Architettura MIC-2

Come rendere più veloce la CPU:

- ridurre il numero di cicli di clock nelle istruzioni;
- accorciare la durata del ciclo semplificando l'organizzazione;
- sovrappore l'esecuzione di più istruzioni (Pipeline);

MIC-1 risulta "lento" perché, oltre ad eseguire fetch e decode delle microistruzioni, ha un passaggio forzato di un operando al registro H.

La nuova architettura MIC-2 estende il BUS A a tutti i registri del cammino dei dati ed esegue il fetch anticipato delle istruzione grazie ad un componente chiamato IFU (Instruction Fetch Unit).

ESEMPIO

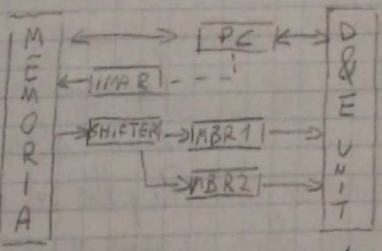
ILOAD MIC-1

```
H = LV
MAR = MBRU+H; rd
MAR = SP = SP+1
PC = PC+1; fetch; wr
TOS = MDR; goto Main1
```

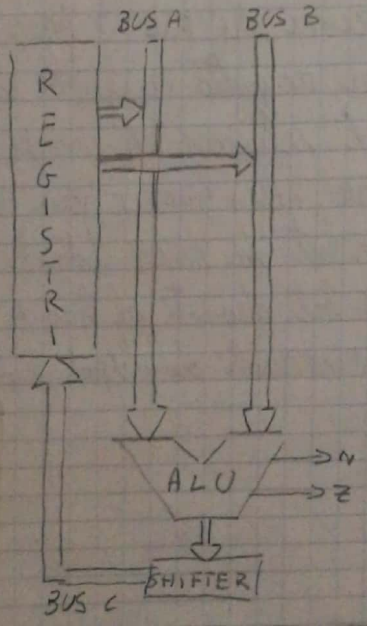
ILOAD MIC-2

```
MAR = MBRU+LV; rd
MAR = SP = SP+1
TOS = MDR, wr; goto Main1
```

IFU struttura



STRUTTURA MIC-2



IFU è un componente, non sincronizzato con D&EU (Decode & Execute Unit), che esegue il fetch delle istruzioni IJVM.
Quando D&EU legge il valore di MBR1/MBR2, l'IFU incrementa PC di 1 e aggiorna i valori di MBR1/MBR2 col byte successivo.
Allo scadere dello shifter, l'IFU invierà un nuovo fetch dalla RAM.

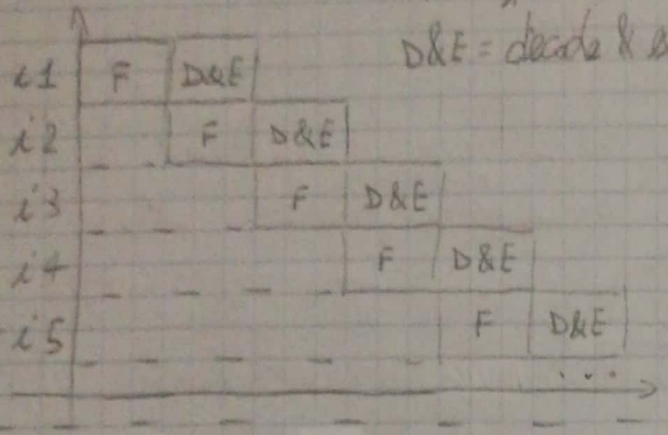
Nel caso D&EU modifichi il valore di PC, l'IFU aggiornerà in automatico IMAR e aggiornerà MBR1 e MBR2 dopo un ciclo di lettura. Prima di tutto questo, IFU e D&EU devono scambiarsi segnali di controllo per far attendere quest'ultima nel caso MBR1 e MBR2 non siano ancora pronti.

Pipeline nel MIC-2

Attraverso l'IFU è possibile sovrapporre i fetch delle istruzioni IJVM a operazioni di decodifica ed esecuzione.

ESEMPIO

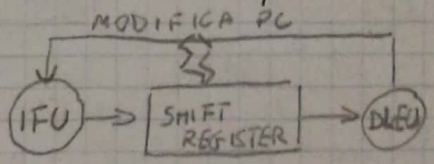
F = fetch
D&E = decode & esecute



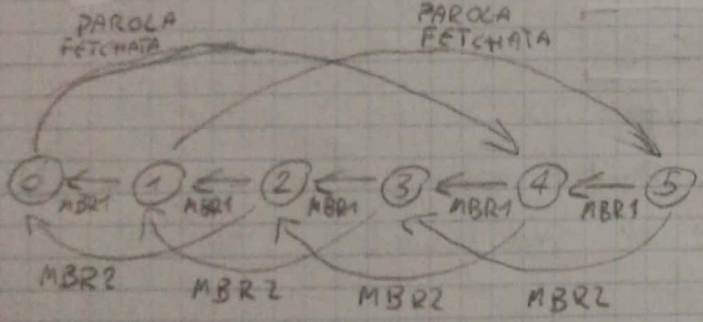
Lo schema di interazione tra IFU e D&EU è di tipo produttore-consumatore che è di tipo pipeline.

FUNZIONAMENTO

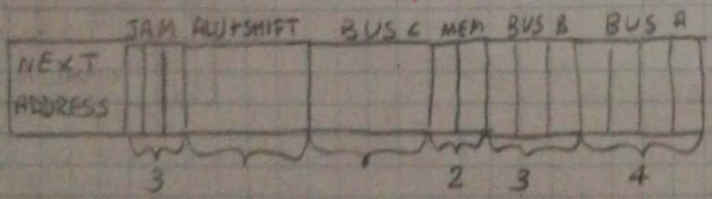
l'IFU "produce" le istruzioni da eseguire e le inserisce nello shift register solo quando vi è abbastanza memoria in quest'ultimo. D&EU "consuma" 1 o 2 bit alla volta quando lo shift register è pieno, altrimenti dovrà attendere. In più, vi è un feedback per gestire i salti: quando D&EU modifica PC, lo shift register si resetta e verrà riempito con le istruzioni a partire dall'indirizzo in PC.



Come si riempie lo shift register



Codifica delle microistruzioni



Le microistruzioni di MIC-2 presentano una struttura rispetto a quelle di MIC-1. Rispetto a quest'ultimo, i campi NEXT ADDRESS, JAM, ALU+SHIFT, BUS C e BUS B rimangono gli stessi, il campo MEM, invece, avrà un bit in meno dato che il fetch è ora eseguito dall'IFU. In aggiunta, vi sono 4 bit dedicati al bus A con relativo decoder per convertire i bit del campo in 11 segnali di output enable da inviare ai rispettivi registri.

Architettura CISC

L'architettura CISC (Complex Instruction Set Computer) è un tipo di architettura dove vi è un set di istruzioni più complesso e ricco, molto vicine ai linguaggi d'alto livello, la microprogrammazione per la realizzazione di istruzioni più complesse, uno spazio più ampio nella Control Store rispetto ai registri ed una pipeline inserita a posteriori.

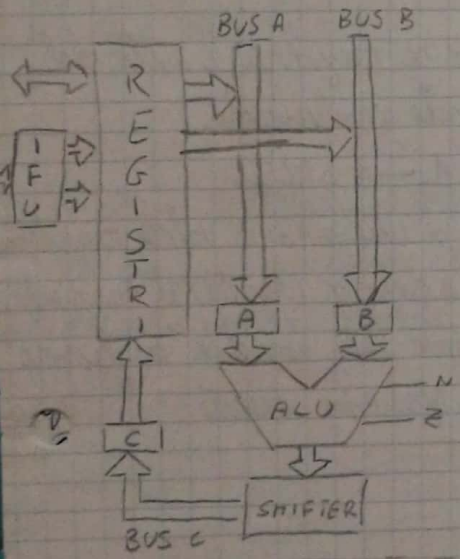
Architettura RISC

L'architettura RISC (Reduced Instruction Set Computer) è un tipo di architettura dove vi è un set d'istruzioni semplice ed essenziale, eseguibile direttamente dall'hardware, di cui due (LOAD e STORE) possono accedere alla memoria. Oltre a ciò, vi è un elevato numero di registri ed una struttura a pipeline come parte fondamentale del progetto.

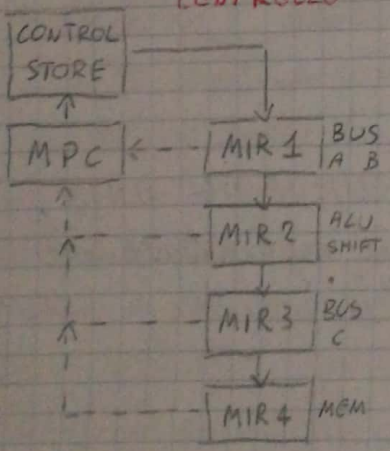
Architettura MIC-3

L'architettura MIC-3 è molto simile a MIC-2, con l'aggiunta di 3 latch nei bus

STRUTTURA



CONTROLLO



Grazie alla pipeline, l'esecuzione delle istruzioni risulta più veloce. L'esecuzione è divisa in 4 fasi:

1) scrittura dei registri selezionati nei latch A e B;
2) calcolo dei valori N e Z e scrittura del risultato dell'ALU nel BUS C;
3) scrittura del contenuto del latch C nei registri selezionati;
4) fase di read/write;

UNITÁ DI CONTROLLO

- se MIR1 contiene un salto semplice o a molte vie, MPC sarà caricato dal NEXTADDRESS oppure da MBR1;
- se MIR1 contiene un salto condizionato, si crea una bolla per poi essere risolta successivamente oppure si carica NEXTADDRESS in MPC;
- se viene risolta una bolla del salto condizionato, la destinazione viene calcolata da MIR2, N e Z;
- se c'è un conflitto si forza una bolla in MIR1.

DIPENDENZE

RAW: non si può leggere un operando non ancora aggiornato dall'istruzione precedente;
WAR: non posso aggiornare un registro se ancora in lettura;
WAW: gli aggiornamenti di un registro devono avvenire nel giusto ordine.

ESEMPI DI DIPENDENZE

H = PC - 1
PC = H + MBR2

RAW: devo aspettare che H si aggiorni.

MAR = SP = SP - 1; rd
MAR = SP = SP + 1

RAW di SP (vedi sopra)
WAR di rd e MAR nel caso duri più di un ciclo

LV = MAR = MDR; rd
MAR = LV + 1
PC = MDR; rd

WAR se rd dura più di un ciclo
RAW di LV (vedi sopra)
RAW tra rd e MDR

Branch Prediction

La Branch Prediction è una tecnica che permette di risolvere il problema dei salti incondizionati tirando a indovinare: se la previsione è corretta, non si perde alcun ciclo, altrimenti si esegue lo squashing, ovvero si svuota parte della pipeline e si riparte dall'istruzione giusta.

TECNICHE STATICHE:

- euristiche;
- con indicazione compilatore;

TECNICHE DINAMICHE

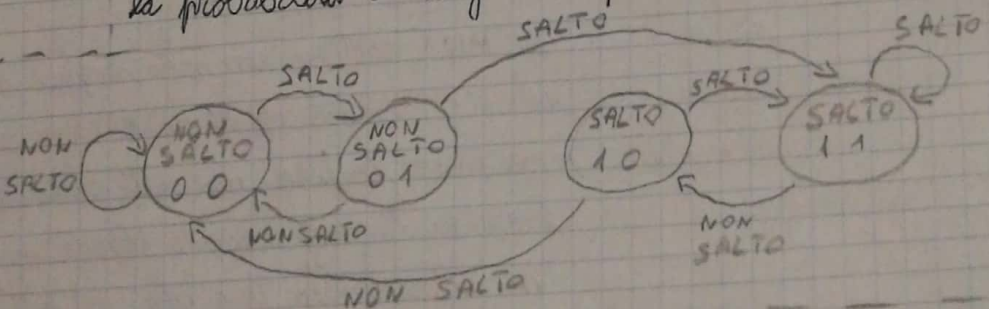
- tabella che registra le informazioni

PREVISIONE DINAMICA

Si utilizza una tabella dove si registra l'esito dell'ultima esecuzione di ogni salto condizionato.

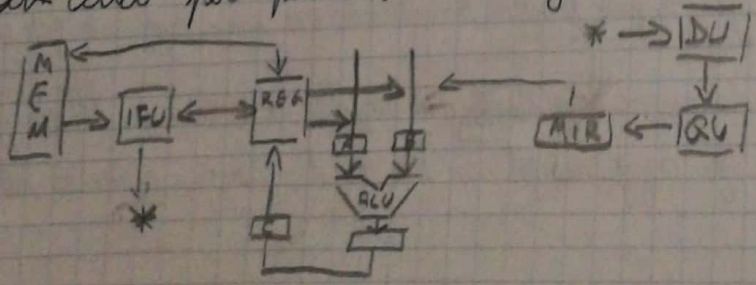
0 -> NON SALTO 1 -> SALTO

Quando si reincontra il salto, si utilizza il valore registrato precedentemente come previsione e dopo si registra il nuovo esito.
Con l'introduzione di un secondo bit, dovrebbe migliorare la probabilità di scegliere la previsione corretta.



Architettura MIC-4

L'architettura MIC-4 non differisce molto da ~~MIC~~ MIC-3 se non per l'aggiunta di un componente chiamato DECODING UNIT che prende in input l'output dell'IFU e, attraverso la QUEUEING UNIT, trova l'istruzione ISVM interessata ed esegue ogni microistruzione nella ~~coda~~ coda per poi essere eseguite dal resto dell'hardware.



Cache

La cache è una memoria di piccola capacità ma molto più veloce rispetto a quella principale che permette di ridurre i tempi di accesso alla memoria.

NOTA: la cache risulta efficace solo solo se i dati e le istruzioni da eseguire si trovano spesso all'interno di essa. Questa condizione è vera grazie alle proprietà di località.

PROPRIETÀ LOCALITÀ TEMPORALE

Se in un tempo t accedo ad un dato/istruzione nell'indirizzo i, è probabile che in futuro accederò nuovamente a quell'indirizzo.

In un loop, ad esempio, riesegue ogni volta le stesse istruzioni e riutilizza le stesse variabili.

PROPRIETÀ LOCALITÀ SPAZIALE

Se in un tempo t accedo ad un dato/istruzione nell'indirizzo i, è probabile che in futuro accederò a indirizzi vicini ad i

Per questo motivo si prelevano istruzioni in sequenza e si utilizzano variabili locali in posizioni vicine nello stack.

Funzionamento della cache

Ad ogni lettura in memoria, si controlla la parola interessata è presente nella cache: se sì, si preleva da lì, altrimenti viene prima caricata nella cache ed in seguito verrà portata alla CPU.

TEMPO MEDIO DI ACCESSO ALLA MEMORIA

$$E(t) = t_c + (1-h)\cdot t_m$$

- h è l'hit rate, cioè la percentuale di accessi alla memoria serviti dalla cache;
- t_c è il tempo di accesso alla cache;
- t_m è il tempo di accesso alla memoria centrale.

Se h tende a 1 allora $E(t)$ tende a t_c, in quanto riusciamo ad avere tempi di accesso medi vicini a quelli della cache.

STRUTTURA INTERNA

La cache è suddivisa in linee e ognuna di esse può contenere un blocco della RAM. Ogni blocco è identificato con un numero NB mentre ogni byte all'interno di esso viene rappresentato con un offset.

$$NB = \frac{\text{indirizzo}}{\text{dim. blocco}}$$

$$\text{offset} = \text{indirizzo} \,\%\, \text{dim. blocco}$$

La linea è la parte di cache che può contenere un blocco ed è divisa in 3 campi:

- Valid: bit che indica se la linea è piena;
- Tag: indica quale blocco è contenuto nel campo Data;
- Data: campo che contiene il blocco di memoria.

Funzionamento cache

Si può inserire un blocco in qualunque linea oppure si può calcolare con l'indirizzo.
Per identificare un blocco si calcola il TAG utilizzando NB

$$NB = I / dim.blocco$$

Per rimpiazzare un blocco o si sceglie in modo casuale oppure si determina attraverso l'indirizzo.
Infine vi sono 2 modi per scrivere in memoria:
- write through: aggiorna cache e RAM;
- write back: aggiorna cache e aggiornerà RAM quando la linea verrà rimpiazzata.

Cache associativa a n vie

la cache associativa a n vie è l'equivalente di n cache a corrispondenza diretta allineate e, in più, attenua i problemi di conflitto tra i blocchi associati alla stessa riga. La ricerca avviene in parallelo nelle n vie

FUNZIONAMENTO

l'inserimento dei blocchi di memoria nella cache è analogo alla corrispondenza diretta, con la differenza che c'è più libertà grazie al fatto che si deve scegliere una linea tra le altre nell'insieme.
Per identificare i blocchi (vedi corrispondenza diretta). Per rimpiazzare i blocchi, il procedimento è lo stesso della corrispondenza, ma anche qui bisogna scegliere una linea.
La scrittura in memoria è analoga alla cache normale.

Cache a n vie

la cache a n vie è equivalente a n cache allineate e permette di contenere più blocchi in una sola linea.

Cache a corrispondenza diretta

La cache a corrispondenza diretta è un tipo di cache dove ogni linea può contenere un solo blocco.

FUNZIONAMENTO

Per inserire un blocco di memoria nella cache, si calcola la riga.

$$riga = NB \,\%\, num.righe$$

Per identificare un blocco di memoria nella cache, si calcola il TAG (vedi funzionamento cache).
Per rimpiazzare un blocco, si calcola la riga dove è stato inserito.
La scrittura in memoria è analoga alla cache normale.

FUNZIONAMENTO CACHE ASSOCIATIVA PURA

UNA RIGA ⊆ TUTTI BLOCCHI UNA RIGA ⊆ TUTTE LINEE

In questo caso, posso mettere i blocchi in ogni riga libera più comoda. (compresa la scelta di chi rimpiazzare) e, come nel caso precedente, ha più flessibilità. Ho più, minimizza i conflitti.
La scrittura è analoga alla cache normale, con alcune precauzioni da prendere.

ALTRI UTILIZZI DELLA CACHE

la cache è anche utilizzata anche dagli algoritmi di predizione dei salti per decidere in quale linea cercare accede confrontare il tag con parte dell'indirizzo per verificare i dati e infine, si scrivono le informazioni per la predizione e per il ritorno.

ISA (Instruction Set Architecture)

L'ISA è il livello che definisce il linguaggio macchina eseguibile da un processore, esso rappresenta l'interfaccia tra software e hardware ed è in grado di definire un set d'istruzioni elementari chiamato "linguaggio macchina".

NOTA: Anche se è possibile eseguire direttamente file scritti con linguaggi ad alto livello, questo risulta svantaggioso perché si favorisce l'interpretazione alla compilazione, rendendo l'esecuzione inefficiente.

L'obiettivo di ISA è quello di trovare un compromesso tra i programmatori (che implementano i compilatori) e i progettisti hardware ma, per esigenze di mercato, il livello deve essere compatibile con le versioni precedenti del software (questo è possibile grazie all'emulazione di istruzioni "vecchie" nella nuova architettura).

In alcune architetture, il livello ISA è definito da un consorzio di produttori.

- VANTAGGIO: più produttori realizzano la medesima architettura;
- SVANTAGGIO: l'architettura non è più proprietaria.

ISA permette di definire almeno due livelli di esecuzione:

- KERNEL MODE: pieno controllo sulle risorse;
- USER MODE: controllo limitato sulle risorse.

REGISTRI

Ogni elaboratore possiede un certo numero di registri (di solito pochi decine), su cui opera l'ALU, essi possono essere:

- tutti visibili alla microarchitettura;
- alcuni visibili a ISA.

Alcuni registri, come PC e SP, hanno funzioni specifiche. La principale funzione di un registro è quella di fornire un'elevata accessibilità ai dati.

Un registro chiamato FLAG o PSW contiene i condition code dell'ALU:

- N: risultato negativo;
- Z: risultato zero;
- V: overflow;
- C: risultato con riporto;

essi sono importanti perché determinano la condizione di un salto condizionato.

ORGANIZZAZIONE DELLA MEMORIA

La memoria è divisa in celle (solitamente di 8 bit) aventi indirizzi consecutivi. Una cella da 8 bit è chiamata byte, più byte formano una parola (solitamente grande 4 o 8 byte) e più parole formano un blocco.

Per convenienza, le parole/blocchi sono allineate.

Gran parte delle architetture ha un solo spazio d'indirizzamento lineare di 2^{32} o 2^{64} byte, tuttavia ne esistono altre che hanno spazi diversi per dati e istruzioni (indirizzamento memorie più grandi e gli accessi sono più controllati, se specificati i permessi d'accesso di ogni segmento).

ACCESSO ALLA MEMORIA

Per garantire la corretta esecuzione delle istruzioni d'accesso alla memoria, si utilizzano 3 differenti approcci:

- Lasciare al compilatore il compito di memorizzare i dati tramite SYNC;
- Realizzare, in hardware, la verifica automatica di RAW e WAR durante una LOAD/STORE.
- Sequenzializzare tutte le LOAD/STORE.

SET D'ISTRUZIONI

- LOAD/STORE permettono di leggere e scrivere in memoria;
- MOVE copia i dati da un registro all'altro;
- ARITMETIC esegue operazioni aritmetico-logiche;
- BOOLEAN esegue operazioni booleane;
- SALTO CONDIZIONATO E INCONDIZIONATO;
- ISTRUZIONI I/O.

DATI NON NUMERICI (BOOLEANI)

I valori booleani possono essere rappresentati su un singolo bit anche se, in pratica, viene utilizzato un intero byte o una parola, il motivo è impossibile e non conveniente indirizzare un bit, quindi, per convenienza, si è deciso di interpretarli nel seguente modo:

- FALSO quando il byte o la parola è 0;
- VERO quando il byte o la parola è diverso da 0.

Si recupera efficienza nel bit-map dove si utilizza un'intera parola per memorizzare i booleani.

DATI NON NUMERICI (BOOLEANI)

I caratteri rappresentano un tipo di dato importante anche se sono pochi i calcolatori con un'hardware in grado di manipolarli. Le codifiche principali sono ASCII (7bit) e UNICODE (16 bit).

PUNTATORI

I puntatori sono tipi di dati, fatti di indirizzi di memoria, che permettono di fare riferimento a diverse locazioni di memoria con la stessa istruzione.

TIPI DI DATI

Un aspetto importante di un'architettura è la gestione di diversi tipi di dati in cui:

- se le istruzione prevedono un certo formato, deve essere rispettato.
- Se un dato per essere rappresentato ha bisogno di specifici requisiti ma l'hardware non lo supporta, si procede via software con una diminuzione dell'efficienza.

DATI NUMERICI (INTERI)

Per rappresentare un numero intero, i calcolatori utilizzano la notazione "complemento a 2", sia che si tratta di un numero con segno che di un numero senza segno. ASSOLUTO

4 Byte → con segno → Valore massimo $2^{31} - 1$

4 Byte → senza segno → Valore massimo $2^{32} - 1$

DATI NUMERICI (NON INTERI)

I numeri non interi si rappresentano con la tecnica "floating point" che può andare da 32 bit a 128 bit in base alla precisione.

FORMATO DELLE ISTRUZIONI

La codifica binaria di un istruzione è formata da:

- un codice operativo;
- campi indicare i dati e memorizzare i risultati.

Il problema dell'indicazione del luogo da cui prelevare o memorizzare si chiama indirizzamento, il numero di indirizzi in una istruzione varia da 0 a 3.
Un'altro problema è la lunghezza: istruzioni corte occupano meno spazio per l'indirizzamento CPU-MEMORIA sono più difficili da decodificare e riducono la possibilità che la CPU rimanga in fetch per le nuove istruzioni.
Anche il codice operativo è preferibile corto perché garantisce una decodifica veloce ed hanno grande spazio d'indirizzamento. Le istruzioni a lunghezza fissa sono più efficienti di quelle variabili nella gestione. La complessità del formato di un istruzione dipende dal codice operativo e dalla modalità di specifica degli operandi.

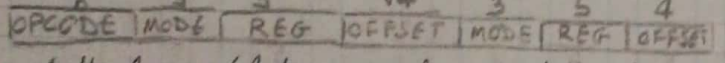
RIDURRE LA LUNGHEZZA DI UN'ISTRUZIONE

Per ridurre la lunghezza di un'istruzione, si utilizzano i registri:

- i dati più frequenti vengono spostati in registri di lavoro;
- utilizzare registri per puntare nella memoria;
- utilizzare operandi impliciti (stack, registri specializzati, medesimo operando sia per input che per output).

Ortogonalità tra opcode e indirizzamento

esempio: architettura a due indirizzi per istruzioni aritmetico-logiche:



NOTA: tutte le modalità si applicano a tutti gli opcode, indipendentemente dal codice operativo.

MODE può codificare fino a 8 modalità d'indirizzamento

ISTRUZIONI I/O

Ogni dispositivo possiede un'interfaccia chiamata controller che, attraverso apposite istruzioni (IN/OUT) oppure con delle LOAD/STORE (Memory Mapped I/O), è possibile modificare i valori dei registri al suo interno.

MEMORY MAPPED I/O

Il memory mapped I/O è una tecnica che, attraverso l'utilizzo di LOAD/STORE a indirizzi associati, modifica i valori dei registri nel controller.
Il controller contiene un decoder che riconosce gli indirizzi assegnati e restituisce/riceve dati ai suoi indirizzi.

INDIRIZZAMENTO

esistono diverse modalità d'indirizzamento:

- IMMEDIATO: operando contenuto nell'istruzione;
  `BIPUSH 10`
- DIRETTO: operando all'indirizzo specificato;
  `LOAD R1, 0x302815` R1 ← m.[0x302815]
- TRAMITE REGISTRO: operando contenuto nel registro;
  `ADD R1, R2, R3` R1 ← R2+R3
- INDIRETTO: operando all'indirizzo contenuto nel registro;
  `MOVE R1, (R2)` R1 ← m.[R2]
- INDICIZZATO: operando all'indirizzo ottenuto da un registro e una costante;
  `MOVE #A(R2), R1` m.[#A+R2] ← R1
- BASE-INDICE: operando all'indirizzo ottenuto da 2 registri;
  `MOVE R3, (R1+R2)` R3 ← m.[R1+R2]

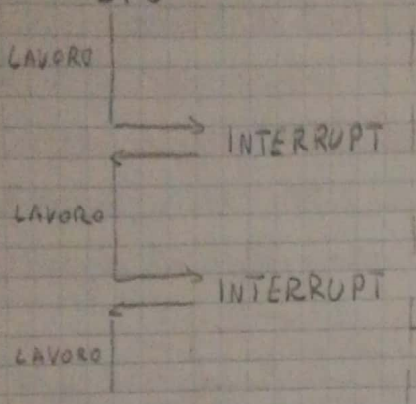
TIPI DI ISTRUZIONI I/O

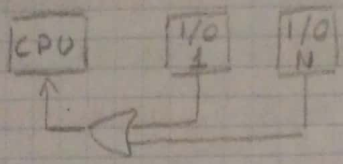
- BUSY WAITING: Il busy waiting è una tecnica dove la CPU verifica ciclicamente la disponibilità dei dispositivi I/O e, quando disponibile, procede alla modifica dei registri nel controller. L'unico svantaggio è che tiene la CPU impegnata nel controllo ciclico quando effettivamente può fare altro.
- INTERRUPT: La periferica I/O manda un segnale alla CPU la quale interrompe quello che stava facendo per dedicarsi ad esso. Finito l'interrupt, la CPU riprende il precedente lavoro. La gestione degli interrupt deve essere trasparente al lavoro della CPU.

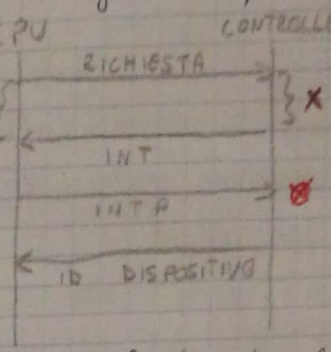
FUNZIONAMENTO INTERRUPT

Nei registri di controllo dei dispositivi sono presenti interrupt bit che servono per attivare gli interrupt da parte dei dispositivi. Questi bit sono in AND col bit di stato e connessi alla linea INT della CPU la quale, quando il valore della linea è 1, interromperà quello che stava facendo per eseguire l'interrupt.

SCHEMA FUNZIONAMENTO INTERRUPT







PROBLEMI INTERRUPT

Dato che le periferiche sono connesse in logica OR alla linea INT della CPU, il dispositivo interessato viene individuato dal dispositivo tramite Polling o Interrupt Vettorizzato.*

° la CPU esegue un codice di un altro processo.

× Il dispositivo esegue l'operazione I/O richiesta.

⊗ segnale di acknowledgment.

*L'interrupt vector è una tabella dell'OS indirizzata in base al tipo di dispositivo dove è indicato l'ID.

Nel caso vi è l'arrivo di un altro interrupt, deve essere possibile disattivare gli interrupt:

- è opportuno definire livelli di priorità per permettere un'interrupt ad alta priorità rispetto ad un altro anche se in esecuzione (Nel caso vi siano interrupt con livelli divergenti).
- In altri casi la CPU non è in grado di gestire la priorità in modo autonomo e quindi c'è bisogno di un controller esterno che esegua questo lavoro.

Nel caso in cui più interrupt arrivano in contemporanea, la CPU riceve un solo segnale:

- Se si utilizza il polling, tutto dipende dall'ordine in cui le periferiche vengono scansionate;
- Se viene utilizzato l'interrupt vettorizzato, si imposta un protocollo di comunicazione tra CPU e dispositivo: quando la prima ha ricevuto l'interrupt poi è pronta a gestirlo, invia un segnale di acknowledgment e, quando ricevuto dal controller, esso invia l'ID del dispositivo. Nel caso in cui l'acknowledgment venga propagato in sequenza dai controllori e assorbito dal primo controller con interrupt pendente, tutto dipende dall'ordine dei dispositivi collegati in daisy chain.

DISABILITARE UN INTERRUPT

Quando la CPU deve eseguire codice senza interruzioni, gli interrupt pendenti vengono ignorati fino alla riabilitazione.

GESTIONE DI UN INTERRUPT

La CPU, per gestire gli interrupt, deve riuscire a rilevarli, leggere l'ID e abilitarli/disabilitarli impostando un bit su un registro. ISA dovrà quindi:

- avere un'istruzione di ritorno da interrupt che ripristinerà lo stato salvato prima del richiamo;
- avere istruzioni di abilitazione/disabilitazione dell'interrupt;

NOTA: queste istruzioni devono essere usate con attenzione e per questo sono utilizzabili solo dall'OS, occorrerà quindi eseguire una microistruzione "modo Kernel" prima di eseguire l'interrupt, al fine si ritornerà a "modo utente".

DIRECT MEMORY ACCESS

Il controller DMA è in grado di gestire lo spostamento dati dalla memoria al buffer e viceversa. Per evitare conflitti, dato che il bus è condiviso, occorre un arbitro per gestirli.