ESERCIZIO PREMIANTE
Architettura degli Elaboratori

Elisa Pioldi
mat. 856591

Appello di Giugno 2020

1 Esecuzione in QtSpim

Il codice proposto è il seguente:

```
main:
beq $0,$0, realMain
realMain:
jr $31
```

e viene eseguito in QtSpim senza particolari problemi. Le istruzioni vengono assemblate in questo modo:

```
[00400024] 10000001  beq $0, $0, 4 [realMain-0x00400024]
[00400028] 03e00008  jr $31                    ; 4: jr $31
```

Si nota pertanto che nel momento in cui la branch viene effettuata, l'offset risulta pari a `4`, dovendo saltare all'istruzione successiva etichettata da *RealMain*.

L'esecuzione delle istruzioni preliminari nell'`User Text Segment` di SPIM, non facenti parte nel nostro codice effettivo, assicura che il programma nel momento dell'istruzione `jr` salti al registro `$31`, corrispondente al *register address*, e con l'esecuzione dell'istruzione riferita dall'indirizzo contenuto in `$ra`, il *register address*, si conclude per un'opportuna syscall finale.

```
[00400018] 00000000  nop                       ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10            ; 191: li $v0 10
[00400020] 0000000c  syscall                   ; 192: syscall # syscall 10 (exit)
```

Durante l'esecuzione il ramo branch viene effettuato e il programma si conclude in QtSpim.

2 Esecuzione in Logisim

Dopo aver salvato le istruzioni assemblate in un file di testo con estensione `.img`:

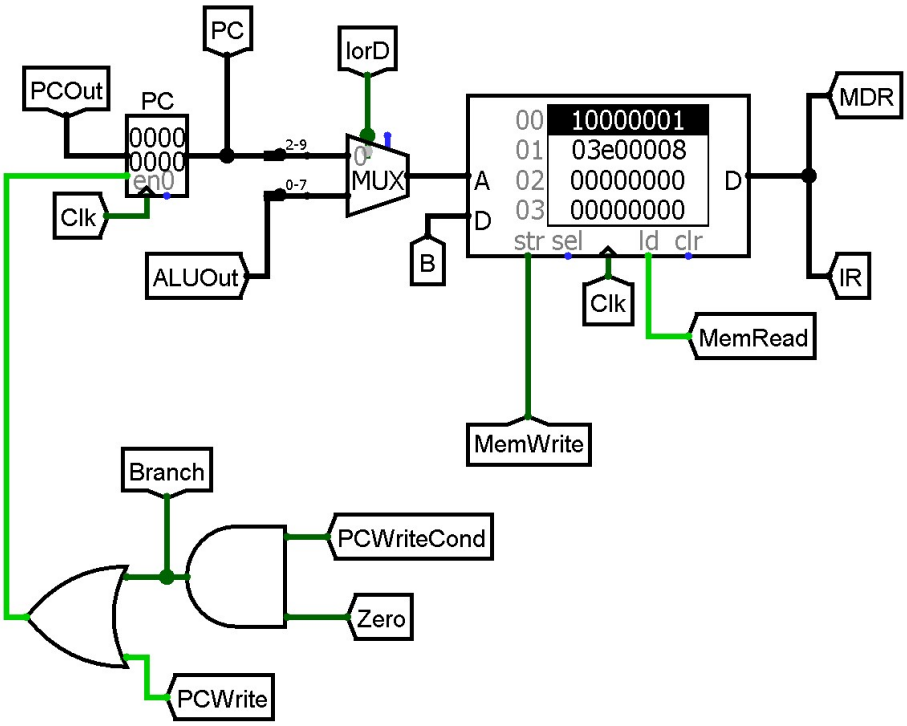
```
v2.0 raw
10000001
03e00008
```

Le carichiamo nella memoria del circuito simulazione del Datapath Muticiclo (figura 1).

Procedendo quindi con l'esecuzione delle istruzioni notiamo subito che il comportamento si rivela inatteso. Per prima cosa mancano le istruzione dell'`User Text Segment`, quindi il programma continua a eseguire le istruzioni anche nulle indefinitamente. Inoltre il *register address* non contiene il valore di default di SPIM che fa eseguire la conclusione del programma, ma il contenuto è nullo.

Ci si aspetterebbe quindi che dopo l'istruzione jr venga eseguita l'istruzione contenuta all'indirizzo segnato dal *register address*, l'istruzione di indirizzo nullo, ovvero la *branch*. Ciò che si verificherebbe sarebbe un ciclo infinito.

Figura 1: Istruzioni caricate in Logisim



Avanzando con i cicli di clock, osserviamo dunque che il *program counter* viene incrementato, ma controllando l'*instruction register* l'istruzione jr codificata come `03e00008` non viene mai eseguita, passando dall'istruzione di *branch* a quelle nulle successive a `jr`, e così via (figura 2). In un primo momento sembra pertanto che durante la *branch* occorra un problema.

Provando quindi a caricare in Logisim alcune istruzioni assemblate simili, ma tali che con la *branch* non venga effettuato l'offset

```
main:
li $t0, 2
beq $t0, $0, realMain
realMain:
jr $31
```

si vede che il programma non presenta problemi, e l'istruzione `jr` viene effettuata.

Se ne deduce quindi che il problema risiede nella consecutività dell'istruzione `jr` quando viene valutato il ramo della *branch*. Osservando attentamente il *program counter* si nota che, come dovrebbe essere, ad ogni *fetch* viene incrementato di `4`. La discordanza va ricercata nell'istruzione assemblata da SPIM di *branch* presenta un offset di `4` che va a sommarsi al già incrementato *program counter*, causando quindi un salto di un'istruzione, in questi caso la `jr`. Se il ramo della *branch* non venisse valutato, allora il *program counter* verrebbe incrementato solo di `4` e inoltre l'istruzione `jr` verrebbe eseguita.

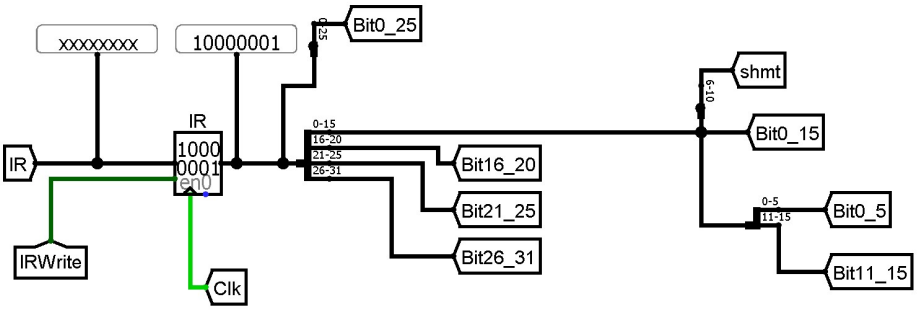
3 Correzione delle istruzioni e confronto con SPIM

Per correggere le istruzioni assemblate, dal momento che i problemi sono presenti nel ramo della *branch*, è necessario quindi operare sull'offset relativo a questo: tenendo presente che il *program counter* verrà incrementato di `4`, per passare all'istruzione successiva l'offset dovrà essere di `0`.

I byte modificati risulteranno quindi come i seguenti:

```
10000000
03e00008
```

Figura 2: Instruction Register



Inserendoli come immagine nella memoria del circuito, si vedrà che non ci saranno problemi, l'istruzione `jr` verrà eseguita e avrà luogo un ciclo infinito.

Resta un'ultima osservazione da fare al riguardo: il simulatore QtSpim nell'assemblare le istruzioni sembra non tenere conto dell'incremento del *program counter*, in quanto l'istruzione viene assemblata secondo un offset di `4` e non di `0`.