Calcolatori Elettronici L-A

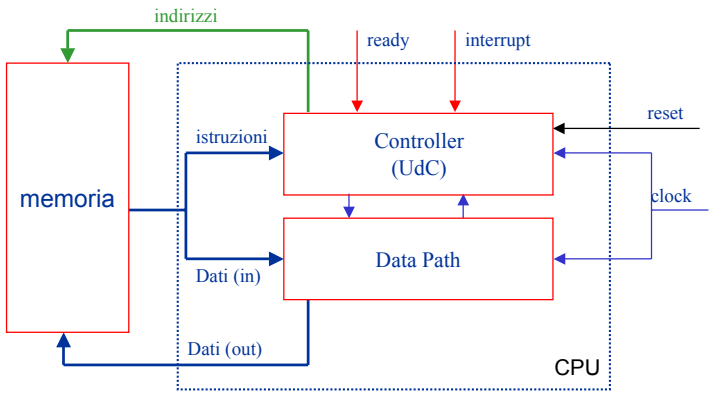
ISA DLX: Implementazione Tramite Struttura Sequenziale

Richiamo: caratterirtiche della CPU “DLX”

- “DLX” è un nome di fantasia (sta per De LuXe). Questa CPU è stata progettata a scopo didattico da John Hennessy e da David Patterson (ne è stata realizzata una molto simile a questa a Stanford Univ., il “MIPS”)
- Contiene molte delle soluzioni progettuali adottate nelle CPU commerciali di oggi ma (essendo teorica) non obbliga all’approfondimento di complicati dettagli tecnici, inutili dal punto di vista didattico
- Load-Store cioè carica gli operandi dalla memoria ai registri ➔ esegue le operazioni ➔ salva nuovamente in memoria il risultato contenuto nel registro. E’ una macchina R-R le cui istruzioni assembler hanno formato (3-operandi : 0-indirizzi di memoria)
- Progettata con filosofia RISC: poche e semplici istruzioni nel LM per favorire l’implementazione della pipeline
- Spazio di I/O memory-mapped
- Assenza di istruzioni assembler per la gestione dello Stack
- contiene registri da 32bit di uso generale (intercambiabili nella sintassi del LM = ortogonali)
- Sia il Bus Dati che quello Indirizzi hanno 32 linee (4GB indirizzabili)
- 1 word = 32 bit

Datapath e Unità di Controllo (1)

- La struttura di una CPU, come tutte le reti logiche sincrone che elaborano dati, può essere strutturata in due blocchi Controller (Unità di Controllo) e Datapath.
- La CPU, per funzionare, ha bisogno della memoria esterna su cui risiedono il programma e i dati.



DLX "sequenziale" 3

•Come tutte le RSS la CPU può essere pensata come la composizione di un “Datapath” e di un “Controller”

•Il datapath (percorso dei dati) può essere pensato come una Rete Combinatoria corredata di registri (memoria) ed è fondamentalmente il blocco dedicato all’elaborazione e ai calcoli (ogni istr. è eseguita mediante una successione di operazioni elementari).

•Il controller è una rete sequenziale sincrona progettata per comandare e dare una successione logica ai blocchi del datapath. Ad ogni ciclo di clock il controller comanda segnali che decidono quali devono essere le attività svolte dal datapath

•Il controller riceve le istruzioni contenute in memoria come risultato della fase di fetch. Dopo aver decodificato l’istruzione corrente il controller comanda al datapath lo svolgimento di tutte quelle micro attività che porteranno al completamento dell’esecuzione dell’istruzione. Il datapath invece preleva i dati dalla memoria alla CPU, elabora gli operandi in ingresso e genera risultati in uscita

•Un segnale di Reset fa si che subito dopo l’accensione la CPU si trovi in uno stato ben preciso che corrisponde all’esecuzione di una istruzione (BIOS)

•Il segnale di Ready è fatto per sincronizzare la CPU con la memoria e le altre periferiche; l’effetto del Ready sulla CPU è la modulazione della durata dei cicli di bus di lettura e scrittura esterne (mem o i/o) in funzione dei tempi di risposta delle periferiche

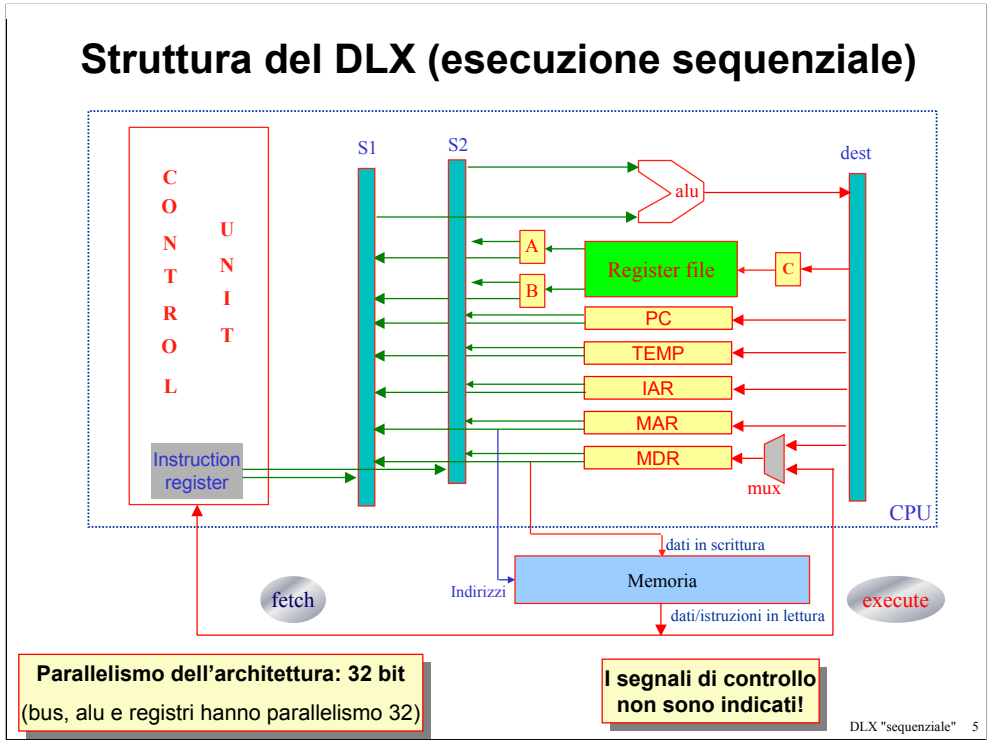
•il segnale di interrupt è un ingresso della CPU sensibile ad eventi esterni asincroni. Se al termine dell’esecuzione dell’istruzione corrente la CPU trova interrupt attivo salva il PC e trasferisce il controllo all’inizio del codice (procedura) che è stato scritto per “servire” l’interrupt.

Datapath e Unità di Controllo (2)

• DATAPATH: contiene tutte le unità di elaborazione ed i registri necessari per l'esecuzione delle istruzioni della CPU. Ogni istruzione appartenente all'Instruction Set è eseguita mediante una successione di *operazioni elementari,* dette *micro-operazioni*.

• *Micro-operazione*: operazione eseguita all'interno del DATAPATH *in un ciclo di clock* (esempi: trasferimento di un dato da un registro ad un altro registro, elaborazione ALU)

• CONTROLLER: è una RSS che in ogni ciclo di clock invia un ben preciso insieme di segnali di controllo al DATAPATH al fine di specificare l'esecuzione di una determinata *micro-operazione*.



•Durante la fase di Fetch arrivano le istruzioni dalla memoria all'IR che è il registro deputato a contenere tutto il codice dell'istruzione che deve essere eseguita.

•Ora comincia l'esecuzione ad opera del datapath. Gli operandi giungono alla ALU attraverso due bus S1 ed S2 mentre il risultato in uscita alla ALU va su un terzo bus detto "dest". In quanto bus essi non hanno la capacità di memorizzare questi dati. L'unica via di comunicazione tra S1-S2 e Dest passa attraverso l'ALU che quindi è necessario attraversare anche nel caso di semplici istruzioni di copia e spostamento dati da un registro all'altro.

•Se l'istruzione comporta la lettura/scrittura di dati in ram (Load e Store) questi saranno dapprima scritti/letti sul Memory Data Register e gli indirizzi di accesso alle celle di memoria saranno contenuti nel Memory Address Register

•Se si tratta di una istruzione ALU gli operandi e i risultati stanno nel Register File (banco dei 32 registri GPR da 32bit). Sono registri Edge Triggered cioè campionano sul fronte (positivo) del ck, hanno un input WE e DUE input OE perchè le loro uscite possono andare su A o su B

•La ALU possiede tre buffer A, B (di lettura dai regs) e C (di scrittura sui regs). Ad ogni ciclo di clock (sull'edge) è possibile accedere in lettura ad una coppia di registri interni campionandone il contenuto su A, B rendendo così i dati disponibili in modo stabile agli ingressi dell'ALU. Allo stesso modo in un Tck è possibile scrivere su un registro il contenuto di C.

•Program Counter contiene sempre l'indirizzo in memoria della prossima istruzione da eseguire calcolato durante il fetch sommando 4 (byte perchè ogni istruzione ha lunghezza fissa 4 byte) all'indirizzo di provenienza dell'istruzione appena prelevata.

•Interrupt Address Register contiene l'indirizzo di ritorno cioè il contenuto del Pc salvato nel momento di trasferire il controllo all'indirizzo della procedura di servizio all'interrupt

•Temporary register è usato per contenere temporaneamente dati

•I Segnali di Controllo (omessi nel lucido) provengono tutti dalla Control Unit. Oltre ad indirizzare i registri del RF comandano le abilitazioni di tutti registri che insistono sui tre bus interni, le opzioni di calcolo della ALU, i WE dei registri dedicati, la selezione del mux ed i comandi RD e WR della ram.

Struttura del DLX (esecuzione sequenziale)

- Instruction Register il registro deputato a contenere l'opcode dell'istruzione che deve essere eseguita.
- Bus S1 ed S2 portano gli operandi alla ALU. Sono multiplexer distribuiti con parallelismo 32bit
- Bus dest trasporta il risultato in uscita dalla ALU. In quanto bus essi non hanno la capacità di memorizzare questi dati. L'unica via di comunicazione tra S1-S2 e Dest passa attraverso l'ALU.
- Memory Data Register memorizza i dati per istruzioni Load/Store in ram prima che arrivino in ram (store) o nei registri (load)
- Memory Address Register contiene l'indirizzo di accesso alle celle di memoria oggetto delle istruzioni del caso precedente
- Register File banco dei 32 registri GPR da 32bit. Campionano sul fronte positivo del ck (ET), hanno un input WE# e due input OE# perché le loro due uscite possono andare su A che su B...
- A, B e C sono tre buffer del banco di registri RF: A, B (di lettura dai regs) e C (di scrittura sui regs). Ad ogni ciclo di clock (sull'edge) è possibile accedere in lettura ad una sola coppia di registri interni campionandone il contenuto su A, B. Allo stesso modo in un Tck è possibile scrivere su un solo registro il contenuto di C.
- Program Counter contiene sempre l'indirizzo in memoria della prossima istruzione da eseguire calcolato durante il fetch sommando 4 all'indirizzo di provenienza dell'istruzione appena prelevata.
- Interrupt Address Register contiene "l'indirizzo di ritorno" cioè il contenuto del Pc salvato al momento di trasferire il controllo all'indirizzo della procedura di servizio all'interrupt
- Temporary register registro temporaneo

Tempo necessario ad un trasferimento dati sul Datapath

- Al fine di valutare la massima frequenza a cui è possibile far funzionare il datapath è importante conoscere le seguenti temporizzazioni:
  - $T_C\,(max)$: ritardo max tra il fronte positivo del clock e l'istante in cui i segnali di controllo generati dall'unità di controllo sono validi;
  - $T_{OE}\,(max)$: ritardo max tra l'arrivo del segnale OE e l'istante in cui i dati del registro sono disponibili sul bus;
  - $T_{ALU}\,(max)$: ritardo massimo introdotto dalla ALU;
  - $T_{SU}\,(min)$: tempo di *set-up* minimo dei registri (requisito minimo per il corretto campionamento da parte dei registri).
- La massima frequenza di funzionamento del datapath si calcola come segue:

$$T_{CK} > T_C\,(max) + T_{OE}\,(max) + T_{ALU}\,(max) + T_{SU}\,(min)$$

$$f_{CK}(max) = 1/T_{CK}$$

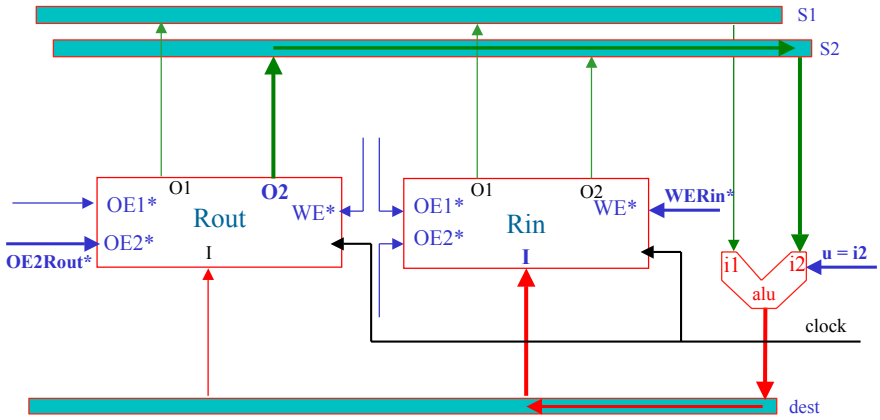
DLX "sequenziale" 7

La microistruzione fondamentale eseguita dal datapath consiste nella lettura di due registri tra quelli accessibili attraverso S1 o S2 (IR, A, B, PC, TEMP, IAR, MAR, MDR) nella elaborazione mediante la ALU (al limite, la mera copia) e nella scrittura del risultato in un terzo registro tra quelli accessibili attraverso dest (C, PC, TEMP, IAR, MAR, MDR).

Per microistruzione fondamentale si intende ad esempio TEMP ← A+ PC oppure PC ← B - MAR, ecc.

Esempio: esecuzione della microistruzione Rin ← Rout

I segnali in blu (*segnali di controllo*) provengono dall'Unità di Controllo

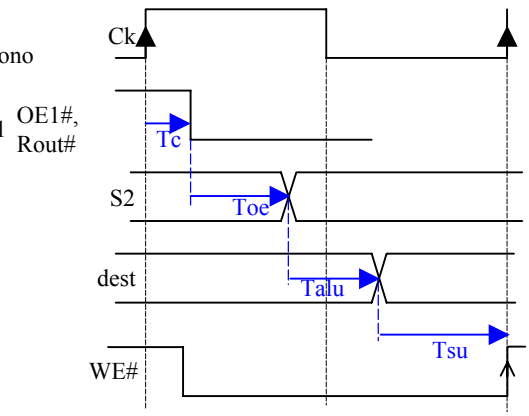


I segnali di controllo in grassetto sono attivi nel ciclo di clock in cui il micro-step Rin ← Rout viene eseguito

Esercizio

• **Si disegnino le forme d'onda dei segnali di controllo e dei dati per la microistruzione vista nel lucido precedente riferendole al clock e considerando i parametri temporali visti nel calcolo di f_{ck}(max).**

La generica micro-istruzione che il datapath visto è in grado di eseguire è "Rk ← Ri OP Rs" dove Rk è uno dei registri che possono essere scritti attraverso il bus "dest" (C, PC, TEMP, IAR, MAR, MDR) ed Ri, Rs sono due dei registr che insistono sui due bus S1 ed S2 (IR, A, B, PC, TEMP, IAR, MAR, MDR) ed OP è una generica operazione logico-aritmetica eseguita dala ALU. Nell'esempio a cui fa riferimento il testo di questo esercizio la micro-istruzione è Rin ← Rout dove non c'è operatore e tutto consiste in un semplice trasferimento dati tra due registri.



Rispetto a quanto detto finora esistono delle micro-istruzioni che non coinvolgono la ALU e non impegnano nemmeno i bus S1, S2 e dest e sono quelle che consistono nella lettura e scrittura del register file:

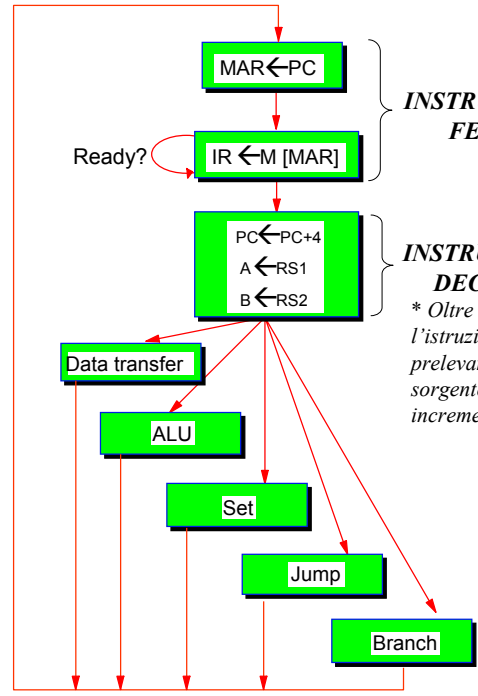
A ← Rs1 B ← Rs2 Rd ←C

Esse possono avvenire in parallelo tra loro e simultaneamente rispetto alla generica operazione del tipo Rk ← Ri OP Rs vista sopra

Il progetto dell'Unità di Controllo

• Una volta definito il Set di Istruzioni e progettato il DATAPATH, il passo successivo del progetto di una CPU è il progetto dell'Unità di Controllo (*CONTROLLER*).
•Il CONTROLLER è una RSS: il suo funzionamento può essere specificato tramite un *diagramma degli stati*.
•Il CONTROLLER (come tutte le RSS) permane in un determinato stato per un ciclo di clock e transita (*può transitare*) da uno stato all'altro in corrispondenza degli istanti di sincronismo (fronti del clock).
•Ad ogni stato corrisponde quindi un ciclo di clock. Le *micro-operazioni* che devono essere eseguite in quel ciclo di clock sono specificate (in linguaggio RTL) nel diagramma degli stati che descrive il funzionamento del CONTROLLER *all'interno degli stati*.
•A partire dalla descrizione data in RTL si sintetizzano poi i segnali di controllo che devono essere inviati al DATAPATH per eseguire le operazioni elementari associate ad ogni stato.

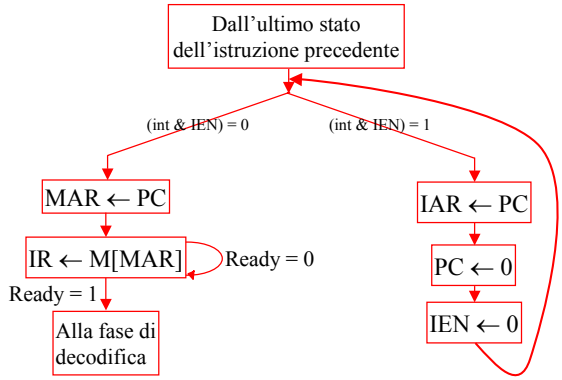
Il diagramma degli stati del controller (1)



Quanti clock sono necessari per eseguire la fase di fetch?

Fase di Fetch e gestione interrupt

- In questa fase si deve verificare se è presente un interrupt (evento esterno asincrono che la CPU deve "servire" con apposito software);
- se l'interrupt è presente e può essere servito (IEN = true, Interrupt Enable Flag) si esegue implicitamente l'istruzione di chiamata a procedura all'indirizzo 0, e si salva l'indirizzo di ritorno nell'apposito registro IAR (Interrupt Address Register);
- se l'interrupt non è presente e le interruzioni non sono abilitate, si va a leggere in memoria la prossima istruzione da eseguire (il cui indirizzo è in PC)



DLX "sequenziale" 12

Nel diagramma precedente non era stata considerata la gestione del segnale di interrupt. Ecco come si modifica la fase di fetch in presenza di una richiesta di interrupt.

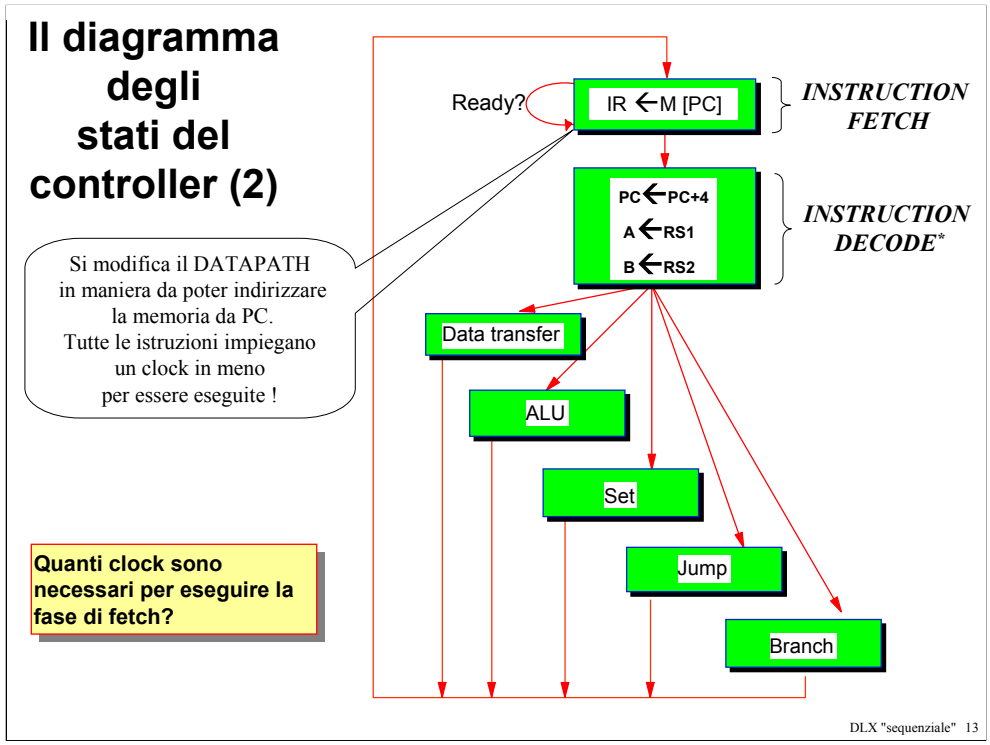
Affinchè un interrupt sia "servito" devono essere soddisfatte due condizioni:

1) l'ingresso Int della CPU deve essere a 1

2) l'Interrupt ENable flag è settato a 1.

In corrispondenza dell'ultimo stato di ciascuna istruzione si fa il check di entrambe le condizioni: se risultano entrambe vere il controllore non torna nello stato di fetch per l'istruzione successiva ma esegue altre micro-istruzioni, quelle mostrate a destra la cui spiegazione andiamo a leggere sul lucido.

Notiamo che un annidamento (nesting) delle interruzioni non sarebbe possibile perché c'è un unico registro per salvare l'indirizzo di ritorno.



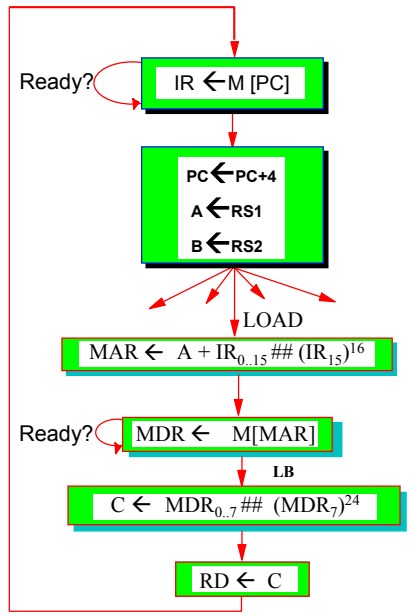
Nel diagramma visto poco fa la fase di fetch è eseguita in due tempi perché il MAR è l'unico a poter pilotare il bus indirizzi per accedere in memoria. Questo però causa uno spreco di tempo pari al clock necessario a copiare sul MAR l'indirizzo che avremmo già pronto nel PC. Inoltre questo stato è comune a TUTTE le istruzioni.

Potrebbe allora valere la pena procedere a modificare il datapath in modo che anche il PC possa accedere al bus indirizzi.

Questa modifica è molto semplice perché si tratta solo di mettere un MUX tra il MAR e la ram e collegare l'input libero del MUX all'uscita del PC. La selezione del canale da mandare agli indirizzi di memoria deve avvenire in funzione dello stato in esecuzione: se FETCH il PC va agli indirizzi di memoria, in tutti gli altri casi ci va il MAR. Questo segnale ci costa anche una piccola modifica al controllore ma abbiamo risparmiato un Tck per tutte le istruzioni!

Con questa nuova configurazione serve un clock per ogni stadio della fase di FETCH più, come prima, tutti i clock per i Twait dell'accesso in ram. Quindi servono in tutto 2 Tck per il FETCH.

Controllo per l'istruzione LB (Load Byte)



DLX "sequenziale" 14

Consideriamo l'istruzione LB

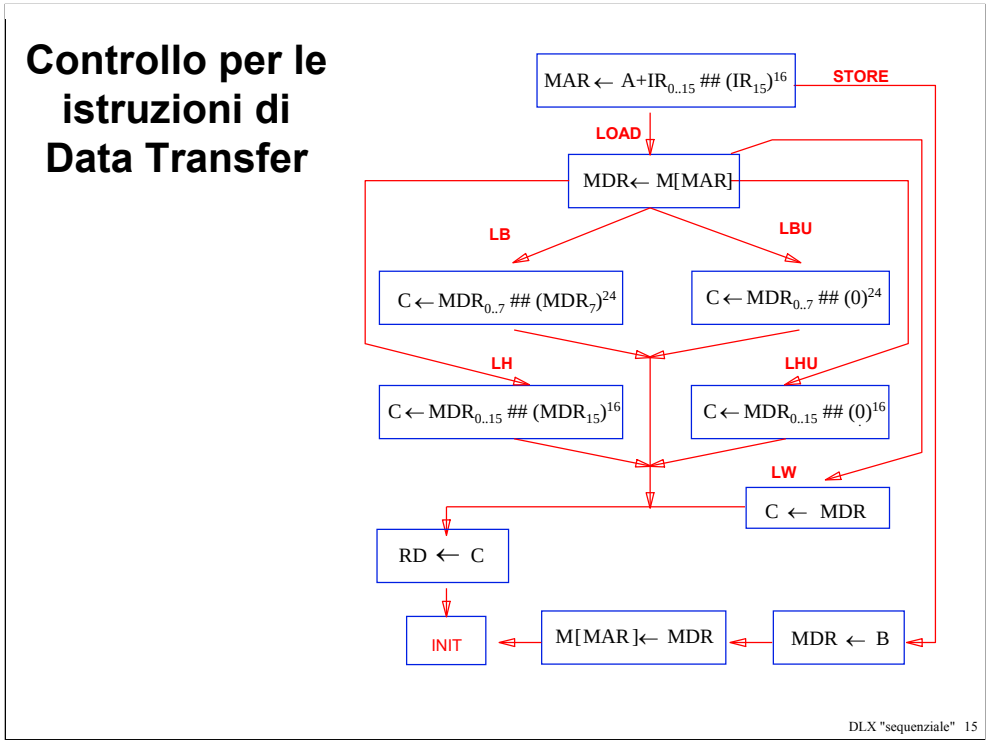
Il passo di decodifica è seguito dagli stati specifici dell'istruzione LOAD. Il primo consiste nel caricare nel MAR il calcolo dell'indirizzo ottenuto sommando A (contenente il campo Rs1 = registro sorgente) con l'estensione di segno dei 16 bit meno sign di IR (contenenti l'operatore immediato "spiazzamento")

Il secondo passo consiste nell'accesso in memoria all'indirizzo contenuto nel MAR e nella conseguente copia del dato su MDR.

La lunghezza del dato caricato può essere varia: byte, byte senza segno, mezza parola (16bit), mezza parola senza segno. Per ciascuno di questi casi è necessario allineare il dato in modo opportuno nel registro C prima che esso sia scritto nel RF. Nel lucido è mostrato l'esempio del caricamento di un byte che viene copiato nella parte "bassa" di C il quale viene poi riempito con l'estensione di segno.

Infine il contenuto di C è campionato nel RF all'indirizzo del registro destinazione (specificato nel campo Rd del formato di istruzione I).

Da ora in poi, per brevità, nei diagrammi del controllo le fasi di FETCH e DECODE verranno indicate con INIT in quanto questa fase è comune alla prima parte di tutte le istruzioni.

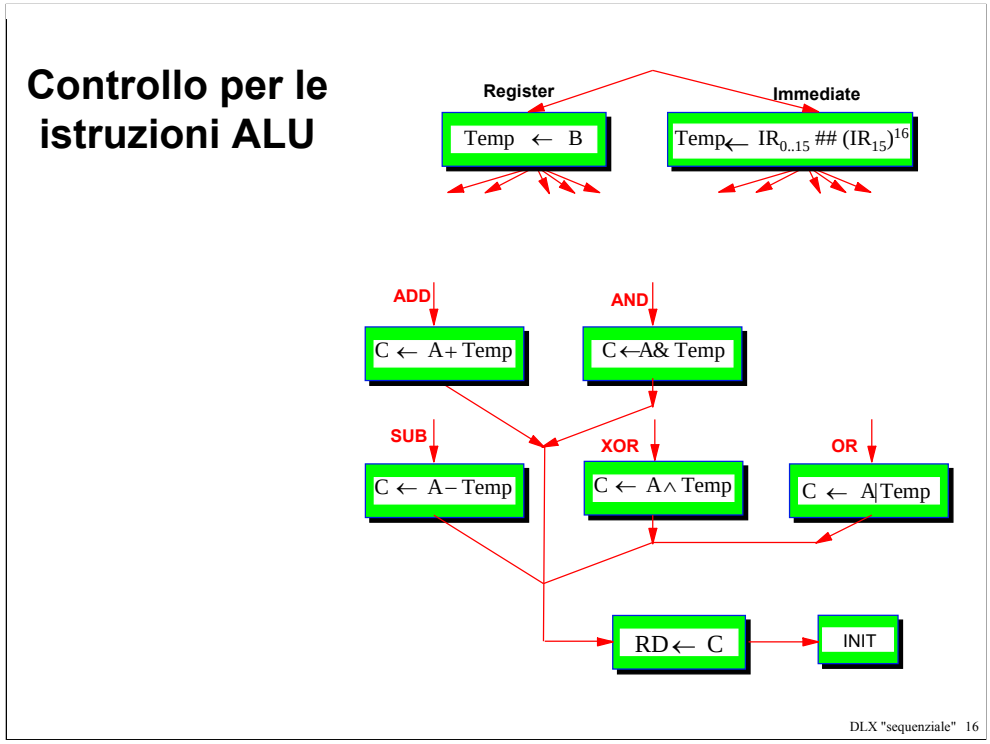


In questo lucido sono mostrati le successioni di stato per le istruzioni LOAD/STRORE differenziate per direzione dei dati (LOAD o STORE) e per formati di dato (LB, LBU, LH, LHU, LW).

Tutte le istr L/S hanno in comune la micro-istruzione dove si calcola l'indirizzo e lo si carica in MAR, come mostrato nel lucido precedente.

Tutte le LOAD accedono prima in memoria e portano in MDR il contenuto a 32 bit della locazione letta. Poi, a seconda dell'opcode, si copia l'MDR nel registro di C di "pre-scrittura": le istruzioni per dati unsigned non hanno la parte per l'estensione del segno e prevedono semplicemente il riempimento di C con degli zeri. Un esempio per dati con segno l'abbiamo visto nel lucido precedente. L'istruzione LW (32 bit) non ha stati per l'aggiustamento e copia MDR in C. Tutte le LOAD hanno la parte finale vista anche nel lucido prec di scrittura nel RF.

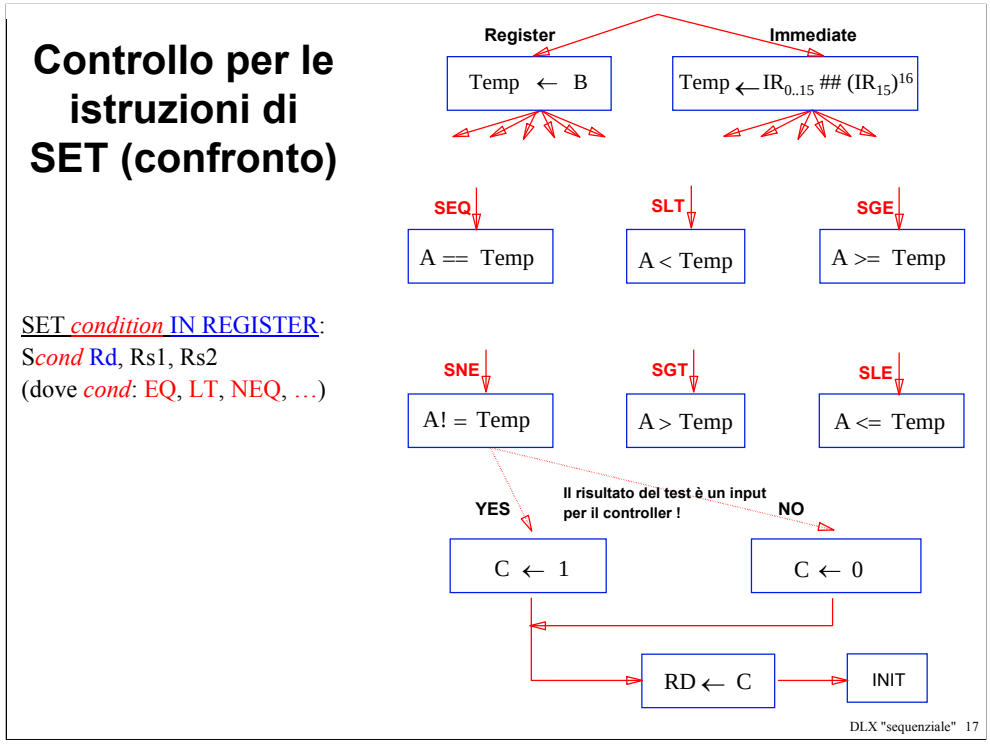
Per le STORE si ha che B è già stato caricato con il contenuto del registro sorgente del campo Rs durante il passo di decode. Quindi si procede copiando B in MDR e successivamente accedendo in memoria per scrivere MDR all'indirizzo di MAR.



Nell'esecuzione delle istruzioni ALU uno dei due operandi può essere un registro o un immediato. Per uniformare i due casi si fa in modo di salvarlo in TEMP prima di compiere l'esecuzione. Quindi, a seconda che nell'istruzione il secondo operando sia un registro o un immediato, bisogna prima copiare in TEMP l'operando registro specificato in Rs2 (che sta già in B) oppure l'operando "immediato" che sta in IR con l'opportuna estensione del segno.

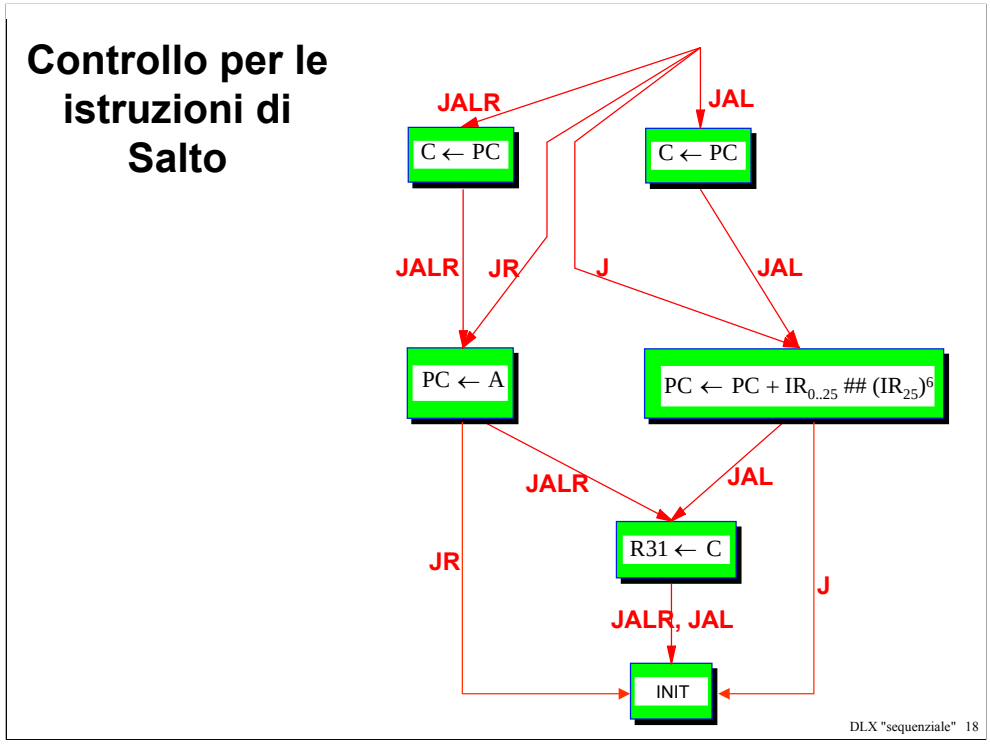
Fatto ciò, il passo successivo corrisponde alla vera e propria elaborazione prendendo come primo termine A e come secondo TEMP e scrivendo il risultato in C.

Alla fine C viene copiato nel RF sul registro specificato nel campo Rd dell'istruzione e si va al fetch successivo.



Il test si compie confrontando (come per le istruzioni ALU) A con un operatore che può essere un registro o un immediato. Quindi -anche qui- per uniformare tutti i casi, il secondo termine di paragone viene prima scritto in TEMP (come per le ALU).

A questo punto si può effettuare il confronto tra A e TEMP e scrivere 1 o 0 sul registro C e poi sul registro destinazione specificato in Rd dell'istruzione, a seconda che l'ipotesi sia o meno verificata.



JALR salta a registro e salva link per ritorno: prima salva il PC corrente in C e poi copia l'indirizzo del salto in PC. Questo è il contenuto del registro (che sta già in A dal passo di decode).

JR salta a registro: viene semplicemente copiato in PC l'indirizzo di salto che è il contenuto del registro (che sta già in A) come parte finale di quanto visto sopra

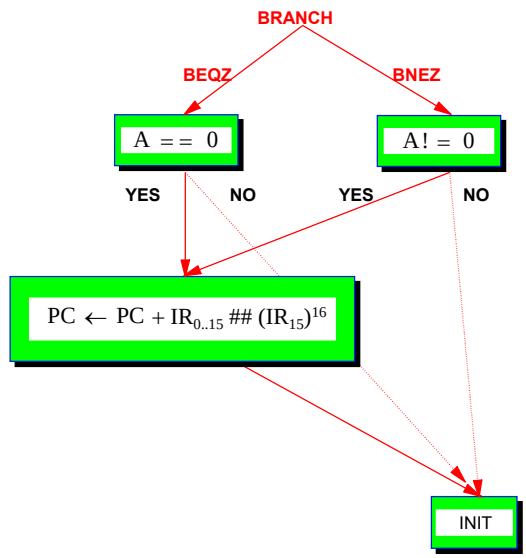
JAL salta a immediato e salva link per ritorno: prima salva il PC corrente in C e poi copia l'indirizzo del salto in PC. Questo risulta dalla somma del PC corrente con lo spiazzamento immediato da 26 bit (come da formato J) e contenuto in $IR_{0..25}$.

J salta a immediato: viene semplicemente copiato in PC l'indirizzo del salto ottenuto come nel caso precedente (PCcorrente+Spiazzamento_di_IR)

JALR e JAL sono gli unici due casi in cui il link per il ritorno va salvato in R31 prima di tornare al fetch successivo (init)

Solo J e JAL sono PC-relative, nelle altre i salti non dipendono dal contenuto del PC.

Controllo per le istruzioni di Branch



DLX "sequenziale" 19

Nelle istruzioni di BRANCH si controlla il contenuto di un registro (precedentemente aggiornato da una SET e contenuto in A) e si salta ad un offset specificato come immediato a 16 bit.

L'indirizzo risultante per il salto è relativo al PC, infatti in caso di branch-taken si ottiene sommando l'offset al PC. L'offset sta in IR ed è parte del codice dell'istruzione. Prima di essere sommato al contenuto di PC, l'offset viene esteso in segno.

Numero di clock necessari per eseguire le istruzioni

Istruzione	Cicli	Wait	Totale
Load	6	2	8
Store	5	2	7
ALU	5	1	6
Set	6	1	7
Jump	3	1	4
Jump and link	5	1	6
Branch (taken)	4	1	5
Branch (not taken)	3	1	4

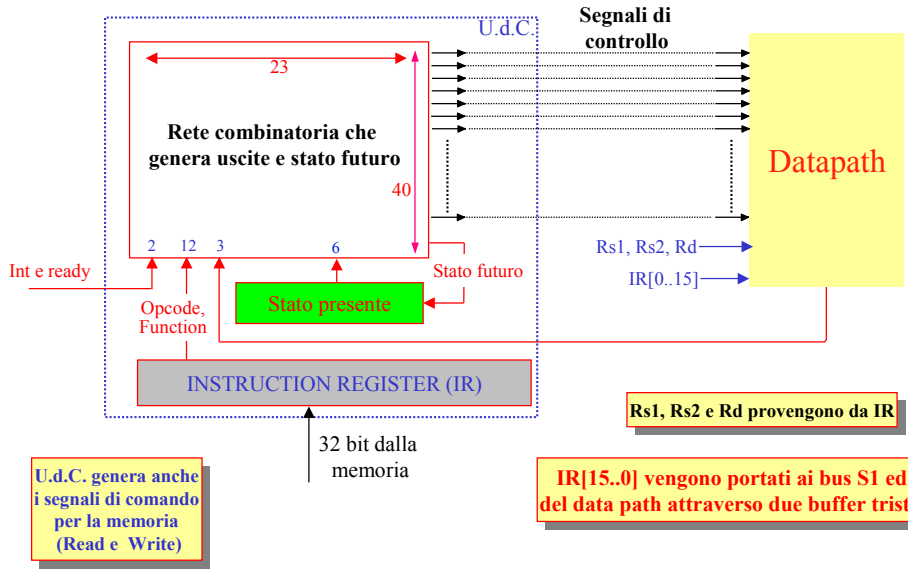
$$CPI = \sum_{i=1}^{n} \left(CPI_i * \frac{N_i}{\textit{numero totale di istruzioni}} \right)$$

Esempio: GCC su DLX
LOAD: 21%, STORE: 12%, ALU: 37%, SET: 6%, JUMP: 2%, BRANCH (taken): 12%, BRANCH (not-taken): 11%

→ CPI = 6.3

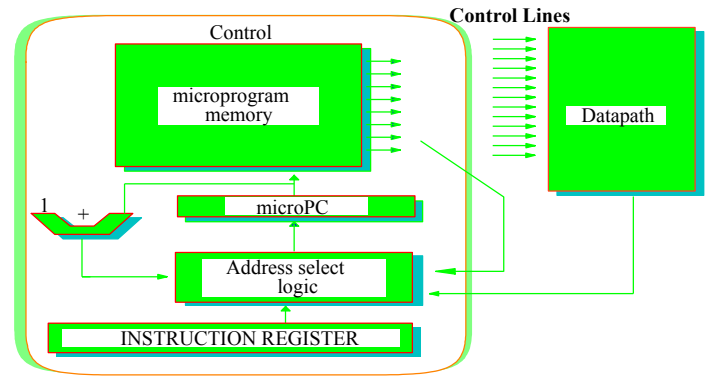
DLX "sequenziale" 20

Controllo cablato (“*hardwired*”)



DLX "sequenziale" 21

Unità di controllo con architettura microprogrammata



μ - INSTRUCTION

Dest	ALU	SR1	SR2	Const	Misc	Cond	Jump

DLX "sequenziale" 22

Esercizi

- Quali segnali di controllo devono essere attivi per eseguire le micro-operazioni di prelievo degli operandi sorgente dal Register File (A ← Rs1 , B ← Rs2) ?
- Si scrivano le micro-operazioni necessarie ad eseguire l'istruzione di ritorno dalla procedura di servizio dell'interrupt (detta anche Return from Exception o RFE)
- Si consideri la sequenza di micro-operazioni necessarie per mettere in esecuzione la procedura di servizio dell'interrupt. Perché è necessario eseguire la micro-operazione: IEN ← 0?; cosa succederebbe senza questa micro-operazione ?
- Qual è l'istruzione assembler del DLX che si deve eseguire per tornare da una procedura al programma chiamante? (si ricordi come viene eseguita l'istruzione JAL)

I passi dell'esecuzione delle istruzioni

Nel DLX l'esecuzione di tutte le istruzioni può essere scomposta in *5 passi*, ciascuno eseguito in uno o più cicli di clock.

Tali passi sono detti:

1) FETCH: l'istruzione viene prelevata dalla memoria e posta in IR.

2) DECODE: l'istruzione in IR viene decodificata e vengono prelevati gli operandi sorgente dal Register File.

3) EXECUTE: elaborazione aritmetica o logica mediante la ALU.

4) MEMORY: accesso alla memoria e, nel caso di BRANCH aggiornamento del PC ("branch completion").

5) WRITE-BACK: scrittura sul Register File (registro del banco).

DLX "sequenziale" 24

Sinora, applicando il modello di Von Neumann, abbiamo distinto tra stadi di Fetch ed Execute ma ci siamo accorti che questi comportano implicitamente altre operazioni: ad es. il FETCH di Von Neumann implica che l'istruzione sia decodificata prima di poter essere eseguita.

Esiste una scomposizione in 5 passi dei due passi fondamentali di Von Neumann: sono presentati in questo lucido.

A seconda della classe a cui appartiene un'istruzione assembler, nei 5 passi corrispondono micro-istruzioni diverse. Nei lucidi seguenti vedremo proprio cosa succede per ciascun passo in funzione del tipo di istruzione eseguita.

Non è presente nessun contenuto nuovo riguardante le fasi del processore DLX che abbiamo visto nella descrizione a stati: esse vengono soltanto inquadrate nell'ambito di questa nuova descrizione basata sui 5 passi fondamentali.

Le *micro-operazioni* eseguite in ciascun passo (1)

1) FETCH

MAR ← PC ;
IR ← M[MAR];

2) DECODE

A← RS1, B← RS2, PC← PC+4

DLX "sequenziale" 25

In realtà in questo lucido non ci sono differenze per i passi FETCH e DECODE eseguiti per qualsiasi tipo di istruzione:

il loro scopo è sempre quello di prelevare codice dalla RAM, incrementare il PC (preparandolo per il prossimo prelievo di codice) e copiare gli operandi in A e B.

Le *micro-operazioni* eseguite in ciascun passo (2)

3) EXECUTE

- **Memoria:**

  MAR ← A + $IR_{0..15}$ ## $(IR_{15})^{16}$;

  MDR ← B; ***(RD=RS2 nelle STORE)***

- **ALU:**

  C ← A *op* B (*oppure* $IR_{0..15}$ ## $(IR_{15})^{16}$) ;

- **Branch:**

  Temp ← PC + $IR_{0..15}$ ## $(IR_{15})^{16}$;

  Cond ← A *op* 0 ;

DLX "sequenziale" 26

Il passo EXECUTE invece è diverso a seconda del tipo di istruzione.

Per le istruzioni che fanno riferimento alla memoria, nel passo di EXECUTE si caricano MAR con l'indirizzo di accesso (viene eseguita la somma dell'indirizzo contenuto nel registro con lo spiazzamento -offset- immediato a 16bit) e il MDR col dato, ma quest'ultimo solo se l'istruzione è una STORE.

Per le istruzioni ALU l'"esecuzione" corrisponde al copiare in C il risultato dell'elaborazione dell'ALU che può essere su operandi registro o immediato.

Per le diramazioni BRANCH si esegue l'operazione di verifica della condizione e intanto si carica in TEMP l'indirizzo col salto (relativo al PC, calcolato come PCcorrente+spiazzamento) che sarà usato solo in caso di branch taken e cioè se la verifica della condizione ha dato esito positivo.

Le *micro-operazioni* eseguite in ciascun passo (3)

4) MEMORY

- **Memoria:**

  MDR ← M[MAR]; (*LOAD*)

  M[MAR] ← MDR; (*STORE*)

- **Branch:**

  If (Cond) PC ← Temp;

5) WRITE-BACK

C ← MDR; (*se è una LOAD*)

RD ← C ;

DLX "sequenziale" 27

4 - MEMORY

Per le istruzioni che fanno riferimento alla memoria si accede di fatto in ram nei due modi descritti a seconda che l'istruzione sia una LOAD o una STORE.

Per le istruzioni ALU non viene eseguita nessuna operazione nello stadio MEMORY.

Per i BRANCH, se il check sulla condizione calcolato al passo precedente ha dato esito positivo, si copia l'indirizzo di salto calcolato al passo precedente su PC, compiendo quindi il "branch completion" = aggiornamento del PC con l'indirizzo del salto (anche detto "Branch Target Address").

5 - WRITE-BACK

Per le sole LOAD si scrive sul registro il dato letto dalla ram e contenuto in MDR passando per il registro C. Per le STORE non si fa nulla nel WRITE-BACK.

Per le istruzioni ALU si scrive il contenuto di C sul registro di destinazione (come per le LOAD)

Per i BRANCH non si fa nulla durante il WRITE-BACK.

Esercizio tipo esame

Si supponga di voler estendere il Set di Istruzioni del DLX con la seguente istruzione:

ADD Rx, IMMEDIATE (Ry)

dove Rx, Ry sono due registri di uso generale e IMMEDIATE è un numero di 16 bit con segno.
L'istruzione deve eseguire la somma fra Rx e la word (32 bit) situata all'indirizzo di memoria Ry+IMMEDIATE e quindi deve scrivere il risultato in Rx.

a) Quale fra i 3 formati delle istruzioni DLX risulta il più idoneo alla codifica di questa nuova istruzione ? Come potrebbe essere codificata in binario l'istruzione ADD R1, 20 (R2)?
b) Con riferimento al datapath visto a lezione, si sviluppi il diagramma degli stati che controlla l'esecuzione dell'istruzione ADD Rx, IMMEDIATE (Ry), inserendo anche gli stati necessari alle fasi di fetch e decodifica.
c) A partire dal diagramma degli stati ottenuto e ipotizzando che ogni accesso alla memoria richieda 3 clock, si calcoli il CPI (clock-per-istruzione) della nuova istruzione.