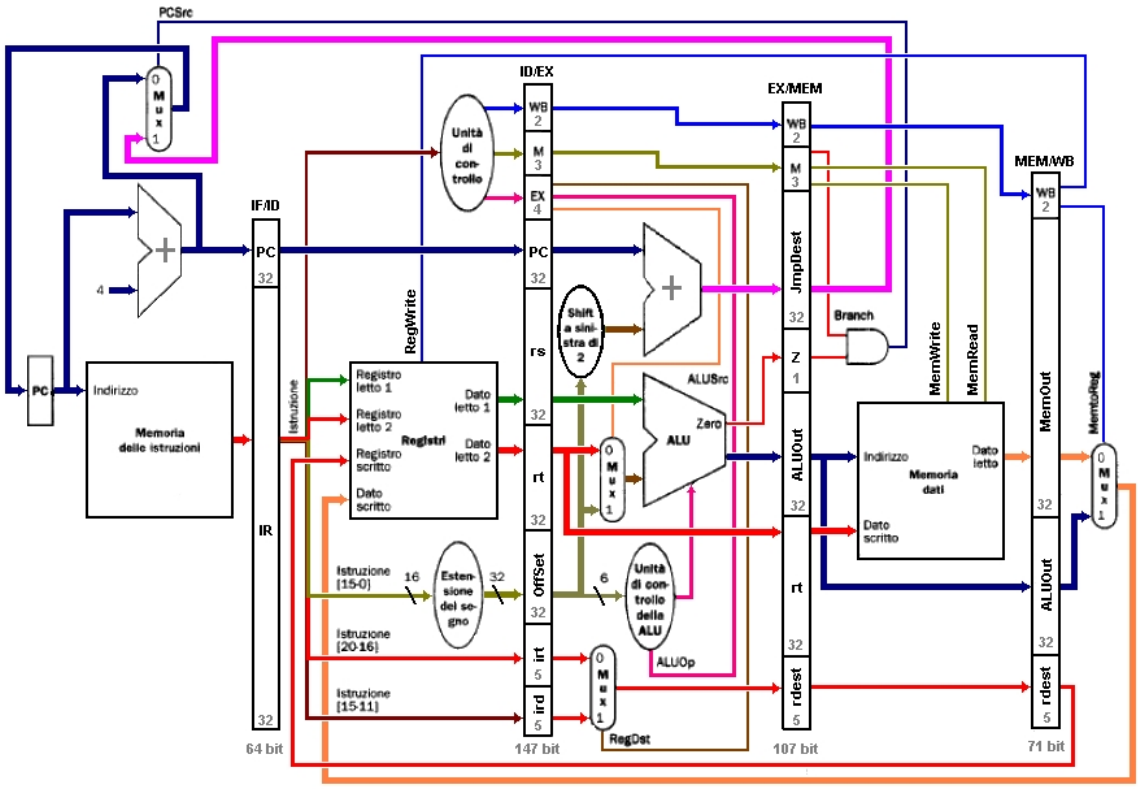
Esercitazione sulle CPU pipeline

Una CPU a ciclo singolo come pure una CPU multi ciclo eseguono una sola istruzione alla volta. Durante l'esecuzione parte dell'hardware della CPU rimane inutilizzato perché in attesa dei dati intermedi da un circuito precedente o perché già usato (o non usato) ed in attesa della fine dell'istruzione.
In una CPU pipeline l'hardware è diviso in stadi. Ogni stadio si occupa di un'operazione elementare (*Fetch, Decode, Execute, Memory Operation, WriteBack*) ed è separato dal successivo tramite un registro temporaneo. Ad ogni ciclo di clock, ogni stadio legge il registro di separazione con lo stato precedente, elabora i dati e scrive i risultati nel registro di separazione dallo stato successivo. Al successivo ciclo di clock ogni stadio potrà occuparsi dell'istruzione che segue senza attendere la fine di quella corrente. Se una CPU contiene **n stadi** potrà, in linea teorica, mantenere in esecuzione contemporaneamente **n istruzioni** ognuna ad un diverso stadio dell'esecuzione.



Consideriamo per semplicità una CPU pipelined semplificata schematizzata in figura. Questa CPU è in grado di eseguire solo operazioni tra registri, operazioni di load/store e salti condizionati con condizione di uguaglianza.
Sia data la seguente sequenza di istruzioni indipendenti tra loro di tipo R:

```
0x04000000  lw $1, 0($0)
0x04000004  add $2, $3, $4
0x04000008  ori $5, $6, 128
0x0400000C  beq $7,$8, 4
```

Massimo Marchi – marchi@dsi.unimi.it

Primo ciclo di clock

La trama di una generica istruzione di LOAD è:

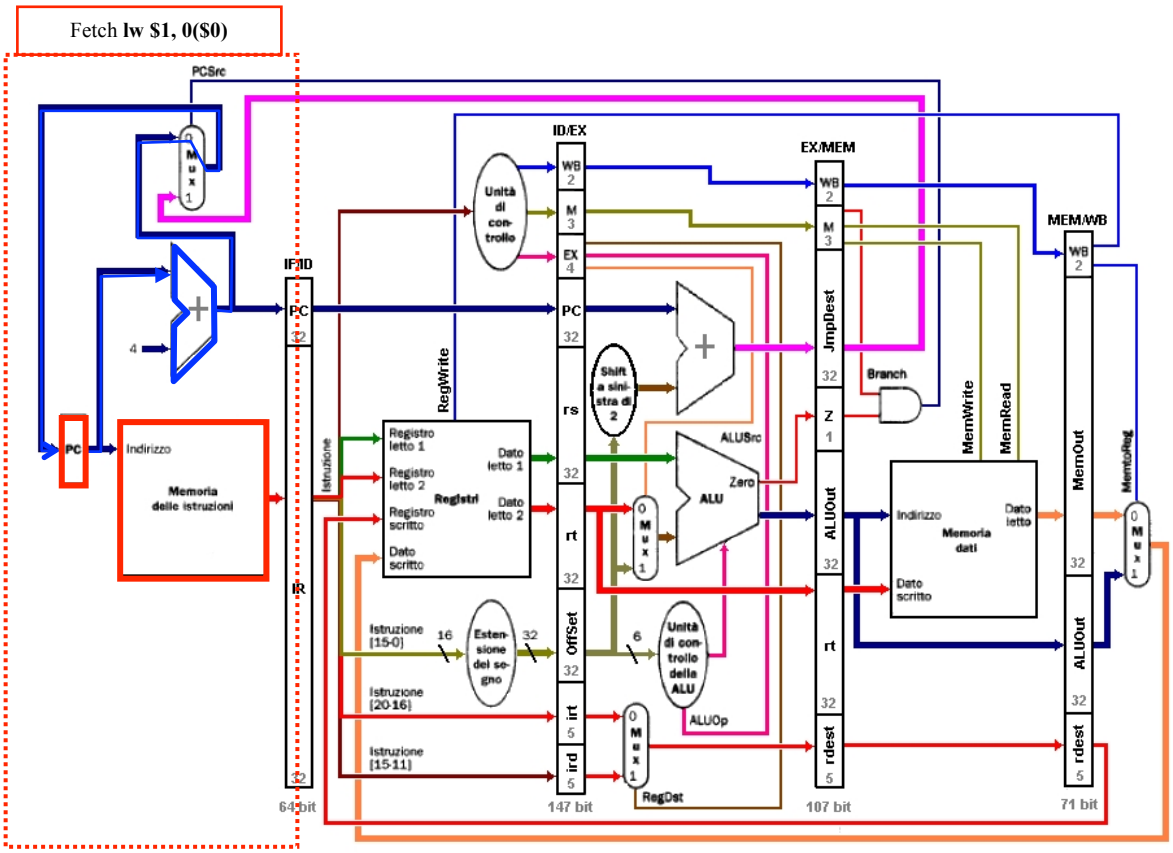
lw rt , offset (base)

31 26	25 21	20 16	15 0
LW 100011	**Base** 00000	**rt** 00001	**Offset** 0000 0000 0000 0000

Lo stadio di *fetch* si occupa di:

- Caricare la prossima istruzione da eseguire
- Calcolare l'indirizzo dell'istruzione immediatamente seguente

Ne segue che al primo ciclo di clock viene caricata la prima istruzione nel registro **IF/ID** (*Istruction Fetch/ Istruction Decode*). Il PC è incrementato di 4, **0x04000004**.



Il contenuto del registro **IF/ID** è alla fine del ciclo:

- **PC = 0x4000.0004**
- **IR = 1000 11. 00 000.0 0001. 0000 0000 0000 0000 = 0x8c01.0000**

Il registro **PC** viene aggiornato con il valore **0x4000.0004** dell'istruzione successiva (supponiamo non si siano verificati salti in precedenza).
In questa implementazione il registro **IF/ID** memorizza 64 bit: 32 bit del PC e 32 bit dell'istruzione

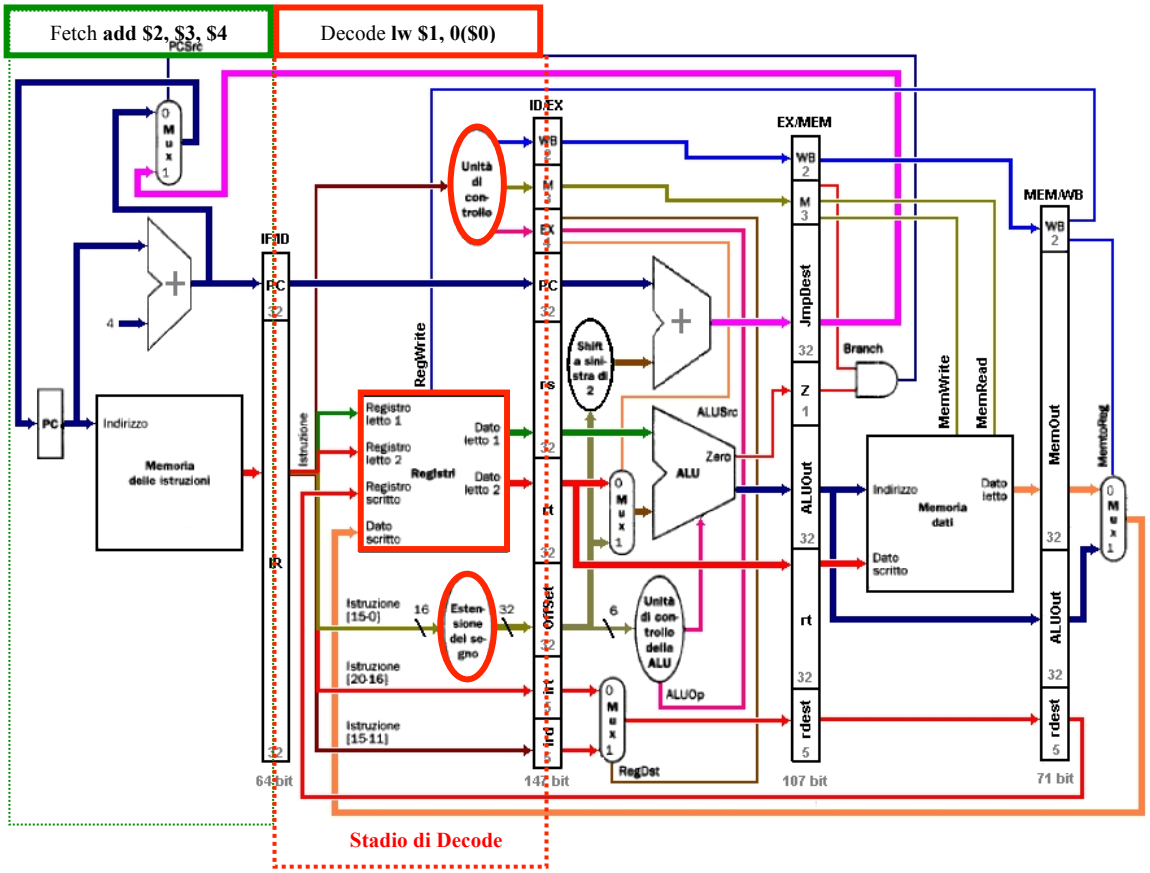
Secondo ciclo di clock

Al secondo ciclo di clock, il primo stadio esegue la fase di fetch della seconda istruzione mentre il secondo stadio esegue la decodifica della prima istruzione.
Nel primo stadio viene ripetuta la fase di fetch. La trama di una generica istruzione di tipo R è:

add rd , rs, rt

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 000000	**rs** 00011	**rt** 00100	**rd** 00010	**shift** 00000	**ADD** 100000

L'operazione di *fetch* (scrittura in **IF/ID**) non interferisce con l'operazione di decodifica (lettura da **IF/ID**) poiché i registri sono realizzati con architettura Master/Slave.



Lo stadio di *Decode* si occupa di:

- Decodificare l'istruzione da eseguire
- Estrarre dal register File il contenuto dei registri *rs* e *rt* in un ipotetica trama di tipo R
- Espandere l'ipotetico offset da 16 bit a 32 bit estendendo il segno.

Nello stadio di decodifica vengono eseguite tutte le operazioni che si possono eseguire senza sapere ancora di che tipo di istruzione si tratta, come nel caso della CPU mono/multi-ciclo. Tutti i dati letti dal *register file* vengono passati allo stadio successivo tramite il registro **ID/EX** (*Istruction Decode/Execute*). Al prossimo ciclo l'unità di controllo dovrà decodificare l'istruzione successiva. Ne segue che i segnali di controllo generati ma non ancora utilizzati (*eventuale lettura/scrittura dei registri, l'operazione da impostare sulla ALU, etc.*) andrebbero persi se non venissero memorizzati. Ad esempio, l'istruzione **lw** prevede al quinto ciclo di clock che il dato letto da memoria venga scritto nel registro indicato dal campo *rt***.** Questo valore però non potrà essere recuperato dall'**IF/ID** al momento della scrittura, come avviene nelle CPU non-pipelined, in quanto al quinto ciclo di clock **IF/ID** conterrà il codice operativo della quarta istruzione seguente. Analogamente il segnale di abilitazione alla scrittura nel register file, ottenibile dalla decodifica dell'istruzione **lw** andrà applicato anch'esso al quinto ciclo di clock. E' necessario quindi far percorrere la pipeline insieme ai dati temporanei anche i segnali di controllo usati negli stadi seguenti. In maniera analoga andranno trasferiti anche i segnali di controllo per i multiplexer presenti nei vari stadi in modo da realizzare i giusti data path.

I segnali di controllo presenti nel registri temporanei sono:

• **WB** (WriteBack) viene usato per trasportare i segnali di scrittura nel register file: 2 bit (RegWrite e MemToReg).
• **M** (Memory) controlla le operazioni sulla memoria dati e il segnale di salto condizionato: 3 bit (MemRead, MemWrite, Branch).
• **EX** (Execute) controlla le operazioni eseguite dalla **ALU**: 4 bit (RegDst,AluSrc, 2 bit di AluOp)

Nel dettaglio, alla fine del secondo ciclo il contenuto del registro **IF/ID** è (fetch di **add $2,$3,$4**):

- **PC = 0x4000.0008**
- **IR = 0000 00. 00 011.0 0100. 0001 0.000 00.10 0000 = 0x0064.1020**

Il registro **PC** viene aggiornato con il valore **0x40000008** dell'istruzione successiva (supponiamo non ci siano stati salti in precedenza). Il contenuto del registro **ID/EX** è (decode di **lw $1, 0($0)**):

- **WB.MemtoReg = 0**, prendi il dato da scrivere nei registri dalla memoria
- **WB.RegWrite = 1,** scrivi il dato nei registri
- **M.MemRead = 1**, leggi la memoria dati
- **M.MemWrite = 0** , non scrivere la memoria dati
- **M.Brach = 0** , non è un salto
- **EX.ALUOp = 00 ,** esegui la somma
- **EX.ALUSrc = 1** , usa l'**offset** esteso a 32 bit come secondo termine della somma,
- **EX.RegDest = 0 ,** usa **rt** come indirizzo del registro di destinazione
- **PC** = **0x4000.0004** l'indirizzo dell'istruzione successiva alla **lw**
- **rs** = **0x0000.0000**: il contenuto del registro puntato dal campo **irt**, **$0**, che vale sempre 0.
- **rt** = **0x1111.1111**: il contenuto del registro puntato dal campo **irt**, **$1**, supponiamo **0x1111.1111**.
- **offset = 0x0000.0000 ,** l'offset esteso su 32 bit con segno.
- **irt = 1** , indice del registro **rt** dell'istruzione**,**
- **ird = 0** , indice del registro **rd** dell'istruzione**,** non usato dall'istruzione **lw**.

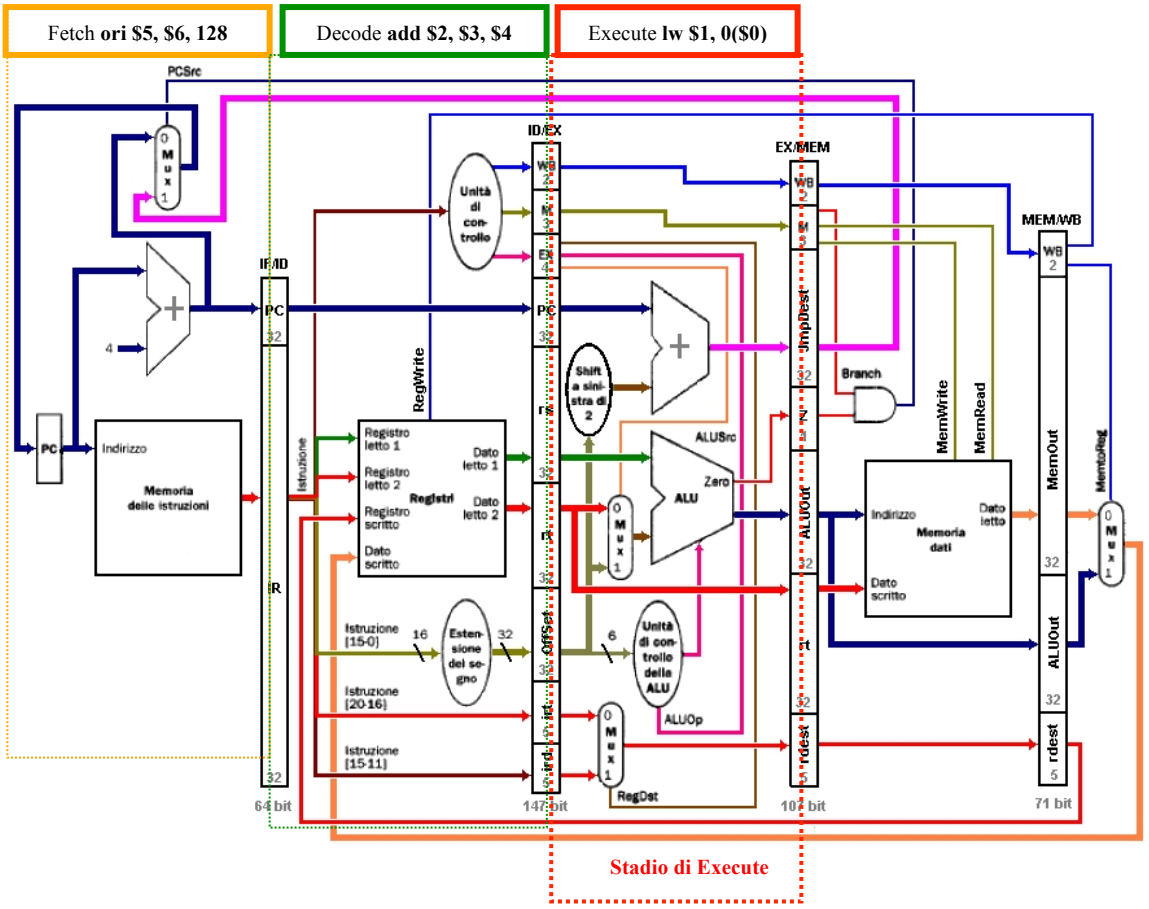
In questa implementazione il registro **ID/EX** memorizza 147 bit: 9 bit per i registri *WB*,*M* e *EX*, 32 bit per il *PC*, 64 bit per il contenuto dei registri *rs* e *rt*, 32 bit per l'offset esteso, 10 bit per gli indici *irs* e *irt*.

Terzo ciclo di clock

Al terzo ciclo di clock il primo stadio esegue la fase di fetch della terza istruzione dell'esempio, il secondo stadio esegue la decodifica della seconda istruzione mentre il terzo stadio esegue la prima istruzione.
La trama di una generica istruzione di tipo I è:

ori rt, rs, immediate

31 26	25 21	20 16	15 0
ORI 001101	**rs** 00110	**Rt** 00101	**immediate** 0000 0000 1000 0000



Lo stadio di *Execute* si occupa di:

- calcolare l'indirizzo di un eventuale salto relativo considerando il PC e l'eventuale offset esteso a 32 bit e moltiplicato per 4;
- calcolare attraverso l'ALU l'operazione di una eventuale istruzione di tipo R;
- selezionare l'indice del registro di destinazione tra rd e rt di un istruzione di scrittura su registri.

Nel registro **EX/MEM** (*Execute/Memory*) il terzo stadio memorizza in questo caso:

- i controlli di **lw** per gli stadi successivi (**WB** e **M**);
- l'indirizzo **0($0)** calcolato dalla **ALU;**
- l'indice del registro **rt ($1)** selezionato dal MUX attraverso il segnale **RegDst**.
- l'indirizzo di un eventuale salto condizionato calcolato tra l'indirizzo dell'istruzione successiva di **lw** e l'offset esteso e moltiplicato per 4, in questo caso non usato.

Contemporaneamente nel registro **ID/EX** vengono memorizzati i controlli ed i valori dei registri relativi alla decodifica dell'instruzione seguente, in questo caso l'istruzione di tipo R **add**. Poiché è un operazione tra registri vengono impostati i controlli in modo che alla ALU arrivino i valori dei registri. L'operazione da eseguire viene decisa tramite il campo **funct**. Nessuna operazione sulla memoria è impostata.
Nel registro **IF/ID** il primo stadio memorizza il codice operativo della terza istruzione, **ori $5,$6,128** in questo esempio.
Il registro **PC** viene aggiornato con il valore **0x4000000c** dell'istruzione successiva (supponiamo che non si siano verificati salti in precedenza).

Nel dettaglio, alla fine del ciclo il contenuto del registro **IF/ID** è (fetch di **ori $5,$6,128**):

- **PC = 0x4000.000c**
- **IR = 0011 01. 00 110.0 0101. 0000 0000 1000 0000 = 0x34c5.0080**

Il contenuto del registro **ID/EX** è (decode di **add $2,$3,$4**):

- **WB.MemtoReg = 1** (prendi il dato da scrivere dalla ALU)
- **WB.RegWrite = 1** (scrivi il register file)
- **M.MemRead = 0, M.MemWrite = 0** (nessuna operazione sulla memoria)
- **M.Branch=0** (non è un salto condizionato)
- **EX.ALUOp = 10** (operazione decisa dal campo **funct**),
- **EX.ALUSrc = 0** (usa **rt** come secondo operando)
- **EX.RegDest = 1** (usa il campo **ird** come destinazione)
- **PC** = **0x4000.0008** l'indirizzo dell'istruzione successiva alla **add**
- **rs** = **0x3333.3333**: il contenuto del registro puntato dal campo **irt**, **$3**, supponiamo **0x3333.3333.**
- **rt** = **0x4444.4444**: il contenuto del registro puntato dal campo **irt**, **$4**, supponiamo **0x4444.4444**
- **offset = 0x1111.1020 (**l'**offset** esteso su 32 bit con segno, non usato)
- **irt = 4 (**indice del registro **rt,** non usato**)**
- **ird = 2 (**indice del registro **rd**)

Il contenuto del registro **EX/MEM** è (execute **lw $1,0($0)**):

- **WB** e **M** come **ID/EX** al clock precedente:
- **WB.MemtoReg = 0**, **WB.RegWrite = 1,**
- **M.MemRead = 1** , **M.MemWrite = 0** , **M.Branch=0**
- **JBranch=0x4000.0004** (0x4000.0004+0x0000.0000*4), indirizzo di un eventuale salto, non usato
- **Z=1**, flag risultato della ALU uguale a zero, non usato
- **ALUOut = 0** (è il risultato calcolato dalla ALU, in questo caso il valore del registro **rt** più l'**offset** presi da **ID/EX**, **0($0)**)
- **rt = 0x1111.1111 (**il contenuto del registro puntato dal campo **irt**, **$1 = 0x1111.1111**, da usare in un ipotetica operazione di scrittura in memoria, non usato)
- **idest = 1** (indice del registro da usare nell'operazione di WriteBack, **$1** in questo caso).

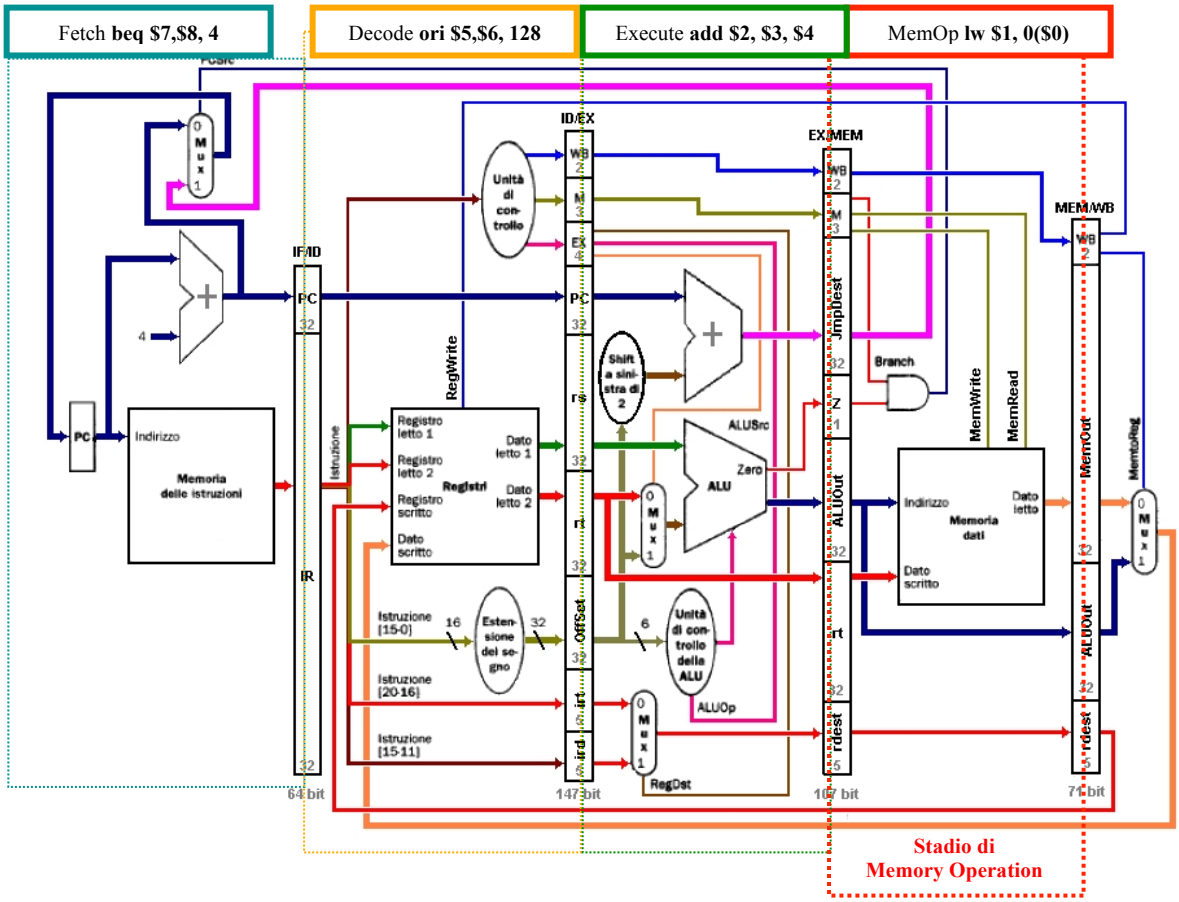
In questa implementazione il registro **EX/MEM** memorizza 107 bit: 5 bit per i registri *WB* ed *M*, 32 bit per l'indirizzo di salto condizionato, 1 bit per il flag Z della ALU, 32 bit per il risultato della ALU, 32 bit per il contenuto del registro *rt*, 5 bit per l'indice *idest*.

Quarto ciclo di clock

Al quarto ciclo di clock il primo stadio esegue la fase di fetch della quarta istruzione, il secondo stadio esegue la decodifica della terza istruzione, il terzo stadio esegue la seconde istruzione mentre il quarto stadio esegue le operazione sulla memoria dati relative alla prima istruzione.
La trama di una generica istruzione di salto condizionato è (l'offset è riportato così come indicato nell'istruzione in linguaggio macchina):

beq rs, rt, offset

31 26	25 21	20 16	15 0
BEQ 000100	**rs** 00111	**rt** 01000	**Offset** 0000 0000 0000 0100



Lo stadio di Memory Operation si occupa di:

- eseguire l'eventuale operazione di lettura/scrittura in memoria dati,
- decidere se esegure il salto di un eventuale operazione di salto condizionato.

Nel registro **MEM/WB** il quarto stadio memorizza oltre al dato contenuto nella cella di memoria dell'indirizzo **0($0)** anche l'indice **idest** del registro **rt** passatogli dallo stadio precedente ed i segnali di controllo **WB**.
Nel registro **EX/EM** memorizza il risultato dell'operazione di *Execute* sulla seconda istruzione dell'esempio. Il registro **ID/EX** memorizza il risultato della deccodifica della terza istruzione. Infine il registro **IR/ID** il primo stadio memorizza il codice operativo della quarta istruzione.

Nel dettaglio, alla fine del ciclo il contenuto del registro **IF/ID** è (fetch di **beq $7,$8, 4**):

- **PC = 0x4000.0010**
- **IR = 0001 00.00 111.0 1000. 0000 0000 0000 0100 = 0x10e8.0004**

Il registro **PC** viene aggiornato con il valore **0x40000010** dell'istruzione successiva.

Il contenuto del registro **ID/EX** è (decode di **ori $5,$6,128**):

- **WB.MemtoReg = 1** (prendi il dato da scrivere dalla ALU)
- **WB.RegWrite = 1** (scrivi il register file)
- **M.MemRead = 0, M.MemWrite = 0** (nessuna operazione sulla memoria)
- **M.Branch=0** (non è un salto condizionato)
- **EX.ALUOp = 11 (OR)** (ALU deve eseguire una operazione **OR**),
- **EX.ALUSrc = 1** (usa **offset** come secondo operando)
- **EX.RegDest = 0** (usa il campo **irt** come destinazione)
- **PC** = **0x4000.000c** l'indirizzo dell'istruzione successiva alla **ori**
- **rs** = **0x5555.55555**: il contenuto del registro puntato dal campo **irs**, **$5**, supponiamo. **0x5555.5555**
- **rt** = **0x6666.6666**: il contenuto del registro puntato dal campo **irt**, **$6**, supponiamo **0x6666.6666**
- **offset = 0x0000.0080 (**l'**offset** esteso su 32 bit con segno)
- **irt = 6 (**indice del registro **rt)**
- **ird = 0 (**indice del registro **rd**, non usato)

I

l contenuto del registro **EX/MEM** è (execute **add $2,$3,$4**):

- **WB, M** come **ID/EX** al clock precedente**:**
- **WB.MemtoReg = 1**, **WB.RegWrite = 1,**
- **M.MemRead = 0**, **M.MemWrite = 0**, **M.Branch=0**
- **ALUOut** = **0x7777.7777** (**0x3333.3333** + **0x4444.4444)**, è il risultato calcolato dalla ALU, in questo caso il valore del registro **rs** più il valore del registro **rt**.
- **JBranch = 0x8444.4088** (0x4000.0008+0x1111.1020*4), indirizzo di un eventuale salto, non usato
- **Z=0**, flag risultato della ALU uguale a zero, non usato
- **rt=0x4444.4444.** il contenuto del registro puntato dal campo **irt**, **$4**, da usare in un ipotetica operazione di scrittura in memoria, non usato,
- **idest = 2** , indice del registro da usare nell'operazione di WriteBack.

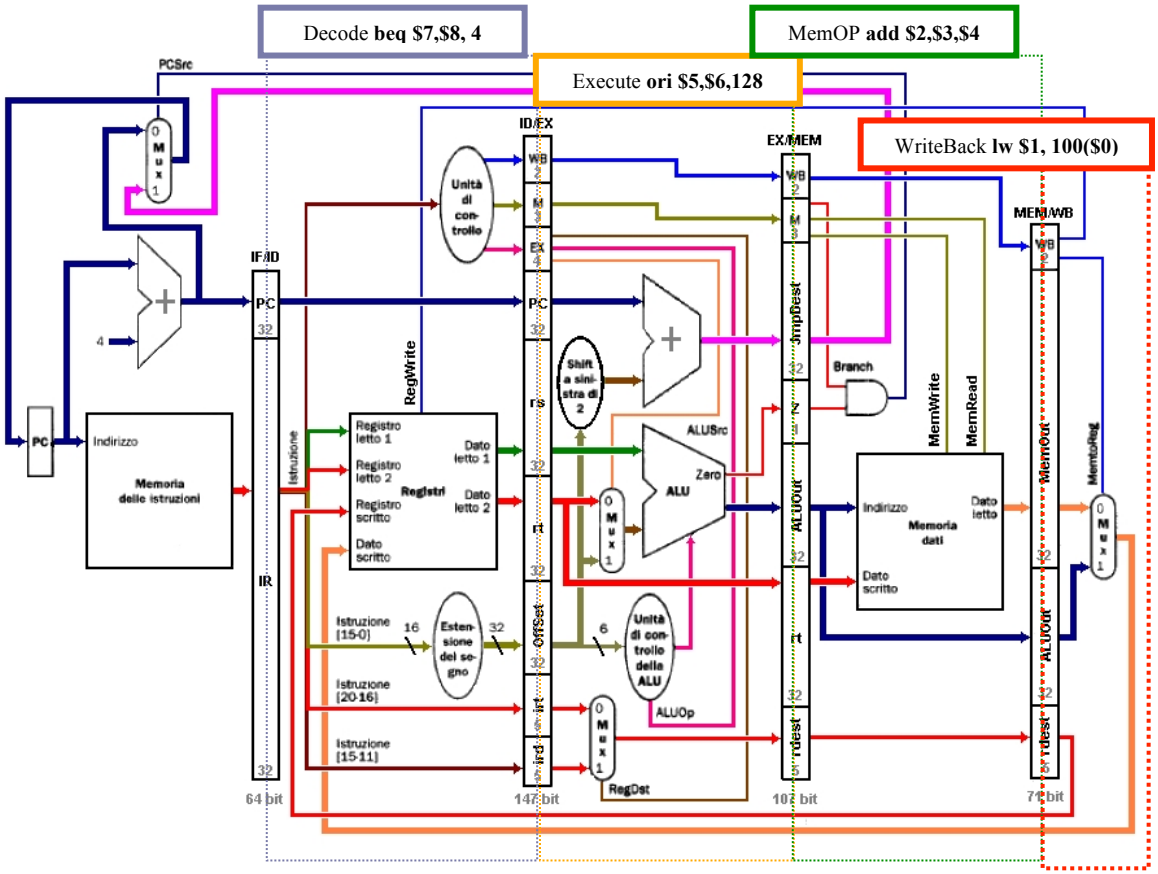
Il contenuto del registro **MEM/WB** è (MemOp di **lw $1,0($0)**):

- **WB** come **EX/MEM** al clock precedente:
- **WB.MemtoReg = 0**, **WB.RegWrite = 1,**
- **MemOut**, è il dato estratto dalla memoria alla posizione 0($0), ex **0x1234.5678**.
- **AluOut**, il risultato ALU precedente, 0, non usato,
- **idest = 1** , indice del registro da usare nell'operazione di WriteBack.

In questa implementazione il registro **MEM/WB** memorizza 71 bit: 2 bit per i registri *WB*, 32 bit per l'eventuale dato in uscita dalla memoria, 32 bit per il risultato della ALU, 5 bit per l'indice *idest*.

Quinto ciclo di clock

Al quinto ciclo di clock è possibile terminare la prima istruzione; il quinto stadio scrive il dato letto dalla memoria nel *Register File*. Questa operazione, usando un *Register File* che consente letture e scritture contemporanee, può avvenire in contemporanea alla operazione di decodifica/lettura dei registri relativa alla quarta istruzione. Nello stesso ciclo il primo stadio esegue la fase di fetch dell'istruzione seguente alla quarta istruzione (non riportata), il terzo stadio esegue la terza istruzione mentre il quarto stadio esegue le operazione sulla memoria dati relative alla seconda istruzione.



Sfruttando l'indirizzo **rd** relativo alla prima istruzione trasportato dalla pipe, il quinto stadio *scrive indietro* nel register file il contenuto della memoria letto nello stadio precedente e selezionato attraverso il Mux controllato dal segnale *MemToReg=0*.
Nel registro **MEM/WB** il quarto stadio passa avanti il dato calcolato dalla ALU ed anche l'indice del registro **rd** passatogli dallo stadio precedente ed il segnale di controllo **WB**.
Nel registro **EX/EM** il terzo stadio la ALU esegue una OR tra rs e l'offset, l'indice del registro **rd** (**$2**) passatogli dallo stadio precedente, i segnali di controllo **WB** ed **M** generati nel quarto ciclo dalla **UC** relativi alla terza istruzione.
Nel registro **ID/EX** il secondo stadio memorizza il contenuto dei registri **rs**, **rt** estratti dal *Register File* relativi alla quarta istruzione, nonché i segnali di controllo ottenuti dalla decodifica dell'istruzione.

Il contenuto del registro **ID/EX** è (decode di **beq $7,$8,4**):

- **WB.MemToReg=0:** nel caso della *beq* questo segnale non ha importanza poiché non vengono eseguite operazioni di scrittura sui registri può essere settato come meglio serve per minimizzare la logica della Control Unit; per completezza poniamolo uguale a 0.
- **WB.RegWrite = 0** (non scrivere il register file)
- **M.MemRead = 0, M.MemWrite = 0** (nessuna operazione sulla memoria)
- **M.Branch=1** (è un salto condizionato)
- **EX.ALUOp = 01 (Diff)** (ALU deve eseguire una sottrazione),
- **EX.ALUSrc = 0** (usa **rt** come secondo operando)
- **EX.RegDest:** non verrà eseguita nessuna operazione sui registri, questop valore può essere preso come meglio occorre come per MemToReg; poniamolo a 0 per completezza.
- **PC** = **0x4000.0010** l'indirizzo dell'istruzione successiva alla **beq**
- **rs=0x7777.7777** (il contenuto del registro puntato dal campo **irs**, **$7**, supponiamo **0x7777.7777)**
- **rt=0x8888.8888** (il contenuto del registro puntato dal campo **irt**, **$6**, supponiamo **0x8888.8888**)
- **offset = 0x0000.0004 (**l'**offset** esteso su 32 bit con segno)
- **irt = 8 (**indice del registro **rt,** non usato**)**
- **ird = 0 (**indice del registro **rd**, non usato)

Il contenuto del registro **EX/MEM** è (execute **ori $5,$6,128**):

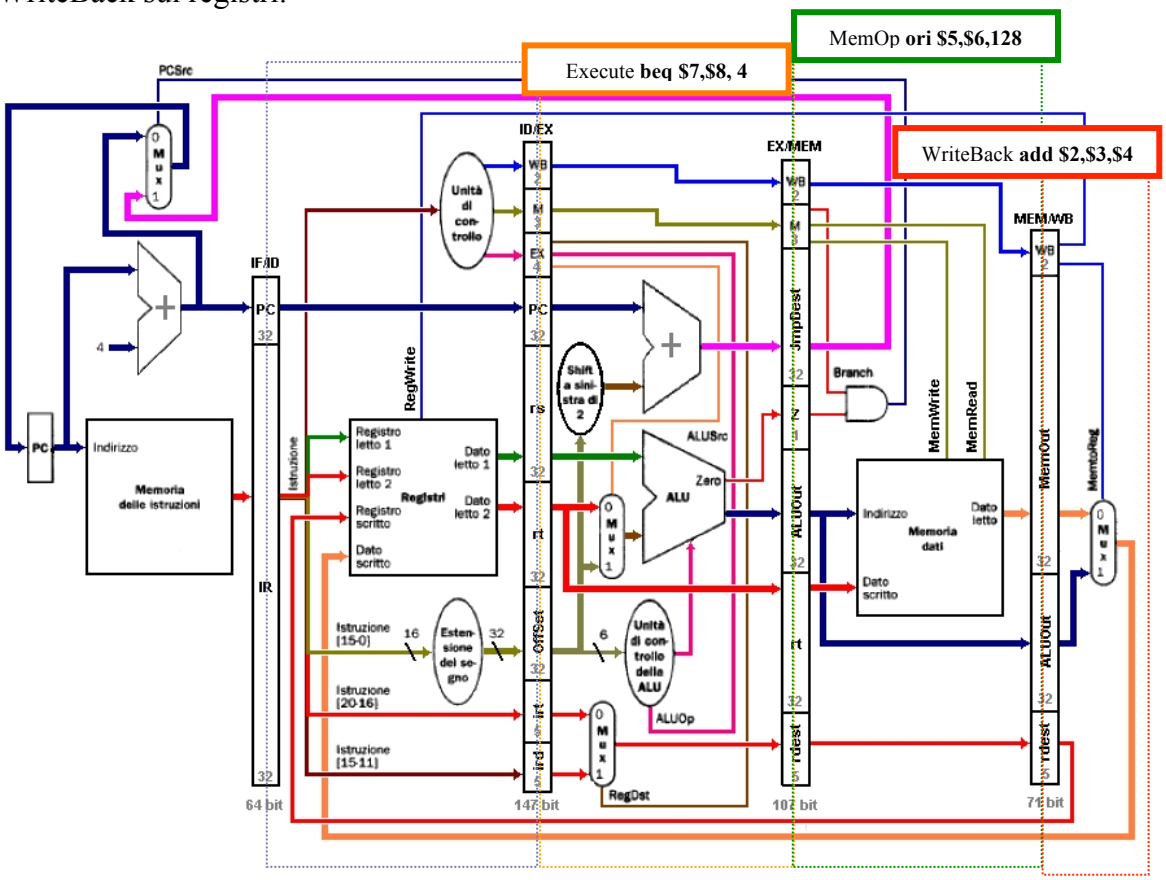
- **WB, M** come **ID/EX** al clock precedente**:**
- **WB.MemtoReg = 1**, **WB.RegWrite = 1,**
- **M.MemRead = 0**, **M.MemWrite = 0**, **M.Branch=0**
- **ALUOut = 0x6666.66e6** (0x6666.6666 or 0x0000.0080), è il risultato calcolato dalla ALU, in questo caso il valore del registro **rs** OR il valore dell'offset
- **JBranch =0x4000.020c** (0x4000.000c+0x0000.0080*4), indirizzo di un eventuale salto, non usato
- **Z=0**, flag risultato della ALU uguale a zero, non usato
- **rt=0x6666.6666**, il contenuto del registro puntato dal campo **irt**, **$6**, da usare in un ipotetica operazione di scrittura in memoria, non usato,
- **idest = 5** , indice del registro da usare nell'operazione di WriteBack.

Il contenuto del registro **MEM/WB** è (MemOp di **add $2,$3,$4**):

- **WB** come **EX/MEM** al clock precedente:
- **WB.MemtoReg = 1**, **WB.RegWrite = 1,**
- **MemOut=0x0000.0000**: L'uscita della memoria è indeterminata poiché non viene eseguita nessuna lettura; per completezza supponiamo sia 0x0000.0000.
- **AluOut = 0x7777.7777**, il risultato ALU precedente.
- **idest = 2** , indice del registro da usare nell'operazione di WriteBack.

Sesto ciclo di clock

Al successivo ciclo di clock la ori passa in fase di MemOp e si comporta come l'istruzione add precedente mentre la **beq** esegue la differenza tra i due registri $7 ed $8. Se il risultato è uguale a zero allora il segnale Z viene messo a 1 ed al prossimo ciclo di clock provocherà la modifica del PC come indicato nel registro JBranch, altrimenti il flusso di istruzioni proseguira in maniera sequenziale. L'istruzione add esegue la fase di WriteBack sui registri.



Il contenuto del registro **EX/MEM** è (execute **beq $7,$8,4**):

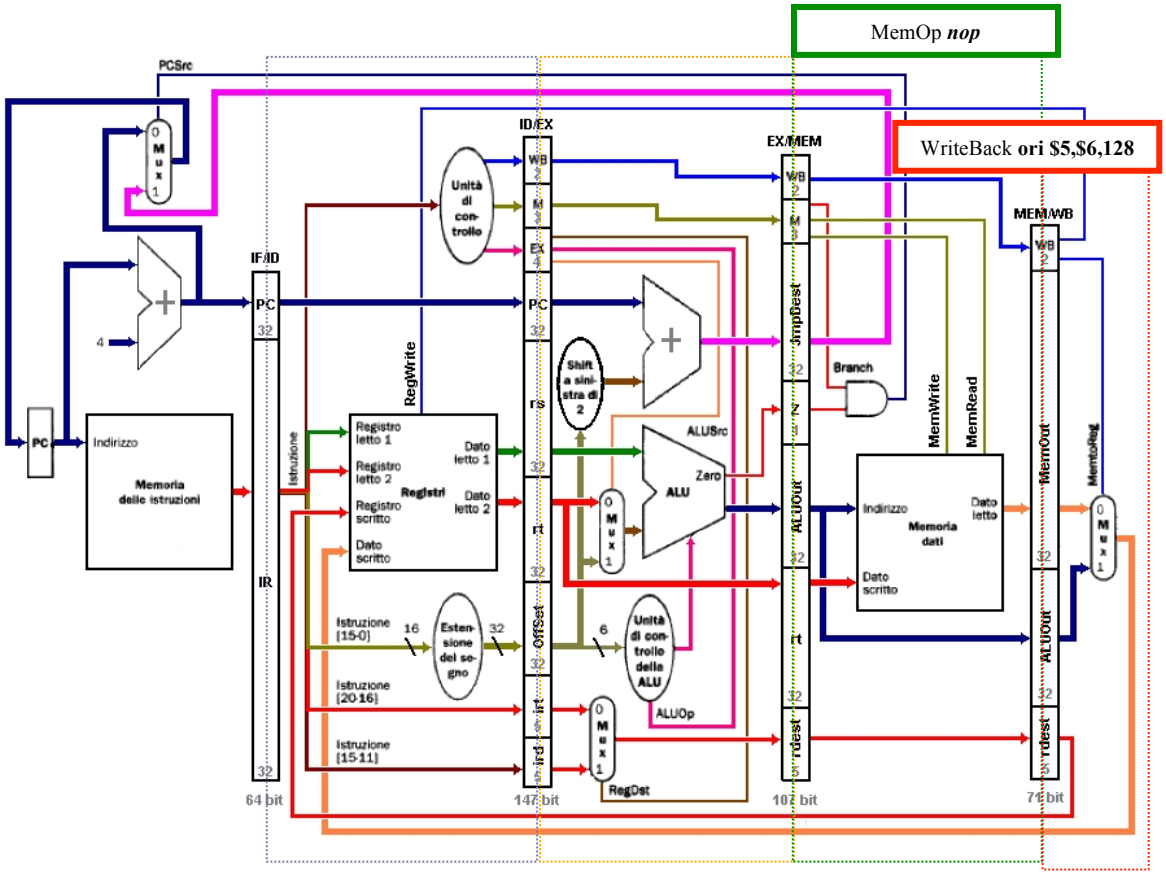
- **WB, M** come **ID/EX** al clock precedente**:**
- **WB.MemtoReg = x**, **WB.RegWrite = x,**
- **M.MemRead = 0**, **M.MemWrite = 0**, **M.Branch=1**
- **ALUOut**, è il risultato calcolato dalla ALU, in questo caso il valore del registro **rs** meno il valore del registro **rt**, **0x7777.7777 - 0x8888.8888** = **0xEEEE.EEEF** (-0x1111.1111).
- **JBranch = 0x4000.0020** (0x4000.0010 + 0x0000.0004*4), l'indirizzo del salto condizionato, in questo caso non usato perché il risultato della differenza non è zero.
- **Z**, flag risultato della ALU uguale a zero, in questo caso uguale a **0**
- **rt**, **0x8888.8888**, non usato
- **idest = 8** (l'indice di **irt**, non usato).

Il contenuto del registro **MEM/WB** è (MemOp di **ori $5,$6,128**):

- **WB** come **EX/MEM** al clock precedente:
- **WB.MemtoReg = 1**, **WB.RegWrite = 1,**
- **MemOut=0x0000.0000**, L'uscita della memoria è indeterminata poiché non viene eseguita nessuna lettura; per completezza supponiamo sia 0x0000.0000.
- **AluOut = 0x6666.66e6**, il risultato ALU precedente, ,
- **idest = 5** , indice del registro da usare nell'operazione di WriteBack.

Settimo ciclo di clock

Al successivo ciclo di clock la ori passa in fase di WriteBack mentre la **beq** si trasforma all'atto pratico come una istruzione di *NoOperation* (non deve effettuare nessuna operazione di lettura/scrittura sulla memoria o sui registri). In questo caso il flusso di istruzioni proseguira in maniera sequenziale; al prossimo ciclo di clock nel *PC* verrà incrementato di 4. Se si fosse verificato il salto, al prossimo ciclo di clock sarebbe stato caricato il contenuto del registro *JumpDest* attraverso il Mux controllato dal segnale PCSrc.



Il contenuto del registro **MEM/WB** è banalmente:

- **WB** come **EX/MEM** al clock precedente:
- **WB.MemtoReg = x**, **WB.RegWrite = 0,**
- **MemOut = 0x0000.0000**, L'uscita della memoria è indeterminata poiché non viene eseguita nessuna lettura; per completezza supponiamo sia 0x0000.0000.
- **AluOut** = **0xEEEE.EEEF**, il risultato ALU precedente, non usato
- **idest = 8**, non usato.

Ogni istruzione richiede esattamente 5 cicli di clock per essere eseguita totalmente ma grazie al parallelismo dei 5 stadi si potranno eseguire 5 istruzioni contemporaneamente riuscendo quindi ad eseguire mediamente un istruzione per ogni ciclo di clock.