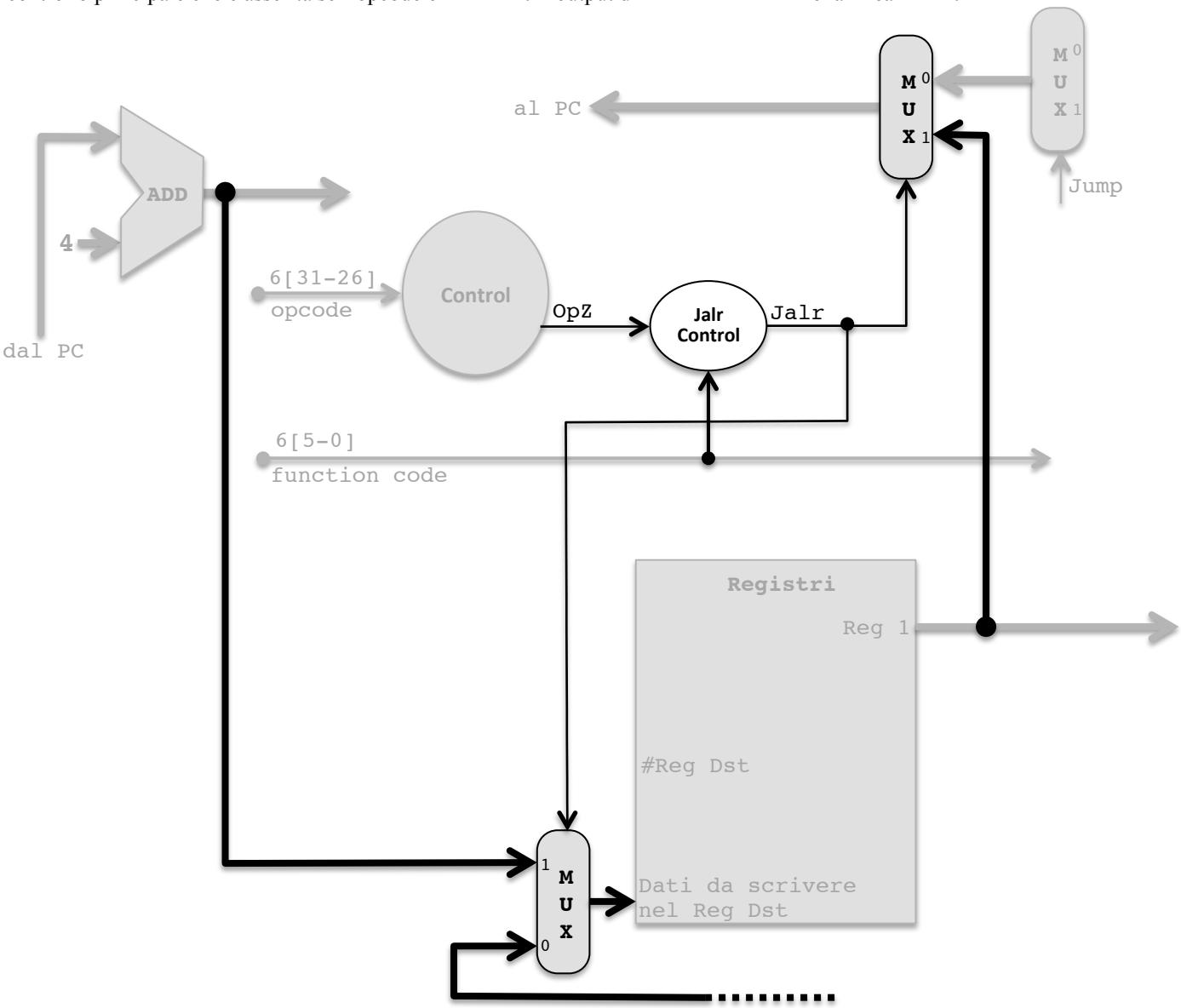
SOLUZIONE ESERCIZIO 1

Occorre aggiungere un multiplexer a monte o a valle del multiplexer del jump per effettuare il salto. Bisogna anche aggiungere un nuovo multiplexer in entrata a "`Dati da scrivere nel Reg Dst`" che prende come input 1 il valore di `PC+4`. Inoltre, a differenza dell'istruzione `j`, l'istruzione `jalr` è determinata anche dal function code. Quindi la linea di controllo per i nuovi multiplexer, che chiamiamo `Jalr`, è determinata dal opcode (`000000`) e dal function code (`001001`). Risulta più facile introdurre una unità di controllo dedicata, `Jalr Control`, che prende in input il function code e una linea `OpZ` dall'unità di controllo principale che è asserita se l'opcode è `000000`. L'output di `Jalr Control` è la linea `Jalr`.



	OpZ	**Jalr**	Jump	Branch	MemToReg	MemWrite	MemRead	ALUOp	ALUSrc	RegWrite	RegDst
jalr	1	1	0	0	X	0	0	XX	X	1	1

SOLUZIONE ESERCIZIO 2

1. Le istruzioni che non funzionano correttamente, se il processore è difettoso, sono quelle che richiedono che `RegWrite` non sia asserito: `sw`, `beq`, `j`.
2. Il programma è il seguente:

```
addi $s0, $zero, 0        #s0 <- 0
addi $s1, $zero, 1        #s1 <- 1
beq $s1, $s0, label       #non esegue il salto. Se è difettoso,
                          #scrive in s0 il risultato della
                          #sottrazione s1 – s0 = 1, altrimenti s0
                          #rimane con il valore 0
```

SOLUZIONE ESERCIZIO 3

Ad ogni iterazione del loop il valore di `$s0` è memorizzato nel elemento dell'array di indice `$s1` poi se `$s0` <= `$s1` esce dal loop, altrimenti il valore di `$s0` è aggiornato in modo tale che se `$s0` è pari viene diviso per 2 e altrimenti è moltiplicato per 3 e incrementato di 1, infine `$s1` è incrementato di 1. Al termine del loop, in `$s0` è caricato il valore dell'array di indice `$s0`. Allora, si può scrivere in pseudo-codice come segue:

```
          s0 = 3
          s1 = 0
loop:     array[s1] = s0
          if s0 <= s1 goto end
          if s0 è pari
              s0 = s0/2
          else
              s0 = 3*s0 + 1
          s1 += 1
          goto loop
end:      s0 = array[s0]
```

Adesso si può facilmente simulare e trovare che al termine del programma si ha `$s0` = 8 e `$s1` = 5.

SOLUZIONE ESERCIZIO 4

Affinché il loop possa scandire tutti gli elementi dell'array è necessario che in `$s1` ci sia l'indirizzo dell'elemento immediatamente successivo all'ultimo. Quindi, la prima istruzione mancante deve essere:

```
sll $a1, $a1, 2.
```

La seconda istruzione mancante deve necessariamente scrivere il valore di ouput di `func` nell'elemento corrente dell'array:

```
sw $v0, ($s0)
```

Infine, l'ultima istruzione mancante è semplicemente il ripristino di `$ra`:

```
lw $ra, -12($sp).
```