ARCHITETTURA DEGLI ELABORATORI

A.A. 2020-2021

Università di Napoli Federico II
Corso di Laurea in Informatica

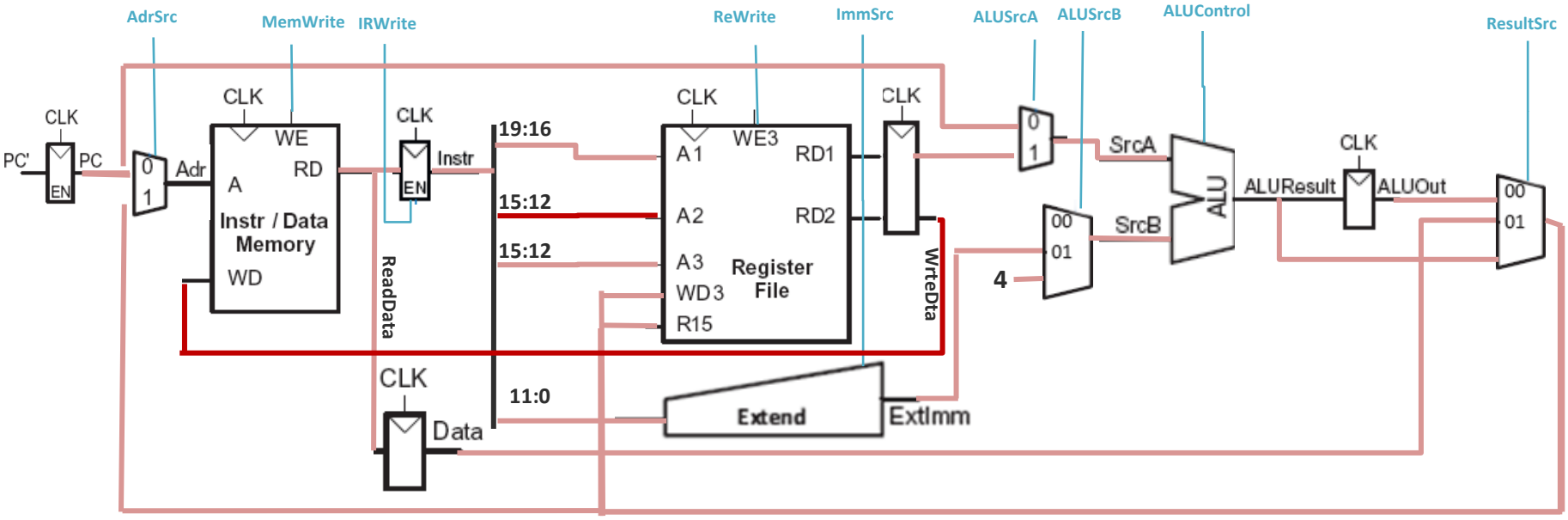
Docenti
Proff. Luigi Sauro gruppo 1 (A-G)
Silvia Rossi gruppo 2 (H-Z)

MICROARCHITETTURA ARM

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** da 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.

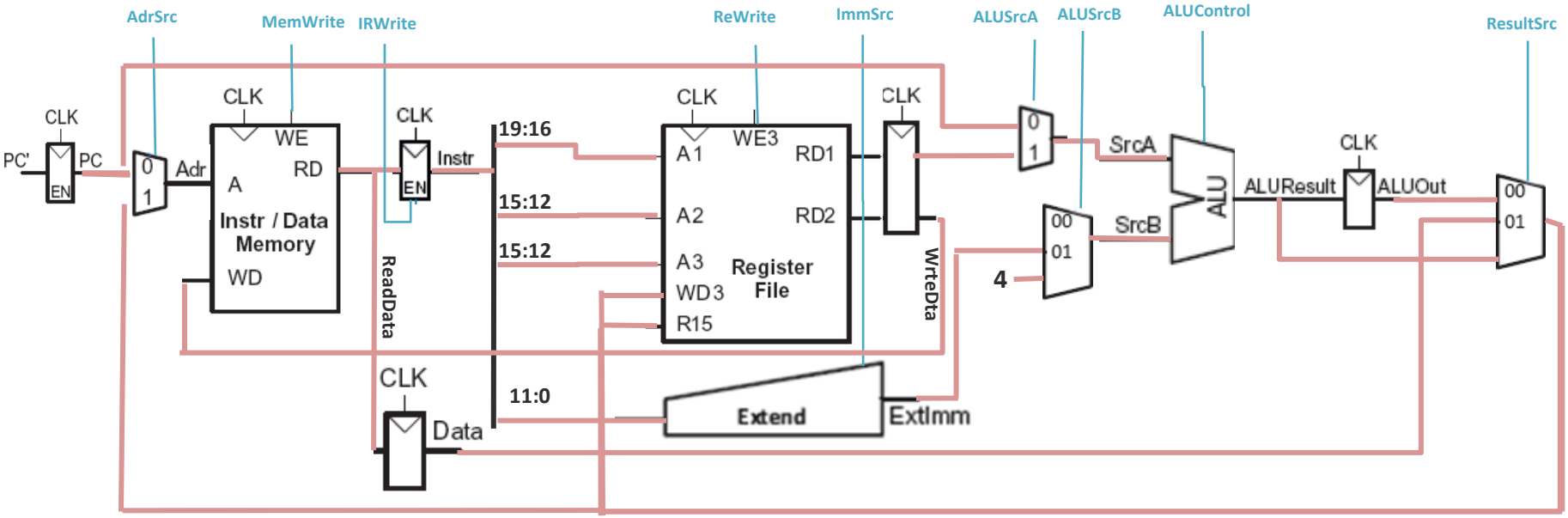


Write data in `Rn` to memory

Datapath DataProcessing

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.

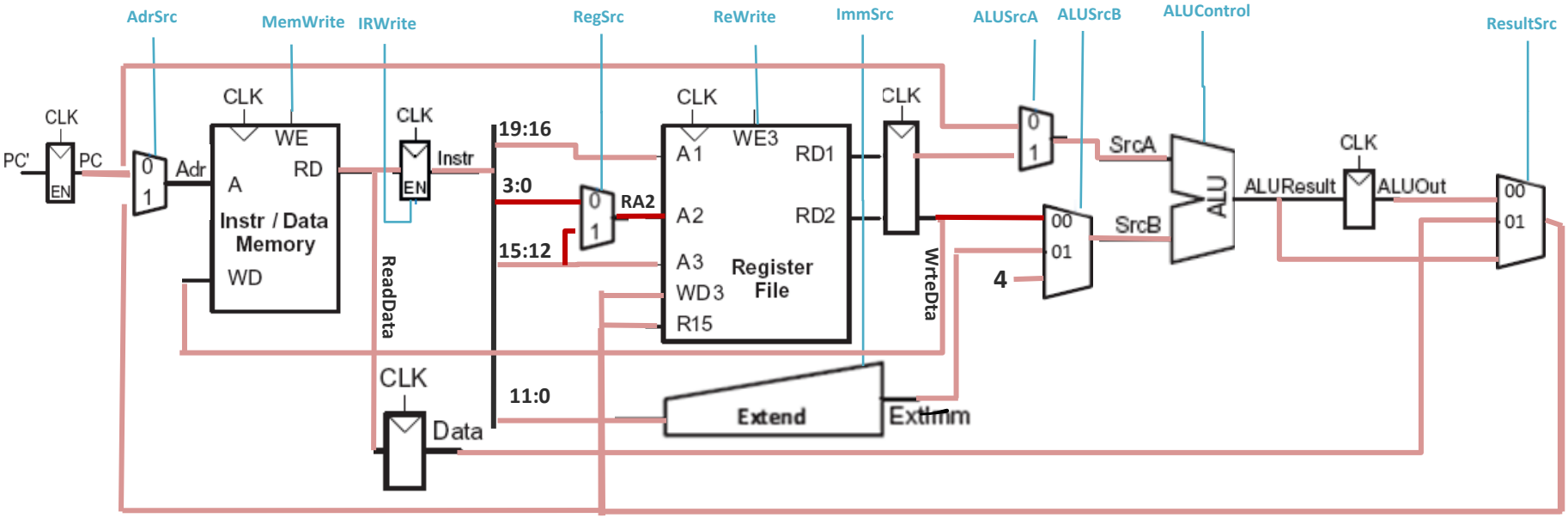
L'operazione da effettuare è specificata dal segnale **ALUControl**, mentre gli **ALUFlags** permettono di aggiornare il registro di stato.



Datapath DataProcessing

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.

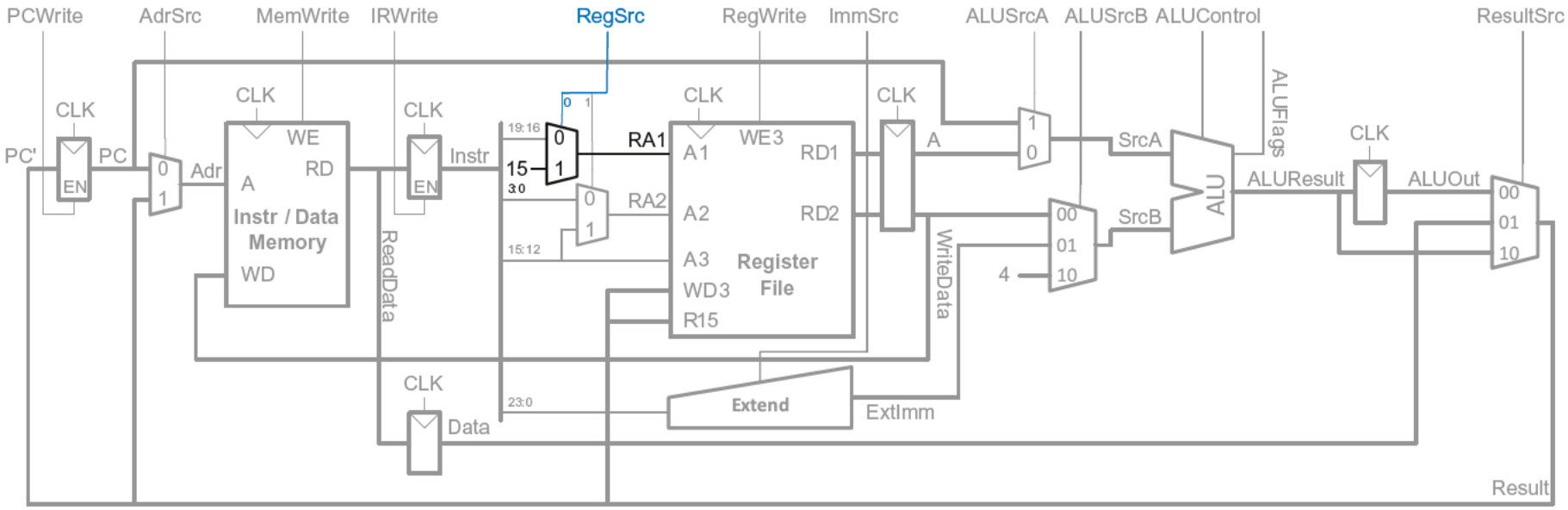


Multicycle Datapath: B

Calculate branch target address:

BTA = (*ExtImm*) + (PC+8)

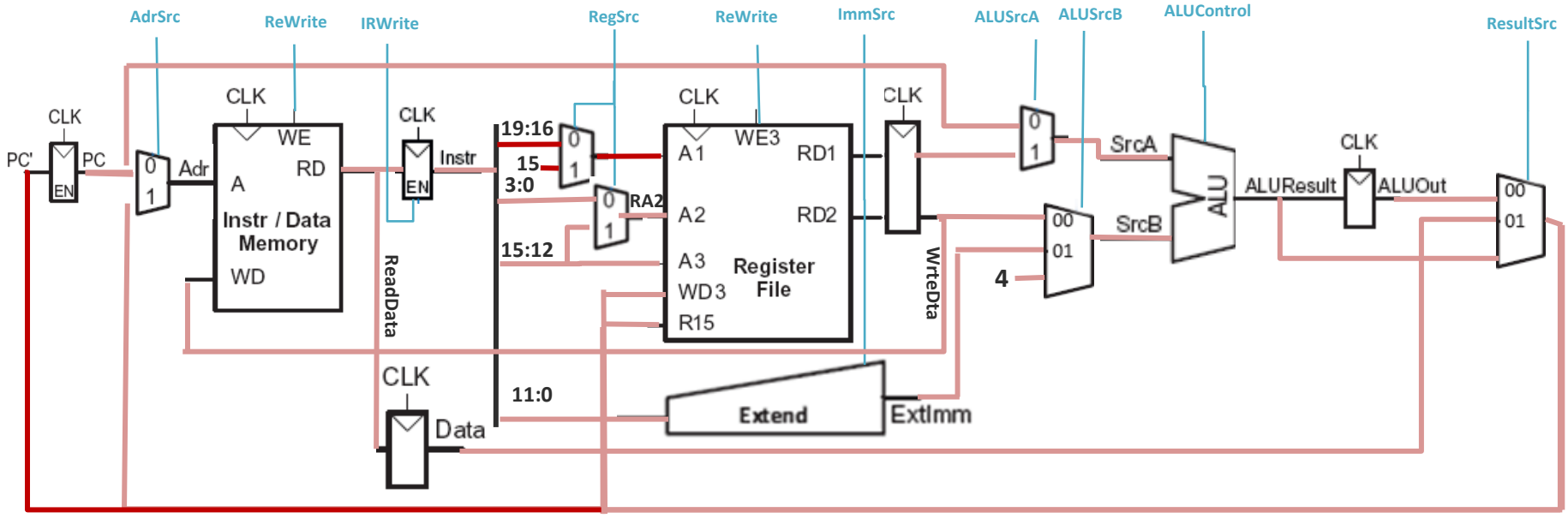
ExtImm = Imm24 << 2 and sign-extended



Datapath Branch

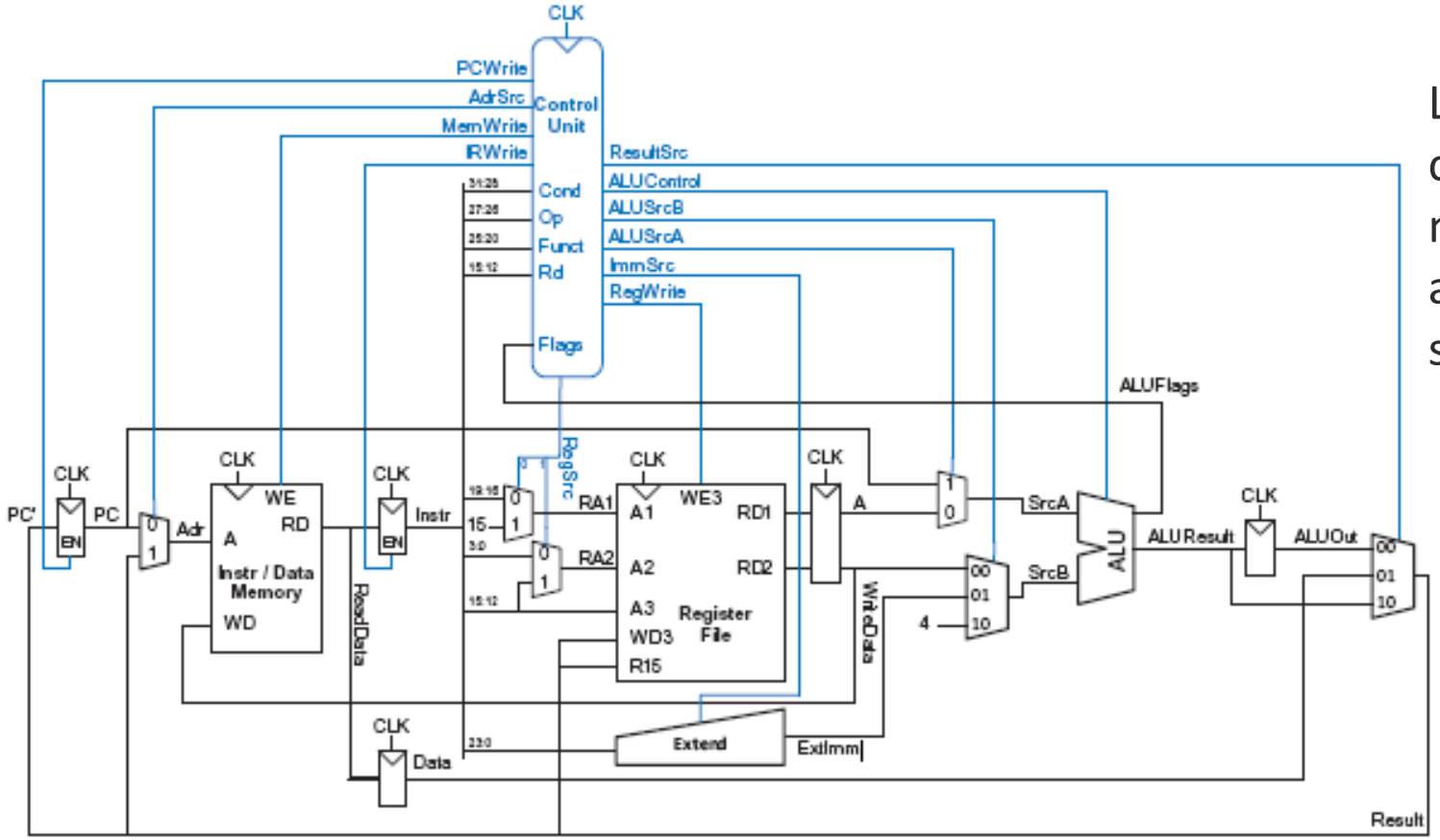
Per le istruzioni di branch, il datapath legge **PC+8** e una costante a **24** bit, che viene estesa a **32** bit. La somma di questi due valori è addizionata al **PC**.
Si ricorda, inoltre, che il registro **R15** contiene il valore **PC+8** e deve essere letto per tornare da un salto. È sufficiente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

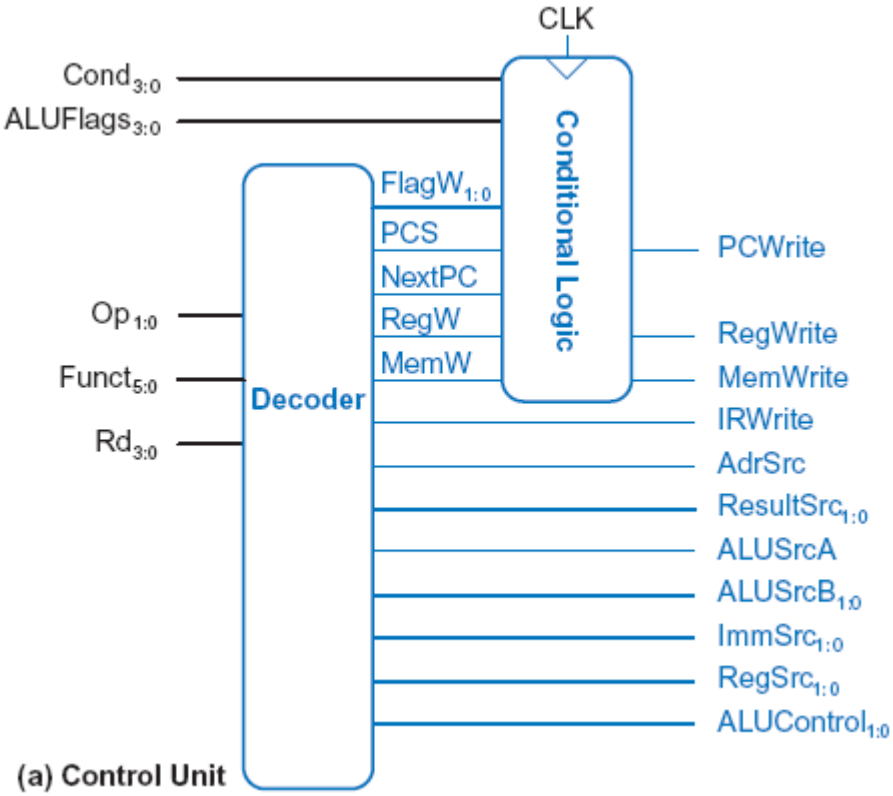


L'unità di controllo memorizza e aggiorna i flag di stato.

Unità di controllo

Come nel processore a ciclo singolo, l'unità di controllo è suddivisa in decodificatore e logica condizionale. Il decodificatore è progettato come una FSM, 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.



Dataflow 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).

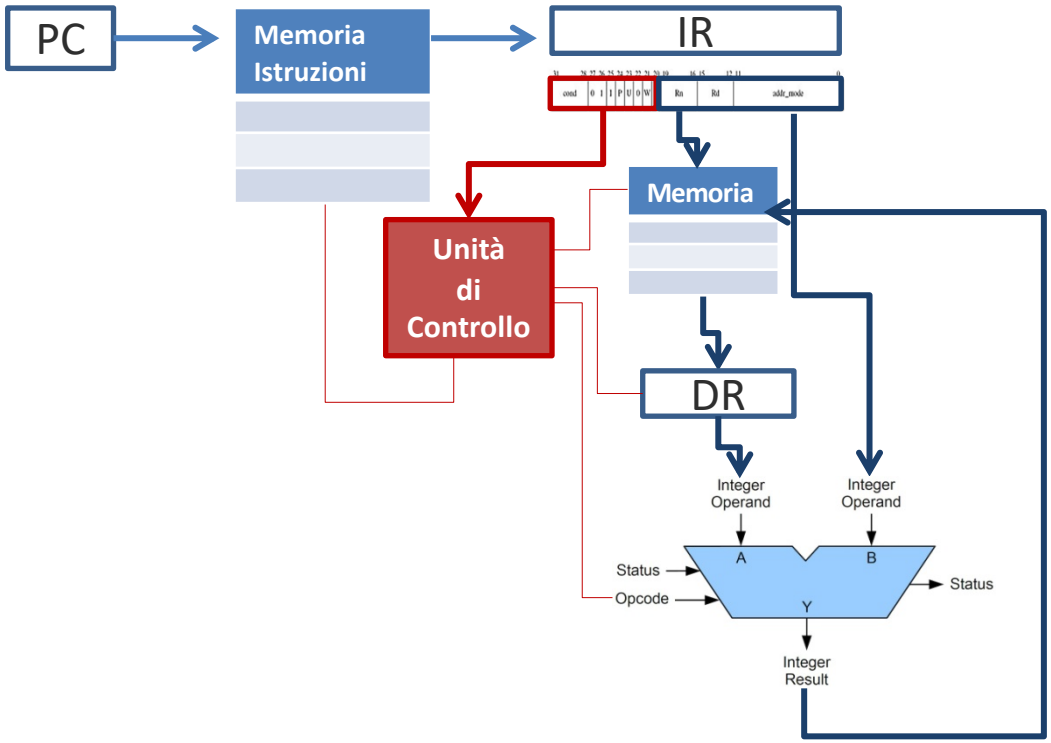
Uno stato dell'automa che implementa il **main decoder** non è altro che lo stato dei segnali in un determinato momento.

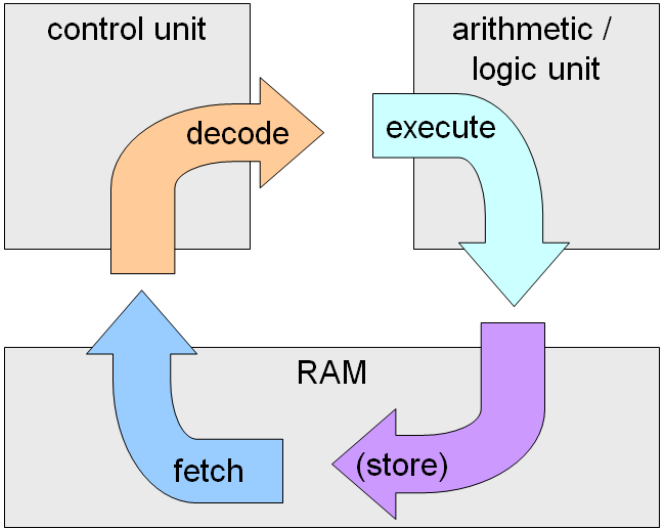
Per avere una visione più chiara di quali siano gli stati e di come vengano effettuate le transizioni da uno stato all'altro è utile considerare il **data flow** del processore.

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**.

Le microarchitetture

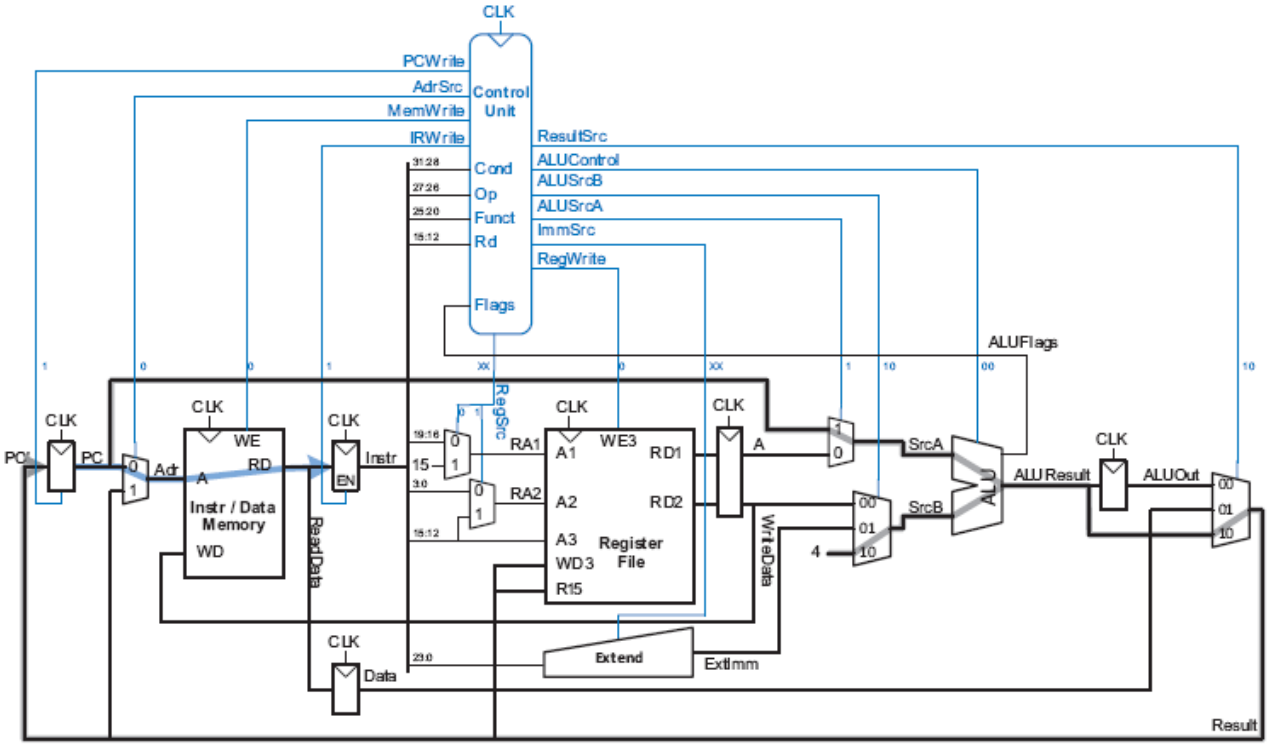
Una istruzione ha un ciclo di vita che consta di quattro fasi principali





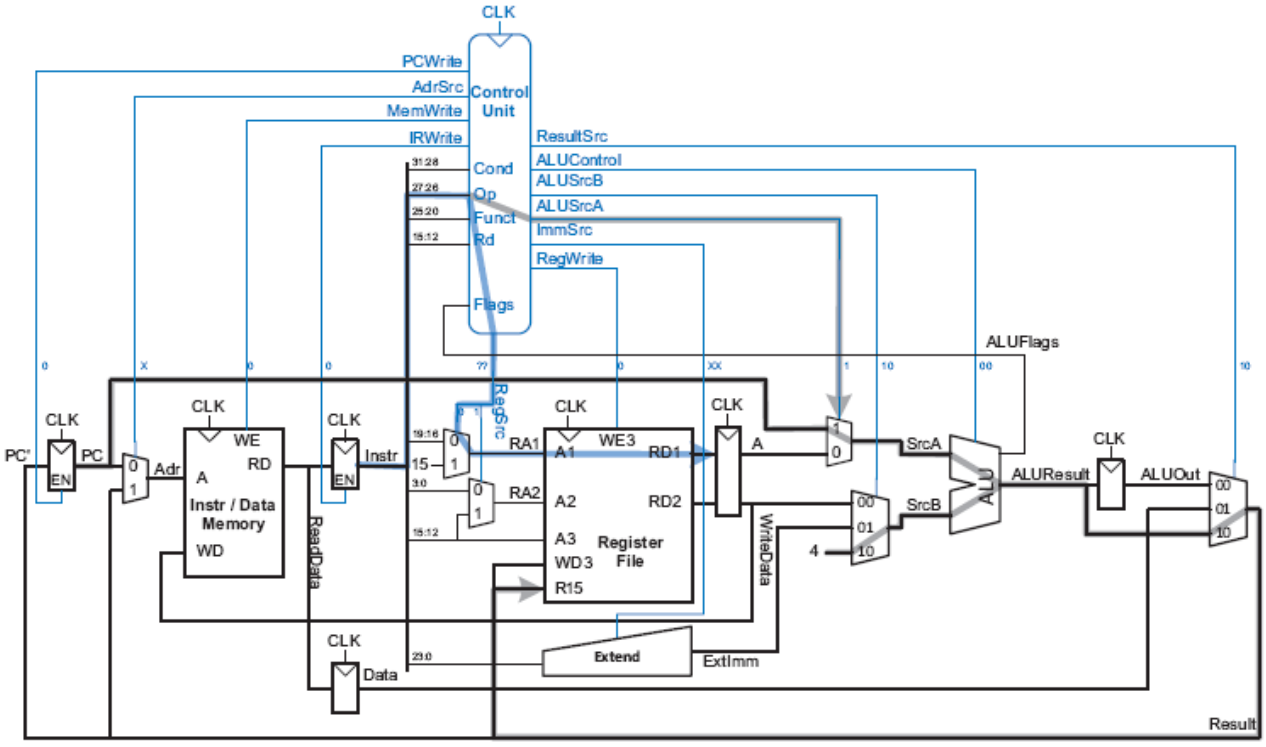
Dataflow di LDR

Operazione: **Fetch**



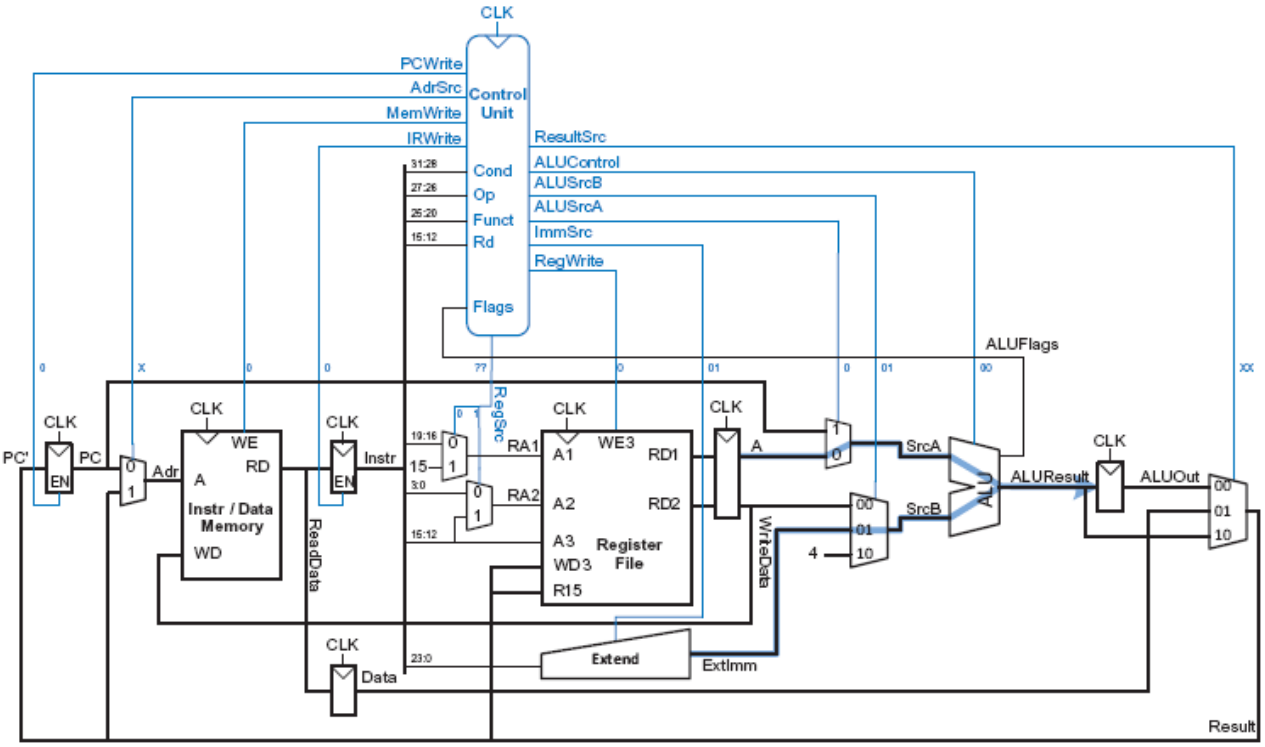
Dataflow di LDR

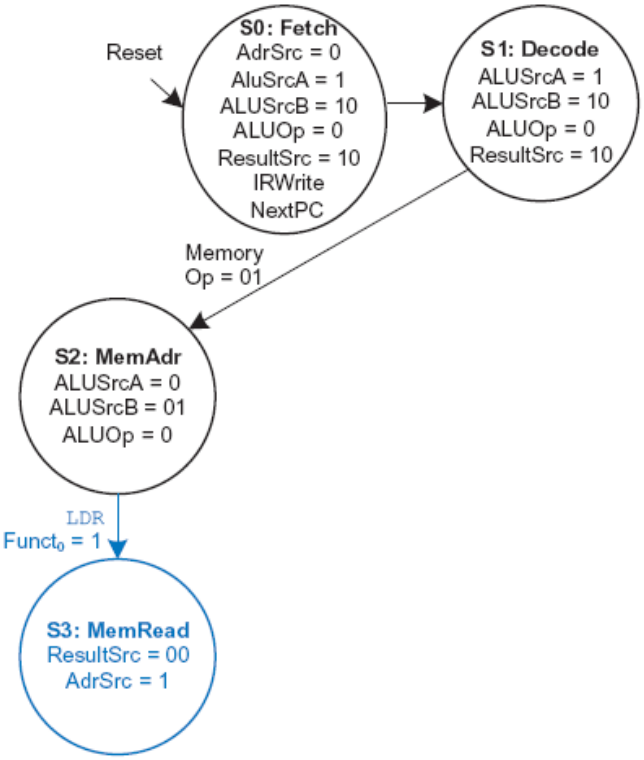
Operazione: **Decode**



Dataflow di LDR

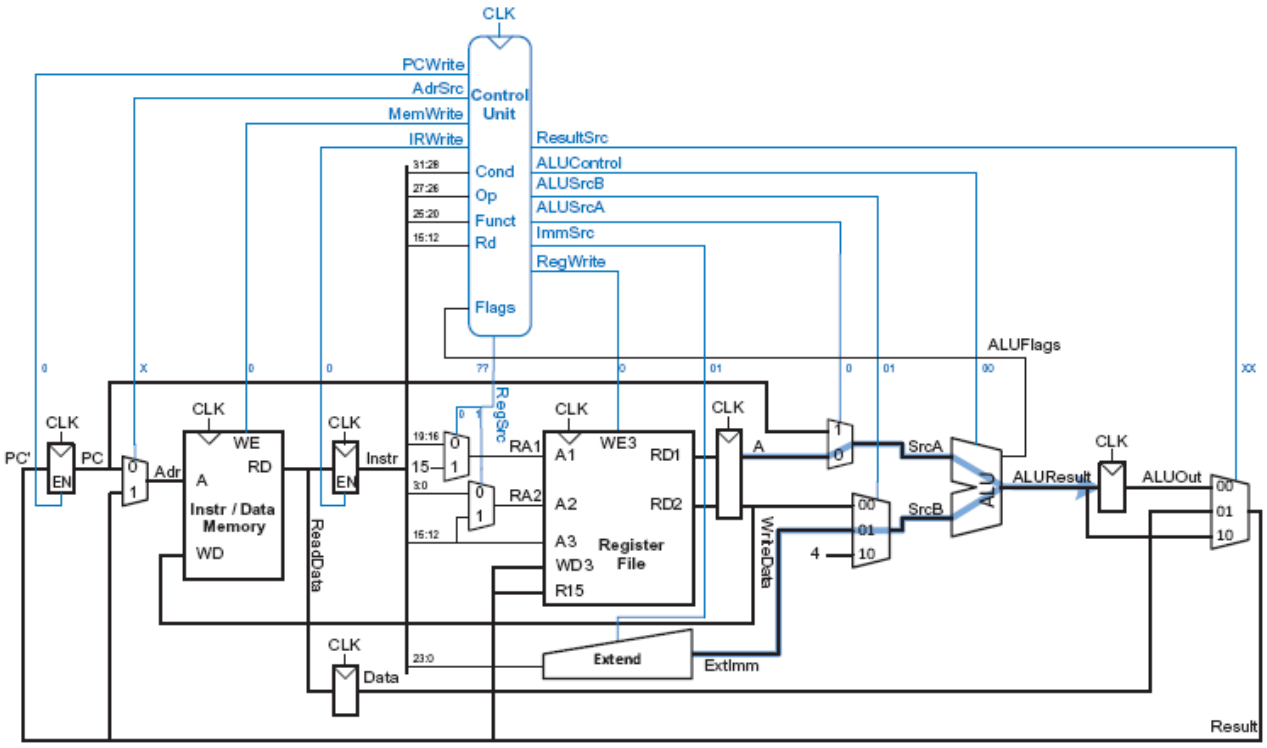
Operazione: Execute (memory address computation)

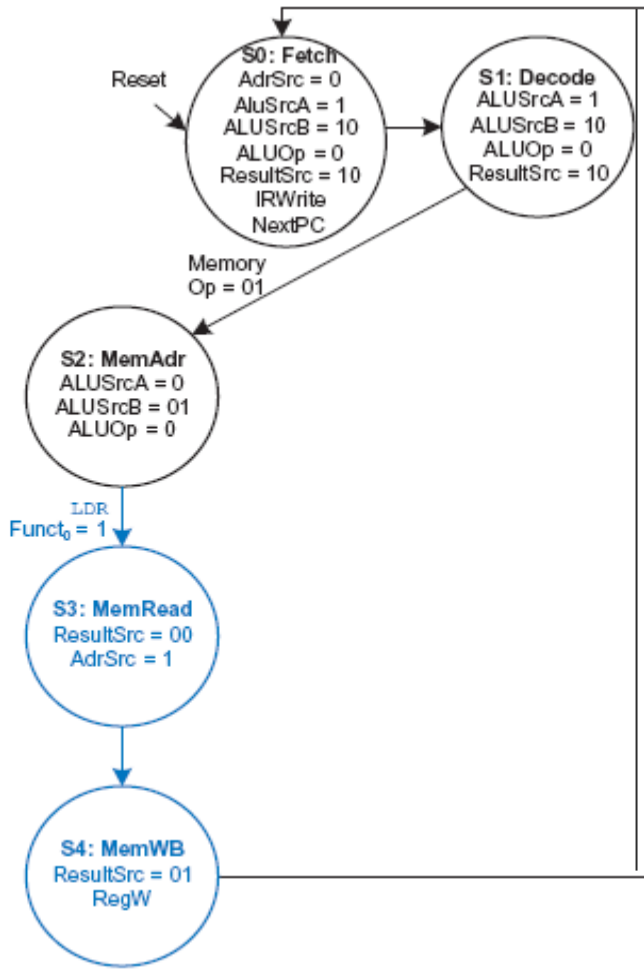




Dataflow di LDR

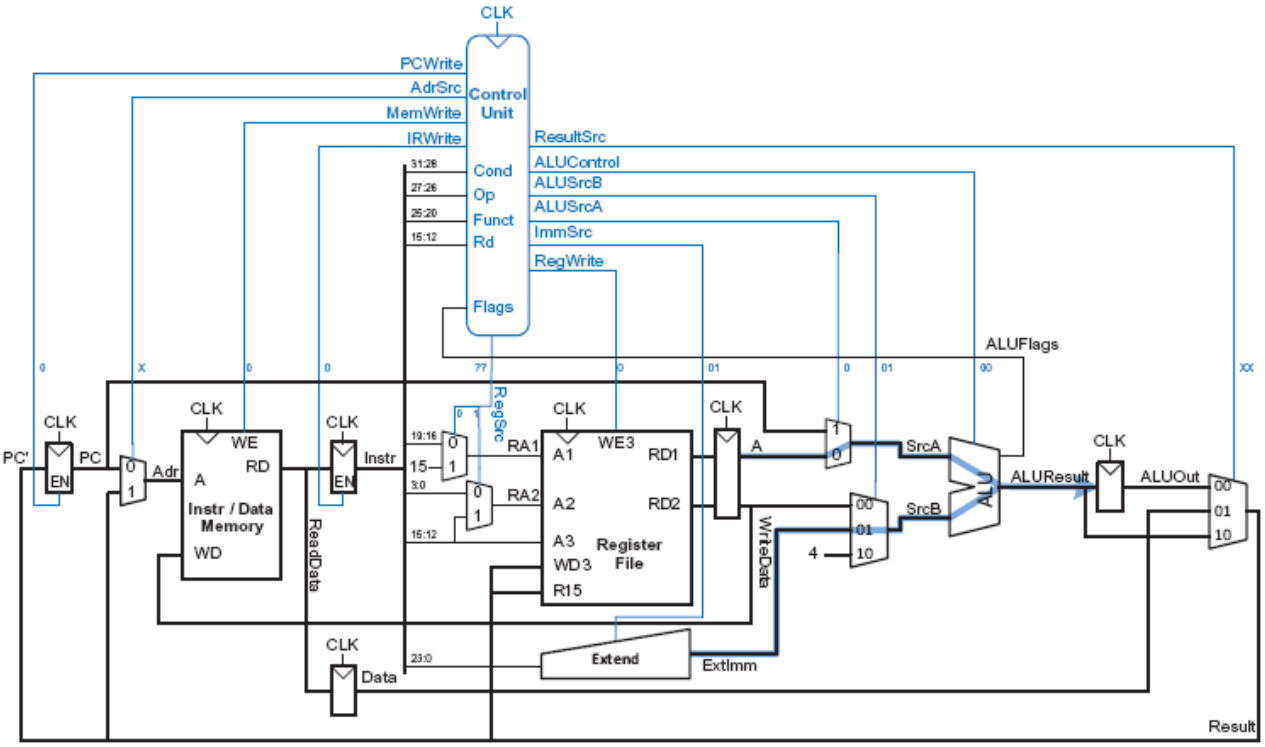
Operazione: Store (memory read)

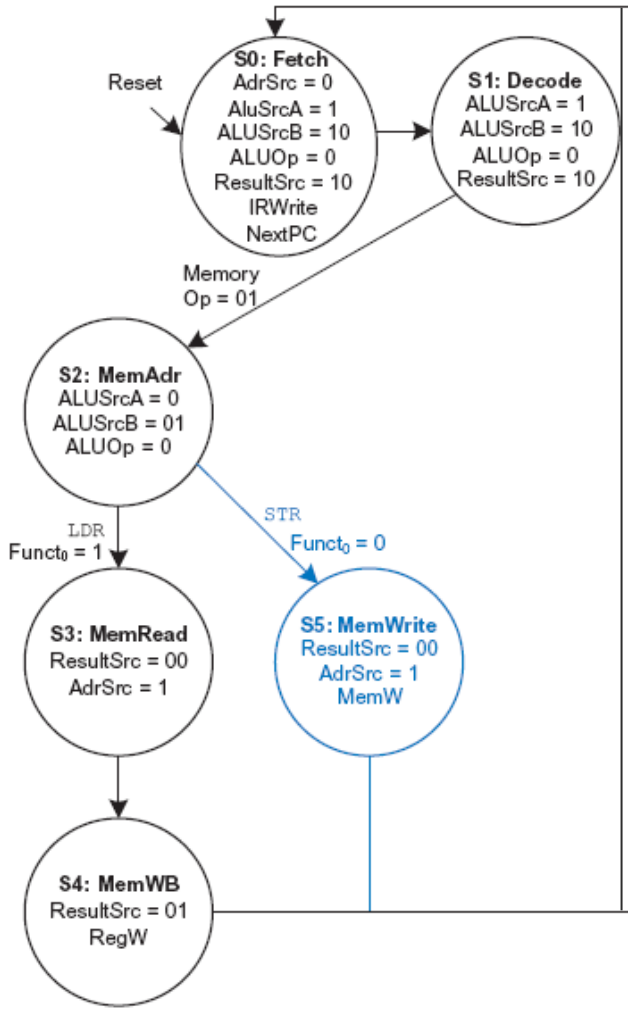




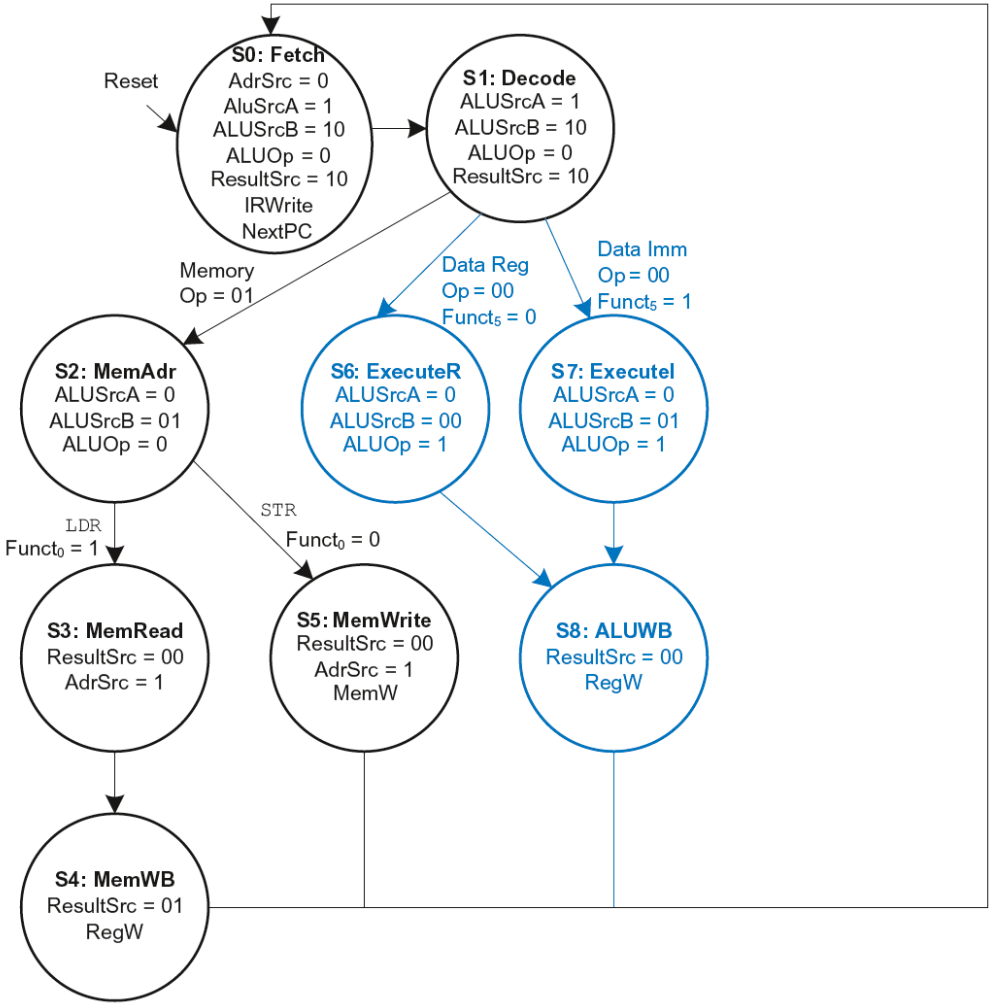
Dataflow di STR

Operazione: Execute (memory write)



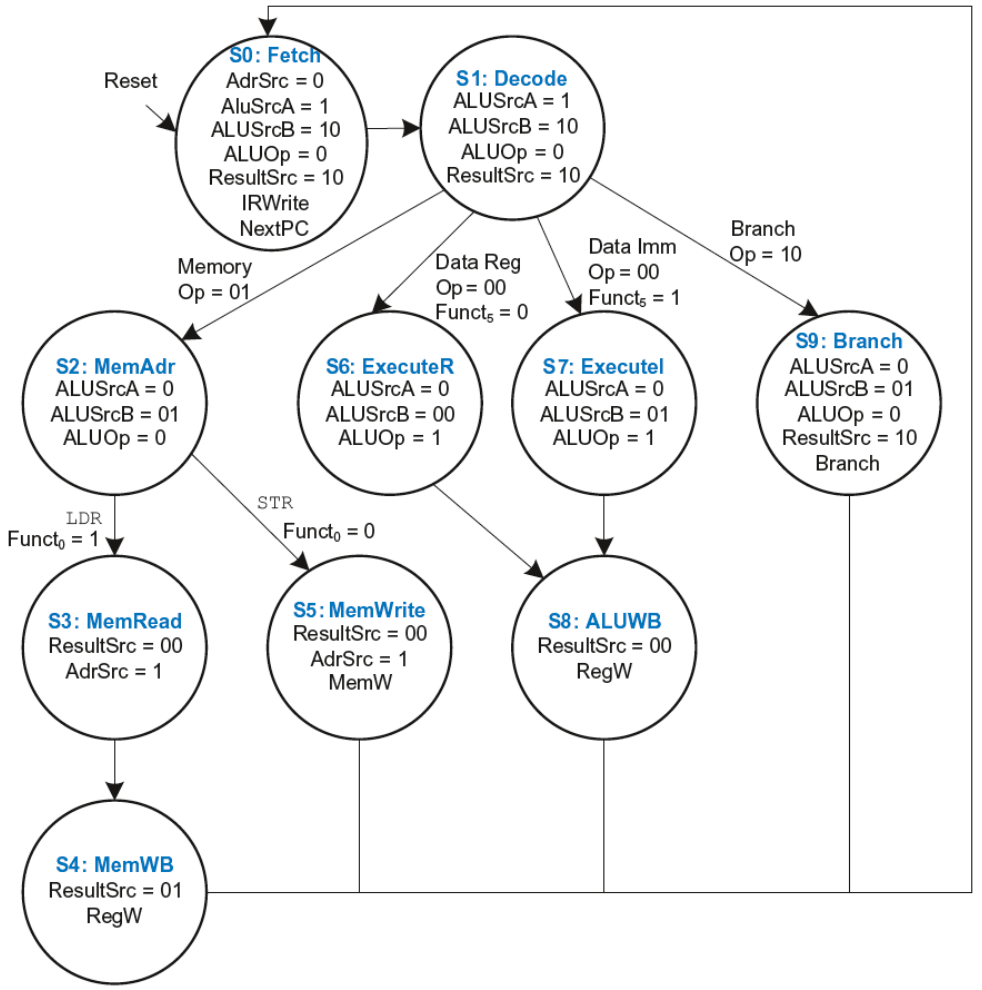


Main Controller FSM: Data-processing



Multicycle Controller FSM

State	Datapath μOp
Fetch	Instr ← Mem[PC]; PC ← PC+4
Decode	ALUOut ← PC +4
MemAdr	ALUOut ← Rn + Imm
MemRead	Data ← Mem[ALUOut]
MemWB	Rd ← Data
MemWrite	Mem[ALUOut] ← Rd
ExecuteR	ALUOut ← Rn op Rm
ExecuteI	ALUOut ← Rn op Imm
ALUWB	Rd ← ALUOut
Branch	PC ← R15 + offset



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** – **13**%;
- **Data Processing** – **52**%;

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 registro **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

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, la frequenza di ciclo del processore non è aumentata 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.

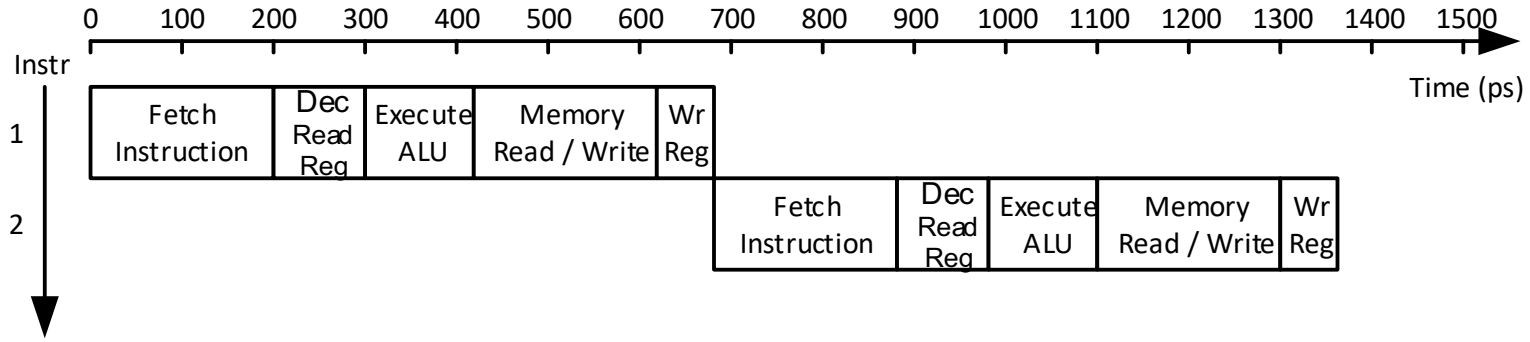
In generale, ci si è resi conto che il fatto che alcuni calcoli siano più veloci di altri è difficile da sfruttare, a meno che le differenze sono grandi.

Pipelined ARM Processor

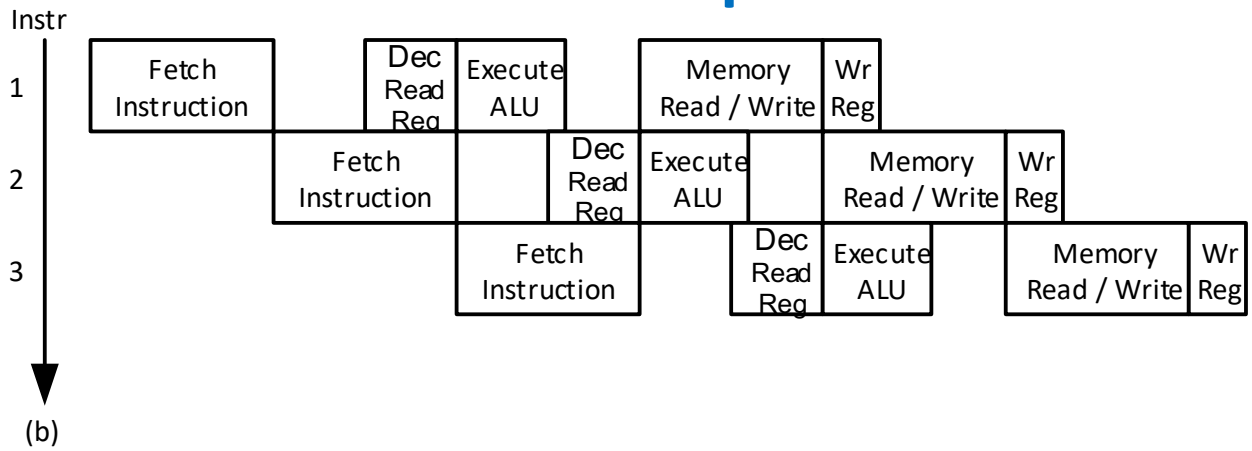
- Temporal parallelism
- Divide single-cycle processor into 5 stages:
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- Add pipeline registers between stages

Single-Cycle vs. Pipelined

Single-Cycle



Pipelined



3. Si riportino i valori dei flag V C N Z risultanti dalla somma delle parole a 32 bit #800348CC e #8CFFFFFA:

   V C N Z: _____

4. Il seguente codice assembly rappresenta la funzione fattoriale, ma ci sono due errori. Scrivere nell'apposito riquadro il codice corretto. [Si ricordi che SP sta per stack pointer, LR per link register e PC per program counter.]

```
FACTORIAL
  PUSH R0, LR
  CMP R0, #1
  BEQ ELSE
  MOV R0, #1
  ADD SP, SP, #8
  MOV PC, LR
ELSE
  SUB R0, R0, #1
  B FACTORIAL
  POP R1, LR
  MUL R0, R1, R0
  MOV PC, LR
```