Esercizio I-DLX

Calcolatori Elettronici L-A

Esercizio 1

Si supponga di voler estendere il set di istruzioni del DLX con l'istruzione

BEQ Ri, Rj, (Rz)

L'istruzione confronta i due registri Ri e Rj e, in caso di uguaglianza, inserisce nel PC il contenuto della locazione di memoria puntata da Rz. Nel caso risulti Ri!=Rj il flusso del programma deve proseguire con l'istruzione successiva alla BEQ.

Nell'ipotesi che non siano necessari *wait-state* nel colloquio con la memoria, individuare:

- il diagramma degli stati ottimizzato che controlla l'esecuzione della nuova istruzione con riferimento al datapth del DLX *sequenziale* (cioè *non pipelined*) visto a lezione

- la sequenza di istruzioni assembler DLX equivalente alla BEQ

- nell'ipotesi che il salto sia effettuato: confrontare il numero di cicli di clock necessari per eseguire l'istruzione BEQ e il codice DLX equivalente

Esercizio 1

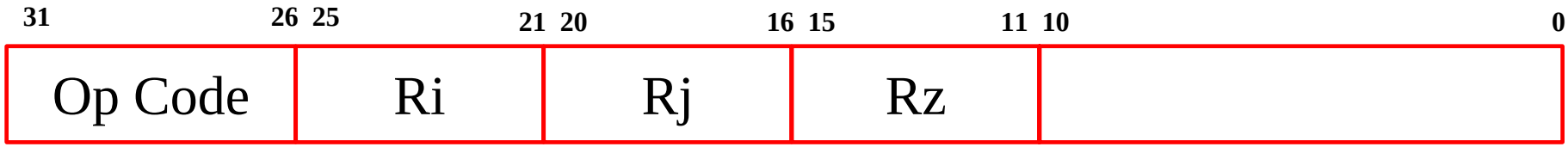
BEQ Ri, Rj, (Rz)

Nel caso in cui Ri e Rj siano uguali, l'istruzione **BEQ Ri, Rj, (Rz)**, deve effettuare un salto all'indirizzo di memoria specificato dal contenuto della locazione di memoria puntato da Rz.

Il formato delle istruzioni DLX è composto da 32 bit.

Il codice operativo è codificato con 6 bit e i registri coinvolti nell'operazione (Ri,Rj,Rz) sono codificati con 5 bit (32 possibili registri).

Il formato dell'istruzione è di **tipo R**.

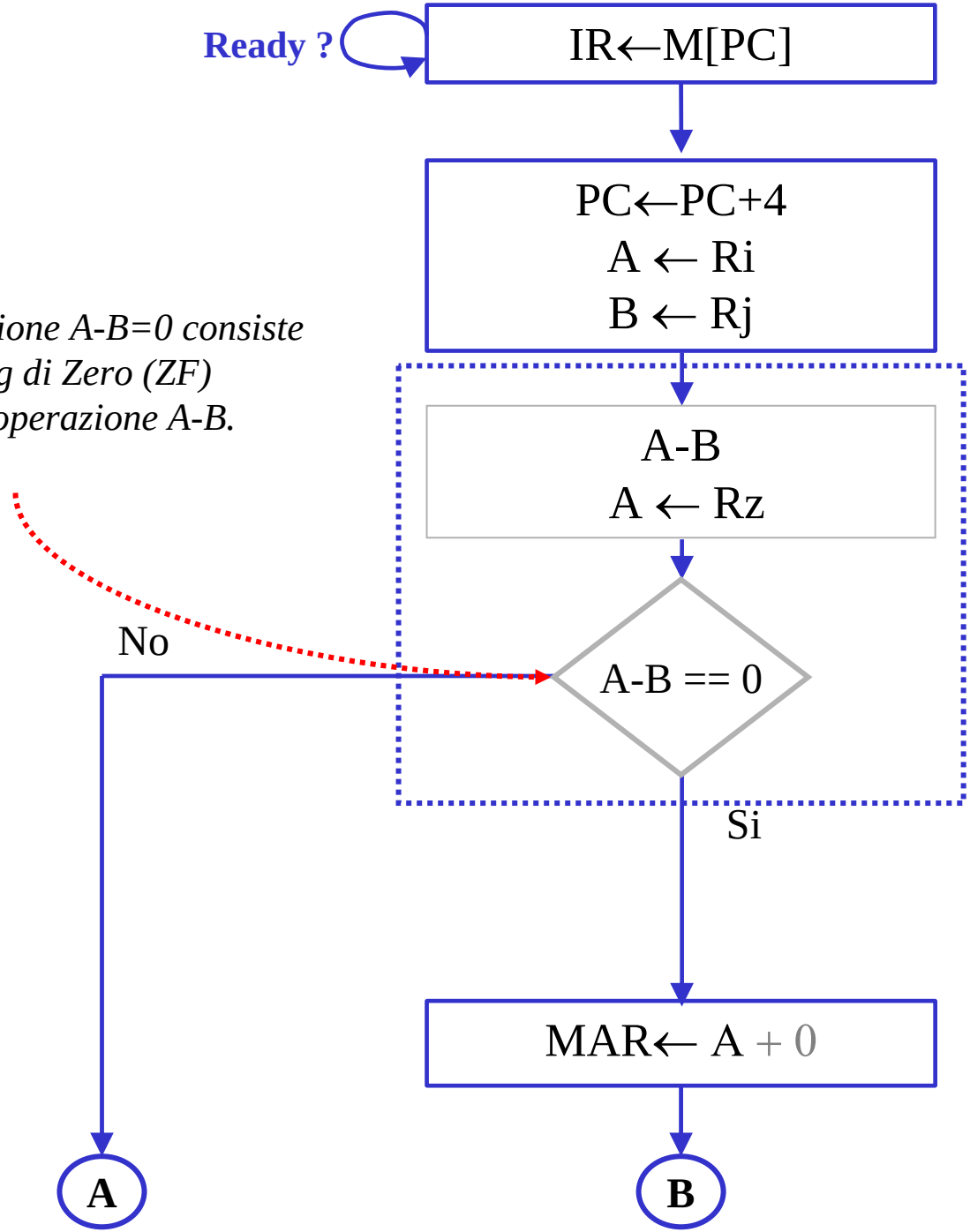


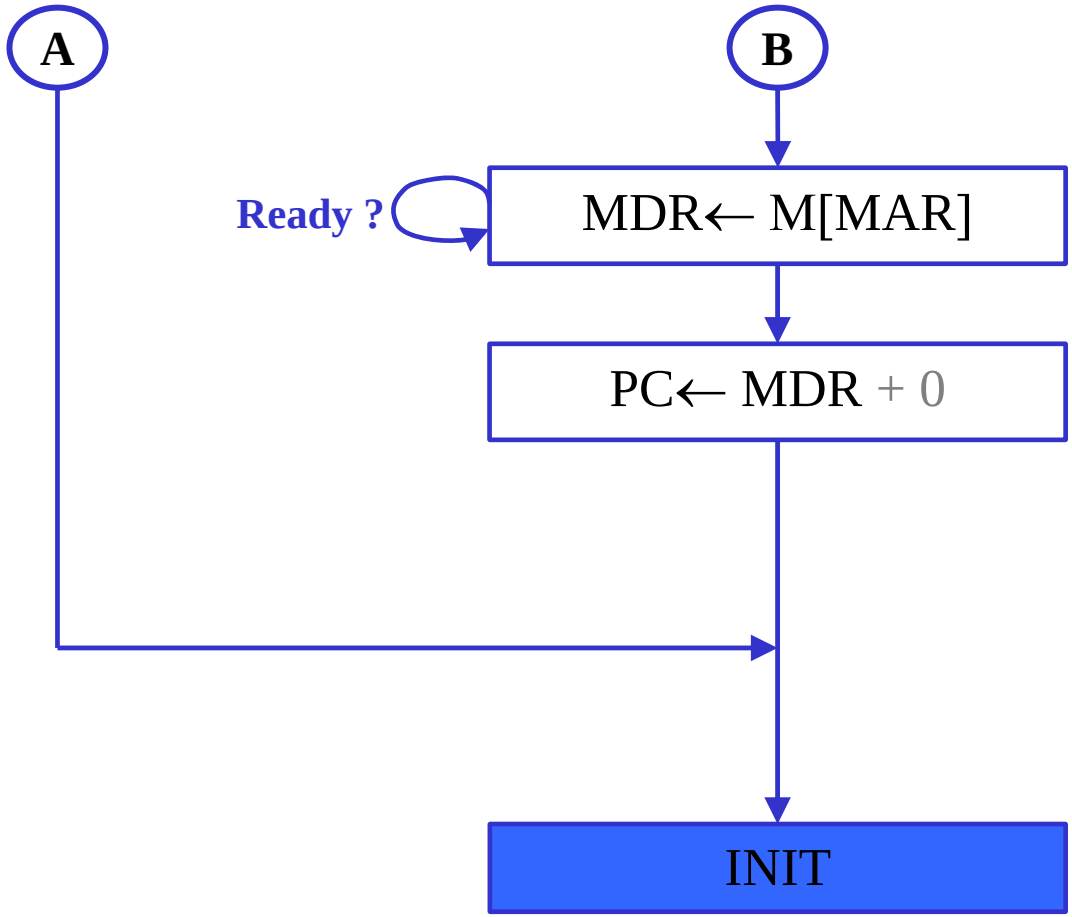
Il diagramma degli stati di questa istruzione è riportato nel lucido successivo.

Diagramma degli stati dell'istruzione

BEQ Ri, Rj, (Rz)

Il test della condizione A-B=0 consiste nell'analisi del flag di Zero (ZF) della ALU dopo l'operazione A-B.




A
B
Ready ?
MDR← M[MAR]
PC← MDR + 0
INIT

La sequenza di istruzioni DLX equivalente alla istruzione **BEQ Ri, Rj, (Rz)** è la seguente.

```
                SEQ  Rk, Ri, Rj        ; pone Rk=1 se Ri==Rj
                                       ; (6 clocks)

                BEQZ Rk, NOT_EQUAL     ; se Rk==0 salta a NEQ
                                       ;(4 clocks)


                LW   Ri, 0(Rz)         ; Ri←M[Rz]
                                       ; (6 clocks)

                JR   Ri                ; salta all'ind. contenuto in Ri
                                       ; (3 clocks)

NOT_EQUAL:      ................
```

Nel caso in cui Ri == Rj il numero di clock(*) necessari per eseguire l'istruzione equivalente alla BEQ Ri,Rj,(Rz) è pari a 19 clock (6+4+6+3).

Nel caso dell'istruzione BEQ Ri,Rj,(Rz) sono necessari solo 6 cicli di clock.

(*) *Il numero di clock delle istruzioni DLX è stato calcolato con riferimento ai diagramma degli stati presenti nei lucidi ipotizzando zero wait state.*

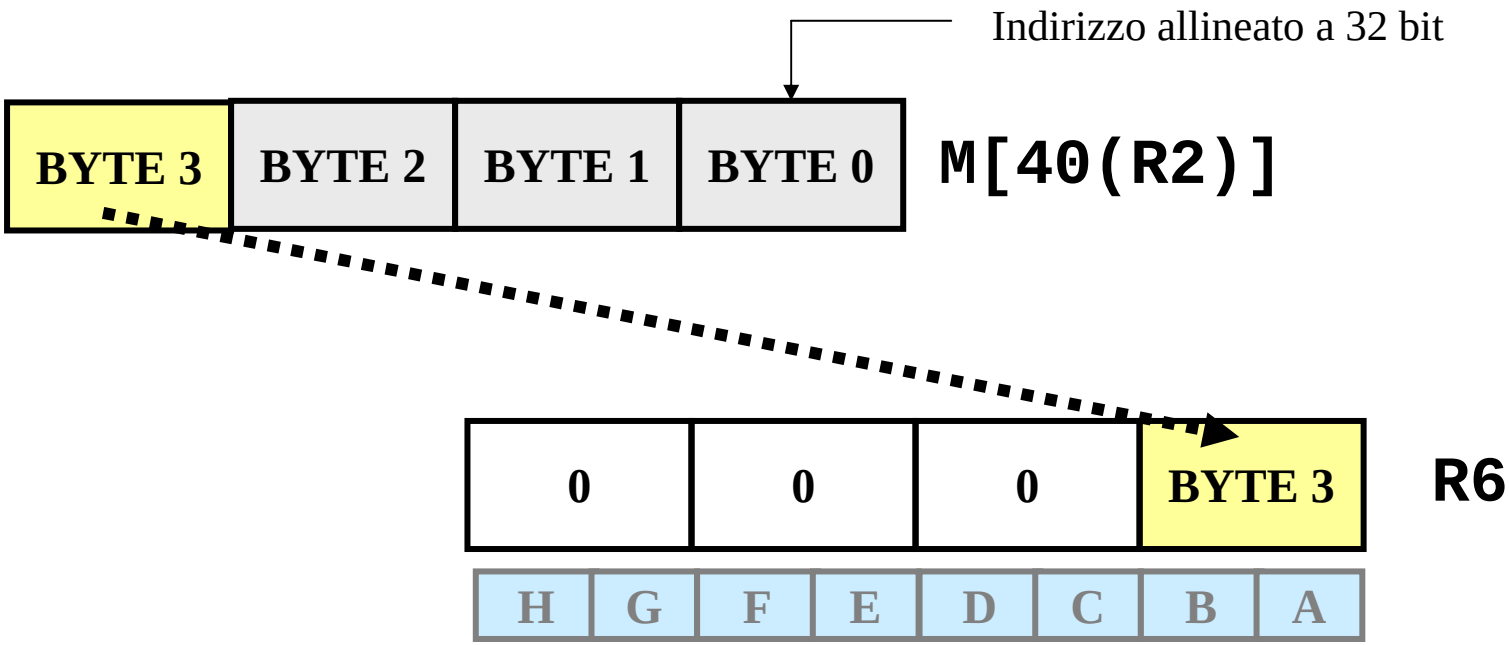
Esercizio 2

Si desiderano implementare via software istruzioni equivalenti alla *store byte* (SB) e *load byte unsigned* (LBU) del byte 3 di una word utilizzando solo istruzioni LW e SW.

Si supponga di poter utilizzare le istruzioni ***SLLI Ri, Rj, n*** *e* ***SLRI Ri, Rj, n*** *che permettono di traslare logicamente risp. verso sinistra e verso destra di* ***n*** *posizioni il contenuto del registro Rj.*

Il testo del problema richiede di leggere e scrivere il **BYTE 3** (di tipo *unsigned*) di una word utilizzando solo istruzioni di LW e SW (senza cioè utilizzare istruzioni SB e LBU).

a) Si supponga di voler prima leggere in R6 il BYTE 3 della word memorizzato all'indirizzo 40(R2) utilizzando solo istruzioni di tipo LW

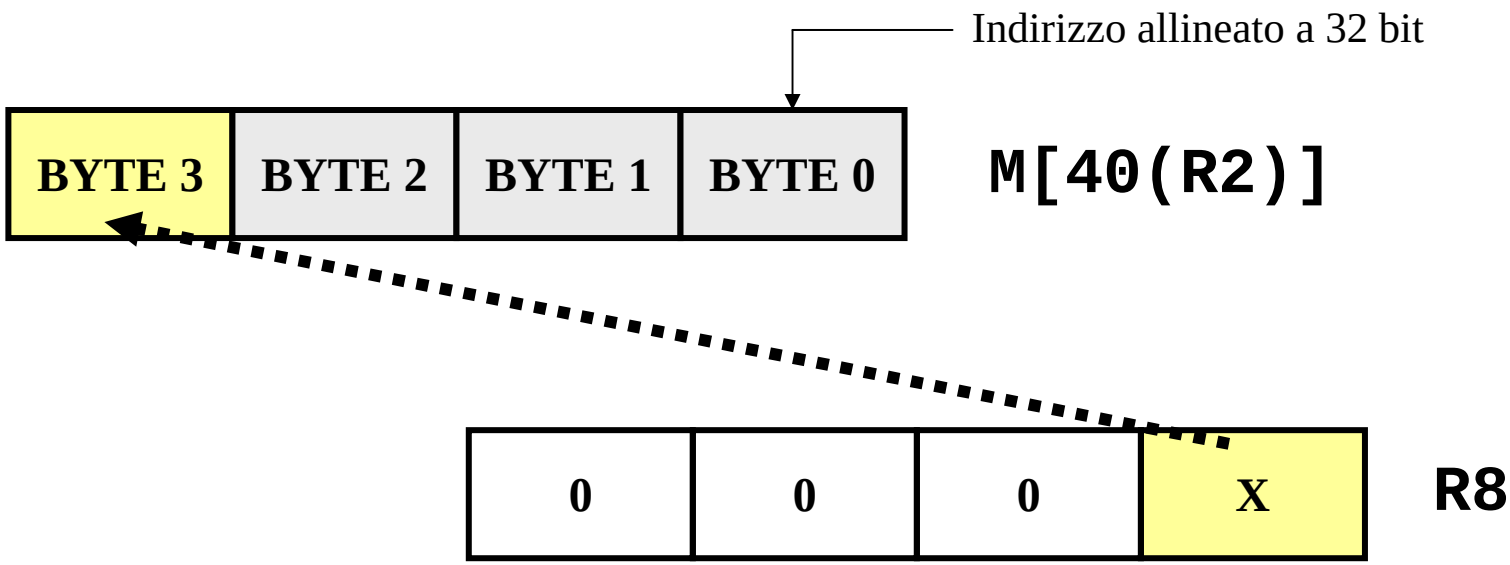



```
ORI  R3, R0, 00FFh      ; R3 ← 000000FF
SLLI R3, R3, 18h        ; R3 ← FF000000
LW   R6, 40(R2)         ; R6 ← M[40(R2)] = HGFEDCBA
AND  R6, R6, R3         ; R6 ← HGFEDCBA AND FF000000=HG000000
SLRI R6, R6, 18h        ; R6 ← 000000HG
```

Una versione piu' efficiente del codice precedente…

```
LW   R6, 40(R2)        ; R6 ← M[40(R2)] = HGFEDCBAh
SLRI R6, R6, 18h       ; R6 ← 000000HG
```

b) Si supponga di voler modificare solo il BYTE 3 della word memorizzata all'indirizzo 40(R2) con il primo byte (BYTE 0) contenuto nel registro R8 = HGFEDCBAh utilizzando solo istruzioni di tipo SW



Si supponga di voler modificare solo il byte 3 della word memorizzata all'indirizzo 40(R2) con il primo byte (byte 0) contenuto nel registro R8 = HGFEDCBAh

```
ORI  R3, R0, 00FFh     ; R3 ← 000000FF
AND  R8, R8, R3        ; R8 ← 000000BA
SLLI R8, R8, 18h       ; R8 ← BA000000

LW   R6, 40(R2)        ; R6 ← M[40(R2)] = ZVUTSRQP

SLLI R4, R3, 18h       ; R4 ← FF000000
ADDI R5, R0, -1h       ; R5 ← FFFFFFFF
XOR  R5, R5, R4        ; R5 ← 00FFFFFF
AND  R6, R6, R5        ; R6 ← 00UTSRQP
OR   R6, R6, R8        ; R6 ← BAUTSRQP
SW   40(R2), R6        ; M[40(R2)] ← BAUTSRQP
```

Una versione piu’ efficiente del codice precedente…

```
SLLI R8, R8, 18h        ; R8 ← BA000000

LW   R6, 40(R2)         ; R6 ← M[40(R2)] = ZVUTSRQP

SLLI R6, R6, 8h         ; R6 ← UTSRQP00
SLRI R6, R6, 8h         ; R6 ← 00UTSRQP
OR   R6, R6, R8         ; R6 ← BAUTSRQP
SW   40(R2), R6         ; M[40(R2)] ← BAUTSRQP
```