CORSO DI
ARCHITETTURA DEGLI ELABORATORI

Corso di Laurea in Informatica - Unical

Progettazione di un processore general purpose

Un calcolatore didattico con architettura ad accumulatore

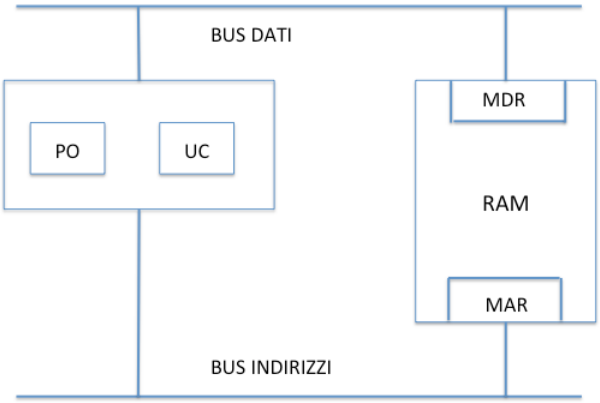
Pasquale Rullo

1 Premessa

Un processore **general purpose** è un sistema di elaborazione programmabile, cioè, in grado di eseguire programmi. Esso è pertanto una macchina universale, capace di risolvere tutti i problemi per i quali esiste un algoritmo (a patto che il linguaggio di programmazione sia sufficientemente potente). Un processore general purpose è, in ultima analisi, un interprete del linguaggio usato per descrivere gli algoritmi.
L'architettura di un calcolatore digitale si basa (1) su un processore general purpose (CPU - Central Processing Unit) e (2) su una memoria RAM (detta anche memoria centrale) in cui risiede il programma in esecuzione. Tale modello concettuale è noto come **macchina di Von Neumann** (o modello a **programma memorizzato**).

La CPU, come ogni sistema di elaborazione, ha due macro-componenti:
- la Parte Operativa (PO) che contiene l'ALU ed un certo numero di registri
- l'Unità di Controllo (UC) che sovrintende al funzionamento della Parte Operativa.

CPU e RAM formano l'**Unità Centrale** –uno schema è riportato nella seguente figura. In esso, la RAM è collegata alla CPU attraverso un bus dati ed un bus indirizzi. I registri MAR e MDR interfacciano la CPU con la RAM.



Quando si vuole eseguire un programma, lo si deve preventivamente caricare nella RAM. A questo punto, la sua esecuzione avviene secondo il seguente schema, detto"ciclo della CPU":

1. reperisci in memoria la prossima istruzione da eseguire trasferendola in qualche registro R della Parte Operativa
2. esegui l'istruzione corrente memorizzata nel registro R, e torna al passo 1.

Tale ciclo viene eseguito fino a quando tutte le istruzioni del programma non sono state eseguite. Ai fini della realizzazione della CPU è pertanto necessario progettare la "logica" che implementa il ciclo della CPU.

2 Architettura del Calcolatore Didattico

L'Unità Centrale del calcolatore didattico comprende (vedi figura seguente):

- una CPU la cui Parte Operativa si basa su una architettura con registro *accumulatore*
- una memoria RAM di 4K byte collegata alla CPU attraverso un bus dati ed un bus indirizzi.

La Parte Operativa include i seguenti componenti:

– Registro Acc (Accumulatore): contiene i risultati e gli operandi impliciti delle istruzioni (vedi descrizione linguaggio macchina)
– Registro IR (Instruction Register): contiene l'istruzione in fase di esecuzione
– Registro PC (Program Counter): contiene l'indirizzo di memoria della prossima istruzione da eseguire
– T0 e T1: registri tampone che memorizzano l'input all'ALU
– ALU (Unità Logico-Aritmetica): è una rete combinatoria che esegue le seguenti operazioni logiche e aritmetiche: ALU(T0+T1), ALU(T0-T1), ALU(T1) (funzione identità) e ALU(not T1) (complementazione).

La RAM è interfacciata con il processore tramite due registri: MAR e MDR. Il primo contiene l'indirizzo della cella di memoria da cui leggere o in cui scrivere. Il secondo contiene il dato da scrivere o il risultato della lettura.

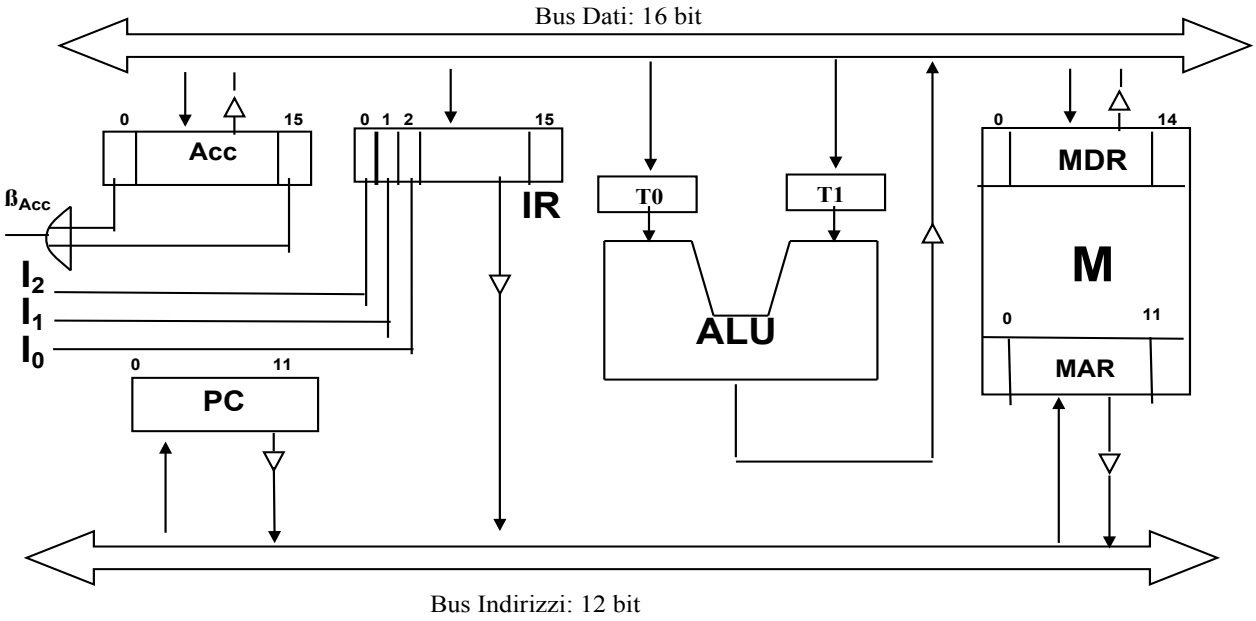


Figura 1 - Parte Operartiva e RAM

3 Linguaggio Macchina del Calcolatore Didattico

Il linguaggio macchina è l'insieme delle istruzioni eseguibili da un dato processore. Ogni programma scritto in un linguaggio di alto livello (C, C++, etc.) deve essere preventivamente tradotto in un programma equivalente scritto in linguaggio macchina per poter essere eseguito. I *traduttori* sono quei programmi che permettono di effettuare la traduzione dal programma sorgente (non eseguibile dalla CPU) al programma eseguibile scritto in linguaggio macchina. Ogni modello di CPU è caratterizzato da un suo proprio linguaggio macchina (cioè, il linguaggio che è in grado di eseguire).
Il Linguaggio Macchina del Calcolatore Didattico (LMCD) consiste delle istruzioni riportate nella seguente tabella.

Istruzione	Semantica	Commento
STORE x	Acc → M[x] – trasferisci il contenuto dell'accumulatore nella locazione di memoria con indirizzo x	Istruzione di trasferimento dati dalla CPU alla memoria centrale M
LOAD x	M[x] → Acc - trasferisci il contenuto della locazione di memoria con indirizzo x nell'accumulatore	Istruzione di trasferimento dati dalla memoria centrale M alla CPU
ADD x	Acc + M[x] → Acc – somma il contenuto dell'accumulatore con il contenuto della locazione di memoria con indirizzo x e salva il risultato nell'accumulatore	Istruzione airtmetica; uno dei due operandi è esplicito (locazione di memoria con indirizzo x), l'altro è implicito (accumulatore)
SUB x	Acc - M[x] → Acc	Come sopra
JZ x	If Acc == 0 salta all'istruzione che si trova all'ind. x	Istruzione di salto condizionato
JUMP x	salta all'istruzione che si trova all'ind. x	Istruzione di salto incondizionato
HALT	fine programma	Istruzione di chiusura del programma

Come si può notare, tutte le istruzioni di LMCD hanno un unico operando (operando esplicito) che è un indirizzo di memoria. Ciò perché, nel caso di operazioni binarie (STORE, LOAD, ADD, SUB, JZ), gli altri operandi sono impliciti. Ad esempio, l'istruzione ADD x ha il seguente significato: somma il contenuto M[x] della locazione di memoria (RAM) che ha indirizzo x (operando esplicito) al contenuto del registro Accumulatore (operando implicito), ed il risultato memorizzalo nel registro Accumulatore. In generale, il registro Accumulatore è la destinazione dei risultati delle istruzioni, nonché il secondo operando delle istruzioni con due operandi.

3.1 Esempi di Frammenti di Programmi in Linguaggio Macchina

Esempio 1. Si consideri l'istruzione di assegnazione X=Y, il cui significato è: trasferisci il contenuto della variabile (locazione di memoria) X nella variabile (locazione di memoria) Y. Essa viene tradotta in LMCD come segue:

- LOAD Y
- STORE X

Esempio 2. L'assegnazione y=(a+b)-(c+d) viene tradotta come segue:

1. LOAD c
2. ADD d /* c+d → acc
3. STORE y /* acc → y, cioé, c+d→ y
4. LOAD a
5. ADD b /* a+b → acc
6. SUB y /* acc – y → acc, cioé, (a+b) – (c+d) → acc
7. STORE y /* (a+b)-(c+d) → y

Esempio 3. Il frammento di programma

```
...
if a != 0  a = a+b;
else a = a-b;
.....
```

viene tradotto nel seguente codice in linguaggio macchina:

```
...
1. LOAD a  //a → acc
2. JZ  5
```

3. ADD b // a+b → acc
4. JUMP 6
5. SUB b // a-b → acc
6. STORE a

Esempio 4. Il frammento di programma

```
const c=1;
int i = 5;
int b=50;
while i != 0 {b=b-i;  i=i-c;}
```

viene tradotto come segue:

1. LOAD i
2. JZ 10
3. LOAD b
4. SUB i //b-i → acc
5. STORE b //acc → b
6. LOAD i
7. SUB c // i-1 → acc
8. STORE i //acc → a
9. JUMP 2
10. HALT

3.2 Codifica delle Istruzioni del Linguaggio Macchina

Un programma P in linguaggio macchina, per essere eseguito, deve essere caricato in memoria centrale (RAM), a partire da un dato indirizzo. Ogni istruzione viene memorizzata in un certo numero di byte, in funzione della sua struttura.
Nel caso del linguaggio LMCD, le istruzioni sono tutte caratterizzate da:

- un Codice Operativo che le identifica univocamente
- un indirizzo di memoria che individua la locazione dell'operando esplicito.

Codice Operativo - Poiché il numero di istruzioni di LMCD è pari a 7, sono sufficienti 3 bit per rappresentare il Codice Operativo; in particolare, possiamo utilizzare la seguente codifica:

Istruzione	*Codice Operativo*
LOAD	001
STORE	010
ADD	011
SUB	100
JZ	101
JUMP	110
HALT	111

Operando - Considerato che la memoria ha una capacità di 4 Kbyte (4096 byte), cioè 2^{12} byte, ogni indirizzo è lungo 12 bit. Pertanto, sono necessari 12 bit per memorizzare l'operando (esplicito) di ogni singola istruzione.

Ne consegue che ogni istruzione è lunga 15 bit (3 per il Codice Operativo e 12 per l'indirizzo di memoria). Sono pertanto necessari 2 byte (16 bit) per la sua memorizzazione (sono utilizzati i 15 bit più significativi). Utilizziamo il termine CELLA (o PAROLA) per indicare una sequenza di due byte.

Nel seguito assumeremo che anche i dati siano memorizzati su 2 byte (2^{16} – 1 è quindi il più grande numero rappresentabile, assumento che siano trattati solo numeri naturali).

4 Dimesionamento dei registri e dei bus

- registro MAR: Poiché dobbiamo indirizzare una memoria di 4K byte, cioè di 2^{12} byte, abbiamo bisogno di un registro MAR di 12 bit.
- registro IR: Poiché le istruzioni sono memorizzate in una cella, il registro IR è di 2 byte (viene trascurato solo il bit meno significativo). I tre bit del codice operativo vengono inviati all'Unità di Controllo
- registro PC: siccome contiene l'indirizzo di memoria in cui si trova la prossima istruzione che deve essere eseguita, il PC è un registro di 12 bit; il PC è un registro contatore ad incremento
- registro MDR: Poiché le istruzioni e i dati sono memorizzati in una cella di memoria, il registro MDR è anch'esso di 2 byte
- registro Acc: Deve essere in grado di contenere il valore di una cella di memoria, quindi 2 byte.

Da quanto detto consegue che il bus indirizzi dovrà essere a 12 bit ed il bus dati dovrà essere a 16 bit.

5 Segnali di Controllo della Parte Operativa

La Parte Operativa è controllata dai seguenti segnali di controllo:

- Aacc = segnale di abilitazione alla scrittura (caricamento) del registro Acc
- Sacc = segnale di abilitazione alla lettura del registro Acc
- A_{T0} = segnale di abilitazione al caricamento del registro T_0
- A_{T1} = segnale di abilitazione al caricamento del registro T_1
- A_{IR} = segnale di abilitazione alla scrittura (caricamento) del registro IR
- S_{IR} = segnale di abilitazione alla lettura del registro IR
- Z_{IR} = segnale di azzeramento del registro IR
- A_{MAR} = segnale di abilitazione alla scrittura (caricamento) del registro MAR
- S_{MAR} = segnale di abilitazione alla lettura del registro MAR
- A_{MDR} = segnale di abilitazione alla scrittura (caricamento) del registro MDR
- S_{MDR} = segnale di abilitazione alla lettura del registro MDR
- A_{PC} = segnale di abilitazione alla scrittura (caricamento) del registro PC
- S_{PC} = segnale di abilitazione alla lettura del registro PC
- K_{PC} = segnale per il controllo dell'incremento del PC (ricorda che PC è un registro contatore)
- A_{L0} e A_{L1} per l'ALU: se A_{L0}=0 ed A_{L1}=0 si ordina la somma, se A_{L0}=0 ed A_{L1}=1 la differenza, se A_{L0}=1 ed A_{L1}=0 in uscita avremo il secondo operando immutato, se A_{L0}=1 ed A_{L1}=1 in uscita avremo il complemento del secondo operando
- S_{alu} = segnale di abilitazione al caricamento sul bus dati dell'output dell'ALU
- R e W per la Memoria: se W=1 ed R=0 si ordina una scrittura, se W=0 ed R=1una lettura, se W=0 ed R=0 rimane invariata, se W=1 ed R=1 una reset

Il funzionamento del registro PC è definito come segue:

- se A_{PC} = 0 allora il contenuto del registro rimane invariato;

- A_{PC} = 1 allora
    - Se K_{PC} = 0 allora il PC carica dall'esterno (bus indirizzi)
    - Se K_{PC} = 1 allora il PC incrementa il suo contenuto (indirizzo della prossima istruzione)

Il funzionamento del registro IR è definito come segue:

- se A_{IR} = 0 allora il contenuto del registro rimane invariato;
- A_{IR} = 1 allora
    - Se Z_{IR} = 0 allora l'IR carica dall'esterno (bus dati)
    - Se Z_{IR} = 1 allora l'IR si azzera

6 Segnali di condizione della Parte Operartiva

Per verificare che il contenuto del registro Acc sia pari a zero (come richiesto dall'istruzione JZ x – vedi il paragrafo sul linguaggio macchina), è necessaria una porta logica che metta in OR tutti i bit del registro; se l'uscita di tale porta β_{acc} = 0 allora il valore contenuto in Acc è nullo, altrimenti almeno un bit di Acc è pari ad 1 e, quindi, il valore contenuto in Acc è diverso da zero.

Per il riconoscimento della istruzione corrente nel registro IR, i tre bit più a sinistra vengono inviati all'Unità di Controllo – essi rappresentano il codice operativo della istruzione.

7 Progettazione dell'Unità di Controllo

Come già accennato, quando si vuole eseguire un programma P, lo si deve preventivamente caricare nella memoria RAM, a partire da un certo indirizzo. Tale indirizzo viene copiato nel registro PC della CPU. A questo punto, l'esecuzione di P avviene secondo il seguente schema, detto "ciclo della CPU":

1. reperisci in memoria la prossima istruzione da eseguire (il cui indirizzo è nel PC) trasferendola nel registro IR della Parte Operativa, ed aggiorna il PC perché punti alla prossima istruzione – questa operazione è chiamata FETCH
2. esegui l'istruzione corrente memorizzata nel registro IR, e torna al passo 1.

Tale ciclo viene eseguito fino alla terminazione del programma (istruzione halt).

Ai fini della realizzazione della UC è pertanto necessario progettare la "logica" che implementa il ciclo della CPU. Ciò richiede la definizione dei microprogrammi che ne definiscono i vari passi, in particolare, il microprogramma del FETCH (passo 1) e i microprogrammi delle istruzioni del linguaggio macchina (passo 2).

7.1 Microprogrammi per le istruzioni del linguaggio macchina

Ogni istruzione del linguaggio macchina viene decomposta in una sequenza di microistruzioni, ognuna delle quali è atomica e direttamente eseguibile dalla parte operativa della CPU. La sequenza di microistruzioni associate ad una istruzione IS del linguaggio macchina costituisce il ***microprogramma*** MP di IS, che ne definisce l'implemetazione. L'esecuzione di IS (passo 3 del ciclo della CPU) richiede quindi l'esecuzione di MP.

NOTA: durante l'esecuzione di una istruzione del linguaggio macchina, tale istruzione è memorizza nel registro IR

STORE x

- m1: $IR_{3\text{-}14}$ → MAR; : Acc → MDR
- m2: WriteMem // MDR→M[MAR]

LOAD x

- m3: $IR_{3\text{-}14}$ → MAR
- m4: ReadMem (cioè M[MAR]→MDR)
- m5: MDR → Acc

ADD x

- m6: Acc → T0 ; $IR_{3\text{-}14}$ → MAR
- m7 : ReadMem
- m8 : MDR → T1
- m9: Alu(T0+T1) → Acc

SUB x

- m10: Acc → T0 ; $IR_{3\text{-}14}$ → MAR
- m11 : ReadMem
- m12 : MDR → T1
- m13: Alu(T0-T1) → Acc

JUMP x

- m14: $IR_{3\text{-}14}$ → PC

JZ x

- m15: If β_{Acc} == 0 $IR_{3\text{-}14}$ → PC

HALT

- m16: Ø // istruzione vuota

7.2 Microrpogramma del Fetch

Come si evince dal ciclo della CPU sopra schematizzato, l'istruzione FETCH va eseguita all'inizio del programma ed alla fine dell'esecuzione di ogni singola istruzione. Essa consiste in una semplice lettura in RAM della locazione di memoria il cui indirizzo è nel registro PC, e nel trasferimento del suo contenuto nel registro IR. Essa inoltre prevede l'incremento del PC. Il microprogramma dell'istruzione FETCH è quindi il seguente:

- m16: PC → MAR
- m17: ReadMem; PC = PC+2;
- m18: MDR → IR

Si noti che il PC viene incrementato di una quantità pari a 2 per "puntare" alla cella successiva che contiene la prossima istruzione del programma nella RAM. Tale incremento viene effettuato in parallelo alla ReadMem in quanto non vi è "interferenza" tra le due microistruzioni (non vi è ragione perché una preceda l'altra ed inoltre la loro esecuzione coinvolge componenti diversi dell'architettura).

NOTA: ReadMem e PC = PC+2 costituiscono un'unica microistruzione (la m17)

7.3 Esecuzione del Ciclo della CPU

L'esecuzione di un programma P richiede l'esecuzione dei microprogrammi delle istruzioni in P e del FETCH, secondo l'ordine definito dal ciclo della CPU:

CICLO DELLA CPU
Esegui il microprogramma del Fetch - sia ***IS*** *l'istruzione trasferita in IR*
While ***IS*** *<> halt*
1. *Esegui il microprogramma associato a IS*
2. *Esegui il microprogramma del Fetch; sia IS l'istruzione trasferta in IR*

End

Una volta che l'istruzione IS da eseguire è stata trasferita nel registro IR tramite il FETCH, il suo riconoscimento avviene attraverso il codice operativo. A tal fine, i tre bit I_0, I_1, I_2 più a sinistra di IR (che codificano il Codice Operativo di IS) vengono inviati all'UC (vedi Figura 1). Questa, sulla base dei valori di I_0, I_1, I_2, procede all'esecuzione del microporgramma MP di IS, inviando alla Parte Operativa (PO) una sequenza di segnali di controllo che codificano la sequenza delle miscroistruzioni di MP.

Così come per le istruzioni del linguaggio macchina, anche l'esecuzione del FETCH richiede l'esecuzione del relativo microprogramma. A tal fine, si può assimilare il FETCH alle altre istruzioni del linguaggio macchina, e gli si assegna un codice operativo, ad esempio 000.

Per "forzare" l'esecuzione del FETCH è quindi necessario, di volta in volta, azzerare i tre bit I_0, I_1, I_2 nel registro IR. Il valore 000 del codice operativo viene infatti interpretato dall'UC come una richiesta di esecuzione del relativo microporgramma (così come, ad esempio, il codice operativo 001 viene interpretato come una richiesta di esecuzione del microporgramma del LOAD). Più precisamemnte, l'azzeramento di IR deve avvenire

- quando il programma P viene caricato nella RAM (passo 1 del Cilco della CPU)
- alla fine della esecuzione di ogni singola istruzione di P (passo 2 del Cilco della CPU). A tale scopo, basta estendere il microprogramma di ogni istruzione del linguaggio macchina con un comando finale di azzeramento del registro IR.

Nella seguente tabella è riportata la versione finale dei microprogrammi.

cod Oper	istruzione	microprogramma
000	FETCH	• m1: PC → MAR • m2: ReadMem; PC = PC+2; • m3: MDR → IR
001	LOAD x	• m4: IR_{3-14} → MAR • m5: ReadMem • m6: MDR → Acc; Azzera(IR)
010	STORE x	• m7: IR_{3-14} → MAR; : Acc → MDR • m8: WriteMem; Azzera(IR)
011	ADD x	• m9: Acc → T0 ; IR_{3-14} → MAR • m5: ReadMem • m10: MDR → T1 • m11: Alu(T0+T1) → Acc; Azzera(IR)
100	SUB x	• m9: Acc → T0 ; IR_{3-14} → MAR • m5 : ReadMem • m10 : MDR → T1 • m12: Alu(T0-T1) → Acc; Azzera(IR)
101	JZ x	• m13: If β_{Acc} == 0 IR_{3-14} → PC; • m14: Azzera(IR)
110	JUMP x	• m15: IR_{3-14} → PC; • m14: Azzera(IR)
111	HALT	• m14: Ø

Alla luce di quanto appena detto, possiamo codificare il ciclo della CPU come segue:

CICLO DELLA CPU
esegui il microprogramma FETCH – sia cop il codice operativo della istruzione trasferita nel registro IR (i primi 3 bit del registro IR)
while cop ≠ 111 //111 è il cod operativo di HALT
switch (cop) {
- *case 000: Esegui il microprogramma FETCH*
- *case 001: Esegui il microprogramma LOAD*
- *....*
- *case 110: Esegui il microprogramma JUMP*

}
esegui microprogramma HALT
end.

7.4 Codifica delle Microistruzioni

Ogni microistruzione è codificata assegnando un valore ad ogni segnale di controllo della Parte Operativa. I valori assegnati sono tali per cui la PO viene "forzata" ad eseguire la data microistruzione. Nella seguente tabella è riportata la codifica delle miscroistruzioni del microprogramma della STORE (il simbolo "–" indica che il valore può essere indifferentemente 0 o 1):

Segnali di controllo	IR → MAR	Acc → MDR	WriteMem	Azzera(IR)
Aacc	0	0	-	0
Sacc	-	1	-	-
A_{IR}	0	0	0	1
S_{IR}	1	-	-	-
Z_{IR}	0	0	0	1
A_{MAR}	1	-	0	0
S_{MAR}	0	-	0	-
A_{MDR}	-	1	0	0
S_{MDR}	-	0	0	-
A_{PC}	0	0	0	0
S_{PC}	0	-	-	-
K_{PC}	-	-	-	-
A_{L0}	-	-	-	-
A_{L1}	-	-	-	-
Salu	-	0	-	-
R	0	0	0	-
W	0	0	1	-

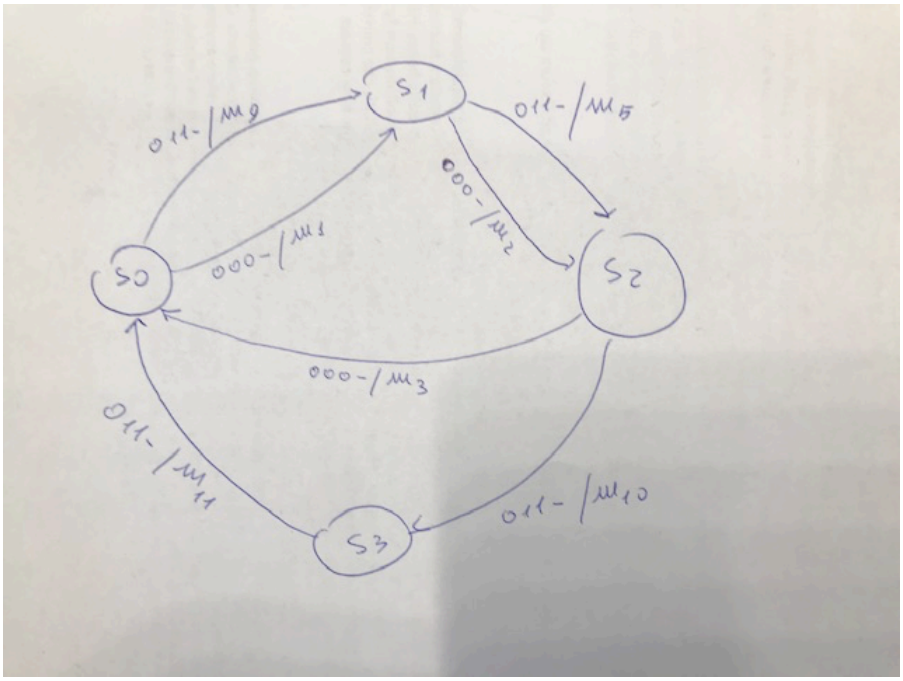
7.5 Unita' di Controllo come Automa a Stati Finiti

L'Unità di Controllo (UC) è un automa a stati finiti che genera, nella sequenza opportuna, i segnali di controllo che codificano i vari passi del Ciclo della CPU.

A tal fine, l'UC riceve in ingresso i segnali di condizione: 3 bit I_0, I_1, I_2 del codice operativo delle istruzioni del linguaggio macchina e del FETCH, nonché il segnale β_{acc}.

Il grafo delle transizioni che descrive il comportamento dell'UC del calcoltaore didattico si costruisce come segue:

- si parte dal microprogramma più lungo (quello associato alla istruzione ADD o SUB). S_0 è lo stato nel quale l'UC si trova all'inizio ed alla fine della esecuzione di ogni singola istruzione. Quando I_0=0, I_1 = 1 e I_2=1, ad esempio, l'UC "sa" di dover eseguire l'istruzione ADD. Pertanto, il sistema evolve nello stato S_1 (indipendentemente dal valore di β_{ACC}) inviando alla PO i segnali di controllo che codificano la microistruzione m6 (prima microistruzione del microprogramma di ADD); poi passa nello stato S_2, eseguendo la microistruzione m7, ecc. ecc. Una volta eseguita la microistruzione AzzeraIR, l'automa torna nello stato S_0. Questa parte del grafo delle transizioni consta di 4 stati (S_0-S_3)
- Utilizzando un sottoinsieme dei suddetti stati S_0-S_3, si rappresentano anche i microprogrammi delle altre istruzioni. Ad esempio, il grafo delle transizioni per ADD e FETCH è riportato nella seguente figura – si lascia per esercizio la rappresentazione delle altre istruzioni.



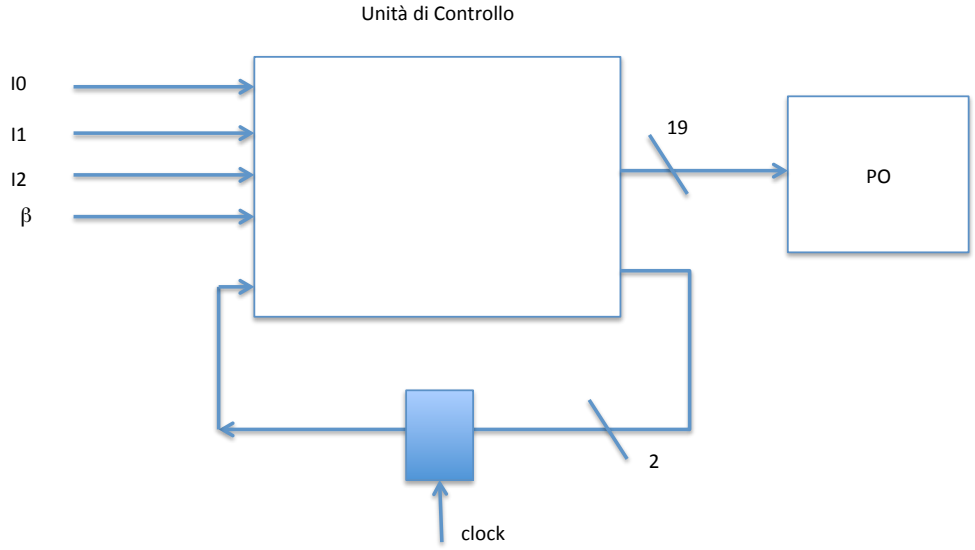
L'Unità di Controllo è quindi una rete sequenziale che

1. riceve in ingresso i segnali di condizione I0, I1, I2 del codice operativo dell'istruzione corrente memorizzata nel registro IR, ed il segnale βAcc
2. fornisce in uscita i segnali di controllo della parte operativa (19 bit di controllo: Aacc, Sacc , …., W, R) che codificano le miscroistruzione dei vari microporgrammi
3. ha due variabili di anello necessarie per codificare i 4 stati attraverso i quali evolve l'automa (il numero di variabili di anello è $\log_2 n$, dove n è il massimo numero di micro-istruzioni contenute in un micro-programma).

Quando un nuovo programma P viene caricato nella RAM per l'esecuzione, l'avvio del ciclo della CPU avviene

1. Ponendo l'indirizzo della RAM che contiene la prima istruzione di P nel registro PC
2. Azzerando il registro IR al fine di forzare il FETCH della prima istruzione
3. Mettendo la UC nello stato iniziale S_0.

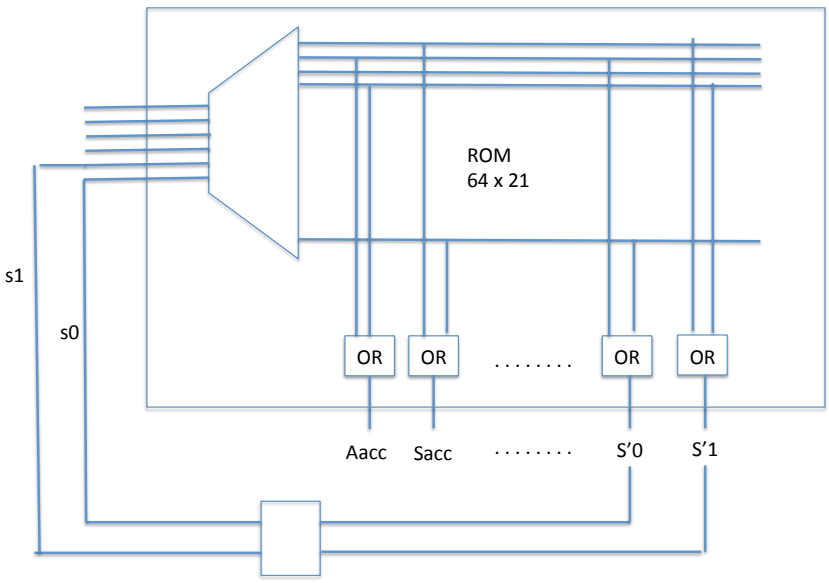
La parte combinatoria dell'UC è una rete con 6 variabili d'ingresso (I0, I1, I2 e β, più le 2 variabili di anello) e 22 uscite (i 19 segnali di controllo per la Parte Operativa più le 3 variabili di anello). Il comportamento ingresso-uscita di tale rete si evince facilmente dal grafo delle transizioni



Il segnale di clock sincronizza il funzionamento del sistema. Il passaggio da uno stato all'altro dell'UC avviene tra un impulso di clock ed il successivo. In tale intervallo di tempo, la PO esegue una mocroistruzione. Ne consegue che la velocità di esecuzione della CPU è pari ad una microistruzione per ogni ciclo di clock. Pertanto, un processore con un clock, ad esempio, di 1 GHz, esegue un miliardo di microistruzioni al secondo.

7.6 Unità di Controllo realizzata con un circuito ROM

La parte combinatoria dell'Unità di Controllo può essere realizzata con una ROM. Questa è un circuito combinatorio che comprende un Decoder a 6 ingressi (I0, I1, I2 e β, più le 2 variabili di anello, s0 e s1) e 64 segnali di uscita, collegati a 21 porte OR che generano i 21 segnali di uscita (19 segnali di controllo più e segnali che definiscono il nuovo stato). Lo schema della ROM è riportato nella seguente figura.



La ROM può essere vista come una memoria che memorizza la rappresentazione tabellare dell'automa descritto nel paragrafo precedente. In particolare, una memoria che contiene 2^6=64 locazioni (quante sono le uscite del decondificatore) in ognuna delle quali sono memorizzati i 19 bit di controllo che codificano le microistruzioni e i due bit s'0 e s'1 che codificano lo stato futuro. I 6 bit in ingresso (I0, I1, I2 e β, s0 e s1) possono essere interpretati come l'indirizzo di una particolare microistruzione memorizzata nella ROM. Ad esempio, (0,0,0,0,0,0) è l'indirizzo della microistruzione m1, (0,0,0,0,0,1) della microistruzione m2, etc. Quindi, il microporgramma del FETCH è, ad esempio, memorizzato agli indirizzi (0,0,0,0,0,0), (0,0,0,0,0,1), (0,0,0,0,1,0), etc.

Indirizzo ROM	Contenuto ROM	Commento
I0 I1 I2 β s_0 s_1	m s'_0 s'_1	
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0	m1 0 1 m2 1 0 m3 0 0	microprogramma FETCH
0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0	m4 0 1 m5 1 0 m6 0 0	microprogramma LOAD
0 1 0 0 0 0 0 1 0 0 0 1	m7 1 0 m8 0 0	microprogramma STORE
0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 0	m9 0 1 m5 1 0 m10 1 1 m11 0 0	microprogramma ADD
1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0	m9 0 1 m5 1 0 m10 1 1 m12 0 0	microprogramma SUB
1 0 1 0 0 0 1 0 1 0 0 1	m13 0 1 m14 0 0	microprogramma JZ
1 1 0 0 0 0 1 1 0 0 0 1	m15 01 m14 0 0	microprogramma JUM
1 1 1 0 0 0	m14 0 0	microprogramma HALT

Come si può vedere nella precedente tabella, all'interno di ogni microprogramma, il passaggio da una micrositruzione alla successiva è determinata dai valori delle variabili di stato s'0 e s'1 che, andando in ingresso al decodificatore (vedi schema architetturale sopra), generano l'indirizzo della prossima miscroistruzione (a parità di codice operativo della istruzione in esecuzione). Si noti anche che nell'ultima microistruzione di ogni microporgramma le variabili di stato valgono entrambe zero; in tal modo, il FETCH successivo partirà dalla prima microistruzione m1.
Si noti infine che il contenuto della ROM è largamente inutilizzato, in quanto non tutti i 64 indirizzi (configurazioni dei segnali di ingresso) vengono sfruttati (le microistruzioni sono infatti solo 21, ed ognuna di esse corrisponde ad un indirizzo).

8 Dispositivi di I/O

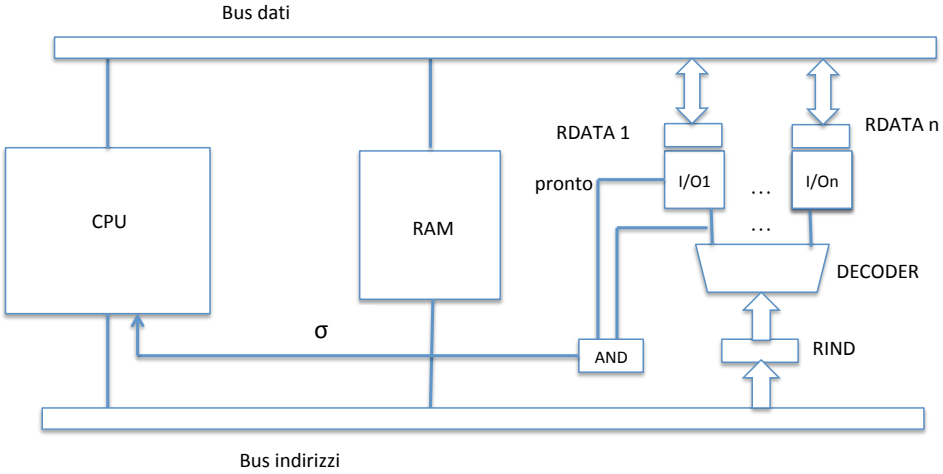
Un sistema reale è equipaggiato con un insieme di dispositivi di I/O attarverso i quali scambia dati con il mondo esterno. Una rappresentazione semplificata dell'architettura è riportata nella seguente figura. Ogni dispotivo ha un indirizzo che viene caricato dal processore nel registro RIND quando il dispositivo deve utilizzato. Tale indirizzo viene poi fornito in input ad un decodifivcatore che provvede ad attivare il dispositivo selezionato.

Per l'utilizzo di tali dispositivi, estendiamo il linguaggio macchina tramite due istruzioni aggiuntive:

- READ x: trasferisci nell'accumulatore il dato introdotto dall'esterno dal disposiitivo di ingresso x – dato contenuto del registro RDATA x
- WRITE x: scrivi il contenuto dell'accumulatore mediante l'unità di uscita x

L'interazione tra il processore ed i dispositivi di I/O è condizionata dal fatto che questi ultimi sono normalmente molto più lenti. Per gestire questa situazione, si usano due tecniche: I/O programmato e I/O ad interruzione.

Nel caso di I/O programmato, il processore deve attendere che il dato letto da un dispositivo di lettura D sia effettivamente disponibile. A tal fine, D mette in uscita un segnale *pronto* che indica quando il dato è disponibile (vedi figura). Analogamente, quando il processore deve scrivere un dato mediante un dispositivo di uscita D, attende il segnale *pronto* da D prima di inviare il dato (per evitare che ci siano sovrascritture).



Di seguito riportiamo i microprogrammi delle due istruzioni di I/O nel caso di I/O programmato.

READ x;

1. IRx → RIND;
2. if σ == 0 Ø goto 2
3. RDATAx → Acc

WRITE x

4. IRx → RIND;
5. if σ == 0 Ø goto 5
6. Acc → RDATAx
7. Stampa

Nei suddetti microporgrammi, σ rappresenta un nuovo segnale di condizione per l'Unità di Controllo. Esso è pari ad 1 quando il dispositivo selezionato è pronto ad interagire con il processore. In attesa che ciò avvenga, il processore si mette in attesa (vedi cicli alle linee 2 e 5).

Da quanto detto risulta evidente che l'I/O programmato introduce inefficienza, in quanto il processore è costretto ad attendere (senza far nulla) la disponibilità di un dispositivo molto più lento.

La tecnica dell'I/O ad interruzione, utilizzata in quasi tutti i sistemi reali, consente viceversa al processore di dedicarsi ad altri calcoli mentre il dispositivo di I/O opera.

9 Conclusioni

Un programma P in esecuzione è una sequenza di istruzioni del linguaggio macchina $<IS_1, IS_2, ..., IS_n>$ memorizzate nella RAM. L'esecuzione di P richiede l'esecuzione di ogni singola istruzione nella sequenza data. Ciò è a carico della CPU (processore), un circuito logico che funge da interprete del linguaggio macchina. A tal fine, il compito del processore è essenzialmente quello di eseguire il cosiddetto Ciclo della CPU:

1. Reperisci in memoria la prossima istruzione da eseguire e portala nel registro IR. A tal fine, viene lanciata l'esecuzione del microporgramma del FETCH
2. Esegui l'istruzione nell'IR lanciando l'esecuzione del rispettivo microprogramma; torna al passo 1.

L'esecuzione del FETCH (1) inizializza l'esecuzione del programma P, e (2) si inserisce tra la fine di ogni istruzione e la successiva. L'esecuzione di P può quindi essere vista come l'esecuzione di una sequenza di microporgrammi $<MP(F), MP(IS_1), MP(F), ..., MP(F), MP(IS_k), MP(F) ..>$, dove MP(F) è il microprogramma del FETCH e $MP(IS_k)$ è il microprogramma della istruzione IS_k (si noti che, a causa della presenza di cicli, la stessa istruzione può essere eseguita più volte).

Un processore genel purpose è pertanto un sistema di eleborazione in grado di eseguire un *unico* algoritmo, per l'appunto, il Ciclo della CPU. Si dà il caso, tuttavia, che tale algoritmo consenta di interpretare ogni programma scritto in linguaggio macchina. Pertanto, se il linguaggio macchina è sufficientemente ricco di istruzioni tale da poter tradurre qualsiasi programma scritto in un linguaggio di alto livello (ad esempio, C o C++), allora un processore genel purpose è un sistema in grado di eseguire qualsiasi programma (cioè, di risolvere qualsiasi problema per il quale esista una soluzione algoritmica).

Il Ciclo della CPU ed i microprogrammi sono "cablati" nella logica dell'automa a stati finiti che rappresenta l'UC. Questa è una rete sequenziale che interpreta ogni istruzione del linguaggio macchina tramite l'esecuzione del suo micro-programma. La sintesi dell'UC può essere realizzata con una ROM. In tal caso, la ROM può essere vista come una memoria nella quale sono memorizzati, in maniera permanente, i microporgrammi. Pertanto, l'esecuzione di P (che risiede nella RAM) può essere vista come una sequenza di chiamate a procedure cablate nella ROM dell'UC.