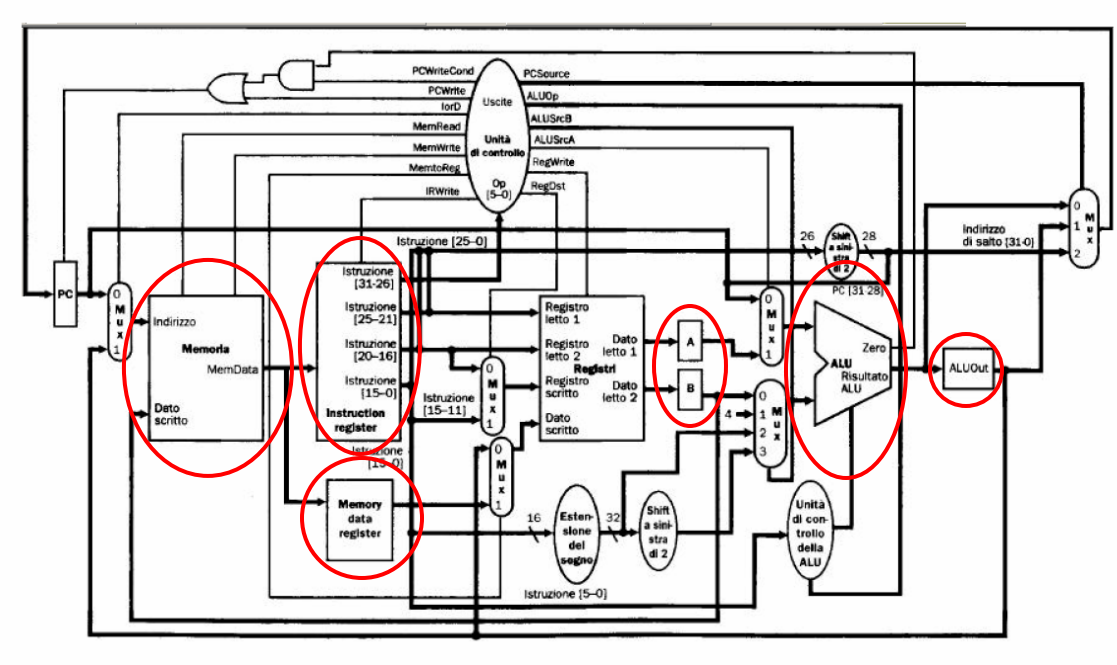
Esercitazione del 05/05/2005 - Soluzioni

Una CPU a ciclo singolo richiede un ciclo di clock di durata sufficiente a permettere la stabilizzazione del circuito nel caso dell'istruzione più complicata (con il cammino più lungo). Supponiamo che i cammini critici nelle varie sezioni del circuito siano quelli indicati in tabella e calcoliamo i tempi di esecuzione per i vari tipi di istruzione:

Tipo Istruzione	**Fetch Istruzione**	**decodifica/ lettura registri**	**Operazione ALU**	**Lettura/ scrittura Memoria dati**	**Scrittura registri**	**Cammino critico**
	2ns	1ns	2ns	2ns	1ns	
Load	2ns	1ns	2ns	2ns	1ns	9ns
Store	2ns	1ns	2ns	2ns		8ns
branch	2ns	1ns	2ns			5ns
Tipo R	2ns	1ns	2ns		1ns	6ns
J	2ns	1ns				3ns

Nel caso della CPU semplificata considerata il caso peggiore si verifica nelle operazioni di load da memoria, 9ns. Ne segue che la CPU a ciclo singolo, per le istruzioni più veloci, resterà per buona parte del tempo inattiva in attesa che termini il ciclo di clock.
In aggiunta si può notare che alcuni componenti simili vengono spesso usati in tempi diversi come ad esempio l'operazione di somma di 4 compiuta dal sommatore degli indirizzi e le operazioni ALU. Ne segue che parte del circuito resta per una parte del tempo inattivo.
Si può allora pensare di riorganizzare il circuito della CPU in modo da realizzare in un singolo ciclo di clock solo una parte dell'esecuzione di un istruzione con l'intento di mantenere il ciclo di clock breve e utilizzare intensivamente (o equivalentemente semplificare il circuito) ogni parte dell'hardware. Un istruzione quindi sarà eseguita nel caso di una **CPU multi-ciclo** in più cicli di clock. La divisione delle operazioni da eseguire in ogni singolo ciclo di clock andrà fatta cercando un buon compromesso tra larghezza del ciclo di clock e massimo utilizzo dei singoli componenti della CPU.
Durante l'esecuzione di una singola istruzione avremo bisogno di aree di memoria temporanee dove memorizzare i calcoli parziali, una sorta di registri di lavoro nascosti, non accessibili al programmatore. Alcuni di questi dovranno memorizzare informazioni utilizzate per più cicli di clock (come l'**Instruction Register** che memorizzerà l'opcode dell'istruzione in esecuzione**)**; l'unità di controllo in questo caso dovrà controllare con una opportuna linea di comando quando aggiornare il dato contenuto. Altri registri invece memorizzeranno informazioni utilizzate solo al ciclo di clock immediatamente successivo. Per questi ultimi non sarà necessario un segnale di scrittura esplicito in quanto potranno essere riscritti ad ogni passo, eventualmente con dati fittizzi.

Consideriamo quindi la seguente **CPU multi-ciclo** semplificata:



Confrontandola con la CPU a singolo ciclo si notano le seguenti differenze:

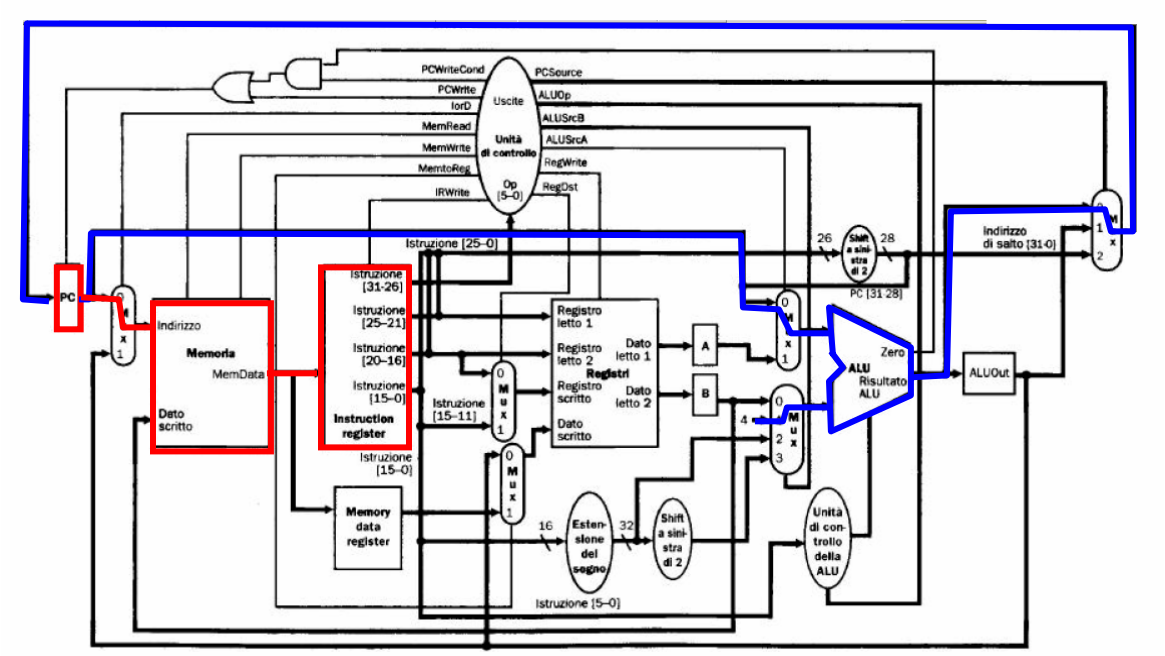
1. sono presenti i registri temporanei A, B, ALUOut e MDR che saranno riscritti ad ogni ciclo di clock.
2. è presente il registro temporaneo IR che memorizza il codice dell'istruzione da eseguire. Poiché il contenuto andrà riscritto solo durante la fase di fetch è necessaria una linea di controllo apposita.
3. è presente una sola memoria condivisa sia per le istruzioni e per i dati. Ciò deriva dalla considerazione che il fetch dell'istruzione e la lettura/scrittura nella memoria dati avvengono in tempi diversi e quindi possono essere gestite in due cicli di clock differenti.
4. manca il sommatore "+4" degli indirizzi ed il sommatore per il calcolo degli indirizzi di destinazione dei salti relativi. Le corrispondenti funzioni possono essere svolte dalla ALU ridisegnando le connessioni e aumentando il numero di ingressi dei multiplexer.
5. L'unità di controllo è più complessa soprattutto a causa del maggior numero di linee di controllo necessarie per i multiplexer che devono realizzare più percorsi di trasferimento dati.
6. sono presenti un maggior numero di collegamenti tra i singoli componenti.

Consideriamo l'istruzione di tipo R: **add $10, $9 , $8** (*add rd, rs, rt*) memorizzata all'indirizzo 0x00400000. Supponiamo di avere un ciclo di clock di 2ns. I primi due cicli di clock sono identici per tutte le istruzioni in quanto l'unità di controllo impiega due cicli per decodificare l'istruzione.

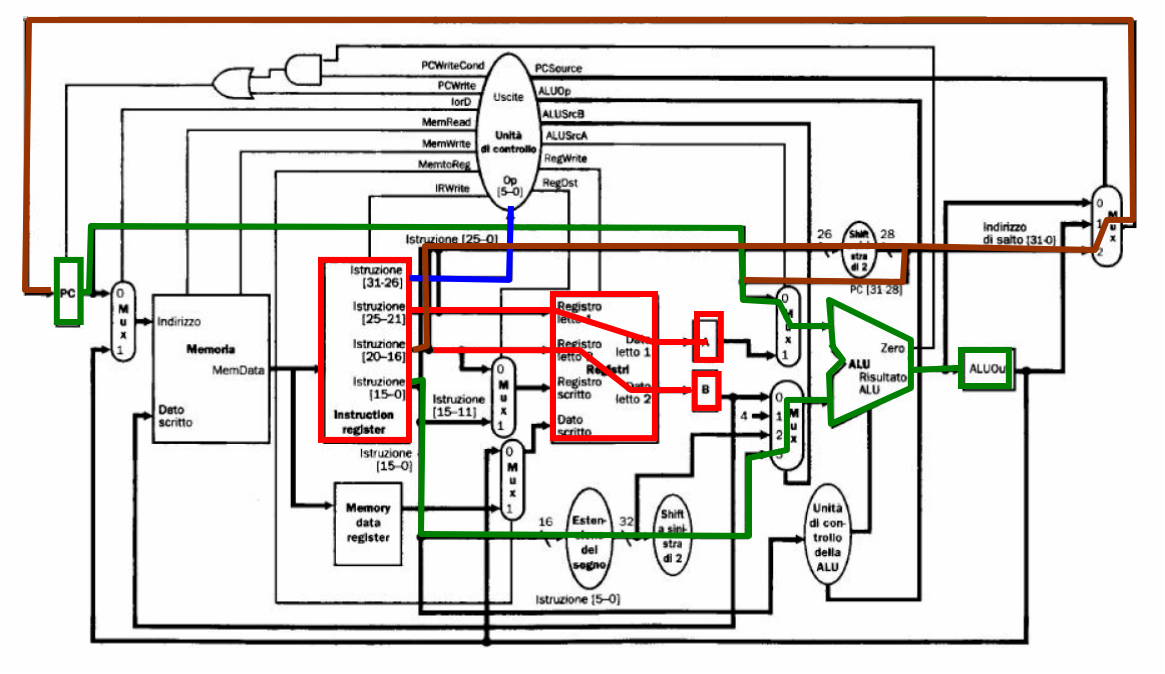
Durante il **primo ciclo** di clock si realizza la fase di **fetch.** Vengono eseguite le seguenti operazioni in parallelo:

- Il **Program Counter** (PC) viene usato per leggere l'istruzione da eseguire. L'istruzione viene memorizzata nell'**Istruction Register** (IR).
- Usando la ALU viene calcolato l'indirizzo successivo sommando a **PC** la costante **4** e memorizzando il risultato direttamente in **PC.**

Entrambe le operazioni saranno eseguite all'interno del ciclo di clock poiché richiedono, secondo quanto ipotizzato nella tabella precedente, al più 2ns.



Nel **secondo ciclo** di clock si realizza la fase di **decodifica** dell'istruzione e si preparano alcuni dati eventualmente utilizzati nei restanti cicli di clock:



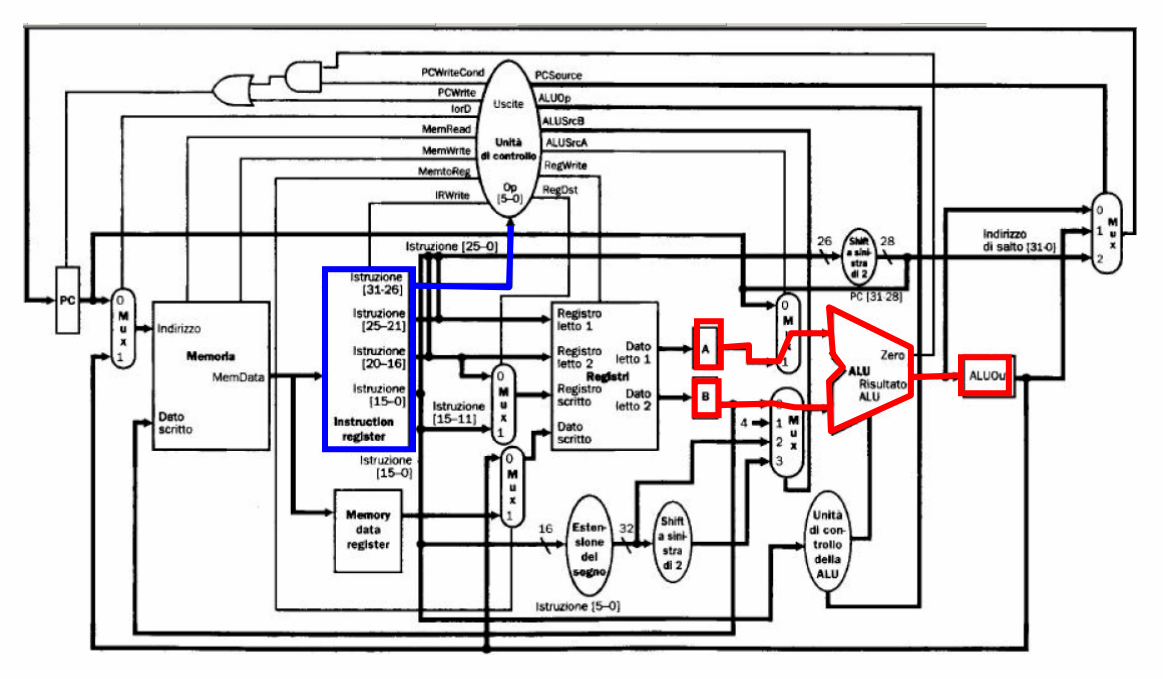
Vengono eseguite le seguenti operazioni in parallelo:

- Vengono letti e memorizzati in **A** e **B** i registri puntati dai bit nell'IR corrispondenti a **rs** ed **rt**. Ancora l'istruzione non è stata decodificata e quindi non si sa se effettivamente nella trama dell'istruzione è presente il riferimento ai registri. Leggere comunque adesso i registri non è dannoso ed accelera le istruzioni che richiedono tutti o parte dei dati estratti, come la **add** che stiamo considerando.
- Usando la **ALU** viene calcolato l'indirizzo di un ipotetico salto relativo sommando al **PC** il dato immediato estratto dall'**IR** in posizione opportuna. L'indirizzo calcolato viene memorizzato in **ALUOut**. Nel caso dell'istruzione **add** questa operazione risulta inutile.
- Usando parte del **PC** ed un opportuna parte dell'**IR** viene calcolato l'indirizzo di destinazione di una istruzione di salto incondizionato. Nel caso dell'istruzione **add** anche questa operazione risulta inutile.

Tutte le operazioni saranno eseguite all'interno del ciclo di clock poiché richiedono, secondo quanto ipotizzato nella tabella precedente, al più 2ns.

Nel **terzo ciclo** di clock l'unità di controllo è in grado di personalizzare l'esecuzione a seconda del tipo di istruzione in esecuzione.
Nel caso dell'istruzione di tipo R **add rt, rs ,rt** considerata l'esecuzione procede come segue:

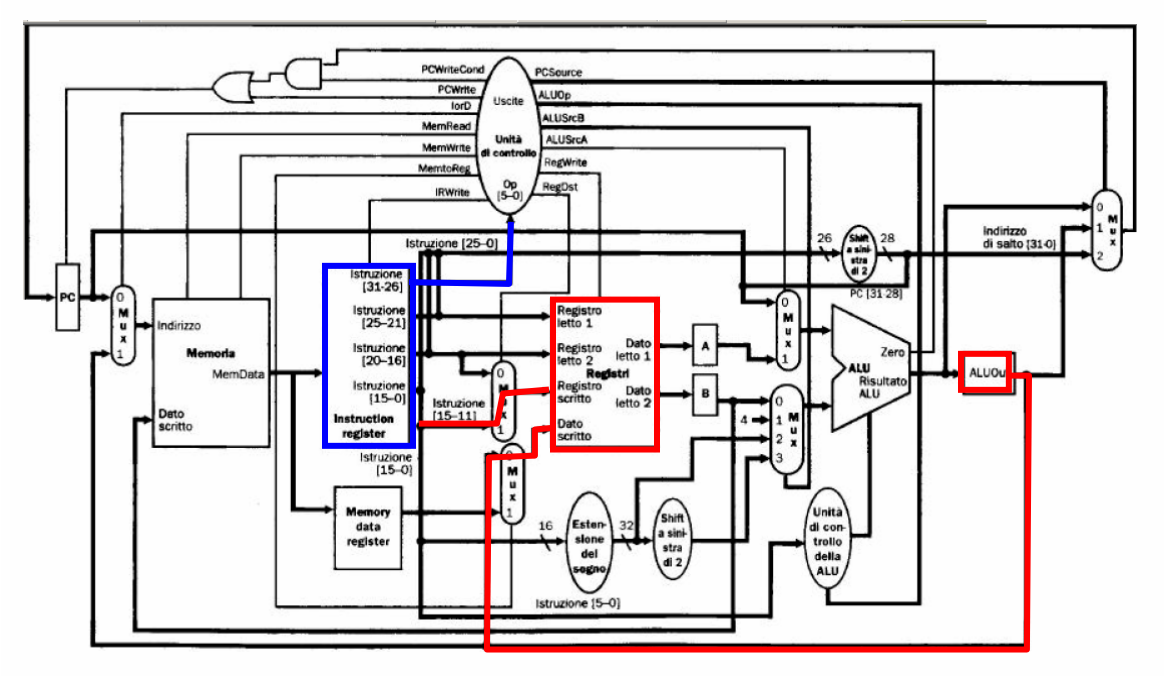
- Usando la **ALU** viene calcolato il risultato della somma dei registri **A** e **B** contenenti i dati rei registri **rs** ed **rt** ed il risultato è memorizzato in **ALUOut**



L'operazione compiuta dalla ALU viene eseguita in al più 2ns. Sarebbe anche possibile circuitalmente memorizzare il dato calcolato direttamente nel **register file.** Questo però richiederebbe in tutto un tempo almeno di 3ns che costringerebbe ad adottare un ciclo di clock più lungo.

Nel **quarto ciclo** di clock, nel caso di un istruzione di tipo R, viene eseguita la scrittura nei registri:

- il dato presente in **ALUOut,** risultato della computazione realizzata dalla ALU, viene scritto nel registro indicizzato dai bit 15-11 in **IR** indicanti il registro **rd**.



L’operazione viene eseguita secondo la tabella precedente in al più 1ns. La CPU però attende la fine del ciclo di clock, 2ns, prima di caricare la nuova istruzione. Il tempo totale di esecuzione quindi per le istruzioni di tipo R sarà di 4 cicli di clock, cioè 8ns.

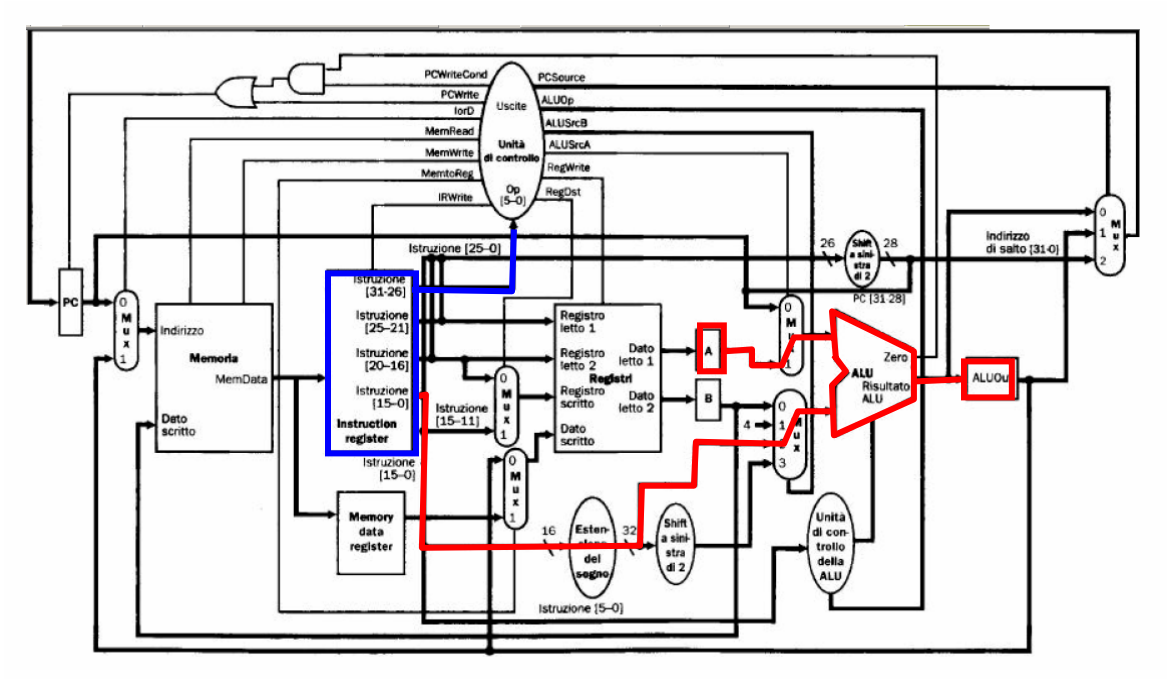
Consideriamo ora l'istruzione di tipo load: **lw $9, 3($8)** (*lw rt, 3(rs)*) memorizzata all'indirizzo 0x00400000:

I **primi due cicli di clock** vengono svolti in maniera analoga alle istruzioni di tipo R:

- viene incrementato il PC di 4
- viene caricato IR con l'istruzione da eseguire
- vengono caricati i valori dei registri **rs** ed **rt** in **A** e **B**
- Viene calcolato e memorizzato in **ALUOut** l'indirizzo di destinazione di un eventuale salto condizionato. Viene preparato l'indirizzo per un eventuale salto incondizionato. Questo dato non vengono usati.

Nel **terzo ciclo** di clock l'unità di controllo è in grado di personalizzare l'esecuzione a seconda del tipo di istruzione in esecuzione. Nel caso dell'istruzione di tipo load considerata l'esecuzione procede come segue:

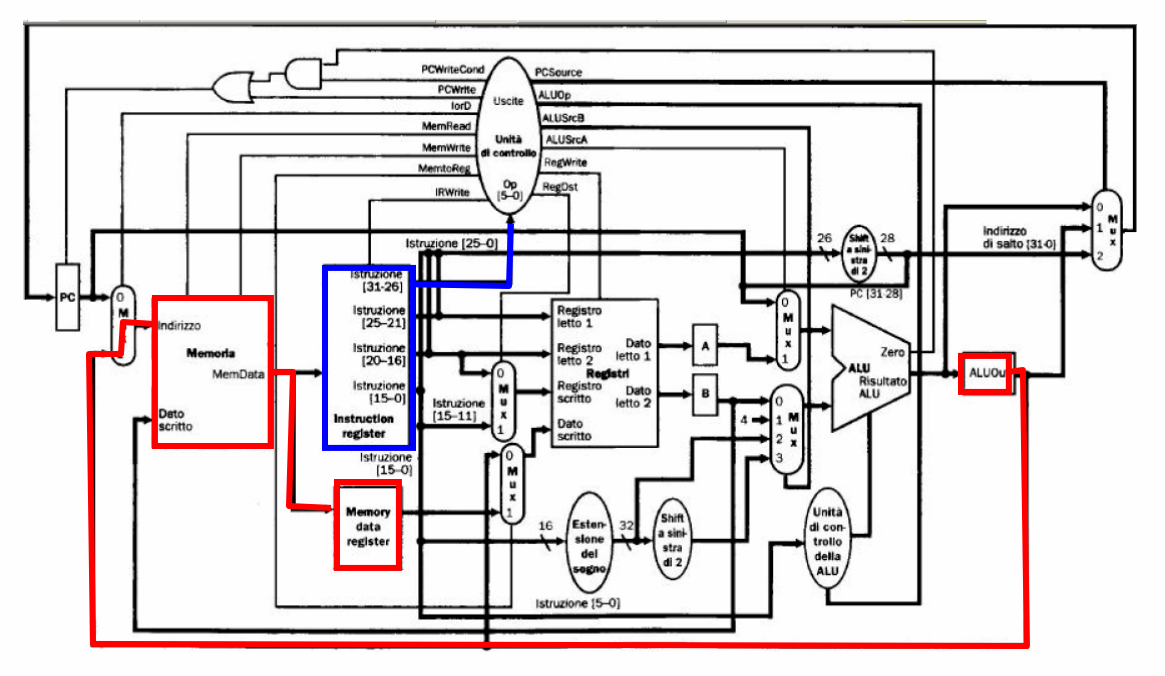
- Usando la **ALU** viene calcolato l'indirizzo della cella di memoria da leggere, cioè **rs** + **offset**. L'offset viene letto dai bit 15-0 dell'**IR**. L'indirizzo viene memorizzato in **ALUOut.**



L'operazione compiuta dalla ALU viene eseguita in al più 2ns, che esaurisce il tempo del ciclo di clock a disposizione.

Nel **quarto ciclo** di clock dell'istruzione di tipo load considerata viene letto il dato dalla memoria:

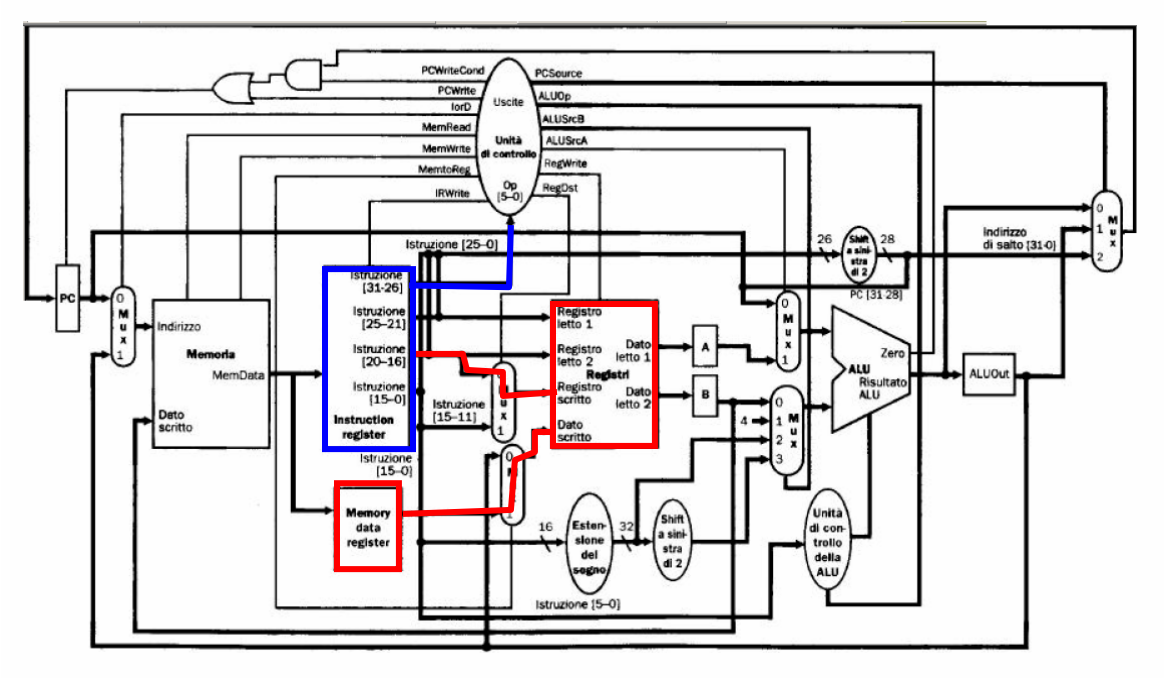
- Usando l'indirizzo contenuto in **ALUOut** viene letto il dato dalla memoria e memorizzato nel registro temporaneo **Memory Data Register** (MDR)**.**



Valgono le stesse considerazioni fatte per il terzo ciclo di clock delle istruzioni di tipo R. L'operazione viene eseguita in al più 2ns, che esaurisce il tempo del ciclo di clock a disposizione.

Nel **quinto ciclo** di clock di un'istruzione di tipo load viene scritto il dato letto dalla memoria nel **register file**:

- Il contenuto in **MDR** viene scritto nel registro **rt** indicato dai bit 20-16 dell'**IR**.



Le istruzioni di tipo load richiedono quindi 5 cicli di clock per essere eseguite, cioè 10ns.

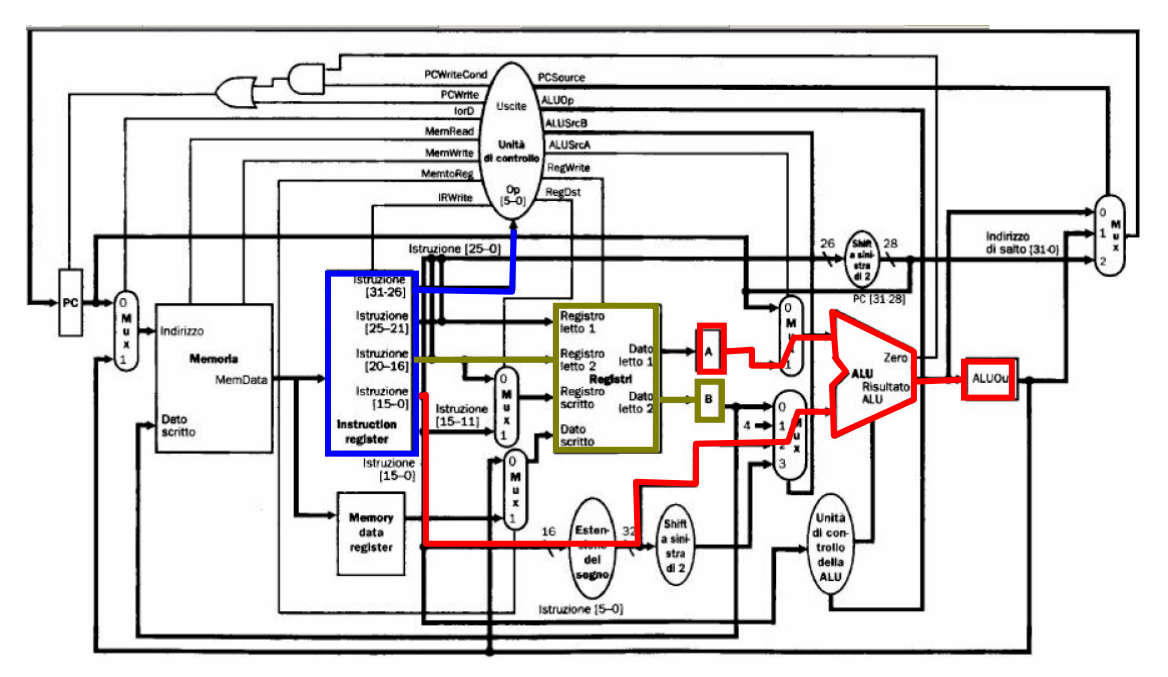
Analogo discorso si può fare con le istruzione di tipo store, esempio **sw $9, 3($8)** (*sw rt, offset(rs)*) memorizzata all'indirizzo 0x00400000:

I **primi due cicli di clock** vengono svolti in maniera analoga alle istruzioni di tipo R:

- viene incrementato il PC di 4
- viene caricato IR con l'istruzione da eseguire
- vengono caricati i valori dei registri **rs** ed **rt** in **A** e **B**
- Viene calcolato e memorizzato in **ALUOut** l'indirizzo di destinazione di un eventuale salto condizionato. Viene preparato l'indirizzo per un eventuale salto incondizionato. Questo dato non vengono usati.

Nel **terzo ciclo** di clock di un istruzione di tipo store l'esecuzione procede come segue:

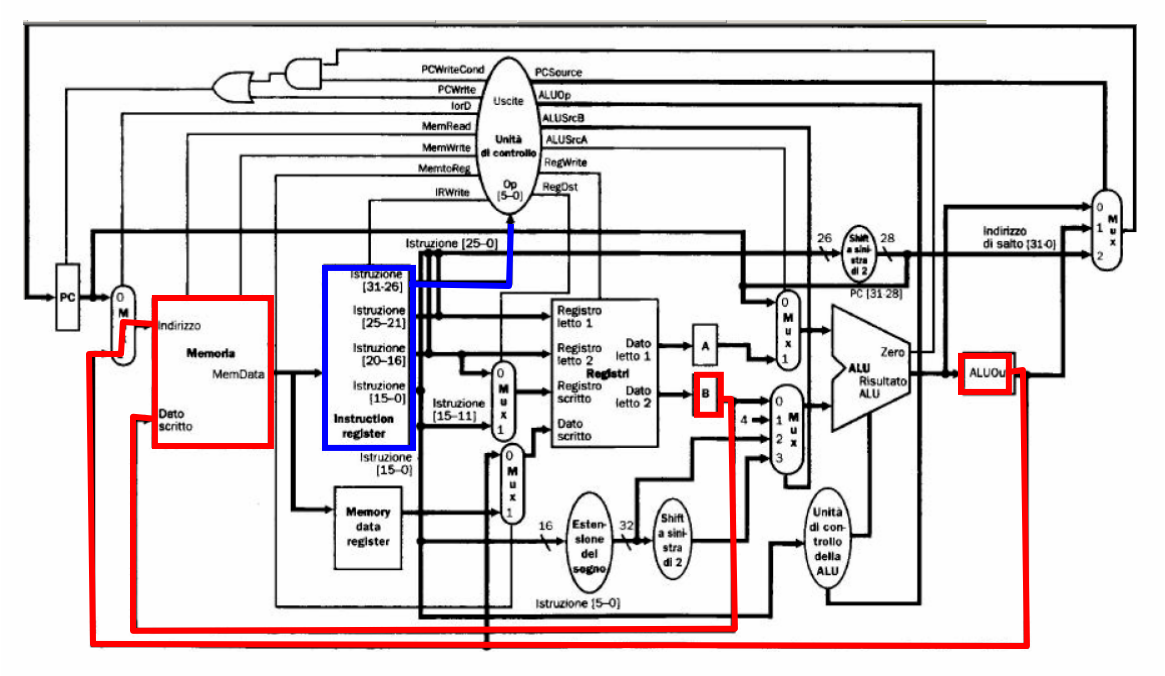
- Usando la **ALU** viene calcolato l'indirizzo della cella di memoria da scrivere, cioè **rs** + **offset**. L'offset viene letto dai bit 15-0 dell'**IR**. L'indirizzo viene memorizzato in **ALUOut.**
- Viene riletto il registro rt e riscritto in B. Questo passaggio è necessario in quanto B non ha una linea di controllo per la riscrittura e quindi viene riscritto ad ogni ciclo di clock.



Tutte le operazioni terminano in al più 2ns, il che esaurisce il tempo del ciclo di clock a disposizione.

Nel **quarto ciclo** di clock dell'istruzione di tipo store considerata viene scritto il dato nella memoria:

- Usando l'indirizzo contenuto in **ALUOut** viene scritto nella memoria il dato presente in **B.**



Le istruzioni di tipo store richiedono quindi 4 cicli di clock per essere eseguite, cioè 8ns.

Un altra sottoclasse di istruzioni di tipo I sono le istruzioni di salto relativi:

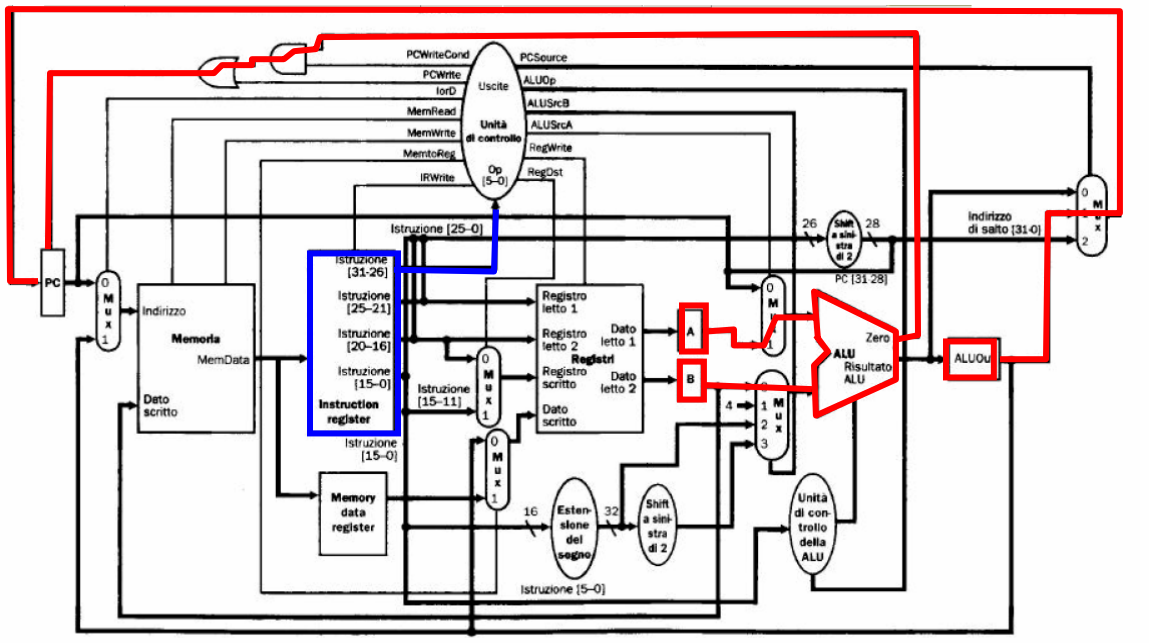
Consideriamo l'istruzione di tipo salto condizionato relativo: **beq $8, $9 , 12** (*beq rs, rt, offset*) memorizzata all'indirizzo 0x00400000:

I **primi due cicli di clock** vengono svolti in maniera analoga alle istruzioni di tipo R:

- viene incrementato il PC di 4
- viene caricato IR con l'istruzione da eseguire
- vengono caricati i valori dei registri **rs** ed **rt** in **A** e **B.**
- Viene calcolato e memorizzato in **ALUOut** l'indirizzo di destinazione di un eventuale salto condizionato.
- Viene preparato l'indirizzo per un eventuale salto incondizionato. Questo dato non viene usato.

Nel **terzo ciclo** di clock di un istruzione di tipo branch l'esecuzione procede come segue:

- Usando la **ALU** vengono confrontati i registri **rs** ed **rt** memorizzati in A e B. Il risultato del confronto è presente sul segnale **zero** della **ALU**. Se **rs** ed **rt** sono uguali (**zero = 1**) l'indirizzo calcolato nel ciclo precedente presente in **ALUOut** viene scritto in **PC**, altrimenti (**zero=0**) **PC** viene lasciato inalterato.

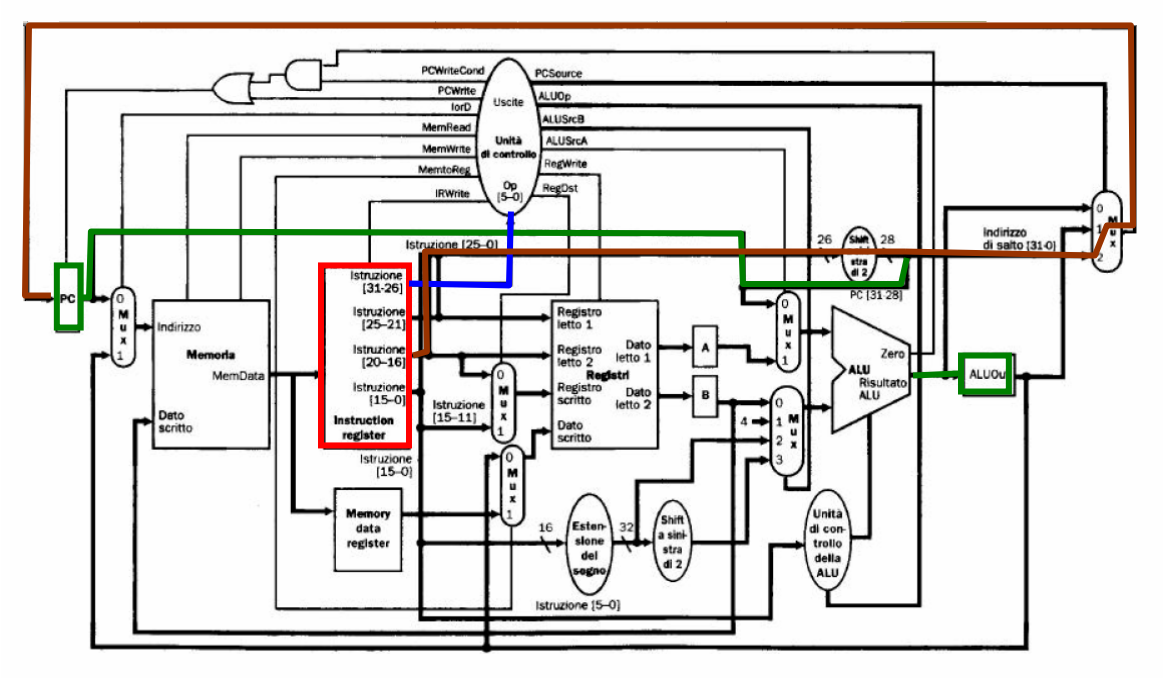


E' da notare che ALUOut viene riscritto alla fine del terzo ciclo di clock con la differenza calcolata dalla ALU per effettuare il confronto. Questo però non influenza il dato eventualmente scritto in PC se si implementano i registri temporanei con F/F di tipo Master/Slave.
Le operazioni di branch richiedono 3 cicli di clock, pari a 6ns.

Consideriamo l'istruzione di tipo **J**: **j 0x00400010** (*j address*) memorizzata all'indirizzo 0x00400000.

I **primi due cicli di clock** vengono svolti in maniera analoga alle istruzioni di tipo R:

- viene incrementato il PC di 4
- viene caricato IR con l'istruzione da eseguire
- vengono caricati i valori dei registri **rs** ed **rt** in **A** e **B.** Questo dato non viene usato.
- Viene calcolato e memorizzato in **ALUOut** l'indirizzo di destinazione di un eventuale salto condizionato. Questo dato non viene usato.
- Viene preparato l'indirizzo per un eventuale salto incondizionato.



Nel **terzo ciclo** di clock l'unità di controllo è in grado di riconoscere l'istruzione di salto e sistemare le linee di controllo in modo da riscrivere il **PC** con l'indirizzo estratto dall'**IR** e completato con i bit 31-28 del **PC** stesso.

In tutto quindi un operazione di salto condizionato richiede 3 cicli di clock, pari a 6ns.

Riepilogando:

Tipo Istruzione	**Cammino critico**	**Tempo di esecuzione su CPU singolo ciclo**	**Tempo di esecuzione su CPU multi ciclo**
Tipo R	6ns	9ns	8ns
Load	10ns		10ns
Store	8ns		8ns
branch	5ns		6ns
J	3ns		6ns

Si può verificare una CPU multi-ciclo richiede più tempo per l'esecuzione di istruzioni di tipo load sia rispetto al caso ideale (cammino critico) sia rispetto alla CPU a ciclo singolo. Questo dato però è compensato dal minore tempo di esecuzione richiesto per tutte le altre operazioni. Se avessimo considerato anche le operazioni in virgola mobile che richiedono molto più tempo di una istruzione load, il guadagno sarebbe stato più netto.

Le maggiori difficoltà nell'implementare CPU multi ciclo si verificano nella progettazione dell'unità di controllo, concettualmente più complicata rispetto all'unità della CPU a ciclo songolo. In quest'ultimo caso infatti l'unità di controllo si riduce ad una funzione booleana senza memoria. Nel caso di CPU multi ciclo il valore delle uscite, a parità di segnali di ingresso deve variare in funzione del passo di esecuzione a cui è arrivata l'esecuzione dell'istruzione. In pratica l'unità di controllo deve *ricordare* in che passo di calcolo si ritrova.

Il modo più comodo per sviluppare questo è considerare l'unità di controllo come un'automa a stati finiti: