indichi valore di bit a 0).

La linea $\overline{\text{Wait}}$ (attesa) indica trasferimento completato (1) o trasferimento in corso (0), situazione che dimostra il collo di bottiglia di von Neumann: i tempi di accesso alla memoria sono più lunghi dei tempi di elaborazione del processore.

Analogamente, se sulla linea I/O-$\overline{\text{Mem}}$ troviamo 1, il trasferimento riguarda processore e I/O; se troviamo 0, il trasferimento riguarda processore e memoria e se sulla linea R/$\overline{\text{W}}$ troviamo 1, si tratta di un'operazione di lettura (se 0, scrittura).

Le quantità delle linee dell'ABus e del DBus, non necessariamente coincidenti, dipendono dalle caratteristiche specifiche del processore e, comunque, sono spesso potenze di due (4, 8, 16, 32, 64 linee, anche se in alcuni processori troviamo 20, 36 o 80). In generale la dimensione dell'ABus specifica la quantità di memoria raggiungibile dai programmi (spazio di indirizzamento) e si calcola elevando 2 al numero di linee dell'ABus. La dimensione del DBus, invece, rappresenta il grado di parallelismo del processore, ovvero la massima quantità di dati che è in grado di elaborare in un solo trasferimento di bus.

ESEMPIO

Trasferimenti sul bus: memoria

Dato un modello di bus con 8 linee di DBus, 8 linee di ABus e 3 linee di CBus, si voglia indicare lo stato del bus al completamento del trasferimento del valore 23 alla cella di indirizzo 17 della memoria.

Indicare la sequenza dei valori binari delle linee del bus se il CBus ha le seguenti linee: I/O-$\overline{Mem}$, R/$\overline{W}$, $\overline{Wait}$.

Prima di tutto si trasformano i valori decimali in binario, riempiendo i risultati al «byte» (cioè aggiungendo tanti zeri a sinistra fino a ottenere sequenze di 8 bit):

(23)d=(10111)b=(**00010111**)b (valore da trasferire)
(17)d=(10001)b=(**00010001**)b (indirizzo di destinazione)

Quindi si impostano i valori sulle linee del CBus in accordo con la richiesta:

I/O-$\overline{Mem}$ = **0** (indica la direzione processore-memoria)
R/$\overline{W}$ = **0** (indica verso di scrittura)
$\overline{Wait}$ = **1** (indica operazione completata)

In definitiva:

DBus	*ABus*	*CBus*		
76543210	*76543210*	*I/O-$\overline{Mem}$*	*R/$\overline{W}$*	*$\overline{Wait}$*
00010111	**00010001**	**0**	**0**	**1**

All'interno della scheda madre (**Motherboard**) di un calcolatore è abbastanza complicato isolare la sezione del bus di sistema.

In effetti l'insieme di circuiti, linee e chip dedicati al bus di sistema viene indicato con il termine complessivo di **Chipset**, che viene fornito e montato in base alle specifiche del processore utilizzato.

Il Chipset realizza il bus di sistema tramite l'interconnessione di due aree distinte normalmente denominate **NorthBridge** (che si occupa della connessione processore-memoria) e **SouthBridge** (dedicato alle connessioni tra processore e sezione di I/O).

Le tecnologie con cui viene realizzato il bus di sistema sul classico PC Intel si sono evolute dallo standard **ISA** (DBus a 8 bit e clock a 8,33 MHz), allo standard **EISA** (DBus a 16 bit e clock a 8,33 MHz), allo standard **PCI** (*Peripheral Component Interconnect*, DBus a 32 bit e clock a 33 MHz).

Nel tempo sono comparsi bus interni dedicati per veicolare i dati della scheda video e del processore, a partire dal VESA fino al più recente AGP.

Allo stato attuale la tecnologia di bus più diffusa è il PCI a 32 bit, 33MHz e 133MBytes/s e il PCI-X (32-64bit, 66MHz, da 528 MB/s a 1035 MB/s).

L'evoluzione più recente in termini di tecnologie di bus è il **PCI Express**, per ora utilizzato come sostituto di AGP come bus dedicato per le schede video. Altri tipi di bus hanno avuto alterne fortune commerciali, tra cui Microchannel, lo SCSI e il PMCIIA dedicato ai dispositivi portatili.

3 Input/output

La sezione di **input/output** di un calcolatore è dedicata all'acquisizione dei dati e dei programmi, e alla rappresentazione degli stessi in varie forme, dal video alla stampa a valori memorizzati su memorie secondarie (per esempio tutti i tipi di memorie di massa, dal DVD all'hard disk al pendrive).

Rappresentazioni numeriche e convenzioni

Dato un valore numerico intero senza segno, esso può essere rappresentato attraverso i vari sistemi numerici posizionali.
Bisogna però ricordare che il segno 123 non è un numero, ma solo la sua rappresentazione nel sistema numerico decimale.
In informatica i numeri sono spesso rappresentati nei sistemi numerici **binario** e **esadecimale**, ovvero il segno 123 rappresenta la stessa cosa del segno (1111011)b e del segno (7B)h.
A volte 123 viene indicato con (123)d, per mostrare che la rappresentazione è decimale, oppure con 1111011b in notazione binaria oppure ancora con 7bh in notazione esadecimale.
Ricordare sempre che i numeri espressi in forma esadecimale sono molto usati perché due cifre esadecimali significano sempre un singolo byte.
Altre convenzioni esadecimali equivalenti usate in ambito informatico sono:

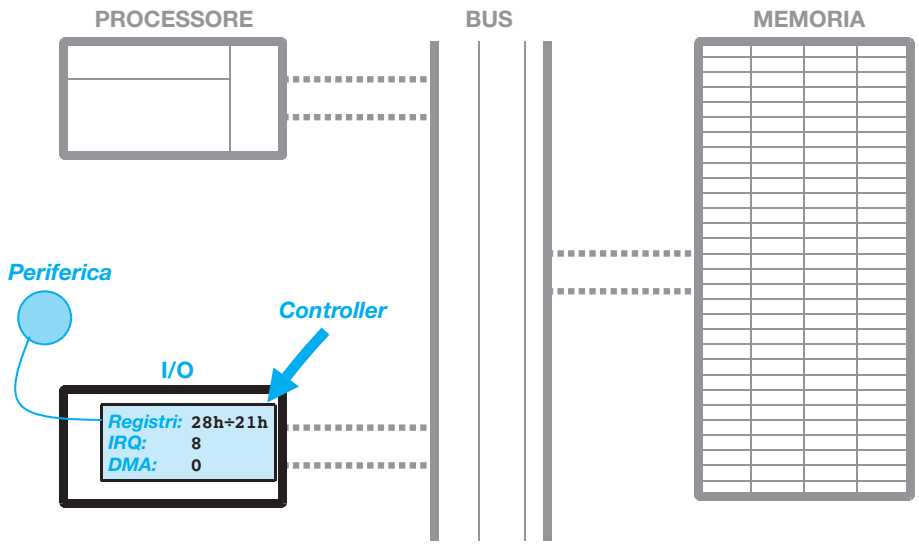
(7B)H	0x7B	
&H7B	16#7B#	$7B
$7B_{16}$	7BHex	$(7B)_{16}$

Concettualmente la sezione di I/O è ancora rappresentabile come un contenitore di celle del tutto analogo alla memoria, anche se dotato di uno spazio di indirizzamento (**spazio degli indirizzi di I/O**) molto più ridotto. Ogni dispositivo periferico, di input o di output, possiede un proprio range di indirizzi di I/O riservato, detti anche **registri di I/O** o **porte di I/O** all'interno dello spazio di indirizzamento di I/O.

Qualche volta alcuni dispositivi, che magari necessitano di grandi spazi di I/O, usano indirizzi di memoria invece di indirizzi di I/O. In questo caso si parla di **dispositivi mappati in memoria**.

La sezione di I/O dispone tuttavia di almeno un'altra modalità fondamentale per consentire la gestione delle attività di I/O con le attività generali del bus, ovvero linee di sincronizzazione dedicate denominate **linee di interruzione** (*Interrupt*).

Per gestire i segnali di interruzione, sul CBus sono implementati normalmente due segnali tipici (**INTR** e **INTA**) che segnalano, rispettivamente, la richiesta di un'interruzione e il suo completamento. Con un segnale di interruzione, il dispositivo periferico chiede al processore di sospendere temporaneamente la sua esecuzione per eseguire una parte di programma che lo riguarda, sotto forma di procedura associata a quella interruzione (**ISR**, *Interrupt Service Routine*). Questo è molto importante per quell'I/O denominato **asincrono**, ovvero che può intervenire senza preavviso e in ogni momento (per esempio il movimento del mouse).



In realtà i calcolatori prevedono anche speciali modalità di trasferimento di I/O che evitano di occupare il processore, veicolando i valori di I/O direttamente verso (e dalla) memoria, mediante tecniche denominate **DMA** (*Direct Memory Access*) o di bus Mastering. In questi casi ampie quantità di informazioni vengono spostate sul bus senza impegnare il processore.

La circuiteria di una periferica, dedicata ad affacciarsi sul bus di sistema, a rendere disponibili i propri indirizzi di I/O, a sincronizzarsi con i trasferimenti tramite il CBus e a condividere le proprie linee dedicate di interruzione e/o DMA, è detta **scheda controller** della periferica.

ESEMPIO

Trasferimenti sul bus: I/O

Dato un modello di bus con 8 linee di DBus, 8 linee di ABus e 3 linee di CBus, si indichino gli stati del bus in conseguenza alla scrittura all'indirizzo 73 dell'I/O della somma dei valori che si trovano agli indirizzi 22 e 23 della memoria.

Si suppone che agli indirizzi specificati in memoria siano presenti rispettivamente i valori 10 e 20 e che il CBus abbia le seguenti linee: I/O-$\overline{Mem}$, R/$\overline{W}$, $\overline{Wait}$.

Prima di tutto si trasformano i valori decimali in binario, riempiendo i risultati al «byte» (cioè aggiungendo tanti zeri a sinistra fino a ottenere sequenze di 8 bit):

(73)d=(1001001)b=(**01001001**)b (indirizzo I/O destinazione)
(22)d=(10110)b=(**00010110**)b (primo indirizzo di memoria)
(23)d=(10111)b=(**00010111**)b (secondo indirizzo di memoria)
(10)d=(1010)b=(**00001010**)b (primo valore in memoria)
(20)d=(10100)b=(**00010100**)b (secondo valore in memoria)

Ricordando che le linee sul CBus hanno i seguenti significati:

I/O-$\overline{Mem}$ **0** processore-memoria; **1** processore-I/O;
R/$\overline{W}$ **0** scrittura; **1** lettura;
$\overline{Wait}$ **0** attesa; **1** operazione completata

ecco i singoli passaggi.

DBus	*ABus*	*CBus*		
76543210	*76543210*	*I/O-$\overline{Mem}$*	*R/$\overline{W}$*	*$\overline{Wait}$*

Trasferimento del primo valore dalla memoria alla CPU (lettura):

.....	**00010110**	**0**	**1**	**0**
00001010	**00010110**	**0**	**1**	**1**

Trasferimento del secondo valore dalla memoria alla CPU (lettura):

00001010	**00010111**	**0**	**1**	**0**
00010100	**00010111**	**0**	**1**	**1**

Trasferimento del risultato della somma (30)d = (00011110)b all'I/O (scrittura):

00011110	**01001001**	**1**	**0**	**0**
00011110	**01001001**	**1**	**0**	**1**

Molti dispositivi di I/O sono interni al calcolatore, cioè hanno la scheda controller integrata nel Chipset della scheda madre (per esempio tastiera, scheda di rete e scheda video). Anche speciali elementi di I/O dedicati al controllo dei trasferimenti di I/O sono integrati nel Chipset, come ad esempio il dispositivo di controllo delle interruzioni (**Interrupt Controller**) e di gestione del DMA (**DMA Controller**).

Con l'introduzione della tecnologia PCI per il bus di sistema (che gestisce anche i collegamenti con l'I/O tramite il SouthBridge) viene a risolversi l'annosa questione dell'esatta distribuzione dei registri di I/O e delle linee di sicronizzazione dell'I/O (per esempio, le linee di interruzione e canali DMA). Infatti, essendo tutti indirizzi, linee e canali privati e dedicati a singole periferiche, essi devono essere ben separati e distribuiti tra le diverse schede controller installate sul sistema.

Prima del bus PCI, infatti, ogni scheda controller doveva essere configurata «a mano» (tramite ponticelli modificabili sulla scheda controller) per specificarne gli indirizzi di I/O, il numero di interruzione e il canale DMA in modo coerente con eventuali altre schede controller presenti sul sistema, per evitare conflitti di attribuzione.

Attualmente il bus PCI consente di utilizzare una tecnica denominata **Plug&Play** per la quale BIOS, sistema operativo e firmware residente sulla scheda controller stabiliscono, all'avvio del calcolatore (o nel momento dell'installazione «a caldo» di un dispositivo), indirizzi, canali e linee completamente separati per ogni scheda controller presente sul sistema, eliminando la possibilità di conflitti ed evitando operazioni di configurazione manuale da parte dell'utente.

Altri elementi di I/O o periferiche, opzionali o comunque gestibili come componenti installati dall'utente, sono detti esterni e si connettono al bus

Codici Ascii

Siccome in una memoria sono memorizzabili solo valori numerici, il modo classico per specificare caratteri alfabetici, ma anche simboli o codici speciali, è usare una tabella di corrispondenza tra numeri e simboli.

La tabella utilizzata più frequentemente è la cosiddetta **tabella dei codici Ascii** (*American Standard Code for Information Interchange*) nella sua versione originale a 7 bit, che associa ai valori numerici contenuti in 7 bit (128 valori), altrettanti simboli.

I codici superiori a 127 (su 8 bit) si dicono caratteri **Ascii estesi**.

È importante ricordare che i numeri da 0 a 31 (0h-1fh) associano **caratteri speciali**, da 48 a 57 (30h-39h) i **caratteri numerici decimali**, da 65 a 90 (41h-5ah) i **caratteri maiuscoli dell'alfabeto inglese**, da 97 a 122 (61h-7ah) i **caratteri minuscoli dell'alfabeto inglese**.

tramite connessioni standard che vengono rese disponibili attraverso appositi connettori.

Tra questi ricordiamo le connessioni dirette al bus di sistema tramite i cosiddetti **Slot** (per esempio slot PCI), sui quali l'utente può connettere direttamente (ma a calcolatore spento) una scheda controller di I/O (per esempio una seconda scheda di rete o schede di I/O dedicate).

Ancora per poco disponibili sono le connessioni IDE-EIDE per hard disk (conosciute anche con il nome ATA-ATA2) e/o dispositivi DVD (che utilizzano lo standard ATAPI su IDE-EIDE).

Le connessioni di questo tipo constano di due connettori IDE-EIDE su piastra madre (primario e secondario) a cui collegare due dispositivi, un Master e uno Slave selezionabili tramite ponticelli (jumper) sulle periferiche a scelta tra hard drive e dispositivi DVD.

Attualmente questi dispositivi possono avvalersi di una più recente tecnologia di connessione, di tipo **SATA** (Serial ATA), per hard disk e dispositivi ottici di nuova generazione, collegabili anche a caldo.

Tra gli standard di connessione più diffusi ricordiamo lo standard **USB** (*Universal Serial Bus*), il **Firewire** (o IEEE 1394) e l'**Ethernet 802.x**.

In via di dismissione invece altri standard di connessione storici, tipo porte seriali Rs232, parallele Centronics e seriali PS/2, progressivamente realizzate attraverso standard più moderni (tipicamente convertitori USB).

USB e Firewire sono modi di connessione dell'I/O molto utilizzati, sia per l'ampia **larghezza di banda** (480 Mbit/s per USB 2.0, 400 Mbit/s per Firewire), sia per la possibilità di installare «a caldo» i dispositivi che se ne servono. Inoltre forniscono anche una certa quantità di corrente sullo stesso cavo dati, così da rendere molti dispositivi del tutto autonomi anche dall'alimentazione. In entrambi i casi USB e Firewire consentono connessioni multiple in catena, tramite Hub nel caso di USB, direttamente in modo passante per Firewire.

4 Processore

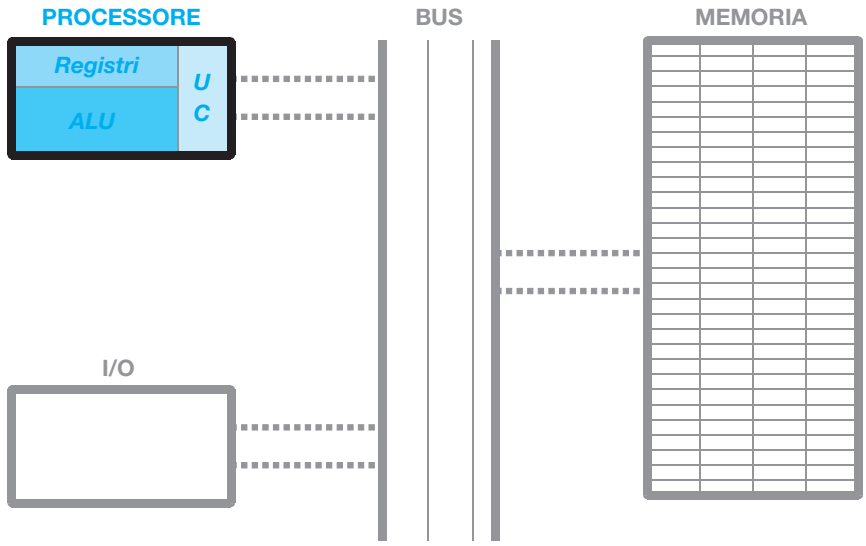
Un processore è un singolo circuito integrato in grado di effettuare operazioni decisionali, di calcolo o di elaborazione dell'informazione; spesso il processore è indicato con la sigla **CPU** (*Central Processor Unit*).

Il processore può essere visto come suddiviso in tre unità funzionali, l'**UC** (*unità di controllo*), l'area dei **registri** e l'**ALU** (*unità aritmetico-logica*).

L'UC si affaccia sul bus, lo arbitra impostando i valori sulle linee ABus, DDus e CBus, legge il DBus e il CBus, legge dalla memoria (e dall'I/O) i dati o li aggiorna in memoria (o nell'I/O) dopo aver compiuto operazioni.

I registri contengono i dati letti dall'UC sul bus per predisporli all'esecuzione delle istruzioni che avverranno nell'ALU; oppure contengono i risultati delle operazioni compiute dall'ALU in attesa di essere passati all'UC e quindi sul bus.

L'ALU è l'unità di esecuzione effettiva del processore, all'interno della quale si trovano **microprogrammi** cablati direttamente in hardware, scritti nel cosiddetto microcodice con relative microistruzioni.



Ogni processore viene progettato con un set di istruzioni specifico denominato **ISA** (**Instruction Set Architecture** o **Instruction Set**), in corrispondenza di ognuna delle quali è implementato un preciso microprogramma in ALU.

Ogni istruzione dell'ISA è contraddistinta da un numero specifico, denominato **Op.Code** (*Operation Code* o *op.cod.*) e ogni istruzione dotata di Op. Code necessita di un numero preciso e definito di parametri (**operandi**) che, assieme all'Op.Code, determinano la **lunghezza dell'istruzione** (in byte).

Ad ogni Op.Code si associa anche una breve descrizione in lingua naturale che ne ricorda la funzione, detta **codice mnemonico**.

Un registro speciale del processore, detto **PC** (*Program Counter*), si incrementa automaticamente della lunghezza dell'istruzione appena eseguita.

Codici Ascii speciali e numerici

Tra i caratteri speciali vanno ricordati:

10 (0ah), detto **Line Feed** (LF).
13 (0dh), detto **Carriage Return** (CR)
32 (20h), detto **Space** (spazio).

Ricordare sempre che il simbolo 0 (zero) ha codice Ascii 48 (30h) e non è il numero zero (a cui è assegnato, nella tabella Ascii, un simbolo grafico).
È utile ricordare anche che per ottenere i simboli Ascii dei numeri decimali basta aggiungere 48 (30h) al numero corrispondente e viceversa.
Esempio: dato il numero 7, il suo codice Ascii corrispondente vale 7 + 48, cioè 55 (37h).
Infine si ricorda che in molti contesti è possibile indicare il codice Ascii di un carattere alfabetico con la notazione tra singolo apice, cosicché 'A' è il numero 65.

ESEMPIO

ISA e programma

Dato un modello di calcolatore con ABus e DBus da 8 linee e CBus da 3 linee; un microprocessore con due registri da 8 bit (registro A e registro B), e la seguente ISA:

Op. cod.	Mnemonico	Operando 1	Operando 2	Descrizione
0	STORE	indirizzo	valore	Pone **valore** a **indirizzo**
10	LOADA	indirizzo		Carica il **valore** che si trova a **indirizzo** nel **registro A**
20	OUT			Pone sullo schermo il simbolo del codice Ascii che si trova nel **registro A**
50	END			Termina il programma

stampare sullo schermo la stringa HAL.

Se si suppone che inizialmente il Program Counter valga 16, il programma in memoria diventa così:

Indirizzi	Memoria							
0								
8								
16	0	0	'H'	2	1	65	0	2
24	'L'	10	0	20	10	1	20	10
32	2	20	50					

Si nota che il programma inizia all'indirizzo 16 con istruzione op.cod. 0 (STORE) e termina all'indirizzo 34 con istruzione op.cod. 50 (END).

L'istruzione op.cod. 0 (STORE) ha due operandi e ha lunghezza 3 byte, mentre l'istruzione op.cod. 50 (END) non ha operandi per cui la sua lunghezza vale 1 byte.

Il programma è ampio 34 – 16 + 1 = 19 byte.

Lo stesso programma può essere scritto, in codice mnemonico: ▶

ESEMPIO

Indirizzo	*Istruzione*	*Commento*
(16)	STORE 0,'H'	' Salva in memoria la lettera H a indirizzo 0
(19)	STORE 1,'A'	' Salva in memoria la lettera A a indirizzo 1
(22)	STORE 2,'L'	' Salva in memoria la lettera L a indirizzo 2
(25)	LOADA 0	' Carica nel registro A la prima lettera…
(27)	OUT	' ... e la stampa
(28)	LOADA 1	
(30)	OUT	
(31)	LOADA 2	
(33)	OUT	
(34)	END	' Termine del programma

Il processore tipico quindi agisce secondo una rigida sequenza di passi che si ripetono fino all'arresto della macchina:

- **Fetch**: l'UC pone sull'ABus il valore del registro Program Counter, imposta lettura da memoria e carica l'Op.Code ivi memorizzato
- **Decode**: l'UC, a partire dall'Op.Code appena letto, determina la lunghezza dell'istruzione, cioè la quantità di parametri di cui necessita, quindi attiva una fase intermedia di caricamento degli operandi (Operand Fetch) che si trovano necessariamente e in modo ordinato agli indirizzi adiacenti a quello dell'Op.Code. Gli operandi caricati andranno a depositarsi nei registri.
- **Execute**: viene avviato il microprogramma relativo all'Op.Code attuale, che usa i propri parametri correttamente memorizzati nei registri. La frequenza in base alla quale vengono eseguiti i microprogrammi è regolata dal **clock di CPU** (frequenza del microprocessore).
- **Store**: al termine della fase di Execute gli eventuali risultati, posti nei registri, vengono scritti sul bus dall'UC, o verso la memoria, o verso l'I/O.

Il ciclo del processore qui descritto termina, in effetti, consultando il segnale **INTR** per capire se il controller di una periferica ha richiesto una interruzione, ovvero la sospensione temporanea dell'esecuzione per servire il codice associato alla interruzione pendente.

Ogni singola istruzione dell'ISA di un processore è contraddistinta da un proprio Op.Code, una determinata lunghezza (in base al numero di operandi che utilizza) e un preciso numero di **cicli di bus** per il suo completamento (compresi tra il fetch, il decode e lo store). Il tempo di effettiva esecuzione del microprogramma influisce relativamente sulla durata dell'istruzione, essendo il clock di CPU di almeno un ordine di grandezza superiore al clock di bus.

Questa considerazione ricorda di nuovo il già citato collo di bottiglia dell'architettura CISC, causato da una fase di decode molto lunga e onerosa a causa della quantità di istruzioni dell'ISA e della loro complessità e varietà in termini di numero di operandi.

Ovviamente lo schema è semplificato. Per esempio un microprocessore reale contiene, oltre all'ALU, una **FPU** (*Float Point Unit*) per i calcoli in virgola mobile e varie unità di calcolo per istruzioni complesse (per esempio per calcoli vettoriali), ma il modello di riferimento rimane ancora concettualmente significativo.

ESEMPIO

ISA completa

Dato un modello di calcolatore con ABus e DBus da 8 linee e CBus da 3 linee; un microprocessore con due registri da 8 bit (registro A e registro B), e la seguente ISA:

Op. cod.	Mnemonico	Operando 1	Operando 2	Descrizione
0	STORE	indirizzo	valore	Pone **valore** a **indirizzo**
10	LOADA	indirizzo		Carica il **valore** che si trova a **indirizzo** nel **registro A**
11	LOADB	indirizzo		Carica il **valore** che si trova a **indirizzo** nel **registro B**
20	OUT			Pone sullo schermo il simbolo del codice Ascii che si trova nel **registro A**
21	INPUT	indirizzo		Il codice Ascii del tasto digitato viene posto a **indirizzo**
30	JMP	indirizzo		Il Program Counter passa a **indirizzo**
31	JE	indirizzo		Salta a **indirizzo** solo se **registro A** è uguale a **registro B**
50	END			Termina il programma

stampare sullo schermo la stringa HAL solo se l'utente digita il tasto H.

Se si suppone che inizialmente il Program Counter valga 16, il programma in memoria diventa così:

Indirizzi	Memoria							
0								
8								
16	0	0	'H'	0	1	65	0	2
24	'L'	21	3	10	0	11	3	31
32	40	30	49					
40	10	0	20	10	1	20	10	2
48	20	50						

Si nota che il programma inizia all'indirizzo 16 con istruzione Op.Cod. 0 (STORE) e termina all'indirizzo 49 con istruzione Op.Cod. 50 (END).

Il programma è ampio 49 – 16 + 1 = 34 byte.

Lo stesso programma può essere scritto, in codice mnemonico:

```
Indirizzo  Istruzione        Commento
(16)  STORE 0,'H'
(19)  STORE 1,'A'
(22)  STORE 2,'L'
(25)  INPUT 3          ' memorizza all'indirizzo 3 il tasto digitato
(27)  LOADA 0          ' carica la 'H' nel registro A
(29)  LOADA 3          ' carica il codice Ascii del tasto digitato nel registro B
(31)  JE 40            ' salta alla stampa solo se registro A è uguale a registro B
(33)  JMP 49           ' se no, termina saltando all'ultima istruzione
(40)  LOADA 0
(42)  OUT
(43)  LOADA 1
(45)  OUT
(46)  LOADA 2
(48)  OUT
(49)  END
```

5 CISC, RISC, CRISC

Il modello di processore appena descritto contiene microprogrammi e microcodice, cioè si dice che è un microprocessore «a interprete». In altre parole, ogni istruzione dell'ISA deve essere decodificata (decode), quindi dotata degli operandi che richiede (operand fetch) e, infine, avviata alla fase di esecuzione (execute) che richiede l'avvio di uno specifico microprogramma. Tutto questo significa che ogni singola istruzione di un'ISA del genere ha uno svolgimento (o data path) a più cicli.

Il **data path** è il percorso dei dati all'interno del processore, e i suoi cicli sono scanditi dal clock della CPU.

Questi tipi di ISA sono denominate **CISC** (*Complex Instruction Set Code*). Le architetture CISC possiedono un set di istruzioni numeroso; le ampiezze delle istruzioni sono molto variabili, con corrispondente fase di decode complessa, da svolgersi con un data path a più passi. Si tratta di architetture che facilitano la portabilità del software, dato che l'insieme dei microprogrammi (o interprete del processore) può essere trasportato su processori più recenti, e quindi sono processori adatti per essere programmati anche in Assembly.

Al contrario, un'architettura **RISC** (*Reducted Instruction Set Code*), possiede un data path a singolo passo. Il set di istruzioni di una architettura RISC è limitato, contiene istruzioni di lunghezza costante (con un numero di operandi fisso), con fase di decode breve e senza microprogrammi da eseguire nel processore: ogni istruzione è eseguita direttamente in hardware con pochi cicli di clock. In questo modo una elaborazione RISC appare nettamente più veloce (almeno di un ordine 10). In sostanza un'istruzione CISC – con molti passi nel data path – equivale a numerose istruzioni RISC con data path singolo. Per questo i programmi per ISA RISC sono molto più lunghi di un analogo programma per ISA CISC. Tutto ciò implica maggiori difficoltà per la portabilità del software e maggiore complessità dei compilatori. I calcolatori RISC sono difficilmente programmabili in Assembly, causa la mancanza di istruzioni ISA di alto livello.

Nessuna delle due architetture è ideale.

Piuttosto la tendenza attuale è l'implementazione di processori su base CISC, come descrive il modello di von Neumann, dotati di sottosistemi interni basati su RISC, soprattutto dedicati alla computazione delle istruzioni semplici (e più comuni) dell'ISA adottata. In questo caso si parla di architetture **CRISC** (*Complex Reducted Instruction Set Code*).

Il modello SISD/CISC dell'architettura di von Neumann si scontra con un paio di problemi che ne limitano, strutturalmente, la performance.

Una singola istruzione su singolo dato eseguita nell'unità di tempo è un limite da tempo inaccettabile. Molte delle soluzioni per incrementare le prestazioni di un calcolatore tendono a parallelizzare l'esecuzione.

Questo è il limite di un'architettura SISD.

Così come il cronico ritardo con cui si accede alla memoria (clock del bus, sull'ordine dei MHz), rispetto alla velocità di esecuzione del processore (clock della CPU, sull'ordine dei GHz), penalizza enormemente il modello CISC con una fase di decode troppo lenta e complessa. Molte delle soluzioni per incrementare le prestazioni di un calcolatore tendono a fornire istruzioni e operandi dalla memoria alla stessa velocità che impiega il processore per eseguirle.

Questo è il limite di un'architettura CISC.

6 Cache

Un modo ingegnoso per diminuire gli accessi al bus e alla memoria, e quindi di contrastare i limiti di un'architettura CISC, è quello di dotare il calcolatore di una memoria «tampone» (**Cache Memory**) tra il processore e il bus.

Man mano che il processore legge dalla memoria, ad un determinato indirizzo, molte locazioni di memoria con indirizzi prossimi a quello, vengono spostati nella memoria cache (nello stesso tempo di bus). Questi gruppi di valori che vengono portati nella cache sono detti linee di cache.

Il motivo è che, secondo il principio della **località spazio-temporale**, un programma utilizza, entro un breve intervallo di tempo, solo una piccola parte del suo spazio degli indirizzi, una parte composta da indirizzi numericamente prossimi tra di loro. Cosicché la memoria cache (realizzata in SRAM), che è molto più veloce di una memoria DRAM, può fornire, nell'immediato futuro (per esempio, la prossima istruzione da eseguire), i valori desiderati senza dover accedere al bus.

Quindi, ad ogni operazione di lettura (del processore dalla memoria), l'informazione viene cercata prima di tutto nella cache; se è presente (**hit**), non è necessario accedere al bus; se non è presente (**miss**) si accede alla memoria e, oltre all'informazione richiesta, si carica una nuova linea in cache, sovrascrivendo la linea di cache meno usata di recente.

Nei calcolatori sono montate almeno tre tipi di memorie cache (livelli): Livello 1, all'interno del processore; Livello 2, collegato al processore; Livello 3 sulla piastra madre.

7 Prefetch, pipeline, superscalarità

Un modo per aumentare il parallelismo d'esecuzione, e quindi di superare i limiti di un'architettura SISD, fu quello di caricare nel processore più istruzioni oltre a quella richiesta. Fin dagli esordi, per esempio, i processori erano dotati di una **coda di Prefetch**, ovvero di un buffer interno in cui il processore memorizzava i successivi 6 o 8 byte consecutivi a quello appena letto dalla memoria. In questo modo, con un solo accesso alla memoria, si aveva a disposizione una serie di valori che potevano essere usati successivamente (come istruzioni od operandi) senza dover accedere di nuovo al bus.

Ben presto, alla coda di Prefetch, fu affiancato un sistema a **Pipeline** che ha lo scopo di sfruttare il concetto di catena di montaggio: invece di eseguire un'istruzione completamente e solo al termine la sua successiva, si può avviare l'istruzione subito dopo che la precedente è stata inserita nel data path. Per esempio, basta che la prima istruzione si trovi in fase di decode, e la successiva può essere posta in stato di fetch. Così come in una catena di montaggio, un nuovo pezzo può essere lavorato anche se il precedente non è ancora stato completato: basta che le fasi (dette anche **stadi della pipeline**) non si sovrappongano. Così, una Pipeline a 5 stadi trasporta cinque istruzioni in catena di montaggio.

La Pipeline sopperisce così alle attese di CPU veloci nei confronti di memorie lente (collo di bottiglia di von Neumann).

Una volta dotato di Pipeline, in un processore si è notato che lo stadio di esecuzione è il più lento: lo stadio precedente fornisce più valori di quanto lo stadio di esecuzione, implementato nell'ALU, può elaborare. Ecco allora che sui processori sono montate più ALU in modo da servire velocemente

ogni istruzione che arriva allo stadio di execute. In questo caso il processore è detto **Superscalare**.

In questo modo è possibile dotare i processori anche di due o quattro pipeline differenti.

Tutte queste metodologie, però, vengono vanificate da due ovvie situazioni:

a Istruzioni di salto;
b Dipendenza dei dati tra le istruzioni.

Nel primo caso la pipeline viene del tutto persa se l'istruzione corrente è un salto (**branch**) distante dall'istruzione successiva che è attualmente in pipeline. In questo caso viene persa anche la cache.

Nel secondo caso una pipeline deve essere interrotta se l'istruzione successiva necessita, come operando, del risultato finale dell'istruzione precedente.

Esempio:

- Istruzione1: A = B + C;
- Istruzione2: D = A + 1.

La pipeline che sta servendo l'Istruzione2 deve interrompersi allo stadio di operand fetch, dato che A non è disponibile se non quando l'Istruzione1 non è del tutto terminata.

8 Esecuzione predicativa e speculativa

L'esecuzione predicativa, implementata in moduli del processore denominati **unità di previsione dei salti** (*Dynamic Branch Prediction*), cerca di prevenire la perdita delle pipeline a causa delle istruzioni di salto. In questo caso le unità cercano, con vari algoritmi che usano tabelle simili a memorie cache, di capire se una istruzione di salto avverrà o meno (cosa del tutto non deterministica ma solo statistica o «storica», dato che il fatto viene computato solo durante l'esecuzione). Per esempio, alcuni criteri considerano sempre «presi» (taken) i salti all'indietro, tipici dei cicli (in un ciclo il salto avviene molto più spesso all'indietro). Il problema di questa tecnica, che in realtà è molto efficiente, si ha quando la previsione è sbagliata: le istruzioni eseguite inutilmente devono essere gettate e lo stato della macchina ripristinato.

L'esecuzione predicativa è anche nota come **esecuzione speculativa**, intendendosi quella elaborazione che computa anche il codice che potrebbe non essere mai utilizzato.

Con l'esecuzione **fuori ordine** (*out of order execution*) si cerca di prevenire lo svuotamento delle pipeline a causa di dipendenze tra le istruzioni. Quando il processore individua una dipendenza in una pipeline, invece «di buttarla» e attendere l'operando mancante, la salta e prosegue con istruzioni "future" che, in teoria, dovrebbero essere eseguite solo dopo quella interrotta. In questo caso la pipeline viene quasi del tutto conservata (un

solo stadio rimane bloccato, fino all'arrivo dell'operando mancante) ma, una volta risolta la dipendenza, saranno già state eseguite altre istruzioni (quelle che avevamo chiamato istruzioni «future»), con conseguente risalita delle prestazioni.

Naturalmente le istruzioni "future" possono essere eseguite solo se non hanno, a loro volta, dipendenze con istruzioni in corso. Non appena le istruzioni con dipendenze terminano, il processore continuerà l'esecuzione in ordine (in order execution).

La condizione più critica per questa tecnica si presenta quando il processore deve essere interrotto a causa di un interrupt; se il processore si trova in fase di fuori ordine, lo stato del sistema potrebbe non essere coerente. In questi casi il processore deve ripristinare lo stato della CPU ritirando «in ordine» tutte le fasi fuori ordine.

Gran parte del lavoro delle unità che gestiscono l'esecuzione fuori ordine è dovuta all'individuazione delle dipendenze nelle istruzioni. La dipendenza classica e più complicata (**RAW**, *Read After Write*) è proprio quella descritta nell'esempio precedente (Istruzione1: A = B + C; Istruzione2: D = A + 1): l'Istruzione2 contiene una dipendenza RAW.

Per poter riordinare il giusto flusso di esecuzione dopo aver saltato e ricalcolato una istruzione con dipendenza, i processori utilizzano una serie di registri d'appoggio (interni e invisibili al programmatore) su cui memorizzare i calcoli temporanei delle istruzioni fuori ordine. All'atto del riordinamento, per evitare di spostare i valori dai registri interni a quelli effettivamente usati nel data path, i processori sono in grado di rinominare i registri interni nei nomi dei registri effettivi, risparmiando il tempo del trasferimento (**register renaming**).

Anche se non esplicitamente, tutte queste innovazioni (pipeline, superscalarità, predicazione, esecuzione fuori ordine) cercano di implementare un modello di esecuzione parallelo molto studiato nei centri di calcolo, e denominato **VLIW** (*Very Long Instruction Word*). In questo modello, oltre alla parallelizzazione dell'esecuzione ottenuta in hardware, si presuppone che lo stesso codice esecutivo generato dai compilatori sia «pre-cucinato» per essere parallelizzato ottimamente dalle CPU.

ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1. Oltre a John von Neumann, indicare almeno altri due studiosi che hanno dato slancio all'evoluzione dei calcolatori, in quale periodo e con quale contributo.
2. Disegnare su un foglio lo schema dell'architettura di von Neumann, indicando i nomi dei componenti e la loro funzione.
3. Citare le due tecnologie per la memoria RAM e illustrare il concetto di collo di bottiglia di von Neumann.
4. Illustrare i concetti di spazio di indirizzamento e di memoria volatile.
5. Mostrare le tre sezioni costituenti un bus e descriverne le funzioni.
6. Spiegare in cosa consiste un trasferimento sul bus e scriverne un esempio.
7. Spiegare la nozione di porta di I/O e saperla collocare tra controller e periferica.
8. Elencare i principali modi di connessione per l'I/O.
9. Illustrare il concetto di interruzione anche con un esempio tratto dalla vita quotidiana.
10. Disegnare su un foglio lo schema di una CPU riportando i nomi delle unità funzionali.
11. Saper commentare le nozioni di ISA, Op.Code, Program Counter e lunghezza dell'istruzione.
12. Elencare i tipi di ISA in base alla natura del set delle istruzioni.
13. Spiegare perché una memoria cache è efficiente e perché una modalità a pipeline è efficiente.

Requisiti avanzati

1. Perché il modello di John von Neumann ha dato la svolta alla tecnologia degli elaboratori e quale ne è stata l'evoluzione più recente?
2. Spiegare come è possibile che un calcolatore si avvii regolarmente se la memoria RAM è volatile.
3. Illustrare i termini Motherboard, Chipset, Northbridge, Southbridge collocando in ognuno le relative funzionalità.
4. Spiegare perché l'uso di memoria cache contrasta il collo di bottiglia di von Neumann.
5. Elencare le principali tecnologie commerciali di bus.
6. Elencare i principali elementi di I/O che vengono impostati dal plug&play.
7. Commentare il concetto di ISR (*Interrupt Service Routine*).
8. Elencare e commentare le fasi di lavoro di una CPU.
9. Definire e spiegare il termine datapath e le sue implicazioni sull'ISA di una CPU.
10. Mettere in relazione il concetto di pipeline con la locuzione Dynamic Branch Prediction.
11. Definire l'acronimo RAW e illustrarne il concetto con un esempio.
12. Elencare gli elementi tecnologici che spostano il tradizionale modello SISD verso i modelli SIMD/MIMD dei moderni calcolatori.
13. Elencare tutti i codici Ascii più importanti, singolarmente e per categorie.

ESERCIZI PER LA VERIFICA SCRITTA

I prerequisiti richiesti per affrontare questi esercizi sono la conoscenza dei sistemi di numerazione in base 2 e in base 16 (binario ed esadecimale), la conversione decimale-binario e viceversa, la conversione decimale-esadecimale e viceversa, anche con applicativi (esempio Calc).

È necessaria la consultazione di una tabella di codici Ascii.

Trasferimenti

1 Si voglia indicare lo stato del bus al completamento del trasferimento del valore 23 alla cella di indirizzo 17 della memoria, su un bus con 8 linee di DBus, 8 linee di ABus e 3 linee di controllo I/O-Mem, R/W, Wait. Indicare la sequenza dei valori binari delle linee del bus.

2 Si voglia indicare lo stato del bus al completamento del trasferimento del valore 15 alla cella di indirizzo 19 della memoria, su un bus con 8 linee di DBus, 8 linee di ABus e 3 linee di controllo I/O-Mem, R/W, Wait. Indicare la sequenza dei valori binari delle linee del bus.

3 Si voglia indicare lo stato del bus al completamento del trasferimento del valore 29 alla cella di indirizzo 14 della memoria, su un bus con 8 linee di DBus, 8 linee di ABus e 3 linee di controllo I/O-Mem, R/W, Wait. Indicare la sequenza dei valori binari delle linee del bus.

4 Scrivere la sequenza dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti la lettura del dato presente sull'I/O alla porta di indirizzo 234 se sulla porta c'è il valore 33.

5 Scrivere la sequenza dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti la lettura del dato presente in memoria nella cella di indirizzo 235, supponendo che a tale indirizzo ci sia il codice Ascii dell'iniziale del proprio cognome.

6 Scrivere la sequenza dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti la lettura del dato presente sull'I/O nel registro 100 se sul registro c'è il valore 78.

7 Scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti una operazione completa di somma dei valori nelle celle con indirizzi 45 e 179 della memoria nelle quali si trovano, rispettivamente, i valori 7 e 5, riponendo il risultato alla cella di memoria di indirizzo 2.

8 Scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti un'operazione completa di sottrazione dei valori presenti nelle celle di indirizzi 45 e 179 della memoria nelle quali si trovano, rispettivamente, i valori 7 e 5, riponendo il risultato alla cella di I/O di indirizzo 7.

9 Scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti un'operazione completa di moltiplicazione dei valori presenti nelle celle di indirizzi 71 e 24 della memoria, nelle quali si trovano, rispettivamente, i valori 8 e 9, riponendo il risultato alla cella di memoria di indirizzo 9.

10 Ipotizzando che sulla porta di I/O di indirizzo 15 ci sia il valore 22 e sulla porta di I/O di indirizzo 24 ci sia il valore 12, scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti la moltiplicazione dei due valori di I/O da mandare in output sulla porta di I/O di indirizzo 2.

11 Scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti la somma dei valori 20 e 3, riponendo il risultato alla cella di memoria 56.

12 Scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti la moltiplicazione dei valori 12 e 7, riponendo il risultato alla cella di memoria 12.

13 Scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che rappresenti la differenza dei valori 21 e 4, riponendo il risultato alla cella di memoria 37.

14 Se in memoria, a partire dall'indirizzo 117 troviamo la stringa HAL, scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che nella cella di memoria di indirizzo 27 risulti la somma dei tre codici Ascii della stringa data.

15 Se in memoria, a partire dall'indirizzo 120 troviamo la stringa IBM, scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale la stringa sia copiata in memoria a partire dall'indirizzo 0.

16 Se in memoria, a partire dall'indirizzo 10 sono memorizzati i primi tre caratteri minuscoli del proprio cognome, copiarli a partire dall'indirizzo di memoria 20, indicando le sequenze dei valori dei bit di un bus con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait.

17 Scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che i primi tre numeri pari siano scritti sulla porta di indirizzo 119 di un controller di scheda I/O.

18 Impostare i primi tre byte di memoria con il codice Ascii dei primi tre caratteri del proprio nome indicando le sequenze dei valori dei bit di un bus con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait.

19 Scrivere le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) in modo tale che i primi tre numeri interi siano scritti sulla porta di indirizzo 129 di un controller di scheda I/O.

20 Se sul registro di indirizzo 61 si trovano i codici Ascii dei caratteri premuti da tastiera, indicare le sequenze dei valori dei bit di un bus (con 8 linee di ABus, 8 di DBus e 3 di CBus I/O-Mem, R/W, Wait) che memorizzano nelle prime tre celle di memoria i valori che risultano avendo premuto i tasti 1, 2 e 3 della tastiera.

ISA e programmi

1 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
20	OUT		
50	END		

scrivere un programma in memoria che stampi sullo schermo l'iniziale del proprio mese di nascita.

2 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
20	OUT		
50	END		

scrivere un programma in memoria che stampi sullo schermo il proprio giorno (numerico) di nascita, su due cifre.

3 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
20	OUT		
50	END		

scrivere un programma in memoria che stampi sullo schermo il proprio mese (numerico) di nascita, su due cifre.

4 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
20	OUT		
21	INPUT	indirizzo	
50	END		

scrivere un programma in memoria che stampi sullo schermo un carattere digitato da tastiera.

5 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
20	OUT		
21	INPUT	indirizzo	
50	END		

scrivere un programma in memoria che stampi sullo schermo il proprio nome dopo che l'utente ha premuto un tasto.

6 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
20	OUT		
21	INPUT	indirizzo	
50	END		

scrivere un programma in memoria che stampi sullo schermo i primi quattro numeri, ognuno dopo la pressione di un tasto.

7 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
20	OUT		
21	INPUT	indirizzo	
50	END		

scrivere un programma in memoria che stampi sullo schermo le prime 4 lettere dell'alfabeto minuscole su quattro righe diverse, ognuna dopo la pressione di un tasto.

8 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
50	END		

scrivere un programma con codici mnemonici che stampi sullo schermo la parola OK se l'utente preme il tasto o.

9 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
50	END		

scrivere un programma con codici mnemonici che stampi sullo schermo la parola OK se l'utente preme il tasto o; la parola NO altrimenti.

10 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
50	END		

scrivere un programma con codici mnemonici che stampi sullo schermo la parola OK se l'utente preme il tasto 1.

11 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
50	END		

scrivere un programma con codici mnemonici che stampi sullo schermo la parola OK se l'utente preme l'iniziale minuscola del proprio cognome.

12 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
50	END		

scrivere un programma in formato mnemonico che, presi due tasti numerici in input, stampi la lettera D se i due tasti sono diversi.

13 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
50	END		

scrivere un programma in formato mnemonico che, presi due tasti numerici in input, stampi la lettera U se i due tasti sono uguali.

14 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo	
50	END		

(ove l'istruzione JG salta a **indirizzo** se il valore nel **registro A** è maggiore del valore nel **registro B**), scrivere un programma in formato mnemonico che, presi due tasti numerici in input, stampi il simbolo > se il primo tasto numerico è maggiore del secondo.

15 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo (cfr. *Esercizio 14*)	
50	END		

scrivere un programma in formato mnemonico che, presi due tasti numerici in input, stampi il simbolo < se il primo tasto numerico è minore del secondo.

16 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo (cfr. *Esercizio 14*)	
50	END		

scrivere un programma in formato mnemonico che, presi due tasti numerici in input, stampi il simbolo >, o il simbolo < o il simbolo = se, rispettivamente, il primo tasto numerico è maggiore del secondo, il secondo maggiore del primo o se uguali.

17 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
12	LOADC	indirizzo	
15	MOV	registroD	registroS
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo (cfr. *Esercizio 14*)	
33	LOOP	indirizzo	
40	INC	registro	
50	END		

considerare che l'istruzione **LOADC** si comporta come le analoghe LOADA e LOADB ma sul registro C; considerare che l'istruzione **MOV** sposta il contenuto della cella che si trova nell'indirizzo contenuto in **registroS**, nel **registroD** (regA=0; regB=1; regC=2); considerare che l'istruzione **LOOP** prima diminuisce di una unità il registro C, poi salta all'indirizzo specificato solo se il registro C è diverso da zero; considerare che l'istruzione **INC** aumenta di una unità il valore contenuto nel registro specificato.
Scrivere un programma in formato mnemonico che stampa sullo schermo il proprio cognome.

18 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
12	LOADC	indirizzo	
15	MOV	registroD	registroS
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo (cfr. *Esercizio 14*)	
33	LOOP	indirizzo	
40	INC	registro	
50	END		

scrivere un programma in formato mnemonico che stampa sullo schermo i primi dieci numeri naturali.

19 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
12	LOADC	indirizzo	
15	MOV	registroD	registroS
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo (cfr. *Esercizio 14*)	
33	LOOP	indirizzo	
40	INC	registro	
50	END		

scrivere un programma in formato mnemonico che stampa sullo schermo l'alfabeto inglese.

20 Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
12	LOADC	indirizzo	
15	MOV	registroD	registroS
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo (cfr. *Esercizio 14*)	
33	LOOP	indirizzo	
40	INC	registro	
50	END		

scrivere un programma in formato mnemonico che stampa sullo schermo l'alfabeto inglese se si preme a, i primi dieci numeri interi se si preme 0.

21 Considerando l'ISA dell'esercizio 20, terminare il programma con la stampa del proprio cognome solo se premuto il tasto corrispondente all'iniziale del cognome.

22 Considerando l'ISA dell'esercizio 20, solo se l'utente preme un carattere numerico, stamparlo a video.

23 Considerando l'ISA dell'esercizio 20, terminare il programma stampando il carattere digitato dall'utente solo se il carattere è numerico.

24 Considerando l'ISA dell'esercizio 20, se premuto un tasto numerico inferiore a 9, stampare sullo schermo il numero successivo.

25 Considerando l'ISA dell'esercizio 20, stampare a schermo la parola NUM se premuto un tasto numerico, la parola ALF se premuto un tasto letterale minuscolo.

26 Considerando l'ISA dell'esercizio 20, stampare a schermo il carattere iniziale del numero se premuto un tasto numerico.

27 Considerando l'ISA dell'esercizio 20, se premuto un tasto alfabetico minuscolo diverso da z, stampare sullo schermo il carattere successivo.

28 Considerando l'ISA dell'esercizio 20, terminare il programma stampando il carattere digitato dall'utente solo se l'utente preme una vocale.

29 Considerando l'ISA dell'esercizio 20, stampare Maiuscolo se premuto un tasto maiuscolo, Minuscolo se premuto un tasto minuscolo.

Architetture Intel

A2

1 x-86

Con **x-86** si intende l'architettura di microprocessori inizialmente sviluppata dall'azienda Intel Corporation negli anni '70, e che è ancora oggi (2013) predominante sul mercato mondiale. Altre importanti aziende producono calcolatori basati su questa tecnologia, i cui diritti sono stati a suo tempo venduti, per esempio AMD.

Il nome x-86 deriva dal primo microprocessore della serie, Intel 8086 (1976), a cui sono seguite numerose versioni più potenti che hanno mantenuto il suffisso nel nome: 8088 (1979), 80186 (1980), 80286 (1982), 80386 (1986), 80486 (1989).

Sono da considerarsi macchine x-86 anche i modelli successivi all'80486, che hanno dovuto rinunciare al nome «numerico» per l'impossibilità di brevettarlo: Pentium (o P5, 1993), Pentium Pro (o P6, 1995), Pentium II (1997), Pentium III (1999), Pentium 4 (o P7, 2000), Pentium M (2003), Pentium D (2005), Core 2 Duo (o P8, 2006), Core i7 (2008), Core i5 (2009), Core i3 (2010).

Tutti i microprocessori di questa famiglia condividono una qualità comune che ne ha decretato il successo, ma anche la notevole complessità progettuale: sono tutti **retrocompatibili** e, in particolare, tutti ancora in grado di eseguire le istruzioni originali dell'ISA primitiva del progenitore, l'Intel 8086, benché esso fosse una macchina a 16 bit, mentre le ultime citate sono tutte a 32 bit e, in parte, a 64 bit.

Tutti questi modelli sono sostanzialmente macchine CISC/SISD, benché dal Pentium in avanti si sia cercato di aumentare progressivamente il grado di parallelismo dell'esecuzione (prefetch, pipeline, superscalarità, esecuzione predicativa e speculativa, esecuzione fuori ordine).

2 Intel 8086

L'**Intel 8086** (consideriamo l'Intel 8088 come del tutto equivalente, anche se il bus dati è a 8 bit) è un microprocessore a 16 bit (ampiezza dei registri e del DBus) con 20 linee sull'ABus per un totale di 1 MiB di spazio di indirizzamento fisico (2^{20} = 1048756 celle). L'unità di interfaccia con il bus o Unità di Controllo è denominata **BIU** (*Bus Interface Unit*), e passa le istruzioni all'ALU (detta **EU** da *Execution Unit*) attraverso una coda di

prefetch di 6 byte (4 nell'8088), implementando una sorta di rudimentale meccanismo di Pipeline.

L'Intel 8086 possiede 14 registri da 16 bit, di cui quattro **registri per uso generale** (AX, BX, CX, DX), a cui si può accedere anche come se fossero otto registri a 8 bit (AH e AL, BH e BL, CH e CL, DH e DL), due **registri indice** per indirizzare in memoria (SI, DI) e due registri dedicati alla gestione dello stack (BP o **Base Pointer** e SP o **Stack Pointer**).

A questi si aggiungono altri quattro registri detti **di segmento** (CS, DS, ES e SS), dedicati specificatamente all'indirizzamento della memoria. Completano il set di registri l'**Instruction Pointer** (IP) e il registro **PSW** (*Program Status Word*), denominato **flag register**.

Lo spazio degli indirizzi di I/O si avvale di un indirizzamento a 16 bit, per un totale di 64KiB ($2^{16} = 65536 = 64 \times 1024$) registri di Input/Output disponibili. Completa la sezione di I/O un set di 8 linee di interruzione hardware (poi ampliato a 16) e un canale DMA per dispositivi di I/O con ampio traffico. La frequenza originale del clock di CPU valeva 4,77 MHz.

Specifiche soluzioni adottate da questo microprocessore sono l'uso della **segmentazione della memoria** e l'utilizzo in comune delle sedici linee del DBus con altrettante linee dell'ABus (*Multiplexing Bus*), cosicché è necessario un segnale speciale nel CBus (M/IO) che segnali se su quelle linee è presente attualmente un dato o (parte di) un indirizzo. La memorizzazione delle parole di byte in memoria avviene in modalità **little endian** (il byte meno significativo all'indirizzo più basso), modo caratteristico di tutte le macchine Intel.

Le istruzioni dell'ISA 8086, lunghe da 1 a 4 byte, prevedono spesso l'uso implicito di alcuni registri, complicando le operazioni di salvataggio temporaneo e rendendo il processore cronicamente povero di registri disponibili.

La ridotta quantità di registri dell'8086 – così come nelle macchine successive fino al Pentium – ha designato l'Instruction Set dell'8086 come ISA **a due indirizzi**, ovvero come architettura che ha continuamente bisogno di depositare dati temporanei in memoria, invece che su registri supplementari. I molti accessi alla memoria, come abbiamo visto, determinano una forte penalizzazione delle prestazioni.

Ordine dei byte

Esistono vari modi per immagazzinare in memoria dati di dimensione superiore al byte. Tra questi si distinguono il **little endian** e il **big endian**.

Per esempio, il numero 2561, non rappresentabile con un solo byte ma con due byte, e la cui rappresentazione esadecimale su due byte vale (0a01)h, deve essere memorizzato in due celle contingue, per esempio, di indirizzi 12 e 13.

Si possono avere le seguenti possibilità (big endian):

…	*12*	*13*	…
	0A	**01**	

oppure (little endian):

…	*12*	*13*	…
	01	**0A**	

L'ordine di memorizzazione, per quanto riguarda i microproces- →

2.1 Registri

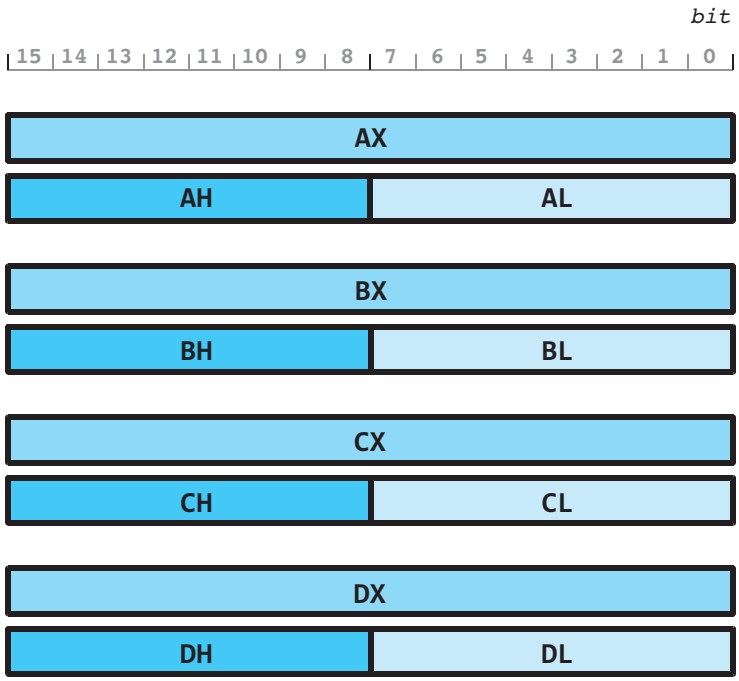
I quattro registri di uso generale, pur utilizzati frequentemente come registri di memorizzazione temporanea (a 16 o a 8 bit), sono dedicati a precisi compiti e sono coinvolti implicitamente in numerose istruzioni dell'Instruction Set x-86.

AX, o registro **accumulatore**, è predisposto per le istruzioni aritmetiche (somme, sottrazioni, moltiplicazioni e divisioni).

BX, o registro **base**, è l'unico dei registri di uso generale che può specificare un indirizzo di memoria (gli altri sono DI, SI e BP).

CX, o registro **contatore**, è utilizzato implicitamente nelle istruzioni di conteggio dei cicli o di operazioni che esigono una numerazione.

DX, o registro di **I/O** è l'unico registro dell'8086 che consente di indirizzare le porte di I/O. Usato anche in moltiplicazioni e divisioni.



registri di uso generale e semiregistri

I due registri indice, pur utilizzabili come registri di memorizzazione temporanea (ma solo a 16 bit), sono usati implicitamente nelle istruzioni dedicate alla manipolazione di array di caratteri (stringhe).

SI, o registro **indice sorgente**, specifica l'indirizzo da cui leggere l'array;

DI, o registro **indice destinazione**, specifica l'indirizzo in cui scrivere l'array.

I due registri dedicati allo stack sono in grado di indirizzare in memoria, anche se non liberamente.

BP, o **Base Pointer**, contiene l'indirizzo di partenza della pila di stack, per poter gestire il passaggio dei parametri delle procedure.

SP, o **Stack Pointer**, contiene sempre l'indirizzo di memoria dell'ultimo elemento sullo stack.

Infine, i due registri IP e Flag, sono registri non modificabili esplicitamente dato che il processore li mantiene aggiornati automaticamente.

IP, o **Instruction Pointer**, contiene la parte meno significativa dell'indirizzo della prossima istruzione da eseguire. Il programmatore non lo modifica mai.

Flag, o registro dei **Flags**, è l'unico registro intepretato a singolo bit, ove ogni bit ha un significato differente e concorre a descrivere lo stato attuale del processore dopo l'esecuzione di un'istruzione.

2.2 Indirizzamento

Ai dieci registri precedenti nell'Intel 8086 bisogna aggiungere 4 registri di segmento (CS, DS, ES, SS, rispettivamente **Code Segment**, **Data Segment**, **Extra Segment** e **Stack Segment**) che hanno sempre il compito di contenere i 16 bit più significativi di un qualsiasi indirizzo di memoria. Infatti l'Abus è a 20 linee, mentre i registri solo a 16 bit, per mantenere la compatibilità con un modello di microprocessore precedente, l'Intel 8080. Cosicché un

→
sori, è del tutto trasparente al programmatore, a meno che il programmatore non intenda accedere alla memoria in modo puntuale, byte per byte.
L'etimologia del nome proviene dal romanzo *I viaggi di Gulliver* (Jonathan Swift, 1726).
Esiste anche il modo **middle endian**, quando l'ordine di valori rappresentabili su più di due byte non è consecutivo rispetto agli indirizzi.
Per dare un'idea del concetto, si pensi alla rappresentazione delle date in diversi contesti geopolitici:

gg/mm/aaaa: data europea, little endian.
aaaa/mm/gg: data giapponese, big endian.
mm/gg/aaaa: data statunitense, middle endian.

Indirizzi segmentati

Dato un indirizzo segmentato nel formato (Seg:Spz), per esempio a100h:023dh, la trasformazione nella sua versione lineare è abbastanza veloce:

```
a100h:023dh=
=a100h*10h+23dh=
=a1000h+23dh=
=a123dh=(660029)d
```

Si deve tenere presente che moltiplicare per 10h un numero esadecimale significa solo aggiungere uno zero (esadecimale) a destra, cosicché a100h * 10h = a1000h.

Si noti che un qualsiasi indirizzo lineare può essere rappresentato da **più coppie di indirizzi segmentati**, dato che non tutti i 32 bit dell'indirizzo segmentato sono utilizzati.

→

solo registro è insufficiente per rappresentare un indirizzo completo, e ne servono due, accoppiati, per formare un cosiddetto **indirizzo segmentato**.

L'indirizzo segmentato è quindi formato da due parti, e si indica con le notazioni (Seg:Spz) o (Seg:Ofs), ove la parte più significativa (Seg) è detta parte **segmento** dell'indirizzo, mentre la parte meno significativa (Spz o Ofs) è detta parte **spiazzamento** (o *Offset*) dell'indirizzo.

La regola per ottenere un **indirizzo lineare** (o **flat**, cioè la sequenza di 20 bit da collocare sull'ABus) da un indirizzo segmentato (Seg:Spz) è la seguente:

indirizzo lineare = Seg * 10h + Spz

I registri di segmento sono spesso impliciti, ovvero, stabiliti gli accoppiamenti di default con i registri di indirizzamento standard, non è necessario indicarli espressamente nelle istruzioni dell'x-86.

CS, o **Code Segment**, contiene sempre la parte più significativa del Program Counter, ed è associato a IP (CS:IP = Program Counter dell'x-86).

DS, o **Data Segment**, contiene sempre la parte più significativa della memoria dei dati. È associato a BX, SI, DI.

ES, o **Extra Segment**, contiene sempre la parte più significativa di una seconda memoria dati. È associato a BX, SI, DI.

SS, o **Stack Segment**, contiene sempre la parte più significativa della memoria in cui è allocato lo stack. È associato a BP e a SP.

3 IA-32

Con **IA-32** (*Intel Architecture 32 bit*) si indica il modello di architettura e l'Instruction Set dei Processori Intel a partire dal 1986, anno di commercializzazione del microprocessore Intel 80386, il primo microprocessore Intel a 32 bit. Questa architettura si è mantenuta presente sulle macchine Intel fino ad oggi, seppur con numerose varianti introdotte per aumentarne le prestazioni.

La terminologia IA-32 si contrappone all'IA-64, l'architettura a 64 bit che dovrebbe modificare radicalmente la struttura dei calcolatori Intel introducendo in modo nativo il calcolo parallelo e abbandonando il modello CISC/SISD originale dell'IA-32.

In realtà anche l'IA-32, nelle sue evoluzioni, si è sempre più avvicinata a un modello SIMD, già a partire dallo stesso 80386, che fu il primo microprocessore Intel a montare regolarmente una **FPU** (*Floating Point Unit* o coprocessore matematico), un parziale ma effettivo elemento SIMD. Successivamente l'IA-32 si arricchisce di vere e proprie istruzioni SIMD denominate **MMX** (*MultiMedia eXtension*, 1996) e **SSE** (*Streaming SIMD Extensions*, 2000).

È con l'introduzione dell'IA-32, nello specifico con l'80386, che Intel perde il monopolio dell'architettura x-86 (anche se il primo 8086 fu clonato su licenza da NEC già nei primi anni '80) a favore, dapprima, del produttore statunitense AMD (1991).

Naturalmente tutte le macchine con architettura IA-32 sono ancora compatibili con l'architettura x-86 a 16 bit dei modelli precedenti all'Intel 80386.

Le specifiche caratteristiche dell'IA-32 rispetto alla classica x-86 a 16 bit si hanno soprattutto nel nuovo spazio di indirizzamento fisico a 32bit per

un totale di 4GiB di locazioni ($2^{32} = 4\,294\,967\,296 = 4 \times 2^{30}$) e la gestione completa di un modello di memoria virtuale e protetta.

La gestione della **memoria virtuale**, infatti, consente a più programmi contemporaneamente di usare tutto lo spazio di indirizzamento a prescindere da quanta memoria fisica sia effettivamente installata. Ciò significa la possibilità di far funzionare i programmi in *multitasking* (cioè più programmi contemporaneamente), potendo usare uno spazio lineare (e non più segmentato) molto ampio.

La gestione della **memoria protetta**, infine, garantisce che i programmi non interferiscano tra di loro né con il sistema operativo. La protezione della memoria e dell'I/O, quindi, garantiscono anche che i dispositivi non possano essere gestiti con codice errato, tramite livelli di esecuzione differenziati detti **privilegi** (o **ring**).

In pratica l'accesso completo a memoria e I/O è concesso solo ai programmi che hanno privilegi alti (**kernel mode** o permessi a **Ring0**, riservati ai programmi di sistema operativo), mentre i programmi applicativi standard hanno privilegi bassi (**user mode** o permessi a **Ring3**). Tutto ciò ha reso sostanzialmente più stabile l'esecuzione delle applicazioni, superando l'annoso problema dei frequenti blocchi di sistema sulle macchine x-86.

Per questi motivi l'IA-32 può essere avviata o usata dai programmi in tre modalità di microprocessore differenti: **modalità protetta** (la più evoluta, con indirizzamento lineare), **modalità reale** (come un 8086 segmentato), **modalità virtuale 8086** (emulazione 8086 segmentata, ma con protezione).

In altri termini i programmi possono essere eseguiti, anche contemporaneamente, in modalità protetta o virtuale 8086. In questo modo il blocco di un programma non arresta il sistema. Se invece il processore viene avviato e poi utilizzato in modalità reale, esso si comporta solamente ed esattamente come un grosso Intel 8086 (se avviene un blocco sul programma, il sistema si arresta) e non può più cambiare modalità.

In ogni caso tutte le macchine con IA-32, quando si avviano, sono in modalità reale, per poi cambiare modalità a seconda del sistema operativo che verrà caricato.

I registri dell'IA-32 sono sostanzialmente gli stessi della x-86, ma tutti a 32 bit. Solo due registri di segmento sono stati aggiunti, per un totale di 16 registri a 32 bit.

Per analogia, ogni nuovo registro eredita il nome del vecchio, ma con un prefisso E (**Extended**): EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, EIP, EFlags, ECS, EDS, EES, ESS, EFS, EGS.

Il significato e le funzioni dei registri rimangono immutati, considerando che i 16 bit meno significativi dei registri di uso generale assumono lo stesso nome della vecchia architettura (AX, BX, CX, DX) e, a loro volta, si scompongono nei soliti 8 registri a 8 bit (AH, AL, BH, BL, CH, CL, DH, DL).

→
Per esempio i due indirizzi segmentati a100h:023dh e a110h:013dh rappresentano lo stesso indirizzo (lineare) a123dh.
Infatti:

```
a110h:013dh=
=a110h*10h+13dh=
=a1100h+13dh=
=a123dh=(660029)d
```

Inoltre, molti indirizzi segmentati sono **illeciti**, cioè rappresentano valori superiori all'indirizzo lineare massimo (che vale 2^{20}-1 = ffffh = 1048575d).
Per esempio f31ah:e100h, che risulta essere l'indirizzo lineare 1012a0h.
Ma 1012a0h > ffffh, dato che 1012a0h = 1053344d > 1048575d

3.1 Netburst, EM64T

Con **netburst** si indica una tecnologia che mira ad aumentare le prestazioni dei calcolatori agendo sull'aumento progressivo della frequenza di clock del processore (clock di CPU) insistendo sulla continua miniaturizzazione

dei circuiti. Per questo motivo i processori equipaggiati con questa tecnologia posseggono ALU che «girano» a frequenza doppia di quella della CPU. Incrementando la velocità interna alla CPU si possono implementare anche pipeline molto più ampie, aumentando quindi il numero di istruzioni eseguite nell'unità di tempo. Netburst ha infatti introdotto pipeline a 20 stadi, contro i dieci dei microprocessori precedenti. Naturalmente il fallimento della pipeline diventa una evenienza critica, per questa tecnologia. Se la predizione di un salto è errata o se le istruzioni contengono molte dipendenze, la pipeline si svuota e le prestazioni crollano. Per prevenire queste eventualità netburst ha elevato la complessità dell'unità di predizione dei salti (esecuzione predicativa e speculativa) e aggiunto una cache di livello 2 in grado di conservare le microistruzioni, così da non dover accedere alla memoria in caso di ricostruzione della pipeline.

L'**EM64T** (*Extended Memory 64 Technology*) è una tecnologia che viene utilizzata nei processori IA-32 per poter sfruttare alcuni benefici di calcolo eseguiti a 64 bit, all'interno di processori a 32 bit.

Tramite una speciale modalità (64bit Mode) in cui può operare il processore, modalità impostabile solo da programmi appositamente scritti in tal senso, diviene disponibile un indirizzamento lineare a 64 bit, per uno spazio degli indirizzi a disposizione di tali programmi, di 17 179 869 184 GiB o 16 EiB ($2^{64} = 16 \times 2^{60}$, dove 2^{60} equivale a 1 exbibyte) e si possono utilizzare 8 nuovi registri a 64 bit che diminuiscono gli accessi alla memoria per salvataggi temporanei. Naturalmente il processore dotato di EM64T può operare anche in modo classico IA-32, nella modalità Compatibility Mode.

3.2 MultiCore

Secondo la strada indicata dalla tecnologia netburst, si sarebbero dovute ottenere frequenze al limite dei 10GHz. Tecnologicamente, però, tali frequenze non sono state raggiunte né sembra lo saranno a breve; per ora le frequenze di clock massime non hanno raggiunto la metà di quella previsione. A frequenze così alte, infatti, si crea una enorme instabilità sui chip e la potenza richiesta pone seri problemi di dissipazione del calore all'interno dei circuiti.

L'orientamento attuale, denominato **MultiCore**, si basa sulla progressiva implementazione di calcolo parallelo, ottenuto inizialmente progettando processori con due (o più) unità complete di calcolo, ognuna basata su CPU che operano a frequenze più basse di una netburst.

Questo tipo di architettura, al pari dei sistemi semplicemente dual core e, più generalmente, di tutti i sistemi biprocessore e multiprocessore, consente di aumentare la potenza di calcolo senza aumentare la frequenza di lavoro, a tutto vantaggio del minore calore dissipato. Se con la tecnologia netburst il grado di parallelismo raggiunto era di tre istruzioni per ciclo di clock (usando grandi pipeline), un processore multicore arriva a 4 istruzioni per ciclo di clock per ogni core implementato, anche se le frequenze di clock interno sono normalmente inferiori a quelle di una CPU netburst.

La tecnologia parallela di un sistema MultiCore è detta **Wide Dynamic**

Execution e usa una pipeline con meno stadi (14 invece dei 31 raggiunti dalla netburst), risultando meno vulnerabile allo svuotamento della pipeline e operando a frequenze minori di clock con abbassamento della potenza richiesta e del calore sviluppati.

In teoria un processore MultiCore assume le caratteristiche di una macchina MIMD, dato che i singoli processori operano in modo indipendente, condividendo solo la cache di livello 2 per evitare di accedere continuamente al bus.

Per sfruttare i Core multipli però è necessario che i programmi siano scritti in modo adeguato, ovvero contengano istruzioni dedicate all'esecuzione parallela, con la granularità del processo o del sottoprocesso detto anche **thread**. Infatti un programma non multithread sarà eseguito da un singolo processore, senza alcun aumento delle prestazioni.

In ogni caso tutti i sistemi operativi multiprogrammati (*multitasking*) godono di maggiore efficienza su un sistema MultiCore dato che i processi sono distribuiti su più unità di calcolo (Core) invece di condividerne uno solo.

Il notevole risparmio in potenza e di calore da dissipare, a causa dell'abbassamento delle frequenze di clock, rende le tecnologie MultiCore estremamente idonee per i calcolatori portatili.

4 IA-64

Con **IA-64** (*Intel Architecture 64 bit*) si intende una famiglia completamente nuova di architettura per microprocessore, contrapposta a IA-32 e radicalmente differente dall'architettura x-86.

Tale progetto è stato affrontato già da molti anni ma ha da sempre incontrato notevoli resistenze per imporsi sul mercato dato che, per forza di cose, esso perde completamente la retrocompatibilità con i programmi scritti per l'x-86, cioè quasi tutti i programmi disponibili sul mercato a tutt'oggi.

Il primo processore IA-64 completamente a 64 bit è datato 2001 ed è conosciuto come **Intel Itanium**. Nel frattempo l'azienda ha continuato a sviluppare la tecnologia IA-32 cercando di spremergli ogni possibilità di evoluzione e prestazione fino agli attuali sistemi MultiCore.

I vantaggi di una tecnologia a 64 bit suppliscono agli svantaggi della tecnologia a 32 bit, prima di tutto alla struttura fondamentalmente CISC dell'IA-32, con la presenza di molte istruzioni di lunghezze differenti e formati diversi che rallentano inevitabilmente la fase di decode.

Inoltre il Set dell'IA-32 è zeppo di istruzioni che fanno riferimento alla memoria (ISA a due indirizzi) mentre sono più efficienti le ISA a tre indirizzi (o **ISA load/store**), che accedono alla memoria solo per caricare gli operandi e memorizzare i risultati.

Un altro limite strutturale è la cronica povertà di registri dell'IA-32 che costringe i compilatori a spostare in memoria molti risultati temporanei, con conseguente abbassamento delle prestazioni.

Infine la complessità delle istruzioni dell'IA-32 comporta la presenza di molte dipendenze WAR (vedi esecuzione fuori ordine e VLIW), compli-

cando notevolmente l'uso delle pipeline e la loro profondità (numerosi stadi). E molti stadi rendono le pipeline vulnerabili ai salti, ottenendo perciò un sistema continuamente in bilico con le prestazioni.

Un'architettura a 64 bit, poi, consente uno spazio di indirizzamento di una grandezza veramente notevole (2^{64} = 18 446 744 073 709 551 616 celle), in linea (e forse molto di più) con l'ampiezza del software richiesto dai sistemi operativi moderni.

La struttura dell'IA-64 è quindi quella di una ISA load/store, con 64 registri a 64 bit. Tutte le istruzioni dell'IA-64 hanno lo stesso formato, in modo da ottenere un'architettura fondamentalmente RISC. Tutti i registri sono dotati di un meccanismo di finestra (*register window*) che riesce a «srotolare» efficacemente i cicli (*loop*), cioè a esplicitare un ciclo di istruzioni eseguite più volte elencandole tutte per esteso, il che in genere porta a un miglioramento delle prestazioni.

L'innovazione computazionale determinante di IA-64 è l'adozione di **EPIC** (*Explicitly Parallel Instruction Computing*), ovvero una tecnica che sposta l'analisi del flusso delle istruzioni di un programma a livello del compilatore in modo tale che in CPU arrivino sequenze di istruzioni (fasci o bundle) già pronte per una elaborazione parallela. Anche l'analisi della predizione dei salti è preconfezionata a livello della compilazione, pertanto una IA-64 ottiene previsioni di salto corrette nel 98% dei casi. Si può dire che EPIC è la implementazione completa e specifica di VLIW già adottata a suo tempo su IA-32.

Da notare che le prime versioni di macchine con IA-64 sono a singolo Core, malgrado la presenza di una tecnologia esplicitamente orientata al calcolo parallelo. Il calcolo parallelo, infatti, si ottiene con la superscalarità e le pipeline ottimizzate da EPIC.

Naturalmente, a causa della sua natura completamente rinnovata e dipendente dalla preanalisi del codice da parte del compilatore, l'architettura IA-64 deve rinunciare a quasi tutto il software di sistema (sistemi operativi) e utente (programmi applicativi) disponibile per IA-32, e questo rappresenta un notevole impedimento alla sua diffusione.

ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1. Elencare i principali microprocessori Intel x-86 a partire dal primo modello, ricordandone almeno altri tre compresi gli ultimi della serie. Indicare quale caratteristica fondamentale li accomuna.
2. Citare le caratteristiche base del microprocessore 8086, compresa l'ampiezza dei registri e lo spazio di indirizzamento.
3. Elencare tutti i 14 registri dell'8086 e ricordare quali interpretano il Program Counter e la PSW.
4. Spiegare perché per l'8086 sono necessari due registri per memorizzare un indirizzo.
5. Illustrare il concetto di segmentazione della memoria per l'8086.
6. Elencare e commentare i registri di uso generale e i registri indice dell'8086, indicandone nomi e funzionalità.
7. Illustrare la gestione della memoria in IA-32 rispetto all'8086.
8. Trasformare un indirizzo segmentato in un indirizzo lineare con un esempio.
9. Scrivere su un foglio il contenuto di un registro di uso generale dell'8086, quindi estrarre i valori dei suoi due semiregistri.
10. Mostrare la relazione tra i registri dell'IA-32 con i registri dell'8086.
11. Mettere a confronto la tecnologia netburst con la tecnologia multicore.
12. Elencare i principali limiti dell'architettura IA-32 rispetto a IA-64.
13. Ricordare il nome commerciale del processore Intel IA-64 e le sue caratteristiche base.

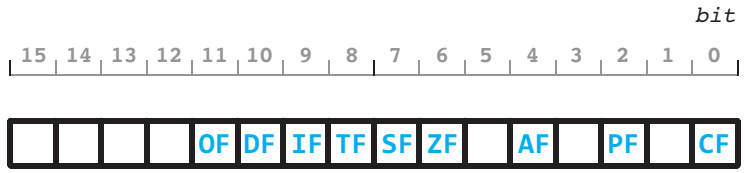
Requisiti avanzati

1. Commentare l'ampiezza dei registri dell'8086 in relazione all'ampiezza del suo spazio di indirizzamento.
2. Spiegare a cosa servono i registri di segmento dell'8086.
3. Mostrare con due esempi le memorizzazioni little e big endian.
4. Spiegare il significato delle locuzioni "ISA a due indirizzi" e "ISA a tre indirizzi".
5. Elencare e spiegare le funzioni dei registri indice dell'8086.
6. Mostrare con un esempio come ricavare due indirizzi segmentati equivalenti da un indirizzo flat.
7. Illustrare il concetto di protezione della memoria in IA-32.
8. Elencare le tre modalità operative dei microprocessori di tipo IA-32, commentandone le caratteristiche.
9. Spiegare la differenza tra codice user mode e kernel mode.
10. Spiegare perché sistemi a microprocessore con frequenze relativamente basse possono risultare più efficienti di sistemi che operano a frequenze maggiori.
11. Mostrare perché lo stile di scrittura del software può incrementare le prestazioni nei sistemi multicore.
12. Spiegare come e perché i compilatori influiscono nel funzionamento dei sistemi IA-64.
13. Spiegare perché IA-64 incontra problemi nell'imporsi commercialmente, benché più evoluta dei sistemi IA-32.

A3 Assembly x-86

Per ogni Instruction Set, prima di cominciare a programmare, è fondamentale conoscere il registro **PSW** (*Program Status Word*) che descrive lo stato della CPU al termine di ogni singola istruzione eseguita.

Nell'x-86 tale registro è chiamato **Flags** e il suo contenuto è interpretato a livello di singolo bit. Naturalmente si tratta di un registro che non è modificabile esplicitamente dal programmatore (non è operando di nessuna istruzione), ma soltanto implicitamente tramite istruzioni che lo modificano nel modo eventualmente voluto dal programmatore.



registro Flags

I 16 bit del registro di Flags non sono tutti utilizzati. Le sigle di quelli significativi sono le seguenti: **CPAZSTIDO**, dove ogni flag indica una precisa condizione desunta dal risultato dell'ultima istruzione eseguita (**CPAZSO**) oppure una modalità impostabile dal programmatore (**TID**):

- C, flag di **Carry** o **CF**, a 1 segnala l'avvenuto riporto (carry) o prestito (borrow) di una somma o di una sottrazione tra numeri naturali (senza segno);
- P, flag di **Parità** o **PF**, a 1 segnala che il risultato ha parità pari; a 0, parità dispari;
- A, flag **Ausiliario** o **AF**, come il flag di Carry, ma relativo a un nibble (solo i quattro bit meno significativi degli operandi);
- Z, flag di **Zero** o **ZF**, a 1 segnala che il risultato è zero. Fondamentale per sapere se due operandi sono uguali (esempio: tramite una sottrazione);
- S, flag di **Segno** o **SF**, a 1 segnala che il risultato ha segno negativo, nella rappresentazione in complemento a due;
- T, flag di **Trap** o **TF**, impostato a 1 obbliga il processore a eseguire le istruzioni passando il controllo all'utente dopo ogni singola esecuzione (per effettuare il Debug);
- I, flag di **Interruzione** o **IF**, impostato a 1 abilita le linee di interruzione hardware, impostato a 0 le disabilita;

- D, flag di **Direzione** o **DF**, impostato a 1 indica la direzione del trasferimento inverso (da un indirizzo alto a uno basso) nelle istruzioni stringa;
- O, flag di **Overflow** o **OF**, come il flag di Carry, ma su numeri interi (con segno).

I flag T, I e D si impostano a 1 e a 0 con le corrispondenti istruzioni: STT, CLT, STI, CLI, STD, CLD.

1 Sintassi e indirizzamenti

L'ISA x-86 è una ISA **a due indirizzi**, ovvero usa una sintassi in cui ogni istruzione possiede al più due operandi.

Questo implica che una normale operazione aritmetica, per esempio, deve prevedere il terzo operando (esempio, il risultato) in modo implicito, cioè predeterminato dall'istruzione stessa. Detti **destinazione** e **sorgente** i due operandi di un'istruzione, il primo operando (destinazione) conterrà anche il risultato dell'istruzione.

La sintassi generale di un'istruzione x-86 è la seguente:

2 indirizzi: **Op.Code** destinazione, sorgente
1 indirizzo: **Op.Code** destinazione
0 indirizzi: **Op.Code**

Le regole generali previste dalla sintassi:

- sorgente e destinazione possono essere un numero costante (**immediato**, o imm), un registro (reg) o una locazione di memoria (mem), ma mai, contemporaneamente, due locazioni di memoria;
- una costante (imm) non può mai essere una destinazione;
- sorgente e destinazione devono essere **concordi**, ovvero devono avere la stessa dimensione in bit;
- per indicare una locazione di memoria (mem) si pone l'indirizzo tra **parentesi quadre** (esempio [0] è la cella di indirizzo zero);
- se la dimensione del trasferimento non è deducibile dall'istruzione, bisogna specificarla con le parole riservate **byte ptr**, **word ptr**, ecc;
- i numeri costanti (imm) devono essere prefissati con uno 0 se espressi in esadecimale. Per esprimere in esadecimale i numeri costanti si usa il suffisso h.

Sorgenti e destinazioni delle istruzioni possono essere specificati in determinati modi previsti dall'ISA x-86, modi comunemente denominati **indirizzamenti**.

Per elencare le modalità degli indirizzamenti previsti dall'x-86 si usa l'istruzione di gran lunga più utilizzata dai programmi, l'istruzione che attiva un trasferimento, l'istruzione MOV, che trasferisce un byte o una word.

Flags

La **parità** di un valore indica se il numero di bit a 1 nel valore è un numero pari o dispari (0 è considerato pari).

Si dice che un valore ha **parità pari** se il numero di bit a 1 nel valore è pari (esempio: 44h ha parità pari, infatti 44h = 1000100b ha due bit a 1).

Si dice che un valore ha **parità dispari** se il numero di bit a 1 nel valore è dispari (esempio: 45h = 1000101b ha tre bit a 1, quindi ha parità dispari).

L'**overflow aritmetico** (o traboccamento) avviene quando il risultato di un'operazione (aritmetica) genera un risultato che eccede il contenitore destinato a rappresentarlo.

Esempio, 127×10 genera un overflow aritmetico se il risultato deve essere rappresentato in un byte (infatti 1270 > 255), mentre non genera overflow se il risultato deve essere rappresentato in una word (infatti 1270 < 65535).

Il **complemento a due** è un modo efficiente di rappresentare numeri interi (con segno) in formato binario.

I numeri positivi hanno la consueta rappresentazione, ma devono possedere necessariamente il bit MSB a zero.

Esempio: 7 = 111b = 0111b = + 7

Per un numero negativo, si prende il corrispondente numero positivo, lo si inverte bit a bit e poi si aggiunge 1 (operazione detta di complemento a due).

Esempio: −7 = C2(7) = C2(0111)b = (1000)b + (1)b = (1001)b = −7.

Il numero 0 ha sempre tutti i bit a 0.

Per questa ragione i numeri interi in complemento a due esprimibili in un byte vanno da – 128 a 0 e da 0 a + 127.

1.1 Istruzione MOV

Sintassi: **`MOV destinazione, sorgente`**
Scopo: Il contenuto di `sorgente` viene trasferito in `destinazione`.
Esempi: `MOV reg, imm` (**indirizzamento immediato**)

```
MOV AL,2
MOV CX, 0A
```

Nota: L'operando **`imm`** viene automaticamente convertito nel contenitore destinazione, cosicché il valore **`0Ah`** del secondo caso riempie anche il byte più significativo di CX con uno zero.

`MOV reg, reg` (**indirizzamento registro**)

```
MOV AX, BX
MOV AL, BL
```

Nota: I due registri operandi devono avere la stessa dimensione.

`MOV mem, reg` (**indirizzamento diretto**)

```
MOV reg, mem
MOV mem, imm
MOV [102], AL
MOV CX, [106]
MOV byte ptr [200], 3
```

Nota: Uno dei due operandi deve sempre specificare la dimensione del trasferimento. Nel primo caso vengono spostati 8 bit; nel secondo 16 bit, prelevati dall'indirizzo 106 e 107; nel terzo caso va specificata la dimensione del trasferimento (byte ptr).

`MOV mem, reg` (**indirizzamento indiretto**)

```
MOV reg, mem
MOV mem, imm
MOV [BX], AL
MOV AX, [DI+3]
MOV ES:[BX], 23
```

Nota: All'interno delle parentesi quadre si possono solo usare registri indice: BX, SI, DI, BP, corrispettivamente associati ai registri di segmento DS, DS, DS, SS. Nel terzo caso, volendo usare un diverso registro di segmento, bisogna specificarlo espressamente (**override di segmento**).

2 x-86 e MSDOS

Per scrivere programmi in Assembly per x-86 è necessario decidere su quale sistema operativo operare per avere a disposizione tutta la serie di **API** (*Application Programming Interface*) che consentono di accedere correttamente all'hardware di Input/Output della macchina, come per esempio stampare caratteri sullo schermo o acquisire valori da tastiera.

Senza le API di un sistema operativo a disposizione, la programmazione in Assembly rimane abbastanza frustrante, non potendo fornire dati a

runtime (input) né poter visualizzare i risultati delle elaborazioni (output). Inoltre le API di un sistema operativo consentono di usare correttamente e velocemente tutti gli altri dispositivi di I/O necessari per un programma applicativo standard, a partire dalla gestione dei dischi e relativa gestione dei file.

Storicamente l'architettura x-86 è stata distribuita con il sistema operativo **MSDOS** (o equivalente come PCDOS e DRDOS); inoltre anche i modelli più recenti che implementano l'IA-32 prevedono l'esecuzione di codice x-86 in particolari modalità del processore (modalità reale e modalità virtuale86), comunemente attraverso quelle che, sotto i sistemi operativi Windows, sono note come **Shell di MSDOS** (attraverso il comando **Cmd**).

Ciò non esclude il fatto che sia proficua anche la programmazione in Assembly x-86 per sistemi operativi differenti, come per esempio Linux (tramite la **sintassi AT&T**) o Windows-Win32 (denominata **WinAsm**).

2.1 Memoria

MSDOS è stato progettato per l'Intel 8086, pertanto «vede» una memoria a 20 bit, per un totale di $2^{20} = 1\,048\,576$ locazioni di memoria con indirizzi numerati da 0 a 1 048 575 (o da 00000h a fffffh). Per la particolare gestione della memoria utilizzata, gli indirizzi sono rappresentati con la **notazione segmentata** (Seg:Ofs). Qualora la parte spiazzamento (Ofs) di un indirizzo segmentato valga zero, esso si dice **indirizzo di paragrafo**.

MSDOS è un sistema operativo monoprogrammato e monoutente; può mandare in esecuzione un solo programma alla volta.

La memoria principale è suddivisa in tre aree tipiche, la **memoria di sistema**, la **memoria convenzionale**, la **memoria riservata**.

Nella memoria di sistema, molto ridotta, vengono salvate all'avvio alcune strutture dati fondamentali per il sistema operativo, come l'area dei vettori di Interrupt (o **IDT**, *Interrupt Descriptor Table*), che contiene gli indirizzi segmentati di 256 procedure associabili ad altrettante interruzioni.

Nella memoria convenzionale, la più ampia, viene conservato il **kernel** di MSDOS e il programma applicativo dell'utente.

Nella memoria riservata viene allocata una porzione di firmware (BIOS) e, in generale, viene allocato l'I/O mappato in memoria di schede controller di I/O installabili dall'utente.

Lo schema generale è il seguente:

Indirizzi (Hex)	Dimensione byte	Dimensione kB	Descrizione	Nome
00000-005FF	1536	61,5	Strutture dati e valori riservati a MSDOS	Memoria di sistema
00600-9FFFF	653823	640	IO.SYS, MDOS.SYS e COMMAND.COM, programmi applicativi	Memoria convenzionale
A0000-FFFFF	393215	384	ROM BIOS, ROM d'espansione di schede di I/O	Memoria riservata

2.2 Boot

Ogni sistema operativo deve avviarsi contando su software scritto all'interno del calcolatore (il firmware del BIOS) e su software scritto nel settore di avvio di un dispositivo avviabile (esempio, un hard disk, o in generale un dispositivo di memoria secondaria avviabile) nel cosiddetto **Boot Sector**.

Tutta la fase di avvio di un calcolatore è denominata **Bootstrap** o **Boot**, e la prima parte dell'avvio è indipendente dal sistema operativo, cioè è identica per ogni sistema operativo installabile sulla macchina.

Questa fase, denominata in breve **POST** (*Power On Self Test*) è a carico del firmware scritto dalla casa costruttrice nel BIOS, come si nota dalla classica schermata di avvio che ne riporta il logo.

I programmi presenti nel BIOS sono scritti in linguaggio Assembly, non essendo ancora disponibile alcun supporto per programmi ad alto livello.

Quando la fase di POST termina, essa individua un dispositivo avviabile, secondo l'ordine che è memorizzato nella EEPROM del BIOS, e cede, «alla cieca», il controllo al Boot Sector del dispostivo avviabile.

Il codice nel Boot Sector è ancora scritto in linguaggio Assembly, e usa qualche sottoprogramma messo a disposizione dal BIOS per cominciare a caricare i dati del sistema operativo da disco fisso (se il dispositivo avviabile era il disco fisso).

Il codice scritto nel Boot Sector di un dispositivo avviabile che contiene MSDOS carica quindi il primo programma che è contenuto nel file IO.SYS. Questo programma a sua volta carica il secondo file del kernel di MSDOS, che si chiama MSDOS.SYS.

Infine, MSDOS.SYS carica il terzo programma fondamentale di MSDOS, denominato COMMAND.COM.

Il processo di avvio prevede che i file di sistema, durante il boot, vadano a consultare due file di testo in cui l'utente può scrivere alcune istruzioni di configurazione specifiche per il proprio calcolatore. Questi file di configurazione sono CONFIG.SYS e AUTOEXEC.BAT.

CONFIG.SYS contiene parametri di configurazione personalizzata, solo letti da MSDOS, mentre AUTOEXEC.BAT contiene una lista di istruzioni personalizzate dall'utente da eseguire al termine della fase di avvio del sistema operativo.

Si può dire che il **kernel** di MSDOS (ovvero la parte di sistema operativo più profonda) è costituito da IO.SYS e MSDOS.SYS, che interagisce con l'hardware e con i programmi, mentre la **shell** di MSDOS (ovvero la parte di sistema operativo più superficiale) è costituita da COMMAND.COM, che interagisce con l'utente e il kernel.

MSDOS è completato da una serie di programmi residenti su disco che ne estendono la funzionalità (esempio, FORMAT.EXE). Questi programmi di sistema sono anche detti **comandi esterni**, per differenziarli da quelli direttamente eseguiti dalla shell, denominati **comandi interni** (esempio, COPY).

File eseguibili

Un file eseguibile per un sistema operativo, normalmente, viene creato da un programma specifico detto **linker** (in italiano, correlatore, programma che è parte del processo di compilazione).

Una delle attività del linker consiste nell'aggiungere due sezioni fondamentali al file eseguibile: il **formato** (o header) e l'**area di startup**.

I sistemi operativi di classe Windows usano il formato eseguibile denominato **PE** (*Portable Executable*), mentre i sistemi operativi di classe Linux usano il formato eseguibile denominato **ELF** (*Executable Linkable Format*).

Lo header di un file eseguibile contiene informazioni fondamentali affinché il sistema operativo possa, in fase di caricamento di un programma (**load time**), sapere come caricarlo correttamente in memoria e gestirlo durante l'esecuzione (**runtime**).

Il formato eseguibile di MSDOS è denominato **DOS MZ executable**, è ampio almeno 512 byte e inizia sempre con i codici Ascii 'M' e 'Z', le iniziali di uno storico programmatore Microsoft, Mark Zbikowski (anche il formato PE inizia sempre con la stringa "MZ").

L'area di startup invece viene collocata dal linker in modo che la prima istruzione del programma utente sia richiamabile dal sistema operativo per avviare il runtime.

I file eseguibili MSDOS di tipo .COM non possiedono né formato, né area di startup.

2.3 Formato dei programmi eseguibili e rilocazione

MSDOS riconosce come programmi eseguibili due tipi di formati che corrispondono a due tipi di file a loro volta distinti da un'estensione caratteristica: file eseguibili di tipo **COM** (con estensione .COM) e file eseguibili di tipo **EXE** (con estensione .EXE). Inoltre la shell è in grado di eseguire in serie una lista di comandi (interni o esterni) scritti, riga per riga, su un file di testo denominato *batch* e di estensione BAT (.BAT). Pur esssendo eseguito, un file .BAT non è un file eseguibile ma uno **script di shell**.

Gli eseguibili di tipo COM non possono essere più ampi di 64kB, mentre gli eseguibili di tipo EXE possono essere di qualsiasi dimensione, purché inferiore alla dimensione della memoria convenzionale disponibile (tipicamente 640 kB).

L'eseguibile di tipo COM è molto semplice: contiene direttamente la lista di istruzioni Assembly da eseguire, senza alcuna informazione ulteriore.

Quando la shell deve eseguire un file COM – su ordine dell'utente o perché presente in una riga di un file batch, decide a quale indirizzo di memoria caricarlo, operazione detta di **rilocazione statica** (o caricamento rilocante statico) di MSDOS.

Nei primi 256 byte, a partire da quell'indirizzo, MSDOS scrive una serie di informazioni canoniche dette **PSP** (*Program Segment Prefix*), quindi legge il file .COM da disco byte per byte e lo ricopia nello stesso ordine in memoria, subito dopo il PSP. Terminato il caricamento, MSDOS cede il controllo alla prima istruzione del programma COM in memoria, impostando il program counter sull'indirizzo iniziale del programma, che sarà necessariamente il valore 256 (100h). Quando il programma terminerà, dovrà preoccuparsi di riconsegnare il controllo a MSDOS, affinché il sistema ritorni nella condizione di partenza.

Il caricamento rilocante statico di MSDOS per i file COM avviene in questo modo:

1. MSDOS cerca una zona di memoria libera contigua di ampiezza 64 kB (ampiezza denominata **segmento**), a partire da un indirizzo di paragrafo nella memoria convenzionale (esempio: 10a0h:0000h).
2. A partire dall'indirizzo di paragrafo così prescelto scrive il PSP, all'interno del quale compaiono informazioni utili al programma che sta per essere caricato, come per esempio il suo nome e gli eventuali parametri su linea di comando che l'utente può aver aggiunto (per esempio nel comando FORMAT C:, la stringa C: è un parametro su linea di comando).
3. Subito dopo il PSP, il loader ricopia ordinatamente, byte per byte, il contenuto del file COM in memoria.
4. Imposta tutti i registri di segmento al valore della parte segmento dell'indirizzo di paragrafo prescelto (nel nostro caso 10a0h), il registro SP a ffffh, il registro IP a 0100h e i rimanenti registri a 0. L'impostazione del registro IP a 0100h coincide con l'impostazione del program counter (nel nostro caso CS:IP = 10a0h : 0100h), quindi l'esecuzione viene di fatto ceduta al programma.

Si deve ricordare che l'operazione di caricamento di un programma in

PSP

Il **PSP** (*Program Segment Prefix*) è una struttura dati residente in memoria e creata da MSDOS appena prima del caricamento di un programma applicativo.

In questa zona il sistema operativo scrive una serie di informazioni utili al programma e alcune sue zone sono dedicate a memorizzare dati di interscambio tra programma e sistema operativo.

Per i file eseguibili .COM il PSP inizia sempre all'indirizzo di paragrafo del segmento di memoria scelto da MSDOS per allocare il programma applicativo, cosicché la prima istruzione di un programma COM deve avere la parte offset dell'indirizzo iniziale uguale a **100h**.

Nel PSP, tra le altre informazioni, sono memorizzati i valori digitati sulla linea di comando.

Esempio **C:\>pippo ciao mare**

In questo caso il programma pippo.com è stato avviato con due parametri su linea di comando (ciao e mare), pertanto consultando il PSP del programma queste due stringhe possono essere ricavate. **Il primo parametro su linea di comando è sempre il nome completo del file eseguibile**.

memoria, e sua effettiva esecuzione, viene fatta dal cosiddetto **loader** del sistema operativo; nel nostro caso il loader di MSDOS risiede all'interno di COMMAND.COM.

Una volta in memoria e in esecuzione, il **programma** prende il nome di **processo** (mentre «programma» è il nome delle istruzioni contenute nel file).

Il valore costante 0100h impostato su IP, coincide con la dimensione del PSP (100h = 256d), così da indicare sempre ed esattamente l'indirizzo di memoria della prima istruzione del programma caricato.

La ragione dell'impostazione al valore costante ffffh del registro SP la si vedrà quando si studierà lo stack.

I file eseguibili di tipo EXE vengono caricati in modo sostanzialmente differente, dato che la loro ampiezza può superare il limite del segmento (limite di 64kB, cioè la dimensione ottenibile usando i 16 bit di un registro x-86).

È necessario che ogni file EXE contenga, oltre al proprio codice eseguibile, anche una serie di informazioni supplementari che serviranno al loader di MSDOS per poterlo caricare correttamente in memoria. In questo caso si parla ancora di rilocazione statica, ma anche di **caricamento rilocante dinamico**.

Le informazioni supplementari contenute in un file EXE si dicono **header** del file EXE, e sono l'intestazione del file.

Esse sono ampie almeno 512 byte, cosicché un file EXE è sempre almeno ampio 514 byte (due byte servono comunque per l'istruzione di terminazione). L'header di un file EXE inizia sempre con due byte costanti, che sono i primi due byte di qualsiasi file EXE: MZ. Nell'header del file EXE sono memorizzate in un formato specifico, e a cura dei linker dei compilatori, tutte le informazioni necessarie al loader di MSDOS per caricare il programma in memoria. Queste informazioni riguardano per esempio la quantità di blocchi da 64kB (segmenti) di cui è costituito il programma EXE e le loro effetive dimensioni, e quale, tra questi, è il segmento di codice di partenza.

Il caricamento rilocante dinamico di MSDOS per i file EXE avviene in questo modo:

1. Il loader di MSDOS legge lo header del file EXE e cerca tante zone di memoria libere (e contigue) di ampiezza 64kB (segmenti), quanti ne sono elencati nello header, a partire da indirizzi di paragrafo. Uno di questi sarà l'indirizzo di paragrafo (e il relativo segmento) di avvio del programma (esempio: 10B0h:0000h), mentre un altro sarà l'indirizzo di paragrafo (e il relativo segmento) dei dati del programma.
2. A partire dall'indirizzo di paragrafo dei dati, MSDOS scrive il PSP del programma EXE.
3. A partire dall'indirizzo di paragrafo del segmento di avvio del programma, il loader ricopia il file EXE nei vari segmenti individuati precedentemente, avendo cura di correggere gli indirizzi che ne sono contenuti in accordo con la posizione dei valori nei segmenti di memoria libera individuati precedentemente (rilocazione su caricamento).
4. Infine imposta il registro di segmento del codice (CS) al valore (seg)

dell'indirizzo di avvio del programma (esempio: 10b0h), il registro di segmento dei dati (DS) al valore (seg) del segmento del PSP, il registro SP a ffffh, il registro IP a 0000h e i rimanenti registri a 0. L'impostazione del registro IP a 0000h coincide con l'impostazione del Program Counter (nel nostro caso CS:IP = 10b0h : 0000h), quindi l'esecuzione viene di fatto ceduta al programma.

Come è evidente, il caricamento di un file EXE è più laborioso e richiede più tempo. Come si può notare, IP viene posto inizialmente a 0, a differenza del valore iniziale che assume nel caso di caricamento di file COM, dato che il PSP, nei file EXE, è allocato sull'indirizzo di paragrafo dei dati e non del codice.

3 API, interruzioni software e servizi

Quando MSDOS è operativo, esso fornisce una interfaccia pubblica a numerose funzioni fondamentali che consentono ai programmatori di accedere velocemente al video, alla tastiera, ai dischi e a tutti i dispositivi installati dal sistema, evitando di conoscere i dettagli e le complessità dell'accesso all'hardware di I/O.

Queste funzioni, in generale, sono dette **API** (*Application Programming Interface*) del sistema operativo.

Per MSDOS le API di sistema sono scritte in Assembly e invocabili tramite una speciale istruzione dell'ISA x-86 denominata **INT** o interruzione software. Per i sistemi operativi Win32 (Windows) e Linux, invece, le API sono scritte in linguaggio C, e sono radicalmente differenti.

Tutte le 256 interruzioni del sistema operativo sono numerate da 0 a 255 (0-ffh), e si distinguono in **interruzioni hardware** (o **IRQ**, le prime 16, da 0 a fh), **interruzioni software del BIOS** (da 10h a 1fh), **interruzioni software di MSDOS** (da 20h a 2fh). Le successive, da 30h a 3fh, non sono documentate, mentre le rimanenti (da 40h a ffh) sono disponibili per usi definibili dall'utente.

Molto spesso le interruzioni software forniscono più di una funzione d'uso, cosicché è necessario selezionare una precisa sottofunzione di una interruzione specifica, prima di invocarla. Ogni sottofunzione di una interruzione è anche detta **servizio dell'interruzione**, ed è adeguatamente numerata in modo che la selezione sia univoca. Il numero di sottofunzione di una interruzione va sempre specificato nel semiregistro AH prima della chiamata all'interruzione.

Il funzionamento delle interruzioni, che sono in realtà dei sottoprogrammi (o subroutine), è semplice: quando il chiamante invoca l'interruzione di numero X, il sistema calcola l'indirizzo assoluto X * 4, si reca in memoria a questo indirizzo (0000h : X * 4), legge i quattro byte consecutivi a partire da questo indirizzo, imposta il Program Counter CS:IP con il valore di questi 4 byte e cede il controllo, «alla cieca», al programma che si trova a quell'indirizzo, programma detto anche routine associata all'interruzione (o **ISR**, *Interrupt Service Routine*).

Interruzioni

L'interruzione (o **interrupt**) è un concetto fondamentale per ogni architettura. Si tratta di un sistema per far intervenire un programma improvvisamente e mentre un altro è attualmente in esecuzione. Per questa ragione l'interruzione è un meccanismo **asincrono**, quasi sempre realizzato tramite un meccanismo integrato tra hardware, sistema operativo e programma.

Si immagini di dover gestire il puntatore di un mouse affinché un programma possa visualizzarlo sempre al posto giusto e reagire se premuti i tasti.

Siccome non è possibile sapere quando l'utente agirà con il mouse (evento asincrono), il programmatore è costretto a leggere la posizione e i pulsanti del mouse in continuazione. Questa tecnica di programmazione è detta a **controllo di programma** (o **polling**).

Purtroppo la tecnica polling ha molti limiti, tra cui quello di occupare senza effettiva ragione molto tempo di CPU.

Se invece ci si affida ad una **tecnica ad interruzione**, è sufficiente informare il sistema operativo che si intende avviare il codice di controllo del mouse (detto ISR, *Interrupt Service Routine*) solo quando la periferica è azionata.

La tecnica ad interruzione è molto efficiente ma anche più complessa da programmare, dovendo accordarsi con le regole del sistema operativo.

Per quanto riguarda le API di MSDOS superiori a 0fh, pur essendo ISR, esse non vengono avviate con l'automatismo descritto, ma con una chiamata utente (INT n) che avvia l'ISR associata ottenendo l'effetto di una chiamata a procedura standard.

Le istruzioni di I/O

Nell'ISA x-86 le istruzioni Assembly per accedere (in lettura e in scrittura) allo spazio di indirizzamento di Input/Ouptut sono IN e OUT.

Istruzione IN

Sintassi: **IN acc, porta**

Scopo: Viene letto il valore dall'indirizzo `porta` e depositato in `acc`.

Esempi:
```
IN AL,45h
IN AX,DX
```

Nota: `acc` può essere solo il registro AX (o AL). Se AX, verranno letti due byte a partire da `porta`; se AL, uno solo. `Porta` può essere solo il registro DX.

→

Una speciale istruzione, che deve sempre essere presente come ultima di una ISR, consente di reimpostare il program counter all'indirizzo successivo a quello dell'invocazione, così l'esecuzione ritorna automaticamente al chiamante esattamente nel punto in cui era stata interrotta.

Infatti l'inizio della memoria di MSDOS (memoria di sistema), contiene la tabella dei vettori di interruzione (**IDT**, *Interrupt Descriptor Table*), cioè una serie di 256*4 = 1024 byte che corrispondono agli indirizzi di 256 sottoprogrammi già disponibili in memoria.

La differenza tra interruzioni software e interruzioni hardware riguarda solo il modo in cui vengono invocate: se l'interruzione è software, è il programmatore che la invoca con l'istruzione x-86 INT che ha inserito nel codice del suo programma. La chiamata è detta sincrona.

Se l'interruzione è hardware, la chiamata viene fatta automaticamente dalla CPU, in questo caso si parla di **eccezione**, o dal dispositivo di I/O a cui quell'interruzione è associata. La chiamata è detta asincrona.

Bisogna dire che non tutte le interruzioni software possiedono una ISR all'indirizzo della IDT corrispondente. In alcuni casi presso quell'indirizzo risiedono strutture dati in tabelle, contenenti dati necessari per la configurazione di determinati dispositivi.

3.1 Istruzione INT

Sintassi: **INT num**

Scopo: Viene avviata l'interruzione software di numero `num`.

Esempi:
```
INT 20
INT 21
INT 10
```

Nota: In molte occasioni, prima di usare l'istruzione INT, deve essere impostato nel registro AH il numero del servizio.

In generale, il meccanismo dell'interruzione così descritto consente di virtualizzare l'accesso all'hardware, che è la parte più complessa della programmazione per qualsiasi ambiente.

La possibilità di modificare la IDT, e quindi le varie ISR associate, consente di caricare in memoria il codice più aggiornato, o più efficiente o più adeguato per ogni dispositivo di I/O disponibile anche in futuro, lasciando intatte le chiamate dei programmi alle funzioni di gestione. La IDT può essere aggiornata sia in fase di caricamento del sistema operativo, così come fa MSDOS con i file IO.SYS, MSDOS.SYS e COMMAND.COM, sia caricando «al volo» codice specifico in moduli speciali detti **driver**, che in MSDOS hanno estensione .SYS, sia operando la modifica della IDT durante l'esecuzione di un programma utente (cioè a runtime, badando però di risistemare le impostazioni al termine dell'esecuzione).

In ogni caso le API del BIOS (10h-1fh) sono sempre disponibili nella loro forma originale, garantite dalla immodificabilità del firmware su BIOS (benché IO.SYS ne intercetti le chiamate e le normalizzi).

4 Video e tastiera con le interruzioni software del BIOS e di MSDOS

Le istruzioni assembler necessarie per accedere alla console (video e tastiera) sono necessarie per poter scrivere un qualsiasi programma significativo.

Le principali funzioni per accedere al video e alla tastiera sono contenute nel sottoinsieme delle interruzioni software del BIOS, in particolare la 10h (e sue sottofunzioni) per il video e la 16h (e sue sottofunzioni) per la tastiera.

Le funzioni di accesso alla console fornite dalle interruzioni del BIOS sono importanti perché sempre disponibili fin dall'avvio del calcolatore, essendo scritte in firmware.

n. Interruzione BIOS	Descrizione
10h	**Funzioni per la gestione del video**
11h	Funzioni per la determinazione della dotazione del computer
12h	Funzioni per determinare la memoria
13h	Funzioni per la gestione dei dischi
14h	Funzioni per la gestione delle porte seriali
15h	Funzioni per la gestione estesa del sistema
16h	**Funzioni per la gestione della tastiera**
17h	Funzioni per la gestione della stampante
18h	Caricatore del Basic IBM (obsoleta)
19h	Esecuzione del Bootstrap da disco
1Ah	Funzioni per la gestione dell'orologio in tempo reale
1Bh	Procedura utente per la gestione della tastiera
1Ch	Procedura utente per la gestione del timer di sistema
1Dh	Tabella di inizializzazione del video
1Eh	Tabella dei parametri dei floppy disk
1Fh	Tabella dei caratteri video

→

Istruzione OUT

Sintassi: **OUT porta, acc**

Scopo: Viene scritto il valore in `acc` all'indirizzo `porta`

Esempi:
```
OUT 41,AL
OUT DX,AX
```

Nota: `acc` può essere solo il registro AX (o AL). `porta` può essere solo il registro DX.

Per i sistemi operativi win32 (e win64) come WindowsXP, Vista e Seven **le istruzioni IN e OUT sono ammesse solo in kernel mode**, pertanto il codice eseguibile x-86 user mode che contiene istruzioni IN e OUT su tali sistemi non avrà effetto.
Per eseguire questo tipo di programmi bisogna che il microprocessore operi in real mode, per esempio effettuando il boot tramite il sistema operativo MSDOS.

4.1 INT 10h – Stampa di un carattere sullo schermo

```
MOV AL, 30    ; codice Ascii del carattere da stampare a video (esempio:
              ; 30h è lo zero, o '0')
MOV AH, 0E    ; numero del servizio
INT 10        ; interruzione sw del BIOS gestione video
```

Il carattere da stampare a schermo va sempre fornito con il suo codice Ascii, pertanto la stampa di singoli numeri decimali (da 0 a 9) deve essere sempre *normalizzata*, aggiungendo 48 (o 30h) al numero da stampare.

Si può indicare il codice Ascii di un qualsiasi carattere indicandolo tra singoli apici, così come in linguaggio C.

4.2 INT 16h – Input di un carattere da tastiera

```
MOV AH, 00    ; numero del servizio
INT 16        ; interruzione sw del BIOS gestione tastiera
              ; in AL il codice Ascii del carattere premuto, in AH il codi-
                ce di scansione
```

Naturalmente l'esecuzione di questa interruzione blocca il flusso del programma in esecuzione, che rimane in attesa di un carattere digitato dalla tastiera. Non appena un carattere viene premuto, la routine ritorna avendo memorizzato in AL il codice Ascii del carattere premuto.

Le stesse funzioni di accesso alla console possono essere richieste tramite l'interruzione software di MSDOS denominata INT 21h.

L'effetto è sostanzialmente identico, anche se bisogna ricordare che tali routine non sono disponibili fin dall'avvio del sistema, ma solo allorquando MSDOS è completamente caricato. Per i programmi utente questa distinzione non è rilevante.

4.3 INT 21h - Stampa di un carattere sullo schermo

```
MOV DL, 30    ; codice Ascii del carattere da stampare a video (esempio:
                30h è lo zero, o '0')
MOV AH, 02    ; sottofunzione
INT 21        ; interruzione sw di MSDOS
```

Il carattere da stampare a schermo va sempre fornito con il suo codice Ascii, pertanto la stampa di singoli numeri decimali (da 0 a 9) deve essere sempre *normalizzata*, aggiungendo 48 (o 30h) al numero da stampare.

Si può indicare il codice Ascii di un qualsiasi carattere indicandolo tra singoli apici, così come in linguaggio C.

4.4 INT 21h - Input di un carattere da tastiera

```
MOV AH, 01    ; sottofunzione
INT 21        ; interruzione sw di MSDOS
              ; in AL il codice Ascii del carattere premuto
```

Naturalmente l'esecuzione di questa interruzione blocca il flusso del programma in esecuzione, che rimane in attesa di un carattere digitato alla tastiera. Non appena un carattere viene premuto, la routine ritorna avendo memorizzato in AL il codice Ascii del carattere premuto.

A differenza della sua gemella del BIOS, questa funzione mostra a video il carattere premuto.

Ogni programma scritto per MSDOS in assembler x-86 quando termina deve avvisare il sistema operativo tramite un'interruzione sw specifica.

In questo modo il sistema operativo riacquisisce il controllo del calcolatore correttamente, riconfigurandosi opportunamente per riprendere la

sessione di lavoro in attesa del lancio di un nuovo programma eseguibile da parte dell'utente. Tramite l'avviso di terminazione, che deve essere sempre l'ultima istruzione assembler di ogni programma sia EXE, sia COM, il programma può avvisare il sistema operativo sullo stato della propria terminazione, indicando, per esempio, eventuali terminazioni anomale o, più frequentemente, una terminazione regolare.

4.5 INT 20h – Terminazione di un programma COM

```
INT 20          ; interruzione sw di MSDOS Terminazione programma
                  COM
```

L'interruzione non necessita di alcun parametro, ma non consente di avvisare MSDOS sullo stato di terminazione.

4.6 INT 21h – Terminazione di un programma EXE

```
MOV AL, 00      ; codice di terminazione. Se 0 = terminazione regolare
MOV AH, 4C      ; sottofunzione
INT 21          ; interruzione sw di MSDOS Terminazione programma
                  EXE
```

L'interruzione di terminazione dei file eseguibili EXE consente di avvisare MSDOS sullo stato di terminazione, inseribile nel semiregistro AL. MSDOS può valutare questo valore usando l'istruzione ERRORLEVEL, magari in un comando batch.

Se si volesse usare lo stato di terminazione anche per un file eseguibile COM, si può usare questa interruzione.

ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1. Elencare tutti i bit del registro Flags dell'8086, mandando a memoria la sequenza delle loro iniziali.
2. Mostrare un esempio di operazione che imposta a 1 lo zero flag.
3. Mostrare un esempio di operazione che imposta a 1 il flag di overflow.
4. Spiegare cosa significa che due operandi sono concordi.
5. Elencare almeno tre tipi di indirizzamento.
6. Fornire tre esempi di istruzione MOV per tre tipi di indirizzamento differenti.
7. Spiegare la fase di avvio di un sistema, suddividendola tra la fase di avvio della macchina e del sistema operativo MSDOS.
8. Confrontare le fasi di caricamento di un file COM e di un file EXE per il sistema operativo MSDOS.
9. Definire e commentare la nozione generale di API per un generico sistema operativo e per MSDOS.
10. Mostrare un esempio di istruzione INT in relazione al concetto di sottofunzione (o servizio).
11. Scrivere su un foglio le istruzioni Assembly per stampare a schermo l'iniziale maiuscola del proprio cognome.
12. Scrivere su un foglio le istruzioni Assembly per acquisire un tasto da console.
13. Elencare i soli registri dell'8086 che possono indirizzare in memoria (ovvero che possono comparire all'interno delle parentesi quadre).

Requisiti avanzati

1. Riportare gli esempi di quattro operazioni che originano rispettivamente PF=0, PF=1, SF=0, SF=1.
2. Mostrare con un esempio l'istruzione MOV usata correttamente anche se gli operandi sono discordi.
3. Riportare l'esempio di tre numeri (massimo 7 bit) con segno negativo in rappresentazione complemento a due.
4. Scrivere su un foglio due istruzioni MOV che utilizzano l'indirizzamento indiretto.
5. Definire la locuzione «indirizzo di paragrafo» e spiegare quando viene usata.
6. Spiegare la nozione di script di shell ed elencare i file di configurazione di MSDOS.
7. Spiegare il PSP e la sua funzione.
8. Illustrare il concetto di header (di un file eseguibile) ed elencarne le sigle per i maggiori sistemi operativi.
9. Riportare un esempio in cui si distinguano alcuni parametri su linea di comando. Ricordare anche quale parametro è sempre presente.
10. Elencare le tre categorie di interruzioni in MSDOS e confrontarle.
11. Riportare le istruzioni Assembly dell'x-86 per l'I/O e commentarne i limiti di utilizzazione.
12. Scrivere su un foglio la sequenza di istruzioni per acquisire in input un carattere, stamparlo a video e terminare un programma.
13. Scrivere su un foglio i codici Ascii *normalizzati* delle cifre del numero 529.

Assembly con Debug.exe

A4

Il modo più semplice per scrivere programmi in Assembly in formato x-86 è farlo tramite il programma di MSDOS denominato **Debug.exe** fornito su tutte le piattaforme Microsoft Win32 tramite la **Shell di MSDOS**, invocabile a sua volta usando il comando **Cmd**.

Con questo programma si scrive codice Assembly x-86 in formato COM e con indirizzi assoluti numerici, imponendo una rigorosa disciplina nella scrittura delle istruzioni e mostrando la pratica reale della programmazione in assembler. Il programma Debug.exe è a sua volta scritto in codice x-86 a 16 bit. Inoltre, il fatto che Debug.exe sia disponibile su ogni piattaforma Microsoft x-86 (ovvero da MSDOS fino a Win32), ne garantisce la continuità didattica e la reperibilità per gli studenti.

La scelta ricade su questo «ambiente» (utilizzabile anche tramite file di testo «sorgenti») proprio per mettere in evidenza l'aspetto didattico della programmazione a basso livello, le sue interazioni con il sistema operativo e i suoi meccanismi dipendenti dall'ISA del microprocessore.

In particolare, il dover usare indirizzi assoluti numerici (e non etichette come negli ambienti classici tipo Tasm, Masm o Nasm), spinge lo studente a organizzare il codice in modo ordinato e razionale, e lo rende cosciente dei meccanismi più profondi che riguardano la compilazione del codice sorgente in generale.

Tutto ciò anche alla luce del fatto che ormai la programmazione Assembly sta scomparendo anche dalle piattaforme più a basso livello (come la programmazione di microprocessori e microcontrollori vari), sostituita da ambienti di programmazione più versatili (dal linguaggio C a Java) e quindi non ha molto significato proporla come reale caso d'uso per la programmazione di applicazioni utente. *In ogni caso, è disponibile online una versione completa del corso per l'ambiente Tasm/Masm, dotata di lezione testuale, codice esemplificativo e guida all'uso del compilatore Tasm e del linker Tlink.*

EQUIVALENTE CORSO ASSEMBLY PER TASM/MASM

MSDOS e Windows

I moderni sistemi operativi Microsoft a 32 bit (**Win32**) consentono di avviare sessioni del sistema operativo MSDOS in due modi: tramite l'avvio del programma **Command.com** o del programma **Cmd.exe**.

Nel primo caso la finestra simula nel modo più preciso possibile l'interprete dei comandi di MSDOS (esempio: non vengono riconosciuti i nomi di file lunghi, cioè di lunghezza superiore a 8 caratteri). Il codice eseguibile a disposizione è solo il codice x-86 a 16bit.

Nel secondo caso viene «virtualizzato» il programma Command.com in una sua versione analoga, ma a 32bit (modo virtual86 del microprocessore). In questo caso si parla di **Shell di MSDOS**.

Per sperimentare la programmazione Assembly per MSDOS è meglio, quando possibile, usare la Shell di MSDOS, dato che consente di sfruttare tutte le estensioni previste da Win32.

In ogni caso è sempre possibile avviare la macchina da un dispositivo avviabile (esempio: Cd-Rom o pendrive) contenente il sistema operativo originale, oggi gratuito (modalità reale del microprocessore).

1 Debug.exe

Debug è un ambiente con interfaccia a carattere, con un prompt (il trattino) che attende un comando dell'utente.

Tutti i numeri utilizzati con Debug sono sempre in formato esadecimale; per usare una diversa rappresentazione va specificato il formato in coda al numero.

I comandi sono singoli caratteri di facile comprensione. Debug non è *case sensitive*, ovvero non distingue se il comando è scritto in maiuscolo o in minuscolo, comportandosi allo stesso modo in entrambi i casi. In evidenza, i comandi principali.

Comando	Nome	Descrizione
`A [indirizzo]`	**Assembla**	Consente di scrivere un programma
`C intervallo indirizzo`	Confronta	
`D [intervallo]`	**Dump**	Esplora la memoria
`E indirizzo [elenco]`	Immetti	
`F intervallo elenco`	Riempi	
`G [=indirizzo] [indirizzi]`	Vai	
`H valore1 valore2`	Esadecimale	
`I portadiI/O`	Input	
`L [indirizzo][unità][settore][numero]`	**Carica**	
`M intervallo indirizzo`	Muovi	
`N [nomefile][elencoargomenti]`	**Nomina**	Assegna il nome al programma
`O portadiI/O valore`	Output	
`P [=indirizzo][numero]`	Procedi	
`Q`	**Esci**	Esce da Debug
`R [registro]`	**Registro**	Esplora e/o imposta i registri
`S intervallo elenco`	Cerca	
`T [=indirizzo][valore]`	Traccia	
`U [intervallo]`	**Disassembla**	Consente di modificare il programma
`W [indirizzo][unità][settore][numero]`	**Scrivi**	Salva su disco il programma

Win64

Con i sistemi operativi Microsoft a 64bit (Win64, come XP64, Vista64 e Seven64), non è più possibile usare Command.com, e la shell di MSDOS non consente più di lanciare programmi x-86 a 16bit come, per esempio, Debug.exe (e anche Tasm/Tlink).
Per ovviare a questa situazione si può installare, su questi sistemi, un programma emulatore gratuito di MSDOS x-86 denominato **DOSBox**, oppure il programma emulatore gratuito di processore x-86 denominato **EMU8086** (vedi Appendice).

Con il **Comando D** (Dump) Debug consente di esplorare la memoria principale, dall'indirizzo 0 all'indirizzo fffffh.

Naturalmente la modalità d'uso e rappresentazione del comando prevede la sintassi con indirizzi segmentati (Seg:Ofs) e i numeri rappresentanti gli indirizzi e i contenuti delle locazioni sono sempre espressi in formato esadecimale.

Al prompt di Debug (il trattino) si può quindi usare **D** [**intervallo**], potendo specificare l'indirizzo di partenza da cui analizzare la memoria o un intervallo:

```
d          Mostra la memoria a partire dall'indirizzo CS:IP
d 0:0      Mostra la memoria a partire dall'indirizzo 0h:0h
d 0:0 100  Mostra la memoria a partire dall'indirizzo 0h:0h e per 256
           celle (100h = 256)
```

Le videate del comando **D** sono costituite normalmente da 8 righe così strutturate:

- Sezione di sinistra: mostra l'indirizzo segmentato della prima locazione di memoria riportata nella sezione centrale. L'indirizzo avanza di 10h, dato che su ogni riga sono rappresentate 16 celle di memoria da un byte l'una.

```
C:\>debug
-d
0CE2:0100  4D 00 00 5F 00 B4 02 CD-21 FE C2 E2 F8 CD 20 6F   M.._....!..... o
0CE2:0110  43 4F 4D 53 50 45 43 3D-43 3A 5C 57 34 00 D1 0C   COMSPEC=C:\W4...
0CE2:0120  57 53 5C 53 59 53 54 45-4D 33 32 5C 43 4F 4D 4D   WS\SYSTEM32\COMM
0CE2:0130  41 4E 44 2E 43 4F 4D 00-41 4C 4C 55 53 45 52 53   AND.COM.ALLUSERS
0CE2:0140  50 52 4F 46 49 4C 45 3D-43 3A 5C 44 4F 43 55 4D   PROFILE=C:\DOCUM
0CE2:0150  45 7E 31 5C 41 4C 4C 55-53 45 7E 31 00 41 50 50   E~1\ALLUSE~1.APP
0CE2:0160  44 41 54 41 3D 43 3A 5C-44 4F 43 55 4D 45 7E 31   DATA=C:\DOCUME~1
0CE2:0170  5C 6F 6C 6C 61 72 69 5C-44 41 54 49 41 50 7E 31   \ollari\DATIAP~1
-d 0:0
0000:0000  68 10 A7 00 8B 01 70 00-16 00 8D 03 8B 01 70 00   h.....p.......p.
0000:0010  8B 01 70 00 B9 06 0C 02-40 07 0C 02 FF 03 0C 02   ..p.....@.......
0000:0020  46 07 0C 02 EC 06 97 05-3A 00 8D 03 54 00 8D 03   F.......:...T...
0000:0030  6E 00 8D 03 88 00 8D 03-A2 00 8D 03 FF 03 0C 02   n...............
0000:0040  A9 08 0C 02 A4 09 0C 02-AA 09 0C 02 5D 04 0C 02   ............]...
0000:0050  B0 09 0C 02 0D 02 DB 02-C4 09 0C 02 8B 05 0C 02   ................
0000:0060  0E 0C 0C 02 14 0C 0C 02-1F 0C 0C 02 AD 06 0C 02   ................
0000:0070  AD 06 0C 02 A4 F0 00 F0-37 05 0C 02 61 90 00 C0   ........7...a...
-q
C:\>
```

- Sezione centrale: elenca il contenuto delle 16 locazioni consecutive, a partire da quella indirizzata dalla prima sezione. Per maggior chiarezza la sequenza dei 16 byte di ciascuna riga è divisa in due gruppi da 8 byte divisi da un trattino.
- Sezione destra: mostra le stesse 16 celle di memoria in formato Ascii stampabile, cioè escludendo i simboli con codici Ascii da 00h a 1fh (caratteri di controllo) e da 80h a ffh (caratteri Ascii estesi). Se il carattere non è stampabile, viene sostituito da un punto.

Se il comando **D** viene ripetuto, è riportata a video la successiva serie di 128 byte, e così ad oltranza.

Nel secondo blocco dell'esempio, i 128 byte riportati sono i 32 indirizzi (128/4 = 32) delle prime 32 interruzioni di MSDOS. Infatti, alla base della memoria si trova la IDT.

1.1 Consultare i registri

Con il **Comando R** (Register) Debug consente di visualizzare il contenuto e lo stato dei registri x-86, nonché impostarne il valore.

Naturalmente i valori visualizzati sono espressi in esadecimale, mentre il registro dei Flags è mostrato attraverso delle sigle convenzionali.

Al prompt di Debug (il trattino) si può quindi usare **R** [**registro**], specificando un registro se si vuole impostarne il valore. Infine, si può modificare un singolo bit del registro dei Flags con il comando **RF**, specificando la lista dei nuovi valori (consultare la tabella di pagina seguente):

`r`	Visualizza lo stato dei registri
`r CX`	Imposta il registro CX
`r f`	Imposta uno o più bit del registro dei Flags

Si nota che oltre ai registri, il comando **R** mostra sempre l'indirizzo, l'Op. Code e il codice mnemonico dell'istruzione puntata dal program counter (CS:IP).

```
C:\>debug
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0CED  ES=0CED  SS=0CED  CS=0CED  IP=0100   NV UP EI PL NZ NA PO NC
0CED:0100 B80100        MOV  AX,0001
-r cx
CX 0000
:12
-r
AX=0000  BX=0000  CX=0012  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0CED  ES=0CED  SS=0CED  CS=0CED  IP=0100   NV UP EI PL NZ NA PO NC
0CED:0100 B80100        MOV  AX,0001
-rf
NV UP EI PL NZ NA PO NC  -pe
-rf
NV UP EI PL NZ NA PE NC  -zr po
-r
AX=0000  BX=0000  CX=0012  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0CED  ES=0CED  SS=0CED  CS=0CED  IP=0100   NV UP EI PL ZR NA PO NC
0CED:0100 B80100        MOV  AX,0001
-q
C:\>
```

Nell'esempio riportato, all'indirizzo (0CEDh:0100h) si trova l'Op.Code B80100h, che corrisponde all'istruzione mnemonica MOV AX, 1

Per quanto riguarda il registro Flags, vengono mostrati solo otto stati di otto bit significativi, ognuno con una coppia di caratteri.

La decodifica delle coppie di caratteri del registro Flags è la seguente:

Flag	Descrizione		Descrizione	
	bit = 0	bit = 1		
Overflow	NV	OV	NV: no Overflow;	OV: Overflow
Direzione	UP	DN	UP: incremento;	DN: decremento
Interruzione	DI	EI	DI: disabilitate;	EI: abilitate
Segno	PL	NG	PL: segno positivo;	NG: segno negativo
Zero	NZ	ZR	NZ: no zero;	ZR; risultato = 0
Ausiliario	NA	AC		
Parità	PO	PE	PO: parità pari;	PE: parità dispari
Carry	NC	CY	NC: nessun riporto;	CY: riporto

2 Scrivere un programma

La sequenza di passi per scrivere un programma x-86 per MSDOS in formato COM con Debug è la seguente:

1. **Comando A** – scrivere le istruzioni (edit)
2. Terminare la fase di edit con un invio su una riga vuota
3. **Comando R** – impostare la lunghezza del programma
4. **Comando N** – assegnare il nome al programma
5. **Comando W** – salvare il programma su disco

Il **comando A** (Assembly) deve essere specificato con l'[indirizzo] impostato a

100h, per rispettare la regola di MSDOS circa il PSP: la prima istruzione di un programma .COM deve sempre trovarsi all'indirizzo di spiazzamento 100h.

Quindi si digita A 100 per cominciare la fase di edit.

Ad ogni pressione del tasto Enter (invio) l'istruzione è confermata e si può scrivere la successiva.

Debug riporta sempre l'indirizzo delle istruzioni digitate.

Per terminare l'editazione del programma, bisogna premere Enter su una riga vuota. Debug, a questo punto, mostra il prompt (il trattino), ed è pronto per un nuovo comando.

Controllando l'indirizzo dell'ultima istruzione scritta, sottraendo l'indirizzo iniziale (sempre 100h), si calcola facilmente la dimensione, in byte, del programma scritto. Questa dimensione va specificata nel registro CX tramite il **comando R**, nella forma **R CX**. Si digita la dimensione e la si conferma con Enter. Debug mostra di nuovo il prompt.

Ora si può assegnare il nome al programma con il **comando N** (Name). Il nome deve rispettare le regole per gli identificatori di MSDOS, ovvero non può essere più lungo di 8 caratteri e non può contenere caratteri speciali (esempio: lo spazio). Va sempre specificata l'estensione COM. Quindi il comando da utilizzare è il seguente (se si sceglie pippo come nome): **N PIPPO.COM.** Digitato Enter, Debug mostra di nuovo il prompt (il trattino).

PROGRAMMA

Il programma più corto del mondo

Il programma più corto del mondo per x-86 è un programma eseguibile COM la cui unica attività è terminarsi correttamente e restituire il controllo a MSDOS.
Il programma è lungo 2 byte.

Nel dettaglio, tutti i passi da usare con Debug per scrivere questo programma e poi avviarlo.

`C:\>`	Prompt di MSDOS
`C:\>debug`	Lancio del programma Debug da MSDOS
`-a 100`	Edit del programma a partire dall'indirizzo 100h
`0CE1:0100 int 20`	Unica riga di codice: terminazione di file .COM
`0CE1:0102`	Riga vuota: termine dell'edit
`-r cx`	Comando per l'impostazione della dimensione
`CX 0000`	Visualizzazione del contenuto attuale del registro CX da parte di Debug
`:2`	Impostazione della dimensione (**100h-102h=2h**)
`-n corto.com`	Assegnazione del nome (corto.com)
`-w`	Comando per la scrittura del file di programma su disco
`Scrittura di 002 byte in corso`	Visualizzazione della quantità di byte scritti da parte di Debug
`-q`	Uscita da Debug
`C:\>corto`	Lancio del programma corto.com da MSDOS
	Nessun output del programma
`C:\>`	Prompt di MSDOS

Si noti come, su ogni riga digitata, Debug riporti l'indirizzo Seg:Ofs dell'istruzione, calcolando automaticamente l'ampiezza, in byte, dell'istruzione utilizzata. Il valore della parte segmento dell'indirizzo non è significativo, mentre la parte offset dell'indirizzo è l'esatta posizione in memoria che quell'istruzione avrà nel segmento scelto dal caricatore di MSDOS una volta lanciato il programma.

Per salvare il programma su disco, nella cartella corrente, basta usare il **comando W** (Write) senza argomenti. Debug risponde con la quantità di byte scritti su disco e mostra il prompt.

Ora si può uscire con il **comando Q** (Quit) e lanciare il programma dal prompt di MSDOS.

2.1 Istruzione NOP

Sintassi: **NOP**
Scopo: Non fa nulla. Usata per scopi di servizio, per esempio per allineare una sequenza di istruzioni.
Esempi: `NOP`
Nota: A volte viene inserita NOP se si vuole aumentare di un'unità l'indirizzo del codice in esecuzione. In altri casi per rallentare l'esecuzione.

3 Modificare un programma

Per verificare il programma scritto ed eventualmente modificarlo, si usano i **comandi L** (Load) e **U** (Unassembly) dopo aver specificato il nome del programma da manipolare. Verrà aggiunta una sola istruzione NOP al codice precedente, per mostrare il modo in cui Debug consente di modificare un programma:

`C:\>`	Prompt di MSDOS
`C:\>debug`	Lancio del programma Debug da MSDOS
`-n corto.com`	Assegnazione del nome (corto.com)
`-l`	Caricamento del file da disco presente nella cartella corrente
`-u 100 L2`	Disassemblaggio dall'indirizzo 100h, per 2 byte (dimensione del file)
`0D5C:0100 CD20 INT 20`	Disassemblato, all'indirizzo 100h, Op.Code CD20h e codice mnemonico INT 20
`-a 100`	Modifica dell'istruzione all'indirizzo 100h
`0D5C:0100 nop`	Nuova istruzione (NOP = non fa nulla)
`0D5C:0101 int 20`	Terminazione di file .COM
`0D5C:0103`	Riga vuota: termine dell'edit
`-r cx`	Comando per l'impostazione della dimensione
`CX 0002`	Visualizzazione del contenuto attuale del registro CX da parte di Debug
`:3`	Impostazione della dimensione (100h-103h=3h)
`-w`	Comando per la scrittura del file di programma su disco
`Scrittura di 003 byte in corso`	Visualizzazione della quantità di byte scritti da parte di Debug
`-q`	Uscita da Debug
`C:\>`	Prompt di MSDOS

In alternativa, il file di programma da disassemblare può essere caricato direttamente citandolo come argomento di Debug:

`C:\>`	Prompt di MSDOS
`C:\>debug corto.com`	Lancio del programma Debug da MSDOS con caricamento del file corto.com
`-u 100 L2`	
`0D5C:0100 CD20 INT 20`	
`-a 100`	
`0D5C:0100 nop`	
`0D5C:0101 int 20`	
`0D5C:0103`	
`-r cx`	
`CX 0002`	
`:3`	
`-w`	
`Scrittura di 003 byte in corso`	
`-q`	
`-r cx`	
`CX 0002`	
`C:\>`	

PROGRAMMA

Stampa a video

Un programma che si limita a richiedere l'immissione di un carattere e a stampare la stringa "CIAO" può essere scritto nel seguente modo:

`C:\>`	Prompt di MSDOS
`C:\>debug`	Lancio del programma Debug da MSDOS
`-a 100`	Edit del programma a partire dall'indirizzo 100h
`0CED:0100 mov ah,0`	Sottofunzione 00h di INT 16h, Input di un carattere da tastiera
`0CED:0102 int 16`	Lancio interruzione sw BIOS 16h
`0CED:0104 mov al,43`	Codice Ascii da stampare (43h = 'C')
`0CED:0106 mov ah,0e`	Sottofunzione 0Eh di INT 10h, stampa carattere sullo schermo
`0CED:0108 int 10`	Lancio interruzione sw BIOS 10h
`0CED:010A mov al,49`	Codice Ascii da stampare (49h = 'I')
`0CED:010C mov ah,0e`	Sottofunzione 0Eh di INT 10h, stampa carattere sullo schermo
`0CED:010E int 10`	Lancio interruzione sw BIOS 10h
`0CED:0110 mov al,41`	Codice Ascii da stampare (41h = 'A')
`0CED:0112 mov ah,0e`	Sottofunzione 0Eh di INT 10h, stampa carattere sullo schermo
`0CED:0114 int 10`	Lancio interruzione sw BIOS 10h
`0CED:0116 mov al,4f`	Codice Ascii da stampare (4Fh = 'O')
`0CED:0118 mov ah,0e`	Sottofunzione 0Eh di INT 10h, stampa carattere sullo schermo
`0CED:011A int 10`	Lancio interruzione sw BIOS 10h
`0CED:011C int 20`	Terminazione di file COM
`0CED:011E`	Riga vuota: termine dell'edit
`-n ciao.com`	Assegnazione del nome (ciao.com)
`-r cx`	Comando per l'impostazione della dimensione
`CX 0003`	Visualizzazione del contenuto attuale del registro CX da parte di Debug
`:1e`	Impostazione della dimensione (11Eh -100h=1Eh)
`-w`	Comando per la scrittura del file di programma su disco
`Scrittura di 01e byte in corso`	Visualizzazione della quantità di byte scritti da parte di Debug
`-q`	Uscita da Debug
`C:\>`	Prompt di MSDOS

4 Strutture di controllo

Le principali strutture di controllo utilizzate in Assembly x-86, oltre alla sequenza, sono la condizione (se-allora) e l'iterazione (ripeti n volte), rispettivamente implementate dalle istruzioni di salto condizionato (istruzioni tipo **J*** [**indirizzo**]) e dall'istruzione di ciclo a conteggio (tipo **LOOP*** [**indirizzo**]).

Le istruzioni di salto condizionato, che spostano l'esecuzione all'indirizzo in esse specificato, sono numerose e vanno utilizzate, di norma, subito dopo l'istruzione di confronto **CMP**; tale istruzione, che equivale ad una sottrazione, imposta i flag del registro omonimo in base al risultato della sottrazione, consentendo alle istruzioni di salto condizionato di operare.

In base allo stato dei singoli flag del registro dei Flags, infatti, le varie istruzioni di salto condizionato effettuano il salto all'indirizzo specificato o meno.

4.1 Istruzione CMP

Sintassi: **`CMP operando1, operando2`**

Scopo: Viene eseguita la sottrazione `operando1 - operando2` e impostati i flag opportuni in base all'esito della sottrazione.

Esempi:
```
CMP AL,2
CMP AX, BX
MOV [BX], AL
```

Nota: L'istruzione viene utilizzata in base alle regole generali della sintassi x-86 (cfr. 3.1 Sintassi e indirizzamenti). Naturalmente `operando1` e `operando2` rimangono invariati dopo la CMP.

4.2 Istruzione J*

Sintassi: **`J* indirizzo`**

Scopo: La notazione **J*** significa un intero gruppo di istruzioni analoghe (esempio: JE, JG, JLE, ecc.), tutte con la medesima sintassi. Il flusso dell'esecuzione si sposta su `indirizzo` se le condizioni sui flag previste dalla **J*** sono verificate, altrimenti il flusso dell'esecuzione prosegue regolarmente in sequenza. Queste istruzioni sono dette **salti condizionati**, e si usano spesso dopo l'istruzione CMP per sfruttare le modifiche di stato dei flag.

Esempi:
```
JE 109
JG 110
JB 112
```

Nota: Se l'esito del confronto CMP precedente ha impostato il flag di zero, allora i due operandi di CMP sono uguali e la JE salta all'indirizzo 109h. Analogamente per gli altri due casi, se gli operandi sono, rispettivamente, il primo maggiore del secondo (numeri con segno), il primo minore del secondo (numeri senza segno). *Ricordare che la distanza tra l'indirizzo dell'istruzione di salto e l'indirizzo presso il quale si vuole saltare non deve superare 128.*

Infine vediamo l'istruzione di **salto incondizionato**, fondamentale per dirigere il flusso del codice in modo da saltare alcune parti del programma che per qualche motivo non devono essere eseguite. Spesso il salto incondizionato è usato per realizzare strutture di controllo diverse da quelle standard (esempio: cicli do-while).

4.3 Istruzione JMP

Sintassi: **`JMP indirizzo`**
Scopo: Sposta il flusso dell'esecuzione a indirizzo.
Esempi:
```
JMP 120
JMP CS:120
```
Nota: L'istruzione viene usata per spostare il flusso dell'esecuzione ad un `indirizzo` specifico in modo incondizionato. L'istruzione viene detta salto incondizionato e non ha i limiti di estensione del salto condizionato. Può infatti essere specificato un indirizzo completo Seg:Ofs.

Una tabella di riferimento per consultare velocemente il comportamento delle istruzioni di salto condizionato è la seguente:

Istruzioni di salto	Flags					Operatore equivalente	Descrizione
	Z	C	S	O	P		
JE, JZ	1					=	Salta se uguali
JNE, JNZ	0					≠	Salta se diversi
JA, JNBE	0	0				>	Salta se maggiore, senza segno
JAE, JNB, JNC		0				>=	Salta se maggiore o uguale, senza segno
JB, JC, JNAE		1				<	Salta se minore, senza segno
JBE, JNA	1	1				<=	Salta se minore o uguale, senza segno
JG, JNLE	0		=	=		>	Salta se maggiore, con segno
JGE, JNL			=	=		>=	Salta se maggiore o uguale, con segno
JL, JNGE			≠	≠		<	Salta se minore, con segno
JLE, JNG	1		≠	≠		<=	Salta se minore o uguale, con segno
JNO				0			Salta se non c'è overflow
JNP, JPO					0		Salta se c'è non c'è parità (ovvero c'è parità dispari)
JNS			0				Salta se non c'è segno
JO				1			Salta se c'è overflow
JP, JPE					1		Salta se c'è parità (pari)
JS			1				Salta se c'è segno

PROGRAMMA

Input di un carattere

Ecco come si presenta un codice Debug che attende un tasto, se il tasto è lo zero (0), viene stampato a schermo una zeta maiuscola, altrimenti una enne minuscola:

`C:\>`	Prompt di MSDOS
`C:\>debug`	Lancio del programma Debug da MSDOS
`-a 100`	Edit del programma a partire dall'indirizzo 100h
`0d53:0100 mov ah,00`	Sottofunzione 00h di INT 16h, input di un carattere da tastiera
`0d53:0102 int 16`	Lancio interruzione sw BIOS 16h
`0d53:0104 cmp al,30`	Confronto carattere in input (AL) con carattere zero (30h = '0')
`0d53:0106 je 0110`	Se uguali, salta all'indirizzo 110h ove si stamperà 'Z'
`0d53:0108 mov al,6e`	Altrimenti si stampa il carattere 'n' (6eh = 'n')
`0d53:0100 mov ah,0e`	Sottofunzione 0Eh di INT 10h, Stampa carattere sullo schermo
`0d53:010c int 10`	Lancio interruzione sw BIOS 10h
`0d53:010e jmp 0116`	Salta alla fine del programma
`0d53:0110 mov al,4a`	Si stampa il carattere 'Z' (5ah = 'Z')
`0d53:0112 mov ah,0e`	Sottofunzione 0Eh di INT 10h, stampa carattere sullo schermo
`0d53:0114 int 10`	Lancio interruzione sw BIOS 10h
`0d53:0116 int 20`	Terminazione di file .COM
`0d53:0118`	Riga vuota: termine dell'edit
`-n cmpj.com`	Assegnazione del nome
`-r cx`	Comando per l'impostazione della dimensione
`CX 0003`	Visualizzazione del contenuto attuale del registro CX da parte di Debug
`:18`	Impostazione della dimensione (118h-100h=18h)
`-w`	Comando per la scrittura del file di programma su disco
`Scrittura di 018 byte in corso`	Visualizzazione della quantità di byte scritti da parte di Debug
`-q`	Uscita da Debug
`C:\>`	Prompt di MSDOS

4.4 Istruzione INC

Sintassi: **`INC destinazione`**

Scopo: L'istruzione INC incrementa di un'unità `destinazione`, che può essere un registro o una locazione di memoria.

Esempi:
```
INC  AX
INC  DL
INC  byte ptr [102]
INC  word ptr [102]
```

Nota: Non modifica il registro dei Flags, pertanto se servisse valutarli, si può usare l'equivalente ADD destinazione,1 (cfr. 5.3 Istruzioni aritmetiche, Istruzione ADD). La sua duale è **DEC**, che ha la stessa sintassi e decrementa di un'unità `destinazione`.
Notare che quando destinazione è una cella di memoria, va specificata l'ampiezza della cella da incrementare con le parole chiave **byte ptr** (incrementa solo quella locazione) o **word ptr** (incrementa il valore in memoria a partire da quella locazione e ampio due byte).

4.5 Istruzione LOOP

Sintassi: **`LOOP indirizzo`**

Scopo: All'esecuzione di LOOP la CPU decrementa di un'unità il registro contatore CX; se CX è diverso da zero, il flusso passa all'istruzione posta ad `indirizzo`, altrimenti il flusso dell'esecuzione prosegue regolarmente in sequenza.

Esempi: `LOOP 110`

Nota: Naturalmente l'iterazione automatica di LOOP funziona solo se, prima del blocco da ripetere chiuso da LOOP, si imposta il registro CX con il numero delle iterazioni desiderate. LOOP salta quasi sempre all'indietro, ovvero `indirizzo` è quasi sempre una locazione di memoria precedente a LOOP, ma seguente all'impostazione di CX.

PROGRAMMA

Cicli

Stampare le 26 lettere minuscole dell'alfabeto inglese.

C:\>	Prompt di MSDOS
C:\>debug	Lancio del programma Debug da MSDOS
-a 100	Edit del programma a partire dall'indirizzo 100h
0d53:0100 mov cx,1a	Numero di iterazioni: 26 (1ah = 26)
0d53:0103 mov dl,61	Codice Ascii della a minuscola (61h = 'a')
0d53:0105 mov ah,02	Sottofunzione 02h di INT 21h, stampa carattere sullo schermo
0d53:0107 int 21	Lancio interruzione sw MSDOS 21h
0d53:0109 inc dl	Incrementa di un'unità il codice Ascii
0d53:010b loop 0105	Ripeti CX volte dall'indirizzo 105h, quindi prosegui
0d53:010d int 20	Terminazione di file .COM
0d53:010f	Riga vuota: termine dell'edit
-n alfabe.com	Assegnazione del nome
-r cx	Comando per l'impostazione della dimensione
CX 18	Visualizzazione del contenuto attuale del registro CX da parte di Debug
:f	Impostazione della dimensione (100h-10Fh=Fh)
-w	Comando per la scrittura del file di programma su disco
Scrittura di 00f byte in corso	Visualizzazione della quantità di byte scritti da parte di Debug
-q	Uscita da Debug
C:\>	Prompt di MSDOS

ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1. Elencare le fasi per avviare il programma Debug, quindi i comandi per ispezionare la memoria e i registri, e infine tornare alla shell di MSDOS.
2. Elencare tutti i comandi di Debug per scrivere un programma in Assembly 8086.
3. Scrivere su un foglio le fasi per avviare Debug, scrivere il programma più corto del mondo e verificare la sua presenza su disco.
4. Spiegare come si calcola la lunghezza di un programma COM tramite Debug.
5. Spiegare per quale motivo i programmi COM iniziano a indirizzi con spiazzamento 100h.
6. Fornire degli esempi di nomi per file COM, alcuni corretti e alcuni errati.
7. Spiegare come si termina la fase di scrittura del codice (edit) con Debug.
8. Spiegare come si usa Debug quando si vuole modificare un programma COM già presente su disco.
9. Mostrare con un esempio in Assembly come si verifica se il contenuto di AX è minore del contenuto BX.
10. Mostrare con un esempio una sequenza di istruzioni Assembly tali per cui una istruzione CMP imposti ZF=1.
11. Scrivere su un foglio una sequenza di istruzioni in cui si evidenzi l'uso dell'istruzione JMP.
12. Scrivere su un foglio le istruzioni per stampare dieci volte una zeta maiuscola.
13. Spiegare che istruzioni servono per sapere se un tasto acquisito da tastiera è la lettera P o meno.

Requisiti avanzati

1. Spiegare con quali versioni dei sistemi operativi Microsoft è possibile usare il programma Debug e perché.
2. Spiegare la differenza tra i programmi Command.com e Cmd.com.
3. Riportare l'esatto indirizzo segmentato della routine ISR associata all'interruzione software 10h del BIOS sul proprio sistema.
4. Spiegare come è possibile risalire all'Op.Code d'una qualsiasi istruzione Assembly 8086 tramite Debug.
5. Commentare l'istruzione NOP.
6. Ricordare il limite delle istruzioni di salto condizionato.
7. Spiegare come si può realizzare un ciclo do-while con le istruzioni Assembly dell'8086.
8. Illustrare tutti i meccanismi impliciti dell'istruzione LOOP. Fare un esempio.
9. Mostrare i due modi di Debug per modificare un file COM già presente su disco.
10. Mostrare con un esempio in Assembly come si verifica se il contenuto della cella di indirizzo 103h è maggiore del contenuto nella cella a indirizzo 104h.
11. Scrivere su un foglio le istruzioni per stampare dieci volte una zeta maiuscola senza usare l'istruzione LOOP.
12. Spiegare come un programma COM può stampare il proprio PSP.
13. Descrivere come installare e usare il programma DOSBox su un sistema operativo Win64. Fare una relazione scritta.

ESERCIZI PER LA VERIFICA DI LABORATORIO

I prerequisiti per affrontare questi esercizi sono la disponibilità del programma Debug.exe e la possibilità di consultare una tabella di codici Ascii.

Debug

1 Scrivere un programma Assembly con Debug che stampi sullo schermo la lettera A.

Avviare una shell di MSDOS (esempio, da Windows, Start/Esegui cmd).
Avviare il programma Debug e scrivere il programma, ricordando che il codice Ascii della lettera A, in esadecimale, vale 41h.
Il nome del programma risultante è ***stampaa.com****; qualsiasi nome sarebbe stato corretto, con l'accortezza che non superi gli otto caratteri, che non contenga caratteri speciali (esempio non deve contenere spazi, slash o backslash, ecc.) e che abbia estensione COM.*
La dimensione vale 8h, perché il programma inizia a 100h e termina a 108h.

```
C:\dbg>debug
-a 100
13F1:0100 mov ah,2
13F1:0102 mov dl,41
13F1:0104 int 21
13F1:0106 int 20
13F1:0108
-rcx
CX 0000
:8
-n stampaa.com
-w
Writing 00008 bytes
-q
C:\dbg>
```

OUTPUT

```
C:\dbg>stampaa
A
C:\dbg>
```

2 Scrivere un programma Assembly con Debug che stampi sullo schermo il proprio cognome.

Notare che il cognome stampato è Rossi, i cui codici Ascii esadecimali sono: 52h, 6Fh, 73h, 69h.

```
C:\dbg>debug
-a 100
13F1:0100 mov ah,2
13F1:0102 mov dl,52
13F1:0104 int 21
13F1:0106 mov dl,6f
13F1:0108 int 21
13F1:010A mov dl,73
13F1:010C int 21
13F1:010E mov dl,73
13F1:0110 int 21
13F1:0112 mov dl,69
13F1:0114 int 21
13F1:0116 int 20
13F1:0118
-rcx
CX 0008
:18
-n cognome.com
-w
Writing 00018 bytes
-q
C:\dbg>
```

OUTPUT

```
Rossi
C:\dbg>
```

3 Scrivere un programma Assembly con Debug che stampi sullo schermo i numeri da 0 a 9.

Ricordare che il codice Ascii del simbolo zero (0) vale 30h, e che i successivi sono in sequenza, cioè 31h, 32h, 33h, 34h, 35h, 36h, 37h, 38h, 39h.
Notare che il ciclo deve essere ripetuto dieci volte, che in esadecimale equivale a ah.
I numeri esadecimali che iniziano con una lettera vanno scritti premettendo uno zero (0ah).

```
C:\dbg>debug
-a 100
13F1:0100 mov ah,2
13F1:0102 mov dl,30
13F1:0104 mov cx,0a
13F1:0107 int 21
```

```
13F1:0109 inc dl
13F1:010B loop 107
13F1:010D int 20
13F1:010F
-rcx
CX 0018
:f
-n numeri.com
-w
Writing 0000F bytes
-q
C:\dbg>
```

OUTPUT

```
C:\dbg>numeri
0123456789
C:\dbg>
```

4 **Scrivere un programma Assembly con Debug che si limita ad aspettare la pressione di un tasto qualsiasi.**

```
Layout:
c:\>inp
? a
C:\>
```

Nel testo dell'esercizio è indicato anche il layout, cioè come il programma deve presentarsi a video.

Nel layout sono «nascoste» due specifiche:

1. *Prima di attendere la pressione di un tasto qualsiasi, bisogna stampare un punto interrogativo (3fh) e uno spazio (20h). Di solito ciò che precede un input dell'utente è chiamato* ***prompt.*** *Pertanto i caratteri " ?" sono il prompt di questo programma.*
   Nell'esempio di layout proposto nel testo, l'utente ha premuto una lettera a (minuscola).
2. *Il nome del programma deve essere* ***inp.com***

```
C:\dbg>debug
-a 100
13EB:0100 mov ah,2
13EB:0102 mov dl,3f
13EB:0104 int 21
13EB:0106 mov dl,20
13EB:0108 int 21
13EB:010A mov ah,1
13EB:010C int 21
13EB:010E int 20
13EB:0110
-rcx
CX 0000
:10
-n inp.com
-w
Writing 00010 bytes
-q
C:\dbg>
```

OUTPUT

```
C:\dbg>inp
? a
C:\dbg>
```

5 **Scrivere un programma Assembly con Debug che, preso in input un carattere dall'utente, stampi un punto esclamativo se il carattere è una a (minuscola).**

```
Layout:
c:\>inpa
? a!
C:\>
```

oppure

```
c:\>inpa
? b
C:\>
```

Si noti che, quando è stata scritta l'istruzione a indirizzo 13EB:0110 (jne 100), non si era in grado di sapere a quale indirizzo saltare per terminare il programma (infatti, se in AL non c'è il codice Ascii della lettera a minuscola, il programma deve terminare).
Non sapendo a quale indirizzo saltare, è stato usato un indirizzo «falso» e temporaneo, per esempio 100 (jne 100).
Quindi, dopo aver scritto l'istruzione di terminazione (int 20, all'indirizzo 13EB:0118), si è ritornati a correggere l'istruzione jne 100 in jne 118.
Si ricorda che il codice Ascii della lettera a (minuscola) vale 61h e il codice Ascii del punto esclamativo vale 21h.

```
C:\dbg>debug
-a 100
13EB:0100 mov ah,2
13EB:0102 mov dl,3f
13EB:0104 int 21
13EB:0106 mov dl,20
13EB:0108 int 21
13EB:010A mov ah,1
13EB:010C int 21
13EB:010E cmp al,61
13EB:0110 jne 100
```

```
13EB:0112 mov ah,2
13EB:0114 mov dl,21
13EB:0116 int 21
13EB:0118 int 20
13EB:011A
-a 110
13EB:0110 jne 118
13EB:0112
-rcx
CX 0000
:1a
-n inpa.com
-w
Writing 0001A bytes
-q
C:\dbg>
```

OUTPUT

```
C:\dbg>inpa
? a!
C:\dbg>
```

oppure

```
C:\dbg>inpa
? b
C:\dbg>
```

6 Scrivere un programma Assembly con Debug che stampi sullo schermo il proprio cognome (seconda versione).

In questo caso vengono allocati i codici Ascii del cognome Rossi, cioè 52h, 6Fh, 73h, 73h, 69h, all'inizio del codice Assembly con la parola chiave ***db****.*
È quindi necessario saltare questi dati affinché il microprocessore non li interpreti come istruzioni.
Non si può sapere esattamente a quale indirizzo saltare fino a quando non si sono allocati tutti i dati, quindi il salto iniziale viene scritto con indirizzo temporaneo (jmp 100).
Al termine del programma si corregge questa istruzione sostituendola con l'indirizzo corretto (jmp 107).
Siccome si usa un ciclo, si deve impostare il registro CX con il numero di ripetizioni, ovvero 5 (i cinque caratteri del cognome).
Quindi si carica in BX (uno dei pochi registri assieme a DI e SI che può contenere indirizzi) l'indirizzo del primo dato da stampare, cioè 102.
Infine si legge il dato, ricorsivamente, utilizzando la sintassi delle parentesi quadre sul registro BX, depositandolo nel semiregistro DL affinché sia stampato.
Il ciclo deve ripetersi a partire dall'istruzione ad indirizzo 10d, ovvero l'istruzione mov ah, 2.
Sarebbe stato corretto anche far saltare il ciclo all'indirizzo 10F.

```
C:\dbg>debug
-a 100
13EB:0100 jmp 100
13EB:0102 db 52
13EB:0103 db 6f
13EB:0104 db 73
13EB:0105 db 73
13EB:0106 db 69
13EB:0107 mov cx,5
13EB:010A mov bx,102
13EB:010D mov ah,2
13EB:010F mov dl,[bx]
13EB:0111 int 21
13EB:0113 inc bx
13EB:0114 loop 10d
13EB:0116 int 20
13EB:0118
-a 100
13EB:0100 jmp 107
13EB:0102
-rcx
CX 0000
:18
-n cognome2.com
-w
Writing 00018 bytes
-q
C:\dbg>
```

OUTPUT

```
C:\dbg>cognome2
Rossi
C:\dbg>
```

7 Scrivere un programma Assembly con Debug che stampi sullo schermo una riga di 10 asterischi.

```
Layout:
c:\dbg>as
**********
c:\>
```

8 Scrivere un programma Assembly con Debug che stampi sullo schermo due righe di 10 asterischi.

```
Layout:
c:\dbg>as1
**********
**********
c:\>
```

9 Scrivere un programma Assembly con Debug che, preso in input un carattere, stampi N se numerico, A altrimenti.

```
Layout:
c:\dbg>an
? 1
N
c:\>
```

oppure

```
c:\dbg>an
? z
A
c:\>
```

10 Scrivere un programma Assembly con Debug che, presi in input due caratteri, stampi U se sono uguali.

```
Layout:
c:\dbg>ug
? 1 ? e
c:\>
```

oppure

```
c:\dbg>ug
? a ? a
U
c:\>
```

11 Scrivere un programma Assembly con Debug che, preso in input un carattere, stampi Numerico se numerico, Altro altrimenti.

```
Layout:
c:\dbg>an1
? 1
Numerico
c:\>
```

oppure

```
c:\dbg>an1
? z
Altro
c:\>
```

12 Scrivere un programma Assembly con Debug che, preso in input un carattere numerico, stampi tanti asterischi quanto vale quel numero.

```
Layout:
c:\dbg>as2
? 6******
c:\>
```

(per ottenere il numero dal codice Ascii di un carattere numerico provare l'istruzione SUB:

```
SUB AL,30   ;il risultato rimane AL)
```

13 Scrivere un programma Assembly con Debug che, preso in input un carattere numerico ad esempio n, stampi sullo schermo un «quadrato» di asterischi di lato n.

```
Layout:

c:\dbg>as3
? 6
******
******
******
******
******
******
c:\>
```

(per ottenere il numero dal codice Ascii di un carattere numerico provare l'istruzione SUB:

```
SUB AL,30   ;il risultato rimane AL)
```

14 Scrivere un programma Assembly con Debug che, presi in input due caratteri, stampi U se sono uguali, > se il primo ha codice Ascii maggiore del secondo, < altrimenti.

```
Layout:
c:\dbg>conf
? 1 ? 7
<
c:\>
```

oppure

```
c:\dbg>conf
? a ? a
U
c:\>
```

15 Scrivere un programma Assembly con Debug che, presi in input due caratteri, stampi Uguali se sono uguali, p>s se il primo ha codice Ascii maggiore del secondo, p<s altrimenti.

```
Layout:
c:\dbg>conf1
? 1 ? 7
p<s
c:\>
```

oppure

```
c:\dbg>conf1
? a ? a
Uguali
c:\>
```

16 Scrivere un programma Assembly con Debug che si fermi ad aspettare un input per 3 volte, dopo ogni volta va a capo e alla fine stampa il simbolo dell'ultimo carattere digitato.

```
Layout:
c:\>v3
1?a
2?b
3?c
c
c:\>
```

Per «andare a capo» è sufficiente stampare a schermo e in sequenza i codici Ascii 0ah (LF) e 0dh (CR).

17 Scrivere un programma Assembly con Debug che stampa sullo schermo una riga di asterischi lunga quanto la lunghezza del proprio cognome.

```
Layout:
c:\dbg>lacogn
******
```

18 Scrivere un programma Assembly con Debug che, presi in input due caratteri, stampi a schermo = se i due caratteri sono uguali.

```
Layout:
c:\>ugu
1?h
2?h
=
c:\>
```

19 Scrivere un programma Assembly con Debug che riporti sullo schermo tutti i numeri a una cifra, attenda un tasto, quindi tutte le lettere minuscole, attenda un tasto, quindi tutte le maiuscole.

```
Layout:
c:\>nminmai
0123456789
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
c:\>
```

20 Scrivere un programma Assembly con Debug che acquisito un tasto alfabetico da tastiera stampi il suo maiuscolo se minuscolo o viceversa.

```
Layout:
c:\>minmai
? kK
c:\>
```

oppure

```
c:\>minmai
? Aa
c:\>
```

21 Scrivere un programma Assembly con Debug che acquisito un tasto da tastiera stampi num se numerico, min se minuscolo, mai se maiuscolo, ? altrimenti.

```
Layout:
c:\>minmain
? 1
num
c:\>
```

oppure

```
c:\>minmain
? a
min
c:\>
```

oppure

```
c:\>minmain
? W
mai
c:\>
```

22 Scrivere un programma Assembly con Debug che acquisisca un tasto da tastiera e termini solo quando è numerico.

```
Layout:
c:\>inuma
? 1
c:\>
```

oppure

```
c:\>inuma
? awY u9
c:\>
```

23 Scrivere un programma Assembly con Debug che mostra a video il proprio cognome, prenda un tasto numerico da tastiera quindi stampi tanti asterischi quanto indicato nel valore dell'input.

```
Layout:
c:\>ast
Oppici
? 3
****************************************
**********
c:\>
```

oppure

```
c:\>ast
Oppici
? 3
***
c:\>
```

24 Scrivere un programma Assembly con Debug presi in input due caratteri, stampi C se consecutivi.

```
Layout:
c:\>cnsc
? a
? b
C
c:\>
```

oppure

```
c:\>cnsc
? 1
? 3
c:\>
```

25 Scrivere un programma Assembly con Debug presi in input due caratteri, termini solo quando il secondo è consecutivo al primo.

```
Layout:
c:\>2c1
? a
? b
c:\>
```

oppure

```
c:\>2c1
? a
? 1
? c
? b
c:\>
```

26 Scrivere un programma Assembly con Debug che acquisisca un tasto numerico da tastiera e stampi tutti i numeri inferiori o uguali a quello.

```
Layout:
c:\>serie
? 7 01234567
c:\>
```

27 Scrivere un programma Assembly con Debug che acquisisca un tasto numerico da tastiera e ne stampi il successivo.

```
Layout:
c:\>succ
? 1
2
c:\>
```

28 Scrivere un programma Assembly con Debug che acquisisca un tasto da tastiera e termini solo quando si preme invio.

```
Layout:
c:\>invio
? 8
c:\>
```

oppure

```
c:\>invio
? awY u98
c:\>
```

29 Scrivere un programma Assembly con Debug che acquisisca un tasto numerico da tastiera e ne stampi il successivo solo se il tasto in input non è 9.

```
Layout:
c:\>succ9
? 1
2
c:\>
```

oppure

```
c:\>succ9
? 9
c:\>
```

30 Scrivere un programma Assembly con Debug che acquisisca un tasto da tastiera e stampi voc se il tasto era una vocale minuscola.

```
Layout:
c:\>voc
? x
c:\>
```

oppure

```
c:\>voc
? a voc
c:\>
```

31 Scrivere un programma Assembly con Debug che acquisisca due tasti da tastiera e stampi tutti i caratteri compresi tra i due in input.

```
Layout:
c:\>tra2
? g
```

```
? a
c:\>
```

oppure

```
c:\>tra2
? h
? v
hijklmnopqrstuv
c:\>
```

32 **Scrivere un programma Assembly con Debug che, premuto un tasto, calcoli il logaritmo in base 2 del codice Ascii del tasto premuto.**

```
Layout:
c:\>log2
?h
7
```

A5 Editare un programma

La redirezione dei comandi

Le shell testuali di molti sistemi operativi consentono la **redirezione** dei comandi, ovvero la possibilità che l'output (o l'input) di un comando possa essere inviato su una destinazione diversa da quella prestabilita.

Essendo MSDOS un sistema operativo con shell a carattere, anche in MSDOS è possibile redirezionare i comandi.

Per esempio il comando **dir**, che normalmente invia sullo schermo l'output della lista di file e cartelle può redirezionare la stessa lista su un file di testo:

```
c:\>dir > lista.txt
```

Nella cartella corrente ora esiste il file `lista.txt` il cui contenuto potrebbe essere:

```
il volume nell'unità C è OSDisk
Numero di serie del volume:
7CBE-4B80

Directory di C:\
11/06/2012  14:59 <DIR> documenti
01/07/2013  15:09 <DIR> dosbox
14/07/2009  04:37 <DIR> PerfLogs
26/06/2012  16:42 <DIR> tmp
07/07/2011  14:34 <DIR> Users
06/12/2011  00:40 <DIR> utility
19/06/2012  11:41 <DIR> Windows
10/06/2009  23:42    10 config.sys
06/07/2013  11:15   172 lista.txt
               2 File         10 byte
               7 Directory

69 201 444 864 byte disponibili
```

→

La scrittura di programmi direttamente con Debug è noiosa, soprattutto per quanto riguarda il calcolo degli indirizzi, per esempio, per le istruzioni di salto.

Per rendere più agevole l'uso di Debug, si può utilizzarlo sfruttando i simboli di **redirezione** di MSDOS, ottenendo così la possibilità di poter agire su un codice sorgente scritto in un file di testo. Debug non è *case sensitive*, per cui le istruzioni e i comandi nel file di testo possono essere scritti indifferentemente in minuscolo che in maiuscolo.

La scrittura di un file sorgente per Debug è semplice.

1. Avviare la scrittura di un file di testo (esempio: **alfabe.txt**) tramite un editor di testo, esempio, Notepad: `C:\>notepad alfabe.txt`.
2. Ogni istruzione va riportata su una riga, e la prima riga deve essere sempre il comando di assemblaggio (esempio, a 100h).
3. Dopodiché vanno specificate le righe di codice, magari scrivendo indirizzi temporanei (esempio, usando sempre 100h) laddove non è possibile, in un primo tempo, sapere il valore preciso dell'indirizzo da specificare nel programma (**prima passata**).
4. Il codice va terminato, come al solito, con una riga vuota, quindi si riportano i comandi di Debug soliti per il salvataggio del file eseguibile COM, specificando una dimensione temporanea abbondante (esempio, 100h).

Editando un file per Debug, si può proficuamente dotare il testo di commenti, cioè di righe che non verranno prese in considerazione dal programma Debug, e che contengono indicazioni di aiuto alla comprensione del listato del codice.

I commenti si possono porre su righe che abbiano come primo carattere il punto e virgola (;).

Ricordare di lasciare una riga vuota dopo l'ultima riga di codice e una riga vuota al termine del file.

Il programma dell'esempio precedente (Cicli. Stampare le 26 lettere dell'alfabeto inglese), scritto su file di testo (esempio: **alfabe.txt**), appare nel seguente modo:

```
a 100
mov cx,1a
mov dl,61
mov ah,2
int 21
```

```
inc dl
loop 100          ; 100h = indirizzo temporaneo
int 20

n alfabe.com
rcx
100               ; 100h = dimensione temporanea
w
q
```

5. Per ottenere il file eseguibile (esempio, **alfabe.com**), è ora sufficiente usare la seguente redirezione al prompt di MSDOS:

   `C:\>debug < alfabe.txt > alfabe.lst`

   Se il codice è corretto, su disco viene salvato regolarmente il file eseguibile e un secondo file di testo di tipo listing (esempio, **alfabe.lst**) che ha la seguente forma:

```
-a 100
0CE2:0100 mov cx,1a
0CE2:0103 mov dl,61
0CE2:0105 mov ah,2
0CE2:0107 int 21
0CE2:0109 inc dl
0CE2:010B loop 100
0CE2:010D int 20
0CE2:010F
-n alfabe.com
-rcx CX 0000
:100
-w
Scrittura di 100 byte in corso
-q
```

6. Se nella prima passata si erano riportati indirizzi temporanei, consultando il file di listing si possono agevolmente desumere gli indirizzi reali, nonché la dimensione effettiva del programma. Con una **seconda passata** sul file sorgente e una nuova operazione di redirezione, il programma viene messo a punto definitivamente. Ecco come appare dopo la messa a punto:

```
a 100
mov cx,1a
mov dl,61
mov ah,2
int 21
inc dl
loop 105          ; 105h = indirizzo effettivo
int 20

n alfabe.com
rcx
```

→
La riga di MSDOS:

`C:\>debug < alfabe.txt > alfabe.lst`

Redireziona sia l'input (<) che l'output (>) operando così:

1. il file alfabe.txt viene inviato all'input del programma Debug (<), che viene eseguito;
2. il programma Debug invia l'output (>) nel file di testo alfabe.lst.

Un altro carattere di redirezione (|) è detto **pipe**:

`comando1 | comando2`

In questo caso viene eseguito comando1 e il suo output diventa l'input per l'esecuzione di comando2.

```
f                       ; fh = 10fh-100h, dimensione effettiva
w
q
(riga vuota)
```

7. Infine, si ottiene il file eseguibile definitivo (esempio, **alfabe.com**) riutilizzando di nuovo la redirezione al prompt di MSDOS:

   ```
   C:\>debug < alfabe.txt > alfabe.lst
   ```

   L'intera sequenza per editare, mettere a punto e generare il file eseguibile COM a partire da un file di testo TXT è la seguente:

```
C:\>
C:\>notepad alfabe.txt                ; creazione del file di testo contenente il programma alfabe.txt
C:\>debug < alfabe.txt > alfabe.lst   ; prima passata: creazione del file di listing alfabe.lst da consultare
C:\>notepad alfabe.lst                ; apertura del file di listing per consentire la messa a punto
C:\>notepad alfabe.txt                ; apertura del file di testo per consentirne la messa a punto
C:\>debug < alfabe.txt > alfabe.lst   ; seconda passata: creazione del file eseguibile alfabe.com
C:\>alfabe                            ; esecuzione del programma alfabe.com
Abcdefghijklmnopqrstuvwxyz            ; output del programma
C:\>
```

1 Area Dati e area Codice

Naturalmente le API di MSDOS forniscono gli strumenti per memorizzare dati singoli (variabili), array di caratteri (stringhe), array di numeri e l'I/O di stringhe sullo schermo e dalla tastiera.

L'allocazione di dati in memoria avviene tramite una **pseudoistruzione**, ovvero un'indicazione contenuta nel codice affinché il dato da allocare sia semplicemente posto in memoria, per distinguerlo dall' Op. Code di un'istruzione.

Le pseudoistruzioni non generano alcun codice macchina, ma servono solo per indicare come deve comportarsi un traduttore (nel nostro caso Debug). Le pseudoistruzioni per allocare dati in memoria sono:

1.1 Pseudoistruzione D*

Sintassi: **D* dato**

Scopo: Alloca dato all'indirizzo corrente.

Esempi:

```
DB  0
DB  41h
DB  'A'
DB  ?
DB  10 DUP (0)
```

Nota: Nel primo caso viene allocato un byte in memoria (0), nel secondo un byte che rappresenta il codice Ascii della A maiuscola, nel terzo caso un modo equivalente al secondo e nel terzo si allocano due byte contigui.

Nel caso DB ? si alloca un byte senza inizializzarlo (il valore in quella cella sarà casuale), mentre con DB 10 DUP (0) si indica l'allocazione di 10 byte tutti inizializzati a 0.

Accedere alla memoria

Per leggere e scrivere valori in area Dati ma in generale per accedere alla memoria di un programma x-86 bisogna usare la sintassi con le **parentesi quadre**. Esempio: per leggere il primo byte del PSP per un file .COM si usa l'istruzione:

```
MOV DL,[0]
```

All'interno delle parentesi quadre si specifica un indirizzo di memoria e le parentesi quadre significano «la cella che si trova a quell'indirizzo». →

È disponibile anche DW (alloca due byte) DD (4 byte) D (8 byte) e DT (10 byte), queste ultime spesso usate per memorizzare numeri in virgola fissa per i calcoli con il coprocessore matematico.

Ovviamente l'area di memoria destinata a contenere i dati (variabili e array) non deve essere eseguita come codice. Essa è detta **area Dati**, e deve essere separata dall'**area Codice**, che contiene le istruzioni da eseguire.

Nel modello COM, area Dati e area Codice risiedono nello stesso segmento, cioè in locazioni di memoria contigue. Se, come spesso si opta, l'area Dati viene posta all'inizio della zona della memoria del programma, è necessario porre un'istruzione iniziale di salto incondizionato per evitare l'area Dati e avviare correttamente l'area Codice.

In altri casi si può optare allocando l'area Dati immediatamente dopo l'area di Codice.

→
Questa operazione è detta **dereferenziazione**, del tutto analoga a quella ottenuta in linguaggio C con l'operatore * agente su un puntatore.
Analogamente, per scrivere in una locazione di memoria (p.es. modificare il primo byte del PSP):

```
MOV [0],65
```

PROGRAMMA

Area Dati e area Codice

Allocare i tre codici Ascii della parola HAL e stamparli sullo schermo.

File di testo **hal.txt**

```
a 100
jmp 105          ; Salto dell'area Dati per raggiungere l'area del Codice che si trova ad indirizzo 105h (mov bx, 102)
db 'H'           ; Area Dati. Si allocano i tre byte dei codici Ascii della stringa 'HAL'. Il primo si trova ad indirizzo 102h
db 'A'
db 'L'
mov bx,102       ; Area Codice, siamo a indirizzo 105h. In BX l'indirizzo del primo byte dell'area Dati
mov cx,3         ; Contatore del ciclo a 3 (3 caratteri da stampare)
mov dl,[bx]      ; Indirizzamento indiretto. In DL il codice Ascii che si trova in area Dati all'indirizzo specificato in BX
mov ah,2         ; Sottofunzione 02h di MSDOS, stampa carattere
int 21           ; Interruzione sw MSDOS
inc bx           ; Incremento indirizzo in area Dati: La prossima cella contiene il prossimo codice Ascii da stampare
loop 10b         ; Iterazione a partire dall'istruzione mov dl,[bx], che si trova ad indirizzo 10bh
int 20

n hal.com
rcx
16
w
q
```

La compilazione, la messa a punto e l'esecuzione:

```
C:\>
C:\>notepad hal.txt                ; creazione del file di testo contenente il programma hal.txt
C:\>debug < hal.txt > hal.lst      ; prima passata: creazione del file di listing hal.lst da consultare
C:\>notepad hal.lst                ; apertura del file di listing per consentire la messa a punto
C:\>notepad hal.txt                ; apertura del file di testo per consentirne la messa a punto
C:\>debug < hal.txt > hal.lst      ; seconda passata: creazione del file eseguibile hal.com
C:\>hal                            ; esecuzione del programma hal.com
HAL                                ; output del programma
C:\>
```

Le tre locazioni di memoria così allocate possono ospitare a tutti gli effetti delle variabili. Infatti in quelle locazioni i dati possono essere cambiati a runtime per memorizzare altri valori. Nell'esempio, le tre locazioni vengono manipolate, aggiungendo un'unità ad ogni cella, ottenendo i tre codici Ascii della stringa 'IBM', che poi verrà stampata a schermo.

PROGRAMMA

Variabili

Allocare in un buffer la parola HAL, quindi aggiungere un'unità ad ogni codice Ascii e stampare il buffer risultante.

File di testo **ibm.txt**

```
a 100
jmp 105          ; Salto dell'area Dati per raggiungere l'area del Codice che si trova ad indirizzo 105h (mov bx, 102)
db 'H'           ; Area Dati. Si allocano i tre byte dei codici Ascii della stringa 'HAL'. Il primo si trova ad indirizzo 102h
db 'A'
db 'L'
mov bx,102       ; Area Codice. In BX l'indirizzo del primo byte dell'area Dati
mov cx,3         ; contatore a 3 (3 locazioni di memoria da manipolare)
mov al,[bx]      ; lettura della variabile (locazione di memoria puntata da BX)
inc al           ; incremento di un'unità
mov [bx],al      ; salvataggio della variabile (nella locazione di memoria puntata da BX)
inc bx           ; prossima variabile in memoria
loop 10b         ; iterazione. 10bh è l'indirizzo dell'istruzione mov al,[bx]
mov bx,102       ; indirizzo della prima locazione di memoria, per stampare a schermo
mov cx,3         ; contatore a 3 (3 stampe da effettuare)
mov dl,[bx]
mov ah,2
int 21
inc bx
loop 11a         ; iterazione. 11ah è l'indirizzo dell'istruzione mov dl,[bx]
int 20

n ibm.com
rcx
25
w
q
```

La compilazione, la messa a punto e l'esecuzione:

```
C:\>
C:\>notepad ibm.txt                  ; creazione del file di testo contenente il programma ibm.txt
C:\>debug < ibm.txt > ibm.lst        ; prima passata: creazione del file di listing ibm.lst da consultare
C:\>notepad ibm.lst                  ; apertura del file di listing per consentire la messa a punto
C:\>notepad ibm.txt                  ; apertura del file di testo per consentirne la messa a punto
C:\>debug < ibm.txt > ibm.lst        ; seconda passata: creazione del file eseguibile ibm.com
C:\>ibm                              ; esecuzione del programma ibm.com
IBM                                  ; output del programma
C:\>
```

1.2 Allineamento

Quando si alloca l'area Dati prima dell'area Codice, non sempre la prima istruzione di codice può essere collocata immediatamente dopo l'ultimo byte dell'area Dati. Infatti il microprocessore interpreta i dati come codice

e raggruppa i byte dei dati in base alle combinazioni di Op.Code che essi formano 'inconsapevolmente'.

Per evitare questa situazione, che impedisce l'avvio del codice, bisogna allineare i dati, ovvero aggiungere in coda all'area Dati alcuni byte che consentano alla CPU di interpretare correttamente la prima istruzione di codice.

Si veda questa situazione nel codice che segue, che effettua l'input di tre caratteri, proponendo un prompt a forma di punto interrogativo, e indica a video il maggiore con la stringa 'il max vale:'.

In una prima versione, i dati della stringa causano il disallineamento del codice (riga azzurra):

```
C:\>                                                  Prompt di MSDOS
C:\>debug m3.com                                      Debug carica il file m3.com
-u 100                                                Edit del programma a partire dall'indirizzo 100h
0D54:0100   EB0A        JMP     010E                  JMP 10E
0D54:0102   49          DEC     CX                    49 ('i')
0D54:0103   6C          DB      6C                    6C ('l')
0D54:0104   206D61      AND     [DI+61],CH            20 (' ') 6D ('m') 61 ('a')
0D54:0107   7820        JS      0129                  78 ('x') 20 (' ')
0D54:0109   7661        JBE     016C                  76 ('v') 61 ('a')
0D54:010B   6C          DB      6C                    6C ('l')
0D54:010C   65          DB      65                    65 ('e')
0D54:010D   3AB402B2    CMP     DH,[SI+B202]          3A (':') B402 (=MOV AH,02) B2
0D54:0111   3F          AAS                           3F (=MOV AL,3F)
0D54:0112   CD21        INT     21                    CD21 INT 21
0D54:0114   B401        MOV     AH,01                 B401 MOV AH,01
0D54:0116   CD21        INT     21                    CD21 INT 21
```

Come si può notare, la sequenza di byte allocati in memoria che rappresentano la stringa 'il max vale:', non vengono correttamente allineati.

Infatti all'indirizzo 0D54:010D il byte 3A (il codice Ascii dei 'due punti') viene compattato nell'Op.Code dell'istruzione CMP DH,[SI+B202], inglobando la prima effettiva istruzione di codice (MOV AH,02) e facendo fallire il salto iniziale ad essa (JMP 10E).

Aggiungendo un solo byte al termine dell'area Dati, con la pseudoistruzione DB 0, il codice viene riallineato e il salto incondizionato deve essere aggiornato con un'unità in più:

```
C:\>                                                  Prompt di MSDOS
C:\>debug m3.com                                      Debug carica il file m3.com
-u 100                                                Edit del programma a partire dall'indirizzo 100h
0D54:0100   EB0A        JMP     010F                  JMP 10F
0D54:0102   49          DEC     CX                    49 ('i')
0D54:0103   6C          DB      6C                    6C ('l')
0D54:0104   206D61      AND     [DI+61],CH            20 (' ') 6D ('m') 61 ('a')
0D54:0107   7820        JS      0129                  78 ('x') 20 (' ')
0D54:0109   7661        JBE     016C                  76 ('v') 61 ('a')
0D54:010B   6C          DB      6C                    6C ('l')
0D54:010C   65          DB      65                    65 ('e')
0D54:010D   3A00        CMP     AL,[BX+SI]            3A (':') 00 (per allineare)
0D54:010F   B402        MOV     AH,02                 B402 MOV AH,02
0D54:0111   B23F        MOV     DL,3F                 B23F MOV AL,3F
0D54:0113   CD21        INT     21                    CD21 INT 21
0D54:0115   B401        MOV     AH,01                 B401 MOV AH,01
0D54:0117   CD21        INT     21                    CD21 INT 21
```

Il programma completo è il seguente:

PROGRAMMA

Variabile allocata

Scrivere un programma Assembly con Debug e file sorgente che, presi in input tre caratteri numerici, indichi il maggiore.

Il layout:

```
C:\>m3
?1?5?3Il max vale:5
C:\>
```

Ecco il codice sorgente nel file di testo m3.txt, già con gli indirizzi effettivi desunti dal file di listing corrispondente. Si noti come è possibile allocare una stringa con la sintassi tra apici, senza dover allocare ogni singolo codice Ascii numerico.

```
a 100
JMP 10f                 ; Si salta l'area Dati per arrivare all'istruzione MOV AH,02
DB 'Il max vale:'       ; area Dati
db 0                    ; byte di allineamento. Usato anche come variabile per contenere il maggiore
MOV AH,02               ; sottofunzione per la stampa di un carattere
MOV DL,3f               ; codice Ascii del punto interrogativo (il prompt prima dell'input)
INT 21                  ; Interruzione sw MSDOS stampa un carattere
MOV AH,01               ; sottofunzione per l'input di un carattere da tastiera
INT 21                  ; Interruzione sw MSDOS
MOV BX,10e              ; in BX l'indirizzo del byte di memoria che fungerà da variabile (il byte di allineamento)
MOV [BX],AL             ; memorizzazione dell'input nella variabile
MOV AH,02               ; secondo prompt per il secondo input
MOV DL,3f
INT 21
MOV AH,01               ; secondo input
INT 21
CMP AL,[BX]             ; confronto tra l'input e la variabile in memoria
JL 12e                  ; se inferiore, non memorizzo (salto all'istruzione MOV AH,02)
MOV [BX],AL             ; altrimenti salvo nella variabile il nuovo massimo
MOV AH,02               ; terzo prompt per il terzo input
MOV DL,3f
INT 21
MOV AH,01               ; terzo input
INT 21
CMP AL,[BX]             ; confronto tra l'input e la variabile in memoria
JL 13e                  ; se inferiore, non memorizzo (salto all'istruzione MOV CX,0d)
MOV [BX],AL             ; altrimenti salvo nella variabile il nuovo massimo
MOV CX,0d               ; contatore a 13 (0dh=13) per 12 caratteri della stringa da stampare più il byte della variabile
                        ; che contiene il max
MOV AH,02               ; sottofunzione di stampa carattere
MOV BX,102              ; indirizzo iniziale dei caratteri da stampare
MOV DL,[BX]             ; in DL il carattere da stampare, prelevato dalla memoria
INT 21                  ; interruzione MSDOS
INC BX                  ; prossimo indirizzo per il prossimo carattere
LOOP 146                ; iterazione dalla istruzione MOV AH,02
INT 20

n m3.COM
rcx
4f
w
q
```

2 Input e output di stringhe

Spesso conviene rappresentare in memoria le stringhe in buffer di formato **AsciiZ** (*Ascii Zero terminated*) dato che si tratta della rappresentazione usata da numerosi ambienti e linguaggi, compreso il linguaggio C.

In questo caso sia l'input sia l'output di stringhe avviene tramite cicli che controllano se il byte analizzato è l'ultimo previsto, che deve avere valore zero (0 o **NULL**, come spesso indicato in linguaggio C).

Per acquisire in input una stringa il programma verificherà se l'utente ha premuto il carattere di '**Acapo**' (CR = 0dh) e quindi scriverà il byte terminatore NULL (0) nel buffer di memorizzazione.

PROGRAMMA

Output stringhe AsciiZ

Scrivere un programma Assembly con Debug e file sorgente che, preso in input un numero (da 0 a 9) stampi <>Zero se il numero non è zero.

Il layout:

```
C:\>nzero
Input numero: 1
<>Zero
C:\>
```

oppure

```
C:\>nzero
Input numero: 0
C:\>
```

Nello svolgimento vengono riportate entrambe le passate, con i file di listing corrispondenti.

Si noti l'uso dei commenti, l'uso dei caratteri speciali **CR** (0dh) e **LF** (0ah) per rispettare il layout richiesto andando a capo correttamente e la gestione di due stringhe AsciiZ terminate con zero (0).

Prima passata, con indirizzi temporanei tutti uguali a 100h:

file nzero.txt

```
a 100
; salto i dati
jmp 100
; messaggi a video
db 'Input numero: ',0
db 0d,0a,'<>Zero',0
; Stampo a video la descrizione
; ..................................
;
mov bx,100
mov ah,2
mov dl, [bx]
cmp dl,0
je 100
int 21
inc bx
jmp 100
; Input del numero
; ..................................
;
mov ah,1
int 21
; Confronto
;...................................
cmp al,30
je 100 ; salto alla fine...
; ...altrimenti stampo messaggio
; ..................................
;
mov bx,100
mov ah,2
mov dl, [bx]
```

File nzero.lst

```
-a 100
13F1:0100 ; salto i dati
13F1:0100 jmp 100
13F1:0102 ; messaggi a video
13F1:0102 db 'Input numero: ',0
13F1:0111 db 0d,0a,'<>Zero',0
13F1:011A ; Stampo a video la descrizione
13F1:011A ;..............................
13F1:011A ;
13F1:011A mov bx,100
13F1:011D mov ah,2
13F1:011F mov dl, [bx]
13F1:0121 cmp dl,0
13F1:0124 je 100
13F1:0126 int 21
13F1:0128 inc bx
13F1:0129 jmp 100
13F1:012B ; Input del numero
13F1:012B ; ..............................
13F1:012B ;
13F1:012B mov ah,1
13F1:012D int 21
13F1:012F ; Confronto
13F1:012F ;...............................
13F1:012F cmp al,30
13F1:0131 je 100 ; salto alla fine...
13F1:0133 ; ...altrimenti stampo messaggio
13F1:0133 ; ..............................
13F1:0133 ;
13F1:0133 mov bx,100
13F1:0136 mov ah,2
13F1:0138 mov dl, [bx]
```

```
cmp dl,0                                13F1:013A cmp dl,0
je 100                                  13F1:013D je 100
int 21                                  13F1:013F int 21
inc bx                                  13F1:0141 inc bx
jmp 100                                 13F1:0142 jmp 100
; Fine programma                        13F1:0144 ; Fine programma
; ..............................        13F1:0144 ; ..........................
int 20                                  13F1:0144 int 20
                                        13F1:0146
Rcx                                     -rcx
100                                     CX 0100
n nzero.com                             -n nzero.com
w                                       -w
q                                       Writing 00100 bytes
                                        -q
```

Seconda passata, con indirizzi effettivi desunti dal file di listing precedente (in evidenza):

```
            file nzero.txt
a 100
; salto i dati
jmp 11A
; messaggi a video
db 'Input numero: ',0                   ; Ecco una stringa AsciiZ, terminata con zero
db 0d,0a,'<>Zero',0                     ; Anteponendo CR e LF otterremo la stampa su una nuova riga.
; Stampo a video la descrizione
; ..............................
;
mov bx,102                              ; stampa della stringa del prompt…
mov ah,0032
mov dl, [bx]
cmp dl,0                                ; … fino al terminatore 0 della stringa AsciiZ
je 12b
int 21
inc bx
jmp 11d
; Input del numero
; ..............................
;
mov ah,1
int 21
; Confronto
;..............................
cmp al,30
je 144 ; salto alla fine...
; ...altrimenti stampo messaggio
; ..............................
;
mov bx,111
mov ah,2
mov dl, [bx]
cmp dl,0
je 144
int 21
inc bx
jmp 136
; Fine programma
; ..............................
int 20

rcx
46
n nzero.com
w
q
```

PROGRAMMA

Input stringhe AsciiZ

Scrivere un programma Assembly con Debug e file sorgente che, preso in input il proprio cognome lo stampi a video.

Il layout:

```
C:\>icogn
Inserire cognome: Rossi
Rossi
C:\>
```

Nello svolgimento risultano già gli indirizzi effettivi e la dimensione effettiva desunti dalla prima passata consultando il file di listing corrispondente.

file icogn.txt

```
a 100
jmp 129
;Area dei dati
;.....................................
db 'Inserire cognome: ',0                    ; Ecco la stringa AsciiZ del prompt di input
db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0   ; Buffer adeguato per contenere la stringa in input. Sarà AsciiZ

; stampa stringa prompt di input a schermo
;.....................................
mov bx,102                                   ; 102 è l'indirizzo della stringa del prompt di input
mov ah,2                                     ; stampa della stringa del prompt…
mov dl,[bx]
cmp dl,0
je 13A                                       ; … fino al terminatore 0 della stringa AsciiZ
int 21
inc bx
jmp 12E
;Input stringa
;.....................................
mov bx,115                                   ; 115 è l'indirizzo del Buffer per contenere la stringa di input
mov ah,1                                     ; input di un carattere (sarà memorizzato in AL)
int 21
cmp al,0d                                    ; il carattere digitato è CR (Enter)? …
je 14A                                       ; ... sì, allora l'input è terminato
mov [bx],al                                  ; … no, allora proseguo l'input
inc bx                                       ; … incrementando l'indirizzo di memorizzazione nel Buffer
jmp 13D                                      ; … e tornando a acquisire un carattere
;A capo
;.....................................
mov ah,2                                     ; Nuova riga, per rispettare il layout richiesto
mov dl,0d
int 21
mov dl,0a
int 21
;stampa stringa di input
;.....................................
mov bx,115                                   ; stampa del Buffer di input
mov ah,2
mov dl,[bx]
cmp dl,0
je 165
int 21
inc bx
jmp 157
;Fine programma
;.....................................
int 20

n icogn.com
rcx
67
w
q
```

3 Istruzioni aritmetiche

Una stringa numerica (decimale) è una stringa i cui codici Ascii sono compresi tra 48 (30h) e 57 (39h), pertanto per verificare se una stringa è numerica è sufficiente controllarne ogni codice Ascii e verificare che sia compreso entro questi due limiti.

D'altra parte se si deve stampare un numero decimale ad una singola cifra sullo schermo, è sufficiente trovarne il codice Ascii aggiungendo 48 (30h).

Vediamo quindi le quattro istruzioni per il calcolo aritmetico.

3.1 Istruzione ADD

Sintassi: **`ADD sorgente, destinazione`**

Scopo: Effettua la somma (anche con segno) tra `sorgente` e `destinazione`. Il risultato viene collocato in `sorgente`. `destinazione` non può essere un immediato, ma può essere una zona di memoria.

Esempi:
```
ADD BX, 256
ADD BX, CX
ADD [102], CX
ADD byte ptr [BX], 1
```

Nota: Come per INC/DEC, se agisce su una zona di memoria, va precisata la dimensione del sorgente con le parole chiave **byte ptr** o **word ptr**.

3.2 Istruzione SUB

Sintassi: **`SUB minuendo, sottraendo`**

Scopo: Sottrae da minuendo il sottraendo (anche con segno). Il risultato viene collocato in `minuendo`. `minuendo` non può essere un immediato, ma può essere una zona di memoria.

Esempi:
```
SUB BX, 256
SUB BX, CX
SUB [102], CL
SUB word ptr [BX], 1
```

Nota: Vedi ADD.

3.3 Istruzione MUL

Sintassi: **`MUL moltiplicatore`**

Scopo: Effettua la moltiplicazione senza segno tra: AL e `moltiplicatore`, se `moltiplicatore` è a 8 bit, oppure tra AX e `moltiplicatore`, se `moltiplicatore` è a 16 bit. Nel primo caso colloca in AX il risultato, nel secondo caso in DX:AX. `moltiplicatore` non può essere un immediato, ma può essere una cella di memoria.

Esempi: `MUL CH`
`MUL [102]`
`MUL AX`

Nota: Se il risultato è maggiore del contenitore, saranno impostati i flag di Overflow o di Carry, altrimenti azzerati.

3.4 Istruzione DIV

Sintassi: **`DIV divisore`**

Scopo: Effettua la divisione senza segno tra: AX e `divisore`, se `divisore` è a 8 bit, oppure tra DX:AX e `divisore`, se `divisore` è a 16 bit. Nel primo caso colloca in AL il quoziente e in AH il resto, nel secondo caso in AX il quoziente e in DX il resto. `divisore` non può essere un immediato, ma può essere una cella di memoria.

Esempi: `DIV BL`
`DIV [102]`
`DIV AX`

Nota: Se il quoziente non sta nel contenitore, avviene un errore di overflow o di divisione per zero. Esempio, MOV AX,0A100; MOV BL,2; DIV BL; genera un errore perché A100h / 2 = 5080h, che non sta in un byte.

PROGRAMMA

Verifica se una stringa è numerica

Effettuare l'input di una stringa e, se numerica, stamparne la lunghezza.

Layout:

```
C:\>snum
Inserire stringa: 123
numerica: 3
C:\>
```

oppure

```
C:\>snum
Inserire stringa: ciao
C:\>
```

Nel codice proposto si verifica se ogni singolo codice Ascii della stringa in input rispetta la condizione (compreso tra 30h e 39h) e si salva la lunghezza della stringa in area dati, calcolandola come differenza tra indirizzo finale e iniziale.

Per stampare la lunghezza, supposta su una cifra, si aggiunge 30h per ottenere il codice Ascii equivalente.

file snum.txt

```
a 100
jmp 12c
;Area dei dati
;.....................................
db 'Inserire stringa: ',0                ; stringa prompt di input
db 'numerica: ',0                        ; stringa risultato
db 0,0,0,0,0,0,0,0,0,0,0                 ; Buffer per la stringa in input
db 0                                     ; Locazione che conterrà la lunghezza della stringa numerica

; stampa stringa prompt di input a schermo
;.....................................
mov bx,102                               ; 102h è l'indirizzo della stringa del prompt
mov ah,2
mov dl,[bx]
cmp dl,0
je 13D
int 21
```

PROGRAMMA

```
inc bx
jmp 12F
;Input stringa
;.....................................
mov bx,120                              ; 120h è l'indirizzo del Buffer di input
mov ah,1
int 21
cmp al,0d
je 14d
mov [bx],al
inc bx
jmp 140
;Verifica se numerica
;.....................................
mov bx,120
mov dl,[bx]
cmp dl,0                                ; Abbiamo finito?
je 164                                  ; sì, allora andiamo a stampare
cmp dl,30                               ; il codice Ascii è inferiore a 30h?
jl 194                                  ; sì, allora termina programma
cmp dl,39                               ; il codice Ascii è inferiore a 39h?
jg 194                                  ; sì, allora termina programma
inc bx                                  ; prossimo carattere
jmp 150
; Salvo la lunghezza
; .....................................
mov si,12b                              ; 12bh è l'indirizzo in cui memorizzare la lunghezza
sub bx,120                              ; Il Buffer iniziava a indirizzo 120h e termina al valore di BX
mov dl,bl
add dl,30                               ; aggiungo 30h per ottenere il codice Ascii equivalente
mov [si],dl                             ; salvo in memoria
;A capo
;.....................................
mov ah,2
mov dl,0d
int 21
mov dl,0a
int 21
;stampa stringa risultato
;.....................................
mov bx,115
mov ah,2
mov dl,[bx]
cmp dl,0
je 18d
int 21
inc bx
jmp 17f
;stampa lunghezza
;.....................................
mov bx,12b                              ; recupero la lunghezza e la stampo
mov dl,[bx]
int 21
;Fine programma
;.....................................
int 20

n snum.com
rcx
96
w
q
```

4 Operatori orientati ai bit

Tutte le ISA possiedono operatori in grado di manipolare direttamente i bit di un valore numerico (**operatori bitwise**). Tipicamente si tratta di operatori di confronto di tipo booleano come OR, AND e NOT o di operatori di scorrimento tipo SHIFT.

L'operatore bitwise più semplice è la negazione **NOT** che, agendo su un solo operando, gli inverte tutti i bit (quelli a 0 in 1 e viceversa).

4.1 Istruzione NOT

Sintassi: **`NOT destinazione`**
Scopo: Inverte tutti i bit di `destinazione` (gli 1 in 0, gli 0 in 1). Agisce sia su byte sia su word, a seconda della dimensione di `destinazione`.
Esempi:
```
NOT AL
NOT BX
```
Nota: Non modifica alcun flag nel registro Flags.

ESEMPIO

NOT

Dato il valore 23, calcolare NOT 23.

Prima di tutto si trasforma in binario il valore dato:

`(23)d = (10111)b`

quindi si invertono tutti i bit:

`NOT(23)d = NOT(10111)b = (01000)b = (1000)b.`

La stessa operazione, ma riferita al byte (8bit):

`NOT(23)d = NOT(10111)b = (00010111)b = (11101000)b`

Un'applicazione tipica degli operatori logici è, dato un valore, impostare a 0 o a 1 un bit di peso dato; oppure verificare, dato un valore, se un bit di peso dato vale 0 o 1.

Per impostare un bit di peso dato a 1 in un valore assegnato, si opera così (se il valore è val e il peso n):

- Si calcola 2^n.
- Si trasformano val e 2^n in binario.
- Si calcola val OR 2^n.

4.2 Istruzione OR

Sintassi: **`OR destinazione, sorgente`**
Scopo: Calcola l'OR logico tra `destinazione` e `sorgente`, depositando il valore in `destinazione`.
Esempi:
```
OR AL,3
OR BX, AX
```

Operatori booleani e logici

Gli **operatori booleani** definiscono operazioni all'interno dell'**algebra di Boole** (1854), una struttura algebrica che comprende due soli valori.

Gli operatori dell'algebra booleana possono essere rappresentati in vari modi a seconda dei contesti in cui sono usati. In informatica sono usati proficuamente per agire sui valori 1 e 0 intesi come bit (invece in Elettronica digitale i due valori sono in genere tensioni, nulla e +5V).

Quando gli operatori booleani sono usati in contesto informatico si dicono **operatori logici** (mentre in Elettronica digitale sono detti **porte logiche**).

Gli operatori logici possono essere esemplificati tramite tabelle che ne descrivono il comportamento:

A	NOT A	
0	1	
1	0	
A	**B**	**A OR B**
0	0	0
0	1	1
1	0	1
1	1	1
A	**B**	**A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

Nota: `destinazione` e `sorgente` devono essere concordi, cioè avere la stessa dimensione (byte o word). Vengono modificati opportunamente OF e CF nel registro Flags.

ESEMPIO

OR: impostare un bit a 1

Dato il valore 38, impostare a 1 il bit di peso 3.

Si calcola 2^n:

$2^3 = 8$

Si trasformano i dati in binario:

```
(38)d = (100110)b
 (8)d = (1000)b
```

Quindi, tramite l'algebra booleana, si calcola val OR 2^n:

```
100110 OR
  1000 =
101110
```

In definitiva:

```
38 OR 8 = (100110)b OR (1000)b = (101110)b = (46)d = 46
```

In Assembly:

```
MOV AL,26          ; 38d = 26h
OR AL,8            ; ora in AL c'è 46d (2Eh)
```

Per impostare un bit di peso dato a 0 in un valore assegnato, si opera così (se il valore è val e il peso n):

- Si calcola 2^n.
- Si trasformano val e 2^n in binario; 2^n va riempito alla dimensione di val.
- Si calcola val AND (NOT 2^n).

4.3 Istruzione AND

Sintassi: **`AND destinazione, sorgente`**
Scopo: Calcola l'AND logico tra `destinazione` e `sorgente`, depositando il valore in `destinazione`.
Esempi:
```
AND AL,3
AND BX, AX
```
Nota: `destinazione` e `sorgente` devono essere concordi, cioè avere la stessa dimensione (byte o word). Vengono modificati opportunamente OF e CF nel registro Flags.

ESEMPIO

AND: impostare un bit a 0

Dato il valore 38, impostare a 0 il bit di peso 2.

Si calcola 2^n:

$2^2 = 4$

Si trasformano i dati in binario:

```
(38)d = (100110)b
 (4)d = (100)b
```

ESEMPIO

Quindi, tramite l'algebra booleana, si calcola NOT 2^n:

`NOT (4)d = NOT (100)b = NOT (000100)b = (111011)b` (*attenzione al riempimento*)

E quindi si calcola val AND (NOT 2^n):

```
100110 AND
111011 =
100010
```

In definitiva:

`38 AND (NOT 8) = (100110)b AND (111011)b = (100010)b = (34)d = 34`

In Assembly:

```
MOV AL,26          ; 38d = 26h
MOV AH,8
NOT AH
AND AL,AH          ; ora in AL c'è 34d (22h)
```

Per verificare se in un valore assegnato un determinato bit di peso dato vale 0 o 1 si opera così (se il valore è val e il peso n):

- Si calcola 2^n.
- Si trasformano val e 2^n in binario; 2^n va riempito alla dimensione di val.
- Si calcola val AND 2^n: se il risultato è 0, il bit vale 0, altrimenti il bit vale 1.

ESEMPIO

Verificare il valore di un bit

Dato il valore 38, verificare il valore del bit di peso 4.

Si calcola 2^n:

$2^4 = 16$

Si trasformano i dati in binario (*attenzione al riempimento*):

```
(38)d = (100110)b
(16)d = (10000)b = (010000)b
```

Quindi, tramite l'algebra booleana, si calcola val AND 2^n:

```
100110 AND
100000 =
100000
```

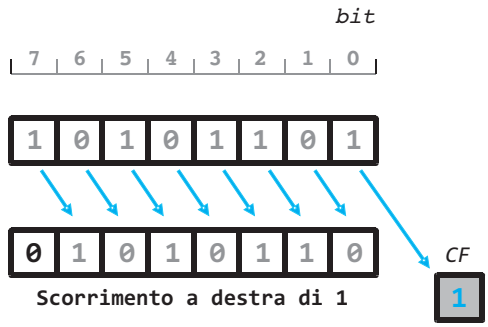
Siccome `(38)d AND (16)d = (100110)b AND (100000)b = (100000)b = 16` è diverso da zero, il bit di peso 4 in 38 vale 1

In Assembly:

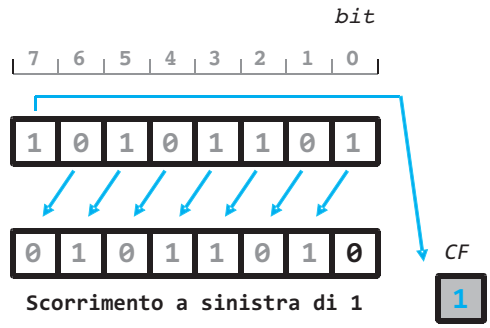
```
MOV AL,26          ; 38d = 26h
MOV AH,16
AND AL,AH          ; ora in AL c'è 34d (22h)
JZ indirizzo       ; se vera, il bit vale 0
JNZ indirizzo      ; se vera, il bit vale 1
```

Le operazioni bitwise di scorrimento spostano i bit di un valore a destra (o a sinistra) un numero di volte specificato. Il bit che fuoriesce (a destra o a sinistra) va a finire nel CF (Carry Flag), mentre il bit che compare dall'altra parte vale sempre zero.

Lo scorrimento a destra su un byte, per esempio, si comporta così:



Lo scorrimento a sinistra su un byte, per esempio, si comporta così:



4.4 Istruzione SAL

Sintassi: **`SAL destinazione, conteggio`**

Scopo: Scorre i bit di `destinazione` a sinistra (Left) tante volte quanto vale conteggio. Ogni singolo bit che esce a sinistra transita nel Carry Flag e ogni singolo bit che entra a destra vale 0.

Esempi:
```
SAL AL,3
SAL BX, AL
```

Nota: Il CF contiene l'ultimo bit uscito a sinistra. Notare che in `destinazione` risulta la moltiplicazione tra `destinazione` e $2^{\text{conteggio}}$.

Nel primo esempio, quindi, in AL risulta AL*8.

4.5 Istruzione SAR

Sintassi: **`SAR destinazione, conteggio`**

Scopo: Scorre i bit di `destinazione` a destra (Right) tante volte quanto vale conteggio. Ogni singolo bit che esce a destra transita nel Carry Flag e ogni singolo bit che entra a sinistra vale 0.

Esempi:
```
SAR AL,3
SAR BX, AL
```

Nota: Il CF contiene l'ultimo bit uscito a destra. Notare che in `destinazione` risulta la divisione tra `destinazione` e $2^{\text{conteggio}}$.

Nel primo esempio, quindi, in AL risulta AL/8.

PROGRAMMA

Uso di SAR: trasformazione binaria

Digitato un carattere, mostrare sullo schermo la sua conversione binaria.

Layout:

```
C:\>shiftbit
Digitare un tasto: 3
bit: 00110011
C:\ >
```

Attenzione: il tasto 3 ha codice Ascii 33h, ovvero (00110011)b

Per convertire il valore digitato in binario, viene effettuato, sul valore, uno scorrimento a destra di una posizione per 8 volte. Ad ogni scorrimento viene valutato il CF (Flag di carry) e memorizzato 30h o 31h a seconda che valga 0 o 1.

Per ottenere la giusta sequenza dei valori binari a schermo, gli zeri e gli uni vengono memorizzati da destra a sinistra in memoria.

file shiftbit.txt

```
a 100
jmp 127
;Area dei dati
;.....................................
db 'Digitare un tasto: ',0                ; stringa prompt di input
db 0d,0a,'bit: ',0,0,0,0,0,0,0,0,0        ; Buffer per la stringa di output. Notare gli 8 byte destinati ai
; stampa stringa prompt di input (valore)  ; codici Ascii dei 'bit' convertiti, Il penultimo è a indirizzo 125h
;.....................................
mov bx,102                                ; 102h è l'indirizzo della stringa del prompt di input
mov ah,2
mov dl,[bx]
cmp dl,0
je 138
int 21
inc bx
jmp 12a
;Input valore da valutare
;.....................................
mov ah,1
int 21                                    ; In AL il valore da convertire
; Conversione
;.....................................
mov bx,125                                ; Decido da dove iniziare a memorizzare i 'bit'
mov cx,8                                  ; farò lo scorrimento a destra di AL per 8 volte (un byte)
sar al,1                                  ; ogni scorrimento di AL è di una posizione
jnc 14b                                   ; se il CF vale 0, salto per memorizzare '0' (30h)
mov [bx], byte ptr 31                     ; altrimenti memorizzo '1' (31h)
jmp 14e                                   ; e proseguo
mov [bx], byte ptr 30
dec bx                                    ; la prossima locazione in cui memorizzare il 'bit' (all'indietro)

loop 142
;stampa stringa risultato
;.....................................
mov bx,117                                ; Stampa del risultato. Stamperà su una nuova riga (a causa di
mov ah,2                                  ; 0dh e 0ah) e proseguirà stampando i 'bit' che avevo
mov dl,[bx]                               ; memorizzato. L'ultima locazione è comunque 0, per terminare
cmp dl,0                                  ; la stringa AsciiZ
je 162
int 21
inc bx
jmp 154
;Fine programma
;.....................................
int 20

n shiftbit.com
rcx
64
w
q
```

PROGRAMMA

Uso di AND: verifica del bit

Digitato un carattere numerico rappresentante un valore e un carattere numerico indicante il peso di un bit, mostrare a schermo se quel bit in quel valore vale 0 o 1.

Layout:

```
C:\>testbit
Digitare un tasto: 7
Digitare il peso: 2
Il bit 2 vale 1
```

```
C:\>
```

Per effettuare il test del bit di peso n deve essere calcolato 2^n. Il calcolo viene fatto con moltiplicazioni per 2 ripetute n volte. Quindi, effettuato il test con l'operatore AND, si controlla lo ZF (Zero Flag, ovvero se l'operazione AND ha generato uno zero).

Sia il valore sia il peso vengono 'normalizzati', cioè dal loro codice Ascii si ottiene il numero equivalente (sottraendo 30h).

file testbit.txt

```
a 100
jmp 13e
;Area dei dati
;.....................................
db 'Digitare un tasto: ',0                  ; stringa prompt di input per il valore
db 0                                        ; Locazione per memorizzare il valore da convertire (116h)
db 0d,0a,'Digitare il peso: ',0             ; stringa prompt di input per il peso
db 0d,0a,'Il bit vale ',0,0                 ; stringa di output. Notare il 'buco' e il penultimo byte.
; stampa stringa prompt di input (valore)   ; Il 'buco' è a indirizzo 135h; il penultimo a indirizzo 13ch
;.....................................
mov bx,102
mov ah,2
mov dl,[bx]
cmp dl,0
je 14f
int 21
inc bx
jmp 141
;Input valore da valutare
;.....................................
mov ah,1
int 21                                      ; In AL il valore da convertire
; Normalizzato a numero e salvato
;.....................................
sub al,30                                   ; da codice Ascii a numero (sottraggo 30h)
mov [116],al                                ; e salvo in memoria.
; stampa stringa prompt di input (peso)
;.....................................
mov bx,117
mov ah,2
mov dl,[bx]
cmp dl,0
je 169
int 21
inc bx
jmp 15b
;Input peso e salvataggio
;.....................................
mov ah,1
int 21                                      ; In AL il peso
mov [135],al                                ; salvataggio nella stringa di output (il 'buco')
; Normalizzato a numero
;.....................................
sub al,30                                   ; da codice Ascii a numero
mov cx,0
```

PROGRAMMA

```
mov cl,al                                   ; calcolo la potenza di due in base al peso
mov dh,2                                    ; preparo la moltiplicazione
mov al,1
cmp cl,0                                    ; ma se il peso è zero, non moltiplico
je 184
mul dh                                      ; sto calcolando la potenza di due
loop 180
; Carico il valore
;.....................................
mov dl,[116]                                ; riprendo il valore da convertire
and dl,al                                   ; faccio il test con l'AND e la potenza di due
jz 193                                      ; se il test dà risultato zero, memorizzo 30h ('0')
mov [13c], byte ptr 31                      ; altrimenti memorizzo '1' (=31h)
jmp 198
mov [13c], byte ptr 30
;stampa stringa risultato
;.....................................
mov bx,12c
mov ah,2
mov dl,[bx]
cmp dl,0
je 1a9
int 21
inc bx
jmp 19b
;Fine programma
;.....................................
int 20

n testbit.com
rcx
64
w
q
```

5 Stack

Se un numero non è rappresentabile su una sola cifra, la sua trasformazione in stringa (numerica) è più complessa: deve essere diviso ripetutamente per 10 (fino ad ottenere come risultato 0), memorizzare i resti delle divisioni (al contrario) e, singolarmente, trasformarli in codici Ascii aggiungendo 30h.

Quella descritta è la conversione di un numero in formato decimale; se al posto del divisore 10 usassimo il divisore 2, avremmo la rappresentazione dello stesso numero in formato binario.

Per realizzare questa conversione in modo efficiente si utilizza una struttura dati fondamentale per ogni Assembly di ogni ISA, ovvero lo **stack**.

Ricordiamo che tutti i linguaggi ad alto livello usano lo stack, ma in modo trasparente al programmatore, sia per allocare/deallocare le variabili locali, sia per far transitare i parametri alle procedure e per gestire gli indirizzi di andata e ritorno delle procedure.

Per velocizzare tutti questi processi, lo stack assume la forma di una

struttura dati a **Pila** (o **LIFO**, *Last In First Out*). L'immissione di un valore nello stack si appoggia sull'ultimo valore presente nello stack, in modo tale che l'ultimo valore immesso, sempre in cima alla pila, sia immediatamente accessibile. Per raggiungere i valori sotterrati nella pila è necessario scaricare quelli che lo ricoprono, come quando si vuole prelevare un piatto in mezzo ad una pila di piatti.

Per gestire velocemente le operazioni di scrittura (inserimento) e lettura (prelevamento) dallo stack, l'ISA x-86 prevede istruzioni specifiche (rispettivamente **PUSH** e **POP**) e automatismi specifici su alcuni registri: il registro **Stack Pointer** (SP) contiene sempre e automaticamente l'indirizzo dell'ultimo elemento sulla cima dello stack, mentre il registro **Base Pointer** (BP) contiene l'indirizzo del primo elemento sulla base dello stack.

Lo stack x-86 è organizzato a **word** (due byte), ovvero ogni elemento in pila è sempre ampio due byte.

Lo stack x-86 inizia (ha la base) sempre alla fine di un segmento di memoria, ovvero l'indirizzo del primo elemento di uno stack ha sempre valore di offset pari a FFFEh.

In altre parole, all'avvio di un qualsiasi programma eseguibile (EXE o COM) il registro SP contiene sempre il valore FFFEh.

Ciò significa che la pila dello stack x-86 cresce diminuendo gli indirizzi (dello Stack Pointer) di due unità alla volta per ogni elemento.

Questa scelta è opportuna, dato che lo stack si amplia a runtime senza controllo: se si perde il controllo dello stack e lo si riempie indefinitamente (**Stack Overflow**), vengono sovrascritte locazioni di memoria del programma, ma non del sistema operativo.

Per evitare gli Stack Overflow bisogna fare attenzione che il bilancio delle chiamate di PUSH e POP sia uguale a zero, ovvero che un programma, complessivamente, effettui tante POP quante PUSH.

Le istruzioni per la gestione esplicita dello stack sono:

5.1 Istruzione PUSH

Sintassi: **`PUSH sorgente`**

Scopo: Decrementa SP di due unità e pone `sorgente` sullo stack all'indirizzo contenuto in SP.

Esempi:
```
PUSH AX
PUSH [BX]
```

Nota: `sorgente` non può essere un valore immediato, almeno nell'8086/88, ma può essere una locazione di memoria, purché ampia due byte.

5.2 Istruzione POP

Sintassi: **`POP destinazione`**

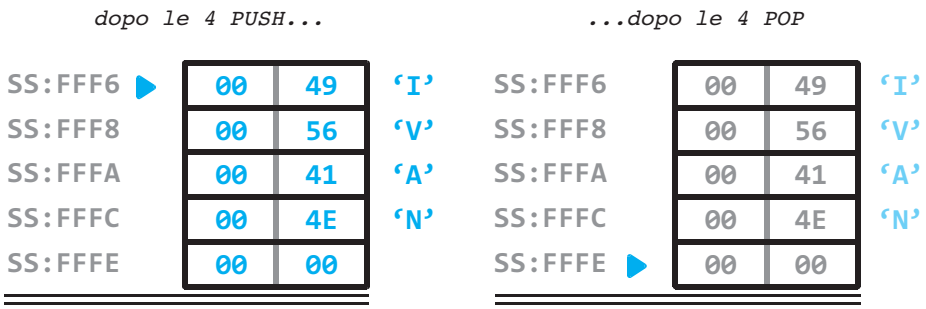
Scopo: Preleva una word dallo stack, dall'indirizzo contenuto in SP, e la deposita in `destinazione`, quindi incrementa SP di due unità.

Esempi: `POP CX`
`POP [SI]`

Nota: Nel primo caso, il valore a due byte in cima allo stack viene posto in CX. Nel secondo caso il valore in cima allo stack viene posto direttamente in memoria, occupando due celle contigue a partire dall'indirizzo specificato (SI).

Per esempio, per salvare sullo stack la parola NAVI, bisogna inserire sullo stack i quattro codici Ascii 4eh ('N'), 41h ('A'), 56h ('V'), 49h ('I') con quattro istruzioni PUSH. I codici Ascii andranno memorizzati in uno o più registri (non è importante), ma a 16 bit:

```
MOV AX, 4e
PUSH AX
MOV AX, 41
PUSH AX
MOV AX, 56
PUSH AX
MOV AX, 49
PUSH AX
```



Quindi si potrebbero riprendere e stampare a video:

```
MOV AH,2
POP DX
INT 21
POP DX
INT 21
POP DX
INT 21
POP DX
INT 21
```

ottenendo la parola bifronte IVAN.

Sfruttando lo stack possiamo effettuare un'efficace trasformazione da numero a stringa (decimale), utilizzando le istruzioni per la manipolazione dello stack, copiando su di esso tutti i resti delle divisioni per 10 del numero dato (con PUSH) e poi estraendoli per stamparli a video (con POP).

Il numero da convertire viene estratto a sorte (**random**) utilizzando un servizio MSDOS che restituisce l'ora di sistema (*Get System Time*).

5.3 INT 21h – Get System Time

```
MOV AH, 2Ch     ; sottofunzione
INT 21h         ; interruzione sw di MSDOS
```

Il servizio restituisce in CH l'ora corrente (0-23), in CL il minuto corrente (0-59) in DH i secondi (0-59) e in DL i centesimi di secondo (0-99).

PROGRAMMA

Conversione decimale-stringa

Effettuare la conversione in stringa numerica decimale di un numero casuale.

Layout:

```
C:\>dec2str
Numero casuale: 12611
C:\>
```

Nel codice proposto si predispongono due Buffer: uno per memorizzare i quozienti delle divisioni ripetute per 10; l'altro per memorizzare quante divisioni sono state fatte (fino a quoziente 0), e che corrispondono al numero di cifre del numero casuale da convertire.

Si noti che sostituendo il divisore 0ah (=10d) con 2, si ottiene la stampa a video della rappresentazione binaria del numero.

file dec2str.txt

```
a 100
jmp 116
;Area dei dati
;.....................................
db 'Numero casuale: ',0                 ; stringa AsciiZ prompt di Output
db 0,0                                  ; Buffer per memorizzare il numero casuale
db 0                                    ; Buffer per memorizzare il n. di cifre del numero
; stampa prompt di output               ; Locazione che conterrà la lunghezza della stringa numerica
;.....................................
mov bx,102
mov ah,2
mov dl,[bx]
cmp dl,0
je 127
int 21
inc bx
jmp 119
;Estrazione e salvataggio numero casuale
;.....................................
mov ah,2c                               ; Servizio 2ch di MSDOS (per INT 21h)
int 21                                  ; dopo la chiamata in DX si trovano i secondi attuali e in
                                        ; DL i centesimi. È un buon numero casuale su una word.
mov si,113                              ; 113h è l'indirizzo del Buffer di memorizzazione del numero
mov [si],dx                             ; memorizzazione del numero casuale
mov di,115                              ; 115h è l'indirizzo della cella che conterrà il n. di cifre del numero
;Conversione numero decimale in stringa
;.....................................
cmp [si],word ptr 0                     ; il numero da dividere è zero?
je 14c                                  ; sì, abbiamo finito
mov ax,[si]                             ; no, preparo la divisione
mov cx,0a                               ; divisore: 0ah = 10d (stringa decimale)
mov dx,0
div cx                                  ; divisione; il resto in DX, il quoziente in AX
mov [si],ax                             ; salvo il quoziente
add dl,30                               ; aggiusto il resto, trovando il codice Ascii equivalente
push dx                                 ; salvo il resto 'normalizzato' sullo stack
inc byte ptr [di]                       ; conto quante cifre e salvo il conteggio
jmp 133                                 ; proseguo con le divisioni successive
;Stampa a schermo della stringa decimale
;.....................................
mov cx,0                                ; inizializzo CX a zero (CL conterrà il n. di cifre da stampare)
mov cl,[di]                             ; recupero il n. di cifre da stampare
mov ah,2
pop dx                                  ; cifra da stampare prelevata dallo stack
int 21                                  ; stampa cifra (si trova già in DL)
loop 151                                ; ripeto per il n. di cifre
```

PROGRAMMA

```
;Fine programma
;.....................................
int 20

n dec2str.com
rcx
100
w
q
```

Nel caso riportato del testo, in cui il numero casuale è 12611 (=3143h), lo stack al termine delle divisioni progressive risulta:

```
ss:FFF4 ▶ 0031        posizione dello Stack Pointer dopo le divisioni
ss:FFF6   0032
ss:FFF8   0036
ss:FFFA   0031
ss:FFFC   0031
ss:FFFE   0000        posizione iniziale Stack Pointer
```

Infatti:

`12611 /10 = 1261 (=4edh)` con resto `=1; 1 + 30h =` **`31h`**
`1261 /10 = 126 (=7eh)` con resto `=1; 1 + 30h =` **`31h`**
`126 /10 = 12 (=0ch)` con resto `=6; 6 + 30h =` **`36h`**
`12/10 = 1 (=01h)` con resto `=2; 2 + 30h =` **`32h`**
`1/10 = 0 (=0h)` con resto `=1; 1 + 30h =` **`31h`**

Ovvero:

31h 32h 36h 31h 31h, cioè '1' '2' '6' '1' '1'.

6 Procedure

Una nozione base per la programmazione è quella di **procedura** (o routine, o subroutine o, in alcuni contesti, funzione). Individuata una funzionalità ripetitiva e riutilizzabile, come per esempio andare a capo o stampare una stringa sullo schermo, è possibile riunire il codice equivalente in un blocco autonomo e separato dal programma, in modo da riutilizzarlo senza doverlo riscriverlo nel codice, semplicemente invocandolo con una chiamata (CALL).

Una volta esaurito il compito, la procedura torna al programma chiamante (RET), esattamente all'indirizzo successivo dell'istruzione CALL (**indirizzo di ritorno**) e l'esecuzione prosegue.

Il meccanismo che consente la chiamata e il ritorno delle procedure è realizzato tramite lo stack, questa volta in modo implicito, ovvero senza usare le istruzioni PUSH e POP, ma sfruttando le proprietà delle istruzioni CALL e RET:

6.1 Istruzione CALL

Sintassi: **`CALL indirizzo`**
Scopo: Effettua la chiamata ad una procedura che si trova ad `indirizzo`. L'istruzione CALL esegue le seguenti operazioni:

1) salva nello stack l'indirizzo di ritorno;
2) trasferisce il controllo all'indirizzo della procedura tramite un salto incondizionato.

L'indirizzo di ritorno è l'indirizzo dell'istruzione successiva a quella di CALL.

Esempi: `CALL 116`
`CALL word ptr [BX]`

Nota: La CALL 116 può essere vista come l'unione delle due istruzioni PUSH IP; JMP 116.

Nel secondo caso un esempio di chiamata dinamica, ovvero una chiamata che assume valore solo a runtime (in base al valore attuale di BX).

6.2 Istruzione RET

Sintassi: **RET**

Scopo: Effettua il ritorno al chiamante.

L'istruzione `RET` assume che l'indirizzo di ritorno si trovi attualmente in cima allo stack.

Essa esegue le seguenti operazioni:

1) preleva dallo stack dell'indirizzo di ritorno;
2) salto all'indirizzo di ritorno.

Esempi: `RET`

Nota: La RET, che va sempre posta come ultima istruzione di un blocco di procedura, esegue, praticamente, le seguenti istruzioni: POP BX/JMP [BX], oppure, con una sola istruzione logica: POP IP.

PROGRAMMA

Procedura senza parametri: Acapo

Scrivere i numeri 0, 1 e 2 su tre righe differenti usando una procedura per andare a capo.

Layout:

```
C:\>acapo
0
1
2
C:\>
```

file acapo.txt

```
a 100
jmp 10d                                 ; Salto all'inizio del programma (saltando la procedura)
;.....................................
; Procedura Acapo
;.....................................
mov ah,2                                ; indirizzo 102h; Inizio della procedura
mov dl,0d
int 21
mov dl,0a
int 21
ret                                     ; ecco il ritorno al chiamante
;.....................................
; Main
```

PROGRAMMA

```
;.....................................
mov ah,02                              ; indirizzo 10dh: inizio del programma
mov dl,30
int 21
call 102                               ; chiamata alla procedura
mov ah,02                              ; la RET ritornerà automaticamente qui (indirizzo 116h)
mov dl,31
int 21
call 102                               ; chiamata alla procedura
mov ah,02                              ; la RET ritornerà automaticamente qui
mov dl,32
int 21
;Fine programma
;.....................................
int 20

n acapo.com
rcx
27
w
q
```

Il movimento (implicito) dello stack risulta essere:

```
ss:FFFA   ....
ss:FFFC   ....
ss:FFFE ► 0000      Valore iniziale dello Stack Pointer

ss:FFFA   ....
ss:FFFC ► 0116      Indirizzo di ritorno, dopo la CALL 102
ss:FFFE   0000

ss:FFFA   ....
ss:FFFC   0116
ss:FFFE ► 0000      Dopo la RET nella procedura
```

Molto spesso le procedure diventano fondamentali quando ad esse possono essere forniti valori in ingresso, in modo che possano agire su dati differenti, come per esempio una procedura che stampa sullo schermo una stringa.

In questo caso è necessario «passare» alla procedura un valore (**parametro**), cioè l'indirizzo della stringa da stampare, in modo che la procedura possa agire, di volta in volta, sulla stringa desiderata.

In Assembly i parametri possono essere passati **per registro**, ovvero si considera un registro prefissato come locazione comune a chiamante (il programma) e procedura: la procedura assume che in quel registro sia pronto il valore che dovrà usare.

Naturalmente è possibile adoperare più registri per più parametri.

È possibile, tuttavia, passare i parametri anche nel modo classico attraverso lo stack, organizzandolo in un formato convenzionale (esempio, modo **CDecl** caratteristico del linguaggio C).

PROGRAMMA

Procedura con parametro: stampa stringa

Stampare sullo schermo due stringhe su righe differenti usando una procedura.

Layout:

```
C:\>sstr
ciao
mare
C:\>
```

```
                file sstr.txt
a 100
jmp 11B                                 ; Salto all'inizio del programma
;Area dei dati
;.....................................
db 'Ciao',0                             ; prima stringa da stampare a indirizzo 102h
db 'mare',0                             ; seconda stringa da stampare a indirizzo 107h
;.....................................
; Procedura stampa stringa AsciiZ
;.....................................
mov ah,2                                ; indirizzo 10ch; Inizio della procedura
mov dl,[bx]                             ; si assume che in BX ci sia l'indirizzo della stringa da
                                          stampare
cmp dl,0
je 11A
int 21
inc bx
jmp 10c
ret                                     ; ecco il ritorno al chiamante
;.....................................
; Main
;.....................................
; Chiamata a procedura di stampa .........
mov bx,102                              ; passaggio per registro: in BX indirizzo della stringa da
                                          stampare
call 10c                                ; chiamata alla procedura
; A capo ...............................
mov ah,2                                ; la RET ritornerà automaticamente qui (indirizzo 121h)
mov dl,0d
int 21
mov dl,0a
int 21
; Chiamata a procedura di stampa .........
mov bx,107                              ; passaggio per registro: in BX indirizzo della stringa da
                                          stampare
call 10c                                ; chiamata alla procedura
;Fine programma
;.....................................
int 20                                  ; la RET ritornerà automaticamente qui (indirizzo 131h)

n proc.com
rcx
33
w
q
```

La chiamata a procedura implica un inconveniente: dopo la chiamata, i registri usati dalla procedura per eseguire il suo compito risultano sovrascritti: ciò che si era memorizzato in quei registri prima della chiamata è andato perso (sovrascritto).

Per ovviare a questo inconveniente una procedura deve, prima di iniziare il proprio compito, salvare sullo stack – con l'istruzione PUSH – i contenuti di tutti i registri che userà quindi, appena prima dell'istruzione RET, recuperarli ordinatamente con altrettante istruzioni POP (**preservare i registri**).

ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1. Commentare la redirezione dei comandi per una shell a carattere. Fornire degli esempi.
2. Mostrare la linea di comando di una shell di MSDOS per il programma Debug, per generare il file COM e il file di listing di un programma di nome pluto.
3. Elencare e commentare i passaggi per scrivere un file di testo sorgente per il programma Debug.
4. Illustrare i concetti di prima passata e seconda passata per la scrittura di un programma Assembly con file sorgente per Debug.
5. Spiegare come è possibile commentare i file sorgenti per il programma Debug.
6. Definire e commentare la nozione di file di listing per il programma Debug.
7. Mostrare la pseudoistruzione Assembly che alloca in memoria il codice Ascii dell'iniziale maiuscola del proprio nome e di una variabile inizializzata a zero.
8. Spiegare i concetti di area codice e area dati di un programma.
9. Mostrare come si imposta a 1 il bit5 di un valore dato.
10. Mostrare la pseudoistruzione Assembly che alloca in memoria la stringa AsciiZ del proprio cognome e che stampata vada anche a capo.
11. Calcolare NOT, OR e AND di due byte a piacere.
12. Mostrare la sequenza di istruzioni per memorizzare sullo stack la parola Pippo e poi recuperarla stampandola.
13. Spiegare il meccanismo di chiamata a procedura e ritorno da procedura.

Requisiti avanzati

1. Commentare la nozione di dereferenziazione e mostrane un esempio in Assembly.
2. Spiegare i meccanismi impliciti delle istruzioni Assembly DIV e MUL.
3. Elencare gli intervalli dei codici Ascii numerici, alfabetici minuscoli e maiuscoli.
4. Mostrare come si verifica se il bit4 di un certo numero vale 0 o 1.
5. Mostrare come si inverte il valore di un bit di peso dato.
6. Spiegare il rapporto tra le istruzioni SAR/SAL e alcune operazioni aritmetiche.
7. Illustrare i ruoli dei registri SP e BP nella gestione dello stack.
8. Illustrare il concetto di stack overflow.
9. Riportare con esattezza come le istruzioni PUSH e POP agiscono sul registro SP.
10. Spiegare i meccanismi impliciti delle istruzioni Assembly CALL e RET.
11. Spiegare perché lo stack nei file COM parte sempre da un indirizzo con spiazzamento uguale a FFFFh.
12. Spiegare l'importanza di preservare i registri per una procedura.
13. Scrivere un file batch per MSDOS che «compili» e crei il file di listing tramite Debug per un programma di nome ciccio.

ESERCIZI PER LA VERIFICA SCRITTA

I prerequisiti richiesti per affrontare questi esercizi sono la conoscenza dei sistemi di numerazione in base 2 e in base 16 (binario e esadecimale), la conversione decimale-binario e viceversa, la conversione decimale-esadecimale e viceversa, anche con applicativi (esempio, Calc).

È necessaria la consultazione di una tabella di codici Ascii.

1 Scrivere su un foglio i passaggi per ottenere un file di testo di nome lstroot.txt contenente tutti i nomi di file e directory della radice di un disco fisso.

2 Scrivere su un foglio i passaggi per ottenere un file di testo di nome lstpen.txt contenente tutti i nomi di file e directory della radice di un pendrive.

3 Scrivere su un foglio i passaggi per ottenere un file di testo di nome lstsys.txt contenente tutti i nomi di file e directory della cartella c:\windows\system32.

4 Scrivere su un foglio i passaggi per creare, editare, mettere a punto e generare il file eseguibile paperino.com con Debug, a partire da un file di nome paperino.txt.

5 Mostrare la pseudoistruzione Assembly per allocare il codice Ascii della lettera A.

6 Mostrare la pseudoistruzione Assembly per allocare il carattere 0 e il numero 0.

7 Mostrare la sequenza di istruzioni Assembly per memorizzare a indirizzo 103 il codice Ascii della lettera A.

8 Mostrare la sequenza di istruzioni Assembly per memorizzare a indirizzo 103 il carattere 0 e a indirizzo 104 il numero 0.

9 Mostrare la pseudoistruzione Assembly per allocare la stringa AsciiZ della parola gennaio e per allocare nel modo del Pascal la stessa stringa. *Il modo del Pascal prevede che il primo byte dell'allocazione di una stringa contenga il numero dei caratteri di quella stringa*.

10 Indicare la sequenza di istruzioni Assembly per stampare una stringa AsciiZ allocata a partire dall'indirizzo di spiazzamento 102h.

11 Indicare la sequenza di istruzioni Assembly per stampare una stringa allocata nel modo del Pascal a partire dall'indirizzo di spiazzamento 102h. *Il modo del Pascal prevede che il primo byte dell'allocazione di una stringa contenga il numero dei caratteri di quella stringa*.

12 Scrivere due diverse sequenze di istruzioni Assembly usando l'istruzione ADD in modo che nel registro AX risulti il valore 250d.

13 Scrivere due diverse sequenze di istruzioni Assembly usando l'istruzione SUB in modo che nel registro CX risulti il valore 100d.

14 Scrivere una sequenza di istruzioni Assembly in modo tale che nel registro AX risulti la somma dei primi cinque numeri naturali, zero compreso.

15 Scrivere una sequenza di istruzioni Assembly in modo tale che nel semiregistro AL risulti il valore 12 come risultato di una divisione.

16 Eseguire l'operazione NOT del valore 13 prima riferito al nibble, poi riferito al byte.

17 Eseguire l'operazione A AND B con A=23 e B=12.

18 Eseguire l'operazione A OR B con A=23 e B=12.

19 Eseguire l'operazione NOT (A OR B) con A=33 e B=22.

20 Eseguire l'operazione NOT (A AND B) con A=33 e B=22.

21 Eseguire l'operazione NOT ((NOT A) OR B) con A=33 e B=22.

22 Eseguire le operazioni A OR B e A + B con A=23 e B=12. Fornirne una versione in Assembly.

23 Eseguire i passaggi per impostare a 1 il bit4 del valore 34.

24 Eseguire i passaggi per impostare a 0 il bit1 del valore 34.

25 Eseguire i passaggi per impostare a 1 il bit8 del valore 34.

26 Eseguire i passaggi per impostare a 0 il bit8 del valore 334.

27 Eseguire i passaggi per ottenere il valore del bit4 del valore 56.

28 Eseguire i passaggi per ottenere il valore del bit8 del valore 56.

29 Eseguire l'operazione A SAR B con A=33 e B=1.

30 Eseguire l'operazione A SAL B con A=33 e B=1.

31 Eseguire l'operazione A SAL B con A=33 e B=3.

32 Eseguire l'operazione A SAR B con A=33 e B=3.

33 Eseguire le operazioni A SAL B e A * 2^B con A=10 e B=3. Fornirne una versione in Assembly.

34 Eseguire le operazioni A SAR B e A / 2^B con A=80 e B=3. Fornirne una versione in Assembly.

35 Scrivere una sequenza di istruzioni Assembly per depositare sullo stack la parola ROMA.

36 Scrivere una sequenza di istruzioni Assembly per depositare sullo stack la parola ROMA usando solo due istruzioni PUSH.

ESERCIZI PER LA VERIFICA DI LABORATORIO

I prerequisiti per affrontare questi esercizi sono la disponibilità del programma Debug.exe e la disponibilità di un editor di testo (esempio, notepad.exe).

Può essere molto utile usare un **debugger** per codice x-86, come ad esempio Turbo Debugger (td.exe) facilmente reperibile come programma gratuito. L'uso di Turbo Debugger è estremamente semplice: dato il file pippo.com da controllare, usare la linea di comando:

```
C:\>td pippo.com          ; se il file pippo.com si trova nella cartella corrente
```

Ora si può verificare il programma passo-passo (con i tasti F7 o F8), eseguire fino ad una istruzione desiderata (F4) e consultare memoria, registri e stack contemporaneamente.

Debug con sorgente

1 **Scrivere un programma Assembly con Debug e file sorgente che stampi sullo schermo il proprio cognome (terza versione).**

La prima stesura del file di testo sorgente (cognome3.dbg) risulta essere la seguente:
NB: Ricordare di lasciare una riga vuota dopo l'ultima riga del codice e di mettere l'acapo dopo il comando q di uscita.
file di testo scritto con Blocco Note: ***cognome3.txt***

```
a 100
jmp 100
db 52
db 6f
db 73
db 73
db 69
mov cx,5
mov bx,100
mov ah,2
mov dl,[bx]
int 21
inc bx
loop 100
int 20

rcx
100
n cognome3.com
w
q
```

Come prima, l'indirizzo del salto iniziale è temporaneo (non si può sapere a cosa corrisponderà), così come l'indirizzo del primo byte dell'area Dati (il codice Ascii 52h), così come l'indirizzo per il ciclo loop, così come la dimensione del programma.

Tutti e quattro questi valori sono stati impostati al valore temporaneo 100h.
Il file di listing corrispondente (cognome3.lst, ottenuto con il comando MSDOS debug < cognome2.dbg > cognome3.lst) dà le indicazioni per i valori effettivi degli indirizzi e della dimensione del programma (in evidenza):
(file di listing: cognome3.lst)

```
-a 100
13EB:0100 jmp 100
13EB:0102 db 52
13EB:0103 db 6f
13EB:0104 db 73
13EB:0105 db 73
13EB:0106 db 69
13EB:0107 mov cx,5
13EB:010A mov bx,100
13EB:010D mov ah,2
13EB:010F mov dl,[bx]
13EB:0111 int 21
13EB:0113 inc bx
13EB:0114 loop 100
13EB:0116 int 20
13EB:0118
-rcx CX 0000
:100
-n cognome3.com
-w
Writing 00100 bytes
-q
```

Quindi la seconda stesura del file sorgente risulta: (file: cognome3.txt)

```
a 100
jmp 107
db 52
db 6f
db 73
```

```
db 73
db 69
mov cx,5
mov bx,102
mov ah,2
mov dl,[bx]
int 21
inc bx
loop 10d
int 20

rcx
18
n cognome3.com
w
q
```

OUTPUT

```
C:\dbg>cognome3
Rossi
C:\dbg>
```

2 **Scrivere un programma Assembly con Debug e file sorgente che, presi in input due numeri (da 0 a 9) stampi quale è il maggiore o se sono uguali.**

```
Layout:
c:\>duen
Input primo numero: 1
Input secondo numero: 2
Il secondo è maggiore
C:\>
```

oppure

```
c:\>duen
Input primo numero: 3
Input secondo numero: 3
Sono uguali
C:\>
```

In questo esercizio si può notare come i due numeri in input, sottoforma di codici Ascii, vengono memorizzati in due semiregistri inutilizzati (ch e cl), in modo da poter fare i confronti successivamente. Ciò accade agli indirizzi 0181 e 0198, mentre il confronto si trova all'indirizzo 019a.

NB: Si noti come gli indirizzi temporanei, da un certo punto in avanti, sono stati cambiati da (100)H a (200)H.

Questo è necessario quando il salto tra istruzione e indirizzo è maggiore di 128.

Prima stesura con indirizzi temporanei:

```
a 100
; salto i dati
jmp 100
; messaggi a video
db 'Input primo numero: ',0
db 0d,0a,'Input secondo numero: ',0
db 0d,0a,'Il primo è maggiore',0
db 0d,0a,'Il secondo è maggiore',0
db 0d,0a,'Sono uguali',0
; Stampo a video la prima descrizione
; ..............................
;
mov bx,100
mov ah,2
mov dl, [bx]
cmp dl,0
je 100
int 21
inc bx
jmp 100
; Ora faccio l'input del primo numero
; ..............................
;
mov ah,1
int 21
mov cl,al
; Stampo a video la seconda descrizione
; ..............................
;
mov bx,100
mov ah,2
mov dl, [bx]
cmp dl,0
je 200
int 21
inc bx
jmp 200
; Ora faccio l'input del secondo numero
; ..............................
mov ah,1
int 21
mov ch,al
; Confronti
cmp cl,ch
jg 200
jl 200
; Allora sono uguali: lo stampo
; ..............................
mov bx, 100
mov ah,2
mov dl, [bx]
cmp dl,0
je 200      ; a fine programma
int 21
inc bx
jmp 200
```

```
; Il primo è maggiore: lo stampo
; ..............................
mov bx, 100
mov ah,2
mov dl, [bx]
cmp dl,0
je 200      ; a fine programma
int 21
inc bx
jmp 200
; Il secondo è maggiore: lo stampo
; ..............................
mov bx, 100
mov ah,2
mov dl, [bx]
cmp dl,0
je 200      ; a fine programma
int 21
inc bx
jmp 200
; Fine programma
; ..............................
int 20

rcx
200
n duen.com
w
q
```

Seconda stesura con indirizzi effettivi:

```
a 100
; salto i dati
jmp 16c
; messaggi a video
db 'Input primo numero: ',0
db 0d,0a,'Input secondo numero: ',0
db 0d,0a,'Il primo è maggiore',0
db 0d,0a,'Il secondo è maggiore',0
db 0d,0a,'Sono uguali',0
; Stampo a video la prima descrizione
; ..............................
;
mov bx,102
mov ah,2
mov dl, [bx]
cmp dl,0
je 17d
int 21
inc bx
jmp 16f
; Ora faccio l'input del primo numero
; ..............................
;
mov ah,1
int 21
mov cl,al
; Stampo a video la seconda descrizione
; ..............................
;
mov bx, 117
mov ah,2
mov dl, [bx]
cmp dl,0
je 194
int 21
inc bx
jmp 186
; Ora faccio l'input del secondo numero
; ..............................
mov ah,1
int 21
mov ch,al
; Confronti
cmp cl,ch
jg 1b1
jl 1c2
; Allora sono uguali: lo stampo
; ..............................
mov bx, 15e
mov ah,2
mov dl, [bx]
cmp dl,0
je 1d3      ; a fine programma
int 21
inc bx
jmp 1a3
; Il primo è maggiore: lo stampo
; ..............................
mov bx, 130
mov ah,2
mov dl, [bx]
cmp dl,0
je 1d3      ; a fine programma
int 21
inc bx
jmp 1b4
; Il secondo è maggiore: lo stampo
; ..............................
mov bx, 146
mov ah,2
mov dl, [bx]
cmp dl,0
je 1d3      ; a fine programma
int 21
inc bx
jmp 1c5
```

```
; Fine programma
; ..............................
int 20

rcx
D5
n duen.com
w
q
```

OUTPUT

```
C:\dbg>duen
Input primo numero: 2
Input secondo numero: 7
Il secondo è maggiore
C:\dbg>
```

3 **Scrivere un programma Assembly con Debug e file sorgente che prenda in input una stringa di massimo 10 caratteri e la stampi al contrario.**

```
Layout:

c:\>reverse
Input stringa: ciao
oaic
C:\>
```

In questo esercizio si mostra come è possibile allocare dati in memoria, per esempio una stringa. Prima di tutto viene allocato un buffer adeguato (esempio, di 11 byte) con la parola chiave db seguita da 11 byte a zero (NULL) separati da virgole. La memorizzazione avviene ancora tramite il registro bx e la sintassi con parentesi quadre, incrementando bx ad ogni input di carattere.
L'input termina quando il carattere digitato dall'utente è CR (0d)H.
Per stampare al contrario si accede alla fine del buffer e si cicla all'indietro (dec bx). Se il byte letto è diverso da NULL, si stampa. Se l'indirizzo in bx scende sotto quello iniziale (memorizzato nel registro si), la stampa termina.
Stesura con indirizzi effettivi:

```
a 100
; salto i dati
jmp 11d
; messaggi a video
db 'Input stringa: ',0
db 0,0,0,0,0,0,0,0,0,0,0
; Stampo a video il messaggio
; ..............................
;
mov bx,102
mov ah,2
mov dl, [bx]
cmp dl,0
je 12e
int 21
inc bx
jmp 120
; Input della stringa
; ..............................
;
mov bx,112
mov ah,1
int 21
cmp al,0d
je 13e      ; input terminato
mov [bx],al
inc bx
jmp 131
; ..............................
; A capo
; ..............................
mov ah,2
mov dl,0d
int 21
mov dl,0a
int 21
; Stampa al contrario
;..............................
mov bx,112
mov si,bx   ; memorizzo l'indirizzo iniziale
add bx,0a   ; vado al termine del buffer
mov ah,2
cmp bx,si
jl 162      ; ho finito
mov dl,[bx]
dec bx
cmp dl,0
je 150      ; non è stampabile
int 21      ; stampo
jmp 150
; Fine programma
; ..............................
int 20

rcx
64
n reverse.com
w
q
```

OUTPUT

```
C:\dbg>reverse
Input stringa: Informatica
acitamrofnI
c:\dbg>
```

4 Scrivere un programma Assembly con Debug e file sorgente che, preso in input un carattere numerico per esempio n, stampi sullo schermo un «quadrato» di cancelletti di lato n.

```
Layout:
c:\dbg>can
? 4
####
####
####
####
c:\>
```

5 Scrivere un programma Assembly con Debug e file sorgente che, preso in input un carattere qualsiasi, verifichi che sia numerico e consenta di proseguire solo in questo caso, altrimenti riproponga l'input. Quindi stampi una linea «lunga» quel numero.

```
Layout:
c:\dbg>linea
? a
? ;
? 7

_______
c:\>
```

6 Scrivere un programma Assembly con Debug e file sorgente che, preso in input tre caratteri, stampi a video quello con codice Ascii maggiore (se uguali, stampi quel carattere).

```
Layout:
c:\dbg>tremag
1? a
2? c
3? b
Maggiore: c
c:\>
```

7 Scrivere un programma Assembly con Debug e file sorgente che, presa in input una stringa di massimo 12 caratteri, stampi M se c'è almeno una maiuscola o un trattino altrimenti.

```
Layout:
c:\dbg>maiu
? Ciao
M
c:\>
```

8 Scrivere un programma Assembly con Debug e file sorgente che, presa in input una stringa di massimo 12 caratteri, stampi S se c'è almeno uno spazio.

```
Layout:
c:\dbg>spa
? Ciao mare
S
c:\>
```

9 Scrivere un programma Assembly con Debug e file sorgente che, presa in input una stringa di massimo 20 caratteri, stampi N se c'è almeno un carattere numerico, un trattino altrimenti.

```
Layout:
c:\dbg>almn
? A.S. 2013
N
c:\>
```

oppure

```
c:\dbg>almn
? Ciao mare
-
c:\>
```

10 Scrivere un programma Assembly con Debug e file sorgente che, presa in input una stringa di massimo 20 caratteri, la ristampi senza spazi.

```
Layout:
c:\dbg>spa1
? Ciao mare
Ciaomare
c:\>
```

oppure

```
c:\dbg>almn
? Rossi
Rossi
c:\>
```

11 Scrivere un programma Assembly con Debug e file sorgente che, presa in input una stringa di massimo 10 caratteri, e un carattere numerico, stampi il carattere a quella posizione.

```
Layout:
c:\dbg>mid
Inserire stringa (max 10 char): Ciao mare
Inserire numero: 5
m
c:\>
```

12 Scrivere un programma Assembly con Debug e file sorgente che, presa in input una stringa di massimo 16 caratteri, stampi a video Vocali! se nella stringa c'è almeno una vocale, altrimenti un trattino.

```
Layout:
c:\dbg>voc
Inserire stringa (max 16 char): Ciao mare
Vocali!
c:\>
```

oppure

```
c:\dbg>voc
Inserire stringa (max 16 char):
   LLRPLM63B25G337M
-
c:\>
```

13 Scrivere un programma Assembly con Debug e file sorgente che stampi il PSP.

```
Layout:
c:\dbg>psp
Stampa PSP:
µ ?♬¶ ↑ ?♬          ♣ =!╦
c:\>
```

oppure (versione controllata)

```
c:\dbg>psp Ciao mare
Stampa PSP:
.µ. .?.♬.¶. .↑. .?.♬. . . . . .♣. . . . .
. . . . . .=.!.╦. . .
. .  . . .C.I.A.O. . . . . .
. .M.A.R.E. . . . . .
. . .
. . . . . .
. . . . . .
. . . . . .
c:\>
```

14 Scrivere un programma Assembly con Debug e file sorgente che stampi sullo schermo il proprio nome e cognome su due righe. Quindi prenda in input un carattere. Se il codice Ascii del carattere è uguale all'iniziale del nome, stampare un punto esclamativo, altrimenti un punto interrogativo. Andare a capo, prendere in input un altro tasto e fare la stessa cosa (stavolta con l'iniziale del cognome).

```
Layout:
c:\>s00
Primo
Occhini
n? P!
c? O!
c:\>
```

oppure

```
c:\>s00
Paolo
Ollari
n? p?
c? O!
c:\>
```

15 Scrivere un programma Assembly con Debug e file sorgente che, presi in input due tasti numerici stampi quale dei due è maggiore o se sono uguali.

```
Layout:
c:\>s001
1? 5
2? 7
Il secondo è maggiore
c:\>
```

oppure

```
c:\>s001
1? 5
2? 5
Sono uguali
c:\>
```

16 Scrivere un programma Assembly con Debug e file sorgente che, dopo aver mostrato un prompt, acquisisca una stringa con terminatore (max 20 caratteri), la allochi in memoria e verifichi se è tutta numerica o no.

```
Layout:
c:\>numono
inserire stringa (max 20 char): ciao
numerica: no
c:\>
```

17 Scrivere un programma Assembly con Debug e file sorgente che, dopo aver mostrato un prompt, acquisisca una stringa con terminatore (max 20 caratteri), la allochi in memoria e verifichi se è tutta alfabetica o no.

```
Layout:
c:\>alfono
inserire stringa (max 20 char): ciao
alfabetica: si
c:\>
```

18 Scrivere un programma Assembly con Debug e file sorgente che, dopo aver mostrato un prompt, acquisisca una stringa con terminatore (max 20 caratteri), la allochi in memoria e verifichi se è tutta maiuscola o no.

```
Layout:
c:\>maiono
inserire stringa (max 20 char): Ciao
maiuscola: no
c:\>
```

19 Scrivere un programma Assembly con Debug e file sorgente che estrae un numero casuale a una cifra e lo stampi sullo schermo.

```
Layout:
c:\>randoms
Numero casuale: 5
c:\>
```

20 Scrivere un programma Assembly con Debug e file sorgente che, presi in input i tasti speciali «freccia», sposti un carattere ('*') sullo schermo in base alla direzione.

Se si preme 'Esc' il programma termina.

```
Layout:
c:\>frecce
Tasti speciali

***********
         *****
c:\>
```

Usare le nuove interruzioni software:

– Posizionamento cursore a video

```
Mov dl, x  ; la coordinata X in dl
Mov dh, y  ; la coordinata Y in dh
mov bh,0   ; bh sempre a 0
mov ah,2   ; servizio Bios posizione cur-
             sore
int 10
```

– Input senza echo sullo schermo

```
mov ah,7   ; servizio MSDOS input senza
             echo
int 21     ; in al il codice Ascii in input
```

Per acquisire un tasto speciale freccia (su, giù, destra, sinistra), bisogna verificare se l'input del tasto vale 0 (NULL).

Se vale 0 bisogna leggere di nuovo l'input e il valore letto ha le seguenti corrispondenze:

```
48h=su, 50h=giù, 4bh=destra,
4dh=sinistra
```

21 Scrivere un programma Assembly con Debug e file sorgente che, presi in input tre tasti numerici, stampi sullo schermo qual è il maggiore.

```
Layout:
C:\>max
Maggiore dei tre.
?4,5,2
Il maggiore: 5
C:\>
```

oppure

```
C:\>max
Maggiore dei tre.
?2,2,2
Il maggiore: 2
C:\>
```

oppure

```
C:\>max
Maggiore dei tre.
?2,s,3,9
Il maggiore: 9
C:\>
```

22 Scrivere un programma Assembly con Debug e file sorgente che acquisisce un tasto numerico da tastiera ed estrae un numero casuale a una cifra. Stampare se il numero estratto è maggiore di quello in input.

```
Layout:
c:\>randomc
? 6
Numero casuale: 7
maggiore
c:\>
```

23 Scrivere un programma Assembly con Debug e file sorgente che stampi a schermo tutti i caratteri numerici con il codice Ascii avente il bit 3 a 1.

```
Layout:
C:\>numbit
Numeri con bit 3 a 1: 5 6 7
C:\>
```

24 Scrivere un programma Assembly con Debug e file sorgente che, presi in input un valore e un peso, imposti a 1 il bit specificato.

```
Layout:
C:\>setbit
Digitare un tasto: 6
Digitare il peso: 3
Il valore ottenuto: >
C:\>
```

25 Scrivere un programma Assembly con Debug e file sorgente che usi una procedura per effettuare l'input di due stringhe.

26 Scrivere un programma Assembly con Debug e file sorgente che usi una procedura per copiare una stringa in un'altra.

27 Scrivere un programma Assembly con Debug e file sorgente che stampi un numero casuale in formato binario tramite una procedura a cui passare il numero.

28 Scrivere un programma Assembly con Debug e file sorgente che stampi un numero casuale in formato esadecimale tramite una procedura a cui passare il numero.

29  Scrivere un programma Assembly con Debug e file sorgente che stampi un numero casuale in formato decimale tramite una procedura a cui passare il numero.

30 Scrivere un programma Assembly con Debug e file sorgente che implementa una procedura che restituisce (per registro) se una stringa data è numerica o meno.

31 Scrivere un programma Assembly con Debug e file sorgente che implementa una procedura che restituisce la lunghezza di una stringa data.

B

Reti. Standard di riferimento, organizzazione in livelli, reti locali e geografiche, protocolli

La storia delle reti di calcolatori ha origine alla fine degli anni '50 negli Stati Uniti sotto la spinta delle ricerche militari del **DARPA** (Defence Advanced Research Project Agency).

L'evoluzione delle idee e delle innovazioni che ne seguirono condussero a realizzare le due tipologie di reti di calcolatori ancora oggi più diffuse, dapprima le **reti geografiche** (WAN), quindi le **reti locali** (LAN).

In un primo momento **Leonard Kleinrock** (1962) del MIT (Massachusetts Institute of Technology) ideò le reti switching basate sull'infrastruttura telefonica statunitense e nel 1965 fu inaugurata la prima rete geografica tra il MIT e l'università di Berkeley. Poco più tardi **Lawrence Roberts** (1967), per conto di DARPA, ideò la rete **ARPANET**, antesignana di Internet, prodotta dalla Bolt, Beranek e Newman Inc. e realizzata tramite **IMP** (i primi apparati di rete, antesignani dei router). Il primo protocollo di rete viene scritto da **Robert Kahn**, per ARPANET. Era **NCP** (*Network Control Program*), precursore di TCP, e fu pubblicato nel 1972. Nello stesso anno compare il primo programma applicativo di rete pubblico, la **Email** a cura di Ray Tomlinson. Quindi Vinton Cerf e Robert Kahn scrissero la prima versione di **TCP/IP** (1974).

Pochi anni dopo videro la luce le prime reti locali; dai lavori di Norman Abramson, **Robert Metcalfe** e **David Boggs** dell'azienda Xerox Inc. diedero vita a **Ethernet** (1976), la prima rete locale a grande diffusione.

Gli anni successivi videro il boom delle tecnologie di rete e il panorama mondiale si arricchì di numerose soluzioni proprietarie, sia pubbliche sia locali (tra cui SNA di IBM, XNS, NetBios, Novell, ecc.), fino alla decretazione degli standard **LAN 802.3** (1983, ex Ethernet) e standard WAN **TCP/IP** (con la migrazione ufficiale di ARPANET a TCP/IP, 1983).

Lo scenario generale viene anche regolamentato mediante l'introduzione di uno standard di riferimento dovuto a **OSI** (*Open Systems Interconnection*), in cui viene proposta una visione delle reti di calcolatori a 7 livelli (**Charles Bachman**, *Honeywell Information Services*, 1981).

B1 Reti di calcolatori

Le reti di calcolatori, assieme alla diffusione di massa dei personal computer, sono protagoniste del rinnovamento informatico che inizia dagli anni '90 su scala planetaria. La prima grande innovazione prende il nome di **downsizing** (la migrazione da sistemi centralizzati a modelli distribuiti). Le precedenti reti infatti dipendevano totalmente dal modello **centralizzato** a «mainframe», cioè grandi e potenti calcolatori supportati dalle unità usate dagli utenti di tipo **terminale**, cioè console composte solo da monitor e tastiera senza unità di calcolo.

Tali sistemi sono molto costosi (per esempio, il mainframe) e poco scalabili (cioè non facilmente ridimensionabili), ma soprattutto sono **sistemi proprietari**, quindi assolutamente poco versatili.

Dapprima le reti locali, poi le reti geografiche hanno scalzato quel modello diffondendo molto velocemente un modello **distribuito** basato su rete locale, molto più economico, scalabile e versatile.

I sistemi operativi che sono alla base di questo nuovo modello sono detti **sistemi aperti** o anche **open source**, come Windows e Linux.

1 Enti di standardizzazione

Siccome una rete è un sistema che coinvolge elementi tra di loro non omogenei, come apparati differenti, calcolatori differenti, sistemi operativi differenti, programmi differenti, e prevede il trasferimento di dati attraverso confini pubblici e internazionali, è molto importante che l'informazione e le strutture ad essa preposte siano regolamentate. È importante che le normative siano proposte da enti terzi rispetto ai produttori per evitare monopoli, garantire l'interoperabilità e anche la legalità delle operazioni in rete.

Questi enti quindi sono consultati dai produttori, dai sistemisti ma anche dai programmatori per poter produrre materiale hardware e/o software, in grado di rispettare gli standard previsti per il corretto funzionamento dei prodotti:

- **PTT** (*Post*, *Telegraph*, *Telephone*). Con le nuove ristrutturazioni economiche (dopo il 1996) come PTT dobbiamo considerare solo i governi nazionali che danno in concessione e in libera concorrenza le licenze d'uso delle infrastrutture pubbliche a gruppi privati (in Italia, tramite il ministero delle Telecomunicazioni o affine).

- **ITU-T** (*International Telecommunication Union*, *Telecommunication Standardization Bureau*, *www.itu.int/ITU-T/*), sezione dell'ITU, organo delle Nazioni Unite con sede a Ginevra, che si occupa di regolare le telecomunicazioni telefoniche e telegrafiche. Fino al 1993 era noto come CCITT (Comité consultatif international téléphonique et télégraphique). Recepisce e coordina i vari PTT nazionali.
- **ISO** (*www.iso.org*) che va inteso come International Standard Organization, ed è il maggiore ente di standardizzazione mondiale, non governativo, con sede a Ginevra. I suoi membri sono gli enti nazionali di standardizzazione, per l'Italia l'UNI (Ente Nazionale Italiano di Unificazione).
- **ANSI** (*American National Standards Institute*, *www.ansi.org*), Istituto Americano di Normalizzazione, è un'organizzazione privata non a fini di lucro che produce standard industriali per gli Stati Uniti. È un membro dell'ISO e dell'IEC. Ha sede negli Stati Uniti.
- **IEEE** (*Institute of Electrical and Electronics Engineers*, *www.ieee.org*), istituto degli ingegneri elettrici ed elettronici, è un organismo privato di standardizzazione e sviluppo, a fine di lucro con sede negli Stati Uniti, la cui letteratura tecnica sta alla base di molti documenti ISO e IEC.
- **EIA/TIA** (*Electronic Industries Alliance/Telecommunications Industry Association*, *www.eia.org*), associazione di industrie elettroniche con sede negli Stati Uniti che emana raccomandazioni e standard industriali.
- **W3C** (*World Wide Web Consortium*, *www.w3.org*), non è un ente di standardizzazione ma un consorzio che si limita a sollecitare i produttori a seguire le sue raccomandazioni. È lo standard *de facto* per le tecnologie del web.

2 Tipi di reti

Una classificazione abbastanza semplice e sufficientemente operativa delle reti di calcolatori si basa sulla loro estensione fisica:

- **PAN** (*Personal Area Network*), sviluppate su aree circoscritte inferiori a 100 m, e concepite per connessioni senza cavo, soprattutto per dispositivi di uso personale come palmari e cellulari, per ottenere servizi da reti preesistenti, relativamente veloci (<10Mbit/s), conformi a IEEE.
- **LAN** (*Local Area Network*), sviluppate su aree limitate senza attraversamento di suolo pubblico per la condivisione di informazioni e periferiche. Esse non sono compatibili con gli standard di telecomunicazioni pubbliche perché progettate per essere veloci (>10Mbit/s) su aree normalmente inferiori al km. Conformi a ISO-IEEE-ANSI.
- **MAN** (*Metropolitan Area Network*), sviluppate su aree private e pubbliche ma limitate in estensione (qualche km), supportate quindi da infrastrutture pubbliche che garantiscono comunque alte velocità (>100 Mbit/s) conformi a ITU-T, ISO-IEEE-ANSI.
- **WAN** (*Wide Area Network*), reti geografiche che si basano su sistemi di

RFC e IETF

Le **RFC** (*Request For Comment*), sono il modo in cui si è creata la documentazione tecnica dapprima di Arpanet, poi di Internet. Attraverso vari enti informali come l'Internet Society o l'Internet Architecture Board, vari tecnici e studiosi contribuiscono alla scrittura di nuovi standard, o alla modifica di quelli in uso, attraverso i documenti RFC.
Una volta sottoposti al vaglio dell'**IETF** (*Internet Engineering Task Force*), le RFC possono diventare standard ufficiali per Internet.
Ogni RFC è pubblica e possiede un proprio URL (esempio, *http://www.rfc-editor.org/rfc/rfc5000.txt*).
Tra le più famose, **RFC 791 Internet Protocol** (1981) e **RFC 793 Transmission Control Protocol** (1981).
La prima RFC pubblicata fu invece **RFC 1 Host Software** (1969) di Steve Crocker.

Velocità delle reti

Uno dei parametri più importanti di una rete di calcolatori è la **velocità di trasmissione**, detta anche **bit rate**, cioè la quantità di bit che possono essere trasferiti in un dato intervallo di tempo attraverso un canale di comunicazione.
In alcuni contesti il bit rate viene anche chiamato portata o **banda**, oppure **larghezza di banda**, anche se questi due termini non sono propriamente corretti.
L'unità di misura adottata per la velocità di trasmissione è il **bit/s** (bit al secondo), a volte abbreviato in **bps**, da non confondere con Bps (byte al secondo).
I multipli del bit/s sono:
kilobit al secondo, **kbit/s**=10^3 bps
megabit al secondo, **Mbit/s** =10^6 bps
gigabit al secondo, **Gbit/s**=10^9 bps
terabit al secondo, **Tbit/s**=10^{12} bps

telecomunicazioni usati per la telefonia convenzionale o apparati dedicati di proprietà pubblica, tipicamente poco veloci ma di grandissima estensione geografica. Velocità tipicamente variabili tra i 64kbit/s e 10Mbit/s. Conformità ITU-T.

La struttura generale di una rete di calcolatori tipica comprende l'unione di più PAN, LAN, MAN e WAN, anche di architetture diverse, configurando un sistema informativo denominato **internetworking**.

3 Tipi di comunicazione

Una prima classificazione sulle tipologie di comunicazione adottate nelle reti di calcoltaori riguarda il tipo di occupazione del canale che hanno a disposizione mittente e destinatario. Si parla cioè di comunicazioni:

- **simplex**: quando l'informazione si veicola sempre in una sola direzione del canale (per esempio, reti televisive);
- **half duplex**: quando l'informazione si veicola in entrambe le direzioni del canale, ma non simultaneamente (per esempio, walkie talkie);
- **full duplex**: quando l'informazione si veicola in entrambe le direzioni anche simultaneamente (per esempio, autostrada).

Un'altra classe di modalità di comunicazione riguarda il tipo di inoltro delle informazioni sulla rete. In questi casi si parla di comunicazioni:

- **broadcast,** quando l'informazione viene inviata a tutti (o a molti) destinatari contemporaneamente e solo i destinatari interessati la trattengono;
- **punto-punto** (*point-to-point*) quando l'informazione è inviata all'unico destinatario connesso direttamente al mittente.

Quando invece si ha a che fare con mittenti e destinatari non direttamente connessi sulla rete (per esempio, reti WAN), subentrano due modi di comunicare molto importanti che determinano le due modalità tipiche con cui mittente e destinatario realizzano la loro interconnessione.

In questi casi si parla di comunicazione basata sui modelli:

- **orientato alla connessione**: mittente e destinatario suddividono il processo di comunicazione in tre fasi. Stabilire una connessione, comunicare attraverso quella connessione, rilasciare la connessione. È il modello usato, per esempio, quando si effettua una telefonata;
- **non orientato alla connessione**: in questo caso la comunicazione viene scomposta dal mittente in varie unità dotate ognuna delle informazioni necessarie per raggiungere il destinatario; una volta giunte a destinazione il ricevente dovrà ricostruirne il flusso ordinato e completo. Il modello è simile a quello usato dal servizio postale.

Dal punto di vista dell'infrastruttura utilizzata sulle reti geografiche, si possono adottare alcune modalità tipiche con cui realizzare la comunicazione, denominate **commutazioni**.

3.1 Commutazione di circuito

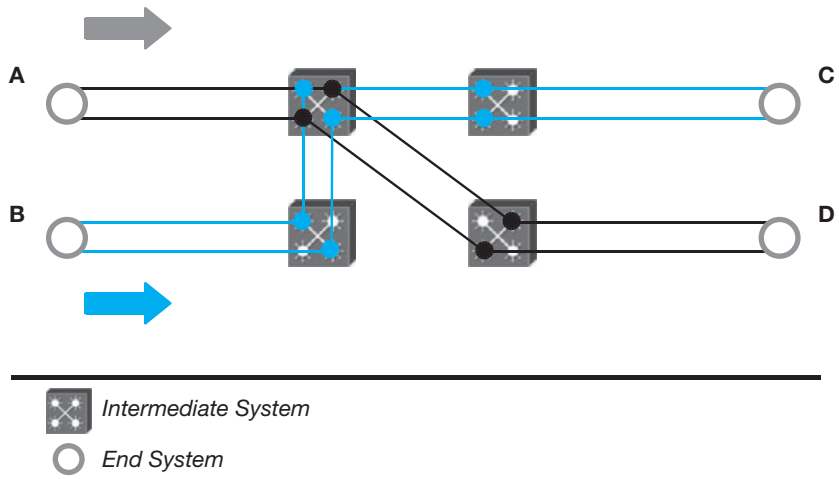
Si tratta di una connessione tra apparati intermedi concepita per la telefonia, ovvero per linee analogiche.

La connessione viene stabilita una volta per tutte prima dell'avvio della comunicazione e rilasciata al termine. Gli elementi intermedi, o centraline di commutazione, creano circuiti fisici punto-punto e sono di uso esclusivo dei soggetti che hanno instaurato la connessione.

La tariffazione è a tempo e non in base allo scambio di quantità di informazione.

Caratteristiche principali:

- bassa utilizzazione canale, impegnato anche in assenza di comunicazione;
- canale di trasmissione dati molto tollerante;
- poca tolleranza ai guasti sulla caduta della connessione;
- problemi nella gestione delle congestioni sui nodi più affollati;
- buona velocità;
- tariffazione a tempo.



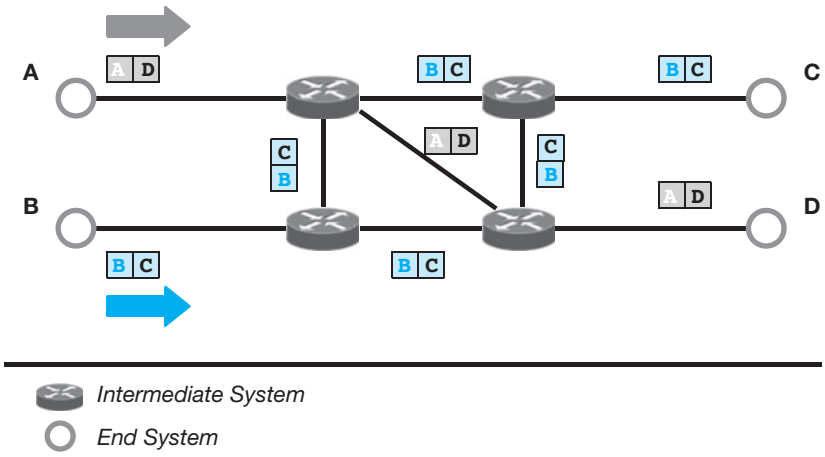
3.2 Commutazione di pacchetto

Il modo di trasferimento a commutazione di pacchetto è indicato per linee digitali, anche se si adatta alle linee analogiche. L'informazione viene suddivisa in tanti gruppi numerati contenenti indirizzo mittente e destinatario, pertanto ogni singolo gruppo di dati (pacchetto) circola nella rete verso la destinazione autonomamente e opportunamente inoltrato dai nodi intermedi. Non esistono circuiti privilegiati e i nodi intermedi hanno il compito di instradare ogni singolo pacchetto. Giunti a destinazione, i pacchetti verranno riordinati per ottenere l'informazione originale.

Caratteristiche principali:

- buona utilizzazione del canale, impegnato solo in presenza di comunicazione;
- buona tolleranza ai guasti;
- buona gestione delle congestioni;

- bassa velocità;
- tariffazione a quantità.

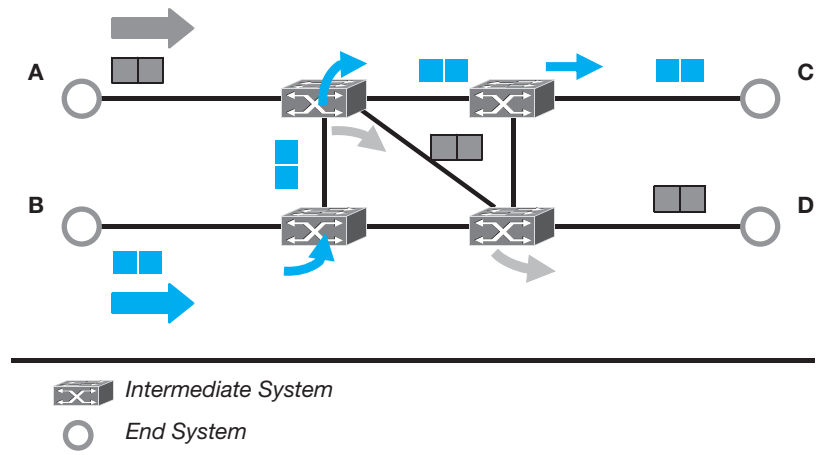


3.3 Commutazione di circuito virtuale

Modo di trasferimento che tende a riunire i pregi di entrambe le tecniche, soprattutto la velocità della commutazione di circuito e la robustezza della commutazione di pacchetto. Non vengono creati circuiti fisici tra i nodi ma gli elementi intermedi memorizzano la «strada» per i pacchetti facenti parte dello stesso flusso di informazione, con la possibilità di dirottarli in caso di problemi o congestioni. In pratica i nodi intermedi realizzano dei canali (circuiti virtuali) non fisici quindi gestibili in tempo reale. Come per la commutazione di circuito, la connessione tra i due estremi va stabilita prima dello scambio dei dati e rilasciata al termine.

Caratteristiche principali:

- buona utilizzazione del canale;
- buona tolleranza ai guasti;
- buona gestione delle congestioni;
- buona velocità;
- difficoltà di realizzazione.



ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1 Indicare i nomi degli studiosi più rappresentativi riguardo lo sviluppo delle reti WAN e delle reti LAN. Saper collocare cronologicamente l'evoluzione storica delle reti WAN e LAN.

2 Mettere a confronto i modelli dell'informatica centralizzata e dell'informatica distribuita.

3 Spiegare la differenza tra personal computer e terminale.

4 Elencare le sigle dei maggiori enti di standardizzazione operanti nel mondo delle reti.

5 Riportare la classificazione delle reti in base alla loro estensione fisica, indicandone i parametri base.

6 Confrontare i modelli di comunicazione simplex, half e full duplex.

7 Mettere a confronto le comunicazioni broadcast e punto-punto. Fornire esempi tratti dalla vita quotidiana.

8 Illustrare le modalità di comunicazione orientate e non orientate alla connessione.

9 Commentare il modo di comunicazione a commutazione di circuito ed elencarne le caratteristiche.

10 Commentare il modo di comunicazione a commutazione di pacchetto ed elencarne le caratteristiche.

11 Commentare il modo di comunicazione a commutazione di circuito virtuale ed elencarne le caratteristiche.

12 Elencare le tipologie di rete in base alla velocità trasmissiva tipica.

13 Calcolare quanto tempo serve per veicolare 1 Gb di informazioni in una rete con bitrate di 1 Gbit/s.

Requisiti avanzati

1 Indicare i nomi degli standard storicamente consolidati per reti WAN e per reti LAN.

2 DARPA e ARPANET: illustrarne significato e ruoli nella storia dell'evoluzione delle reti.

3 La documentazione tecnica: RFC. Commentare l'importanza di tali contributi.

4 Gli enti per Internet: W3C e IETF. Commentarne il ruolo e le attività.

5 Commentare il concetto di downsizing.

6 Enti governativi, non governativi e privati nel mondo delle reti: elencarne i nomi e le funzioni.

7 Illustrare il concetto di internetworking.

8 Comunicazione broadcast e punto-punto: collocarne l'utilizzazione nelle principali tipologie di reti.

9 Commentare i modi di comunicazione orientato e non orientato alla connessione in funzione della tolleranza ai guasti.

10 Spiegare perché nella commutazione di circuito il canale non è utilizzato in modo ottimale.

11 Spiegare come la commutazione di circuito virtuale sia il modo ottimale di comunicazione, ma tuttavia non sia effettivamente molto diffuso.

12 Calcolare quanto tempo serve per veicolare 1 Gb di informazioni su un percorso composto da due tratte di rete con bitrate da 1 Gbit/s e 1 Mbit/s.

13 Riportare la classificazione delle reti in base alla loro estensione fisica, indicandone i valori tipici d'ampiezza territoriale e di bitrate.

Modelli per le reti di calcolatori

B2

Dopo un primo periodo nel quale progettisti e costruttori hanno ideato e realizzato reti proponendo sistemi anche molto differenti tra loro cercando di imporre i propri punti di vista progettuali, ci si è resi conto che per affrontare razionalmente il problema era necessario un coordinamento generale che limitasse tali differenze.

A partire dagli anni '80 i vari soggetti operanti nel mondo delle reti hanno cominciato a fissare alcuni standard operativi per aumentare il grado di **internetworking**, che via via sono sfociati in due modelli tuttora considerati riferimenti stabili per il mondo delle reti di calcolatori.

Entrambi i modelli, **ISO-OSI** e **TCP/IP**, si basano sull'approccio a **livelli**, ovvero isolando le varie problematiche di una rete in sezioni omogenee e ben circoscritte, tra di loro in relazione con regole ben definite, realizzando di fatto la metodologia classica di risoluzione di un problema complesso denominato *divide et impera*.

Nel processo di standardizzazione, OSI è partito dai livelli più bassi – quelli più vicini all'hardware, ed è salito verso quelli alti – quelli più vicini all'utente, ricevendo gradimento e consensi abbastanza diffusi soprattutto per gli standard suggeriti circa i primi livelli. In questo caso lo standard *de jure* OSI ha raggiunto il suo obiettivo.

Per i livelli superiori gli standard hanno avuto maggiori difficoltà ad imporsi per l'alto impatto che la loro adozione ha sul software dei sistemi informatici e sui dispositivi utilizzati. In questo senso ha raggiunto migliori risultati lo standard *de facto* TCP/IP.

1 Modello ISO-OSI

L'**OSI** (*Open System Interconnections*) è un progetto di ampio respiro formulato dall'**ISO** (*International Standard Organization*) alla fine degli anni '70 con lo scopo principale di servire come modello di riferimento per le reti di calcolatori. Esso infatti doveva rappresentare la base comune per coordinare gli sforzi dei vari sviluppatori e produttori, per esempio standardizzando la terminologia e definendo quali sono le funzionalità di una rete. Per gestire la complessità del problema, l'OSI ha adottato un approccio a **livelli** (*layers*): l'intero problema della comunicazione tra due sistemi è stato suddiviso in un insieme di **sette** diverse problematiche, ciascuna delle quali deve essere affrontata tramite modalità specifiche.

OSI, spesso in accordo con IEEE, ha avuto inoltre il merito di coordinare tutte le attività di standardizzazione, scopo che è stato senza dubbio raggiunto.

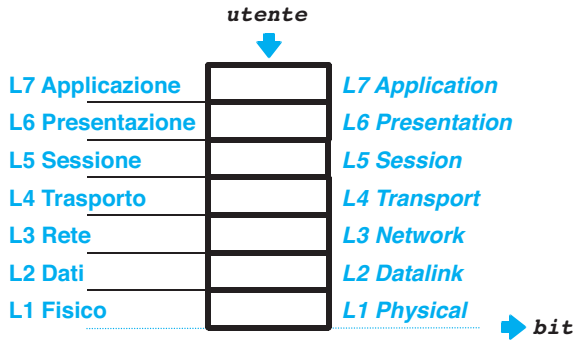
2 I livelli

I **livelli** previsti da OSI devono garantire modularità delle applicazioni, intercambiabilità, mascheramento dei servizi interni, interfacce di lavoro strutturate e protocolli comuni.

Generalmente i livelli sono numerati e i livelli più bassi sono vicini alla macchina, mentre i più alti sono vicini all'utente. L'insieme dei livelli è anche detto **pila** o **stack**.

I livelli di rete OSI sono sette:

- L1, **Livello Fisico** (*PH Layer* o *Physical*). Si occupa della trasmissione/ricezione materiale delle sequenze binarie sul mezzo di trasmissione. Prevede la definizione del supporto (cavo), dei connettori, delle grandezze fisiche coinvolte, ecc. È un livello prevalentemente hardware.
- L2, **Livello di Collegamento dei Dati** (*DH Layer* o *Datalink*). Si occupa della trasmissione (verso PH Layer) o ricezione (da NH Layer) di gruppi di bit detti frames, assicurandosi che i bit giungano a destinazione correttamente. Garantisce una comunicazione sostanzialmente affidabile tra sistemi direttamente connessi.
- L3, **Livello di Rete** (*NH Layer* o *Network*). Gestisce l'instradamento dei pacchetti sulla rete decidendo quali sistemi intermedi devono essere attraversati per giungere a destinazione. Deve possedere delle tabelle contenenti le regole per decidere i percorsi. È il livello più complesso.
- L4, **Livello di Trasporto** (*TH Layer* o *Transport*). Gestisce trasferimenti end-to-end (mittente-destinatario) e indipendenti dalla rete fisica sottostante; gestisce il congestionamento delle stazioni intermedie, frammenta e ricostruisce le informazioni se necessario.
- L5, **Livello di Sessione** (*SH Layer* o *Session*). Gestisce il dialogo e la sincronizzazione tra due programmi che comunicano tra di loro operando mutue esclusioni, sequenze coerenti di comandi, attese, ritrasmissioni su richiesta, modo di tariffazione, qualità del servizio, ecc.
- L6, **Livello di Presentazione** (*PH Layer* o *Presentation*). Traduce i formati e le sintassi dei dati – che su due sistemi in comunicazione possono essere differenti – in formati standard. Garantisce anche la riservatezza e la protezione dei dati tramite algoritmi di criptaggio e autenticazione.
- L7, **Livello di Applicazione** (*AH Layer* o *Application*). Applicativi standard di rete scritti da programmatori e utilizzati dagli utenti; per Internet un protocollo di L7 Applicazione è per esempio HTTP, per la realizzazione di programmi che navigano nel World Wide Web come i browser.

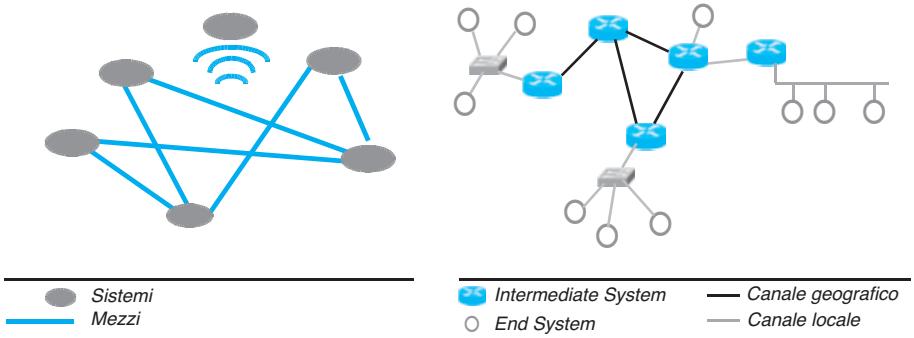


2.1 Terminologia

Le seguenti terminologie sono estremamente generiche e servono sostanzialmente per uniformare il linguaggio dei progettisti e quindi evitare ambiguità. Si tratta quindi di notazioni e concetti astratti detti in gergo *international bureaucrat speak* o linguaggio burocratico internazionale.

La terminologia di OSI comprende alcuni elementi base: i **sistemi** (elaboratori, terminali, apparati), i **mezzi fisici** (elementi di interconnessione), le **applicazioni** (programmi) e le **connessioni** (canali di comunicazione).

I *sistemi* comunicano tra di loro attraverso i *mezzi fisici* e sono anche i luoghi che ospitano le *applicazioni*; alcune *applicazioni* possono prevedere *connessioni* per comunicare con altre *applicazioni*. Le *connessioni* sono realizzate tramite i mezzi fisici.



In una rete i *sistemi* si distinguono in **End System** (ES) e **Intermediate System** (IS).

Gli ES sono i *sistemi* che ospitano ed eseguono le *applicazioni*. Sono detti anche host o end node; devono possedere l'implementazione di tutti i livelli OSI, cioè la pila completa.

Gli IS invece devono realizzare almeno i primi tre livelli OSI.

Implementano l'instradamento dei messaggi sulla rete. Sono detti anche router o gateway.

L'insieme di regole, convenzioni e formati che gestiscono lo scambio di informazione tra gli stessi livelli ma su sistemi diversi è detto **protocollo**. Ogni livello possiede un protocollo. Livelli uguali comunicano tramite il protocollo di quel livello.

I dati manipolabili dai protocolli sono detti in generale **pacchetti** e assumono il nome di **PDU** (*Protocol Data Unit*).

Lo scambio di informazioni operante tra livelli adiacenti sullo stesso sistema viene invece regolato dai **servizi** che trovano i punti di contatto tramite elementi codificati denominati **SAP** (*Service Access Point*).

3 Il pacchetto

I dati che circolano su una rete possiedono un formato generale ormai universalmente condiviso denominato **pacchetto**:

Il pacchetto consiste di tre parti fondamentali:

- una **intestazione** (*header*);
- una sequenza **dati** (*data* o *payload*);
- una **coda** (*tail* o *trailer*).

In alcuni casi il pacchetto è premesso da una sequenza di segnali denominata **preambolo** (*preamble*). Non sempre queste componenti sono presenti nel formato e a volte ne sono presenti altre.

L'**intestazione** contiene in genere informazioni di sincronizzazione per il ricevente, informazioni come indirizzi per l'instradamento del pacchetto, informazioni per la decodifica della parte dati.

I **dati** rappresentano l'oggetto della comunicazione e sono le informazioni propriamente dette.

La **coda** invece contiene quasi sempre informazioni per poter valutare l'integrità e la validità dell'intero pacchetto.

L'unità di misura del pacchetto e delle sue parti di solito è il **bit** o il **byte**, che nel linguaggio delle reti si preferisce chiamare a volte **ottetto**.

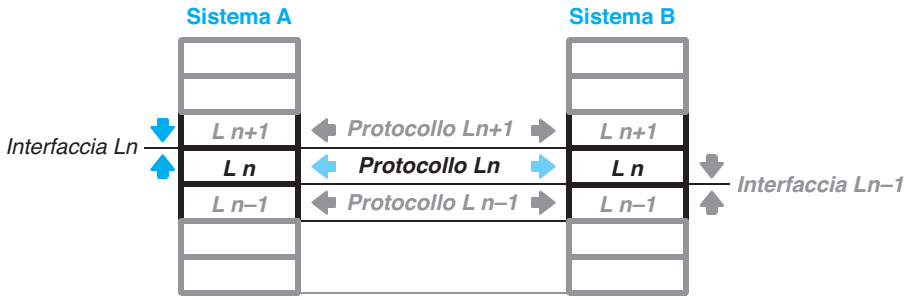
Il termine alternativo **frame** (o trama) è sinonimo di pacchetto, ma più spesso è riferito ad un particolare pacchetto, oggetto di una particolare classe di protocolli, quelli di livello L2 (Collegamento Dati).

3.1 Interfacce e protocolli

In un'architettura a livelli si hanno due tipi di attività: una tra i **livelli adiacenti** (in verticale) dello stesso sistema e una tra **livelli paritari** (in orizzontale) tra sistemi diversi.

In verticale i livelli adiacenti si scambiano i **servizi** (che cosa fare) attraverso una **interfaccia** (regole su come fare) cosicché un livello superiore (per esempio, Livello n) chiede Servizi al livello inferiore n-1 (e un livello inferiore n-1 fornisce Servizi al livello superiore n) tramite una interfaccia n-1.

In orizzontale i livelli paritari si scambiano dati (normalmente pacchetti o PDU) attraverso un **protocollo** (regole su come fare), cosicché un generico Livello n utilizza un Protocollo di livello n per scambiare dati **nPDU** con il Livello n equivalente di un altro sistema (che usa il medesimo Protocollo di livello n). Ogni livello possiede quindi Protocolli e Servizi peculiari.



ESEMPIO

Pacchetto e protocollo

Un protocollo di livello 6 Presentazione dispone che se nell'header di ampiezza un ottetto compare il valore 1, allora la parte dati è costituita da codici Ascii; se compare il valore è 2, allora la parte dati è un valore numerico intero senza segno little endian; se il valore è 3, la parte dati è come prima, ma big endian.

Interpretare correttamente i seguenti pacchetti di livello 6:

1) `01h,43h,68h,61h,8fh;`
2) `02h,0ah,01h;`
3) `03h,0ah,01h;`
4) `01h,32h,36h,36h.`

Siccome non viene citata la coda, i pacchetti ne sono sprovvisti. Quindi tutti i pacchetti hanno un header (il primo byte o ottetto) e la parte dati costituita dai rimanenti valori.

1) L'header di valore 1 impone che i dati siano codici Ascii, quindi 43h='C', 68h='i', 61h='a', 8fh='o', il pacchetto contiene la stringa "Ciao".
2) L'header di valore 2 impone che i dati siano un numero little endian, ovvero il primo byte è il meno significativo (0ah), il secondo il più significativo (01h). Pertanto abbiamo il numero esadecimale (010a)h, ovvero il pacchetto contiene il numero 266 (=10ah).
3) L'header di valore 3 impone che i dati siano un numero big endian, ovvero il primo byte è il più significativo (0ah), il secondo il meno significativo (01h). Il pacchetto contiene il numero 0a01h, cioè 2561.
4) L'header di valore 1 impone che i dati siano codici Ascii, quindi 32h='2', 36h='6', 36h='6', ovvero il pacchetto contiene la stringa numerica "266".

PROGRAMMA

Pacchetto e protocollo

Un protocollo di livello 7 (Applicazione) genera una stringa e un numero intero.

Scrivere un programma in linguaggio C per stampare sullo schermo i due pacchetti di livello 6 (Presentazione) se l'intestazione vale un byte 'A' o 'N' che indicano, rispettivamente, parte dati in Ascii e parte dati numerica in big endian.

Layout:

```
Livello7, digitare una stringa: Ciao
Livello7, digitare un numero: 258
Pkt1: 41h 43h 69h 61h 6fh
Pkt2: 4eh 01h 02h
```

Il codice in linguaggio C potrebbe essere il seguente:

```
00 #include <stdio.h>
01
02 int main(void)
03 {
04   char szInput[256];
05   int n;
06   int i;
07
08   printf("Livello7, digitare una stringa: ");
09   gets(szInput);
10   printf("Livello7, digitare un numero: ");
11   scanf("%d", &n);
12
13   printf("Pkt1: %xh",'A');
14   i=0;
15   while (szInput[i])
16   {
17     printf(" %02xh",szInput[i]);
18     i++;
19   }
20
21   printf("\nPkt2: %xh",'N');
22   printf(" %02xh %02xh",n/256, n%256);
23
24   return 0;
25 }
```

Notare a **riga 15** il ciclo che si interrompe al termine della stringa (sul terminatore nullo, ovvero quando il test è falso).

La stampa a **riga 17** `%02xh` viene interpretata come la stampa esadecimale minuscola del valore su due caratteri; se il primo manca, viene messo uno 0.

Infine, per ottenere la sequenza big endian (prima il byte più significativo), si estrae dal valore il byte più significativo dividendo per 256, mentre il byte meno significativo è il resto della divisione per 256.

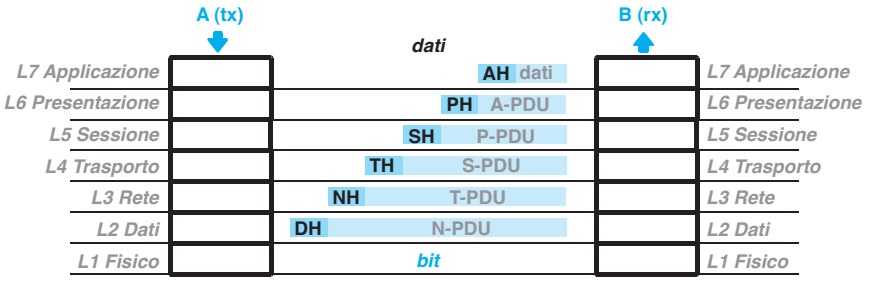
3.2 Imbustamento

Un concetto fondamentale per OSI, ma anche per tutte le altre architetture di rete a livelli, prende il nome di **imbustamento multiplo**.

Ogni livello, in spedizione (tx), genera il pacchetto, aggiungendo dati caratteristici del livello (nell'intestazione) tramite il proprio protocollo, e lo cede al livello sottostante.

Ogni livello, in ricezione (rx), acquisisce un pacchetto e, tramite la sua intestazione e il suo protocollo, riesce a trattarne i valori contenuti.

I sette livelli OSI descritti precedentemente generano pertanto pacchetti di protocollo (PDU) caratteristici per ogni livello.



Si noti come, per esempio, il livello 6 del sistema A abbia creato un pacchetto P-PDU (Presentation PDU) costituito da una intestazione PH e una parte dati A-PDU che è il pacchetto del livello 7 (Application PDU).

L'intestazione PH è anche detta **busta** (in questo caso di livello 6).

Se in trasmissione ogni livello si comporta in questo modo, sul livello 1 (fisico) del sistema A avremo un pacchetto di bit costituito da 7 buste, ognuna contenente pacchetti conformi al protocollo del rispettivo livello.

In ricezione il sistema B apre la busta di livello 1 e la usa per estrarre i dati del livello 1 tramite un protocollo di livello 1; ora questi dati contengono la busta di livello 2, usata per estrarre i dati di livello 2 e interpretarli tramite un protocollo di livello 2; e così via. Ogni livello apre la propria busta e sa come interpretare i dati in essa contenuti.

ESEMPIO

Busta di livello 2 Datalink

Un protocollo di livello 2 Datalink (collegamento dati) che trasporta codici Ascii prevede un'intestazione di un byte: se il valore è 'T', allora il pacchetto ha una coda di un byte che contiene il n. di ottetti dell'intero pacchetto; se l'intestazione vale '0', allora il pacchetto non ha coda.

Interpretare correttamente i seguenti pacchetti di livello 2:

1) `54h,43h,68h,61h,8fh,36h;`
2) `30h,6dh,61h,72h,65h;`
3) `30h,6dh,61h,72h,65h,3fh;`
4) `54h,68h,65h,6ch,6ch,6fh,35h.`

1) La busta del pacchetto (cioè la sua intestazione) vale 'T'=54h. Allora l'ultimo byte è la coda: 36h='6'. Siccome i byte del pacchetto sono proprio 6, il pacchetto è integro.

   La parte dati vale: 43h='C', 68h='i', 61h='a', 8fh='o', cioè la stringa "Ciao".

2) La busta del pacchetto vale '0'=30h, quindi il pacchetto non ha coda e non è possibile verificarne l'integrità. La parte dati vale: 6dh='m',61h='a', 72h='r',65h='e', cioè la stringa "mare".

3) La busta del pacchetto vale '0'=30h, quindi il pacchetto non ha coda e non è possibile verificarne l'integrità. La parte dati vale: 6dh='m',61h='a', 72h='r',65h='e', 3fh='?' cioè la stringa "mare?"

4) La busta del pacchetto vale 'T'=54h. Allora l'ultimo byte è la coda: 35h='5'. Siccome i byte del pacchetto sono 7, il pacchetto non è integro.

   La parte dati (non sicura) vale: 68h='h',65h='e', 6ch='l',6ch='l',6fh='o', cioè la stringa "hello".

PROGRAMMA

Busta di livello 2 Datalink

Un protocollo di livello 3 (Network) consegna una stringa a un livello 2 (Datalink) che prevede una coda di un ottetto contenente il n. di ottetti del pacchetto.

Prendere in input una stringa e stampare il pacchetto di livello 2 appropriato.

Layout:

Livello3, digitare una stringa: Ciao mondo
Pkt: 43h 69h 61h 6fh 20h 6dh 6fh 6eh 64h 6fh 0bh

Il codice in linguaggio C potrebbe essere il seguente:

```
00  #include <stdio.h>
01
02  int main(void)
03  {
04    char szInput[256];
05    int i;
06
07    printf("Livello7, digitare una stringa: ");
08    gets(szInput);
09
10    printf("Pkt: ");
11
12    i=0;
13    while (szInput[i])
14    {
15     printf(" %02xh",szInput[i]);
16     i++;
17    }
18
19    printf(" %02xh",i+1);
20
21    return 0;
22  }
```

La variabile di conteggio `i` per la stampa serve anche a preparare la coda, che viene stampata per ultima in esadecimale a **riga 19**. (Nell'esempio il pacchetto è formato da 11 ottetti, 11 = 0bh).

3.3 Indirizzi e frammentazione

Molto spesso l'intestazione di un livello contiene un identificativo numerico denominato **indirizzo**, a volte detto anche SAP.

L'indirizzo ha lo scopo di identificare senza ambiguità il mittente e il destinatario del pacchetto: spesso nell'intestazione abbiamo quindi due indirizzi. L'indirizzo non riguarda necessariamente l'identificazione di un sistema nella rete (per esempio, un personal computer), ma anche l'identificativo di un programma o di una procedura.

I livelli che contengono indirizzi, solitamente, sono il livello 2 di Collegamento Dati, il livello 3 di Rete e il livello 4 di Trasporto.

Un altro concetto fondamentale riguarda la **frammentazione** dei pacchetti.

Se il pacchetto di un certo livello n (un nPDU) diviene troppo grande rispetto ad una trasmissione senza errori per quel livello, lo stesso livello frammenterà la nPDU in tanti pacchetti di minori dimensioni per consentire la trasmissione. Ogni pacchettino conterrà naturalmente le informazioni necessarie per poter ricomporre il pacchetto originale, come ad esempio la quantità totale di frammenti e il numero di frammento su ogni frammento.

Il limite di dimensione massima di una PDU per un determinato livello è detto **MTU** (*Maximum Transmission Unit*).

La frammentazione avviene soprattutto a livello 2 Collegamento e a livello 4 Trasporto, anche se può avvenire a qualsiasi livello nelle numerose architetture di rete operanti su un sistema complesso.

ESEMPIO

Frammentazione

Un protocollo di livello 4 Transport (trasporto) con parte dati Ascii, prevede un header di due byte, di cui primo può valere 'L' o 'F'. Se vale 'L', allora il secondo byte contiene il numero di frammenti totale; se vale 'F', allora il secondo byte indica il numero di sequenza del frammento.

Interpretare correttamente i seguenti pacchetti di livello 4 se l'MTU vale 6:

1) `46h,00h,42h,75h,6fh,6eh;`
2) `4ch,04h;`
3) `46h,03h,6fh,6eh,64h,6fh;`
4) `54h,68h,65h,6ch,6ch,6fh,35h;`
5) `46h,01h,67h,69h,6fh,72h;`
6) `46h,02h,6eh,6fh,20h,6dh.`

1) Siccome il primo byte dell'header vale 'F'=46h, si tratta di un frammento, proprio il primo (00h). Il suo contenuto: 42h='B',75h='u',6fh='o',6eh='n', cioè la stringa "Buon".
2) Siccome il primo byte dell'header vale 'L'=4ch, il secondo byte contiene il numero totale dei pacchetti frammentati, ovvero 4 (=04h). Il pacchetto non ha parte dati.
3) Siccome il primo byte dell'header vale 'F'=46h, si tratta di un frammento, il quarto (03h). Il suo contenuto: 6fh='o', 6eh='n',64h='d',6fh='o', cioè la stringa "ondo".
4) Si vede subito che il pacchetto va scartato: la sua lunghezza (7 ottetti) supera l'MTU che vale 6. Inoltre l'header non è a norma.
5) Siccome il primo byte dell'header vale 'F'=46h, si tratta di un frammento, il secondo (01h). Il suo contenuto: 67h='g',69h='i',6fh='o',72h='r', cioè la stringa "gior".
6) Siccome il primo byte dell'header vale 'F'=46h, si tratta di un frammento, il terzo (02h). Il suo contenuto: 6eh='n',6fh='o',20h=spazio,6dh='m', cioè la stringa "no m".

In definitiva i pacchetti ricompattati sono 4, che ricostruiti in ordine danno origine alla stringa "Buongiorno mondo".

PROGRAMMA

Frammentazione

Un protocollo di livello 4 (Transport) che trasporta dati in codici Ascii, ha un'intestazione di due byte.

Il primo byte può valere 'F' se si tratta di un pacchetto frammentato; vale 'L' se contiene il numero di frammenti totali.

Il secondo byte contiene il n. di sequenza (pacchetti di tipo 'F') o il n. totale dei frammenti (pacchetti di tipo 'L').

Scrivere un programma in linguaggio C che acquisisce in input una stringa e un numero che rappresenta l'MTU (di livello 4); frammentare i dati rappresentati dalla stringa e stampare a schermo i pacchetti.

Layout:

```
Livello4, digitare una stringa: Ciao mondo
Livello4, digitare l'MTU: 5

Pkt1: 46h 00h 43h 69h 61h
Pkt2: 46h 01h 6fh 20h 6dh
Pkt3: 46h 02h 6fh 6eh 64h
Pkt4: 46h 03h 6fh
Pkt5: 4ch 04h
```

Il codice in linguaggio C potrebbe essere il seguente:

```
00 #include <stdio.h>
01
02 int main(void)
03 {
04   char szInput[256];
05   int iMtu;
06   int i;
07   int j;
08
09   printf("Livello4, digitare una stringa: ");
10   gets(szInput);
11   printf("Livello4, digitare l'MTU: ");
12   scanf("%d", &iMtu);
13
14   j=0;
15   i=0;
16   while (szInput[i])
17   {
18     if ((i % (iMtu-2))==0)
19     {
20       printf("\nPkt%d: %02xh 02xh",j+1,'F',j);
21       j++;
22     }
23     printf(" %02xh",szInput[i]);
24     i++;
25   }
26
27  printf("\nPkt%d: %02xh %02xh",j+1,'L',j);
28
29   return 0;
30 }
```

La variabile j conta i pacchetti.

Dato l'MTU (esempio, 5), ogni pacchetto ha 5-2=3 byte di dati e 2 di intestazione fissa. Quindi, ogni 3 caratteri della stringa si ha un pacchetto di tipo 'F' (vedi **riga 18**).

In fondo si stampa l'ultimo pacchetto, di tipo 'L' contenente il numero di pacchetti frammentati (**riga 27**).

4 Modello TCP/IP

Nella prima metà degli anni ’70, la Defence Advanced Research Project Agency (DARPA) degli Stati Uniti dimostrò interesse per lo sviluppo di una rete a commutazione di pacchetto per l’interconnessione di calcolatori eterogenei, da utilizzarsi come mezzo di comunicazione tra istituzioni. DARPA finanziò a tal scopo l’Università di Stanford e l’azienda BBN (Bolt, Beranek and Newman Inc.) affinché sviluppassero un insieme di protocolli di comunicazione. Verso la fine degli anni ’70, tale sforzo portò al completamento dell’**Internet Protocol Suite**, di cui i due protocolli più noti sono il **TCP** (*Transmission Control Protocol*) e l’**IP** (*Internet Protocol*). Questi protocolli furono utilizzati da un gruppo di ricercatori per la rete ARPANET e ottennero un elevato successo, anche perché utilizzati fin dall’inizio pubblicamente e quindi utilizzabili gratuitamente da chiunque.

Il nome più preciso di questa rete è Internet Protocol Suite, anche se comunemente si fa riferimento ad essa con la sigla **TCP/IP**.

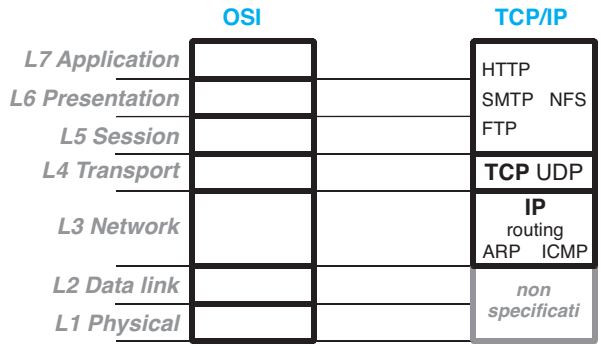
I protocolli appartenenti a questa architettura sono specificati tramite standard pubblici che si chiamano **RFC** (*Request For Comments*) facilmente reperibili sulla rete Internet. Generalmente la rete TCP/IP si ispira ad un modello **client/server**, in cui i due capi di una comunicazione assumono ruoli differenti, l’uno il ruolo del fornitore di informazioni (server), l’altro il ruolo di richiedente di informazioni (client).

Come in OSI, i sistemi di una rete TCP/IP si differenziano tra **host** (l’equivalente dell’End System di OSI) e **gateway** (l’equivalente di Intermediate System di OSI).

Ad oggi TCP/IP è lo standard di fatto per le reti geografiche (soppiantando interamente OSI) e di fondamentale supporto anche per le reti locali.

5 I livelli

La figura mostra l’architettura dell’Internet Protocol Suite e la paragona con il modello di riferimento ISO-OSI, riportando i protocolli più famosi operanti nei livelli della suite TCP/IP.



La natura di **rete disconnessa** di TCP/IP è data dal **livello 3 di Rete IP** che è basato su un protocollo disconnesso (non orientato alla connessione).

La parte orientata alla connessione della suite TCP/IP la si ritrova nel **livel-**

ICANN e IPv6

Gli indirizzi IP per la rete Internet, dovendo essere univoci a livello mondiale, vengono assegnati tramite un organismo controllato dal ministero del commercio degli Stati Uniti, l’**ICANN** (*Internet Corporation for Assigned Names and Numbers*), il quale a sua volta delega lo **IANA** (*Internet Assigned Numbers Authority*), il quale a sua volta delega organismi locali denominati **RIR** (*Regional Internet Registry*), uno per continente che, a loro volta, delegano l’effettiva distribuzione a organizzazioni nazionali denominate **LIR** (*Local Internet Registry*).

Il RIR per l’Europa si chiama **RIPE** (*Réseaux IP Européen*) e ha sede in Olanda. I LIR italiani sono circa 600 (2012), tra aziende e istituti, tra cui i grandi operatori telefonici nazionali come Fastweb, Telecom, Tiscali e Vodafone.

Siccome la distribuzione degli indirizzi Ip è messa in crisi dalla notevole domanda di indirizzi (la cosiddetta **saturazione di IPv4**, avvenuta ufficialmente il 3 febbraio 2011 con una dichiarazione pubblica dell’IANA), esiste uno standard per nuovi indirizzi IP espressi su 128 bit anziché 32 denominato **IPv6**.

Questo nuovo standard, già operativo in alcuni stati e presso le principali aziende operanti sul web (6 giugno 2012) risolve alla radice il problema della scarsità degli indirizzi IP, così come sostiene Andrew S. Tanenbaum: «Se l’intero pianeta, terraferma e acqua, fosse coperto di computer, IPv6 permetterebbe di utilizzare $7x10^{23}$ indirizzi IP per metro quadrato [...] questo numero è più grande del numero di Avogadro.»

lo 4 di Trasporto, attraverso il protocollo **TCP** che è un protocollo connesso (mentre il protocollo **UDP** fornisce, allo stesso livello 4, servizi disconnessi).

I livelli L5 Sessione e L6 Presentazione non esistono in TCP/IP e sono demandati ai protocolli di livello superiore (ovvero al livello L7 Applicazione), pertanto non esiste uno standard univoco per gestire le problematiche di livello Sessione e di livello Presentazione.

L'architettura di rete TCP/IP non specifica i livelli 1 e 2 della rete, ma utilizza quelli normalmente disponibili e conformi agli standard IEEE/OSI.

Per esempio, nell'ambito delle reti locali LAN, opera sul protocollo di livello 2 di Collegamento denominato LLC (quello di Ethernet/IEEE802.3); nell'ambito delle reti geografiche WAN, TCP/IP opera indifferentemente su PPP, HDLC, X.25, Frame Relay, ATM, che sono altrettanti protocolli di livello 2. Esistono anche realizzazioni per reti poco diffuse o proprietarie, come ad esempio AIX.25, la rete a pacchetto dei radioamatori, o per reti industriali tipo ModBus.

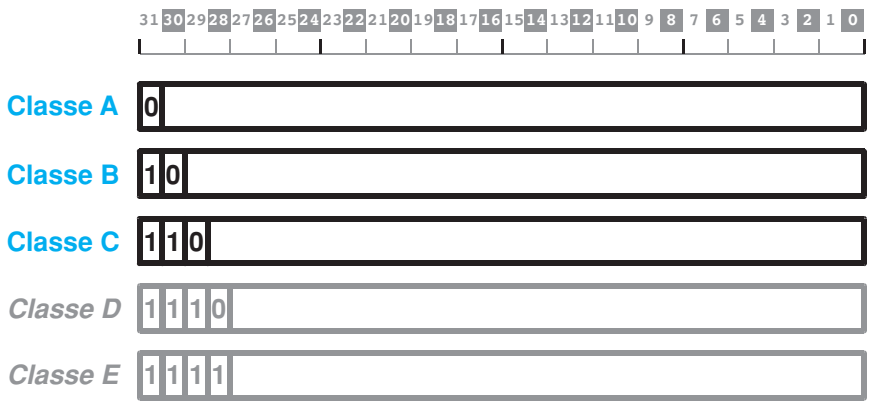
6 Indirizzi IP e porte TCP

Di notevole importanza sono i SAP di TCP/IP, ovvero gli identificativi che abbiamo chiamato indirizzi.

Per individuare un host su una rete TCP/IP si usano gli indirizzi di livello 3 Network, o **indirizzi IP**. Ogni host quindi deve possedere almeno un indirizzo IP univoco sulla sua rete. Un gateway ne possiede più di uno, dato che deve avere almeno due connessioni sulla rete TCP/IP (un indirizzo IP per ogni connessione). Se consideriamo la rete TCP/IP mondiale (Internet), ogni host su Internet deve avere un indirizzo IP univoco su scala mondiale.

D'altra parte esistono reti TCP/IP locali (per esempio, all'interno di un'azienda o di una scuola): in questo caso ogni host deve avere un indirizzo IP univoco all'interno di quell'azienda o scuola.

L'indirizzo IP è un numero intero senza segno espresso con 32 bit (detto **IPv4**), nel quale i bit MSB hanno un significato speciale che classifica gli indirizzi IP in cinque categorie o **classi di indirizzi** (A, B, C, D e E). Possono quindi esistere, teoricamente, 2^{32} indirizzi IP differenti: cioè la rete TCP/IP più ampia possibile può gestire al massimo $2^{32} = 4\,294\,967\,296$ host, circa 4 miliardi di host.



Maschere di bit: impostare un bit a 1

Per modificare un singolo bit di un valore (per esempio, un byte) bisogna adoperare un operando speciale detto **maschera** (di bit). Per esempio, per impostare a 1 il bit di peso 3 in un byte, bisogna costruire la maschera `00001000`, cioè una sequenza di 8 bit a zero tranne quello relativo al peso desiderato.

In linguaggio C questo si ottiene con l'operatore di scorrimento a sinistra <<:

```
unsigned char m = 1;
m = m << 3;
```

Ora m vale 8, cioè (00001000)b. Esempio, per impostare a 1 il bit 3 di un numero dato n = 18, si usa l'operatore logico OR con la maschera opportuna: n OR m = 18 OR 8 = 26

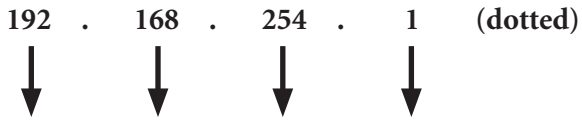
```
( 18) 0 0 0 1 0 0 1 0  OR
(  8) 0 0 0 0 1 0 0 0
      ---------------
( 26) 0 0 0 1 1 0 1 0
```

In linguaggio C:

```
unsigned char m = 1,
n=18;
m = m << 3;
n = n | m;
```

Il modo più diffuso per rappresentare un indirizzo IP è detto **dotted decimal** e riporta i 32 bit dell'indirizzo su quattro byte ordinati, separati da un punto.

Per esempio, l'indirizzo IP 3232300545 (C0A8FE01)h può essere espresso:



(11000000.10101000.11111110.00000001)b

Infatti:

(11000000)b = (C0)h = 192;
(10101000)b = (A8)h = 168;
(11111110)b = (FE)h = 254;
(00000001)b = (01)h = 1.

ESEMPIO

La classe di un indirizzo Ip

Dato l'indirizzo IP 192.168.254.1, determinarne la classe di appartenenza.

Ragionando in base alla figura, si nota che per determinare la classe di un indirizzo IP basta esaminare il byte più significativo della sua notazione dotted (cioè 192).

Quindi si trasforma 192 in binario: 192 = (11000000)b, eventualmente riempiendolo al byte, cioè ottenendo una rappresentazione su 8 bit con i bit mancanti a sinistra impostati a 0 (in questo caso non serve).

Confrontando i tre bit MSB della rappresentazione (110)b con i valori riportati nella figura, si deduce che l'indirizzo IP 192.168.254.1 è di classe C.

Per comodità si è soliti associare ad un indirizzo IP una stringa testuale, detta **nome** dell'host. Solo nel caso della rete TCP/IP mondiale, cioè Internet, il nome dell'host è detto **nome di dominio**, che ha la forma di varie stringhe separate da punti. Per esempio, su una rete TCP/IP locale, un nome di host potrebbe essere *pc-casa*, mentre un nome di dominio su Internet è *google.it*. Così come gli indirizzi IP, i nomi degli host devono essere univoci[NB] all'interno della propria rete TCP/IP.

[NB] Dovrebbe essere così, ma in realtà non è così, altrimenti un sito come Google, con nome di dominio *www.google.it*, coinciderebbe con una unica macchina a livello mondiale, cosa impossibile dato l'enorme carico di richieste che ha il sito.
Cfr. *Applicazioni: ping.*

PROGRAMMA

La classe di un indirizzo IP

Scrivere un programma in linguaggio C che genera un indirizzo IP casuale, quindi stampi sullo schermo se si tratta di un indirizzo di classe C o meno.

Layout:

```
L'indirizzo IP 194.3.7.8 è di classe C
```

Il codice in linguaggio C potrebbe essere il seguente:

```
00 #include <stdio.h>
01 #include <stdlib.h> // per rand()
02 #include <time.h>  // per time()
03
04 #define IPMASK_C (0xE0) //E0h=11100000b
05 #define IPCLASS_C (0xC0)//C0h=11000000b
06
07 int main(void)
08 {
09   int i;
10   unsigned char byt0,byt1,byt2,byt3;
11
12   srand(time(0));
13   i = 1 + rand() % 254; // Generazione
     numero da 1 a 255
14   byt0 = (unsigned char) i;
15
16   i = 1 + rand() % 254;
17   byt1 = (unsigned char) i;
18
19   i = 1 + rand() % 254;
20   byt2 = (unsigned char) i;
21
22   i = 1 + rand() % 254;
23   byt3 = (unsigned char) i;
24
25   printf("L'indirizzo IP %d.%d.%d.%d",
     byt0,byt1,byt2,byt3);
26
27   if ((byt0 & IPMASK_C) == IPCLASS_C)
28   {
29    printf(" è di classe C");
30   }
31   else
32   {
33    printf(" non è di classe C");
34   }
35
36   return 0;
37 }
```

PROGRAMMA

I quattro byte dell'indirizzo IP sono dichiarati `unsigned char` (**riga 10**).

L'estrazione pseudocasuale di ogni byte dell'indirizzo IP usa la funzione `rand()` che si trova nella libreria `stdlib`, quindi l'intestazione della libreria va riportata in testa al codice, come l'intestazione della libreria `time` che serve per la funzione `time()` usata con `srand()`, come in **riga 12**. In questo modo la generazione dei numeri pseudocasuali varia ad ogni avvio del programma. Si nota che viene escluso dall'estrazione il numero `0`.

Per ogni estrazione viene fatta un'assegnazione usando l'operatore di cast come in **riga 14** `byt0 = (unsigned char) i;` in modo che i contenitori siano coerenti con il contenuto (che deve essere un byte).

Per determinare se un indirizzo IP è di classe C (o meno) si sfrutta la proprietà dell'operatore logico AND (in linguaggio C è il carattere `&`): un bit a `1` si ottiene solo se entrambi i bit corrispondenti valgono `1`.

Definito quindi un valore opportuno detto **maschera**, nel nostro caso un numero con i primi tre bit a `1` e i rimanenti a `0`, e dato un valore da esaminare, si possono isolare i suoi primi tre bit, imponendo a `0` i rimanenti calcolandone l'AND con la maschera.

Scelto come maschera il valore `IPMASK_C (0xE0)`, che in binario vale 11100000, si isolano i primi tre bit del primo byte dell'indirizzo IP con l'operatore logico AND, come a **riga 27** `(byt0 & IPMASK_C)` e il risultato viene quindi confrontato con il valore caratteristico dei bit delle reti di classe C, che vale `11000000b`, cioè `C0h`, definito come `IPCLASS_C`.

Maschere di bit: impostare un bit a 0

Per impostare un bit a zero di un valore dato si usa l'operatore logico AND, e la maschera va invertita: una sequenza di bit tutti a uno tranne quello del peso interessato, che deve valere zero.

Per esempio, per impostare a 0 il bit di peso 3 in un byte, bisogna costruire la solita maschera `00001000`, quindi invertirne i bit con l'operatore logico NOT.

In linguaggio C:

```
unsigned char m = 1;
m = m << 3;
m = ~ m;
```

Ora m vale 247, cioè (11110111)b.

Esempio, per impostare a 0 il bit 3 di un numero dato n = 28, si usa l'operatore logico AND con la maschera opportuna: n AND m = 28 AND 247 = 20

```
( 28) 0 0 0 1 1 1 0 0
                    AND
(247) 1 1 1 1 0 1 1 1
      ---------------
( 20) 0 0 0 1 0 1 0 0
```

In linguaggio C:

```
unsigned char m = 1,
n=28;
m = m << 3;
m = ~ m;
n = n & m;
```

L'indirizzo di livello 4 Trasport, o indirizzo TCP, è detto invece **porta**.

Dopo aver individuato un determinato host su una rete TCP/IP tramite l'indirizzo IP, bisogna individuare, su quello stesso host, l'applicativo di livello 7 (in pratica, il programma) a cui l'informazione è destinata tra le varie applicazioni eventualmente operanti su quell'host. Ecco allora che nei pacchetti TCP/IP, oltre agli indirizzi IP, è necessario specificare gli indirizzi di porta TCP: ad ogni porta TCP corrisponde un programma a cui «appartiene» quel pacchetto.

Le porte TCP sono numeri interi senza segno espressi con 16 bit, per un totale di $2^{16} = 65536$ numeri di porta possibili.

Anche in questo caso esiste una classificazione delle porte TCP: da 0 a 1023 si trovano le **porte note** (*well known ports*, riservate ad applicativi server); da 1024 a 49151 si trovano le **porte registrate** (per i programmi applicativi degli utenti); da 49152 a 65535 si trovano le **porte dinamiche** (riservate al sistema operativo).

Maschere di bit: confronti

Per verificare se in un certo valore alcuni bit di peso noto sono impostati in una determinata combinazione, è sufficiente costruire una maschera con bit indicati, lasciando i rimanenti a 0 e calcolare l'AND tra la maschera e il valore da esaminare: se il risultato coincide con la maschera, allora il confronto è positivo.

Per esempio, per verificare se i due bit più significativi di un byte valgono 01, bisogna costruire la maschera `01000000`, con i due bit più significativi uguali al valore da esaminare. Quindi si effettua il test.

In linguaggio C:

```
#define MASK (0x80)
//10000000b
unsigned char n=18;

if ((MASK & n) = MASK)
{
  // i bit coincidono
}
else
{
  // i bit non coincidono
}
```

ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1 Elencare i sette livelli OSI e indicarne le funzionalità base. Riportare un diagramma dei livelli.

2 Elencare e descrivere i ruoli della terminologia base del modello OSI. Riportare un diagramma esplicativo.

3 Descrivere e commentare il concetto di pacchetto, le sue parti e la sua unità di misura.

4 Commentare la nozione di protocollo e di imbustamento multiplo.

5 Commentare la nozione di indirizzo per un pacchetto di rete.

6 Mettere in relazione i concetti di frammentazione e MTU.

7 Collocare i protocolli TCP e IP nello schema a livelli di OSI. Indicare i corrispettivi dei 7 livelli OSI per lo standard TCP/IP.

8 Illustrare dimensione, forma, notazione e classi degli indirizzi IPv4.

9 Spiegare come si stabilisce che la rete TCP/IP è una rete disconnessa (ovvero non orientata alla connessione).

10 Descrivere come è possibile determinare se un valore dato contiene una determinata sequenza di bit.

11 Descrivere cosa si intende per saturazione di IPv4.

12 Fornire un esempio di indirizzo IP dotted valido e uno non valido. Commentare la locuzione «nome di dominio» e fornirne un esempio.

13 Protocollo TCP: natura del protocollo; indirizzi TCP, formato e classificazioni.

Requisiti avanzati

1 Collocare le seguenti attività nei rispettivi livelli OSI: immissione della password per un collegamento Wi-Fi, frammentazione di un pacchetto, chat su Twitter, calcolo della spesa di una SIM durante la navigazione cellulare, connessione di un modem alla presa telefonica.

2 Commentare gli acronimi PDU e SAP.

3 Livelli paritari e livelli adiacenti in OSI: commentare i due concetti.

4 Elencare la terminologia TCP/IP dei sistemi rispetto alla terminologia OSI.

5 Descrivere il meccanismo di distribuzione mondiale degli indirizzi IP per la rete Internet.

6 Gateway, host e indirizzi IP: mettere in relazione i tre concetti.

7 Descrivere come si distinguono indirizzi di classe A, B e C.

8 Mostrare come impostare ad 1 un bit di peso dato in un valore assegnato, usando il linguaggio C.

9 Spiegare il concetto generale di maschera di bit e una sua applicazione per gli indirizzi IP.

10 Illustrare il concetto di univocità degli indirizzi IP nelle reti TCP/IP.

11 Elencare qualche protocollo di livello 2 (Collegamento Dati) su cui opera TCP/IP.

12 Elencare la sigla di qualche protocollo di livello 7 (Applicazione) operante in TCP/IP.

13 Protocolli TCP e UDP: tipi di protocolli e porte.

ESERCIZI PER LA VERIFICA SCRITTA

I prerequisiti richiesti per affrontare questi esercizi sono la conoscenza del sistema di numerazione in base 16 (esadecimale) e relative conversioni, anche con applicativi (esempio, Calc).

È necessaria la consultazione di una tabella di codici Ascii.

Pacchetti e protocolli

1 Un protocollo di livello 6 (Presentazione) prevede che se nell'header di ampiezza un byte compare il valore 'A', allora la parte dati è costituita da codici Ascii; se compare il valore 'L', allora la parte dati è un valore numerico intero senza segno little endian; se il valore è 'B', la parte dati è come prima, ma big endian.
Interpretare correttamente i seguenti pacchetti di livello 6:

1. `41h,50h,69h,70h,70h,6fh`;
2. `4ch,06h,10h`;
3. `41h,73h,65h,69h,20h,75h,6eh,6fh,20h,7ah,65h,72h,6fh`;
4. `42h,06h,10h`;
5. `41h,36h,31h,30h`.

2 Un protocollo di livello 6 (Presentazione) trasporta dati numerici little endian oppure dati Ascii a seconda dell'intestazione di un byte che può valere 'N' o 'A'.
Riportare il pacchetto che contiene il proprio nome e cognome.

3 Un protocollo di livello 6 (Presentazione) trasporta solo dati numerici little endian.
Riportare il pacchetto che contiene la data di oggi nel formato aammgg.

4 Un protocollo di livello 6 (Presentazione) trasporta solo dati Ascii.
Riportare il pacchetto che contiene il giorno odierno della settimana.

5 Un protocollo di livello 6 (Presentazione) prevede che se nell'header di ampiezza un byte compare il valore 0, allora la parte dati è costituita da codici Ascii; se compare il valore 1, allora la parte dati è un valore numerico intero senza segno little endian; se il valore è 2, la parte dati è come prima, ma big endian.
Interpretare correttamente i seguenti pacchetti di livello 6:

a `00h,35h,35h,68h,41h,41h,68h`;
b `01h,AAh,55h`;
c `00h,50h,69h,70h,70h,6fh`;
d `02h,55h,AAh`;
e `00h,50h,6ch,75h,74h,6fh`.

6 Un protocollo di livello 6 (Presentazione) trasporta solo dati numerici big endian.
Riportare il pacchetto che contiene la data dell'esaurimento degli indirizzi IP nel formato ggmmaa.

7 Un protocollo di livello 6 (Presentazione) trasporta dati numerici little endian oppure dati Ascii a seconda dell'intestazione di un byte che può valere 'N' o 'A'.
Riportare tre pacchetti che contengono la propria città di nascita e la propria età.

8 Un protocollo di livello 6 (Presentazione) prevede che se nell'header di ampiezza un byte compare il valore 'A', allora la parte dati è costituita da codici Ascii maiuscoli; se compare il valore è 'a', allora la parte dati è costituita da codici Ascii minuscoli.
Riportare i pacchetti che contengono il proprio codice fiscale e l'istruzione del linguaggio C che stampa sullo schermo.

9 Un protocollo di livello 6 (Presentazione) prevede che se nell'header di ampiezza un byte compare il valore 'A', allora la parte dati è costituita da codici Ascii; se compare il valore è 'L', allora la parte dati è un valore numerico intero senza segno little endian.
Riportare i pacchetti che contengono il proprio cognome e la propria data di nascita nel formato aammgg.

10 Un protocollo di livello 2 (Datalink) che trasporta codici Ascii prevede un'intestazione di un byte: se il valore è 1, allora il pacchetto ha una coda di

un byte che contiene il n. di byte dell'intero pacchetto; se l'intestazione vale 0, allora il pacchetto non ha coda.
Interpretare correttamente i seguenti pacchetti di livello 2:

1. `31h,36h,31h,30h,34h`;
2. `30h,36h,31h,30h,34h`;
3. `31h,36h,31h,30h,04h`;
4. `30h,36h,31h,30h,04h`

11 Un protocollo di livello 2 (Datalink) che trasporta solo codici Ascii prevede una coda di un byte che contiene il n. di ottetti del pacchetto.
Riportare i seguenti pacchetti:

a Contenente l'acronimo dell'Open System Interconnections.
b Contenente l'acronimo di Protocol Data Unit.
c Contenente le proprie iniziali.

12 Un protocollo di livello 2 (Datalink) con parte dati in Ascii e il rimanente in formato numerico, prevede un'intestazione di un byte: se il valore è 1, allora il pacchetto ha una coda di un byte che contiene il n. di byte dell'intero pacchetto; se l'intestazione vale 0, allora il pacchetto non ha coda.
Interpretare correttamente i seguenti pacchetti di livello 2:

a `01h, 49h,70h,76h,34h,06h`;
b `01h,69h,6eh,74h,65h,72h,66h,61h,63h 63h,69h,61h,0ch`;
c `01h, 49h,70h,76h,34h,36h`;
d `30h,36h,31h,30h,04h`;
e `00h, 49h,70h,76h,36h`.

13 Un protocollo di livello 2 (Datalink) che trasporta codici Ascii prevede una coda di due byte che contiene il n. di ottetti del pacchetto.
Riportare i seguenti pacchetti:

a Contenente il proprio cognome.
b Contenente il proprio codice fiscale.
c Contenente la propria data di nascita.

14 Un protocollo di livello 2 (Datalink) che trasporta dati in codici Ascii prevede un'intestazione di un byte che vale 'N' se la coda di un byte è in formato numerico; 'A' se la coda è in formato Ascii. La coda contiene il numero di byte dell'intero pacchetto.
Riportare i seguenti pacchetti:

a Contenente il proprio nome.
b Contenente la propria residenza.
c Contenente la propria data di nascita.

15 Un protocollo di livello 2 (Datalink) che trasporta dati in codici Ascii prevede una coda di un byte numerico che contiene il n. di byte del pacchetto.
Interpretare correttamente i seguenti pacchetti di livello 2:

1. `49h,43h,41h,4eh,4eh,06h`;
2. `49h,41h,4eh,41h,05h`;
3. `49h,70h,76h,34h`.

16 Un protocollo di livello 4 (Transport) con parte dati Ascii, prevede un header di due byte, in cui il primo può valere 0 o 1. Se vale 0, allora il secondo byte contiene il numero di frammenti totale; se vale 1, allora il secondo byte indica il numero di sequenza del frammento.
Interpretare correttamente i seguenti pacchetti di livello 4 se l'MTU vale 4:

a `01h,04h,6dh,61h`;
b `01h,00h,43h,69h`;
c `00h,05h`;
d `01h,01h,61h,6fh`;
e `01h,02h,20h,6dh`;
f `01h,03h,61h,6dh`.

17 Un protocollo di livello 4 (Transport) con parte dati Ascii, prevede un header di due byte, in cui il primo può valere 0 o 1. Se vale 0, allora il secondo byte contiene il numero di frammenti totale; se vale 1, allora il secondo byte indica il numero di sequenza del frammento.
L'MTU del protocollo vale 6.
Mostrare il pacchetto di tipo 0 per spedire la stringa "Dal tramonto all'alba".

18 Un protocollo di livello 4 (Transport) con parte dati Ascii, prevede un header di due byte, in cui il primo può valere 'F' o 'N'. Se vale 'F', allora il secondo byte contiene il numero di frammenti totale; se vale 'N', allora il secondo byte indica il numero di sequenza del frammento.
L'MTU del protocollo vale 6.
Mostrare i pacchetti necessari per spedire la stringa "Planet terror".

19 Un protocollo di livello 4 (Transport) con parte dati Ascii, prevede un header di due byte, in cui il primo può valere 'F' o 'N'. Se vale 'F', allora il secondo byte contiene il numero di frammenti totale; se vale 'N', allora il secondo byte indica il numero di sequenza del frammento.
Inoltre prevede una coda numerica di un byte contenente il n. di byte del pacchetto. L'MTU del protocollo vale 8.
Mostrare i pacchetti necessari per spedire la stringa "C'era una volta in Messico".

20 Un protocollo di livello 4 (Transport) con parte dati Ascii, prevede un header di due byte, in cui il primo può valere 't' o 'n'. Se vale 't', allora il secondo byte contiene il numero di frammenti totale; se vale 'n', allora il secondo byte indica il numero di sequenza del frammento.
Interpretare correttamente i seguenti pacchetti di livello 4 se l'MTU vale 10:

- **a** `6eh,03h,6dh,61h,67h,69h,63h,61h`;
- **b** `6eh,01h,72h,6fh,20h,64h,65h,6ch,6ch,61h`;
- **c** `6eh,02h,20h,70h,69h,65h,74h,72h,61h,20h`;
- **d** `74h,05h`;
- **e** `6eh,00h,49h,6ch,20h,6dh,69h,73h,74h,65h`.

Indirizzi IP

21 Dato l'indirizzo IP 160.1.2.8, determinarne la classe di appartenenza.

22 Dato l'indirizzo IP 225.230.8.20, determinarne la classe di appartenenza.

23 Dato l'indirizzo IP 220.34.1.250, determinarne la classe di appartenenza.

24 Dato l'indirizzo IP 241.43.100.61, determinarne la classe di appartenenza.

25 Dato l'indirizzo IP di 127.0.0.1, determinarne la classe di appartenenza.

26 Dato l'indirizzo IP 0.0.0.0, determinarne la classe di appartenenza.

27 Dato l'indirizzo IP 127.255.255.255, determinarne la classe di appartenenza.

28 Dato l'indirizzo IP 128.0.0.0, determinarne la classe di appartenenza.

29 Dato l'indirizzo IP 191.255.255.255, determinarne la classe di appartenenza.

30 Dato l'indirizzo IP 192.0.0.0, determinarne la classe di appartenenza.

31 Dato l'indirizzo IP 223.255.255.255, determinarne la classe di appartenenza.

32 Dato l'indirizzo IP 224.0.0.0, determinarne la classe di appartenenza.

33 Dato l'indirizzo IP 92.168.254.1, determinarne la classe di appartenenza.

34 Dato l'indirizzo IP 239.255.255.255, determinarne la classe di appartenenza.

35 Determinare quanti sono gli indirizzi IP di classe A.

36 Determinare quanti sono gli indirizzi IP di classe B.

37 Determinare quanti sono gli indirizzi IP di classe C.

38 Determinare quanti sono gli indirizzi IP di classe D.

39 Determinare quanti sono gli indirizzi IP di classe E.

40 Determinare l'intervallo degli indirizzi IP di classe A (minimo ÷ massimo).

41 Determinare l'intervallo degli indirizzi IP di classe B (minimo ÷ massimo).

42 Determinare l'intervallo degli indirizzi IP di classe C (minimo ÷ massimo).

ESERCIZI PER LA VERIFICA DI LABORATORIO

I prerequisiti per affrontare questi esercizi sono la consultazione di una tabella di codici Ascii e la disponibilità di un ambiente di sviluppo per programmi in linguaggio C. Per compilare gli esercizi proposti è stato usato l'ambiente gratuito multipiattaforma Windows/Linux ***Code::Blocks 10.5*** con **gcc 4.4.1**.

I testi degli esercizi presentano uno o più **layout** di input/output richiesti; in questo modo sono indicate, a volte, alcune specifiche del programma come l'output, il controllo dell'input o i valori ammissibili.

Pacchetti e protocolli

1 Un protocollo di livello 7 (Applicazione) consegna una stringa e un numero intero casuale al livello 6 (Presentazione).
Scrivere un programma in linguaggio C per stampare sullo schermo i due pacchetti di livello 6 (Presentazione) se l'intestazione vale un byte 'a' o 'n' che indicano, rispettivamente, parte dati in Ascii e parte dati numerica in big endian.

```
Layout:
Livello7, digitare una stringa: Pluto
Livello7, numero casuale: 330
Pkt1: 61h 50h 6ch 75h 74h 6fh
Pkt2: 6eh 01h 4ah
```

L'esercizio è del tutto simile a quello svolto nel libro.
Cambiano i valori costanti delle intestazioni: in quel caso erano 41h ('A') e 4eh ('N'), ora sono 61h ('a') e 6eh ('n') e il numero viene estratto in modo pseudocasuale, utilizzando, per esempio, la riga di codice:

```
n = rand();
```

Per fare in modo che ad ogni avvio il numero pseudocasuale cambi, anteporre la riga di codice:

```
srand(time(0));
```

Infine, per una corretta compilazione, ricordare di aggiungere nella sezione delle inclusioni le intestazioni:

```
#include <stdlib.h>  // per rand() e srand()
#include <time.h>    // per time()
```

2 Un protocollo di livello 6 (Presentazione) rappresenta dati Ascii o un numero big endian a seconda di un'intestazione di un byte che vale 'a' o 'b'.
Scrivere un programma in linguaggio C che acquisisce prima il formato e poi il dato, quindi stampi il relativo pacchetto.

Layout:

OUTPUT 1

```
Digitare il formato (a,b): c
Digitare il formato (a,b): a
Digitare una stringa: Paperino

Pkt: 61h,50h,61h,70h,65h,72h,69h,6eh 6fh
```

OUTPUT 2

```
Digitare il formato (a,b): b
Inserire un numero (<32768): 3456

Pkt: 62h,0dh,80h
```

3 Un protocollo di livello 6 (Presentazione) rappresenta dati Ascii o un numero big endian a seconda di un'intestazione di un byte che vale 'a' o 'b'.
Scrivere un programma in linguaggio C che genera un pacchetto di tipo 'b' casualmente.

Layout:

OUTPUT

```
Formato casuale (a,b): a
Formato casuale (a,b): a
Formato casuale (a,b): b
Numero casuale da rappresentare: 2ee0h
Pkt: 62h,2eh,e0h
```

4 Un protocollo di livello 6 (Presentazione) rappresenta dati Ascii se il byte di intestazione vale 'A'.
Scrivere un programma in linguaggio C che acquisisce un numero e genera il pacchetto corrispondente.

Layout:

OUTPUT

```
Inserire un numero (<32768): 23450

Pkt: 41h,32h,33h,34h,35h,30h
```

5 Un protocollo di livello 6 (Presentazione) rappresenta dati Ascii o un numero big endian o un numero little endian a seconda di una intestazione di un byte che vale 'a' o 'b' o 'c'.

Scrivere un programma in linguaggio C che genera casualmente un pacchetto numerico.

`Layout` (*Nota: 59d9h = 23001d*):

OUTPUT 1

```
Formato casuale (a,b,c): b
Numero casuale da rappresentare: 59d9h

Pkt: 62h,59h,d9h
```

OUTPUT 2

```
Formato casuale (a,b,c): a
Numero casuale da rappresentare: 59d9h

Pkt: 61h,32h,33h,30h,30h,31h
```

OUTPUT 3

```
Formato casuale (a,b,c): c
Numero casuale da rappresentare: 59d9h

Pkt: 62h,d9h,59h
```

6 Un protocollo di livello 6 (Presentazione) rappresenta dati Ascii numerici decimali se il byte di intestazione vale 'A', dati Ascii numerici esadecimali se l'intestazione vale 'H'.
Scrivere un programma in linguaggio C che acquisisce un numero e genera i due pacchetti Ascii numerici.

`Layout:`

OUTPUT

```
Inserire un numero (<32768): 4519

Pkt1: 41h,34h,35h,31h,39h
Pkt1: 48h,31h,31h,41h,37h
```

7 Un protocollo di livello 3 (Network) consegna una stringa a un livello 2 (Datalink) che prevede una coda di un byte contenente il n. di byte del pacchetto, ma in formato Ascii.
Scrivere un programma in linguaggio C che prenda in input una stringa e stampare il pacchetto di livello 2 appropriato.

`Layout:`

OUTPUT

```
Digitare una stringa (<9 car): Ciao

Pkt: 43h,69h,61h,6fh,35h
```

8 Un protocollo di livello 2 (Datalink) trasporta dati Ascii e prevede una coda di un byte contenente il n. di byte del pacchetto solo se nell'intestazione compare il valore 1.
Scrivere un programma in linguaggio C che scelga a caso se generare la coda o meno per una stringa in input e stampi il pacchetto appropriato.

`Layout:`

OUTPUT 1

```
Digitare una stringa: Marzo
Con coda: Sì

Pkt: 01h,4dh,61h,72h,7ah,6fh,07h
```

OUTPUT 2

```
Digitare una stringa: Aprile
Con coda: No

Pkt: 41h,70h,72h,69h,6ch,65h
```

9 Un protocollo di livello 2 (Datalink) trasporta dati Ascii e prevede una coda di un byte contenente il n. di byte del pacchetto. L'MTU vale 8.
Scrivere un programma in linguaggio C che prenda in input i byte di un pacchetto e stampi se appropriato.

`Layout:`

OUTPUT 1

```
Byte n. 0: 12
Byte n. 0: 99
Byte n. 1: 105
Byte n. 2: 97
Byte n. 3: 111
Byte n. 4: 5

Il pacchetto è corretto.
```

OUTPUT 2

```
Byte n. 0: 67
Byte n. 1: 73
Byte n. 2: 65
Byte n. 3: 79
Byte n. 4: 6

Il pacchetto è scorretto.
```

OUTPUT 3

```
Byte n. 0: 67
Byte n. 1: 73
Byte n. 2: 65
Byte n. 3: 79
Byte n. 4: 32
Byte n. 5: 77
Byte n. 6: 65
Byte n. 7: 82

Il pacchetto è scorretto.
```

10 Un protocollo di livello 2 (Datalink) trasporta dati Ascii e prevede una coda di due byte contenente il n. di byte del pacchetto in formato Ascii. L'MTU vale 16.

Scrivere un programma in linguaggio C che acquisisce una stringa e stampa il pacchetto appropriato.

Layout:

OUTPUT 1

```
Digitare una stringa: Attenti

Pkt: 41h,74h,74h,65h,6eh,74h,69h,00h,39h
```

OUTPUT 2

```
Digitare una stringa: Dicembre

Pkt:44h,69h,63h,65h,6dh,62h,72h,65h,31h,30h
```

OUTPUT 3

```
Digitare una stringa: Attenti al cane
Digitare una stringa: Lunga

Pkt: 4ch,75h,6eh,67h,61h,30h,37h
```

11 Un protocollo di livello 2 (Datalink) trasporta dati numerici big endian e prevede una coda da un byte con il n. totale di byte del pacchetto.
Scrivere un programma in linguaggio C che acquisisca un numero intero e stampi il pacchetto appropriato.

Layout:

OUTPUT

```
Digitare un numero (<32768): 32768
Digitare un numero (<32768): 32767

Pkt: 7fh,ffh,03h
```

12 Un protocollo di livello 2 (Datalink) trasporta dati Ascii e prevede una coda da un byte con il n. totale di byte del pacchetto.
Scrivere un programma in linguaggio C che acquisisca un numero intero e stampi il pacchetto appropriato.

Layout:

OUTPUT

```
Digitare un numero (<32768): 32768
Digitare un numero (<32768): 32767

Pkt: 33h,32h,37h,36h,37h,06h
```

13 Un protocollo di livello 4 (Transport) che trasporta dati in codici Ascii, ha una intestazione di due byte.
Il primo byte può valere 'f' se si tratta di un pacchetto frammentato; vale 't' se contiene il numero di frammenti totali.
Il secondo byte contiene il n. di sequenza (pacchetti di tipo 'f') o il n. totale dei frammenti (pacchetti di tipo 't').
Scrivere un programma in linguaggio C che data la stringa "Attenti al cane" e l'MTU che vale 5, stampi sullo schermo il pacchetto di tipo 't'.

Layout:

OUTPUT

```
Il pacchetto di tipo 't' per la stringa "Attenti al cane" con MTU=5 vale:

Pkt: 74h,05h
```

NB. La stringa data è lunga 15 caratteri, ma i pacchetti frammentati possono contenerne solo 3 alla volta, dato che il massimo è 5 (MTU), e 2 byte sono riservati all'intestazione. Quindi 15/3=5 (05h).

14 Un protocollo di livello 4 (Transport) che trasporta dati in codici Ascii, ha una intestazione di due byte.
Il primo byte può valere 'f' se si tratta di un pacchetto frammentato; vale 't' se contiene il numero di frammenti totali.
Il secondo byte contiene il n. di sequenza (pacchetti di tipo 'f') o il n. totale dei frammenti (pacchetti di tipo 't').
Scrivere un programma in linguaggio C che data la stringa "Attenti al cane" e un MTU preso in input, stampi il pacchetto di tipo 't'.

Layout:

OUTPUT 1

```
Digitare l'MTU (tra 3 e 9): 7
Il pacchetto di tipo 't' per la stringa "Attenti al cane" con MTU=7 vale:

Pkt: 74h,03h
```

OUTPUT 2

```
Digitare l'MTU (tra 3 e 9): 9
Il pacchetto di tipo 't' per la stringa "Attenti al cane" con MTU=9 vale:

Pkt: 74h,03h
```

OUTPUT 3

```
Digitare l'MTU (tra 3 e 9): 3
Il pacchetto di tipo 't' per la stringa "Attenti al cane" con MTU=3 vale:

Pkt: 74h,0fh
```

15 Un protocollo di livello 4 (Transport) che trasporta dati in codici Ascii, ha un'intestazione di due byte.
Il primo byte può valere 'f' se si tratta di un pacchetto frammentato; vale 't' se contiene il numero di frammenti totali.
Il secondo byte contiene il n. di sequenza (pacchetti di tipo 'f') o il n. totale dei frammenti (pacchetti di tipo 't').

Scrivere un programma in linguaggio C che presi in input una stringa dati e un MTU, stampi il pacchetto di tipo 't'.

Layout:

OUTPUT

```
Digitare una stringa: Attenti al gatto
Digitare l'MTU (tra 3 e 9): 5
Il pacchetto di tipo 't' per la stringa "Attenti al gatto" con MTU=5 vale:

Pkt: 74h,06h
```

16 Un protocollo di livello 4 (Transport) che trasporta dati in codici Ascii, ha un'intestazione di due byte.
Il primo byte può valere '0' se si tratta di un pacchetto frammentato; vale '1' se contiene il numero di frammenti totali.
Il secondo byte contiene il n. di sequenza (pacchetti di tipo '0') o il n. totale dei frammenti (pacchetti di tipo '1').
Scrivere un programma in linguaggio C che acquisisce in input una stringa e un numero che rappresenta l'MTU; frammentare i dati rappresentati dalla stringa e stampare a schermo i pacchetti.

Layout:

OUTPUT

```
Digitare una stringa: Attenti al gatto
Digitare l'MTU (tra 3 e 9): 7

Pkt1: 30h,00h,41h,74h,74h,65h,6eh
Pkt2: 30h,01h,74h,69h,20h,61h,6ch
Pkt3: 30h,02h,20h,67h,61h,74h,74h
Pkt4: 30h,03h,6fh
Pkt5: 31h,04h
```

Indirizzi IP

17 Scrivere un programma in linguaggio C che acquisisca da tastiera un indirizzo IP, quindi stampi sullo schermo se si tratta di un indirizzo di classe A o meno.

Layout:

OUTPUT 1

```
Immettere un indirizzo IP.
Primo byte: 192
Secondo byte: 23
Terzo byte: 300
Terzo byte: 2
Quarto byte: 3

L'indirizzo IP 192.23.2.3 non è di classe A
```

OUTPUT 2

```
Immettere un indirizzo IP.
Primo byte: 10
Secondo byte: 230
Terzo byte: 8
Quarto byte: 1

L'indirizzo IP 10.230.8.1 è di classe A
```

18 Scrivere un programma in linguaggio C che stampi sullo schermo tutti gli indirizzi IP della forma 192.168.0.x, dove x è un valore variabile.

Layout:

OUTPUT

```
IP n. 001: 192.168.0.0
IP n. 002: 192.168.0.1
IP n. 003: 192.168.0.2
IP n. 004: 192.168.0.3
IP n. 005: 192.168.0.4
...
IP n. 254: 192.168.0.253
IP n. 255: 192.168.0.254
IP n. 256: 192.168.0.255
```

19 Scrivere un programma in linguaggio C che acquisisca da tastiera un indirizzo IP, quindi stampi sullo schermo se si tratta di un indirizzo di classe B o meno.

Layout:

OUTPUT 1

```
Immettere un indirizzo IP.
Primo byte: 130
Secondo byte: 9
Terzo byte: 2
Quarto byte: 254

L'indirizzo IP 130.9.2.254 è di classe B
```

OUTPUT 2

```
Immettere un indirizzo IP.
Primo byte: 224
Secondo byte: 2
Terzo byte: 7
Quarto byte: 3

L'indirizzo IP 224.2.7.3 non è di classe B
```

20 Scrivere un programma in linguaggio C che acquisisca da tastiera un indirizzo IP, quindi stampi sullo schermo se si tratta di un indirizzo di classe C o meno.

Layout:

OUTPUT

```
Immettere un indirizzo IP.
Primo byte: 192
Secondo byte: 168
Terzo byte: 0
Quarto byte: 1

L'indirizzo IP 192.168.0.1 è di classe C
```

21 Scrivere un programma in linguaggio C che genera un indirizzo IP valido.

Layout:

OUTPUT

```
Indirizzo IP generato: 220.234.56.1
```

22 Scrivere un programma in linguaggio C che acquisisca da tastiera un indirizzo IP, quindi stampi sullo schermo se si tratta di un indirizzo di classe D o meno.

Layout:

OUTPUT

```
Immettere un indirizzo IP.
Primo byte: 240
Secondo byte: 0
Terzo byte: 1
Quarto byte: 1

L'indirizzo IP 240.0.1.1 non è di classe D
```

23 Scrivere un programma in linguaggio C che genera un indirizzo IP valido fino a quando l'indirizzo è di classe C.

Layout:

OUTPUT

```
Indirizzo IP generato: 10.24.6.255
Indirizzo IP generato: 120.7.212.211
Indirizzo IP generato: 225.4.78.12
Indirizzo IP generato: 1.2.111.29
Indirizzo IP generato: 220.19.55.87
```

24 Scrivere un programma in linguaggio C che stampi sullo schermo tutti gli indirizzi IP della forma 192.168.x.y.

Layout:

OUTPUT

```
IP n. 00001: 192.168.0.0
IP n. 00002: 192.168.0.1
IP n. 00003: 192.168.0.2
IP n. 00004: 192.168.0.3
IP n. 00005: 192.168.0.4
...
IP n. 65531: 192.168.255.250
IP n. 65532: 192.168.255.251
IP n. 65533: 192.168.255.252
IP n. 65534: 192.168.255.253
IP n. 65535: 192.168.255.254
IP n. 65536: 192.168.255.255
```

25 Scrivere un programma in linguaggio C che genera un indirizzo IP valido e ne determini la classe.

Layout:

OUTPUT

```
Indirizzo IP generato: 240.12.1.255
L'indirizzo è di classe E
```

26 Scrivere un programma in linguaggio C che acquisisca da tastiera un indirizzo IP, quindi stampi sullo schermo se si tratta di un indirizzo di classe E o meno.

Layout:

OUTPUT

```
Immettere un indirizzo IP.
Primo byte: 250
Secondo byte: 13
Terzo byte: 45
Quarto byte: 100

L'indirizzo IP 250.13.45.100 è di classe E
```

B3 Reti locali e reti geografiche

Rimangiarsi le parole

Uno dei pionieri nel mondo delle reti, l'inventore di Ethernet **Robert Metcalfe** (1946), pur protagonista indiscusso dell'innovazione tecnologica mondiale, fu autore di una famosa previsione errata.

Nel 1995 sostenne che «[...] la rete Internet, in brevissimo tempo, esploderà in maniera spettacolare come una supernova, e nel 1996 collasserà catastroficamente a causa della sua crescita esponenziale[...]» (*Infoworld Magazine, dicembre 1995*).

La sua posizione era così categorica che promise di rimangiarsi le parole se ciò non fosse puntualmente avvenuto.

Infatti, nel 1997, in una conferenza pubblica, inserì il suo articolo di *Infoworld Magazine* in un frullatore e ne mangiò la poltiglia sotto gli occhi stupiti della platea.

Le reti locali LAN e le reti geografiche WAN sono i tipi di rete più diffuse e utilizzate. Naturalmente nel primo caso abbiamo reti che insistono su aree limitate, quasi sempre private e con bit rate alto (≥100Mbit/s); nel secondo caso abbiamo una rete che opera sul territorio mondiale, attraversando suolo pubblico e superando confini statali e continentali, con larghezze di banda variabili ma relativamente basse (<100Mbit/s).

LAN e WAN si differenziano anche per la **topologia** adottata, ovvero il modo in cui i sistemi sono interconnessi fisicamente.

Nelle LAN i sistemi sono connessi, normalmente, **a stella**, con gli apparati di interconnessione che prendono il nome di **switch**, a loro volta interconnessi **ad albero**.

Nelle WAN i sistemi sono connessi, normalmente, **a maglia** (incompleta), con l'apparato di interconnessione che prende il nome di **router**. In questi contesti i termini router, gateway e Intermediate System possono essere considerati sinonimi.

Switch e router si connettono ai nodi di una rete attraverso delle prese di interfaccia denominate **porte**.

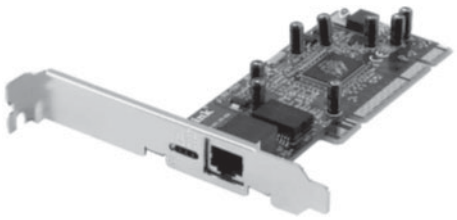
Sulle LAN i pacchetti circolano in modalità **broadcast**, ovvero un qualsiasi pacchetto spedito da un nodo circola sulla rete fino a raggiungere ogni altro nodo della rete, se necessario. In base al modo di indirizzamento utilizzato sulla LAN, solo il nodo destinatario, però, trattiene il pacchetto e tutti gli altri lo scartano.

Al contrario, sulle WAN il pacchetto spedito da un nodo circola sulla rete in modalità (generalmente) **punto-punto**, giungendo al nodo destinatario attraversando numerose tratte e nodi intermedi che si preoccupano di indirizzarlo correttamente verso la destinazione in base al modello **packet switching** (la commutazione di pacchetto).

1 Indirizzi di livello 2

In una rete LAN i nodi possiedono necessariamente un indirizzo univoco su scala mondiale denominato **MAC address**.

Il MAC address è un numero intero senza segno rappresentato su 48 bit (o 6 byte) che si trova «prestampato»nella memoria ROM di ogni scheda di rete (**NIC**, *Network Interface Card*), che è il dispositivo hardware necessario ad ogni nodo per connettersi ad una rete LAN.



Sui personal computer la scheda di rete è spesso integrata, cioè fa parte dell'I/O della macchina; se necessario si possono aggiungere altre schede di rete sugli slot disponibili.

Gli indirizzi MAC si rappresentano in esadecimale, con 12 cifre accoppiate separate da un trattino, come ad esempio **5C-26-0A-29-BE-34**, oppure separate da «due punti», come per esempio **00:c0:26:f0:33:fc**.

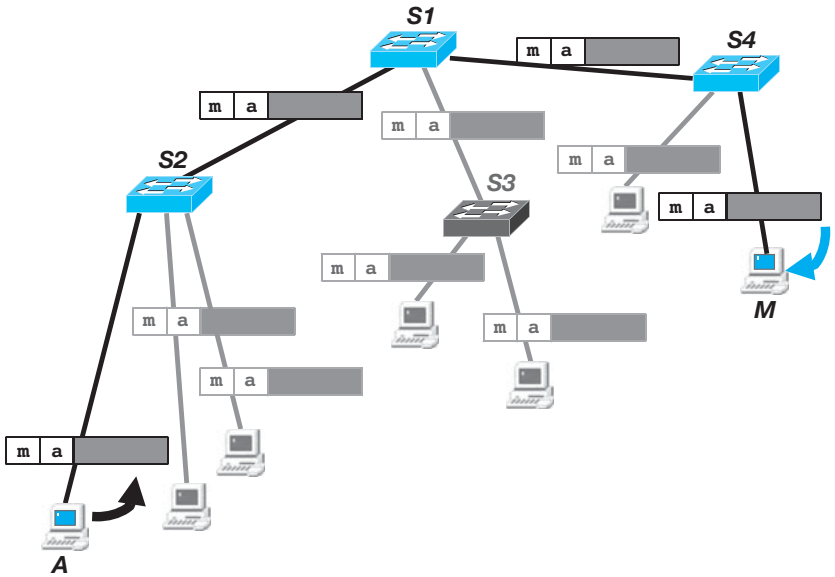
Affinché ogni NIC prodotta nel mondo abbia un indirizzo MAC univoco, ad ogni costruttore viene assegnato un identificativo privato su 3 byte; i rimanenti 3 byte li decide autonomamente il costruttore in modo che ogni scheda che produrrà abbia un numero di serie differente.

Per esempio, Intel Corporation ha un OUI (*Organization Unique Number*) che vale 00-AA-00, mentre IBM Inc. possiede 08-00-5A.

L'indirizzo MAC, essendo parte del livello 1 Fisico (è integrato nell'hardware), consente ad una rete LAN di individuare i propri nodi senza problemi, cosicché è sufficiente che i nodi di una rete LAN **implementino i soli livelli 1 Fisico e 2 Collegamento dati** per funzionare correttamente.

Per questo motivo, anche se in modo leggermente scorretto, gli indirizzi MAC si dicono anche indirizzi di livello 2 o, più correttamente, **indirizzi fisici**.

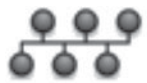
Affinché un pacchetto in una LAN raggiunga il destinatario, è sufficiente che esso contenga l'indirizzo MAC del destinatario e sia poi spedito sulla rete: il modello broadcast farà il resto, il pacchetto giungerà ad ogni nodo e solo il nodo che possiede quell'indirizzo MAC prenderà in carico il pacchetto. Nel pacchetto circolante su una LAN deve essere inserito anche l'indirizzo MAC del mittente, affinché il destinatario possa sapere chi gli sta inviando i dati.



Nella figura il nodo ***A*** di una LAN invia un pacchetto al nodo ***M***.

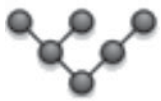
Topologie

Nelle telecomunicazioni si distinguono varie configurazioni geometriche che rappresentano i modi in cui i sistemi sono connessi fisicamente (**topologie**).
Tra queste riportiamo la topologia bus, stella, anello, albero, maglia (o mesh), maglia completa. Ogni topologia conferisce un modo di circolazione dei pacchetti proprio, detta **topologia logica**.


Bus


Stella


Anello


Albero


Maglia


Maglia completa

Simboli

Per rappresentare graficamente i nodi intermedi e gli apparati di rete si usano simboli grafici derivati dall'azienda **CISCO Inc.**, oramai divenuti internazionali, per es:


Router


Switch


Access Point


Modem (DCE)

Gli indirizzi MAC dei due nodi sono, rispettivamente, **a** e **m**, cosicché il pacchetto MAC ha la forma [m|a|] dove compaiono l'indirizzo MAC destinatario, l'indirizzo MAC mittente e la busta di livello 3 (in grigio) contenente l'informazione.

Giunto al primo switch ***S2***, il pacchetto viene rispedito su tutte le porte dello switch, ma nessun nodo su queste porte conserverà il pacchetto.

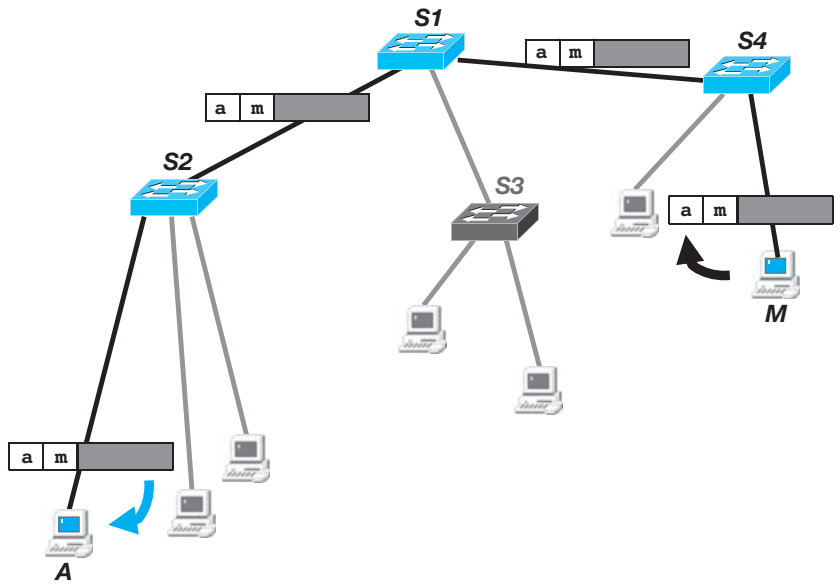
Il pacchetto continua a circolare in broadcast attraverso tutti gli switch ***S1***, ***S3*** e ***S4*** collegati ad albero e tutti i nodi, fino a giungere a destinazione sul nodo ***M***.

Infatti il NIC di ***M*** è l'unico a possedere l'indirizzo MAC **m** specificato nel pacchetto.

Gli switch si comportano in questo modo, inoltrando ogni pacchetto proveniente su tutte le proprie porte, solo quando sono stati appena avviati.

Infatti ogni porta dello switch, non appena riceve un pacchetto su una sua porta, memorizza su quella porta l'indirizzo MAC mittente del pacchetto: ora sa che il nodo con quell'indirizzo si raggiunge attraverso quella porta.

Per esempio, se in seguito ***M*** dovesse a sua volta comunicare con ***A***, il pacchetto [a|m|] non verrebbe più propagato dagli switch ***S4***, ***S1*** e ***S2***, ma inoltrato direttamente attraverso le porte «informate» degli switch:



2 Indirizzi di livello 3

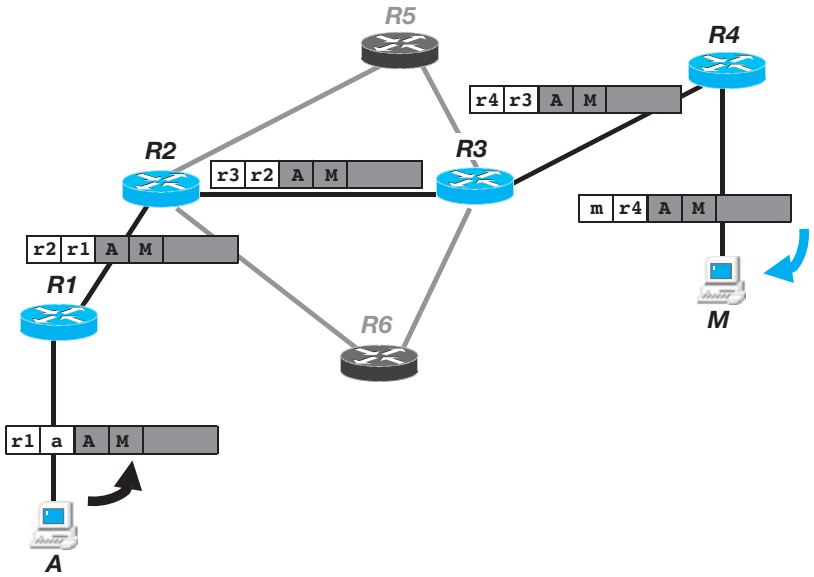
In una rete WAN invece sono gli indirizzi di livello 3 che hanno il compito di individuare i nodi nella rete in modo univoco (vedi Indirizzi IP di TCP/IP).

Ogni pacchetto circolante su una WAN deve contenere l'indirizzo di livello 3 del mittente e del destinatario. Al termine di ogni tratta il router li analizza e decide la tratta successiva più idonea per dirigere a destinazione il pacchetto (**instradamento**).

In questo modo su una WAN è necessario che i nodi intermedi **implementino solo i primi 3 livelli OSI** per poter instradare verso la desti-

nazione i pacchetti. Ogni tratta potrebbe essere realizzata fisicamente in modo diverso, cosicché i nodi intermedi hanno la capacità di modificare il contenuto dei livelli 1 e 2 del pacchetto, adeguandoli al cammino. Naturalmente la parte dati del pacchetto non viene modificata per preservarne il contenuto originale: solo le buste di livello 1 e 2 saranno oggetto della modifica.

La circolazione di un pacchetto su una WAN, da un mittente ***A*** a un destinatario ***M*** può essere rappresentata come in figura:



Ora si nota che il pacchetto trasmesso da ***A***, cioè `r1 a A M` contiene due coppie di indirizzi, quelli più esterni di livello 2 (esempio, indirizzi MAC) come **`r1`** e **`a`**, e quelli più interni di livello 3 (esempio, indirizzi IP) come **`A`** e **`M`**.

È fondamentale notare come la **busta di livello 2 cambia** per ogni tratta attraversata, adattandosi alla nuova rete (e quindi al nuovo livello 2 incontrato sul cammino), mentre la **busta di livello 3** in grigio **non cambia mai**, arrivando a destinazione intatta.

Anche i router memorizzano le informazioni di instradamento sulle loro porte, ma con regole molto più articolate rispetto a quella vista per gli switch.

3 Applicazioni

TCP/IP prevede molti programmi applicativi di livello 7 di uso generale, sia per controllare il funzionamento della rete, sia per effettuare configurazioni e verifiche delle configurazioni.

I sistemi operativi più importanti rendono disponibili questi programmi all'utente, quasi sempre attraverso la shell a caratteri e con sintassi uniforme, pertanto i sistemi operativi di classe Windows e Linux consentono di usare proficuamente alcuni programmi TCP/IP, tra i quali ifconfig/ipconfig, ping, netstat, arp e traceroute/tracert.

Con il programma **ifconfig/ipconfig** è possibile controllare ma anche

configurare un'interfaccia di rete (NIC). Si tratta del programma ideale per conoscere l'indirizzo IP del proprio host (o gli indirizzi IP se si tratta di un gateway) e i relativi indirizzi MAC.

Con il programma **ping** si può verificare se un host con un determinato indirizzo IP (o con il nome equivalente) è attualmente attivo sulla rete TCP/IP. Si tratta del programma ideale per verificare se un certo host è raggiungibile o meno.

Con il programma **netstat** è possibile verificare quali e quante connessioni di rete TCP/IP sono attualmente presenti. Si tratta del programma ideale per controllare le porte TCP utilizzate da un host.

Con il programma **arp** si può visualizzare la tabella delle corrispondenze tra indirizzi di livello 2 MAC e indirizzi di livello 3 IP, infine con il programma **traceroute/tracert** si può visualizzare l'elenco dei nodi attraversati da un pacchetto per arrivare a destinazione.

3.1 ifconfig/ipconfig

Per controllare gli indirizzi di livello 2 (esempio, indirizzi MAC) e di livello 3 (esempio, indirizzi IP) di un host si usa il programma ipconfig (sotto sistemi operativi Windows) o ifconfig (sotto sistemi operativi Linux).

Da una shell a caratteri si digita direttamente il comando ifconfig per avere le informazioni su ogni interfaccia (NIC) presente sull'host (Linux):

LINUX

```
linux:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:c0:26:f0:33:fc
          inet addr:192.168.10.2  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::2c0:26ff:fef0:33fc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:94199212 errors:48 dropped:0 overruns:0 frame:125
          TX packets:150793338 errors:1 dropped:0 overruns:1 carrier:1
          collisions:0 txqueuelen:1000
          RX bytes:3466099535 (3.2 GiB)  TX bytes:3836887750 (3.5 GiB)
          Interrupt:11 Base address:0xc000


lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:6331489 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6331489 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2555224645 (2.3 GiB)  TX bytes:2555224645 (2.3 GiB)
linux:~#
```

Si nota che questo host ha due interfacce di rete: **`eth0`** e **`lo`**; la prima ha indirizzo MAC **00:c0:26:f0:33:fc** e indirizzo IP **192.168.10.2**. Viene anche riportato l'indirizzo IPv6.

La seconda interfaccia non rappresenta una reale scheda di rete, ma una scheda di rete virtuale (fittizia) sempre presente su ogni host TCP/IP denominata **interfaccia di loopback**, che ha sempre indirizzo IP **127.0.0.1** e non ha, ovviamente, un indirizzo fisico MAC.

Le interfacce di loopback sono estremamente utili per fare prove ed esperimenti, per esempio nel caso in cui un host non sia acceso o non sia disponibile e quindi non sia possibile accedervi tramite il suo indirizzo IP. Nell'impossibilità di colloquiare con un host indisponibile, basta riproporre in locale il programma con cui si dovrebbe colloquiare in remoto e usare l'indirizzo di loopback per fare delle prove sulla stessa macchina, quasi come se si operasse realmente in rete.

L'analogo programma per Windows ipconfig si avvia da una shell di MSDOS con il parametro /all:

WINDOWS

```
C:\windows>ipconfig /all
Configurazione IP di Windows

   Nome host . . . . . . . . . . . . : DellE4310
   Suffisso DNS primario . . . . . . :
   Tipo nodo . . . . . . . . . . . . : Ibrido
   Routing IP abilitato. . . . . . . : No
   Proxy WINS abilitato . . . . . . . : No

Scheda PPP KRZR BlueTooth:
   Suffisso DNS specifico per connessione:
   Descrizione . . . . . . . . . . . : KRZR BlueTooth
   Indirizzo fisico. . . . . . . . . :
   DHCP abilitato. . . . . . . . . . : No
   Configurazione automatica abilitata   : Sì
   Indirizzo IPv4. . . . . . . . . . : 2.193.39.84(Preferenziale)
   Subnet mask . . . . . . . . . . . : 255.255.255.255
   Gateway predefinito . . . . . . . : 0.0.0.0
   Server DNS . . . . . . . . . . . . : 213.230.155.10, 217.200.200.42
   NetBIOS su TCP/IP . . . . . . . . : Disattivato

Scheda Ethernet BT:
   Stato supporto. . . . . . . . . . : Supporto disconnesso
   Suffisso DNS specifico per connessione:
   Descrizione . . . . . . . . . . . : Bluetooth Device (Personal Area Network)
   Indirizzo fisico. . . . . . . . . : C0-CB-38-A9-FD-A8
   DHCP abilitato. . . . . . . . . . : Sì
   Configurazione automatica abilitata   : Sì

Scheda Ethernet LAN:
   Stato supporto. . . . . . . . . . : Supporto disconnesso
   Suffisso DNS specifico per connessione:
   Descrizione . . . . . . . . . . . : Intel(R) 82577LM Gigabit Network Connection
   Indirizzo fisico. . . . . . . . . : 5C-26-0A-29-BE-34
   DHCP abilitato. . . . . . . . . . : No
   Configurazione automatica abilitata   : Sì
C:\windows>
```

L'output di ipconfig è molto diverso da quello di ifconfig, pur fornendo informazioni simili. Su questo host sono presenti tre interfacce di rete fisiche (**`KRZR Bluetooth`**, **`BT`**, **`LAN`**), ma non viene riportata l'interfaccia di loopback (pur presente e sempre con indirizzo IP 127.0.0.1).

Si nota che la prima interfaccia non ha indirizzo MAC, cioè non è una interfaccia di rete LAN, e possiede indirizzo IP **2.193.39.84**.

Le altre due hanno invece indirizzo fisico MAC (**C0-CB-38-A9-FD-A8, 5C-26-0A-29-BE-34**), ma non fanno parte di reti TCP/IP, non possedendo (ancora) un indirizzo IP.

3.2 ping

Il programma ping è fondamentale per capire se un determinato nodo di cui si conosce l'indirizzo IP è attualmente attivo sulla rete. Per esempio, serve per sapere se un nodo è avviato o meno oppure se è raggiungibile o meno.

Per il sistema operativo Linux il comando va digitato specificando il parametro **`-c [numero]`**, oltre all'indirizzo IP del nodo da controllare:

LINUX

```
linux:~# ping -c 4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=48 time=17.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=48 time=17.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=48 time=17.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=48 time=17.3 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4015ms
rtt min/avg/max/mdev = 17.017/17.287/17.470/0.170 ms
linux:~#
```

In questo caso il comando ping tenta per quattro volte di raggiungere l'host con indirizzo IP **8.8.8.8**, che è l'indirizzo di un host normalmente sempre avviato sulla rete Internet (è l'indirizzo della interfaccia di una macchina di Google, il DNS server).

«*Pingare*» questo indirizzo così semplice da ricordare consente di sapere immediatamente se la macchina da cui si effettua il comando è connessa a Internet. Nel caso proposto la macchina Linux è connessa a Internet, dato che i quattro tentativi hanno avuto successo.

Nella versione Windows del comando non è necessario specificare il numero di tentativi, che sono comunque quattro. In questo caso invece dell'indirizzo IP 8.8.8.8 si è specificato il **nome di dominio** equivalente:

WINDOWS

```
C:\windows>ping google-public-dns-a.google.com
Esecuzione di Ping google-public-dns-a.google.com [8.8.8.8] con 32 byte di dati:
PING: trasmissione non riuscita. Errore generale.
PING: trasmissione non riuscita. Errore generale.
PING: trasmissione non riuscita. Errore generale.
PING: trasmissione non riuscita. Errore generale.

Statistiche Ping per 8.8.8.8:
    Pacchetti: Trasmessi = 4, Ricevuti = 0,
    Persi = 4 (100% persi),
C:\windows>
```

Il programma ping in questo caso mostra come l'host da cui è stato effettuato il comando non sia connesso a Internet, nell'ipotesi che la macchina di Google di indirizzo 8.8.8.8 sia attiva.

Si nota come ping sia anche un **risolutore di nomi di dominio**, ovvero fornito il nome di dominio (in questo caso **google-public-dns-a.google.com**), fornisca l'equivalente indirizzo IP (in questo caso **8.8.8.8**).

Ping **«risolve» anche gli indirizzi IP**, ovvero fornito l'indirizzo IP, è in grado di restituire il nome di dominio equivalente, usando l'opzione **-a**.

WINDOWS

```
C:\windows>ping -a 8.8.8.8
Esecuzione di Ping google-public-dns-a.google.com [8.8.8.8] con 32 byte di dati:

Risposta da 8.8.8.8: byte=32 durata=3499ms TTL=49
Risposta da 8.8.8.8: byte=32 durata=611ms TTL=49
Risposta da 8.8.8.8: byte=32 durata=1316ms TTL=49
Richiesta scaduta.

Statistiche Ping per 8.8.8.8:
    Pacchetti: Trasmessi = 4, Ricevuti = 3,
    Persi = 1 (25% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
    Minimo = 611ms, Massimo =  3499ms, Medio =  1808ms
C:\windows>
```

Per verificare che un nome di dominio non è sempre associato ad un unico indirizzo IP, come invece dovrebbe essere, si può *pingare* consecutivamente il sito di Google e verificare che allo stesso nome di dominio *www.google.it* risponde più di una interfaccia con diversi indirizzi IP:

WINDOWS

```
C:\windows>ping www.google.it
Esecuzione di Ping www-cctld.l.google.com [173.194.35.152] con 32 byte di dati:
Risposta da 173.194.35.152: byte=32 durata=37ms TTL=54
Risposta da 173.194.35.152: byte=32 durata=36ms TTL=54
Risposta da 173.194.35.152: byte=32 durata=59ms TTL=54
Risposta da 173.194.35.152: byte=32 durata=37ms TTL=54

Statistiche Ping per 173.194.35.152:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
    Minimo = 36ms, Massimo =  59ms, Medio =  42ms

C:\Windows>ping www.google.it
Esecuzione di Ping www-cctld.l.google.com [173.194.35.159] con 32 byte di dati:
Risposta da 173.194.35.159: byte=32 durata=39ms TTL=54
Risposta da 173.194.35.159: byte=32 durata=39ms TTL=54
Risposta da 173.194.35.159: byte=32 durata=41ms TTL=54
Risposta da 173.194.35.159: byte=32 durata=39ms TTL=54

Statistiche Ping per 173.194.35.159:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
C:\windows>
```

3.3 netstat

Il programma netstat serve per visualizzare le **connessioni attive** sull'host su cui si esegue il comando, ovvero quante porte TCP (e UDP) sono attivate in attesa di ricevere pacchetti o in procinto di spedirne.

Con netstat si possono visualizzare tutti i numeri di porta TCP (ma anche UDP) utilizzate sull'host:

LINUX

```
linux:~# netstat -n
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 192.168.0.100:41569     192.168.0.122:445       ESTABLISHED
tcp        0      1 192.168.10.2:44758      193.206.139.37:80       SYN_SENT
tcp        0      0 192.168.10.2:143        37.101.255.198:44749    ESTABLISHED
tcp        0     26 192.168.10.2:143        37.101.255.198:47960    FIN_WAIT1
tcp        0      0 192.168.10.2:143        93.47.33.246:1708       ESTABLISHED
linux:~#
```

In questo caso l'host Linux sta utilizzando varie porte TCP, alcune porte **note** (80, 143 e 445), una **registrata** (1708) e le rimanenti **dinamiche**[NB].

Tra le porte note ricordiamo la **porta numero 80**, riservata per applicazioni server Web.

Lo stesso applicativo per Windows ha un output più sintetico ma analogo:

WINDOWS

```
C:\windows>netstat
Connessioni attive

  Proto  Indirizzo locale       Indirizzo esterno       Stato
  TCP    127.0.0.1:4573         DellE4310:49156         ESTABLISHED
  TCP    127.0.0.1:49156        DellE4310:4573          ESTABLISHED
C:\windows>
```

In questo caso si notano una porta **registrata** (4573) e una porta **dinamica** (49156).

3.4 arp

Il programma arp usa l'omonimo protocollo di TCP/IP (RFC 826) per visualizzare la tabella delle corrispondenze tra indirizzi fisici MAC e indirizzi IP. È anche in grado di risolvere gli indirizzi IP in nomi di dominio.

Per il sistema operativo Linux il comando va digitato specificando il parametro **-n** se non si vogliono risolvere gli indirizzi:

LINUX

```
linux:~# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.0.138            ether   00:21:04:04:e5:ed   C                     eth0
192.168.0.176            ether   04:46:65:4c:2d:84   C                     eth0
192.168.0.130            ether   00:15:f2:0a:67:51   C                     eth0
linux:~#
linux:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
A580-IP.apogeo.com       ether   00:21:04:04:e5:ed   C                     eth0
android-954f93f671fa1b3   ether   04:46:65:4c:2d:84   C                     eth0
master.apogeo.com        ether   00:15:f2:0a:67:51   C                     eth0
linux:~#
```

[NB] Si può notare come il sistema operativo Linux usi come porte dinamiche anche numeri di porte registrate. Infatti le porte dinamiche vanno da 49151 fino a 65535, mentre i numeri precedenti fino a 1024 sono numeri di porte registrate. Questo è possibile perché all'interno dei numeri di porte registrate alcuni gruppi sono *Unassigned* dallo IANA, e quindi risultano disponibili.
Il sistema operativo Windows invece si comporta regolarmente.

Nella versione Windows del comando è necessario specificare il parametro –a:

WINDOWS

```
C:\windows>arp -a
Interfaccia: 192.168.0.17 --- 0xb
  Indirizzo Internet    Indirizzo fisico      Tipo
  192.168.0.1           00-1f-c6-24-7c-15     dinamico
  192.168.0.200         00-1f-3c-75-63-be     dinamico
  192.168.0.255         ff-ff-ff-ff-ff-ff     statico
  224.0.0.22            01-00-5e-00-00-16     statico
  224.0.0.252           01-00-5e-00-00-fc     statico
  239.255.255.250       01-00-5e-7f-ff-fa     statico
C:\windows>
```

Il protocollo **ARP** (*Address Resolution Protocol*), invece, ha lo scopo di calcolare l'indirizzo MAC a partire da un indirizzo IP.

3.5 traceroute/tracert

L'applicazione consente di ricavare il percorso seguito dai pacchetti per giungere a destinazione, riportando gli indirizzi IP dei nodi intermedi attraversati.

Per il sistema operativo Linux il comando è *traceroute* e il parametro classico è l'indirizzo IP da raggiungere o il suo nome di dominio:

LINUX

```
linux:~# traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 64 hops max
 1   81.174.0.1 (81.174.0.1) 54.076ms 17.246ms 17.104ms
 2   81.174.0.188 (81.174.0.188) 26.135ms 17.541ms 17.412ms
 3   217.29.66.96 (217.29.66.96) 37.345ms 18.488ms 33.017ms
 4   216.239.47.128 (216.239.47.128) 18.481ms 17.992ms 19.437ms
 5   72.14.232.78 (72.14.232.78) 50.209ms 26.910ms 27.726ms
 6   209.85.254.112 (209.85.254.112) 41.976ms 28.078ms 87.643ms
 7   8.8.8.8 (8.8.8.8) 27.037ms 27.828ms 28.492ms
linux:~# traceroute  google-public-dns-a.google.com
traceroute to google-public-dns-a.google.com (8.8.8.8), 64 hops max
 1   81.174.0.1 (81.174.0.1) 21.835ms 33.995ms 17.133ms
 2   81.174.0.188 (81.174.0.188) 17.101ms 19.149ms 18.363ms
 3   217.29.66.96 (217.29.66.96) 33.355ms 18.045ms 17.250ms
 4   216.239.47.128 (216.239.47.128) 18.140ms 45.403ms 18.247ms
 5   72.14.232.78 (72.14.232.78) 56.806ms 51.376ms 27.893ms
 6   209.85.254.112 (209.85.254.112) 27.688ms 42.558ms 27.360ms
 7   8.8.8.8 (8.8.8.8) 38.249ms 27.329ms 27.792ms
linux:~#
```

Per il sistema operativo Windows il comando equivalente si chiama *tracert* e l'uso è analogo:

WINDOWS

```
C:\windows> tracert 8.8.8.8
Traccia instradamento verso google-public-dns-a.google.com [8.8.8.8]
su un massimo di 30 punti di passaggio:

  1    <1 ms     *        <1 ms  Myxp.mshome.net [192.168.0.1]
  2     *        *         *     Richiesta scaduta.
  3    21 ms    22 ms     22 ms  172.18.9.161
  4    22 ms    23 ms     22 ms  172.18.8.21
  5    75 ms    33 ms     54 ms  172.17.10.57
  6    28 ms    27 ms     42 ms  172.17.10.73
  7    31 ms    30 ms     27 ms  pos1-10-0-0.milano50.mil.seabone.net [93.186.128.101]
  8    27 ms    27 ms     27 ms  74.125.51.12
  9    62 ms    63 ms     72 ms  66.249.94.235
 10    37 ms    37 ms     37 ms  72.14.232.76
 11    36 ms    37 ms     37 ms  209.85.254.116
 12     *        *         *     Richiesta scaduta.
 13    84 ms    82 ms     85 ms  google-public-dns-a.google.com [8.8.8.8]
Traccia completata.
C:\windows>
```

4 Cablaggio strutturato

Sia dovendo organizzare una piccola rete LAN/WAN casalinga, sia a maggior ragione dover organizzare una LAN/WAN per un ufficio o un'azienda, ciò comporta affrontare il problema della cablatura dei sistemi all'interno dell'area interessata.

Utilizzare fili volanti o canaline; nascondere i cavi nei battiscopa e sotto i tappeti, forare muri per far passare cavi, decidere dove collocare il router, lo switch e il modem anche rispetto alla presa della linea telefonica, sono tutte operazioni che riguardano l'organizzazione materiale di un sistema di comunicazione all'interno di un ambiente.

Per **cablaggio strutturato** si intendono quindi tutte quelle soluzioni che vengono adottate negli edifici moderni (e non) per consentire l'allestimento di un sistema di comunicazioni destinato sia ad una rete locale, ma anche telefonica, audio-video, per sistemi domotici, videosorveglianza, ecc.

Il cablaggio strutturato viene regolamentato da numerose normative tecniche, quasi tutte facenti capo agli standard di riferimento **EIA/TIA 568** (una normativa statunitense) e **ISO/IEC DIS 11801** (una normativa internazionale). In Italia le norme suddette sono recepite dal CEI (Comitato Elettronico Italiano).

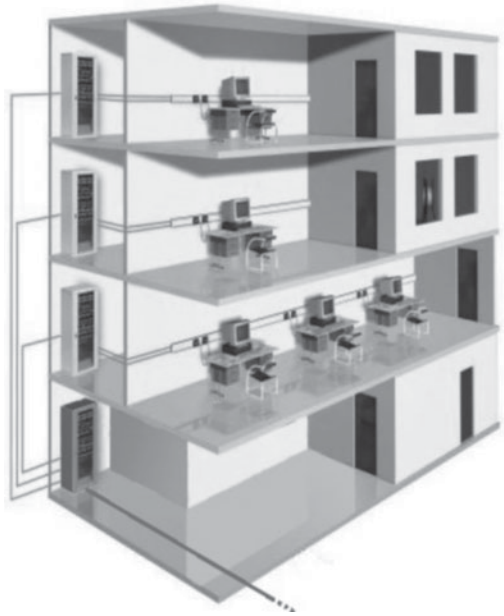
Questi standard descrivono i requisiti per il cablaggio in aree che tecnicamente sono denominate **campus**, ovvero una zona generica costituita da un insieme di edifici disposti su un'area privata; essi inoltre isolano cinque sottosistemi di cablature:

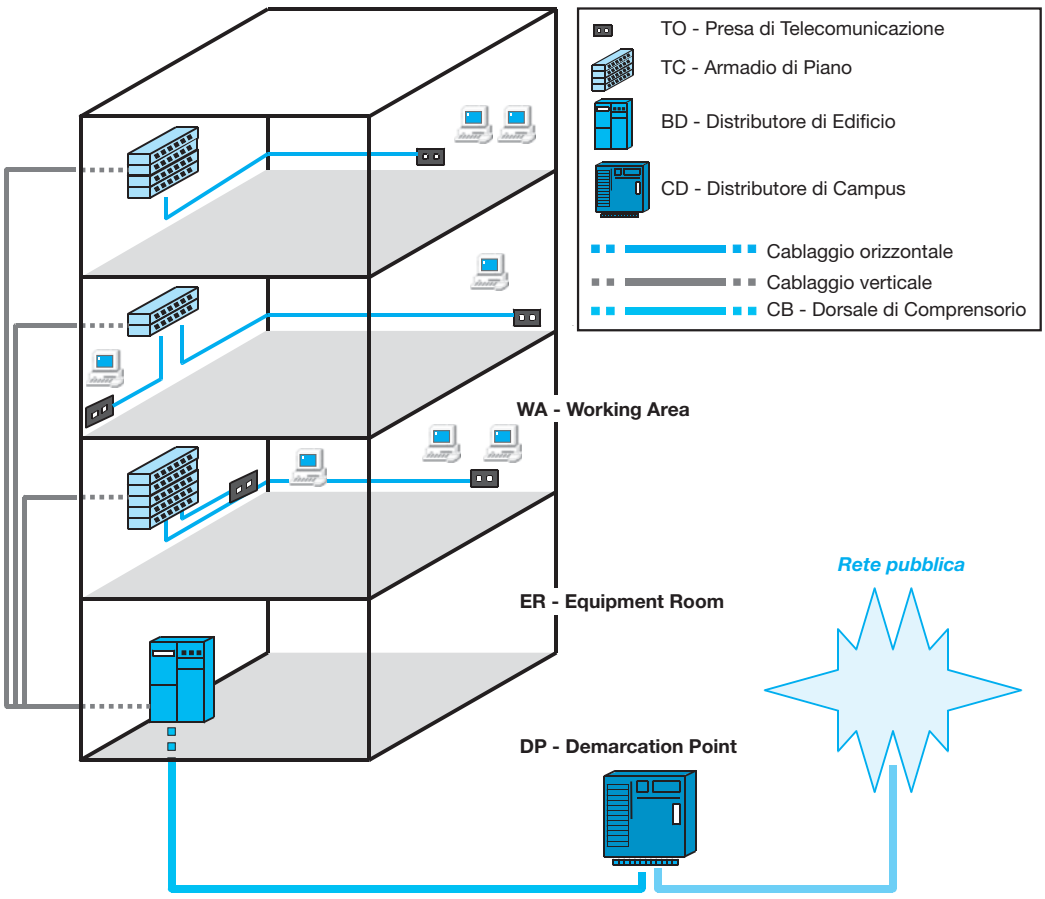
- L'**area di lavoro** (*working area*), la singola stanza in cui si trovano i dispositivi in rete.
- Il **cablaggio orizzontale**, che si occupa della cablatura situata su ogni singolo piano di un edificio del campus.
- Il **cablaggio verticale**, che si occupa della cablatura tra i vari piani di un edificio del campus.
- La **dorsale** (*backbone*), cioè i collegamenti tra gli edifici del campus.
- Il **punto di demarcazione**, cioè la zona in cui risiede l'interfaccia tra la rete privata del campus e i servizi esterni pubblici.

Tra gli elementi classici considerati da un cablaggio strutturato riconosciamo l'**Armadio di Piano** (TC), o armadio centro stella, una struttura a rack che contiene gli elementi attivi (esempio, switch e router) ed elementi passivi (**patch panel**); le **Prese di Telecomunicazione** (TO), cioè i terminali collocati nell'area di lavoro (WA) a cui si collegano i dispositivi finali (esempio, personal computer).

Ogni edificio dovrebbe poi avere un locale detto **Sala Apparati** (ER) contenente il **Distributore di Edificio** (BD), da cui parte la dorsale verticale verso i piani.

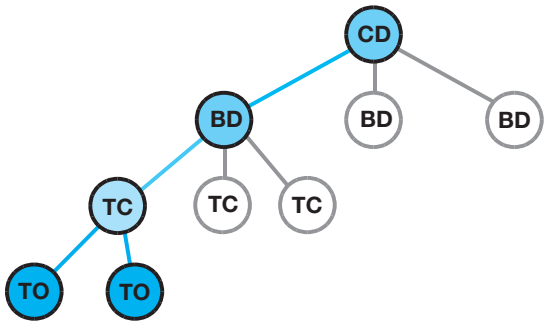
A loro volta le Sale Apparati sono connesse tramite una **Dorsale di Comprensorio** (CB) al **Punto di Demarcazione** (DP) su un armadio di apparati detto **Distributore di Campus** (CD).





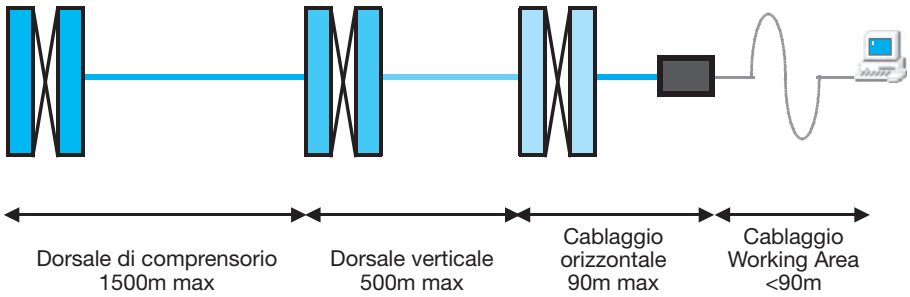
Il cablaggio strutturato presenta quasi sempre una **topologia ad albero**, in cui la radice è il *punto di demarcazione* DP (per esempio, dove si trova il router su rete pubblica), mentre i nodi intermedi sono i *distributori di edificio* BD e i *nodi terminali* TC gli armadi di piano (esempio, switch).

Le foglie, infine, sono i terminali di rete (per esempio, i computer) connessi alle *prese di telecomunicazione* TO.



Per tutti questi elementi, ma anche altri che non sono stati riportati, le normative citate precedentemente stabiliscono ogni caratteristica fisica, dalla forma alle dimensioni al tipo di cablaggio, specificando anche i parametri minimi richiesti in termini di errore tollerabile per ogni singola cablatura.

In particolare è abbastanza utile ricordare le prescrizioni sulle distanze ammesse per i cablaggi delle dorsali di comprensorio e verticali e per il cablaggio orizzontale.



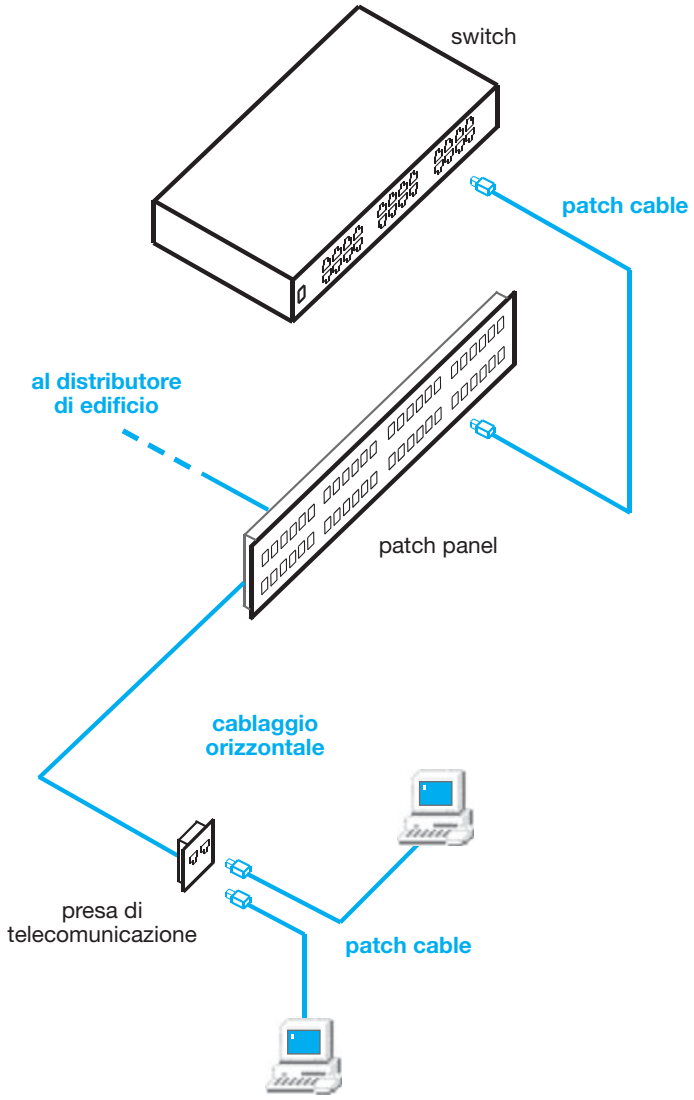
4.1 Cablaggio orizzontale

Una nota va fatta per quanto riguarda il cablaggio orizzontale, quello più direttamente a contatto con l'utente.

Tra i cavi previsti vengono utilizzati i cosiddetti **patch cable**.

All'interno degli armadi di rete, infatti, abbiamo almeno due tipi di dispositivi, quelli **attivi** come gli switch e quelli passivi come i *selezionatori* o **patch panel**. I patch panel sono pannelli multipresa a cui fanno capo i collegamenti *permanenti* o primari, come per esempio le dorsali. Le prese dei patch panel sono ordinate e ben etichettate, per contrastare l'effetto «spaghetti wiring» all'interno degli armadi.

Dalle porte dei patch panel vengono effettuati i collegamenti con le porte degli apparati attivi tramite patch cable. Il patch cable viene anche utilizzato per il cablaggio orizzontale, dalle prese di telecomunicazione al dispositivo di rete (per esempio, il computer).



5 Mezzi, connettori, cablaggi

I mezzi fisici utilizzati nelle reti di calcolatori si distinguono tra mezzi costituiti da **cavi in rame**, **cavi in fibra ottica** o senza cavo (**wireless**).

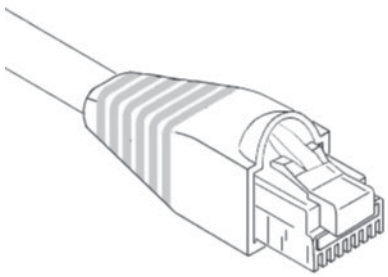
Nelle reti locali sono generalmente utilizzati cavi in rame, a volte per connettere gli switch si usano cavi in fibra ottica mentre per le stazioni mobili si usano connessioni wireless.

5.1 Cavi in rame

Rimanendo all'interno della rete LAN più diffusa al mondo cioè Ethernet, o standard **IEEE 802.3** e successivi, i cavi in rame sono denominati

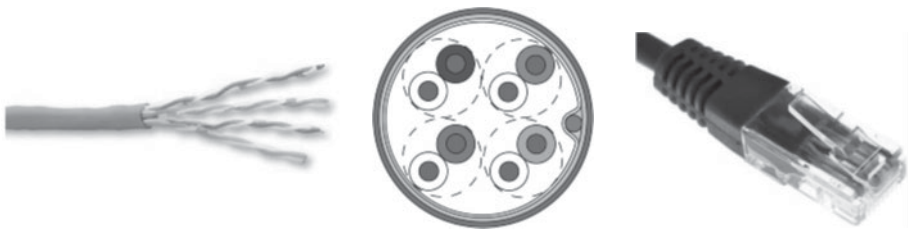
doppini (*twisted pair*) costituiti da quattro coppie di otto fili di rame intrecciati, e sono classificati in categorie, dalla Categoria 1 (usata per i telefoni standard), alla Categoria 7 per le LAN/MAN più veloci. A seconda della schermatura del doppino, e quindi a seconda della sua resistenza alle interferenze elettromagnetiche, si possono acquistare cavi UTP (*Unshielded Twisted Pair*), FTP (*Foiled Twisted Pair*) e STP (*Shielded Twisted Pair*).

I doppini usano un connettore denominato **RJ45**, del tipo *8P8C* ovvero a 8 posizioni e 8 contatti, collegati in **modo dritto** (*pin to pin*) o in **modo incrociato** (*crossover*), in questo caso per consentire di connettere due calcolatori tra di loro senza usare uno switch.

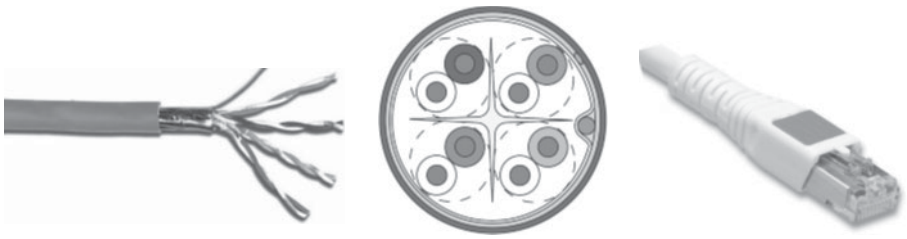


Le categorie di doppino usate nelle reti locali sono le seguenti:

- **Categoria 5**, la più utilizzata, per velocità classiche a 100Mbit/s per tecnologie denominate 100BASE-T (**FastEthernet**).



- **Categoria 6**, per velocità alte fino a 1000Mbit/s (1Gbit/s), per tecnologie denominate 1000BASE-T (**GigaEthernet**).



Alternative name

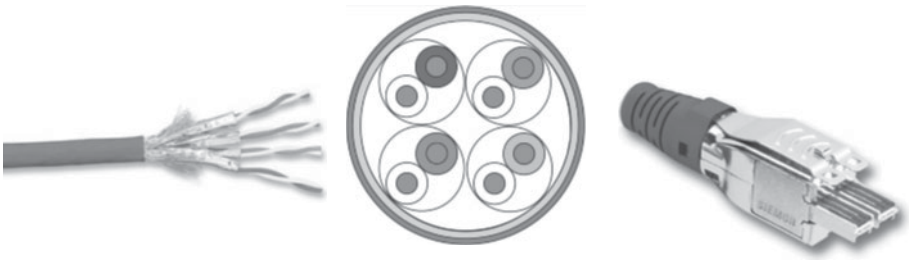
Molto spesso nel gergo delle reti locali si usano nomi speciali per indicare le varie tecnologie trasmissive di Ethenet 802.3 (e successive).

Queste sigle, stabilite negli standard IEEE, sono dette **Alternative Name** e hanno una forma autodescrittiva del tipo [*velocità*][*tipo*]-[*mezzo*], come ad esempio 100BASE-TX o 1000BASE-FX. I numeri 100 e 1000 specificano il bit rate (100 Mbit/s e 1000 Mbit/s), BASE significa baseband (sigla fissa per le LAN/WAN), mentre T in genere significa doppino (*twisted*) e F fibra ottica (*fiber*).

Gli alternative name non usano sigle rigorose. Ad esempio **100BASE-T** è un nome collettivo per tutte le tecnologie **FastEthernet** (100Mbit/s) sia su rame che su fibra, mentre **1000BASE-T** indica solo **GigaEthernet** (1Gbit/s) su rame, con le relative tecnologie su fibra ottica che prendono i nomi di 1000BASE-FX, 1000BASE-LH, 1000BASE-LX (L sta per Long, fino a 5000m) e 1000BASE-SX (S sta per *Short*, entro 500m).

Sono oramai abbandonati i nomi 10BASE2 e 10BASE5 ove i numeri finali indicavano le distanze massime (in centinaia di metri) del cavo coassiale utilizzato.

- **Categoria 7**, per l'ultimo e più veloce standard per LAN/MAN, fino a 10000Mbit/s (10 Gbit/s), per tecnologie denominate 10GBASE-T (**TeraEthernet**).



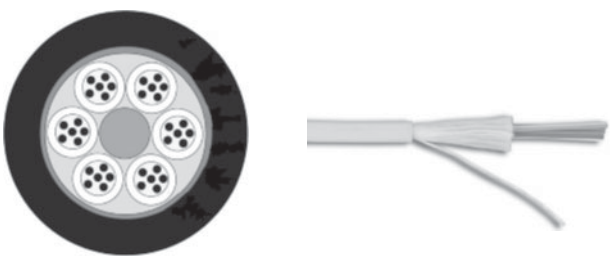
Il doppino è l'ideale per il *cablaggio orizzontale* e, in quasi tutti i casi descritti, i tratti di doppino (patch cable) non possono superare il centinaio di metri di lunghezza.

5.2 Cavi in fibra ottica

I mezzi in fibra ottica sono utilizzati per ottenere grandi larghezze di banda (≥100 Mbit/s, fino a 10Gbit/s), per tratte medio-alte (≥100 m e anche sull'ordine dei km), magari in ambienti dove è importante l'isolamento dai disturbi elettromagnetici. Queste caratteristiche sono garantite dai mezzi in fibra ottica (mono o multimodali), che si possono distinguere in:

- **Tight buffer**, cavi realizzati con materiali più indicati per una posa all'interno degli edifici e per distanze più ridotte. Mancando il gel di protezione all'interno, le connettorizzazioni sono più semplici; supportano tecnologie 1000BASE-SX, 10GBASE-L, 10GBASE-E.

- **Loose tube**, cavi realizzati con materiali più indicati per installazioni in ambienti esterni o canalizzazioni interrate, con guaine rinforzate e presenza di gel per prevenire infiltrazioni. Supportano tecnologie 1000BASE-FX, 10GBASE-LX4, 10GBASE-LX.



Ogni singola fibra è in grado di veicolare un solo senso della comunicazione (simplex) quindi i cavi sono sempre composti da un numero di fibre pari, in modo tale che ogni coppia possa realizzare una comunicazione full duplex (trasmissione e ricezione contemporanee).

I cavi in fibra ottica usano molti tipi di connettori tra i quali i più diffusi sono: **ST** (*Straight Tip*), il primo ad essere utilizzato, ora in disuso; **SC** (*Standard Connector*), il connettore attualmente più utilizzato, **MT-RJ** (*Mechanical Transfer Registered Jack*, simile all'RJ45), il probabile standard futuro.

Nelle LAN la fibra ottica è utilizzata spesso per realizzare le *dorsali*, sia di *comprensorio* che eventualmente le *dorsali verticali*.

5.3 Wireless

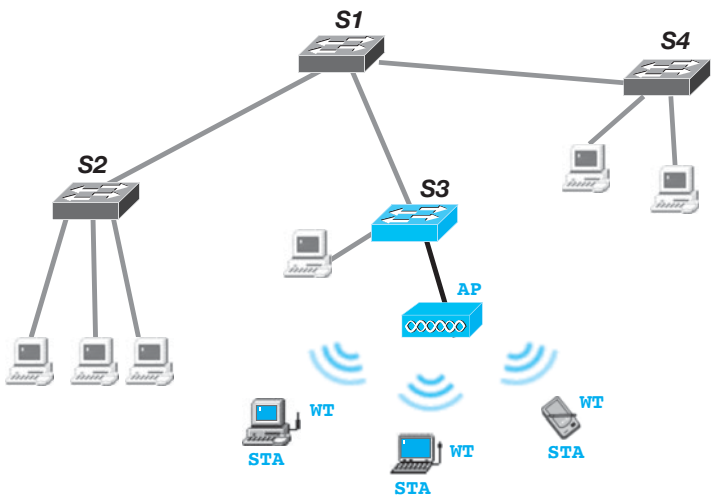
Per effettuare un internetworking fisico completo una LAN può dotarsi di connessioni senza cavo denominate **Wi-Fi**, uno standard che a partire dalla versione **IEEE 802.11** arriva a raggiungere velocità fino a 52 Mbit/s e oltre (con la versione 802.11n).

Una connessione Wi-Fi per rete LAN consiste di due apparati.

L'**Access Point** (**AP**), che ha una normale connessione cablata Ethernet 802.3 per uno switch, ma che funge anche da base radio per il modulo **Wireless Terminal** (**WT**), normalmente collocato come periferica di un sistema detto **STA**.

I computer portatili hanno il modulo wireless WT integrato, mentre un personal computer desktop deve dotarsi di un modulo WT, per esempio realizzato attraverso una «chiavetta» USB.

Il sistema STA dotato del modulo WT si comporta come un normale sistema connesso allo switch su cui l'Access Point relativo è connesso. Un Access Point può fornire l'accesso alla LAN per una trentina di client WT differenti operanti nella sua cella radio.



Siccome la comunicazione avviene tramite onde radio, essa può essere messa in crisi da particolari condizioni ambientali (per esempio, ostacoli come alberi, pilastri, muri di edifici), cosicché la portata dei dispositivi WiFi non sempre rispetta i valori promessi, che sono 30 m per ambienti interni e 100 m per ambienti esterni.

Ogni Access Point è riconoscibile tramite un nome sottoforma di stringa descrittiva (**SSID**, *Service Set Identifier*) in modo tale che un client possa consultare l'elenco dei SSID presenti nel suo raggio d'azione.

Le connessioni wireless, per loro natura, sono vulnerabili, cioè si prestano ad essere violate sia per sottrarre dati, sia per utilizzare le connessioni abusivamente. Lo standard Wi-Fi prevede quindi l'impostazione di una **chiave privata** che Access Point e client WT devono condividere e che consente la criptazione dei pacchetti con algoritmi di tipo **WPA2**, ad oggi ancora inviolati a differenza degli algoritmi basati su WEP e WPA. In questo caso si parla di reti Wi-Fi protette.

6 Mezzi e sistemi per una WAN

In una rete WAN i mezzi utilizzati sono molto più eterogenei e relativamente meno standardizzati. Data l'estensione nazionale e sovranazionale della rete Internet gli standard di collegamento sono spesso molto complessi e la infrastruttura di supporto esula dagli obiettivi di questa trattazione. Ciononostante è interessante indicarne alcuni elementi di base per avere un'idea di come i sistemi intermedi di una WAN organizzano le proprie connessioni su scala geografica.

Inoltre è abbastanza utile distinguere le interconnessioni delle WAN in due sezioni ben distinte, la parte di interconnessione privata e la parte di interconnessione pubblica.

Per quanto riguarda le interconnessioni interamente pubbliche, dorsali o linee punto-punto per collegamenti geografici, si distinguono due grosse categorie di protocolli fisici adottati, entrambe in condivisione di tempo, il modello **plesiocrono** (PDH, *Plesiochronous Digital Hierarchy*) e il modello **sincrono** (SDH, *Synchronous Digital Hierarchy*).

6.1 Infrastruttura pubblica

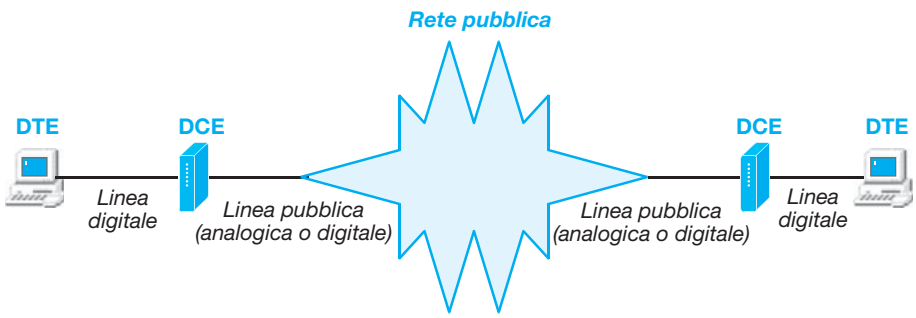
Nel modello PDH gli apparati possono veicolare 30 gruppi di dati da 64kbit/s, per un totale di 2Mbit/sec (detta **trama E1** in Europa e **T1** negli USA) i quali sono raggruppati 4 alla volta per ottenere un flusso da 8Mbit/sec, costruito recuperando i bit uno a uno dai quattro flussi originari con un'operazione di multiplazione. Con tecniche analoghe si raggruppano flussi fino a 140Mbit/s (**trama E4**). Lo svantaggio della tecnica PDH risiede nelle continue operazioni di multiplazione e demultiplazione per raggruppare ma anche accedere alle informazioni nei singoli flussi originali da 2Mbit/sec. Anche per questo motivo i sistemi PDH spesso oggi vengono sostituiti da sistemi SDH in numerose reti di telecomunicazioni. I formati

di multiplazione PDH sono definiti nella raccomandazione ITU-T G.702. Le caratteristiche elettriche e fisiche delle interfacce PDH sono invece specificate nella raccomandazione ITU-T G.703, come il formato delle trame E1-E4.

Il modello SDH non ha i limiti di PDH in termini di carico e overhead di multiplazione e demultiplazione. Gli apparati, con codifiche denominate STM (*Synchronous Transport Module*), possono trasportare 63 flussi a 2 Mbit/s, oppure 3 flussi a 34 Mbit/s o ancora 3 flussi a 45 Mbit/s o infine 1 solo flusso a 140Mbit/s nel caso di STM1. Con le versioni più evolute, si ottengono flussi da 40Gbit/sec. Il protocollo SDH è diffuso soprattutto in Europa e in Asia. In America viene invece usato un protocollo molto simile, il **Sonet**, che segue uno standard proprietario. Il protocollo SDH è standardizzato dall'ITU-T nelle normative G.707 e G.708.

6.2 Infrastruttura privata

Esistono varie modalità d'accesso alle linee pubbliche e quindi alla rete pubblica, generalmente Internet. Quasi tutte queste modalità necessitano di un elemento hardware intermedio che deve trasformare i segnali digitali del sistema (per esempio, personal computer) in segnale opportuno per il mezzo (per esempio, il mezzo di telefonia pubblica, che spesso è analogico).



In questi casi si adotta la terminologia **DTE-DCE** per indicare, rispettivamente, il sistema digitale (**DTE**, *Data Terminal Equipment*) e il sistema di codifica/decodifica (**DCE**, *Data Communication Equipment*).

Nella maggior parte dei casi il DTE coincide con un **computer**, mentre il DCE con un **modem**.

Si possono distinguere varie tecnologie di accesso alle reti pubbliche, ovvero di tipo telefonico (PSTN, ISDN), a pacchetto (X.21/X.25, Frame Relay e ATM), di tipo xDSL, via radio wireless (UMTS, WiMAX) o satellitare. Spesso queste tecnologie si mescolano, per utilizzare al meglio e con minor spesa apparati e linee preesistenti. Esempio, i CDA non sono quasi più utilizzati come connessioni analogiche ma come supporto a HDSL, mentre in molti casi l'accesso su linea pubblica, ovvero dal DCE verso la linea pubblica, avviene secondo il modello plesiocrono in accordo con gli standard G.703/704.

Di fondamentale importanza, quando si parla di accesso a linee pubbliche è il problema dell'**ultimo miglio** (*subscriber line*), ovvero della tratta

pubblica che connette l'utente residenziale con la centrale di smistamento pubblico, spesso una centralina telefonica di quartiere.

Essendo il tratto più diffuso su scala geografica e il meno adattabile a modifiche o sostituzioni, il problema delle tecnologie adottate sulle linee *subscriber* obsolete è fondamentale.

La rivoluzione dell'accesso **ADSL**, per esempio, ha risolto il grosso problema dell'ultimo miglio che collega le abitazioni private alle centraline telefoniche di quartiere, essendo in grado di riutilizzare la connessione di rame preinstallata per l'utenza telefonica come veicolo per l'alta velocità delle reti digitali.

ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1. Topologie e modi di circolazione dei pacchetti su reti LAN e su reti WAN. Ricordare i nomi degli apparati di interconnessione.
2. Illustrare dimensione, notazione e univocità del MAC address.
3. Mettere a confronto l'uso degli Indirizzi fisici e degli indirizzi di livello 3 nei pacchetti circolanti su una rete WAN.
4. Elencare i principali programmi multipiattaforma per analizzare indirizzi e flussi di pacchetti su una rete.
5. Mostrare uso e funzionalità del programma applicativo ping. Fare un paio di esempi con argomenti differenti.
6. Commentare le locuzioni «risoluzione dei nomi» e «risoluzione di indirizzi IP».
7. Dimostrare di sapere come ottenere gli indirizzi IP e gli indirizzi MAC di un nodo di cui si ha accesso.
8. Elencare le due normative di riferimento per il cablaggio strutturato.
9. Disegnare uno schema in cui sono messi in evidenza cablaggio orizzontale e verticale, dorsali e armadi di piano.
10. Elencare il connettore e le categorie più diffuse di cavi in rame indicandone il bitrate supportato.
11. Spiegare in quali casi di cablaggio strutturato si usano cavi in fibra e per quali ragioni.
12. Elencare gli elementi base di una rete WiFi e commentarne le funzioni.
13. Commentare le sigle DTE e DCE, e la locuzione «subscriber line».

Requisiti avanzati

1. Illustrare le topologie di interconnessione tra host e tra switch in una LAN e la topologia di interconnessione tra router in una WAN.
2. Calcolare quante aziende possono produrre schede di rete e quante schede di rete può produrre un'azienda.
3. Livelli OSI sufficienti in una LAN e necessari in una WAN: commentarne le ragioni.
4. Disegnare il diagramma del percorso di un pacchetto originato da un host A di una LAN che raggiunge un host Z su un'altra LAN attraverso una WAN. Usare i simboli CISCO.
5. Mostrare come è possibile verificare che un nodo di una rete è connesso a Internet.
6. Mostrare quali nodi si attraversano per raggiungere il sito *www.rai.tv.*
7. Mostrare con esempi la risoluzione di due nomi di dominio e la risoluzione di due indirizzi IP.
8. Mostrare i principali limiti di lunghezza per i cavi utilizzati in un cablaggio strutturato, compresi i patch cable.
9. Illustrare le caratteristiche, i limiti e le vulnerabilità di un sistema WiFi.
10. Spiegare come sia possibile connettere due PC con 802.3 senza usare uno switch.
11. Elencare gli Alternative Name più diffusi e indicarne le caratteristiche.
12. Spiegare perché i cavi in fibra hanno sempre un numero di fibre pari ed elencarne le due tecnologie commerciali più usate.
13. Elencare i principali tipi di trame usate nell'infrastruttura di rete pubblica.

Il livello 1: Fisico

B4

1 Protocolli di accesso per reti WAN

Sono disponibili varie modalità con cui è possibile l'interconnessione tra una rete LAN e una rete WAN, tipicamente Internet. Tutte le modalità elencate rispettano il modello DTE-DCE, ove il DCE (modem) determina la qualità e il tipo di connessione.

1.1 PSTN (o dialup)

PSTN sta per *Public Switched Telephone Network* e consiste di una modalità di comunicazione a commutazione di circuito. Si tratta della forma base del collegamento alla linea pubblica, che viene effettuato con un DCE modem in banda fonica (il modem propriamente detto) sulla linea commutata della telefonia analogica tradizionale, detta anche POTS (*Post Office Telephone System*).

La linea che collega il DTE (computer) al DCE (modem) è tipicamente una linea seriale asincrona full duplex in base allo standard **EIA Rs232C**, che può raggiungere la velocità di 115200 bit/s. Lo standard EIA Rs232C è stato poi inserito nei cosiddetti *standard V* dell'ITU con il nome di V.24/V.28.

Sull'altro lato, verso la rete pubblica, la comunicazione si basa su altri standard V dell'ITU, dal **V.21** fino al **V.90**.

In questo senso le velocità massime ottenibili con modem classici valgono 4800 baud (baud = numero di modulazioni al secondo ottenibili sulla banda analogica disponibile sul cavo telefonico). Con tecniche di modulazione speciali, che consentono di associare più bit ad ogni baud, si ottengono velocità superiori, fino a 28800 bit/s (con compressione dati anche a velocità di 38400 bit/s). L'ultimo standard V commercializzato, il V.90, raggiunge la velocità di 57600 bit/sec (56kbit/s) associando 12 bit per ogni baud.

Le connessioni PSTN vengono utilizzate solo da utenze residenziali, per esempio laddove il servizio DSL non è attivo.

La mancata copertura del servizio ADSL su un territorio dà origine al divario digitale (*digital divide*) tra le persone. In Italia da tempo si discute su come offrire la banda larga alla maggior parte della popolazione.

Comunicazioni seriali asincrone Rs232

Lo standard di comunicazione tra calcolatori più longevo è sicuramente il protocollo seriale asincrono **Rs232** (EIA Rs232C), proposto addirittura nel 1962.

La ragione del suo successo è dovuta al fatto che con soli tre canali (modo null modem) è in grado di veicolare dati full duplex su distanze anche di qualche decina di metri senza altri ausili.

Come canale trasmissivo è oggi in disuso, rimanendo tuttavia ancora utilizzato in applicazioni industriali.

Recentemente ha ricevuto nuovo slancio dovuto al fatto che quasi tutti i linguaggi di programmazione contengono le librerie per impostarne il protocollo (di livello L1 Fisico e L2 Datalink) e per spedire e ricevere pacchetti su **porte seriali** virtuali (COM) realizzate su USB o Bluetooth.

Il **protocollo** classico indica velocità, tipo di parità, ampiezza del pacchetto, n. di bit di stop, come per esempio **9600,n,8,1** (9600 bps, nessuna parità, 8 bit di dato, 1 bit di stop).

ESEMPIO

Protocollo fisico per Rs232

Una interfaccia seriale EIA Rs232C deve spedire il byte del codice Ascii della lettera A (maiuscola). Rappresentare l'onda quadra simbolica del pacchetto completo di livello 1 se il protocollo vale 9600, n, 8, 1.

Il codice Ascii della lettera A (maiuscola) vale 41h, ovvero (1000001)b.

Siccome il protocollo da utilizzare prevede otto bit di dato, la sequenza dei Dati risulta essere (01000001)b.

Siccome Rs232 è una codifica a **logica negativa**, il valore del bit a 1 è basso (**–**), mentre il valore del bit a 0 è alto (**+**).

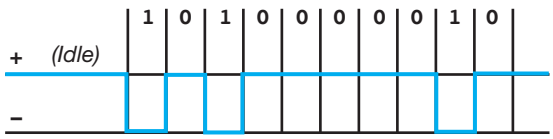
Siccome lo stato a riposo della linea (**Idle**) è sempre alto (**+**), il bit di start deve originare una transizione, cioè valere il suo contrario, ovvero sempre 1 (**–**).

Siccome il bit di stop deve garantire che la linea si trovi a Idle per almeno un tempo di bit (**bit time**) prima di ricevere il pacchetto seguente, il bit di stop vale sempre 0 (**+**), cioè come Idle.

Dato che il protocollo non prevede bit di parità e un solo bit di stop, l'intero pacchetto fisico è composto da 10 bit; il pacchetto di livello 1 Fisico, completo di Intestazione, Dati e Coda è il seguente:

Intestazione	*Dati*	*Coda*
start	'A'	stop
1	**01000001**	**0**

Lo schema simbolico dell'onda quadra corrispondente:



PROGRAMMA

Stampa di onda quadra simbolica

Scrivere un programma in linguaggio C che stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii esteso.

Per lo scopo è sufficiente stampare con la funzione **printf()** i simboli dei codici Ascii estesi corrispondenti ai valori (196)d, (191)d, (192)d, (217)d e (218)d relativi ai simboli –, ┐, └, ┘, ┌. Per esempio:

```
00 #include <stdio.h>
01
02 int main(void)
03 {
04  printf("Caratteri:\t – ┐ └ ┘ ┌\n");
05  printf("Ascii:\t %c %c %c %c %c",196,191,192,217,218);
06  return 0;
07 }
```

L'ouptut però potrebbe diventare:

OUTPUT 1

```
Caratteri:  – ┐ └ ┘ ┌
Ascii:      – ┐ └ ┘ ┌
```

oppure

OUTPUT 2

```
Caratteri:  ÔöÇ ÔöÉ ÔöÖ Ôöÿ ÔöîÎ
Ascii:      – ┐ └ ┘ ┌
```

In questo secondo caso va impostato un set di caratteri conveniente nell'editor utilizzato.
Per esempio, con Notepad++ si imposta Formato/Set di caratteri/Europa occidentale/OEM 850.
In ogni caso per ottenere uno schema simbolico di onda quadra come riportata nell'**Esempio** precedente:

```
00 #include <stdio.h>
01
02 int main(void)
03 {
04  printf("Schema simbolico Rs232 9600,n,8,1 per 'A'\n");
05  printf("          \n");
06  printf("          \n");
07  return 0;
08 }
```

PROGRAMMA

oppure:

```
00 #include <stdio.h>
01
02 int main(void)
03 {
04  printf("Schema simbolico Rs232 9600,n,8,1 per 'A'\n");
05  printf("%c%c%c%c%c%c%c%c%c\n",191,218,191,218,196,196,196,196,191,218);
06  printf("%c%c%c%c%c%c%c%c%c\n",192,217,192,217, 32, 32, 32, 32,192,217);
07  return 0;
08 }
```

che danno origine:

OUTPUT

```
Schema simbolico Rs232 9600,n,8,1 per 'A':
┐┌──┐┌
└┘  └┘
```

1.2 ISDN

ISDN sta per *Integrated Services Digital Network*, anche in questo caso modalità in commutazione di circuito, ma su linea digitale.

La rete ISDN prevede due tipi di accesso: l'accesso base, principalmente concepito per l'utente finale, e l'accesso primario, destinato a centri a loro volta erogatori di servizi.

L'accesso base consiste in due canali a 64 kbit/s (detti canali B) e in un canale dati di servizio a 16 kbit/s (detto canale D). L'accesso base prevede una velocità di trasmissione di 192 kbit/sec, di cui 144 utilizzati per i 2 canali B e il canale D, e i restanti 48 per informazioni di controllo e di sincronismo. Trattandosi di comunicazione su linea digitale, il DCE non è necessariamente un modem, benché sia necessaria una apparecchiatura d'interfaccia per il collegamento alla rete ISDN. Esso è detto NT (*Network Termination*) o **borchia ISDN**, che, integrato con un TA (*Terminal Adapter*), completa il DCE lato utente. Il DCE ISDN può essere realizzato con varie tecnologie che supportano le interfacce digitali lato utente più comuni: EIA Rs232C (V.24), X.21, V.35 (con adattatore), USB, o addirittura realizzate su bus PCI o PcCard/PCMCIIA. I protocolli sul lato della centrale sono descritti nelle raccomandazioni I.430 e I.431 dell'ITU.

ISDN è in disuso, sostituita progressivamente dalle connessioni di tipo xDSL. A volte viene utilizzata come linea sostitutiva a queste.

1.3 CDN

CDN sta per *Canale Diretto Numerico*, tipo di connessione che non opera in commutazione di circuito su linee commutate ma su linee digitali dedicate appositamente prese in affitto dal gestore della telefonia pubblica.

Si tratta di un tipo di accesso digitale su linea dedicata, ove il DCE è detto anche Terminazione di rete CDN, che può fornire prestazioni dai 2 kbit/s ai 2Mbit/s. In alcuni casi, e con supporti adatti, la CDN può essere supportata

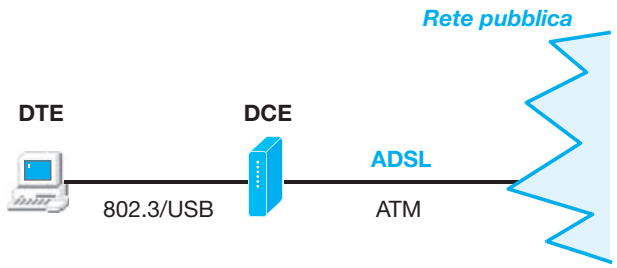
dalla cosiddetta CTN (*Canale Trasmissivo Numerico*) in grado di ottenere anche 155Mbit/s e oltre e realizzato in fibra ottica.

Anche per CDN il DCE non è propriamente un modem (a volte è detto modem digitale) e, dal lato utente, i protocolli utilizzati sul livello fisico partono dai protocolli seriali asincroni EIA Rs232C (V.24), protocolli seriali sincroni V.35, protocolli seriali evoluti EIA449, EIA530 e HSSI (*High Speed Serial Interface*).

Le connessioni CDN non sono utilizzate normalmente da utenti residenziali.

1.4 DSL

DSL sta per *Digital Subscriber Line* (o Loop), modo di accesso digitale alla rete pubblica su linee commutate effettuato sul doppino di rame della telefonia standard. Le versioni DSL sono **simmetriche** (HDSL, IDSL, SDSL) o **asimmetriche** (ADSL, RADSL, UADSL, VDSL) a seconda del fatto che la banda disponibile sia la stessa in trasmissione e ricezione (upstream e downstream), o maggiore in downstream.



Per **ADSL** (*Asymmetric DSL*), dedicata agli accessi residenziali, il DCE è comunemente detto **modem ADSL** e, verso la centrale, sfrutta le frequenze sopra i 4KHz (le stesse della filodiffusione), potendo così convivere con lo scambio dati fonico che usa le frequenze nell'intervallo dei 300-3400Hz.

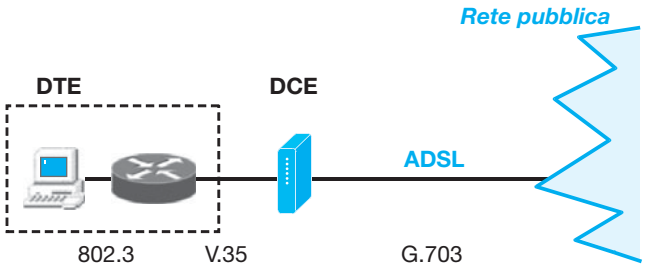
Per ottenere la separazione delle frequenze della fonia standard, va comunque installato un dispositivo di «splitter» sulla presa telefonica dell'utente (**POTS** splitter).

In Italia le velocità stanno aumentando e le offerte da 4, 6 e più Mbit/s sono comuni, ma stanno diventando sempre più frequenti anche le offerte **ADSL 2+**, a 12, 20 e 24 Mbit/s. Per ADSL questi valori indicano prestazioni nominali (PCR, *Peak Cell Rate*), dato che poi bisogna fare i conti con la condivisione del canale con gli altri utenti, la distanza dalla centralina di aggancio e l'**overbooking**. Infatti gli operatori telefonici spesso attestano sulle centraline un numero di utenti la cui somma di traffico medio garantito (MCR, *Minimum Cell Rate*) supera la larghezza di banda effettivamente fornita, cosicché non di rado i singoli utenti vengono privati della banda promessa.

Per quanto riguarda il lato DTE-DCE, la connessione ADSL utilizza linee Ethernet 802.3 a 10-100 Mbit/s con protocollo di livello 2 PPPoE (PPP over Ethernet) o linee USB con protocollo di livello 2 PPPoA (PPP over Atm). Più di rado viene utilizzato direttamente un protocollo IPoA (IP Over Atm). Si ricorda che per ognuna di queste tecniche di incapsulamento,

compresa PPPoE, la trama che giunge alla centrale è una trama ATM, infatti in ognuno di tali protocolli vanno specificate le coppie di parametri VPI/VCI che non sono altro che i valori di **VC** (*Virtual Circuit* ovvero circuito virtuale) e cella ATM. Gli standard ADSL sono contenuti nei documenti ITU-T G992.x

Per **HDSL** (*High Data Rate DSL*), dedicata agli accessi aziendali, le prestazioni in entrambi i versi sono garantite e più affidabili, tipicamente dai 2Mbit/s garantiti, fino a 8-16 Mbit/s, senza condivisione di banda con altri utenti, infatti HDSL è implementata su linee dedicate CDN o ISDN.



L'accesso HDSL utilizza un modello DTE-DCE mediato da un router dedicato con interfaccia Ethernet 802.3 o seriale sincrona V.35, pertanto il DCE non è collegato direttamente al PC.

Per HDSL il protocollo con la centrale si basa sugli standard G.703/704 del modello Plesiocrono con trame E1. Giunto alla centrale, il segnale G.703 viene convertito per passare su celle ATM. Gli standard HDSL sono contenuti nel documento ITU-T G991.x.

1.5 WiMAX

La tecnologia wireless **WiMAX** (*Worldwide Interoperability for Microwave Access*) consente l'accesso alla rete pubblica con velocità fino a 70 Mbit/s in aree metropolitane, larghezza di banda sufficiente per supportare simultaneamente almeno 40 aziende con connettività equivalente di tipo T1 (ovvero a 2Mbit/s). Gli elementi base per WiMAX sono la Base Station (BS), la Repeater Station (RS), la Subscriber Station (SS), i Terminal Equipment (TE).

Il Terminal Equipment è il classico DTE, mentre la Subscriber Station è il DCE, che comunica via radio con la Base Station collocata in un raggio contiguo, spesso nello stesso edificio, e su cui convergono vari SS.

La Base Station poi si connette, con varie tecnologie, anche su cavo (o con ripetitori radio come le Repeater Station), alla rete pubblica. La tecnologia consente connessioni tra utenze wireless nel raggio di 50 km. Il collegamento DTE-DCE sarà realizzato su Bus Pc (tramite PcCard PMCIIA) o con connessioni standard tipo USB.

Lo standard WiMAX si basa sui documenti ITU-T 802.16x.

1.6 Sim Card

Alternative al progetto wireless WiMAX sono le tecnologie basate sui sistemi di telefonia cellulare 3G (Terza generazione), come l'**UMTS** (*Uni-*

versal Mobile Telecommunications System) o 4G come **HSDPA** (*High Speed Downlink Packet Access*), a loro volta basati sui sistemi di 2G GSM o 2,5G come il **GPRS** (*General Packet Radio Service*).

La larghezza di banda di UMTS per l'accesso alla rete WAN è di circa 2Mbit/s, mentre HDSPA arriva a 10 Mbit/s.

Altra modalità di accesso alla rete pubblica tramite telefonia cellulare è il tipo **EDGE** (*Enhanced Data rates for GSM Evolution*) che amplia la larghezza di banda della connessione GPRS.

In tutti questi casi, se i servizi avanzati non sono disponibili o momentaneamente disattivati, il supporto viene garantito dalla connettività GPRS o GSM preesistente, anche se a velocità decisamente inferiori (da 60 a 200 kbit/s).

Il DCE è un modulo radio (implementato su PcCard PMCIIA o con connessione diretta standard tipo USB) che si accredita direttamente sulla rete di telefonia cellulare. In tutte queste tecnologie l'accesso prevede la presenza di una **Sim Card** sul DCE per l'autenticazione e la tariffazione.

I documenti standard che regolano gli standard per le comunicazioni cellulari sono emessi dall'ETSI e dal 3GPP (*3rd Generation Partnership Project*).

2 Protocolli di accesso per reti LAN

Codificare i singoli bit con segnali analogici (per esempio, di tensione), comporta associare un determinato livello di segnale analogico al valore di un determinato bit, all'interno di un modello di comunicazione seriale, ovvero su singolo canale. Questa operazione comporta un problema di sincronizzazione tra trasmettitore e ricevitore che non possono avere, per forza di cose, i componenti di clock sempre perfettamente fasati (sincronizzati) o i mezzi sempre esenti da disturbi o attenuazioni.

La teorica che descrive il rapporto tra la banda passante di un mezzo trasmissivo e la sua capacità di trasportare segnali è descritta dai **teoremi di Nyquist** (per un mezzo privo di disturbi) e di **Shannon** (per un mezzo con disturbi).

L'associazione tra segnale e bit deve essere ottenuta preservando la sincronizzazione anche e soprattutto in assenza di segnali dedicati alla sincronizzazione (**clock**). Inoltre la comunicazione dovrebbe essere ottenuta riducendo al minimo le caratteristiche dei componenti, sia per questioni di costi, sia per problemi legati alle infrastrutture. Ciò significa poter ottenere la maggiore larghezza di banda utilizzando la minor frequenza possibile degli apparati di comunicazione, oppure eliminare la componente continua del segnale.

La problematica maggiore è la sincronizzazione del segnale per ottenere la discriminazione del bit: il segnale deve essere costruito in modo tale che il ricevitore sappia quando la codifica di un singolo bit inizia e termina attraverso il segnale analogico, ovvero poter «leggere» il valore del segnale analogico nel momento giusto.

Ciò si ottiene con varie tecniche di sincronizzazione che fondono i se-

gnali di sincronizzazione all'interno dei segnali stessi che veicolano i dati, codifiche dette **clock & data encoding**.

La sincronizzazione viene ottenuta quando si hanno **transizioni** di livello del segnale, i fronti di salita e di discesa, normalmente presenti quando un bit passa da 0 a 1, o viceversa da 1 a 0.

PROGRAMMA

Conteggio delle transizioni

Scrivere un programma in linguaggio C che indichi quante transizioni ha un byte acquisito da tastiera.

Una volta acquisito un carattere da tastiera è necessario calcolarne la sua rappresentazione binaria su 8 bit, come richiesto.

Quindi vanno contate le transizioni, ovvero tutte le presenze di coppie binarie **01** o **10**.

Una versione del programma può essere la seguente:

```
#include <stdio.h>
#include <conio.h> // per getche()

#define CONTENITORE (8)
char bit[CONTENITORE+1]; // i bit da memorizzare

int main(void)
{
 char ch;
 int i,j,t,k;
 int q;
 int r;

 printf("Inserire un carattere: ");
 ch = getche();

 q = ch;
 i = 0;
 k = 0;
 do
 {
     r = q % 2;
     q = q / 2;
     bit[i] = r;
     i++;
     k++;
 }
 while (q); // finché la divisione non è zero

 // Riempio al CONTENITORE
 for (j=0; j<(CONTENITORE-k); j++)
 {
       bit[i++] = 0;
 }

 // Stampa bit al contrario e conteggio delle transizioni
 t = 0;
 printf("\nStampa binaria di '%c' al contrario:\t ",ch);
 for (i=0; i<CONTENITORE; i++)
 {
       printf("%c",bit[i] + '0');
       if (i>0)
          {
       if (bit[i]!=bit[i-1]) t++;
       }
 }

 // Stampa dei bit 'raddrizzati'
 printf("\nStampa binaria di '%c' al raddrizzata:\t ",ch);
 for (i=0; i<CONTENITORE; i++)
 {
```

PROGRAMMA

```
51          printf("%c",bit[CONTENITORE-i-1] + '0');
52      }
53
54      printf("\nLe transizioni sono:\t%d ",t);
55
56      return 0;
57  }
```

La conversione binaria viene fatta nel ciclo a **righe 29÷27**, con divisioni per 2 e memorizzazione dei resti (al contrario) nell'array **bit[ ]**.

Il conteggio delle transizioni viene eseguito sull'array **bit[ ]** nella selezione presente alle **righe 41÷44**.

Il layout di questo programma, nel caso del codice Ascii della lettera **a** minuscola:

OUTPUT

```
Inserire un carattere: a
Stampa binaria di 'a' al contrario:   10000110
Stampa binaria di 'a' al raddrizzata: 01100001
Le transizioni sono: 3
```

Dato che 'a' = 61h = 01100001b, le transizioni sono 3, ovvero, messe in evidenza:

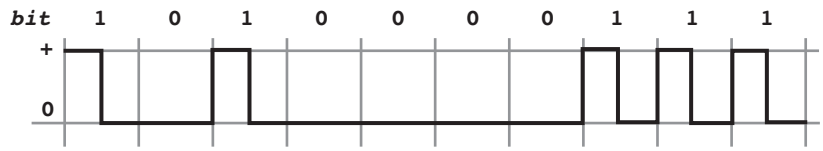
```
 1   2       3
0 1 1 0 0 0 0 1
```

Molti di questi sistemi di codifica fisica dei bit non garantiscono le transizioni, per esempio quando i dati sono composti da lunghe sequenze costanti di bit a zero o a uno, pertanto si devono prevedere codifiche supplementari **a garanzia di transizione**, che prima di immettere i bit originali sul mezzo con la tecnica adottata, trasformano i dati in altre sequenze equivalenti in cui viene garantita l'alternanza minima delle presenze di bit a 0 e di bit a 1.

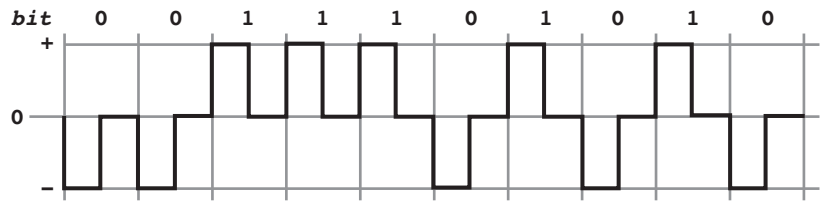
Gli schemi di codifica più diffusi sono i seguenti:

RZ

Lo schema base per la codifica dei bit è denominato **RZ** (*Return Zero*), unipolare (un solo segno nel segnale), con transizione sul bit e a metà del bit 1 (ritorno a zero). Non garantisce transizioni sui bit a zero, quindi necessita di una codifica supplementare a causa delle sequenze di zeri.



La versione **RZI** invece è bipolare (due segni opposti nel segnale), con medesime caratteristiche, ma le transizioni sono garantite per ogni bit, sia i bit a 1, sia i bit a zero.



Queste codifiche non sono usate, di norma, per la comunicazione.

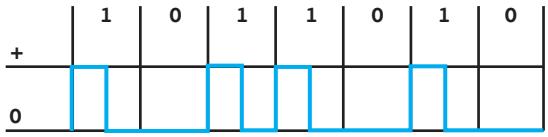
ESEMPIO

Codifica RZ

Una interfaccia di rete deve spedire il codice Ascii della lettera Z con codifica RZ. Rappresentare l'onda quadra simbolica.

Il codice Ascii della lettera Z (maiuscola) vale 5ah, cioè (1011010)b.

Siccome le transizioni in RZ sono **0+0** e avvengono sulla metà del bit e per i soli bit a 1, la rappresentazione grafica simbolica è:



NRZ

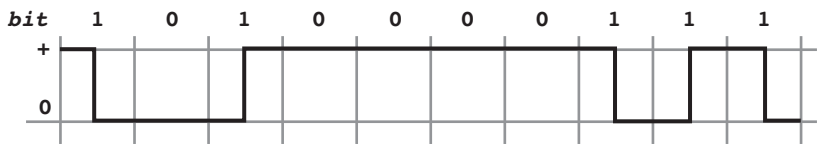
NRZ o *Non Return to Zero*, è una codifica unipolare. I valori sono codificati in modo diretto sul bit: i bit a 1 con tensione positiva, i bit a zero con tensione nulla. Il rapporto tra la frequenza della fondamentale e la frequenza del bit vale f=R/2, ovvero per una velocità trasmissiva sul cavo di 10 Mbit/sec è sufficiente un clock di 5 MHz. Anche in questo caso sequenze costanti di uni o di zeri non causano transizioni e fanno decadere la sincronizzazione. Si ripara con codifiche di bit a garanzia di transizione.

Usata per EIA Rs232C (ma invertita) e anche utilizzata su 1000BASE-SX (fibra ottica fino a 500m) e 1000BASE-LX (fibra ottica fino a 10 km) con codifica supplementare 8B10B.

NRZI

NRZ Inverted on One, codifica unipolare in cui i bit sono codificati con transizione a metà dei bit a 1 e nessuna transizione per i bit a zero.

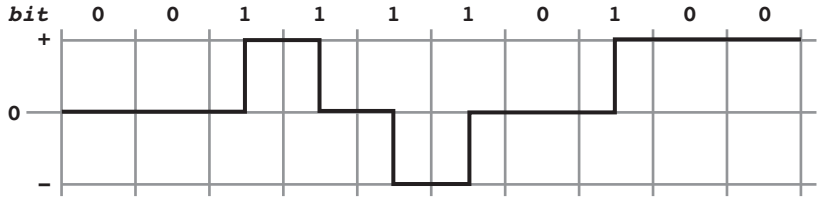
Come prima, il rapporto frequenza fondamentale e frequenza del bit vale f=R/2, ovvero per una velocità trasmissiva sul cavo di 125 Mbit/s è sufficiente un clock di 62,5 MHz. In questo caso perdono la sincronizzazione le sequenze di zeri, pertanto bisogna integrare con codifiche a garanzia di transizioni.



La NRZI è utilizzata su 100BASE-FX (fibra ottica) e sui canali USB.

MLT3

MLT3 o *Multilevel Threshold-3*, codifica bipolare in cui i bit sono segnalati con transizione a metà sui bit 1 e nessuna transizione per i bit 0. Le transizioni per i bit 1 si susseguono nell'ordine 0+, +0, 0–, –0.



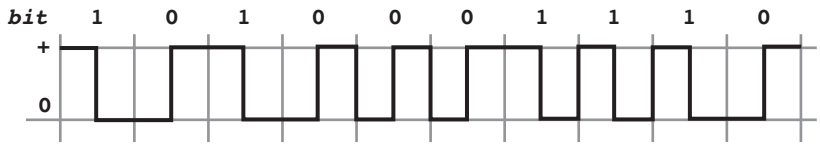
Il rapporto frequenza fondamentale e frequenza del bit vale f=R/4 (per esempio, una sequenza di 4 uni, sono codificati all'interno di un solo ciclo di fondamentale). Si deduce che per una velocità trasmissiva sul cavo di 100 Mbit/s, è necessaria una frequenza della fondamentale di 25 MHz.

In presenza di sequenze di zeri, il segnale non contiene transizioni e quindi la sincronizzazione viene persa, pertanto bisogna integrare con codifiche a garanzia di transizioni. La codifica MLT3 è utilizzata per Fast Ethernet 100BASE-TX in rame.

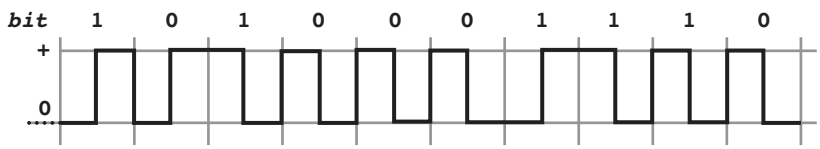
Manchester

Codifica unipolare bifase, che contiene sempre una transizione sul bit, sia per i bit 1 che per i bit 0.

Per la **codifica di Manchester**, la transizione è costante: per i bit 1 vale +0, per i bit 0 vale 0+.



Per la **codifica di Manchester differenziale** invece dipende dal fronte precedente: per i bit 1 +0 se il livello precedente è +; 0+ se il livello precedente è 0. Per i bit 0, +0 se il livello precedente è 0; 0+ se il livello precedente è + (si suppone che prima della codifica dei bit il segnale sia in uno stato conosciuto, o 0 o +)[NB].



In entrambi i casi la frequenza del segnale fondamentale è uguale alla frequenza del bit (ogni ciclo «porta» un bit), cosicché per aumentare la larghezza di banda è necessario accrescere dello stesso ordine la frequenza del segnale, cosa che ha impedito a questo tipo di codifica di evolversi.

Tuttavia la codifica di Manchester non necessita di ulteriori metodi di conversione, avendo garanzia di transizione e quindi sincronizzazione su ogni bit.

La codifica di Manchester è usata su Ethernet 802.3 a 10 Mbit/s, mentre la codifica di Manchester differenziale è utilizzata su Token Ring 802.5 a 4 e 16 Mbit/s. Per ognuna di queste tecniche le violazioni di codice, necessarie per la sincronizzazione dei frames, si ottiene con segnale costante su un livello determinato.

NB. Alcuni testi riportano le transizioni di Manchester invertite rispetto a quelle proposte. Per questo si rimanda al testo *Reti locali: dal cablaggio all'internetworking*, Silvano Gai, P. L. Montessoro e P. Nicoletti, Scuola Superiore G. Reiss Romoli, 1997.

ESEMPIO

Codifica di Manchester

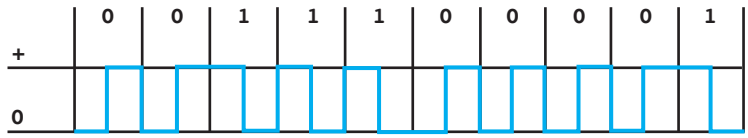
Un'interfaccia di rete deve spedire il codice Ascii esteso del simbolo β su 10 bit con codifica di Manchester.

Rappresentare l'onda quadra simbolica.

Il codice Ascii esteso del simbolo β vale (225)d, ovvero (11100001)b.

Riempito a 10, risulta (0011100001)b.

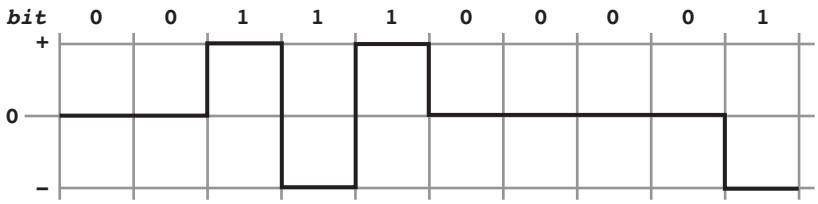
La transizione avviene su tutti i bit e a metà del bit, per i bit 0 una **0+**, per i bit 1 una **+0**; la rappresentazione logica è:



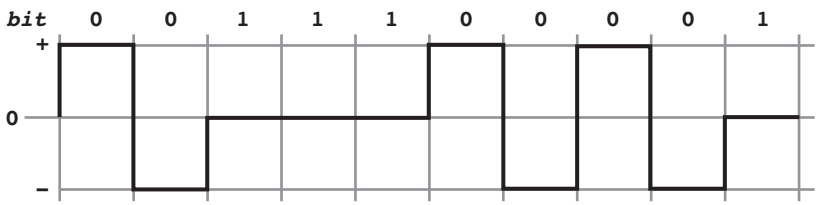
AMI

AMI o *Alternate Mark Inversion* è una codifica bipolare (due segni nel segnale), con sola transizione sul bit a 1 in modo alternato 0+ e 0–.

Lunghe sequenze di zeri fanno perdere la sincronizzazione e vanno contrastate con codifiche a garanzia di transizione.



Analoghe considerazioni per **AMI invertita** o Pseudoternaria, in cui sono i bit a zero a prevedere le transizioni, e le sequenze di «uni» a perdere sincronizzazione.



Con AMI è possibile anche la rilevazione dell'errore, data l'alternanza garantita degli impulsi delle transizioni. È così possibile individuare immediatamente le **violazioni di codice**, ovvero transizioni consecutive con lo stesso segno.

AMI si utilizza per gli accessi alla rete PSTN tramite modem fonici fino a 57 kbit/s, e per tutti i modelli di comunicazione Plesiocroni con trame E1/T1, assieme alle codifiche supplementari B8ZS e HDB3, o su trame superiori En/Tn con codifiche B6ZS e B3ZS.

AMI con codifica HDB3 viene usata per numerose implementazioni HDSL lato linea, dal DCE alla centrale.

Codifiche per AMI

Per le codifiche AMI sono previste ulteriori codifiche supplementari di sincronizzazione, dato che sui mezzi in cui AMI viene implementata le codifiche in trasmissione e in ricezione devono avere le stesse lunghezze. Pertan-

Codifiche multilivello

Per determinate linee vengono adottate anche **codifiche multilivello**, cioè codifiche che associano a gruppi di bit insiemi di segnali in modo da ottenere sequenze più brevi sfruttando basi di numerazione superiore a due.

2B1Q (*2 Binary 1 Quaternary*), codifica che associa alle 4 coppie di 2 bit un simbolo quaternario (su base 4), in base allo schema, 10 +3v, 11 +1v, 01 –1v, 00 –3v. Usata sui canali di rete pubblica ISDN e HDSL.

8B1Q4 (*8 Binary plus1 binary 4 Quinary*), codifica che associa 9 simboli binari a 4 simboli quinari (base 5), tipicamente i valori di tensione –2v, –1v, 0, +1v, +2v. Usata in 1000BASE-T.

MBNT, codici che trasformano m-uple di cifre binarie (B) in n-uple di cifre ternarie da trasmettere utilizzando tre livelli sulla linea (T). Esempi sono il codice 4B3T, che trasforma quaterne di cifre binarie in terne di cifre ternarie, e il codice 6B4T, che trasforma sestine di cifre binarie in quaterne di cifre ternarie. 4B3T e 3B2T sono usate per ISDN. 8B6T è usata per 100BASE-T4.

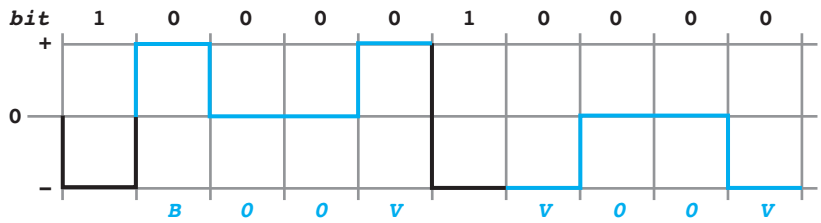
PAM5 (*Pulse Amplitude Modulation 5*), in cui sequenze di bit vengono associate a coppie di simboli quinari (a base 5) sui 25 a disposizione usando i valori di tensione –2v, –1v, 0, +1v, +2v (costellazione). Usata in 1000BASE-T.

to le sequenze costanti di bit sono sostituite con sequenze canoniche che il ricevitore dovrà poi escludere automaticamente.

HDB3

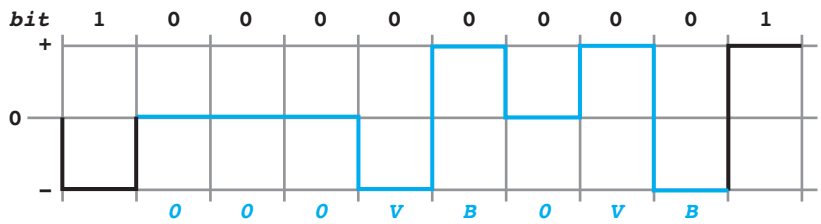
In questo caso vengono sostituite le sequenze di 4 zeri con la sequenza **x00V**, dove **x** può essere una transizione regolare (**B**) o una transizione in violazione (**V**). Una transizione regolare è quella che rispetta l'alternanza dei segni sulle transizioni; la violazione è una transizione di segno identico alla precedente. Si usa **B** se il numero di bit a 1 dall'ultima sostituzione è pari (0 considerato pari).

Nell'esempio, il numero di bit a 1 precedenti alla prima sostituzione è per ipotesi pari.



B8ZS

In questo caso vengono sostituite le sequenze di 8 zeri con **000VB0VB**. Sono possibili due sostituzioni: se l'ultimo impulso di tensione era positivo, con **000+−0−+**; con **000−+0+−** altrimenti. In entrambe le sequenze usate per la sostituzione si generano due violazioni della codifica AMI.



2.1 Codici a garanzia di transizione

Per molte delle codifiche suddette, esistono sequenze di dati che non generano transizioni di bit e quindi non garantiscono più la sincronizzazione tramite i fronti di salita/discesa.

In questi casi, prima di inviare i segnali originali sul mezzo, è necessario codificare ulteriormente i bit, allungandone la sequenza, ma garantendo l'alternanza minima dei bit 1 e 0 per avere un numero sufficiente di transizioni, codifica per codifica.

Le tabelle per effettuare la sostituzione (in trasmissione), e la ricostruzione (in ricezione), sono denominate **tabelle xByB** (con x<y).

Tra le più note e già citate codifiche xByB, distinguiamo 4B5B, 5B6B, 8B10B. Per tecnologie di rete molto evolute su fibra ottica della famiglia 10GBASE-xx si usano anche codifiche del tipo 64B66B.

dato	3B/4B		5B/6B		8B/10B	
0	000	**0100**	00000	**011000**	00000000	**0110001011**
1	001	**1001**	00001	**100010**	00000001	**1000101011**
2	010	**0101**	00010	**010010**	00000010	**0100101011**
3	011	**0010**	00011	**001010**	00000011	**0010101011**
4	100	**1010**	00100	**000111**	00000100	**0001110100**
5	101	**0110**	00101	**000110**	00000101	**0001101011**
6	110	**1101**	00110	**101000**	00000110	**1010001011**
7	111	**1100**	00111	**100100**	00000111	**1001001110**
8			01000	**100010**	00001000	**1000101110**
...			...	...	00001001	**1000011110**
31			...	...	00001010	**1000001110**
...					...	...
255					...	...

Come si può notare dalla tabella 3B/4B, alcune combinazioni originali di 3 bit possono dar vita a sequenze con 5 o più bit uguali (per esempio i due dati 0 e 0 originano 000 e 000 mentre 3 e 7 originano 011 e 111).

Usando invece i quartetti proposti, non è possibile nessuna sequenza con 5 o più bit uguali (max. 4 bit uguali).

ESEMPIO

Garanzia di transizioni con 3B/4B

Applicare la codifica 3B/4B agli ottetti della sequenza Ascii ABC e ottenere una nuova sequenza con un numero massimo di bit costanti uguale a 4.

Prima di tutto si calcola la sequenza binaria a partire dai codici Ascii dei singoli caratteri 'A' = 41h =(01000001)b , 'B' = 42h =(01000010)b , 'C' = 43h =(01000011)b.

Le rappresentazioni binarie vanno riempite al byte, perché il testo parla di ottetti.

Si ottiene la sequenza binaria: **010000010100001000011**, nella quale si nota una sequenza costante di 5 zeri.

Ora si effettua la sostituzione in base alla tabella 3B/4B:

```
3B  010  000  010  100  001  001  000  011
4B  0101 0100 0101 1010 1001 1001 0100 0010
```

La sequenza risultante: **0101010001011010100110 0101000010**, ha al massimo sequenze di 4 bit costanti.

Si nota che una rappresentazione su 24 bit originale viene trasformata in una rappresentazione su 32 bit codificata, con un overhead di 8 bit (25% di 32).

Naturalmente l'uso delle tabelle xByB comporta uno «spreco» di banda (detto anche **overhead**), aumentando il numero di bit complessivi da spedire.

Nel caso della codifica 4B5B i quintetti previsti, per ogni quartetto possibile in ingresso, induce un overhead del 25% (un bit aggiuntivo ogni quattro), pertanto per ottenere una trasmissione di 100 Mbit/s a livello MAC, il bit rate effettivo sul mezzo dovrà essere di 125 Mbit/s.

Spesso queste codifiche sono sottoposte a ulteriore trattamento prima della trasmissione fisica, operazioni dette di **scrambling**, per evitare che le codifiche sul mezzo riportino frequenze troppo regolari e che quindi il sistema hardware diventi soggetto a emissioni di interferenze elettromagnetiche (EMI).

PROGRAMMA

Garanzia di transizioni con 2B/3B

Scrivere un programma in linguaggio C che codifichi un ottetto con una tabella 2B/3B opportuna.

Una tabella opportuna 2B/3B potrebbe essere:

```
2B/3B
00 010
01 101
10 110
11 011
```

Si nota che con questa tabella la massima sequenza costante di bit in codifica 3B vale 4 (per esempio, 011110, che decodificata in 2B vale 1110). Un codice C per la sequenza binaria **10000011** espressa in codici Ascii potrebbe essere:

```
00 #include <stdio.h>
01 #include <string.h>
02
03 char sz[]="10000011"; // in 2B, da codificare
04
05 char* asz2B[]={"00","01","10","11"};
06 char* asz3B[]={"010","101","110","011"};
07
08 char sz2B3B[128];    // risultato codificato in 3B
09
10 int main(void)
11 {
12   int i,j,k,l;
13
14   sz2B3B[0]=0;
15   k = 1;
16   l = strlen(sz);
17
18   for (i=0; i<l; i=i+2)
19   {
20     for (j=0; j<4; j++)
21     {
22       if (sz[i]==asz2B[j][0])
23       {
24         if (sz[i+1]==asz2B[j][1])
25         {
26           strcat(sz2B3B,asz3B[j]);
27           k = k + 1;
28         }
29         }
30       }
31     }
32
33     sz2B3B[k*3-1] = 0;
34
35     printf("2B=%s\n",sz);
36     printf("3B=%s\n",sz2B3B);
37
38     return 0;
39 }
```

La variabile i salta di due in due (0,2,4,6, vedi **riga 18**).

Ad ogni passo si cerca la coppia binaria **sz[i]** e **sz[i+1]** nella tabella ***asz2B[]**, che ha quattro elementi, vedi **righe 20÷24**. Quando si trova la coppia nella tabella, si concatena l'equivalente elemento nella tabella ***asz3B[]** alla stringa risultato come in **riga 26**. La variabile **k** conta il n. di caratteri della stringa risultato, per poter collocare il terminatore di stringa correttamente (**riga 33**).

L'output di questo programma è:

OUTPUT

```
2B=10000011
3B=110010010011
```

In effetti:

```
2B    10  00  00  11
3B   110 010 010 011
```

ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1. Elencare tutti i modi di interconnessione a una rete WAN, indicandone il bitrate tipico.
2. Indicare i modi di interconnessione alla rete pubblica wireless, indicandone le caratteristiche.
3. Elencare le modalità d'accesso DSL, indicandone le caratteristiche.
4. Spiegare per quale motivo il bitrate di una linea ADSL è variabile.
5. Indicare la relazione tra sincronizzazione di un segnale digitale e transizione.
6. Commentare la nozione di clock & data encoding.
7. Mostrare di saper contare le transizioni su una sequenza binaria.
8. Elencare gli schemi di codifica di livello 1 (Fisico) per reti LAN.
9. Riportare su un foglio le rappresentazioni logiche del byte aah per almeno quattro schemi di codifica di livello 1 (Fisico).
10. Spiegare la differenza fondamentale tra lo schema della codifica di Manchester e tutte le altre.
11. Indicare quale schema di codifica di livello 1 (Fisico) è utilizzato in Fast Ethernet.
12. Illustrare la nozione di violazione di codice.
13. Commentare gli schemi che consentono la garanzia di transizione.

Requisiti avanzati

1. Spiegare per quale motivo il protocollo EIA Rs232C è ancora diffuso.
2. Illustrare la nozione di overbooking per le connessioni ADSL.
3. Spiegare a cosa serve il bit di stop nel protocollo fisico EIA Rs232C.
4. Elencare le principali tecnologie di accesso a Internet tramite Sim Card.
5. Indicare i due protocolli di livello 2 usati per le connessioni residenziali ADSL.
6. Spiegare per quale motivo la banda su telefonia cellulare dati a volte subisce un crollo di prestazioni.
7. Confrontare i modi di accesso ADSL e HDSL, fornendone anche i parametri di bitrate.
8. Spiegare quali codifiche di livello 1 (Fisico) necessitano di ulteriore codifica a garanzia di transizione e perché.
9. Spiegare cosa contraddistingue le codifiche AMI da tutte le altre.
10. Spiegare cosa si intende per codifiche multilivello.
11. Illustrare le codifiche ulteriori per lo schema AMI, spiegandone la ragione.
12. Spiegare in cosa consiste l'overhead delle codifiche a garanzia di transizione.
13. Mostrare che proprietà devono possedere le codeword delle tabelle xB/yB.

ESERCIZI PER LA VERIFICA SCRITTA

I prerequisiti richiesti per affrontare questi esercizi sono la conoscenza dei sistemi di numerazione in base 2 e in base 16 (binario e esadecimale), la conversione decimale-binario e viceversa, la conversione decimale-esadecimale e viceversa, anche con applicativi (per esempio Calc).

È necessaria la consultazione di una tabella di codici Ascii.

1 Un'interfaccia seriale EIA Rs232C deve spedire il byte del codice Ascii della lettera Z. Rappresentare l'onda quadra simbolica del pacchetto completo di livello 1 se il protocollo vale 9600, n, 8, 1.

2 Un'interfaccia seriale EIA Rs232C deve spedire il byte del codice Ascii della lettera a (minuscola). Rappresentare l'onda quadra simbolica del pacchetto completo di livello 1 se il protocollo vale 9600, e, 8, 1.

3 Un'interfaccia seriale EIA Rs232C deve spedire il byte del codice Ascii dello zero. Rappresentare l'onda quadra simbolica del pacchetto completo di livello 1 se il protocollo vale 9600, n, 7, 2.

4 Un'interfaccia seriale EIA Rs232C deve spedire i byte Ascii del numero 65. Rappresentare l'onda quadra simbolica del pacchetto completo di livello 1 se il protocollo vale 9600, n, 8, 1.

5 Un'interfaccia seriale EIA Rs232C deve spedire i due byte big endian del numero 42929. Rappresentare l'onda quadra simbolica del pacchetto completo di livello 1 se il protocollo vale 9600, o, 8, 1.

6 Effettuare il procedimento necessario per calcolare il numero di transizioni nei codici Ascii dei caratteri della stringa Ciao.

7 Effettuare il procedimento necessario per calcolare il numero di transizioni nei codici Ascii del numero 1243.

8 Effettuare il procedimento necessario per calcolare il numero di transizioni nei due byte del numero 45057.

9 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica RZ. Rappresentare l'onda quadra simbolica.

10 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica RZI. Rappresentare l'onda quadra simbolica.

11 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica NRZ. Rappresentare l'onda quadra simbolica.

12 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica NRZI. Rappresentare l'onda quadra simbolica.

13 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica MLT3. Rappresentare l'onda quadra simbolica.

14 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica Manchester. Rappresentare l'onda quadra simbolica.

15 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica Manchester differenziale. Rappresentare l'onda quadra simbolica.

16 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica AMI. Rappresentare l'onda quadra simbolica.

17 Un'interfaccia di rete deve spedire il codice Ascii della lettera A con codifica AMI invertita. Rappresentare l'onda quadra simbolica.

18 Rappresentare l'onda quadra simbolica in codifica MLT3 dei codici Ascii delle proprie iniziali maiuscole (nome e cognome).

19 Rappresentare l'onda quadra simbolica in codifica AMI dei codici Ascii delle proprie iniziali maiuscole (nome e cognome).

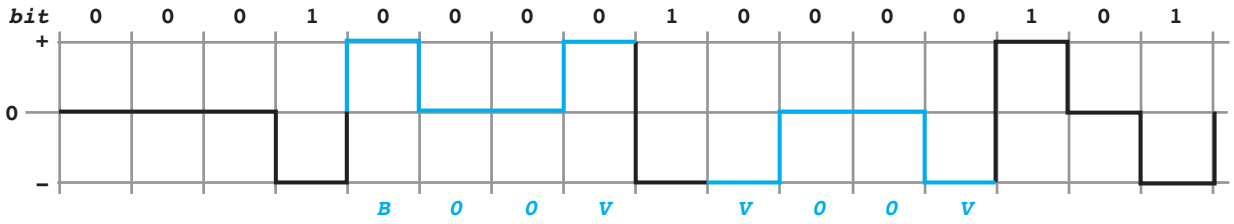
20 Un'interfaccia di rete deve spedire il numero 4229 con codifica AMI/HDB3. Rappresentare l'onda quadra simbolica.

Prima di tutto si converte in binario il numero dato, riempiendolo su 2 ottetti:

```
4229 = (1085)h = (1000010000101)b = (0001000010000101)b
```

Dovendo usare HDB3, si notano 2 sostituzioni, cioè due gruppi da 4 bit a zero. Ipotizzando che il n. di bit a 1 precedenti a questa codifica sia pari, la prima sostituzione sarà B00V.

Per la seconda sostituzione, i bit a 1 precedenti sono dispari, infatti un bit a 1 si frappone tra i due gruppi, pertanto la sostituzione sarà V00V.



La transizione del primo bit a 1 è 0–, ma poteva essere anche 0+, dato che non si sa come fosse la transizione precedente.

Nella prima sostituzione, la transizione su B è 0+, cioè regolare, dato che la precedente era 0–. Inoltre la transizione su V è 0+, identica alla precedente, come vuole una violazione.

Analoghe considerazioni per la seconda sostituzione.

21 Un'interfaccia di rete deve spedire il numero 32849 con codifica AMI/B8ZS. Rappresentare l'onda quadra simbolica.

22 Un'interfaccia di rete deve spedire il numero 2081 con codifica AMI/HDB3. Rappresentare l'onda quadra simbolica.

23 Un'interfaccia di rete deve spedire il numero 2081 con codifica AMI/B8ZS. Rappresentare l'onda quadra simbolica.

24 Applicare la codifica 3B/4B agli ottetti della sequenza Ascii abc e ottenere una nuova sequenza con un numero massimo di bit costanti uguale a 4.

25 Applicare la codifica 3B/4B ai byte del numero 32840 e ottenere una nuova sequenza con un numero massimo di bit costanti uguale a 4.

26 Decodificare la sequenza di bit secondo la tabella 3B/4B, sapendo che si tratta di una sequenza Ascii (su otto bit).

Sequenza:

```
10101011001010101001010110101001101O
```

27 Decodificare la sequenza di bit secondo la tabella 3B/4B, sapendo che si tratta di una sequenza Ascii (su otto bit).

Sequenza:

```
10101011010110101001011010100111O1
```

28 Scrivere una nuova tabella 3B/4B in modo che il n. massimo di bit costanti sia uguale a 4. Quindi codificare la sequenza Ascii Ciao.

ESERCIZI PER LA VERIFICA DI LABORATORIO

I prerequisiti per affrontare questi esercizi sono la consultazione di una tabella di codici Ascii e la disponibilità di un ambiente di sviluppo per programmi in linguaggio C. Per compilare gli esercizi proposti è stato usato l'ambiente gratuito multipiattaforma Windows/Linux ***Code::Blocks 10.5*** con **gcc 4.4.1**.

I testi degli esercizi presentano uno o più **layout** di input/output richiesti; in questo modo sono indicate, a volte, alcune specifiche del programma come l'output, il controllo dell'input o i valori ammissibili.

1 **Scrivere un programma in linguaggio C che, estratto un numero casuale rappresentabile con un byte, ne stampi la rappresentazione binaria.**

Layout:

```
                                        OUTPUT
Numero casuale 203
In binario: 11001011
```

2 **Scrivere un programma in linguaggio C che, preso in input un numero intero senza segno, ne stampi la rappresentazione binaria.**

Layout:

```
                                        OUTPUT
Inserire numero: 3492
In binario: 110110100100
```

3 **Scrivere un programma in linguaggio C che, presa in input una stringa Ascii, stampi la rappresentazione binaria di ogni suo carattere.**

Layout:

```
                                        OUTPUT
Inserire caratteri (max 32 char): HAL
In binario: 01001000 . 01000001 . 01001100
```

4 **Scrivere un programma in linguaggio C che, estratto un numero casuale rappresentabile con un byte, stampi la sequenza di caratteri della sua rappresentazione esadecimale.**

Layout:

```
                                        OUTPUT
Numero casuale 203
In esadecimale: C B
```

Nota: Rispettare il layout, in particolare lo spazio tra i caratteri.

5 **Scrivere un programma in linguaggio C che, preso in input un numero intero senza segno, ne stampi la rappresentazione esadecimale.**

Layout:

```
                                        OUTPUT
Inserire numero: 3492
In esadecimale: D A 4
```

Nota: Rispettare il layout, in particolare lo spazio tra i caratteri.

6 **Scrivere un programma in linguaggio C che, presa in input una stringa Ascii, stampi la rappresentazione esadecimale di ogni suo carattere.**

Layout:

```
                                        OUTPUT
Inserire caratteri (max 32 char): HAL
In esadecimale: 4 8 . 4 1 . 4 C
```

Nota: Rispettare il layout, in particolare lo spazio tra i caratteri.

7 **Scrivere un programma in linguaggio C che, presa in input una stringa Ascii binaria (soli caratteri '0' e '1') di otto bit, la decodifichi sapendo che l'ottetto di bit è un codice Ascii.**

Layout:

```
                                        OUTPUT
Stringa binaria (8char): 0100100
Stringa binaria (8char): 01001000
Codice Ascii: 'H'
```

8 **Scrivere un programma in linguaggio C che, presa in input una stringa Ascii binaria (soli caratteri '0' e '1'), stampi il numero delle transizioni dei bit presenti nella sequenza.**

Layout:

```
                                        OUTPUT
Inserire stringa binaria: HAL
Inserire stringa binaria: 101010101111
8 transizioni
```

9 Scrivere un programma in linguaggio C che, preso in input un numero intero senza segno, stampi il numero delle transizioni dei bit presenti nella sua rappresentazione binaria.

Layout:

OUTPUT

```
Inserire numero: HAL
Inserire numero: 6454
7 transizioni
```

10 Scrivere un programma in linguaggio C che, presa in input una stringa Ascii, stampi il numero delle transizioni dei bit di ogni suo singolo carattere.

Layout:

OUTPUT

```
Inserire caratteri (max 32 char): HAL
'H': 4 transizioni
'A': 3 transizioni
'L': 4 transizioni
```

11 Scrivere un programma in linguaggio C che, presa in input una stringa Ascii binaria (soli caratteri '0' e '1'), individui la sequenza costante di bit più lunga.

Layout:

OUTPUT

```
Inserire stringa binaria: 101010101111
N. bit sequenza costante maggiore: 4
```

12 Scrivere un programma in linguaggio C che, presa in input una stringa Ascii binaria (soli caratteri '0' e '1'), e un numero n (>2), conti quante sequenze costanti di bit lunghe pù di n sono presenti nella sequenza.

Layout:

OUTPUT

```
Inserire stringa binaria: 100000101111
Inserire numero (>2): 2
Inserire numero (>2): 3

Sequenze costanti (>3): 2
```

13 Scrivere un programma in linguaggio C che, presa in input una stringa Ascii binaria (soli caratteri '0' e '1'), la riempia a sinistra con tanti bit a zero per fare in modo che la sua lunghezza sia un multiplo di 3.

Layout:

OUTPUT 1

```
Inserire stringa binaria: 100000101111
Sequenza corretta.
```

OUTPUT 2

```
Inserire stringa binaria: 10000101111
Sequenza riempita: 010000101111
```

14 Scrivere un programma in linguaggio C che, presa in input una stringa Ascii binaria (soli caratteri '0' e '1') e un numero n, la riempia a sinistra con tanti bit a zero per fare in modo che la sua lunghezza sia un multiplo di n.

Layout:

OUTPUT

```
Inserire stringa binaria: 10000010111
Inserire numero (0<n<11): 4

Sequenza a gruppi di 4: 010000010111
```

15 Scrivere un programma in linguaggio C che, presa in input una stringa Ascii binaria (soli caratteri '0' e '1') decodifichi la sequenza sapendo che si tratta di ottetti Ascii.

Layout:

OUTPUT

```
Inserire ottetti: 0100100001000
Inserire ottetti: 010010000100000101001100

Ottetto n.1, codice Ascii: 'H'
Ottetto n.2, codice Ascii: 'A'
Ottetto n.3, codice Ascii: 'L'
```

Nota: Il primo input è errato perché il n. di caratteri non è multiplo di 8. Il secondo input è corretto.

16 Scrivere un programma in linguaggio C che prenda in input una stringa Ascii binaria (soli caratteri '0' e '1') rappresentante una codifica 4B.

Layout:

OUTPUT

```
Inserire stringa binaria 4B: 10000010111
Inserire stringa binaria 4B: 01001000
Inserire stringa binaria 4B: 01001001
```

Nota: Il primo input è errato perché il n. di caratteri non è multiplo di 4; nel secondo caso un quartetto non è previsto nella tabella. Il terzo input è corretto.

17 Scrivere un programma in linguaggio C che prenda in input una stringa Ascii binaria (soli caratteri '0' e '1') rappresentante una codifica 4B e ne stampi la decodifica in 3B.

Layout:

```
                                                    OUTPUT
Inserire stringa binaria 4B: 010010010101
In 3B: 000.001.010
```

18 Scrivere un programma in linguaggio C che prenda in input una stringa Ascii binaria (soli caratteri '0' e '1') rappresentante una codifica 3B e ne stampi la decodifica in 4B.

Layout:

```
                                                    OUTPUT
Inserire stringa binaria 3B: 0000010
Inserire stringa binaria 3B: 00000100
Inserire stringa binaria 3B: 000001001

In 4B: 0100.1001.1001
```

Nota: Il primo input è errato perché il n. di caratteri non è multiplo di 3, così come il secondo input. Il terzo input è corretto.

19 Scrivere un programma in linguaggio C che prenda in input una stringa Ascii e ne stampi la codifica in 4B.

Layout:

```
                                                    OUTPUT
Inserire stringa: HAL
I caratteri in binario:
'H' = 01001000
'A' = 01000001
'L' = 01001100

3B = 010 010 000 100 000 101 001 100
4B = 01010101010010100100011010011010
```

20 Usando la tabella 2B/3B seguente:

2B	3B
00	010
01	101
10	110
11	011

scrivere un programma in linguaggio C che trasformi la seguente sequenza binaria Ascii 10000010 (solo caratteri '0' e '1') da 2B a 3B.

Layout:

```
                                                    OUTPUT
Stringa binaria 2B: 10000010
Stringa binaria 3B: 110010010110
```

21 Usando la tabella 2B/3B seguente:

2B	3B
00	101
01	010
10	001
11	100

scrivere un programma in linguaggio C che presa in ingresso una sequenza binaria Ascii (solo caratteri '0' e '1') normalizzata a 2, la codifichi in 3B.

Layout:

```
                                                    OUTPUT
Stringa binaria 2B: 1000001
Normalizzata a 2: 01000001
Stringa binaria 3B: 010101101010
```

Nota: Per normalizzare a 2 una stringa binaria bisogna riempire a destra con uno '0' se la lunghezza della stringa è un numero dispari.

22 Usando la tabella 2B/3B seguente:

2B	3B
00	101
01	010
10	001
11	100

scrivere un programma in linguaggio C che presa in ingresso una stringa di codici Ascii, la codifichi in 3B.

Layout:

```
                                                    OUTPUT
Inserire stringa: HAL
I caratteri in binario:
'H' = 01001000
'A' = 01000001
'L' = 01001100

2B = 01 00 10 00 01 00 00 01 01 00 11 00
3B = 010101001101010101101010010101100101
```

23 Scrivere un programma in linguaggio C con interfaccia a carattere che stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii impostando le transizioni di un pacchetto dato.
Si provi con il pacchetto (1010000010)b per Rs232 9600, n, 8, 1.
Come analizzato in un esercizio precedente, il pacchetto dà origine alla seguente forma d'onda:

```
 1 0 1 0 0 0 0 0 1 0
```

Si nota che vanno codificate quattro transizioni: `+0`, `0`, `0+`, `+`, che risultano essere i quattro simboli grafici da stampare a video a seconda della situazione.
Nel pacchetto dato, la sequenza delle transizioni è la seguente:

```
(idle+), +0, 0+, +0, 0+, +, +, +, +0, 0+, +, (+idle)
```

Nel programma si codificano queste quattro transizioni e quindi si prepara una funzione `BitOnda()` che, per ogni condizione, memorizza su due array la coppia di caratteri grafici che rappresentano quella transizione.
Per esempio, la transizione `+0` sarà costituita dalla coppia ┐ e └.
In definitiva:

```
 +0 0 + 0+
```

```
#include <stdio.h>

#define ZERO (0) // Codifiche delle transizioni
#define PIU (1)
#define ZP (2)
#define PZ (3)

char oqa[32]; // array che contiene la parte grafica superiore della condizione di stato
char oqb[32]; // array che contiene la parte grafica inferiore della condizione di stato

void BitOnda(int i, int come); // compone gli array di stato

int main()
{
    int i,j;

    i = 0;
    BitOnda(i++,PIU); // 0 idle+
    BitOnda(i++,PZ);  // 1 start
    BitOnda(i++,ZP);  // 0
    BitOnda(i++,PZ);  // 1
    BitOnda(i++,ZP);  // 0
    BitOnda(i++,PIU); // 0
    BitOnda(i++,PIU); // 0
    BitOnda(i++,PIU); // 0
    BitOnda(i++,PIU); // 0
    BitOnda(i++,PZ);  // 1
    BitOnda(i++,ZP);  // 0 stop
    BitOnda(i,PIU);   // idle+
```

```
    printf("Segnale di (1010000010)b per Rs232 9600,n,8,1\n");

    for (j=0;j<i;j++)
    {
      printf("%c",oqa[j]);
    }
    printf("\n");

    for (j=0;j<i;j++)
    {
      printf("%c",oqb[j]);
    }

    return 0;
}

void BitOnda(int i, int come)
{
  switch (come)
  {
    case ZERO:
      oqa[i] = ' ';
      oqb[i] = 196;
    break;

    case PIU:
      oqa[i] = 196;
      oqb[i] = ' ';
    break;

    case ZP:
      oqa[i] = 218;
      oqb[i] = 217;
    break;

    case PZ:
      oqa[i] = 191;
      oqb[i] = 192;
    break;
  }

}
```

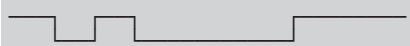
OUTPUT

```
Segnale di (1010000010)b per Rs232 9600,n,8,1
```

24 Scrivere un programma in linguaggio C con interfaccia a carattere che stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii impostando le transizioni di un pacchetto dato con la codifica NRZ. Si provi con il pacchetto (1010000111)b.

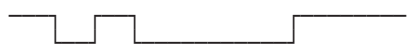
Layout:

```
                                        OUTPUT
Segnale di (1010000111)b per NRZ:
```



Sfruttando le impostazioni del precedente programma, si vuole ottenere un risultato del genere:

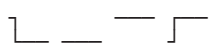
```
  1   0 1 0   0   0   0   1   1   1
```



```
1..0..1..0..0..0..0..1..1..1..
```

Come si può notare, abbiamo di nuovo quattro condizioni di stato: +, +0, 0+, 0, anche se questa volta corrispondono ai simboli

```
  +0   0    +    0+
```



Si modificherà quindi la funzione `BitOnda()` prevedendo tre caratteri per condizione.
Per implementare l'algoritmo NRZ, dopo aver predisposto un array contenente i bit da codificare, si progetterà una semplice condizione sul bit in base al bit precedente, selezionando la condizione opportuna.

25 Scrivere un programma in linguaggio C con interfaccia a carattere che, una volta acquisito un byte da tastiera, ne stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii secondo la codifica di Manchester.

26 Scrivere un programma in linguaggio C con interfaccia a carattere che, una volta acquisito un byte da tastiera, ne stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii secondo la codifica di Manchester differenziale.

27 Scrivere un programma in linguaggio C con interfaccia a carattere che, una volta acquisito un byte da tastiera, ne stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii secondo la codifica MLT3.

28 Scrivere un programma in linguaggio C con interfaccia a carattere che, una volta acquisito un byte da tastiera, ne stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii secondo la codifica AMI.

29 Scrivere un programma in linguaggio C con interfaccia a carattere che, una volta acquisito un byte da tastiera, ne stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii secondo la codifica RZ.

30 Scrivere un programma in linguaggio C con interfaccia a carattere che, una volta acquisito un byte da tastiera, ne stampi sullo schermo uno schema di onda quadra con i simboli grafici del codice Ascii secondo la codifica RZI.

B5 Il livello 2: Collegamento dati

Il livello 2 del modello OSI deve garantire il trasferimento privo di errori di un consistente blocco di dati detto, a questo livello, **trama** (o **frame**) tra due sistemi collegati direttamente (o adiacenti). Si dice anche che il livello 2 deve garantire una gestione completa **point-to-point**.

Le competenze tipiche del livello riguardano quindi:

- la delimitazione dei blocchi di dati, detta operazione di **framing** tra le diverse trame;
- l'aggiunta di codifiche dedicate all'individuazione e al **controllo degli errori**;
- la sincronizzazione della comunicazione delle trame, ovvero il **controllo del flusso**.

Il servizio globale che il livello 2 offre al livello superiore è quello di «nascondere» l'implementazione fisica della rete e offrire una comunicazione priva di errori con trame uniformi a prescindere dalla rete realizzata fisicamente. Il livello 2 infatti corrisponde al **confine della sottorete fisica**.

L'informazione ricevuta dal livello superiore L3 (Network) non tiene conto della rete fisica sottostante e quindi ne è indipendente; ciò causa una serie di situazioni, come esempio, che la grandezza del pacchetto N-PDU può non essere compatibile con la grandezza massima della trama (MTU) prevista per il livello L2 Datalink.

In questo caso il livello 2 dovrà operare frammentazione della N-PDU in svariate trame D-PDU.

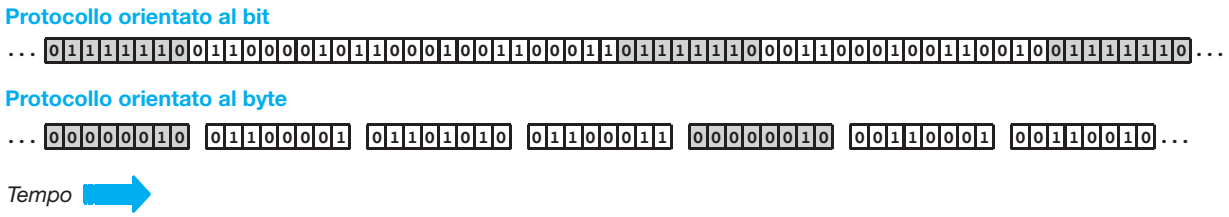
Si può affermare che il tipo di pacchetto di livello L2 Datalink da sempre preso come riferimento da quasi tutte le implementazioni di rete comprese le reti locali, è il pacchetto derivante dal protocollo «storico» **SDLC** (*Synchronous Data Link Control*), poi trasformatosi in **HDLC** (*High-Level Data Link Control*). In definitiva i diversi livelli L3 Network delle tipologie di reti esistenti si aspettano quasi sempre pacchetti di livello L2 piuttosto simili.

I protocolli di livello L2 possono essere distinti in due grandi categorie: protocolli **orientati al carattere** (o al byte) e protocolli **orientati al bit**.

Nel primo caso si tratta di protocolli il cui livello L1 Fisico gestisce direttamente gruppi di bit, 7 o 8 bit nelle comunicazioni asincrone come per esempio le interfacce Rs232C o il protocollo PPP. Le trame sono quindi gruppi di caratteri singoli gestiti autonomamente a livello L1.

Nel secondo caso si tratta di protocolli il cui livello L1 Fisico gestisce gruppi di bit relativamente lunghi in modalità sincrona, con clock di riferimento interno ai bit (come su LAN) o su linea dedicata.

Questi gruppi di bit non sono fisicamente separati tra loro all'interno delle serie circolanti sul mezzo, tuttavia logicamente rappresentano le singole informazioni generate:



Nella figura sono riprodotte due serie di bit che rappresentano i pacchetti "abc" e "12" in codice Ascii, rispettivamente per un protocollo orientato al bit e un protocollo orientato al byte:

'a' = 61h = (01100001)b
'b' = 62h = (01100010)b
'c' = 63h = (01100011)b
'1' = 31h = (00110001)b
'2' = 32h = (00110010)b

Nel primo caso la serie di bit è continua; in grigio sono messe in evidenza le sequenze speciali di bit separatrici (01111110)b che consentono di individuare i due pacchetti all'interno della serie.

Nel secondo caso la serie è composta da blocchi di 8 bit, e i due pacchetti sono separati da un blocco speciale (00000010)b.

Violazioni di codice

La **violazione del codice** è una tecnica di framing adottata dai protocolli orientati al bit che si basano su realizzazioni del livello fisico L1 nei quali è possibile discriminare con precisione un segnale fisico che non veicola dati.

È il caso del protocollo AMI e, in maggior misura, del protocollo di Manchester.

In questi casi l'apparato fisico che controlla il mezzo inserisce apposite sequenze di segnali di violazione della codifica tra le trame, cioè segnali non interpretabili come bit. Si tratta quindi di una operazione di framing che avviene per metà a livello L1.

Per la codifica di Manchester, la violazione del codice è un segnale con mancanza di transizioni nei bit time; dopo una violazione d'inizio trama, normalmente detta preambolo (preamble), si immette una sequenza di bit delimitatrice canonica detta **SFD** (*Start Frame Delimiter*).

1 Framing

Come si è visto, le informazioni sul mezzo viaggiano in modo indifferenziato in serie di bit o in gruppi di byte. Affinché le unità logiche di informazione, cioè i pacchetti o trame, possano essere ricostruite dal livello L2 Datalink ricevente, il livello L2 Datalink trasmettitore deve prevedere delle tecniche di delimitazione dei bit o dei byte. L'operazione è detta **framing**.

1.1 Delimitazione a carattere

Per protocolli orientati al byte, la delimitazione delle trame avviene inserendo uno o più caratteri speciali posti all'inizio e/o alla fine di una trama. L'inconveniente naturale di questa tecnica è rappresentato dall'eventuale presenza del carattere delimitatore all'interno della trama ma sottoforma di dato e non come delimitatore.

Esistono alcune soluzioni adottate dai protocolli per superare questa situazione:

- **Character stuffing** (riempimento di carattere). L'eventuale dato/delimitatore interno alla trama viene sempre duplicato in trasmissione e, in ricezione, all'estrazione del carattere delimitatore, si assegna il significato corretto in base al carattere letto successivamente. Caratteri delimitatori «storici» sono i codici Ascii STX (02h, *Start transmission*) e ETX (03h, *End transmission*) oppure i byte aah e 55h.
- **Tipo SLIP** (*Serial Line IP*). Metodo usato su trame di tipo IP tramite il carattere FEND (*Frame End*, Ascii c0h) posto all'inizio e al termine dei dati. All'interno dei dati si usano i caratteri speciali FESC (Ascii dbh), TFEND (Ascii dch) e TFESC (Ascii ddh) con le regole:

  1 un carattere dati che coincide con FEND, viene riempito con la sequenza FESC+TFEND;
  2 un carattere dati che coincide con FESC, viene riempito con la sequenza FESC+TFESC.

- **Esclusione di carattere**. Il protocollo impedisce la presenza dei caratteri delimitatori all'interno della parte dati della trama. Per esempio, limitando alla parte dati della trama i soli caratteri Ascii da 20h in su e usando come delimitatori i caratteri Ascii inferiori a 20h.

ESEMPIO

Character stuffing

Un livello L2 Datalink trasporta numeri senza segno su 16 bit. Delimitare il pacchetto dati di livello L3 Network che contiene i seguenti numeri 9127, 5083, 4288, 49153 con un char stuffing di tipo IP.

Prima di tutto si trasformano i numeri in esadecimale, per poter individuare eventuali caratteri speciali all'interno dei dati:

```
 9127 = 23a7h
 5083 = 13dbh
 4288 = 10c0h
49153 = c001h
```

Si decide per una rappresentazione di tipo big endian, più intuitiva, pertanto il pacchetto delimitato sarebbe:

FEND 23 a7 13 db 10 c0 c0 01 **FEND**

Si nota che all'interno dei dati compaiono caratteri speciali, lo stesso FEND (c0h) e FESC (dbh). È necessario fare char stuffing in base alle regole del tipo SLIP:

```
FEND 23 a7 13     db     10     c0         c0     01 FEND
                  ↓↑           ↓↑         ↓↑
FEND 23 a7 13 FESC TFESC 10 FESC TFEND FESC TFEND 01 FEND
```

In definitiva:

c0 23 a7 13 **db dd** 10 **db dc db dc** 01 **c0**

Spesso le tecniche di framing orientate al carattere usano immettere nell'header della trama anche il numero di caratteri contenuti nel campo dati della trama, in modo da fornire un'ulteriore verifica della delimitazione.

PROGRAMMA

Byte stuffing

Un livello L2 Datalink trasporta numeri senza segno su 16 bit. Applicare il byte stuffing con il valore AAh a un pacchetto dati che contiene i numeri 9157, 4778, 43523, 21771.

Un programma in linguaggio C potrebbe essere il seguente:

```
00 #include <stdio.h>
01
02 int main(void)
03 {
04     unsigned char byte[16];
05     unsigned int n0,n1,n2,n3;
06     int c,i,j,l;
07
08     n0 = 0x23C5; // 9157
09     n1 = 0x12AA; // 4778
10     n2 = 0xAA03; // 43523
11     n3 = 0x550B; // 21771
12
13     byte[0]= n0 >> 8; byte[1]= (unsigned char) n0;
14     byte[2]= n1 >> 8; byte[3]= (unsigned char) n1;
15     byte[4]= n2 >> 8; byte[5]= (unsigned char) n2;
16     byte[6]= n3 >> 8; byte[7]= (unsigned char) n3;
17
18     l = 4 * sizeof(short); // 4*2
19     printf("La sequenza originale: ");
20     for (i=0; i<l; i++) printf("%02x ",byte[i]);
21
22     c = 0;
23     for(i=0; i<l; i++)
24     {
25       if(byte[i]==0xAA)
26       {
27         for(j=l+1; j>i; j--)
28         {
29           byte[j]=byte[j-1];
30         }
31         byte[i]=0xAA;
32         i++;
33         c++;
34       }
35     }
36
37     printf("\nLa sequenza 'riempita': ");
38     for (i=0; i<l+c; i++) printf("%02x ",byte[i]);
39
40     return 0;
41 }
```

La preparazione del vettore di byte **`byte[]`** avviene estraendo il byte MSB del numero dato con uno scorrimento a destra di 8 bit (per esempio **`n0 >> 8`**) e il byte LSB sfruttando il cast (**`unsigned char n0`**).

Ottenuta la sequenza di byte originale, si effettua il controllo per il byte stuffing sul valore speciale AAh (**riga 25**).

Se si incontra, si traslano tutti i byte in avanti di una posizione (**righe 27÷30**) e si aggiunge il byte AAh (**riga 31**).

Si nota che la dimensione del vettore (8) viene calcolata sfruttando la dimensione del contenitore (**`sizeof(short)`**) che vale 2, ma solo se il compilatore è a 32 bit.

L'output del programma:

OUTPUT

```
La sequenza originale : 23 c5 12 aa aa 03 55 0b
La sequenza 'riempita': 23 c5 12 aa aa aa aa 03 55 0b
```

I due valori speciali aah che compaiono nei dati sono stati «riempiti» con aah aah aah aah.

1.2 Bit stuffing

La tecnica **bit stuffing** si applica ai protocolli orientati al bit. In particolare si parla di questa tecnica per i protocolli di linea tipo HDLC e derivati.

La delimitazione avviene con un gruppo di bit caratteristici e univoci, come ad esempio 7eh (=01111110b) per HDLC.

Come per il char stuffing, la presenza di configurazioni di bit identiche a quella adottata per la delimitazione sono escluse dal protocollo con l'inserimento, in trasmissione, di un bit a 0 per ogni sequenza di 5 bit a 1; in fase di ricezione, ad ogni gruppo di 5 bit a 1 la decodifica del pattern avviene sul bit successivo: se uguale a 1 si tratta di un delimitatore, se uguale a 0, lo si scarta e i successivi bit andranno a comporre normalmente i dati.

PROGRAMMA

Bit stuffing in HDLC

Data una sequenza binaria, operare il bit stuffing HDLC, cioè di un bit a zero per ogni sequenza di 5 bit a 1.

Per semplicità si opera con caratteri «binari», cioè i bit saranno rappresentati dai caratteri '0' e '1'.

Acquisita in input una sequenza di caratteri «binari», bisogna rilevare le sequenze di 5 bit consecutivi a 1; una volta individuata va inserito un bit a 0 (dopo i 5 bit a 1).

Un esempio di programma in linguaggio C potrebbe essere:

```
00 #include <stdio.h>
01 #include <string.h>
02
03 int main(void)
04 {
05     char bit[128];
06     int c,i,j,l,k;
07
08     printf("Immettere una sequenza binaria: ");
09     scanf("%s",bit);
10     l = strlen(bit);
11
12     for(i=0; i<l; i++)
13     {
14       if(bit[i]=='1')
15       {
16         k = i;
17          c = 0;
18          while(bit[k]=='1')
19          {
20            c++;
21            k++;
22            if(c == 5)
23            {
24              for(j=l+1; j>k; j--)
25              {
26                bit[j]=bit[j-1];
27              }
28              bit[k]='0';
29              l++;
30              break;
31            }
32            i = k;
33          }
34        }
35      }
36
37     printf("La sequenza 'riempita': %s",bit);
38     return 0;
39 }
```

PROGRAMMA

Si scandisce carattere per carattere il vettore di caratteri binari `bit[]` con un ciclo a conteggio (**riga 12**).

Quando si incontra un bit a 1, si effettua un ciclo che conta quanti bit consecutivi ad 1 possiede, come in **riga 18**. Se si arriva a 5 bit consecutivi (**riga 22**), si «fa un posto» traslando tutti i successivi (**righe 24÷27**), quindi si aggiunge il bit a zero (**riga 28**)

Il programma presenta il seguente layout:

OUTPUT

```
Immettere una sequenza binaria: 011111110
La sequenza 'riempita': 0111110110
```

Si nota che il settimo bit (che vale 0) è l'effetto del bit stuffing dopo la sequenza di 5 bit a 1.

2 Controllo dell'errore

Il livello L2 (Datalink) deve garantire una comunicazione tra sistemi direttamente connessi esente da errori.

Ciò non significa che necessariamente debba correggere l'eventuale errore dovuto al mezzo, ma che almeno lo rilevi richiedendo nuovamente il pacchetto errato o segnali la condizione di errore al livello superiore (L3 Network).

La teoria dei codici che si occupa della rilevazione e della correzione degli errori in una sequenza di **m** bit, si basa sull'aggiunta di una sequenza supplementare di **r** bit di servizio dedicati al controllo.

In totale si ottiene una sequenza di **n = m + r** bit denominata **codeword**.

Date due codeword di ugual lunghezza, si dice **distanza di Hamming** il numero di bit differenti tra le due sequenze, e viene indicato con **d**.

Per esempio, date le due codeword 01010101 e 01010011, la distanza di Hamming tra esse vale **2**: per confronto a parità di posizione, solo due bit sono diversi.

2.1 Codici di Hamming

La codifica di Hamming prevede sia l'individuazione dell'errore, sia la sua correzione da parte del ricevente.

Stabilito un numero massimo di errori **d** che possono aversi in una trama di **m** bit, bisogna aggiungere tanti bit **r** alla trama originale in modo tale che si ottenga un codice di codeword lungo **m + r** bit con distanza minima di Hamming **d + 1** per rilevare gli errori, oppure distanza minima di Hamming **2d + 1** per correggerli.

ESEMPIO

Codifica di Hamming

Si consideri una codifica di questo tipo, che è in grado di rappresentare 4 informazioni usando 10 bit:

dato	Codeword
0 (00)b	0000011111
1 (01)b	1111100000
2 (10)b	0000000000
3 (11)b	1111111111

Calcolarne il numero massimo di bit errati che può rilevare e quanti bit è in grado di correggere.

Si nota che la distanza minima di Hamming è **d = 5**, per esempio confrontando le due codeword centrali: nessun altro confronto tra codeword fornisce una distanza di Hamming inferiore.

Dalla formula riportata, il numero massimo di bit errati **e** che può rilevare, si ricava:

d = e + 1, cioè **5 = e + 1**, da cui **e = 4**.

ESEMPIO

In altri termini, la codifica proposta in tabella riesce a «capire» se una sequenza è errata anche se contiene 4 bit errati.

Dalla seconda formula si possono ricavare quanti bit `c` la codifica è in grado di correggere:

`d = 2c + 1`, cioè `5 = 2c +1`, da cui `c = 2`.

In altri termini, la codifica proposta è in grado di correggere la sequenza anche se in presenza di un errore doppio, cioè di due bit errati.

La **codifica di Hamming** insegna come costruire le codeword in questo senso.

Si noti dall'esempio che la codifica proposta è in grado di trasportare solo 4 informazioni. In altri termini, per un codice originale su due bit (4 informazioni), l'esigenza di correggere solo due errori obbliga alla creazione di un codice con un numero di bit 5 volte superiore, cioè un codice a 10 bit.

Per questo motivo le codifiche di Hamming *non* vengono usate nelle reti di calcolatori, che si «accontentano» di tecniche per la sola rilevazione dell'errore.

2.2 Checksum

Per i protocolli orientati al carattere la codifica di controllo a volte è implementata con l'aggiunta di caratteri di **checksum** (*somma di controllo*) sull'intera trama.

Il trasmettitore calcola con un semplice algoritmo (per esempio, una somma troncata al byte o alla word) uno o più caratteri di checksum e li accoda alla trama da spedire; il ricevente legge la trama e la checksum, calcola con il medesimo algoritmo la checksum a partire dai dati di trama e confronta la checksum calcolata con quella ricevuta: se i due valori discordano, la trama è errata.

ESEMPIO

Calcolo della checksum

Dato un pacchetto Ascii che contiene la sequenza di caratteri "Buongiorno", calcolarne la checksum troncata al byte.

Prima di tutto si trovano i codici Ascii dei singoli caratteri:

'B'=66; 'u'=117; 'o'=111; 'n'=110; 'g'=103; 'i'=105; 'o'=111; 'r'=114; 'n'=110; 'o'=111.

Quindi si effettua la somma aritmetica:

66+117+111+110+103+105+111+114+110+111 = 1058 = (422)h

Per troncare al byte la somma è sufficiente una maschera in AND opportuna con 00ffh:

`422h AND 00ffh = 22h`

Un algoritmo di checksum su due byte, denominato **Internet checksum**, è piuttosto diffuso:

1) si sommano i caratteri `char` del messaggio troncati alla word:

```
x = (x + char) AND ffffh
```

2) dopodiché, sul risultato **x**, si calcola il complemento a uno:

```
x = NOT x
```

Il modo di controllo dell'errore effettuato con il calcolo di checksum, quando è positivo, *non dà alcuna garanzia* sulla bontà della trama ricevuta, mentre garantisce la presenza di almeno un errore quando il controllo risulta negativo.

2.3 CRC

Tipicamente per i protocolli orientati al bit (ma estendibile anche ai protocolli orientati al byte), la codifica **CRC** (*Cyclic Redundancy Code*) consiste nell'immettere al termine della trama una sequenza di bit – spesso 16 o 32 – ricavati con un algoritmo dai bit presenti nella trama.

Analogamente al controllo mediate checksum, in ricezione il ricevitore ricalcolerà quei bit – a partire dai bit della trama – con il medesimo algoritmo, quindi li confronterà con quelli ricevuti.

CRC prevede che il trasmettitore aggiunga, ad ogni sequenza **dati** di bit da spedire, una sequenza supplementare di bit **crc**, in modo da rendere la sequenza totale **dati + crc** divisibile (in aritmetica modulo 2) senza resto, per una sequenza base **G(x)** concordata a priori e denominata **polinomio generatore**.

Il ricevitore, ricevendo la sequenza completa **dati + crc**, la divide (in aritmetica modulo 2) per **G(x)**, e se ottiene resto non nullo sa che tale sequenza contiene almeno un errore.

Polinomi generatori G(x) tipici utilizzati nelle reti di calcolatori sono i seguenti:

```
CRC_16: (11000000000000101)b
CRC_CCITT: (10001000000100001)b, usato in HDLC
CRC_32: (100000100110000010001110110110111)b, usato in 802.3
```

La teoria algebrica dei Campi garantisce, per i CRC a 16 bit, le seguenti qualità:

- rilevazione di tutti gli errori singoli e doppi;
- rilevazione di tutti gli errori con numero dispari di bit;
- rilevazione di tutti i **burst** (sequenze consecutive di errori) fino a 16;
- rilevazione al 99,997% dei burst di 17 errori (se gli errori sono casuali);
- rilevazione al 99,998% dei burst di 18 errori (se gli errori sono casuali).

Spesso la codifica e la decodifica CRC viene eseguita in hardware con opportune circuiterie.

Il campo contenente l'informazione CRC è detto – almeno sui pacchetti di rete locale – **FCS** (*Frame Control Sequence*), controllo di sequenza del frame.

ESEMPIO

Calcolo del CRC

Data la sequenza dati 10011010 da trasmettere, e scelto G(x)=1101, calcolare il CRC della trama.

1. Siccome i bit di **G(x)** sono quattro (grado del polinomio = 3), si genera una sequenza

   T(x) = dati + 000, cioè T(x) = 10011010000

2. Si divide **T(x)** in aritmetica 2 per il generatore, con operazioni di XOR successive per trovare il resto:

```
          ⌒⌒⌒⌒⌒⌒⌒⌒
          10011010000 : 1101
XoR       1101
          -1001
XoR        1101
          -1000
XoR         1101
          -1011
XoR          1101
          -1100
XoR           1101
          ---1000
XoR              1101
                 -101      Resto
```

3. Si sottrae da **T(x)** il resto in aritmetica 2 (ancora uno XOR), ottenendo una sequenza sicuramente divisibile per **G(x)** e senza resto.
4. La trama da trasmettere con CRC risulta quindi **10011010101**, dove:

   10011010 è il messaggio e
   101 è il CRC.

Il ricevitore dividendo la trama completa per **G(x)** deve ottenere resto nullo, altrimenti c'è almeno un errore nella trama.

PROGRAMMA

Calcolo del CRC

Scrivere un programma in linguaggio C che, data una sequenza dati binaria in codici Ascii e dato un polinomio generatore, calcoli il CRC della parte dati e stampi la sequenza dati+crc.

Un possibile programma in linguaggio C che calcola il CRC di una sequenza dati binaria in Ascii:

```
00 #include <stdio.h>
01 #include <string.h>
02
03 char dati[128],crc[32],g[32];
04 int i,a,c,n;
05
06 int main()
07 {
08   printf("Sequenza dati binari: ");
09   scanf("%s",dati);
10   a = strlen(dati);
11   printf("Polinomio G(x) : ");
12   scanf("%s",g);
13   n = strlen(g);
14
15   for(i=a; i<a+n-1; i++) dati[i]='0';
16   printf("Sequenza binaria T(x) %s",dati);
17
18   for(i=0; i<n; i++) crc[i]=dati[i];
```

PROGRAMMA

```
19   do
20   {
21     if(crc[0]=='1')
22     {
23       for(c = 1; c < n; c++)
24         crc[c] = ((crc[c] == g[c])?'0':'1'); // XOR
25     }
26     for(c=0; c<n-1; c++) crc[c]=crc[c+1];
27     crc[c]=dati[i++];
28   }
29   while(i<=a+n-1);
30
31   printf("\nIl CRC calcolato : %s",crc);
32
33   for(i=a; i<a+n-1; i++) dati[i]=crc[i-a];
34   printf("\nSequenza dati + CRC : %s",dati);
35
36   return 0;
37 }
```

Il programma, applicato all'esempio precedente, presenta il seguente layout:

OUTPUT

```
Sequenza dati binari  : 10011010
Polinomio G(x)        : 1101
Sequenza binaria T(x) : 10011010000
Il CRC calcolato      : 101
Sequenza dati + CRC   : 10011010101
```

Dopo aver preparato adeguatamente la stringa T(x) a **riga 18** e averla memorizzata nel vettore **`crc[]`**, il ciclo tra le **righe 19** e **29** effettua le 'divisioni' in aritmetica modulo 2.
In particolare l'operazione di XOR viene effettuata alla **riga 24**: se i bit corrispondenti sono uguali, il risultato è '0', altrimenti è '1', come dalla tabella di verità dell'operatore XOR.
Il test a **riga 24** è svolto in forma compatta:

```
crc[c] = ((crc[c] == g[c])?'0':'1')
```

In forma tradizionale diventa:

```
if (crc[c] == g[c]) crc[c] = '0'; else crc[c] = '1';
```

3 Controllo di flusso

Per controllo di flusso si intendono quelle tecniche che servono al trasmettitore e al ricevitore per condividere correttamente un canale trasmissivo half duplex (affinché non vi sia sovrapposizione) o, nel caso di canali full duplex, per gestire lo scambio delle trame in andata e ritorno con la maggiore efficienza possibile in termini di velocità e completezza.

Più in generale il controllo di flusso riguarda le operazioni che una coppia di sistemi a livello 2 devono compiere affinché una serie di trame vengano inoltrate correttamente da trasmettitore a ricevitore senza errori né duplicazioni, nel minor tempo possibile e con garanzia che tutti i frame siano stati spediti e ricevuti.

Sono infatti possibili eventuali condizioni erronee denominate **trama errata, trama duplicata** e **trama mancante** che impediscono un flusso continuo e regolare della comunicazione.

Per garantire l'effettivo flusso delle trame, il livello 2 Datalink usa pacchetti speciali di servizio denominati **riscontri** come **ACK** (*Acknowledge-*

ment) o **NACK** (*Not ACK*), che indicano a chi li riceve, normalmente il trasmettitore, l'avvenuta ricezione corretta o l'avvenuta ricezione scorretta di un frame da parte del ricevitore.

In molti casi il trasmettitore associa alla spedizione di una trama un **timeout**, un conteggio a tempo di tipo «countdown»: se il countdown scade senza che sia giunto un riscontro, il trasmettitore è in grado di sapere che la trama inviata è andata persa.

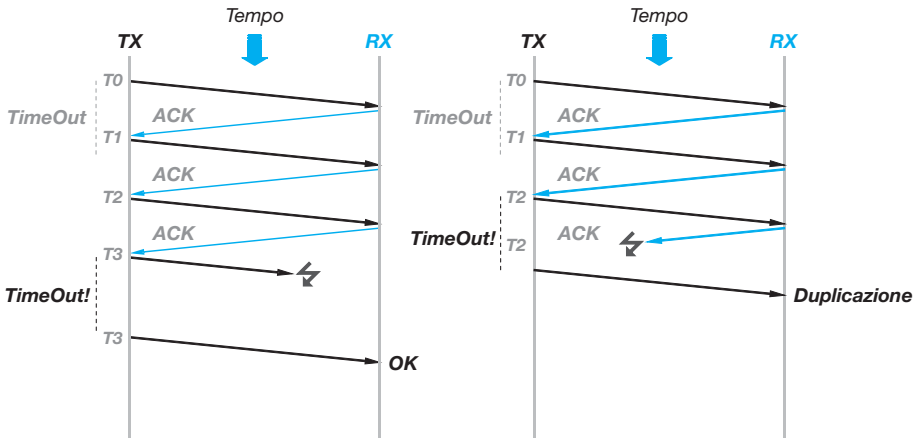
3.1 Stop and wait

Nel caso di protocolli orientati al carattere, la tecnica di controllo del flusso più adottata prende il nome di **Stop and wait**: su un canale generalmente half-duplex, il trasmettitore invia la trama e si ferma aspettando un riscontro dal ricevente, ACK o NACK.

Se il ricevitore riscontra un problema nella trama ricevuta, invia una trama NACK che obbligherà il trasmettitore a reinviare l'ultima trama spedita.

Il riscontro positivo ACK viene inoltrato dal ricevente non quando ha ricevuto la trama, ma quando è in grado di riceverne un'altra. Ciò significa che la trama ricevuta dovrà essere inoltrata ai livelli superiori e il ricevitore invierà l'ACK solo quando avrà avuto via libera da questi.

L'intera operazione è ripetuta per ogni trama inviata.



Il caso d'errore classico è definito **trama mancante**, e avviene quando una trama spedita non giunge al ricevitore.

Per prevenire il caso di trama mancante, il trasmettitore può attivare un timer all'invio di ogni trama; se il timer scade senza aver ricevuto riscontri (timeout), la trama verrà automaticamente ritrasmessa.

Stop and wait non riesce a gestire il problema dell'**ACK mancante**, qualora dovesse andar persa la trama di riscontro, sia essa ACK che NACK.

L'ACK mancante fa scattare il timeout del timer sulla trama precedentemente trasmessa, cosicché il trasmettitore reinvia la trama precedente. Il ricevitore non è in grado di stabilire se questa seconda trama è duplicata volontariamente o meno e quindi insorge il problema della **trama duplicata**.

Controllo di flusso per comunicazioni asincrone Rs232

Per gestire il controllo di flusso su una comunicazione seriale asincrona tipo EIA Rs232C nel caso di comunicazione half duplex, si possono usare almeno due metodologie consolidate, di carattere hardware (**RTS/CTS**) e software (**Xon/Xoff**).

RTS/CTS

Lo standard prevede, oltre ai segnali Tx, Rx e Gnd, due segnali fisici denominati **RTS** (*Request To Send*) e **CTS** (*Clear To Send*). Quando il Tx vuole trasmettere, alza la linea RTS. Quando il Rx rileva questo segnale, risponde alzando CTS, così che il Tx possa iniziare a trasmettere.
Per terminare la comunicazione il Rx può abbassare il CTS, cosicché il Tx interromperà disattivando l'RTS.
Si tratta di un classico protocollo fisico *Three Way Handshake*.

XON/XOFF

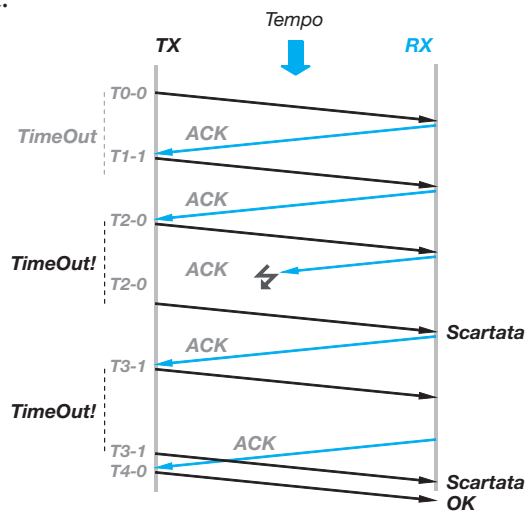
Senza utilizzare linee fisiche, è abbastanza consolidato il protocollo Xon/Xoff. In sostanza si attiva lo stesso meccanismo a tre vie descritto in precedenza, ma usando particolari byte sulla linea dati. Infatti Xon (Ascii 17) corrisponde al CTS, mentre Xoff (Ascii 19) all'RTS.

Lo stesso problema si presenta nel caso di un ricevitore lento, che invia il riscontro correttamente ma in ritardo (ACK ritardato), lasciando scattare, come prima, il timeout del trasmettitore e la successiva rispedizione di trama duplicata.

3.2 PAR

Il caso di trama ACK mancante o ACK ritardato obbliga il protocollo Stop and wait ad adottare una tecnica denominata **PAR** (*Positive Acknowledgement with Retransmission*), che consiste nell'aggiungere nell'header delle trame inviate un bit con lo scopo di conteggiare le trame in alternanza stretta (**trame numerate**).

Le trame ritrasmesse a causa di ACK mancanti o ritardati avranno lo stesso conteggio di bit della trama precedente. In questo modo il ricevitore, se riceve due trame consecutive con lo stesso valore di bit, scarta la seconda come duplicata.



In ogni caso si rivela critica la scelta del valore di tempo del timeout, necessario per regolare il flusso.

Un tempo troppo lungo limita le prestazioni in caso di numerosi errori; un tempo troppo corto, inferiore al tempo di trasmissione di una trama e ricezione di un riscontro detto **RTD** (*Round Trip Delay*), implica numerose inutili ritrasmissioni.

Il protocollo Stop and wait con PAR è poco efficiente, dovendo attendere un intero RTD per ogni singola trama, e ritorna veramente utile solo nei casi di canali half duplex.

3.3 Finestra scorrevole

Per evitare di attendere un RTD per ogni singola trama da spedire, si adotta il protocollo a **finestra scorrevole** (*Sliding Window*).

In questo caso è necessario numerare le trame da spedire (per esempio, T0, T1, T2,...) e numerare i relativi riscontri, positivi o negativi (per esem-

pio, ACK0, ACK1, ACK2,...), aggiungendo la numerazione nell'header del pacchetto.

I compiti di trasmettitore e ricevitore possono essere sintetizzati nel seguente modo:

- *Trasmettitore*. Il trasmettitore decide a priori una certa quantità di trame che è in grado di mantenere sempre pronte da spedire nel proprio buffer di trasmissione, denominando questa quantità **SWS** (*Send Window Size*) e ricorda il numero dell'ultima trama di cui ha ricevuto un riscontro positivo in un indice denominato **LAR** (*Last Ack Received*). Esso trasmetterà sempre tutte le trame contenute nella finestra di trasmissione, ovvero le trame che hanno indice maggiore di LAR, per un totale di SWS trame, attivando per ognuna un timer.

  Ad ogni ACK numerato ricevuto aggiornerà l'indice LAR con quel numero e la finestra si sposterà in avanti.
- *Ricevitore*. Il ricevitore si limita a spedire un riscontro numerato ad ogni trama ricevuta, positivo o negativo a seconda dei casi (si dice anche che il ricevitore ha una finestra di dimensione 1). Nel caso ricevesse una trama errata (NACK), nessuna altra trama successiva ricevuta sarà trattenuta fino alla ricezione corretta della trama errata.

  Quando riceverà una trama fuori ordine, situazione che indica trama mancante, scarterà quella trama e tutte le successive ricevute fino alla ricezione della trama mancante (altre versioni del protocollo impongono l'invio di un NACK numerato con la trama mancante oppure l'invio di un ACK numerato con l'ultima trama in ordine, ma è ininfluente).

Per ottimizzare il flusso, il trasmettitore può, all'avvio della comunicazione, imporre la dimensione della sua finestra a 1, aumentandola di una unità ogniqualvolta riceve un ACK fino a raggiungere l'ampiezza stabilita SWS.

Allo stesso scopo può decidere di decrementare la dimensione SWS per ogni timeout.

3.4 Go back n

Il protocollo a finestra scorrevole diventa molto efficiente se si adotta la tecnica **go back n**. Infatti le varie situazioni anomale che possono presentarsi sono affrontate in questo modo:

- **Trama persa.** Mancando una trama, il ricevitore avrà una successiva trama fuori ordine e quindi non invierà l'ACK numerato sulla trama persa. Le successive trame ricevute verranno scartate. Il trasmettitore non avanzerà più la sua finestra, non ricevendo l'ACK della trama persa e sarà fermo con la finestra piena di trame oramai già spedite. Scatterà il timeout del trasmettitore sulla trama persa, che gli imporrà di rispedirla. Il ricevitore confermerà con il relativo ACK numerato, il trasmettitore ritrasmetterà tutte le trame della finestra bloccata avanzando la finestra di un indice.
- **ACK mancante.** Il trasmettitore avrà la finestra ferma in attesa dell'ACK

(mancante), ma presumibilmente arriverà l'ACK della trama successiva. Ciò significa effettivamente «ACK mancante» per il trasmettitore che, a quel punto, lo considererà come giunto e avanzerà la finestra continuando la comunicazione. Infatti la ricezione di un ACK significa sempre che sono state riscontrate positivamente tutte le trame precedenti (un NACK eventualmente perso impedisce l'invio di ACK successivi da parte del ricevitore).

- **NACK mancante.** Se il trasmettitore non ottiene un NACK del ricevitore, non riceverà neppure successivi ACK. La finestra di trasmissione sarà ferma sul timer della trama che ha causato il NACK e, presumibilmente, piena di altre trame già spedite. Scatterà il timeout di questa trama e la reinvierà; pertanto, appena ricevuto il riscontro positivo, reinvierà tutte le trame in attesa nella sua finestra.

3.5 Selective reject

SOFTWARE DI EMULAZIONE DIDATTICA PER SLIDING WINDOW

Per migliorare considerevolmente l'efficienza dello Sliding Window Go Back N si può dotarlo del **selective reject** in modo che il ricevitore, invece di scartare tutte le trame ricevute successivamente ad una trama mancante (o ad una trama errata) le possa conservare in attesa del ripristino della comunicazione.

In questo caso anche il ricevitore deve dotarsi di un buffer in cui mantenere le trame ricevute ma non ancora accettate e ricordare su un indice l'ultima trama confermata. L'ampiezza della finestra di ricezione è indicata con **RWS** (*Receiver Window Size*) e con **LAF** (*Last Acceptable Frame*) l'indice dell'ultima trama accettata.

Come prima, le varie situazioni anomale che possono presentarsi sono affrontate in questo modo:

- **Trama persa.** Come prima, il ricevitore otterrà una trama fuori ordine, ma questa volta non scarterà le trame successive ricevute, spedendo per loro i relativi riscontri e salvandole nella propria finestra come non ancora accettate. Al trasmettitore scatterà il timeout sulla trama persa e la reinvierà. Ricevuta la trama che era fuori ordine e spedito il relativo ACK, il ricevitore potrà spostare l'indice LAF di numerose posizioni (dato che non ha scartato le trame successivamente ricevute), avendo ripristinato l'ordine di ricezione della propria finestra.

  Il trasmettitore, ricevendo l'ACK sulla trama mancante rispedita, si ritroverà la finestra già quasi completa, avendo ottenuto nel frattempo gli ACK delle trame successive in finestra. Potrà quindi spostare LFR di numerose posizioni e far avanzare la finestra per trasmettere diverse nuove trame.
- **ACK mancante.** Come Go Back N.
- **NACK mancante.** Analogo a trama persa, dato che il trasmettitore non riceve riscontro ad una sua trama (quella errata) e si bloccherà sul timer di quella trama. Alla scadenza reinvierà la trama avviando le fasi descritte per trama persa.

3.6 Piggybacking

Quando la tecnica a finestra scorrevole è adottata in una comunicazione full duplex con mittente A e destinatario B che devono scambiarsi trame di dati reciprocamente e contemporaneamente, si può applicare la tecnica di **piggybacking**.

Essa consente di evitare alcune delle trame dedicate ai riscontri prevedendo all'interno della trama di dati un campo dedicato al riscontro per la comunicazione nel verso opposto.

In questo caso quando la stazione A deve trasmettere un dato alla stazione B, la trama consegnerà, oltre al dato previsto, anche il riscontro (positivo o negativo) relativo alla contemporanea comunicazione nell'altro senso tra B e A.

Il piggybacking abbassa l'occupazione di banda per i riscontri, garantendo un servizio generale di conferme ancora affidabile e preciso.

4 Protocollo HDLC

I protocolli di livello 2 Datalink realmente utilizzati nelle reti WAN come Internet o LAN come Ethernet 802.3 derivano dal protocollo **HDLC** (*High Level Data Link Control*). Si può quindi affermare che i protocolli di livello L2 Datalink per le reti geografiche come PPP, e per reti locali come IEEE 802 LLC coprono gran parte dei casi reali, ed entrambi fanno riferimento a HDLC, pur se con qualche variante specifica.

Studiare HDLC significa quindi avere un'idea abbastanza precisa su come funzionino i protocolli di livello L2 nel loro complesso e per una grande quantità di reti, comprese X.25 e Frame Relay che, pur non trattate in questo libro, sono ancora piuttosto diffuse.

HDLC può operare in varie modalità, orientato al byte o orientato al bit, in half duplex o in full duplex, implementa il framing, il controllo dell'errore e il controllo di flusso con le principali tecniche descritte (bit stuffing, CRC e sliding windows), adattandosi a numerose realizzazioni fisiche.

4.1 Framing

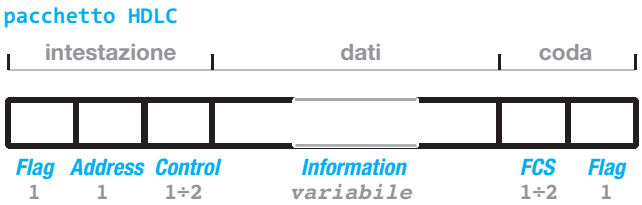
In modalità sincrona e orientata al bit, la trama è delimitata da due caratteri **flag** che corrispondono alla configurazione binaria 01111110 (7eh) e che marcano univocamente l'inizio e la fine di una trama.

Affinché il flag risulti un marcatore univoco, HDLC usa la tecnica del *bit stuffing* che garantisce che solo il carattere flag contenga sei 'uni' consecutivi. Infatti il bit stuffing analizza la trama prima di trasmetterla e inserisce un bit a zero dopo eventuali cinque 'uni' consecutivi indipendentemente dal valore del bit successivo; quindi giustappone i flag agli estremi.

Il ricevitore, se riceve una sequenza di cinque 'uni' e uno zero, elimina lo zero, che era stato inserito dal bit stuffing; se riceve sei 'uni' e uno zero identifica il carattere flag.

4.2 Pacchetto

Il formato della trama HDLC è riportato in figura ove si nota che la trama è composta da tre parti principali: un'intestazione (header), un campo dati (information) a lunghezza variabile e una coda (trailer). I numeri esprimono la dimensione in ottetti:



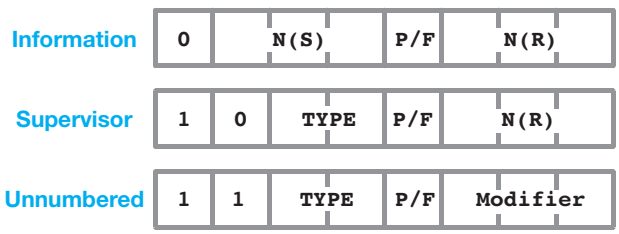
Il campo *Address* è lungo un ottetto e ha il significato di indirizzo della stazione. Per connessioni punto-punto non è considerato.

Il campo *Control* è un campo estremamente importante e può essere lungo uno o due ottetti.

Il campo *Information*, opzionale, è la parte dati del pacchetto ed è di dimensione variabile.

Il campo *FCS* (*Frame Control Sequence*) si occupa del controllo dell'errore con il modo CRC a 16 o a 32 bit.

La decodifica del campo *Control* è fondamentale. Esso determina i tre formati tipici del pacchetto, come mostrato in figura (versione a 8 bit).



- Il formato **Information** è usato per le trame che trasportano i dati e ha anche la possibilità di trasportare un riscontro per la trasmissione nella direzione inversa (piggybacking). Le trame di questo formato sono dette **I-frame**.

  Il campo *N(S)* è il numero di trama, mentre il campo *N(R)* è il numero di ACK nel verso opposto.
- Il formato **Supervisor** non prevede la presenza del campo *Information* nella trama ed è usato per trasportare informazioni di controllo relative agli I-frame; per esempio, fornire un ACK in assenza di traffico nella direzione opposta, operare il controllo di flusso, ecc. Le trame di questo formato sono dette **S-frame**.
- Il formato **Unnumbered** è utilizzato per due scopi diversi: trasportare dati di utente in modalità non connessa e trasportare messaggi di controllo del collegamento come inizializzazioni e diagnostica. Le trame di questo formato sono dette **U-frame**.

Il bit P/F (Poll/Final) non è particolarmente interessante.

ESEMPIO

Decodifica pacchetto HDLC

Decodificare il pacchetto HDLC, con campo control su un ottetto e FCS su due ottetti, considerando che il campo dati contiene ottetti di codici Ascii:

01111110111110111011000010100001101001111010011010100010100100000010101100100000100111111010111011011111010001111110

Con un po' di pazienza si cominciano a individuare i delimitatori, posizionati in effetti all'inizio e alla fine della trama. Senza delimitatori la sequenza risulta:

11111011101100001010000110100111101001101010001010010000001010110010000010011111101011111011011111010

Ora vanno eliminati i bit a zero del bit stuffing (in evidenza in grigio): dopo ogni sequenza di 5 'uni', si elimina il bit successivo (che deve essere uno zero).

La trama rimane:

1111111101100001010000110100111101001101010001010010000001010110010000010011111101111111101111110

Ora si possono estrarre i campi (in evidenza):

address: **11111111**

control: **01100001**, dove
- 01 indica **I-Frame**
- 100 indica **trama n. 4**
- 001 indica **ACK 1**

information:
- **01000011** = 43n = 'C'
- 01001111 = 4fh = 'O'
- 01001101 = 4dh = 'M'
- 01000101 = 45h = 'E'
- 00100000 = 20h = ' '
- 01010110 = 56h = 'V'
- 01000001 = 41h = 'A'
- 00111111 = 3Fh = '?'

FCS: **0111111101111110**

4.3 Flusso

U-frame

HDLC è un protocollo **negoziabile**, ovvero contiene comandi che possono essere usati per modificare il proprio funzionamento, come per esempio stabilire l'ampiezza del campo *Control* o del campo FCS. Inizialmente HDLC funziona sempre in modo normale, quindi, attraverso U-frame può negoziare. Alcuni tipi di U-frame, distinguibili in funzione dei valori assunti dai sottocampi **type** e **modifier** del campo control, sono di seguito elencati.

→

Nella modalità connessa è necessario numerare le trame. HDLC prevede due possibili numerazioni alternative: la prima impiega un numero di trama su tre bit (modulo 8) come riportato nella figura del formato del pacchetto HDLC, mentre la modalità estesa impiega un numero di trama su 7 bit (modulo 128).

I sottocampi N(S) e N(R) del campo *Control* sono destinati ad ospitare i numeri di trama. Quando si opera in modalità normale (modulo 8) il campo *Control* è sempre lungo 8 bit (1 ottetto), poiché N(S) e N(R) sono lunghi 3 bit ciascuno. Quando invece si opera in modalità estesa (modulo 128), N(S) e N(R) sono lunghi 7 bit e quindi il campo *Control* è lungo un ottetto nel caso degli U-frame, due ottetti nel caso degli I-frame e degli S-frame.

Gli S-frame sono usati in associazione agli I-frame per implementare un controllo di flusso a finestra scorrevole con Go back n e Selective reject. Il sottocampo N(R) del campo *Control* contiene il numero di sequenza del

prossimo frame che la stazione si aspetta di ricevere; questo serve anche da ACK per tutti i frame con numero di sequenza minore di N(R).

Sono previsti quattro tipi di S-frame, distinguibili in funzione dei valori assunti dai due bit del sottocampo **type** del campo *Control*:

- **RR** (00b, *Receiver Ready*). È un frame utilizzato per fornire un ACK in assenza di traffico, oppure per indicare che la stazione è pronta a ricevere nuovi I-frame.
- **RNR** (01b, *Receiver Not Ready*). È un frame utilizzato per indicare che la stazione è impossibilitata a ricevere altri I-frame.
- **REJ** (10b, *REJect*). È un frame utilizzato per chiedere la ritrasmissione di tutti gli I-frame già inviati a partire da quello con numero di sequenza N(R).
- **SREJ** (11b, *Selective REJect*). È un frame utilizzato per chiedere la ritrasmissione del solo I-frame con numero di sequenza N(R).

→

SABM/SABME (*Set Asynchronous Balanced Mode*). Trama utilizzata per inizializzare connessioni con numerazioni su 3 o 8 bit.

UI (*Unnumbered Information*). Trama utilizzata per inviare dati in modalità non connessa. È molto utilizzata dalla variante LLC, che opera sulle LAN.

XID (*eXchange station IDentification*). Trama usata per negoziare parametri tra le stazioni, per esempio l'uso di FCS su 16 o 32 bit.

PROGRAMMA

Decodifica del tipo di pacchetto HDLC

Scrivere un programma in linguaggio C che dato un byte che rappresenta il campo Control di un pacchetto HDLC, stampi di che tipo di pacchetto si tratta: Information, Supervisor o Unnumbered.

Sfruttando maschere di bit opportune, un esempio di codice C potrebbe essere:

```
00 #include <stdio.h>
01
02 #define MASK_FRAME (0xC0) //C0h = 11000000b
03
04 #define SUPERVISOR (0x80) //80h = 10000000b
05 #define UNNUMBERED (0xC0) //C0h = 11000000b
06
07 int main(void)
08 {
09     int i;
10     unsigned char control;
11
12     control = 0x61; // 0x61 = 01100001b
13
14     printf("Il pacchetto HDLC con control = %02xh", control);
15
16     if ((control & MASK_FRAME) == SUPERVISOR)
17     {
18       printf(" e' di tipo Supervisor");
19     }
20     else
21     {
22       if ((control & MASK_FRAME) == UNNUMBERED)
23       {
24         printf(" e' di tipo Unnumbered");
25       }
26       else
27       {
28         printf(" e' di tipo Information");
29       }
30     }
31
32     return 0;
33 }
```

Il programma presenta il seguente layout:

OUTPUT

```
Il pacchetto HDLC con control = 61h e' di tipo Information
```

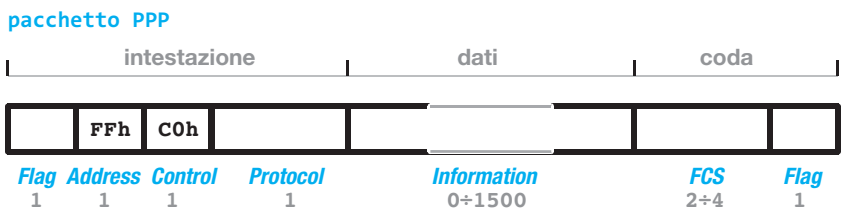
5 Protocollo PPP

Il protocollo **PPP** (*Point-to-Point Protocol*), usato sul lato linea (DCE-centrale) per la maggior parte delle reti WAN come Internet, deriva da HDLC in modalità orientata al carattere.

La differenza più importante con HDLC è l'aggiunta di un campo nel formato del pacchetto che consente di capire quale protocollo di livello superiore ha generato la trama PPP, consentendo a PPP di veicolare anche contemporaneamente trame appartenenti a protocolli differenti.

5.1 Pacchetto

Il formato del pacchetto PPP è mostrato in figura:



Il nuovo campo di PPP si chiama **Protocol** ed è lungo 2 ottetti. Tale campo contiene la codifica del protocollo di livello superiore utilizzato ma anche la codifica di un sottoprotocollo PPP, la cui PDU è contenuta nel campo *Information*. Si nota che PPP pone limitazioni ai valori leciti per alcuni campi e in particolare:

- Il campo *Address* deve sempre contenere la sequenza binaria 11111111. PPP non assegna indirizzi alle stazioni essendo un protocollo punto-punto.
- Il campo *Control* deve sempre contenere la sequenza 11000000, cioè la trama deve essere un U-frame. La lunghezza del campo *Control* è quindi sempre pari a un ottetto e la trasmissione è di tipo non connesso.
- Il campo *Information* ha una lunghezza compresa tra 0 e 1500 ottetti. La lunghezza massima può essere modificata di comune accordo dalle stazioni.
- Il campo *FCS* ha una lunghezza di 2 ottetti, ma può essere portata a 4 ottetti di comune accordo dalle stazioni.

PPP usa una serie di **sottoprotocolli**, sempre utilizzando il formato di trama descritta, per negoziare il tipo di protocollo di livello L3 Network da usare, la compressione dei dati, la cifratura dei dati e l'autenticazione. In realtà una trama PPP generica contiene, nel campo *Information*, uno specifico pacchetto da interpretarsi con il valore contenuto nel campo *Protocol*. PPP è un esempio di protocollo a tunnel.

I principali sottoprotocolli di PPP ne determinano anche le fasi di lavoro:

- fase **Establish**, con sottoprotocollo **LCP** (*Link Control Protocol*). Si negoziano i parametri tipo l'MTU, se prevista o meno la compressione dei dati, la cifratura, l'autenticazione, ecc.

- Fase **Authenticate** (se richiesta), con sottoprotocollo **CHAP** (*Challenge-Handshake Authentication Protocol*). Dato che PPP è un protocollo di linea pubblica, spesso è necessario che mittente e destinatario si autentichino a vicenda (come avviene con le connessioni ad Internet).
- Fase **Network**, con sottoprotocollo **NCP** (*Network Control Protocol*). Si decide quale protocollo di livello 3 Network deve essere usato.
- Fase di **Flusso**, effettivo scambio dei dati attraverso il sottoprotocollo identificato nella fase precedente, per esempio il protocollo **IPNCP**, responsabile anche della negoziazione di un eventuale indirizzo IP dinamico.

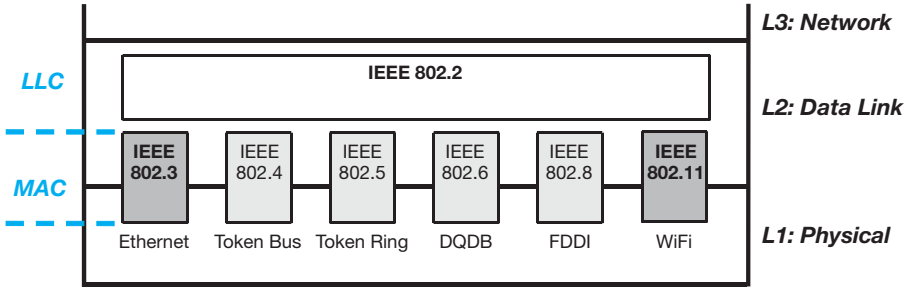
Altri sottoprotocolli noti per PPP sono **CCP** (*Compression Control Protocol*) per la compressione e **ECP** (*Encryption Control Protocol*) per la cifratura dei dati.

In una connessione tipica host-ISP su ADSL, come ad esempio tra un pc casalingo e il provider Internet, si usa comunemente PPP, eventualmente nelle versioni **PPPoE** (*PPP over Ethernet*) o **PPPoA** (*PPP over Atm*) se il personal computer è connesso al modem ADSL con una connessione Ethernet o con una connessione USB.

6 Protocollo LLC

Il protocollo di livello 2 Datalink **LLC** (*Logical Link Control* o IEEE 802.2) è in effetti un protocollo «sottile», che ha il compito di uniformare le varie tecnologie di rete locale LAN verso il livello superiore L3 Network.

Date infatti varie tecnologie di accesso al mezzo su rete locale, come Ethernet 802.3 (e successive) oppure Wi-Fi 802.11 (e successive), ma anche modelli di rete locale radicalmente differenti come **Token Ring 802.5** o **DQDB 802.6**, ognuna di queste tecnologie utilizza uno strato intermedio tra L2 Datalink e mezzo fisico – non previsto da OSI – denominato **MAC** (*Medium Access Control*).



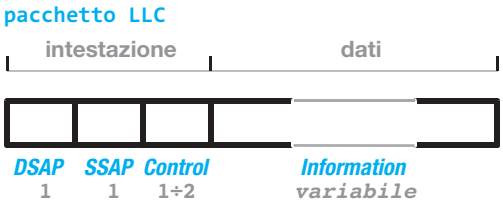
Ogni MAC è quindi molto specifico, pur conservando le nozioni fondamentali relativamente agli indirizzi MAC del proprio adattatore di rete (NIC).

In definitiva il protocollo LLC si limita a presentare in modo standardizzato il contenuto del pacchetto MAC della tecnologia adottata sul nodo.

6.1 Pacchetto

Il pacchetto LLC è una versione semplificata del pacchetto HDLC dato che, appoggiandosi su uno strato intermedio MAC e non su un livello fisico non ha necessità né di delimitare le trame, né di fare controllo di errore (è senza coda), operazioni demandate allo strato MAC.

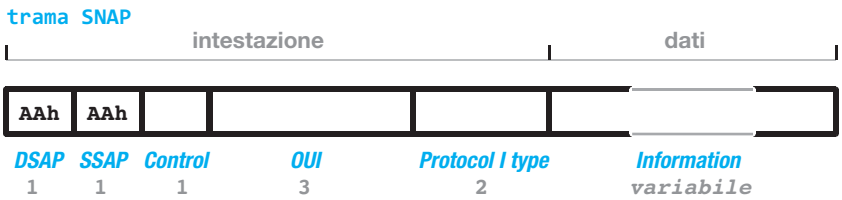
Inoltre le reti locali, per definizione e progettazione, non soffrono di pacchetti mancanti, per cui non è necessario neppure che LLC utilizzi un qualsiasi protocollo di tipo **ARQ** (*Automatic Repeat-reQuest*) per la gestione del flusso, operazione demandata ai livelli superiori (questa volta tipicamente al livello 4 di Trasporto).



I due campi *DSAP* e *SSAP* in realtà sono sempre uguali e contengono l'identificativo del protocollo di livello L3 Network che ha generato – o a cui è destinato – il pacchetto.

Il campo *Control* assume gli stessi valori riportati in HDLC.

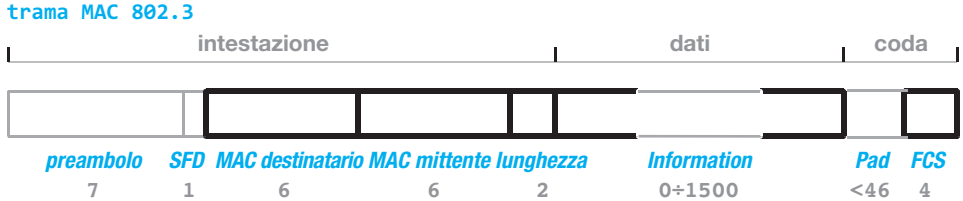
Siccome il campo *DSAP/SSAP* è lungo solo un ottetto, può codificare al massimo 256 tipi di protocolli di L3 Network differenti; l'OSI ha introdotto una variante al pacchetto 802.2 LLC denominato **SNAP** (*SubNetwork Access Protocol*), il cui formato è il seguente:



Il valore AAh per il campo *DSAP/SSAP* identifica il pacchetto SNAP, l'*OUI* identifica l'organizzazione che ha proposto il protocollo e *Protocol type* identifica il protocollo di livello L3 Network a cui appartiene il pacchetto.

Il pacchetto SNAP è fondamentale per le reti Ethernet. Esso consente che sulla stessa rete Ethernet possano circolare anche i pacchetti MAC progettati prima della standardizzazione di Ethernet in 802.3, pacchetti MAC denominati **Ethernet 2**.

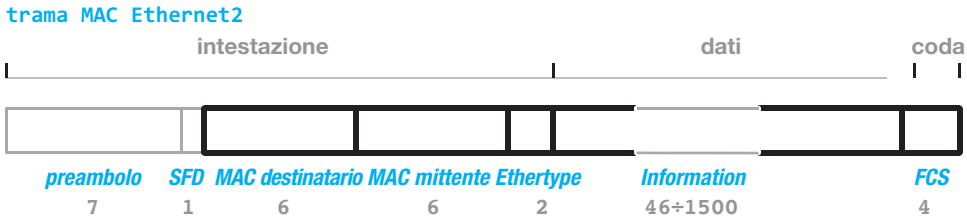
I formati dei pacchetti MAC Ethernet 2 e MAC 802.3 sono leggermente diversi; inoltre il pacchetto MAC Ethernet 2 non prevede di essere imbustato nel pacchetto LLC, a differenza di 802.3.



Ethernet 2 e Ethernet 802.3

Se l'ufficializzazione dello standard 802.3 da parte di IEEE e ISO risale al 1989, le reti Ethernet erano già attive fin dalla seconda metà degli anni '70 a cura del consorzio di produttori Digital, Intel e Xerox (DIX) che avevano ufficializzato la tecnologia **Ethernet 2** in un documento del 1982.

Tra le altre particolarità, Ethernet 2 non prevede la presenza del livello LLC, cioè manca del livello 2 Datalink dato che non c'era alcuna necessità di uniformare altri MAC essendo Ethernet 2 una rete proprietaria. I compiti del livello 2 Datalink erano quindi svolti dal MAC di Ethernet 2, in particolare l'individuazione del protocollo di livello 3 trasportato dal pacchetto, e codificato in un campo apposito del MAC detto *Ethertype*.



Il *Preamble* sincronizza l'inizio del pacchetto. In assenza di segnale sul cavo, questa prima informazione serve per la sincronizzazione iniziale e per tarare i clock delle stazioni sulla linea. Contiene violazioni del codice di Manchester.

Il campo SFD (*Start Frame Delimiter*) è ulteriore delimitatore iniziale di pacchetto che contiene una sequenza di bit canonici che consentono all'hardware di sincronizzare i seguenti bit di trama.

Si notano quindi i campi dedicati agli indirizzi fisici (MAC Address) che, come si è visto, devono essere ampi 6 byte: **indirizzo MAC del destinatario** e **indirizzo MAC del mittente**.

Il campo FCS ha la funzione di garantire l'integrità dello stesso. È implementato a CRC con campi di Galois.

I pacchetti si differenziano nel campo *lunghezza/Ethertype*.

In 802.3 si stabilisce il tipo di protocollo di livello 3 Network all'interno del campo *Information*, che è la PDU di LLC di tipo SNAP: dentro il pacchetto SNAP c'è *Protocol type*.

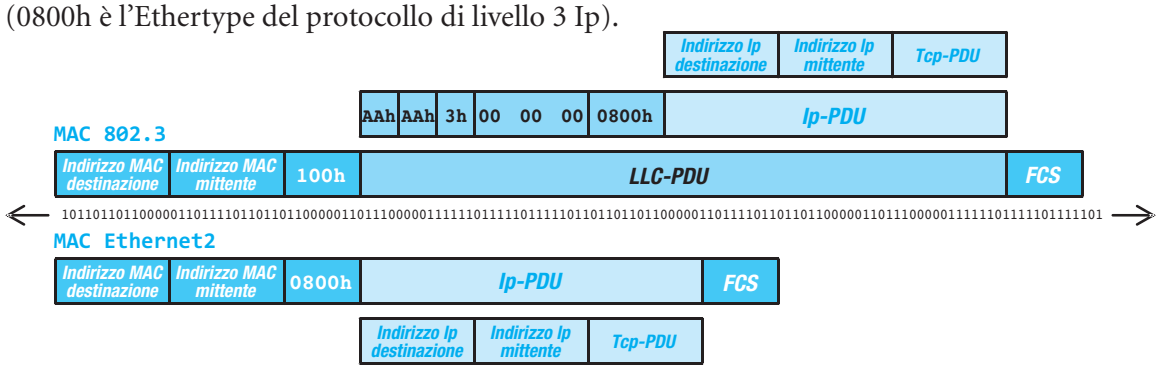
In Ethernet 2 invece non c'è LLC e circola direttamente il pacchetto MAC contenente *Ethertype* per individuare il tipo di protocollo di livello 3 Network.

La lunghezza del campo *Information*, per entrambi i casi non deve superare 1500 ottetti e non deve essere inferiore a 46 ottetti per rispettare i vincoli di CSMA/CD. Entrambi i tipi di pacchetto convivono sulla medesima LAN dato che il campo *lunghezza* non può superare 1500 (5dch) mentre *Ethertype* contiene codifiche sempre superiori a 1536 (600h).

Una scheda di rete NIC in ricezione controlla il campo *lunghezza/Ethertype* e:

- se il valore è superiore a 1500, si tratta di un pacchetto Ethernet 2 e il valore è l'*Ethertype* di livello 3 Network che prenderà in carico direttamente il pacchetto;
- se il valore è minore o uguale a 1500, allora il pacchetto è 802.3 e il campo *Information* conterrà lo SNAP al cui interno si trova il *Protocol type* di livello 3 Network a cui consegnare il pacchetto.

I due tipi di pacchetti, applicati ad uno stesso segmento Ip generato su una LAN, circolano in formato MAC con i due schemi rappresentati in figura (0800h è l'Ethertype del protocollo di livello 3 Ip).



ESERCIZI PER LA VERIFICA ORALE

Saper rispondere ai **requisiti fondamentali** dà una sufficiente garanzia per sentirsi pronti all'interrogazione. Saper anche rispondere ai **requisiti avanzati** dimostra una padronanza eccellente degli argomenti del capitolo.

Requisiti fondamentali

1. Elencare e commentare i compiti del livello 2 (Collegamento dati o Datalink).
2. Ricordare le due categorie di protocolli operanti a livello 2 (Collegamento dati) e commentarle.
3. Illustrare la nozione di frame (trama) e spiegare perché è fondamentale.
4. Esporre la tecnica di framing con character stuffing. Fare un esempio.
5. Saper calcolare la distanza di Hamming tra due codeword a piacere. Fare un esempio.
6. Commentare le proprietà di una codifica di Hamming.
7. Illustrare la tecnica di controllo tramite checksum e indicarne le proprietà.
8. Spiegare a cosa serve il controllo del flusso a livello 2 (Collegamento dati) ed elencarne tutte le tecniche.
9. Illustrare la tecnica di controllo del flusso PAR in relazione alla tecnica Stop & Wait.
10. Ricordare quale protocollo è la base di molti protocolli di livello 2 (Collegamento dati) e i nomi dei protocolli di livello 2 più diffusi su LAN e WAN.
11. Elencare i tipi di pacchetto HDLC e commentarli.
12. Mettere in relazione i protocolli HDLC e PPP.
13. Collocare LLC nello schema a livelli OSI e commentare il livello MAC.

Requisiti avanzati

1. Spiegare il concetto espresso dalla locuzione «point-to-point» e di «sottorete fisica».
2. Esporre la tecnica di framing con bit stuffing. Fare un esempio.
3. Illustrare il concetto sotteso dal termine codeword.
4. Mostrare l'algoritmo per il calcolo della Internet checksum su un pacchetto a piacere. Discuterne le proprietà
5. Illustrare le operazioni in aritmetica modulo 2 utilizzate per il calcolo del CRC.
6. Spiegare le nozioni di trama errata, mancante e duplicata.
7. Illustrare il ruolo del timeout nel controllo del flusso, riportando un esempio tratto da una tecnica a piacere.
8. Illustrare la tecnica di controllo del flusso denominata Sliding window (finestra scorrevole).
9. Spiegare le sigle SWS, LAR, SWR e LAF.
10. Spiegare la tecnica di piggybacking: quando si può applicare e perché è efficiente.
11. Riportare lo schema del pacchetto HDLC e commentarne i campi.
12. Spiegare cosa significa che HDLC è un protocollo negoziabile.
13. Mostrare i principali sottoprotocolli usati da PPP e la loro funzione.

ESERCIZI PER LA VERIFICA SCRITTA

I prerequisiti richiesti per affrontare questi esercizi sono la conoscenza dei sistemi di numerazione in base 2 e in base 16 (binario ed esadecimale), la conversione decimale-binario e viceversa, la conversione decimale-esadecimale e viceversa, anche con applicativi (per esempio Calc).

È necessaria la consultazione di una tabella di codici Ascii.

Framing

1 Un livello L2 Datalink trasporta numeri senza segno su 16 bit. Delimitare il pacchetto dati di livello L3 Network che contiene i seguenti numeri 40961, 4315, 448, 254 con un char stuffing di tipo IP.

2 Un livello L2 Datalink trasporta numeri senza segno su 16 bit e delimitatore iniziale aah. Delimitare il pacchetto dati di livello L3 Network che contiene i seguenti numeri 257, 2730, 40978, 2727.

3 Un livello L2 Datalink trasporta numeri senza segno su 16 bit con delimitatore iniziale 55h e finale aah. Delimitare il pacchetto dati di livello L3 Network che contiene i seguenti numeri 40961, 4522, 21761, 12.

4 Un livello L2 Datalink trasporta numeri senza segno su 16 bit con un char stuffing di tipo IP. Estrarre i numeri presenti nel pacchetto c0h 23h a7h 13h dbh ddh 22h dbh dch dbh dch 01h c0h.

5 Un livello L2 Datalink trasporta numeri senza segno su 16 bit con un char stuffing sul delimitatore 55h. Estrarre i numeri presenti nel pacchetto 55h 23h a7h 55h 55h ddh 22h dbh 55h 55h dch 01h 04h 55h.

6 Un livello L2 Datalink trasporta numeri senza segno su 16 bit con un char stuffing sul delimitatore iniziale aah e finale 55h. Estrarre i numeri presenti nel pacchetto aah 33h a7h aah aah aah aah dbh 55h 55h dch 01h 04h ffh 55h.

7 Un livello L2 Datalink trasporta codici Ascii con un char stuffing sul delimitatore iniziale e finale 55h. Costruire il pacchetto che trasporta la stringa «Unforgiven».

8 Un livello L2 Datalink deve usare una delimitazione a esclusione di carattere con delimitatori STX e ETX e deve trasportare numeri interi senza segno su 16 bit. Costruire il pacchetto che trasporta i numeri 8707, 3, 256.
Nota: Siccome il protocollo è ad esclusione di carattere il byte 03h non può comparire nei dati. Quindi bisogna codificare i numeri in Ascii. Considerare anche che i numeri devono essere su 16 bit.

9 Un livello L2 Datalink orientato al bit trasporta dati in codici Ascii e delimita con la sequenza binaria 01111110b.
Mostrare il pacchetto «binario» per la stringa «Oppala».

10 Un livello L2 Datalink orientato al bit trasporta dati in codici Ascii (7 bit) e delimita con la sequenza binaria 01111110b.
Mostrare il pacchetto «binario» per la stringa «Oibo».

11 Un livello L2 Datalink orientato al bit trasporta numeri interi senza segno su 16 bit e delimita con la sequenza binaria 01111110b.
Mostrare il pacchetto «binario» per i numeri 12, 253, 127, 1017.

Hamming

12 Indicare le distanze di Hamming per le coppie di numeri 12, 13; 255, 256; a027h, a028h; ffffh, 0101h.

13 Indicare la distanza di Hamming delle due stringhe «Ivan» e «Navi».

14 Indicare la distanza minima di Hamming per i numeri 12, 13, 127, 255.

15 Indicare la distanza minima di Hamming tra le stringhe «Vani», «Ivan» e «Navi».

16 Data la tabella

dato	Codeword
0 (00)b	0000011111
1 (01)b	1111100000
2 (10)b	0000000000
3 (11)b	1111111111

codificare il numero 2800.

17 Data la tabella

dato	Codeword
0 (00)b	00001111
1 (01)b	11110000
2 (10)b	00000000
3 (11)b	11111111

stabilire il numero massimo di bit errati che può rilevare e quanti bit è in grado di correggere.

18 Data la tabella

dato	Codeword
0 (00)b	00001111
1 (01)b	11110000
2 (10)b	00000000
3 (11)b	11111111

codificare il numero 13071 e quindi inserirlo in un pacchetto di livello 2 Datalink con delimitatore 01111110b.

Controllo dell'errore

19 Dato un pacchetto Ascii che contiene la sequenza di caratteri «Gran Torino», calcolarne la checksum troncata al byte.

20 Dato un pacchetto Ascii che contiene la sequenza di caratteri «Mystic river», calcolarne la checksum troncata alla word.

21 Data una checksum troncata al byte che vale 34h, indicare due pacchetti che generano questa checksum.

22 Dato un pacchetto di livello 2 Datalink che trasporta numeri interi senza segno su 16 bit, calcolare la checksum del pacchetto composto dai numeri 40961, 4522, 21761, 12.

23 Individuare due stringhe differenti che originano la stessa checksum troncata al byte.

24 Calcolare il CRC della sequenza 10101010b con G(x)=1101.

25 Calcolare il CRC della sequenza 10101010b con G(x)=1011.

26 Calcolare il CRC della sequenza 1110101010b con G(x)=1101.

27 Data la sequenza crc+dati=11110000010b e G(x)=1101b, verificare se il pacchetto è corretto.

28 Data la sequenza 11110001010b (dati+crc) e G(x)=1101b, verificare se il pacchetto è corretto.

29 Data la sequenza 11110000010b (dati+crc) e G(x)=1101b, verificare se il pacchetto è corretto.

30 Data la sequenza 1100110011001001b (dati+crc) e G(x)=11101b, verificare se il pacchetto è corretto.

31 Data la sequenza 1101110011101001b (dati+crc) e G(x)=11101b, verificare se il pacchetto è corretto.

HDLC

32 Decodificare il pacchetto HDLC, con campo control su un ottetto e FCS su due ottetti, considerando che il campo dati contiene ottetti di codici Ascii:

```
01111110111110001111000110100001101001111010011010100010100100000010101100100000
10011111010111110010111110000111111 0.
```

33 Data la sequenza 01110001b che rappresenta il campo control di un pacchetto HDLC, decodificarla riportando tutti i dati che contiene.

34 Decodificare il pacchetto HDLC, con campo control su un ottetto e FCS su due ottetti, considerando che il campo dati contiene ottetti di codici Ascii:

```
01111110111110111010110110100100101001111001010110010010001101010100010101010
10100110111110101111100100111111 0.
```

35 Data la sequenza 10110101b che rappresenta il campo control di un pacchetto HDLC, decodificarla riportando tutti i dati che contiene.

36 Costruire il campo control per HDLC che rappresenta un NACK per la trama 7. Riportare il valore binario.

37 Costruire il campo control per HDLC che rappresenta un ACK per la trama 100. Riportare il valore binario.

38 Costruire il campo control per HDLC che rappresenta un ACK per la trama 5 e un ACK piggybacked della trama 3. Riportare il valore binario.

ESERCIZI PER LA VERIFICA DI LABORATORIO

I prerequisiti per affrontare questi esercizi sono la consultazione di una tabella di codici Ascii e la disponibilità di un ambiente di sviluppo per programmi in linguaggio C. Per compilare gli esercizi proposti è stato usato l'ambiente gratuito multipiattaforma Windows/Linux ***Code::Blocks 10.5*** con **gcc 4.4.1**.

I testi degli esercizi presentano uno o più **layout** di input/output richiesti; in questo modo sono indicate, a volte, alcune specifiche del programma come l'output, il controllo dell'input o i valori ammissibili.

1 **Scrivere un programma in linguaggio C che, acquisiti da tastiera 3 numeri interi senza segno rappresentabili su 16 bit, crei un pacchetto di livello L2 Datalink con delimitatore 55h.**

Layout:

OUTPUT

```
Inserire numero 1: 9121
Inserire numero 2: 4693
Inserire numero 3: 35

La sequenza originale : 23 a1 12 55 00 23
Il pacchetto: 55 23 a1 12 55 55 00 23 55
```

2 **Scrivere un programma in linguaggio C che, acquisita da tastiera una stringa, crei un pacchetto di livello L2 Datalink con delimitatori STX ed ETX.**

Layout:

OUTPUT

```
Inserire stringa: Un mondo perfetto

Il pacchetto:
02h, 55h, 6eh, 20h, 6dh, 6fh, 6eh, 64h,
6fh, 20h, 70h, 65h, 72h, 66h, 65h, 74h,
74h, 6fh, 03h
```

3 **Scrivere un programma in linguaggio C che, acquisita da tastiera una stringa, crei un pacchetto di livello L2 Datalink con delimitatore aah e 55h.**

Layout:

OUTPUT

```
Inserire stringa: Un mondo perfetto

Il pacchetto:
aah, 55h, 55h, 6eh, 20h, 6dh, 6fh, 6eh,
64h, 6fh, 20h, 70h, 65h, 72h, 66h, 65h,
74h, 74h, 6fh, 55h
```

4 **Scrivere un programma in linguaggio C che, acquisita da tastiera una stringa, crei un pacchetto di livello L2 Datalink con delimitatore 55h.**

Layout:

OUTPUT

```
Inserire stringa: LA RECLUTA

Il pacchetto:
55h, 4ch, 41h, 20h, 52h, 45h, 43h, 4ch,
55h, 55h, 54h, 41h, 55h
```

5 **Scrivere un programma in linguaggio C che, acquisita da tastiera una stringa, ne mostri la rappresentazione binaria e stampi quanti bit sono a 1 e a 0.**

6 **Scrivere un programma in linguaggio C che, acquisita da tastiera una stringa, crei un pacchetto di livello L2 Datalink con delimitatori STX ed ETX e ne stampi la sequenza binaria.**

Layout:

OUTPUT

```
Inserire stringa: Gli spietati

Il pacchetto binario:
00000010.01000111.01101100.01101001.001000
00.01110011.01110000.01101001.01100101.011
10100.01100001.01110100.01101001.00000011
```

7 **Scrivere un programma in linguaggio C che, acquisita da tastiera una stringa, ne mostri la rappresentazione binaria e poi la delimiti con 01111110b.**

Layout:

OUTPUT

```
Inserire stringa: Che?

La stringa binaria:
01000011.01101000.01100101.00111111

La stringa binaria del pacchetto:
0111111001000011011010000110010100111110101
1111110
```

Nota: Siccome il codice Ascii del carattere ? vale 00111111b, è necessario il bit stuffing.

8 Scrivere un programma in linguaggio C che acquisisca da tastiera i byte di un pacchetto Ascii con terminatore 55h (85d).

Layout:

OUTPUT 1

```
Inserire byte n. 1: 67
Inserire byte n. 2: 23
Inserire byte n. 2: 105
Inserire byte n. 3: 97
Inserire byte n. 4: 111
Inserire byte n. 5: 0

Il pacchetto:
55h 43h 69h 61h 6fh 55h
```

Nota: Il secondo input non è accettabile perché 23 non è un codice Ascii alfanumerico. L'input termina quando si immette il valore 0.

OUTPUT 2

```
Inserire byte n. 1: 67
Inserire byte n. 2: 104
Inserire byte n. 3: 101
Inserire byte n. 4: 85
Inserire byte n. 5: 0

Il pacchetto:
55h 43h 68h 65h 55h 55h 55h
```

9 Scrivere un programma in linguaggio C che acquisisca da tastiera i byte in esadecimale di un pacchetto Ascii.

Layout:

OUTPUT

```
Inserire byte n. 1: 47
Inserire byte n. 2: 72
Inserire byte n. 3: 61
Inserire byte n. 4: 6e
Inserire byte n. 5: g1
Inserire byte n. 5: 20
Inserire byte n. 6: 54
Inserire byte n. 7: 6f
Inserire byte n. 8: 72
Inserire byte n. 9: 69
Inserire byte n. 10: 6e
Inserire byte n. 11: 6f
Inserire byte n. 12: 0

Il pacchetto:
47h, 72h, 61h, 6eh, 20h, 54h, 6fh, 72h, 69h,
6eh, 6fh
```

10 Scrivere un programma in linguaggio C che acquisisca da tastiera i byte in esadecimale di un pacchetto Ascii e lo delimiti con il byte 40h.

11 Scrivere un programma in linguaggio C che, acquisiti da tastiera 4 numeri interi senza segno su 16 bit, crei un pacchetto L2 Datalink di tipo SLIP.

Layout:

OUTPUT

```
Inserire numero 1: 9127
Inserire numero 2: 5083
Inserire numero 3: 4288
Inserire numero 4: 49153

Il pacchetto SLIP:
c0h, 23h, a7h, 13h, dbh, ddh, 10h, dbh,
dch, dbh, dch, 01h, c0h
```

12 Scrivere un programma in linguaggio C che, acquisite da tastiera due stringhe Ascii binarie (soli caratteri '0' e '1'), ne stampi la distanza di Hamming.

Layout:

OUTPUT

```
Inserire stringa binaria n.1: 10101111
Inserire stringa binaria n.2: 1000111
Inserire stringa binaria n.2: 10001110

Distanza di Hamming: 2
```

13 Scrivere un programma in linguaggio C che, estratti casualmente due byte, ne stampi la distanza di Hamming.

Layout:

OUTPUT

```
Primo byte: 175
Secondo byte: 142

Distanza di Hamming: 2
```

14 Scrivere un programma in linguaggio C che, presi in input due byte in esadecimale, ne stampi la distanza di Hamming.

15 Scrivere un programma in linguaggio C che, estratti casualmente tre byte, ne stampi la distanza minima di Hamming.

Layout:

OUTPUT

```
Primo byte: 175
Secondo byte: 142
Terzo byte: 128

Le 3 distanze di Hamming: 2, 3, 5
La distanza di Hamming minima: 2
```

16 Scrivere un programma in linguaggio C che, acquisita una stringa da tastiera, ne calcoli la checksum troncata al byte.

Layout:

OUTPUT

```
Inserire una stringa: Potere assoluto

La checksum di:
50h, 6fh, 74h, 65h, 72h, 65h, 20h, 61h,
73h, 73h, 6fh, 6ch, 75h, 74h, 6fh
vale 09h
```

17 Scrivere un programma in linguaggio C che, acquisiti 3 ottetti binari da tastiera, ne calcoli la checksum troncata al byte.

18 Scrivere un programma in linguaggio C che, acquisiti da tastiera alcuni byte in formato esadecimale, ne calcoli la checksum troncata al byte. Si termina l'input con il carattere zero.

19 Scrivere un programma in linguaggio C che, acquisita una stringa da tastiera, ne calcoli la checksum troncata alla word.

Layout:

OUTPUT

```
Inserire una stringa: Potere assoluto

La checksum di:
50h, 6fh, 74h, 65h, 72h, 65h, 20h, 61h,
73h, 73h, 6fh, 6ch, 75h, 74h, 6fh

vale 0609h
```

20 Scrivere un programma in linguaggio C che, acquisiti 5 ottetti binari da tastiera, ne calcoli la checksum troncata alla word.

21 Scrivere un programma in linguaggio C che, acquisita una stringa da tastiera, ne calcoli la internet checksum.

Layout:

OUTPUT

```
Inserire una stringa: Potere assoluto

La internet checksum di:
50h, 6fh, 74h, 65h, 72h, 65h, 20h, 61h,
73h, 73h, 6fh, 6ch, 75h, 74h, 6fh

vale f9f6h
```

22 Scrivere un programma in linguaggio C che dato un byte che rappresenta il campo control di un pacchetto HDLC, se di tipo I-Frame, lo decodifichi.

Layout:

OUTPUT

```
Inserire il campo control: 193
Inserire il campo control: 97

In binario: 01100001
I-Frame
ACK 4
ACK piggybacked 1
```

Nota: Il primo input non è accettabile perché 193 = 11000001b, cioè non è un I-Frame.

23 Scrivere un programma in linguaggio C che dato un byte che rappresenta il campo control di un pacchetto HDLC, se di tipo S-Frame, lo decodifichi.

Layout:

OUTPUT

```
Inserire il campo control: 193
Inserire il campo control: 167

In binario: 10100111
S-Frame
NACK
per trama 7
```

Nota: Il primo input non è accettabile perché 193 = 11000001b, cioè non è un S-Frame.

24 Scrivere un programma in linguaggio C che dato un byte che rappresenta il campo control di un pacchetto HDLC, se di tipo U-Frame, lo decodifichi.

Layout:

OUTPUT

```
Inserire il campo control: 193

In binario: 11000001
U-Frame
ACK
Modifier=1
```

25 Scrivere un programma in linguaggio C che estratto casualmente un byte che rappresenta il campo control di un pacchetto HDLC, lo decodifichi.

Layout:

OUTPUT

```
Campo control: 107

In binario: 01101011
I-Frame
ACK 5
ACK piggybacked 3
```

Appendice

1 Installazione e configurazione DOSBox per sistemi a 64bit

Con le versioni a 64 bit dei sistemi operativi Windows (XP64, Vista, Windows 7) non viene più mantenuta la compatibilità con il codice x86-16 bit. Alcuni programmi, come per esempio Tasm e Debug non possono essere avviati.

Una soluzione è installare su questi sistemi un emulatore di MSDOS a 16bit come **DOSBox**, applicazione gratuita e dotata di licenza GNU.

Lo stesso emulatore può essere installato su diversi sistemi operativi, come GNU/Linux e Mac OS.

Installazione DOSBox per Windows x86-64

Il file di installazione si recupera dal sito *www.dosbox.com*, ed è un programma eseguibile autoinstallante che non richiede scelte durante l'installazione.

L'effetto è la creazione di una cartella *%Program Files(x86)%\DOSBox-0.74*, con relativo collegamento sul desktop e file di configurazione *dosbox-0.74.conf* nella cartella del profilo utente in:

(per Windows XP)
%USERPROFILE%\Local Settings\Application Data\DOSBox\dosbox-{versione}.conf

(per Windows Vista e Windows 7)
{system drive}:\Users\{username}\AppData\Local\DOSBox\dosbox-{versione}.conf

Configurazione DOSBox per Windows x86-64

Una configurazione che consenta di avviare i programmi Debug.exe e TD.EXE (ma anche l'insieme di programmi per l'ambiente assembly Borland Tasm), può essere organizzata utilizzando proficuamente il file di configurazione in dotazione con il programma.

1) Creare una directory nella radice del sistema operativo a 64 bit (esempio, *c:\dosbox*) e copiarvi il file di configurazione *dosbox-0.74.conf*.
2) Creare due cartelle: *c:\dosbox\assembly* e *c:\dosbox\assembly\programs*.
3) Nella cartella *c:\dosbox\assembly\programs* copiare i files x-86 a 16 bit che si intendono usare(NB).

NB Per l'ambiente **Debug.exe**, copiare nella cartella i seguenti files:

- Debug.exe
- TD.EXE

Per l'insieme di programmi per l'ambiente **assembly Borland Tasm**, copiare nella cartella i seguenti files:

- DPMI16BI.OVL
- DPMILOAD.EXE
- DPMIMEM.DLL
- TASM.EXE
- TD.EXE
- TLINK.EXE

Ora si possono scrivere i sorgenti Assembly con un editor di testo come Notepad++, salvandoli nella cartella *c:\dosbox\assembly* e avviare i programmi per compilare e fare il Debug all'interno della finestra di DOSBox.

Per i files sorgenti da utilizzare con **Debug.exe** ricordare di impostare Notepad++ in modo che usi come carattere di fine linea il solo CR (*Carriage Return*). Dal menu di Notepad++, *Modifica-Converti carattere di fine linea- Formato MAC*.

4) Aprire con un editor di testo il file di configurazione *c:\dosbox\dosbox-0.74.conf*, portarsi in fondo al file e aggiungere queste righe (al termine della sezione [autoexec]):

```
mount C C:\DOSBOX\ASSEMBLY
c:
path=%path%;c:\programs
```

5) Infine modificare la proprietà del collegamento di DOSBox sul desktop.

Alla voce *Destinazione*, sostituire il valore presente con:

"C:\Program Files (x86)\DOSBox-0.74\DOSBox.exe" –conf c:\dosbox\dosbox-0.74.conf

Norton Guide

NORTON GUIDE PER MSDOS

Un classico supporto per la programmazione Assembly x-86 sono le cosiddette **Norton Guide** (NG).

Si tratta di un programma residente che consente la consultazione dell'Instruction Set x-86, delle API di MSDOS (Interruzioni software) e di altri piccoli tool d'utilità per la programmazione.

Anche in questo caso il pacchetto, costituito dai soli file *ng.exe*, *ng.ini* e *asm.ng*, non è avviabile su sistemi a 64 bit.

Per poterlo utilizzare comunque, è sufficiente copiare i tre files del pacchetto nella cartella indicata al passo 3) del procedimento descritto per DOSBox.

EMU8086

Didatticamente interessante è anche il programma **EMU8086** che può essere recuperato da *www.emu8086.com*.

L'installazione è praticamente silenziosa e del tutto priva di dipendenze.

Si tratta di un ambiente sostanzialmente completo per la programmazione Assembly x-86 a 16 bit, comprendente un editor con sintassi colorata, un assemblatore, un linker e un emulatore 8086 dotato anche di debugger.

Si presenta con interfaccia grafica, quindi è abbastanza intuitivo, pur generando files binari sia COM che EXE per x-86.

Accetta codice quasi del tutto compatibile con la sintassi Tasm/Masm (eccetto che per qualche parola chiave) e consente l'esecuzione passo passo combinata tra codice sorgente e codice macchina, molto utile per chiarire l'uso delle istruzioni meno intuitive.

Interessante la possibilità di scrivere codice di input/output sia su sistemi Win32 che su sistemi a 64bit, superando la barriera della modalità kernel e quindi offrendo la possibilità di scrivere codice per semplici dispositivi di I/O.

In dotazione, vengono proposti numerosi esempi di codice sorgente a volte anche interessanti.

2 Tabella codici Ascii

DEC	HEX	char	Descrizione
0	00	**NULL**	(Null character)
1	01	**SOH**	(Start of Header)
2	02	**STX**	(Start of Text)
3	03	**ETX**	(End of Text)
4	04	**EOT**	(End of Transmission)
5	05	**ENQ**	(Enquiry)
6	06	**ACK**	(Acknowledgement)
7	07	**BEL**	(Bell)
8	08	**BS**	(Backspace)
9	09	**HT**	(Horizontal Tab)
10	0A	**LF**	(Line feed)
11	0B	**VT**	(Vertical Tab)
12	0C	**FF**	(Form feed)
13	0D	**CR**	(Carriage return)
14	0E	**SO**	(Shift Out)
15	0F	**SI**	(Shift In)
16	10	**DLE**	(Data link escape)
17	11	**DC1**	(Device control 1)
18	12	**DC2**	(Device control 2)
19	13	**DC3**	(Device control 3)
20	14	**DC4**	(Device control 4)
21	15	**NAK**	(Negative acknowledgement)
22	16	**SYN**	(Synchronous idle)
23	17	**ETB**	(End of transmission block)
24	18	**CAN**	(Cancel)
25	19	**EM**	(End of medium)
26	1A	**SUB**	(Substitute)
27	1B	**ESC**	(Escape)
28	1C	**FS**	(File separator)
29	1D	**GS**	(Group separator)
30	1E	**RS**	(Record separator)
31	1F	**US**	(Unit separator)
32	20		(space)
33	21	**!**	(exclamation mark)
34	22	**“**	(Quotation mark)
35	23	**#**	(Number sign)
36	24	**$**	(Dollar sign)
37	25	**%**	(Percent sign)
38	26	**&**	(Ampersand)
39	27	**‘**	(Apostrophe)
40	28	**(**	(round brackets or parentheses)
41	29	**)**	(round brackets or parentheses)
42	2A	*	(Asterisk)
43	2B	+	(Plus sign)
44	2C	,	(Comma)
45	2D	-	(Hyphen)
46	2E	.	(Full stop , dot)
47	2F	/	(Slash)
48	30	**0**	(number zero)
49	31	**1**	(number one)
50	32	**2**	(number two)
51	33	**3**	(number three)
52	34	**4**	(number four)
53	35	**5**	(number five)
54	36	**6**	(number six)
55	37	**7**	(number seven)
56	38	**8**	(number eight)
57	39	**9**	(number nine)
58	3A	:	(Colon)
59	3B	;	(Semicolon)
60	3C	<	(Less-than sign)
61	3D	=	(Equals sign)
62	3E	>	(Greater-than sign ; Inequality)
63	3F	?	(Question mark)

DEC	HEX	char	Descrizione
64	40	@	(At sign)
65	41	**A**	(Capital A)
66	42	**B**	(Capital B)
67	43	**C**	(Capital C)
68	44	**D**	(Capital D)
69	45	**E**	(Capital E)
70	46	**F**	(Capital F)
71	47	**G**	(Capital G)
72	48	**H**	(Capital H)
73	49	**I**	(Capital I)
74	4A	**J**	(Capital J)
75	4B	**K**	(Capital K)
76	4C	**L**	(Capital L)
77	4D	**M**	(Capital M)
78	4E	**N**	(Capital N)
79	4F	**O**	(Capital O)
80	50	**P**	(Capital P)
81	51	**Q**	(Capital Q)
82	52	**R**	(Capital R)
83	53	**S**	(Capital S)
84	54	**T**	(Capital T)
85	55	**U**	(Capital U)
86	56	**V**	(Capital V)
87	57	**W**	(Capital W)
88	58	**X**	(Capital X)
89	59	**Y**	(Capital Y)
90	5A	**Z**	(Capital Z)
91	5B	**[**	(square brackets or box brackets)
92	5C	\	(Backslash)
93	5D	**]**	(square brackets or box brackets)
94	5E	^	(Caret or circumflex accent)
95	5F	_	(underscore)
96	60	`	(Grave accent)
97	61	**a**	(Lowercase a)
98	62	**b**	(Lowercase b)
99	63	**c**	(Lowercase c)
100	64	**d**	(Lowercase d)
101	65	**e**	(Lowercase e)
102	66	**f**	(Lowercase f)
103	67	**g**	(Lowercase g)
104	68	**h**	(Lowercase h)
105	69	**i**	(Lowercase i)
106	6A	**j**	(Lowercase j)
107	6B	**k**	(Lowercase k)
108	6C	**l**	(Lowercase l)
109	6D	**m**	(Lowercase m)
110	6E	**n**	(Lowercase n)
111	6F	**o**	(Lowercase o)
112	70	**p**	(Lowercase p)
113	71	**q**	(Lowercase q)
114	72	**r**	(Lowercase r)
115	73	**s**	(Lowercase s)
116	74	**t**	(Lowercase t)
117	75	**u**	(Lowercase u)
118	76	**v**	(Lowercase v)
119	77	**w**	(Lowercase w)
120	78	**x**	(Lowercase x)
121	79	**y**	(Lowercase y)
122	7A	**z**	(Lowercase z)
123	7B	**{**	(curly brackets or braces)
124	7C	**|**	(vertical-bar)
125	7D	**}**	(curly brackets or braces)
126	7E	~	(Tilde ; swung dash)
127	7F	**DEL**	(Delete)

DEC	HEX	char	Descrizione
128	80	Ç	(Majuscule C-cedilla)
129	81	ü	(“u-umlaut”)
130	82	é	(“e-acute”)
131	83	â	(“a-circumflex”)
132	84	ä	(“a-umlaut”)
133	85	à	(letter “a” with grave accent)
134	86	å	(letter “a” with a ring)
135	87	ç	(minuscule c-cedilla)
136	88	ê	(“e-circumflex”)
137	89	ë	(“e-umlaut”)
138	8A	è	(letter “e” with grave accent)
139	8B	ï	(“i-umlaut”)
140	8C	î	(“i-circumflex”)
141	8D	ì	(letter “i” with grave accent)
142	8E	Ä	(“A-umlaut”)
143	8F	Å	(letter “A” with a ring)
144	90	É	(“E-acute”)
145	91	æ	(Latin diphthong “ae”)
146	92	Æ	(Latin diphthong “AE”)
147	93	ô	(“o-circumflex”)
148	94	ö	(“o-umlaut”)
149	95	ò	(letter “o” with grave accent)
150	96	û	(“u-circumflex”)
151	97	ù	(letter “u” with grave accent)
152	98	ÿ	(letter “y” with diaeresis)
153	99	Ö	(“O-umlaut”)
154	9A	Ü	(“U-umlaut”)
155	9B	ø	(slashed zero)
156	9C	£	(Pound sign)
157	9D	Ø	(slashed zero)
158	9E	×	(multiplication sign)
159	9F	ƒ	(function sign)
160	A0	á	(“a-acute”)
161	A1	í	(“i-acute”)
162	A2	ó	(“o-acute”)
163	A3	ú	(“u-acute”)
164	A4	ñ	(letter “n” with tilde)
165	A5	Ñ	(letter “N” with tilde)
166	A6	ª	(feminine ordinal indicator)
167	A7	º	(masculine ordinal indicator)
168	A8	¿	(Inverted question marks)
169	A9	®	(Registered trademark)
170	AA	¬	(Logical negation symbol)
171	AB	½	(One half)
172	AC	¼	(Quarter or one fourth)
173	AD	¡	(Inverted exclamation marks)
174	AE	«	(Guillemets)
175	AF	»	(Guillemets)
176	B0	░	(Box 0)
177	B1	▒	(Box 1)
178	B2	▓	(Box 2)
179	B3	│	(Box drawing character)
180	B4	┤	(Box drawing character)
181	B5	Á	(“A-acute”)
182	B6	Â	(“A-circumflex”)
183	B7	À	(letter “A” with grave accent)
184	B8	©	(Copyright symbol)
185	B9	╣	(Box drawing character)
186	BA	║	(Box drawing character)
187	BB	╗	(Box drawing character)
188	BC	╝	(Box drawing character)
189	BD	¢	(Cent symbol)
190	BE	¥	(YEN and YUAN sign)
191	BF	┐	(Box drawing character)

DEC	HEX	char	Descrizione
192	C0	└	(Box drawing character)
193	C1	┴	(Box drawing character)
194	C2	┬	(Box drawing character)
195	C3	├	(Box drawing character)
196	C4	─	(Box drawing character)
197	C5	┼	(Box drawing character)
198	C6	ã	(“a-tilde”)
199	C7	Ã	(“A-tilde”)
200	C8	╚	(Box drawing character)
201	C9	╔	(Box drawing character)
202	CA	╩	(Box drawing character)
203	CB	╦	(Box drawing character)
204	CC	╠	(Box drawing character)
205	CD	═	(Box drawing character)
206	CE	╬	(Box drawing character)
207	CF	¤	(generic currency sign)
208	D0	ð	(lowercase “eth”)
209	D1	Ð	(Capital letter “Eth”)
210	D2	Ê	(“E-circumflex”)
211	D3	Ë	(“E-umlaut”)
212	D4	È	(letter “E” with grave accent)
213	D5	ı	(lowercase dot less i)
214	D6	Í	(“I-acute”)
215	D7	Î	(“I-circumflex”)
216	D8	Ï	(“I-umlaut”)
217	D9	┘	(Box drawing character)
218	DA	┌	(Box drawing character)
219	DB	█	(Block)
220	DC	▄	(Semiblock down)
221	DD	¦	(vertical broken bar)
222	DE	Ì	(letter “I” with grave accent)
223	DF	▀	(Semiblock up)
224	E0	Ó	(“O-acute”)
225	E1	ß	(“scharfes S”)
226	E2	Ô	(“O-circumflex”)
227	E3	Ò	(letter “O” with grave accent)
228	E4	õ	(“o-tilde”)
229	E5	Õ	(“O-tilde”)
230	E6	µ	(micron)
231	E7	þ	(“Thorn”)
232	E8	Þ	(“Thorn”)
233	E9	Ú	(“U-acute”)
234	EA	Û	(“U-circumflex”)
235	EB	Ù	(letter “U” with grave accent)
236	EC	ý	(letter “y” with acute accent)
237	ED	Ý	(Capital letter “Y” with acute accent)
238	EE	¯	(macron symbol)
239	EF	´	(Acute accent)
240	F0	¬	(Hyphen)
241	F1	±	(Plus-minus sign)
242	F2	‗	(underline)
243	F3	¾	(three quarters)
244	F4	¶	(pilcrow)
245	F5	§	(Section sign)
246	F6	÷	(Obelus)
247	F7	¸	(cedilla)
248	F8	°	(degree symbol)
249	F9	¨	(Diaeresis)
250	FA	•	(Interpunct)
251	FB	¹	(superscript one)
252	FC	³	(cube)
253	FD	²	(square)
254	FE	■	(black square)
255	FF	nbsp	(no-break space)

3 Esercizi ISA virtuale

ESEMPIO

Uso di JG

Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo	
50	END		

(ove l'istruzione **JG** *salta a* indirizzo *se il valore nel registro A è maggiore del valore nel registro B), scrivere un programma che acquisito in input un tasto numerico, stampi >5 se il tasto premuto è 6,7,8 o 9.*

Se si suppone che inizialmente il Program Counter valga 16, il programma in memoria diventa così (in evidenza **JG**):

Indirizzi	Memoria							
0								
8								
16	21	0	0	1	'5'	10	0	11
24	1	**32**	30	50			0	2
32	'>'	10	2	20	10	1	20	50
40								

Commento:

```
   21  0   Effettua l'input del tasto numerico e ne deposita il codice Ascii nella cella di indirizzo 0.
0  1  '5'  Colloca in memoria all'indirizzo 1 il codice Ascii del carattere 5.
   10  0   Carica dalla cella di indirizzo 0 il valore (dell'input) nel registro A.
   11  1   Carica dalla cella di indirizzo 1 il valore (Ascii di 5) nel registro B.
   32 30   Effettua il confronto tra il valore nel registro A e il valore nel registro B; se registro A maggiore di
           registro B, allora il Program Counter varrà 30 (salto all'indirizzo 30).
      50   Termine del programma.

0  2  '>'  Colloca in memoria all'indirizzo 2 il codice Ascii del simbolo >.
   10  2   Carica nel registro A il valore della cella di indirizzo 2 (Ascii di >).
      20   Stampa sullo schermo il simbolo corrispondente al codice Ascii presente nel registro A (>).
   10  1   Carica nel registro A il valore della cella di indirizzo 1 (Ascii di 5).
      20   Stampa sullo schermo il simbolo corrispondente al codice Ascii presente nel registro A (5).
      50   Termine del programma.
```

ESEMPIO

Uso di LOOP

Data la seguente ISA:

op. cod.	Mnemonico	Operando 1	Operando 2
0	STORE	indirizzo	valore
10	LOADA	indirizzo	
11	LOADB	indirizzo	
12	LOADC	indirizzo	
15	MOV	registroD	registroS
20	OUT		
21	INPUT	indirizzo	
30	JMP	indirizzo	
31	JE	indirizzo	
32	JG	indirizzo	
33	LOOP	indirizzo	
40	INC	registro	
50	END		

Considerare che l'istruzione **LOADC** *si comporta come le analoghe LOADA e LOADB ma sul registro C; considerare che l'istruzione* **MOV** *sposta il contenuto della cella che si trova nell'indirizzo contenuto in* registroS*, nel* registroD *(regA=0; regB=1; regC=2); considerare che l'istruzione* **LOOP** *prima diminuisce di una unità il registro C, poi salta all'indirizzo specificato solo se il registro C è diverso da zero; considerare che l'istruzione* **INC** *aumenta di una unità il valore contenuto nel registro specificato.*

Scrivere un programma che stampa sullo schermo ITIS.

Se si suppone che inizialmente il Program Counter valga 16, il programma in memoria diventa così (in evidenza **LOOP**):

Indirizzi	Memoria							
0								
8								
16	0	0	'I'	0	1	'T'	0	2
24	'I'	0	3	'S'	0	6	7	12
32	6	0	8	0	11	8	15	0
40	1	20	40	1	**33**	38	50	
48								

Commento:

```
0   0  'I'   Colloca in memoria all'indirizzo 0 il codice Ascii del carattere I.
0   1  'T'   Colloca in memoria all'indirizzo 0 il codice Ascii del carattere T.
0   2  'I'   Colloca in memoria all'indirizzo 0 il codice Ascii del carattere I.
0   3  'S'   Colloca in memoria all'indirizzo 0 il codice Ascii del carattere S.
0   6   7    Colloca in memoria all'indirizzo 6 il numero 4 (sarà il n. di ripetizioni del ciclo).
   12   6    Carica dalla cella di indirizzo 6 il valore (4) nel registro C.
0   8   0    Colloca in memoria all'indirizzo 8 il numero 0.
   11   8    Carica dalla cella di indirizzo 8 il valore (0) nel registro B.
15  0   1    Colloca nel registro A (0) il valore contenuto all'indirizzo specificato nel registro B (1): nel registro B
             c'è 0, all'indirizzo 0 c'è il codice Ascii di I. Allora nel registro A ora c'è il codice Ascii di I.
       20    Stampa sullo schermo il simbolo corrispondente al codice Ascii presente nel registro A (I).
   40  1    Incrementa di una unità il registro B (1). Ora nel registro B c'è 1.
   33 38    Diminuisce di una unità il registro C, ora in C c'è 3. Siccome 3 è diverso da zero, il Program Counter
             vale 38 (salto all'indirizzo 38).
15  0   1    Colloca nel registro A (0) il valore contenuto all'indirizzo specificato nel registro B (1): nel registro B
             ora c'è 1, all'indirizzo 1 c'è il codice Ascii di T. Allora nel registro A ora c'è il codice Ascii di T.
       20    Stampa sullo schermo il simbolo corrispondente al codice Ascii presente nel registro A (T).
   40  1    Incrementa di una unità il registro B (1). Ora nel registro B c'è 2.
   33 38    Diminuisce di una unità il registro C, ora in C c'è 2. Siccome 2 è diverso da zero, il Program Counter
             vale 38 (salto all'indirizzo 38).
(ecc.)
```

Indice analitico

Paolo Ollari

Corso di sistemi e reti

per Informatica

Architetture e network

Il corso guida gli studenti a diventare competenti nella gestione dei sistemi di elaborazione dati e reti, e nella scelta degli strumenti e dei dispositivi da utilizzare sulla base delle loro caratteristiche funzionali.

La prima sezione, dedicata ai sistemi, illustra il **linguaggio Assembly** come esempio di accesso all'architettura, facendo uso dell'algoritmica indispensabile.
La seconda sezione, dedicata alle reti, parte dallo stato fisico sino ad arrivare ai problemi tipici del livello di collegamento dati.

Nel libro

- Numerosi **esempi applicativi**, accompagnati dai relativi **programmi**.
- Al termine di ogni capitolo esercizi per la **verifica orale** (divisi in due livelli: *requisiti fondamentali* e *requisiti avanzati*), esercizi per la **verifica scritta** ed esercizi per la **verifica di laboratorio**.

NOVITÀ