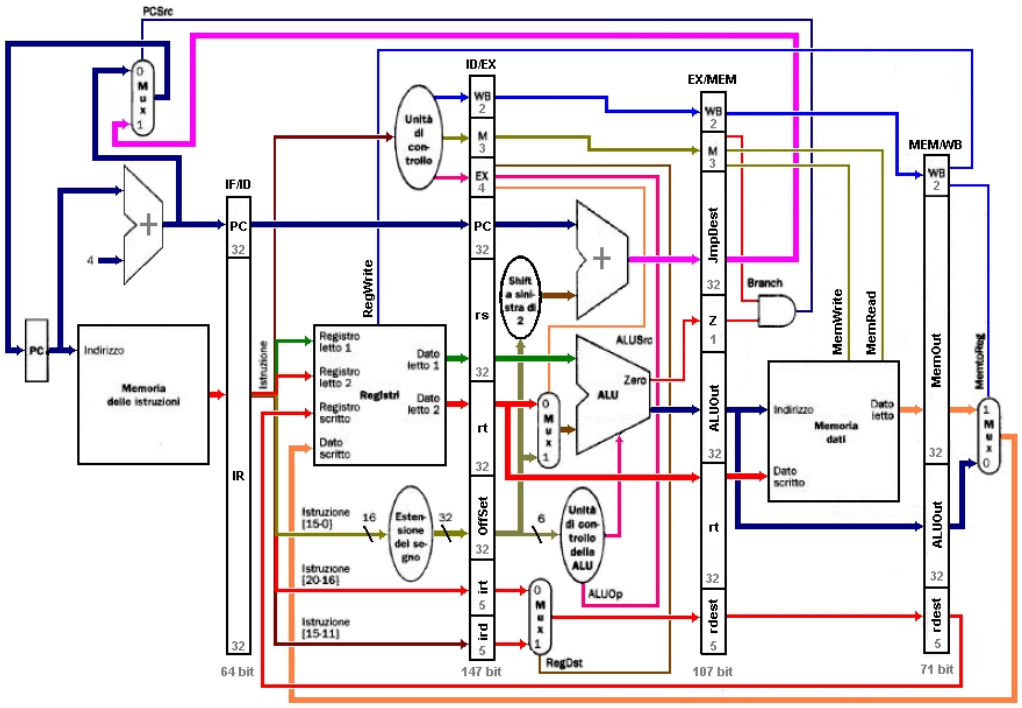
Esercitazione del 30/04/2010 - Soluzioni

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 in attesa che dagli stadi precedenti arrivino i dati da elaborare.
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. Ogni stadio, ad ogni ciclo di clock, legge il registro di separazione con lo stato precedente, elabora i dati e scrive i risultati nel registro di separazione dello 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 pipilined semplificata come riportato di seguito. 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
```

La trama di una generica istruzione di LOAD è:

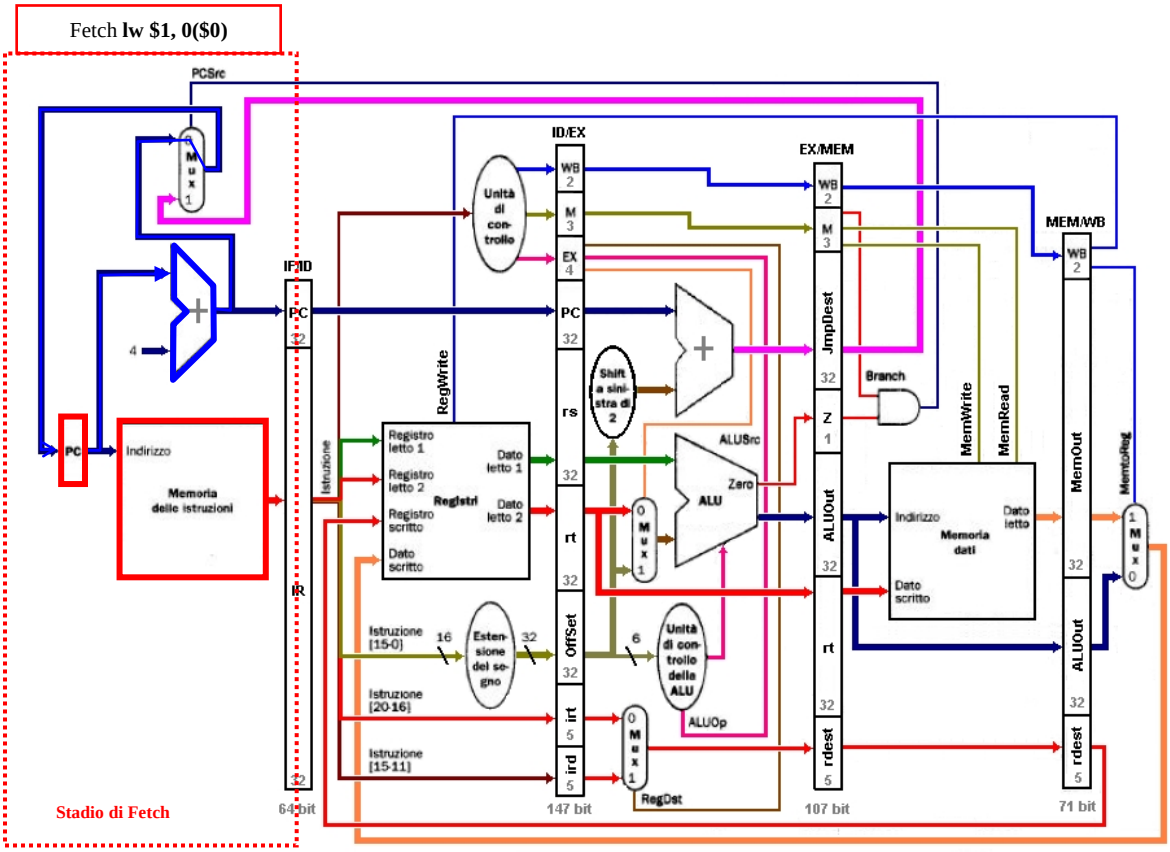
lw rt , offset (base)

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**.

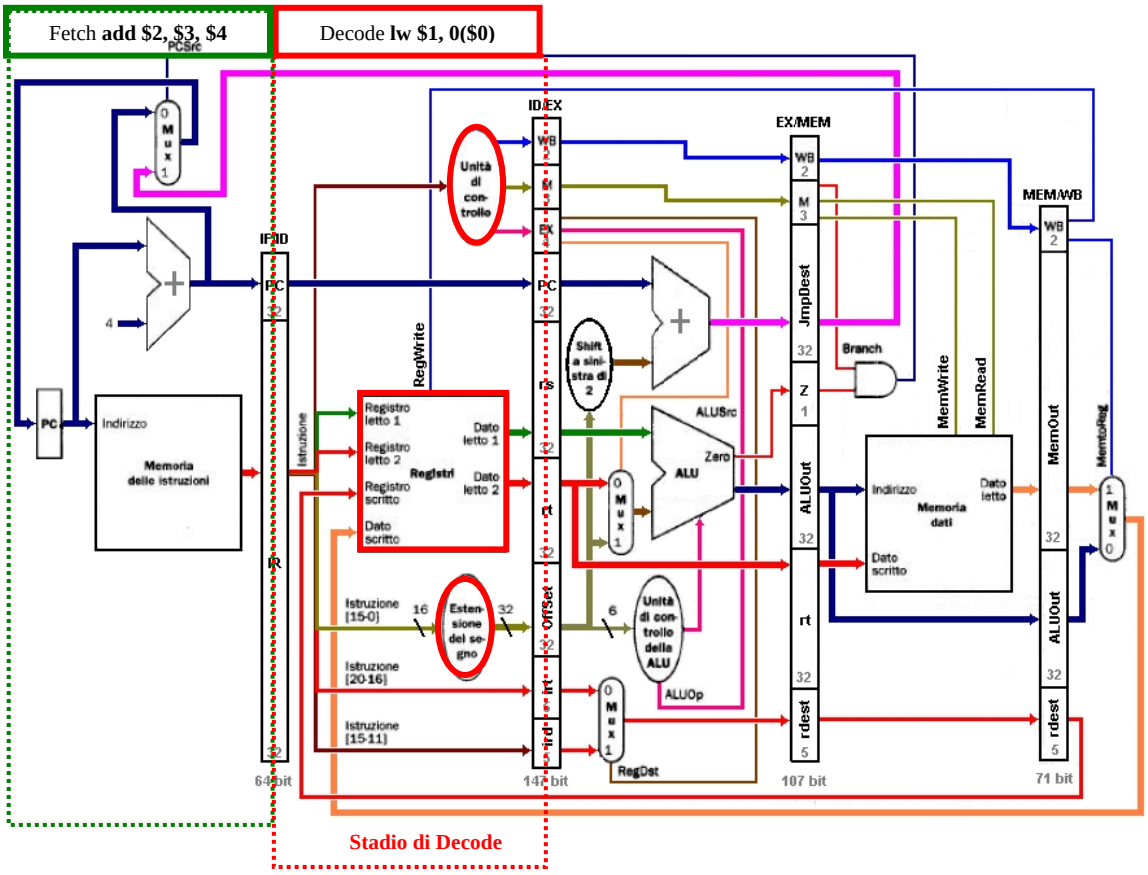


Il contenuto del registro **IF/ID** è alla fine del ciclo:

- **PC = 0x4000.0004**
- **IR = 100011. 00 000.00001. 00000000 00000000 = 0x8c01.0000**

Il registro **PC** viene aggiornato con il valore **0x4000.0004** dell'istruzione successiva (ignoriamo ora eventuali salti).
In questa implementazione il primo registro memorizza 64 bit: 32 bit del PC e 32 bit dell'istruzione.

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.
Il **PC** viene come nel ciclo precedente incrementato di 4. 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'offset 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 la seconda istruzione. 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)

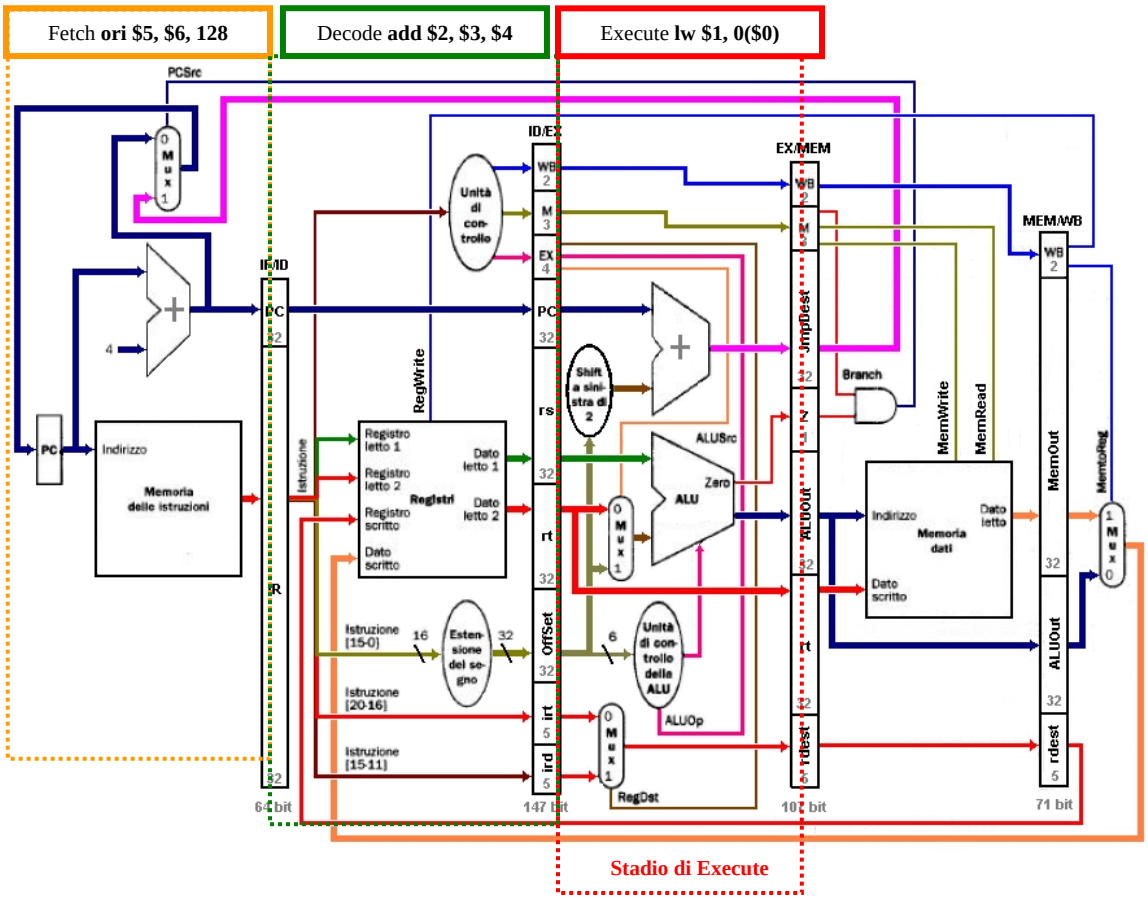
Nel dettaglio, alla fine del secondo ciclo il contenuto del registro **IF/ID** è:

- **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 (ignoriamo eventuali salti). Il contenuto del registro **ID/EX** è:

- **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**, il contenuto del registro puntato dal campo **rs** dell'istruzione, **$0**=**0** in questo caso.
- **rt**, il contenuto del registro puntato dal campo **rt** dell'istruzione, **$1** in questo caso, non usato (supponiamo valga **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 da questa istruzione.

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. Il **PC** come prima è incrementato di 4.



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

- 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.

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 registro **PC** viene aggiornato con il valore **0x4000000c** dell'istruzione successiva.

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

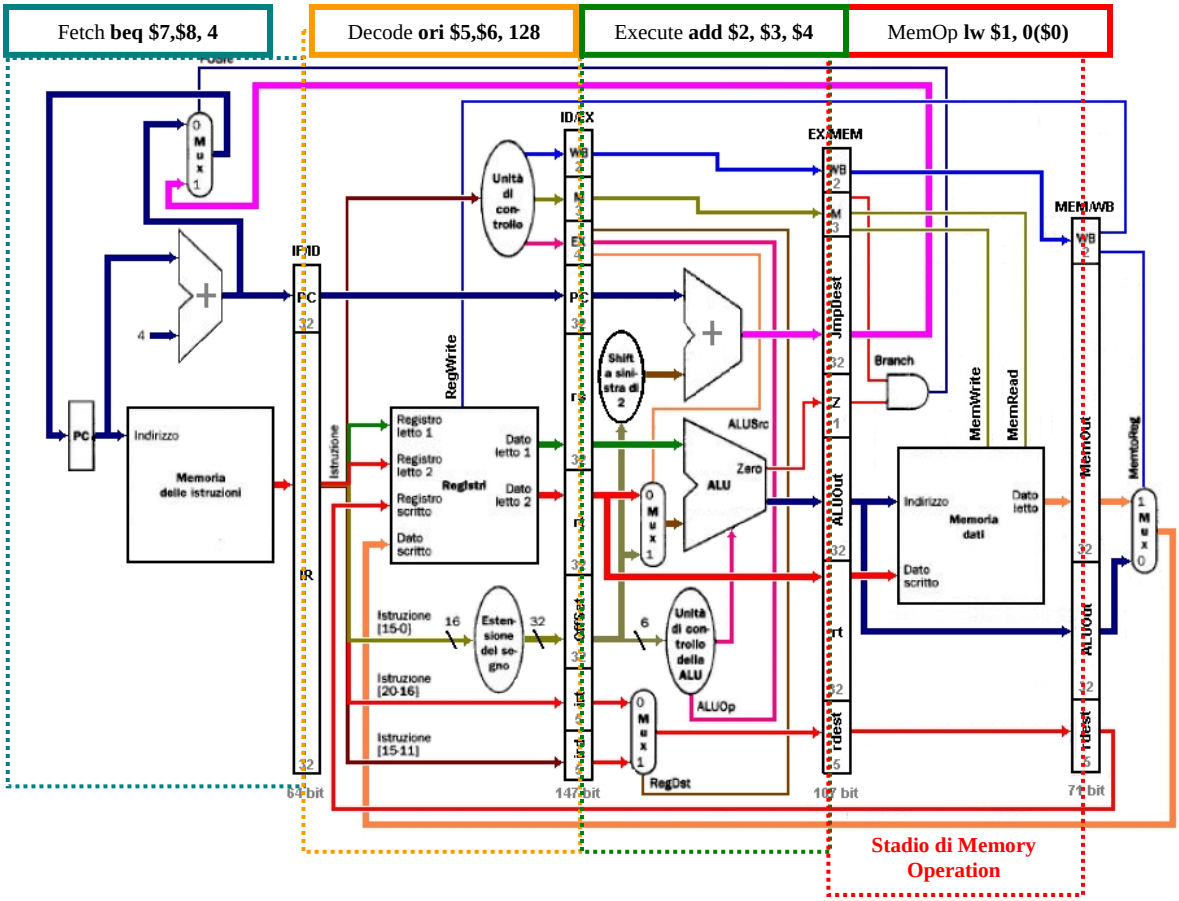
- **WB.MemtoReg = 1** (prendi il dato da scrivere dalla ALU)
- **WB.RegWrite = 1** (scrivi il register file)
- **M.MemRead = 0, M.MemWrite = 0** (nessuna operazione sulla memoria)
- **M.Branch=0** (non è un salto condizionato)
- **EX.ALUOp = 10** (operazione decisa dal campo **funct**),
- **EX.ALUSrc = 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** (il contenuto del registro puntato dal campo **irs**, **$3**, ex. **0x3333.3333)**
- **rt** (il contenuto del registro puntato dal campo **irt**, **$4**, ex: **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**, indirizzo di un eventuale salto, non usato
- **Z**, 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)

.

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. Il **PC** come prima è incrementato di **4**.



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 = "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** (il contenuto del registro puntato dal campo **irs**, **$5**, ex. **0x5555.5555)**
- **rt** (il contenuto del registro puntato dal campo **irt**, **$6**, ex: **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)

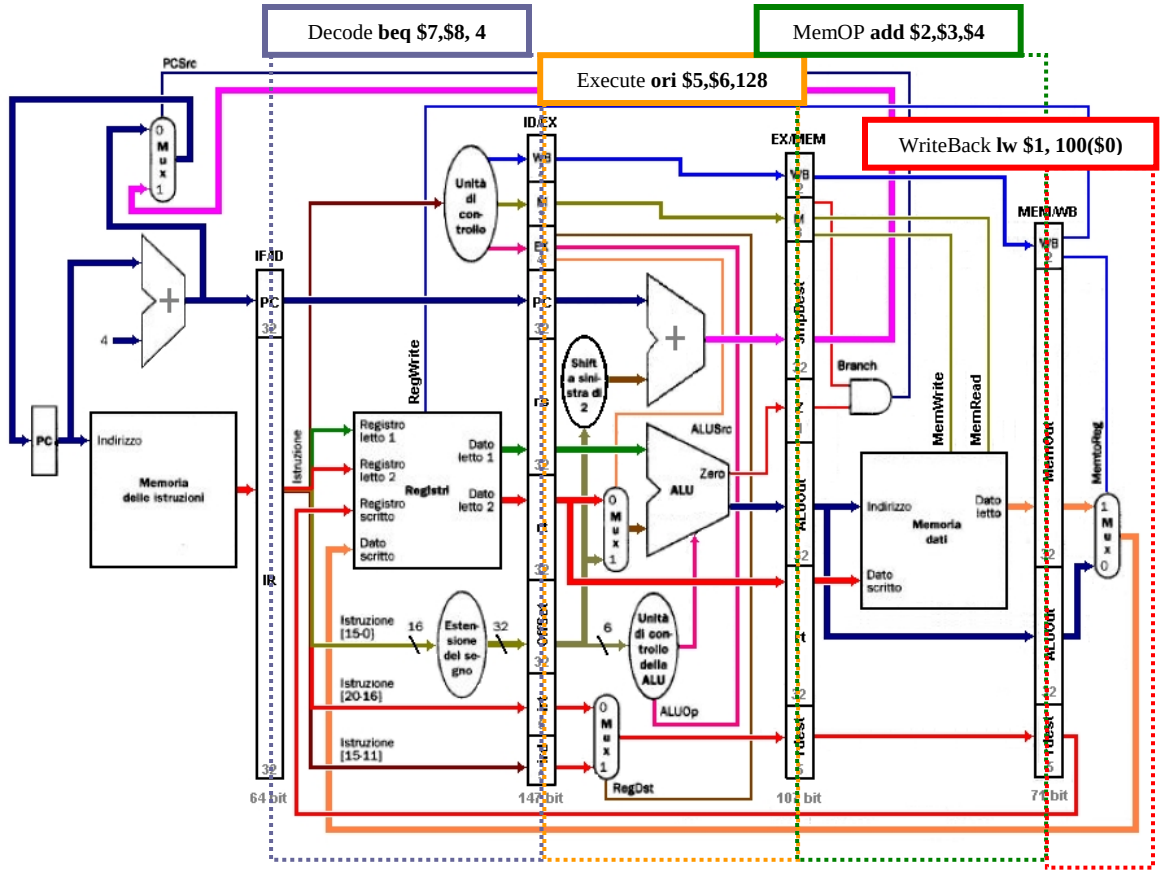
Il contenuto del registro **EX/MEM** è (execute **add $2,$3,$4**):

- **WB, M** come **ID/EX** al clock precedente**:**
- **WB.MemtoReg = 1**, **WB.RegWrite = 1,**
- **M.MemRead = 0**, **M.MemWrite = 0**, **M.Branch=0**
- **ALUOut**, è il risultato calcolato dalla ALU, in questo caso il valore del registro **rs** più il valore del registro **rt**, **0x3333.3333** + **0x4444.4444** = **0x7777.7777**.
- **JBranch**, indirizzo di un eventuale salto, non usato
- **Z**, flag risultato della ALU uguale a zero, non usato
- **rt**, il contenuto del registro puntato dal campo **irt**, **$4 = 0x4444.4444**, 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.

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. Il **PC** come prima è incrementato di 4.



Sfruttando l'indirizzo **rd** trasportato dalla pipe relativo alla prima istruzione il quinto stadio *scrive indietro* nel register file il contenuto della memoria letto nello stadio precedente.
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 = x** (irrilevante)
- **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 = "Diff" = 01** (ALU deve eseguire una sottrazione),
- **EX.ALUSrc = 0** (usa **rt** come secondo operando)
- **EX.RegDest = x** (irrilevante)
- **PC** = **0x4000.0010** l'indirizzo dell'istruzione successiva alla **beq**
- **rs** (il contenuto del registro puntato dal campo **irs**, **$7**, ex. **0x7777.7777)**
- **rt** (il contenuto del registro puntato dal campo **irt**, **$6**, ex: **0x8888.8888**)
- **offset = 0x0000.0004 (**l'**offset** esteso su 32 bit con segno)
- **irt = 8 (**indice del registro **rt)**
- **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**, è il risultato calcolato dalla ALU, in questo caso il valore del registro **rs** OR il valore dell'offset, **0x6666.6666** + **0x0000.0080** = **0x6666.6746**.
- **JBranch**, indirizzo di un eventuale salto, non usato
- **Z**, flag risultato della ALU uguale a zero, non usato
- **rt**, il contenuto del registro puntato dal campo **irt**, **$6 = 0x6666.6666**, 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**, irrilevante.
- **AluOut**, il risultato ALU precedente, **0x7777.7777**,
- **idest = 2** , indice del registro da usare nell'operazione di WriteBack.

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 esatto il segnale Z provocherà al settimo ciclo di clock la modifica del PC come indicato nel registro JBranch altrimenti il flusso di istruzioni proseguira in maniera lineare.

Ogni istruzione richie 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.