UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE E DI ECONOMIA

TESINA DI ELETTRONICA DEI SISTEMI DIGITALI 2

Implementazione di metodi di indirizzamento specifici per un sistema digitale a logica programmata

Giacomo Mammarella

ANNO ACCADEMICO 2017–2018

Indice

Capitolo 1

Introduzione

La realizzazione di un sistema digitale complesso è possibile mediante due diverse strategie realizzative basilari: la sintesi in *hardware diretto* e la sintesi in *logica programmata*. Entrambe queste metodologie realizzative, seppur presentando differenze architetturali sostanziali, possono garantire la sintesi di due sistemi digitali assolutamente equivalenti dal punto di vista del legame ingresso-uscita, che mantengano ossia prestazioni funzionali invariate.
La scelta della metodologia di realizzazione di un sistema digitale dipende dalle circostanze strettamente legate alla funzionalità del sistema da realizzare. Ogni soluzione presenta i propri vantaggi e svantaggi ed è compito del progettista verificare che la soluzione adottata sia compatibile con le specifiche imposte dal problema in esame.

Logica Programmata

In generale, un esecutore a logica programmabile è un sistema dotato di un set di istruzioni finito, la cui combinazione appropriata può garantire l'implementazione di algoritmi generici definibili dall'utente.
Il sistema dunque dispone di un'architettura interna in grado di supportare l'esecuzione di istruzioni base, il cui insieme definisce *instruction-set*. Architettura e set di istruzioni non sono parametri fissi, dunque il ventaglio delle soluzioni disponibili nella realizzazione di un esecutore programmabile è molto ampio, esistendo una molteplicità di soluzioni distinguibili in prima approssimazione secondo i parametri di *area occupata*, *performance* e *potenza dissipata*.

A monte è possibile tuttavia differenziare i sistemi a logica programmabile secondo il tipo di architettura interna adottata. Storicamente, agli albori dello sviluppo dei microcontrollori digitali integrati, nacquero sistemi come l'Intel 4004 basati su un'architettura denominata *archietettura di Von Neumann*, dal nome del matematico che la introdusse. Tale architettura costituisce un esecutore programmabile strettamente sequenziale di tipo master-slave completo ed è internamente strutturata come nella figura seguente.

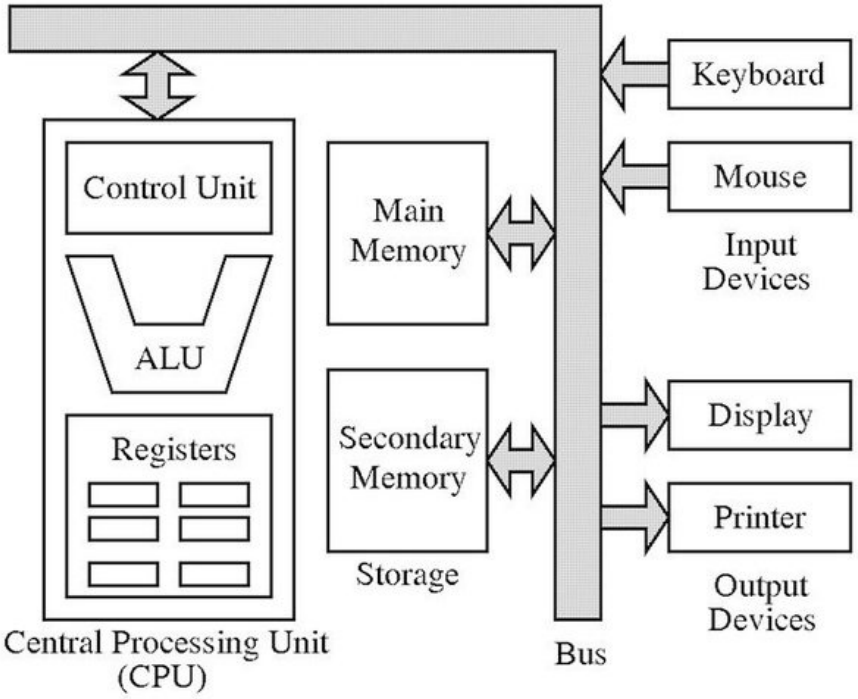


Figura 1.1: Architettura di Von Neumann.

La struttura così costituita è composta da tre componenti fondamentali a cui vanno ad aggiungersi le periferiche collegate tramite un modulo di input - output. Tali componenti sono:

- Il sistema di memoria, dove risiedono le istruzioni ed i dati necessari all'esecuzione del programma. La memoria può essere costituita da diversi livelli gerarchici. In linea generale si può considerare il "sistema di memoria" come una memoria principale di tipo RAM nella quale, a runtime, siano già precaricate le istruzioni ed i dati relativi al programma in esecuzione. La memoria

può essere vista come un insieme di registri indirizzabili singolarmente e di tipo general purpose, aventi ossia la possibilità di mantenere dati ed istruzioni.

- Il sistema di calcolo centralizzato (CPU) che si occupa delle fasi di esecuzione delle istruzioni caricate nella memoria, ossia di eseguire le fasi di FETCH, DECODE, EXECUTE, WRITE BACK per ogni singola istruzione. Al suo interno contiene il sistema di calcolo aritmetico-logico (ALU), l'unità di controllo ed un banco di registri ad alta velocità nel quale i dati vengono salvati mano a mano che viene eseguito il processamento di un'istruzione.

- il BUS, che interconnette con la CPU la memoria ed i dispositivi di ingresso - uscita facenti parte del sistema. É un bus unico per dati, istruzioni e comandi di controllo, pertanto costituisce un limite prestazionale di tale architettura di calcolo.

Come detto su tale architettura è possibile definire un insieme delle istruzioni elementari eseguibili, denominato *instruction-set*. All'interno dell'instruction-set sono disponibili le istruzioni, i registri, le modalità di indirizzamento, l'architettura della memoria, la gestione degli interrupt e delle eccezioni, ed eventualmente l'indirizzamento per i dispositivi di I/O esterni.
La combinazione degli elementi presenti all'interno dell'instruction-set in maniera non univoca e più o meno efficiente, permette l'implementazione di un algoritmo fisicamente eseguibile sull'architettura hardware adottata. L'instruction-set corrisponde dunque a un'interfaccia tra software ed hardware che permette al progettista dell'algoritmo di operare in maniera più semplice sui dispositivi e i segnali interni al sistema digitale col fine dell'implementazione dell'algoritmo stesso.
La progettazione dell'instruction-set dipende dall'architettura adottata nel progetto dell'esecutore, differenziando tra sistemi di tipo "general purpose" come i microprocessori o "special purpose" come ad esempio i microcontrollori. A titolo di esempio di un noto set di istruzioni si riporta di seguito quello relativo al microprocessore Intel 4004, il primo microprocessore integrato a 4 bit rilasciato da Intel nel 1971.

Intel 4004 Instructions Set

INSTRUCTION	MNEMONIC	BINARY EQUIVALENT 1st byte	2nd byte	MODIFIERS
No Operation	NOP	00000000	-	none
Jump Conditional	JCN	0001CCCC	AAAAAAAA	condition, address
Fetch Immediate	FIM	0010RRR0	DDDDDDDD	register pair, data
Send Register Control	SRC	0010RRR1	-	register pair
Fetch Indirect	FIN	0011RRR0	-	register pair
Jump Indirect	JIN	0011RRR1	-	register pair
Jump Uncoditional	JUN	0100AAAA	AAAAAAAA	address
Jump to Subroutine	JMS	0101AAAA	AAAAAAAA	address
Increment	INC	0110RRRR	-	register
Increment and Skip	ISZ	0111RRRR	AAAAAAAA	register, address
Add	ADD	1000RRRR	-	register
Subtract	SUB	1001RRRR	-	register
Load	LD	1010RRRR	-	register
Exchange	XCH	1011RRRR	-	register
Branch Back and Load	BBL	1100DDDD	-	data
Load Immediate	LDM	1101DDDD	-	data
Write Main Memory	WRM	11100000	-	none
Write RAM Port	WMP	11100001	-	none
Write ROM Port	WRR	11100010	-	none
Write Status Char 0	WR0	11100100	-	none
Write Status Char 1	WR1	11100101	-	none
Write Status Char 2	WR2	11100110	-	none
Write Status Char 3	WR3	11100111	-	none
Subtract Main Memory	SBM	11101000	-	none
Read Main Memory	RDM	11101001	-	none
Read ROM Port	RDR	11101010	-	none
Add Main Memory	ADM	11101011	-	none
Read Status Char 0	RD0	11101100	-	none
Read Status Char 1	RD1	11101101	-	none
Read Status Char 2	RD2	11101110	-	none
Read Status Char 3	RD3	11101111	-	none
Clear Both	CLB	11110000	-	none
Clear Carry	CLC	11110001	-	none
Increment Accumulator	IAC	11110010	-	none
Complement Carry	CMC	11110011	-	none
Complement	CMA	11110100	-	none
Rotate Left	RAL	11110101	-	none
Rotate Right	RAR	11110110	-	none
Transfer Carry and Clear	TCC	11110111	-	none
Decrement Accumulator	DAC	11111000	-	none
Transfer Carry Subtract	TCS	11111001	-	none
Set Carry	STC	11111010	-	none
Decimal Adjust Accumulator	DAA	11111011	-	none
Keybord Process	KBP	11111100	-	none
Designate Command Line	DCL	11111101	-	none

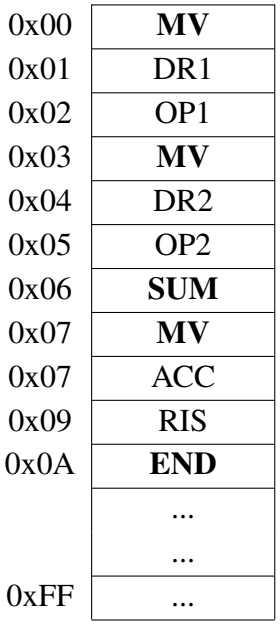
Figura 1.2: Il set di istruzioni relativo al microprocessore Intel 4004

Esecuzione delle istruzioni

Sulla base dell'architettura di Von Neumann esposta, a partire dall'instruction-set precedentemente definito, è possibile implementare un algoritmo specifico combinando le istruzioni con i dati che realizzano la funzione desiderata. In prima approssimazione si può pensare che dati e programmi relativi a tale algoritmo siano presenti all'interno della memoria del sistema in maniera sequenziale e sotto forma di linguaggio macchina.
Prendendo ad esempio un programma in grado di realizzare la somma tra due valori binari caricati all'interno di altrettanti registri (denominati DR1 e DR2) e il salvataggio del risultato su di un terzo registro (denominato RIS), si avrà il seguente codice assembler:

```
MV DR1, OP1      % copia il valore di DR1 in OP1
MV DR2, OP2      % salva il valore di DR2 in OP2
SUM    % somma OP1 e OP2 (risultato in ACC)
MV ACC, RIS    % salva il valore di ACC in RIS
```

in cui le etichette dei registri corrispondono ad un identificativo hardware (indirizzo) anch'esso definito all'interno dell'instruction-set. In memoria RAM saranno presenti quindi le istruzioni ed i dati derivanti da tale esempio di programma in maniera sequenziale e come riportato nella figura seguente.



L'esecuzione di un'istruzione avviene col ciclo di *fetch-execute*, mantenuto da parte dell'unità di processamento centrale CPU. Al suo interno sono infatti presenti tutti i sistemi relativi alla sequenziazione delle operazioni (unità di controllo CU), al calcolo aritmetico nonché registri di appoggio dedicati al salvataggio di dati temporanei ed inidirzzamento della memoria.

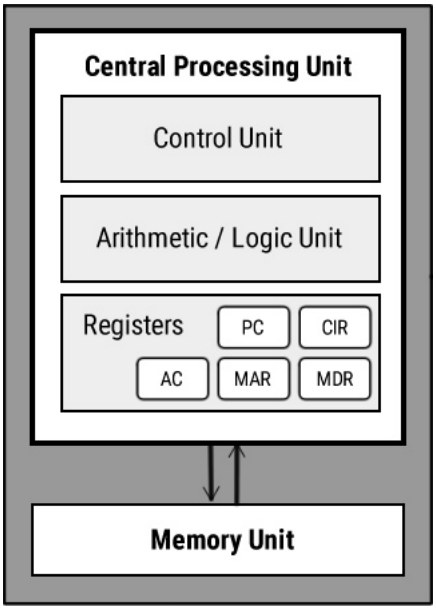


Figura 1.3: La CPU all'interno di un sistema alla Von Neumann

Nel caso pratico dell'esempio precedentemente riportato, all'avvio dell'esecuzione del programma la CPU darà il comando di *fetch* dell'istruzione presente nella locazione di memoria avente indirizzo 0x00.
Al riconoscimento dell'istruzione (*fase di decode*) il sistema diviene al corrente del fatto che ai successivi due prelievi dalla memoria corrisponderanno gli operandi dell'istruzione precedentemente caricata.
La fase di *execute* inizia con il prelievo dei dati dagli indirizzi 0x01 e 0x02. Avendo quindi caricato i dati relativi alla prima MOVE il sistema esegue l'istruzione che ha come risultato il salvataggio nel registro denominato OP1 del valore contenuto nel registro denominato DR1.
A seguire il sistema esegue a seconda istruzione (prelevata all'indirizzo 0x03 ripetendo quanto detto con la prima MOVE) che termina col salvataggio del dato presente in DR2 sul registro OP2.
Al prelievo dell'istruzione all'indirizzo 0x06, corrispondente alla SUM, il siste-

ma esegue la somma dei valori caricati sui registri OP1 ed OP2, precedentemente aggiornati ai valori desiderati. La somma termina col salvataggio del risultato sul registro accumulatore interno (denominato ACC). Infine, con l'istruzione 0x07 si esegue una ulteriore MOVE del dato salvato in ACC sul registro denominato RIS. Il programma termina con l'esecuzione dell'istruzione all'indirizzo 0x0A e denominata END, che comunica alla macchina il termine delle operazioni del programma.

L'esecuzione delle istruzioni in questo modello architetturale di memoria prevede il fatto che i dati ed i programmi siano caricati in maniera sequenziale nelle locazioni di memoria. L'indirizzamento della memoria è affidato ad un particolare registro denominato "Program-Counter", abbreviato con l'identificativo PC. Il valore mantenuto all'interno del PC costituisce quindi l'indirizzo della istruzione in esecuzione o del dato relativo all'istruzione che lo necessita. All'avvenuto prelievo dell'istruzione o del dato corrente (o comunque prima del prelievo della successiva) il PC viene incrementato di 1 puntando di fatto alla successiva locazione di memoria che conterrà un dato o una nuova istruzione.
Inoltre poiché si è supposto che l'avvio dell'esecuzione del programma parta dal fetch dell'istruzione all'indirizzo 0x00, allora questa locazione deve necessariamente contenere la prima istruzione che costituisce l'inizio dell'algoritmo.

Quanto detto costituisce un limite prestazionale all'impiego dell'architettura di Von Neumann in quanto la presenza di un unico bus per dati ed istruzioni (e segnali di controllo) limita fortemente le prestazioni di throughput dei dati da parte del sistema. Unitamente a questo limite prestazionale derivante dall'architettura stessa, la gestione della memoria come descritta, ossia contenente i dati relativi alle istruzioni nelle locazioni di memoria strettamente successive alle istruzioni stesse, non permette la realizzazione di tecniche di parallelizzazione volte all'incremento delle prestazioni del sistema.
Per tale motivo nel seguente paragrafo verrà descritto un sistema di gestione della memoria più efficiente e compatibile con le tecniche di parallelizzazione.

Von Neumann migliorato

Una soluzione al problema descritto nelle ultime righe del precedente paragrafo prevede la modifica della gestione della memoria nel salvataggio di dati e istruzioni. Questa può essere concettualmente divisa in due aree, una dove vengono salvate in

sequenza le istruzioni da eseguire, l'altra che contiene i dati relativi a tali istruzioni, come schematizzato nella figura successiva.

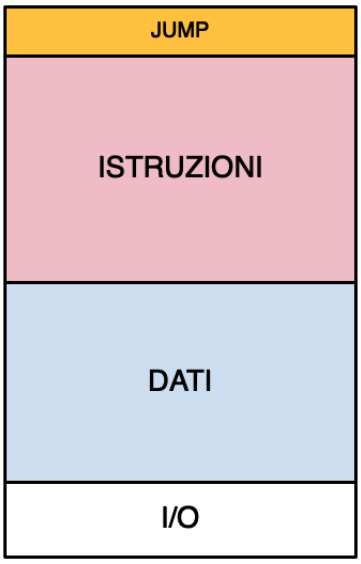


Figura 1.4: Gesione della memoria in un'architettura di Von Neumann migliorata

In tale architettura la gestione della memoria prevede, oltre al classico puntatore all'istruzione denominato program-counter, un puntatore al dato denominato data-counter, abbreviato DC. Tale puntatore contiene l'indirizzo da fornire alla memoria per il prelievo del dato relativo alla istruzione che lo necessita. In questo modo si riescono a garantire due importanti prerogative:

- Nel caso in cui non si debba effettuare un salto l'indirizzo relativo alla prossima istruzione è sempre dato dall'indirizzo corrente di PC sommato ad 1.
- Nel caso in cui si debba effettuare un salto il valore che punta alla prossima istruzione può già essere puntato da DC e non deve essere ricavato eseguendo un ulteriore aggiornamento di PC per il prelievo di tale dato. In questo modo (nel caso di chiamata a subroutine) il valore di PC può essere salvato come indirizzo di ritorno prima di aggiornare il PC all'indirizzo prelevato dalla memoria e al quale eseguire il salto.

L'indirizzamento della memoria si ottiene supponendo delle dimensioni fisse per le aree riservati a istruzioni e dati.

Una qualsiasi istruzione presente nell'area dedicata è indirizzabile tramite due coordinate. Una prima fissa, denominata *base istruzione* ed una seconda che indica *l'offset* rispetto a tale base. L'indirizzamento dunque avviene per mezzo del calcolo della seguente somma:

$$ADDR_P = P_0 + \Delta P \tag{1.1}$$

Analogamente, in riferimento al calcolo della posizione di un dato si avrà:

$$ADDR_D = D_0 + \Delta D \tag{1.2}$$

Metodi di indirizzamento

In base a quanto detto fino ad ora si può procedere alla definizione dell'argomento di questa tesina. Scopo del problema è la progettazione di una UC per un'architettura di Von Neumann curando l'implementazione di parte del set di istruzioni relativo all'indirizzamento della memoria RAM interna, in cui sia prevista la gestione migliorata della memoria precedentemente discussa.
In particolare è richiesta l'implementazione delle seguenti istruzioni:

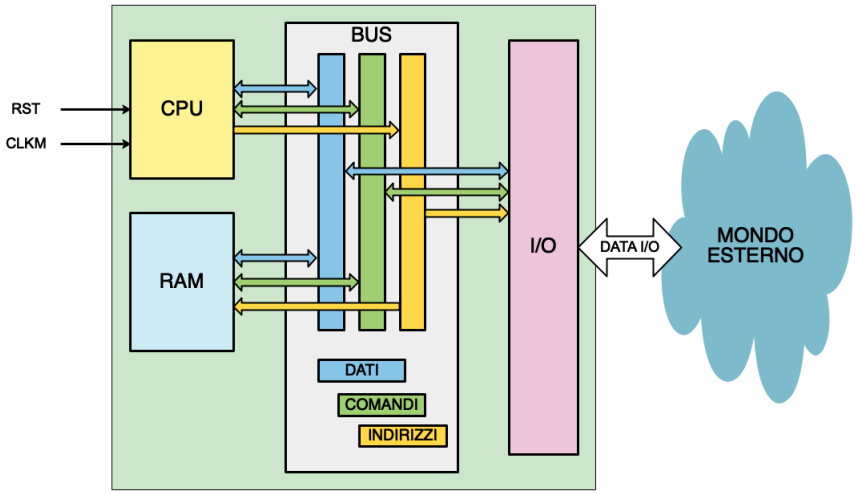
- **RD ADDR**: Esegue una READ della memoria all'indirizzo fornito attraverso il dato ADDR. Il valore letto è salvato su di un registro interno.
- **WR DATA, ADDR**: Esegue la scrittura del valore identificato con DATA nella locuzione di memoria RAM relativa all'indirizzo ADDR. È un'istruzione dal formato "un'istruzione e due dati".
- **JMP ADDR**: Esegue una JUMP incondizionale all'indirizzo presente nel dato ADDR. Tale istruzione realizza un "*salto immediato*". Il formato dell'istruzione è del tipo "una istruzione e un dato".
- **JMP OFFS**: Esegue una JUMP incondizionale all'indirizzo dato dal valore corrente di PC sommato al dato in ingresso denominato OFFS. Costituisce un'istruzione denominata "*salto diretto*" e prende in ingresso un unico dato.
- **JMP BASE, OFFS**: Esegue una JUMP incondizionale all'indirizzo calcolato tramite somma dei due dati in ingresso BASE e OFFS. Costituisce un tipo di salto denominato "*salto indiretto*" e possiede un formato del tipo "una istruzione e due dati".

Inoltre, dotando la CPU di un insieme di registri interni di tipo “general-purpose” in appoggio al sistema di calcolo, si implementerà la seguente istruzione:

- **MV REG1, REG2**: Esegue la copia del dato contenuto nel registro avente etichetta REG1 sul registro avente etichetta REG2. Tale istruzione si applica sui registri interni alla CPU.

Istanziazione zero

Dunque, in base a quanto esposto precedentemente, si parte con un sistema alla Von Neumann basilare come di seguito nuovamente riportato. All’avvio delle operazioni si suppone che la memoria RAM sia già precaricata con le istruzioni ed i dati nelle aree di memoria dedicate.



Le comunicazioni con l’esterno avvengono per mezzo del bus di I/O ad N bit. Si partirà nello svolgimento del problema tipo “carta e penna” con un valore per le dimensioni del bus dati pari a $N = 4$. La dimensione del bus indirizzi si sceglie pari a $M = 2 \cdot N$ bit che definisce uno spazio di indirizzamento della memoria pari a $2^8 = 256$ parole a N bit. In seguito, su ambiente di sviluppo ISE, si modificherà la dimensione del bus dati (e di conseguenza quella del bus indirizzi) per un valore di N = 8 bit. Dunque i segnali presenti a questo livello di definizione del problema sono i seguenti:

- **OPERANDI E RISULTATO:**
  - **DATA_IO** (N bit): bus di input-output di collegamento con l’esterno.

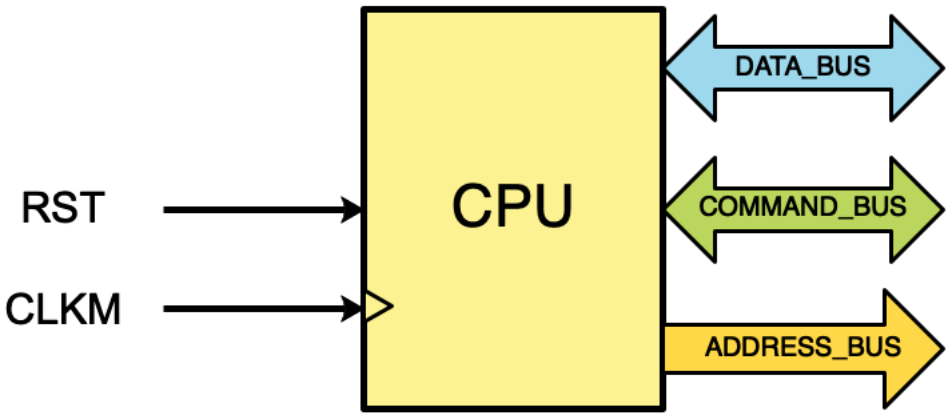
- **CLOCK MACCHINA:**
  - **CLKM**: clock macchina per evoluzione sincrona.
- **RESET:**
  - **RST**: {0: funzionamento normale; 1: reset attivo $\rightarrow$ il sistema riavvia l'esecuzione dalla prima istruzione}.

Capitolo 2

Prima istanziazione

2.1 Schema a blocchi e definizione dei segnali IN/OUT

In base a quanto detto nel precedente capitolo si può procedere ad una prima istanziazione della CPU.



I segnali presenti a questo livello sono:

- **OPERANDI E RISULTATO:**
  - **DATA_BUS** (N bit): bus dati di input-output connesso con il bus DATI di sistema.
  - **COMMAND_BUS** (N bit): bus contenente un raggruppamento dei segnali di controllo connesso con il bus COMANDI di sistema.
  - **ADDRESS_BUS** (2N bit): bus contenente i segnali di indirizzo per la memoria e le periferiche, connesso con il bus INDIRIZZI di sistema.

- **CLOCK MACCHINA:**
  - **CLKM**: clock macchina per evoluzione sincrona.
- **RESET:**
  - **RST**: {0: funzionamento normale; 1: reset attivo $\rightarrow$ il sistema riavvia l'esecuzione dalla prima istruzione}.

2.2 Regola protocollare

- *ESTERNO*:
  1. La macchina procede con l'esecuzione del programma salvato nella memoria RAM in modo sequenziale evolvendo con gli impulsi di clock esterni.
- *INTERNO*: La CPU si trova nella condizione di reset e genera l'indirizzo 0000h per la memoria. Tale locazione conterrà la prima istruzione del programma scritto in RAM. Per ogni istruzione si esegue il ciclo di fetch-execute:
  1. FETCH: La CPU preleva l'istruzione dalla memoria all'indirizzo fornito dal PC e la salva nel registro istruzione IR. Infine esegue un aggiornamento del PC.
  2. DECODE: La CPU riconosce l'istruzione e si occupa del prelievo eventuale dei dati dalla memoria salvandoli nei registri di appoggio interno.
  3. EXECUTE: La CPU esegue l'istruzione.
  4. (se previsto) WRITE-BACK: La CPU si occupa del salvataggio del dato sulla memoria.

2.3 Task eseguibili, set di istruzioni e organizzazione della RAM interna

In questa sezione ci si occupa di definire parte del set di istruzioni relativo a quanto discusso nel paragrafo "Metodi di indirizzamento" di pagina 10. Riassumendo i task da eseguire e considerando i dati relativi a tali istruzioni si ottiene la seguente tabella.

INSTRUCTION	CODE	BINARY EQUIVALENT			MODIFIERS
		COD. OP	1st nibble	2nd nibble	
No operation	NOP	0000	-	-	-
Read RAM	RD	0001	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$	address
Write RAM at address	WR	0011	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$	address, data
Jump at Address	JPA	0100	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$	address
Jump Offset	JPO	0101	$O_3O_2O_1O_0$	-	offset
Jump at Base+Offset	JBO	0111	$O_3O_2O_1O_0$	$B_3B_2B_1B_0$	offset, base
Move reg A to B	MV	1000	$A_3A_2A_1A_0$	$B_3B_2B_1B_0$	reg. A, reg. B

Tabella 2.1: Set di istruzioni

Il numero di bit riservato al codice operazione è sovradimensionato rispetto al numero di operazioni. Per discriminare tra le operazioni possibili sarebbero stati sufficienti $\lceil log_2 7 \rceil = 3$ bit; tuttavia considerando la futura ed eventuale aggiunta di ulteriori istruzioni, nonchè il fatto di avere un bus dati a 4 bit, si è scelto di mantenere un codice operazione a 4 bit.
Come si può notare dalla tabella precedente sistema lavora su istruzioni operanti su 0, 1, o 2operandi a 4 bit. Sarà dunque compito della UC interna, una volta prelevata l'istruzione ed eseguita la decodifica, richiedere il numero di dati necessari all'esecuzione dell'istruzione stessa.

Si ricorda che la memoria è organizzata come in Figura 1.4, dunque le istruzioni del programma troveranno posto in modo sequenziale nell'area "istruzioni", così come i dati nell'area "dati" sottostante. Poiché si dispone di un bus dati a 4 bit, nelle istruzioni dove si debba prelevare un indirizzo dalla memoria, si necessita di più accessi a locazioni di memoria contigue presenti all'area dati. Si suppone dunque che la fase di traduzione da codice assembler a linguaggio macchina tenga conto di tale questione, procedendo ad inserire i dati nell'area di memoria dedicata ordinati in maniera sequenziale.

2.4 Strategia e problematiche

La struttura interna della CPU sarà ricavata in questo paragrafo tenendo conto del set di istruzioni e delle prerogative di progetto.
Anzitutto poiché dati e programmi sono reperiti a runtime dalla memoria RAM interna sarà necessario dotare la CPU di sistemi che permettano l'indirizzamento della memoria ai programmi ed al dato. In particolare si necessiterà quindi di due puntatori: un "program counter" ed un "data counter". Il PC conterrà l'indirizzo relativo alla prossima istruzione da eseguire, mentre il DC contiene l'indirizzo relativo al dato da prelevare. In generale si può considerare che l'indirizzo per la lettura/scrittura della memoria può avere tre sorgenti:

- il PC: come nel caso del prelievo delle istruzioni.
- il DC: come nel caso del prelievo/scrittura dei dati.
- un indirizzo generato internamente dalla CPU, ad esempio a seguito di un calcolo.

A tal proposito, nel procedere al corretto indirizzamento della memoria, l'uscita del bus indirizzi sarà pilotata da un registro al cui ingresso è presente un multiplexer controllato dalla UC interna col fine di discriminare quale dei tre registri sorgenti contenga il valore col quale indirizzare la memoria.
Entrambi i registri PC e DC, devono essere aggiornati (incrementati di 1) ad ogni prelievo relativamente di un'istruzione o di un dato. A tal proposito si doteranno entrambi i registri della possibilità di auto-incremento affiancando agli stessi un sommatore semplice.
Per dotare la CPU della possibilità di indirizzare la memoria da programma (come nel caso della RD e della WR) si può pensare di connettere uno dei registri interni presenti nel banco d'appoggio al mux di selezione. In tale caso l'indirizzamento della memoria sarà possibile senza compromettere i valori presenti in PC e DC. Tale registro avrà la funzione speciale di poter indirizzare la memoria e la sua etichetta sarà definita AR0.

Nel caso dell'operazione di RD l'indirizzo di lettura si suppone già presente nel registro AR0. Successivamente si effettua una lettura a tale indirizzo salvando il dato proveniente dalla RAM sul registro bidirezionale in/out. Nel caso dell'operazione di WR si suppone che il dato da salvare sia già stato precaricato all'interno del

registro bidirezionale in/out che pilota il bus dati dal lato CPU. Come nel caso della read, l'indirizzo su cui salvare il dato sarà stato precedentemente posto in AR0. L'operazione termina abilitando la memoria al salvataggio del dato su tale indirizzo.

Nel caso della jump immediata l'indirizzo di salto è indicato dal valore del dato prelevato dalla memoria. Il valore su cui effettuare il salto viene dapprima composto unendo le due letture della memoria che forniscono le due sottoparti dell'indirizzo nel registro AR0. Successivamente il PC verrà aggiornato al valore contenuto in AR0, indirizzato su di esso tramite multiplexer.

Nel caso delle operazioni denominate JPO e JBO il risultato del salto è dato dalla somma di un indirizzo "*base*" e di un *offset* riferito a tale base. A seguito di una o due letture necessarie al caricamento di offset e base ed al caricamento di queste in due particolari registri di appoggio denominati DRO e DRB il valore di PC viene aggiornato a quello corretto eseguendo il calcolo del nuovo indirizzo su di una unità di calcolo ausiliare. In questo modo non si necessiterà dell'impiego della ALU principale nel caso di calcolo di tali indirizzi di salto. Il collegamento elettrico che permette l'incremento del PC al valore corretto è presentato nel successivo schema di seconda istanziazione.

Infine, considerando la struttura elementare interna di un'architettura di Von Neumann come quella riportata in Figura 1.3, la CPU contiene al suo interno un banco di registri di appoggio al calcolo. In particolare, nel caso relativo a questo esercizio, oltre ai già noti PC e DC trovano posto:

- 10 registri di appoggio per i dati con etichetta.
- 1 registro accumulatore ACC.
- 4 registri di supporto al calcolo di ADDR, tra cui AR0.
- 1 registro bidirezionale per il salvataggio dei dati in ingresso/uscita, connesso al bus dati di sistema.

Su tutti questi registri si applica l'operazione di move parametrica (COD. OP: MV), la quale eseguirà la copia del contenuto di uno dei registri sorgente in un altro di destinazione, ad eccezione del registro ACC che fungerà solo da destinazione e non da sorgente. Per l'esecuzione di tale operazione si può pensare di inserire all'interno dell'area dedicata ai registri una logica combinatoria che apra il "canale di

transito" al dato dal registro sorgente a quello destinazione. Uno strobe da parte della UC master sul registro destinazione ne copierà il dato all'interno completando l'operazione. Per discriminare i registri coinvolgibili in tale operazione si doterà ognuno di questi di una etichetta caratteristica, ossia definita a livello di microcodice. Al programmatore sarà sufficiente scrivere i tag dei registi opportuni nella programmazione a livello assembler per garantire la riuscita dell'operazione in modo corretto.

Detto questo si può procedere alla seconda istanziazione della CPU.

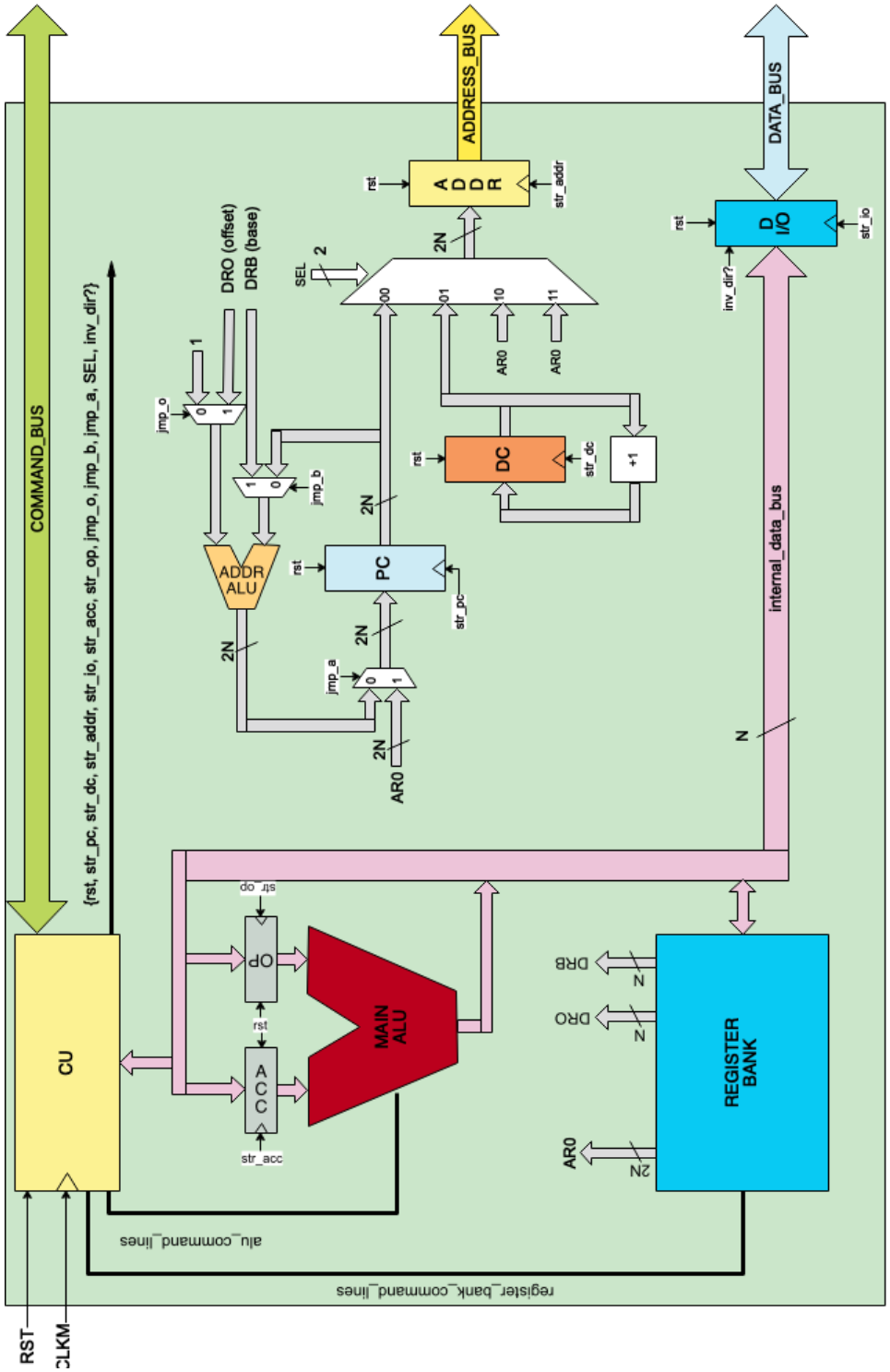
Capitolo 3

Seconda istanziazione

3.1 Schema a blocchi e definizione dei segnali IN/OUT

A partire da quanto esposto nel precedente capitolo segue lo schema di seconda istanziazione della CPU.

3.1.1 Schema a blocchi



3.1.2 Segnali interni

I segnali interni presenti a questo livello sono i seguenti.

- *rst*: {0: funzionamento normale; 1: reset prioritario per i registri interni}.
- *str_pc*: strobe per il registro PC.
- *str_dc*: strobe per il registro DC.
- *str_addr*: strobe per il registro address.
- *str_io*: strobe per il registro di I/O.
- *str_acc*: strobe per il registro accumulatore.
- *str_op*: strobe per il registro operando.
- *jmp_o*: segnale di comando per il mux. {0: uscita = 1, 1: uscita = *DRO*}.
- *jmp_b*: segnale di comando per il mux. {0: uscita = PC, 1: uscita = *DRB*}.
- *jmp_a*: segnale di comando per il mux. {0: uscita = *ALU_ADDR_OUT*, 1: uscita = *AR0*}.
- ***SEL***: 2 bit. Segnale di comando per il mux selezione indirizzo. Vedi 3.1.
- *inv_dir?*: {0: registro I/O in funzionamento IN $\rightarrow$ OUT; 1: registro I/O in funzionamento OUT $\rightarrow$ IN}.
- ***alu_command_lines***: linee di controllo per la ALU principale.
- ***register_bank_command_lines***: linee di controllo per il register bank.
- ***internal_data_lines***: Bus dati interno a N bit.

3.2 Ulteriori considerazioni di seconda istanziazione

Per ciò che concerne l'indirizzamento della memoria si può notare che il registro indirizzo è pilotato da un mux 4-1, controllato tramite ***SEL***. ***SEL*** è un segnale a 2 bit proveniente dalla CU ed il controllo del mux è schematizzato nella seguente tabella:

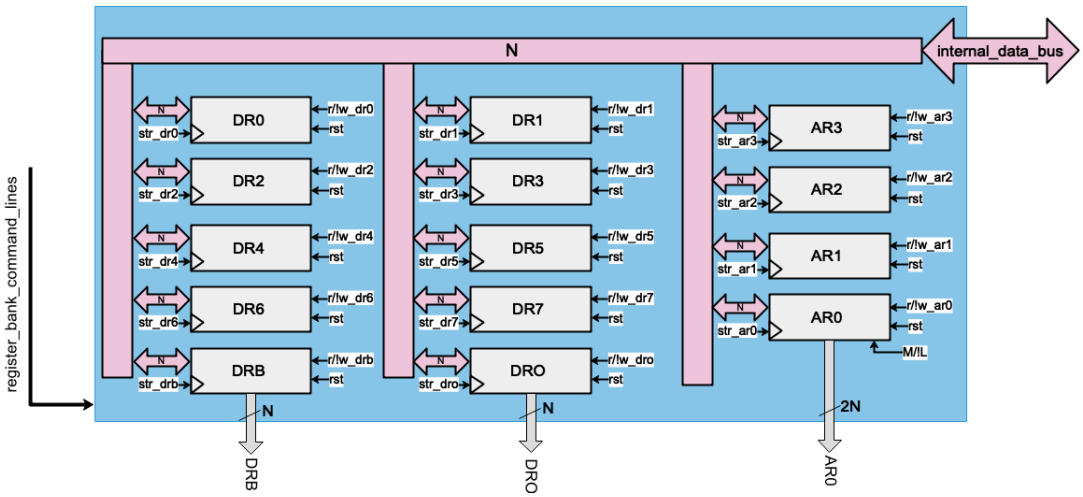
SEL_1	SEL_0	SORGENTE ADDR
0	0	**PC**
0	1	**DC**
1	0	**AR0**
1	1	**AR0**

Tabella 3.1: Indirizzamento della memoria

Questo perché, come detto, è necessario avere tre possibili sorgenti per l'indirizzamento della memoria. Il valore di indirizzamento proviene da PC, DC oppure da AR0, nei casi in cui SEL_1 = 1. AR0 è uno dei tre registri presenti nel banco d'appoggio con "funzione speciale", ossia connesso ad un bus esterno accessibile in modo prioritario, proprio per avere la possibilità di realizzare funzioni come l'indirizzamento della memoria o il caricamento del valore del salto nel caso della jump.

3.3 Dimensionamento interno dei blocchi di seconda istanziazione

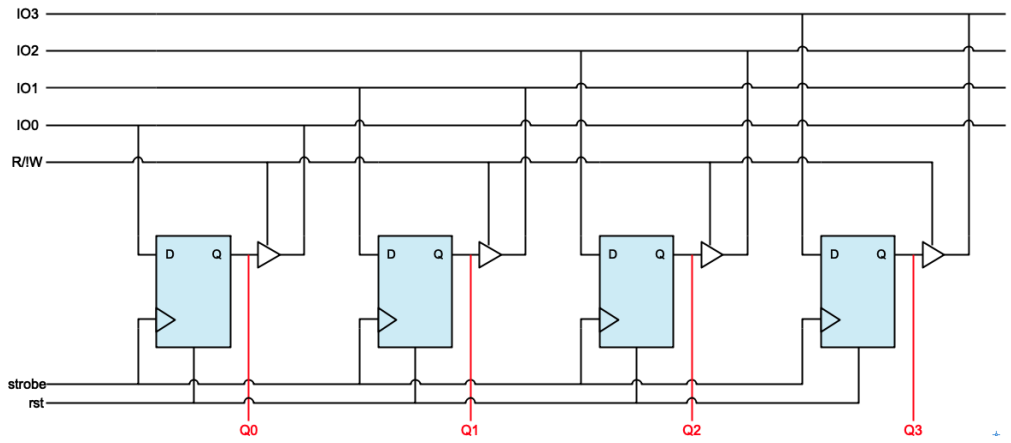
3.3.1 Register bank



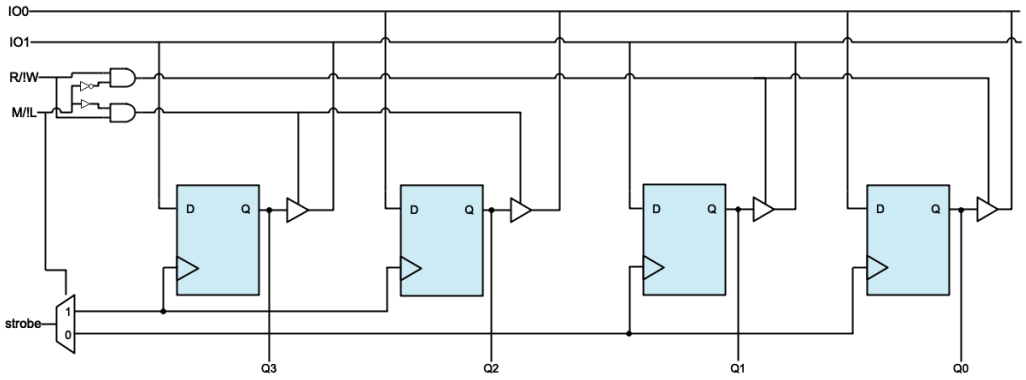
Il register bank contiene 14 registri di supporto al calcolo, di cui 4 riservati al calcolo degli indirizzi. Tra di essi vi sono tre registri accessibili esternamente attraverso altrettanti bus dedicati. Tali registri sono:

- DRB: nel caso dell'operazione JBO in questo registro è presente il valore relativo alla base da sommare all'offset per il calcolo dell'indirizzo di salto.
- DRO: nel caso delle operazioni di JPO o JBO in questo registro è presente il valore relativo all'offset da sommare al PC o alla base per il calcolo dell'indirizzo di salto.
- AR0: in questo registo è presente il valore da dare al PC nel caso di jump immediata, oppure il valore da fornire alla memoria nel caso di RD o WR. Questo registro ha dimensione 2N bit per garantire l'indirizzamento a tutta la memoria, tuttavia è collegato attraverso il bus dati interno solo agli ultimi N bit. Attraverso il comando denominato $M/\overline{L}$ indirizzare il bus dati in scrittura ai primi N flip-flop o agli ultimi N flip-flop che compongono tale registro, per garantire il caricamento su tutto lo spazio dei 2N bit tramite bus a N bit. Di seguito sarà presentato lo schema interno-

Ogni registro componente il register bank è pilotato singolarmente attraverso una terna di segnali: $\{$*str_xxx*, *rst*, *r/*$\overline{w}$*_xxx*$\}$. Nel caso del registro AR0 è presente anche il segnale $M/\overline{L}$ il cui impiego è necessario per avere la possibilità di scrivere o leggere separatemente i LSH (Least Significant Heap) o gli MSH (Most Significant Heap), ossia gli ultimi/primi N bit del registro. Dettagli del funzionamento del registro AR0 saranno esposti in seguito.
L'insieme dei segnali di controllo è stato schematizzato nella figura con l'insieme di linee denominato ***register_bank_command_lines***.
Ogni registro che compone il register bank è inoltre di tipo bidirezionale a singolo bus, per avere la possibilità di essere letto o scritto attraverso il bus dati interno. Lo schema interno del generico registro, supposto N=4 bit, è rappresentato nella figura seguente.



In questo schema il comando $R/\overline{W}$ definisce se il registro è abilitato alla lettura (1) o alla scrittura (0), in modo che il bus possa essere bidirezionale. Quando un registro non è utilizzato deve essere mantenuto in modalità scrittura ($R/\overline{W} = 0$) per far si che i tri-state d'uscita siano disabilitati in modo da non interferire col controllo del bus da parte di altre entità ad esso collegate.
In rosso invece sono rappresentate le linee d'uscita disponibili solamente per i registri con accesso speciale DRO e DRB, che costituiscono i bus dedicati discussi in precedenza.
Lo schema interno del registro AR0 (rappresentato per N=2 bit) è invece il seguente. Si ricorda che la dimensione di tale registro è pari a 2N bit, dunque:



Si può notare il demultiplexer interno che permette lo smistamento del segnale di strobe indipendentemente al primo o al secondo gruppo di flip-flop e la rete logica che pilota i tri-state che realizza la seguente funzione.

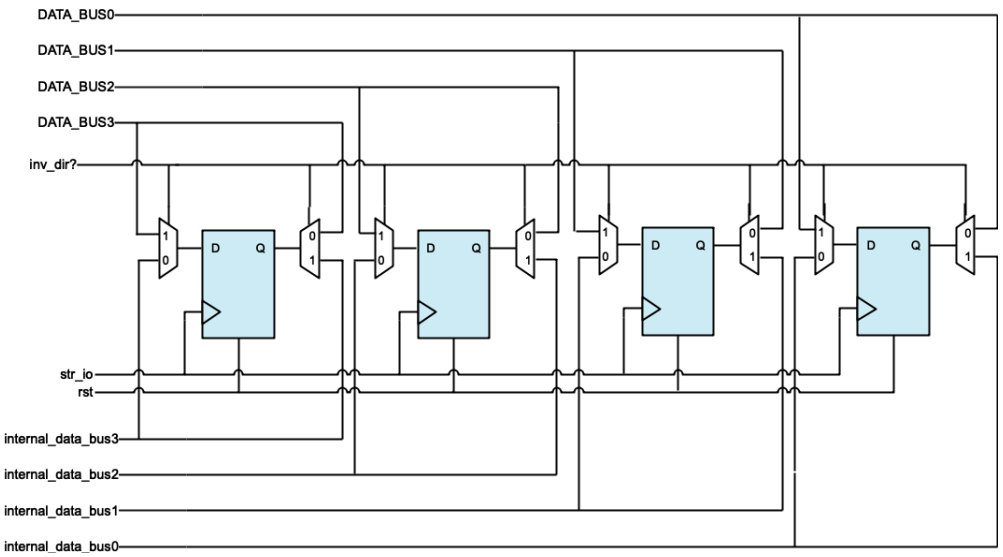
$R/\overline{W}$	$M/\overline{L}$	**modalità**
0	0	scrittura LSH
0	1	scrittura MSH
1	0	lettura LSH
1	1	lettura MSH

Tabella 3.2: Modalità di funzionamento registro AR0

Omesso dal disegno, ma presente, il clear prioritario dei flip-flop connesso al segnale esterno di *reset*.

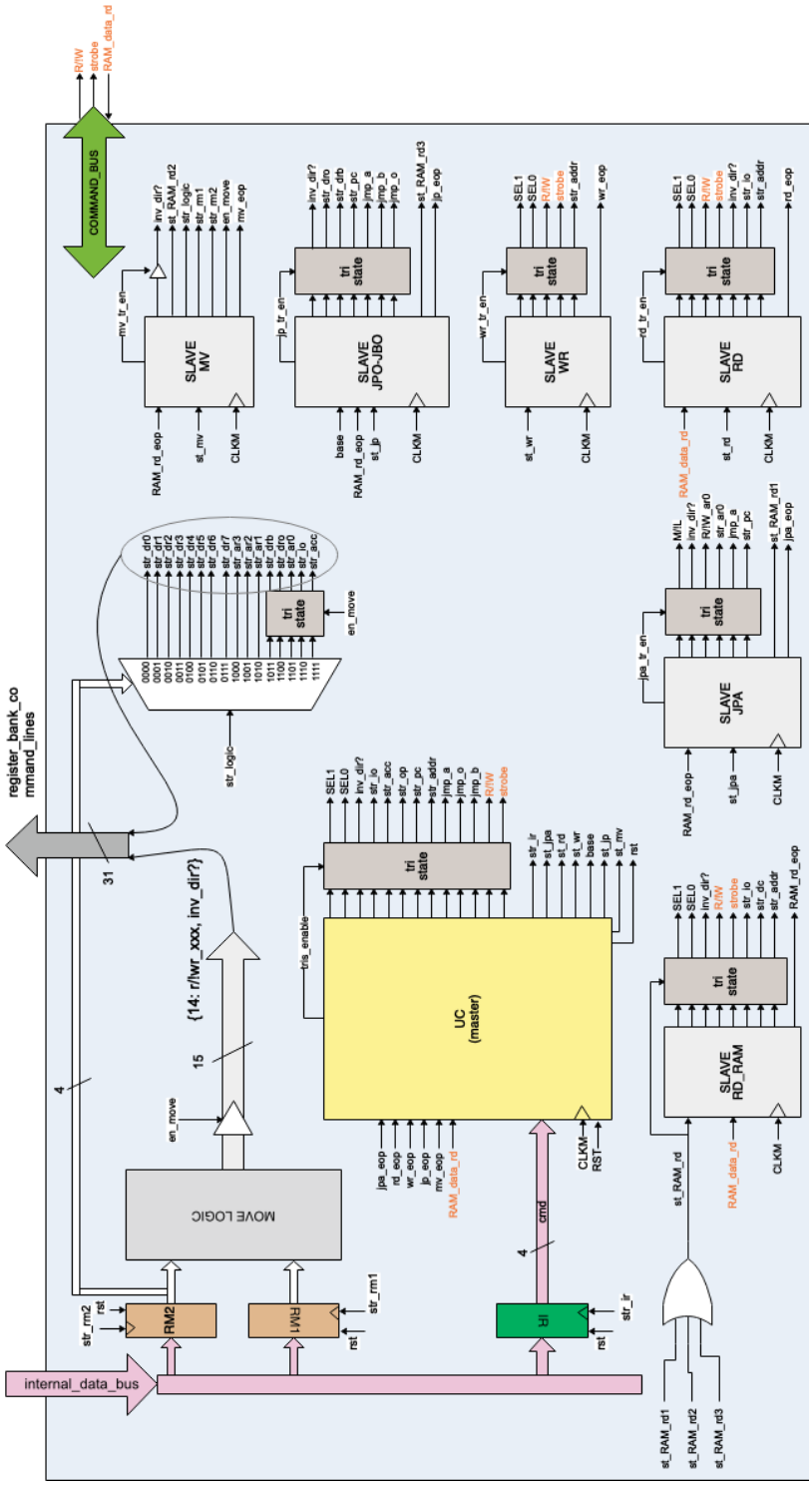
3.3.2 Registro IN-OUT

Un'importante funzione è delegata al registro bidirezionale D_I/O, ossia quella di pilotare il bus dati in uscita e fungere da pozzo per i dati in ingresso. Il registro deve dunque essere di tipo bidirezionale. Lo schema interno è il seguente.



Il registro non è dotato di buffer tri-state in ingresso/uscita, dunque sarà necessario fare in modo che questo sia sempre in modalità *inv_dir? = 0* quando non è in utilizzo.

3.3.3 Unità di controllo: CU



3.3.3.1 Segnali interni

I segnali interni che trovano posto all'interno dello schema di terza istanziazione della UC sono i seguenti:

- *rst*: {0: funzionamento normale; 1: reset prioritario per tutti i registri}.
- *str_ir*: strobe per il registro istruzioni IR.
- ***cmd***: uscita a 4 bit dell'IR.
- *str_rm1*: strobe per il registro di supporto alla move RM1.
- *str_rm2*: strobe per il registro di supporto alla move RM2.
- *st_nomeslave*: segnale di start per lo slave *nomeslave*. Delega il comando da parte della UC master allo slave *nomeslave* per l'esecuzione del proprio task.
- *nomeslave_eop*: se 1 indica che lo slave *nomeslave* ha terminato l'esecuzione delle proprie operazioni.
- *base*: {1: indica allo slave JPO/JBO se è richiesta una somma con base (JBO) prelevata dalla memoria; 0: somma JPO semplice}
- *nomeslave_tr_en*: lo slave *nomeslave* abilita il proprio banco di buffer tri-state per pilotare le linee condivise.
- *en_move*: abilitazione dei tri-state per la logica di move.
- *str_logic*: strobe per il demux di move, esegue la copia del dato sul registro destinazione.
- *str_nomeregistro*: strobe per il registro *nomeregistro*, appartenente al register bank.
- *st_RAM_rdX*: segnale per lo start dello slave RD_RAM. Vi sono 3 segnali provenienti da altrettanti slave che all'occorrenza abilitano RD_RAM. L'abilitazione avviene tramite operazione di OR su questi tre segnali.

Inoltre, connessi al BUS COMANDI si trovano i segnali di controllo per la RAM:

- $R/\overline{W}$: {0: abilitata scrittura della memoria RAM interna alla CPU; 1: RAM abilitata alla lettura.}.

- *strobe*: strobe per la memoria RAM.
- *RAM_data_rd*: {1: la RAM ha terminato la lettura o e il dato è disponibile sul BUS DATI di sistema; 0: la RAM è in stato *busy*}.

3.3.3.2 Logica di Move

Per effettuare l'operazione di move, come detto, si procede dapprima preparando i segnali che permettano lo spostamento del dato da un registro all'altro. Per fare ciò si sintetizza una logica che, a partire dalle etichette contenute in due registri di appoggio scritti a runtime, piloti i segnali da applicare ai registri sorgente e destinazione per permettere la copia del dato. Il principio alla base della move è il fatto che tutti i registri condividono sia in lettura sia in scrittura il bus dati interno a N bit. Il bus può essere pilotato da uno dei registri coinvolti nella move (escluso ACC) attraverso l'abilitazione del segnale $r/\overline{w}\_x$ sul registro x. Si possono ora definire le etichette dei registri coinvolti nella move, come riportato nella seguente tabella.

Registro	*Etichetta*	*Registro*	*Etichetta*
DR0	0000	**DRB**	1000
DR1	0001	**DRO**	1001
DR2	0010	**AR0**	1010
DR3	0011	**AR1**	1011
DR4	0100	**AR2**	1100
DR5	0101	**AR3**	1101
DR6	0110	**ACC**	1110
DR7	0111	**D_I/O**	1111

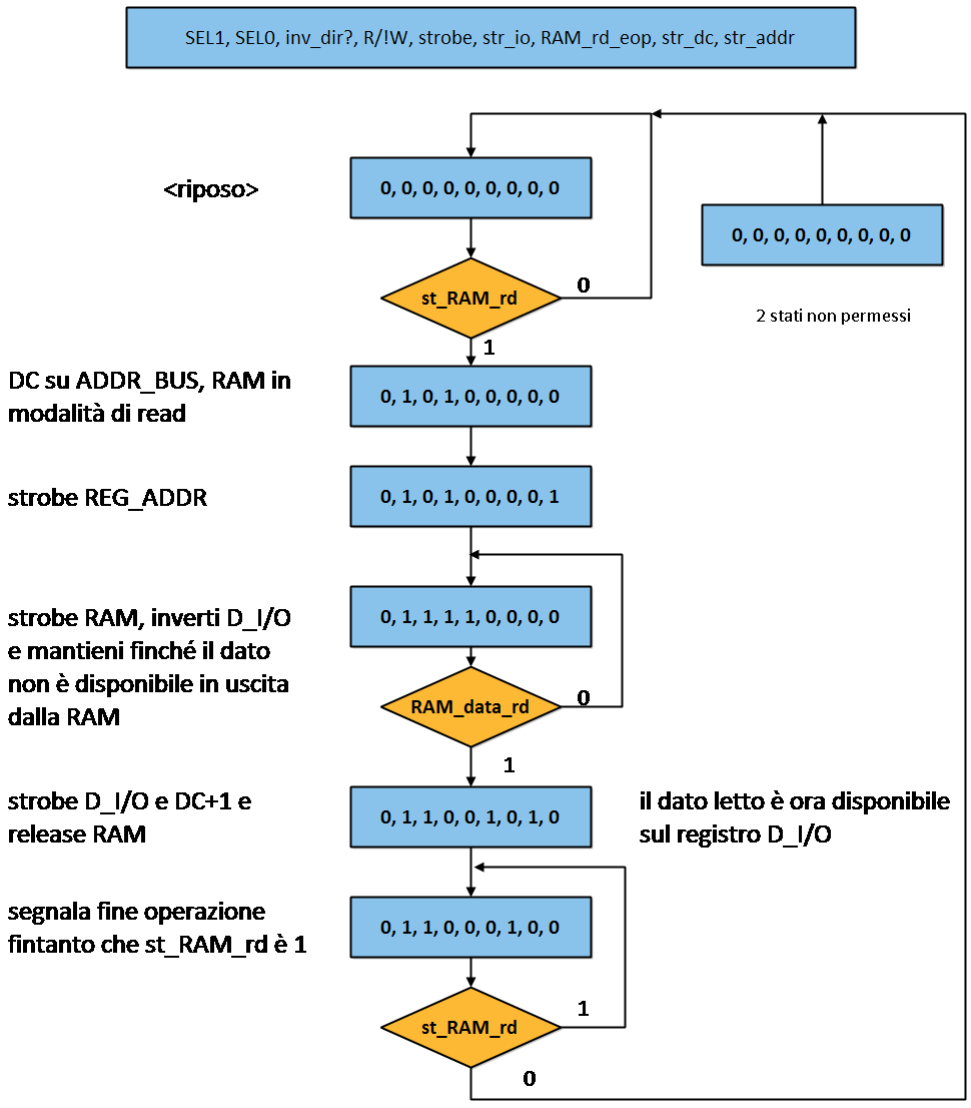
Tabella 3.3: Etichette dei registri abilitati all'operazione di move

Tali etichette possono essere definite a livello di Instruction Set, permettendo all'utente la scrittura dei registri utilizzando direttamente il nominativo nella stesura del codice assembler; es: *MV DR3, ACC*.
La logica che pilota i segnali è di tipo combinatorio e verifica la seguente tabella di verità.

Passiamo ora a definire i diagrammi ASM per le MSF slave interne alla UC.

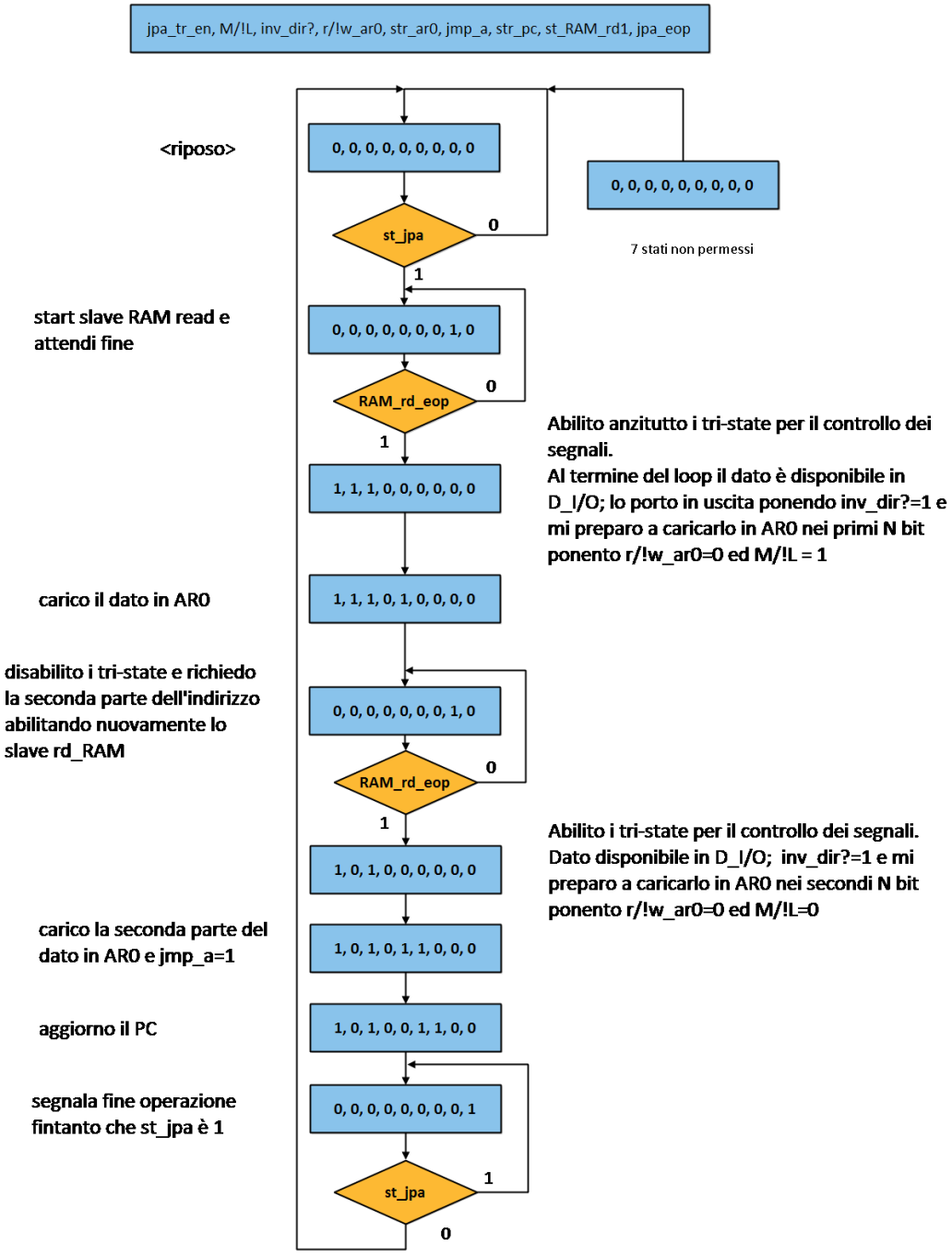
3.3.3.3 Slave RD_RAM

Esegue la lettura della memoria di un dato indirizzando la RAM tramite il DC. Il diagramma ASM è il seguente.



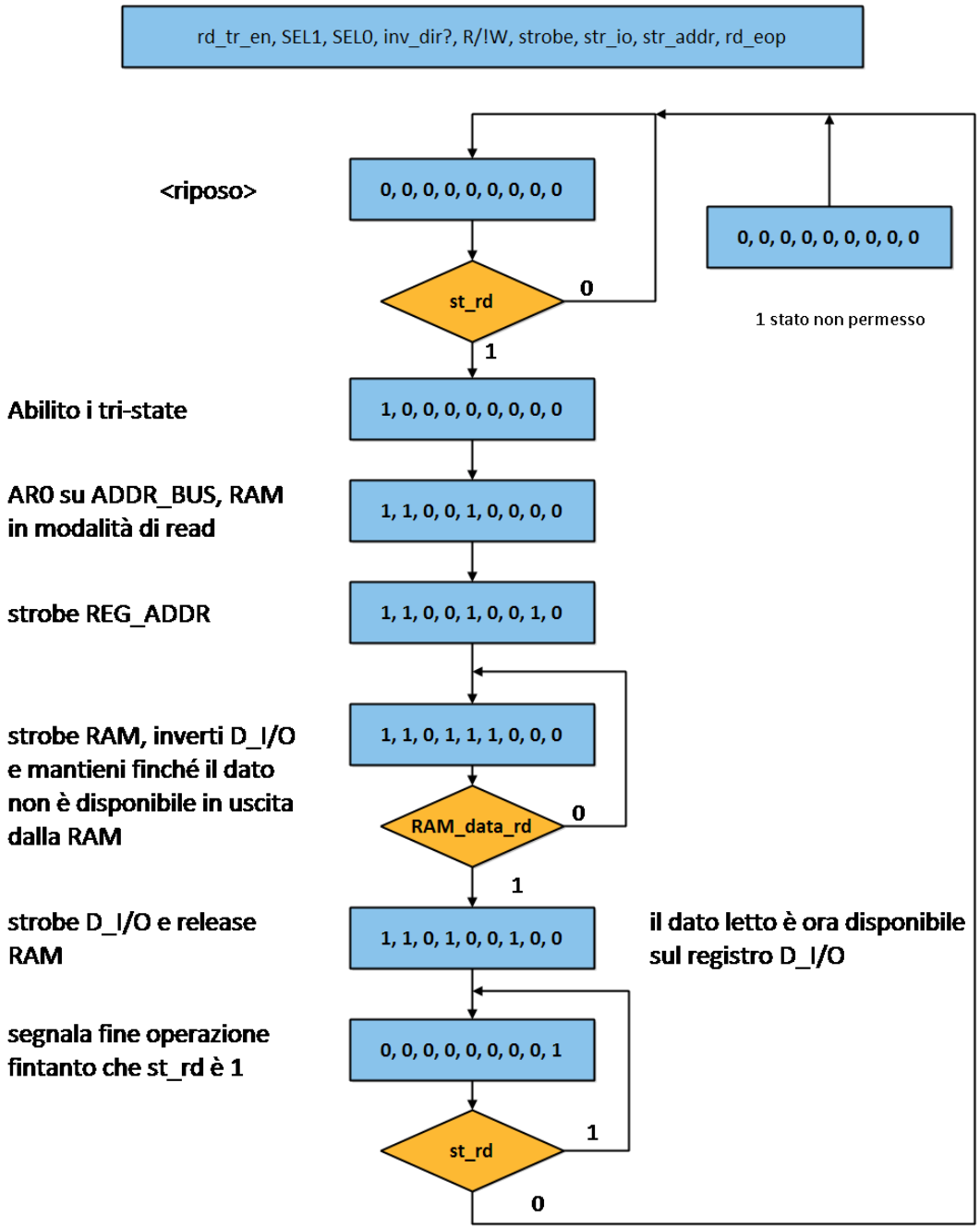
3.3.3.4 Slave JPA

Esegue la jump immediata, aggiornando il Program Counter all'indirizzo di salto ricavato dalla memoria.



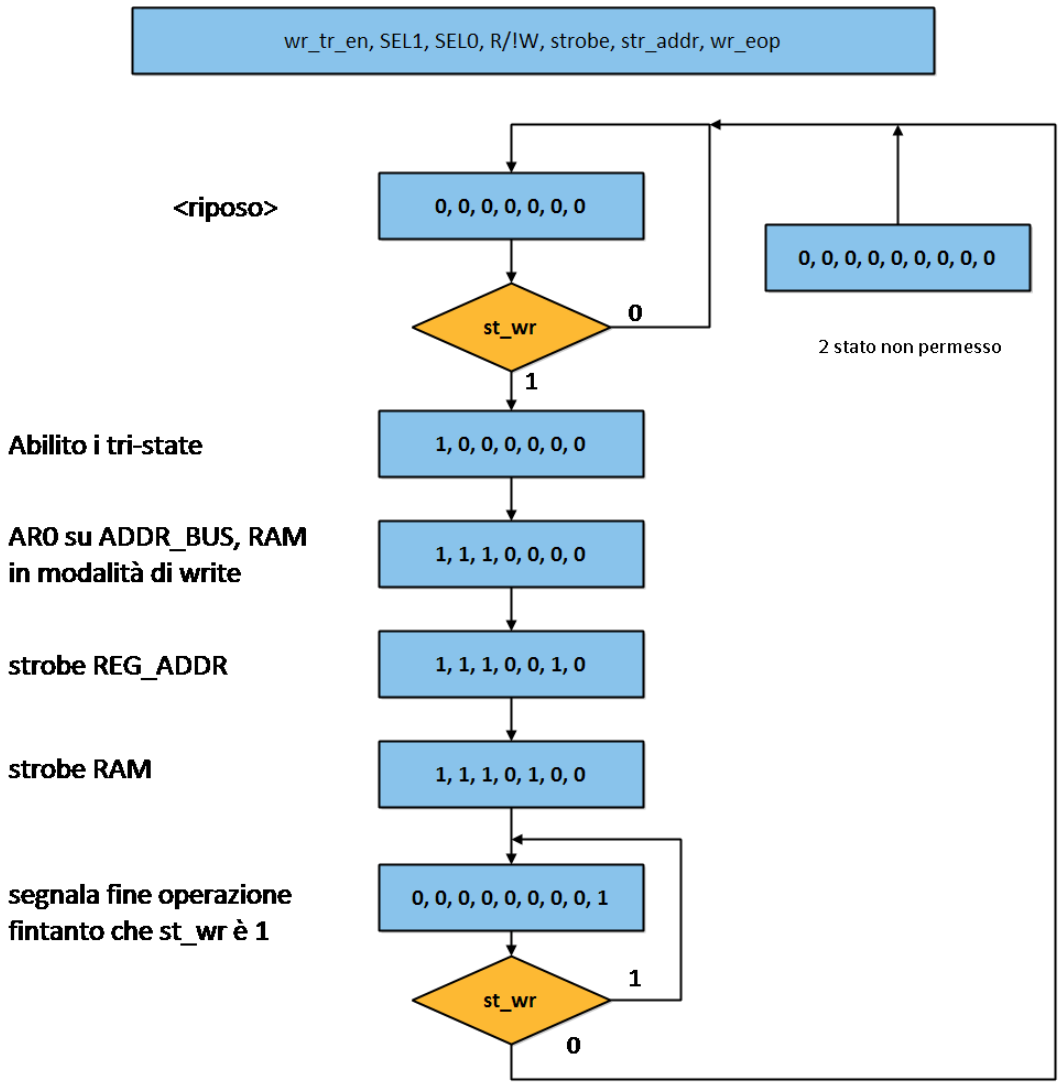
3.3.3.5 Slave RD

Esegue la lettura della memoria all'indirizzo presente in AR0 e salva il dato sul registro D_I/O.



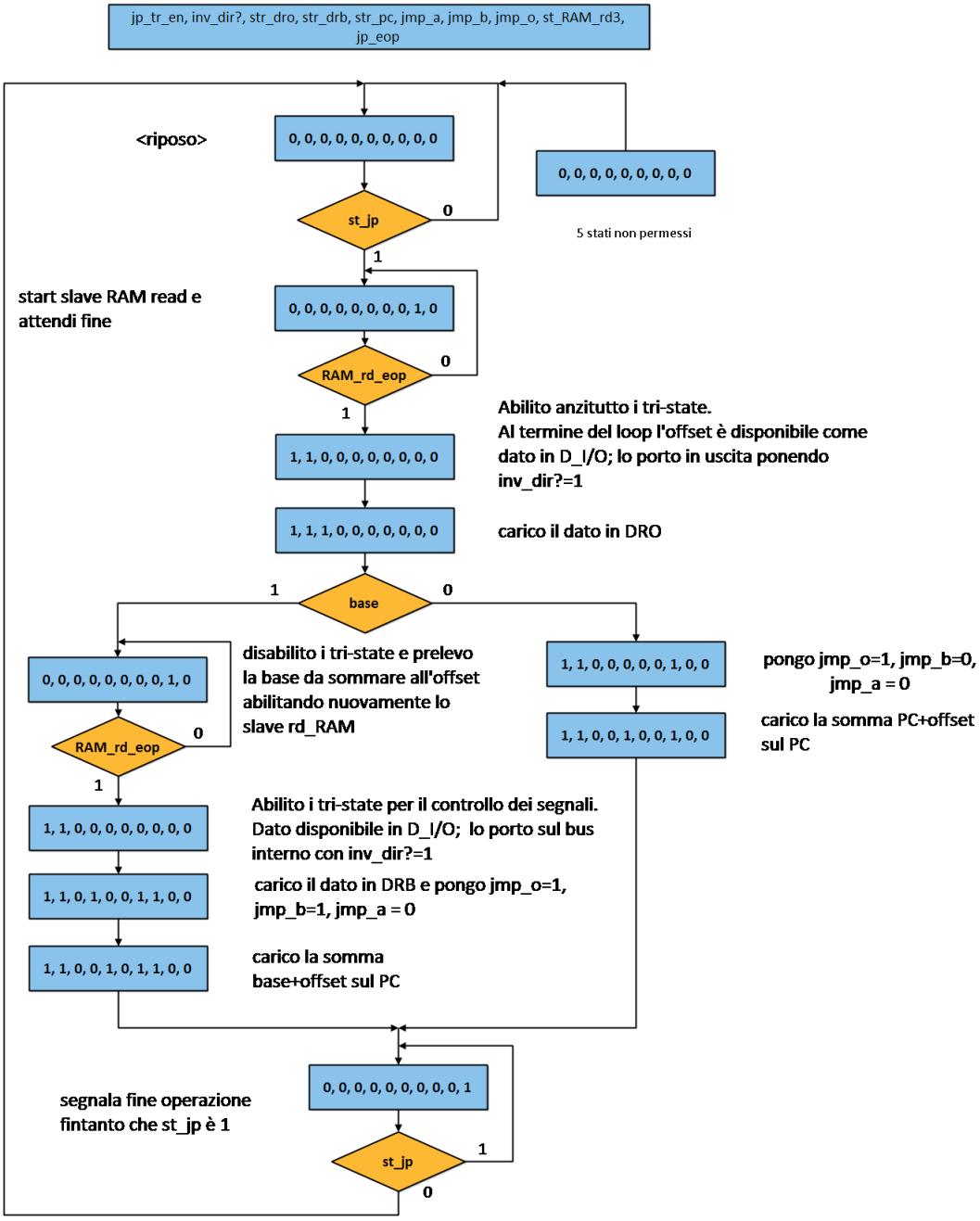
3.3.3.6 Slave WR

Esegue la scrittura del dato presente in D_I/O nella memoria alla locazione imposta dall'indirizzo presente in AR0.



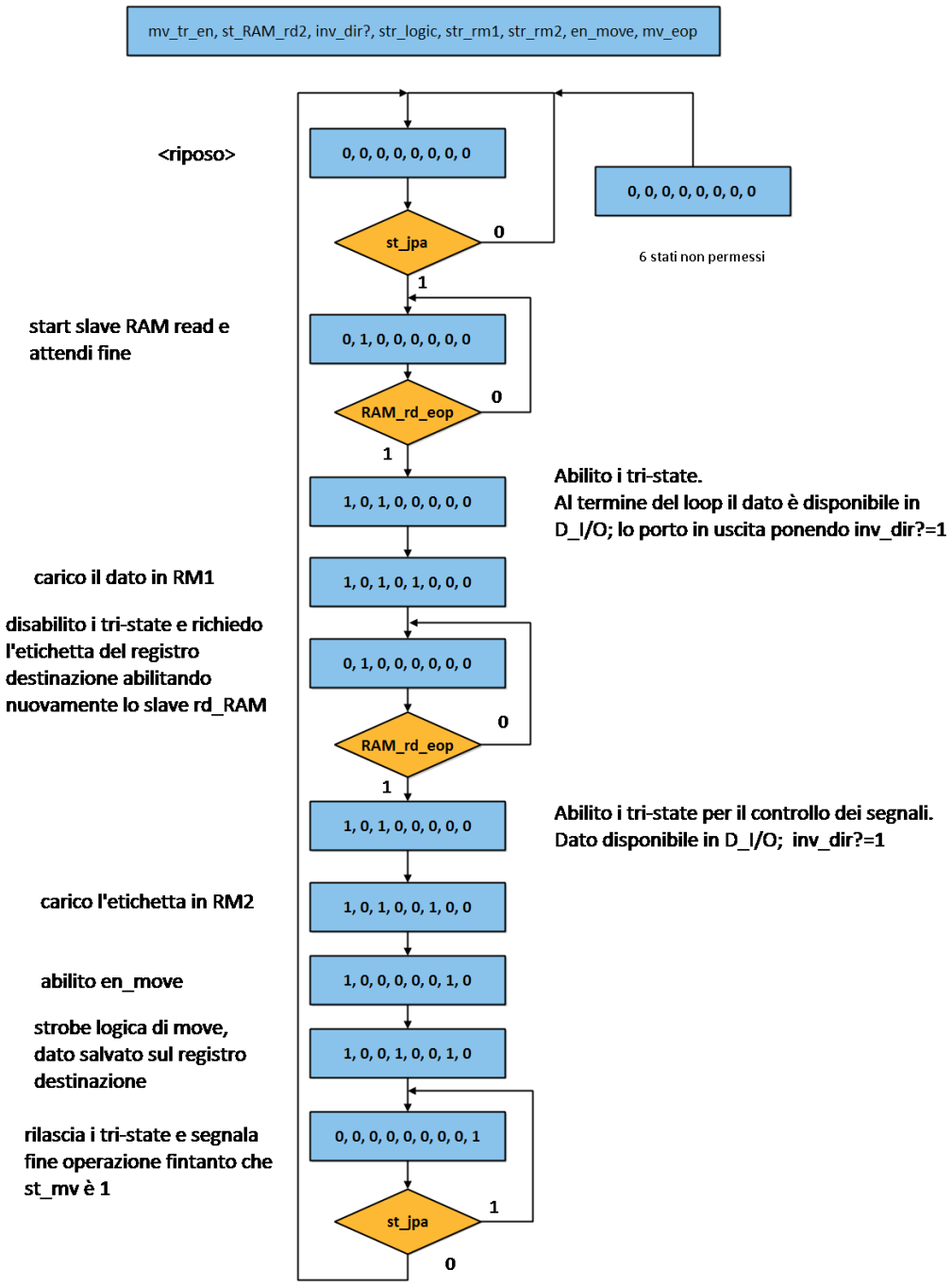
3.3.3.7 Slave JPO_JBO

Esegue la jump all'indirizzo dato dalla somma di una *base* e un offset. Mentre l'offset proviene dalla memoria, la base può essere l'indirizzo corrente del PC o un ulteriore dato prelevato dalla memoria.



3.3.3.8 Slave MV

Esegue la copia del dato da un registro sorgente ad uno destinazione indirizzati tramite etichetta prelevata dalla memoria.



3.3.3.9 ASM MASTER

Infine, il diagramma ASM per la macchina master che esegue le fasi di fetch e decode, delegando l'execute agli slave. Le azioni da svolgere sono:

FETCH:

1. Poni PC su ADDR_BUS, RAM in modalità di read.
2. Strobe REG_ADDR
3. RIPETI: Strobe RAM e poni inv_dir=1 FINCHÉ RAM_data_rd = 1
4. Strobe IR e incrementa PC

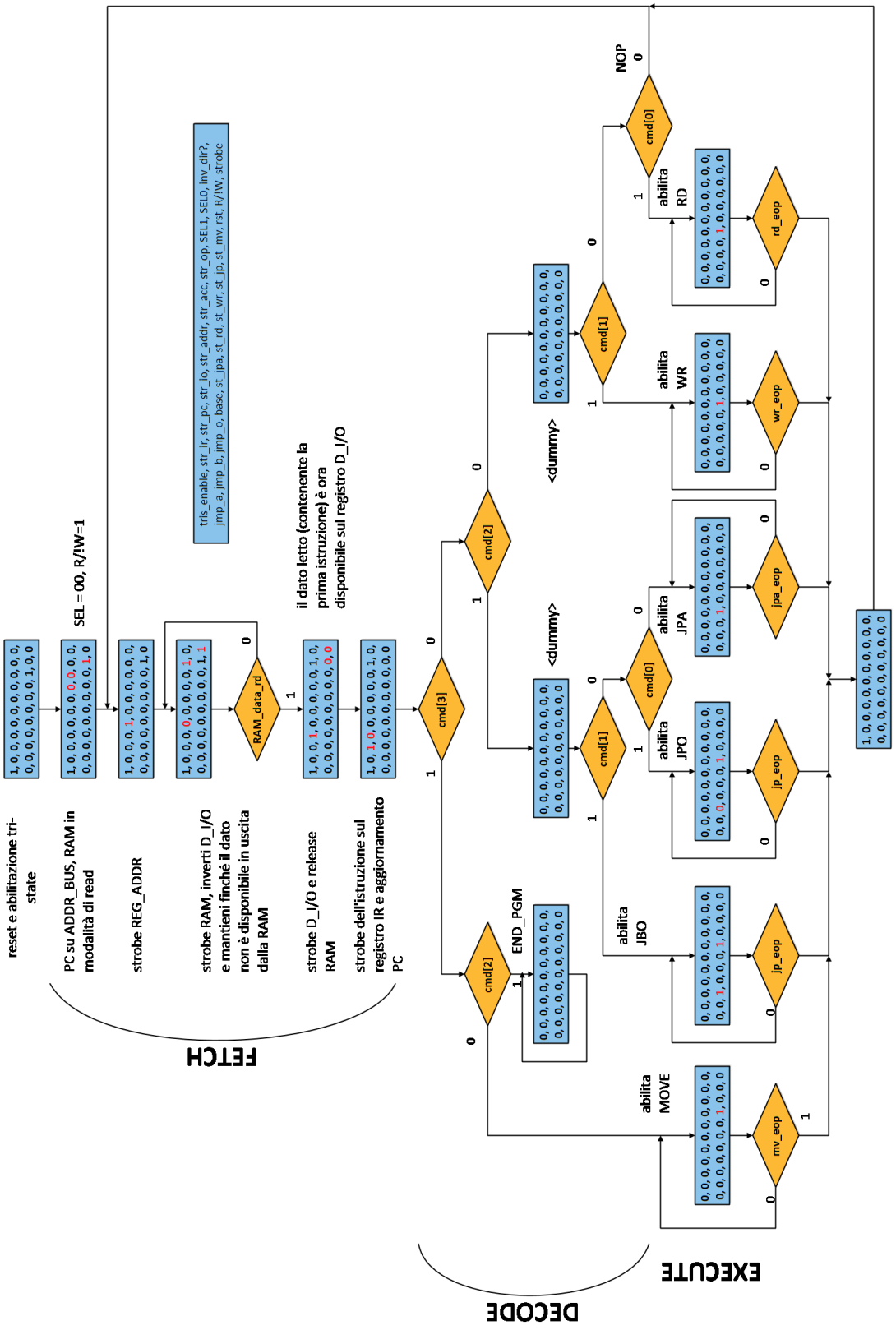
DECODE:

1. L'uscita del registro istruzioni IR contiene il codice relativo all'istruzione da eseguire. Una serie di controlli interni identifica l'operazione definita dai segnali di ***cmd***.

EXECUTE (WRITE-BACK):

1. Riconosciuta l'istruzione la macchina MASTER delega il controllo dell'hardware interno allo slave che eseguirà l'istruzione e cederà nuovamente il controllo alla master. Lo slave si occupa anche di un eventuale WRITE-BACK in memoria.

Da cui il diagramma ASM per la macchina MASTER.


FETCH
DECODE
EXECUTE
reset e abilitazione tri-state
PC su ADDR_BUS, RAM in modalità di read
SEL = 00, R/!W=1
strobe REG_ADDR
strobe RAM, inverti D_I/O e mantieni finché il dato non è disponibile in uscita dalla RAM
tris_enable, str_ir, str_pc, str_io, str_addr, str_acc, str_op, SEL1, SEL0, inv_dir?, jmp_a, jmp_b, jmp_o, base, st_jpa, st_rd, st_wr, st_jp, st_mv, rst, R/!W, strobe
RAM_data_rd
il dato letto (contenente la prima istruzione) è ora disponibile sul registro D_I/O
strobe D_I/O e release RAM
strobe dell'istruzione sul registro IR e aggiornamento PC
cmd[3]
cmd[2]
cmd[1]
cmd[0]
END_PGM
abilita JBO
abilita JPO
abilita JPA
abilita WR
abilita RD
abilita MOVE
<dummy>
<dummy>
jp_eop
jp_eop
jpa_eop
wr_eop
rd_eop
mv_eop
NOP