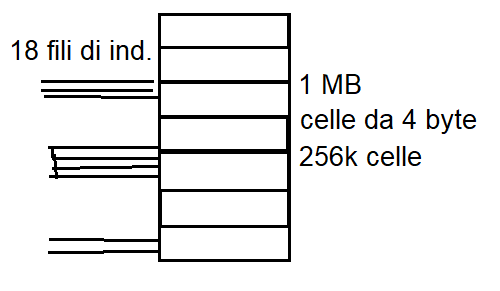
Argomenti principali trattati nel primo semestre:

* Codici (per la rappresentazione di dati): A lunghezza fissa, A lunghezza variabile, A espansione. È possibile costruire un codice a lunghezza variabile utilizzando un terminatore (in C \0), ma anche indicando esplicitamente la lunghezza della sequenza mediante un campo a parte seguito dai byte che costituiscono la sequenza. I codici a espansione hanno più lunghezze fisse disponibili per la rappresentazione dei dati, la lunghezza per la sequenza è determinata da una sequenza di espansione che si trova da qualche parte nel codice. Questi formati di codici a lunghezza non fissa trovano il loro maggior utilizzo in campo di compressione di dati, ma anche nella codifica delle operazioni da far eseguire al processore (ad esempio nel PDP11 le istruzioni utilizzano un codice a espansione).
* Circuiti: Combinatori, Sequenziali. I circuiti combinatori sono i più semplici, essi dipendono dal valore in ingresso e il valore in uscita è sempre una funzione del valore in input. I circuiti sequenziali si suddividono in circuiti sincroni e asincroni. Nei circuiti sequenziali l’output non dipende solo dai valori in ingresso in un dato momento ma anche dalla sequenza di input ricevuti in precedenza. Questo tipo di circuiti possono essere utilizzati per costruire memorie. I circuiti sincroni si caratterizzano per essere dotati di un clock, che fa sì che le uscite non cambino al variare dei valori in ingresso, ma possono farlo solo al variare del valore del clock (nei circuiti asincroni questo non vale). Dal punto di vista pratico i circuiti sincroni sono i più facili da realizzare, poiché si possono realizzare seguendo le tavole di verità, non tenendo conto dei tempi di propagazione (e di assestamento dei valori, in caso si usi un ciclo di clock abbastanza lento), questo non si può fare per i circuiti asincroni (il che ne complica la progettazione).
* Memorie: RAM, Associativa. La memoria RAM prevede che ci siano un filo per bit che permetta sia la lettura che la scrittura di un valore. Tali fili sono connessi a tot. Celle di memora e ne permettono l’accesso a una attraverso l’indirizzamento ottenuto dai fili di indirizzamento. Il tipo di operazione (lettura/scrittura/nulla) viene determinato dai 2 fili Chip-Select e R/Wnegato. La memoria Associativa è più complicata rispetto alla RAM, l’idea è di avere una cella di memoria divisa in due parti, una che viene chiamata Tag, e l’altra che viene chiamata Data. La peculiarità della memoria Associativa è che non richiede all’utilizzatore di conoscere l’indirizzo del valore ricercato (perché tale valore è fornito dal contento del campo Tag), che può essere salvato nel campo Data: il vantaggio è che permette di raggiungere qualsiasi valore sulla base del contenuto del campo Tag. In una normale RAM bisognerebbe fare una scansione dell’intera RAM per trovare un determinato valore, nella memoria associativa ciò avviene in un solo ciclo di clock perché la componente hardware dei tag permette di controllare in una sola volta tutte le celle.\*
* Processori: Il bus dati collega RAM, sistemi di input/output e il Processore. Il processore legge il contenuto dei dispositivi di I/O attraverso la mappatura dei valori in esso contenuti in memoria RAM. Il modo con cui il processore differenzia il codice da eseguire dai dati è quello di eseguire le fasi di fetch, decodifica ed esecuzione, facendo riferimento a un registro chiamato Contatore di Programma (PC). Tale registro trova la successiva istruzione da copiare nel registro delle istruzioni e si auto-incrementa automaticamente alla fine della fase di fetch.
* Meccanismo della memoria virtuale: Il processore genera indirizzi virtuali, che vengono tradotti in indirizzi fisici dall’MMU. La memoria virtuale può seguire il modello di segmentazione, di paginazione o una combinazione delle due. La segmentazione vede un uso razionalizzato della memoria RAM, che viene separata in segmenti a seconda del loro scopo (Codice, Dati Statici, Stack, Heap) la cui dimensione varierà a seconda delle necessità del programma. Il meccanismo della paginazione viene introdotto per evitare occupazioni Mal frammentate della memoria RAM. Essa consiste nel suddividere la memoria RAM in pagine, la cui dimensione varia in potenze di 2. Ciò ci permette di dividere il numero di indirizzo in due parti: Il numero di Pagina e l’Offset. Utilizzando paginazione e segmenti contemporaneamente si fa si che ogni segmento abbia per dimensione un multiplo della dimensione di una pagina. Ecco, quindi che in realtà il numero di indirizzo contiene una terza parte: i bit di segmento (che indicano lo scopo del segmento). Il numero di segmento nella traduzione viene “tagliato” (dopo che l’MMU ha controllato che l’accesso al segmento avvenga in un momento in cui è lecito), il numero di Offset viene mantenuto così com’è e il numero di pagina viene cambiato dall’MMU a quello dell’indirizzo fisico. La traduzione degli indirizzi potrebbe essere molto pesante da realizzare: perché ci si potrebbe aspettare di dover accedere alla RAM per poter ottenere l’indirizzo delle tabelle di traduzione, accedere a tali tabelle e poi effettuare la traduzione in indirizzo fisico attraverso una ricerca sequenziale. Invece, oltre all’MMU viene inserito un dispositivo, il TLB, una piccola memoria associativa fatta apposta per velocizzare la traduzione degli indirizzi: usa infatti i bit corrispondenti al numero di segmento e il numero di pagina come campo tag (ma non i bit di offset perché viene riportato tale e quale) per cercare il numero di pagina fisico nella tabella di traduzione. La prima volta che un processore genera un indirizzo virtuale nel TLB non troverà nulla (perché ancora vuoto), ma dopo la prima traduzione la coppia Tag, Data viene salvata nella TLB e quindi dal secondo accesso in poi troverà lì il valore tradotto e potrà eseguire la traduzione in un solo ciclo di clock. Il TLB funziona quindi come una Cache. Poiché tale memoria associativa è molto costosa si cerca di ridurne la dimensione, quindi non conterrà tutte le informazioni presenti nelle tabelle di impaginazione. Quando si riempie e viene utilizzata una nuova pagina si effettua la sovrascrittura di un dato salvato precedentemente, questo perché secondo uno studio del comportamento dei programmi esistono due proprietà di “località”; nel tempo e nello spazio. Infatti, dopo aver generato un indirizzo I il processore molto probabilmente nel prossimo futuro genererà di nuovo (e più volte) l’indirizzo I: ad esempio un ciclo for in cui viene aggiornato 100 volte una variabile i richiede che la cella di memoria che contiene il valore della variabile i verrà acceduta 100 volte in poco tempo. La località è anche nello spazio perché se a un certo punto viene generato l’indirizzo I è lecito aspettarsi che di lì a poco sia generato anche l’indirizzo i+1 o i-1: ad esempio se, sempre in un ciclo for, azzero tutti gli elementi di un array uno ad uno , ad ogni ciclo utilizzo celle di memoria adiacenti a quelle usate precedentemente. Poiché tutti questi indirizzi sono contenuti in pagine di memoria è lecito ritenere che dopo l’accesso a una cella in una pagina avverranno altri accessi o nella stessa cella o a celle adiacenti nella stessa pagina: questo è il motivo per cui ha senso inserire nella TLB gli ultimi valori che sono stati generati come indirizzi. Questo sistema segue l’algoritmo LRU (Least recently used), che sostiene di sostituire il valore utilizzato meno recentemente (perché probabilmente non gode più dei due principi di località) -> analogamente avviene con il caricamento in RAM di pagine dalla memoria a disco.

\*Approfondimento sulle memorie associative (per completare il programma).

\*Questa è una memoria RAM classica: immaginiamo di modificare un valore e voler poi tornare a modificare quel valore, ma non disponiamo più dell’indirizzo della cella: sarebbe necessario effettuare una ricerca sequenziale su tutte le celle per poter trovare il valore.

Nelle Memorie sequenziale anziché usare i fili di indirizzamento abbiamo delle celle Tag associate a ogni cella Data, all’interno delle quali sono inserite delle chiavi di ricerca. Per i campi delle tag sarebbe quindi necessario usare almeno 18 bit (ma anche di più, come 32) poiché i campi tag devono ovviamente essere tutti diversi (oltretutto i contenuti di tali tag non sono rilevanti per chi usa il dispositivo). Al momento dell’accensione la memoria associativa è vuota (non ci sono tag). Quando si vogliono aggiungere degli elementi bisogna passare al dispositivo due valori: Tag e Data. Il dispositivo prende una qualsiasi cella libera e va a inserire questi valori (e il procedimento si ripete per ogni nuova coppia di valori). Una volta esaurita la memoria a disposizione non ci sono più nuovi tag e quindi si solleverà una situazione di errore. Dopo aver inserito tanti (500) tag diversi si potrebbe voler cambiare un elemento: per fare ciò basta mandare in scrittura una coppia che ha un tag già inserito in precedenza (quello dell’elemento che si vuole cambiare) e un valore x che contiene il nuovo valore da inserire in Data. Per le operazioni di lettura basta inviare l’istruzione “Read(Ti)”, viene fatta la ricerca associativa e se viene trovato il tag Ti (altrimenti restituisce messaggio di errore) viene restituito il valore contenuto nel Data corrispondente. Le memorie associative fan sì che si possano creare associazioni tra stringhe di bit più lunghe e valori. È ovvio che quindi serva anche un bit di rappresentazione che dica per ogni cella se questa è libera o occupata (in tal modo la ricerca associativa viene effettuata solo sulle celle occupate).

Gli utilizzi principali delle memorie associative sono le memorie Cache e un componente del meccanismo di traduzione degli indirizzi dell’MMU.

Questo tipo di dispositivo è costoso da realizzare, perché ci deve essere un circuito comparatore (di uguaglianza) associato a ciascuno dei campi tag. Essi sono molto utili perché sono veloci nella ricerca indipendentemente dalla dimensione della memoria (infatti essa è sempre costante) tuttavia poiché è notevole anche il costo di realizzazione, questo tipo di memorie non possono essere particolarmente grandi. L’operazione di lettura ha la stessa complessità rispetto all’operazione di scrittura.\*