Esercitazione del 28/04/2005 - Soluzioni

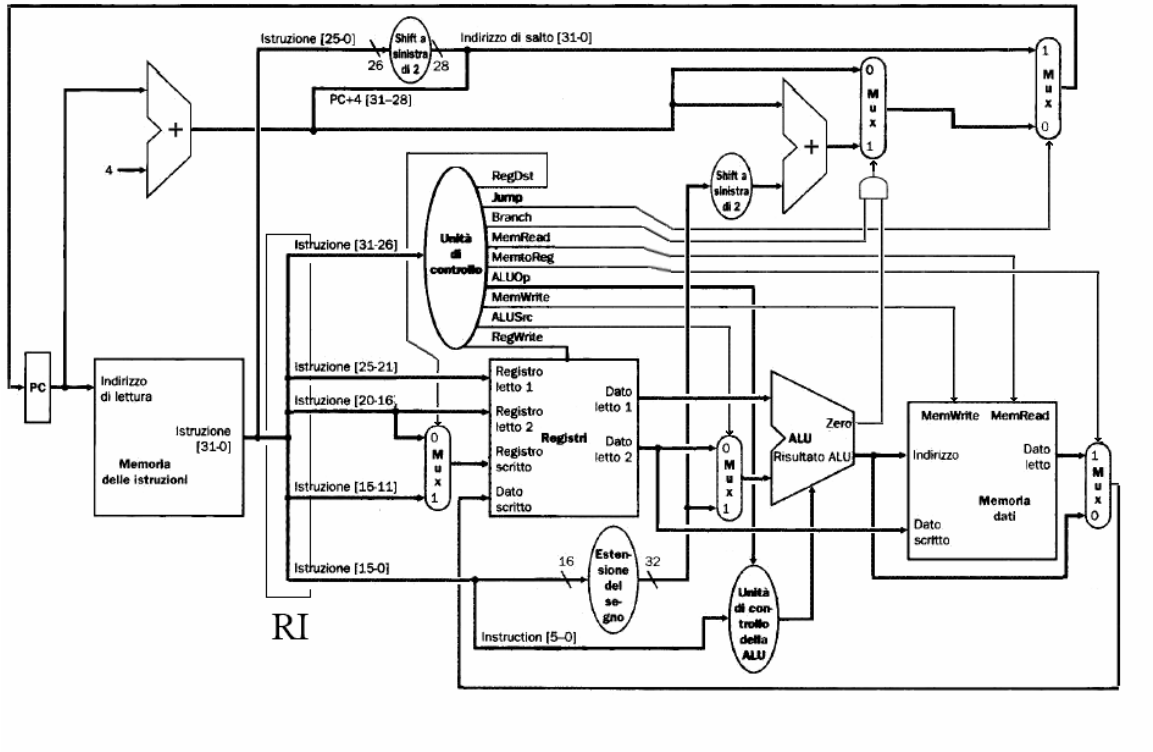
La trama di un istruzione di tipo R del tipo è la seguente:

Op	Rs	Rt	Rd	Sh	Fn
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

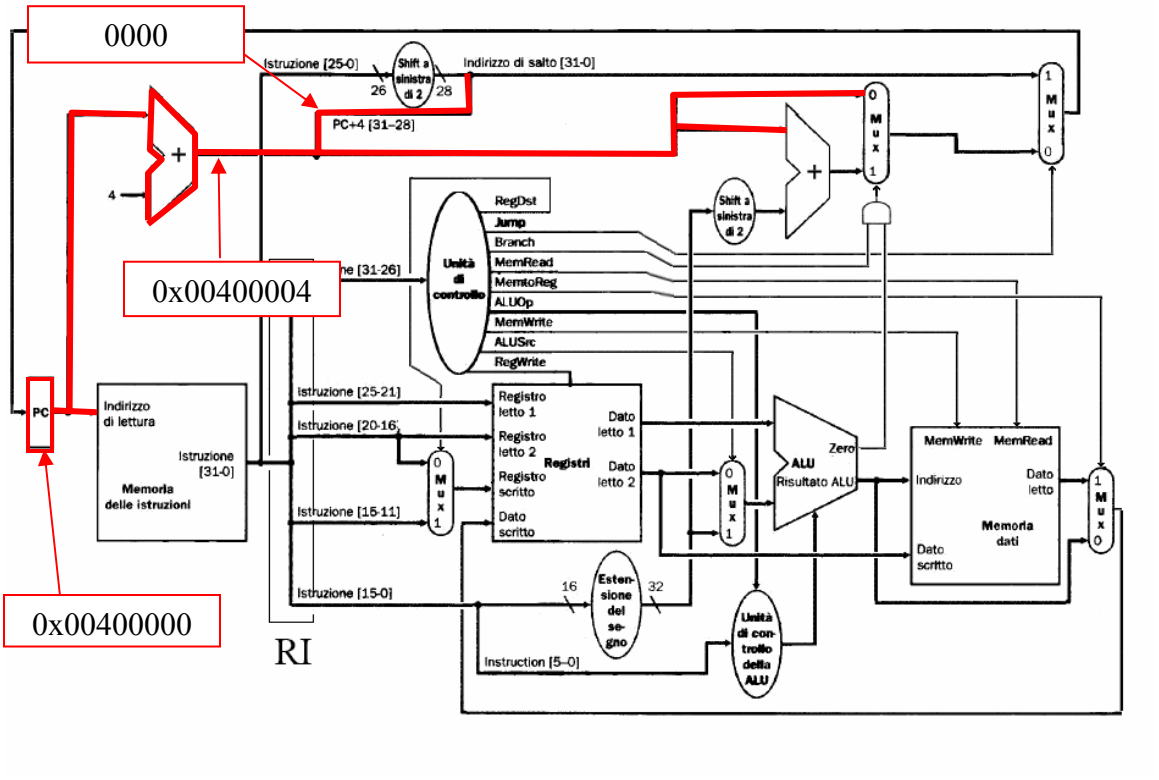
Consideriamo l'istruzione di tipo R: **add $10, $9 , $8** (*add rd, rs, rt*) memorizzata all'indirizzo 0x00400000:

Op	Rs	Rt	Rd	Sh	Fn
000000	01001	01000	01010	00000	100000

cioè la word **0x01285020.** Il circuito di una CPU a ciclo singolo è il seguente:



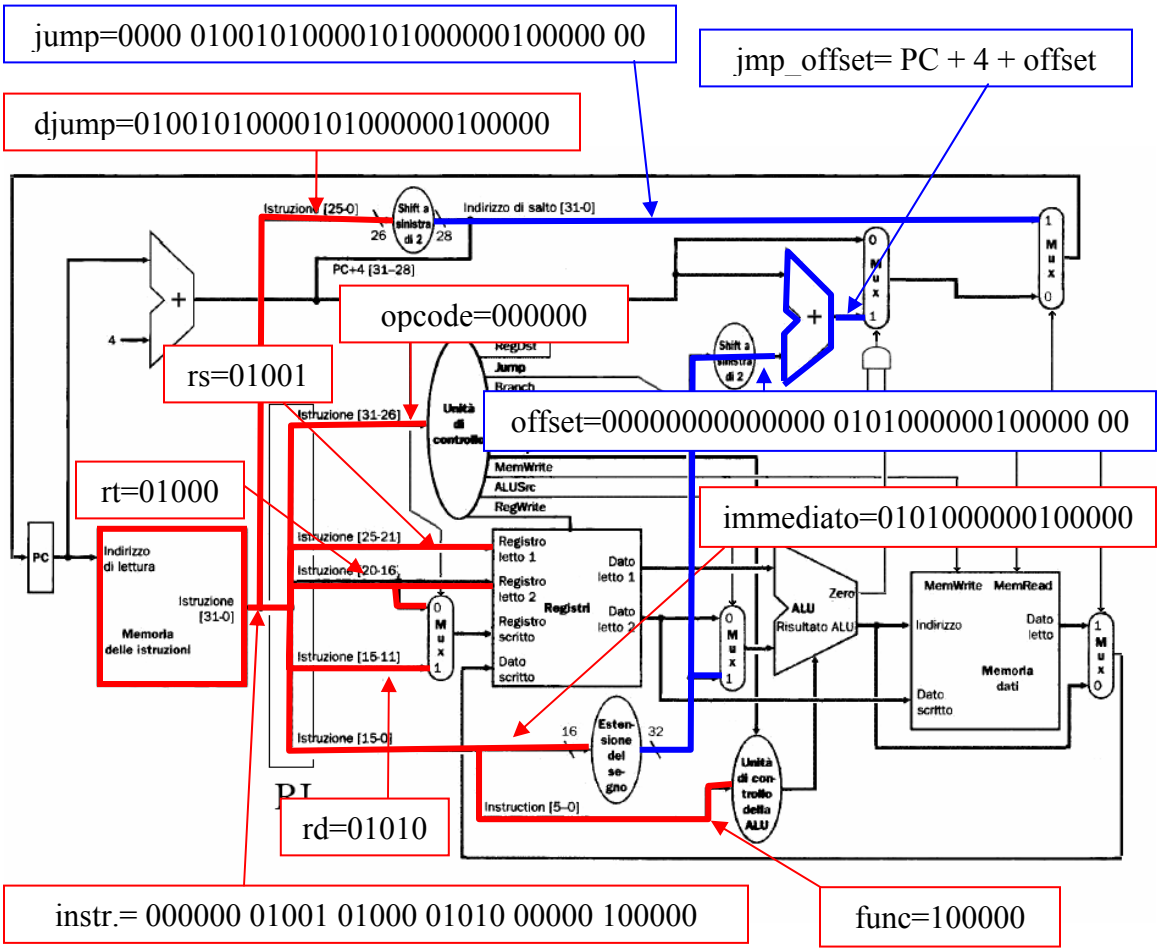
Durante la fase di **fetch** il **Program Counter** (PC) vale (in questo caso) **0x00400000** ed attraverso il bus indirizzi della memoria delle istruzioni punta alla istruzione da caricare. Lo stesso dato giunge al sommatore degli indirizzi che prepara l'indirizzo dell'istruzione immediatamente seguente, cioè in questo caso **0x00400004**. I quattro bit più significativi vengono anche usati per completare l'indirizzo di salto proveniente dalla eventuale istruzione **j** (in questo caso, poiché l'istruzione è una **add**, l'indirizzo generato sarà inconsistente).



Fase di fetch: preparazione dell'indirizzo seguente

L’istruzione viene letta dalla memoria e presentata ai vari moduli per la decodifica.

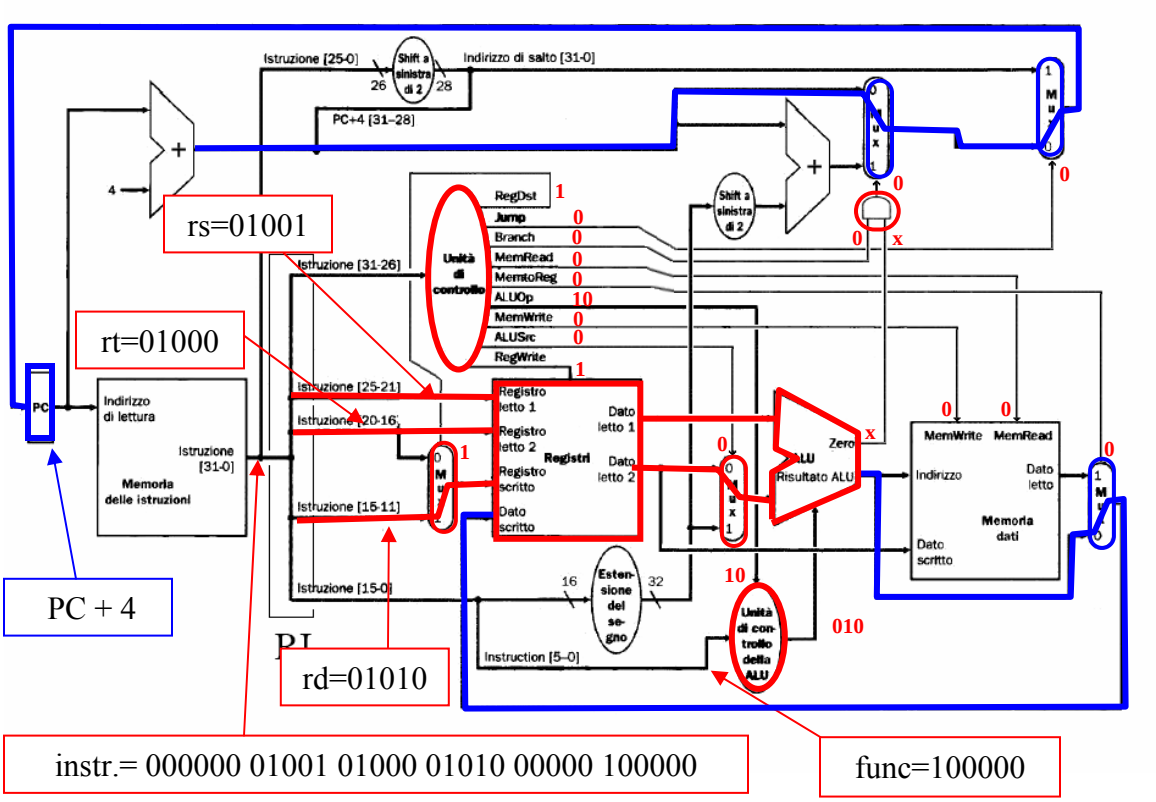
add $10, $9 , $8 = 000000 01001 01000 01010 00000 100000 = 0x01285020



La trama dell’istruzione viene smontata e riassemblata in vari modi per costruire tutti i componenti che si possono costruire senza sapere ancora di che tipo di istruzione si tratta. Alcuni di questi componenti, a seconda del tipo di istruzione, non verranno utilizzati, come ad esempio il dato immediato ricavato dai 16 bit bassi dell’istruzione nel caso di un istruzione di tipo R. I componenti generati sono:

- Usando i bit 25-0 viene costruito l’ipotetico indirizzo **jump** di salto di un’istruzione di tipo **J.**
- Usando i bit 5-0 viene estratto il codice **function** di un istruzione di tipo **R.**
- Usando i bit 15-0 viene costruito l’ipotetico dato immediato in un istruzione di tipo **I.** Lo stesso dato viene usato per generare l’offset e l’indirizzo di arrivo di un ipotetico salto relativo condizionato.
- Usando i bit 25-11 vengono costruiti gli indici relativi a **rs**, **rt** ed **rd.** Gli indici relativi ad **rs** ed **rt** vengono direttamente presentati al register file per selezionare i due registri da leggere.

A questo punto è possibile decodificare tramite l'unità di controllo di che istruzione si tratta e stabilire i percorsi dei dati all'interno della CPU.



add ha come opcode **0 (** comando tipo-r **).** L'unità di controllo principale predispone i segnali **RegDst=1**, **ALUSrc=0, MemToReg=0** e **RegWrite=1** in modo da portare il risultato dell'ALU sui registri **rs** ed **rt** nel registro **rd.** Le operazione sulla memoria dati vengono disabilitate con i segnali **MemWrite=0** e **MemRead=0** mentre la prossima istruzione da eseguire, in questo caso la successiva nella memoria istruzioni è selezionata mandando al PC il risultato del primo sommatore, **Jump=0** e **Branch=0**. La scelta dell'operazione da eseguire nell'ALU viene delegata all'unità di controllo dell'ALU tramite il segnale **ALUOp=10** poiché tale informazione è presente nel campo func dell'istruzione. L'unità di controllo dell'ALU dato che **func=100000** imposta il comando **ALU=010** corrispondente all'operazione di somma. Il segnale **zero** della ALU viene ignorato.

La trama di un istruzione di tipo I del tipo è la seguente:

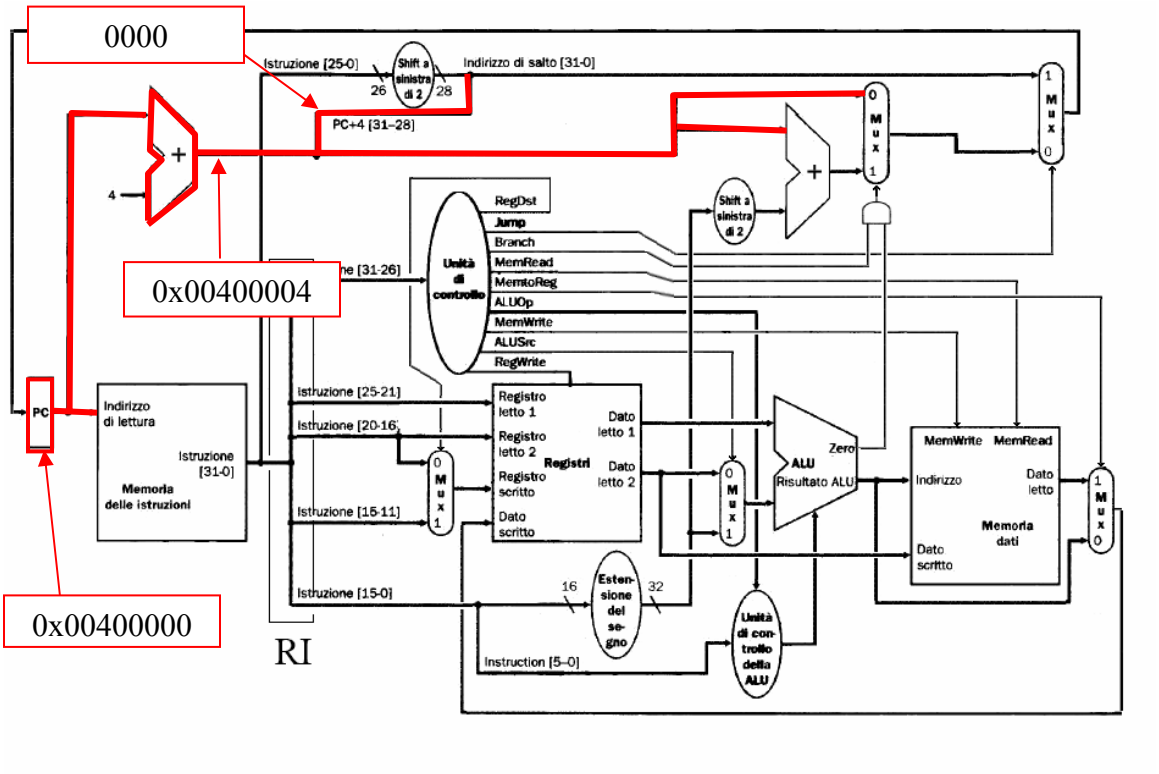
Op	Rs	Rt	Immediato
6 bit	5 bit	5 bit	16 bit

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

Op	Rs	Rt	offset
100011	01000	01001	0000000000000011

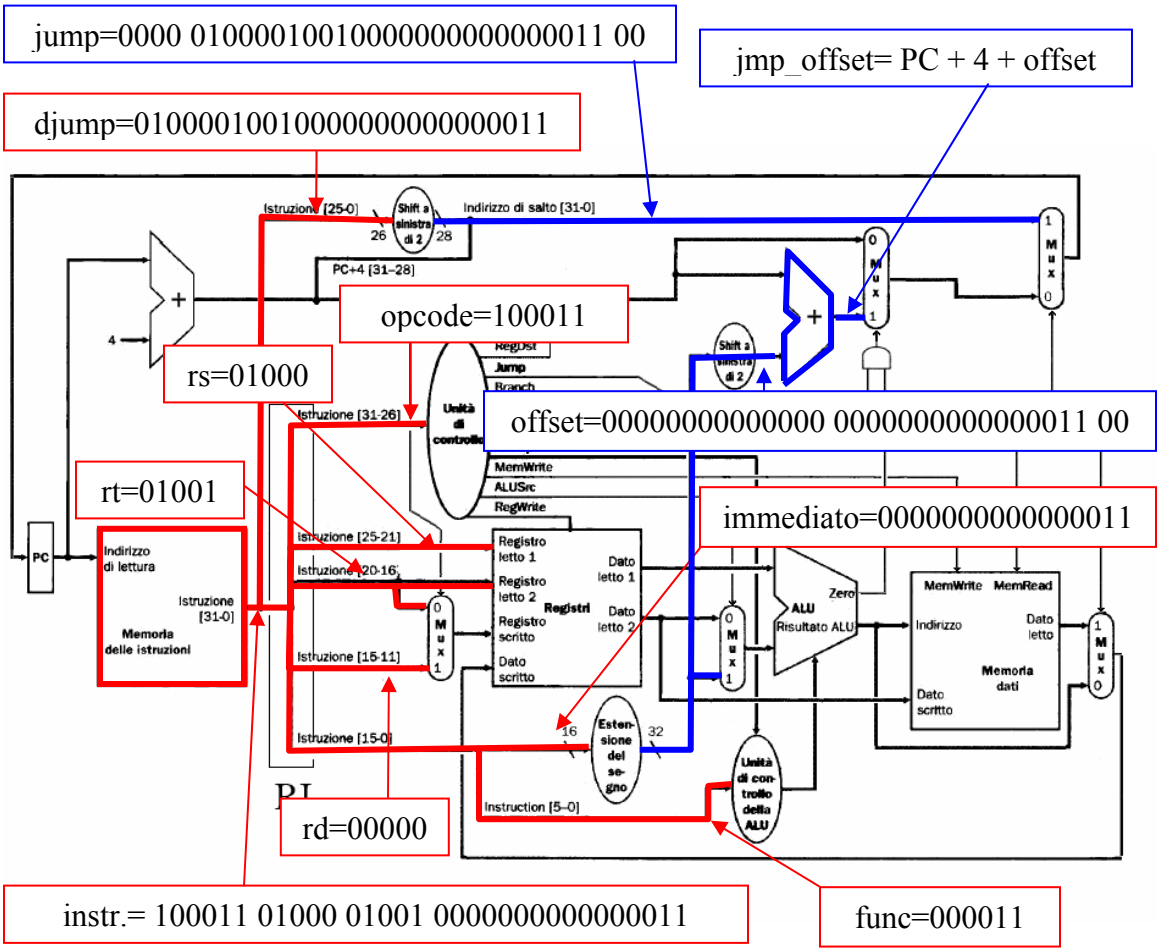
cioè la word **0x8D090003.** Il circuito di una CPU a ciclo singolo è il seguente:

La prima parte della fase di **fetch** è uguale all'esempio precedente: il **Program Counter** (PC) vale **0x00400000,** viene preparato l'indirizzo della prossima istruzione **0x00400004**.



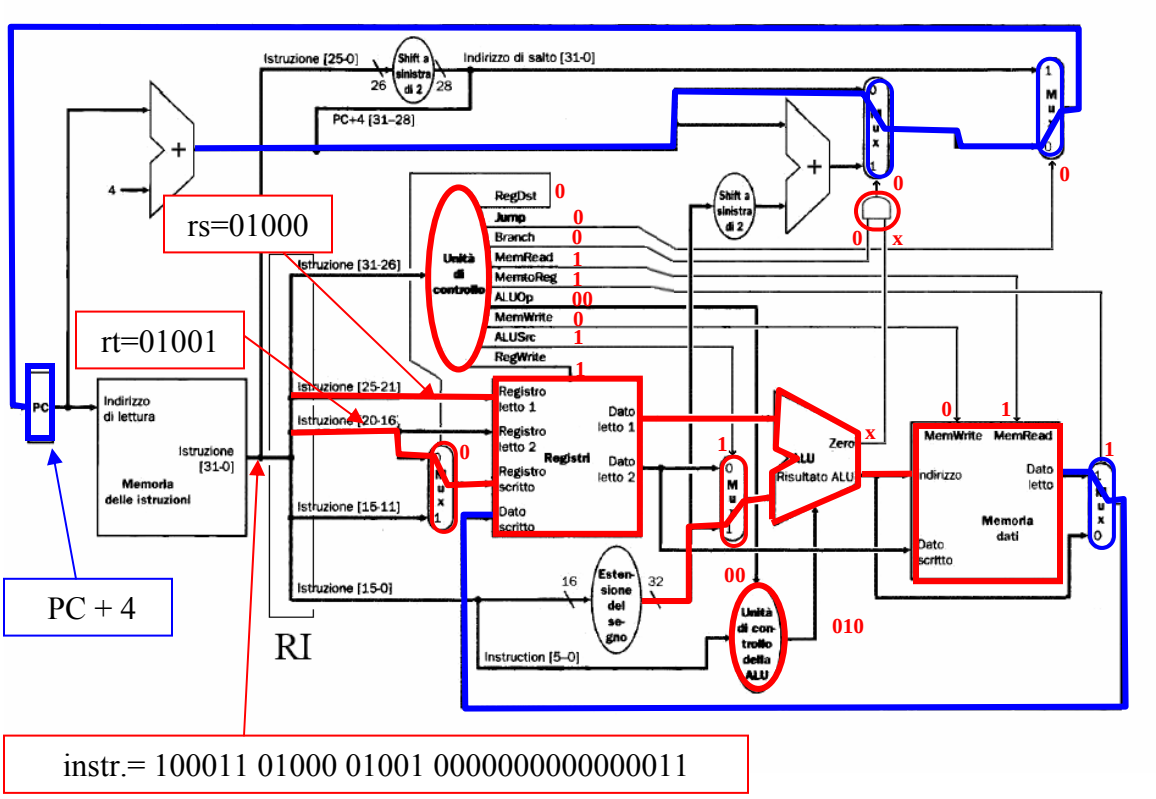
L’istruzione viene letta dalla memoria e presentata ai vari moduli per la decodifica:

lw $9, 3($8) = 100011 01000 01001 0000000000000011= 0x8D090003



Come nell’esempio precedente la trama dell’istruzione viene smontata e riassemblata in vari modi per costruire tutti i componenti che si possono costruire senza sapere ancora di che tipo di istruzione si tratta: **jump, function, immediato**, **rs**, **rt** ed **rd**, l’offset e l’indirizzo di arrivo di un salto condizionale.

A questo punto è possibile decodificare tramite l'unità di controllo di che istruzione si tratta e stabilire i percorsi dei dati all'interno della CPU.



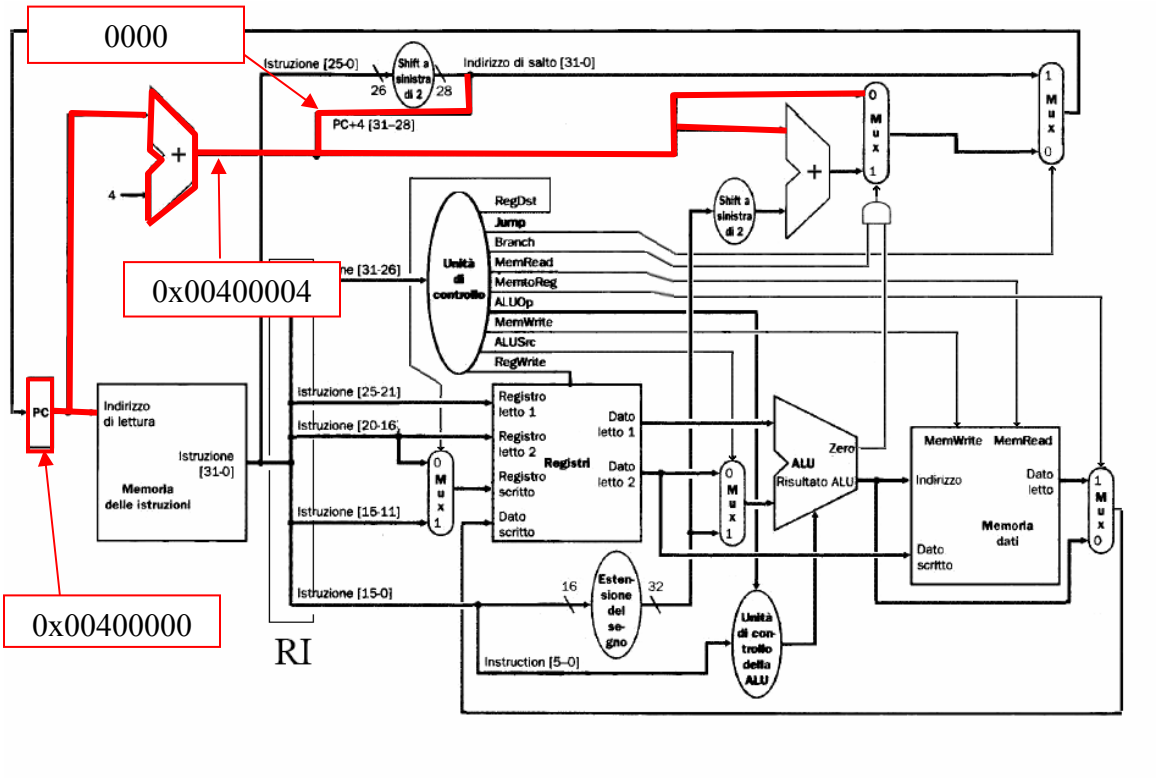
lw ha come opcode **100011 (** comando tipo load **).** L'unità di controllo principale predispone i segnali **RegDst=0**, **ALUSrc=1, MemToReg=1, MemRead=0** e **RegWrite=1** in modo da portare il dato letto dalla memoria dati nel registro **rd.** L'indirizzo del dato da leggere viene gerato dalla ALU sommando il valore di **rs** ed il dato immediato calcolato con i 16 bit meno significativi dell'istruzione**.** Le operazioni di scrittura sulla memoria dati vengono disabilitate con il segnale **MemWrite=0** mentre la prossima istruzione da eseguire, in questo caso la successiva, è selezionata mandando al PC il risultato del primo sommatore, **Jump=0** e **Branch=0**. Poiché l'operazione da compiere nella ALU è sempre la somma nel caso di istruzioni di tipo load/store, l'unità di controllo principale tramite il segnale **ALUOp=00** comunica all'unità di controllo dell'ALU di selezionare l'operazione di somma, **ALUs=010**. Il valore inconsistente di **func** viene ignorato cosi come il segnale **zero** della ALU.

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

Op	Rs	Rt	offset
101011	**01000**	**01001**	**0000000000000011**

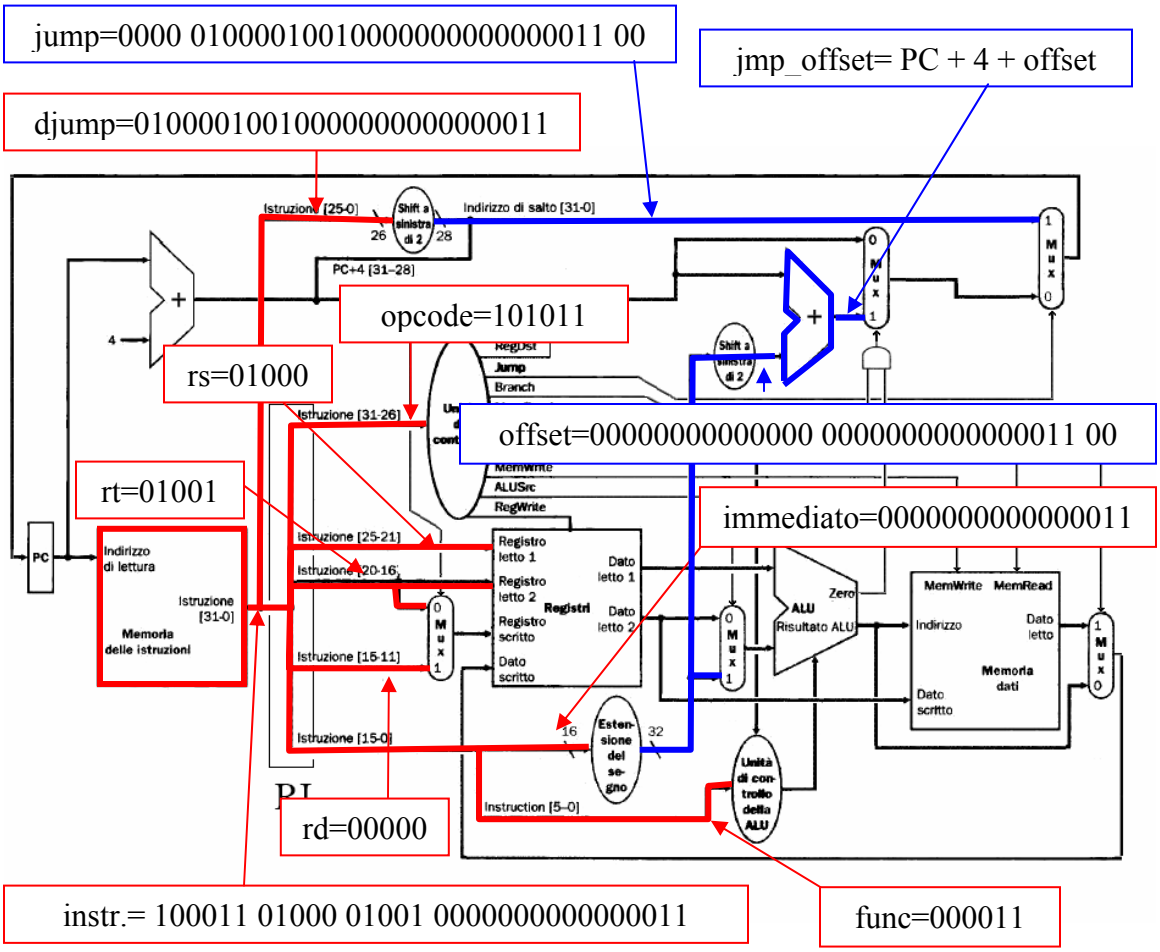
cioè la word **0xAD090003.** Il circuito di una CPU a ciclo singolo è il seguente:

La prima parte della fase di **fetch** è uguale all'esempio precedente: il **Program Counter** (PC) vale **0x00400000,** viene preparato l'indirizzo della prossima istruzione **0x00400004**.



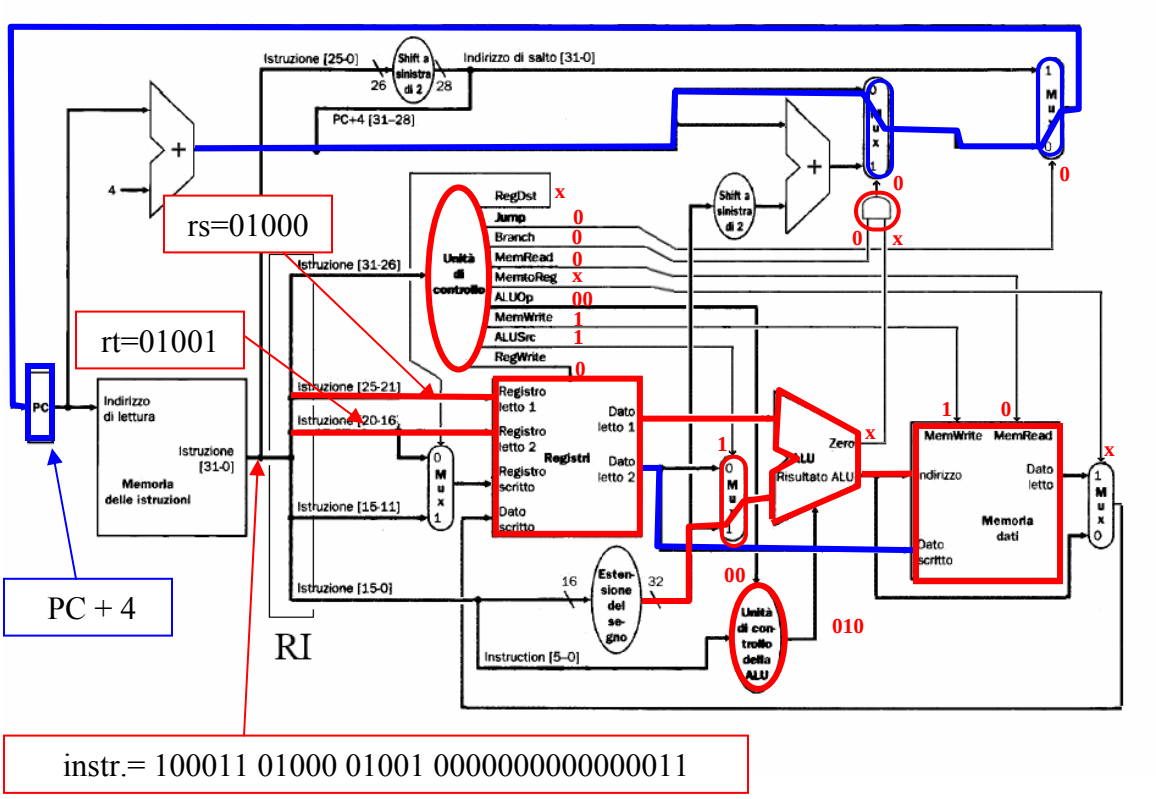
L’istruzione viene letta dalla memoria e presentata ai vari moduli per la decodifica:

sw $9, 3($8) = 101011 01000 01001 0000000000000011= 0xAD090003



I valori di **jump, function, immediato**, **rs**, **rt** ed **rd**, l’offset e l’indirizzo di arrivo di un salto condizionale sono uguali all’esempio dell’istruzione di load.

A questo punto è possibile decodificare tramite l'unità di controllo di che istruzione si tratta e stabilire i percorsi dei dati all'interno della CPU.



sw ha come opcode **101011 (** comando tipo store **).** L'unità di controllo principale predispone i segnali **ALUSrc=1, MemWrite=1** in modo da portare il dato letto dal registro **rt** nella memoria dati**.** L'indirizzo del dato da scrivere viene generato dalla ALU sommando il valore di **rs** ed il dato immediato calcolato con i 16 bit meno significativi dell'istruzione**.** Le operazioni di scrittura sul file register vengono disabilitate con il segnale **RegWrite=0**. Questo rende i segnali **RegDst=X e MemToReg=X** ininfluenti. La prossima istruzione da eseguire, in questo caso la successiva, è selezionata con **Jump=0** e **Branch=0**. In maniera analoga alle istruzioni di tipo load, l'unità di controllo principale tramite il segnale **ALUOp=00** comunica all'unità di controllo dell'ALU di selezionare l'operazione di somma, **ALUs=010**. Il valore inconsistente di **func** viene ignorato cosi come il segnale **zero** della ALU.

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

Op	Rs	Rt	doffset
6 bit	5 bit	5 bit	16 bit

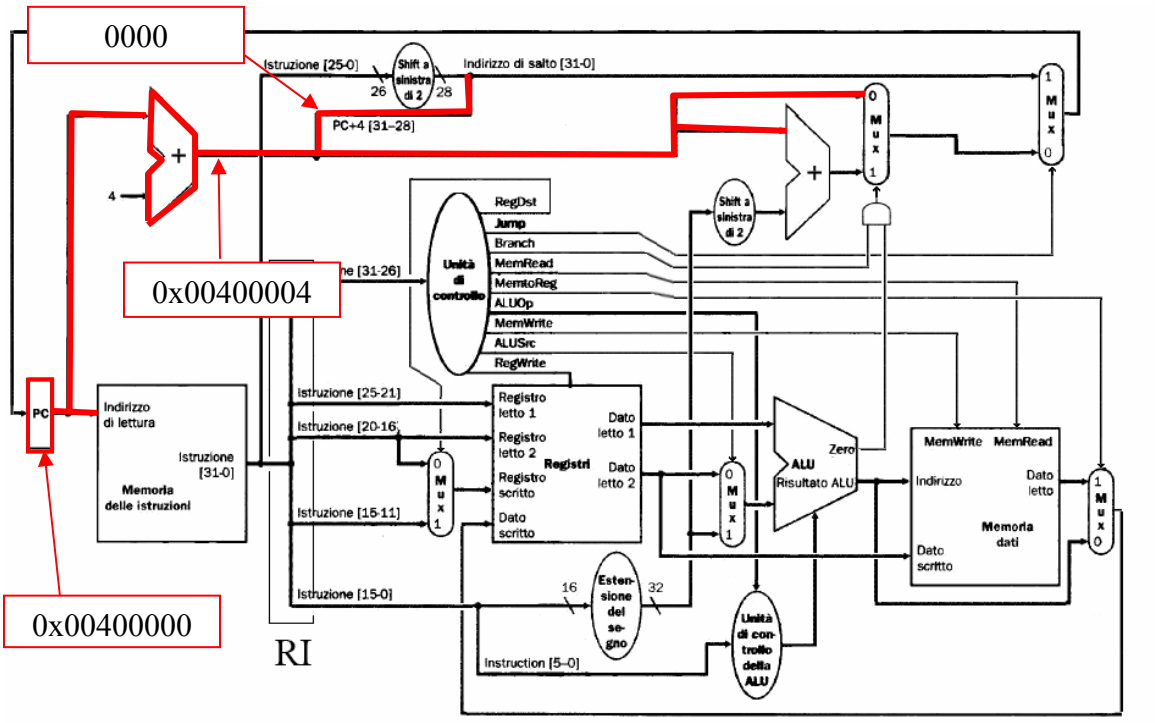
*Nota: L'offset è espresso in **byte**. Dato che nel MIPS gli indirizzi di salto sono sempre allineati alle word, ne segue che l'offset sarà sempre un multiplo di 4, cioè i due bit più significativi dell'offset saranno sempre uguali a 0. Al momento di codificare in linguaggio macchina il branch è quindi conveniente memorizzare nella word l'offset diviso per 4 (shift a destra di 2) cioè memorizzare i 16 bit 18-3 dell'offset (**doffset**). Questo permette di avere una più ampia zona di arrivo per i salti relativi. Un discorso analogo può essere fatto per le istruzioni di salto incondizionato.*

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

Op	Rs	Rt	doffset
000101	01000	01001	0000000000000011

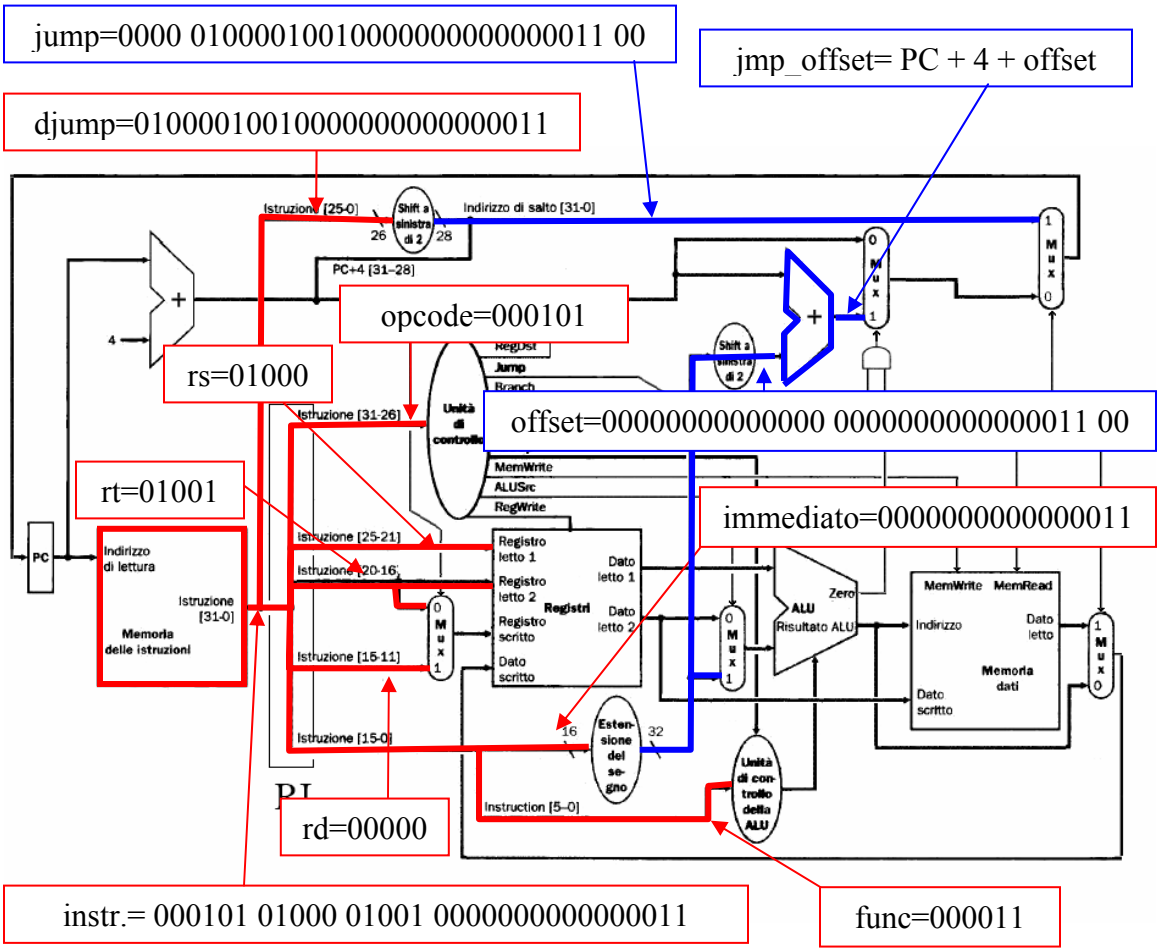
cioè la word **0x15090003.** Il circuito di una CPU a ciclo singolo è il seguente:

La prima parte della fase di **fetch** è uguale agli esempi precedenti: il **Program Counter** (PC) vale **0x00400000,** viene preparato l'indirizzo della prossima istruzione **0x00400004**.



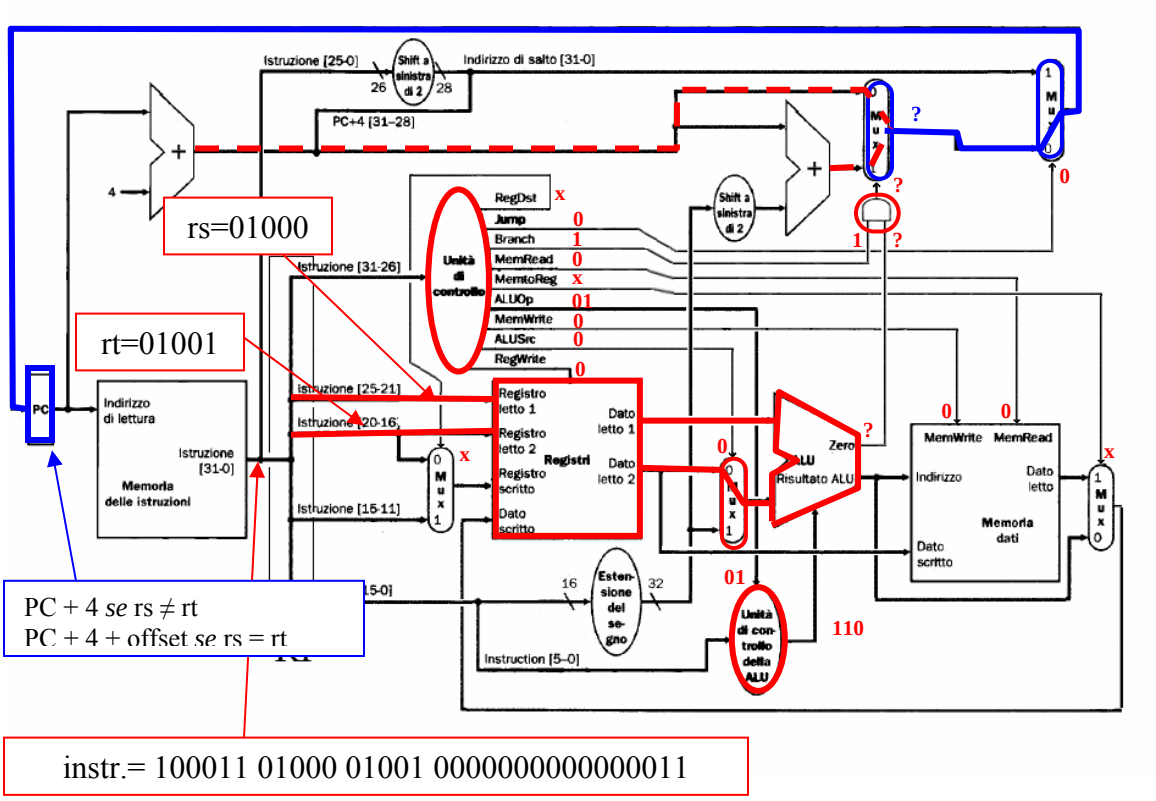
L’istruzione viene letta dalla memoria e presentata ai vari moduli per la decodifica:

beq $8, $9 , 12 = 000101 01000 01001 0000000000000011= 0x15090003



Come nell’esempio precedente la trama dell’istruzione viene smontata e riassemblata in vari modi per costruire tutti i componenti che si possono costruire senza sapere ancora di che tipo di istruzione si tratta: **jump, function, immediato**, **rs**, **rt** ed **rd**, l’**offset** e l’indirizzo di arrivo di un salto condizionale.

A questo punto è possibile decodificare tramite l'unità di controllo di che istruzione si tratta e stabilire i percorsi dei dati all'interno della CPU.



beq ha come opcode **000101 (** comando tipo branch **).** L'unità di controllo principale predispone il segnale **ALUSrc=0** in modo da portare il valore dei registri **rs** e **rt** verso la ALU. Le operazioni sulla memoria dati vengono disabilitate con il segnale **MemWrite=0 MemRead=0** cosi ccome viene disabilitata ogni operazione di scrittura sul register file con **RegWrite=0.** Questo rende irrilevante il valore di **RegDst=X**, **MemToReg=X.** Poiché l'operazione da compiere nella ALU quando si realizzano dei confronti è sempre la sottrazione, l'unità di controllo principale tramite il segnale **ALUOp=01** comunica all'unità di controllo dell'ALU di selezionare l'operazione di sottrazione, **ALUs=110**. Il valore inconsistente di **func** viene ignorato. La prossima istruzione da eseguire viene selezionata tramite il segnale **zero** dell'ALU: se i due dati estratti sono uguali (e quindi la loro differenza vale 0) allora il segnale **zero** è posto dalla ALU ad uno, se sono diversi **zero** vale 1. Questo segnale giunge attraverso la porta and controllata da **Branch=1** al mux posto dopo il sommatore dell'offset: se zero vale 1 allora il PC viene settato al valore **PC+4+offset** calcolato in precedenza altrimenti viene settato a PC+4. Occorre che **Jump=0.**

La trama di un istruzione di tipo **J** del tipo è la seguente:

Op	**daddress**
6 bit	**26 bit**

*Nota: L'indirizzo di un salto assoluto, essendo di 32 bit non può stare totalmente in una sola istruzione. Tolto il codice operativo dell'istruzione rimangono solo 26 bit disponibili. Dato che nel MIPS gli indirizzi di salto sono sempre allineati alle word, ne segue che l'indirizzo di un salto sarà sempre un multiplo di 4, cioè i due bit più significativi saranno sempre uguali a 0. Al momento di codificare in linguaggio macchina l'indirizzo di salto è quindi conveniente memorizzare nella word l'indirizzo diviso per 4 (shift a destra di 2) e quindi troncato nei 26 bit meno significati, cioè memorizzare i 26 bit 28-3 (**daddress**). Al momento di ricostruire l'indirizzo i 4 bit più significativi mancanti verranno rimpiazzati dai 4 bit più significativi del PC corrente. Questo permette salti al più 64M in avanti o indietro rispetto alla posizione attuale. per salti piu lungni (long jump) è necessario prima caricare l'indirizzo completo in un registro e quindi utilizzare l'istruzione jr rd (jump register).*

Consideriamo l'istruzione di tipo **J**: **j 0x00400010** (*j address*) memorizzata all'indirizzo 0x00400000. L'indirizzo da mappare nell'istruzione in binario è:

0x00400010 = ~~0000~~ 0000 0100 0000 0000 0000 0001 00~~00~~

da cui, troncando i primi 4 bit e gli ultimi due, si ricava l'istruzione:

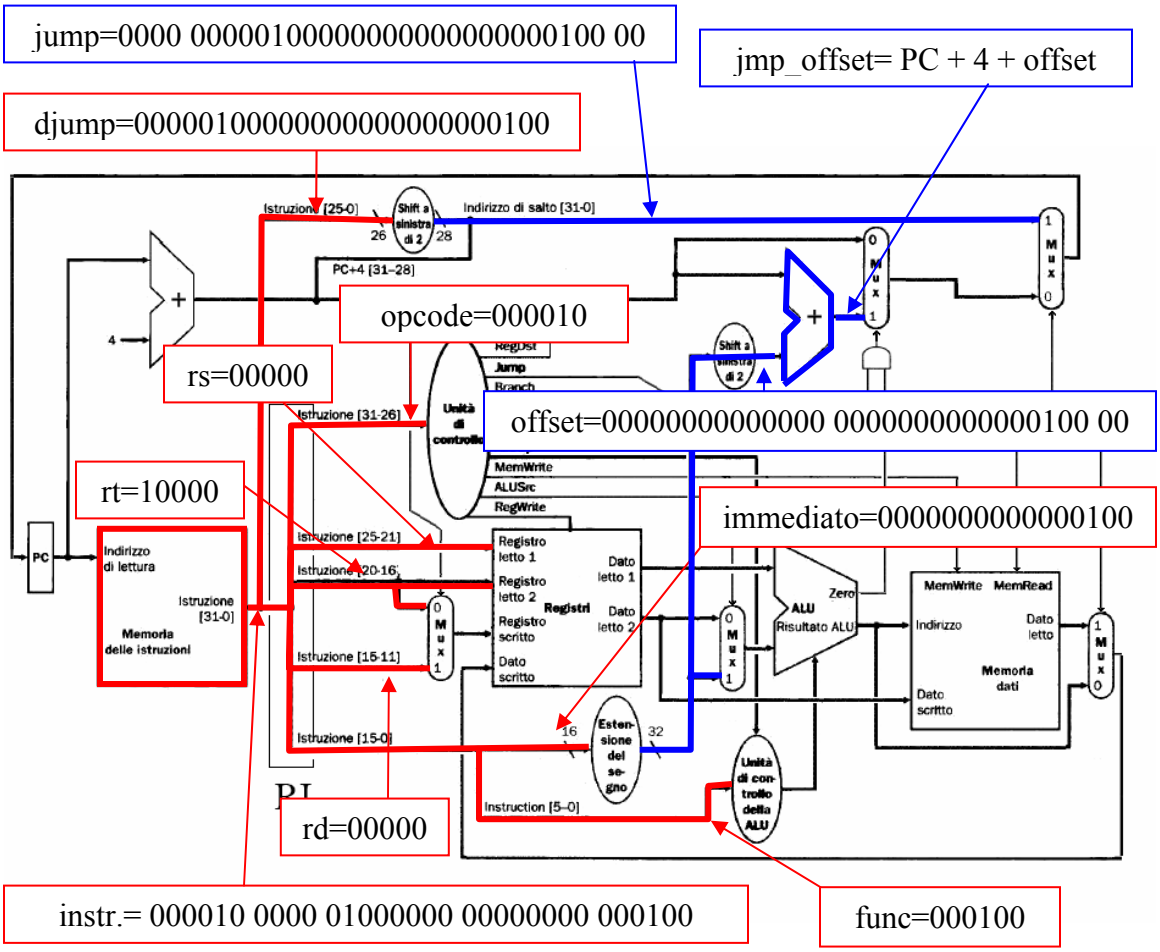
Op	**daddress**
000010	**0000 01000000 00000000 000100**

cioè la word **0x08100004.**

La prima parte della fase di **fetch** è uguale a tutti gli esempi precedenti: il **Program Counter** (PC) vale **0x00400000,** viene preparato l'indirizzo della prossima istruzione **0x00400004** (omettiamo la figura per brevità).

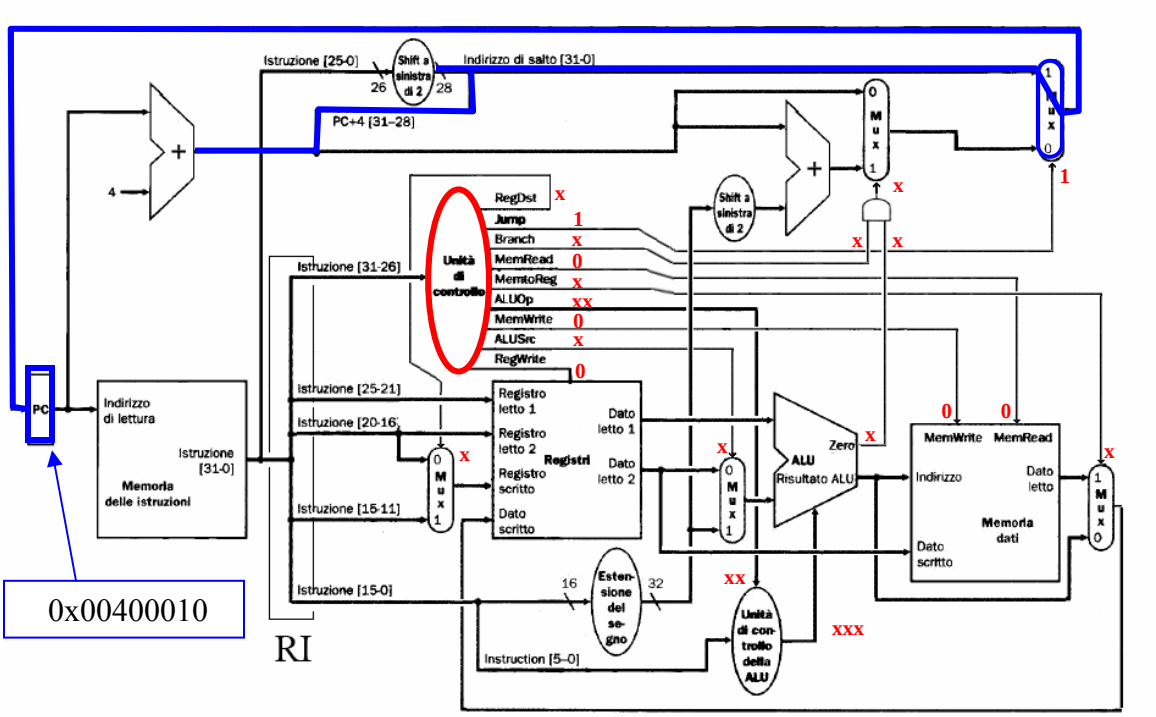
L’istruzione viene letta dalla memoria e presentata ai vari moduli per la decodifica:

j 0x00400010 = 000010 0000 01000000 00000000 000100= 0x08100004



Come nell’esempio precedente la trama dell’istruzione viene smontata e riassemblata in vari modi per costruire tutti i componenti che si possono costruire senza sapere ancora di che tipo di istruzione si tratta: **jump, function, immediato**, **rs**, **rt** ed **rd**, l’offset e l’indirizzo di arrivo di un salto condizionale.

A questo punto è possibile decodificare tramite l'unità di controllo di che istruzione si tratta e stabilire i percorsi dei dati all'interno della CPU.



instr.= 100011 01000 01001 0000000000000011

j ha come opcode **000010 (** comando tipo J **).** L'unità di controllo principale predispone il segnale **Jump=1** in modo da caricare il nuovo indirizzo nel PC. L'indirizzo è ottenuto dal campo **daddress** dell'istruzione shiftato di 2 a sinistra e completato con i due bit più significativi di **PC+4**. Tutte le operazioni sulla memoria dati e sul register file sono disabilitate con i segnali **MemWrite=0, MemRead=0** e **RegWrite=0. I segnali RegDst=X**, **ALUSrc=X, MemToReg=X, ALUOp=XX, ALUs=XXX e Branch=X** diventano quindi irrilevanti. Il valore inconsistente di **func** viene ignorato cosi come il segnale **zero** della ALU.

Diamo come riepilogo le tabelle di associazione della CPU mono-ciclo utilizzate negli esempi:

Istruzione	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Jump	Branch	ALUOp1	ALUOp2
Tipo-R	1	0	0	1	0	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0	0
sw	x	1	x	0	0	1	0	0	0	0
beq	x	0	x	0	0	0	0	1	0	1
j	x	x	x	0	0	0	1	x	x	x

Tabella di verità (semplificata) relativa all'unità di controllo principale

ALUOp		Campo func						
ALUOp1	ALUOp2	F5	F4	F3	F2	F1	F0	ALUs
0	0	x	x	x	x	x	x	010
x	1	x	x	x	x	x	x	110
1	x	x	x	0	0	0	0	010
1	x	x	x	0	0	1	0	110
1	x	x	x	0	1	0	0	000
1	x	x	x	0	1	0	1	001
1	x	x	x	1	0	1	0	111

Tabella di verità (semplificata) relativa all'unità di controllo dell'ALU

ALUs	Operazione
000	AND
001	OR
010	somma
110	sottrazione
111	set on less then

Tabella di corrispondenza tra codice ALUs e relatuva operazione ALU