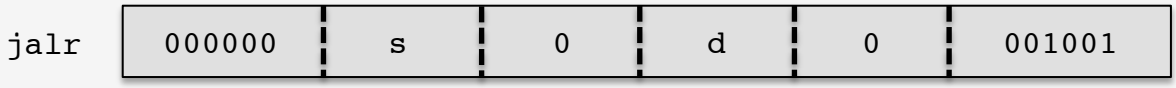
ESERCIZIO 1 Si consideri l'istruzione `jalr d, s` che ha l'effetto `d = PC + 4, PC = s`. È simile a `jal` con la differenza che l'indirizzo di ritorno è salvato in `d` e l'indirizzo di salto è preso dal registro `s`. Implementare l'istruzione `jalr`, eventualmente aggiungendo allo schema generale (riportato sul retro del foglio) linee di controllo, addizionatori, multiplexer, ecc. tenendo presente che è rappresentata nel formato R:



ESERCIZIO 2 C'è il sospetto che una partita di processori abbia il controllo difettoso: la linea `RegWrite` rimane sempre asserita, qualunque sia l'input.

1. Quali istruzioni tra quelle di base (`lw`, `sw`, `add`, `sub`, `and`, `or`, `xor`, `slt`, `beq`, `j`) non funzionano correttamente, se il processore è difettoso.
2. Scrivere un programma in assembly (senza pseudo-istruzioni) che quando termina, il valore del registro `$s0` è 1 se il processore è difettoso e 0 altrimenti. Si assume che `RegDst` sia asserito solo per le istruzioni di tipo R e che `MemtoReg` sia asserito solo per l'istruzione `lw`.

ESERCIZIO 3 Quali sono i valori dei registri `$s0` e `$s1` quando il seguente programma termina?

```
            li $s0, 3
            li $s1, 0
loop:       sll $t1, $s1, 2
            sw $s0, array($t1)    #array è l'indirizzo di un array di word
            ble $s0, $s1, end
            andi $t0, $s0, 1
            beqz $t0, ifeven
            mul $s0, $s0, 3
            addi $s0, $s0, 1
            j endif
ifeven:     srl $s0, $s0, 1
endif:      addi $s1, $s1, 1
            j loop
end:        sll $t1, $s0, 2
            lw $s0, array($t1)
```

ESERCIZIO 4 La seguente procedura `apply` prende in `$a0` l'indirizzo di un array di word e in `$a1` la lunghezza dell'array, riscrive ogni elemento dell'array sostituendolo con l'output della procedura `func` chiamata con input il valore dell'elemento. La procedura `func` prende l'input in `$a0` e ritorna l'ouput in `$v0`. Però, tre istruzioni nelle posizioni indicate dai punti interrogativi sono andate perse. Trovare le istruzioni mancanti.

```
apply:          sw $s0, -4($sp)
                sw $s1, -8($sp)
                sw $ra, -12($sp)
                addi $sp, $sp, -12
                move $s0, $a0
                ?
                add $s1, $s0, $a1
apply_loop:     beq $s0, $s1, apply_end
                lw $a0, ($s0)
                jal func
                ?
                addi $s0, $s0, 4
                j apply_loop
apply_end:      addi $sp, $sp, 12
                lw $s0, -4($sp)
                lw $s1, -8($sp)
                ?
                jr $ra
```

Implementazione ad un ciclo di clock di MIPS (solamente le istruzioni: `add`, `sub`, `and`, `or`, `xor`, `slt`, `lw`, `sw`, `beq`, `j`)