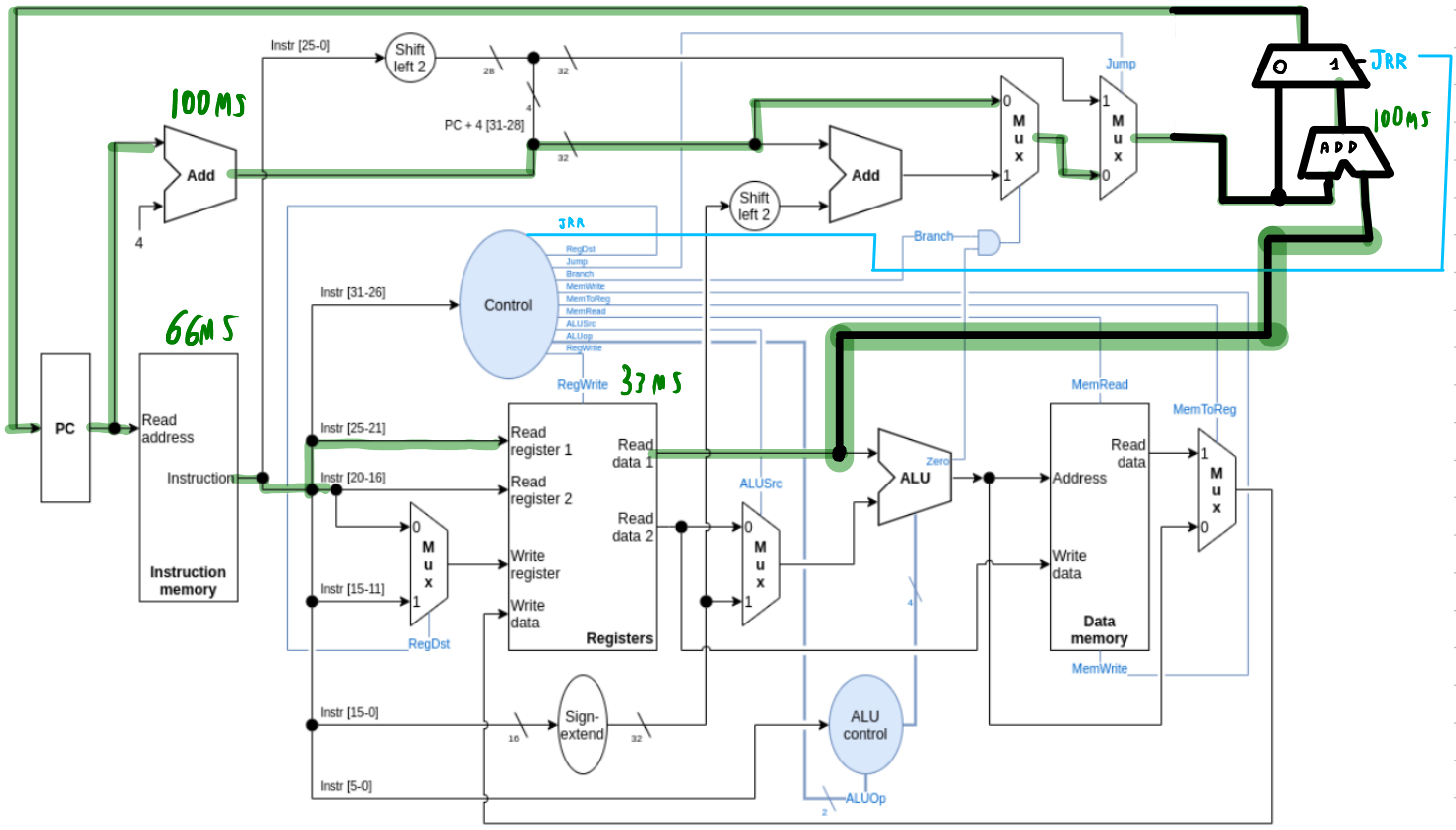
Esercizio per casa: jrr

Aggiungere alla CPU l'istruzione **jrr rs** (Jump Relative to Register)
di tipo R, che salta all'indirizzo (relativo al PC) contenuto nel registro **rs**
Ovvero che esegue come prossima istruzione quella che si trova all'indirizzo
PC+4+Registri[rs]

a) Modificare lo schema per realizzare l'istruzione
b) Indicare tutti i segnali di controllo che la CU deve generare
c) Calcolare il tempo di esecuzione della istruzione assumendo che:
Accesso a memorie = 66ns, accesso ai registri = 33ns, ALU e sommatori = 100ns



JUMP = BRANCH = REGWRITE = MEMWRITE = MEMREAD = 0

JRR = 1

TEMPO = $\boxed{\boxed{66 \mid 33}\ 100}$ = 200 MS

Aggiungere l'istruzione jral

Vogliamo aggiungere l'istruzione di tipo R

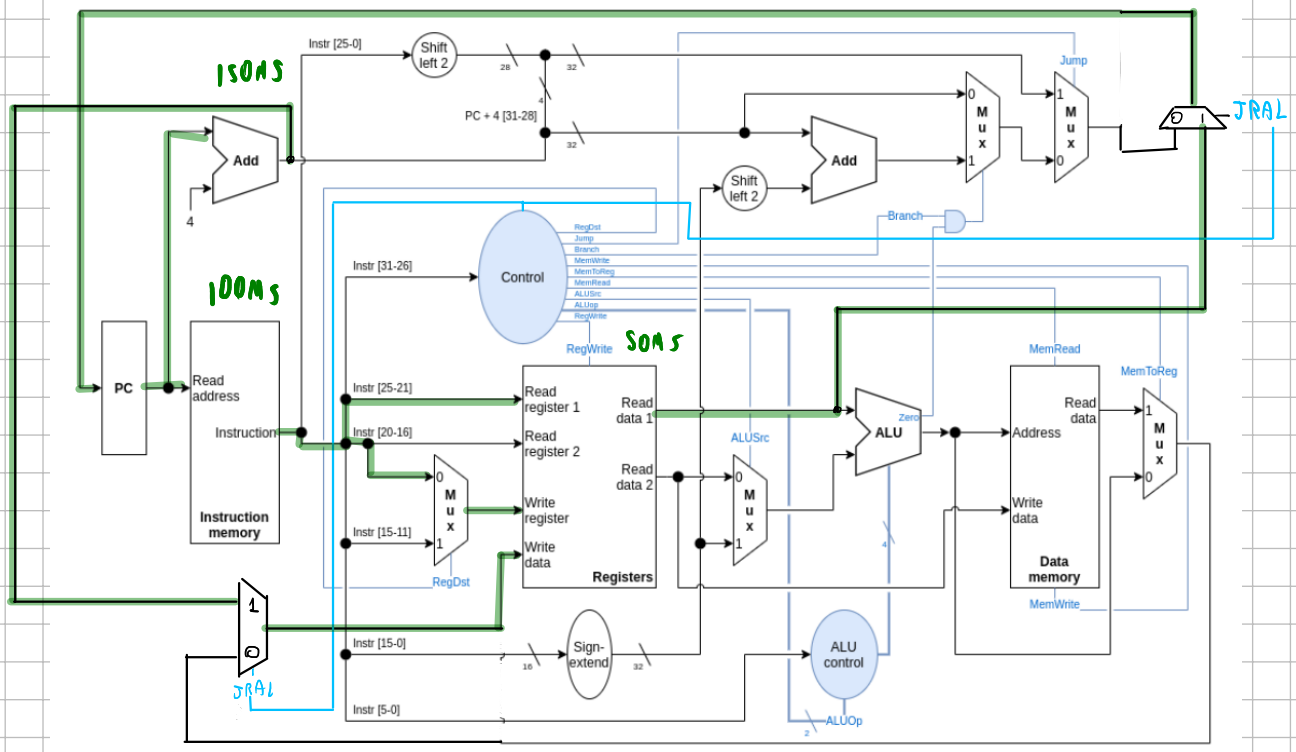
`jral $rs, $rt` (Jump to **R**egister **a**nd **L**ink to rt)

che salta incondizionatamente all'indirizzo contenuto nel registro **rs** e memorizza nel registro **rt** l'indirizzo della istruzione successiva.

a) Modificare lo schema per realizzare l'istruzione

b) Indicare tutti i segnali di controllo che la CU deve generare

c) Calcolare il tempo di esecuzione della istruzione assumendo che:
Accesso a memorie = 100ns, accesso ai registri = 50ns, ALU e sommatori = 150ns



JRAL = REGWRITE = 1

REGDST = MEMREAD = MEMWRITE = 0

GLI ALTRI SEGNALI SONO DONT CARE

TEMPO = $\begin{array}{|c|c|}\hline 150 & \\ \hline 100 & 50 \\ \hline \end{array}$ 50 = 200 ns

Esercizio per casa (nuovo): vj

Si vuole aggiungere alla CPU l'istruzione vectorised jump (**vj**), di tipo **I** e sintassi assembly

```
vj $indice, vettore
```

che salta all'indirizzo contenuto nell'elemento $indice-esimo del *vettore* *di word*.

Esempio: se in memoria si è definito staticamente il

```
vettore: .word 16, 24, 312, 44
```

e al registro **$t0** è assegnato il valore **3** allora

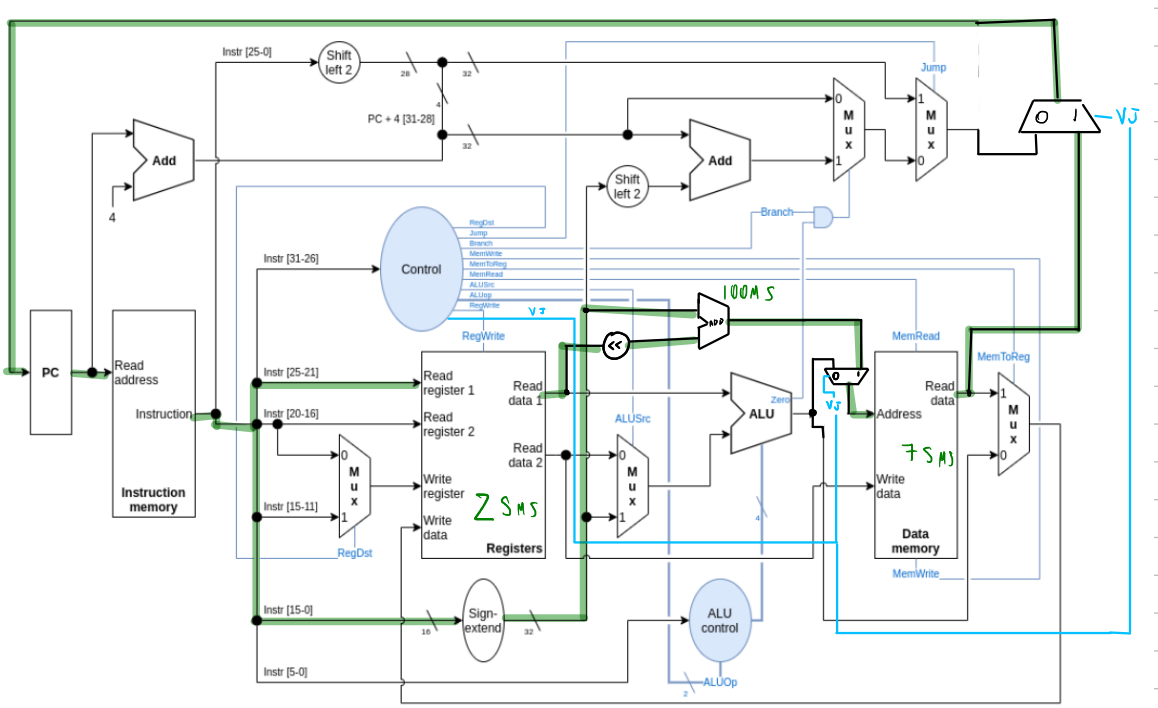
```
vj $t0, vettore
```

salterà all'indirizzo **44** (che è l'elemento con indice 3 del vettore)

a) si disegnino le modifiche necessarie a realizzare la funzione, aggiungendo tutti gli eventuali MUX, segnali di controllo, bus, ALU e sommatori (ecc) che ritenete necessari.

b) indicate i valori di tutti i segnali di controllo, in modo da eseguire l'istruzione **vj**.

c) tenendo conto che il tempo di accesso ai registri (sia in lettura che scrittura) è di **25ns**, l'accesso alla memoria impiega **75ns**, la ALU e i sommatori impiegano **100ns** e ignorando gli altri ritardi, calcolate il tempo di esecuzione minimo della istruzione **vj** e indicate se è necessario aumentare il periodo di clock della CPU per poter svolgere questa nuova istruzione



VJ = MEMREAD = 1

REGWRITE = MEMWRITE = 0

GLI ALTRI SEGNALI SONO DON'T CARE

TEMPO = | 25 | 75 | 100 | = 200 MS

Considerare l'architettura MIPS a ciclo singolo nella figura in basso (ed in allegato).
Si vuole aggiungere l'istruzione di tipo I
`diff_and_store $rs, $rt, addr`
(calcola la differenza in valore assoluto dei registri rs ed rt e salva il risultato in memoria, all'indirizzo addr)
che equivale al seguente frammento di codice

```
          sub $rs, $rs, $rt          # (a)
          bge $rs, $0, save          # (b)
          sub $rs, $0, $rs           # (c)
    save: sw  $rs, addr              # (d)
```

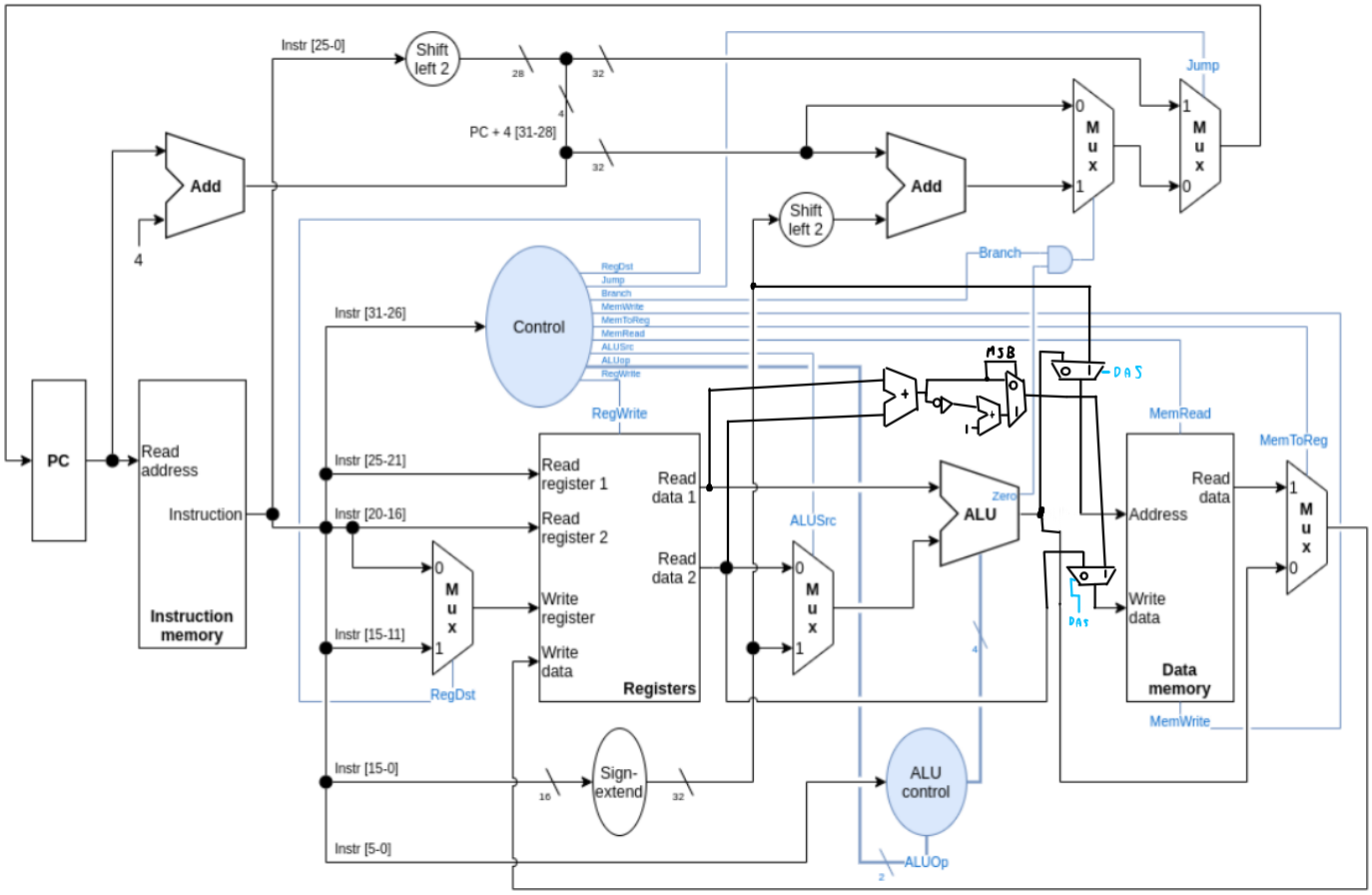
ossia (a) esegue la sottrazione tra `rs` ed rt, (b,c) se il risultato è negativo ne cambia il segno, infine (d) salva il valore risultante in memoria all'indirizzo indicato da `addr`. In particolare, `addr` è la parte immediata dell'istruzione e viene usato come indirizzo *assoluto* di destinazione.

1) Mostrare le **modifiche all'architettura** della CPU MIPS, avendo cura di aggiungere eventuali altri componenti necessari a realizzare l'istruzione. A tal fine, si può alterare la stampa del diagramma architetturale oppure ridisegnare i componenti interessati dalla modifica, avendo cura di indicare i fili di collegamento e tutti i segnali entranti ed uscenti. Indicare inoltre sul diagramma i **segnali di controllo** che la CU genera *per realizzare l'istruzione*.

2) Indicare il contenuto in bit della word che esprime l'istruzione
`diff_and_store $10, $11, 64`
compilando la tabella sottostante (assumiamo che lo *OpCode* di diff_and_store sia 0x38).

1	1	1	0	0	0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3) Supponendo che l'accesso alle **memorie** impieghi **200 ns**, l'accesso ai **registri 50 ns**, le operazioni dell'**ALU e** dei **sommatori 100 ns**, e che gli altri ritardi di propagazione dei segnali siano trascurabili, indicare la durata totale del **ciclo di clock** tale che anche l'esecuzione della nuova istruzione sia permessa *spiegando i calcoli effettuati*.



MEMWRITE = DAS = 1

JUMP = BRANCH = REGWRITE = 0

Si vuole aggiungere l'istruzione di tipo R

beven rs, rt (branch if even)

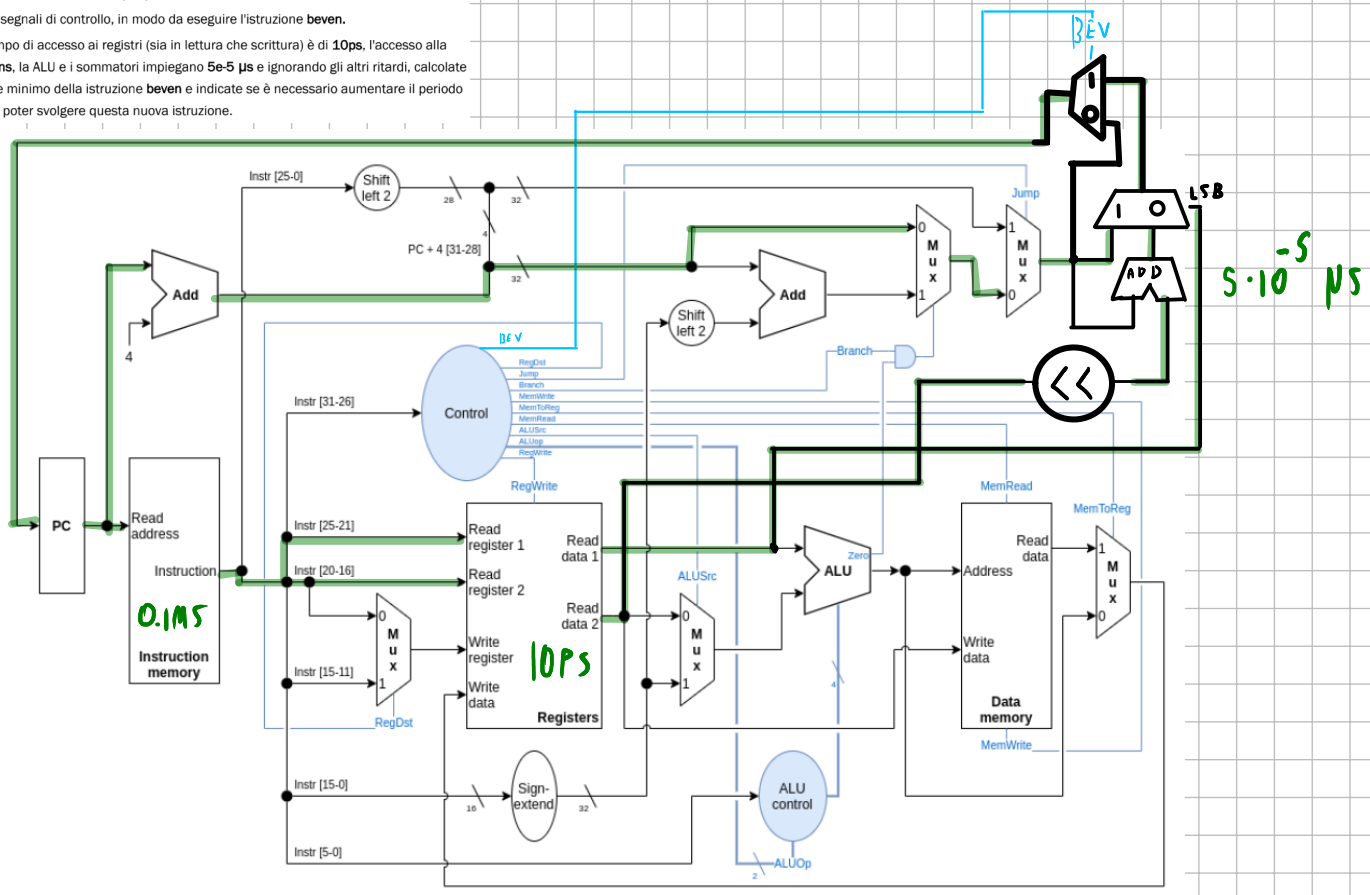
La nuova istruzione:

- esegua un salto relativo al PC (pre-incrementato di 4)
- l'offset del salto è contenuto nel registro rt, il quale contiene il numero di istruzioni da saltare.
- il branch è preso (is taken) solo se il valore del registro rs è pari.

a) si disegnino le modifiche necessarie a realizzare la funzione, aggiungendo tutti gli eventuali MUX, segnali di controllo, bus, ALU e sommatori (ecc) che ritenete necessari.

b) indicate i valori di tutti i segnali di controllo, in modo da eseguire l'istruzione **beven.**

c) tenendo conto che il tempo di accesso ai registri (sia in lettura che scrittura) è di **10ps**, l'accesso alla memoria impiega **0.1ns**, la ALU e i sommatori impiegano **5e-5 µs** e ignorando gli altri ritardi, calcolate il tempo di esecuzione minimo della istruzione **beven** e indicate se è necessario aumentare il periodo di clock della CPU per poter svolgere questa nuova istruzione.



MEMWRIT = REGWRITE = JUMP = BRANCH = 0

LSB È IL BIT PIU' A DESTRA DI QUEL DATO

BEV = 1

$$0,1\,MS + 10PS + 5 \cdot 10^{-5}\,\mu S$$

$$10 + 50 + 100 = 160PS$$