Calcolatori Elettronici A
a.a. 2008/2009

CPU multiciclo: Esercizi

Massimiliano Giacomin

Due tipologie di esercizi standard

- 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)

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

Calcolo CPI e Prestazioni nei sistemi a singolo ciclo e multiciclo

1) Calcolo prestazioni nei sistemi a singolo ciclo

$CPI = 1$

$T_{clock} = \max\{T_a + \ldots T_k\}$

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

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

2) Calcolo CPI e prestazioni nei sistemi multi-ciclo

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

$CPI = f_1*CPI_1 + f_2*CPI_2 + \ldots + f_n*CPI_n$

$T_{clock} = \max\{T_1, \ldots, T_m\}$ [operazioni atomiche eseguite <u>in un ciclo di clock</u>]

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

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

$$\frac{T^1_{esecuzione}}{T^2_{esecuzione}} = \frac{CPI^1 * T^1_{clock}}{CPI^2 * T^2_{clock}}$$

Esercizio 1

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

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

Calcolo delle prestazioni nel caso a singolo ciclo (macchina M1)

CPI = 1

$\Rightarrow T_{esecuzione} = I * T_{clock}$ (I è il numero di istruzioni)

T_{clock} corrisponde all'istruzione "più lunga" che evidentemente è la MUL in virgola mobile (cfr. esercizio sul processore a singolo ciclo)

$T_{clock} = 2 + 1 + 8*2 + 1$ ns = 20 ns

fetch

lettura R.File

ALU

scrittura R.File

$T_{M1} = I * 20$ ns

Calcolo delle prestazioni nel caso multiciclo

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

T_{clock} = 2ns (operazione ALU oppure accesso in memoria)

CPI = somma pesata dei CPI singole istruzioni con le frequenze relative
= (vedi lucido seguente) 4,89

⇒ T_{M2} = I * 4.89 * 2 ns = I * 9.78 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

CPI = 0.31*5 + 0.21*4 + 0.22*4 + 0.05*3 + 0.07*3 + 0.07*7 +0.07*11
= 4.89

M3: in un ciclo di clock ho al più:

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

⇒ T_{clock} = 3ns

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

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 ns = I * 12.66 ns

Riepilogo

	CPI	T_{clock}	T_{esecuz}	$T_{istruzione\text{-}più\text{-}costosa}$
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

$$\frac{T_{M1} = I * 20 \text{ ns}}{T_{M2} = I * 9.78 \text{ ns}}$$

M2 oltre 2 volte più veloce rispetto a M1

$$\frac{T_{M3} = I * 12.66 \text{ ns}}{T_{M2} = I * 9.78 \text{ ns}}$$

M2 circa 1.3 volte più veloce rispetto a M3

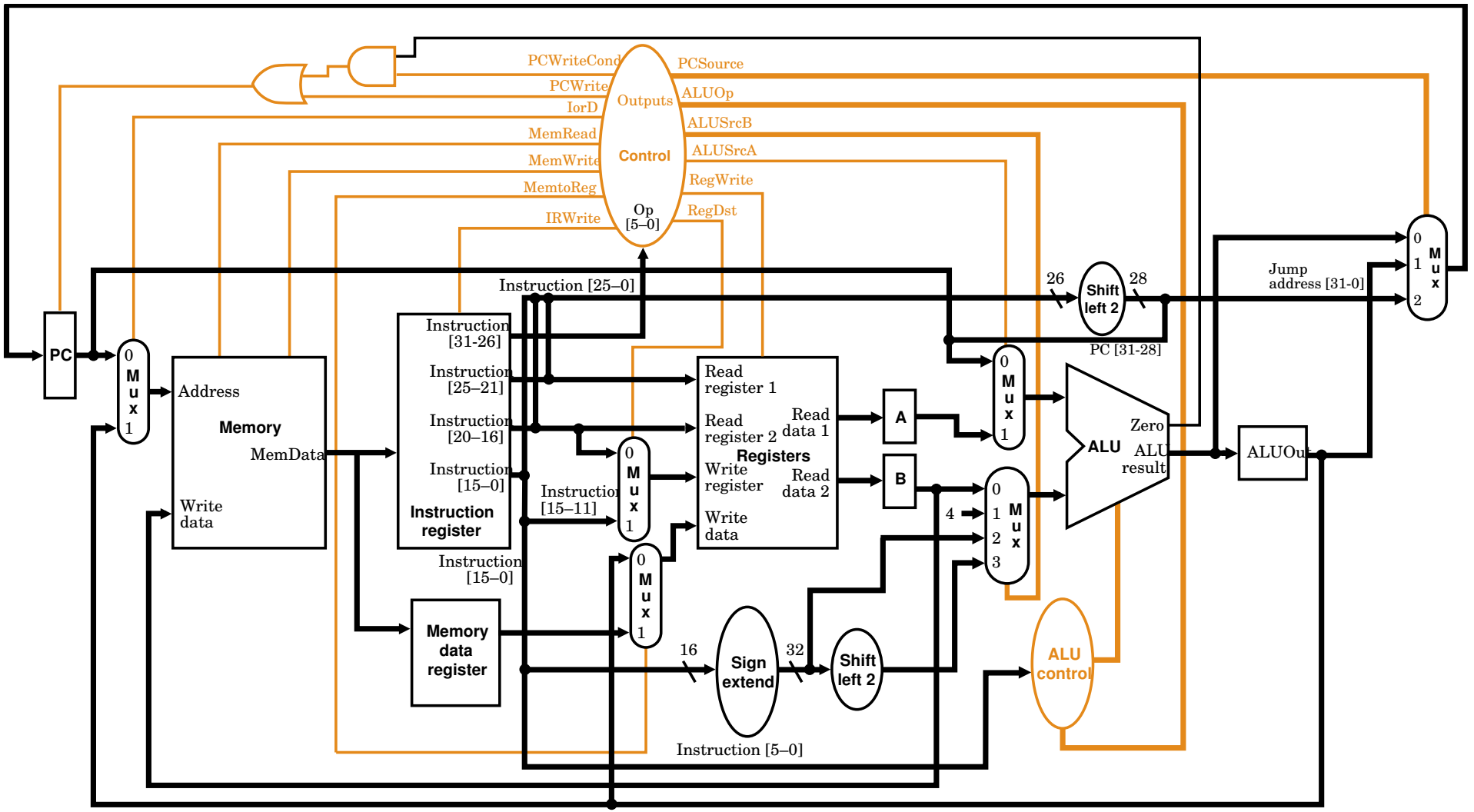
NB: la riduzione di CPI non paga rispetto all'aumento di T_{clock}

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?



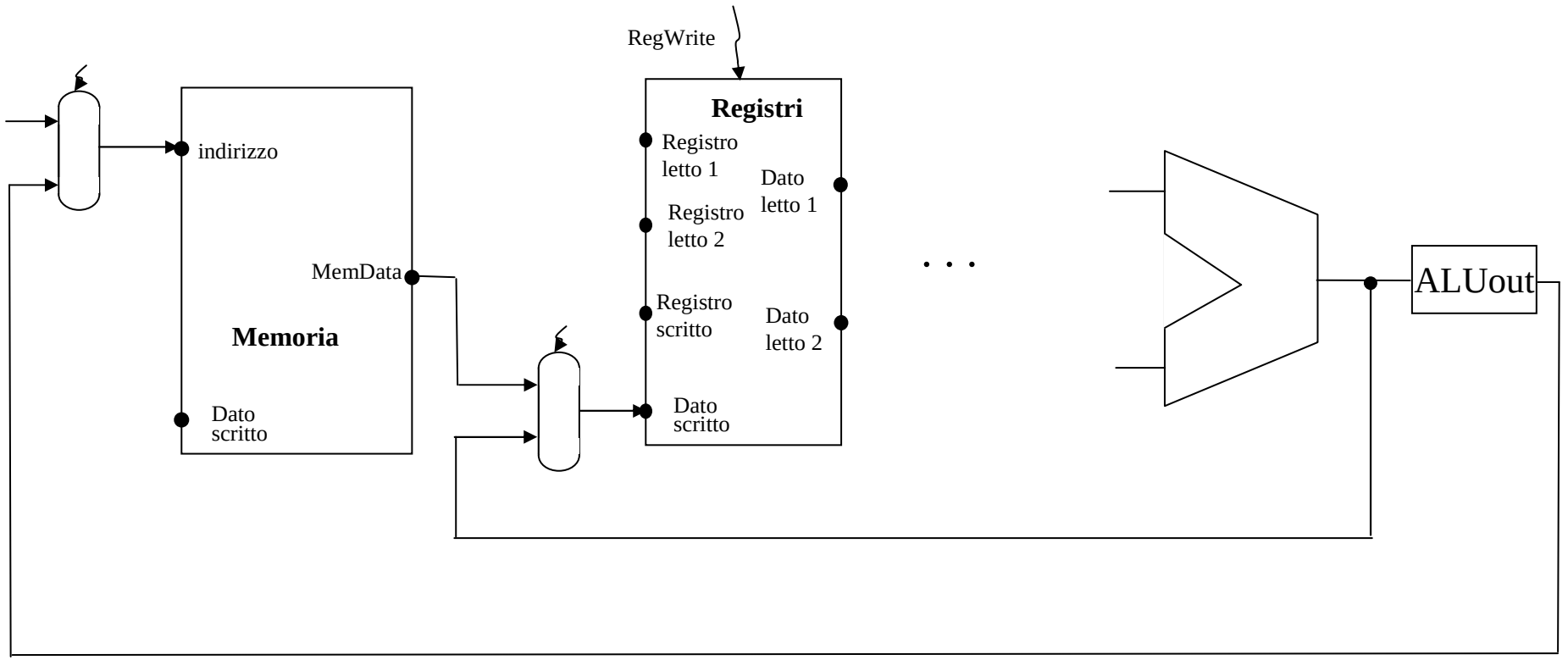
Si tratta di collegare “direttamente in serie”

- ALU a Register File (per scrivere in un registro il risultato di un’operazione)

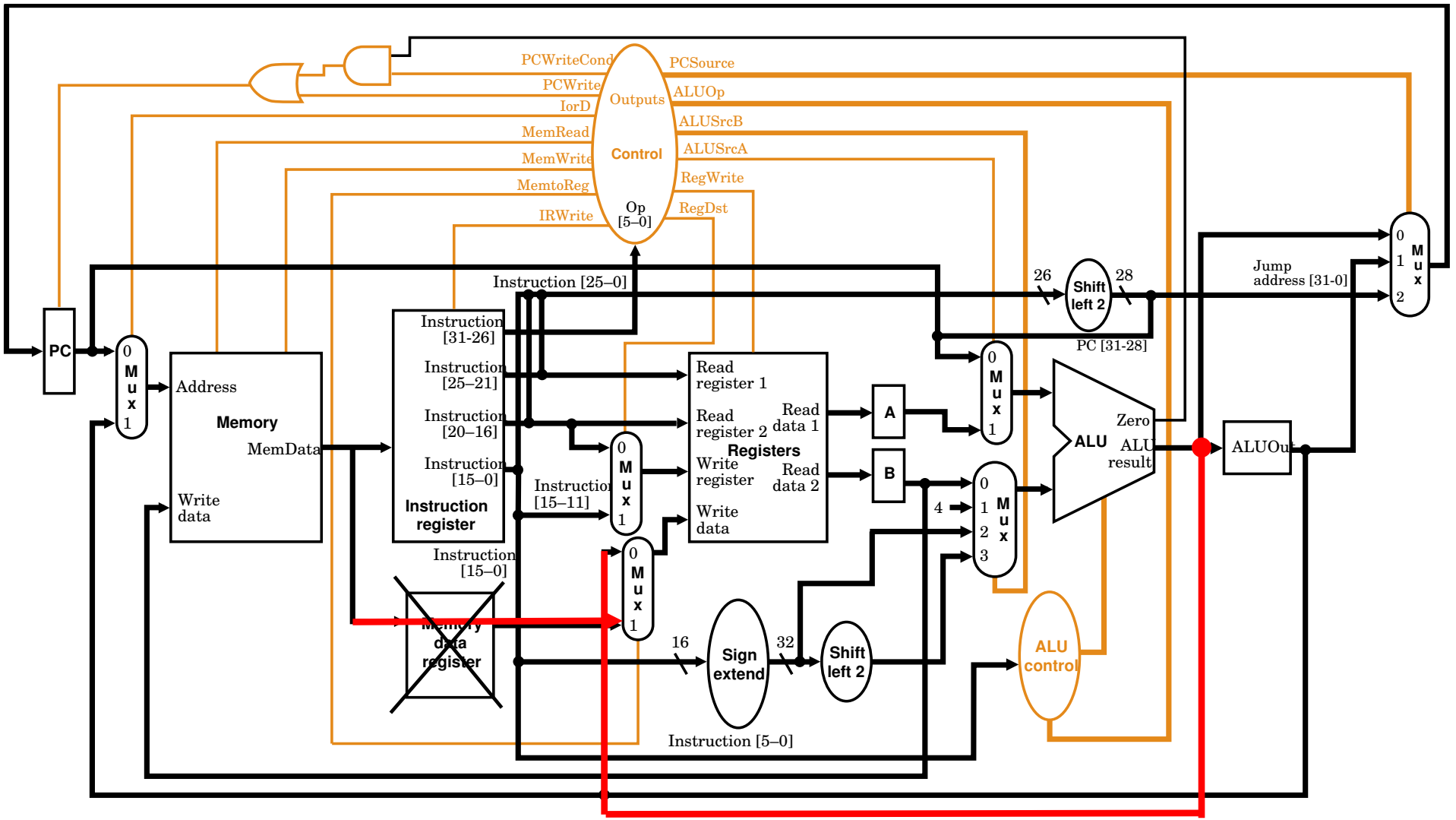
  $\Rightarrow$ linea di collegamento in uscita alla ALU prima del registro ALUout

- Memoria a Register File (per scrivere in un registro il valore letto da Mem)

  $\Rightarrow$ elimino il registro MDR collegando direttamente memoria a Reg.File



Modificando il Datapath completo risulta:



Esercizio 2: dall'appello 20 settembre 2006 [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%:

$f_{beq} + f_j =$ 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).

<u>Prestazioni nel caso di singolo ciclo</u>

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

= 8 ns

CPI = 1

⇒ $T_{SC} = I * 8$ ns (I è il numero di istruzioni)

<u>Prestazioni nel caso multiciclo</u>

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

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

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

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

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

$\cancel{1} * 8 < \cancel{1} * [f_{lw}*10 + f_{sw}*8 + f_{TIPO\text{-}R}*8 + f_{beq}*6 + f_j*6]$

… cerco di "raccogliere" per esprimere $f_{beq} + f_j = 0.1$

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

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

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

E adesso?

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

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

So che

- $f_{lw} + f_{sw} + f_{TIPO\text{-}R} + f_j + f_{beq} = 1$
- $f_{beq} + f_j = 0.1$

$\Rightarrow f_{lw} + f_{sw} + f_{TIPO\text{-}R} = 0.9$

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

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

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

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

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

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:

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

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

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

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

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

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

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

Esercizio 3

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} = #istruzioni * CPI * T_{clock}

T_{clock} = 2ns (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

CPI = 0.25*5 + 0.1*4 + 0.52*4 + 0.11*3 + 0.02*3 = 4.12

⇒ T_{M1} = I * 4.12 * 2 ns = I * 8.24 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} !!!

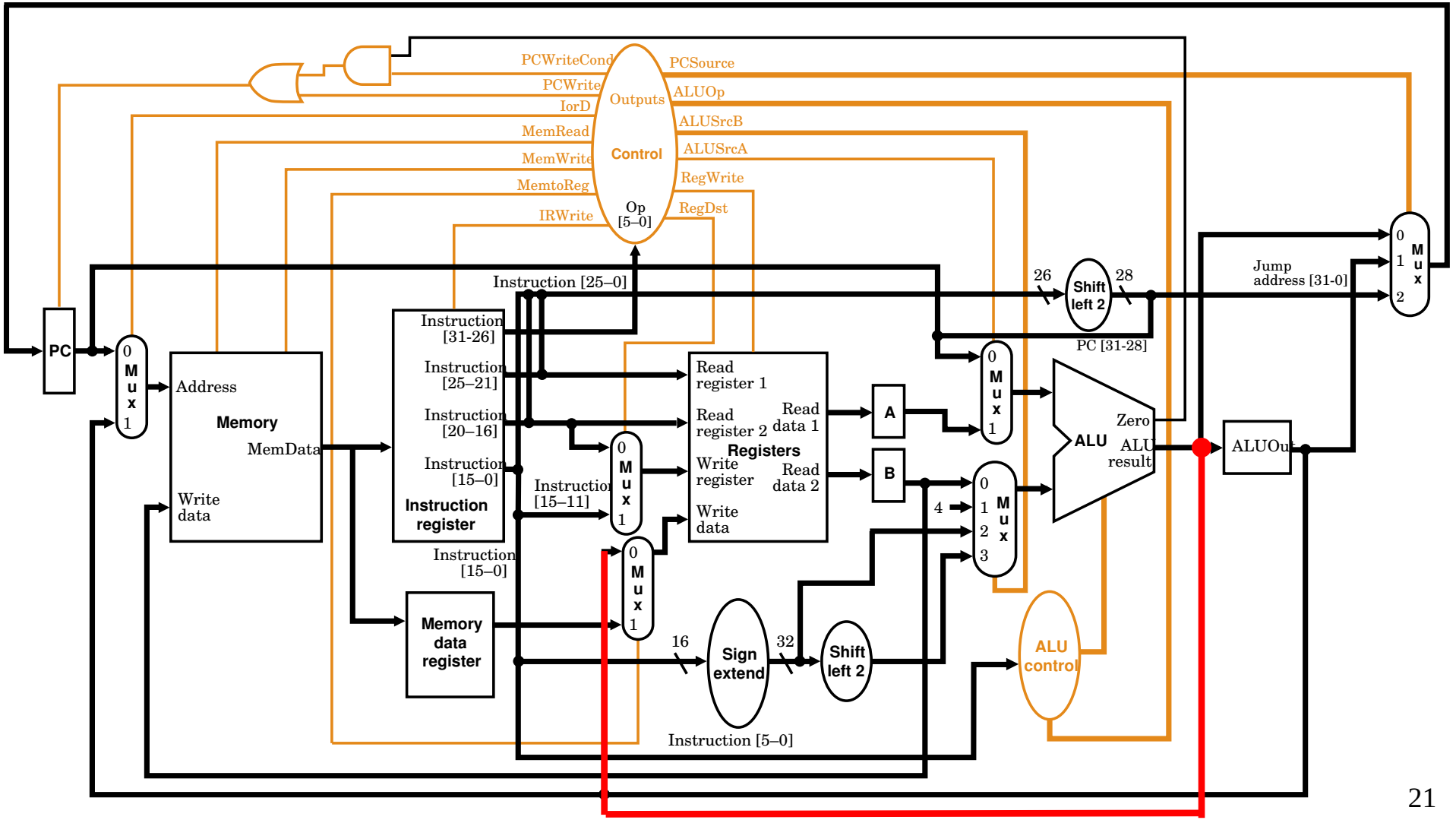
Istruzione	Fetch	Lettura Registri/Decode	ALU	Accesso Memoria	Scrittura Registri	CPI
lw	1	1	1	1	1	5
sw	1	1	1	1		4
Tipo-R	1	1	1		1	~~4~~ 3
beq	1	1	1			3
j	1	1			1	3

$$CPI = 0.25*5 + 0.1*4 + 0.52*3 + 0.11*3 + 0.02*3 = 3.6$$

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

$$\frac{T_{M1} = I * 8.24 \text{ ns}}{T_{M2} = I * 7.2 \text{ ns}}$$

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 4

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

j (rs) ; PC ← rs

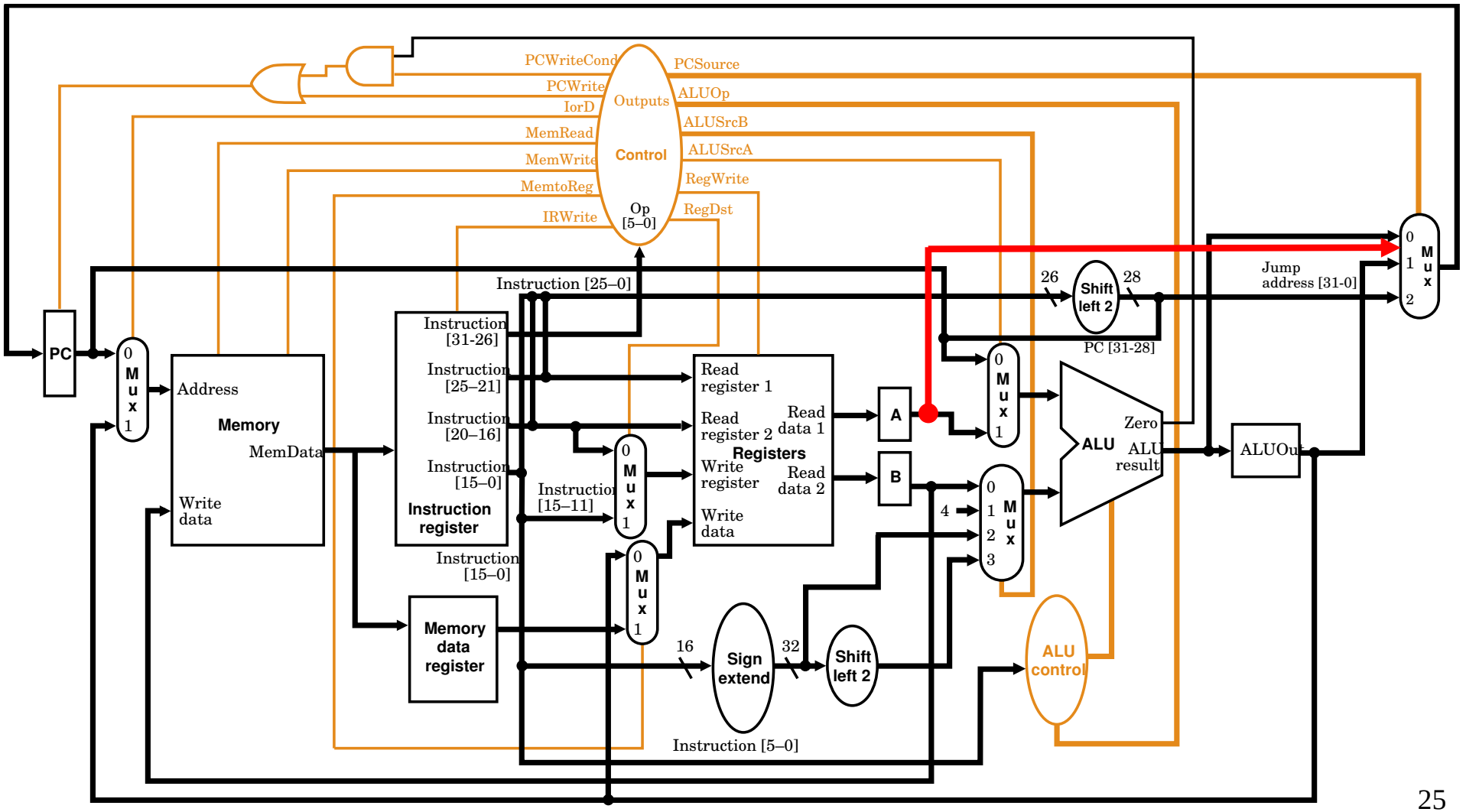
la quale esegue un salto incondizionato all'indirizzo specificato dal registro rs.

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_{clock} = 2ns, calcolare il tempo di esecuzione della nuova istruzione.

j (rs)

- ***Una prima soluzione***: modificare il datapath aggiungendo in ingresso al MUX che determina il valore da scrivere in PC il registro A [che contiene rs]
- Il controllo è modificato di conseguenza (esecuzione simile a j normale)



• ***Una seconda soluzione***: non modificare il datapath

Osservando il datapath, si vede che:

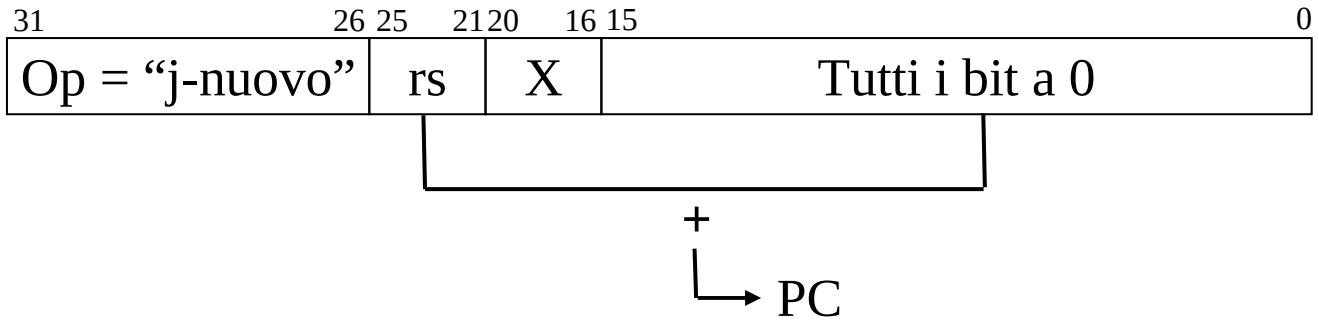
- è possibile forzare la scrittura di PC con l'uscita del MUX [ALUout| ALU| indirizzo j] ponendo PCWrite = 1

  ⇒ consideriamo ALU (è possibile mettere in serie ALU e PC)

- 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:

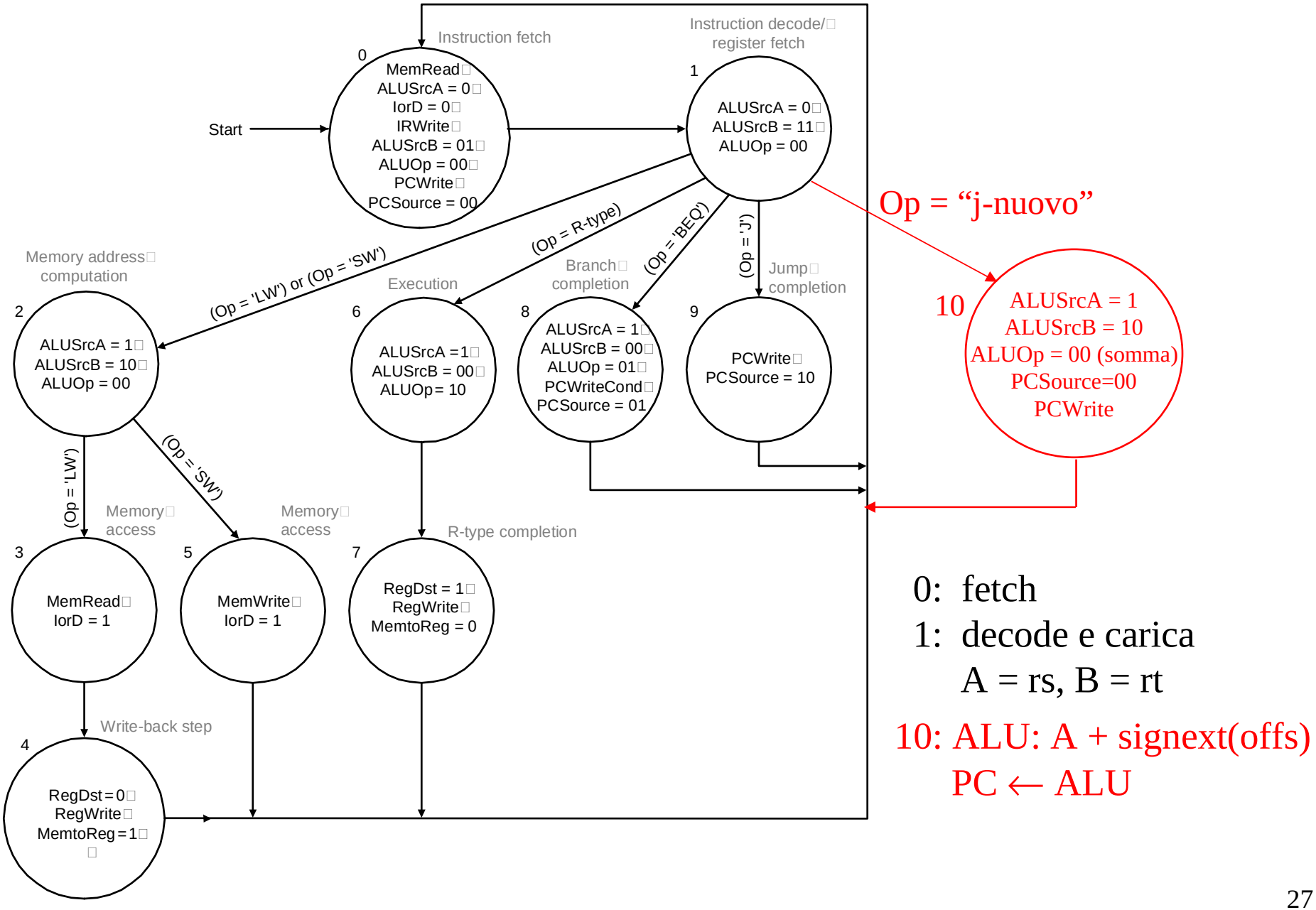
  rs + sign-extend(0) = rs

⇒ 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…



0: fetch

1: decode e carica
A = rs, B = rt

10: ALU: A + signext(offs)
PC ← ALU

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 → ALU → PC:

come nel caso di beq, non poniamo in serie più di un elemento "critico"

Tempo di esecuzione: 3 * 2 ns = 6 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.

//

Esercizio 5: dall'appello 13 luglio 2005 [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←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:

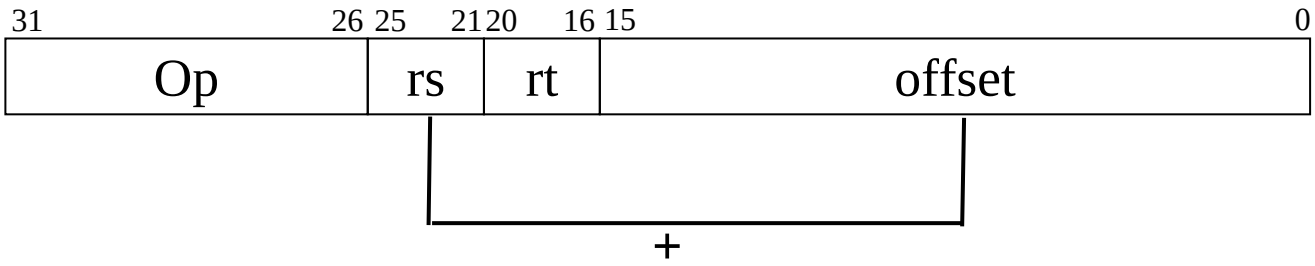
- riportare il formato della nuova istruzione macchina
- riportare, nella corrispondente figura, le modifiche necessarie al datapath
- estendere il diagramma degli stati per implementare la nuova istruzione

addmem rt, offset(rs)

Formato

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

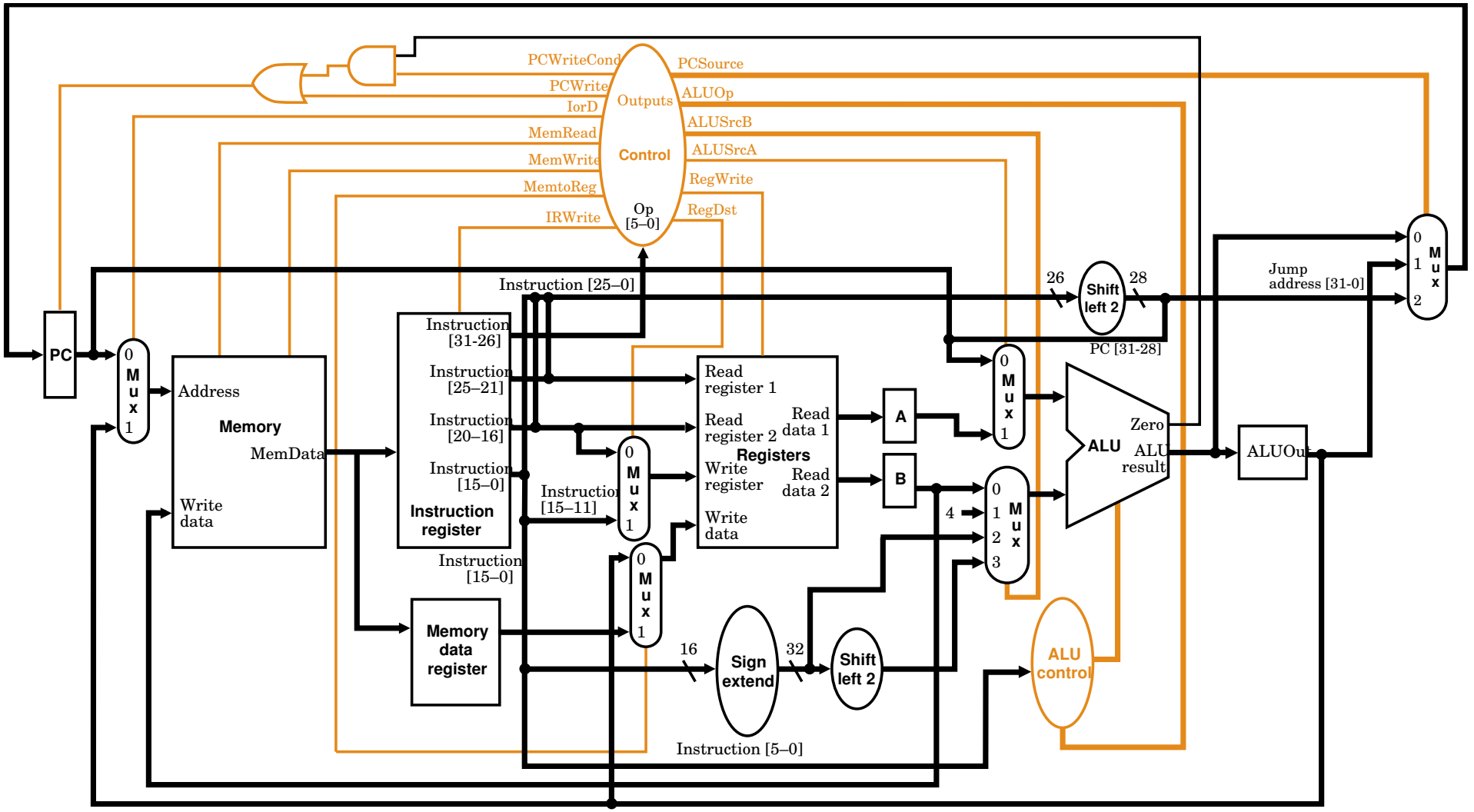
⇒ Sicuramente verrà usato il Tipo-I



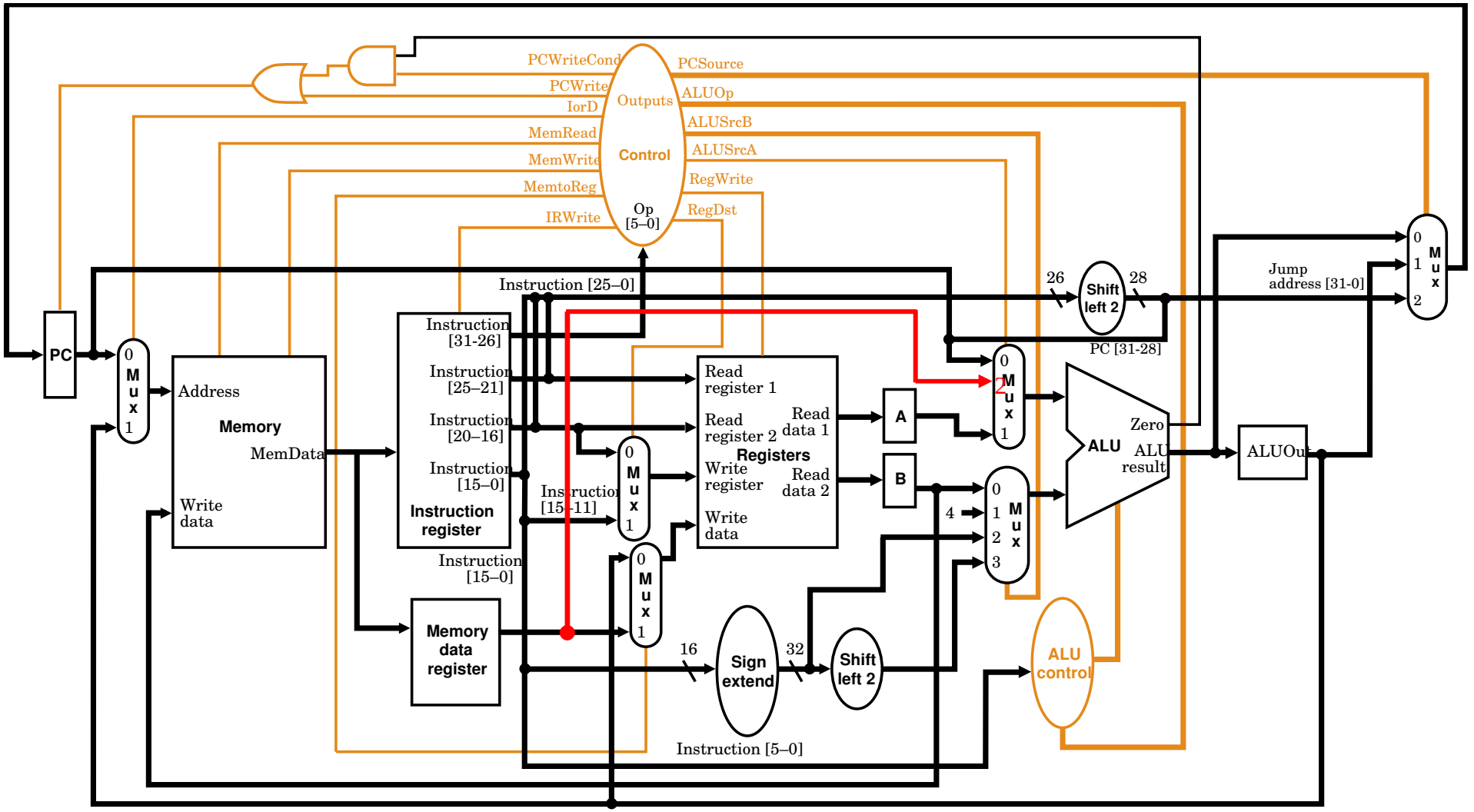
(cfr. anche lw)

Modifica al DataPath

addmem rt, offset(rs) : è una sorta di "lw + add"

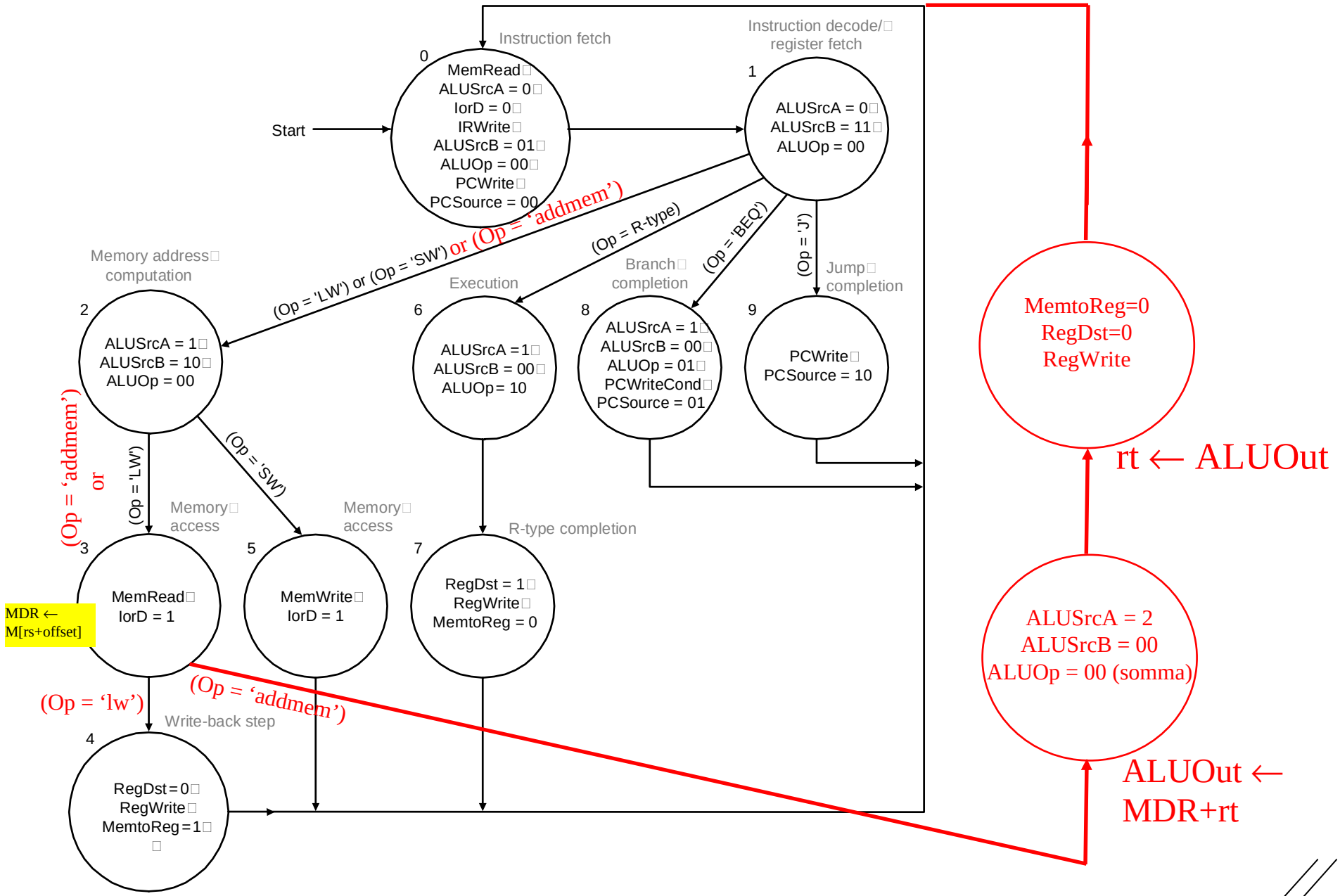


Modifica al DataPath *addmem rt, offset(rs) : è una sorta di "lw + add"*



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



Esercizio 6: dallo stesso appello [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

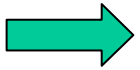
Si confrontino le prestazioni (in termini di rapporto tra tempi di esecuzione)
di un'ipotetica implementazione a singolo ciclo rispetto all'implementazione
multiciclo individuata al punto precedente.

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
	10 ns



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

Multiciclo

$T_{esecuzione}$ = #istruzioni * CPI * T_{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_{clock} = 2 ns

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

⇒ CPI = 0,2 * 5 + 0,2 * 4 + 0,3 * 4 + 0,1 * 3 + 0,1 * 3 + 0,1 * 6 = 4,2

QUINDI SI OTTIENE:

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

Confronto Prestazioni Singolo Ciclo vs. Multiciclo

$T_{singolo}$ = #istruzioni * 10

T_{multi} = #istruzioni * 8,4

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

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