di Andrea de Prisco

Microistruzioni, Microregistri, Microprogrammi

Come già anticipato sul numero scorso, l'argomento di questo mese riguarda il livello di microprogrammazione di un processore: in che modo sono implementate le singole istruzioni di macchina che, dicevamo, possono anche non essere direttamente eseguite dal processore.
In piccolo riabbiamo quello che succede tra livello Basic di un qualsiasi personal computer e linguaggio macchina: si preleva un comando, si interpreta nel livello di programmazione sottostante, se ne preleva un altro e così via. La differenza sta nel fatto che stavolta l'interprete è per linguaggio macchina e non per un linguaggio ad alto livello.

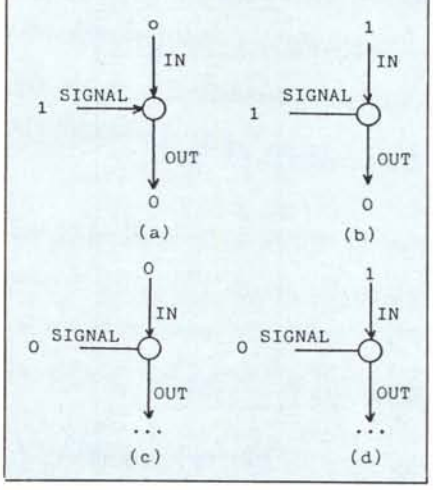
Confine tra hardware e software

Quello visto lo scorso mese era un esempio di linguaggio macchina di una ipotetica CPU. Abbiamo parlato delle sue istruzioni di macchina, come le operazioni per addizionare due numeri, saltare da un punto all'altro di un programma, mettere qualcosa in pila, spostare contenuti di celle ecc. ecc. Come venivano eseguite queste istruzioni all'interno della CPU, volutamente, non ne abbiamo parlato.

I primi processori, eseguivano le operazioni di macchina direttamente in hardware: esisteva, all'interno della CPU, un circuito elettronico per ogni operazione, il quale era attivato ogni qual volta che l'operazione stessa era invocata. Se ad esempio si trattava di fare la somma di due numeri, il circuito «somma» all'interno del processore veniva attivato e provvedeva circa il da farsi. Similmente per le altre operazioni avevamo altrettanti moduli.

Se guardiamo all'interno di un moderno processore, e cerchiamo di individuare il modulo «somma tra due celle» ben difficilmente ci riusciremo perché... non c'è. Analogamente per gli altri moduli. Troveremmo invece delle unità abbastanza diverse, collegate opportunamente tra loro, che sotto la direzione di una parte di processore (la parte controllo) interagiscono per l'esecuzione di tutte le operazioni che la CPU è in grado di eseguire.

Figura 1



Dal canto suo, la parte controllo di un processore, ha bisogno di conoscere una per una tutte le sequenze di operazioni da compiere per implementare le istruzioni di macchina della CPU in questione. Tali sequenze sono dette microprogrammi e ogni operazione di un microprogramma è detta microistruzione. Così facendo, la nostra CPU sarà composta essenzialmente da tre parti: la parte controllo, la parte operativa (che contiene le unità di cui sopra) e i microprogrammi che descrivono le istruzioni da implementare.

A questo punto qualcuno potrebbe obiettare che se il linguaggio macchina non è più di «macchina» essendo esso stesso interpretato dal microprogramma che è più di «macchina» del precedente, non si potrebbe programmare direttamente nel microlinguaggio in modo da non avere interpretazione?

Certo, solo che per microprogrammare una CPU bisogna essere a conoscenza non solo della struttura del computer, ma anche di quella interna al processore: occorre considerare la sincronizzazione tra processore e di-

Figura 2

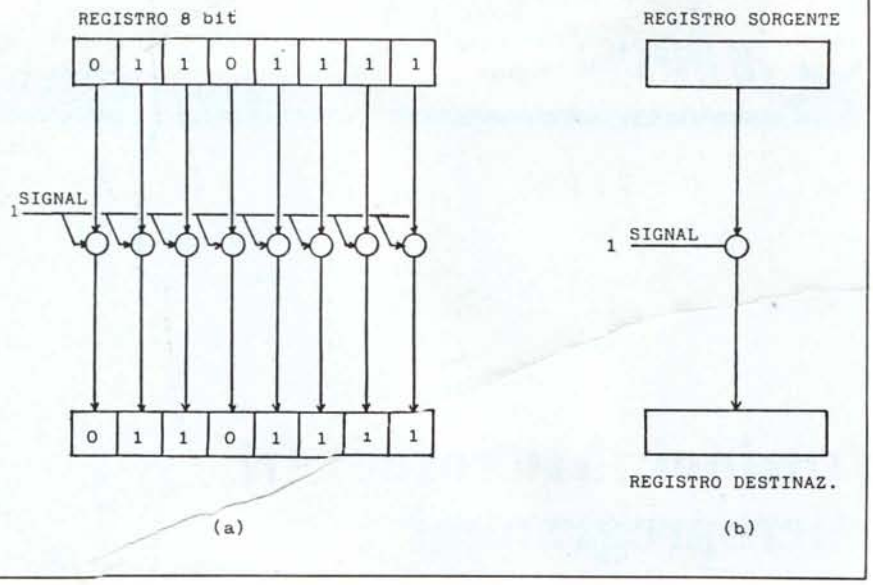
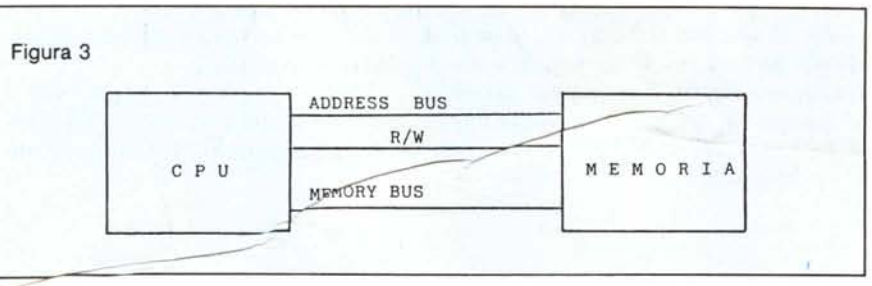


Figura 3



spositivi, il clock di sistema e un mucchio di altre cose che alla fine renderebbero la programmazione così ardua (si noti la stessa radice di hardware) che... l'unica cosa da fare sarebbe quella di reinventare il convenzionale linguaggio macchina (l'arte dei pazzi).

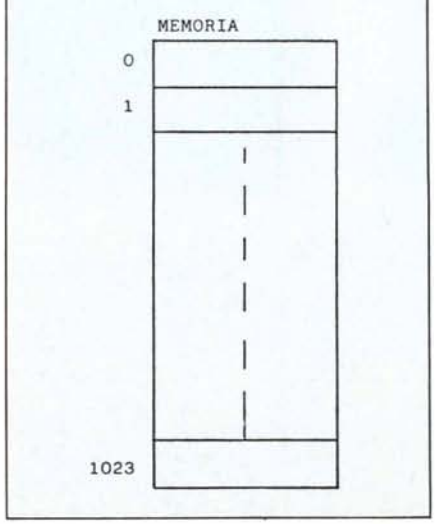
Componenti di un processore

Armati di potente lente di ingrandimento (si fa per dire) andiamo a vedere una moderna CPU come è fatta dentro, quali sono le unità di cui è composta.

Innanzitutto troveremo una manciata di registri: ne abbiamo già parlato il mese scorso, servono per mantenere alcune informazioni, allo stesso modo delle celle di memoria, col grosso vantaggio che essendo locati all'interno del processore, si fa molto presto ad accedervi e/o a modificarli.

Dei registri non accessibili dall'esterno e dei quali non abbiamo ancora parlato troviamo il registro IR nel quale semplicemente è trasferita l'istruzione da eseguire prima di essere eseguita: è come il leggio per lo spartito del direttore d'orchestra (la parte controllo). Esistono poi altri due registri per l'interfacciamento con la memoria, il MAR (Memory Address Register) sul quale si indica la cella di memoria desiderata e l'MBR (Memory Buffer Register) sul quale mandiamo o riceviamo contenuti di celle a/dalla memoria.

Figura 4



Si badi bene che questi due registri stanno nel processore e sono duplicati nel modulo di memoria: due normalissimi bus (bus dati e bus indirizzi) manterranno il collegamento tra le due coppie di registri. Analogamente, gli altri registri interni al processore saranno collegati tra loro da bus di collegamento e, come per le periferiche di ingresso/uscita già trattate alcuni numeri fa, potremo avere un collegamento privato per ogni coppia di registri, o un unico bus cui fanno capo tutti.

Vantaggi e svantaggi delle due possibilità dovrebbero essere ovvi: con unico collegamento si risparmia in soldi, ma si perde in velocità (una sola coppia di registri può usare il bus in uno stesso istante) mentre con i collegamenti dedicati si possono effettuare più trasferimenti in simultanea, a un costo di produzione naturalmente più alto. Come esperienza insegna, in medio stat virtus, e le soluzioni ottime si hanno disponendo collegamenti misti: a bus unico per quei registri che non toccheremo mai simultaneamente e a bus dedicati per gli altri.

Per controllare il traffico di questi trasferimenti useremo le cosiddette porte AND. In figura 1 è mostrata una porta AND: abbiamo un segnale binario di input, un segnale binario di controllo e un terminale di output. In due parole, se il segnale di controlo è a 1, ciò che sta sull'input passa e fuoriesce sul terminale di output, se il segnale di controllo è 0 sul terminale di output non transiterà alcunché, indipendentemente da ciò che abbiamo in input.

In figura 2a abbiamo una manciata di porte AND collegate in modo da controllare i trasferimenti su un bus: abbiamo due registri a 8 bit collegati tra loro tramite 8 fili elettrici, su ognuno dei quali è stata posta una porta. Tutte le 8 porte hanno il terminale di controllo collegate ad una unica linea: in questo modo se tale linea è a 1 il contenuto del registro di sopra è copiato in quello di sotto, se la linea è a zero, il registro destinazione non è modificato. Noi adotteremo la convenzione (peraltro universalmente riconosciuta) mostrata in figura 2b con la quale si schematizza il bus a n bit con un'unica linea sulla quale è posta una singola porta AND a rappresentare le n necessarie al trasferimento.

Un altro componente importante di ogni processore è il clock (o orologio)

Figura 5 - Formato Istruzioni.

che a intervalli di tempo uguali manda un segnale per sincronizzare le operazioni e in generale ogni microistruzione di un processore è eseguita in ognuno di questi cicli. Per semplicità ignoreremo in questa sede le problematiche riguardanti le sincronizzazioni da clock: prevalentemente riguardano le performance cui si mira nella progettazione di un processore.

La nostra ispezione all'interno del processore termina incontrando l'Unità Aritmetico Logica (ALU) che, lo dice il ragionamento stesso, serve per fare le operazioni aritmetiche e logiche. Tra le operazioni aritmetiche annoveriamo la somma e la sottrazione (difficilmente la moltiplicazione e la divisione, che di solito sono interamente microprogrammate) per le operazioni logiche l'and, l'or, il complemento e lo shift a destra o a sinistra di un numero binario.

Progettiamo una CPU

Si, avete letto bene. Anche se a grandi linee, ciò che faremo fino al termine di questo articolo sarà il progetto di un processore, date ovviamente le unità di cui abbiamo bisogno.

La prima cosa da fare, è definire il progetto: ovviamente ci manterremo sul semplice non potendo dedicare una trentina di pagine al problema. La nostra CPU sarà una macchina ad un indirizzo, avrà al suo interno un accumulatore, A, e un registro indice, X, entrambi a 16 bit. Sempre per semplicità immaginiamo di poter indirizzare solo un kappa di memoria, come mostrato in figura 4. In figura 3, è mostrato l'ormai noto collegamento tra CPU e memoria tramite tre bus: di indirizzamento (nel nostro caso è di 10 bit essendo 2 alla 10 uguale a 1024), il memory bus a 16 bit e la linea di Read/Write che consente di comunicare al modulo memoria il tipo di operazione che vogliamo compiere.

Le celle di memoria, come i registri del microprocessore A, X e IR, sono larghe 16 bit, e le istruzioni della nostra CPU potranno occuparne parte di una, una intera o due.

Abbiamo infatti 4 modi di indirizzamento: diretto, indicizzato, immediato e implicito. Il primo, consente di indirizzare una qualsiasi cella di memoria semplicemente specificandola accanto all'istruzione. Il secondo, tramite il registro X permette di accedere alla cella di memoria ottenuta sommando la locazione specificata al contenuto di X, tutto modulo 1024 se tale valore supera il limite massimo di celle disponibili. Il terzo modo di indirizzamento, lo ricordiamo, fornisce direttamente l'operando dell'istruzione occupando in tal caso la cella successiva a quella del codice operativo. Infine, il modo implicito, non specifica alcun operando essendo questo implicito nell'operazione che si vuole compiere.

In figura 5 è mostrato il formato delle istruzioni in base al modo di indirizzare l'operando: si noti che con l'indirizzamento implicito e con l'indirizzamento immediato si ha uno spreco di memoria dato che il codice operativo, come vedremo, occupa sempre e solo 6 bit mentre le celle di memoria come è ovvio sono tutte di 16 bit.

Il processore che stiamo definendo, ha in tutto 16 istruzioni. Ciò vuol dire che per codificarle ci occorrono 4 bit (due alla 4 uguale 16); a questi aggiungeremo due bit per codificare il modo di indirizzamento (vedi sempre figura 5).

Per fare un esempio, l'istruzione LDA che carica nell'accumulatore quanto specificato dall'operando, ha come codice operativo 000000 (6 zeri) se il modo di indirizzamento è diretto, 000001 se il modo di indirizzamento è indiciato, 000010 se è immediato. In figura 6 è mostrato tutto l'insieme di istruzioni con i modi possibili per ognuna di esse e relativo codice operativo (comprensivo di indirizzamento).

Due parole sul modo di indirizzamento implicito: si ha quando non si specifica alcun operando e varia da caso a caso. Ad esempio LDA senza operando sta per «carica in A il conte-

INDIRIZZAMENTO	DIRETTO	INDICIATO	IMMEDIATO	IMPLICITO
LDA	000000	000001	000010	000011
STA	000100	000101	----	----
LDX	001000	----	001010	001011
STX	001100	----	----	----
ASL	----	----	----	010011
LRS	----	----	----	010111
ADD	011000	011001	011010	----
SUB	011100	011101	011110	----
BPL	100000	----	----	----
BEQ	100100	----	----	----
BNE	101000	----	----	----
BMI	101100	----	----	----
BVS	110000	----	----	----
JMP	110100	----	----	----
JSR	111000	----	----	----
RTS	----	----	----	111111

Figura 6 - Codici delle istruzioni.

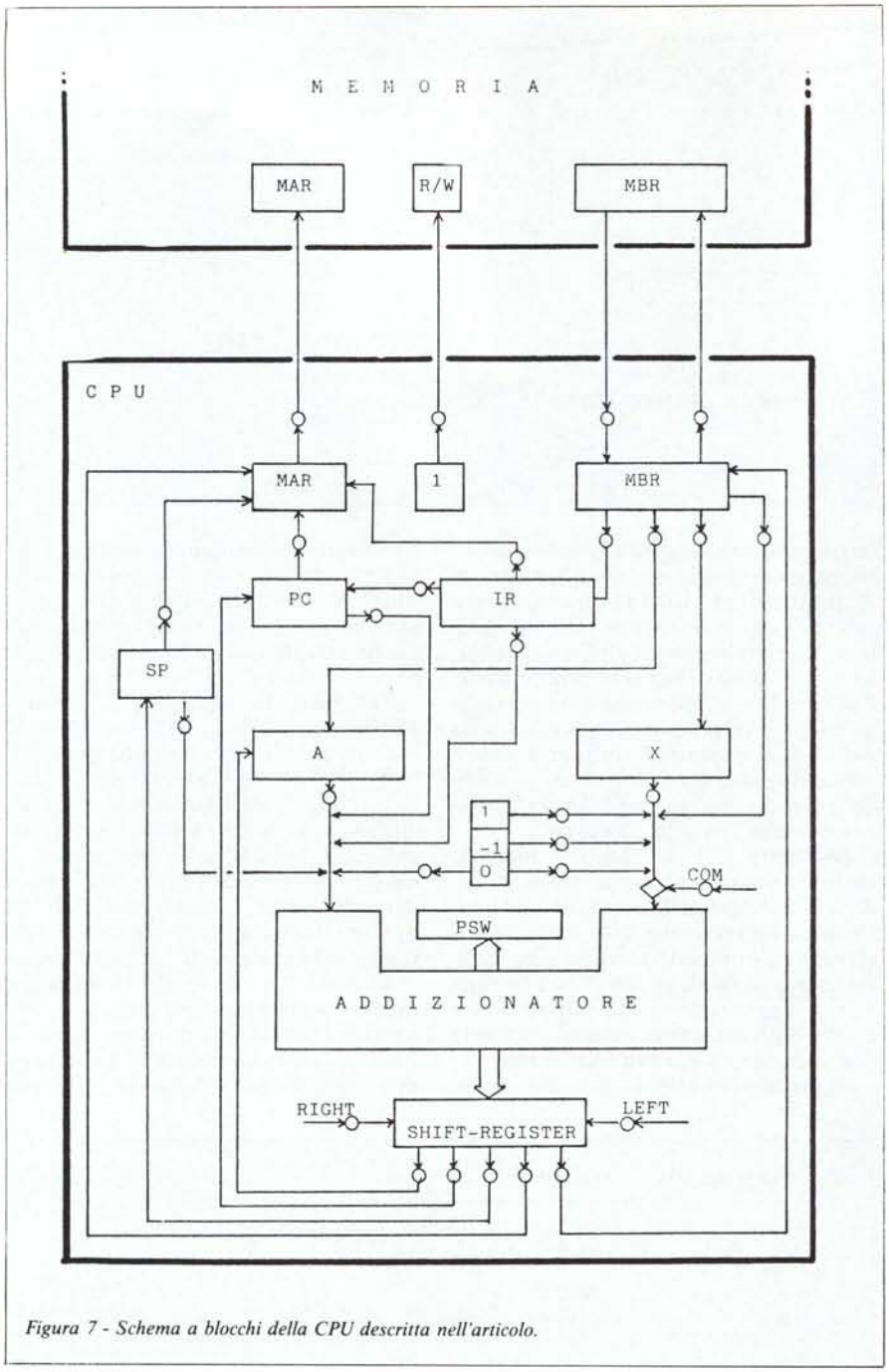


Figura 7 - Schema a blocchi della CPU descritta nell'articolo.

nuto di X» analogamente per LDX, che in caso di indirizzamento implicito fa riferimento al contenuto di A. LSR e ASL che possono essere usati solo in modo implicito, eseguono l'uno lo shift a destra e l'altro lo shift a sinistra di un bit del contenuto dell'accumulatore. Infine RTS, che permette di ritornare da un sottoprogramma (attivato col classico JSR) ovviamente non ha operandi dato che il punto di ritorno è stato salvato nello STACK al momento della chiamata a sottoprogramma.

Le altre istruzioni permettono di saltare se l'operazione precedente ha dato esito positivo, zero, negativo, diverso da zero, se c'è stato un overflow in una somma o sottrazione binaria o incondizionatamente con l'istruzione JMP.

Tutti i salti condizionali avvengono interrogando la Processus Status Word che è aggiornata automaticamente ogniqualvolta si fa uso dell'unità aritmetico-logica (nel nostro caso essenzialmente un semplice addizionatore) contenuta nella CPU.

E passiamo alla figura 7: mostra lo schema a blocchi della CPU che ci stiamo inventando. Com'era da aspettarsi, è interamente costituita da quelle unità di cui parlavamo a inizio articolo. Abbiamo un po' di registri, una ALU e l'interfaccia con la memoria tramite i registri MAR, MBR e la linea Read/Write dei quali abbiamo già parlato. L'unità aritmetico-logica è formata da tre oggetti: un addizionatore, un complemento e uno shift register. Il primo non fa altro che fornire in uscita la somma aritmetica dei due dati presenti all'ingresso; il secondo cambia di segno (da positivo a negativo o viceversa) il dato in ingresso a destra (è rappresentato da quel piccolo rombo); il terzo permette di effettuare lo shift a destra o a sinistra di un bit del dato in uscita dall'addizionatore.

Infine abbiamo delle costanti, 1, −1 e 0 che ci servono, come vedremo, in alcuni casi.

Tutte le unità appena descritte, come mostrato sempre in figura 7, sono collegate tramite bus singoli controllati da porte AND, i pallini lungo le linee di congiunzione: per non complicare troppo il disegno non sono stati riportati i terminali di controllo delle porte, facciamo conto che tali collegamenti passino da sotto, in qualche modo.

Per trasferire, ad esempio, il contenuto del Programm Counter nel registro MAR impulseremo sulla porta posta tra i due; per trasferire il registro X nel registro A, non essendo previsto un collegamento diretto, dovremo passare per l'addizionatore sommando a X la costante 0 (e ciò avviene aprendo la porta tra X e l'ingresso destro, facendo «scivolare» lo 0 sull'ingresso sinistro, non azionando ovviamente né il complementatore né lo shift) e mettendo il risultato così ottenuto in A (un apposito pallino controlla il trasferimento tra lo shift-register e A).

In definitiva la figura 7 mostra la parte operativa del processore. La parte controllo si occuperà di aprire e chiudere i pallini a seconda dell'istruzione da implementare e, come già anticipato, il comportamento di questa unità è descritto dal microprogramma del processore, che nel nostro caso è mostrato in figura 8.

Anche questo, come qualsiasi altro programma scritto in un linguaggio di livello più alto, usa codici mnemonici per descrivere le operazioni da compiere. Non vediamo infatti apri e chiudi porte, ma operazioni più programmerecce per aumentare la comprensi-

```
FETCH-EXECUTE: MAR:=PC; MBR:=MEMORY(MAR);
               PC:=PC+1; IR:=MBR;
               IF BIT(15,IR)=1 THEN GOTO OP8-15
               IF BIT(11-10,IR)=00 THEN MAR:=BIT(9-0,IR)
               IF BIT(11-10,IR)=01 THEN MAR:=BIT(9-0,IR+X)
               IF BIT(11-10,IR)=10 THEN MAR:=PC; PC:=PC+1
               IF BIT(14,IR)=1 THEN GOTO OP4-7
               IF BIT(13,IR)=1 THEN GOTO OP2-3
               IF BIT(12,IR)=1 THEN GOTO STA
        LDA:   IF BIT(11-10,IR)="11" THEN A:=X+0; GOTO FETCH-EXECUTE
                  ELSE  MBR:=MEMORY(MAR); A:=MBR; GOTO FETCH-EXECUTE
        STA:   MBR:=A+0; MEMORY(MAR):=MBR; GOTO FETCH-EXECUTE
      OP2-3:   IF BIT(12,IR)=1 THEN GOTO STX
        LDX:   IF BIT(11-10,IR)="11" THEN X:=A+0; GOTO FETCH-EXECUTE
                  ELSE  MBR:=MEMORY(MAR); X:=MBR; GOTO FETCH-EXECUTE
        STX:   MBR:=X+0; MEMORY(MAR):=MBR; GOTO FETCH-EXECUTE
      OP4-7:   IF BIT(13,IR)=1 THEN GOTO OP6-7
               IF BIT(12,IR)=1 THEN GOTO LSR
        ASL:   A:=LEFT(A+0); GOTO FETCH-EXECUTE
        LSR:   A:=RIGHT(A+0); GOTO FETCH-EXECUTE
      OP6-7:   IF BIT(12,IR)=1 THEN GOTO SUB
        ADD:   MBR:=MEMORY(MAR); A:=A+MBR; GOTO FETCH-EXECUTE
        SUB:   MBR:=MEMORY(MAR); A:=A+COM(MBR); GOTO FETCH-EXECUTE
     OP8-15:   IF BIT(14,IR)=1 THEN GOTO OP12-15
               IF BIT(13,IR)=1 THEN GOTO OP10-11
               IF BIT(12,IR)=1 THEN GOTO BEQ
        BPL:   IF BIT(0,PSW)=0 THEN GOTO JMP
                               ELSE GOTO FETCH-EXECUTE
        BEQ:   IF BIT(1,PSW)=1 THEN GOTO JMP
                               ELSE GOTO FETCH-EXECUTE
    OP10-11:   IF BIT(12,IR)=1 THEN GOTO BMI
        BNE:   IF BIT(1,PSW)=0 THEN GOTO JMP
                               ELSE GOTO FETCH-EXECUTE
        BMI:   IF BIT(0,PSW)=0 THEN GOTO JMP
                               ELSE GOTO FETCH-EXECUTE
    OP12-15:   IF BIT(13,IR)=1 THEN GOTO OP14-15
                     IF BIT(12,IR)=1 THEN GOTO JMP
        BVS:   IF BIT(2,PSW)=0 THEN GOTO FETCH-EXECUTE
        JMP:   PC:=BIT(9-0,IR); GOTO FETCH-EXECUTE
    OP14-15:   IF BIT(12,IR)=1 THEN GOTO RTS
        JSR:   SP:=SP+1; MAR:=SP;
               MBR:=PC+0; MEMORY(MAR):=MBR;
               GOTO FETCH-EXECUTE
        RTS:   MAR:=SP; MBR:=MEMORY(MAR)
               SP:=SP+(-1); IR:=MBR
               GOTO FETCH-EXECUTE
```

Figura 8 - Microprogramma della CPU descritta nell'articolo.

bilità. Infatti, se la porta tra PC e MAR fosse la numero 3, scrivere Apri 3 sarebbe stato meno comprensibile di un più comune MAR:=PC (che è implementato nello stesso modo, Apri 3) e al programmatore, anzi al microprogrammatore, certamente sarebbe piaciuto di meno.

Tutto il microprogramma, per essere eseguito dalla parte controllo, sarà opportunamente compilato (da un microcompilatore?) nella sequenza di apri e chiudi corrispondente. Ad essi saranno aggiunti solo i dovuti test sui bit di alcuni registri che permettono così di variare il flusso dell'esecuzione a seconda dei codici operativi in arrivo dalla memoria.

E passiamo ora a commentare un po' del listato di figura 8.

Le prime istruzioni implementano il ciclo FETCH-EXECUTE del processore: serve per prelevare dalla memoria la prossima istruzione (di linguaggio macchina) e ad eseguirla. Il Programm Counter, che punta alla nuova istruzione, è copiato nel registro MAR, in MBR ci finisce il contenuto della corrispondente cella di memoria, il Programm Counter è incrementato di 1 e MBR è copiato nel registro IR. Tenendo sott'occhio la figura 7, queste prime quattro microistruzioni si implementano aprendo la porta tra PC e MAR, aprendo la porta tra MAR del processore e MAR della memoria (ciò dà anche ordine alla memoria di porre sul suo MBR il contenuto della cella cercata), aprendo la porta tra MBR della memoria e MBR della CPU. Ci resta da incrementare il PC e trasferire MBR in IR: quest'ultima operazione si effettua aprendo la porta corrispondente. Per quanto riguarda l'incremento, occorre una serie di passaggi: PC, tramite l'opportuna porta, è copiato nell'ingresso sinistro dell'addizionatore; simultaneamente la costante 1 è trasferita nell'ingresso destro, al ciclo di clock successivo, il risultato è disponibile nello shift register dal quale con un'altra «apertura» possiamo trasferire il risultato nel PC.

Finora abbiamo solo FETCH-ato l'istruzione, ora dobbiamo eseguirla. Ricordiamo che l'istruzione è posta nel registro IR identicamente a come stava in memoria: avremo i bit 0-9 occupati da un eventuale indirizzo e i bit 10-15 (si conta da destra a sinistra) occupati dal codice operativo (vedi anche figura 5).

Immaginiamo allora di avere in IR un'operazione di LDA indicizzato: come da figura 6 il codice corrispondente è 000001, seguito dall'indirizzo di memoria al quale sommeremo X per ottenere l'operando. Torniamo al microprogramma: la terza linea serve per stabilire se l'istruzione è delle prime 8 o delle seconde 8, nel caso nostro è delle prime, quindi l'IF dà risultato falso e si prosegue. I tre IF che seguono servono per indagare sul modo di indirizzamento dell'istruzione in IR: come già detto questa informazione è ricavabile controllando i bit 10 e 11, sempre di IR. A seconda del modo di indirizzamento (implicito escluso) si predispone il registro MAR in modo da farlo puntare correttamente all'operando in memoria. Nel nostro caso (vedi quinta linea) in MAR ci finiscono i bit 0-9 della somma di IR e di X. Tutti gli altri IF danno esito falso e quindi finiamo alla decima linea, etichettata per l'appunto con LDA. Ancora un IF non verificato, dato che il modo di indirizzamento non è implicito e finalmente in MBR è messo il contenuto della cella puntata da MAR, il tutto è trasferito in A e un salto a inizio microprogramma fa continuare con la prossima istruzione puntata dal Programm Counter.

Con la stessa logica funziona tutto il rimanente microprogramma, al quale vi rimandiamo per scoprire le implementazioni delle altre istruzioni di macchina, con la raccomandazione, se avete voglia di raccapezzarvici, di tenere sott'occhio sia figura 6 che figura 7. Le funzioni LEFT, RIGHT e COM, usate nel microprogramma, attivano le corrispondenti unità di shift a sinistra, shift a destra e cambiamento di segno, come già indicato in precedenza.

Al prossimo «Appunt-amento». MC