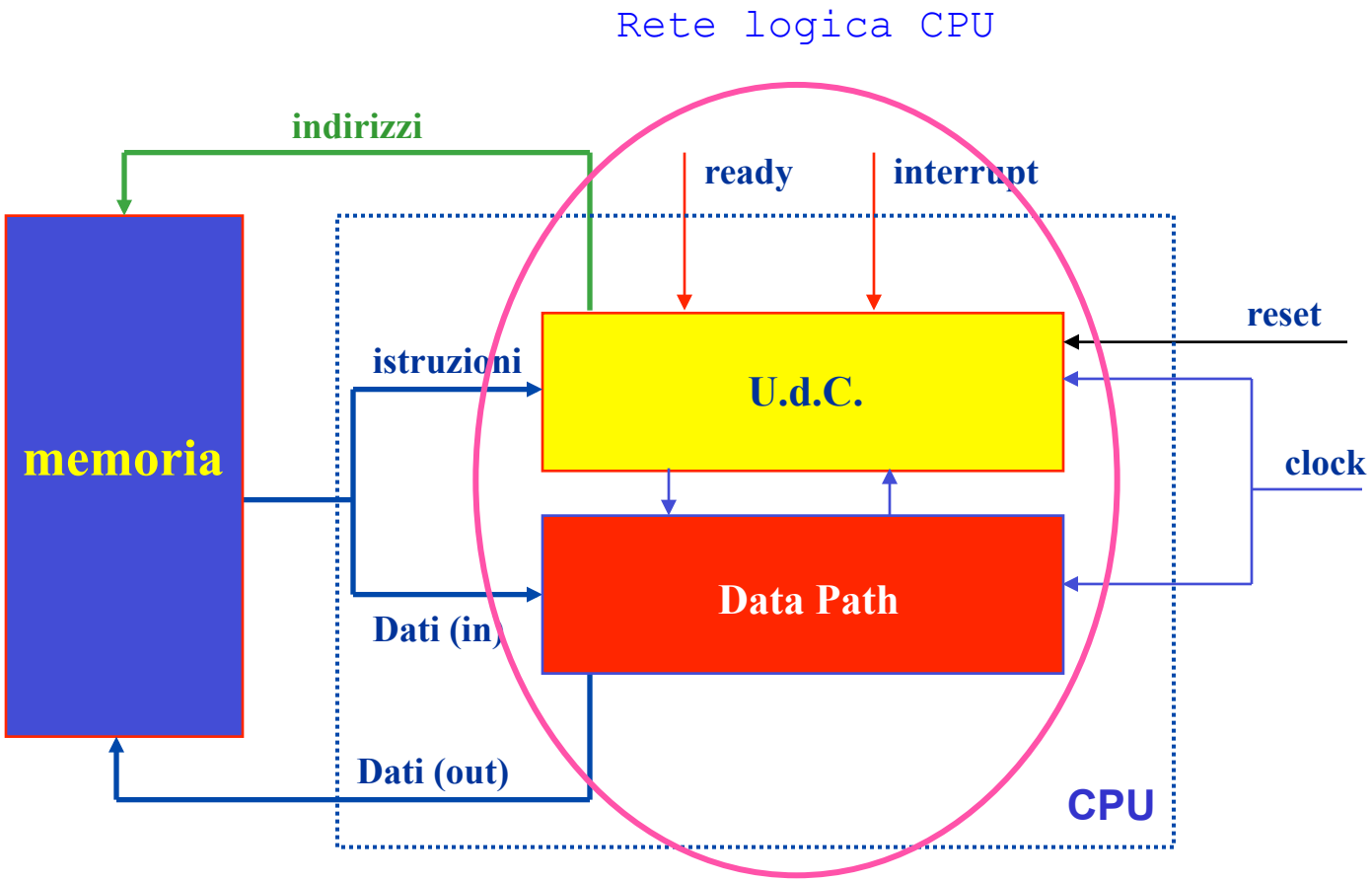
Calcolatori Elettronici T
Ingegneria Informatica

# DLX: implementazione *sequenziale*

# Datapath e Unità di Controllo

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



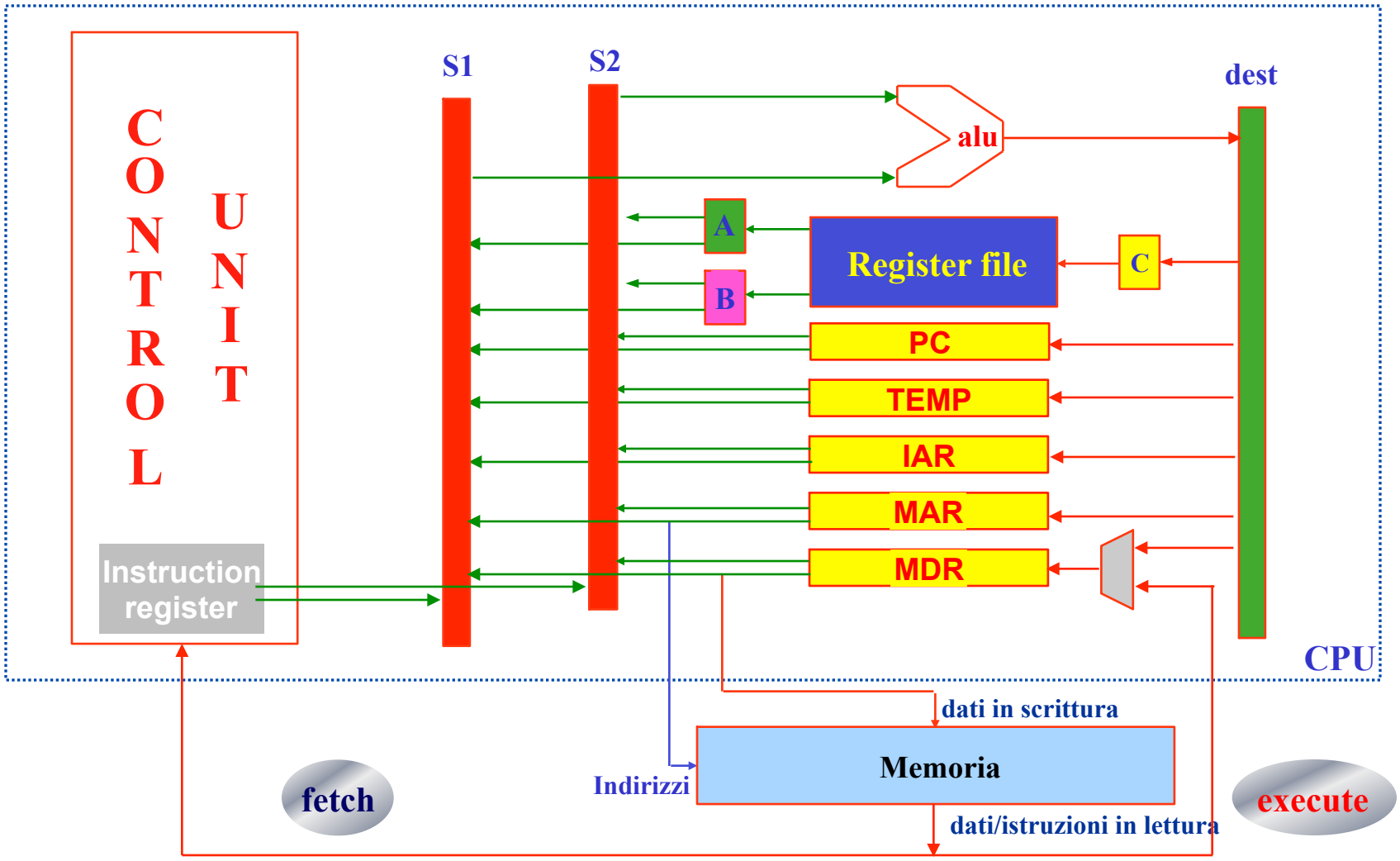
# Datapath e Unità di Controllo

- **Datapath**: contiene tutte le unità di elaborazione ed i registri necessari per l'esecuzione delle istruzioni della CPU. Ogni istruzione appartenente all'ISA 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)

- **Unità di Controllo**: 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*

# Struttura del DLX (esecuzione sequenziale)




Parallelismo dell'architettura: 32 bit

(bus, alu e registri hanno parallelismo 32)

I segnali di controllo non sono riportati !

# I registri del DLX (tutti a 32 bit)

*A parte il Register File questi registri NON sono accessibili al programmatore. In alcuni casi istruzioni speciali per accedere ad alcuni (e.g., IAR)*

- **Register file:** 32 General Purpose Registers R0….R31 con R0=0
- **IAR:** Interrupt Address Register – Deposito dell'indirizzo di ritorno in caso di interruzione
- **PC:** Program Counter
- **MAR:** Memory Address Register – Contiene l'indirizzo del dato da scrivere o leggere in memoria
- **IR:** Instruction Register – Contiene l'istruzione attualmente in esecuzione
- **TEMP:** Temporary Register – Registro di deposito temporaneo di risultati
- **MDR:** Memory Data Register – Registro di transito temporaneo dei dati da e per la memoria
- **A e B:** Registri di uscita dal Register File

# Funzioni della ALU

**Dest (uscite) – 4 bit di comando**

S1 + S2
S1 – S2
S1 and S2
S1 or S2
S1 exor S2
Shift S1 a sinistra di S2 posizioni
Shift S1 a destra di S2 posizioni
Shfit S1 aritmetico a destra di S2 posizioni
S1
S2
0
1

**Flag di uscita**

Zero
Segno negativo
Carry

- La **ALU** è una rete **PURAMENTE combinatoria**
- Non esiste nel DLX un *registro di flag*

# Trasferimento dati sul datapath

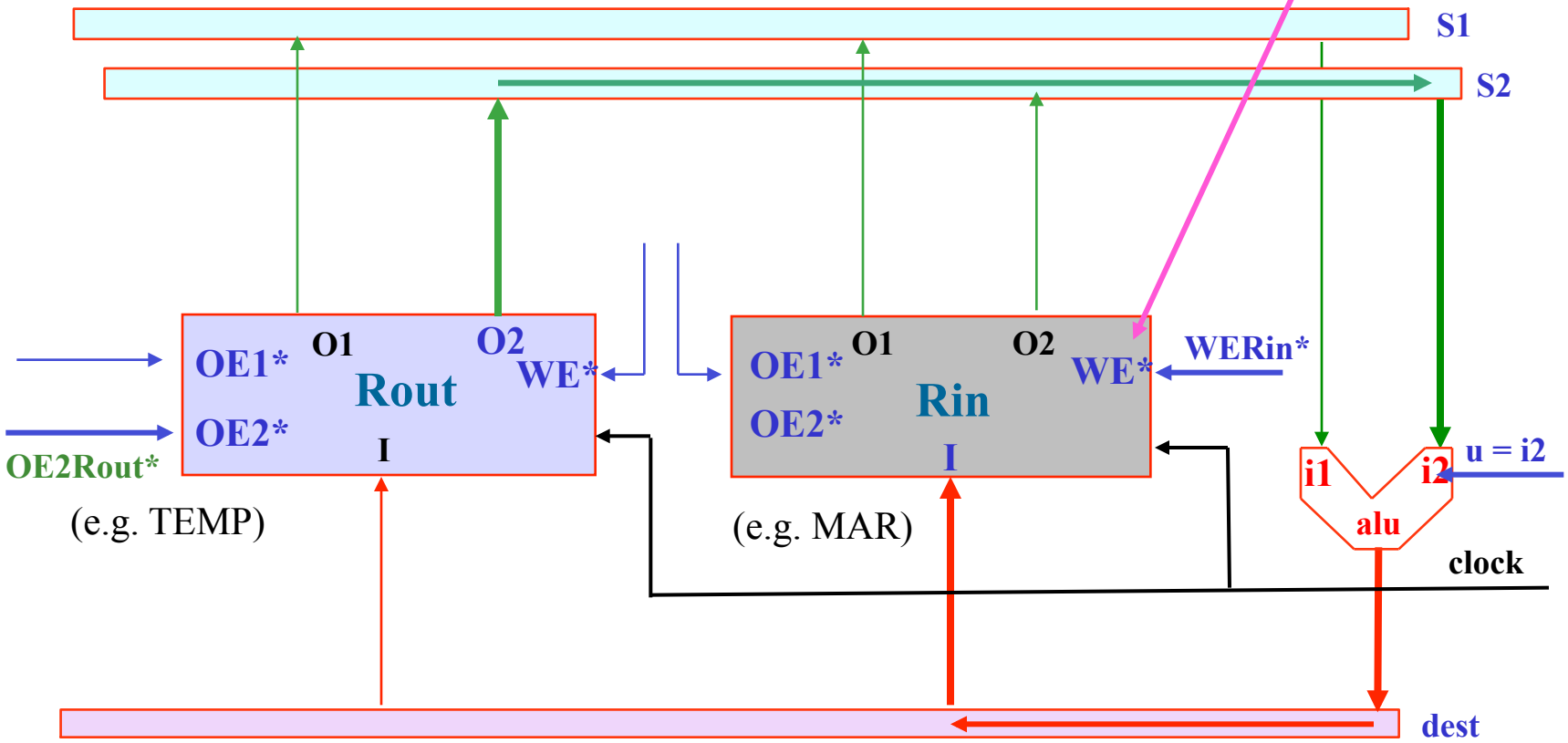
- I bus S1 ed S2 sono multiplexati (tri-state) con parallelismo 32 bit.
- I registri campionano sul fronte positivo del clock, hanno due porte di uscita O1 e O2 per i due bus (o i registri A e B) e dispongono di tre ingressi di controllo:
  - un ingresso di Write Enable (WE*) ed uno di Output Enable per ogni porta di uscita, una per ogni bus S1 e S2 (OE1* e OE2*).
- 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 data path si calcola come segue:

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

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

# Esempio : esecuzione della microistruzione Rin ← Rout

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

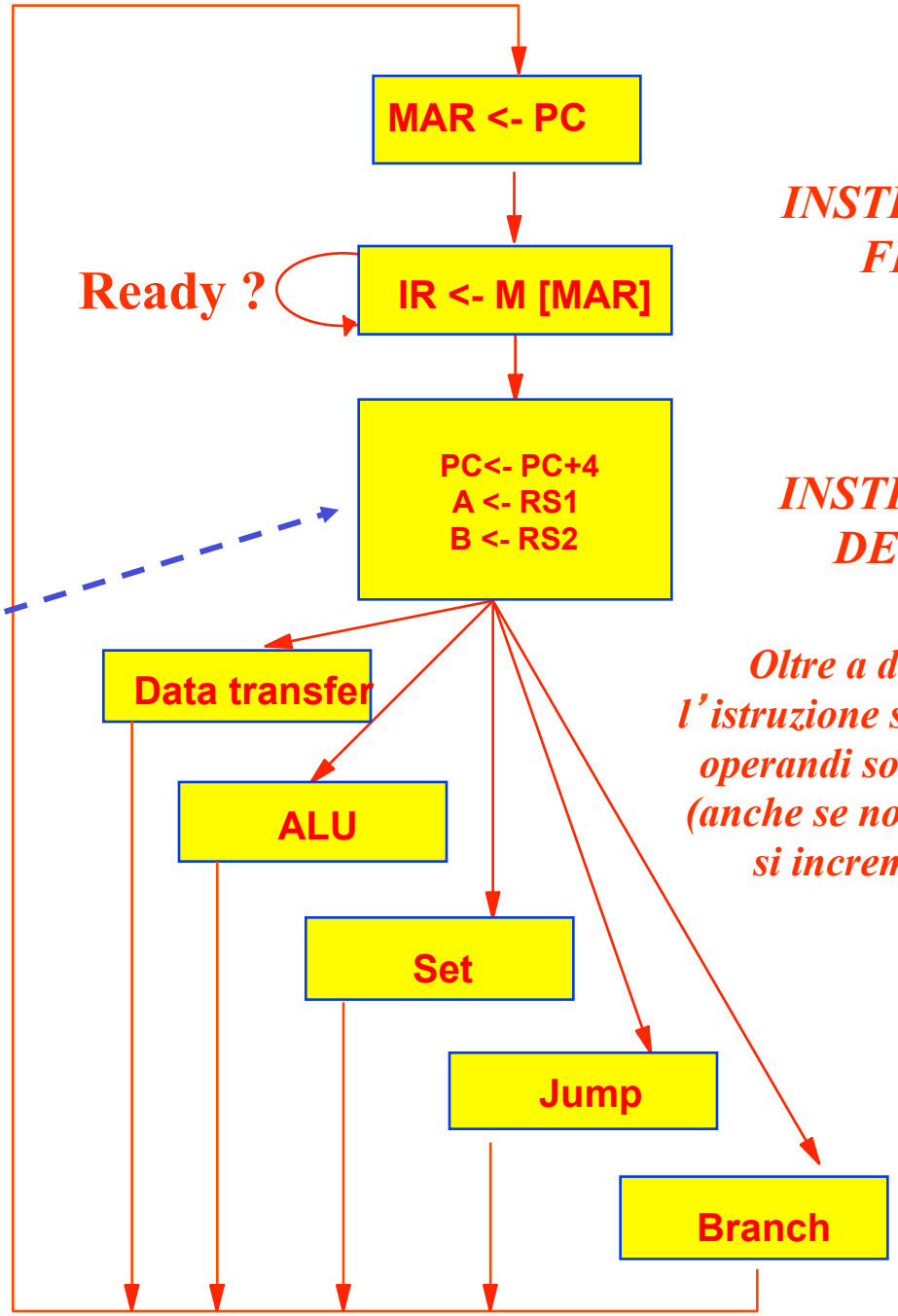



I segnali di controllo in grassetto sono attivi nel ciclo di clock in cui il micro-step Rin ← Rout viene eseguito

# 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 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




*INSTRUCTION FETCH*

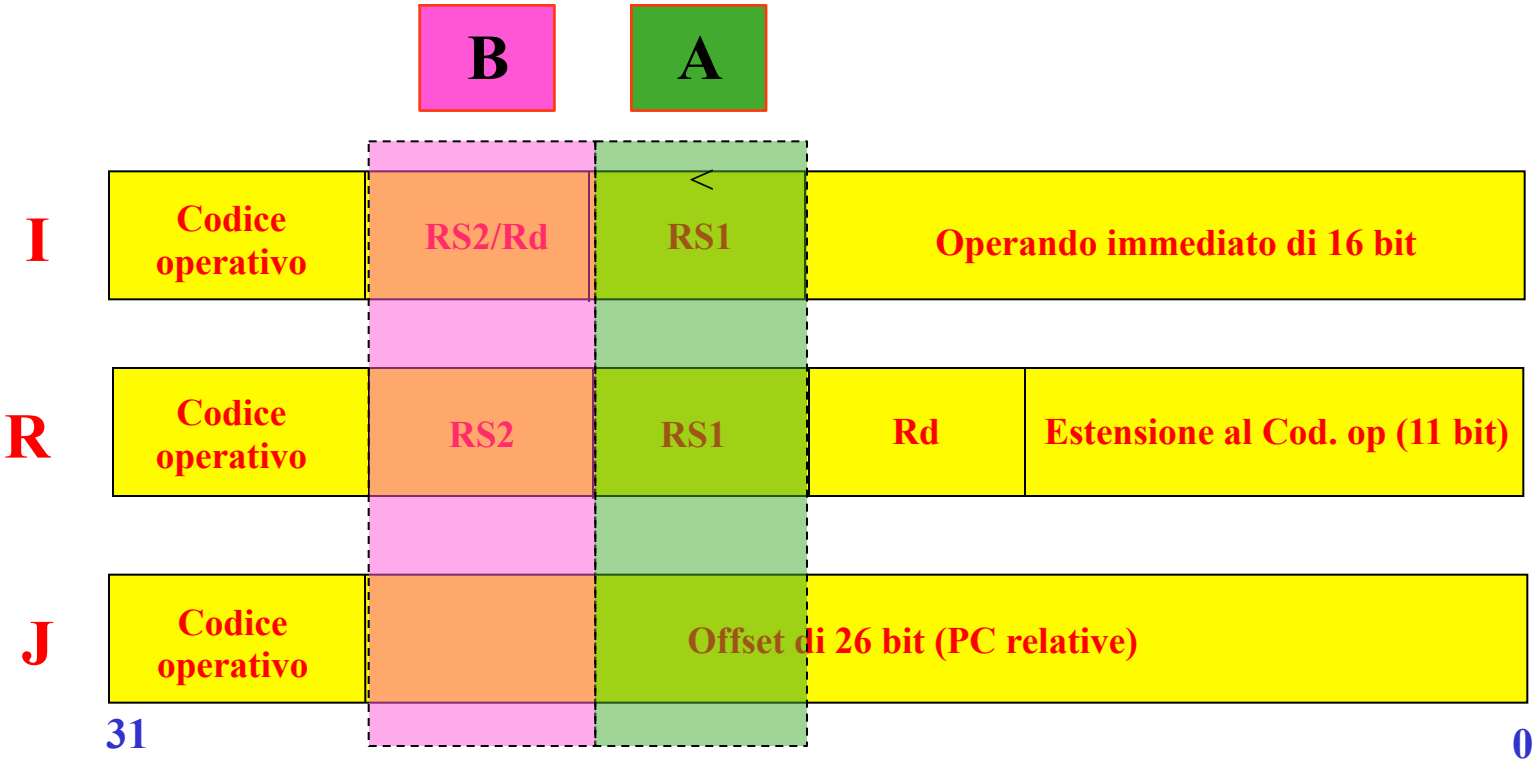
*INSTRUCTION DECODE**

*Oltre a decodificare l'istruzione si prelevano gli operandi sorgente dal RF (anche se non utilizzati !) e si incrementa il PC.*

**Qui non si sa ancora quale sia l'istruzione ma il trasferimento ai registri è fatto comunque !!**

**N.B. I primi tre stadi sono comuni a *tutte* le istruzioni**

# Estrazione “automatica” dei registri durante la fase di *decode* di una istruzione (qualsiasi)

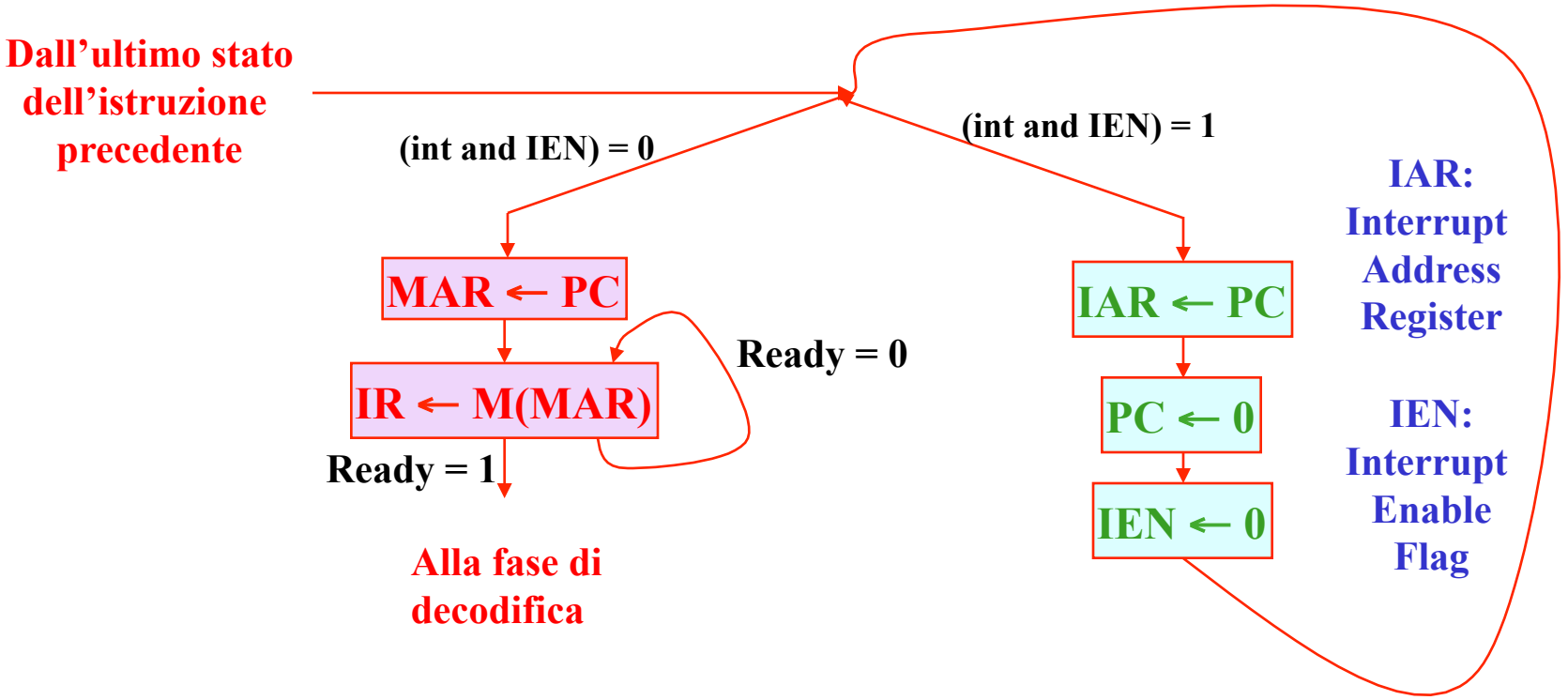



*Questi 5 + 5 bit sono utilizzati per estrarre, preventivamente e ancora prima di conoscere che tipo di istruzione che è stata letta dalla memoria, dal Register File due registri in A e B. Nel caso di istruzione J non ci sono registri coinvolti e quindi saranno estratti bit corrispondenti all’offset. Nel caso di istruzione I, in B potrebbe finire il valore del registro destinazione (e.g. in una LD o operazione ALU (tipo I)).*

*Infine: i 5 + 5 bit rappresentano gli indici (o presunti tali) ma non il valore dei due registri che è contenuto nel Register File.*

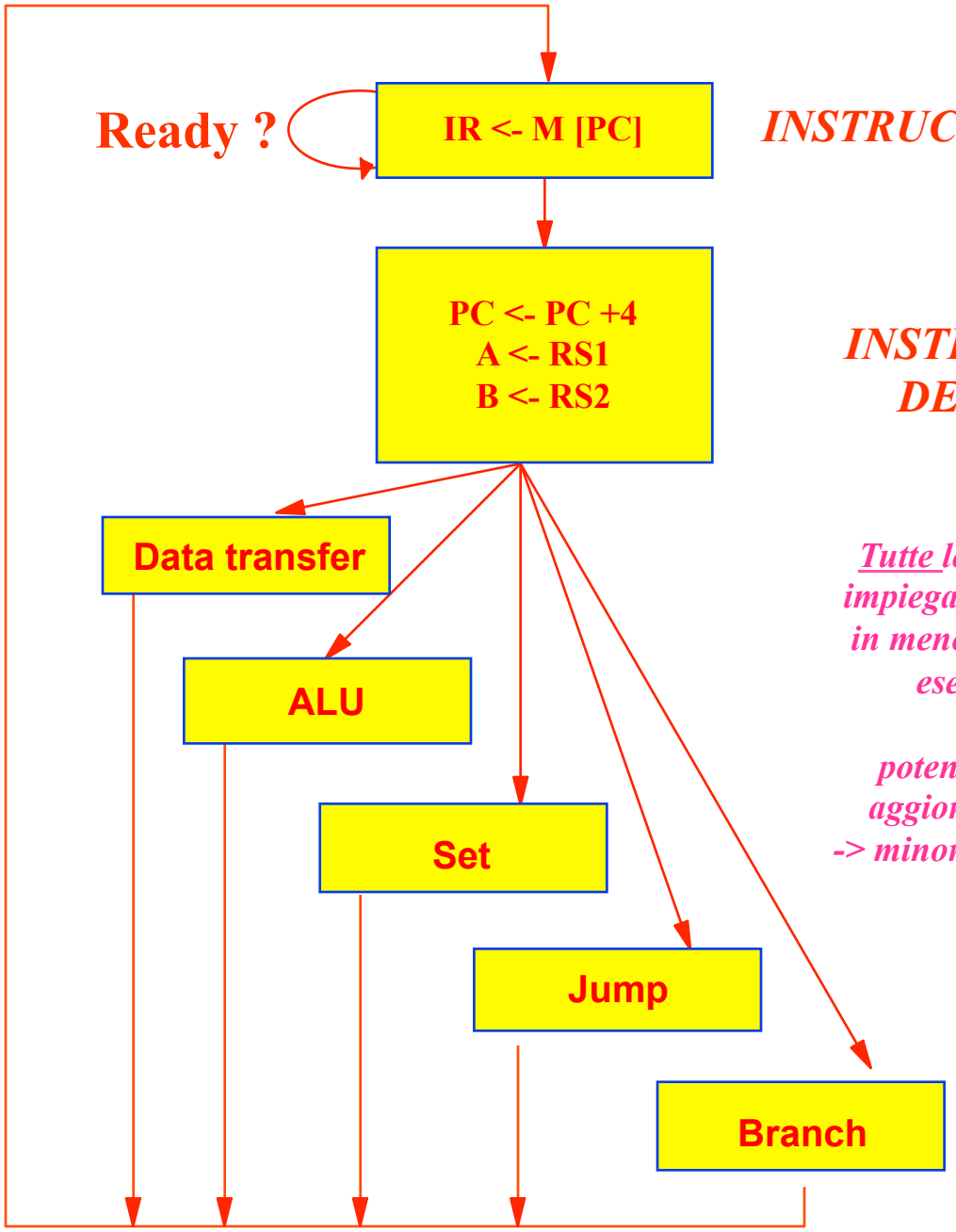
# Gli stati della fase di fetch

- 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) si esegue implicitamente l'istruzione di chiamata a procedura all'indirizzo 0, e si salva l'indirizzo di ritorno nell'apposito registro IAR;
- se l'interrupt non è presente o le interruzioni non sono abilitate, si va a leggere in memoria la prossima istruzione da eseguire (il cui indirizzo è in PC)



# Il diagramma degli stati del controller

*Si modifica il DATAPATH in maniera da poter indirizzare la memoria dal PC. Meno stati ma maggiore complessità*

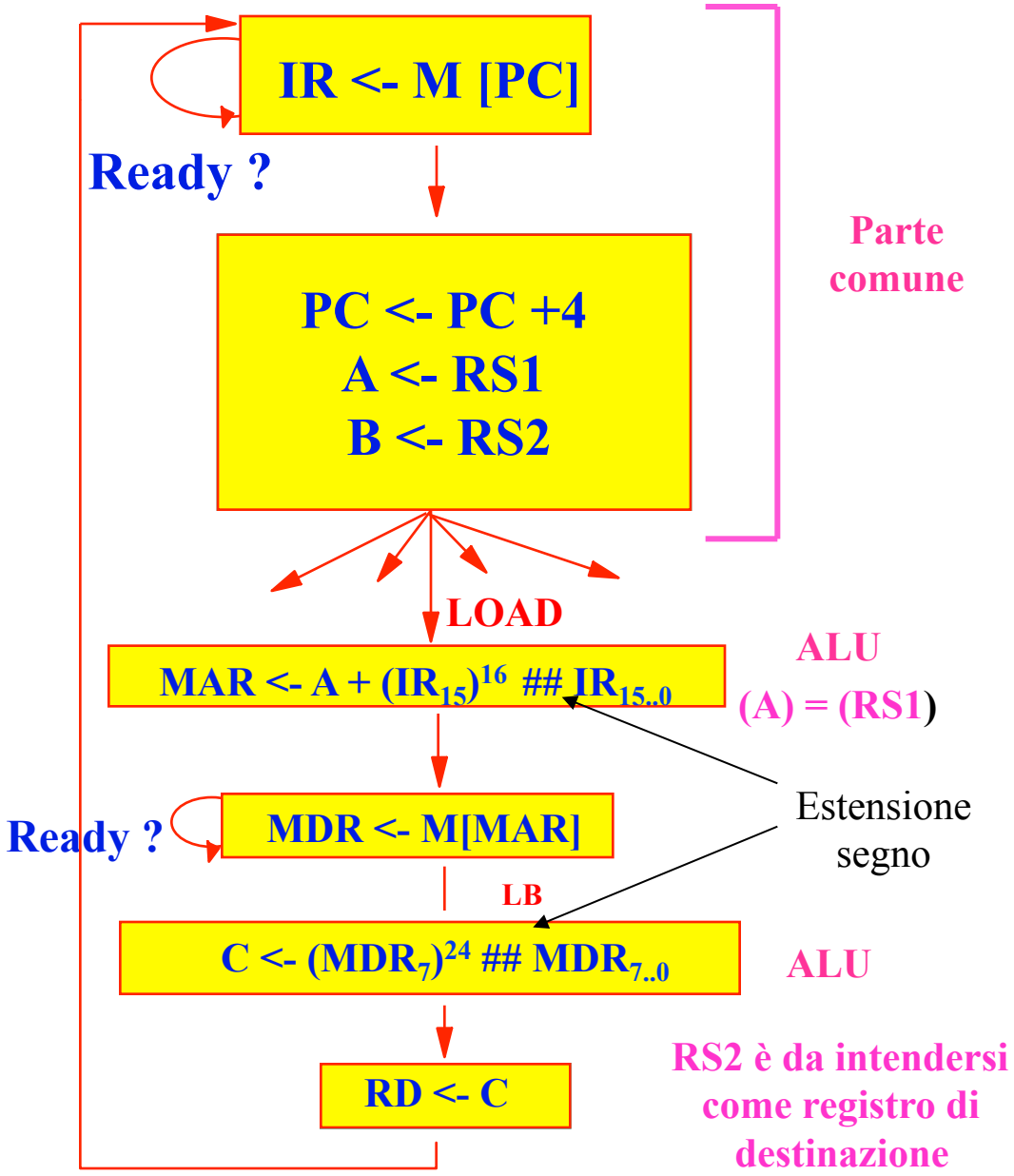



*INSTRUCTION FETCH*

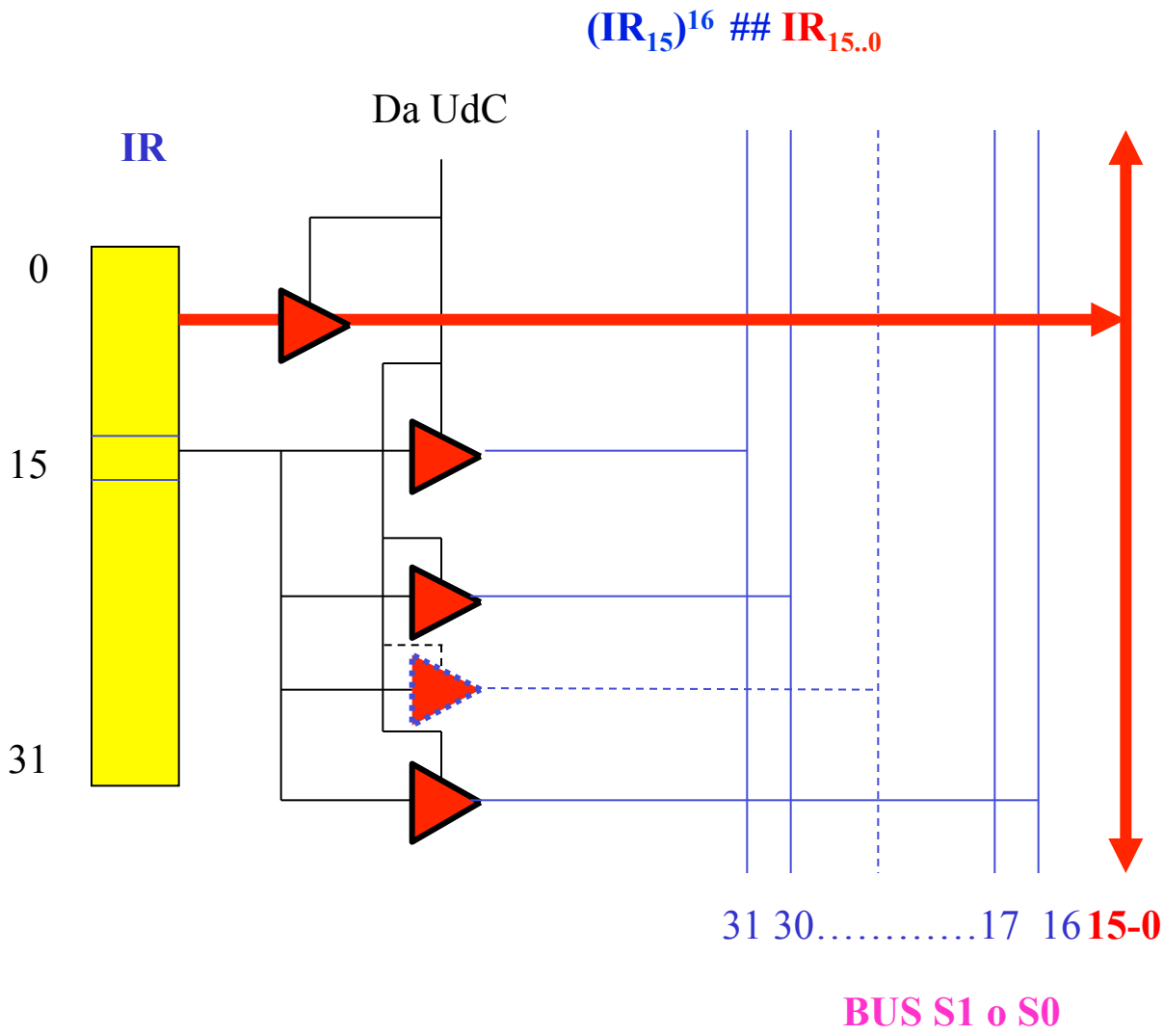
*INSTRUCTION DECODE**

*Tutte le istruzioni impiegano un clock in meno per essere eseguite ! Ma potenzialmente aggiore lentezza -> minore freq. clock*

# Controllo per l'istruzione LB *(LOAD BYTE)*



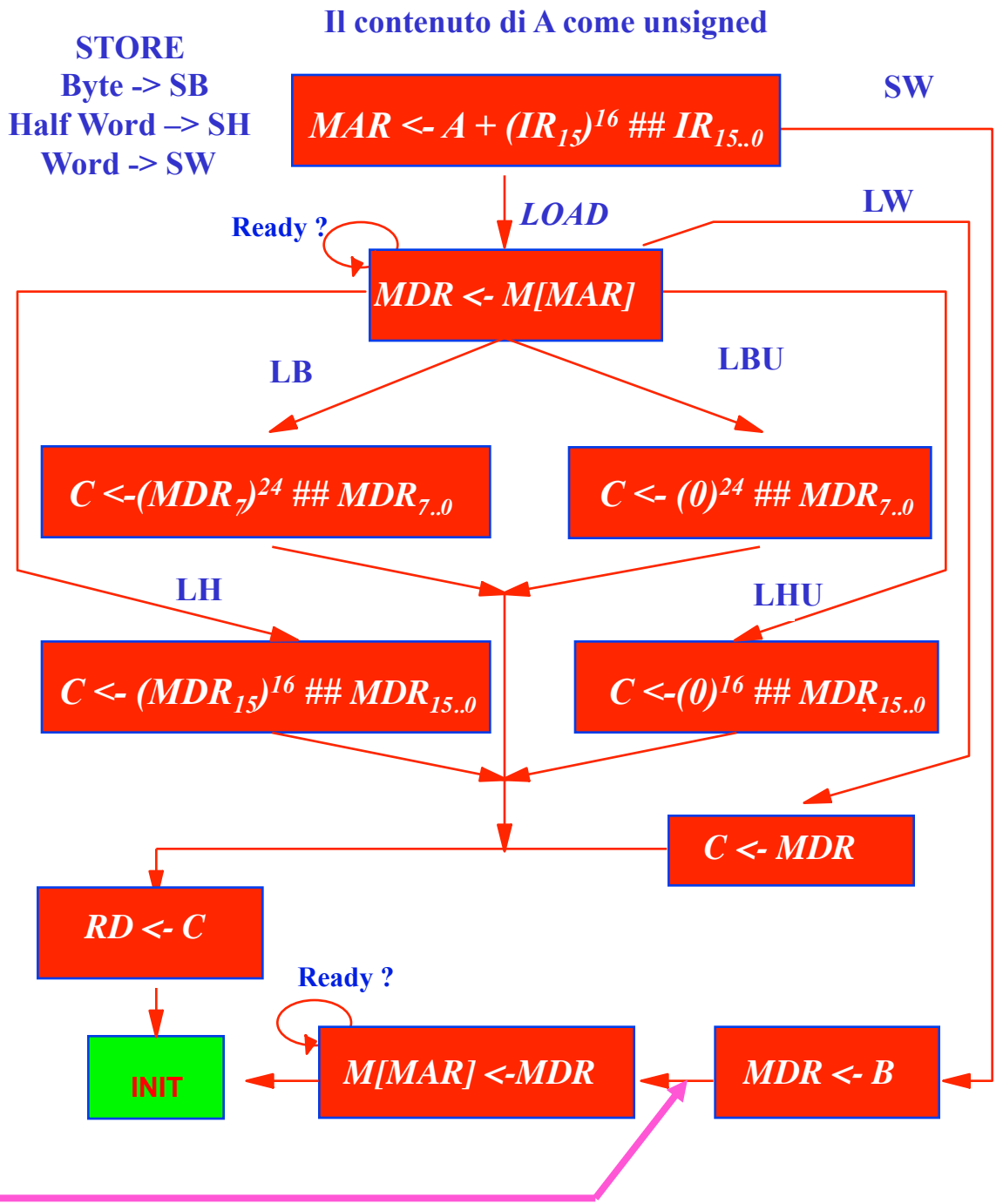
# Estensione del segno



# Controllo per le istruzioni di DATA TRANSFER

***NB: in lettura la parte meno significativa del dato viene letta sempre allineata al registro MDR per permettere il filling***

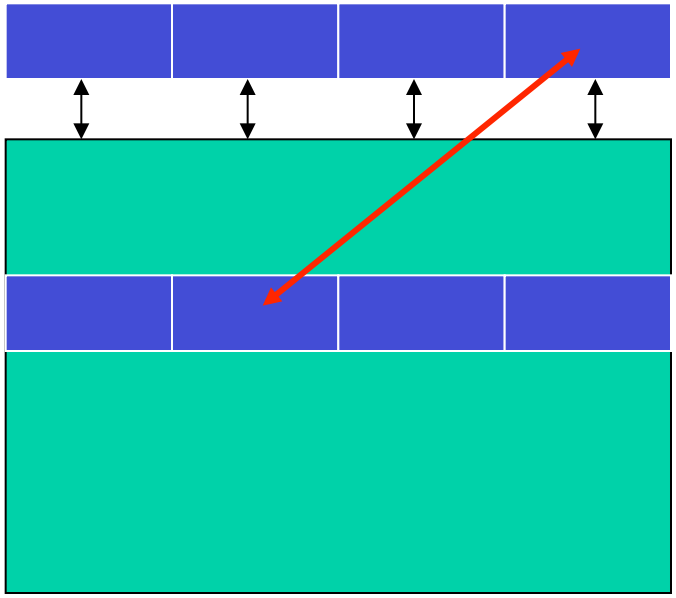
Mancano nell'esempio SH e SB (sempre unsigned) che corrispondono a attivazione degli specifici WE delle memorie e "traslatori" dei bytes del registro MDR.
Come si realizzerebbero ?



# Trasferimenti BYTE, HW

- I trasferimenti di bytes sono SEMPRE considerati allineati
- I trasferimenti di HW debbono avvenire a indirizzi multipli di 2
- I trasferimenti di Word debbono sempre avvenire a indirizzi multipli di 4
- In caso di disallineamento: fault
- Nel caso di store di dati di dimensione inferiore alla word NON si ha estensione del segno
- La lettura/scrittura di bytes e HW (a causa del reciproco disallineamento fra i registri e la memoria) implica che fra i registri e la memoria siano interposti dei mux/demux (realizzati con tristate)

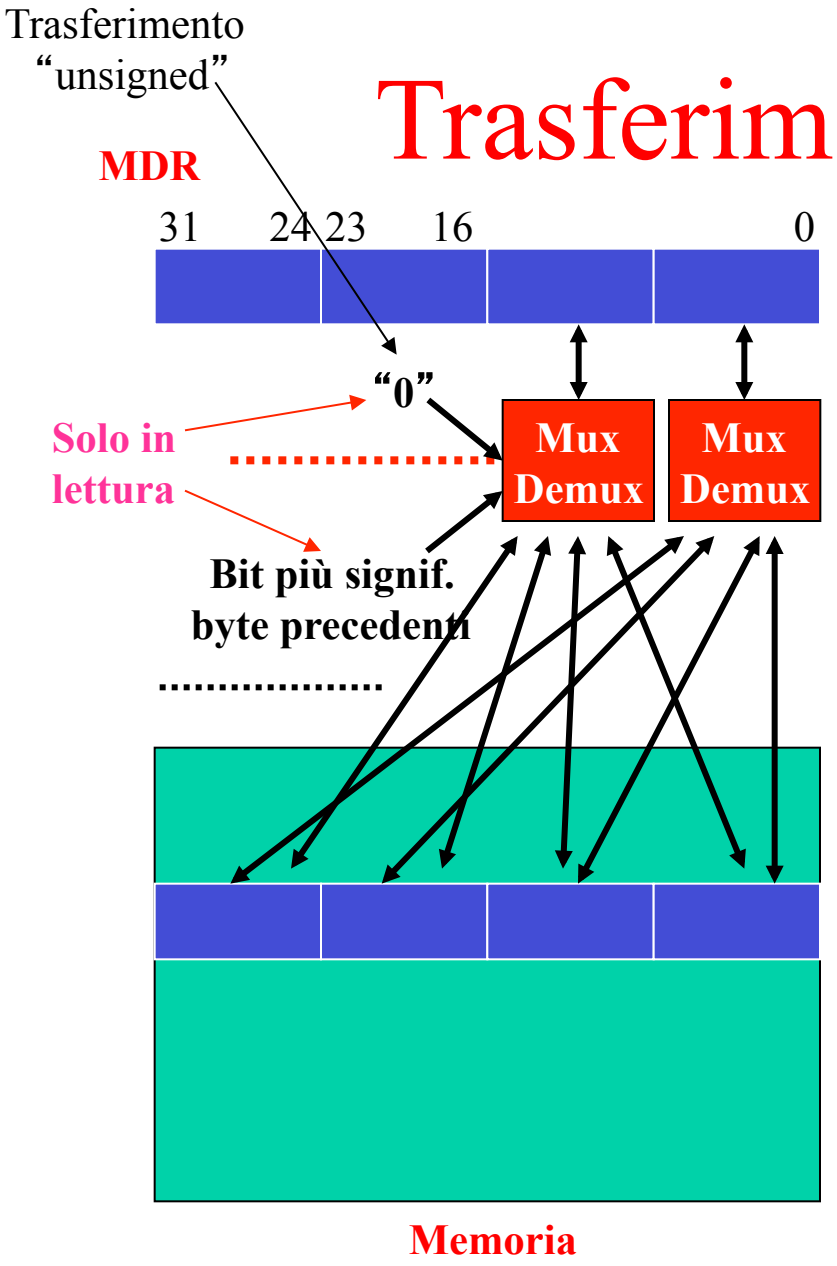
Registro MDR



Memoria

Come sono attivati i WE delle memorie ? Progettare la rete

# Trasferimenti BYTE, HW

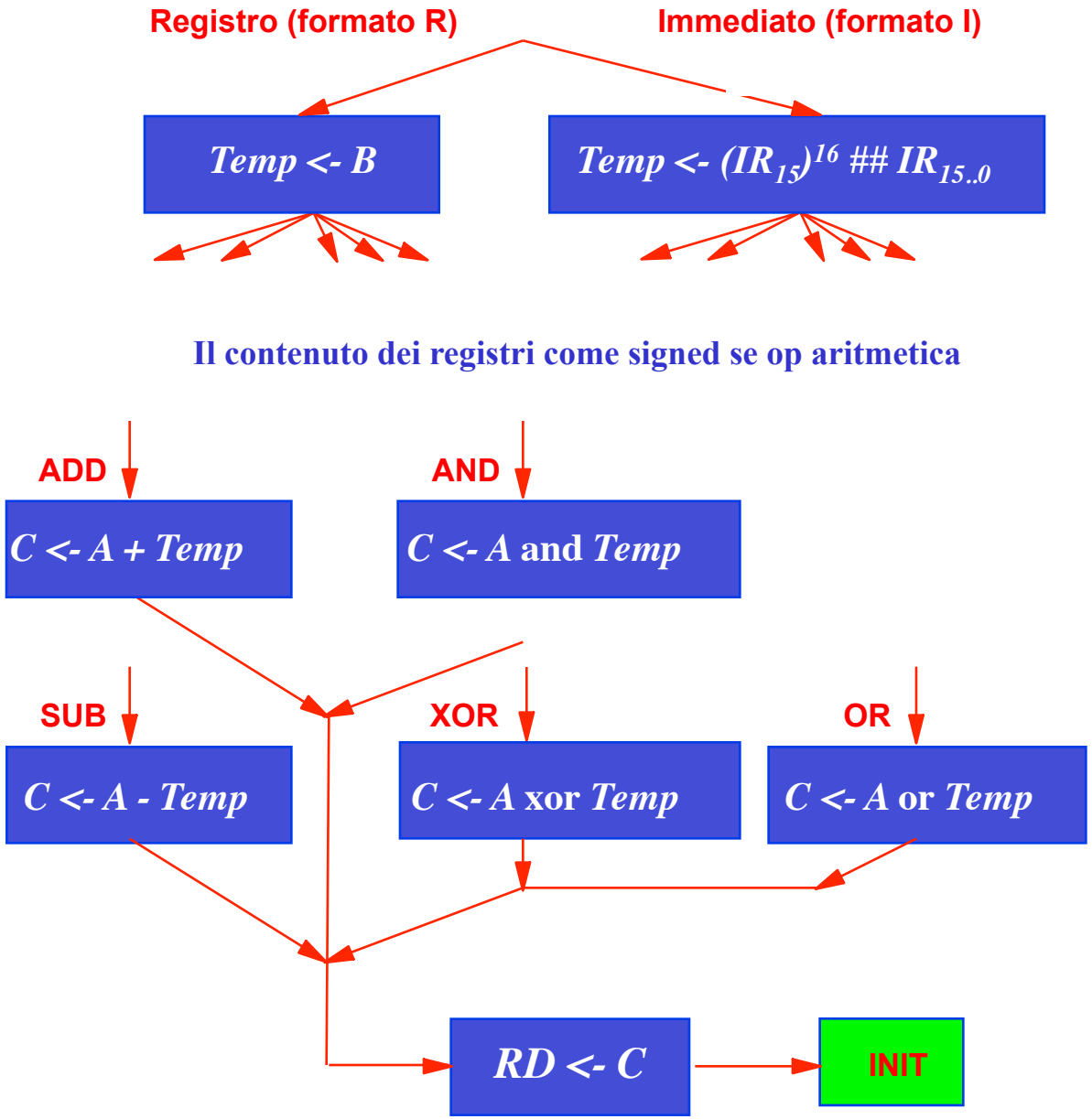



**I MUX 23-16 e 31-24 hanno come ingresso anche il bit 7 del byte 7-0 della memoria (LB) e il bit 15 del byte 15-8 della memoria (LH) Ad esempio in una LB il MUX 7-0 si collega direttamente alla memoria mentre i MUX 15-8, 23-16 e 31-24 si collegano al bit 7 del MUX 7-0 proveniente dalla memoria. In una SH a indirizzo multiplo di 2 e non di 4 il DEMUX 7-0 dal MDR si collega alla memoria 23-16 e il DEMUX 15-8 alla memoria 31-24. Gli altri due bytes della memoria rimangono invariati**

# Esempi di istruzioni ALU

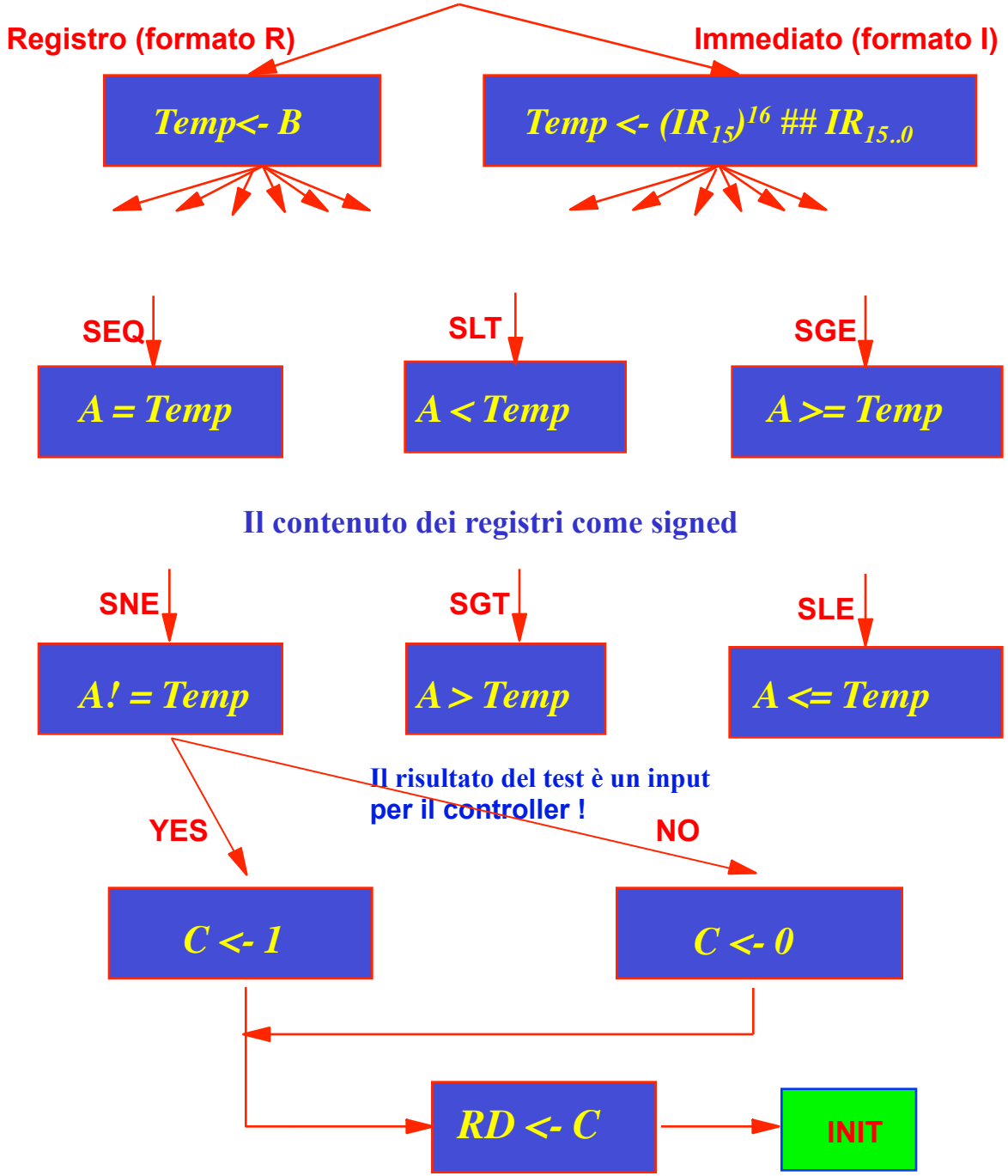
**Duplicando i percorsi si potrebbe risparmiare il passaggio in TEMP**

Lo stesso schema si può usare per gli shift etc.



# Controllo per le istruzioni di SET (*confronto*)

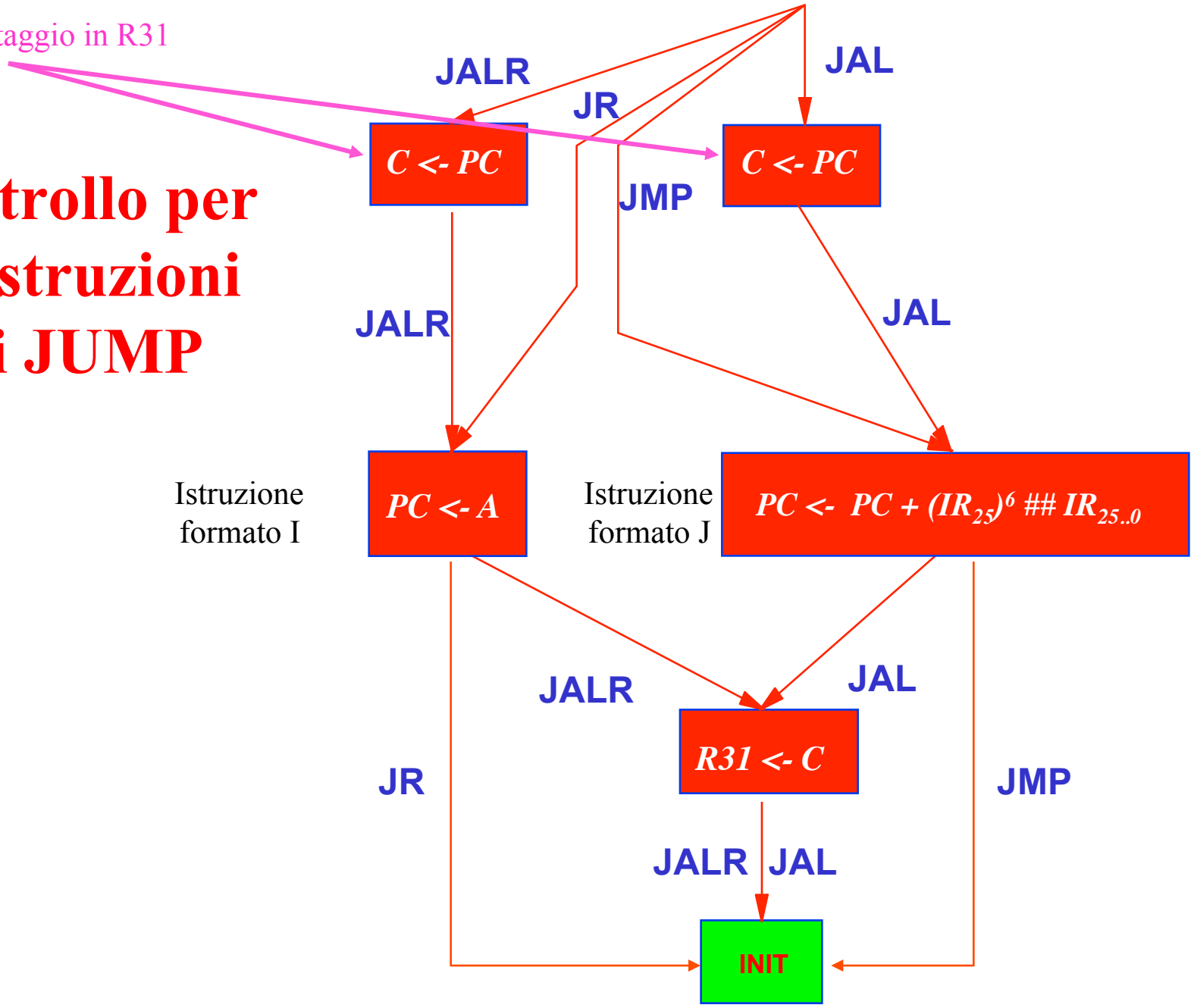
**ex. SLT R1,R2,R3**




**Duplicando i percorsi si potrebbe risparmiare il passaggio in TEMP**

I micropassi sono eseguiti in ALU ma il risultato ***NON è memorizzato*** in un registro: i flag sono utilizzati dalla ALU per impostare (almeno) il bit 0 del registro C

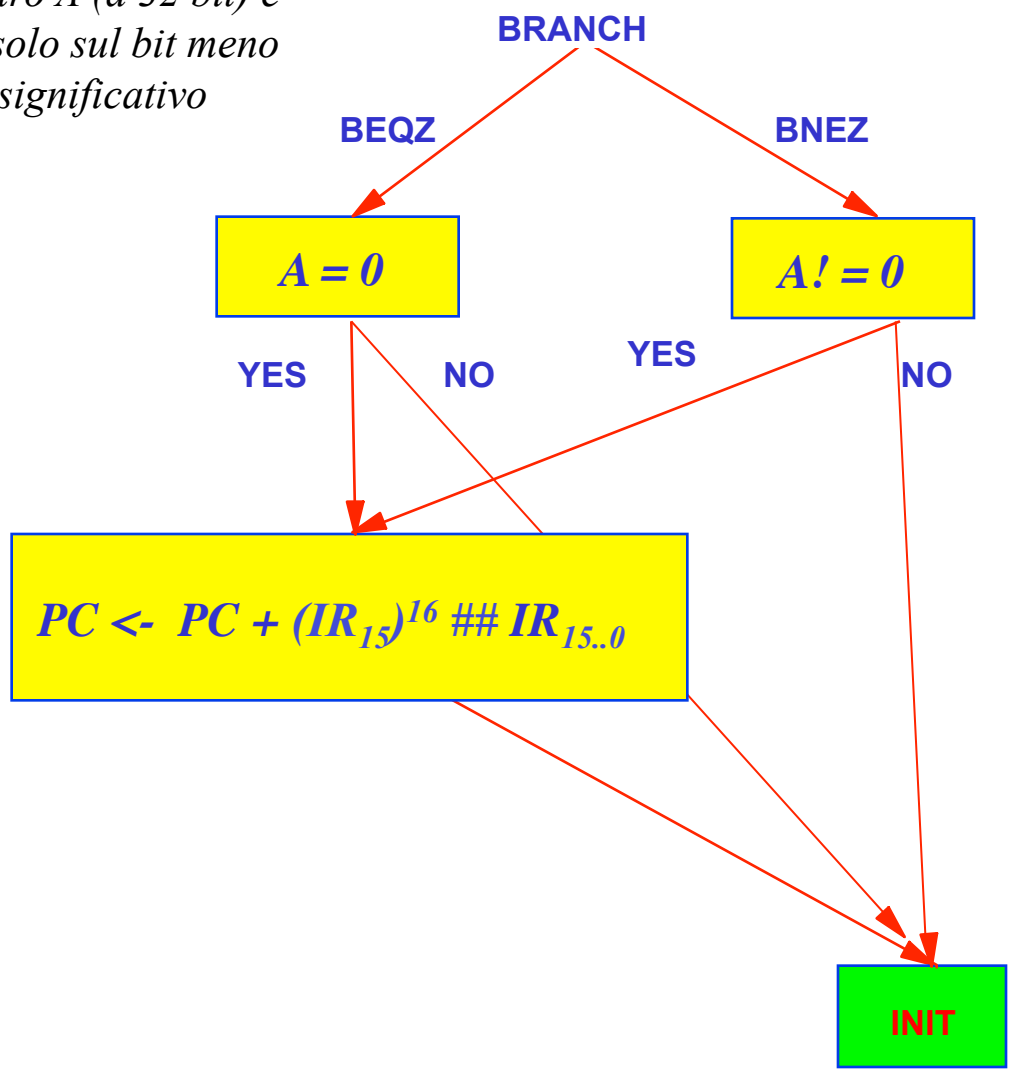
# Controllo per le istruzioni di JUMP



# Controllo per le istruzioni di BRANCH

**Ex. BNEQZ R5, 100**

*Il controllo se 0 (o !=0) è fatto sull'intero registro A (a 32 bit) e non solo sul bit meno significativo*



# 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 |

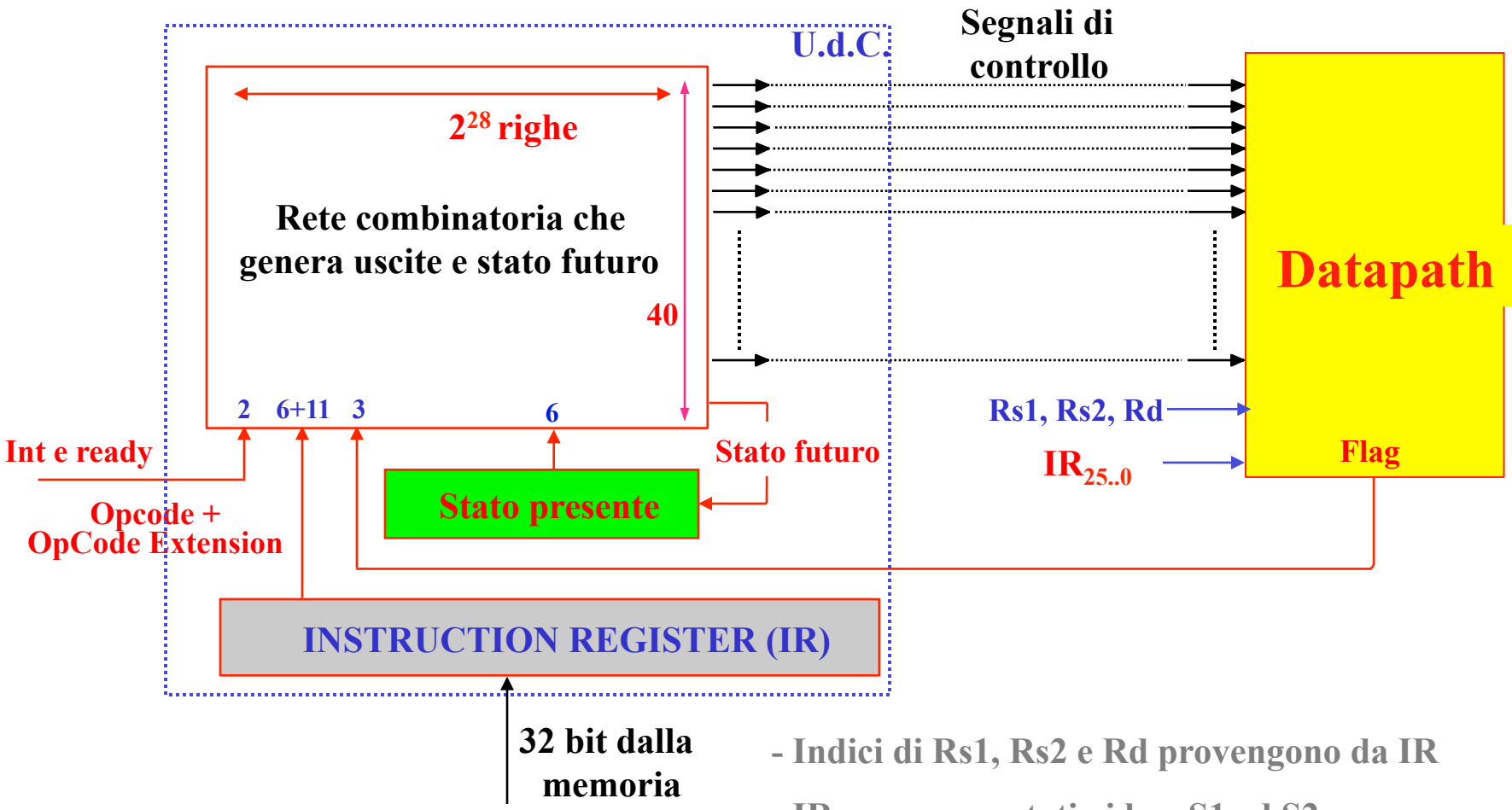
$$CPI = \sum_{i=1}^{n} \left( CPI_i * \frac{N_i}{\text{numero totale di istruzioni}} \right)$$

*Esempio su DLX*

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

→ ***CPI = 6.3***

# Controllo cablato ("*hardwired*")




- Indici di Rs1, Rs2 e Rd provengono da IR

- $IR_{25..0}$ sono portati ai bus S1 ed S2 del data path attraverso due buffer tristate

- U.d.C. genera anche i segnali di comando per la memoria (MEMRD e MEMWR)

# 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.**

# Le *micro-operazioni* eseguite in ciascun passo

**1) FETCH**

**MAR ← PC ;**

**IR ← M[MAR];**

**2) DECODE**

**A← RS1, B← RS2, PC← PC+4**

# Le *micro-operazioni* eseguite in ciascun passo

## 3) EXECUTE

- **MEMORIA:**

**MAR ← A + $(IR_{15})^{16}$ ## $IR_{15..0}$ ;** **(utilizzano ALU, S1, S2, dest)**

**MDR ← B;** ***(NB: serve nelle Store ove RD=RS2 operazione non significativa nelle LOAD)***

- **ALU:**

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

**C <- sign( A *op* B (*oppure* A op $(IR_{15})^{16}$ ## $IR_{15..0}$)); se SCn**

- **BRANCH:**

**Temp ← PC + $(IR_{15})^{16}$ ## $IR_{15..0}$) ;** **(utilizza ALU, S1, S2, dest: qui non si sa ancora se si deve saltare)**

- **J e JAL**

**Temp ← PC + $(IR_{25})^{6}$ ## $IR_{25..0}$) ;**

- **JR e JALR**

**Temp ← A;**

# Le *micro-operazioni* eseguite in ciascun passo

## 4) MEMORY

- **Memoria:**

**MDR ← M[MAR]; (*LOAD*)**

**M[MAR] ← MDR; (*STORE*)**

- **BRANCH:**

**If (Cond) PC ← Temp;**

**[A] è il registro che condiziona il salto (Cond) ;**

- **JAL e JALR:**

**C ← PC;**

# Le *micro-operazioni* eseguite in ciascun passo

## 5) WRITE-BACK

- **istruzioni diverse da J, JR, JAL, JALR**

**C ← MDR; (*se è una LOAD – due micropassi)*)**

**RD ← C ;**

- **istruzioni J, JR, JAL, JALR**

**PC ← Temp;**

**RD ← C ;**