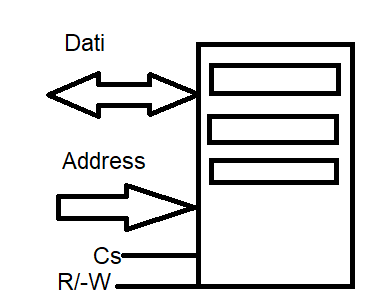
Abbiamo visto come un modulo di memoria RAM può essere interfacciato verso altri dispositivi.



La prima forma di dispositivo di tipo RAM fu ideata da Von Neumann. Secondo la sua progettazione, un calcolatore doveva essere composto di una CPU, che faceva da master, da una RAM, che faceva da slave, e un dispositivo di ingresso (tastiera) e uno di uscita (ai nostri giorni un display). Questo dispositivo così schematico in realtà rispecchia il funzionamento delle macchine anche al giorno d’oggi.

All’interno della memoria Ram troviamo una codifica delle istruzioni da eseguire e anche i dati sul cui il programma lavora. Inizialmente non c’era distinzione tra le due forme di informazioni, erano entrambe codificate sullo stesso numero di bit. Il calcolatore per trovare e eseguire le istruzioni segue quindi una serie di passi: il primo detto fetch, il secondo decode e il terzo execute. A supporto di eseguire i programmi vengono implementati due (in realtà di più) registri nel processore: il contatore di programma (PC) e il registro delle istruzioni (IR. I primi due possono essere registri di tipo D (cioè del tipo che ricevono un numero e lo tengono memorizzato finchè non viene dato l’ordine di memorizzarne un altro).

Durante la fase di fetch, la cpu manda il numero contenuto nel PC come indirizzo e invia Cs e R/-w positivi (accedendo in lettura). Quando la RAM restituisce il valore della cella, questo viene memorizzato nell’IR, dopodiché il valore nel PC viene incrementato di 1 (per passare all’istruzione successiva).

Dopo la fase di fetch avviene la fase di decodifica. La Cpu decodifica il numero salvato nell’IR per capire quale istruzione seguire (poiché a ogni istruzione è assegnato un indice).

A questo punto avviene la fase di esecuzione: vengono eseguite grazie alla ALU le istruzioni decodificate. I risultati di operazioni possono essere (e in generale vengono) salvate in un altro registro: l’accumulatore.

Finita una esecuzione, la Cpu ritorna in fase di fetch, tuttavia essendo aumentato il valore salvato nel PC adesso verrà decodificata ed eseguita l’istruzione successiva.

Questo dà origine a un loop che va avanti fintanto che la macchina rimane accesa e che permette l’esecuzione in ordine di tutte le istruzioni presenti in celle consecutive della RAM. È quindi eseguibile un programma sequenziale.

Per scrivere un programma sequenziale, bisogna innanzitutto definire il punto di partenza della prima istruzione: lì e nelle celle successive vanno inseriti i codici operativi delle varie istruzioni.

Per gestire i casi in cui si devono ripetere più volte le stesse istruzioni vengono introdotte le istruzioni di controllo di flusso (flow control) che permettono al programma di avere delle deviazioni. La prima di queste istruzioni è l’istruzione di “salto”: il suo scopo è cambiare il valore nel PC e inserirvi qualcos’altro (per esempio una costante). L’esecuzione andrà più o meno così:

PC = 000000000000

#Fetch:

IR <- RAM[PC]; PC <- PC+1;

#Decode(si ipotizzi I registri siano organizzati su 16 bit, di cui alcuni, tipo 3, sono dedicati al codice operativo dell’istruzione e gli altri 13 per codificare altre informazioni -> nel caso del “salto” in quei 13 bit va la rappresentazione binaria della costane da inserire nel PC):

PC = 0000000000001

IR = [010-0000000000010];

#Execute:

PC <- IR(const) = [0000000000010];

#Fetch successivo:

IR <- RAM[PC]; PC <- PC+1;

(adesso in IR c’è RAM[0000000000010])

Nel nostro esempio abbiamo dedicato 3 bit alla codifica del codice operativo di un’istruzione, il che vuol dire che l’elenco delle istruzioni comprensibili dalla macchina contiene al massimo 2^3 (=8) elementi.

Oltre all’istruzione di salto possiamo avere l’istruzione di salto condizionale (si controlla che una situazione sia vera e nel caso si effettua un salto). Già questi pochi elementi permettono di creare programmi contenenti deviazioni, loop ecc. Abbiamo poi delle istruzioni di tipo Logico, istruzioni di tipo Aritmetico, istruzioni di scorrimento (di shift) ecc.

Per quanto riguarda la struttura tipica di un’istruzione di tipo aritmetico ci servono in genere due operandi: il risultato deve poi essere memorizzato da un’altra parte (quindi gli operandi in realtà sarebbero tre). Questo ci porta a una serie di complicazioni nella codifica delle istruzioni, questo perché ci serve tanto spazio da poter dedicare ai vari pezzi (il codice operativo e gli indirizzi dei tre operandi). Per poter usare i bit a disposizione in modo ottimale (visto che nel nostro esempio ne abbiamo presi 16) si potrebbe dividerli in maniera eguale, ma 4 bit a ogni pezzo vorrebbe dire: massimo 16 istruzioni riconoscibili, massimo 16 celle di memoria, massimo 16 celle di memoria, massimo 16 celle di memoria. Quindi, tipicamente, le istruzioni con vari operandi vengono implementate usando i registri interni del processore (l’/gli accumulatore/i): si prevede un modo di indirizzamento di tipo registro, nel nostro esempio possiamo avere un massimo di 16 registri interni (poiché 2^4 è il numero di indirizzi che si possono codificare). I registri “accumulatori” vengono formalmente chiamati registri di uso generale.

Per caricare valori dalla RAM in un registro di uso generale è necessaria l’istruzione “load”: essa avrà un codice operativo che la identifichi (4 bit), l’indirizzo di un registro (4 bit, a meno che ci sia un solo registro accumulatore prefissato per queste operazioni, allora non bisogna dedicare alcun bit a questa fase) e l’indirizzo della cella della RAM da cui prelevare il valore (prende tutti i bit rimanenti 8-12), -> Indirizzamento diretto.

Oltre al modo di indirizzamento diretto sono stati inventati tanti modi di indirizzamento diversi: ad esempio il meccanismo di indirizzamento indiretto (chiamata mediante registro o mediante cella di memoria RAM). Nel caso dell’indirizzamento indiretto non viene specificato l’indirizzo nei bit rimanenti, ma viene indicato un posto nel cui si trova l’indirizzo (ad esempio in un altro registro della Cpu o in un'altra cella della RAM).

Nel caso dell’indirizzamento indiretto tramite registro l’execute sarà di questo tipo:

PC <- RAM[reg[IR(address)]]

Altrimenti nel caso di indirizzamento indiretto tramite cella di memoria RAM (decisamente penalizzante dal punto di vista dell’efficienza):

PC <- RAM[RAM[IR(address)]]

Il modo di indirizzamento indicizzato, invece, richiede che vengano specificati un registro sorgente e un valore costante. A quel punto inserisce in un registro il contenuto di una cella della RAM il cui indice è il contenuto del registro sorgente + il valore della costante.

Reg <- RAM[Reg-Sorg + const];

Questo è uno dei metodi preferiti al giorno d’oggi per indirizzare la memoria, essendo più generale rispetto agli altri.

L’ultimo modo di indirizzamento tra quelli attualmente più in uso è il modo di auto-incremento. L’idea è quella di incrementare l’indice del registro e utilizzare il valore successivo (o decrementarlo e quindi usare il valore precedente) per indirizzare la RAM.

(R <- R +- 1)

R <- R + 1; Dest <- RAM[R];

Ci sono casi in cui un’operazione può essere eseguita su un solo registro, per esempio se l’auto-incremento opera su un solo registro questo non necessita di essere indicato esplicitamente (e quindi di occupare bit).

Le operazioni di scorrimento invece possono far scorrere i bit di un numero a destra o a sinistra, essi però sono principalmente applicabili a numeri senza segno: questo perché nel caso di numeri con segno un overflow durante lo scorrimento a sinistra potrebbe causare un inversione di segno -> ciò può venir notato dalla ALU che lo riporta in uno dei suoi fili complementari, questa può essere una condizione per l’attivazione di un Salto condizionale.

Un’altra forma di scorrimento è quella di rotazione: la cifra che andrebbe persa viene inserita all’altro capo del numero.