SAPIENZA
UNIVERSITÀ DI ROMA

Università "Sapienza" di Roma
Facoltà di Informatica

ARCHITETTURA DEGLI ELABORATORI

Author: Alessia Cassetta

Giugno 2024

Indice

Introduzione

Il presente documento è una raccolta di appunti riguardanti il corso di Architettura degli Elaboratori, tenuto dal Prof. Franco Liberati presso l'Università degli Studi di Roma "La Sapienza". Pertanto, il testo è stato redatto in modo da essere il più chiaro e completo possibile, in modo da poter essere utilizzato come supporto allo studio per l'esame.

Il corso di Architettura degli Elaboratori è diviso in due parti: una parte teorica e una parte pratica. La parte teorica riguarda lo studio delle architetture degli elaboratori, ovvero la struttura interna dei computer e il funzionamento dei vari componenti che li compongono. La parte pratica, invece, riguarda l'utilizzo di un linguaggio di programmazione assembly per la scrittura di programmi che sfruttano le funzionalità dell'architettura degli elaboratori.

Questo testo non sarà completo e non sostituirà mai il materiale didattico fornito dal docente, in quanto mancano le esercitazioni svolte in classe. In caso di errori o imprecisioni, si prega di contattare l'autore all'indirizzo email: **c.alessia04@gmail.com**.

Link del sito del Prof. Liberati: Link

1 Storia

1.1 Calcolo manuale

Gli antropologi fanno risalire alla fibula di babbuino ritrovato in una grotta abitata durante il Paleolitico [2.5 milioni a.C. a 10000 a.C.] sui monti Lebombo (Sudafrica) il primo datario. La presenza delle 29 incisioni suggerisce che potrebbe essere stato usato come un contatore di fasi lunari.

Alfabeto numerico, insieme di simboli utili per rappresentare grandi quantità di beni, altrimenti non descrivibili. La **tavoletta di argilla** è il primo dispositivo utilizzato per rappresentare e archiviare delle quantità numeriche nonché per svolgere i calcoli in maniera manuale. Abaco primo dispositivo per rappresentare e manipolare operandi mediante un particolare sistema posizionale in cui dei sassolini (calcolus) indicavano le unità, le decine e le centinaia in accordo alla loro presenza su delle linee pre-segnate.

La macchina di **Antikythera** è il **più antico calcolatore meccanico conosciuto**. Si trattava di un sofisticato planetario, mosso da ruote dentate, che serviva per calcolare il sorgere del sole, le fasi lunari, i movimenti dei cinque pianeti allora conosciuti, gli equinozi, i mesi, i giorni della settimana e le date dei giochi olimpici. Nel 1202, con il Liber Abaci, scritto da Leonardo Fibonacci matematico pisano, si introduce ufficialmente in Europa il **sistema posizionale**.

Un primo sistema di calcolo (temporale) meccanico si riscontrò tra il 1230 ed il 1270 con l'orologio. Era un dispositivo costituito da un organo motore (es.: un peso o una molla); degli ingranaggi a ruote dentate che demoltiplicavano il moto, un elemento di distribuzione di un intervallo di tempo (lo scappamento) regolato da un componente con moto isocrono (es.: un pendolo) e un indicatore della misurazione il quadrante).

1.2 Il Calcolo Semi-AUtomatico

Nel 1623 il matematico tedesco Wilhelm Schickard progetta e realizza un prototipo di **prima macchina calcolatrice meccanica**, il Calculating Clock.

Nel 1642 il matematico e filosofo francese Blaise Pascal propose il **Pascaline**, una macchina automatica in grado di svolgere l'operazione di addizione e di sottrazione.

Nel 1672 il filosofo e matematico Gottfried Wilhelm Leibniz, famoso per lo studio aritmetico del sistema binario, perfezionò il Pascaline con il **Contatore a gradini** (calcolatrice a pignoni o Leibniz Machine). Si trattava di una macchina di calcolo manuale in grado di svolgere le quattro operazioni elementari (somma, sottrazione, moltiplicazione e divisione). Migliorato nel 1727 da Jacob Leupold e, più avanti, dal francese Thomas de Colmar con l'invenzione dell'aritmometro.

Nel 1816 Charles Babbage, richiamando le tecniche di semi automazione dei telai meccanici di Joseph Jacquard, progetta la **macchina differenziale**. La Difference Engine

era una calcolatrice, la cui meccanica era attivata da un motore a vapore. Oltre alle normali operazioni aritmetiche, era in grado di svolgere il calcolo polinomiale (per la risoluzione di funzioni logaritmiche, serie di Taylor).

1.3 Macchine Programmate

Il prototipo di una macchina programmata automatica deputata al calcolo fu proposto nel 1890 da Herman Hollerith per il censimento dei cittadini statunitensi: Hollerith Tabulator o Tabulator Machine.

L'invenzione faceva uso di schede cartacee perforabili (punch cards) su cui erano impresse, in posizioni prestabilite, le caratteristiche demografiche (stato civile, luogo di nascita, età, Stato di residenza).

Le macchine programmate proposte da Hollerith, però, avevano il limite di non poter discriminare, o contare, più caratteristiche demografiche allo stesso momento. Per individuare tutte le donne residenti a Boston bisognava dapprima selezionare le tessere con foratura sul campo F e poi compiere una nuova cernita considerando la colonna indicante la città di residenza Boston. Nei modelli successivi Hollerith introdusse i **pannelli programmati** grazie ai quali si consentì la selezione di schede aventi più caratteristiche demografiche durante la stessa analisi. I pannelli programmati erano dotati di fili interconnessi con dei componenti attivi che simulavano gli operatori logici (and e or) dell'Algebra di Boole. I pannelli erano sostituibili e la disposizione dei cavi e degli interruttori logici fu data in incarico a una nuova figura professionale: **il programmatore**

Nel 1889, Dorr Eugene Felt realizzò Comptometer, **una calcolatrice dotata di tastiera che riproduceva il risultato su un nastro cartaceo**. Il primo esemplare fu realizzato con una scatola per la pasta, tanto da prendere il nome di Macaroni Box. Felt creò un primo sistema di calcolo automatico con un'interfaccia semplice da usare e d'immediata comprensione nel funzionamento (logica ancora visibile nel tastierino numerico delle calcolatrici e degli elaboratori elettronici moderni).

A partire della seconda metà degli anni Trenta l'interesse per i calcolatori automatici fu oggetto di studio nelle accademie e in alcuni centri di ricerca; istituti impiegati nella risoluzione di procedimenti articolati e formati da calcoli complessi. Intanto si ebbero sviluppi tecnologico con l'introduzione dei diodi, triodi e flip flop.

Nel 1935 in Inghilterra, il matematico Alan Turing, presso l'Università di Cambridge, definì un modello di sistema di calcolo in grado di eseguire algoritmi, **la Macchina di Turing**, sfruttando un'Unità di Calcolo, un nastro di lettura-scrittura e un'Unità di Controllo che, mediante un insieme di regole, controllava il comportamento del dispositivo stesso. Questo modello realizzava un sistema veloce e che non necessitava del controllo umano per determinare il flusso di esecuzione.

Nel 1942 Alan Turing fu chiamato dall'esercito britannico presso il centro segreto Code and Cypher School di Bletchley Park di Londra per collaborare al progetto Colossus,

un calcolatore il cui fine era di decifrare, in tempo reale, i messaggi segreti delle forze armate naziste. Il progetto portò alla realizzazione del primo calcolatore digitale, COLOSSUS, che aveva delle dimensioni imponenti (occupava un'intera stanza). Nel 1942 John Atanasoff e Clifford Berry riuscirono a realizzare il primo calcolatore completamente elettronico, Atanasoff Berry Computer. Il calcolatore occupava un'intera scrivania, utilizzava 280 triodi, come amplificatori e 31 tiratron, un tubo riempito di gas che svolgeva la funzione di interruttore. Le componenti deputate al calcolo erano interconnesse con circa 1.5Km di cavi.

Nel 1943, John William Mauchly e John Presper Eckert dell'Università della Pennsylvania furono ingaggiati dallo United States Army's Ballistic Research Laboratory per realizzare **il primo elaboratore programmabile completamente elettronico** (Electronic Numerical Integrator And Computer, **ENIAC**).

Tra i progettisti era presente il fisico ungherese, naturalizzato statunitense, John von Neumann che inquadrò in una teoria matematica coerente il sistema di calcolo e gli automi e realizzò un nuovo modello di elaboratore, definito **Macchina di von Neumann**, che si avvicinava a quello formalizzato anni prima da Alan Turing. Si trattava, cioè, di una macchina automatica in grado di eseguire un algoritmo. Il nuovo dispositivo aveva tutti i componenti automatizzati; aspetto utile per incrementare la velocità di calcolo, ed era dotato di sistemi di auto controllo, per verificare la correttezza di quanto in corso di esecuzione.

Fino agli anni Sessanta gli elaboratori avevano grandi dimensioni, l'energia richiesta per il funzionamento e il raffreddamento dei componenti elettromeccanici era elevata e, naturalmente, erano molto costosi; per questo motivo si tendeva a sfruttarli il più possibile e, quindi, l'utilizzo era suddiviso generalmente fra più programmi. Ciascun utente produceva il proprio programma, lo consegnava all'operatore che lo includeva nella coda dei processi e alla fine ridistribuiva i risultati. Queste macchine furono denominate **mainframe** e spesso la comunicazione, sia in input sia in output, avveniva in binario mediante l'uso di schede o di nastri perforati (alcuni modelli usavano delle telescriventi che riproducevano le istruzioni scritte dai programmatori in codice binario come perforazioni su un nastro cartaceo). Un primo passo in merito al ridimensionamento degli elaboratori e all'incremento rilevante della velocità di computazione avvenne nel 1947 grazie agli studi sui materiali semiconduttori per merito di John Bardeen, Walter Brattain e William Shockley. I tre scienziati, incaricati nel realizzare amplificatori per il sistema di telecomunicazione statunitense, crearono un nuovo commutatore di segnale, il transistore, che nel giro di dieci anni, sostituì il triodo a valvola consentendo di realizzare circuiti elettrici rapidi e compatti e favorendo la nascita della microelettronica.

Nel 1950 Aiken completò il Mark III impiegando esclusivamente valvole e prevedendo all'acquisizione dei dati attraverso il **nastro magnetico** (magnetic tape), una memoria di massa a contenuto permanente in cui l'informazione binaria era stipata su materiale magnetico.

Nel 1951 UNIVersal Automatic Computer, società fondata da Eckert e Mauchly, propose per le grandi aziende UNIVAC-1 un **elaboratore a uso generico**. I programmi furono redatti usando uno dei primi linguaggi assemblativi, lo Short Order Code.

Nel 1954 la società Texas Instruments presentò il transistore con base in **silicio**. Questo nuovo elemento non solo era più economico (in termini di produzione) dei precedenti modelli, ma garantiva valide prestazioni temporali e una maggiore compattezza. Il circuito integrato ottenuto dai transistori con base in silicio, grazie alle innovative tecniche fotolitografiche, garantì un volume ridotto, un'elevata velocità di elaborazione e un minimo consumo di corrente elettrica. Nel 1957 si assistette a un'innovazione nei supporti di memorizzazione permanente: nel sistema Random Access Method Of Accounting And Control (RAMAC) di IBM si installò una unità a **dischi magnetici** (hard-disk), cioè una memoria digitale stabile ad accesso diretto, con tecnologia magnetica, strutturata con componenti elettromeccaniche e a forma di piatto.

In Italia, intanto, il Centro Studi dell'Università di Pisa, su stimolo di Enrico Fermi, realizzò il prototipo della Calcolatrice Elettronica Pisana (CEP).

Negli anni Sessanta il panorama progressivamente cambiò grazie all'affermazione di **nuovi transistori** e alla produzione di sistemi dal costo ridotto e dalla forma compatta. Per distinguerli dai mainframe ai nuovi elaboratori fu associato il termine minicomputer. Tra i minicomputer ebbe un clamoroso successo il PDP-1, proposto nel 1960 da Digital Equipment Corporation. Questo elaboratore era costituito da 2700 transistori e 3000 diodi con una frequenza di clock di 5MHz. L'aspetto innovativo era un videoterminale a tubo catodico, dalla forma di oblò, che aveva una rappresentazione dei segni grafici in modo vettoriale (e non a mappa di punti di colore). Il programmatore scriveva il codice mnemonico utilizzando una tastiera che produceva in uscita un nastro perforato con codifica binaria, il quale poteva essere direttamente sottomesso al sistema. Queste due periferiche decretarono l'obsolescenza delle schede perforate.

Nel 1970 dai laboratori Xerox di Paolo Alto fu proposto l'elaboratore Xerox Alto. Questo modello era dotato di un display con rappresentazione a mappa di punti di colore (bitmap), aveva un rudimentale **Sistema Operativo a interfaccia grafica** basato su delle finestre sovrapponibili per mostrare i documenti e la loro disposizione contenuti nella memoria di massa (cioè c'era la rappresentazione dell'organizzazione logica dei documenti gestita dal file system); e, infine, aveva circuiti integrati che gli consentivano di essere interconnesso sia a una stampante laser sia a una rete locale.

Nei primi anni Settanta, grazie a tecniche litografiche sempre più raffinate e grazie alla tecnologia **nMOS**, si produssero microprocessori in cui tutte le componenti dell'Unità di Elaborazione risiedevano in un singolo chip, cioè una piastrina di silicio. Gli elaboratori, ancor più compatti e veloci, furono rinominati **microcomputer**. Il primo microcomputer per uso generico, Intel 4004, fu progettato nel 1971 dal fisico italiano Federico Faggin e dagli ingegneri statunitensi Marcian Edward Hoff Jr. e Stanley Maze. Intel 4004 aveva una dimensione di circa 42x3mm, era costituito da 2300 transistori e operava ad una frequenza massima di 740KHz; aveva una potenza di calcolo parago-

nabile a quella dell'ENIAC, che aveva bisogno di circa 19000 tubi a vuoto e occupava un'intera stanza.

Il microprocessore lavorava con parole di **4bit e indirizzi di dimensione 12bit**, che consentivano di accedere a 4096 celle di Memoria Centrale (di lunghezza 8bit); infine era **in grado di operare con solo cifre numeriche binarie** (solamente con il passaggio a microprocessori con parole a 8bit fu possibile rappresentare i caratteri alfanumerici e quelli di punteggiatura).

Nel 1972 dopo la stessa azienda propose il primo microprocessore con parole di 8bit, Intel 8008. Il chip aveva solamente 18 piedini (cioè, le piste su cui viaggiano i segnali per la comunicazione con le altre Unità); un numero di porte fisse, 8 per l'input e 24 per l'output; e una struttura ad interconnessione basata su un bus a 8bit. Inoltre, richiedeva molti altri circuiti di supporto per il suo funzionamento. Il processore sfruttava indirizzi a 14bit, che consentivano l'accesso a 16KB di memoria; l'indirizzo era memorizzato in un apposito registro, il **Memory Address Register (MAR)**, esterno all'Unità di Elaborazione e a ridosso della Memoria Centrale. Fu anche proposto il primo sistema ad **interruzioni** per migliorare i tempi di comunicazioni con le periferiche agevolando e migliorando i tempi del trasferimento dei dati.

Nel 1976 esce il processore Intel 8085. Questa nuova architettura aveva dei transistori che lavoravano con un'alimentazione a 5Volt, si accedeva a 64KB celle di Memoria Centrale e aveva un'interfaccia comune per interconnettere e gestire le periferiche. In più tale processore sfruttava l'accesso diretto alla memoria, per spostare velocemente ed indipendentemente dalla CPU dati dalle periferiche alla **Memoria Centrale** (e viceversa), e usava il sistema di interruzione vettorizzata, che perfezionò il **multitasking** e migliorò le prestazioni della macchina per le comunicazioni e i trasferimenti con i **dispositivi di ingresso-uscita**.

Nel 1976 Jobs e Wozniak proposero la serie di elaboratori a uso personale Apple in cui erano inclusi programmi di videoscrittura, fogli di calcolo e giochi. La serie Apple aveva una tastiera, simile a quella di una macchina per scrivere e comunicava con l'esterno attraverso un qualsiasi televisore sul quale era in grado di rappresentare appena quaranta caratteri su sedici righe. La grande novità di questa famiglia di elaboratori non era rappresentata tanto dal processore o dalle periferiche, ma piuttosto dall'adozione del linguaggio ad alto livello, il BASIC, che permetteva la scrittura di programmi con una sintassi molto semplice.

Nel 1978 si affermò la terza generazione di microprocessori in grado di operare con parole a 16bit (tecnologia pMOS). Un esempio fu il processore Intel 8086 attraverso un bus dei dati largo 20bit era in grado di indirizzare direttamente 1MB di memoria, una quantità molto ampia per quei tempi. Il processore gestiva la **memoria segmentata** ovvero la rilocazione dei programmi. In altre parole, era possibile spostare ed eseguire un programma in una qualsiasi zona di memoria. In questo modo si superò la necessità di svolgere il processo di compilazione ogni volta prima di caricare il programma o a posizionare il programma stesso sempre in una locazione prestabilita.

Sempre nel 1979 Motorola presentò il processore 68000. Grazie alla potenza di calcolo, tra cui il multi-bus (o bus multicanale) che offriva trasferimenti concomitanti d'informazioni eterogenee (dati, indirizzi e segnali di controllo), gli elaboratori consentirono agli utenti di lavorare con i dispositivi in tempo reale (es.: sintetizzatori audio elettronici) e una grafica di pregiata qualità.

Nel 1982 Intel presentò il modello 80286 che fu il primo processore completamente a 16bit. Tra le innovative caratteristiche, c'erano cinque nuovi registri per la gestione del multitasking. Per mantenere la compatibilità con i modelli precedenti, il processore Intel 80286 agiva in due modi: reale o protetta. Nella modalità reale si comportava come il modello 8086 e non utilizzava i registri supplementari. La modalità protetta consentiva l'esecuzione di più programmi in maniera pseudo parallela. Un'altra proprietà fu quella della memoria virtuale che, anche grazie alla tecnica delle interruzioni, permise di superare il limite legato all'esecuzione di un programma (o più) di dimensione complessiva inferiore o uguale alla memoria presente fisicamente. La frequenza di clock inizialmente era di 6MHz, divenne presto otto, quindi dieci, fino a modelli a 20MHz.

1.4 Elaboratori elettronici di seconda generazione

Nel 1982 David Patterson e John Hennessy idearono l'architettura Microprocessor without Interlocked Pipelined Stages (**MIPS**), in cui ogni istruzione poteva essere eseguita durante un solo segnale di clock. Il set d'istruzioni malgrado fosse minimo ed avesse pochi modi di indirizzamento era sufficiente a eseguire algoritmi complessi e accedere in tutte le parti della memoria. Il processore, in più, sfruttava la tecnica della **canalizzazione** (pipeline): le istruzioni non erano più eseguite sequenzialmente ma, rendendo indipendenti le fasi in cui si preleva, codifica ed esegue una singola istruzione, si procedeva alla loro sovrapposizione; migliorando le prestazioni complessive della macchina.

Il Microprocessor without Interlocked Pipelined Stages (MIPS) muta il paradigma di elaboratore secondo lo schema di von Neumann adottando una doppia memoria: una riservata alle istruzioni e un'altra riservata ai dati.

Nel 1984 Sony e Philips presentarono il disco ottico a sola lettura (CD-ROM), un supporto portatile di capacità di 640MB impiegato come sostituto del disco in vinile per i contenuti musicali in formato digitale. Nel tempo usciranno i modelli superiori: DVD e Blu-Ray.

A metà degli anni Ottanta ci fu anche il passaggio alla quarta generazione di microprocessori. Si affermò la tecnologia CMOS che garantì frequenze superiori ai 50MHz, minimizzò la dissipazione di potenza e offrì un ridotto consumo di energia rispetto ai modelli precedenti. Un primo modello di questa nuova generazione fu il processore Intel 80386 che poteva operare in tre differenti modi: reale, protetta e virtuale86. Nella modalità **reale** lavorava come un 8086, ma con prestazioni più efficienti. In modalità

protetta era un 80286 dotato di multitasking e gestione della memoria virtuale. La **modalità virtuale86** permetteva di inizializzare un determinato numero di macchine virtuali, assegnando a ciascuna una copia del Sistema Operativo DOS. Ciascuna macchina virtuale era in grado di gestire autonomamente un ambiente simile ad un elaboratore 8086, mantenendolo isolato dalle altre istanze e lavorando in multitasking.

Il modello 80386, inoltre, adottò la tecnica di caching della memoria, saldando in prossimità del chip una memoria statica (Static RAM, SRAM) per velocizzare la trasmissione dati tra il processore e la Memoria Centrale. L'impiego di questa memoria molto veloce (ma costosa) si fonda sul principio di località temporale, alcuni dati appena impiegati possono essere richiesti di nuovo per la successiva elaborazione (es.: le istruzioni nei cicli), e di località spaziale, i dati processati risiedono in un intorno vicino (es.: gli elementi di un vettore). Questi blocchi di dati sono successivamente stipati nella memoria cache, escludendo richieste continue alla più lenta memoria di lavoro.

Nel 1989 uscì, anche, Intel 80486 che su un unico chip ospitava un processore 80386 e tutte quelle parti che erano considerate moduli aggiuntivi nei modelli precedenti come: il coprocessore matematico, la memoria cache e la componentistica per la gestione della grafica tridimensionale. In particolare, grazie ad un algoritmo predittivo statistico, la cache integrata non solo immagazzinava i dati con un accesso più recente, ma anticipava l'importazione di dati residenti nella Memoria Centrale non ancora richiesti, introducendo così la modalità a lettura anticipata (Read-Ahead). Una ulteriore importante novità fu una modifica strutturale dell'architettura, Control ROM. che garantì la retrocompatibilità con le precedenti versioni e con le istruzioni CISC e, allo stesso tempo, la circuiteria in grado di eseguire direttamente istruzioni RISC.

1.5 Elaboratori elettronici di terza generazione

Nel 1993 Intel annunciò il Pentium. L'architettura aveva una suddivisione delle cache in due parti, la prima contenente istruzioni la seconda dati, con dei canali preferenziali di accesso e sfruttava delle tecniche di parallelismo nell'elaborazione delle istruzioni (non sempre efficace). Descrivendo sinteticamente la logica, si consentiva il prelievo e l'esecuzione di due istruzioni (canale U, pipe U, e canale V, pipe V) nello stesso colpo di clock utilizzando due canali separati fisicamente. Se l'esecuzione non fosse potuta avvenire parallelamente, a causa di una relazione tra le istruzioni, si sarebbe cercato di risolvere la criticità modificando l'ordine delle istruzioni, senza svuotare i canali. Infine, il Pentium era dotato di un coprocessore matematico in grado di svolgere i calcoli di addizione, moltiplicazione e divisione tra numeri reali.

Nel 1995 fu proposto il Pentium Pro che rappresentò un vero salto generazionale. In primo luogo c'era la cache di livello due. La canalizzazione raggiunse le 14 fasi. Infine, c'era l'esecuzione fuori ordine (Out of Order): le istruzioni erano convertite in micro-operazioni (micro-ops) per poi essere passate a un componente di esecuzione capace di eseguirle fuori ordine; in altre parole, si processavano quelle pronte, non necessariamente

in sequenza, e si lasciavano in attesa quelle che non lo erano. La sequenza delle istruzioni era, infine, riordinata da una sezione dedicata, detta Memoria di Riordine (Reorder Buffer), alla fine dell'elaborazione.

Nel 1997 fu commercializzato da Intel il Pentium II che incorporava la tecnologia MMX, progettata specificamente per l'elaborazione di dati video, audio e grafici (le funzioni trigonometriche per le rotazioni di punti d'immagine tridimensionali impiegavano calcoli con numeri reali aventi una rappresentazione in virgola fissa).

Nel 1999 fu proposto il Pentium III che ebbe come innovazione principale l'estensione Streaming Simd Extension (SSE). Grazie a questa tecnica si ebbe un potenziamento del coprocessore matematico che operò in modalità Single Istruction Multiple Data (SIMD): cioè, si eseguiva in parallelo la stessa istruzione su più dati prelevati in blocco; orientando parte del processore ad un'architettura vettoriale.

Nel 2001 non riuscendo più ad aumentare significativamente le prestazioni della macchina (la frequenza di clock si attestò all'ordine di grandezza del GHz), le case di produzione decisero di puntare completamente sul parallelismo dei processi ottenuto mediante più elaboratori, i **multiprocessori** (multi-processor), o più Unità di Elaborazione sullo stesso chip, i **multi-nuclei** (multicore).

Tra il 2001 e il 2002, Sunnyvale presentò il microprocessore Athlon XP, per gli elaboratori desktop, e Athlon MP, per i server, entrambi dotati di più di un microprocessore per scheda madre e transistori di dimensione di 0.13 micrometri. L'impiego di più processori era già stato usato per il supercalcolatore Cray-1, prodotto nel 1976, come variante dei mainframe, dall'istituto Cray Research per il calcolo parallelo sui dati. L'elaboratore aveva una memoria di massa di 303MB e una potenza di calcolo di 9megaflops (9 milioni di operazioni con numeri reali rappresentati secondo il formato in virgola mobile).

Nel 2005 fu prodotto il Pentium D, un **processore con due nuclei** (dual core), cioè due Unità di Elaborazione sullo stesso chip, e una memoria cache condivisa. Il suo lancio fu seguito dopo solo pochi giorni dal processore Athlon 64 X2 prodotto da AMD.

1.6 Nuove Prospettive

Negli ultimi anni il processo di sviluppo dei microprocessori si avvia verso limiti progettuali invalicabili: la dimensione dei transistori non è più riducibile e la dimensione del chip non può richiedere uno spazio superiore a quello corrente. Diversi sono i rami di ricerca; quelli che godono menzione sono: la tecnologia tridimensionale, la tecnologia fotonica e i nano-fogli.

La **tecnologia tridimensionale** consente la produzione di microprocessori (e altri moduli funzionali) ad alta densità. In questi chip i transistori non hanno più una disposizione planare, ma hanno uno sviluppo anche in altezza creando dei volumi rettangolari.

La **tecnologia fotonica** opera sulla radiazione luminosa. Questo consente un tempo di trasmissione e di elaborazione dei dati molto rapido (fino 100Gb al secondo) ed esclude le interferenze elettromagnetiche o quelle legate al surriscaldamento (effetto Joule) che possono incrementare il numero di errori (l'alterazione di un bit di una istruzione provoca danni non prevedibili). Le porte logiche sono realizzate con cristalli, l'informazione binaria è ottenuta impiegando un raggio laser e la lettura avviene con dei foto-ricettori.

I **nano-fogli** sfruttano le nano tecnologie che permettono di realizzare dispositivi simili ai transitori con una dimensione di circa 5nanometri, consentendo così di quadruplicare il numero di porte logiche presenti all'interno di un chip. Come materiale non si impiega più il silicio, ma il grafene che ha proprietà superiori.

2 I Principi del Sistema Binario

2.1 Teoria dell'informazione

La teoria enunciata da Shannon poneva l'attenzione su come inviare un messaggio, rappresentato da una concatenazione di parole, simboli (o segnali), e ricostruirlo in modo esatto (o con una buona approssimazione) quando ricevuto da una postazione remota.

Claude Shannon propose uno schema di sistema di comunicazione nel quale indicò gli elementi fondamentali costituenti (la sorgente, trasmettitore, canale, ricevitore, destinatario).

- La sorgente indica l'insieme delle parole possibili.
- Il canale è il mezzo attraverso il quale si propaga il segnale codificato.
- Il Ricevente è l'intermediario che decodifica il segnale nella corrispondente parola.
- Il destinatario, colui al quale si indirizza la collezione di simboli.

Shannon non prese in considerazione il significato dei termini né del messaggio, ma nel definire la sorgente evidenziò che questa può essere specificata come un insieme di simboli possibili. Lo scienziato statunitense comprese che in un sistema efficiente è importante garantire unicamente che siano i singoli segnali a giungere a destinazione in maniera corretta.

Per una comunicazione affidabile Shannon scelse come alfabeto quello costituito da due soli simboli 1 e 0, i bit (o 'unità minima di informazione'). Shannon dimostrò che sfruttando l'alfabeto binario era possibile costituire delle parole rappresentanti un messaggio.

Gli studi di Shannon si concentrarono anche sulla definizione dei criteri (cioè, delle formule matematiche) grazie ai quali è possibile ricevere le parole originarie trasmesse lungo un canale affetto da rumore. La base binaria si dimostrò quella più utile e più robusta nel caso di interferenze. Nel caso di errori, in un sistema di comunicazione a cifre binarie il messaggio originale può essere deducibile, cioè, rimane 'visibile' anche oltre il disturbo. Inoltre, la modifica di una cifra in un sistema non binario induce ad un numero di errori elevato.

Avere un alfabeto binario di riferimento inoltre è comodo perché molti dei componenti negli elettronici (porte logiche, commutatori elettronici, amplificatori) operano proprio con due livelli distinti di corrente elettrica (segnale alto, 1, e segnale basso, 0).

Oltre a queste nozioni Shannon introdusse il concetto di ridondanza, cioè un numero di dati (o caratteri) rappresentanti una parola maggiore di quello richiesto, che sono

aggiunti per garantire l'integrità dei messaggi nel caso di eventuali disturbi, e di codifica economica, cioè una riscrittura delle parole con un numero minimo di caratteri.

Dal concetto di ridondanza si derivarono i codici per il rilevamento e per la correzione degli errori (ECC); molto impiegati nelle trasmissioni lungo la rete internet o nell'archiviazione dei dati sui supporti digitali.

Lo studio inerente ai codici economici, invece, portò alla compressione dati, strategia usata per accelerare i tempi di trasmissione dei segnali e per sfruttare al meglio lo spazio dei dispositivi di memorizzazione.

Poiché Shannon non focalizzò l'attenzione sull'associazione tra il significato e la parola rappresentante, nella teoria dell'informazione questa relazione è di tipo non esclusivo ovvero il significato muta in accordo al dominio di utilizzo: la parola 0 può identificare il colore nero se si considera una immagine o il silenzio se si riproduce un suono o il valore zero se si fa riferimento a numeri.

Per la parte pratica si consiglia di utilizzare il materiale fornito dal professore del corso di Sistemi Digitali. In alternativa c'è il primo capitolo negli appunti realizzati da Simone Bianco, sempre relativi al corso nominato.

2.2 Cosa Bisogna Sapere

1. Numeri binari naturali.
2. Conversioni, Unità di misura e Sistema esadecimale.
3. Operazioni aritmetiche in sistema binario.
4. Numeri binari negativi.
5. Numeri in Sign/Magnitude.
6. Numeri in Complemento a 2.
7. Numeri binari razionali.
8. Numeri a virgola fissa.
9. Numeri a virgola mobile.

3 Reti Combinatorie

3.1 Porte Logiche

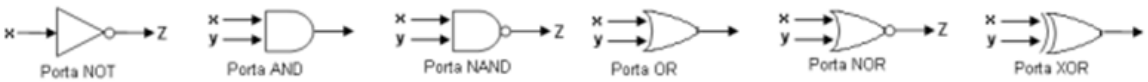
Una **porta logica** (gate) è un elemento di calcolo, realizzato mediante un componente elettromeccanico (relè, anni 1920- 1940) o elettrico (transistor, dal 1950) avente un determinato numero di **linee di ingresso** (fan-in) ed **una linea di uscita** (fan-out) che, eventualmente, può essere collegata all'entrata di una o più porte (eccetto quella da cui esce). I segnali applicati alle linee di ingresso e di uscita sono segnali elettrici e si possono associare a essi due valori convenzionali:

- 1 (presenza di segnale o segnale alto: [2V;5V])
- 0 (assenza di segnale o segnale basso: [0V;1V])

Il **relè** è un componente elettro - meccanico costituito da una bobina di filo conduttore elettrico, generalmente di rame, avvolto intorno ad un nucleo di materiale ferromagnetico. Al passaggio di corrente elettrica nella bobina, l'elettromagnete attrae l'ancora alla quale è vincolato il contatto mobile che quindi cambia posizione. Un relè è utilizzato per controllare un circuito elettrico, interrompendo o stabilendo il flusso di corrente in risposta a un segnale di controllo. In sostanza, funziona come un interruttore che può essere attivato o disattivato da un'altra fonte di energia o segnale elettrico.

Le **tecnologie microelettroniche** oggi più usate per la realizzazione di porte logiche sono: BJT (Bipolar Junction Transistor), ossia transistor bipolari: TTL (transistor-transistor logic) e ECL (emitter coupled logic). MOS (field-effect transistor), pMOS, nMOS, CMOS.

Ciascuna porta logica risolve una funzione (o tabella della verità). Le principali porte logiche utilizzate sono: NOT, OR, NOR, AND, NAND e XOR.



NOT	
x	Z
0	1
1	0

AND		
x	y	Z
0	0	0
0	1	0
1	0	0
1	1	1

NAND		
x	y	Z
0	0	1
0	1	1
1	0	1
1	1	0

OR		
x	y	Z
0	0	0
0	1	1
1	0	1
1	1	1

NOR		
x	y	Z
0	0	1
0	1	0
1	0	0
1	1	0

XOR		
x	y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Figura 1: **Porte Logiche**

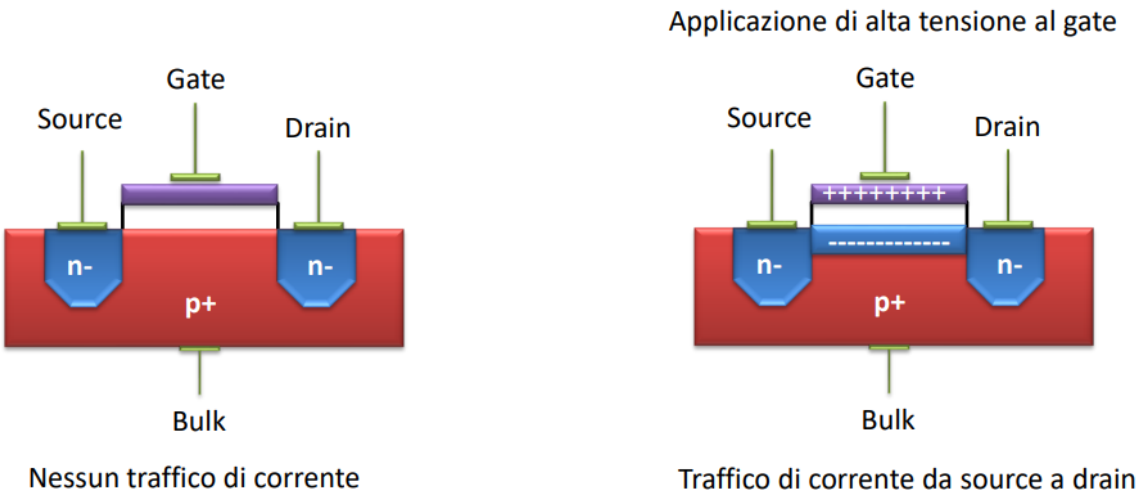


Figura 2: **nMOS**

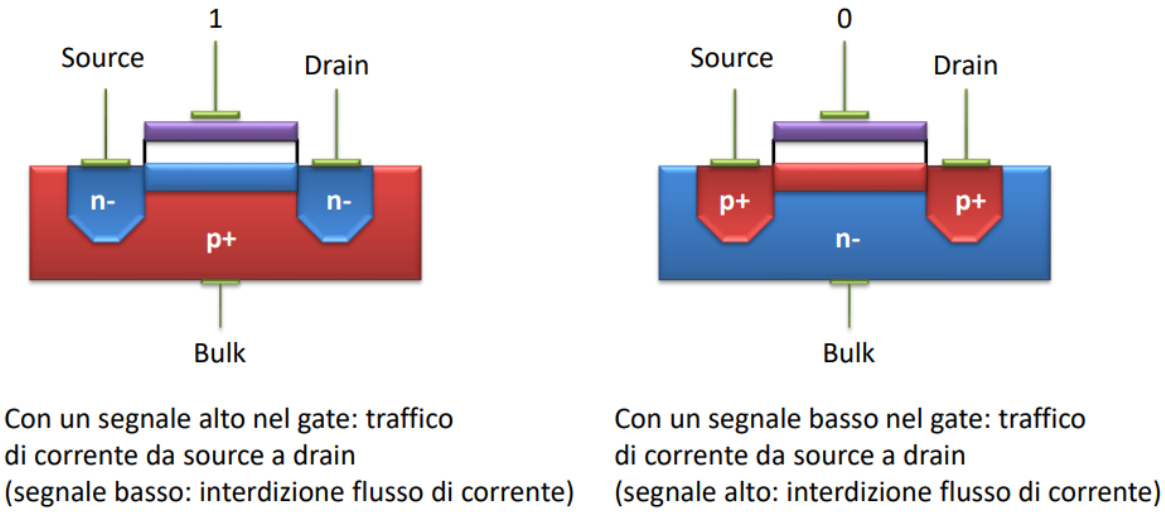


Figura 3: **CMOS: nMOS - pMOS**

3.2 Rete Combinatoria

Una rete è una interconnessione di componenti attivi,le porte, collegati tra loro mediante componenti passivi, le linee.

Una rete combinatoria è un circuito elettronico in grado di elaborare, in modo automatico, funzioni binarie di una o più variabili binarie. Formalmente una rete **combinatoria** è definita come un dispositivo con n linee di ingresso ed m linee di uscite per cui i segnali di uscita dipendono unicamente dai segnali di ingresso.

Osservazione. In una rete combinatoria non sono presenti cicli né cappi Nelle reti combinatorie non può avvenire che gli stessi ingressi forniti in istanti diversi diano luogo ad uscite diverse.

Uno schema circuitale è un collegamento di porte (rappresentate in maniera grafica) tramite linee. **Ci sono tre tipi di linee:**

1. linee di ingresso, ognuna etichettata con una delle n variabili booleane.
2. linee di uscita, ognuna etichettata con una delle m variabili di uscita.
3. linee interne, ciascuna delle quali collega l'uscita di una porta con l'ingresso di un'altra porta.

Vincoli:

- Ogni ingresso di ogni porta deve esserecollegato ad una linea di ingresso oppure aduna linea interna.
- L'uscita di ogni porta deve essere collegata aduna linea di uscita oppure ad una lineainterna.
- Il collegamento di porte tramite linee nondeve dare luogo a cicli

Una rete combinatoria può essere vista come un dispositivo in grado di soddisfare una tabella, la tabella della verità, che per ognuna delle 2n combinazioni possibili relative agli n valori di, x2 ,.., xn) indica gli m valori di uscita (z1, z2, .., zm).

Osservazione. Ad ogni rete caratterizzata da una tabella con n ingressi ed m uscite corrisponde un gruppo di m espressioni booleane (e viceversa).

3.2.1 Caratteristiche Progettuali

Tra i parametri principali nel progetto di una rete combinatoria, è opportuno considerare:

- **L'assorbimento di energia** (che stabilisce un limite complessivo al numero di porte utilizzabili).
- **Il ritardo** (che determina la velocità di calcolo). La velocità di calcolo della rete combinatoria. Varia in base alla profondità della rete (di solito si considera un tempo costante perché la rete combinatoria ha una profondità finita).
- **Il costo di realizzazione**. Dipende dal numero di transistor impiegati che cambia a seconda della tecnologia usata, della funzione da soddisfare e il numero di ingressi. Es.: la porta NOT è costituita da 1 transistor, NAND o NOR 2 transistor; AND e OR 3 o 4 transistor; altre porte: > 4 transistor

3.2.2 Decoder

Il decodificatore è una rete combinatoria che trasforma parole associate a codifiche strette in parole associate a codifiche lasche (le linee di uscita sono in numero maggiore rispetto le linee di ingresso).

Un decodificatore è una rete combinatoria con m linee di ingresso e n=2m linee di uscita. Logicamente il decodificatore riconosce una stringa (es.: una locazione di memoria o unaistruzione).

Il **decodificatore** è usato, ad esempio, per identificare una cella di memoria.

3.2.3 MUX

I **codificatori** sono una famiglia di reti combinatorie che trasformano parole codificate in una codifica lasca in parole d'uscita rappresentate in codifica stretta.

In generale un codificatore è una rete combinatoria che ha n linee di ingresso e m = $log2(n)$ linee di uscita, cioè vi è la produzione della codifica binaria dell'indice dell'unica linea di ingresso attiva.

Logicamente il codificatore è un generatore di codici (es.: dei comandi).

3.3 Circuiti Elementari

3.3.1 Addizionatore (Adder)

L'**addizionatore** è una rete combinatoria che consente l'operazione di somma tra due operandi (addendi).

La rete combinatoria associata ad un addizionatore può essere realizzata mediante una tecnica di decomposizione. Questo perché l'addizione di due numeri binari può essere vista come la somma di due bit alla i-esima posizione (xi e yi) ai quali va aggiunto il **riporto** (ri) per ottenere un **risultato** (zi) ed un eventuale **riporto** ($ri + 1$) per le **cifre successive**.

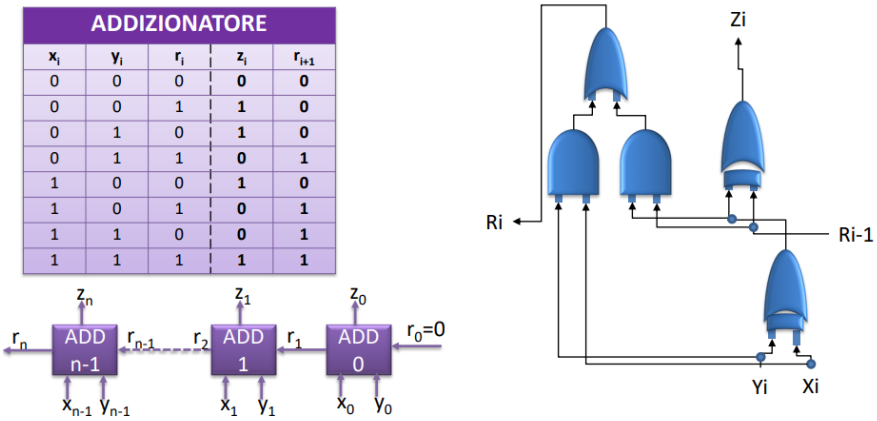


Figura 4: **Adder**

3.3.2 Sottrattore

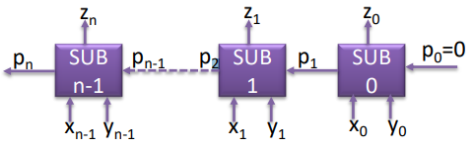
Il sottrattore è una rete combinatoria che permette la sottrazione tra due operandi (minuendo e sottraendo).

Anche in questo caso è possibile fare riferimento ad una struttura modulare. All'i-esimo bit del minuendo (xi) va sottratto sia il sottraendo (yi) sia ilbit di prestito (pi) della posizion eprecedente per poi generare il bit risultante (zi) ed il bit del prestito ($pi + 1$) per le cifre successive.

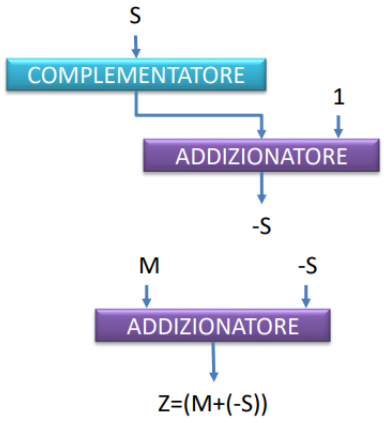
Nel concreto la realizzazione di un sottrattore può essere effettuato con una circuiteria differente da quella vista.

L'operazione di sottrazione $z = m - s$ si riduce alla espressione equivalente $z = (m + (-s))$ Si utilizza un complementatore ed un addizionatore che prende in input come addendi il numero complementato ed il valore 1. L'operazione di sottrazione si ottiene aggiungendo il minuendo.

SOTTRATTORE				
x_i	y_i	p_i	z_i	p_{i+1}
0	0	0	**0**	**0**
0	0	1	**1**	**1**
0	1	0	**1**	**1**
0	1	1	**0**	**1**
1	0	0	**1**	**0**
1	0	1	**0**	**0**
1	1	0	**0**	**0**
1	1	1	**1**	**1**



(a) **Tabella**



(b) **Schema Logico**

Figura 5: **Il Sottrattore**

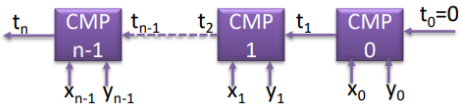
3.3.3 Comparatore Logico

Il **comparatore** è una rete combinatoria che ha una unica linea di uscita che vale 1 se il numero x, di n bit, applicato in ingresso risulta maggiore o uguale (in senso algebrico) al numero y, di n bit, anch'esso preso in ingresso, con cui si effettua il confronto. Anche per tale componente è possibile fare riferimento ad una struttura modulare.

Altresì, invece, è possibile realizzare un comparatore logico utilizzando n porte XOR, a cui in ingresso sono associati gli i-esimi bit dei valori da comparare, le cui uscite sono collegate ad una porta OR.

In questo caso, infatti, è necessario stabilire solamente se le stringhe binarie sono uguali o diverse. Un comparatore logico determina il risultato finale in tempo costante O(1).

COMPARATORE			
x_i	y_i	t_i	t_{i+1}
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**



(a) **Tabella**

Istruzione	Valore in RAX
movb 100,%RAX	0000000000000010
movw 100,%RAX	0000000000003210
movl 100,%RAX	0000000076543210
movq 100,%RAX	fedcba9876543210

(b) **Schema Logico**

Figura 6: **Il comparatore Logico**

4 Macchina di Von Neumann

4.1 Breve Storia

Nel 1642 **Pascaline**. Sistema per il calcolo di addizioni e sottrazioni creato da Pascal Nel 1672 **Stepped Reckoner**, ossia la calcolatrice meccanica a quattro operazioni (addizione, sottrazione, prodotto e divisione) inventata da Von Leibniz. Sistema per il conteggio di informazioni con caratteristiche comuni (tabulatore). Usato da Hollerith per il censimento statunitense del 1890. Uso di un ordinatore (sorter) e schede perforate.

ENIAC (Electronic Numerical lntegrator And Computer). L'elaboratore ENIAC era costituito da 18.000 valvole termoioniche e 1.500 relè, pesava 30 tonnellate e consumava 140KW di energia. Dal punto di vista dell'architettura, la macchina era dotata di 20 registri, ciascuno dei quali in grado di memorizzare un numero decimale a 10 cifre. ENIAC veniva programmato regolando 6000 interruttori multiposizione e connettendo una moltitudine di prese con una vera e propria foresta di cavi.

4.2 Macchine Programmabili

Famiglia di calcolatori con architettura che consente l'esecuzione di programmi memorizzati in memoria. **Un programma** è un insieme di istruzioni elaborate sequenzialmente (a meno di eventuali salti).

Nel 1945 fu presentato **un modello di elaboratore generale** grazie a John von Neumann e Hermann Goldstine.

Il modello di **"macchina di von Neumann"** (o "macchina di Princeton") prevedeva che i dati e le istruzioni fossero archiviate nella stessa memoria (architettura adottata dall'elaboratore EDSAC del 1949). Oltre al calcolo matematico, nel programma era possibile effettuare salti in accordo a precise condizioni. Tale modello, per quanto perfezionato nei singoli componenti, è fino oggi il punto di riferimento per la progettazione di un qualsiasi elaboratore elettronico.

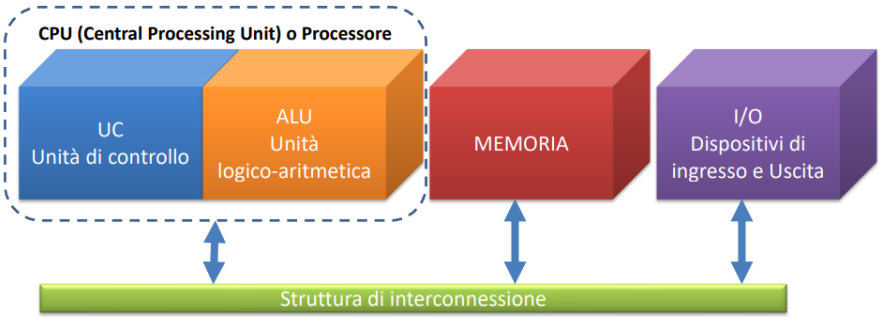


Figura 7: **La CPU**

4.3 CPU

4.3.1 Unità di Controllo

L'Unità di Controllo (Control Unit, **CU**) è predisposta a scandire le sequenze di operazioni elementari necessarie ad eseguire ogni singola istruzione. Le istruzioni devono essere prelevate dalla memoria, e trasferite alla circuiteria interna all'Unità di Controllo.

La **circuiteria dell'unità di controllo** deve riconoscere e generare i comandi atti all'esecuzione dell'istruzione (attivazione della struttura di interconnessione, passaggio dei dati e degli indirizzi in memoria, ...).

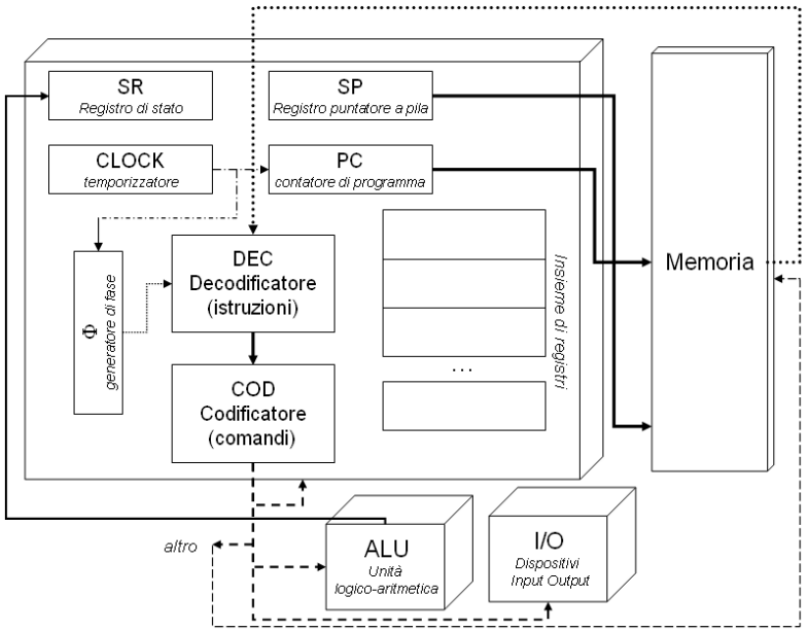


Figura 8: **Unità di controllo**

Nell'Unità di Controllo sono presenti dei registri e le differenze principali tra i essi e le celle di memoria è l'esiguo numero dei primi (sono realizzati con componenti costosi e performanti) e la loro presenza all'interno della CU: c'è una vicinanza e uno scambio di informazione diretto e rapido.

I **registri possono essere classificati** come registri ad **uso generale** o ad **uso speciale**.

I registri ad **uso speciale** sono :

1. Il Contatore di Programma (**Program Counter**, PC): è un registro contatore preselezionabile incrementato ad ogni periodo di clock, contenente l'indirizzo della cella di memoria dove è memorizzata l'istruzione da eseguire.

2. Il Registro di Stato (**Status Registe**r, SR o Processor Status Word, PSW): contiene informazioni che caratterizzano lo stato dell'Unità Centrale, tra cui, ad esempio, quelle relative all'ultima operazione eseguita (i condition codes provenienti dalla ALU).

3. Il Puntatore alla Pila (**Stack Pointer**, SP): contiene l'indirizzo della cima della pila (o canasta) cioè la zona di memoria usata per il passaggio di parametri tra funzioni.

Il **Generatore di Fase** (tipicamente realizzato con un registro contatore) scandisce le fasi delle operazioni elementari eseguite dalla CU che possono essere così schematizzate:

- **Caricamento** (*FETCH*): lettura dalla memoria della parola puntata dal PC. In questa fase la Memoria Centrale ed il codificatore presente nella CU si connettono per consentire il trasferimento dell'istruzione.

- **Decodifica** (*DECODE*): riconoscimento del tipo di istruzione e del modo di riferimento degli operandi. Non vi è alcuna connessione con componenti esterni perché il decodificatore è interno alla CU.

- **Esecuzione** (*EXECUTE*): esecuzione dei comandi definiti dal codice operativo dell'istruzione. Vi è connessione con tutte le unità richieste. Mentre le prime due fasi sono uguali per ogni istruzione, la fase di esecuzione varia in relazione dell'operazione coinvolta (es.: una moltiplicazione impiega più tempo di una addizione; più un modo di indirizzamento è complesso più si impiega tempo a reperire gli operandi).

- **Spostamento** (*MOVE*): esegue una movimentazione di dati (es.: si rimette in memoria il risultato di una istruzione aritmetica).

L'**elaborazione** di una istruzione consta nella successione di fasi che si ripete continuamente all'atto dell'accensione della macchina.

Una istruzione è sempre eseguita in queste fasi ma comprende un numero di cicli macchina variabile dipendenti, ad esempio, dal tipo di operazione, dal numero di accessi in memoria o alle unità di I/O. Ogni ciclo macchina, infatti, è implementato mediante una successione di un piccolo numero di operazione elementari eseguite in circuiti diversi sotto il controllo del transcodificatore. Ogni **operazione elementare** occupa un

periodo di clock e pertanto la durata di una istruzione dipende dal numero di accessi alla memoria, all'esterno della CPU e dal numero di operazioni elementari richieste.

Una volta che l'istruzione è stata caricata viene passata al **transcodificatore** (cioè il decodificatore delle istruzioni connesso col codificatore dei comandi) che riconosce l'istruzione e genera opportuni comandi per eseguire l'istruzione stessa.

Un esempio Pratico: **ADD 0x300,0x100,0x200**

- In fase di **fetch** si preleva l'istruzione dalla Memoria Centrale e si incrementa il PC.
- In fase di **decode**, il decodificatore riconosce l'addizione. Il codificatore, a sua volta, invia dei comandi (dei segnali elettrici) lungo le linee di ingresso della ALU che specificano il tipo di operazione che questa deve offrire.
- In fase di **load**, contestualmente il codificatore lancia dei segnali per prendere gli operandi in memoria alla locazione 0x100 e poi 0x200.
- In fase di execute, una volta reperiti gli operandi si deve generare la connessione con la ALU disconnettendo la memoria.
- La **ALU esegue l'operazione**.
- In fase di **movement**, si riattiva la linea con la Memoria Centrale per trasferire il risultato nella locazione 0x300.

L'insieme dei **registri ad uso generale** è anche denominato FR (**File Register**, archivio di registri); tali registri sono utilizzati per memorizzare, all'interno dell'Unità Centrale, i risultati temporane provenienti dall'ALU e le informazioni di controllo, allo scopo di diminuire il numero di accessi alla Memoria Centrale e di velocizzare il processo di elaborazione.

4.3.2 ALU

L'**Unità Logico-Aritmetica** è il componente che si occupa di effettuare operazioni logiche ed aritmetiche. Per il funzionamento di questa unità di solito sono impiegati un insieme di registri ad uso speciale che servono a contenere gli operandi e il risultato delle operazioni.

I **registri speciali** contenuti nella ALU, denominati **accumulatori**, sono trasparenti al programmatore (cioè il contenuto non può essere modificato dal programmatore mediante istruzioni) e svolgono la funzione di ospitare gli operandi prima e durante l'esecuzione o i risultati dopo l'esecuzione.

Gli **accumulatori** hanno un ruolo fondamentale per l'indirizzamento implicito: sono i registri in cui implicitamente vengono mandati gli operandi nel momento in cui si ricorre ad una istruzione logico-aritmetica.

Oltre agli accumulatori sono presenti delle **linee di ingresso** che individuano la funzione/operazione che deve essere attivata (le operazioni principali sono: ADD, NEG, AND, OR, COMP, TESTB, SHIFT) e delle linee di uscite su cui è ricondotto il risultato. Inoltre ci sono delle linee di uscita denominate condition code o flags che riportano informazioni relative all'ultima operazione eseguita.

Oltre agli accumulatori la ALU ha anche un registro speciale detto registro degli errori nel quale sono riportate situazioni non risolvibili (es.: divisione per zero, radice quadrata di un numero negativo) e che grazie al quale è possibile attivare un'interruzione interna. **La CU e la ALU identificano la CPU dell'elaboratore elettronico (il 'cuore' della macchina).** Col tempo si è provveduto a realizzare ALU specifiche, denominate coprocessori matematici (o ALU Attaccata), che eseguono funzioni complesse come i calcoli in virgola mobile (MULF, DIVF,COMPF,...) con un set di istruzioni dedicato non presente nel set di istruzioni della macchina.

Sebbene questa strategia ormai è stata abbandonata, includendo le funzionalità complesse direttamente nella **ALU-Nativa**, è tuttavia una pratica utilizzata nel caso in cui si voglia aggiungere nuove ALU che fanno operazioni che l'ALU-Nativa non svolge. In questo caso l'**ALU attaccata** è vista come un dispositivo I/O.

4.4 Memoria Centrale

4.4.1 RAM

RAM (Random Access Memory): Memorie volatili, cioè che perdono le informazioni in mancanza della tensione di alimentazione, il cui accesso a ciascuna locazione avviene in tempo costante.

- **SRAM** (Static RAM): Memorie statiche nelle quali l'informazione è memorizzata nell'equivalente di un latch D.
- **DRAM** (Dynamic RAM) Memorie dinamiche nelle quali l'informazione è memorizzata in un condensatore. Anche in presenza della tensione di alimentazione l'informazione contenuta in ogni cella è conservata per un breve periodo di tempo (dell'ordine di grandezza di 2 ms), passato il quale il contenuto deve essere ripristinato: questo avviene ciclicamente tramite un'operazione di "rinfresco" (refresh) che viene effettuata dal sistema.
- **SDRAM** (Synchronous DRAM) consente una maggiore flessibilità di impiego permettendo, grazie ad un apposito registro in uscita, di modificare il dato contenuto in una cella mentre si sta utilizzando il vecchio dato. Una ulteriore evoluzione della SDRAM è la **DDR** (Double Data Rate) che, come indica il nome, consente di operare a frequenza doppia potendo essere pilotata sia sul fronte di salita che sul fronte di discesa del clock.

4.4.2 ROM

ROM (Read Only Memory): Memorie di tipo non volatile che hanno la capacità di conservare l'informazione indipendentemente dalla presenza o meno della tensione di alimentazione. Sono memorie programmate dal costruttore e non sono modificabili dall'utilizzatore: è possibile solo la lettura dei dati contenuti. Anche in questo caso l'accesso ad ogni locazione avviene in tempo costante.

- **PROM** (Programmable ROM). Si tratta di memorie sulle quali l'utilizzatore può scrivere i dati una sola volta, utilizzando un apposito dispositivo di registrazione che brucia dei fusibili.
- **EPROM** (Erasable Programmable ROM). Sono memorie nelle quali l'utilizzatore può memorizzare i dati anche più volte, utilizzando appositi dispositivi di cancellazione (a raggi ultravioletti).
- **EEPROM** (Electrically Erasable PROM) o **EAROM** (Electrically Alterable ROM). Sono memorie EPROM nelle quali la cancellazione si può fare per via elettrica ma di solito è globale (coinvolge cioè tutti i dati registrati).

4.4.3 Memoria Centrale

La Memoria Centrale è una memoria volatile di tipo RAM (di tipologia DDR) costituita da tante locazioni (o celle) ciascuna delle quali può immagazzinare una stringa binaria di lunghezza finita n

Le stringhe presenti in Memoria Centrale possono essere: **istruzioni**, **operandi** o **indirizzi**.

Le locazioni della Memoria Centrale sono numerate in sequenza da 0 a $2m - 1$ (con m dimensione massima della memoria) e tale numero prende il nome di indirizzo della cella.

L'**indirizzo** specifica univocamente una locazione. Si accede a qualsiasi locazione con lo stesso tempo (il tempo di accesso ai dati non varia in relazione alla posizione).

La Memoria Centrale presenta delle aree riservate, in cui risiedono delle informazioni basilari utili al funzionamento della macchina (kernel del Sistema Operativo), ed altre in cui, per comodità sono riservate per operazioni particolari (**stack**, zona per trasferimento I/O,...).

La **Memoria Centrale**, per motivi progettuali ha **locazione di memoria di lunghezza 8bit**. In ogni caso nel momento in cui si stabilisce la lunghezza della parola si realizza una rete combinatoria che permette il prelievo di tante locazioni contigue quante necessarie per raggiungere la lunghezza della parola.

Ad esempio se il processore ha una parla di 32bit la circuiteria durante la fase di fetch preleverà (per default) 4 celle contigue in un solo istante.

La produzione di parole di 8bit non è solo legato a motivi progettuali ma consente anche il prelievo di dati di tipo *byte* (8bit), *halfword* (16bit), *word* (32bit) nella macchina a 32bit (8,32,64 in quelle a 64bit); lunghezze intermedie avvengono manipolando i dati prelevati aventi maggiore lunghezza (mascheramento).

I dati in memoria possono avere **due tipi di ordinamento** (endian): la numerazione comincia a partire dall'estremo più "grande" (cioè dal byte più significativo) è chiamato **big endian** usato nei protocolli internet. In contrapposizione c'è il sistema **little endian**, usato ad esempio in Intel, Digital, Motorola, IBM, SUN (big endian), MIS può essere impostato in entrambe le organizzazioni.

Per poter interagire con la Memoria Centrale è necessario che ci siano:

- **linee di ingresso** che specificano un indirizzo (in alcuni testi si fa riferimento al registro MAR, memory address register).
- **linee di uscita** per poter inviare o trasferire il dato (in alcuni testi si fa riferimento al registro MDR, memory data register).
- **un segnale di controllo** (generato dalla CU) per la lettura o la scrittura del dato.

È prevista, pertanto, una architettura costituita da un **decodificatore** che riceve in ingresso l'indirizzo della locazione di memoria alla quale si vuole accedere ed una linea che abilita questa a porre il suo contenuto in uscita dalla memoria o trascrivere in essa il dato da memorizzare.

Una organizzazione di questo tipo è impraticabile nel caso in cui la memoria abbia una grande dimensione. Con indirizzi, di lunghezza m, è possibile indirizzare 2m celle di memoria. Nel caso il valore di m sia grande (superi il valore 10) una architettura gestita da un singolo decoder è da escludere.

Per questo si ricorre ad una suddivisione logica della Memoria Centrale (multidimensione). **La Memoria Centrale** è suddivisa logicamente in banchi (bank, o piastre) e blocchi (block). L'**indirizzo** è suddiviso in campi ognuno con un significato associato ai banchi, blocchi e locazioni presenti e per ogni campo è presente un proprio decodificatore.

Ad Esempio: nel caso di due banchi con quattro blocchi avremo una suddivisione in tre campi: il **primo campo** di un bit indicante il banco; **il secondo** di due bit il blocco; il terzo dei rimanenti bit la posizione in cui risiede la locazione.

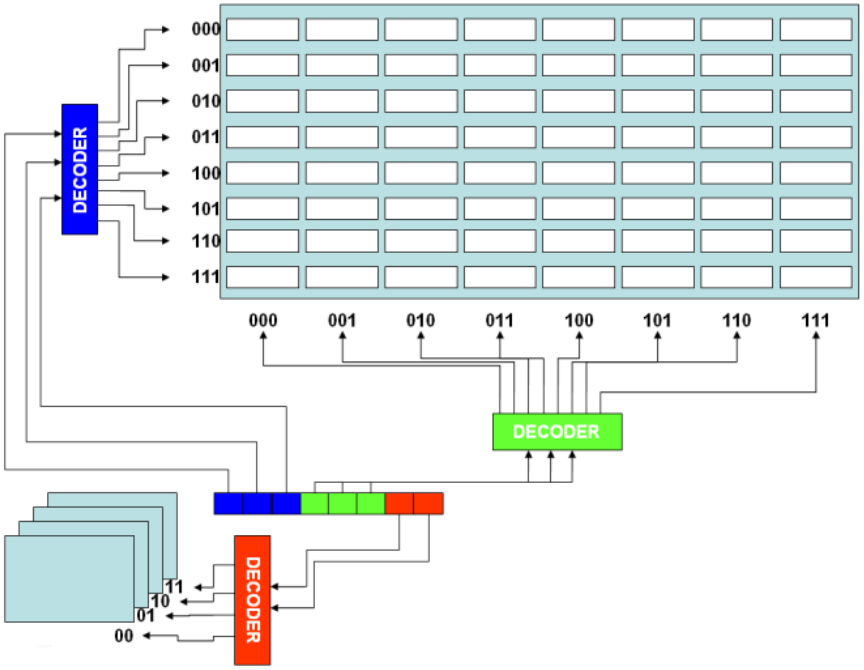


Figura 9: Suddivisione logica della memoria centrale

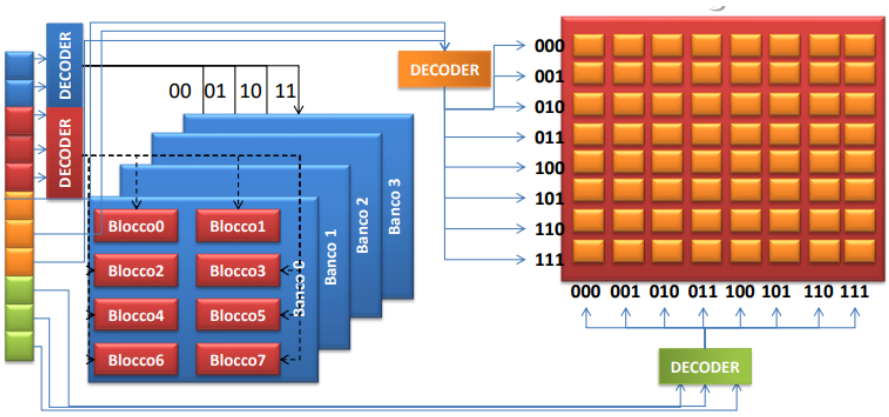


Figura 10: Memoria Centrale

4.5 Dispositivi di Input ed Output

I **dispositivi di input/output** (dispositivi di I/O o periferiche) consentono di collegare l'elaboratore, ed in particolare la Memoria Centrale con il mondo esterno (persone o altri dispositivi). Esistono numerosi tipi di dispositivi di I/O con caratteristiche molto varie che comportano problemi relativi alla conversione tra rappresentazione interna ed esterna dell'informazione.

Le **velocità di trasferimento** sono inferiori a quelle possibili all'interno delle altre componenti (CPU, Memoria Centrale) a causa della natura tecnologica di fabbricazione (componenti meccanici,...) e quindi l'uso delle periferiche comporta problemi di sincronizzazione e di adattamento della velocità.

Quando c'è un trasferimento dati è opportuno che il dispositivo coinvolto e il processore operino in modo coordinato mediante un insieme di regole: **il protocollo**.

Il protocollo consente l'interazione tra dispositivo (identificato da un indirizzo) e la Memoria Centrale sotto il controllo del Processore. Il **dispositivo** deve essere in grado di operare qualora il processore ne richieda l'intervento. Il **processore** deve eseguire le operazioni che consentono il trasferimento solo quando il dispositivo è pronto.

Nel **protocollo di input** una periferica vuole inviare dati all'elaboratore. I dati devono essere stipati in Memoria Centrale. Il protocollo prevede:

- L'individuazione del dispositivo di input che vuole inviare i dati
- La ricerca di un'area libera in cui stipare i dati
- Il prelievo del dato

Ad esempio = si richiede l'immissione di un valore da tastiera:

1. Si individua nella tastiera il dispositivo che vuole inviare i dati.
2. Si trova un'area libera della Memoria Centrale per stipare l'informazione.
3. Avviene il prelievo del dato immesso da tastiera residente in una memoria interna al dispositivo per essere spostato in Memoria Centrale.

Nel **protocollo di output** una periferica ospita i dati prodotti dall'elaboratore eli rielabora in relazione alla propria funzione (stampante, memoria di massa, controreazione nei joystick). Il protocollo prevede:

- L'individuazione del dispositivo che deve ricevere i dati
- Dei controlli sul dispositivo.
- L'invio dei dati

Ad esempio = si richiede il salvataggio di un'immagine su un disco magnetico:

1. Si individua il disco magnetico
2. Si trova un'area libera per stipare l'informazione e si svolgono controlli (ad esempio si controlla il nome del file per evitare che ci siano duplicati).
3. Avviene il trasferimento dei dati dalla Memoria Centrale al disco magnetico

Per interagire con il processore ogni dispositivo deve essere interconnesso ad un **modulo di I/O** (o interfaccia I/O o controller), cioè una rete sequenziale che colloquia con il processore inviando e ricevendo (tramite un bus di I/O) i segnali che, secondo il protocollo, controllano le operazioni di trasferimento.

Il protocollo di I/O pertanto è caratteristico dell'elaboratore, in quanto determinato dal modo di operare del processore, cioè dall'insieme di istruzioni di cui il processore può disporre per i trasferimenti. Questo vuol dire che i diversi dispositivi esterni collegati allo stesso elaboratore devono rispettare tutti lo stesso protocollo di I/O, indipendentemente dalla natura delle informazioni trasferite a dalla struttura fisica del dispositivo. Solamente in seguito il dispositivo da il giusto significato al codice ricevuto.

Schema di un dispositivo di input. Si evidenzia la sotto-rete (controller) che non dipende dal dispositivo e che è interessata nel colloquio con il processore. (Il controller è la parte più significativa dell'interfaccia di I/O).

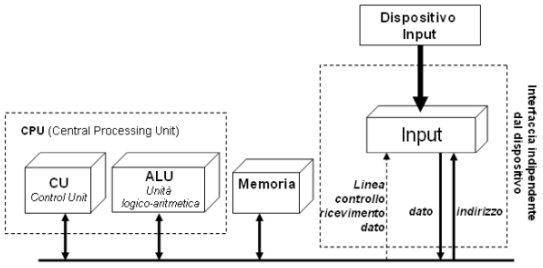


Figura 11: Controller

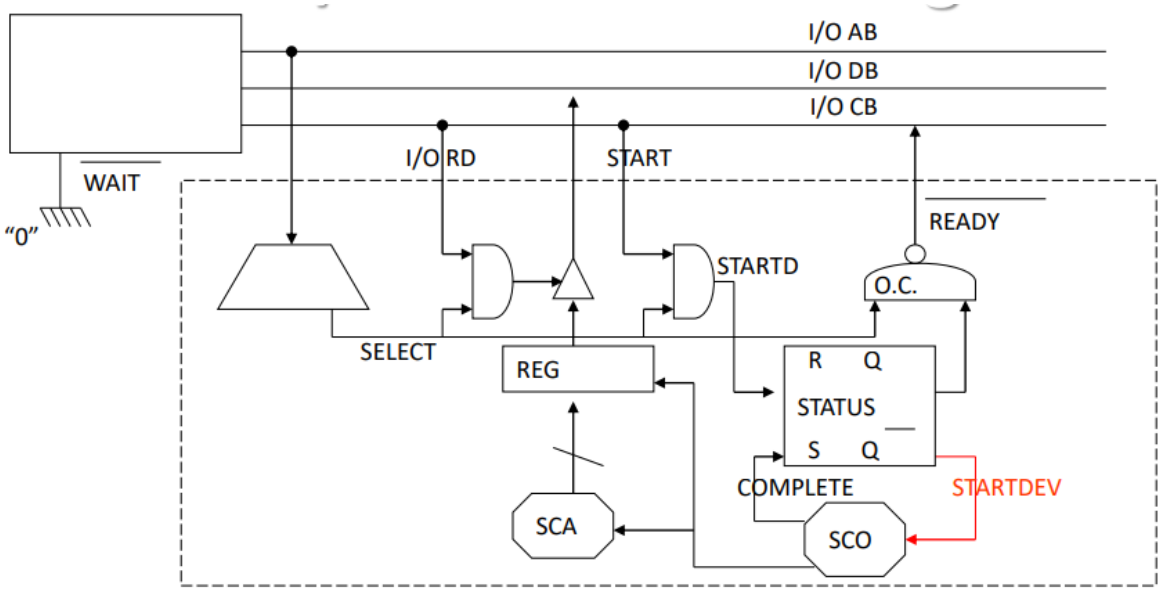


Figura 12: Controller Nel Dettaglio

Protocollo INPUT

1. Il processore invia sull'I/O Address bus l'indirizzo del dispositivo e ne esamina lo stato tramite la linea di controllo READY.

2. Se il dispositivo non è pronto il processore deve attendere e tornare al punto 1 (in alternativa procede elaborando un'altra istruzione e poi ripete il punto 1); se è pronto va la punto 3

3. Il processore avverte il dispositivo che può prendere un dato (seleziona il dispositivo tramite le linee indirizzi e invia il segnale START). START resetta il flip-flop STATUS e in tale stato rimane per tutta la durata delle operazioni di produzione del dato da parte del dispositivo

4. Quando il dato è stato prodotto ed è disponibile in REG, il dispositivo genera il segnale COMPLETE, settando STATUS (READY=0).

5. Nel frattempo il processore, in attesa del dato, esamina il flip flof STATUS campionando il segnale READY

6. Se READY= 1 il processore deve attendere e tornare al punto 5.

   Se READY= 0 il processore invia il segnale di controllo IO/RD per trasferire il dato presente in REG all'interno della locazione di memoria libera (cioè deputata ad ospitare il valore).

Protocollo OUTPUT

1. Il processore invia sull'I/O Address bus l'indirizzo del dispositivo e ne esamina lo stato tramite la linea di controllo READY.

2. Se il dispositivo non è pronto il processore deve attendere e tornare al punto 1 o svolgere un'altra istruzione. Se è pronto va al passo 3

3. Se READY=0 il processore trasferisce il contenuto di una locazione di memoria nel registro di interfaccia del dispositivo (mediante il segnale di controllo I/O WR)

4. Il processore avverte il dispositivo che gli ha trasferito un dato inviando il segnale START. START resetta il flip-flop STATUS e in tale stato rimane per tutta la durata delle operazioni di consumo del dato da parte del dispositivo. Quando il dato è stato letto da REG, il dispositivo genera il segnale COMPLETE, settando STATUS (READY=0).

5. Nel frattempo il processore, in attesa, esamina lo stato di STATUS campionando il segnale READY.

6. Se READY= 1 il processore deve attendere e tornare al punto 5.

   Se READY= 0 il processore può eseguire un'altra istruzione.

Per **indirizzare le unità di I/O** e consentire l'accesso al dato da trasferire o recuperare, il processore ricorre ad una delle due seguenti tecniche :

- Riservare all'I/O uno spazio di indirizzamento indipendente: si utilizzano specifiche istruzioni nelle quali si fornisce anche l'indirizzo identificativo del dispositivo da utilizzare nell'operazione (**I/O -CANONICO**).

- Riservare una porzione dello spazio di indirizzamento in Memoria Centrale ai dispositivi di I/O, in modo che ogni volta che il processore utilizza un indirizzo di questa porzione (con una tipica istruzione di trasferimento dati cioè senza ricorrere a specifiche istruzioni) in realtà fa riferimento ad un dispositivo di I/O (**I/O PROGRAMMATO**)

4.6 Interconnessione tra moduli

Le informazioni elaborate da un calcolatore elettronico prendono in considerazione delle stringhe binarie che hanno il significato di operando, indirizzo o istruzione.

Una stringa binaria, o parola (word), è una sequenza di bit di dimensione prefissata che deve essere considerata come unità indivisibile ed è stabilita a priori dal progettista dell'elaboratore.

Le singole cifre costituenti una parola sono memorizzate in latch e l'insieme risultante è un componente denominato registro (a volte i termini parola e registro si considerano equivalenti).

Il modo più semplice per realizzare un registro è quello di utilizzare n celle di memoria ed almeno due linee: una (write, W) per selezionare simultaneamente le n celle che compongono la parola e consentire la loro sovrascrittura con nuovi valori; mentre l'altra è di azzeramento (clear, C) del registro, cioè impostando, o 'pulendo', il contenuto di ogni latch con il valore 0.

Il transito di informazione è consentito dai sistemi di interconnessione, cioè delle reti che sono in grado di trasferire, o meglio duplicare, l'informazione contenuta nei registri. L'**interconnessione** punto a punto effettua il trasferimento della parola contenuta in un registro sorgente, Rs, a un registro destinazione, Rd.

Tutti gli n latch di Rs (linee di uscita) sono legati agli n latch (linee di entrata) di Rd ovviamente predisponendo una linea (transfer o write) di controllo che indica, con il comando 1, il trasferimento di informazione (la sovrascrittura) e con 0 la conservazione del valore corrente nel registro destinazione.

Il **multiplexer** è la rete d'interconnessione che consente il trasferimento tra m registri sorgenti e un registro destinazione prefissato.

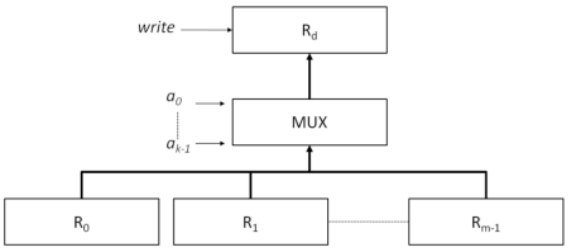


Figura 13: MUX

Il **demultiplexer** è la rete di interconnessione fatta per favorire il trasferimento tra un registro sorgente e uno degli m registri destinatari.

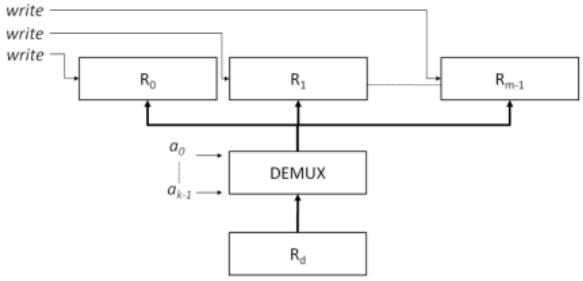


Figura 14: DEMUX

Le **reti mesh** sono reti in grado di interconnettere tra loro m registri, o più in generale m componenti.

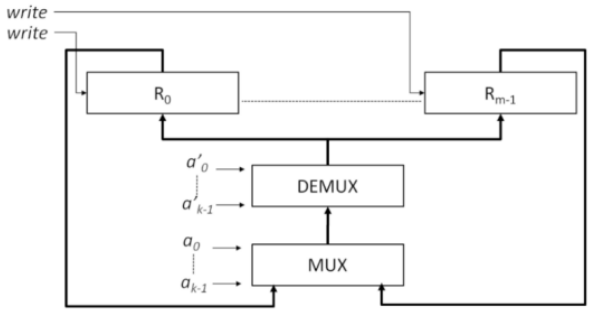


Figura 15: Mesh

Il **bus** è un fascio di k (di solito è uguale o maggiore alla dimensione del registro) linee. Per il trasferimento è sufficiente attivare la linea di ingresso di selezione (s) del registro sorgente e quella del registro destinazione.

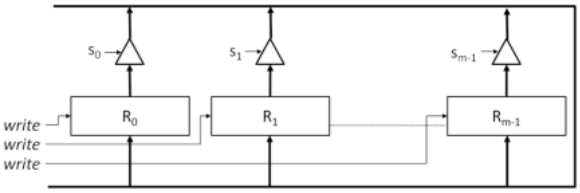


Figura 16: Enter Caption

Il bus sfrutta un buffer tristate, un dispositivo usato per permettere a più porte logiche di pilotare la stessa uscita, generalmente un bus. Se la linea S ha carica positiva si consente il passaggio dei dati, da IN a OUT, altrimenti si inibisce il trasferimento.

4.6.1 Interconnessione bus

I primi elaboratori avevano un unico bus chiamato anche bus di sistema. Esso era composto dai 50 ai 100 fili paralleli di rame che si inserivano nella scheda madre e i cui connettori erano distanziati a intervalli regolari per permettere l'inserimento di memorie e schede di I/O. Attualmente si usano più bus (multi bus): uno specifico tra la CPU e la Memoria Centrale e (almeno) un altro bus per le periferiche.

4.6.2 Interconnessione bus Master/Slave

Alcune periferiche che si collegano al bus sono attive (**master**) e possono iniziare un trasferimento dati, mentre altre sono passive (**slave**) e restano in attesa di una richiesta.

Quando il processore ordina al controllore di un disco di leggere o di scrivere un blocco, svolge il ruolo di master, e il controllore del disco quello di slave. Successivamente però il controllore del disco fa da master nel momento in cui ordina alla Memoria Centrale di accettare le parole che sta leggendo dal disco. La Memoria Centrale non può mai svolgere la funzione di master.

4.6.3 Interconnessione bus multipli e bus multiplexato

Il bus consente il transito di operandi, dati e controlli. Il numero di linee che costituisce il bus influenza la progettazione della macchina (costi, organizzazione topologica,...).

Per aggirare il problema di bus mulitipli si può usare un bus multiplexato. In questa architettura invece di tenere separate le linee d'indirizzo e quelle dei dati, si utilizza un certo numero di linee per entrambi: all'inizio di un'operazione sul bus le linee sono utilizzate per gli indirizzi, mentre in seguito vengono impiegate per i dati.

ES.: nel caso di una scrittura in memoria le linee d'indirizzo devono essere impostate ai valori corretti e propagate fino alla memoria prima di spedire i dati sul bus.

4.6.4 Interconnessione bus: arbitraggio centralizzato

Nel caso di un solo bus e di due o più dispositivi che richiedono contemporaneamente l'uso del bus si ricorre ad un arbitraggio del bus. L'arbitraggio può essere centralizzato o decentralizzato.

Arbitraggio centralizzato: un arbitro del bus (contenuto nella CPU o esterno ad esso) determina chi è il prossimo dispositivo.

L'arbitratore del bus, nel caso più semplice sfrutta la tecnica daisy chaining: ha un'unica linea di richiesta (non sa quanti e quali dispositivi hanno richiesto il bus, ma solo che c'è o non c'è almeno una richiesta)

L'arbitratore abilita l'uso della linea. Il dispositivo di I/O più vicino verifica se ha richiesto l'uso del bus: se si, si impossessa del bus e non consente di trasmettere oltre il segnale di concessione. Se invece non ha fatto richiesta, propaga la concessione sulla linea in direzione del prossimo dispositivo

Per evitare che la scelta ricada sempre sulla distanza e non sul tipo di dispositivo si possono usare delle linee di priorità

Nei sistemi in cui la memoria è collegata al bus principale, la CPU deve competere con tutti i dispositivi di I/O praticamente a ogni ciclo. Di solito la CPU ha la priorità più bassa rispetto gli I/O. I dispositivi di I/O sono obbligati ad acquisire il bus molto velocemente, pena la perdita dei dati in arrivo. I dischi che ruotano ad alte velocità, per esempio, non possono aspettare

4.6.5 Interconnessione bus: arbitraggio decentralizzato

Nell'arbitraggio decentralizzato del bus si usano più linee di richiesta, ciascuna con la propria priorità. Quando un dispositivo vuole utilizzare il bus invia un segnale lungo la linea di richiesta.

Tutti i dispositivi monitorano tutte le linee di richiesta in modo che alla fine di ciascun ciclo di analisi del bus ognuno di loro può sapere se era il richiedente con priorità più elevata e se quindi ha diritto a utilizzare il bus durante il ciclo successivo.

Rispetto al metodo centralizzato questo schema di arbitraggio richiede un maggior numero di linee di bus, ma evita il potenziale costo dell'arbitratore. Un altro limite è che il numero di dispositivi non può superare il numero delle linee di richiesta.

4.7 Macchina Harvard

La prima data ufficiale in cui si realizzò un modello di elaboratore elettronico fu il 1944 con la "**macchina di Harvard**", dal nome del college in cui si trovava il gruppo di lavoro che l'aveva ideata (fu adottata dall'elaboratore MARK).

La macchina di Harvard era costituita da una Unità di Calcolo, una Unità di Controllo, una Memoria delle Istruzioni, una Memoria Dati ed un modulo per i Dispositivi di input ed output, opportunamente collegati per consentire un flusso di comandi e di controlli che ne permettevano il funzionamento e colloquio reciproco e lo svolgimento di una istruzione ad un colpo di clock.

La "macchina di Harvard" fu migliorata nel 1982 con un set appropriato, multibus e una serie di registri ad uso generale che consentivano di eseguire ciascuna istruzione di lunghezza fissa a 32bit in un solo colpo di clock.

Il progetto iniziò nel 1981 per opera di John L. Hennessy dell'Università di Stanford.

Realizzazione di una architettura di tipo RISC (poche istruzioni e pochi modi di indirizzamento) e in grado di realizzare la tecnica della canalizzazione (pipeline).

Suddivisione della Memoria Centrale in due parti fisiche (Memoria Istruzioni e Memoria Dati) per garantire l'esecuzione di una istruzione in un solo ciclo di clock.

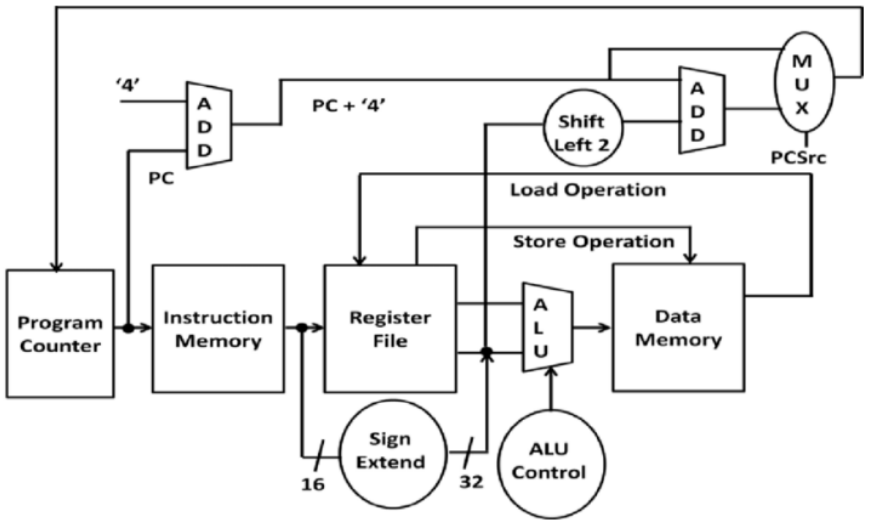


Figura 17: Enter Caption

5 Dal codice sorgente al codice esguibile

L'esecuzione di un programma è il punto di arrivo di una sequenza di azioni che nella maggior parte dei casi iniziano con la scrittura di un programma in un linguaggio simbolico di alto livello.

Le azioni principali che compongono tale sequenza nel caso si parta da un linguaggio ad alto livello sono quelle che vedono in gioco il **compilatore**, l'**assemblatore** e il **collegatore**

Alcuni calcolatori raggruppano queste azioni per ridurre il tempo di traduzione, ma concettualmente tutti i programmi compilati passano sempre attraverso le fasi mostrate.

5.1 Assemblatore

L'assemblatore converte un programma assembly in un file oggetto, che è una combinazione di istruzioni in linguaggio macchina, di dati e di informazioni necessarie a collocare le istruzioni in memoria nella posizione opportuna.

Un programma assembly è tradotto in una sequenza di istruzioni (opcode, indirizzi, costanti, ecc.) attraverso il processo di assemblaggio (assembler) costituito da due passi logici successivi ed in parte indipendenti :

1. il programma assembly è letto sequenzialmente, si identificano le istruzioni e i loro operandi, si calcola la lunghezza e si assegna un indirizzo (relativo) a ciascuna istruzione; inoltre, quando è letto un simbolo (un indirizzo simbolico, cioè una etichetta), nome e indirizzo sono inseriti in una tabella dei simboli (symbol table): nome e indirizzo di un simbolo possono essere inseriti nella symbol table in momenti diversi se un simbolo è usato prima di essere definito.

2. il programma assembly è letto sequenzialmente, a tutti i simboli è sostituito il valore numerico corrispondente presente nella symbol table, a tutte le istruzioni e ai relativi operandi ancora in forma simbolica è sostituito il valore numerico corrispondente (opcode, ecc...).

Il processo di assemblaggio prende il nome di assegnazione interna delle locazioni (internal relocate symbol o internal reference).

5.2 Compilatore

Il **compilatore** trasforma, dopo un controllo sintattico, il programma scritto in un linguaggio ad alto livello in uno in linguaggio assembly, cioè in una forma simbolica che il calcolatore è in grado di capire ma, ancora, non eseguire.

Durante la generazione del codice, il compilatore effettua il riordino delle istruzioni cioè quali istruzioni sono trasmesse al processore e in quale ordine (utile nella canalizzazione o per il calcolo parallelo). Infine il compilatore ottimizza il codice: toglie istruzioni inutili o variabili non utilizzate.

In Linguaggio Assembly (SPIM) - Esempio

```
.text
.globl main

main:
    lw $a0, base    # caricamento valore
    lw $a1, espo    # caricamento valore
    jal pow         # salto a funzione
    sw $a2, ris     # spostamento risultato in memoria
    li $v0, 10
    syscall

pow:
    li $t0, 0       # inizializzazione contatore
    li $t1, 1       # inizializzazione risultato temporaneo
    move $t3, $a0

ciclo:
    bge $t0, $a1, fine    # confronto contatore-esponente
    mul $t1, $t1, $t3     # moltiplicazione per la base
    addi $t0, 1           # incremento contatore
    j ciclo               # salto

fine:
    move $a2, $t1
    jr $ra              # ritorno a funzione

.data                   # dichiarazione variabili
base: .word 2
espo: .word 3
ris: .word 0
```

5.3 Disposizione dei file oggetto in memoria

I file oggetto sono suddivisi e disposti in memoria di solito in sei sezioni distinte:

1. **object file header**: descrive la dimensione e la posizione delle altre sezioni del file oggetto;
2. **text segment**: contiene le istruzioni in linguaggio macchina;
3. **data segment**: contiene tutti i dati che fanno parte del programma;
4. **relocation information**: identifica le istruzioni e i dati che dipendono da indirizzi assoluti e che dovranno essere rilocati dal linker
5. **symbol table**: contiene i simboli che non sono ancora definiti, ad esempio le etichette che fanno riferimento a moduli esterni;
6. **debugging information**: contiene informazioni per il debugger.

In più si riserva uno spazio nel quale può avvenire uno scambio di dati (STACK).

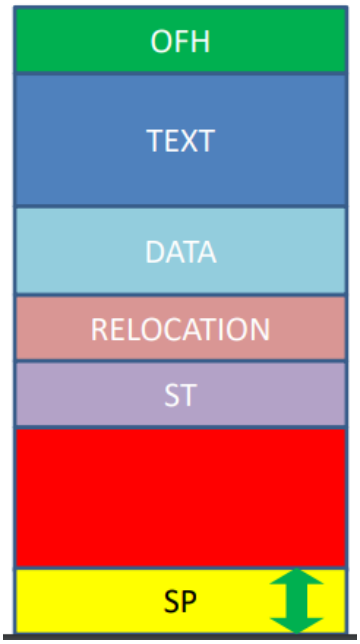


Figura 18: **Disposizione dei file oggetto in memoria**

5.4 Collegatore (Linker)

Quando un programma simbolico è costituito da più moduli contenuti in diversi file sorgenti il processo di traduzione (compilazione e assemblaggio) è ripetuto per ciascun modulo.

I **file oggetto** (object) risultanti devono essere **collegati** (linked) opportunamente tra di loro all'interno di unico file eseguibile, che solo allora può essere caricato in memoria.

Per ogni modulo tradotto separatamente l'indirizzo iniziale è lo stesso: è compito del linker modificare gli indirizzi di ciascun modulo in modo che non ci siano sovrapposizioni. La traslazione dell'indirizzo di ogni istruzione in ciascun modulo permette di unire tutti i moduli ma non è sufficiente, infatti: è necessario traslare in maniera consistente anche tutti gli indirizzi (assoluti) che compaiono come operandi.

Per ogni riferimento da parte di un modulo a un indirizzo di un altro modulo è necessario calcolare coerentemente l'**indirizzo esterno** (riferimento esterno o external reference). Esempi in questo senso sono le **variabili globali**. (che devono essere viste da tutti i moduli) e le chiamate tra procedura appartenenti a moduli diversi.

Per questo, il linker costruisce una tabella dei moduli grazie alla quale è possibile procedere alla rilocazione e al calcolo dei riferimenti esterni a ciascun modulo.

Infine il linker produce un file eseguibile che di norma ha la stessa struttura di un file oggetto, ma non contiene riferimenti non risolti.

5.5 Caricatore

Una volta che il file eseguibile è memorizzato sul supporto di massa (generalmente il disco magnetico), il caricatore, o loader, (un programma afferente al sistema operativo) può caricarlo in memoria per l'esecuzione e quindi effettuare:

1. la lettura dell'intestazione del file eseguibile per determinare la dimensione dei segmenti testo (istruzioni) e dati.
2. la creazione un nuovo spazio di indirizzamento, grande a sufficienza per contenere istruzioni, dati e stack.
3. la copia delle istruzioni e dei dati dal file oggetto al nuovo spazio di indirizzamento.
4. la copia sullo stack degli eventuali argomenti del programma.
5. l'inizializzazione dei registri della CPU.
6. l'inizio dell'esecuzione a partire da una direttiva di inizio che copia gli argomenti del programma dallo stack agli opportuni registri e che chiama la funzione main() o la direttiva di inizio (BEGIN); fino ad una direttiva di terminazione (END).

5.6 Interprete

Un interprete non svolge le operazioni di compilazione e di assemblaggio, ma traduce le istruzioni in linguaggio macchina (memorizzando su file il codice oggetto per essere eseguito dal processore) effettuando solamente una analisi sintattica prima della traslitterazione.

L'uso di un interprete comporta una minore efficienza durante l'esecuzione del programma (run-time); un programma interpretato, in esecuzione, richiede più memoria ed è meno veloce, a causa dell'overhead (maggior numero di operazioni da compiere) introdotto dall'interprete stesso.

Durante l'esecuzione, l'interprete deve infatti analizzare le istruzioni a partire dal livello sintattico, identificare le azioni da eseguire (eventualmente trasformando i nomi simbolici delle variabili coinvolte nei corrispondenti indirizzi di memoria), ed eseguirle; mentre le istruzioni del codice compilato, già in linguaggio macchina, sono caricate e istantaneamente eseguite dal processore.

L'uso di un interprete consente all'utente di agire sul programma in esecuzione sospendendolo, ispezionando o modificando i contenuti delle sue variabili, e così via, in modo spesso più flessibile e potente di quanto si possa ottenere, per il codice compilato.

6 Architettura MIPS

Il **MIPS** (*Microprocessor without interlocked pipeline stages*) è un'architettura informatica per microprocessori RISC sviluppata da MIPS Computer Systems Inc (oggi MIPS Technologies Inc).

Progetto sviluppato nel 1981 da John L. Hennessy dell'Università di Stanford.

Suddivisione della **Memoria Centrale in due parti fisiche** (Memoria Istruzioni e Memoria Dati) per garantire l'esecuzione di una istruzione in un solo ciclo di clock.

Realizzazione di una architettura di tipo RISC e in grado di realizzare la tecnica della canalizzazione (pipeline).

Istruzioni RISC sono caratterizzate da: **semplicità** (nei primi modelli le moltiplicazioni e le divisioni tra interi furono realizzate con somme e sottrazioni successive) e **pochi modi di indirizzamento** (per lo più elementari come LOAD e STORE) per garantire l'esecuzione di ciascuna istruzione in un solo ciclo di clock.

L'esecuzione in un solo ciclo di clock di una istruzione, la suddivisione delle varie unità funzionali e la loro sincronizzazione (ottenuta con l'interposizione di blocchi, un insieme di registri e linee di controllo che sorvegliano il completamento delle varie istruzioni) consentono un **pipeline regolare** e quindi una **prestazione della macchina più efficiente** in termine di quantità di calcoli in unità temporale.

Nel 1984 Hennessy fonda la MIPS Computer Systems.

Nel 1985 la società presenta il **processore R2000** con **lunghezza della parola a 32bit** nel 1988 è commercializzato il modello R3000 (riduzione della dimensione e del numero dei transistori; frequenza: 20Mhz, 33mhz a 35Mhz). Entrambi i processori sono impiegati come CPU delle workstation della società Silicon Graphics. Nel 1991 MIPS presenta **R4000**, il **suo primo processore a 64 bit**. Acquisizione della società da parte di Silicon Graphics (MIPS Technologies).

Agli inizi degli anni Novanta, MIPS Tencologies invade il mercato dei microprocessori grazie al basso prezzo e le ottime prestazioni di calcolo offerte dalla canalizzazione.

Nel 1997 il MIPS supera il numero di processori venduti da Motorola (uno dei leader del mercato mondiale).

Nel 1999 MIPS annuncia la possibilità di acquistare la licenza per due processori base: **MIPS32** a 32 bit e **MIPS64** a 64 bit.

Nascita della SandCraft e sviluppo del processore R7100 (eseguiva istruzioni fuori ordine).

Creazione della SiByte e produzione del modello SB-1250, uno dei primi processori systems-on-a-chip (SOC) ad alte prestazioni basato su architettura MIPS.

Fondazione di Alchemy Semiconductor (in seguito acquisita da AMD), che produce il processore di tipo SOC dal nome Au-1000 che necessita di un basso consumo energetico.

Nella prima metà del XXII secolo l'architettura MIPS trova grossa diffusione nell'ambito dei sistemi embedded, dei device di Windows CE e nei router di Cisco e anche nelle console Nintendo 64, Sony PlayStation, PlayStation 2 e PlayStation Portable.

6.1 Elementi Essenziali

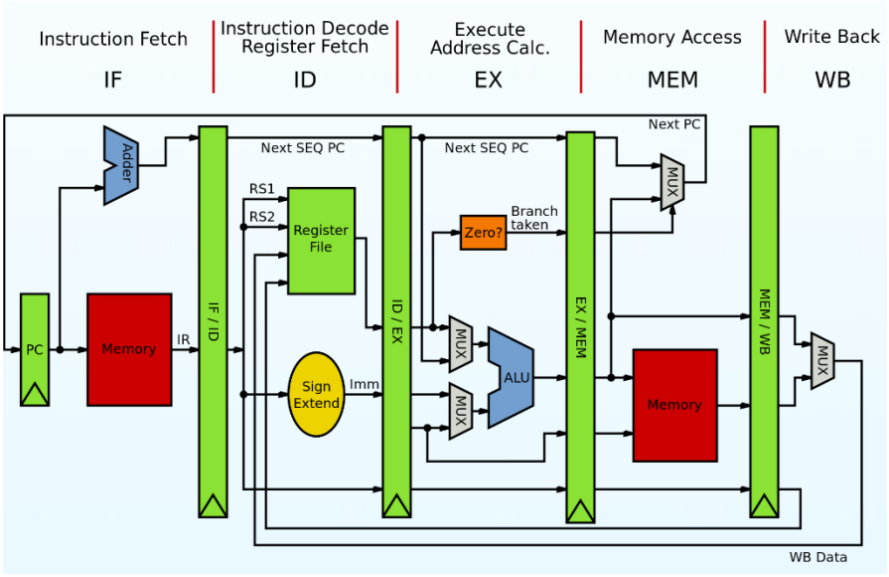


Figura 19: **Componenti schema MIPS**

6.2 I registri

I **registri** sono locazioni di memoria in cui ospitare temporaneamente degli operandi/indirizzi/istruzioni.

REGISTRI TEMPORANEI NON PRESERVANTI: si azzerano dopo un salto a sub-routine.

```
1 $t0 $t1 $t2 $t3 $t4 $t5 $t6 $t7 $t8 $t9
```

REGISTRI TEMPORANEI PRESERVANTI: mantengono sempre i valori.

```
1 $s0 $s1 $s2 $s3 $s4 $s5 $s6 $s7 $s8 $s9
```

REGISTRI TEMPORANEI PER LE SUB ROUTINE:

```
$a0 $a1 $a2 $a3  #Parametri di ingresso della sub-routine
$v0 $v1          #Risultati della sub-routine
```

REGISTRI PER I NUMERI REALI (NUMERI IN VIRGOLA MOBILE, floating point)

```
$fp0 ------------------------- $fp31
```

6.3 ALU

La **ALU** è il modulo nel quale è presente la circuiteria utile per svolgere operazioni logiche – aritmetiche (adder, shifter, comparator...).

Il **MIPS** è dotato di un coprocessore matematico, visto come una unità di I/O con un set di istruzioni specifico, utile per svolgere operazioni aritmetiche con operandi in virgola mobile (espressi in singola e doppia precisione). Questa caratteristica è presente anche nel **simulatore MARS**.

Istruzione	Valore in RAX
movb 100,%RAX	0000000000000010
movw 100,%RAX	0000000000003210
movl 100,%RAX	0000000076543210
movq 100,%RAX	fedcba9876543210

Figura 20: Componenti – ALU e Coprocessore Matematico

6.4 Coprocessore Matematico

Il **coprocessore matematico** opera su numeri reali - rappresentati in Virgola Mobile Singola Precisione e Doppia Precisione - siti in memoria oppure numeri interi derivati dal calcolo dell'ALU e stipati nei registri (o in memoria) previa conversione nel formato IEEE754 mediante apposite funzioni di trasformazione.

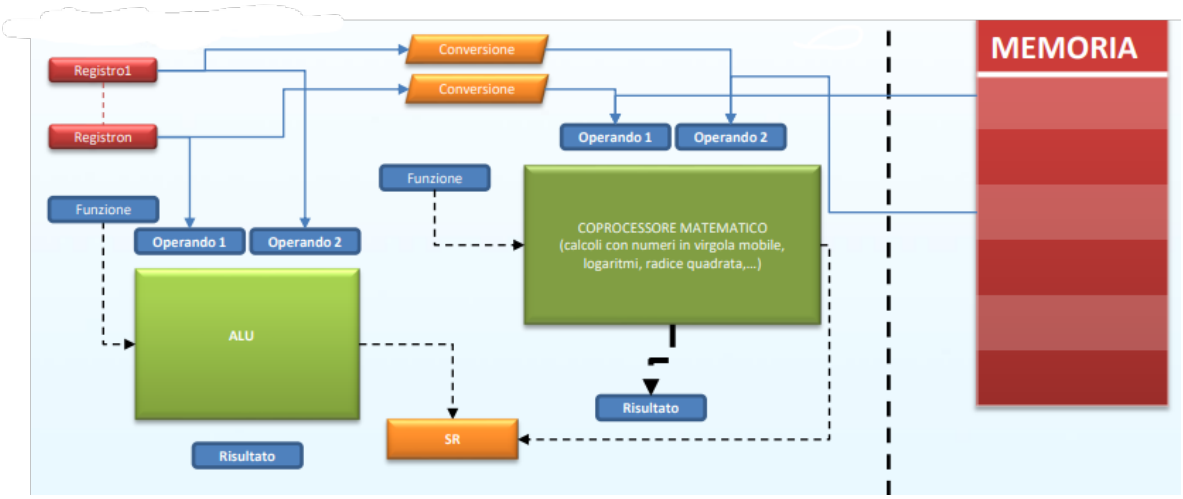


Figura 21: **Coprocessore**

6.5 La memoria

Il MIPS ha **due memorie distinte**:

- **Memoria Istruzioni**: area nella quale sono stipate le istruzioni afferenti ad un programma, e una serie di informazioni ausiliarie (stack, kernel S.O., . . .)
- **Memoria Dati**: area nella quale sono stipati i dati utilizzati dal programma.

6.6 Componenti – I/O

I **dispositivi di Ingresso e di Uscita** (Input Output device o I/O device) permettono l'interazione con il mondo esterno. Il MIPS interagisce con molte periferiche (scheda audio, video, terminale video, tastiera).

Finestra di input per l'immissione dei dati da tastiera

```
1 li $v0,5       #servizio di lettura di un intero da tastiera
2 syscall        #chiamata di sistema
3 move $t0,$v0   #spostamento del valore letto da tastiera
4           #residente in $v0 dopo la syscall
```

Finestra di output (mostrata dal simulatore) per la visualizzazione dei risultati.

```
move $a0,$t0   #spostamento del valore intero da stampare da $t0 a $a0
li $v0,1       #servizio di stampa di un intero
syscall        #chiamata di sistema
```

Esempio 1 (Interazione con i dispositivi di I/O)

```
.text
.globl main
main:
li $v0,5       #servizio di lettura di un intero da tastiera
syscall        #chiamata di sistema
move $t0,$v0   #spostamento del valore letto da tastiera residente
               in $v0 dopo la syscall
add $t0,$t0,1  #calcolo del valore successivo
move $a0,$t0   #spostamento del valore da stampare da $t0 a $a0
li $v0,1       #servizio di stampa di un intero
syscall        #chiamata di sistemali
$v0,10
syscall
```

7 Architettura funzionale: le istruzioni

Una **istruzione** è una stringa binaria che indica all'elaboratore elettronico dei compiti da svolgere. Una istruzione è suddivisa in sottostringhe denominate campi. La suddivisione in campi individua il formato dell'istruzione.

Ricordiamo che i campi principali di un'istruzione sono:

- Il **codice operativo** (o *OPCODE*), che specifica il tipo di operazione da eseguire (addizione, trasferimento dati, ...).
- L'**operando**, che indica il dato su cui devono essere effettuate le operazioni indicate dal codice operativo. L'operando può essere un valore numerico (come avviene nell'indirizzamento immediato) o, come spesso accade, si ha un riferimento: cioè, un indirizzo di memoria in cui è immagazzinato un operando (indirizzamento diretto) o una etichetta che specifica un registro.

Il **formato a lunghezza fissa** prevede un insieme di istruzioni (instruction set) con una dimensione predefinita (una sottoclasse di questa sono le istruzioni a referenziamento implicito, cioè quelle dotate di solo opcode).

In alternativa, un set di istruzioni può avere una **lunghezza variabile**: in relazione al tipo di istruzione cambia la dimensione. Un istruzione a lunghezza variabile di solito ha

i bit in eccesso – cioè non rappresentabili nella parola - ospitati nella parola successiva (richiede più accessi in memoria).

I processori intel X86 hanno un formato a lunghezza variabile. Dopo l'opcode ci sono dei campi che specificano quanti bit appartengono al campo **MODE**.

Istruzione	Valore in RAX
movb 100,%RAX	0000000000000010
movw 100,%RAX	0000000000003210
movl 100,%RAX	0000000076543210
movq 100,%RAX	fedcba9876543210

Il MIPS ha un formato a lunghezza fissa a 32 bit. Qualora si usi un indirizzamento assoluto (o immediato) in cui il riferimento (o l'operando) richieda più di 16bit l'istruzione è suddivisa in due istruzioni elementari che consentono il riempimento dell'operando/indirizzo in un registro.

Le istruzioni sono eseguite quando sono scritte in **linguaggio macchina** (nei primi elaboratori esisteva solo questo tipo di linguaggio).

7.1 Il linguaggio assemblativo

Il programmatore ricorre ad una rappresentazione simbolica delle istruzioni, utilizzando codici mnemonici che possono essere interpretati in maniera più comoda rispetto alle sequenze binarie: **istruzioni assembly**.

La sintassi di una istruzione assembly è costituita da:

- un indirizzo, dove risiede l'istruzione in memoria (spesso omesso, perché impostato dall'assemblatore).
- un'etichetta (opzionale).
- un'istruzione composta da: **codice mnemonico** che descrive l'struzione con pochi caratteri e il **modo di indirizzamento** cioè i dati su cui deve operare o il luogo dove essi risiedono.
- i commenti, indispensabili per la comprensione del codice.

Il legame che intercorre tra **istruzione macchina** e **istruzione assembly** è di uno a uno, nel senso che ad ogni istruzione macchina corrisponde una ed una sola istruzione assembly. Per comodità molti linguaggi assembly utilizzano delle pseudoistruzioni ovvero delle istruzioni che sono composte da una o più istruzione assembly elementare.

7.1.1 Le macro

Un linguaggio assembly consente di definire delle **macro**: una macro sostituisce una serie di istruzioni. Ogni volta che si richiama la macro l'assemblatore riscrive le istruzioni definite nella macro.

Prima della fase di assemblaggio a doppia passata si effettua un pre-assemblaggio dove accadono queste operazioni: si risolvono le macro, le pseudo istruzioni, eventuali file esterni e si inizializzano le direttive.

7.1.2 Esecuzione istruzioni logiche aritmetiche e codici

Ad ogni tempo, dettato dal **clock**, l'elaboratore esegue una istruzione. Ogni istruzione logico-aritmetica, produce dei bit, definiti **flags** (codici di condizione, o condition code), che saranno implicitamente memorizzati nel registro di stato (PSW, processor status word, o STATUS register). I **Condition Codes** svolgono un ruolo fondamentale per le istruzioni di salto condizionato.

I principali flags sono:

- **C - Carry**: Individua il trabocco ed è impostato ad 1 se l'ultima operazione effettuata dall'ALU ha prodotto un riporto (addizione) o un prestito (sottrazione) a sinistra del bit più significato del risultato, 0 altrimenti.
- **N - Negative**: impostato ad 1 se l'ultima operazione effettuata dall'ALU ha prodotto un risultato negativo, 0 altrimenti. Ovvero Negative è una copia del bit più significativo del risultato.
- **Z - Zero**: impostato ad 1 se l'ultima operazione effettuata dall'ALU è nulla, 0 altrimenti.
- **W - Overflow**: impostato ad 1 se l'ultima operazione effettuata dall'ALU ha superato la capacità di rappresentazione data dalla lunghezza della parola, 0 altrimenti.
- **P - Parity**: impostato ad 1 se l'ultima operazione effettuata dall'ALU ha dato un risultato con un numero pari di 1; 0 altrimenti.

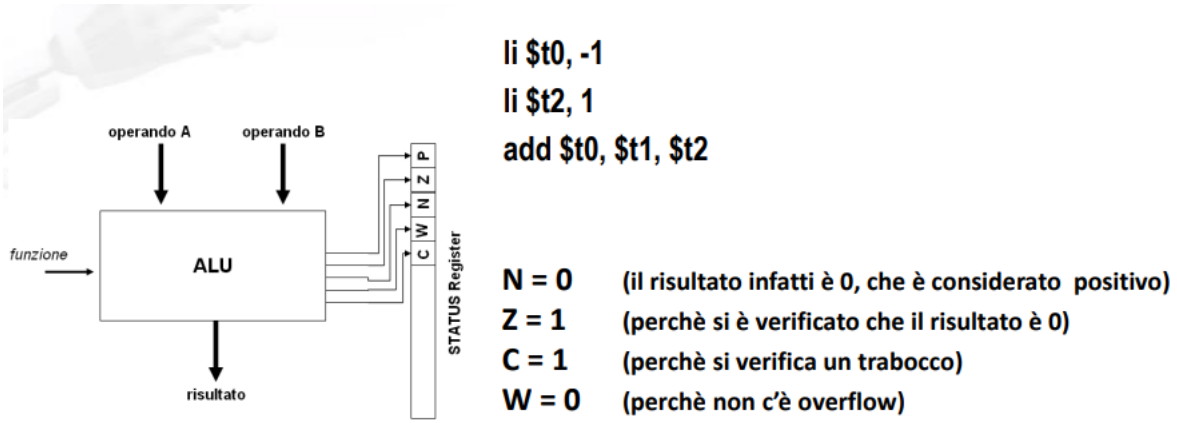


Figura 22: Codici condizione

7.2 Le classi di Istruzioni

Possiamo riassumerle in:

1. istruzione di spostamento dati
2. istruzioni logico ed aritmetiche
3. istruzioni di salto: condizionato, non condizionato, a funzione (o subroutine), trap
4. istruzione di controllo macchina

7.2.1 ISTRUZIONI DI SPOSTAMENTO

Le **istruzioni per lo spostamento** dei dati servono a ricopiare un dato da una sorgente ad una destinazione e cioè da: memoria a registro, registro a memoria, registro a registro, memoria a memoria.

Le istruzioni di spostamento possono interessare la CPU e la Memoria (LOAD, STORE, PUSH e POP) o solamente i registri nella CPU (MOVE). Il contenuto della destinazione non si modifica rispetto alla sorgente.

CODICE	OPERANDI	Commento
LOAD	<Sorgente><Destinazione>	Legge l'operando dalla sorgente (memoria) e lo copia nella destinazione (tipicamente un registro)
STORE	<Sorgente><Destinazione>	Legge l'operando dalla sorgente (tipicamente un registro) e lo copia nella destinazione (una cella di memoria esplicitata)
MOVE	<Sorgente><Destinazione>	Sposta il contenuto di un registro Sorgente ad un registro Destinazione
PUSH	<Sorgente>	Sposta un operando da una Sorgente(un registro o una cella in memoria) in cima allo stack/pila Equivale a STORE sorg,-($SP)
POP	<Destinazione>	Sposta un operando dalla cima dello stack/pila in una Destinazione (un registro o una cella in memoria) Equivale a LOAD ($SP)+,dest

7.2.2 ISTRUZIONI LOGICHE-ARITMETICHE

Istruzioni aritmetiche: consentono di effettuare le operazioni su numeri interi binari rappresentati in complemento a due (in alcuni casi le ALU possono svolgere operazioni anche con numeri in virgola mobile, ma spesso queste operazioni sono demandate ad una unità di calcolo – il coprocessore matematico - che è visto come un dispositivo di I/O). Le funzioni di base offerte dalla ALU sono il complemento, la comparazione e l'addizione; operazioni come la moltiplicazione o la divisione e la sottrazione possono essere ricavati sfruttando algoritmi che impiegano le operazioni elementari sopra citate. Le istruzioni aritmetiche sono eseguite dall'ALU la quale produce due linee di uscita: **il risultato dell'operazione** e un vettore di bit, o **flags** (condition code), che viene implicitamente caricato nello Status Register.

CODICE	OPERANDI	Commento
ADD	<Destinazione><Sorgente><Sorgente>	Legge gli operandi dalla sorgente (memoria/registri), effettua la somma ed il risultato è trasferito nella destinazione (tipicamente un registro).
CMP	<Destinazione><Sorgente><Sorgente>	Legge gli operandi dalla sorgente (memoria/registri), effettua la comparazione ed il risultato è trasferito nella destinazione (tipicamente un registro)
NEG	<Destinazione><Sorgente>	Legge l'operando dalla sorgente (memoria/registro), effettua la negazione ed il risultato è trasferito nella destinazione (tipicamente un registro)

Istruzioni logiche: Le operazioni logiche permettono l'esecuzione delle più importanti operazioni definite nell'algebra booleana su stringhe binarie. Come per le operazioni aritmetiche, anche in questo caso, le operazioni avvengono per tutti i bit in posizione corrispondente. La sintassi è simile alle istruzioni aritmetiche e l'operando sorgente può essere in una locazione di memoria, in un registro, o un dato costante (residente dopo l'istruzione); mentre l'operando destinazione è di solito un registro. Anche in questo caso i passi elementari che costituiscono la fase di decodifica ed esecuzione sono analoghi per tutte le istruzioni. Le istruzioni logiche permettono di modificare alcuni bit di un registro, di esaminare il loro valore o di settarli tutti a 0 o 1.

CODICE	OPERANDI	Commento
AND	Registro <Sorgente>, <Sorgente>	Legge gli operandi dalla sorgente (memoria/registri) ed effettua l'AND riportando il risultato in un registro
OR	Registro <Sorgente>, <Sorgente>	Legge gli operandi dalla sorgente (memoria/registri) ed effettua l'OR riportando il risultato in un registro
XOR	Registro <Sorgente>, <Sorgente>	Legge gli operandi dalla sorgente (memoria/registri) ed effettua l'XOR riportando il risultato in un registro
NOT	Registro <Sorgente>	Legge l'operando dalla sorgente (memoria/registri) ed effettua l'NOT riportando il risultato in un registro

Le istruzioni di rotazione e shift: operano su un solo dato posto in un registro. Queste istruzioni cambiano l'ordine dei bit nel registro ed hanno un significato:

- **logico**: per effettuare lo scorrimento dei bit del registro nella direzione e nel numero di posizioni specificati. Il bit C (carry o trabocco) dello Status Register riceve l'ultimo bit che fuoriesce dal registro;
- **aritmetico**: è opportuno ricordare che uno shift a destra equivale a dividere l'operando per 2 k (con k il numero di posizioni scorse), mentre uno scorrimento verso sinistra equivale a moltiplicare l'operando per 2k (con k il numero di posizioni scorse)

CODICE	OPERANDI	Commento
SL	Registro, k	Shifit a sinistra di k posti del registro
SR	Registro, k	Shifit a destra di k posti del registro
ROL	Registro, k	Ruota a sinistra di k posti del registro
ROR	Registro, k	Ruota a destra di k posti del registro

Istruzioni logico-aritmetiche MIPS: prevedono un OPECODE comune 000000 che individua la classe e poi una sottodivisione negli ultimi 6 bit che specifica il tipo di funzione.

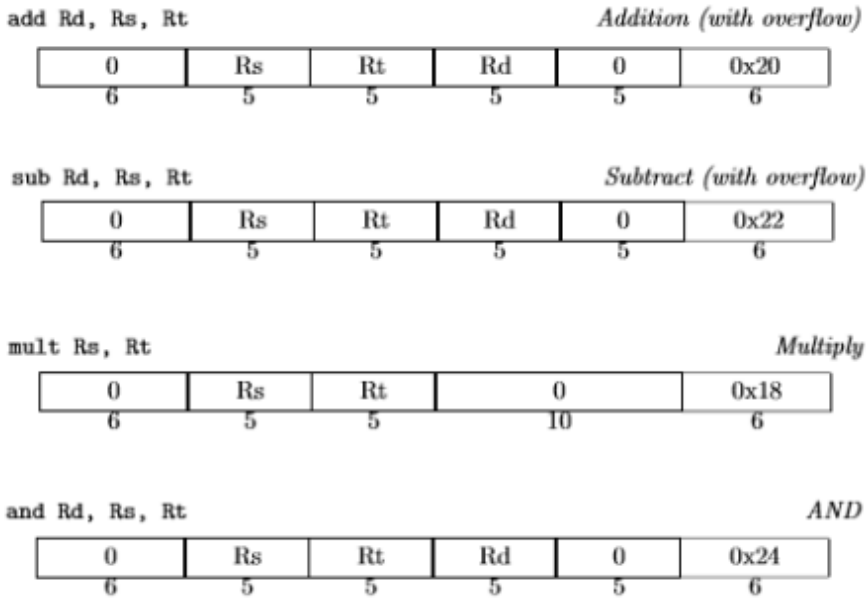


Figura 23: Enter Caption

Esistono inoltre istruzioni, con **referenziamento implicito**, che consento di operare sui singoli bit del Registro di Stato.

CODICE	Commeto	CODICE	Commeto
CLRC	Imposta a 0 il flag C	**SETC**	Imposta a 1 il flag C
CLRN	Imposta a 0 il flag N	**SETN**	Imposta a 1 il flag N
CLRZ	Imposta a 0 il flag Z	**SETZ**	Imposta a 1 il flag Z
CLRW	Imposta a 0 il flag W	**SETW**	Imposta a 1 il flag W

7.2.3 ISTRUZIONI DI SALTO

Le istruzioni di salto individuano una classe particolare in quanto non agiscono direttamente sui dati, ma sono utilizzate per modificare l'ordine sequenziale di esecuzione delle istruzioni del programma stesso o uno esterno.

Le istruzioni di salto si dividono in:

- salto all'interno dello stesso programma. **Condizionato**: il salto viene eseguito in base ad una certa condizione fissata dal programmatore (Branch). **Incondizionato**: il salto viene sempre eseguito (Jump).
- salto ad un altro programma: salto a subroutine (salto a sottoprogramma)
- trap (o interruzioni software)

Le **istruzioni di salto** sono fondamentali perché rompono la sequenzialità offrendo la possibilità di effettuare scelte, cioè, prendere decisioni e perché consentono di eseguire più volte una parte di programma (es.: costrutto IF; ciclo while).

CODICE	OPERANDI	Commento
BEQZ	<Sorg1>, Indirizzo	Se l'operando contenuto in una sorgente (registro/memoria) è uguale a zero salta all'indirizzo specificato
BGT	<Sorg1>,<Sorg2>,Indirizzo	Legge gli operandi dalla sorgente (memoria/registri) e salta all'indirizzo se Sorg1 è maggiore della Sorg2
BLT	<Sorg1>,<Sorg2>,Indirizzo	Legge gli operandi dalla sorgente (memoria/registri) e salta all'indirizzo se Sorg1 è minore della Sorg2
J	Indirizzo	Salto incondizionato all'indirizzo specificato

L'**istruzione di salto a subroutine** (o chiamata a funzione) permette di saltare da un programma (il programma principale) ad un sottoprogramma, di eseguirlo e di tornare alla istruzione immediatamente successiva a quella di chiamata. L'utilizzo di subroutine è utile quando un determinato insieme di istruzioni deve essere eseguito più volte e per avere un codice più chiaro e compatto. Inoltre, le subroutine possono essere realizzate da terzi, essere scambiate e modificate ai propri fini.

CODICE	OPERANDI	Commento
JSR	Indirizzo	Salva il valore del PC incrementato nello Stack e salta all'indirizzo specificato che individua l'inizio del sottoprogramma
RET		Ritorna al programma principale ripristinando il valore del PC recuperato nello stack

Molto spesso però i **sottoprogrammi possono a loro volta chiamare altri programmi** e così via. Può avverarsi, cioè, un **annidamento** di subroutine (nested subroutine). In questo caso è fondamentale salvare i diversi indirizzi di ritorno.

La **gestione di funzioni ricorsive** o l'annidamento di funzioni avviene grazie all'utilizzo della **pila** (stack o canasta). Lo stack è una zona di memoria riservata per il passaggio di parametri e la memorizzazione di informazioni gestita nella modalità LIFO (Last in First Out): ovvero l'ultimo elemento immesso nella pila è anche il primo ad uscire.

7.3 Istruzioni I/O

Per interagire con i dispostivi di I/O si può ricorre ad un set di istruzioni dedicato (IO canonico) o riservare un'area di memoria agli scambi con i dispositivi di I/O ed operare con le istruzioni della macchina (IO programmato).

Le **istruzioni di comando** (o istruzioni di controllo macchina) non operano né sui dati né sui registri né interessano il contatore di programma, ma intervengono direttamente sullo stato della CPU. Le istruzioni di comando sono caratteristiche di ogni CPU: il loro numero può variare da poche unità, per macchine semplici, a decine per macchine complesse.

CODICE	Commento
HALT	Interruzione di sistema
NOP	Nessuna operazione. ***È utile per il Delay Slot del pipeling***
BREAK	Interruzione di programma

7.4 Esempio Pratico

Realizzazione di un elaboratore che svolga la sola funzione dell'elevamento a potenza di un numero (con esponente>0) e che utilizza istruzioni a dimensione fissa es.: $2^3 = 8$, $3^3 = 27$, $5^2 = 25$.

Realizzazione di un elaboratore che svolga la sola funzione dell'elevamento a potenza di un numero (con esponente>0) e che utilizza istruzioni a dimensione fissa.

Possibile implementazione:

```
1      LOAD $R0, BASE
2      LOAD $R1,ESPONENTE
3      LOAD $R2, UNO
4      LOAD $R3,MENOUNO
5  CICLO:
6      BEQZ $R1,FINE
7      MUL $R2,$R2,$R0
8      ADD $R1,$R1,$R3
9      JUMP CICLO
10 FINE:
11     STORE $R2, RISULTATO
```

Di cosa si ha bisogno:

- 4 registri enumerati da R0 a R3
- Una ALU che faccia 4 operazioni: moltiplicazione, somma, salto condizionato al valore zero, salto incondizionato
- Istruzioni di caricamento e archiviazione dati in memoria
- Spazio in memoria per archiviare i dati e gli operandi
- Numero di istruzioni: 6
- Registri: 4 (+ 1 di ausilio alla macchina trasparente al programmatore settato a 0 e non modificabile)

Domanda: quanto deve essere lunga la parola per indirizzare almeno 255 locazioni di memoria (126 per ospitare il programma e 127 per ospitare i dati)?

8 Modi di indirizzamento

Una istruzione macchina contiene generalmente una suddivisione in campi in cui una parte è **operazionale** (OPCODE o Codice operativo) e specifica la classe di istruzione da eseguire ed un secondo **campo indirizzo** (ADDRESS MODE o Campo Indirizzo o modo di indirizzamento) che indica quale è l'operando, cioè quale è il registro/i o la locazione di memoria che contiene l'operando sul quale si deve applicare l'istruzione.

Il luogo dove risiede l'operando può essere espresso in maniera **esplicita** oppure può essere omesso, in questo ultimo caso di parla di **modalità implicita**.

8.1 Implicito

In alcune macchine (es. Intel 8085) si utilizza **l'indirizzamento implicito**. In tale modo non sono esplicitati gli indirizzi dove risiedono gli operandi, ma questi ultimi si trovano in una posizione predeterminata (di solito degli accumulatori).

Grazie al **referenziamento implicito** è possibile utilizzare delle istruzioni con **lunghezza fissa** e **minima**. L'indirizzamento implicito **non può essere utilizzato per le operazioni di trasferimento** come la LOAD e la STORE perché in questo caso è necessario esplicitare dove trasferire o da dove prelevare il dato in memoria.

SET ISTRUZIONI 8085

- PCHL: Trasferisce il contenuto nel registro HL nel program counter
- RRC: Svolge l'operazione di rotate di una posizione verso destra di un valore contenuto in un accumulatore della ALU
- ADD B: Somma il contenuto del registro B con un dato salvato nell'accumulatore della ALU e restituisce il risultato in un altro accumulatore interno alla ALU
- INR B: Incrementa il contenuto del registro B di una unità

È possibile constatare come NON sia possibile ricorrere all'indirizzamento implicito qualora vi siano operazioni di trasferimento in memoria (es.: INTEL 8085).

8.2 Esplicito

La maniera più semplice e diretta di individuare un operando in memoria è quella di indicare il suo indirizzo (**indirizzamento assoluto**) nell'ADDRESS MODE.

- di accedere ad una locazione di memoria il cui indirizzo non è noto nel momento in cui il programma è scritto; ma è calcolato nel momento il cui il programma è eseguito (accesso a strutture dati come vettori, liste)
- di manipolare gli indirizzi, cioè permettere delle operazioni su di essi
- di poter calcolare gli indirizzi relativamente alla posizione dell'istruzione in modo tale che il programma possa essere caricato in memoria in qualsiasi parte della memoria senza prevedere la risoluzione degli indirizzi locali o globali (program indipendent code, PIC)

8.3 Indirizzo Effettivo

- Prima di entrare nel dettaglio dei più comuni modi di indirizzamento bisogna esplicitare il concetto di **indirizzo effettivo** (EA o effective address)
- Nelle istruzioni di trasferimento dati o in quelle logico-aritmetiche **l'indirizzo effettivo è l'indirizzo degli operandi interessati** (si accede in un'area di memoria centrale nel quale risiedono i dati – Memoria Dati MIPS)
- In una istruzione di salto o di chiamata a sottoprogramma, **l'indirizzo effettivo è quello dell'istruzione a cui si vuole saltare** (si accede ad un'area di memoria nella quale è conservato il programma – Memoria Istruzioni MIPS).

8.4 Etichetta

- Parlando di modi di indirizzamento, inoltre, si fa spesso riferimento all'etichetta
- Una etichetta è un identificatore che nel linguaggio assembly è il designatore simbolico di un indirizzo in memoria
- A differenza degli identificatori dei linguaggi ad alto livello (Pascal, C,C++,Java,...) in cui una etichetta ha un valore che le viene assegnato nel momento in cui è definita e questo valore può cambiare; in linguaggio assembly alla etichetta, durante la traduzione, è sostituito il valore che essa rappresenta
- Questa etichetta, esiste solo nel linguaggio assembly e scompare con la traduzione in linguaggio macchina.

+

8.5 Indirizzamento Immediato

L'indirizzamento immediato è così definito perché l'operando si trova nella posizione di memoria immediatamente successiva all'istruzione e pertanto, è nel corpo del programma e non in una area dati. L'indirizzamento immediato è utile per inizializzazioni o per la definizione di costanti (maschere).

Un uso eccessivo può incrementare la lunghezza del programma in memoria. In alcune macchine nel caso in cui si voglia inserire numero lunghi quanto la parola prestabilita dal progettista della macchina si ricorre ad una istruzione a lunghezza variabile dove il resto dell'operando si trova nella parola successiva a quella dell'istruzione.

Nel MIPS il campo è limitato per problemi di efficienza (si preferisce escludere due accessi in memoria) alla lunghezza della parola. Nel caso di valori superiori di 16bit l'istruzione immediate si sdoppia.

Esempio:In assembly 68000 un indirizzamento immediato è MOVE d0, #35.

```
1    li $t0,35
2    li $t0,35
3    li $t0,70000
4    ori $t0,$at,4464
```

8.6 Indirizzamento Diretto

- L'indirizzamento diretto specifica l'indirizzo effettivo di una parola di memoria: è pertanto allocato nella parola immediatamente successiva a quella contente l'istruzione che deve operarlo (istruzione a lunghezza variabile)
- L'indirizzo è fissato nel momento in cui si scrive il programma anche se è descritto da una etichetta
- Osservazione: in alcuni testi è riportato come indirizzamento assoluto.
- L'indirizzamento diretto è utile quando si deve operare con una dato che si trova in una posizione di memoria fissata o quando si deve fare un salto ad una istruzione che si trova ad una posizione prestabilita in memoria.
- In ogni caso, come nell'immediato, occupa spazio in memoria ed è meno efficiente rispetto altri modi di indirizzamento (richiede almeno un ulteriore accesso in memoria se si ha un indirizzo con un valore molto alto).

8.7 Indirizzamento a registro

L'indirizzamento a registro si riferisce all'utilizzo del registro in cui è presente l'indirizzo effettivo. Di solito, il registro contiene un operando o un indirizzo. Ad esempio, il Motorola 68000 aveva dei registri indirizzi. Ogni unità di controllo (CU) è dotata di un certo numero di registri interni chiamati registri a uso generale. Questi registri possono variare da poche unità a qualche centinaio per le grandi macchine.
Le istruzioni che utilizzano l'indirizzamento a registro sono eseguite più velocemente per due motivi principali:

1. Il campo riservato per l'indirizzo è breve perché servono pochi bit per selezionare un registro interno. Ciò consente l'utilizzo di istruzioni a dimensione fissa.

2. Non è necessario accedere alla memoria principale per rintracciare gli operandi, poiché i registri sono integrati nella CU.

L'indirizzamento a registro viene utilizzato quando si ha un operando o un indirizzo che viene utilizzato frequentemente durante l'esecuzione del programma.

8.8 Indirizzamento Indiretto

L'indirizzamento indiretto fa accedere ad un indirizzo effettivo attraverso un indirizzo presente nell'istruzione che punta in memoria ad un altro indirizzo. Tale tecnica è usata, per esempio, per condividere delle variabili tra il programma principale e una funzione (passaggio per riferimento). In questo modo la funzione chiamata dal programma principale è in grado di manipolare il valore della variabile accedendo alla locazione di memoria in cui l'operando è conservato. Questo modo di indirizzamento, usato anche nel Motorola 68000, è impiegato nell'interruzione vettorizzata.

8.9 Indirizzamento Indiretto A Registro

L'indirizzamento indiretto a registro prevede che il registro specificato nella istruzione non contenga direttamente l'indirizzo effettivo dell'operando, ma un indirizzo (definito anche puntatore) che punta all'indirizzo effettivo dell'operando. L'utilità di tale metodo di indirizzamento è quella di poter fare riferimento ad un operando in memoria principale con una istruzione breve, come nel caso dell'indirizzamento a registro, e di poter modificare l'indirizzo contenuto nel registro per puntare a dati diversi utilizzando sempre la stessa istruzione. Questa strategia è spesso utilizzata per scorrere liste e vettori.

8.10 Indirizzamento Diffreito Indiretto

L'indirizzamento differito indiretto individua l'indirizzo effettivo (un operando) mediante un indirizzo memorizzato in registro presente nell'istruzione che punta in memoria ad un altro indirizzo. Questo modo di indirizzamento è utile per gestire strutture dati.

8.11 Indirizzamento con Spiazzamento

L'indirizzamento con spiazzamento consente di raggiungere l'indirizzo effettivo dopo aver sommato al contenuto di un registro, uno spiazzamento (offset) contenuto nella parola successiva all'istruzione

8.12 Indirizzamento Relativo

Quando si parla di indirizzamento relativo si fa riferimento al fatto che l'indirizzo è relativo al Program Counter. L'indirizzo effettivo, in questo caso, è dato dalla somma tra lo spiazzamento (offset) presente nell'istruzione ed il contenuto del PC (questo si può vedere come un indirizzamento indiretto con spiazzamento il cui registro di riferimento è il PC).

Tale modo di indirizzamento è utile per realizzare programmi aventi la caratteristica di essere indipendenti dalla posizione del codice (PIC, Position Indipendent Code). Infatti, i programmi PIC, usano salti relativi che non fanno riferimento ad alcun indirizzo assoluto, ma solamente alla distanza tra le istruzioni.

1. Indirizzamento Pre-Post Incremento: L'indirizzamento con pre/postdecremento (pre/postincremento) è simile all'indiretto a registro solo che il contenuto nel registro è automaticamente decrementato (incrementato) prima o dopo l'esecuzione dell'istruzione stessa. Si elimina quindi l'istruzione di decremento (incremento) che si eseguivano sul registro usato come puntatore (offrendo una maggiore velocità di esecuzione perché non è richiesta una suppletiva fase di fetch per svolgere l'esecuzione dell'istruzione di decremento (incremento).

   L'utilità sta nel poter accedere facilmente ad insiemi di dati allocati in memoria sequenzialmente (cioè dati disposti uno di seguito all'altro, come i vettori o le stringhe) ed in particolare per le operazioni di estrazioni (POP) ed inserimento (PUSH) della pila (o canasta) che ha un modalità LIFO (Last In First out) ed a cui è dedicata una zona di memoria (stack zone).

9 Microprogrammazione

Lo sviluppo tecnologico e la non ottimizzazione dei compilatori portò alla fine degli anni '70 alla realizzazione di architetture in grado di eseguire istruzioni complesse. Nacquero, cioè, le macchine CICS (Complex Instruction Set Computing). Le macchine CICS (es.: Intel x86) hanno un set di istruzioni numeroso (dalle 200 alle 300 istruzioni), spesso a lunghezza variabile alle quali sono associati svariati modi di indirizzamento. di una CPU con nuove istruzioni (bastava sostituire una ROM). Il tempo di accesso alla RAM era superiore al tempo di accesso alla ROM che conteneva i microprogrammi e che era posizionata fisicamente più vicino alla CPU (in quegli anni le CPU non erano circuiti integrati miniaturizzati, ma occupavano lo spazio di più circuiti stampati montati all'interno di armadi della dimensione di qualche metro cubo) Il vantaggio è il codice compatto (miglior debug), mentre lo svantaggio è il transcodificatore complesso.

Per tutti gli anni '50 e '60 l'uso di istruzioni macchina molto complesse era ampiamente giustificato dalla tecnologia disponibile in quegli anni:

1. Le **tecniche di progettazione dei compilatori erano ancora in fase di sviluppo**, e avere a disposizione istruzioni macchina molto espressive permetteva di semplificare il lavoro del compilatore. In altre parole, il set di istruzione complesso consentiva di ridurre il gap semantico tra il linguaggio ad alto livello e il linguaggio macchina.

2. La **RAM era costosa** e scarseggiava, e i processori di quegli anni non erano in grado di gestire grandi spazi di indirizzamento. Dunque l'uso di istruzioni macchina molto espressive permetteva di generare eseguibili più corti.

3. **Non esistevano ancora CPU dotate di cache**, che furono introdotte solo a partire dal 1968 con l'IBM 360/85. Aveva quindi senso, ogni volta che si doveva accedere alla RAM per leggere la successiva istruzione da eseguire cercare di fare in modo che questa istruzione esprimesse una gran quantità di lavoro.

4. Infine, l'uso di microprogrammi per descrivere la funzione di controllo rendeva semplice arricchire il set di istruzioni di una CPU con nuove istruzioni (bastava sostituire una ROM). Il tempo di accesso alla RAM era superiore al tempo di accesso alla ROM che conteneva i microprogrammi e che era posizionata fisicamente più vicino alla CPU (in quegli anni le CPU non erano circuiti integrati miniaturizzati, ma occupavano lo spazio di più circuiti stampati montati all'interno di armadi della dimensione di qualche metro cubo).

9.1 Le istruzioni complesse

Le **istruzioni complesse richiedono più cicli di clock** (o un clock molto lungo) per essere eseguite, molti accessi in memoria ed un tempo riservato alla fase di load molto variabile

Per eseguire un'istruzione complessa è spesso necessario effettuare degli accessi ed operazioni in maniera sequenziale e nel giusto ordine

Il processore può elaborare le istruzioni in due modi, per interpretazione, scomponendo l'istruzione in tante istruzioni più semplici, o in esecuzione diretta, modalità nella quale non avviene una decomposizione dell'istruzione in istruzioni più semplici

Vista la complessità ed il grande numero delle istruzioni, eseguirle direttamente richiederebbe per l'implementazione in hardware un numero improponibile di componenti

La soluzione a questo problema è quella di avere all'interno del processore una ROM (memoria in sola lettura) che contiene la traduzione delle istruzioni complesse in sequenze di istruzioni semplici. Praticamente, non è più il compilatore a scomporre le istruzioni complesse in tante istruzioni semplici, ma è il processore che si preoccupa di scomporle in istruzioni direttamente eseguibili in hardware

L'operazione di decodifica è un punto fondamentale dei processori CISC, dato che, più sono complesse le istruzioni, maggiore sarà il tempo per elaborarle

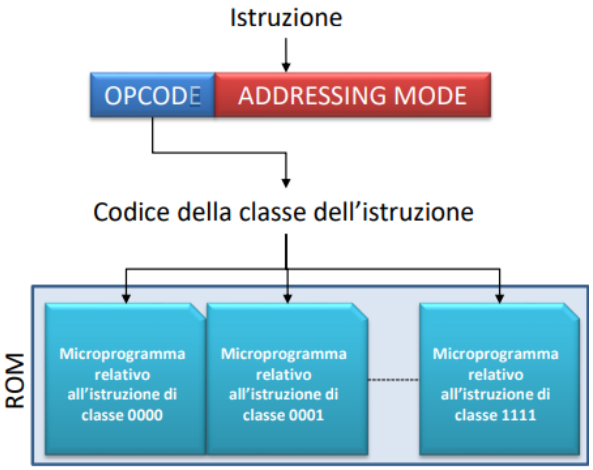
9.2 Micromacchina

Una alternativa alla comune realizzazione di un transcodificatore combinatorio fu proposta negli anni Cinquanta da Maurice Vincent Wilkes che dotò la CU con un decodificatore sequenziale e con una memoria di controllo di sola lettura in cui archiviare le sequenze di comandi che venivano trasformati da circuiterie dedicate in segnali di controllo: l'Unità di Controllo micro-programmata (UC-M). La presenza di un transcodificatore combinatorio richiede un alto numero di componenti ed interconnessioni che risultano essere costosi e scarsamente flessibili ad aggiunte e modifiche. La fase di decodifica delle istruzioni complesse in un UC-M è demandata ad una micro-macchina.

L'idea è quella di far corrispondere ad una singola istruzione macchina l'esecuzione di un microprogramma 'scritto' in una memoria non modificabile (read only memory, ROM). Il processore in questo caso ha una struttura logica e fisica che ha la funzione di interpretare programmi scritti in linguaggio macchina, in programmi scritti in linguaggio microprogrammato. Ad ogni istruzione macchina corrisponde una serie di attività che interessano la UC-M.

9.3 Microprogramma

Per ogni istruzione che la CU deve interpretare e da cui genera un serie di comandi, la stessa CU-M deve fare in modo di eseguire un **microprogramma** che è costituito da un insieme di microistruzioni il cui numero dipende dal numero di micro-operazioni che debbono essere effettuate e dal loro sequenziamento per generare gli stessi comandi. Per accedere correttamente ed in maniera univoca ai diversi microprogrammi si può utilizzare l'OPCODE di ogni istruzione (in questo caso l'OPCODE ha il ruolo di indirizzo iniziale al microprogramma da eseguire).

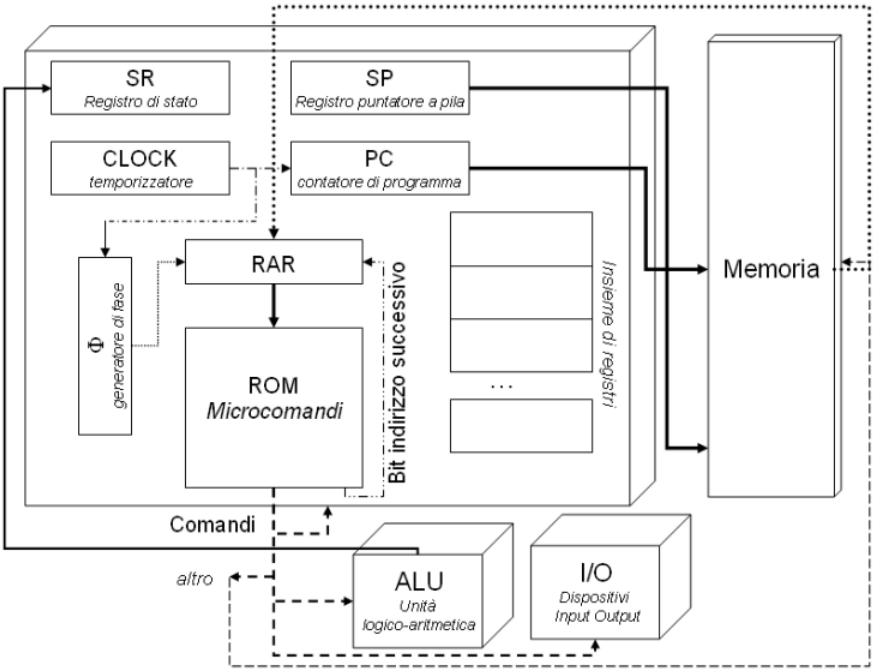


Tra le componenti essenziali della micro-macchina ci sono:

- Il generatore di indirizzo cioè il circuito predisposto al calcolo dell'indirizzo della successiva microistruzione
- Il registro RAR (ROM address register) un registro al cui all'interno è presente l' indirizzo alla micro-istruzione successiva è una sorta di program counter per la macchina microprogrammata
- Il contenuto del RAR ha il valore all'indirizzo della ROM (input) che risponde con una parola di controllo (output)

La parola letta dalla ROM rappresenta una microistruzione costituita da tante micro-operazioni che possono coinvolgere ogni componente della macchina (ALU, memoria). Una volta eseguita una microistruzione bisogna individuare la successiva o predisponendo che alcuni bit della parola di controllo siano dedicati ad assolvere questo compito oppure che alcuni bit specificano esplicitamente l'indirizzo della microistruzione successiva.

I microprogrammi possono usare anche delle microsubroutine (per svolgere compiti ripetitivi) e quindi necessita di un registro di ritorno (es.: SBR)



Riassumendo un indirizzo può essere raggiunto con:

1. Mapping dell'OPCODE dell'istruzione ai bit che rappresentano il primo indirizzo della microistruzione della routine opportuna
2. Incremento del RAR
3. Salto incondizionato all'indirizzo specificato da un campo presente nella microistruzione
4. Salto condizionato dipendente dai bit di stato nei registri della micromacchina
5. Un indirizzo contenuto in un registro di ritorno (SBR) nel caso di salto a subroutine

9.4 Microistruzione nella ROM

La ROM è una memoria di sola lettura che contiene le microistruzioni. Ogni microistruzione è composta da un insieme di bit che rappresentano le micro-operazioni che devono essere eseguite.

Bit	Microps	Descrizione
1	MDR←M	Lettura dalla memoria (nel MAR indirizzo sorgente)
2	M←MDR	Scrittura in memoria (nel MAR indirizzo destinazione)
3	MAR←PC	Indirizzo dell'istruzione (fetch)
4	PC←PC+1	Incremento program counter
5	TMP ←MAR	Trasferimento del MAR in un registro temporaneo
6	MAR ←TMP	Trasferimento da un registro temporaneo al MAR
7	MDR←AC	Trasferimento del valore sito nell'accumulatore in memoria
8	AC←MDR	Trasferimento di un operando nell'accumulatore
9	AC←AC+MDR	Somma dell'accumulatore con l'operando in MDR
10	MAR←MAR>>6	Shift a destra di 6 bit del MAR
11	MAR← MAR&&63	Maschera per estrarre 6 bit meno significativi

Condizioni

Bit	Microps	Descrizione
00	J	Salto incondizionato
01	I=1	Salto
10	AC=1	Salta se operando nell'accumulatore è negativo
11	AC=0	Salta se operando nell'accumulatore è zero

Salti

Bit	Microps	Descrizione
00	Se CD=1 RAR← ADF	Salto incondizionato
01	RAR←ADF e SBR←RAR+1	Chiamata a subroutine
10	RAR←SBR	Ritorno da subroutine
11	RAR←MDR(OP)	Caricamento prossima Micro-operazione

L'esecuzione del microprogramma relativo ad una istruzione è detto ciclo istruzione. Ogni ciclo istruzione è costituito da una o più fasi elementari che comportano l'attivazione di micro-comandi o micro-operazioni relativi ad unità interne o esterne alla CPU. Ogni istruzione è caratterizzata dal relativo numero di fasi elementari.

9.5 Microcodice orizzontale, verticale o diagonale

È proprio il numero delle micro-operazioni possibili ed il numero massimo di microoperazioni che debbono essere eseguite in parallelo che determina la possibilità di optare per una organizzazione orizzontale, verticale o diagonale del microcodice.

La struttura del micro-codice, infatti, è solitamente una soluzione di compromesso tra due esigenze diverse: buona flessibilità e potenza operativa contro la contenuta occupazione di memoria dei micro-programmi. Se si privilegia la compattezza del microcodice allora le istruzioni sono fortemente codificate e limitate nella capacità di diramazione (struttura orizzontale), se si massimizza la potenza del microcodice consentendo la massima parallelizzazione delle microoperazioni e massima flessibilità nel sequenziamento degli indirizzi allora si parla di struttura verticale. Soluzioni intermedie sono dette a struttura diagonale.

Quindi, per poter interpretare un qualsiasi programma la CU-M deve essere strutturata in modo tale da fare corrispondere ad ogni istruzione macchina una (o più) microprogrammi distinti (procedure sequenziali). Per ottenere questo si può utilizzare una ROM considerando i vari micro-programmi come tante sottomacchine in una unica macchina sequenziale.

9.6 Ottimizzazione del microcodice

L'organizzazione di una macchina micro-programmata dipende dalle prestazioni che vogliono essere ottenute e dai costi. Se non si vuole minimizzare il costo, allora il dimensionamento della ROM è fatto tenendo conto di tutte le variabili di ingresso (opcode, condition code,...) e di tutti i segnali di controllo (uscita). Ovviamente questa strategia porta a ROM di dimensioni molto elevate.

Per questo si cerca di minimizzare il numero degli ingressi (quindi una riduzione del numero delle microistruzioni) e delle uscite (operando sulla dimensione di ogni singola parola della ROM). Vediamo alcune strategie per un risparmio:

- Allocare fisicamente in modo adiacente microistruzioni in grado di interpretare una istruzione macchina (si usa solo uno spiazzamento senza codificare tutto l'indirizzo per il salto alla microistruzione seguente).
- Ridurre le variabili di ingresso che possono essere esaminate in un'altra fase (esempio: se si è nella fase di fetch è inutile considerare i dati provenienti dallo SR).
- Ridurre il numero dei segnali di controllo raggruppandoli logicamente quelli che sono mutuamente esclusivi. Ad esempio, se la ALU ha bisogno di 10 segnali di controllo per le operazioni che può effettuare, dato che queste operazioni possono essere attivate una sola alla volta, si potranno pensare di utilizzare 4 linee demandando il compito di identificare l'operazione richiesta alla ALU stessa.
- Strutturare il microprogramma con codice rientrante se differenti istruzioni macchina hanno parti di codice identico. In questo caso, però, è necessario prevedere il salvataggio degli indirizzi in una memoria gestita a pila a livello di microcodice.

9.7 Processori ibridi

Sono esistiti processori che hanno seguito l'approccio CISC per offrire la **retrocompatibilità** con i vecchi programmi (es.: Pentium IV). I processori retrocompatibili decodificano il codice con istruzioni complesse, ma una volta decodificato, esso è mandato in esecuzione su una serie di stadi di elaborazione di tipo RISC e cioè:

- Le istruzioni delle macchine CISC complesse (tante istruzioni di natura diversa), e disomogenee (le istruzioni hanno una lunghezza variabile), sono decodificate attraverso una ROM che contiene la corrispondenza tra le istruzioni complesse e quelle più semplici, e che si adopera alla decodifica delle stesse
- Le istruzioni delle macchine CISC sono, quando possibile, sostituite con istruzioni in formato omogeneo e immesse in una memoria tampone in attesa di essere processate (per essere utilizzate efficientemente con la canalizzazione)

9.8 Generalità

Nel 1980 un gruppo di Berkeley con a capo David Patterson e Carlo Séquin, cominciò a progettare chip VLSI che non usavano l'interpretazione ma avevano un set di istruzione ridotto ed a quello venivano ricondotte tutte le istruzioni anche le più complesse. Vennero realizzate le macchine RISC (Reduced Instruction Set Computer).

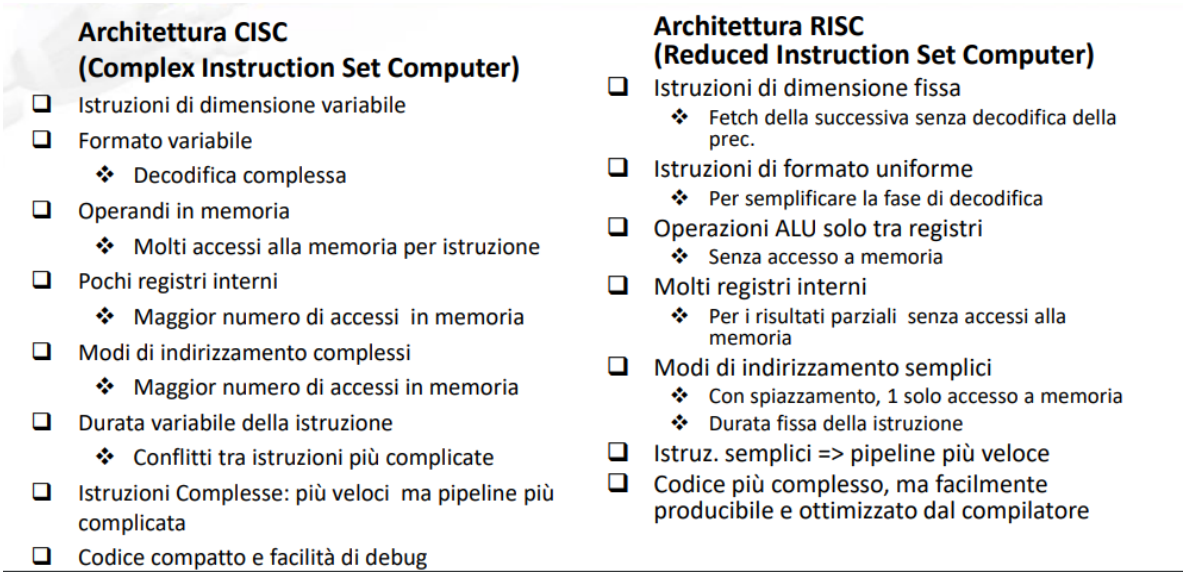
Questi nuovi processori erano molto diversi dal quelli commerciali disponibili in quel momento. Poiché queste nuove CPU non avevano bisogno di essere compatibili con prodotti già esistenti, i progettisti furono liberi di scegliere il set di istruzioni nuovo ed in grado di ottimizzare le prestazioni totali del sistema. Mentre l'enfasi iniziale fu posta su istruzioni semplici e di dimensione limitate che si potessero eseguire in poco tempo, ben presto si capì che era più importante progettare istruzioni che venissero messe in esecuzione velocemente ricorrendo alle pipeline.

Con il passare degli anni, quando i compilatori erano divenuti molto più efficienti, e le memorie meno costose, i progettisti si accorsero dei diversi vantaggi offerti dalle macchine RISC:

1. da test effettuati ci si accorse che per il 90% del tempo il processore utilizza sempre un piccolo sottoinsieme di istruzioni elementari

2. si poteva ottimizzare i tempi predisponendo e realizzando istruzioni semplici e completabili in un singolo ciclo di clock

3. Si poteva provvedere a utilizzare semplici istruzioni di trasferimento tra la CU e la Memoria (store) e tra Memoria e CU (load) e sfruttare (ed aumentare) i registri al'interno della CPU per limitare gli accessi in memoria (e in più si introdusse la memoria cache)



10 Interruzioni

10.1 Introduzione

Una **interruzione** (interrupt) è un evento, in genere determinato da un dispositivo di input o di output, che cambia la normale sequenza di un programma in esecuzione.

Possibili cause:

- I/O: interazione con tastiera, interazione con mouse
- Malfunzionamento hardware: Il bit di parità in memoria è incoerente, abbassamento della tensione di alimentazione, carta inceppata di una stampante
- Programma: overflow, divisione per zero, istruzione non riconosciuta, accesso ad un indirizzo errato, richiesta di interazione con file o dispositivi HW
- Timer: interruzione periodica (es.: ogni 10ms) per analizzare il tempo speso da una applicazione e per cedere eventualmente il processore ad un'altra applicazione (multiprogrammazione)

Le **interruzioni** sono concepite per migliorare l'efficienza. Permettono di liberare il processore da compiti gravosi di sincronizzazione dell'elaborazione. Sono utili soprattutto per gestire le operazioni realizzate da componenti che hanno tempi di risposta superiori a quelli del processore (es. dispositivi di Input ed Output).

Se il ciclo del processore prevedesse esclusivamente il calcolo sorgerebbero alcuni problemi, come per esempio:

- un'applicazione "prepotente" potrebbe impadronirsi della CPU senza mai lasciarla
- non ci sarebbe modo di rimuovere forzatamente un'applicazione che entra per errore in un ciclo infinito
- il Sistema Operativo, in generale, avrebbe un controllo limitato sul sistema (es.: non si potrebbero eseguire più programmi in memoria)

La soluzione comunemente adottata consiste nel permettere al sistema, il "Supervisore", di prendere il controllo del processore al termine dell'elaborazione di una istruzione o in una fase non interrompibile. Questo avviene esclusivamente nel caso si verifichino eventi "eccezionali", di solito asincroni con l'esecuzione del programma correntemente in corso. In assenza di tali eventi l'elaborazione procede nella consueta maniera.

10.2 Classificazioni

- **Interruzione hardware**: il segnale di interruzione è scaturito da un componente esterno alla CPU (es.: l'immissione di un carattere da tastiera)
- **Interruzione software**: il segnale di interruzione è scaturito da un programma
  - **Trap**: interruzione richiesta dal programmatore
  - **Eccezioni**: interruzione determinata durante l'elaborazione di un programma (es.: una divisione per zero)

- Interruzioni possono essere **classificate anche in relazione al clock** della macchina:
  - **Asincrone** (il loro accadimento non è noto, sono indipendenti dal clock)
    - **Interruzioni esterne** (generate da un dispositivo di I/O; es.: inceppamento carta)
    - **Interruzioni interne** (generate da un programma, anche note come eccezioni) es.: la creazione di un file
  - **Sincrone** (il loro accadimento è noto, perché richiesto dal programmatore: è una specifica istruzione)
    - **Trap**

10.2.1 Sistemi di interruzioni

- **Mascherabile**: l'interruzione può essere interrotta a sua volta per svolgerne un'altra (di solito più importante, cioè a priorità più alta)
- **Non mascherabile**: una volta richiesta l'interruzione questa deve essere espletata senza la possibilità di essere interrotta da altre

10.3 Superamento dell'I/O canonico e programmato

I moduli di interfaccia dei dispositivi collegati ad un elaboratore hanno una parte indipendente, il controllore (controller), dal dispositivo stesso che consente di adottare protocolli di I/O uniformi per il trasferimento di dati. Le interazioni tra un processore ed un dispositivo esterno prevedono la possibilità di una attesa nel caso in cui il dispositivo non sia pronto ad intraprendere la transizione. La velocità con cui opera un processore (ordine di grandezza per il trasferimento dati è di 10^7-10^9 parole al secondo) è di parecchi ordini di grandezza più elevata rispetto le operazioni di un dispositivo (ad esempio i dischi magnetici 10^6 byte/secondo ed i nastri magnetici 10^5 byte/secondo). Una interruzione, rispetto ad un I/O programmato consente di ridurre i tempi utili alla CPU per fare altre cose.

10.4 Modalità operativa

Un interrupt interrompe il programma in esecuzione e trasferisce il controllo a un **gestore di interruzione** deputato a svolgere le azioni appropriate. Al loro compimento, il gestore di interruzione restituisce il controllo al programma interrotto. È suo compito far riprendere il processo interrotto esattamente dallo stesso stato e posizione in cui si trovava al momento dell'interruzione, il che implica il ripristino di tutti i registri interni allo stato precedente all'interruzione.

10.4.1 sequenza di attivazione

Sequenza di attivazione:

1. la Periferica o il processore (nel caso di una interruzione interna) alza il segnale di interruzione (INT)
2. il Processore interrompe il programma in esecuzione
3. la Periferica è informata che l'interrupt è stato ricevuto (Ack)
4. è eseguita la procedura che risolve la richiesta di interruzione (ISR)
5. si ripristina il programma originale

10.5 Sistema di interruzione

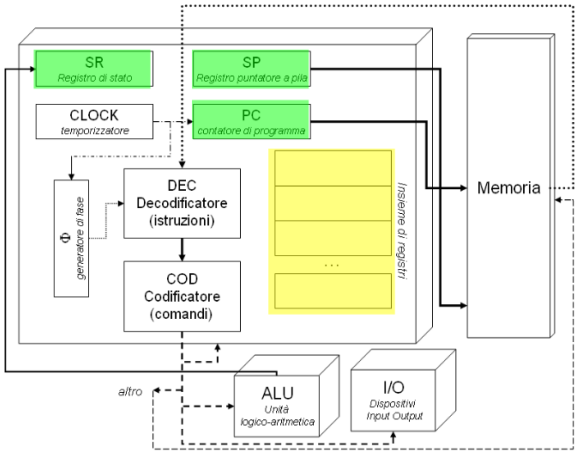
Un sistema di interruzione deve:

- Garantire che non vi siano interferenze indesiderate sul processo interrotto
- La coerenza della ri-esecuzione del processo avviene tramite il salvataggio ed il ripristino del contesto esistente nel momento in cui si verifica una interruzione (**commutazione del contesto**)
- Identificare il dispositivo che ha generato l'interruzione al fine di selezionare ed attivare la routine ad esso associata (**riconoscimento delle interruzioni**)
- Risolvere le diverse interruzioni generate dai svariati dispositivi tramite la definizione di un ordine di gestione nel caso di un sistema a mascheramento (**gerarchia di priorità**)

10.6 Commutazione del contesto

Il contesto sul quale un programma opera è costituito dalle informazioni presenti nei seguenti registri (**lo stato volatile della macchina**):

- il Program Counter, che contiene l'indirizzo della istruzione da cui dovrà essere ripresa l'esecuzione del programma interrotto
- i bit di condizione (implicitamente conservati nel Registro di Stato) per ritornare ad una situazione coerente con le ultime istruzioni elaborate prima del salto (infatti i CC potrebbero essere modificati dalla routine di servizio)
- i registri del modulo ALU, che possono contenere dati che il programma interrotto non ha terminato di elaborare



Il salvataggio del contesto può essere effettuato solo quando si è raggiunta una situazione ben definita e ricostruibile (esempio: prima che inizi la fase di fetch della istruzione successiva) L'operazione di salvataggio può essere intesa come una **commutazione dal contesto** del programma che è interrotto a quella della routine di servizio. Analogamente il ripristino è ancora una commutazione: dal contesto della routine di servizio a quello del programma da proseguire. Pertanto è opportuno che non si verifichino altre interruzioni mentre sono in corso le operazioni di commutazione del contesto (**fase protetta**), altrimenti ci potrebbero essere anomalie che comprometterebbero il corretto funzionamento del sistema. Per far questo si può dotare il processore di un **campo di interruzione** che consenta di definire se può essere interrotto o no, affinché non sia possibile interferire con le operazioni di commutazione. Il campo può essere conservato nel Registro di Stato e può essere di un solo bit, identificato dalla lettera I (**bit di interrompibilità**, Interrupt flag), o di più bit nel caso si vogliano gestire più interruzioni. (Di solito con I=1 si accettano interruzioni con I=0 si rifiutano)

10.7 Riconoscimento delle interruzioni

L'uso di un sistema di interrupt implica che ci siano in Memoria Centrale tante routine di servizio (**interrupt service Routine**, ISR) quanti sono i dispositivi collegati all'elaboratore o quante interruzioni software devono essere soddisfatte.

Questa moltitudine di routine prevede che vi sia la corretta **identificazione del dispositivo** o **della richiesta del trap**

- L'individuazione può avvenire:
  - tramite una sequenza di istruzioni atte all'individuazione del dispositivo che faccia da preambolo comune a tutte le routine e pertanto eseguita all'inizio di ogni interruzione (scansione delle interruzioni o **polling**)
  - prevedendo che il dispositivo, oltre a generare la richiesta di interruzione invii una informazione più completa come ad esempio un codice di identificazione univocamente associato al dispositivo chiamante (**interruzione vettorizzata**)

10.8 Gerarchia delle interruzioni

In alcune applicazioni le elaborazioni previste da una sequenza di istruzioni devono essere eseguite, dopo un evento esterno, entro un **tempo massimo** determinato da esigenze che variano a seconda del dispositivi e del sistema in considerazione e secondo un ordine prestabilito.

Si può pensare ad un sistema per il rilevamento termico del nocciolo di una centrale nucleare, che ogni Δt campiona la temperatura: il prelievo del dato prodotto deve avere luogo entro l'intervallo temporale fissato, altrimenti il dato non è prelevato o è distrutto dal sopraggiungere del successivo (e in un contesto così pericoloso sono due condizioni da scongiurare!!!). Altresì per dispositivi come i terminali video o di stampa, generalmente, la visualizzazione o la riproduzione può essere rimandata senza gravi problemi.

Per questo è opportuno che un sistema di interruzione sia in grado di assicurare che le richiesta di interruzioni provenienti dai dispositivi con esigenze di **tempo reale** (o real time) più critiche siano gestite con priorità superiore rispetto alle altre. La priorità deve essere considerata:

- quando sono presenti contemporaneamente più interruzioni
- quando si presenta una interruzione mentre è in corso il servizio di un'altra

Nel primo caso è necessario che ci sia una discriminante per consentire un ordinamento che porti ad individuare il dispositivo che ha esigenze prioritarie e lasci pendenti gli altri. Nel secondo caso la priorità consente di stabilire se l'interruzione in atto debba essere sospesa a vantaggio di una nuova interruzione, oppure no.

10.9 POLLING

Nel polling o I/O controllato da programma o **scansione delle interruzioni**.
Alla fine di ogni istruzione la CPU interroga i registrai di stati dei dispositivi per vedere se almeno una periferica ha richiesto una interruzione. Il polling impedisce alla CPU di fare altro durante l'attesa; infatti la CPU si blocca in quello che si chiama busy-waiting.

10.9.1 Segnalazione di un Dispositivo

Nel polling è necessario fornire nel modulo di interfaccia dei dispositivi, un bit I (conservato in un registro di stato del dispositivo) che segnala la **richiesta di interruzione**.

Ovviamente questa linea è legata anche al bit di interrompibilità presente nella CPU (tipicamente nel registro di stato) per abilitare (1) o non abilitare (0) le interruzioni.

Dopo aver completato una istruzione il processore deve analizzare il segnale uscente e vedere se è 1, in questo caso ci sarà l'accettazione dell'interruzione.

In una architettura è logico pensare che siano collegati più dispositivi e quindi si può pensare ad una circuiteria realizzata con un OR logico o, più comunemente, ad un collegamento wired OR di porte open collector che consenta di usare un'unica linea per far pervenire alla CPU le diverse richieste.

10.9.2 Commutazione del contesto

Il completamento della commutazione del contesto può avvenire in maniera più o meno radicale (salvando tutte le informazioni contenute nei registri e nella ALU - codice semplice e compatto ma efficienza bassa; oppure solo le informazioni che effettivamente vengono alterate nel corso del servizio dell'interruzione - codice complesso ma efficienza alta).

Per avere un hardware molto semplice è possibile pensare che le uniche operazioni interessate alla commutazione del contesto siano quello di salvataggio del contenuto dello PC e dello SR nello stack. In generale si salva tutto lo stato volatile della macchina grazie ad una procedura nota come.

In seguito, la procedura setta a 0 il flag I per impedire che ci sia un'altra interruzione (architettura non mascherabile).

Infine, è necessario inserire nel PC l'indirizzo della prima istruzione della routine di servizio di interruzione. Se l'hardware consentisse un'identificazione del dispositivo, si potrebbe passare direttamente l'indirizzo della routine di gestione dell'interruzione associata. Tuttavia, in questa modalità non è possibile farlo direttamente, pertanto si punta a una routine di preambolo comune a tutte le routine, atta al riconoscimento del dispositivo (nell'esempio a destra, la routine di preambolo risiede nell'indirizzo riservato 0x1000).

10.9.3 Identificazione dell'interruzione

A questo punto il riconoscimento dell'interruzione può avvenire tramite il **polling**: interrogando ad uno ad uno gli n dispositivi (o le n tipologie di interruzione gestibili dalla macchina) fino a che non si trova il richiedente del servizio. La sequenza di istruzione del polling (**handler**) prevede un salto a subroutine agli indirizzi SERVn delle corrispondenti routine di servizio del dispositivo chiamante DISPn.

10.9.4 Gerarchia di priorità

Per quanto riguarda la gerarchia di priorità si può pensare che l'ordine con cui i dispositivi sono analizzati dalla **sequenza di polling indica anche una gerarchia di priorità** (in caso di richieste contemporanee è esaminata la prima).

Nel caso in cui si debba prevedere la possibilità di interrompere una routine di servizio (architettura mascherabile) è necessario utilizzare una istruzione di SETI che setta il bit di Interruzione (I) a 1 e renda il processore nuovamente interrompibile (inoltre vanno presi ulteriori accorgimenti).

10.10 Interruzioni: strategie di miglioramento

Per migliorare **l'efficienza di un sistema di interruzione** si cerca di ottimizzare il tempo di risposta, cioè il tempo che intercorre tra l'istante in cui un dispositivo sollecita l'intervento del processore e quello in cui il processore inizia l'esecuzione della prima istruzione utile di interazione col dispositivo (dai 1-2 microsecondi a 20-30 per sistemi meno sofisticati) Per ottenere tale obiettivo possono essere considerati diversi approcci nelle diverse fasi che vedono coinvolti una interruzione. Nella parte di **commutazione del contesto** si può pensare di effettuare un salvataggio delle informazioni che costituiscono il contesto in **registri appositi** (in questo modo si aumentano i costi, perché deve essere realizzato un set di registri per ogni livello di priorità corrispondente; ma si incrementa la velocità: non c'è alcuna perdita di tempo per eseguire una salvataggio nello stack).

10.11 Interruzione Vettorizzata

Un ulteriore vantaggio può esserci utilizzando un sistema di **interruzione vettorizzato** (interrupt vectored system) grazie al quale è possibile identificare direttamente il dispositivo che richiede un servizio.

Un sistema siffatto prevede che nel modulo interfaccia di ciascun dispositivo sia presente un registro in cui è memorizzato un codice di identificazione o **numero vettorizzato di interruzione** (o interrupt vector number, IVN) che viene inviato al processore quando si verifica l'interruzione richiesta dal dispositivo.

Il codice potrebbe puntare direttamente alla routine di servizio, ma questa scelta non è la migliore: per questo si utilizza un **IVN** che rimanda ad un indirizzo di una locazione di memoria che contiene l'indirizzo iniziale alla routine di gestione dell'interruzione.

Osservazione. La gerarchia tra i dispositivi può essere dettata dal valore del IVN.

Utilizzando questo modo, il processore non ottiene solo la richiesta di interruzione, ma anche un **indirizzamento indiretto** fornito tramite l'IVN. Il programmatore, in questo caso, deve collocare nelle celle di memoria indirizzate dai codici IVN gli indirizzi delle rispettive routine di servizio. A questo scopo è riservata una tabella in memoria in cui risiedono gli indirizzi alle routine di **servizio (interrupt service Routine, ISR)** Di solito la tabella risiede nella parte alta della memoria in modo tale che si possa rintracciare l'indirizzo utilizzando codici con pochi bit. L'indirizzo effettivo che viene forzato al PC nasconde un indirizzamento indiretto perché nella tabella delle interruzioni c'è l'indirizzo di dove si trova la routine di gestione.

Questo metodo è utile per non tenere tutte le routine di gestione in memoria (ma solo dei dispositivi connessi).

In questo modo è possibile locare le routine in memoria in punti diversi (ottimizzazione della Memoria Centrale) Una volta finito la ISR ha una istruzione che ripristina il sistema (**RTI, return from Interrupt**).

Osservazione. Se si utilizza un IVN si dimensione 256 è possibile usare un codice di soli 8 bit.

10.11.1 Modalita' di funzionamento

La presenza di interruzioni è spesso collegata alla gestione delle funzionalità "di basso livello" dell'elaboratore

- gestione del disco, periferiche, timer, etc.
- Normalmente, i programmi per la gestione di tali funzionalità devono avere accesso a tutte le caratteristiche dell'elaboratore.
- tipicamente, è meglio evitare invece che questo sia concesso ai normali programmi utente.
- Molti processori prevedono pertanto (almeno) due modalità:
  - Supervisore (accesso pieno alle funzionalità del sistema)
  - Utente (accesso limitato alle sole istruzioni standard)
- All'accettazione dell'interruzione, la CPU cambia modalità di esecuzione passando da Utente a Supervisore.

10.12 Driver

La modalità di esecuzione Supervisore è normalmente pensata per l'esecuzione di routine che sono gestite dal Sistema Operativo.

L'insieme di routine, comprese le ISR, che gestiscono l'interazione con una particolare periferica viene detto driver.

Ogni periferica ha il proprio funzionamento e necessita pertanto dei propri driver.

Questo è il motivo per cui l'installazione di una nuova periferica comporta normalmente l'installazione dei corrispondenti driver.

10.13 Interruzioni Multiple

Nel caso di più interruzioni si può procedere in due modi:

- **Mascherabile**: l'interruzione può essere interrotta a sua volta per svolgerne un'altra (di solito on priorità più alta)
- **Non mascherabile**: una volta richiesta l'interruzione questa deve essere espletata senza la possibilità di essere interrotta da altre (si avrà una sequenza di istruzioni lasciando "appese" le interruzioni che intervengono dopo la prima)

Anche se subentra una richiesta di interruzione a più alta priorità, ci sono alcune **fasi del servizio di un'interruzione** (salvataggio e ripristino del contesto) che non possono essere interrotte (**fase protetta**).

- Sospensione esecuzione del programma corrente
- Salvataggio del contesto
- Inizializzazione del PC all'indirizzo di ingresso della routine per la gestione dell'interrupt
- Ripristino del contesto
- Continuazione esecuzione del programma corrente
- Esecuzione della routine di interruzione ISR

Il salvataggio dello stato incrementa il ritardo tra l'istante di ricezione della richiesta di interruzione e l'istante in cui inizia l'esecuzione della routine di interrupt. Questo tempo viene detto latenza di interrupt ed indica l'intervallo temporale massimo che intercorre tra la richiesta di attenzione e l'effettivo servizio dell'interruzione.

10.14 Interruzioni Esterne

Tra le principali interruzioni esterne devono essere evidenziate:

- Stampante
- Richiesta di stampa
- Stampa non collegata
- Mancanza o inceppamento carta per la stampa
- Mouse
- Tastiera
- Mancanza della tastiera
- Video
- Sconnessione con l'elaboratore (segnale assente)

10.15 Interruzioni Interne

All'interno dell'elaboratore possono verificarsi anche delle interruzioni **generate dal processore stesso** (interruzioni interne o eccezioni) e sono asincrone o **volutamente dal programmatore** (trap) e sono sincrone.

Le generazioni di queste richieste di interruzioni possono essere di tipo **recuperabili** (soft o fisiologiche) o **irrecuperabili** (hard o patologiche).

Tra le principali interruzioni interne asincrone vanno evidenziate:

- Il tentativo di eseguire una divisione per zero (zero divide)
- La mancanza di tensione di alimentazione (power failure)
- Indirizzo errato (address error): condizione che si verifica quando si chiede l'accesso ad una cella di memoria non fisicamente presente. Questa interruzione è sfruttata nella memoria virtuale che sarà presentata in seguito
- In modalità asincrona:
  - L'esecuzione della modalità **trace** (single step): si verifica una interruzione dopo ogni istruzione eseguita

10.16 Interruzioni software

Nei processori attuali si prevede la possibilità che un programma richieda in modo esplicito un'interruzione a se stesso. Si ha una **interruzione software** (o trap): una interruzione voluta e scritta dal programmatore. Al contrario delle comuni interruzioni fino ora viste, che sono segnali asincroni (è imprevedibile stabilire quando occorrano), le interruzioni software invece si verificano in corrispondenza dell'istruzione che le richiedono (segnale sincrono).

Il meccanismo delle trap è uguale alle altre interruzioni e **simile al salto a subroutine** (infatti le trap sono incluse di diritto nella classificazione delle istruzioni). In entrambi i casi si ha il salvataggio dello stato volatine della macchina, ma la differenza è nell'attivazione: nel salto a subroutine si conosce l'indirizzo della routine a cui si vuole saltare; nelle trap invece l'indirizzo è ricavato dalla tabella del IVN e non richiede che il programma chiamante conosca l'indirizzo del sotto-programma chiamato. Questa strategia è utile anche per richiamare moduli oggetti e librerie esterni al codice eseguibile in corso di elaborazione.

10.16.1 Esecuzione

1. Esecuzione del programma
2. Caricamento istruzione TRAP con identificativo (o etichetta) #10
3. Decodifica istruzione (riconoscimento di interruzione software)
4. Esecuzione
   (a) Commutazione del contesto: salvataggio SR e PC nello stack e dello stato volatile
5. Forzatura del PC all'indirizzo individuato dall'etichetta #10 ed eventuale mascheramento
6. Salto a IVN
7. Salto alla routine di servizio (si forza anche in questo caso il PC all'indirizzo trovato nell'IVN)
8. Esecuzione routine di servizio
9. Ripristino del contesto
   (a) Recupero dallo stack del SR e del PC dello stato volatile della macchina
10. Prosecuzione istruzione del programma

Le trap sono utilizzate per usufruire delle librerie di sistema che in questo modo possono essere modificate (nelle versioni successive) in modo del tutto trasparente ai programmi degli utenti. Per questo motivo molto spesso si usa il sinonimo di **chiamata a sistema** (syscall).

In questo caso l'indirizzo contenuto nella locazione di memoria puntata dal TRAP riporta l'indirizzo della routine o libreria a cui si deve accedere (esempio di multiprogrammazione), alla fine della libreria c'è un ritorno esplicito al programma originale.

Inoltre grazie alle interruzioni generate nel Sistema Operativo a tempi prestabiliti è possibile la multiprogrammazione: cioè più programmi eseguiti in memoria ad intervalli regolari e separati che danno un effetto di pseudo parallelismo.

10.17 Interruzione nel MIPS

Nel MIPS il controllo della CPU, prima di saltare all'handler predisposto dal Sistema Operativo (sito ad un indirizzo fisso), deve salvare in un registro interno un identificatore numerico del tipo di eccezione/interruzione verificatosi. L'handler accederà al registro interno per determinare la causa dell'eccezione/interruzione. Nel MIPS si usa il registro, denominato **Cause**, per memorizzare il motivo dell'eccezione/interruzione. Il valore del Program Counter corrente è salvato nel **registro EPC** (Exception PC) tutto il resto deve essere salvato dall'handler.

Il sistema (ma anche il datapath corrispondente) deve essere progettato per:

- individuare l'evento inatteso
- interrompere l'istruzione corrente
- salvare il PC corrente (nel registro interno EPC)
- salvare la causa dell'interruzione nel registro Cause (vedere Coprocessore 0 del MARS)
- saltare ad una routine del SO (exception/interrupt handler) ad un indirizzo fisso: 0xC0000000

Il MIPS non salva alcun altro registro oltre il valore del PC:

- È compito dello handler salvare altre porzioni dello stato corrente del programma (es.: tutti i registri generali), se necessario
- Approccio RISC: semplice e minimale
- Esistono CPU dove il "salvataggio esteso" dello stato viene sempre effettuato prima di saltare all'interrupt handler
  - Salvataggio garantito dal microcodice:complesso
  - Approccio CISC: complesso e completo.

11 Canalizzazione

Il **pipeline**, o canalizzazione, è una tecnica che consiste nella scomporre una rete logica **(combinatoria)** in una serie di reti logiche più semplici mediante l'inserimento di opportuni registri di disaccoppiamento **(interlock)**.

Questo accorgimento permette di aumentare la frequenza di clock e svolgere più fasi in parallelo.

L'esecuzione di una istruzione, in una macchina **RISC** (es.: MIPS), è di solito suddivisa in **cinque fasi** (o sezioni o stage):

- **Fetch**: prelevamento dell'istruzione dalla memoria (con incremento del Program Counter)
- **Decode**: decodifica dell'istruzione
- **Execute**: esecuzione dell'istruzione
- **Memory**: accesso alla memoria per scrittura o lettura (load o store)
- **Write Back**: scrittura del risultato registro destinazione

In una macchina senza pipeline in ogni momento, solo una unità funzionale è attiva e le altre non sono impegnate. Mentre in una **macchina con pipeline** ogni unità funzoonale elabora la fase che gli corrisponde e poi passa all'operaqzione successiva.

In questo modo una volta che la canalizzazione arriva a **regime**, cioè quando tutte le unità funzionali sono occupate e si svolgono fino a n istruzioni contemporaneamente in n fasi diverse, alla fine di una fase si completa una istruzione.

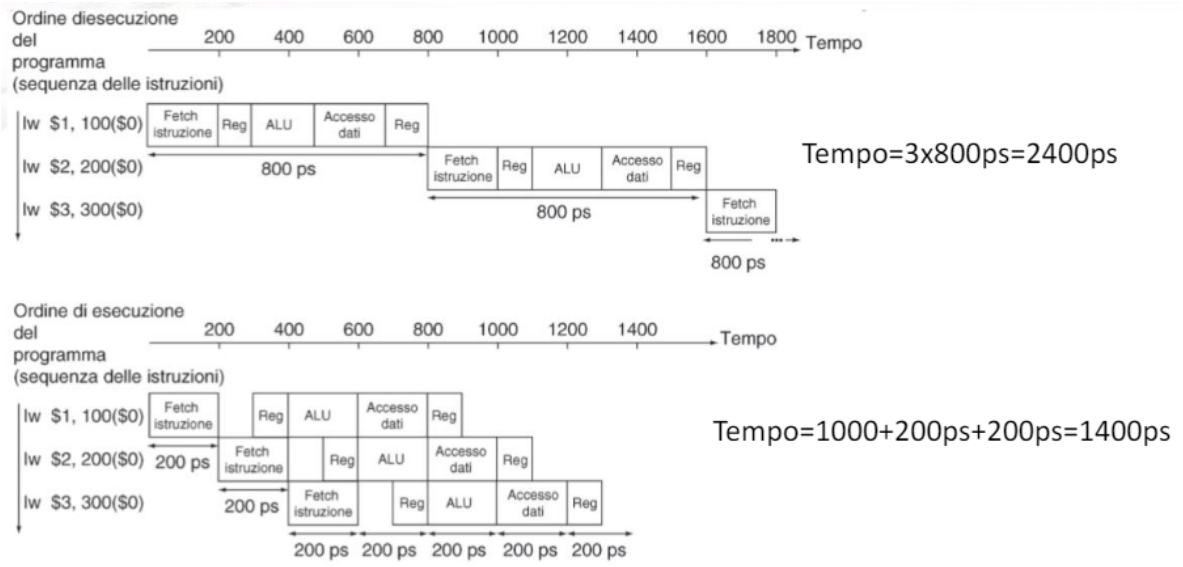
Poiché ciascuna fase deve poter essere svolta contemporaneamente a quelle delle altre istruzioni, è necessario imporre che tutte quante **abbiano la stessa durata** (stabilita valutando la fase più lenta durante la progettazione dell'architettura con test temporali). Questo determina un **clock totale più lungo** (si attribuisce a tutte le classi di istruzione il tempo della classe di istruzione più dispendiosa temporalmente; c'è un periodo più lungo), ma il tempo complessivo di esecuzione del programma si migliora (si ha una frequenzadi elaborazione maggiore una volta che la canalizzazione è a regime).

Per determinare il periodo di clock uniforme che permette di svolgere le fasi in modo da poterle sovrapporre si deve individuare l'istruzione o meglio la classe di istruzione più lenta.

Il guadagno offerto da una canalizzazione è, una volta "a regime", dato del numero delle fasi elementari legate alla esecuzione dell'istruzione (in un caso con cinque fasi il guadagno è di cinque volte).

In alcune macchine c'è una canalizzazione fino a 20 fasi che consente uno sfruttamento completo di tutte le unità funzionali in gioco (es.: l'accesso a delle memorie ausiliare come la cache).
Il **compilatore** assume un ruolo fondamentale: deve infatti presiedere alla suddivisione delle istruzioni complesse in gruppi di istruzioni semplici ed ottimizzare la sequenza di esecuzione, sfruttando appieno i vantaggi della pipeline.

La canalizzazione è trasparente al programmatore.



Il **compilatore** assume un ruolo fondamentale: deve infatti presiedere alla suddivisione delle istruzioni complesse in gruppi di istruzioni semplici ed ottimizzare la sequenza di esecuzione, sfruttando appieno i vantaggi della pipeline.

11.1 Canalizzazione nelle macchine CISC

L'implementazione di pipeline su **elaboratori CISC** risulta difficoltosa a causa della intrinseca complessità delle istruzioni.

L'esecuzione di una istruzione, nelle macchine CISC, è di solito suddivisa in sei fasi (o sezioni) la cui durata è molto variabile:

- **F - Fetch**: prelievo istruzione (con incremento del Program Counter)
- **D - Decode**: decodifica/riconoscimento istruzione
- **L - Load**: prelievo operandi dalla memoria (tipico delle istruzioni con modi di indirizzamento complesso)

- **E - Execution**: esecuzione istruzione
- **M - Memory**: accesso in memoria per scrittura o lettura (load o store)
- **WB - Write Back**: quando il risultato dell'ALU o quello letto dalla memoria viene messo nel registro destinazione

La **fase Fetch** può sovrapporsi alla fase Decode nelle macchine CISC solamente se l'istruzione ha dimensione fissa e quindi è possibile calcolare il prossimo valore del Program Counter senza sapere la classe di istruzione. Altrimenti nelle architetture CISC con **istruzioni a dimensione variabile**, oltre alla presenza di una eventuale fase di **Load** (recupero degli operandi), la fase di **decodifica** è variabile così come quelle di esecuzione e di scrittura dei dati.

L'alternativa è commisurare il tempo più lungo di ciascuna fase delle diverse classi di istruzioni ed uniformare ogni fase di tutte le classi di istruzioni a quelle più lunghe (esattamente come avviene per le macchine RISC); con un dispendio temporale più lungo per le istruzioni "meno complesse". Seppur fattibile, la canalizzazione nelle macchine CISC è molto più complicata da realizzare. Inoltre, le istruzioni CISC compiono funzioni più "articolate" rispetto alle istruzioni RISC, che richiederebbero più istruzioni per essere eseguite. Nonostante queste difficoltà, l'uso delle macchine RISC è preferito perché la maggior parte dei programmi utilizza istruzioni semplici, mentre l'impiego di istruzioni complesse è riservato a pochi applicativi specifici.

Esecuzione RISC																					
Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Istruzione 1	F	D	E	M	WB																
Istruzione 2		F	D	E	M	WB															
Istruzione 3			F	D	E	M	WB														
Istruzione 4				F	D	E	M	WB													
Istruzione 5					F	D	E	M	WB												

Esecuzione CISC																					
Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Istruzione 1	F	D	D	L	L	L	E	E	M	WB											
Istruzione 2				F	D	D	L	L	L	E	M	WB									
Istruzione 3						F	D	D	D	D	L	L	E	M	M	WB					
Istruzione 4										F	F	D	L	L	L	E	M	WB			
Istruzione 5												F	F	D	D	L	L	L	E	M	WB

11.2 Criticità della canalizzazione o hazard

La regolarità delle istruzioni permette di utilizzare e ottimizzare il funzionamento della pipeline, fino a quando non si verifica **una criticità nella esecuzione (hazard)**. Ad esempio, nel caso di un salto è necessario svuotare il pipe per far si che siano caricate le istruzioni che fanno riferimento al punto di arrivo del salto e le successive.

11.2.1 HAZARD

Le criticità nella esecuzione (hazard) possono essere:

- **SUL CONTROLLO (control hazard)**: un salto cambia il flusso sequenziale di esecuzione delle istruzioni (*jump/beq* ma anche una interruzione o un salto a subroutine)
- **SUI DATI (data hazard)**: quando un dato (istruzione o operando) che deve essere elaborato in una istruzione non è ancora pronto perché in corso di calcolo nell'istruzione precedente.
- **STRUTTURALI**: le risorse HW non sono sufficienti (es.: l'ALU sta ancora svolgendo una operazione precedente e quindi non è libera)

Per risolvere i problemi di hazard si possono adottare diverse strategie:

- **STALLING**: si blocca la pipeline fino a che non si risolve il problema. Si ottiene riordinando le istruzioni (ed evitando gli stalli).
- **FORWARDING**: si anticipa il dato che deve essere elaborato.L'adozione del forwarding evita i problemi correlati ad una canalizzazione con solo interlocked block. Una canalizzazione interlocked block le fasi della pipeline NON sono interconnesse in modo tale che l'esecuzione di un'istruzione può dipendere dal completamento di un'altra istruzione nella pipeline. Questo tipo di organizzazione può introdurre ritardi e complicazioni nel design del processore.

  Con il termine "without Interlocked Pipeline Stages" si indica che il design del processore MIPS è stato sviluppato senza questa caratteristica, adottando un approccio più semplice e diretto alla progettazione della pipeline. Questo ha contribuito a rendere il MIPS più efficiente e più facile da implementare rispetto ad altre architetture di processore che utilizzano l'interlocking delle fasi della pipeline.

  Se la fase che ha bisogno del dato si trova prima (nel tempo) di quella che lo produce, il forwarding non è possibile e quindi è necessario inserire un tempo di attesa **(stallo o bolla)**.
- **BRANCH PREDICTION**: si prevede il salto e si carica la pipeline con le istruzioni che si prevede saranno eseguite

11.3 CPU MIPS con Pipeline

La **canalizzazione** per funzionare adegautamente richiede che le varie unità siano sincronizzate e che i dati delle varie istruzioni non si sovrappongano, quindi all'interno delle pipeline sono posti dei blocchi (**interlock**) che sorvegliano il completamento delle varie istruzioni e fanno procedere il pipeline solamente quando tutti gli stadi sono pronti. Questo meccanismo garantisce la corretta esecuzione del programma ma introduce spesso stalli nella CPU che deprimono le prestazioni.

La caratteristica distintiva del progetto MIPS è che tutte le istruzioni sono completate dagli stadi della pipeline in un solo ciclo di clock in modo da non introdurre ritardi e stalli nella pipeline.

CPU MIPS con pipeline (senza gestione hazard)

Obiettivo:

- Fasi veloci (periodo di clock = durata della fase più lenta)
- Ciascuna fase realizza un solo compito (o più compiti ma in parallelo)
- Ciascuna fase riceve informazioni e segnali di controllo
- Ciascuna fase passa alla successiva le informazioni e segnali di controllo
- I segnali necessari devono restare stabili durante tutta la fase. Per rendere stabili dei segnali si usano LATCH oppure registri che cambiano solo alla transizione del clock

Soluzione: separare ciascuna fase dalla successiva con un registro (interblock) che riceve informazioni e segnali di controllo dalla fase precedente e li mette a disposizione alla fase successiva.

11.4 Considerazioni sugli hazard

Un Data hazard si verifica quando una istruzione dà il valore ad una delle due istruzioni successive (prima di WB) ovvero: il registro destinazione della istruzione precedente è uguale ad uno dei due registri argomenti delle due istruzioni successive ed inoltre: la istruzione precedente ha RegWrite = true.

Nel MIPS, per fermare l'istruzione con uno stallo, si deve (nella

- Annullare l'istruzione che deve attendere (una bolla che continuerà senza fare nulla) ovvero azzerare i segnali di controllo MemWrite e RegWrite e bloccare il registro F/D
- Ripetere l'istruzione che è stata letta e deve entrare nella fase Decodifica ovvero impedire l'aggiornamento del registro F/D della pipeline
- Rileggere la stessa istruzione di nuovo perché possa essere rieseguita un clock dopo, ovvero impedire che il PC si aggiorni

Salto incondizionato:

- Anticipo: si costruisce una circuiteria che analizza l'opcode dell'istruzione e se è un salto incondizionato si aggiorna il Program Counter dopo il fetch senza decodificare l'istruzione saltando alla parte di codice indicato dal salto (non si scarta nessuna istruzione perchè la successiva è quella corretta).

Salto condizionato:

- Predizione e Ritardo del salto (delay slot)

Ad ogni istruzione di salto si possono associare alcuni bit che «**predicono**» (rispetto alla storia dei salti già fatti) contando se è più probabile che il salto sia fatto oppure no.
Con un solo bit si può rappresentare l'informazione «l'ultima volta il salto è stato fatto». Nel realizzare un ciclo la previsione sarà sbagliata 2 volte: entrando e uscendo. Con due bit si può realizzare la macchina a stati finiti (figura laterale), che ha bisogno di due sbagli consecutivi per cambiare previsione e quindi sbaglia meno nei cicli a forte prevalenza di uno specifico tipo di scelta (fa una sola predizione errata).

Per recuperare il tempo perso dallo stallo si può ritardare il salto (delay slot), ovvero eseguire in ogni caso l'istruzione che segue il salto condizionato.

- Questo richiede una scrittura del codice assembly diversa, oppure l'uso di un compilatore capace di automatizzare le necessarie modifiche al codice macchina prodotto.
- Nel delay slot è possibile copiare una delle istruzioni che vanno sempre eseguite:

– Caso 1: una istruzione precedente che non abbia dipendenze (anche indirette) con il salto condizionato. NOTA: l'istruzione viene copiata perché potrebbe far parte di altri flussi di controllo.

– Se non ci sono istruzioni precedenti senza dipendenze:

  Caso 2: si copia l'istruzione alla destinazione del salto. NOTA: quando il salto NON viene fatto l'istruzione scelta (che viene sempre eseguita) però non deve creare problemi:

  * ad esempio può calcolare un valore non più necessario nel codice seguente.
  * l'importante è che non sia dannosa per l'esecuzione successiva.