Lezione 12: L'architettura LC-3

Laboratorio di Elementi di Architettura e Sistemi Operativi

24 Maggio 2012

Ricorda...

Il ciclo di esecuzione di un'istruzione è composto da sei fasi:

- FETCH
- DECODE
- ADDRESS EVALUATION
- OPERAND FETCH
- EXECUTE
- STORE RESULT

Istruction Set Architecture

Che cos'è l'ISA?
Il termine ISA (Instruction Set Architecture), specifica l'interfaccia tra i comandi software (forniti attraverso un linguaggio di programmazione) e quello che l'hardware è in grado di eseguire (linguaggio macchina).

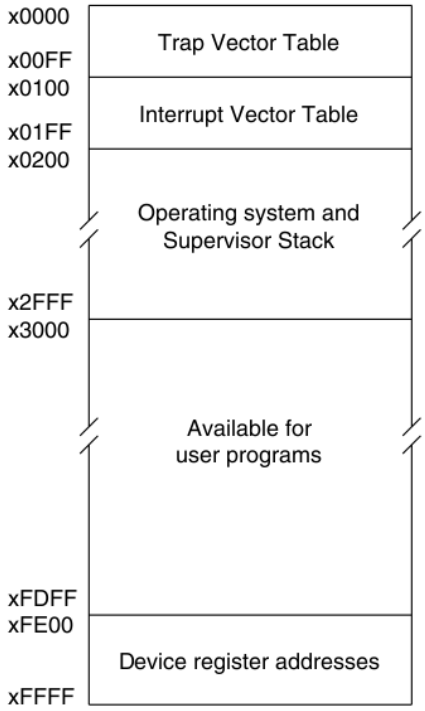
In particolare specifica:

- l'organizzazione della memoria
- l'insieme dei registri
- l'insieme delle istruzioni:
  - opcode
  - tipi di dato
  - metodi di indirizzamento

Vediamole nel dettaglio...

Organizzazione della memoria

- La memoria di LC-3 ha uno spazio di indirizzamento di 2^{16} (= 65536) locazioni
- Gli indirizzi sono a 16 bit.
- Gli indirizzi di memoria sono numerati da 0 (0x0000) a 65535 (0xFFFF)
- Non tutti identificano zone di memoria disponibile per l'utente, come riportato in figura.



Insieme dei registri

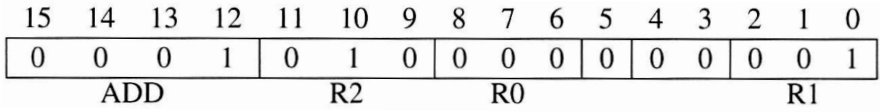
- LC-3 mette a disposizione 8 General Purpose Register (GPR);
- i registri sono a 16 bit (word);
- ognuno degli 8 registri (chiamati R0, R1, ..., R7) è identificato da un numero a 3 bit (da R0=000 a R7=111).

Register 0	(R0)	0000000000000001
Register 1	(R1)	0000000000000011
Register 2	(R2)	0000000000000101
Register 3	(R3)	0000000000000111
Register 4	(R4)	1111111111111110
Register 5	(R5)	1111111111111100
Register 6	(R6)	1111111111111010
Register 7	(R7)	1111111111111000

Insieme delle istruzioni

- Un'istruzione è identificata da DUE componenti:
  - **opcode:** specifica cosa l'istruzione chiede di fare alla macchina;
  - **operandi:** specificano i dati su cui la macchina andrà ad operare;
- Ogni istruzione è codificata su 16 bit.

- *Esempio:* vogliamo fare un'ADD fra i registri R0 e R1 e memorizzare il risultato in R2. L'istruzione sarà:



- L'ISA di LC-3 definisce 15 istruzioni, ciascuna identificata da un opcode univoco;
- L'**opcode** è specificato dai bit [12:15] di un'istruzione. Poichè vengono usati 4 bit, si possono identificare 16 istruzioni; LC-3 ne specifica 15, lasciandone libera una (codice 1101).
- Ci sono 3 diversi tipi di istruzioni:
  - *operazioni*: processano informazioni;
  - *spostamento di dati*: spostano informazioni tra registri e memoria e viceversa;
  - *controllo*: cambiano il flusso d'esecuzione delle istruzioni.
- Per i più temerari: in fondo a queste slide sono riportati i riferimenti all'elenco completo delle istruzioni.

Tipi di dato

- Per *tipo di dato* si intende un modo per rappresentazione l'informazione che sia gestibile dall'ISA
  - ovvero, per la quale l'ISA mette a disposizione *istruzioni in grado di manipolarla*
- Ci sono molti modi per rappresentare un'informazione...
  - caratteri
  - modulo e segno
  - complemento a uno
  - complemento a due
  - virgola mobile
  - ...
- L'ISA di LC-3 utilizza la forma **interi in complemento a due**.

Metodi di indirizzamento

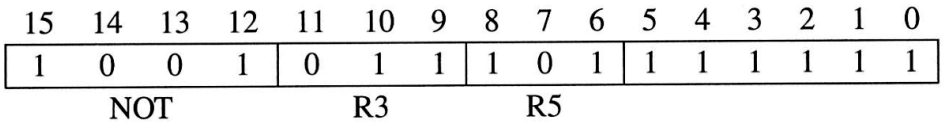
- Il *metodo di indirizzamento* è il meccanismo per specificare dove si trova un operando;
- LC-3 supporta cinque modi di indirizzamento:
  - all'interno dell'istruzione stessa (literal o immediate)
  - a registro
  - tre a memoria:
    - PC-relative
    - indiretto
    - base + offset

Le istruzioni di LC-3: operazioni

- Le operazioni sono istruzioni che processano i dati; possono essere di tipo aritmetico (ADD, SUB, MUL, DIV) o di tipo logico (AND, OR, NOT);
- LC-3 definisce tre operazioni:
  - NOT
  - AND
  - ADD
- Combinandole assieme è possibile ottenere tutte le altre operazioni aritmetiche e logiche.

NOT

- L'istruzione **NOT** (opcode = 1001) è l'unica ad utilizzare un solo operando (operazione unaria);
- Utilizza un registro sia per la sorgente che per la destinazione;
- Esegue il complemento bit a bit sui 16 bit del registro sorgente, e memorizza il risultato nel registro destinazione;
- Esempio: si vuole porre nel registro R3 la negazione del registro R5; l'istruzione sarà:

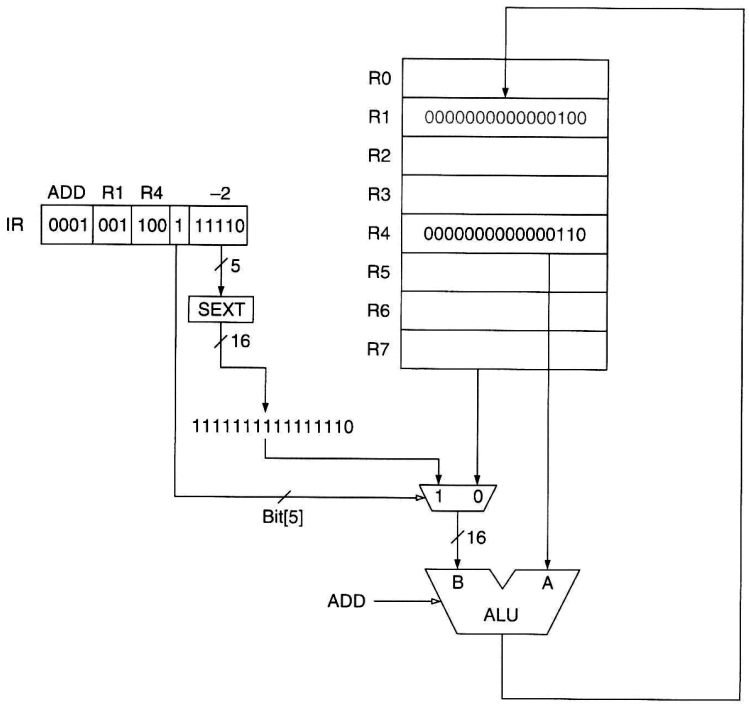


- Nota: i bit [0:5] sono tutti 1.

AND e ADD

- Le istruzioni **AND** (opcode = 0101) e **ADD** (opcode = 0001) operano su due operandi a 16 bit;
- L'ADD esegue l'addizione in complemento a due tra gli operandi;
- L'AND esegue l'and bit a bit tra gli operandi;
- A differenza di NOT, il secondo operando sorgente può essere:
  - un registro,
  - un valore costante.
- Nel primo caso il bit [5] vale 0, così come i bit [3:4]. I bit [0:2] contengono il numero del registro.
- Nel secondo caso invece, il bit [5] è settato a 1, e il valore costante è contenuto nei bit [0:4] (vedi esempio).

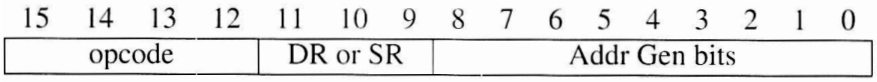
Esempio di esecuzione dell'istruzione "ADD R1, R4, #-2":



Le istruzioni di LC-3: spostamento di dati e metodi di indirizzamento

Struttura delle istruzioni di spostamento dati

- LC-3 definisce 7 istruzioni per spostare dati: LD, LDR, LDI, LEA, ST, STR e STI;
- In generale il formato delle istruzioni di load e store è:



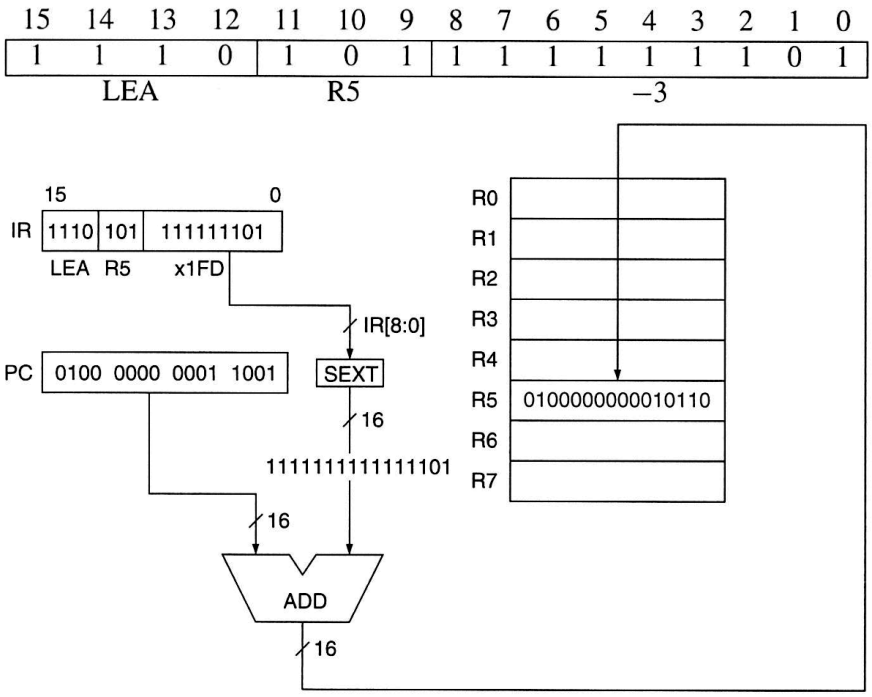
- I bit [0:8] contengono *l'address generation bits*, cioè l'informazione usata per calcolare l'indirizzo a 16 bit che costituisce il secondo operando dell'istruzione.
- **Il *modo di indirizzamento* specifica come devono essere interpretati questi bit**.

Indirizzamento immediato

- Si usa *SOLO* con l'istruzione di caricamento effettivo di un indirizzo (Effective Load Address LEA);

L'istruzione LEA (opcode = 1110) carica nel registro specificato dai bit [9:11] il valore ottenuto sommando il valore del PC incrementato con il valore indicato nei bit [0:8] dell'istruzione (rappresentato su 16 bit).

- Di solito l'istruzione LEA viene usata per inizializzare un registro con un valore molto vicino all'indirizzo di memoria che si sta usando;
- Esempio: supponiamo che la locazione di memoria 0x4018 contenga l'istruzione "LEA R5, #-3" e il PC sia 0x4018; dopo l'esecuzione dell'istruzione 0x4018, R5 conterrà 0x4016.

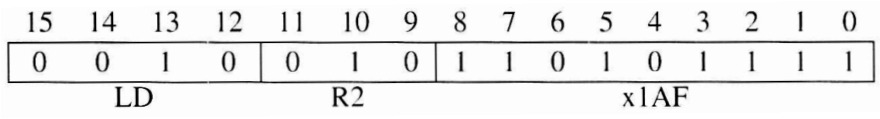


Indirizzamento PC-relative

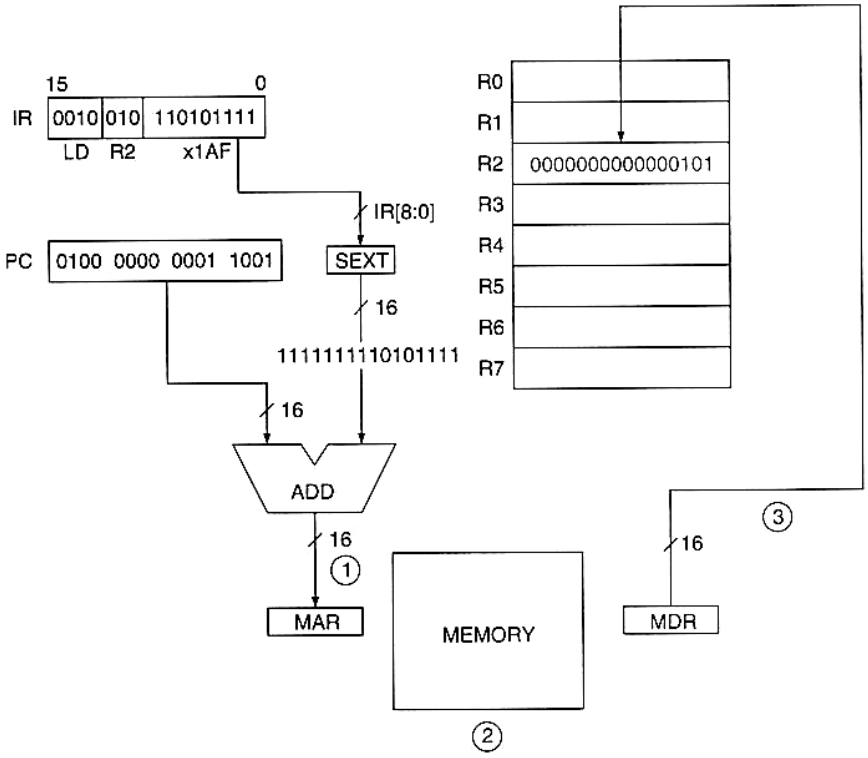
- Le istruzioni **LD** (opcode = 0010) e **ST** (opcode = 0011) specificano l'indirizzamento PC-relative;
- Si chiama così perchè i bit [0:8] specificano l'offset da sommare al PC;

  L'indirizzo di memoria a cui fa riferimento l'operazione viene calcolato sommando al valore del PC incrementato il valore, esteso su 16 bit, indicato dai bit [0:8] dell'istruzione.

- Se l'operazione è di load, viene messo nel registro indicato dai bit [9:11] il valore contenuto all'indirizzo di memoria calcolato; se è di store, viene memorizzato all'indirizzo di memoria calcolato il valore contenuto nel registro indicato dai bit [9:11].
- Esempio: supponiamo che il valore del PC sia 0x4018 e l'istruzione sia "LD R2, x1AF"; in R2 verrà caricato il valore contenuto in memoria all'indirizzo 0x3FC8.

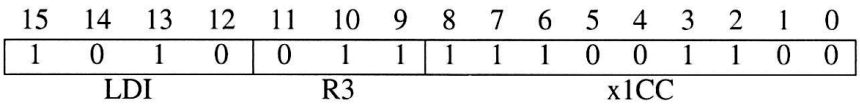


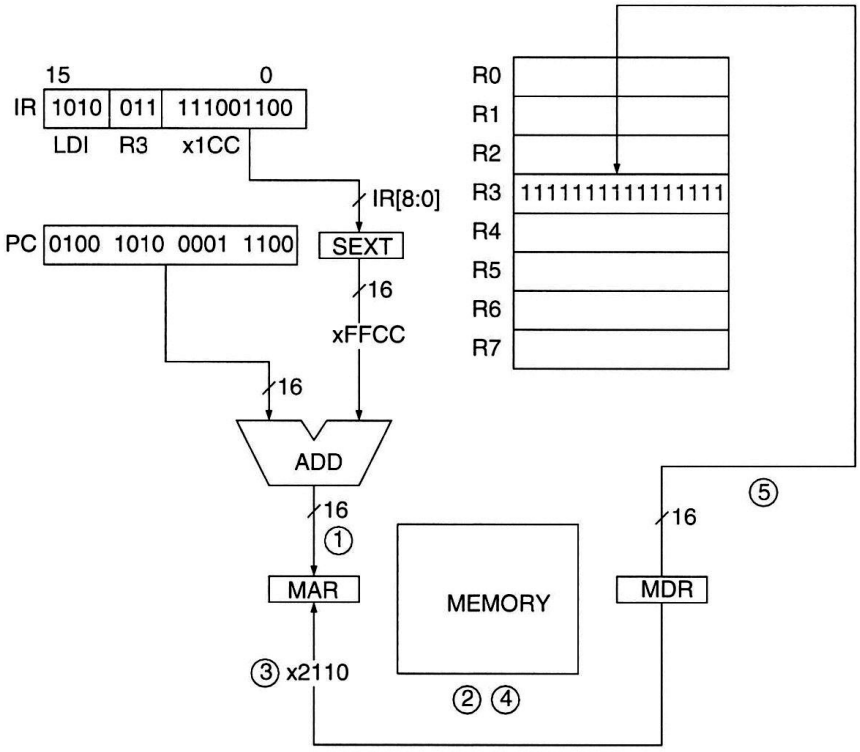
- Per eseguire l'istruzione è necessario effettuare un *accesso alla memoria*.



Indirizzamento indiretto

- Le istruzioni **LDI** (opcode = 1010) e **STI** (opcode = 1011) utilizzano l'indirizzamento indiretto.
- In questo caso l'indirizzo calcolato dalla somma tra il PC e il valore specificato nell'istruzione, *non rappresenta direttamente l'indirizzo di memoria da/su cui leggere/scrivere,* ***ma l'indirizzo di memoria che contiene l'indirizzo dell'operando da caricare/salvare***;
- Esempio: supponiamo che l'istruzione in 0x4A1B sia "LDI R3, x1CC", e che la zona di memoria 0x49E8 (ottenuto sommando il PC incrementato, 0x4A1C, all'estensione su 16 bit con segno dell'offset, che da 0xFFCC) contenga 0x2110; l'esecuzione sarà:
  - somma dell'offset al PC (il cui risultato è l'indirizzo 0x49E8 (fase 1);
  - il contenuto della cella di memoria all'indirizzo 0x49E8 (0x2110) rappresenta l'indirizzo di memoria vero e proprio a cui l'operazione fa riferimento (fase 2);
  - tale indirizzo (0x2110) viene caricato nel MAR (fase 3);
  - viene preso il contenuto della cella di memoria 0x2110 e il suo contenuto viene messo nel registro di destinazione (R3) (fasi 4 e 5).





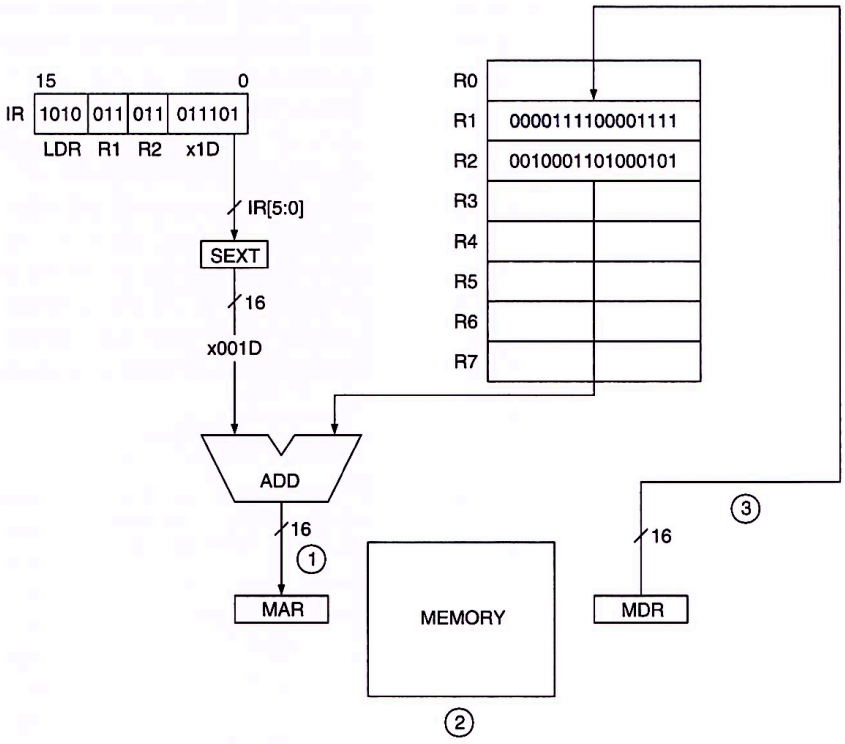
Indirizzamento base+offset

- Le istruzioni **LDR** (opcode = 0110) e **STR** (opcode = 0111) specificano l'indirizzamento base+offset;

  L'indirizzo dell'operando è ottenuto aggiungendo l'offset indicato dai 6 bit [0:5], esteso a 16 bit, al registro di base indicato dai bit [6:8].

- I 6 bit utilizzati per rappresentare l'offset vanno sempre considerati come interi in complemento a 2 (quindi valori compresi fra -32 e +31);

- Esempio: si supponga che R2 contenga il valore 0x2345, e che l'istruzione da eseguire sia "LDR R1, R2, x1D"; dopo la sua esecuzione, in R1 si avrà il contenuto della cella di memoria calcolata sommando 0x2345 a 0x1D.



Le istruzioni di LC-3: controllo del flusso

LC-3 - Controllo

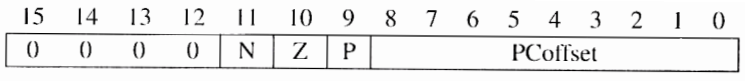
- LC-3 ha 5 operazioni (opcode) riservate per il controllo di flusso:
  - **salto condizionale**
  - **salti non condizionali**
  - chiamata a funzione
  - TRAP
  - return da un'interrupt
- In questo corso vedremo solo i salti condizionali e non.

Nota:
LC-3 ha 3 registri di un bit che sono settati a 1 o a 0 ogni qual volta uno degli otto registri general purpose vengono scritti. Essi sono:

- negative (indicato con N)
- zero (indicato con Z)
- positive (indicato con P)

Salti condizionali

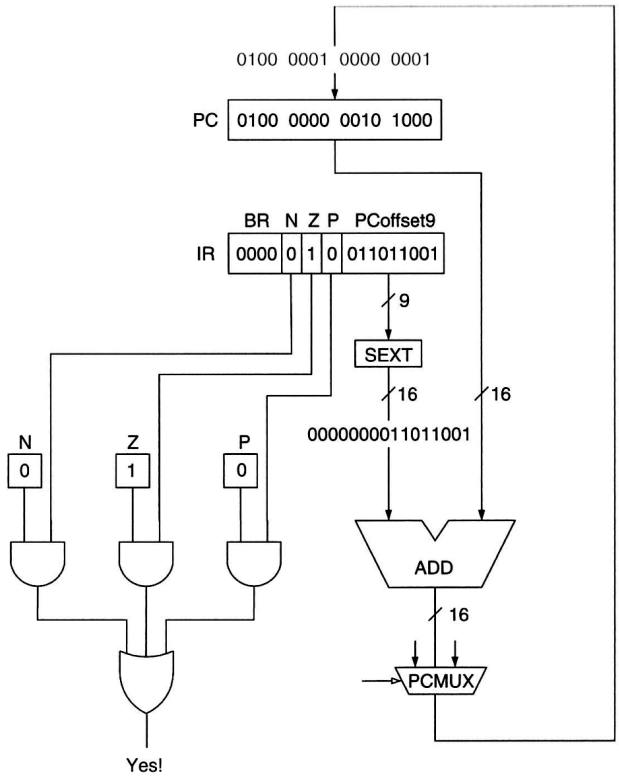
- L'istruzione di branch è **BR** (opcode = 0000);
- Il suo formato è:



- I bit [9:11] corrispondono alle tre condizioni viste; essi sono usati per cambiare il flusso di esecuzione;
- Durante la fase di ESECUZIONE il processore valuta i bit N, Z, P, e:
  - se il bit [11] = 1 viene esaminata la condizione corrispondente a N;
  - se il bit [10] = 1 viene esaminata la condizione corrispondente a Z;
  - se il bit [9] = 1 viene esaminata la condizione corrispondente a P;
- *Solo se nessuna delle condizioni verificate è vera viene caricato nel PC il valore calcolato nella fase di ADDRESS EVALUATION*;
- **Altrimenti...**

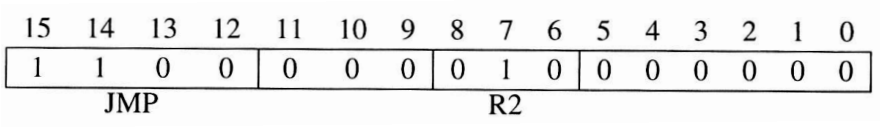
...
Verrà aggiornato il PC con il valore calcolato come somma tra il PC incrementato e il PC offset indicato nell'istruzione.

Esempio: supponiamo che l'istruzione precedente abbia prodotto in un registro general purpose il valore 0; il PC ha valore 0x4027 e l'istruzione corrente è "BRz x0D9". Dopo l'esecuzione di questa istruzione il PC avrà valore 0x4101 e non 0x4028.



Salti non condizionali

- LC-3 mette a disposizione l'istruzione **JMP** (opcode = 1100) per eseguire salti non condizionati;
- JMP carica nel PC il valore contenuto nel registro indicato dai bit [6:8];
- Esempio:



Costrutti condizionali e cicli

- Le istruzioni di controllo viste ci permettono di scrivere programmi per LC-3 che contengono costrutti condizionali e cicli;

  Due metodi per il controllo dei cicli:

  - *for*: a priori si ha la conoscenza del numero di volte di esecuzione; si utilizza quindi un registro contatore (ad esempio R2) che viene decrementato ad ogni ciclo. Esempio: somma dei primi dodici numeri a partire da una locazione di memoria.
  - *while*: non si ha a priori la conoscenza sul numero di volte in cui il ciclo verrà eseguito; si utilizza la tecnica della **sentinella**. Supponiamo ad esempio che di voler sommare un insieme di valori tutti positivi (zero escluso) memorizzati in memoria; al termine di tali valori occorre piazzare una sentinella, che in questo caso potrebbe essere lo zero. In questo modo si può uscire dal ciclo appena si trova uno zero.

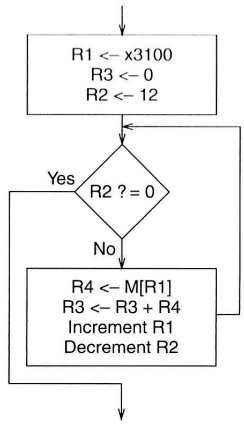
- Analizziamo questi esempi...

Il ciclo FOR in LC-3

Esempio: somma dei primi dodici numeri memorizzati a partire da una locazione di memoria. Di seguito è riportata una possibile implementazione nei linguaggi C ed Assembly.

```
int main(){
 int A[20]={1,2,3,4,5,6,7,8,9,10,11,12,0,0,0,0,0,0,0,0};
 int i;
 int r=0;
 int *p = &A[0];
 for(i=12;i>0;i--){
   r=r+(*p);
   p++;
 }
}
```

```
.ORIG x3000
LD R1, x3100  ; R1 corrisponde a p
              ; x3100 è l'indirizzo di A[0]
AND R3,R3,#0  ; corrisponde a r = 0;
AND R2,R2,#0
ADD R2,R2,#12 ; corrisponde a i = 12
BRz x300A     ; se i == 0 esci dal for
LDR R4,R1,x0
ADD R3,R3,R4  ; corrisponde a r=r+(*p)
ADD R1,R1,#1  ; corrisponde a p++
ADD R2,R2,#-1 ; corrisponde a i--
BRnzp x3004   ; ripeti
.END
```



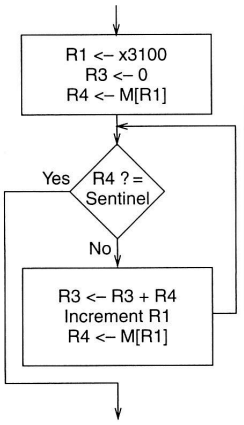
Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
x3000	1	1	1	0	0	0	1	0	1	1	1	1	1	1	1	1	R1<- 3100
x3001	0	1	0	1	0	1	1	0	1	1	1	0	0	0	0	0	R3 <- 0
x3002	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	R2 <- 0
x3003	0	0	0	1	0	1	0	0	1	0	1	0	1	1	0	0	R2 <- 12
x3004	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	BRz x300A
x3005	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	R4 <- M[R1]
x3006	0	0	0	1	0	1	1	0	1	1	0	0	0	1	0	0	R3 <- R3+R4
x3007	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	1	R1 <- R1+1
x3008	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1	1	R2 <- R2-1
x3009	0	0	0	0	1	1	1	1	1	1	1	1	1	0	1	0	BRnzp x3004

Il ciclo WHILE in LC-3

Esempio: somma dei numeri maggiori di zero memorizzati a partire da una locazione di memoria. Di seguito è riportata una possibile implementazione nei linguaggi C ed Assembly.

```
int main(){
 int A[20]={1,2,3,4,5,6,7,8,9,10,11,12,0,0,0,0,0,0,0,0};
 int *p=&A[0];
 int r=0;
 while(*p>0){  //Nota: 0 è la sentinella
   r=r+(*p);
   p++;
 }
}
```

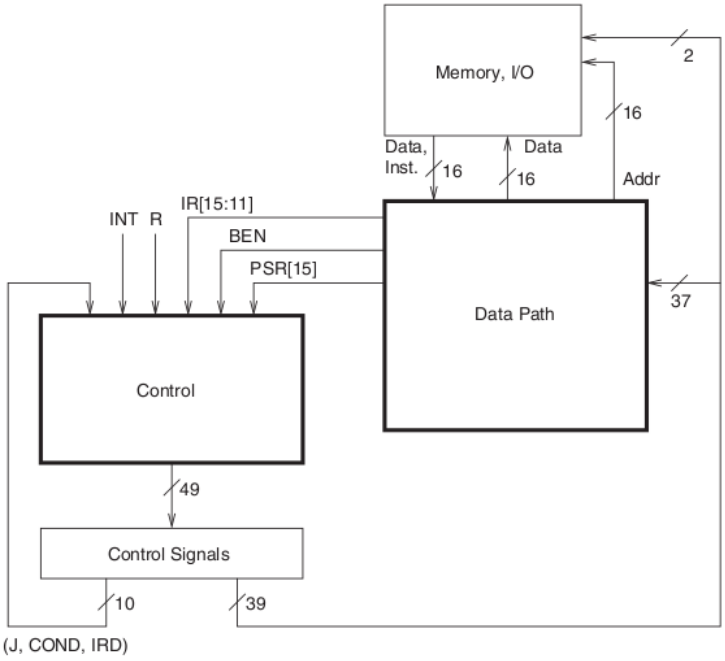
```
.ORIG x3000
LD R1, x3100 ; R1 corrisponde a p
             ; x3100 è l'indirizzo di A[0]
AND R3,R3,#0 ; corrisponde a r = 0
LDR R4,R1,x0
BRn x3008    ; if *p < 0 esci dal ciclo
ADD R3,R3,R4 ; corrisponde a r = r+(*p)
ADD R1,R1,#1 ; corrisponde a p++
LDR R4,R1,x0
BRnzp x3003  ; ripeti
.END
```



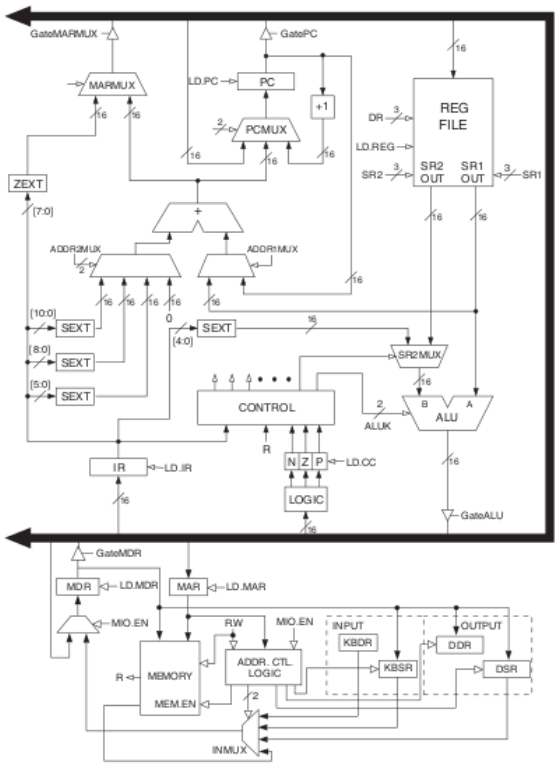
Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
x3000	1	1	1	0	0	0	1	0	1	1	1	1	1	1	1	1	R1<- x3100
x3001	0	1	0	1	0	1	1	0	1	1	1	0	0	0	0	0	R3 <- 0
x3002	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	R4 <- M[R1]
x3003	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	BRn x3008
x3004	0	0	0	1	0	1	1	0	1	1	0	0	0	1	0	0	R3 <- R3+R4
x3005	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	1	R1 <- R1+1
x3006	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	R4 <- M[R1]
x3007	0	0	0	0	1	1	1	1	1	1	1	1	1	0	1	1	BRnzp x3003

LC3 - Approfondimento

- Le due componenti principali di un'ISA sono:
  - **controllore**: contiene tutte le componenti per generare i segnali di controllo necessari per controllare i processi ad ogni istante di tempo;
  - **datapath**: è l'insieme di tutte le componenti che eseguono le istruzioni.
- In questa lezione abbiamo visto le componenti principali del datapath. Vediamo ora la struttura completa del datapath, e un breve cenno al controllore.
- Maggiori dettagli si possono trovare all'appendice C del testo.
- La microarchitettura di LC-3 è la seguente:



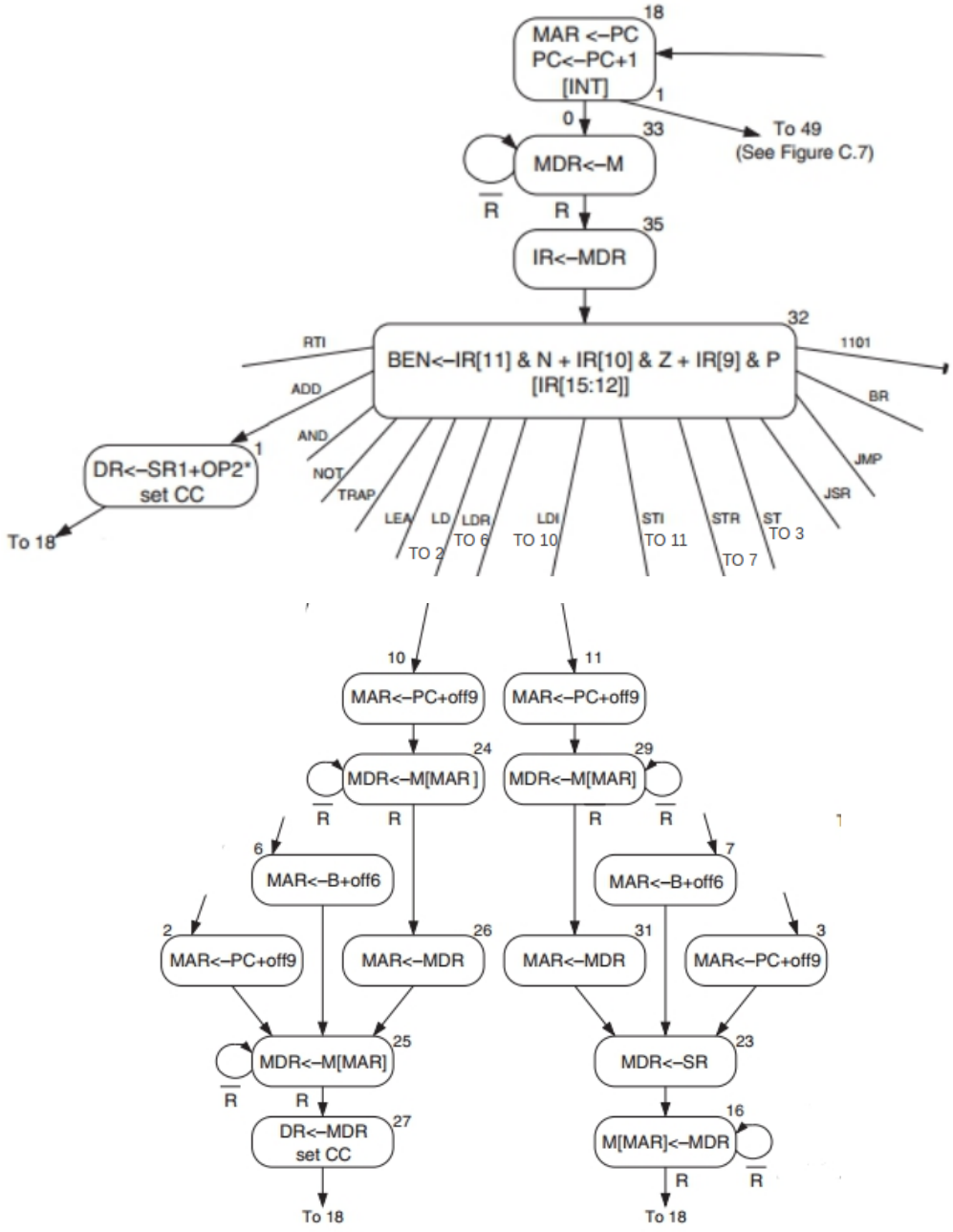
- Il datapath è l'insieme di tutte le componenti che processano le informazioni durante il ciclo d'esecuzione;
- Occorre notare che ad ogni componente è associato un segnale di controllo;
- In tutto sono 39 bit di controllo diretto che, ad ogni ciclo di clock, fanno collaborare le componenti del datapath per processare correttamente i dati;
- Di seguito viene riportato uno schema complessivo del datapath.



- Il controllore è la componente che stabilisce il comportamento della microarchitettura durante un ciclo di clock;
- E' implementato da una macchina a stati finiti (FSM);
- Ogni ciclo di clock è completamente determinato da 49 bit (più ulteriori 9); 39 di questi servono per far si che il datapath processi i dati, e 10 bit servono per determinare lo stato futuro dell'FSM;
- Complessivamente i 49 bit sono chiamati **microistruzioni**;
- L'FSM è composta in tutto da 52 stati distinti, ciascuno dei quali corrisponde ad una microistruzione;

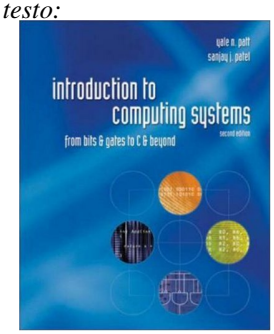
Nota:
Dall'FSM si evince quanti cicli di clock sono effettivamente necessari per eseguire un'istruzione.

- Riportiamo di seguito un frammento dell'FSM:


18
MAR <–PC
PC<–PC+1
[INT]
1
To 49
(See Figure C.7)
0
33
MDR<–M
R
R
35
IR<–MDR
32
BEN<–IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
RTI
ADD
AND
NOT
TRAP
LEA
LD
LDR
LDI
STI
STR
ST
JSR
JMP
BR
1101
1
DR<–SR1+OP2*
set CC
To 18
TO 2
TO 6
TO 10
TO 11
TO 7
TO 3
10
MAR<–PC+off9
11
MAR<–PC+off9
24
MDR<–M[MAR]
29
MDR<–M[MAR]
6
MAR<–B+off6
7
MAR<–B+off6
2
MAR<–PC+off9
26
MAR<–MDR
31
MAR<–MDR
3
MAR<–PC+off9
25
MDR<–M[MAR]
23
MDR<–SR
27
DR<–MDR
set CC
16
M[MAR]<–MDR
To 18
To 18

Riferimenti, simulatore e manuali

Libro di testo:



Yale N. Patt, Sanjay J. Patel.
Introduction to Computing Systems:
From Bits and Gates to C and Beyond.
Seconda edizione.
McGraw-Hill Higher Education, 2003.

`http://www.mhhe.com/patt2`

Contiene il simulatore, il manuale del simulatore e le appendici con l'elenco completo delle istruzioni e la descrizione dell'ISA