CORSO DI
ARCHITETTURA DEGLI ELABORATORI

Corso di Laurea in Informatica - Unical

Progettazione di un processore general purpose

Un calcolatore didattico con architettura ad accumulatore

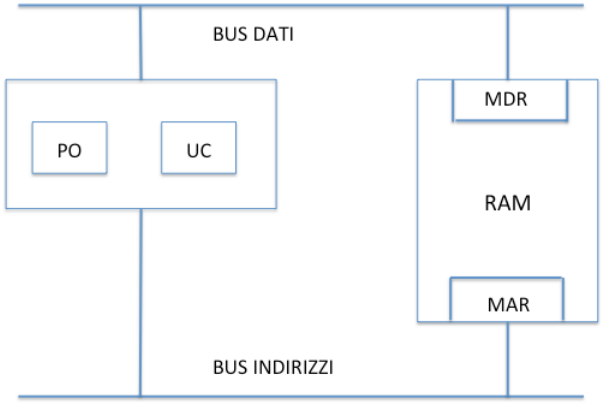
Pasquale Rullo

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 prevede (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.

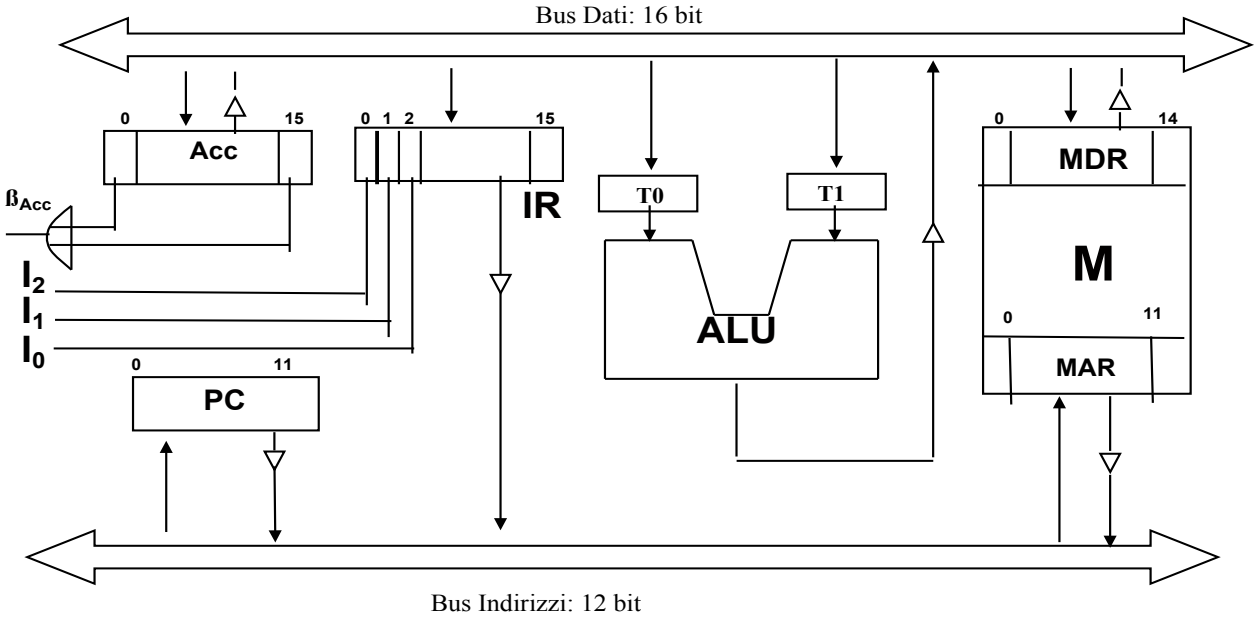
La Parte Operativa comprende dai seguenti componenti:

– 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).

Per verificare che il contenuto del registro Acc sia nullo (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.

I tre bit più a sinistra del registro IR vengono inviati all'Unità di controllo – come vedremo, essi rappresentano il cordice operativa della istruzione in IR.

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.



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

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.

4. 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 some 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 i=1;
Int a = 5;
int b=50;
While a != 0 {b=b-a;  a=a-i;}
```

viene tradotto come segue:

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

5. Codifica delle Istruzioni del Linguaggio Macchina

Un programma P in linguaggio macchina, per essere eseguito, viene 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 6, sono sufficienti 3 bit per rappresentare il Codice Operativo; in particolare, possiamo utilizzare la seguente codifica:

- LOAD: 001
- STORE: 010
- ADD: 011
- SUB: 100
- JZ: 101
- JUMP: 110

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).

6. 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 si ricava che il bus indirizzi dovrà essere a 12 bit ed il bus dati dovrà essere a 16 bit.

7. Progettazione dell'Unità di Controllo

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), come mostrato di seguito.

CICLO DELLA CPU
Esegui il microprogramma del Fetch - sia ***IS*** *l'istruzione reperita in RAM e trasferita in IR*
While ***IS*** *<> halt*
- *Esegui il microprogramma dell'istruzione IS*
- *Esegui il microprogramma del Fetch per trasferire la prossima istruzione in IR; sia* ***IS*** *tale istruzione*

End

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: $IR_{4\text{-}15}$ → MAR; : Acc → MDR
- m2: WriteMem (cioè MDR→M[MAR])

LOAD x
- m3: $IR_{4\text{-}15}$ → MAR
- m4: ReadMem (cioè M[MAR]→MDR)
- m5: MDR → Acc

ADD x
- m6: Acc → T0 ; $IR_{4\text{-}15}$ → MAR
- m7 : ReadMem
- m8 : MDR → T1
- m9: Alu(T0+T1) → Acc

SUB x
- m10: Acc → T0 ; $IR_{4\text{-}15}$ → MAR
- m11 : ReadMem
- m12 : MDR → T1
- m13: Alu(T0-T1) → Acc

JUMP x
- m14: $IR_{4\text{-}15}$ → PC

JZ x
If β_{Acc} == 1 then
- m15: $IR_{4\text{-}15}$ → PC

Microrpogramma del Fetch

L'istruzione FETCH va eseguita all'inizio del programma ed alla fine dell'esecuzione di ogni singola istruzione ed il suo trasferimento nell'IR. Essa consiste in una semplice lettura in RAM della locazione di memoria il cui indirizzo è nel registro PC. Essa inoltre include l'incremento del PC.

Il microprogramma dell'istruzione FETCH è 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)

Esecuzione del Ciclo della CPU

Come descritto in precedenza, i microprogrammi delle istruzioni e del FETCH si inseriscono nella logica del Ciclo della CPU, secondo il quale l'esecuzione di un programma P avviene secondo i seguenti passi:

1. esecuzione del microprogramma del FETCH per caricare la prima istruzione di P nel registro IR
2. esecuzione del microprogramma dell'istruzione nel registro IR
3. esecuzione del microprogramma del FETCH al termine dell'istruzione in IR per caricare in IR la prossima istruzione da eseguire.

I passi 2) e 3) vengono iterati fino all'istruzione finale ***halt***.

Per l'esecuzione del microprogramma di una generica istruzione IS caricata nel registro IR (passo 2) si procede come segue:

- 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
- l'UC, sulla base dei valori di I_0, I_1, I_2, avvia l'esecuzione del microporgramma M dell'istruzione IS - inviando alla Parte Operativa (PO) una sequenza di segnali di controllo che codificano la sequenza delle miscroistruzioni di M.

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 istruzioni del linguaggio macchina, e gli si assegna un codice operativo, ad esempio 000 (diverso da quelli delle istruzioni del linguaggio macchina). Per "forzare" l'esecuzione del FETCH è quindi sufficiente azzerare i tre bit I_0, I_1, I_2 nel registro IR - il codice operativo 000 sarà infatti interpretato dall'UC come una richiesta di esecuzione del microporgramma del FETCH (così come, ad esempio, il codice operativo 001 viene interpretato come una richiesta di esecuzione del microporgramma del LOAD). L'azzeramento dei bit I_0, I_1, I_2 può essere realizzato semplicemente definendo un opportuno segnale di azzeramento sul registro IR, che sarà attivato

- 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 3 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, come mostrato nei seguenti esempi.

STORE x
m1: $IR_{4\text{-}15}$ → MAR; m2: Acc → MDR
m2: WriteMem; Azzera IR
JZ x
If β == 0 goto m16
m15: $IR_{4\text{-}15}$ → PC
m16: Azzera IR

ADD x
m6: Acc → T0 ; IR4-15 → MAR
m7 : ReadMem
m8 : MDR → T1
m9: Alu(T0+T1) → Acc; Azzera IR;

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.

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
- AT0 = segnale di abilitazione al caricamento del registro T_0
- AT1 = segnale di abilitazione al caricamento del registro T_1
- AIR = segnale di abilitazione alla scrittura (caricamento) del registro IR
- SIR = segnale di abilitazione alla lettura del registro IR
- Z IR = segnale di azzeramento del registro IR
- AMAR = segnale di abilitazione alla scrittura (caricamento) del registro MAR
- SMAR = segnale di abilitazione alla lettura del registro MAR
- AMDR = segnale di abilitazione alla scrittura (caricamento) del registro MDR
- SMDR = segnale di abilitazione alla lettura del registro MDR
- APC = segnale di abilitazione alla scrittura (caricamento) del registro PC
- SPC = segnale di abilitazione alla lettura del registro PC
- KPC = segnale per il controllo dell'incremento del PC (ricorda che PC è un registro contatore)
- AL_0 e AL_1 per l'ALU: se AL_0=0 ed AL_1=0 si ordina la somma, se AL_0=0 ed AL_1=1 la differenza, se AL_0=1 ed AL_1=0 in uscita avremo il secondo operando immutato, se AL_0=1 ed AL_1=1 in uscita avremo il complemento del secondo operando
- Salu = 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 APC = 0 allora il contenuto del registro rimane invariato;
- APC = 1 allora
    - Se KPC = 0 allora il PC carica dall'esterno (bus indirizzi)
    - Se KPC = 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

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	m1: IR → MAR	m2: 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	-

NOTA: I segnali di controllo sono inviati alla Parte Operativa dalla Unità di Controllo

Unita' di Controllo come Automa a Stati Finiti

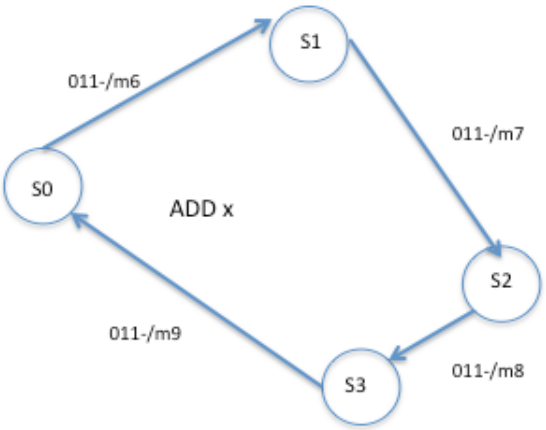
L'Unità di Controllo (UC) è un automa a stati finiti che genera, nella sequenza opportuna, i segnali di controllo che codificano i vari passi del Ciclo della CPU:

*Esegui il microprogramma del Fetch - sia **IS** l'istruzione reperita in RAM*
*While **IS** <> halt*
- *Esegui il microprogramma dell'istruzione IS*
- *Esegui il microprogramma del Fetch - sia **IS** l'istruzione reperita*

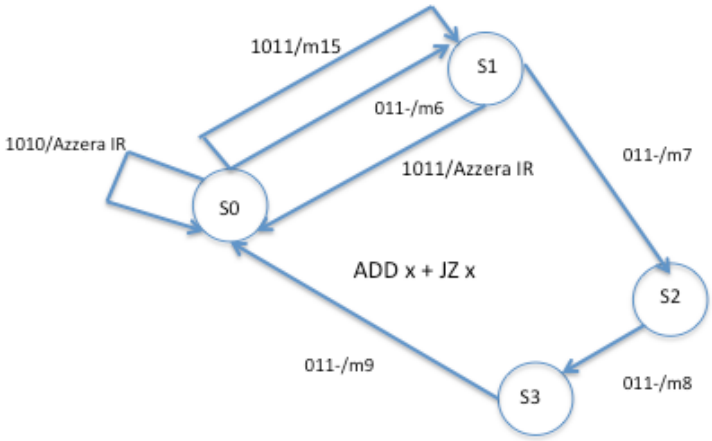
A la fine, l'UC riceve in ingresso i 3 bit del codice operativo delle istruzioni del linguaggio macchina e del FETCH, nonché il segnale β_{acc}.

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)



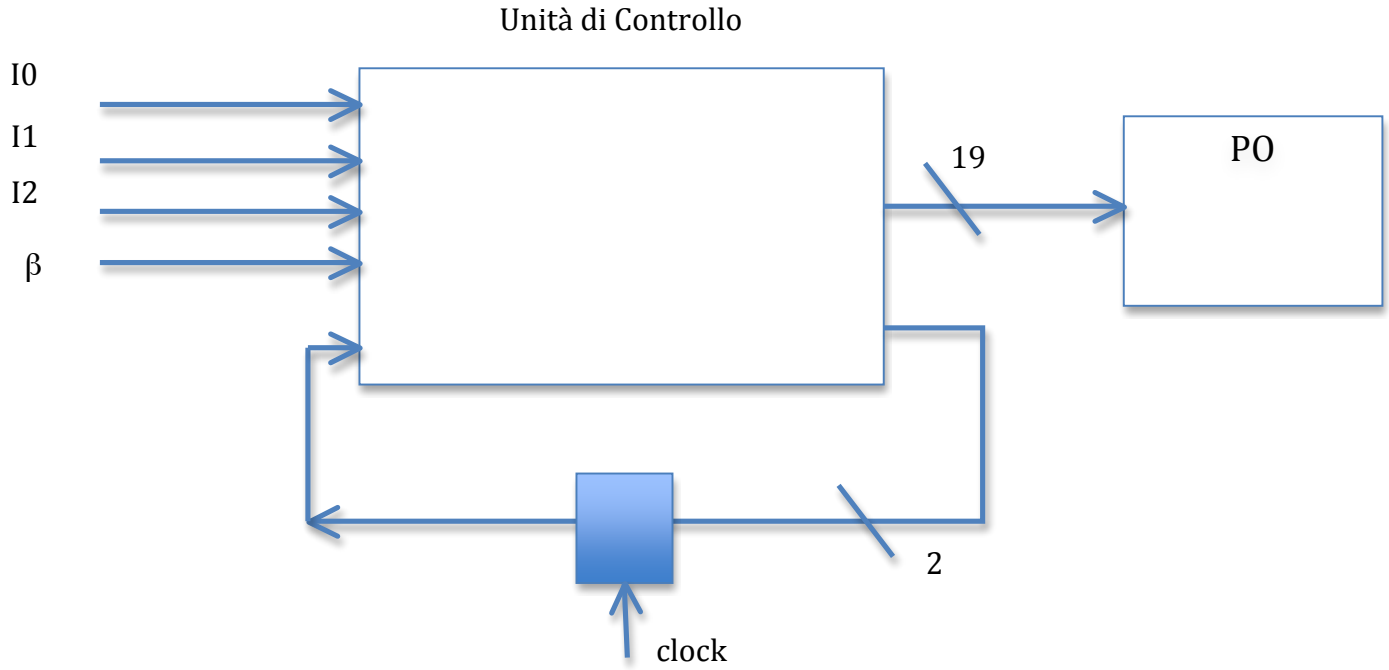
- Utilizzando un sottoinsieme dei suddetti stati S_0-S_3, si rappresentano anche i microprogrammi delle altre istruzioni, compresa la "istruzione" Fetch (il grafo delle transizioni per ADD e JZ è riportato nella seguente figura – si lascia per esercizio la rappresentazione del grafo per altre istruzioni).



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).

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.

8. Conclusioni

Un programma P in esecuzione consiste in 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, lancia 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.