

2.0.7 Processore per impieghi generali

Il processore descritto in precedenza ha solo scopo esemplificativo in quanto esegue sempre la stessa operazione, peraltro alquanto banale, con un investimento in hardware del tutto ingiustificato.

Si può cercare di costruire una macchina digitale che, basata sull'idea di **scomposizione** descritta con la (2.0.2), sappia risolvere relazioni più complesse e modificabili sulla base di un **programma** che determina la **sequenza** con cui eseguire le operazioni elementari (operazioni eseguibili dall'ALU). In pratica è necessario:

- costruire dei circuiti (*hardware*) che sappiano eseguire un insieme di operazioni elementari, dette operazioni primitive (eseguibili col l' ALU);
- aggiungere una struttura di controllo che sappia comandare l'esecuzione di una successione qualsiasi delle suddette operazioni primitive, sulla base di un prefissato **programma**.

La soluzione a queste richieste può essere rappresentata dal sistema di **Fig.2.0.7.1**.

Si nota una sostanziale modifica del data path, anche se la struttura generale rimane la stessa. La funzione svolta dai precedenti quattro registri **D0, D1, D2, D3**, consistente nel contenere i dati e i risultati delle elaborazioni, è adesso delegata al solo registro **D0**. I nuovi registri, **Operando** e **PC**, hanno nomi legati alla particolare funzione cui sono destinati.

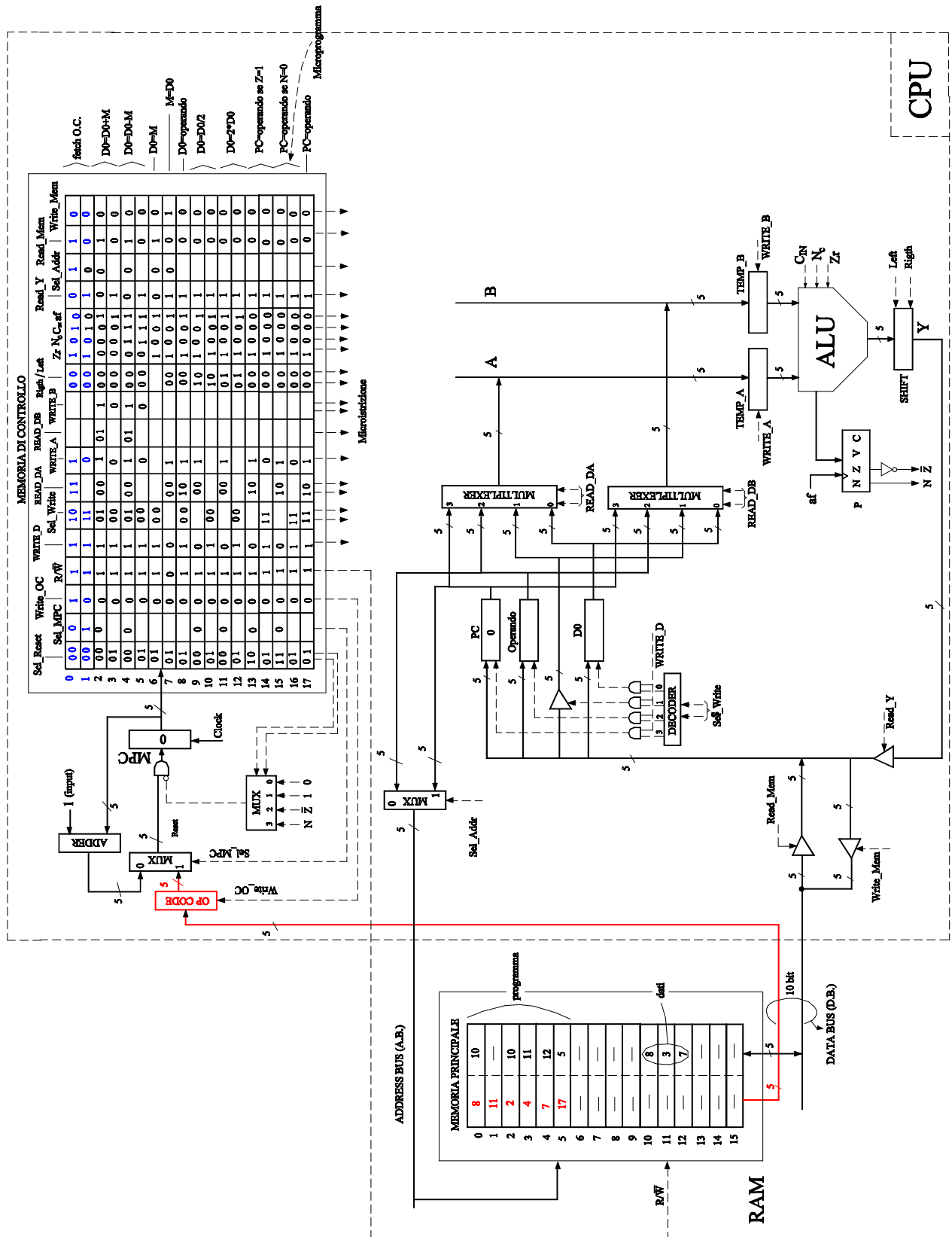
La riduzione dei registri contenenti dati è ampiamente compensata dall'utilizzazione di una **memoria a lettura e scrittura (RAM)**, esterna al processore (**Memoria principale**) e collegata a questo da due canali di comunicazione, **A.B.** e **D.B.**, e dal bit di controllo $R/\bar{W}$. La struttura interna della **RAM** è simile a quella della **ROM**, con la differenza che, ogni locazione, come uno dei quattro registri **D0, D1, D2, D3** di **Fig.2.0.7.1**, oltre ad essere letta, può essere scritta con un nuovo valore. La **Fig.2.0.7.2** mostra questo nuovo importante componente. Le locazioni di questa **RAM** contengono dati da 10 bit: i cinque più significativi sono collegati al registro **OP_CODE** ed i cinque meno significativi con il bus **Y** del data path. I valori indicati all'interno delle locazioni sono in codice decimale, per cui nella locazione 0 la coppia di valori indicata "**8 10**" corrisponde in binario ai dieci bit "**0100001010**".

Nel sistema esemplificato le locazioni dall'indirizzo 10 all'indirizzo 15 sono usate per svolgere la funzione di registri contenenti dati.

L'indirizzo inviato alla **RAM** può essere scelto tra due fonti di provenienza:

- il registro **PC (Program Counter)**
- il registro **Operando**.

Anche l'unità di controllo è ampiamente modificata, rispetto a quella di **Fig.2.0.7.1**. Il contatore di indirizzamento della **ROM** è stato sostituito da una logica circuitale che consente anche un indirizzamento non sequenziale della memoria di controllo, come sarà spiegato nel seguito. La stessa **ROM** di controllo di sole 6 locazioni, in **Fig.2.0.7.1**, è stata sostituita da una memoria di 18 locazioni, necessaria a contenere tutte le microistruzioni di questa nuova macchina.



Per comprendere il funzionamento dell'intero sistema si fa riferimento alla **Fig.2.0.7.1**, intendendo che i contenuti della memoria, dei registri interni alla **CPU** e della memoria principale, rappresentano lo stato del sistema subito dopo l'accensione.

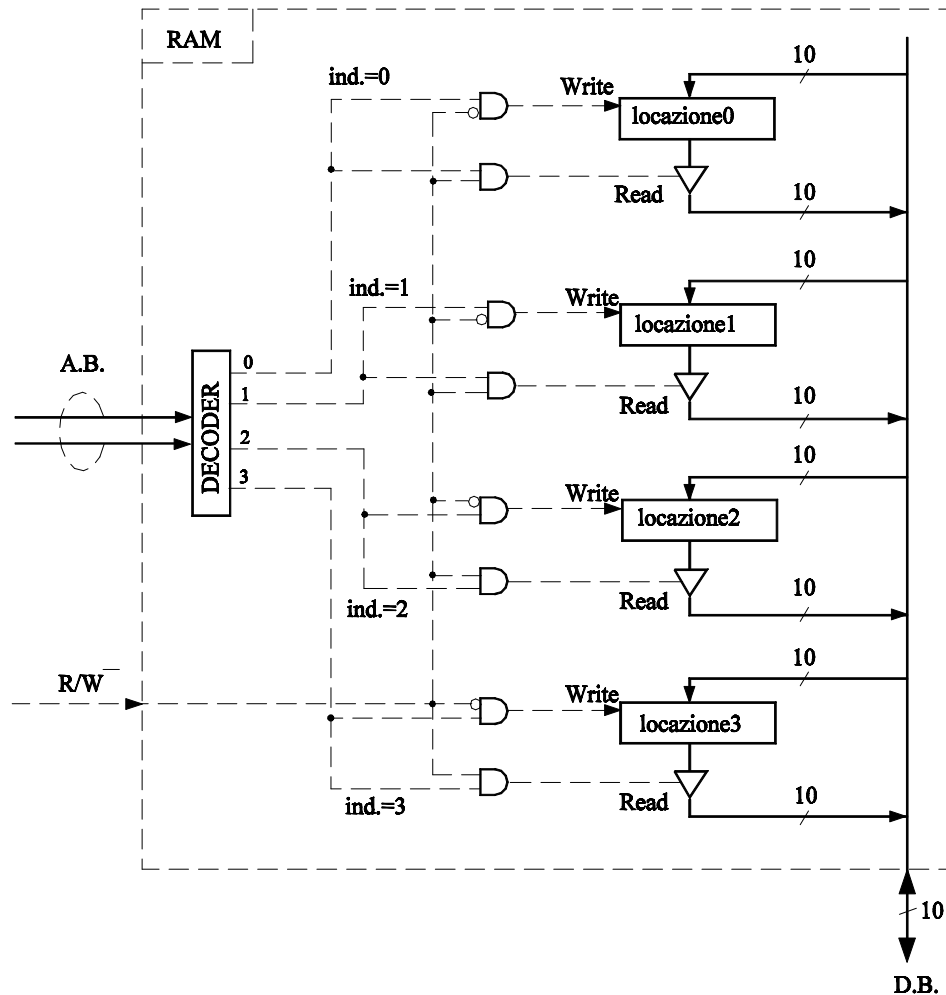


Fig.2.0.7.2 Schema di principio di una memoria a lettura e scrittura (**RAM**) con 4 locazioni.
Lettura $R/W = 1$. Scrittura $R/W = 0$.

Ricordando che le attività che si svolgono sul data path dipendono dai bit di controllo e quindi dalla microistruzione fornita dall'unità di controllo (unità di controllo: **ROM** di controllo più circuito generatore dei suoi indirizzi), l'evoluzione del sistema è:

accensione:

il registro **MPC** (*microprogramcounter*) indirizza la locazione 0 della **ROM**. Quattro bit della microistruzione agiscono sul generatore di indirizzi, uno, R/W , agisce direttamente sulla memoria, mentre i rimanenti 18 agiscono sul data path. Questi 18 bit, come si può facilmente controllare, producono l'invio del contenuto del registro **PC** (*Program Counter*) sul **Address Bus** (con $Sel_Addr=1$), consentendo di indirizzare la locazione 0 della **RAM** e di inviare lo stesso contenuto del **PC** all'**ALU** in modo che sul bus **Y** vi sia il valore $PC+1$ ($Read_DA=11$, $Write_A=1$, $Zr,Nc,Cin,af=1010$, $Rigth/Left=00$). Infine con la stessa microistruzione viene letta

(*R/W*=1) la locazione 0 della **RAM** inviando, tramite il **Data Bus**, i 5 bit più significativi (nel caso il valore decimale 8) al registro **OP_CODE** (*Write_OC*=1) e i 5 bit meno significativi (nel caso il valore decimale 10) al registro **Operando** (*Read_Mem*=1, *Sel_Write*=10, *Write_D*=1).

I quattro bit che agiscono sul generatore di indirizzi, o parte di loro, “decidono” il prossimo indirizzo da inviare alla **ROM**. *Sel_Reset*=00 e *Sel_MPC*=0 collegano l'uscita dell'**Adder**, tramite il multiplexer **MUX**, all'ingresso del registro **MPC**. Dato che l'**Adder** incrementa di 1 il contenuto attuale del registro **MPC**, all'impulso di **Clock**, il contenuto di **MPC** sarà incrementato di 1. In pratica l'anello formato da **MPC**, dall'**Adder**, da **MUX** selezionato sull'ingresso 0 e dalla porta **AND** abilitata, realizzano il funzionamento di un *contatore up*.

(NB. Per motivi di chiarezza nella lettura della **ROM**, non sono stati indicati i bit di controllo il cui valore non influisce sulle operazioni che debbono essere eseguite da ogni microistruzione.)

1°Clock:

il registro **MPC** (*microprogramcounter*) indirizza la locazione 1 della **ROM**. Nel data path la microistruzione di indirizzo 1 mantiene l'**ALU** nella condizione in cui era nella fase precedente (*Write_A*=0, *Zr,Nc,Cin,af*=1010, *Rigth/Left*=00), per cui, con *Read_Y*=1, *Sel_Write*=11 e *Write_D*=1, nel registro **PC** viene memorizzato il valore **PC**+1. Il bit di controllo *af* = 0 disabilita l'attivazione dei segnalatori non utile per l'incremento unitario del contatore di programma. Dato che il valore di **PC** prima dell'impulso di **Clock** era 0, adesso nel **PC** vi è il valore 1 (incremento di **PC**: **PC**=**PC**+1).

I tre bit di controllo interni: *Sel_Reset*=00 e *Sel_MPC*=1, collegano, tramite l'ingresso 1 del multiplexer **MUX**, l'ingresso del registro **MPC** con l'uscita del registro **OP_CODE**. In questo modo il **MPC** non funziona più come contatore up ma come registro nel quale memorizzare un nuovo valore. In particolare, al successivo impulso di **Clock**, dato che **OP_CODE** contiene il numero 8, raccolto dalla locazione 0 della **RAM**, nel **MPC** verrà memorizzato il valore 8. Questo valore viene detto *codice operativo*.

2°Clock:

Le precedenti due microistruzioni sono state eseguite in modo automatico all'accensione del sistema. La terza microistruzione, invece, *dipende dal valore raccolto dalla RAM*. Le prime due microistruzioni (indirizzo di **ROM** 0 e 1) realizzano la *fase di fetch del codice operativo* (raccolta). La terza microistruzione, indirizzata dal codice operativo, produce nel data path il trasferimento del dato contenuto nel registro **Operando** (il valore 10 raccolto dalla **RAM** assieme al *codice operativo* 8) nel registro **D0** (*Read_DA*=10, *Write_A*=1, *Zr,Nc,Cin,af*=1001, *Rigth/Left*=00, *Read_Y*=1, *Sel_Write*=00 e *Write_D*=1). Questa operazione attiva i segnalatori (*af*=1).

Le azioni svolte da questa microistruzione di indirizzo 8 costituiscono la *fase di execute* (esecuzione) del *codice operativo* 8.

L'attività svolta dal processore fino a questo punto può essere così riassunta:

- *raccolta dalla RAM della coppia di valori 8 10 mediante le prime due microistruzioni (fase di fetch)*
- *trasferimento del valore 10 nel registro D0 con la microistruzione di indirizzo 8 (fase di execute).*

Con la stessa microistruzione, i due bit interni **Sel_Reset**=01, inviano, tramite un multiplexer, il valore 1 alla porta AND. Dato che questo valore viene negato prima di agire sulla porta, il dato di uscita della porta AND è azzerato. Ciò garantisce che, al successivo impulso di **Clock**, nel registro **MPC** viene memorizzato ancora il valore 0. L'unità di controllo ritorna nella stessa condizione in cui si trovava subito dopo l'accensione del sistema, per cui viene ripetuta una nuova fase di fetch.

3°4°Clock: Questi due impulsi di clock ripetono le operazioni descritte ai **1°** e **2°** impulso di **Clock**. Si ha così una seconda fase di fetch che raccoglie dalla **RAM** il contenuto della locazione numero 1 (si ricorda che il **PC** nel **1°** impulso di **Clock** è stato incrementato a 1). I 5 bit più significativi raccolti rappresentano il nuovo codice operativo (11: undici) che sostituisce nel registro **OP_CODE** il valore 8. I cinque bit meno significativi, posti ancora nel registro **Operando**, non sono stati indicati in quanto non saranno utilizzati dal *codice operativo 11*. Nel data path il **PC** viene incrementato di 1 per cui si ottiene **PC**=2.

5°Clock: La sesta microistruzione, indirizzata dal *codice operativo 11*, produce nel data path il trasferimento del contenuto del registro **D0** all'**ALU** affinché sia moltiplicato per 2 (**Read_DA**=00, **Write_A**=1, **Zr,Nc,Cin,af** =1001, **Rigth/Left**=01). Inizia la fase di execute del codice operativo 11 che produce sul bus **Y** il valore $2 \cdot D0$. I bit di controllo interni **Sel_Reset**=00 e **Sel_MPC**=0 producono per il registro **MPC** il funzionamento come contatore up in modo che, al successivo impulso di clock si abbia **MPC**=12.

6°Clock: La settima microistruzione, di indirizzo 12, completa l'esecuzione del *codice operativo 11* producendo, nel data path, la memorizzazione in **D0** del valore presente sul bus **Y**. Si ottiene quindi **D0**= $2 \cdot D0$.

Ancora una volta i due bit interni **Sel_Reset**=01 predispongono, per il successivo impulso di clock, il reset del registro **MPC**, in modo da avviare una terza fase di fetch.

7°8°Clock: Si ha la terza fase di fetch che, usando il contenuto del **PC**=2 indirizza la *locazione 2* di **RAM** e porta in **OP_CODE** il valore 2, nuovo codice operativo, e in **Operando** il valore 10. Il **PC** viene incrementato a 3.

9°Clock: Inizia la fase di esecuzione del *codice operativo 2* che, con la microistruzione di indirizzo 2, porta all'ingresso **A** dell'**ALU**, programmata per fare la somma (**Zr,Nc,Cin,af** =0001, **Rigth/Left**=00), il contenuto di **D0** (**Read_DA**=00, **Write_A**=1). La stessa microistruzione porta all'ingresso **B** dell'**ALU** il valore letto dalla locazione di **RAM**, il cui indirizzo si trova nel registro **Operando**, nel caso il valore 12 contenuto nella locazione 10 (**Sel_Addr**=0, **Read_Mem**=1, **Sel_Write**=01, **Read_DB**=00, **Write_B**=1).

10°Clock: Si completa la fase di esecuzione del *codice operativo 2*, con la microistruzione di indirizzo 3, che memorizza in **D0** il valore di uscita dell'**ALU** (**Write_A**=0, **Write_B**=0, **Zr,Nc,Cin,af** =0001, **Rigth/Left**=00, **Read_Y**=1, **Sel_Write**=00, **Write_D**=1).

Si ottiene **D0**=**D0**+**M**[10], dove con **M**[10] si intende *valore del dato che si trova nella locazione di memoria di indirizzo 10* (in questo esempio il valore 12).

11°12°Clock: Si ha la quarta fase di fetch che, usando il contenuto del **PC**=3 indirizza la *locazione 3* di **RAM** e porta in **OP_CODE** il *codice operativo 4* e in **Operando** il valore 11. Il **PC** viene incrementato a 4.

13°14°Clock: Esecuzione del *codice operativo 4* (con operando pari a 11) che produce in due cicli l'operazione **D0**=**D0**-**M**[11]. Quest'operazione è prodotta dalle microistruzioni memorizzate agli indirizzi 4 e 5 della ROM. I bit di controllo differiscono da quelli memorizzati nelle locazioni 2 e 3 (che nei cicli **10°** e **11°** hanno prodotto la somma), solo per i bit **ZrNcCin**=011, dato che, in questo caso, l'operazione programmata nell'**ALU** è la differenza.

15°16°Clock: Si ha la quinta fase di fetch che, usando il contenuto del **PC=4** indirizza la *locazione* 4 di **RAM** e porta in **OP_CODE** il *codice operativo* 7 e in **Operando** il valore 12. Il **PC** viene incrementato a 5.

17°Clock: Esecuzione del *codice operativo* 7 (con operando pari a 12) che, con la microistruzione di indirizzo 7, produce, in un solo ciclo, la copia del dato contenuto in **D0** (**Read_DA=00**, **Write_A=1**, **Zr,Nc,Cin,af** =1001, **Rigth/Left=00**) nella locazione di memoria di indirizzo 12 (**Sel_Addr=0**, **Write_Mem=1**).

18°19°Clock: Si ha la sesta fase di fetch che, usando il contenuto del **PC=5** indirizza la *locazione* 5 di **RAM** e porta in **OP_CODE** il *codice operativo* 17 e in **Operando** il valore 5. Il **PC** viene incrementato a 6.

20°Clock: Esecuzione del *codice operativo* 17 (con operando pari a 5) che, con la microistruzione di indirizzo 17, produce, in un solo ciclo, la copia del dato contenuto del registro **Operando** (**Read_DA=10**, **Write_A=1**, **Zr,Nc,Cin,af** =1000, **Rigth/Left=00**) nel registro **PC** (**Read_Y=1**, **Sel_Write=11** e **Write_D=1**). L'operazione svolta è quindi **PC=operando**, essendo operando=5 si ha **PC=5**.
Ne deriva che il **PC** che, nell'ultima fase di fetch, prodotta dai cicli di clock **19°** e **20°**, era stato incrementato a 6, viene riportato a 5, per cui con la successiva fase di fetch (**22°** e **23°** impulso di clock) sarà raccolto ancora il *codice operativo* 17 con operando 5.

Ciò comporta la ripetizione indefinita dei cicli **19°**, **20°** e **21°** che eseguono le operazioni prodotte dalle microistruzioni di indirizzo 0, 1 e 17. In pratica la macchina raccoglie all'infinito il contenuto dalla locazione di **RAM** di indirizzo 5 ed esegue all'infinito il *codice operativo* 17. In pratica si ha una "fermata dinamica delle elaborazioni".

Per uscire da questa situazione di stallo è necessario prevedere un circuito (non mostrato in figura) che con azione esterna produca il reset dei registri **MPC** e **PC**, in modo da poter o riavviare lo stessa elaborazione con dati diversi, od una nuova elaborazione su un altro tipo di dati. In entrambi i casi sarà necessario modificare adeguatamente, in qualche modo, i contenuti della **RAM**.

Nella seguente tabella si riassumono i venti cicli di esecuzione del programma.

clock	MPC	μI	OP CODE	operando	PC	ALU bus Y	D0
(accen.)	0	0F7056	8	10	0	1	-
1	1	178058	8	10	1	1	-
2	8	265058	8	10	1	10	10
3	0	0F7056	11	-	1	2	10
4	1	178058	11	-	2	2	10
5	11	0410D8	11	-	2	20	10
6	12	2600D8	11	-	2	20	20
7	0	0F7056	2	10	2	3	20
8	1	178058	2	10	3	3	20
9	2	069612	2	10	3	28	20
10	3	260018	2	10	3	28	28
11	0	0F7056	4	11	3	4	28
12	1	178058	4	11	4	4	28
13	4	069672	4	11	4	25	2
14	5	260078	4	11	4	25	25
15	0	0F7056	7	12	4	5	25
16	1	178058	7	12	5	5	25
17	7	201059	7	12	5	29	25
18	0	0F7056	17	5	5	6	25
19	1	178058	17	5	6	6	25
20	17	27D048	17	5	5	5	25
21	0	0F70A6	17	5	5	6	25
22	1	1780A8	17	5	6	6	25
23	17	27D058	17	5	5	5	25

Si ripetono all'infinito i cicli 21, 22 e 23

2.0.8 Descrizione funzionale del processore

La macchina digitale descritta nel paragrafo precedente è formata da due parti fondamentali: la **CPU** (**C**entral **P**rocessing **U**nit), a sua volta costituita dal **data path**, dall'**unità di controllo**, e dalla **Memoria Principale (RAM)**.

La **RAM** contiene nelle locazioni 0, 1, 2, 3, 4, 5 le coppie di valori **codice operativo-operando**, chiamate **istruzioni in linguaggio macchina**.

La **CPU** usa il registro **PC** (**P**rogram **C**ounter), nelle fasi di fetch, per raccogliere ordinatamente le singole istruzioni dalla **RAM**, a iniziare da quella di indirizzo 0 (fasi di fetch) e trasferire il codice operativo nel registro **OP_CODE** e l'operando nel registro **Operando**. Dopo che il **PC** è stato usato per raccogliere una istruzione viene incrementato di uno, in modo da essere predisposto per la raccolta della prossima istruzione, all'indirizzo di memoria consecutivo.

Terminata ogni fase di fetch, la **CPU** esegue l'operazione "comandata" dal codice operativo appena raccolto (*fase di execute*). Questo codice è usato come indirizzo della memoria di controllo (**ROM**) in risposta al quale la **ROM** genera la prima microistruzione che inizia la fase di esecuzione. Per completare l'operazione comandata da un codice operativo alcune volte è sufficiente una microistruzione, altre volte ne servono due. L'insieme di microistruzioni che, per ogni codice operativo, esegue l'operazione comandata è detta *microprogramma*. Il legame tra codici operativi, operazioni comandate e microprogramma è mostrato nella Fig.2.0.8.1.

<i>Codice Operativo</i>	<i>Indirizzi ROM del microprogramma</i>	<i>Operazione Svolta</i>
2	2, 3	D0=D0+M[indirizzo]
4	4, 5	D0=D0-M[indirizzo]
6	6	D0= M[indirizzo]
7	7	M[indirizzo]=D0
8	8	D0=operando
9	9, 10	D0=D0/2
11	11, 12	D0=2*D0
13	13, 14	PC=operando se Z=1
15	15, 16	PC=operando se N=0
17	17	Pc=operando

Gli *operandi* sono le informazioni contenute in ogni istruzione necessarie a individuare i dati coinvolti nelle elaborazioni. E' detto *campo operando* di una istruzione l'insieme dei bit che contengono l'operando (i cinque bit meno significativi in questo esempio). Così l'istruzione di indirizzo tre: "4 11", ha nel suo campo operando il valore undici. Questo valore come quelli delle istruzioni "2 10" e "7 12", di valori rispettivamente dieci e dodici, rappresentano indirizzi di locazioni di memoria contenenti i dati. Agli indirizzi 10, 11 e 12 sono memorizzati rispettivamente i valori 8, 3 e 7. Dato che questi valori possono essere cambiati, si può dire che le tre istruzioni elaborano *dati variabili a indirizzi costanti*. Il dato è indicato nel campo operando con *modo* di *indirizzamento diretto* in memoria. L'istruzione "8 10", nella locazione 0, ha invece nel suo campo operando il valore che è usato come dato. Quando viene eseguita questa istruzione nel registro **D0** viene memorizzato il valore *costante* 10. Si parla in questo caso di *modo* di *indirizzamento immediato*, usato tutte le volte che è necessario inizializzare il valore contenuto in un registro con il dato voluto.

L'insieme delle istruzioni memorizzate nella **RAM** agli indirizzi che vanno da 0 a 5 costituiscono il *programma* che, eseguito dalla **CPU** con le modalità descritte, produce l'operazione:

$$M[12]=2 \cdot D0 + M[10] - M[11]$$

Si noti, per confronto con l'espressione (2.0.2), che le due locazioni di memoria di indirizzo 10 e 11 sostituiscono i registri **D1** e **D2** della macchina esaminata precedentemente. Il risultato dell'operazione, il valore 25, viene memorizzato nella locazione di **RAM** di indirizzo 12.

Si comprende come, con questa macchina, a differenza di quella precedente, sia possibile risolvere una qualunque espressione, anche molto complessa, per ogni programma memorizzato nella **RAM**, purché si disponga di una quantità di locazioni di memoria sufficienti a contenere tutte le istruzioni necessarie.

L'istruzione nella locazione 5 della **RAM**: "17 5", non produce alcuna elaborazione sui dati, ma obbliga la **CPU** ad eseguire sempre se stessa, dato che la sua fase di execute mette nel **PC** il valore 5 che è proprio il suo indirizzo di **RAM**. In senso figurato si dice che l'istruzione produce un *salto all'indirizzo contenuto nel suo campo operando*, impedendo l'avanzamento all'istruzione successiva, che, peraltro, in questo programma non esiste. Per questo viene indicata come **istruzione di salto**.

L'osservazione della **ROM**, in Fig.2.0.7.1, mostra l'esistenza di due codici operativi il 13 ed il 15, ognuno capace di comandare un'operazione di **salto condizionato** al valore, rispettivamente, del segnalatore **Z=1** e **N=0**.

Si supponga di avere in **RAM**, nella locazione di indirizzo 20 (con 5 bit di indirizzo la **RAM** può contenere 32 locazioni) l'istruzione "13 7". Quando questa istruzione è raccolta dalla **CPU** con l'uso di **PC=20** (13 in **OP_CODE** e 7 in **Operando**), la microistruzione di indirizzo 13, generata dalla **ROM**, produce, nel data path, il trasferimento del contenuto del registro **Operando** (il valore 7) nel bus **Y**. La stessa microistruzione seleziona, con i bit di controllo interni **Sel_Reset=10**, l'ingresso **Z** (proveniente, dopo negazione, dallo **Z-flag**) del multiplexer. Se il risultato prodotto dall'**ALU** è un numero uguale a zero (**Z-flag=1** ed il suo negato vale 0) viene abilitata la porta **AND**, in ingresso al **MPC**, che, al successivo impulso di clock, si incrementa di 1 (**Sel_MPC=0**), generando l'indirizzo di **ROM** 14. Conclusione dopo la microistruzione di indirizzo 13 di **ROM** viene generata quella di indirizzo 14 *solo se il risultato fornito dall'ALU è zero*. La microistruzione di indirizzo 14 copia nel **PC** (che contiene il valore 21) il dato presente sul bus **Y**, corrispondente all'operando dell'istruzione. Ne deriva che l'istruzione "13 7", memorizzata nella locazione 20 della **RAM**, produce un *salto* all'indirizzo 7, se l'**ALU** fornisce in uscita il valore zero (**Z-flag=1**). Si parla di **salto condizionato** al valore del segnalatore **Z**.

Se, invece, il dato fornito dall'**ALU** è diverso da zero: **Z-flag=0** ed il suo negato vale 1. L'uscita di MUX, nell'Unità di controllo, obbliga la porta **AND** a produrre il valore 0 in ingresso a **MPC**. Al successivo impulso di clock il contenuto di **MPC** viene azzerato. Dopo la microistruzione di indirizzo 13 si passa direttamente a quella di indirizzo 0 che avvia una nuova fase di fetch con il valore del **PC** inalterato a 21. Il salto non viene eseguito e dopo l'istruzione di indirizzo 20, che non ha prodotto alcuna elaborazione, viene eseguita quella di indirizzo 21.

L'istruzione di salto condizionato su **N=0** avviene con le stesse modalità. Si lascia l'analisi di questo caso come esercizio. Dato che **N=0** quando il risultato dell'**ALU** ha il bit più significativo pari a 0, si può affermare che l'istruzione che ha codice operativo pari a 15 produce un salto all'indirizzo indicato nel suo campo operando, *se l'ALU produce un risultato positivo (o zero)*, quando lo si leggono i dati nel codice in complemento a due.