Oggi introduciamo i dispositivi di ingresso ed uscita. Quindi la cosa si complica, poiché si vanno ad aggiungere alla CPU e alla memoria RAM.

Il metodo più conveniente per inserire tali dispositivi (raggiunto dopo una serie di tentativi) ci si è resi conto che esso era di usare lo stesso meccanismo che il processore usa per comunicare con la memoria per far comunicare col processore i dispositivi di input/output. Come ben sappiamo la Cpu è dotata di fili unidirezionali che trasportano un indirizzo verso la RAM e di altri fili bidirezionali che trasportano i dati per le operazioni di scrittura e lettura (+ i due fili Cs e R/-W).

Quindi la Cpu si comporta come master e qualsiasi dispositivo ad esso legato come uno slave.

L’idea sarebbe di implementare un’operazione di tipo READ per accedere al dispositivo di input (che nel caso più semplice può essere un tastierino alfanumerico, nel quale l’utente può premere un tasto e invia il codice corrispondente al pulsante premuto). Ma mentre le informazioni tra CPU e RAM venivano codificate su 16/32 bit, dal dispositivo di input è più efficiente invece leggere le informazioni inserite su un byte (essendo codificate sul codice ASCII a 7 bit ed inserite un dato alla volta).

Lo stesso vale per le comunicazioni col dispositivo di output, l’operazione di tipo WRITE permette di trasportare un byte di dati dalla CPU al dispositivo di output.

I due dispositivi, quindi, potrebbero aver al loro interno un registro contenente 8 bit (1 byte) sul quale verrebbe salvata l’informazione prima di inviarla / mostrarla all’utente (si lavorerebbe sempre su un dato alla volta).

Si è però introdotto un piccolo trucchetto: la mappatura in memoria dei registri. Si prende la RAM e si “tagliano via” una certa quantità di celle di memoria (quelle che ci servono e ai cui indirizzi la memoria RAM ora non risponderà più). Quindi se abbiamo una RAM di 64 Kb le celle a cui può accedere la RAM, che di norma vanno da 0 a 65534, arrivano fino a poco meno (65532, o 65533 se si accede un byte alla volta): ci restano così liberi 2 byte. Possiamo quindi prendere il nostro registro del dispositivo di input e piazzarlo al posto di uno dei byte della RAM eliminati (il 65534). L’indirizzo 65535 corrisponde invece al registro del dispositivo di uscita.

A questo punto il nostro processore per leggere un carattere dal dispositivo di input deve effettuare la lettura della cella 65534, mentre per scrivere sul dispositivo di output dovrà accedere in scrittura all’indirizzo 65535: questa mappatura serve quindi per permettere al processore di accedervi in maniera semplice.

Sembra che ci sia lo spreco di un bit (essendo la codifica su 8 bit ma la tabella ASCII su 7 bit), ma anche quello può essere usato in maniera efficacie. Per tenere conto del fatto che il processore deve leggere solo quando il programma gli dice di leggere bisogna considerare che non è detto che l’utente abbia premuto un tasto sulla tastiera: il dispositivo si deve chiedere se quello che è nel registro di input è una vera informazione oppure è un dato lì a caso, ecco che torna in gioco il bit in più.

Possiamo interpretare che il bit più significativo (il bit “di segno”) sia 0 (cioè positivo) se è stato effettivamente premuto un pulsante (e quindi si sta leggendo un carattere valido), mentre sia 1 (cioè negativo) quando invece non è stato premuto nessun carattere (e quindi il dato non è valido). Gli altri 7 bit saranno la codifica ASCII del bit premuto.

Se la memoria RAM è indirizzata su 16 fili, per capire quando si sta lavorando con un dispositivo di input o output anziché con una cella standard, basta eseguire una funzione AND sui 15 bit più significativi dei 16 che codificando l’indirizzo (se sono tutti 1 vuol dire che si sta provando ad accedere a una delle ultime due celle e quindi si sta lavorando con un dispositivo di input output). Ovvietà: anche i registri dei dispositivi I/O sono connessi a dispositivi a 3 stati, essendo connessi agli stessi fili per i dati delle altre celle della RAM.

Una problematica da notare è cosa succede se durante l’esecuzione del programma l’utente digita più tasti (e nel frattempo il processore non ha effettuato nessuna lettura): in questo caso i caratteri precedenti vengono persi e nel registro di input rimane solo la codifica dell’ultimo carattere digitato (se il processore non è abbastanza veloce può darsi che vengano persi dei caratteri).

Riassumendo: si effettua la lettura da dispositivo di input quando:

* Cs = 1
* R/-W = 1
* AND(Address[0:15]) = 1
* Address[15]= 0
* Input[0] = 0

Tutto ciò permette ai processori di essere progettati in maniera generica, senza particolari istruzioni per la gestione di input e output.

Per quanto riguarda il dispositivo di uscita, esso deve poter supportare l’operazione di scrittura. Con esso, la cpu interagisce quando:

* Cs = 1
* R/-W = 0
* AND(Address[0:16]) = 1

Nel caso del dispositivo di output , il suo registro viene connesso al bus dati e ciò permette di trascriverli su di esso (essendo appunto mappato sulla RAM). In questo caso il bit “di segno” può servire per gestire il tempo necessario al dispositivo di output per portare a termine un’operazione di scrittura. Se esso equivale a 0, ciò corrisponde all’avvio di un’operazione di stampa. Se a questo punto si effettuasse subito un’operazione di scrittura, senza dare al dispositivo di output di emettere il risultato, i valori ancora in fase di stampa verrebbero sovrascritti e quindi persi. Per ritardare l’operazione di scrittura si cambia/mantiene il valore del bit di segno a 0 finché il dispositivo non è nuovamente pronto per stampare un nuovo carattere (quando è libero il bit di segno torna a 1). All’inizio, quindi, il bit di segno è = 1, poi diventa 0 durante la stampa e ritorna a 1 quando è pronto per subire nuovamente una scrittura.

Prima di effettuare la scrittura, quindi, avviene una fase di lettura che controlli che il primo bit sia effettivamente 1 e non 0.

A questo punto possiamo provare ad abbozzare un piccolo programma che ci consenta di leggere e scrivere attraverso i dispositivi di ingresso e di uscita.

Supponiamo di voler leggere qualche cosa dal dispositivo di ingresso. La lettura deve avvenire velocemente per evitare perdita di caratteri, quindi:

//Leggi Reg 65534 -> R0

MOVE RAM[65534], R0 //C’è estensione del segno (poiché nel registro a 16 bit stiamo inserendo 8 bit)

BNEG -2 //se il valore nel registro è negativo salta all’istruzione precedente (PC viene incrementato di 1, quindi bisogna diminuirlo di 2) -> è come un ciclo do-while. Nota: B sta per BRANCH – viene eseguito se il valore precedente è NEG (valore che è controllabile attraverso il registro di Stato).

Ci sono due modi per fare un salto: Il salto assoluto (con indirizzo esatto di una cella a cui deve puntare il PC) e il salto relativo, che modifica il valore attuale del PC con un valore, come +1, +2, -2 ecc.).

Il salto relativo, nella generalità dei casi, è più comodo/utile.

Relativamente semplice da realizzare, questo codice è anche veloce da eseguire. Tuttavia, ha come effetto collaterale di tenere occupato il processore in modo ciclico finché l’utente non si decide a premere un tasto (attesa attiva / busy waiting), poiché il processore anziché fare cose utili sta aspettando.

Questo è solo uno dei modi possibili di sincronizzazione tra utente e processore e ovviamente non è molto pratico (sebbene sia il più veloce immaginabile per accedere al dispositivo di input). Questo perché se il processore può essere impiegato per eseguire altri programmi la scelta del busy waiting non è affatto opportuna (se il processore può eseguire qualche milione di istruzioni al secondo e l’utente può digitare al massimo una decina di caratteri al secondo c’è uno spreco di risorse nel far attendere il programma per ogni pressione di pulsante).

Dopo il BNEG inizierebbe la parte di programma che usa il carattere che è stato letto dal dispositivo in ingresso (inizierebbe quindi la decodifica del carattere).

Per quanto riguarda l’operazione di scrittura di un carattere:

MOVEB RAM[65535], R1 //MOVEB sta per MOVE BYTE

BPOS -2 //busy waiting se il valore del bit più significativo è 0: cioè positivo

MOVEB R2, RAM[65535]

Se volessi stampare una sequenza di caratteri: es. stringa “ciao”, dovrei innanzitutto avere 4 byte occupati ciascuno con uno dei caratteri da stampare, dopodiché dovrei mandare uno alla volta i caratteri in scrittura al dispositivo di output. SI avrebbe così la stampa più veloce possibile, ma d’altro canto questa struttura causa uno spreco di risorse del processore.