Architettura degli Elaboratori I - B

Le Microarchitetture

Il Ciclo Multiplo

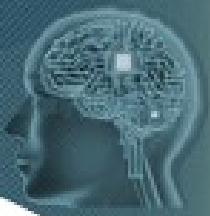
Daniel Riccio/Alberto Aloisio
Università di Napoli, Federico II

10 aprile 2018

Rif. Capitolo 7

Digital Design and Computer Architecture-ARM, Harris-Harris, Edition-Morgan Kaufmann.

Sommario



- Limiti delle architetture a ciclo singolo;
- Architetture a ciclo multiplo;
- Il Datapath e l'Unità di Controllo.

Limiti del ciclo singolo

Le architetture a ciclo singolo hanno tre principali limiti:

- **separazione delle memorie**: la memoria istruzioni e la memoria dati devono essere necessariamente separate, poiché dati e istruzioni devono essere gestiti all'interno dello stesso ciclo.

- **inefficienza temporale**: il ciclo di clock deve avere una durata pari al tempo impiegato dall'istruzione più lenta, sprecando tempo per tutte quelle istruzioni molto più veloci.

- **duplicazione delle componenti**: lo stesso componente non può essere riutilizzato per scopi distinti; ad esempio sono necessarie tre ALU, due per la gestione del PC e una per l'esecuzione delle istruzioni.

Vantaggi del ciclo multiplo

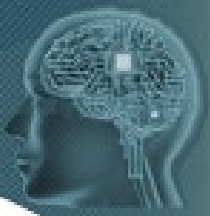
Le architetture a ciclo multiplo risolvono tali problemi, partizionando una intera istruzione in più passi, ciascuno dei quali viene eseguito in un ciclo di clock differente.

- È possibile utilizzare una sola memoria comune sia per le istruzioni, che per i dati. Infatti, l'istruzione viene letta in un ciclo, mentre i dati vengono letti o scritti in memoria in un ciclo differente.

- Istruzioni meno complesse richiedono un minor numero di cicli di clock, evitando sprechi di tempo.

- È possibile utilizzare un'unica ALU sia per gestire il PC, che per eseguire le istruzioni, purché tali operazioni siano effettuate in cicli di clock differenti.

Il datapath



Il **datapath** è sviluppato in modo incrementale aggiungendo di volta in volta agli elementi di stato. I nuovi collegamenti sono evidenziati in **nero** (o **blu**), mentre quanto già introdotto è rappresentato in **grigio**.

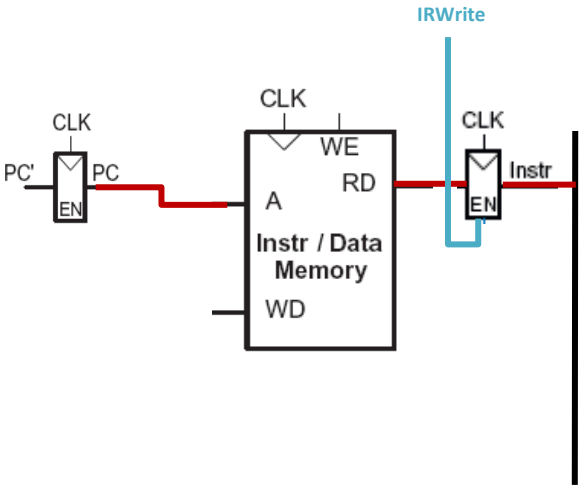
Al fine di facilitare la comprensione dell'architettura di un processore ARM, considereremo un limitato set di istruzioni:

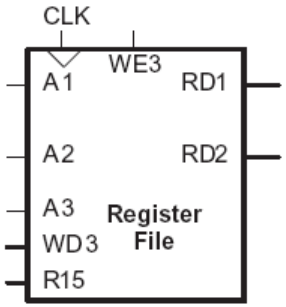
- le istruzioni di elaborazione dati: **ADD**, **SUB**, **AND**, **ORR** (con registro e modalità di indirizzamento diretto e senza shift);
- le istruzioni di Memoria: **LDR**, **STR** (diretto e con offset positivo);
- le istruzioni di salto (branch): **B**.

Il datapath (LDR)

Il **PC** contiene l'indirizzo dell'istruzione da eseguire. Il primo passo è quello di leggere questa istruzione dalla memoria istruzioni, per cui il PC viene collegato all'indirizzo di ingresso della memoria.

L'istruzione a 32 bit viene letta e memorizzata in un registro **IR**. Il registro **IR** riceve un segnale **IRWrite** che indica quando caricare una istruzione.



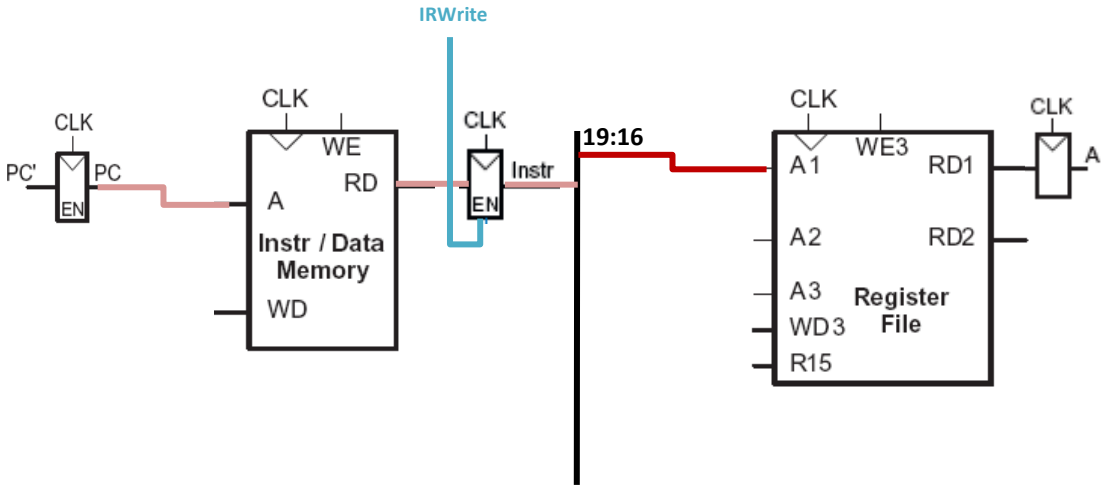


Il datapath (LDR)

Il passo successivo è quello di leggere il registro sorgente contenente l'indirizzo di base. Questo registro è specificato nel campo **Rn** dell'istruzione, **$Instr_{19:16}$**

Questi bit vengono collegati all'ingresso indirizzo di una delle porte del file register (**A1**).

Il register file legge il valore di registro in **RD1** e lo memorizza in un registro **A**.

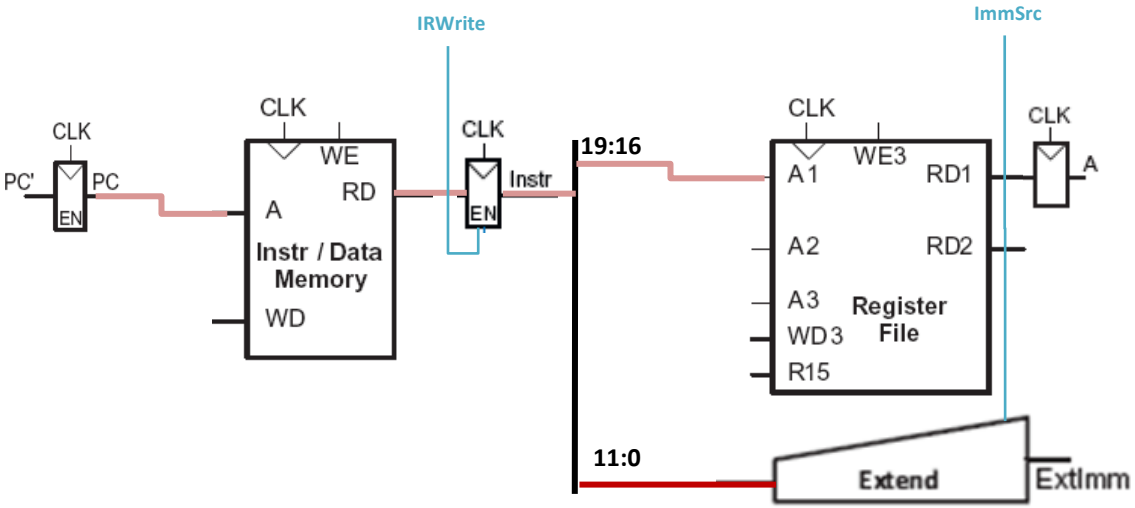


Il datapath (LDR)

L'istruzione **LDR** richiede anche un **offset**, il quale è memorizzato nell'istruzione stessa e corrisponde ai bit $\mathbf{Instr}_{11:0}$.

L'offset è un valore senza segno, quindi deve essere esteso a 32 bit.

Il valore a 32 bit (**ExtImm**) è tale che $\mathbf{ExtImm}_{31:12} = 0$ e $\mathbf{ExtImm}_{11:0} = \mathbf{Instr}_{11:0}$.
ExtImm estende a 32 bit costanti a 8, 12 e 24 bit. Non viene memorizzato in un registro, poiché dipende solo da **Instr**, che non cambia durante l'esecuzione dell'istruzione.

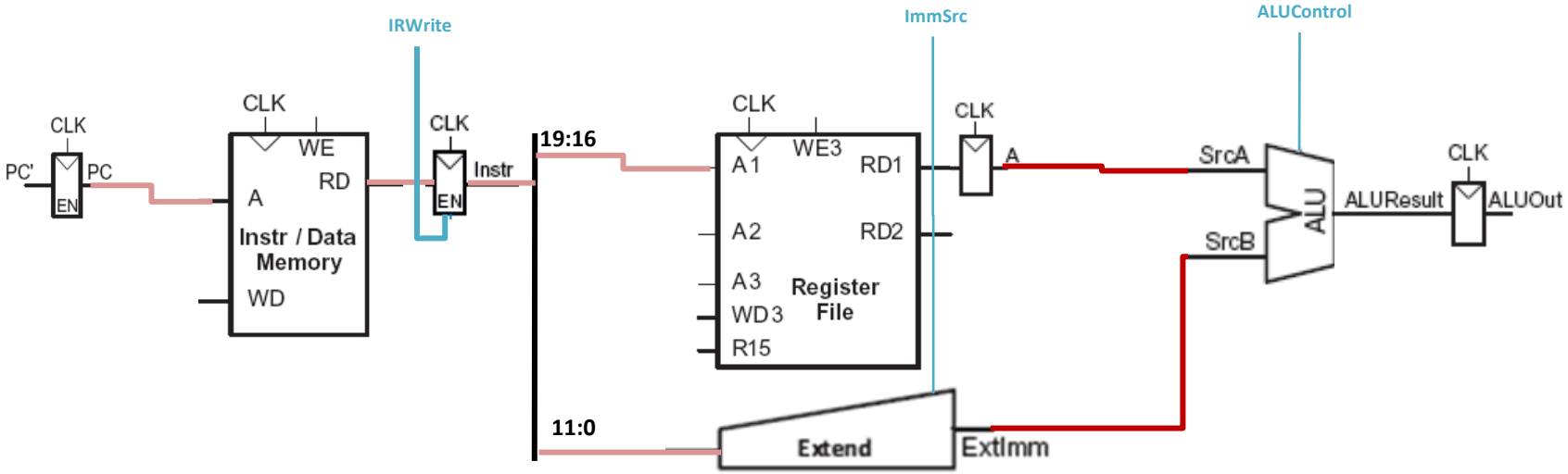


Il datapath (LDR)

Il processore deve aggiungere l'**indirizzo di base** all'offset per trovare l'indirizzo di memoria a cui leggere. La somma è effettuata per mezzo di una **ALU**.

La **ALU** riceve due operandi (**srcA** e **srcB**). **srcA** proviene dal register file, mentre e **srcB** è il valore già contenuto nell'**ALU**. Inoltre, il segnale a 2-bit **ALUControl** specifica l'operazione (una somma è indicata con 00).

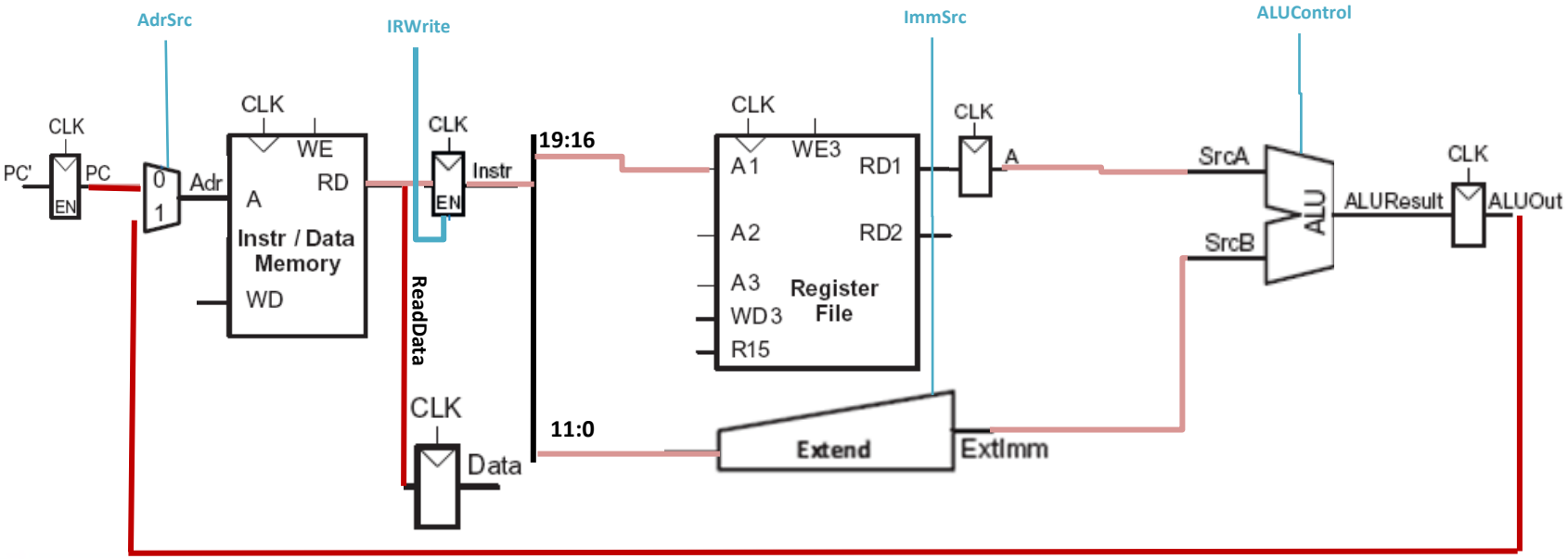
La **ALU** genera un valore a 32 bit **ALUResult**, che viene memorizzato in un registro **ALUOut**.



Il datapath (LDR)

L'indirizzo calcolato dall'**ALU** deve essere inviato alla porta **A** della memoria. Serve un multiplexer per disambiguare l'accesso con **ALUOut** o **PC**. Il multiplexer è controllato dal segnale **AdrSrc**.

I dati vengono letti dalla memoria dati sul bus **ReadData**, e poi vengono memorizzati in un registro chiamato **Data**.

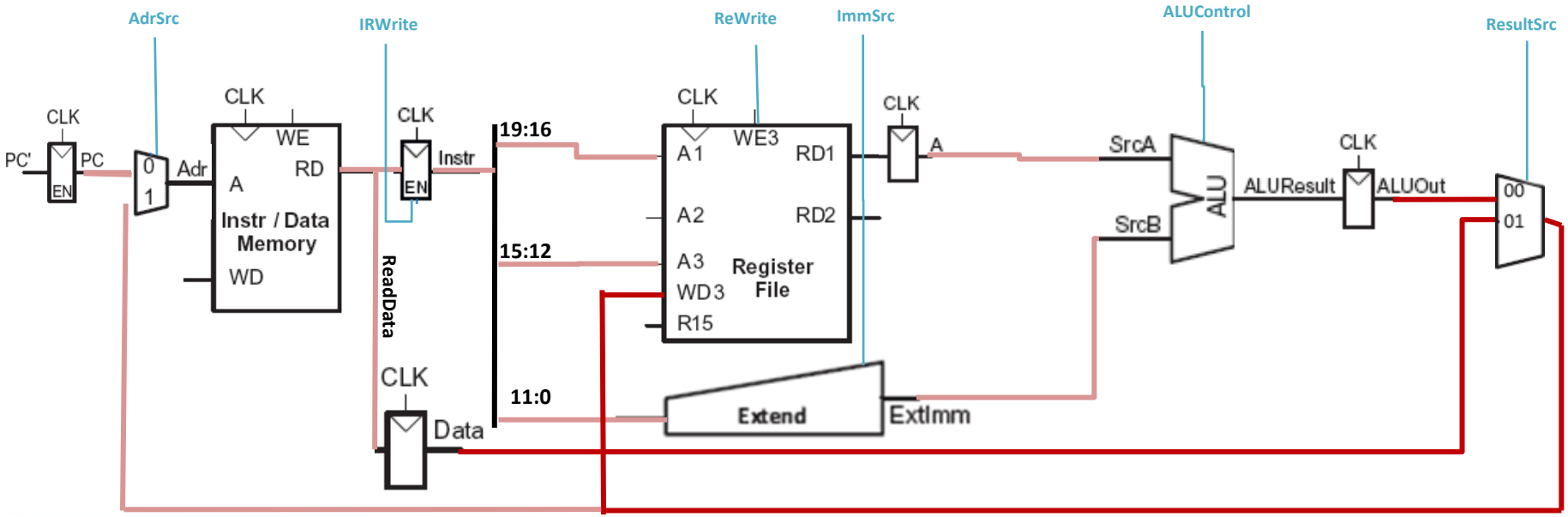


Il datapath (LDR)

Inoltre, i dati appena letti devono essere scritti nel registro **Rd** specificato dai bit **15:12** dell'istruzione **Instr**.

Piuttosto che collegare direttamente **Data** alla porta **WD3**,si consideri che anche il risultato dell'**ALU** potrebbe dover essere scritto in **Rd**. Quindi aggiungiamo un multiplexer che seleziona fra **ALUOut** e **Data**.

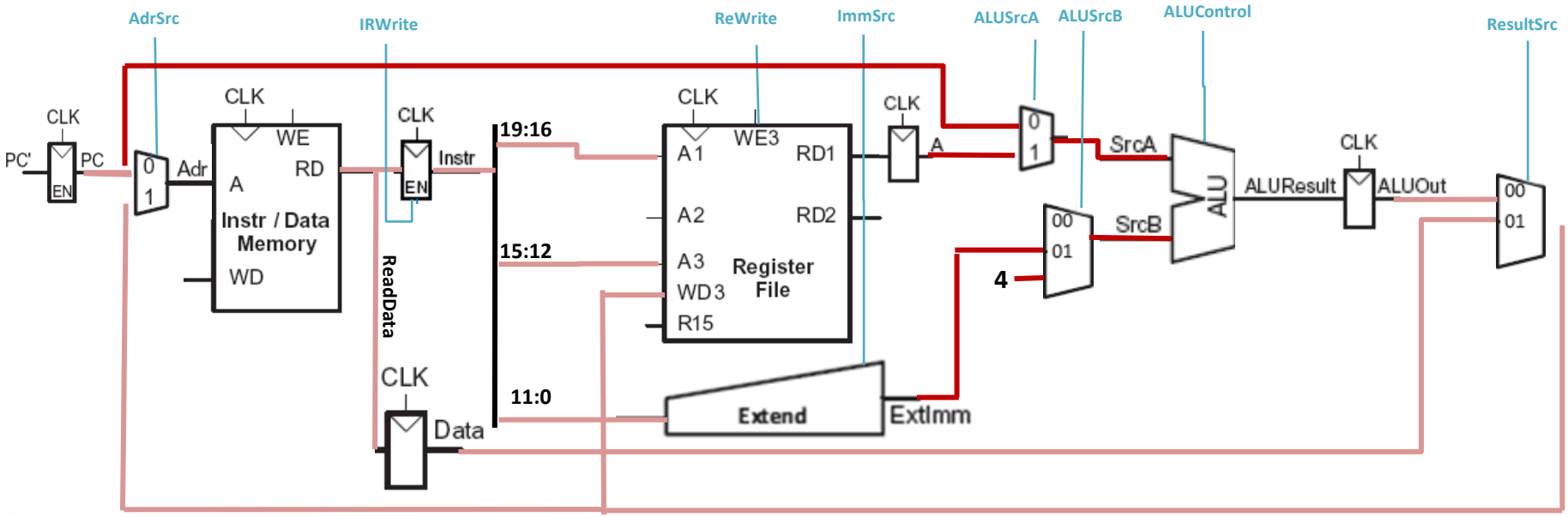
Il segnale **RegWrite** deve essere impostato a **1**, per permettere la scrittura nel registro.



Il datapath (LDR)

Un ulteriore compito a carico dell'**ALU** è l'incremento del **PC**, operazione che prima era svolta da una **ALU** diversa.

Aggiungiamo un multiplexer sul primo ingresso dell'**ALU**, che permette di scegliere fra il contenuto del registro **A** e il **PC**.
Sul secondo ingresso dell'ALU aggiungiamo un ulteriore multiplexer che permetta di selezionare fra ExtImm e la costante 4.

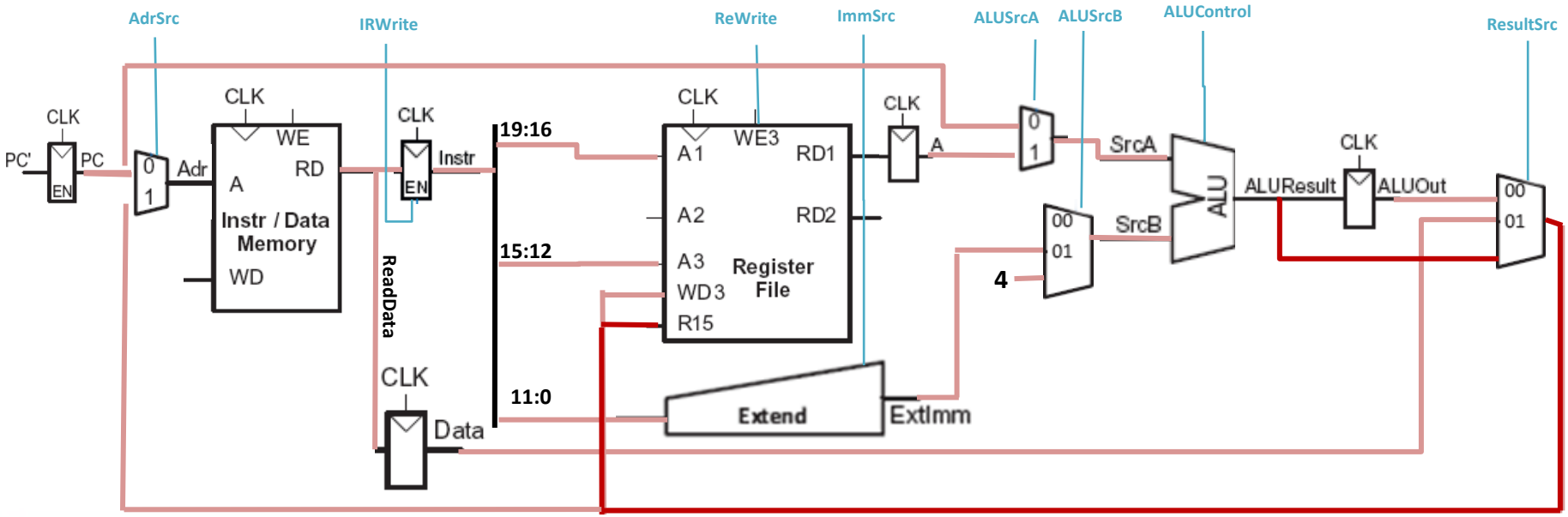


Il datapath (LDR)

Si consideri infine, che il contenuto del registro **R15** nelle architetture ARM corrisponde a **PC+8**.

Durante il passo di fetch, il **PC** è stato aggiornato a **PC+4**, per cui sommare 4 al nuovo contenuto di **PC** produce **PC+8**, che viene memorizzato in **R15**.

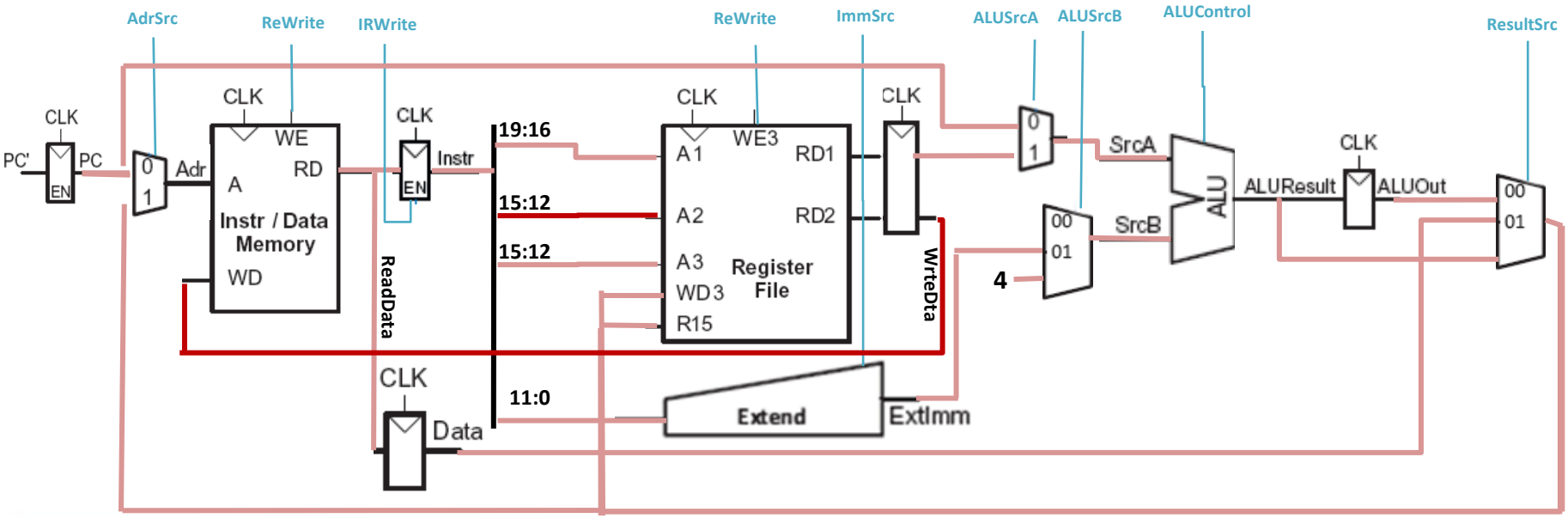
Scrivere **PC+8** in **R15**, richiede che il risultato dell'ALU possa essere collegato a tale registro. A tal fine, colleghiamo **ALUResult** con uno dei tre ingressi del multiplexer.



Il datapath (STR)

Analogamente all'istruzione di caricamento, **STR** legge l'indirizzo di base dalla porta **RD1** del register file, estende la costante e l'**ALU** somma i due valori per calcolare l'indirizzo di memoria. Tutte queste operazioni sono già supportate.

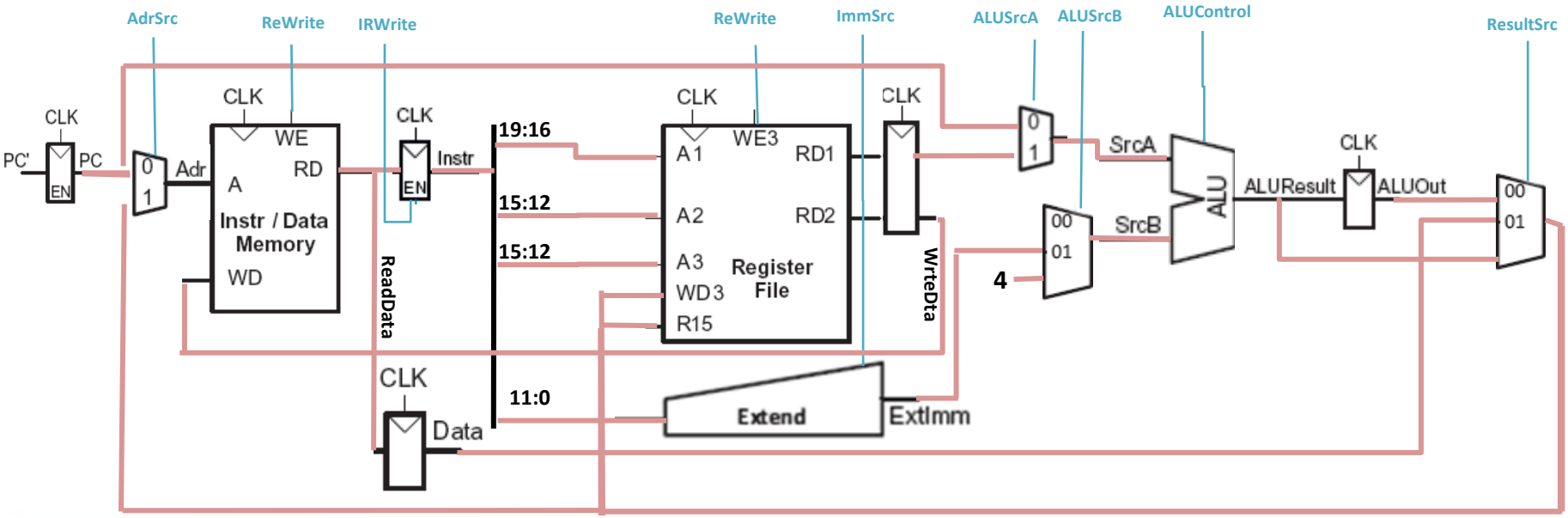
In aggiunta, **STR** legge il registro **Rd** in cui scrivere, che è specificato nei bit **$Instr_{15:12}$**. Il contenuto di **RD2** è inserito in un registro temporaneo WriteData e al passo successivo è inviato alla memoria, con segnale **MemWrite** attivo.



Il datapath (data processing con costante)

Per le istruzioni di data processing con costante (**ADD**, **SUB**, **AND**, **OR**), il datapath legge il primo operando specificato da **Rn**, estende la costante da **8** a **32** bit, esegue l'operazione mediante l'**ALU** e scrive il risultato in un registro del register file. Tutte queste operazioni sono già supportate dal datapath.

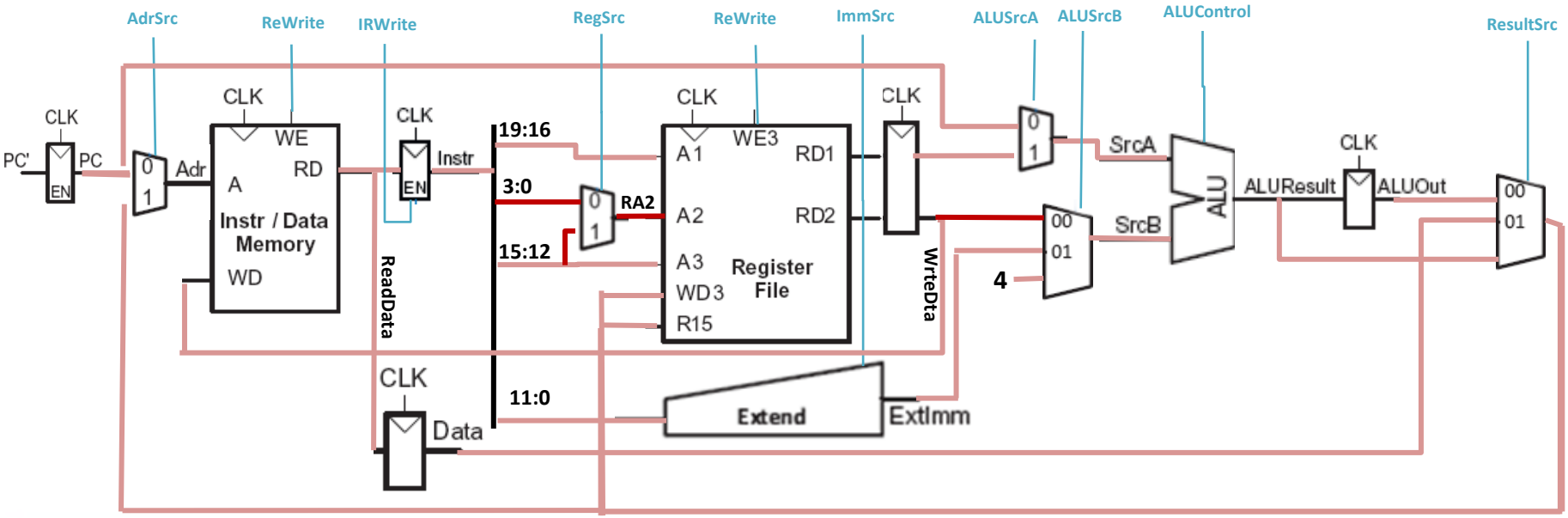
L'operazione da effettuare è specificata dal egnale **ALUControl**, mentre gli **ALUFlags** permettono di aggiornare il registro di stato.



Il datapath (data processing con registro)

Per le istruzioni di data processing con registro (**ADD**, **SUB**, **AND**, **OR**), il datapath legge il secondo operando specificato da **Rm**, indicato nei bit $\mathbf{Instr}_{3:0}$.

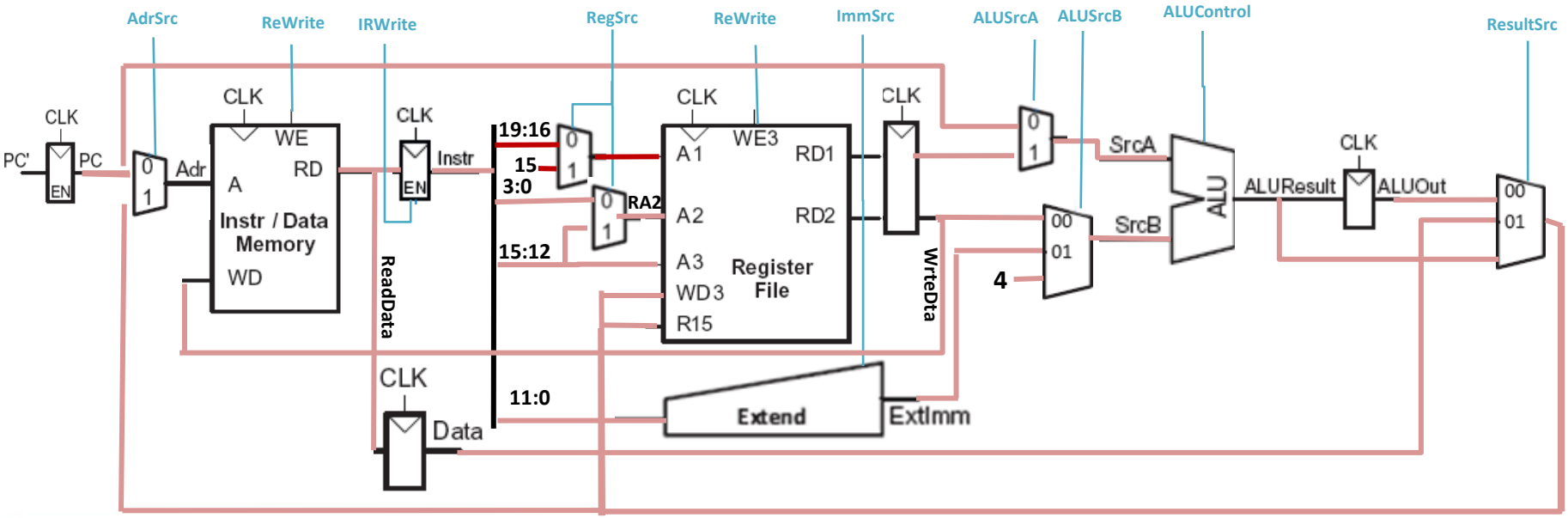
Inseriamo un multiplexer per selezionare tale campo sulla porta **A2** del register file. Il mutiplexer è controllato dal segnale **RegSrc**. Inoltre, estendiamo il multiplexer in **SrcB** in modo da considerare questo caso.



Il datapath (Branch)

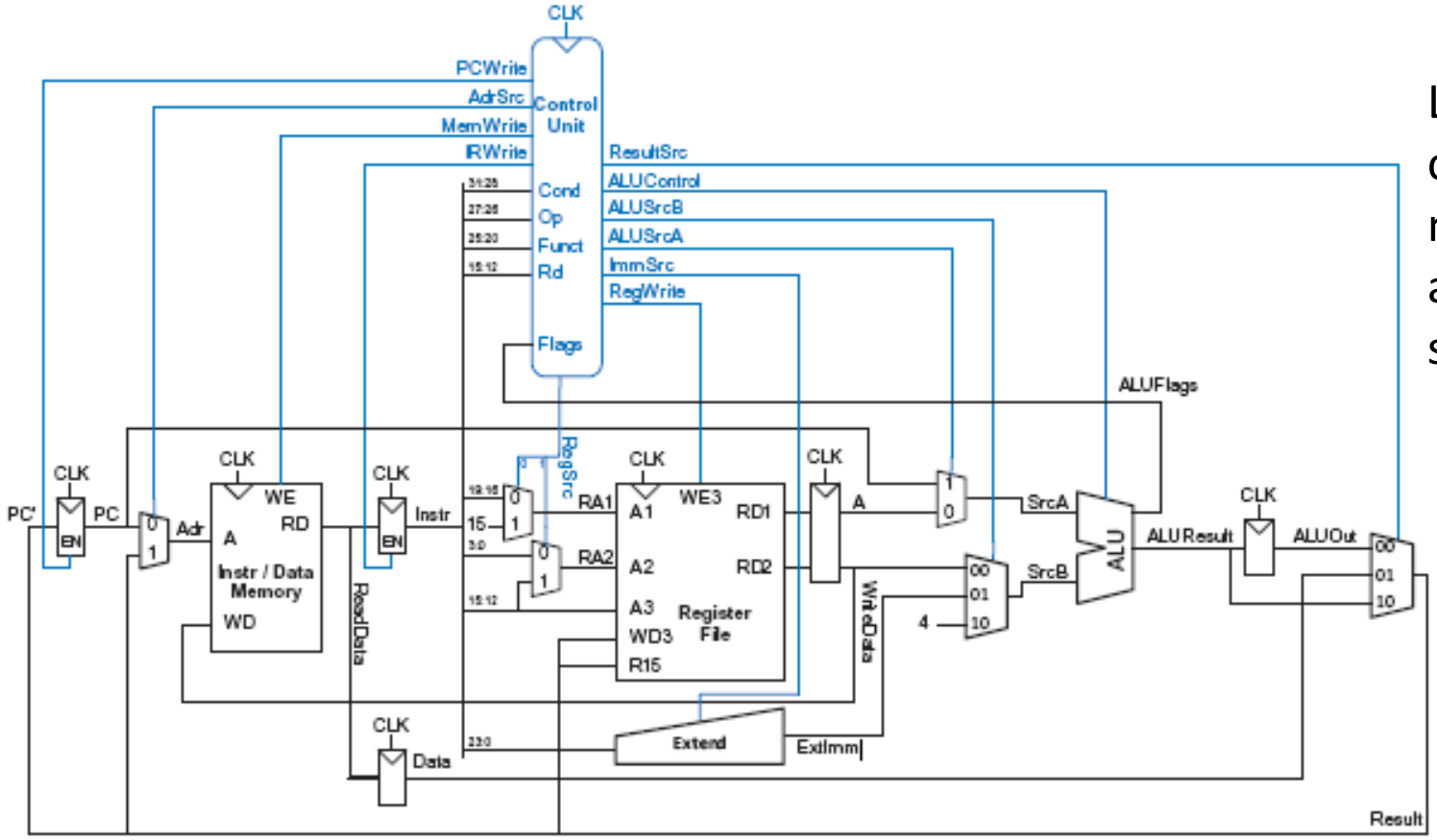
Per le istruzioni di branch, il datapath legge **PC+8** e una costante a **24** bit, che viene estesa a **32** bit. La somma di questi due valori è addizionata al **PC**.
Si ricorda, inoltre, che il registro **R15** contiene il valore **PC+8** e deve essere letto per tornare da un salto. È sufficiente aggiungere un multiplexer per selezionare **R15** come input sulla porta **A1**.

Il multiplexer è controllato dal segnale **RegSrc**.



Unità di Controllo

Come nel processore a ciclo singolo, l'unità di controllo genera i segnali di controllo in base ai campi **cond**, **op** e **funct** dell'istruzione (**Instr**$_{31:28}$, **Instr**$_{27:26}$, e **Instr**$_{25:20}$, ai flag e al fatto che il registro destinazione sia meno il PC.

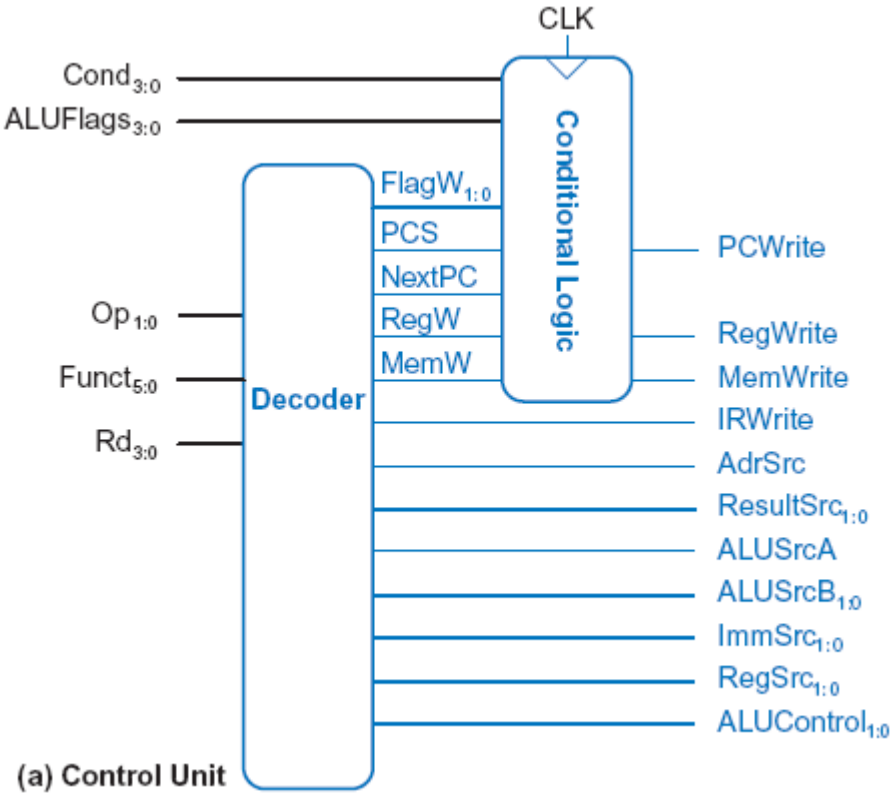


L'unità di controllo memorizza e aggiorna i flag di stato.

Unità di Controllo

Come nel processore a ciclo singolo, l'unità di controllo è suddivisa in Decodificatore e logica combinatoria. Il decodificatore è progettato come una Macchina a stati finiti, che produce i segnali appropriati per i diversi cicli, sulla base del proprio stato.

Il decodificatore è realizzato con una macchina di Moore, in tal modo le uscite dipendono solo dello stato attuale.

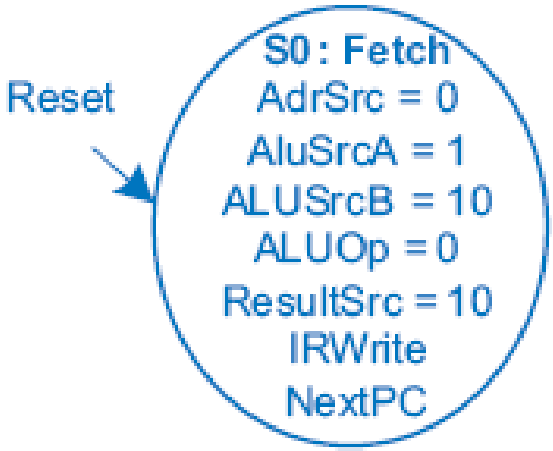


Il data flow di una istruzione

L'unità di controllo produce i segnali di attivazione per tutto il datapath (selezione nei multiplexer, abilitazione dei registri e scrittura in memoria).

Uno stato dell'automa che implementa il **main decoder** non è altro che lo stato dei segnali in un determinato momento.

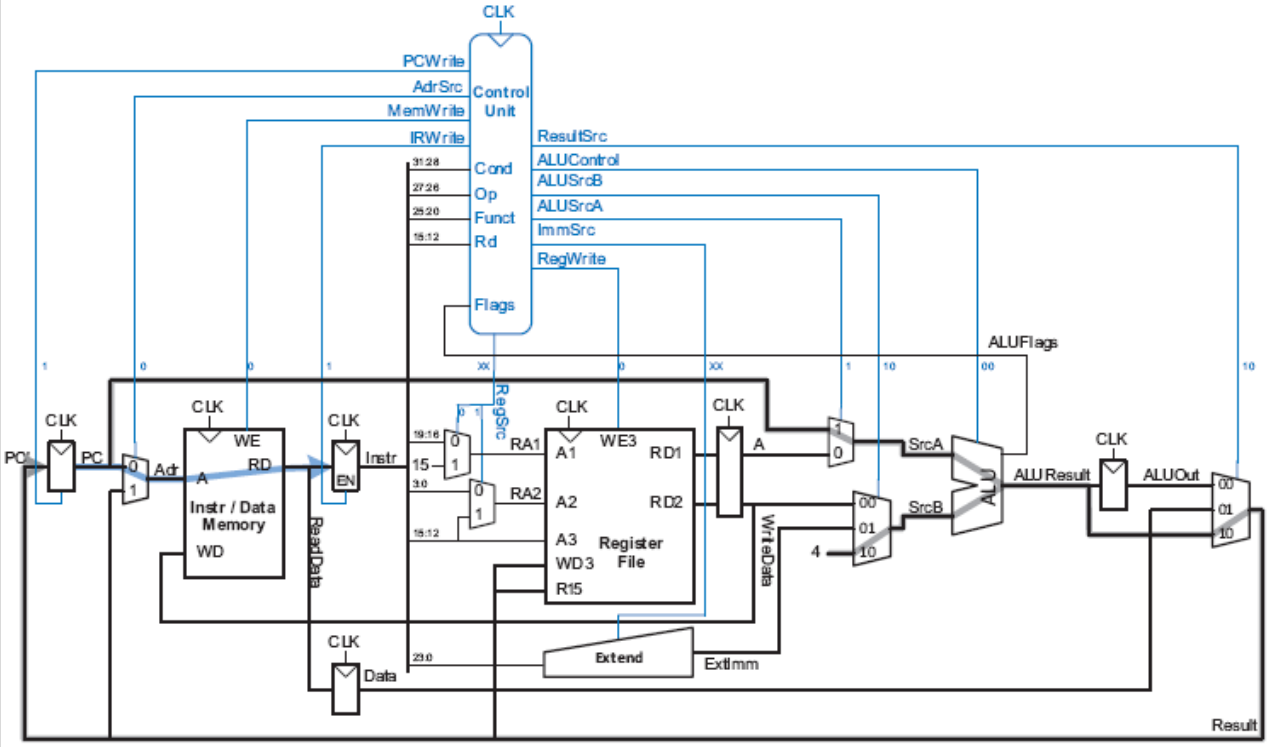
Per avere una visione più chiara di quali siano gli stati e di come vengano effettuate le transizioni da uno stato all'altro è utile considerare il **data flow** del processore.



In altri termini, data una istruzione osserviamo il comportamento del processore nei diversi cicli, in cui avvengono i quattro passi **fetch**, **decode**, **execute** e **store**.

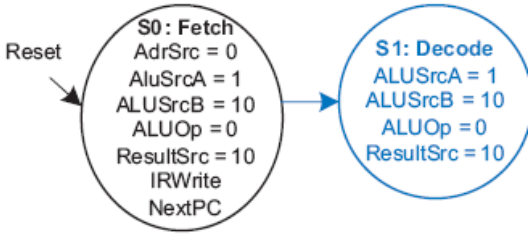
Il data flow di LDR

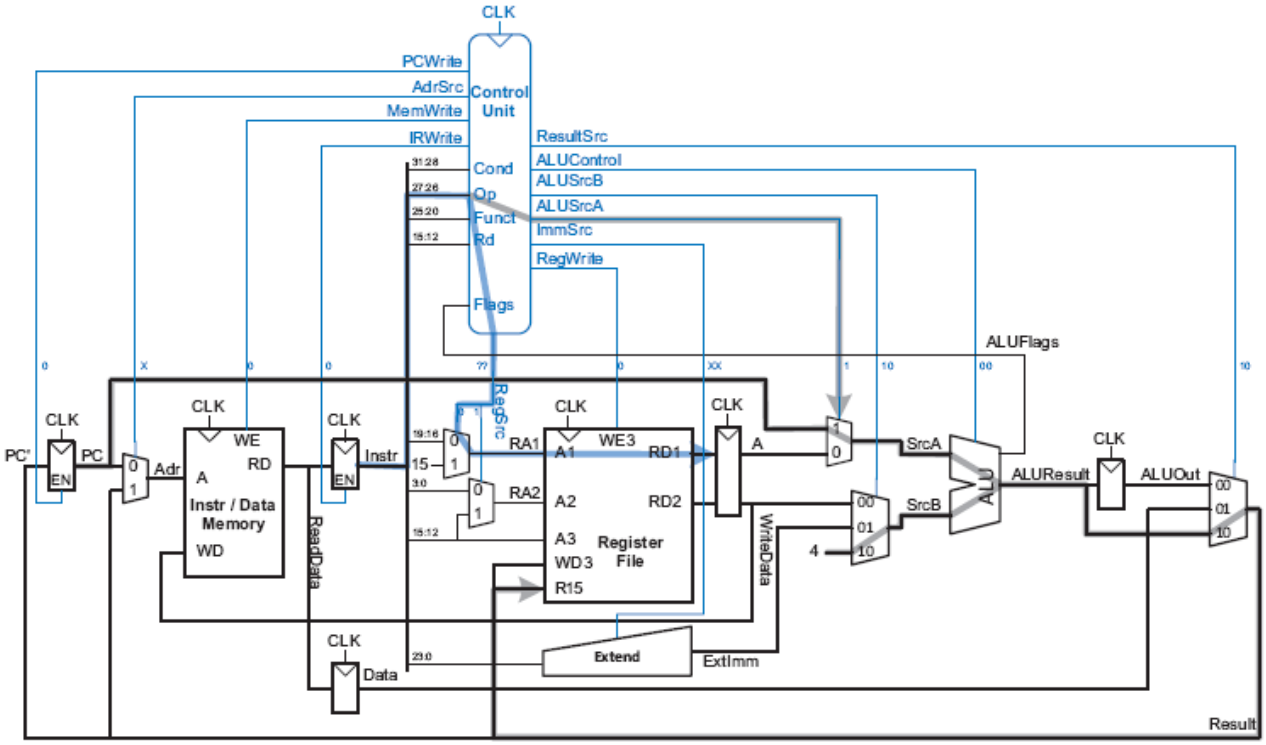
Operazione: **Fetch**



Il data flow di LDR

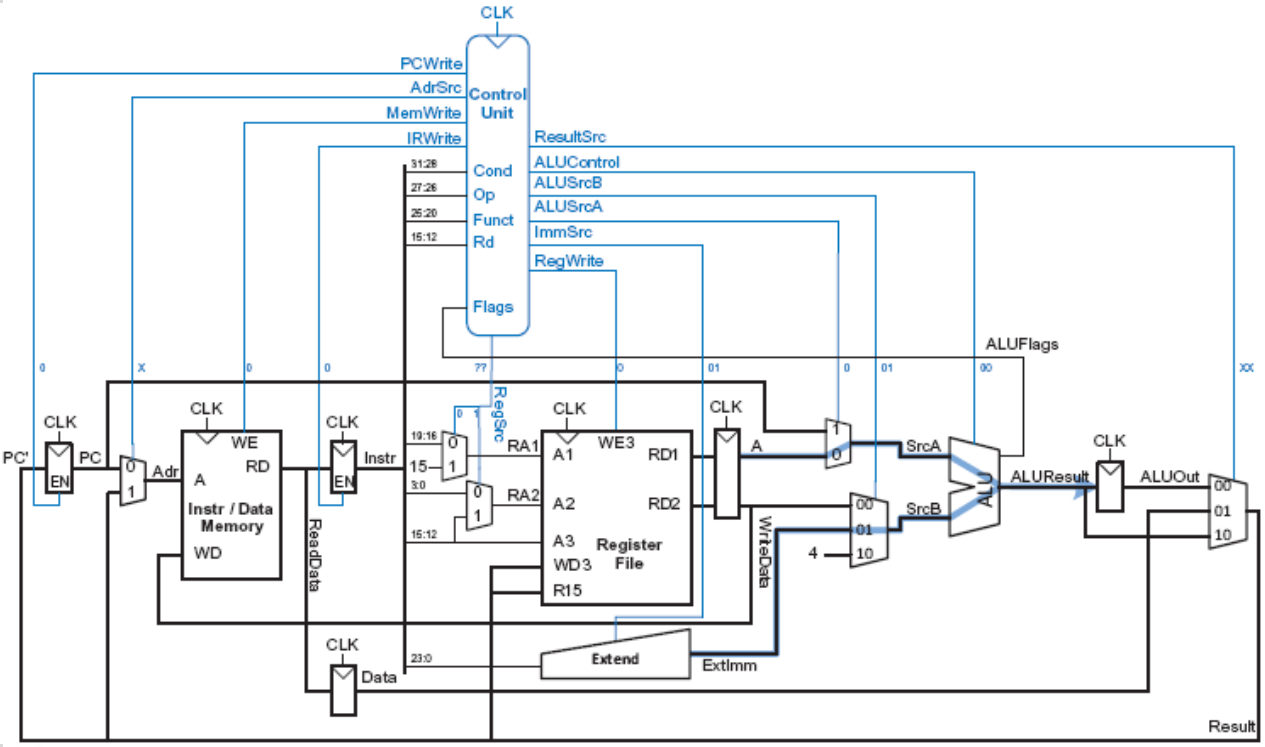
Operazione: **Decode**

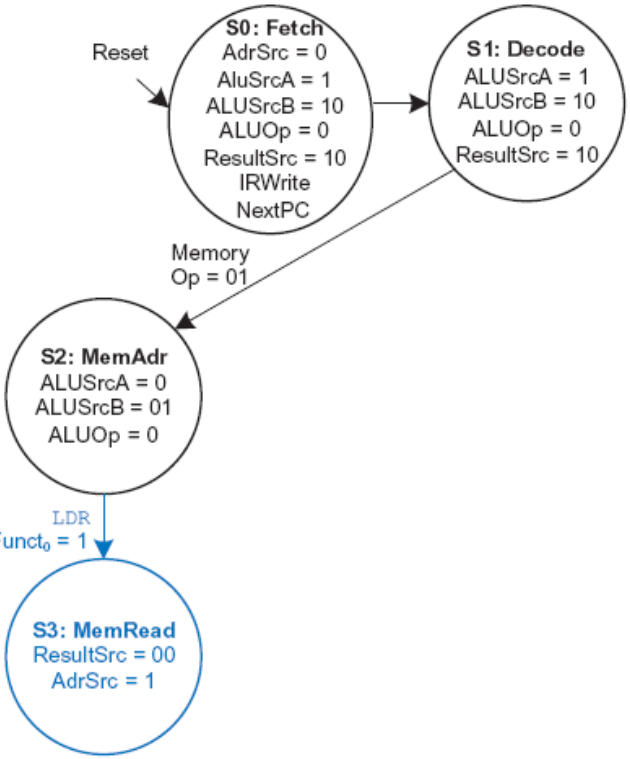




Il data flow di LDR

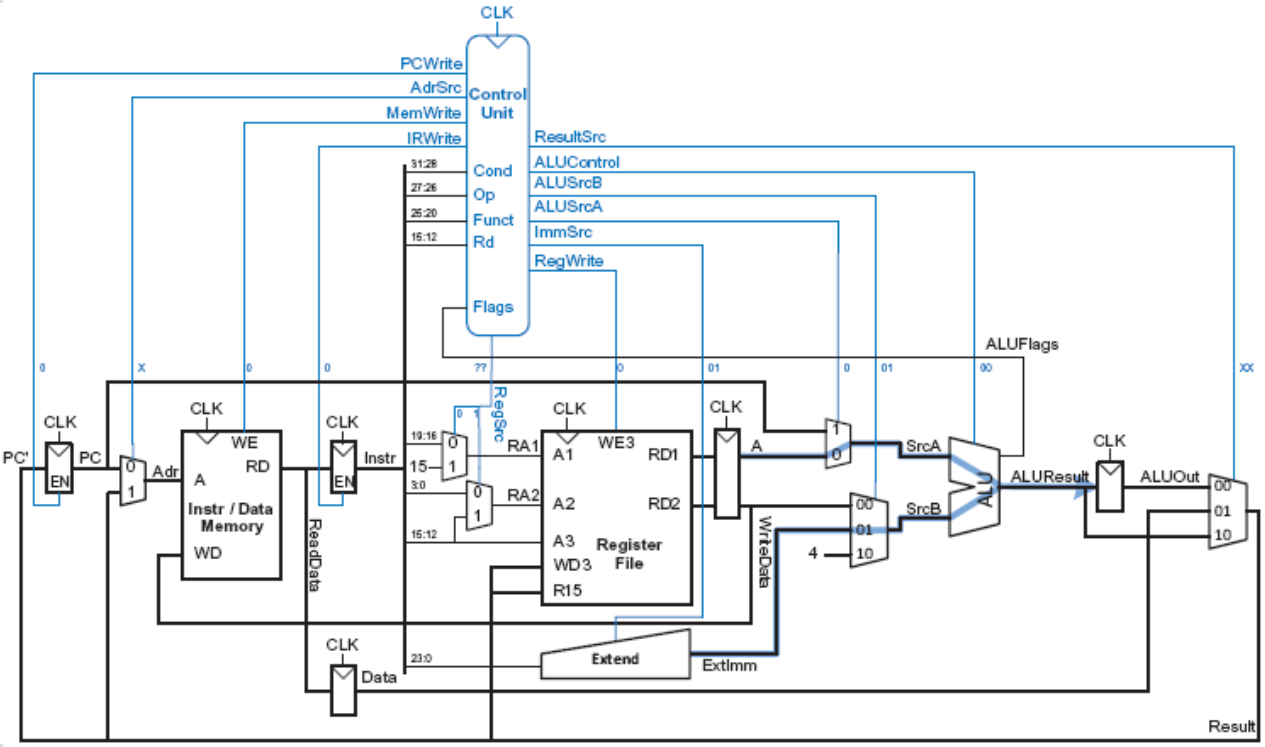
Operazione: **Execute** (memory address computation)

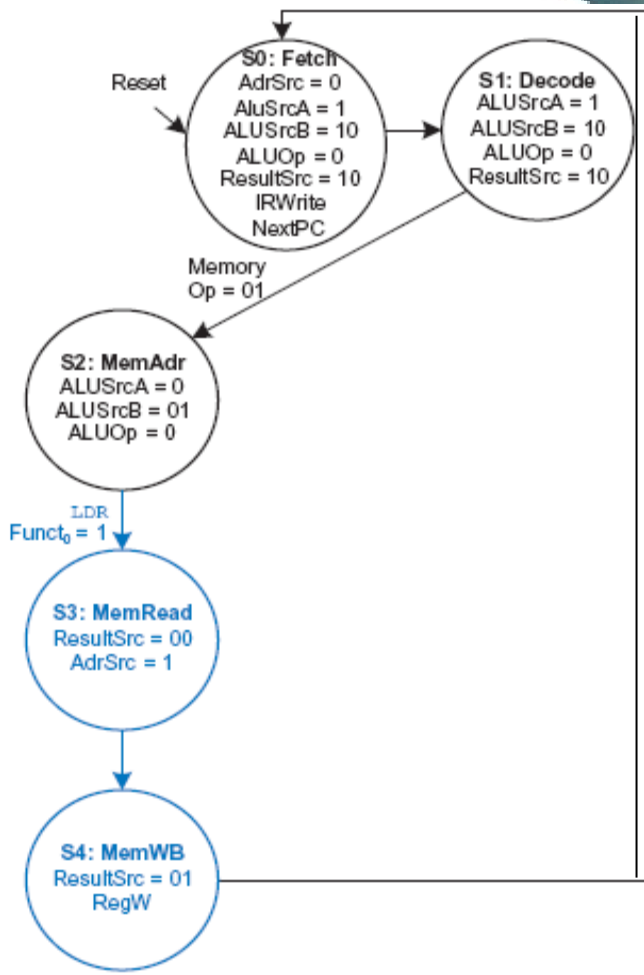




Il data flow di LDR

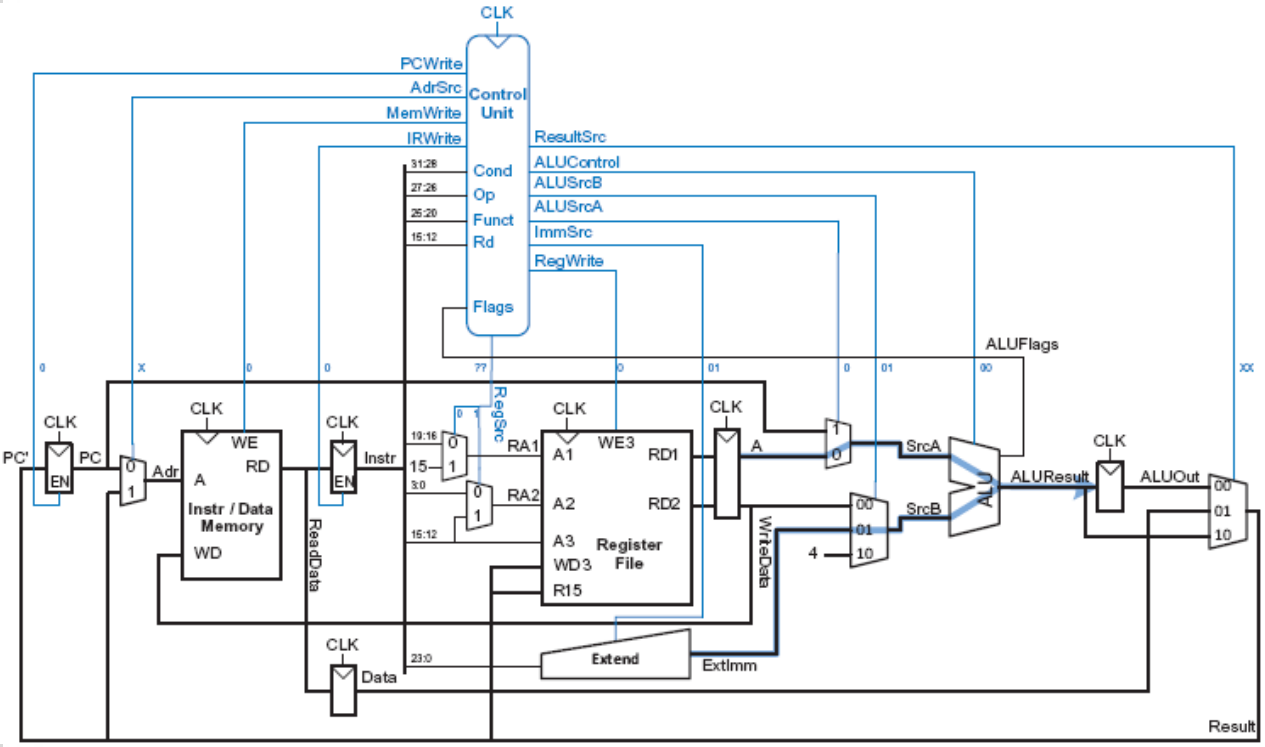
Operazione: **Execute** (memory read)

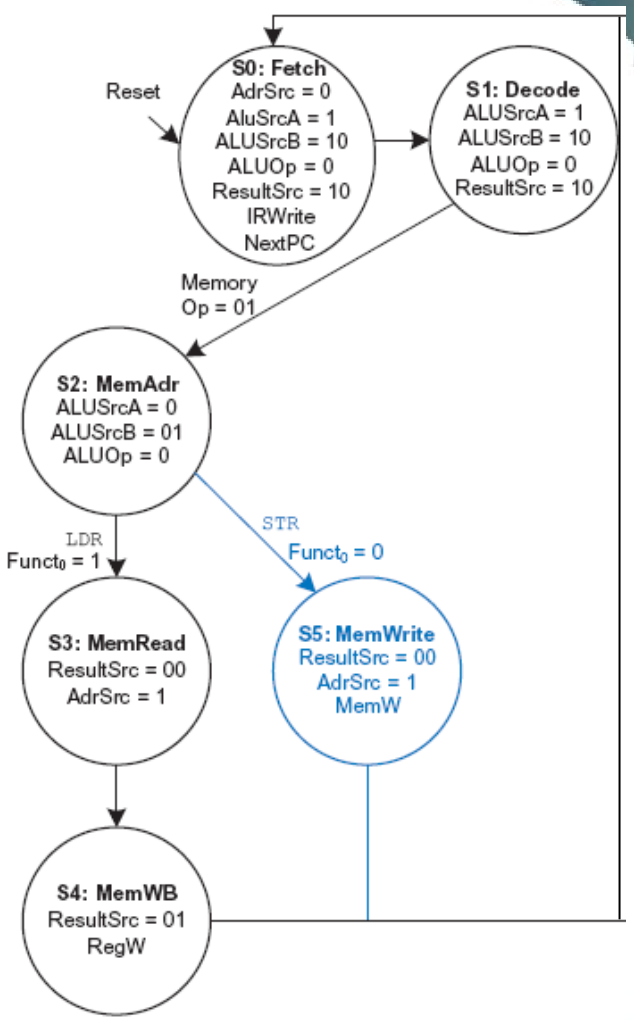




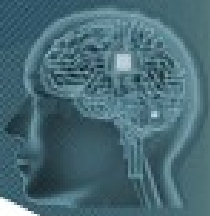
Il data flow di STR

Operazione: **Execute** (memory write)





Analisi delle prestazioni



In un processore a ciclo multiplo, il tempo impiegato per eseguire una istruzione dipende dal numero di cicli di clock, di cui necessita e dalla durata di un singolo ciclo di clock.

Il numero di cicli di clock necessario ad eseguire le diverse istruzioni è di:

- **branch** – **3** cicli;
- **data processing** – **4** cicli;
- **memory store** – **4** cicli;
- **memory load** – **5** cicli;

Valutiamo le prestazioni del processore a ciclo multiplo rispetto al benchmark SPECINT2000, che prevede:

- **LDR** – **25**%;
- **STR** – **10**%;
- **B** – **25**%;
- **Dtata Processing** – **25**%;

Analisi delle prestazioni

Il numero medio di cicli per istruzione è dato da:

CPI = (**perc. di B**)(**#cicli B**) + (**perc. di DP** + **perc. di STR**)(**#cili DP e STR**) + (**perc. di LDR**)(**#cili LDR**) =

= (**0.13**) (**3**) + (**0.52** + **0.10**) (**4**) + (**0.25**) (**5**) = **4.12**

I percorsi critici nel datapath, che richiedono maggior tempo e che quindi sono predominanti, sono due:

- Dal **PC**, attraverso il multiplexer **SrcA**, attraverso l'ALU, attraverso il multiplexer **Result**, attraverso la porta **R15**, fino al regitro **A**.
- Da **ALUOut**, attraverso il registro **Result**, attraverso il multiplexer **Adr**, attraverso la memoria (read), fino al registro **Data**.

- (t_{pcq_PC}) – caricamento di un nuovo indirizzo (PC) sul fronte di salita del clock;
- (t_{mux}) – selezione di un output da parte del multiplexer.
- (t_{ALU}) – l'ALU esegue su srcA e srcB una operazione.
- (t_{setup}) – viene impostato un segnale.

$$T_{c2} = t_{pcq\_PC} + 2t_{mux} + \max[t_{ALU} + t_{mux}, t_{mem}] + t_{setup};$$

Analisi delle prestazioni

Domanda: **qual è il tempo di esecuzione per un programma con 100 miliardi di istruzioni?**

Risposta:

secondo l'equazione

$T_{c2} = t_{pcq\_PC} + 2t_{mux} + \max[t_{ALU} + t_{mux}, t_{mem}] + t_{setup}$;

il tempo di ciclo del processore a ciclo multiplo è

Tc2 = 40 + 2(25) + 200 + 50 = 340 ps.

Secondo l'equazione

$$Tempo\ di\ esecuzione = (\#istruzioni)\left(\frac{cicli}{istruzione}\right)\left(\frac{secondi}{ciclo}\right)$$

il tempo di esecuzione totale è

T1 = (100 × 10^9 istruzioni) (4.12 cicli / istruzione) (340 × 10^{-12} s / ciclo) = 140 secondi.

Nota: **il tempo impiegato dal processore a ciclo singolo per lo stesso benchmark era di 84 sec.**

Table 7.5 Delay of circuit elements

Element	Parameter	Delay (ps)
Register clk-to-Q	t_{pcq}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	25
ALU	t_{ALU}	120
Decoder	t_{dec}	70
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

Alcune considerazioni finali

Una delle motivazioni alla base della progettazione di un processore a ciclo multiplo è stata quella di evitare che il ciclo durasse quanto quello necessario all'istruzione più lenta.

Questo esempio dimostra che il processore a ciclo multiplo è più lento di quello a ciclo singolo a causa delle latenze di propagazione.

Infatti, sebbene l'istruzione più lenta (LDR) sia stata suddivisa in cinque fasi, il tempo di ciclo del processore non è aumentato di cinque volte.

Questo in parte perché:

- non tutti i passaggi hanno esattamente la stessa lunghezza;
- i tempi di setup sono necessari ad ogni passo, e non solo la prima volta per l'intera istruzione.

In generale, ci si è resi conto che il fatto che alcuni calcoli siano più veloci di altri è difficile da sfruttare, a meno che le differenze sono grandi.