

Calcolatori Elettronici B

a.a. 2006/2007

**Tecniche di Controllo:
Esercizi**

Massimiliano Giacomini

Due tipologie di esercizi

- Calcolo delle prestazioni nei sistemi a singolo ciclo e multiciclo (e confronto)
- Implementazione di istruzioni nuove

... cominciamo dalla prima...

Formula fondamentale prestazioni (in ogni caso)

$$T_{\text{esecuzione}} = \# \text{istruzioni} * \text{CPI} * T_{\text{clock}}$$

Calcolo CPI e Prestazioni nei sistemi a singolo ciclo e multiciclo

1) Calcolo prestazioni nei sistemi a singolo ciclo

$$\text{CPI} = 1$$

$$T_{\text{clock}} = \max\{T_a + \dots T_k\}$$

ovvero la serie più lenta di “operazioni atomiche” [cammino critico]

$$T_{\text{esecuzione}} = \#\text{istruzioni} * \text{CPI} * T_{\text{clock}} = \#\text{istruzioni} * T_{\text{clock}}$$

2) Calcolo CPI e prestazioni nei sistemi multi-ciclo

Dato un certo carico di lavoro con frequenze relative delle istruzioni f_1, \dots, f_n

$$\text{CPI} = f_1 * \text{CPI}_1 + f_2 * \text{CPI}_2 + \dots + f_n * \text{CPI}_n$$

$$T_{\text{clock}} = \max\{T_1, \dots, T_m\} \quad [\text{operazioni atomiche eseguite in un ciclo di clock}]$$

$$T_{\text{esecuzione}} = \#\text{istruzioni} * \text{CPI} * T_{\text{clock}}$$

3) Confronto di prestazioni tra sistemi diversi [su un carico/prog. determinato]

$$\frac{T_{\text{esecuzione}}^1}{T_{\text{esecuzione}}^2} = \frac{\text{CPI}^1 * T_{\text{clock}}^1}{\text{CPI}^2 * T_{\text{clock}}^2}$$

Esercizio

Si ipotizzi un **carico di lavoro** che preveda:

lw:	31%	}	Eseguite secondo le fasi viste
sw:	21%		
Tipo-R:	22%		
beq:	5%		
j:	7%	}	Come Tipo-R ma ALU richiede 4 operazioni in sequenza anziché 1
somma virgola mob:	7%		
multipl. Virgola mob:	7%	}	Come Tipo-R ma 8 operazioni ALU

Si supponga che le **operazioni atomiche** che coinvolgono le unità funzionali principali (di cui si tiene conto per il calcolo dei tempi) richiedano:

Unità di memoria (lettura e scrittura):	2 ns
Register File (lettura e scrittura):	1 ns
Operazione-ALU:	2 ns

- Confrontare le prestazioni di una implementazione a ciclo singolo vs. multi-ciclo (vincolo di non mettere in serie due operazioni atomiche) vs. multi-ciclo in cui sono effettuate in serie (nello stesso ciclo):

- operazione ALU per produrre un valore + scrittura del valore nel Register File
- lettura dalla memoria di un valore + " " " " " "

Soluzione

Ricordando le fasi delle istruzioni si hanno le seguenti operazioni atomiche:

Istruzione	Fetch	Lettura Registri/ Decode	ALU	Accesso Memoria	Scrittura Registri
lw	1	1	1	1	1
sw	1	1	1	1	
Tipo-R	1	1	1		1
beq	1	1	1		
j	1	“1”			“1”
ADD V. MOB.	1	1	4		1
MUL V. MOB.	1	1	8		1

M1 (Macchina a singolo ciclo): tutte le operazioni in serie nello stesso ciclo di clock

Istruzione	Fetch	Lettura Registri/ Decode	ALU	Accesso Memoria	Scrittura Registri	Tempo di esecuz.
lw	1	1	1	1	1	8
sw	1	1	1	1		7
Tipo-R	1	1	1		1	6
beq	1	1	1			5
j	1	1			1	2
ADD V. MOB.	1	1	4		1	12
MUL V. MOB.	1	1	8		1	20
Tempo per operaz. atomica	2	1	2	2	1	

Operazione più lunga determina $T_{\text{clock}} = 20 \text{ ns}$



$$\text{CPI} = 1$$

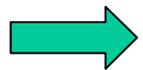
$$T_{\text{clock}} = 20 \text{ ns}$$

$$T_{M1} = I * 20 \text{ ns} \quad (\text{I è il numero di istruzioni})$$

M2 (Macchina multi-ciclo): in un ciclo di clock NO serie di operazioni atomiche

$$T_{\text{clock}} = 2 \text{ ns} \quad (\text{operazione ALU oppure accesso in memoria})$$

$$\begin{aligned} \text{CPI} &= \text{somma pesata dei CPI singole istruzioni con le frequenze relative} \\ &= (\text{vedi lucido seguente}) 4,89 \end{aligned}$$



$$T_{M2} = I * 4.89 * 2 \text{ ns} = I * 9.78 \text{ ns}$$

M2 (Macchina multiciclo): CPI per istruzione

Istruzione	Fetch	Lettura Registri/ Decode	ALU	Accesso Memoria	Scrittura Registri	CPI
lw	1	1	1	1	1	5
sw	1	1	1	1		4
Tipo-R	1	1	1		1	4
beq	1	1	1			3
j	1	1			1	3
ADD V. MOB.	1	1	4		1	7
MUL V. MOB.	1	1	8		1	11

$$\begin{aligned} \text{CPI} &= 0.31*5 + 0.21*4 + 0.22*4 + 0.05*3 + 0.07*3 + 0.07*7 + 0.07*11 \\ &= 4.89 \end{aligned}$$

M3: in un ciclo di clock ho al più:

lettura memoria | ALU [2 ns] + scrittura Reg.File [1 ns]

 $T_{\text{clock}} = 3\text{ns}$

CPI = per TIPO-R, lw e virgola-mobile CPI si riduce di 1

Istruzione	Fetch	Lettura Registri/ Decode	ALU [ev. Scr. Reg.]	Accesso Memoria [ev. Scr. Reg.]	CPI
lw	1	1	1	1	4
sw	1	1	1	1	4
Tipo-R	1	1	1		3
beq	1	1	1		3
j	1	1	1		3
ADD V. MOB.	1	1	4		6
MUL V. MOB.	1	1	8		10

$$\text{CPI} = 0.31*4 + 0.21*4 + 0.22*3 + 0.05*3 + 0.07*3 + 0.07*6 + 0.07*10 = 4.22$$

 $T_{M3} = I * 4.22 * 3 \text{ ns} = I * 12.66 \text{ ns}$

Riepilogo

	CPI	T_{clock}	T_{esecuz}	$T_{\text{istruzione-più-costosa}}$
M1	1	20 ns	$I * 20$ ns	20 ns
M2	4.89	2 ns	$I * 9.78$ ns	$11 * 2 = 22$ ns
M3	4.22	3 ns	$I * 12.66$ ns	$10 * 3 = 30$ ns

Confronto delle prestazioni

$$T_{M1} = I * 20 \text{ ns}$$

$$T_{M2} = I * 9.78 \text{ ns}$$

M2 oltre 2 volte più veloce rispetto a M1

$$T_{M3} = I * 12.66 \text{ ns}$$

$$T_{M2} = I * 9.78 \text{ ns}$$

M2 circa 1.3 volte più veloce rispetto a M3

NB: la riduzione di CPI non paga rispetto all'aumento di T_{clock}

La macchina multiciclo con divisione “equilibrata” delle fasi è la più efficiente.

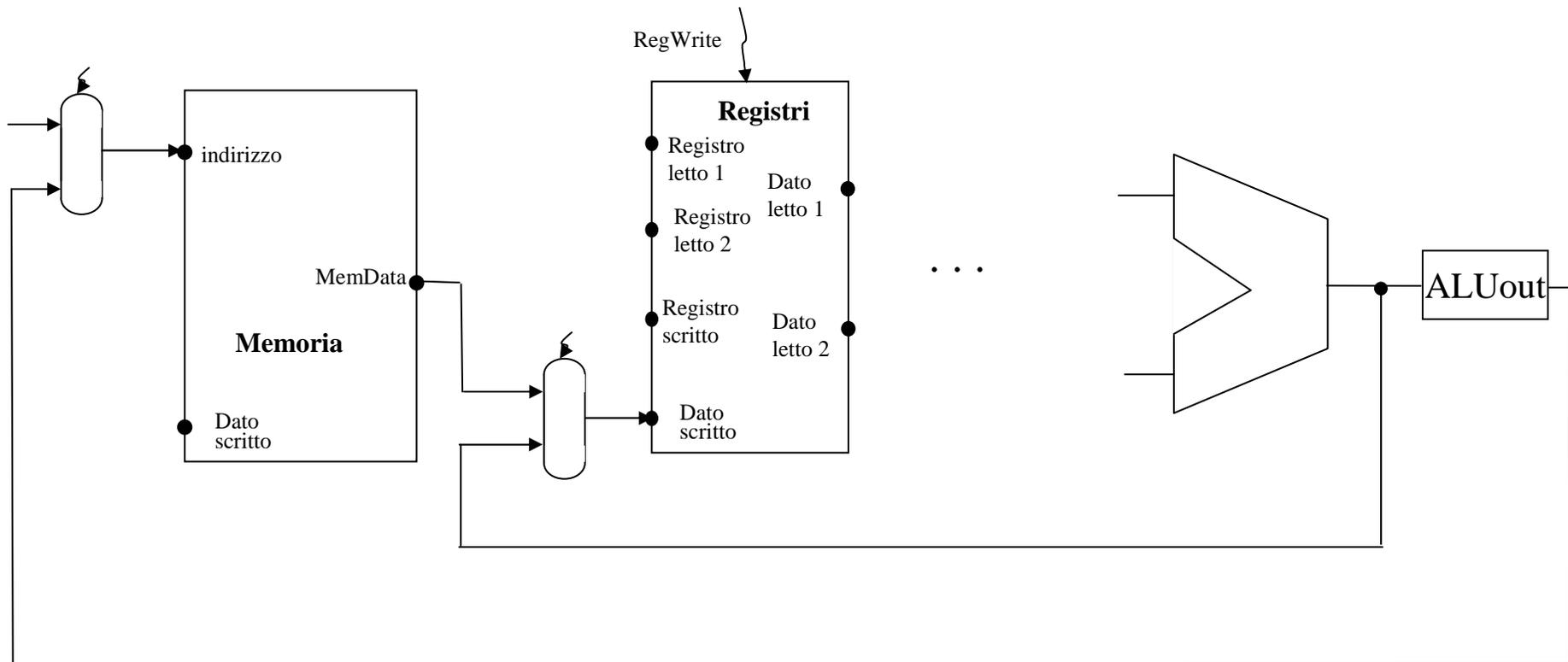
Notare però che l'istruzione più costosa (MUL in V.MOB.) è più veloce nell'implementazione a singolo ciclo.

Domanda

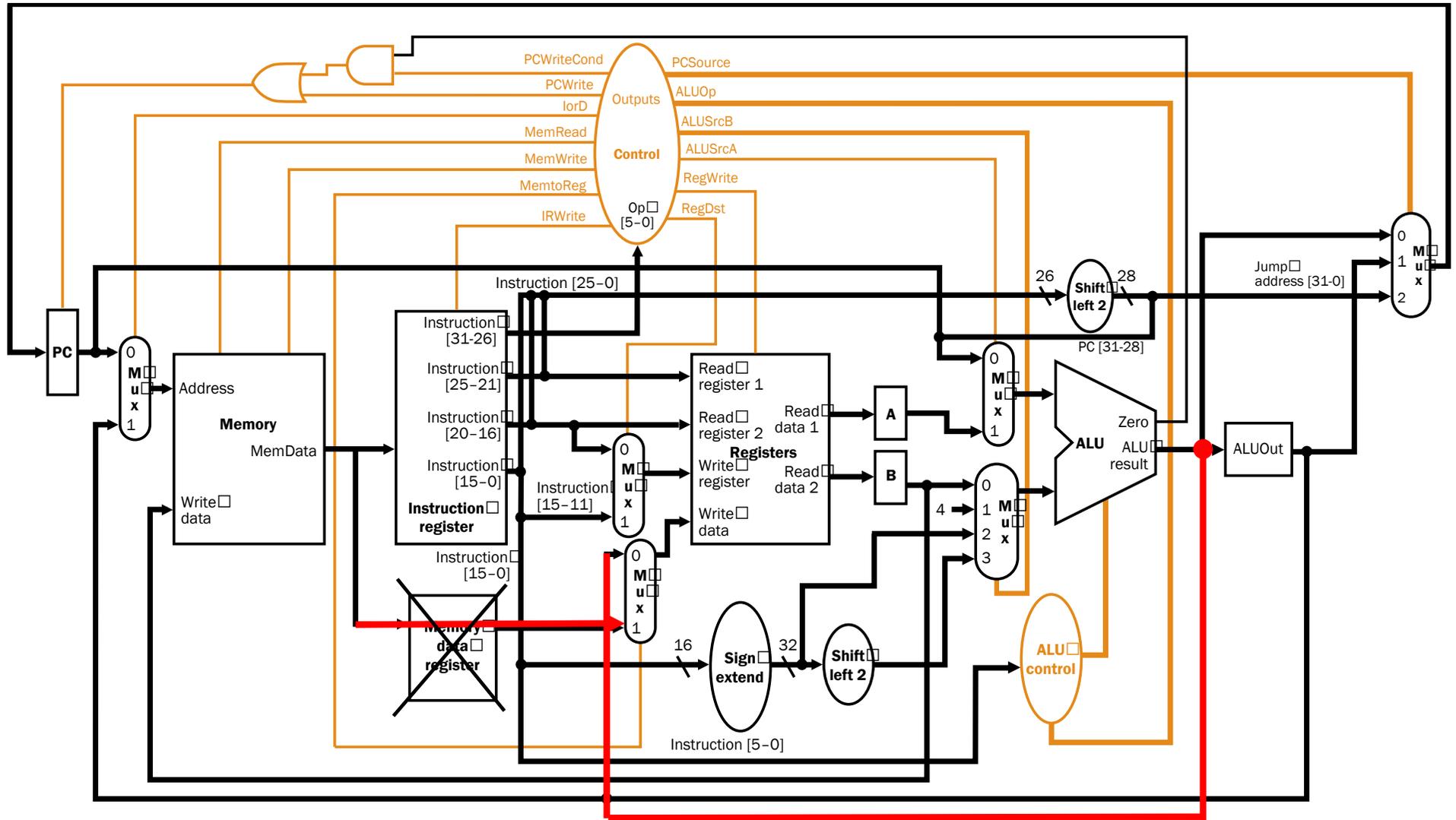
Come deve essere modificato il DataPath per implementare M3?

Si tratta di collegare “direttamente in serie”

- ALU a Register File (per scrivere in un registro il risultato di un'operazione)
⇒ linea di collegamento in uscita alla ALU prima del registro ALUout
- Memoria a Register File (per scrivere in un registro il valore letto da Mem)
⇒ elimino il registro MDR collegando direttamente memoria a Reg.File



Modificando il Datapath completo risulta:



Appello 20 settembre 2006: ES. 1 [4 punti]

Si consideri la nota implementazione dell'unità di controllo secondo la tecnica multiciclo relativamente alle istruzioni MIPS *lw*, *sw*, *beq*, *j* e *TIPO-R* [nel compito, il datapath è riportato in una figura].

Si supponga che le operazioni atomiche che coinvolgono le unità funzionali principali (di cui si tiene conto per il calcolo dei tempi) richiedano:

Unità di memoria (lettura e scrittura): 2 ns

Register File (lettura e scrittura): 1 ns

Operazione-ALU: 2 ns

Si ipotizzi che il carico di lavoro preveda in ogni caso, per le istruzioni *beq* e *j*, una percentuale complessiva del 10%:

$$f_{\text{beq}} + f_j = 10\%$$

- Determinare le condizioni in cui un'implementazione a singolo ciclo risulta più conveniente, in termini di prestazioni, rispetto a quella multiciclo.
- Rispondere alla stessa domanda supponendo che il carico di lavoro preveda per i salti (condizionati e incondizionati) una percentuale complessiva del 20% (al posto del 10%). Come si può giustificare intuitivamente il risultato?

Devo confrontare prestazioni a singolo ciclo vs. multi-ciclo:

- esprimo i tempi di esecuzione nei due casi e li confronto
- mi aspetto che le prestazioni multi-ciclo siano migliori, a meno che l'istruzione più onerosa (la *lw*) non sia “troppo frequente” rispetto al carico di lavoro (vedi le considerazioni alla fine del precedente esercizio).

Prestazioni nel caso di singolo ciclo

$$T_{\text{clock}} - \text{dovuto all'istruzione } lw: 2 \text{ ns (fetch)} + 1 \text{ ns (Reg. file)} + 2 \text{ ns (ALU)} \\ + 2 \text{ ns (Memoria dati)} + 1 \text{ ns (Reg. file)} \\ = 8 \text{ ns}$$

$$\text{CPI} = 1$$

$$\longrightarrow T_{\text{SC}} = I * 8 \text{ ns} \quad (\text{I è il numero di istruzioni})$$

Prestazioni nel caso multiciclo

$$T_{\text{clock}} = 2 \text{ ns (dovuto all'operazione più lunga: accesso a memoria o ALU)}$$

$$\text{CPI} = f_{lw} * 5 + f_{sw} * 4 + f_{\text{TIPO-R}} * 4 + f_{\text{beq}} * 3 + f_j * 3$$

$$\longrightarrow T_{\text{MC}} = I * [f_{lw} * 10 + f_{sw} * 8 + f_{\text{TIPO-R}} * 8 + f_{\text{beq}} * 6 + f_j * 6] \text{ ns}$$

Caso $f_{\text{beq}} + f_j = 10\%$

$T_{\text{SC}} < T_{\text{MC}}$ se e solo se

$$\cancel{I} * 8 < \cancel{I} * [f_{\text{lw}} * 10 + f_{\text{sw}} * 8 + f_{\text{TIPO-R}} * 8 + f_{\text{beq}} * 6 + f_j * 6]$$

... cerco di “raccoliere” per esprimere $f_{\text{beq}} + f_j = 0.1$

$$8 < f_{\text{lw}} * 10 + f_{\text{sw}} * 8 + f_{\text{TIPO-R}} * 8 + (f_{\text{beq}} + f_j) * 6$$

$$8 < f_{\text{lw}} * 10 + f_{\text{sw}} * 8 + f_{\text{TIPO-R}} * 8 + 0.1 * 6$$

$$7.4 < f_{\text{lw}} * 10 + f_{\text{sw}} * 8 + f_{\text{TIPO-R}} * 8$$

E adesso?

Devo cercare di ottenere una sola incognita, che potrebbe essere f_{lw} ...

$$7.4 < f_{lw} * 10 + f_{sw} * 8 + f_{TIPO-R} * 8$$

So che

$$\left. \begin{array}{l} \bullet f_{lw} + f_{sw} + f_{TIPO-R} + f_j + f_{beq} = 1 \\ \bullet f_{beq} + f_j = 0.1 \end{array} \right\} f_{lw} + f_{sw} + f_{TIPO-R} = 0.9$$

$$\rightarrow 7.4 < f_{lw} * 10 + \underbrace{(f_{sw} + f_{TIPO-R})}_{0.9 - f_{lw}} * 8$$

$$\rightarrow 7.4 < f_{lw} * 10 + (0.9 - f_{lw}) * 8$$

$$\rightarrow f_{lw} > 0.2/2 = 0.1$$

\rightarrow Prestazioni a singolo ciclo prevalgono quando lw ha frequenza maggiore del 10%
(NB: corrisponde al “tipo di risultato” che ci aspettiamo)

Caso $f_{\text{beq}} + f_j = 20\%$

Intuitivamente: cresce la frequenza delle istruzioni con tempi di esecuzione inferiore \Rightarrow ci aspettiamo che la tecnica multiciclo “prevalga ancora di più”, ovvero che la percentuale di f_{lw} per cui singolo ciclo è conveniente cresca.

Rifacendo i calcoli con il nuovo vincolo risulta:

$$8 < f_{lw} * 10 + f_{sw} * 8 + f_{TIPO-R} * 8 + 0.2 * 6$$

$$6.8 < f_{lw} * 10 + f_{sw} * 8 + f_{TIPO-R} * 8$$

$$\text{con } f_{lw} + f_{sw} + f_{TIPO-R} = 0.8$$

$$\rightarrow 6.8 < f_{lw} * 10 + (0.8 - f_{lw}) * 8$$

$$\rightarrow 0.4 < f_{lw} * 2$$

$$\rightarrow f_{lw} > 0.4 / 2 = 0.2$$

\rightarrow In questo caso, prestazioni a singolo ciclo prevalgono quando lw ha frequenza maggiore del 20%

Esercizio

Si ipotizzi un **carico di lavoro** che preveda:

lw:	25%
sw:	10%
Tipo-R:	52%
beq:	11%
j:	2%

Si supponga che le **operazioni atomiche** che coinvolgono le unità funzionali principali (di cui si tiene conto per il calcolo dei tempi) richiedano:

Unità di memoria (lettura e scrittura):	2 ns
Register File (lettura e scrittura):	1 ns
Operazione-ALU:	1 ns

- Determinare le prestazioni multi-ciclo (non mettere in serie due operazioni atomiche)
- Delineare, se possibile, una modifica al controllo e al data path in grado di migliorare l'efficienza della soluzione precedente (sempre nell'ambito del controllo multi-ciclo)
- Confrontare le prestazioni delle due soluzioni ottenute

1) Controllo multi-ciclo “usuale”

$$T_{M1} = \# \text{istruzioni} * \text{CPI} * T_{\text{clock}}$$

$$T_{\text{clock}} = 2\text{ns} \quad (\text{accesso in memoria})$$

Istruzione	Fetch	Lettura Registri/Decode	ALU	Accesso Memoria	Scrittura Registri	CPI
lw	1	1	1	1	1	5
sw	1	1	1	1		4
Tipo-R	1	1	1		1	4
beq	1	1	1			3
j	1	1			1	3

$$\text{CPI} = 0.25 * 5 + 0.1 * 4 + 0.52 * 4 + 0.11 * 3 + 0.02 * 3 = 4.12$$

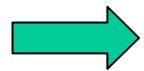
 $T_{M1} = I * 4.12 * 2 \text{ ns} = I * 8.24 \text{ ns}$

2) Soluzione più efficiente

Unità di memoria (lettura e scrittura): 2 ns

Register File (lettura e scrittura): 1 ns

Operazione-ALU: 1 ns



Operazione ALU e accesso a Register File possono essere messe in serie (diminuendo CPI) senza incrementare T_{clock} !!!

Istruzione	Fetch	Lettura Registri/Decode	ALU	Accesso Memoria	Scrittura Registri	CPI
lw	1	1	1	1	1	5
sw	1	1	1	1		4
Tipo-R	1	1	1		1	4 3
beq	1	1	1			3
j	1	1			1	3

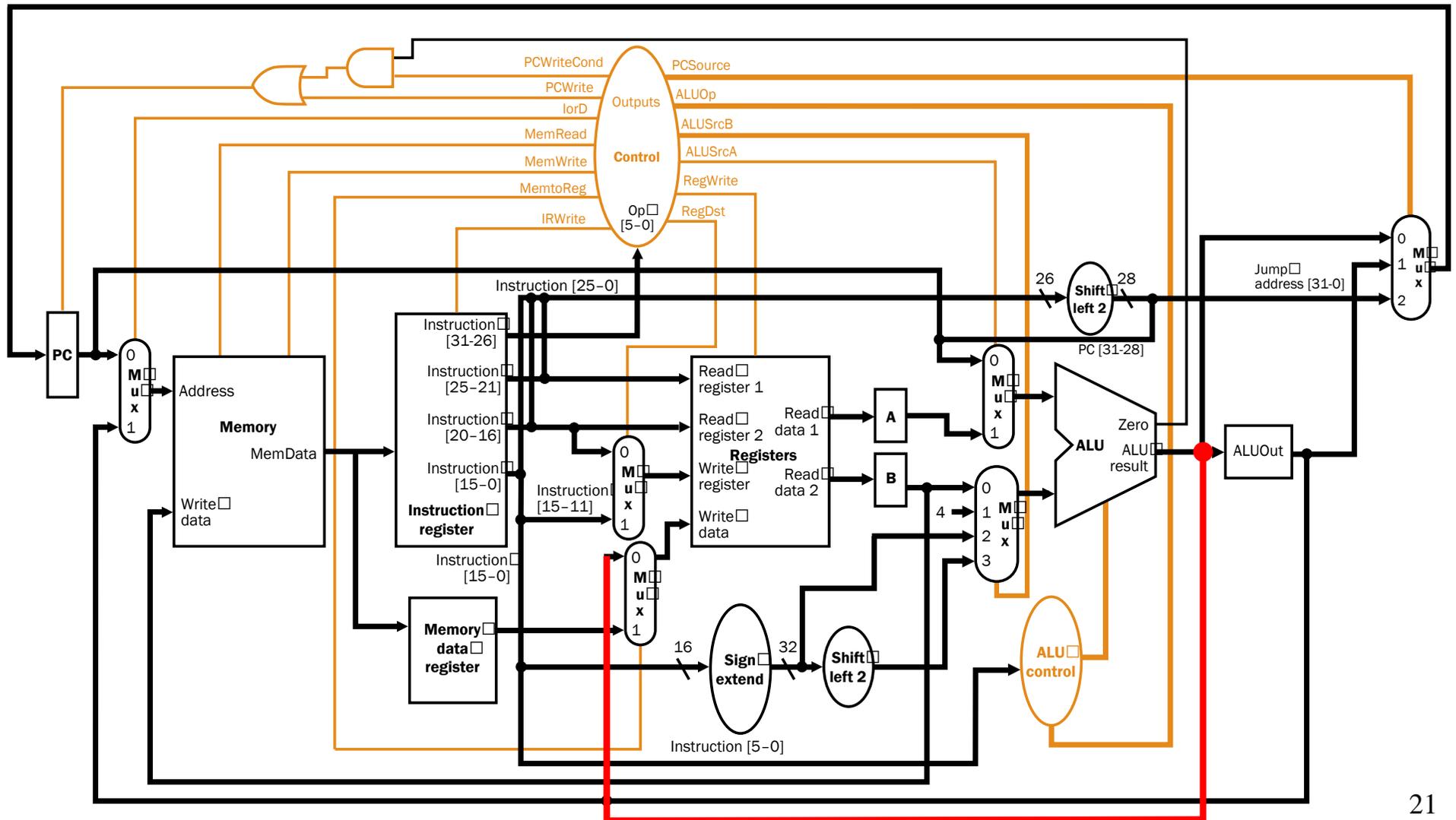


$$\text{CPI} = 0.25 * 5 + 0.1 * 4 + 0.52 * 3 + 0.11 * 3 + 0.02 * 3 = 3.6$$

e quindi risulta

$$T_{M2} = I * 3.6 * 2 \text{ ns} = I * 7.2 \text{ ns}$$

Ovviamente, occorre modificare leggermente il DataPath...



3) Confronto prestazioni

$$T_{M1} = I * 8.24 \text{ ns}$$

$$T_{M2} = I * 7.2 \text{ ns}$$

M2 è circa 1,14 volte più veloce

NB: Perché non la soluzione seguente?

Istruzione	Fetch	Lettura Registri/Decode	ALU	Accesso Memoria	Scrittura Registri	CPI
lw	1	1	1	1	1	5
sw	1	1	1	1		4
Tipo-R	1	1	1		1	4
beq	1	1	1			3
j	1	1			1	3

... pensarci ...

Altra tipologia:

implementazione istruzioni nuove

Esercizio

Si consideri il DataPath per l'implementazione multiciclo del MIPS [che avete a disposizione!]. Si consideri l'ipotetica istruzione assembler:

j (rs) ; $PC \leftarrow rs$

la quale esegue un salto incondizionato all'indirizzo specificato dal registro rs.

Per questa istruzione:

- si dica se (e come) deve essere modificato il **datapath** per permetterne l'implementazione rispettando il vincolo usuale su elementi funzionali;
- supponendo che vi sia un codice operativo "libero" (non impegnato in altre istruzioni) si fornisca un possibile **formato** della corrispondente istruzione macchina e si specifichi come deve essere modificato il **diagramma a stati finiti** dell'unità di controllo per implementarla;
- supponendo $T_{\text{clock}} = 2\text{ns}$, calcolare il tempo di esecuzione della nuova istruzione.

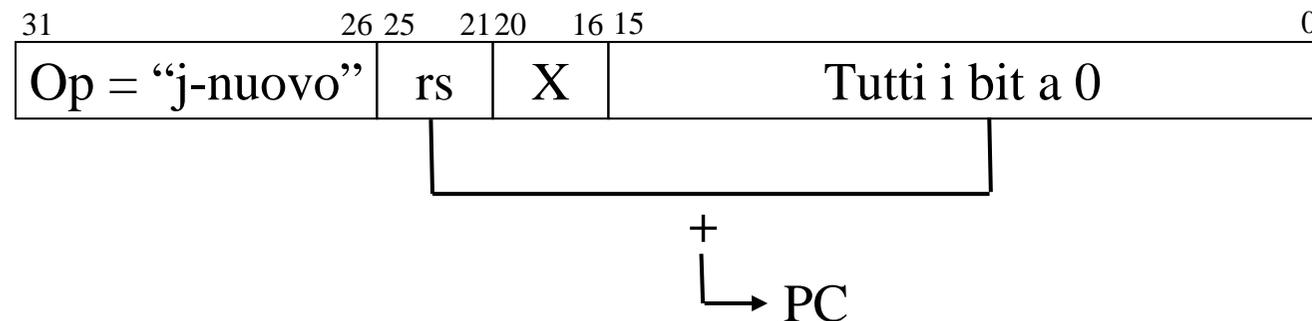
j (rs)

Osservando il datapath, si vede che:

- è possibile forzare la scrittura di PC con l'uscita del MUX [ALUout| ALU| indirizzo j] ponendo PCWrite = 1
- visto che ALUout proviene da ALU, consideriamo ALU!
- in ingresso alla ALU, il primo operando deve essere necessariamente A=rs; per ottenere rs in uscita alla ALU, l'unica possibilità è avere sign-extend(offset) [ev. shiftato di 2 bit] purchè offset=0 e forzare ALU ad effettuare una somma:

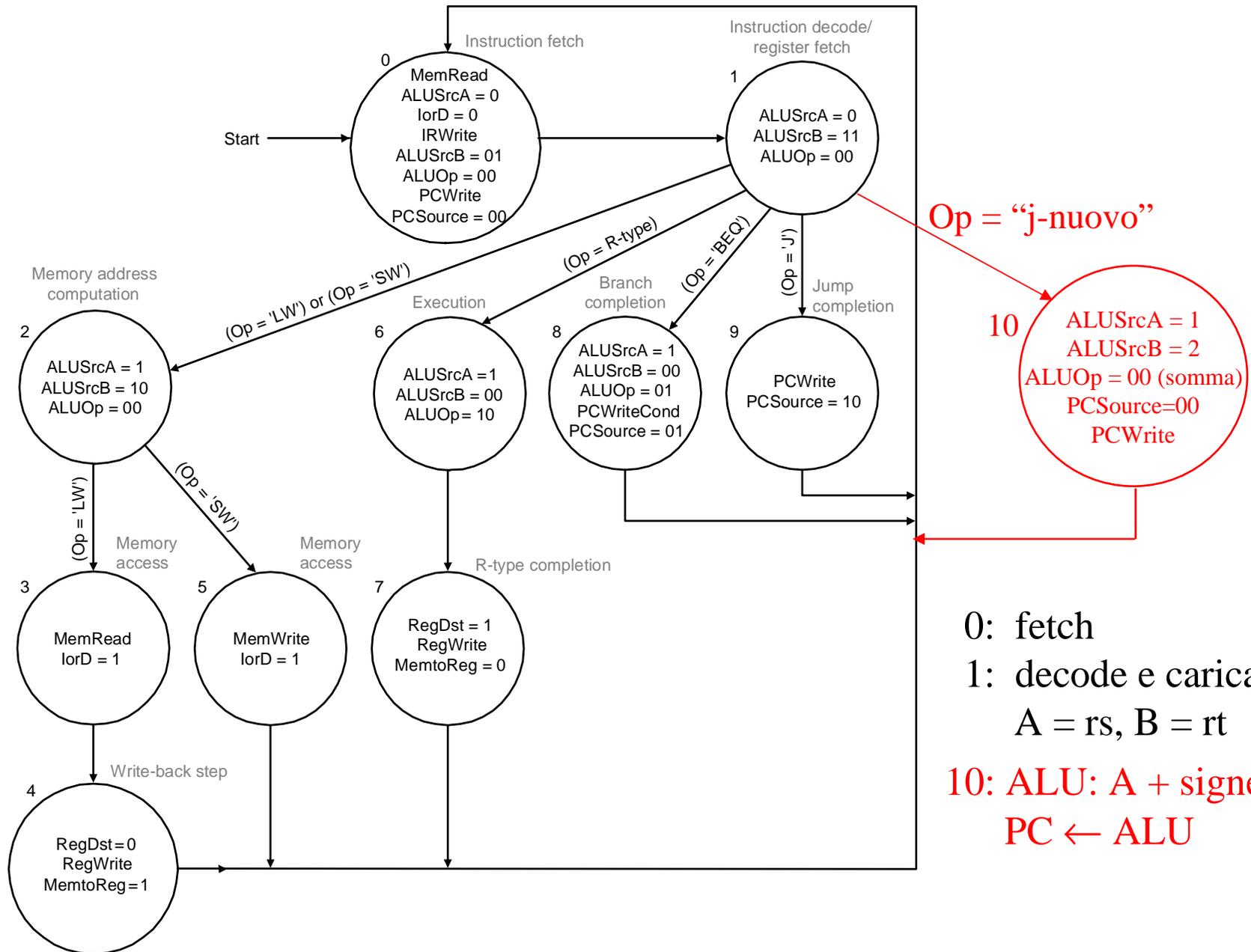
$$rs + \text{sign-extend}(0) = rs$$

➡ E' possibile implementare l'istruzione con il **formato Tipo-I**:



X: don't care

Vediamo il controllo con il diagramma a stati, partendo da quello originario...



NB: in pratica abbiamo modificato l'ultimo stato dell'istruzione j
in modo da selezionare l'uscita della ALU (che calcola l'indirizzo)
al posto dell'usuale indirizzo derivante dall'offset a 26 bit.

Notare che in questo stato il cammino creato è $A-IR \rightarrow ALU \rightarrow PC$:
come nel caso di beq, non poniamo in serie più di un elemento "critico"

Tempo di esecuzione: $3 * 2 \text{ ns} = 6 \text{ ns}$ [come j e beq!]

Notare che è possibile implementare l'istruzione più generale (stesso formato Tipo-I)

j offset(rs) // (il campo offset può essere diverso da zero)

dove però è più ragionevole supporre che offset sia espresso in parole di memoria!

A questo scopo, basta forzare $ALUSrcB=3$ per sommare a rs offset esteso e scalato.

Possibile problema: se il valore di rs non è multiplo di 4, potremmo saltare ad
un indirizzo non corrispondente ad una istruzione
(ciò non si verifica con la beq, dove il registro base è PC)



Appello 13 luglio 2005: ES. 1 [6 punti]

Si considerino, mostrati nelle figure riportate di seguito, il datapath ed il diagramma a stati finiti che specifica l'unità di controllo secondo la tecnica a multiciclo relativamente alle istruzioni MIPS lw, sw, beq, j ed alle istruzioni di tipo-R. Si vuole implementare una nuova istruzione del tipo:

addmem rt, offset(rs) // $rt \leftarrow rt + M[rs+offset]$

che somma al registro rt il contenuto della locazione di memoria indirizzata da rs+offset, dove offset è un numero a 16 bit con segno specificato nell'istruzione macchina (cfr. le istruzioni lw e sw).

Ricordando i tre formati di codifica delle istruzioni (riportati di seguito) si chiede di:

- riportare il formato della nuova istruzione macchina
- riportare, nella corrispondente figura, le modifiche necessarie al datapath
- estendere il diagramma degli stati per implementare la nuova istruzione

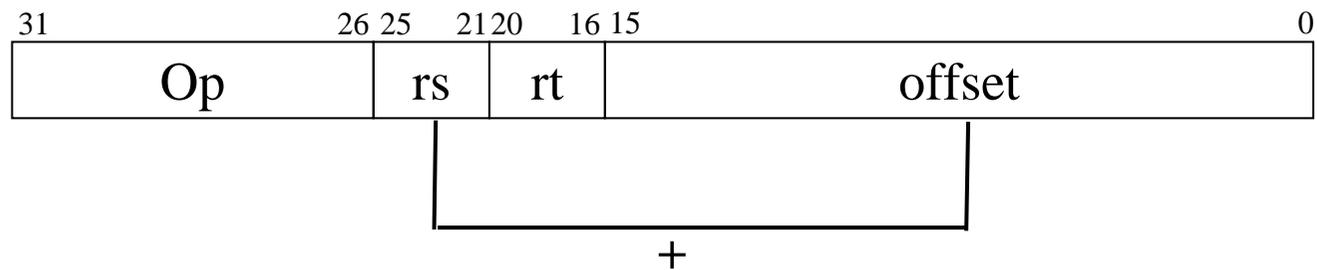
addmem rt, offset(rs)

Formato

devo rappresentare:

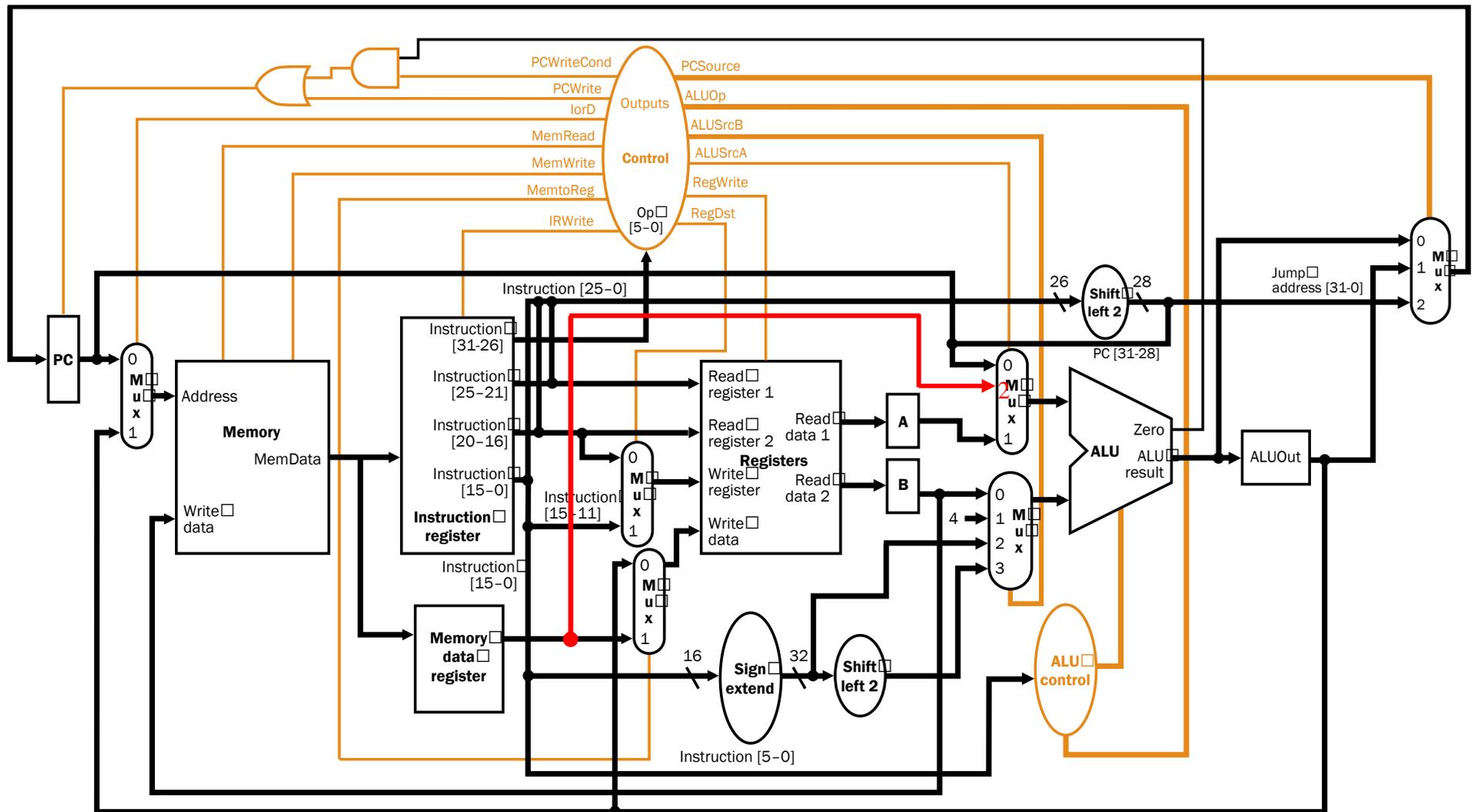
- il registro rs
- il registro rt
- offset (a 16 bit) che va sommato a rs

➡ Sicuramente verrà usato il Tipo-I



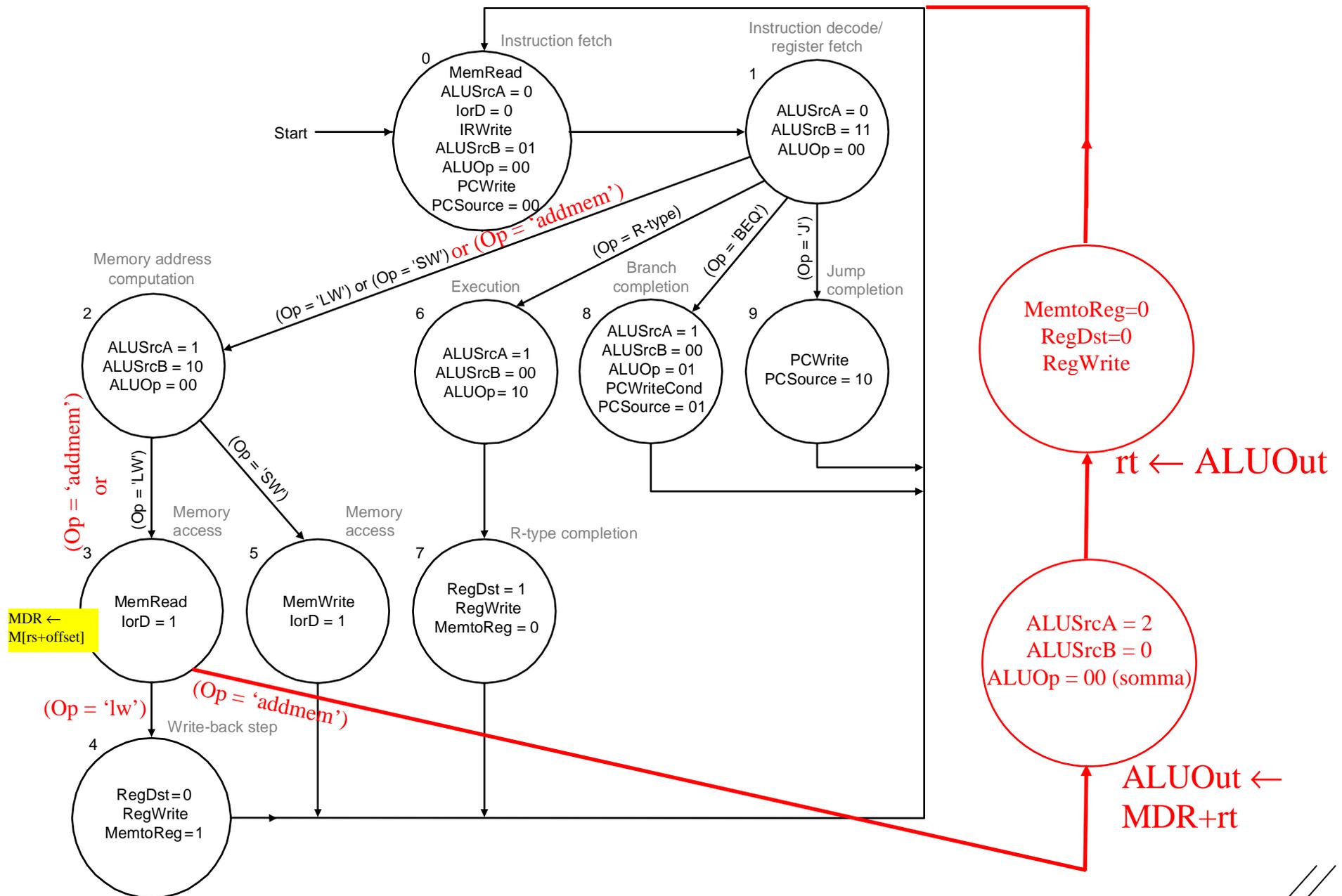
(cfr. anche lw)

Modifica al DataPath *addmem rt, offset(rs) : è una sorta di “lw + add”*



Aggiungiamo ingresso al MUX, con label 2 per non modificare la specifica delle precedenti istruzioni...

Possiamo allora effettuare le stesse operazioni della lw ed aggiungere in coda due stati



Stesso Appello: ES. 2 [4 punti]

Si consideri l'esercizio risolto al punto precedente;
si assuma un carico di lavoro che preveda la seguente distribuzione delle istruzioni:

lw:	20 %
sw:	20 %
Tipo-R:	30 %
beq:	10 %
j:	10 %
addmem:	10 %

Si supponga che le operazioni atomiche delle unità funzionali principali richiedano:

Unità di memoria (lettura e scrittura):	2 ns
Register File (lettura e scrittura):	1 ns
Operazione ALU:	2 ns

Si confrontino le prestazioni (in termini di rapporto tra tempi di esecuzione) di un'ipotetica implementazione a singolo ciclo rispetto all'implementazione multiciclo individuata al punto precedente.

Singolo ciclo

T_{clock} è valutato sull'istruzione più lunga; è facile rendersi conto che questa è addmem, il cui tempo di esecuzione è maggiore della lw che era già l'istruzione più lunga...

In particolare:

Fetch:	2 ns
Prelievo rs:	1 ns
ALU (rs+offset):	2 ns
Accesso Mem:	2 ns
ALU(rt+M[...]):	2 ns
Scrittura rt:	1 ns
	<hr/>
	10 ns



$$T_{\text{esecuzione}} = \# \text{istruzioni} * T_{\text{clock}} = \# \text{istruzioni} * 10$$

Multiciclo

$$T_{\text{esecuzione}} = \# \text{istruzioni} * \text{CPI} * T_{\text{clock}}$$

T_{clock} è valutato sull'operazione più lunga eseguita in un ciclo di clock;
le operazioni più lunghe (accesso a memoria e uso ALU) durano 2 ns

 $T_{\text{clock}} = 2 \text{ ns}$

CPI va valutato sul carico di lavoro, tenendo presente che addmem ha 6 cicli:

 $\text{CPI} = 0,2 * 5 + 0,2 * 4 + 0,3 * 4 + 0,1 * 3 + 0,1 * 3 + 0,1 * 6 = 4,2$

QUINDI SI OTTIENE:

$$T_{\text{esecuzione}} = \# \text{istruzioni} * 4,2 * 2 = \# \text{istruzioni} * 8,4$$

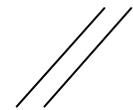
Confronto Prestazioni Singolo Ciclo vs. Multiciclo

$$T_{\text{singolo}} = \# \text{istruzioni} * 10$$

$$T_{\text{multi}} = \# \text{istruzioni} * 8,4$$

Evidentemente la realizzazione multiciclo ha prestazioni migliori.
In particolare risulta:

$$\frac{T_{\text{singolo}}}{T_{\text{multi}}} = \frac{\# \text{istruzioni} * 10}{\# \text{istruzioni} * 8,4} = 1,19$$



Esercizio

lw rt, (rs)++ ; rt ← M[rs] / rs = rs+4 (implementare la nuova istruz.)

Analisi del DataPath:

- possiamo prima procedere al calcolo dell'indirizzo come nel caso della lw:

stato 0: fetch

stato 1: lettura registri $A \leftarrow rs$, $B \leftarrow rt$

stato 3: $ALUOut \leftarrow rs + \text{signext}(\text{offset})$ // offset deve essere nullo...

stato 4: $MDR \leftarrow M[AluOut]$

stato 5: $rt \leftarrow MDR$

- una volta usato rs, deve essere incrementato di 4 byte:

useremo ALU per calcolare $A + 4 [= rs + 4]$ e porre risultato in ALUOut

$rs \leftarrow ALUOut$: dobbiamo poter scrivere rs!

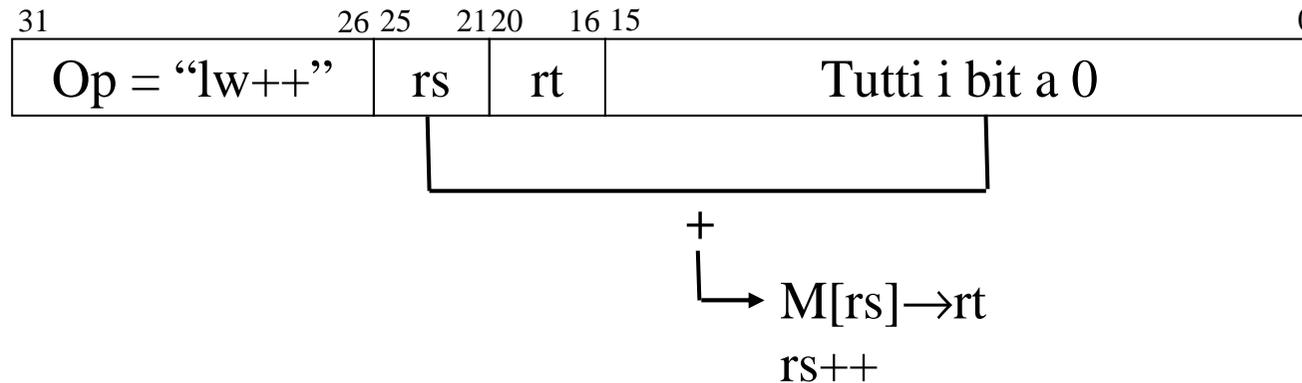
E' necessario estendere MUX in entrata al register file,
che attualmente prevede solo rt [per lw] e rd [per Tipo-R]

Da ciò deriva:

- il formato dell'istruzione (Tipo-I come quello della lw)
- la modifica al DataPath richiesta



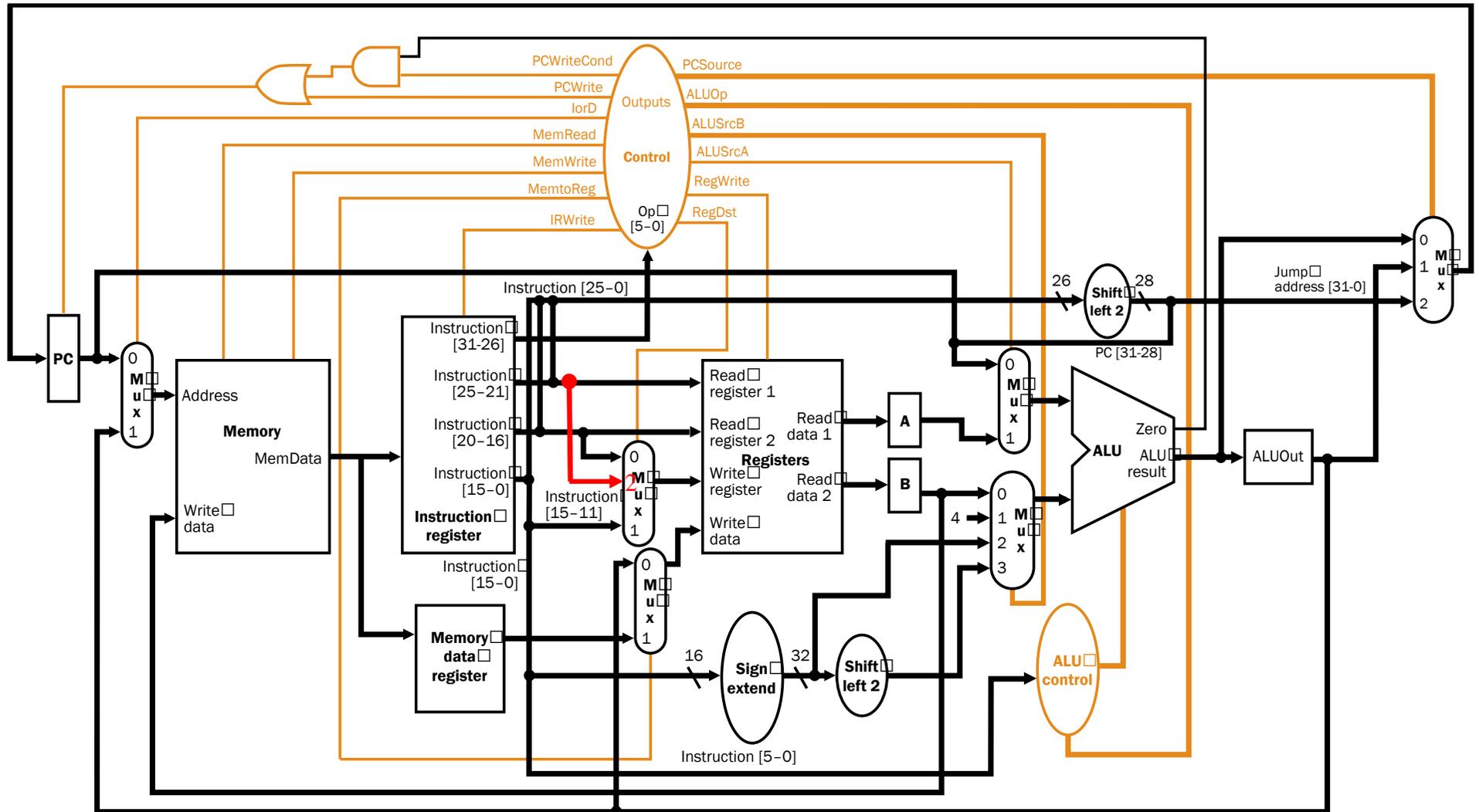
E' possibile implementare l'istruzione con il **formato Tipo-I**:



NB: è chiaro che, permettendo $\text{offset} \neq 0$, possiamo implementare una istruzione del tipo

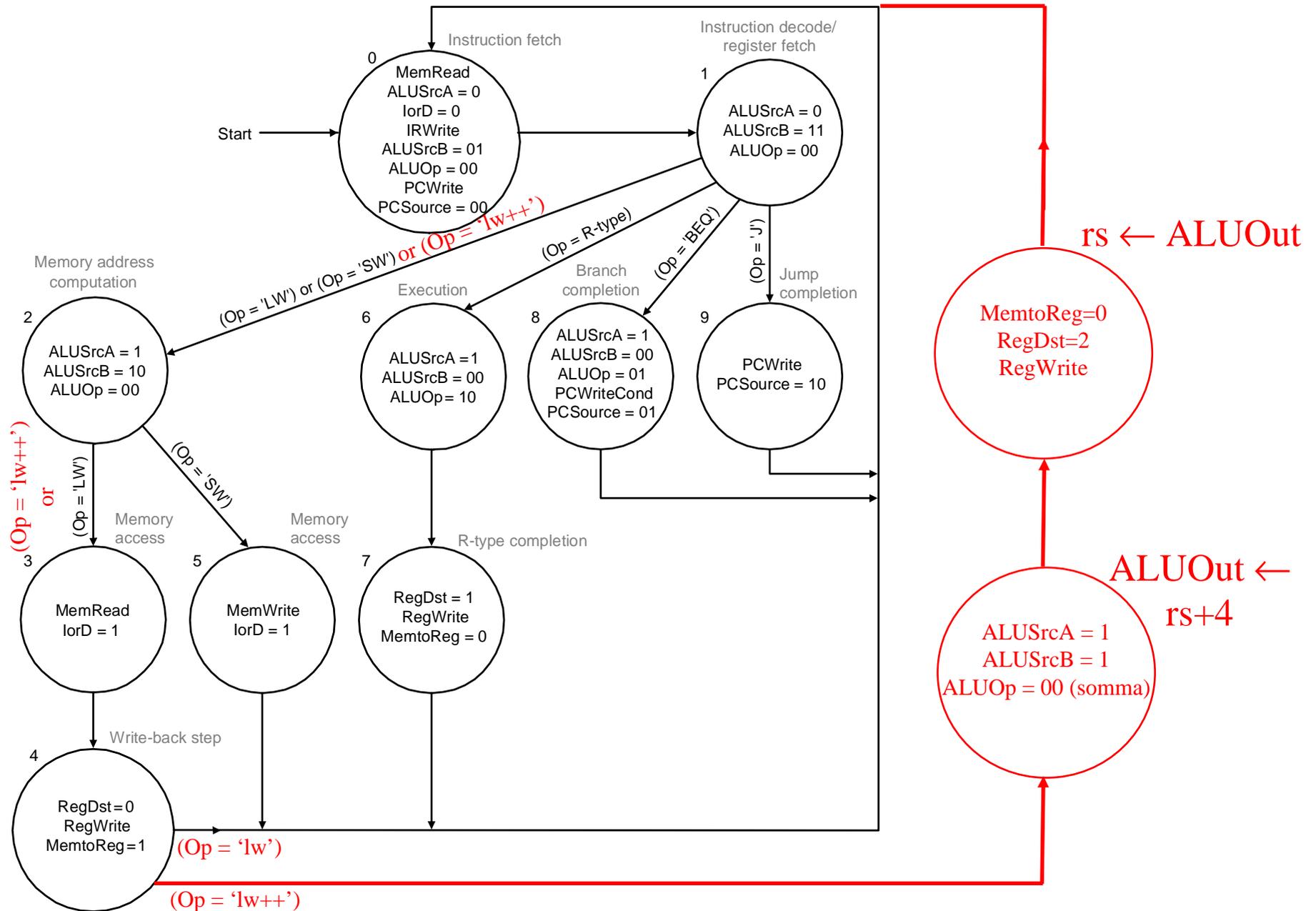
$\text{lw rt, offset(rs)++ ; rt} \leftarrow \text{M[rs+offset]} / \text{rs}=\text{rs}+4$

Modifica al DataPath



Aggiungiamo ingresso al MUX, con label 2 per non modificare la specifica delle precedenti istruzioni...

Possiamo allora effettuare le stesse operazioni della lw ed aggiungere in coda due stati



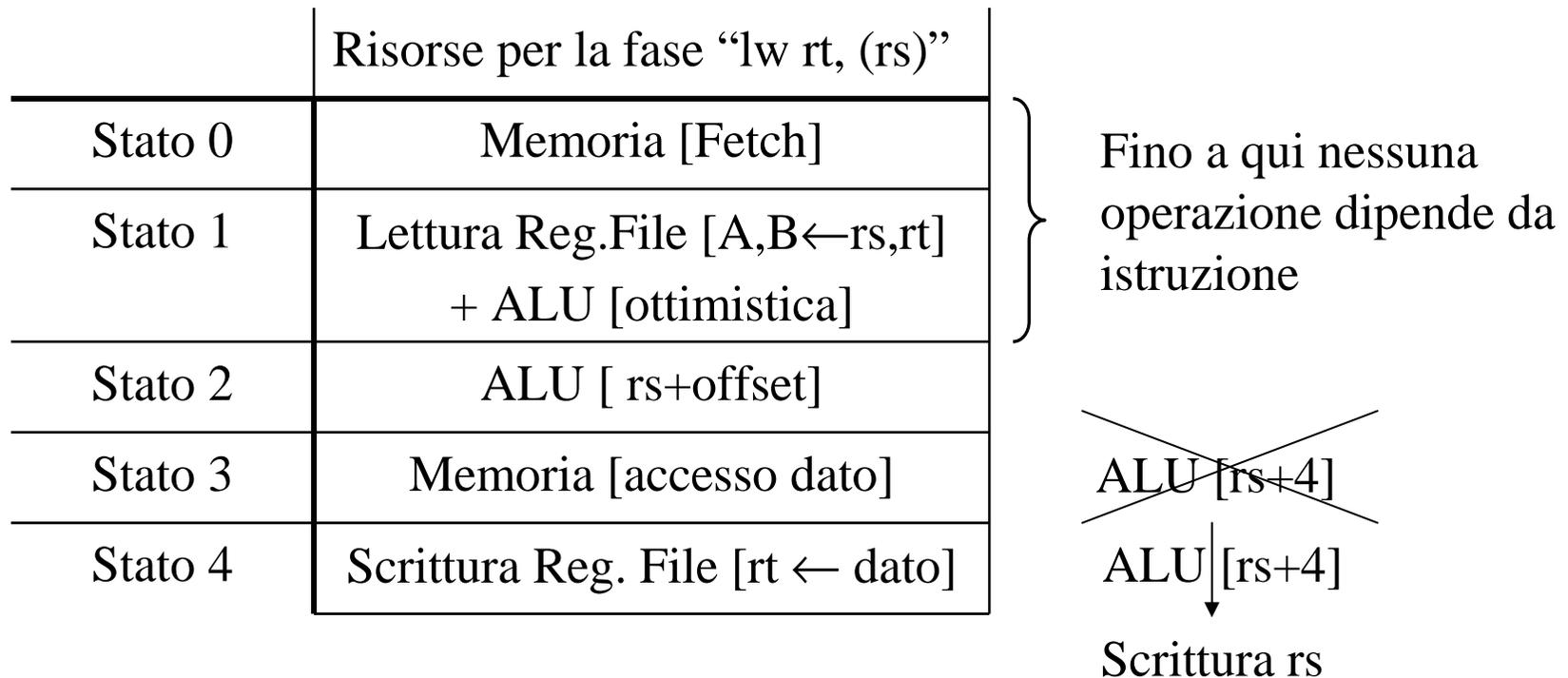
Questa soluzione porta a 7 cicli di clock, esecuzione in $7 * 2ns = 14 ns$.

Soluzione più efficiente:

Il calcolo dell'indirizzo è effettuato nel terzo stato [stato 2] utilizzando valore di rs letto nel secondo stato (di decode) e posto nel registro temporaneo A

- se incremento rs dopo il secondo stato non ci sono problemi
- dopo il terzo stato [stato 2] ALU non è più usata ed è disponibile per il calcolo

 IDEA: anticipare calcolo e scrittura di $rs++$ negli stati precedenti!



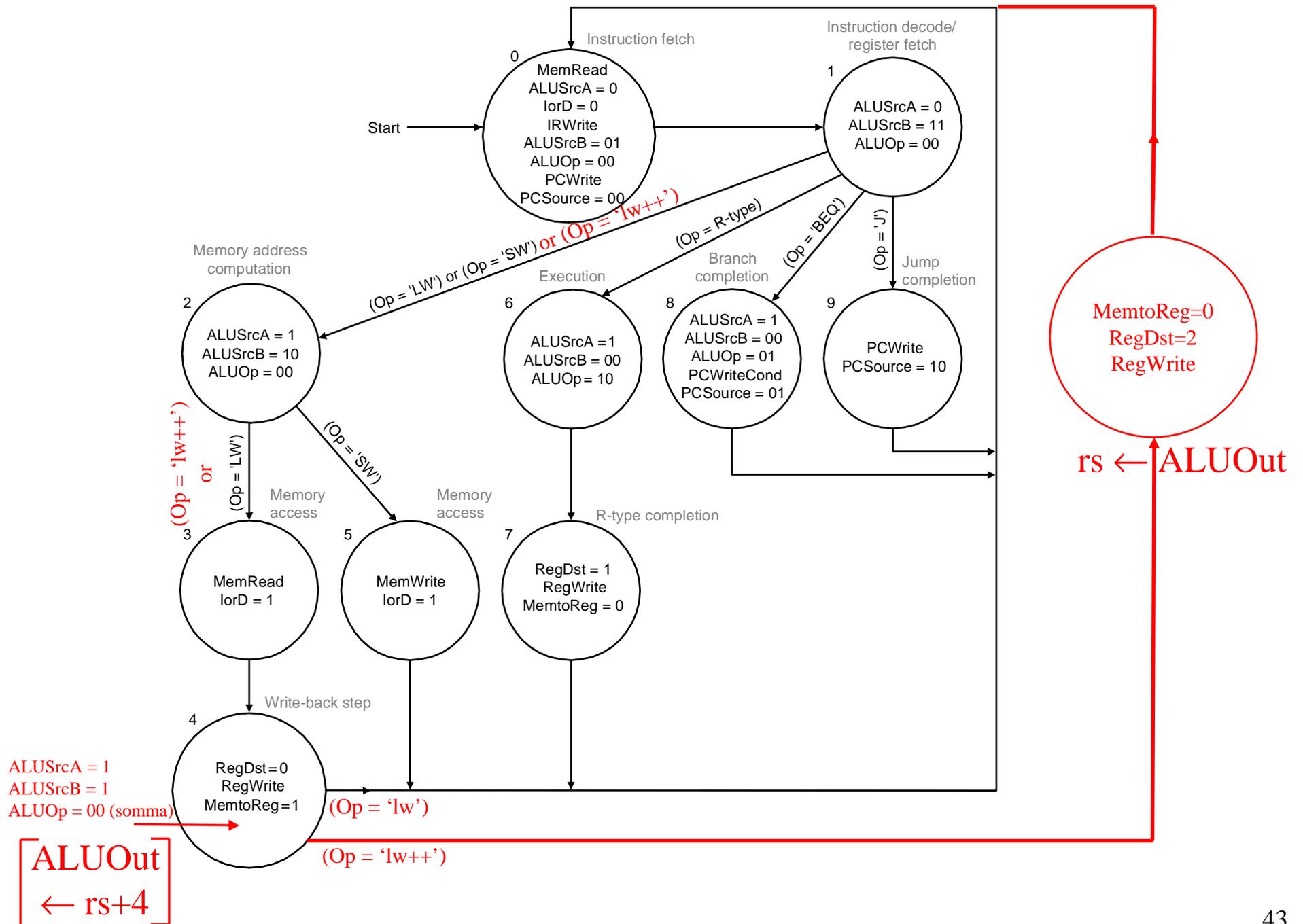
- Se calcolassimo nel quarto stato $rs+4$ [posto in ALUOut] poi comunque dovremmo aspettare uno stato per scriverlo [nel quinto stato accedo al Reg.File per la scrittura di rt] quindi ALUOut sarebbe sovrascritto nel quinto stato...
- Conviene usare la ALU per calcolare $rs+4$ nel quinto stato [stato 4], mentre si sta scrivendo rt con il valore prelevato in memoria. Posso allora aggiungere nel quinto stato (comune a lw e $lw++$) i segnali per il calcolo $ALUOut \leftarrow rs+4$; nel caso della lw , il valore non verrà usato, ma ciò non comporta una inefficienza, visto che l'operazione è svolta in parallelo!
- Basta poi aggiungere solo uno stato in più [esclusivo per la $lw++$] in cui si scrive rs con il valore posto in ALUOut [vedi diagramma a stati modificato nel lucido seguente]



$lw++$ in 6 cicli di clock, pari a $6 * 2ns = 12 ns$

	lw rt, (rs)++
Stato 0	Memoria [Fetch]
Stato 1	Lettura Reg.File [A,B ← rs,rt]
Stato 2	ALU [rs+offset]
Stato 3	Memoria [accesso dato] : MDR ← M[rs] A ← rs
Stato 4	Scrittura Reg. File [rt ← MDR] ALU[rs+4] : ALUout=A+4
Stato 5	Scrittura Reg. File[rs] con ALUOut

 Implementazione a 6 cicli



Confrontiamo il tempo di esecuzione dell'istruzione implementata

lw rt, (rs)++

6 cicli di clock = 12 ns

con l'uso equivalente delle due istruzioni

lw rt, 0(rs)

5 cicli di clock +

addi rs, rs, 4

4 cicli di clock

 Risparmiamo tre cicli di clock:

1 per la fase di fetch [una istruzione vs. 2]

1 per la fase di “decode” [lettura $A \leftarrow rs$] eseguita in parallelo

1 per la fase di calcolo $rs++$ eseguita in parallelo

- In questo caso, l'uso di modalità di indirizzamento più complesse (filosofia “CISC”) risulta più efficiente.
- Come vedremo, con l'introduzione della **pipeline** ciò non è necessariamente vero: un certo grado di parallelismo è “automatico”:
ad esempio, la fase di fetch avviene sempre in parallelo con esecuzione di una o più istruzioni precedenti che si trovano ancora nella pipeline.
Per contro, come si vedrà, modalità di indirizzamento complesse rendono più difficile la gestione delle criticità della pipeline...

ULTIMA DOMANDA:

Come si comporta l'istruzione implementata (nell'ultima maniera) se $rs = rt$?
Come dovrebbe comportarsi?

`lw rs, (rs)++`

Pensarci e provare a fare delle considerazioni in proposito...

lw rs, (rs)++

- 1) rs viene utilizzato per calcolare l'indirizzo [terzo stato: $ALUOut = rs + offset$]
- 2) Quarto stato: $MDR \leftarrow M[rs]$ e intanto viene letto $A \leftarrow rs$
- 3) Quinto stato: $rs \leftarrow M[rs]$ e $ALUOut \leftarrow \text{"rs vecchio"} + 4$
- 4) Sesto stato: $rs \leftarrow \text{"rs vecchio"} + 4$ che sovrascrive il valore precedente!

 L'effetto è solo quello di sommare 4 a rs !!!

Potremmo pensare a varie possibilità:

- accettare questo comportamento

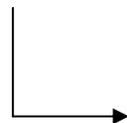
[il programmatore non userà $rt = rs$ per ottenere in 6 cicli il comportamento che la ADD ottiene con 4 cicli!]

- considerare illegale l'istruzione se $rs = rt$

- implementare un comportamento diverso:

$rs \leftarrow M[rs]$

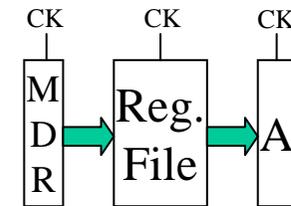
$rs \leftarrow rs + 4$



Notare che, dopo che rs è scritto [alla fine del quinto ciclo di clock] è necessario un altro ciclo di clock per leggere rs e caricarlo in A!

Quindi, neppure l'implementazione meno efficiente funziona!

	lw rs, (rs)++
Stato 0	Memoria [Fetch]
Stato 1	Lettura Reg.File [A,B←rs,rs]
Stato 2	ALU [rs+offset]
Stato 3	Memoria [accesso dato]
Stato 4	Scrittura Reg. File [rs ← dato] A ← rs-vecchio
Stato 5	ALU [rs-vecchio +4]
Stato 6	Scrittura Reg. File [rs-vecchio+4]



Qui rs è utilizzato in lettura con il valore vecchio!!!

Soluzione: o facciamo in modo (con HW supplementare) che, con lettura e scrittura del Register File nello stesso ciclo di clock, si legga il valore “aggiornato” oppure...

...oppure inseriamo una fase in più per leggere rs solo dopo che è stato aggiornato...

	lw rs, (rs)++
Stato 0	Memoria [Fetch]
Stato 1	Lettura Reg. File [A,B←rs,rs]
Stato 2	ALU [rs+offset]
Stato 3	Memoria [accesso dato]
Stato 4	Scrittura Reg. File [rs ← dato]
Stato 5	A ← rs
Stato 6	ALU [rs + 4]
Stato 7	Scrittura Reg. File [rs+4]



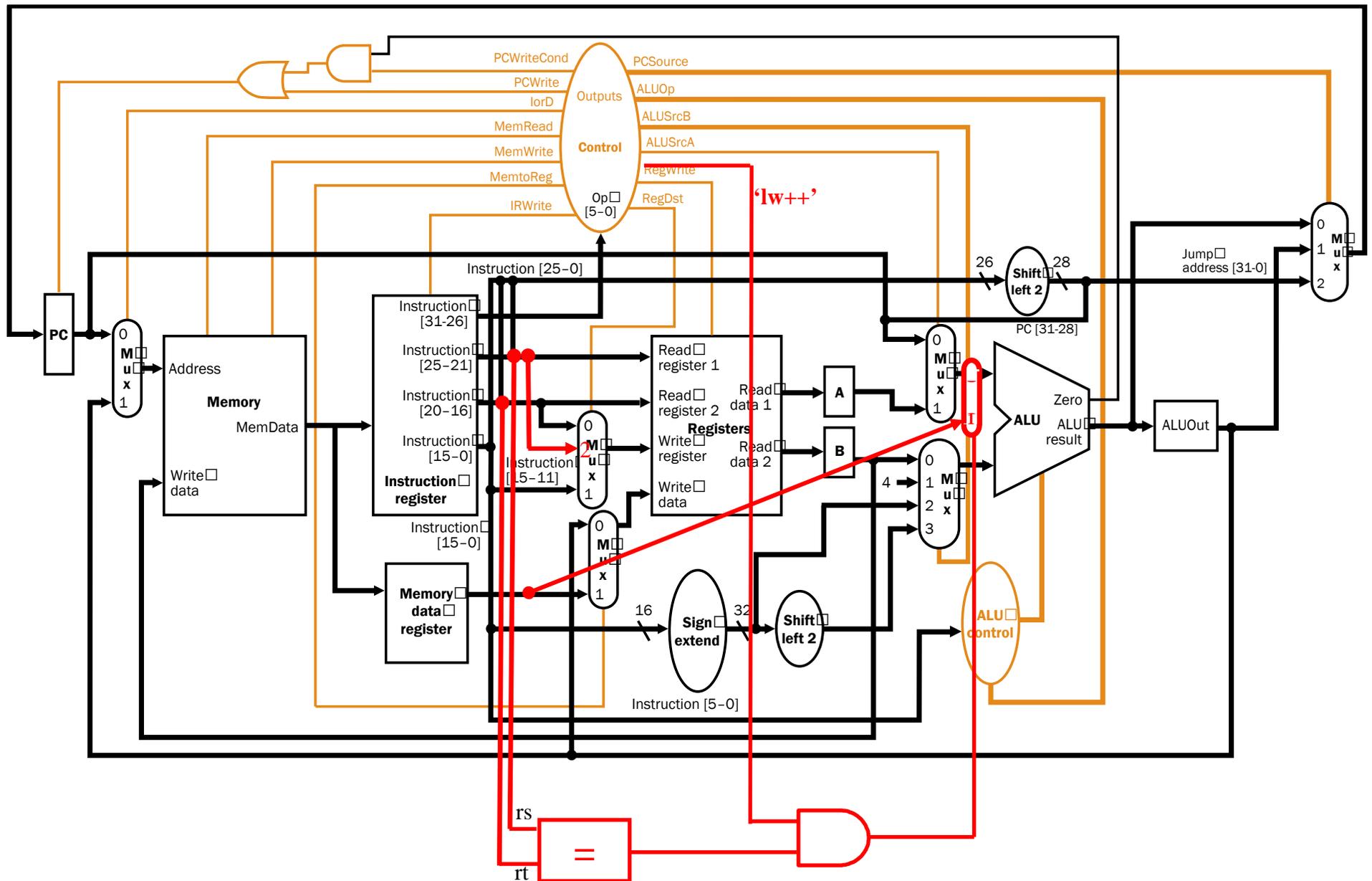
Implementazione con 8 cicli di clock vs. 9 delle due istruzioni elementari (risparmiamo solo il fetch dell'istruzione ADDI rs, rs, 4)

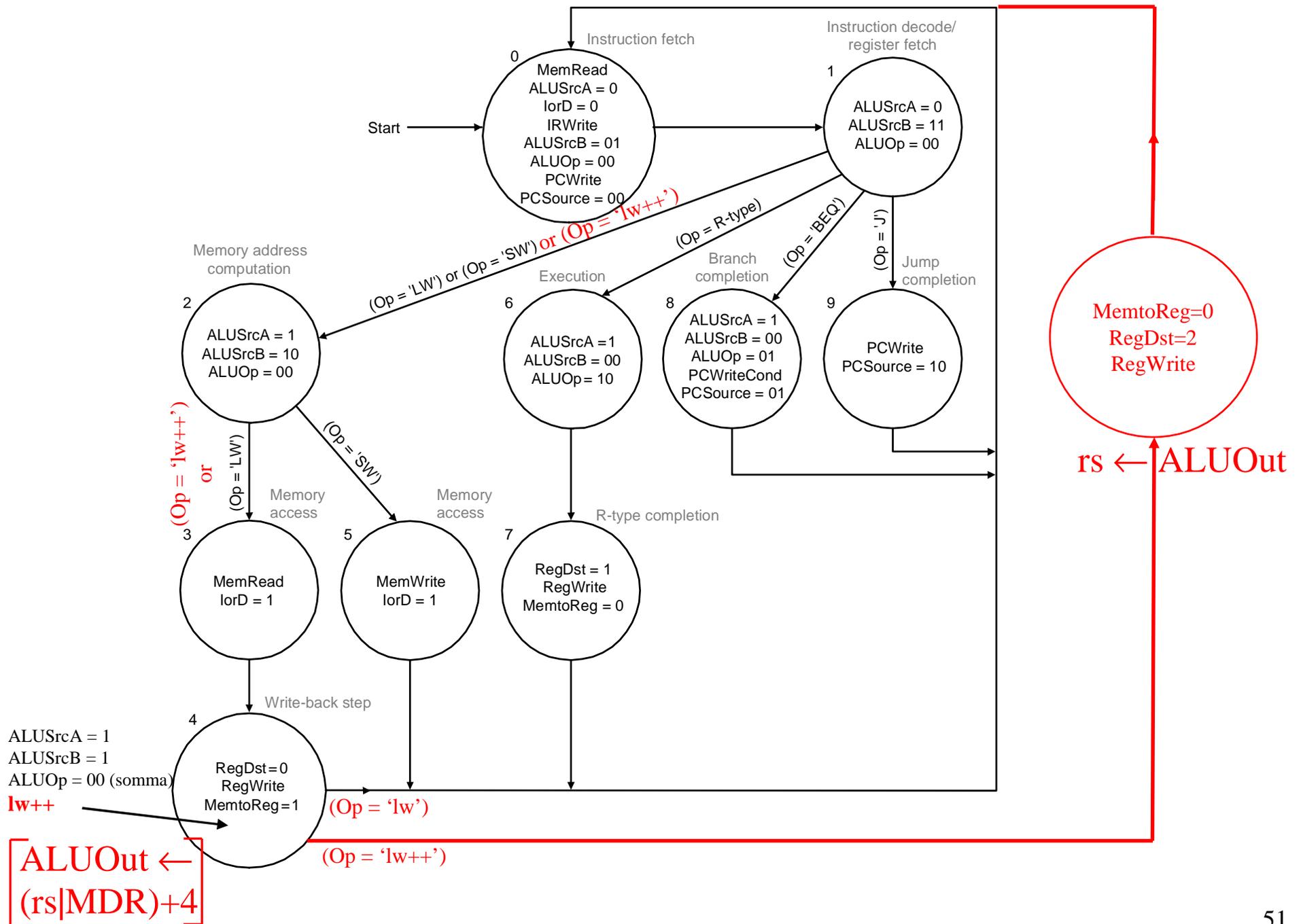
...oppure inseriamo un circuito che consenta di “anticipare” il dato!!!...

	lw rs, (rs)++
Stato 0	Memoria [Fetch]
Stato 1	Lettura Reg.File [A,B ← rs,rs]
Stato 2	ALU [rs+offset]
Stato 3	Memoria [accesso dato] : MDR ← M[rs]
Stato 4	Scrittura Reg. File [rs ← MDR] ALUout=A+4
Stato 5	Scrittura Reg. File [rs+4]

 Implementazione a 6 cicli [-3 rispetto a due istruzioni]

Modifica al DataPath: MUX in ingresso alla ALU per sommare MDR e controllo...





... le considerazioni fatte sono molto simili a quelle che riguarderanno la gestione delle criticità sui dati della pipeline...

... e in effetti, tutto ciò suggerisce l'idea di estendere il tentativo di parallelizzare le operazioni appartenenti ad istruzioni diverse...



CONCETTO DI PIPELINE ... !