ARCHITETTURA LOGICO-DIGITALE

DEL PROCESSORE PD32

BRUNO CICIANI
DIPARTIMENTO DI INFORMATICA E SISTEMISTICA
UNIVERSITA' DI ROMA "LA SAPIENZA"

Indice

0 INTRODUZIONE

0.1 Obiettivi

0.2 Presentazione

0.3 Prerequisiti

1 ORGANIZZAZIONE LOGICA DEGLI ELABORATORI CONVENZIONALI

2 IMPLEMENTAZIONE DEL SOTTOSISTEMA DI CALCOLO

2.1 Organizzazione a blocchi del PD32

2.2 Implementazione dei registri interni usando flip-flop di tipo D

2.3 Organizzazione della struttura di interconnessione

2.3.1 Operazioni di trasferimento dei dati tra registri

2.3.2 Operazioni di trasferimento dei dati verso/da i circuiti di calcolo

2.3.3 Implementazione della struttura di interconnessione interna del PD32

2.3.4 Interfacciamento del bus interno del PD32 con le unità esterne

2.3.4.1 Interfacciamento con la memoria di lavoro

2.3.4.2 Interfacciamento con le porte di ingresso ed uscita

2.4 Circuiti di calcolo

2.4.1 Shifter

2.4.2 Unità Logico Aritmetica (ALU)

3 IMPLEMENTAZIONE DEL SOTTOSISTEMA DI CONTROLLO

3.1 Richiamo alla microprogrammazione

3.2 Comunicazioni con la memoria di lavoro

3.3 Comunicazione con i dispositivi di ingresso ed uscita

3.4 Gestione eventi asincroni

3.4.1 Gestione delle richieste di interruzione

3.4.2 Gestione delle richieste di accesso ai bus

3.4.3 Diagramma di flusso della gestione degli eventi asincroni

3.5 Segnali di controllo del PD32

3.6 Passi elementari dell'interpretazione di una istruzione

3.6.1 Ciclo di fetch

3.6.2 Ciclo di riconoscimento di interruzione

4 CONCLUSIONI

4.1 Sommario

4.2 Risposte alle domande

4.3 Soluzione degli esercizi

0 INTRODUZIONE

0.1 Obiettivi

Obiettivo di queste dispense è quello di progettare il processore didattico con reti logiche combinatorie e sequenziali. A tal fine si espliciteranno prima le motivazioni funzionali e di costo/prestazione che hanno portato alla definizione dell'architettura del PD32 e della memoria di lavoro (che sarà implementata tramite moduli di memoria RAM statica), dopodichè si entrerà nel merito della progettazione di ogni singolo componente con reti logiche combinatorie e sequenziali. Poichè, infine, si farà vedere come vengono eseguite le varie classi di istruzioni, e tra queste vi sono anche quelle relative ai dispositivi di ingresso ed uscita, si descriveranno anche alcune tecniche di interazione con le stesse.

0.2 Presentazione

Come ogni progettazione di sistemi digitali complessi, la progettazione del PD32 è effettuata tenendo conto delle *funzionalità*, del *costo* e delle *prestazioni* che si vogliono ottenere.

Data la complessità delle funzioni che il PD32 deve effettuare, la sua progettazione è effettuata scomponendolo in due entità interagenti: un *Sottosistema di COntrollo* (SCO), con funzioni di coordinamento, ed un *Sottosistema di CAlcolo* (SCA) asservito a quello di controllo. Il SCA è l'insieme dei registri, dei blocchi funzionali (atti ad effettuare le operazioni sul contenuto dei registri) e della struttura di interconnessione necessaria per il trasferimento dei dati tra i registri e i blocchi funzionali. Mentre il SCO è la rete sequenziale che, sotto le direttive del programma da eseguire (scritto in linguaggio macchina), coordina le attività del SCA.

Data il numero delle funzioni di controllo, il modo più semplice per implementare il SCO è tramite la *microprogrammazione* e quindi il problema della progettazione del SCO diviene quello della determinazione di un algoritmo che implementi le funzioni di controllo richieste (ed in seconda fase il tipo di organizzazione della microprogrammazione e della scelta del microlinguaggio da usare nella descrizione dell'algoritmo). Per il SCA, invece, la difficoltà della progettazione risiede nella scelta, tra le possibili alternative, della soluzione tenendo conto del costo e delle ripercussioni sulla complessità dell'algoritmo del SCO. In generale tanto più è complesso il SCA tanto più semplice sarà il SCO. Pertanto particolare attenzione va riposta nella progettazione dei singoli componenti del SCA. Poichè il modello di riferimento scelto per il processore è quello proposto da *Von Neumann*, il

PD32 può eseguire una istruzione alla volta, ciò ha comportato la definizione di un SCA con un basso investimento hardware. Di seguito prima di far vedere le scelte implementative che si sono effettuate per il SCA, si faranno vedere alcune possibili alternative e se ne discuteranno criticamente le loro ripercussioni in termini di *costo* e di *prestazioni*.

0.3 Prerequisiti

Per poter studiare queste dispense è necessario avere una certa familiarità con i seguenti argomenti: rappresentazione delle informazioni e aritmetica binaria, sintesi di reti combinatorie, reti sequenziali, sintesi di macchine sequenziali tramite microprogrammazione, linguaggio macchina del PD32.

1. ORGANIZZAZIONE LOGICA DEGLI ELABORATORI CONVENZIONALI

La macchina di Von Neumann è stata scelta come modello base per la realizzazione del sistema "elaboratore". I suoi componenti base sono:

- una unità di ingresso;
- una unità di uscita;
- una memoria di lavoro;
- una unità di calcolo;
- una unità di controllo.

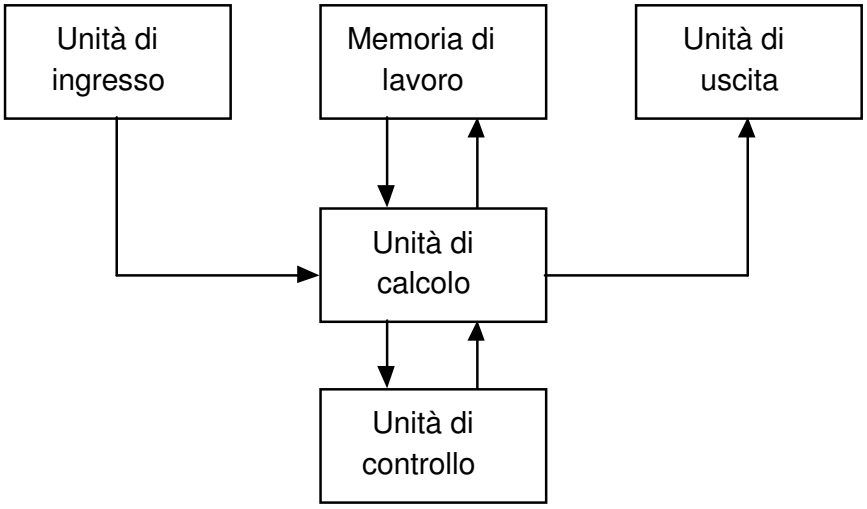


Figura 1: Macchina di Von Neumann

Il sistema elaboratore, come tutti i sistemi digitali complessi, può essere suddiviso in due sottosistemi: un *Sottosistema di COntrollo* (SCO), con funzioni di coordinamento, ed un *Sottosistema di CAlcolo* (SCA) asservito a quello di controllo. Il SCA è l'insieme dei registri, dei blocchi funzionali (atti ad effettuare le operazioni sul contenuto dei registri) e della struttura di interconnessione necessaria per il trasferimento dei dati tra i registri e i blocchi funzionali. Mentre il SCO è la rete sequenziale che, sotto le direttive del programma da eseguire (codificato in linguaggio macchina), coordina, tramite dei *segnali di controllo*, le attività del SCA. Da notare che le attività del SCO sono a loro volta condizionate da quelle del SCA; questo condizionamento è attuato tramite dei segnali denominati *variabili di condizione* .

La suddivisione della macchina di Von Neumann in un SCO ed un SCA può quindi essere schematizzata come in Figura 2.

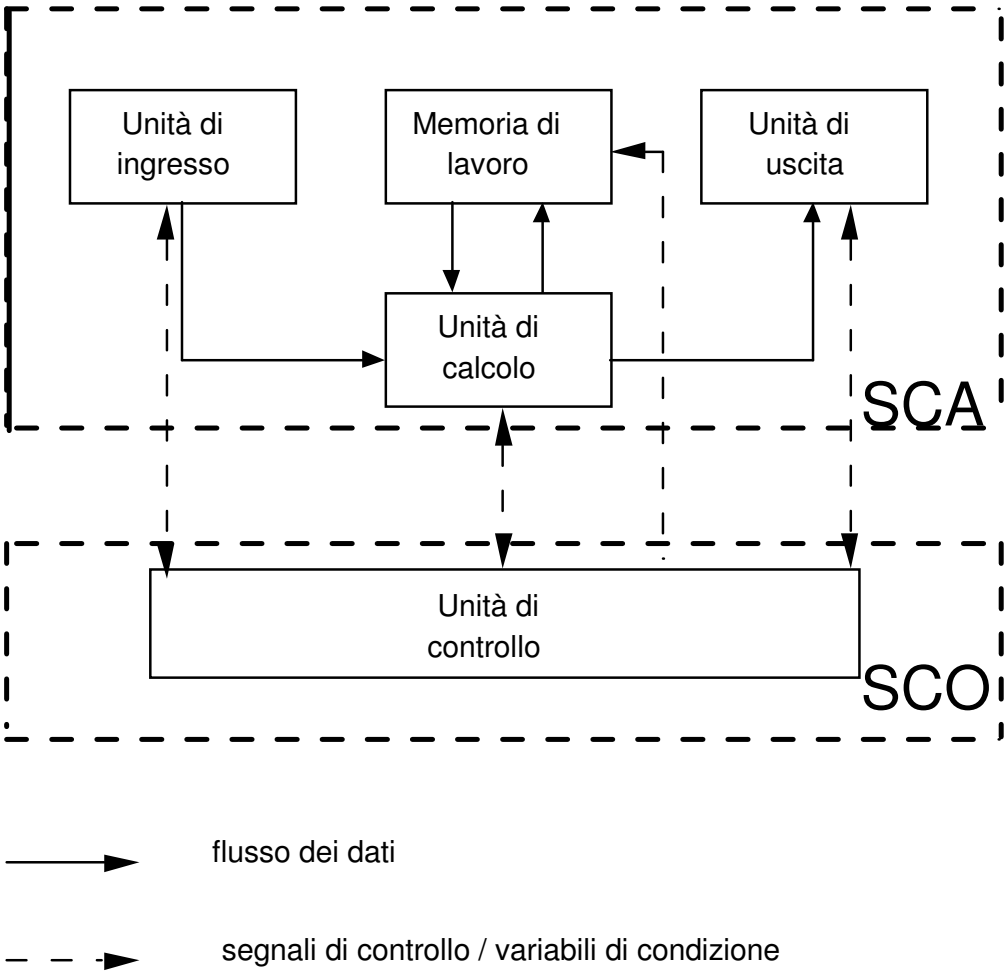


Figura 2: Specializzazione della macchina di Von Neumann secondo l'organizzazione SCO-SCA

In Figura 2 sono evidenziati i segnali di controllo che dal SCO vanno verso le varie unità funzionali, le variabili di condizione che dalle unità funzionali vanno verso il SCO e i flussi delle informazioni tra le unità funzionali.

Per motivi di efficienza negli elaboratori convenzionali (cioè quelli che corrispondono al modello di Von Neumann) l'unità di calcolo, quella di controllo ed una porzione della memoria dati sono stati raggruppati in un'unica entità a cui è stato dato il nome di *processore*. (o *Central Processing Unit - CPU*). Quindi l'organizzazione di un elaboratore convenzionale con processore corrispondente al modello di Von Neumann,

considerando la presenza di una struttura di interconnessione per la trasmissione delle informazioni, può essere specializzata come in Figura 3.

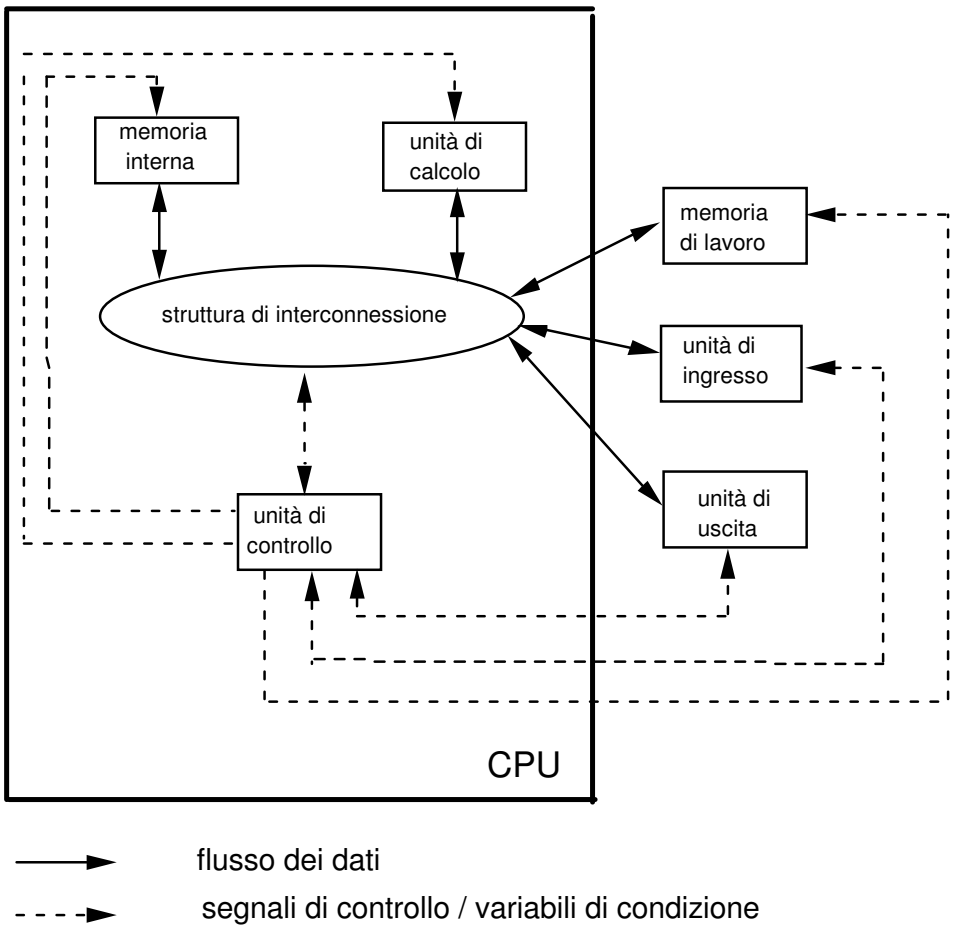


Figura 3: Organizzazione logica di un elaboratore convenzionale con processore

In ogni progettazione è necessario tenere conto del *costo* e delle *prestazioni* che si vogliono ottenere. In particolar modo avendo suddiviso la progettazione in due sottosistemi, nel progetto di ogni singolo sottosistema è necessario valutare ogni scelta, non solo tenendo conto delle possibili ripercussioni che si possono avere al proprio interno, ma anche valutare qualitativamente e quantitivamente l'eventuale interazione con l'altro sottosistema.

Data la complessità delle funzioni di controllo, il modo più semplice per implementare il SCO è tramite la *microprogrammazione* e quindi il problema della progettazione del SCO diviene quello della determinazione di un algoritmo che implementi le funzioni di controllo richieste (ed in seconda fase il tipo di organizzazione della microprogrammazione e della scelta del microlinguaggio da usare nella descrizione dell'algoritmo). Per il SCA,

invece, la difficoltà della progettazione risiede nella scelta, tra le possibili alternative, della soluzione tenendo conto del costo e delle ripercussioni sulla complessità dell'algoritmo del SCO. In generale tanto più è complesso il SCA tanto più semplice sarà il SCO. Pertanto particolare attenzione va riposta nella progettazione dei singoli componenti del SCA. Di seguito prima di far vedere le scelte implementative che si effettueranno per il SCA, si faranno vedere alcune possibili alternative e se ne discuteranno criticamente le loro ripercussioni in termini di costo e di prestazioni.

2. IMPLEMENTAZIONE DEL SOTTOSISTEMA DI CALCOLO

2.1 Organizzazione a blocchi del PD32

Come visto nella Sezione precedente, essendo il PD32 un processore convenionale, esso è costituito da:

- una unità di calcolo;
- una memoria locale;
- una struttura di interconnessione;
- una unità di controllo.

L'unità di controllo è il SCO, mentre il SCA comprende l'unità di calcolo, la memoria locale e la struttura di inteconnessione. L'unità di calcolo a sua volta è suddivisa in due unità funzionali: l'unità logica aritmetica (Aritmetic Logic Unit, ALU) e l'unità che implementa le operazioni di rotazione e spostamento (shifter). La struttura di interconnessione è necessaria per mettere in comunicazione la memoria locale con le unità funzionali e con i dispositivi esterni.

La memoria locale è costituita da:

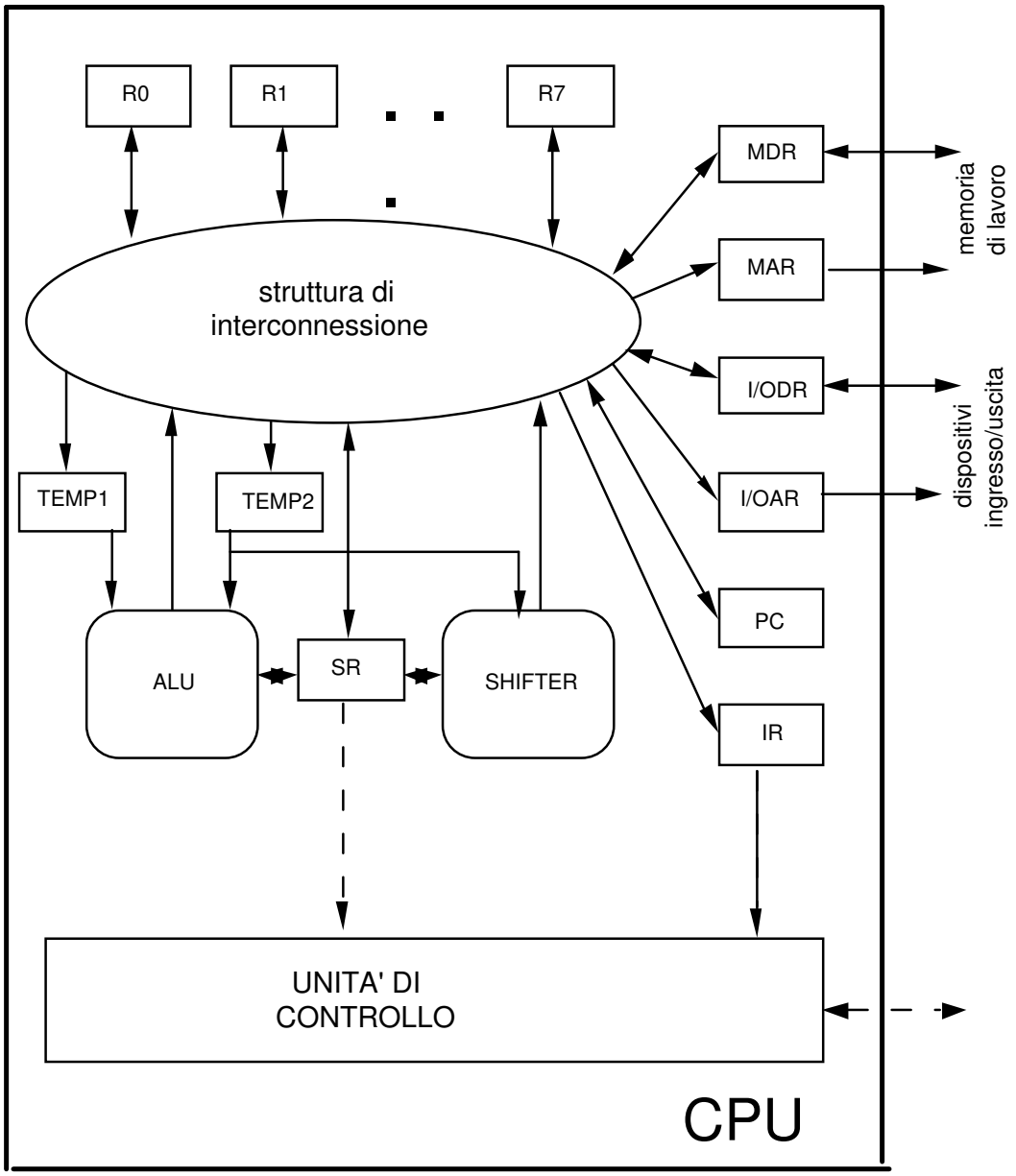
- un insieme di registri di uso generale (R0,. . .,R6), da registro di uso speciale R7 (che memorizza il puntatore dello stack) e da un registro contenente i flag di stato (Status Register, SR)*visibili* o *accessibili* dall'esterno. Questi registri sono indirizzabili e quindi modificabili direttamente dalle istruzioni macchina (vedi set delle istruzioni);

- un insieme di registri utilizzabili solo dal SCO per poter interpretare le istruzioni macchina; in particolare il PC (Program Counter) serve per puntare alla successiva istruzione macchina da eseguire, l'IR (Instruction Register) serve per memorizzare il codice dell'istruzione corrente da interpretare (come vedremo in Sezione 3 il suo contentuto è utilizzato come ingresso al SCO), mentre gli altri registri (TEMP1, TEMP2, MDR, MAR, I/ODR e I/OAR), come vedremo successivamente, avranno funzioni di memorizzazione temporanea di informazioni.

Domanda 1

Quali sono le informazioni che possono transitare nei registri tampone?

In Figura 4 è presentata l'organizzazione a blocchi del PD32.



N.B.
non sono indicati i segnali di controllo che dal SCO vanno verso i registri interni, le unità di calcolo e la struttura di interconnessione

Figura 4: Organizzazione a blocchi del PD32

Le operazioni tipiche che vengono effettuate sui dati memorizzati nei registri sono:

- trasferimento dati:
  - tra registri interni;
  - da/a registro interno a/da memoria di lavoro;
  - da/a registro interno a/da periferica;

- rotazione e spostamento; e

- operazioni logiche e aritmetiche.

Le operazioni di trasferimento vengono coordinate dal SCO utilizzando la struttura di comunicazione che può prevedere il trasferimento simultaneo o meno di più contenuti di registri (vedi Paragrafo 2.3).

Le operazioni di rotazione e spostamento, coordinate sempre dal SCO, vengono effettuate dallo "shifter" (vedi Paragrafo 2.4), mentre le operazioni logiche e aritmetiche vengono effettuate dall'ALU. Ovviamente l'esecuzione delle operazioni di rotazione, spostamento, logiche e aritmetiche necessitano a loro volta di trasferimenti dati tra i registri e le unità funzionali, e viceversa.

Di conseguenza particolare attenzione deve essere posta nella progettazione della struttura di interconnessione. Di seguito dopo aver descritto come si può progettare un registro interno si farà vedere quali sono le possibili scelte per interconnettere tra di loro i registri e i registri con le unità funzionali.

Domanda 2

Quanti segnali di controllo sono necessari per specificare all'ALU il tipo di operazione da effettuare?.

2.2 Implementazione dei registri interni usando flip-flop di tipo D

I registri sono realizzati con flip-flop (reti logiche sequenziali) connessi in banchi il cui numero è pari a quello dei bit da memorizzare. Poichè il processore PD32 è caratterizzato da 32 bit i registri avranno 32 flip-flop. I flip-flop possono essere di vario tipo (S-R, J-K, D). Per semplicità, nell'implementazione del PD32, si useranno flip-flop di tipo D commutanti sul fronte di salita del segnale di abilitazione (o di controlo).

Il simbolo grafico che rappresenta un flip-flop di tipo D commutante sul fronte di salita del segnale di abilitazione è riportato in Figura 5. L'uscita Q assume il valore presente sull'ingresso D (di consequenza l'uscita $\overline{Q}$ prenderà il valore negato di D) nell'istante di commutazione del segnale di abilitazione da 0 ad 1, indicato con C. Il circuito funziona, quindi, da ritardo (da cui il nome D, da delay in inglese) tra l'istante in cui commuta l'ingresso D e l'istante in cui commuta l'uscita; il ritardo è controllato da C.

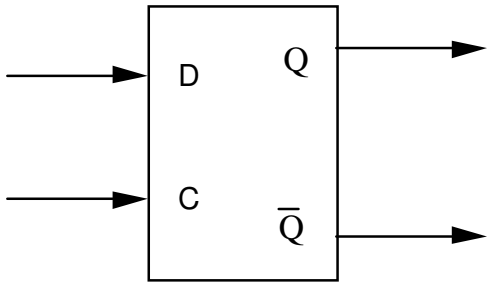


Figura 5: Simbolo grafico del flip-flop di tipo D

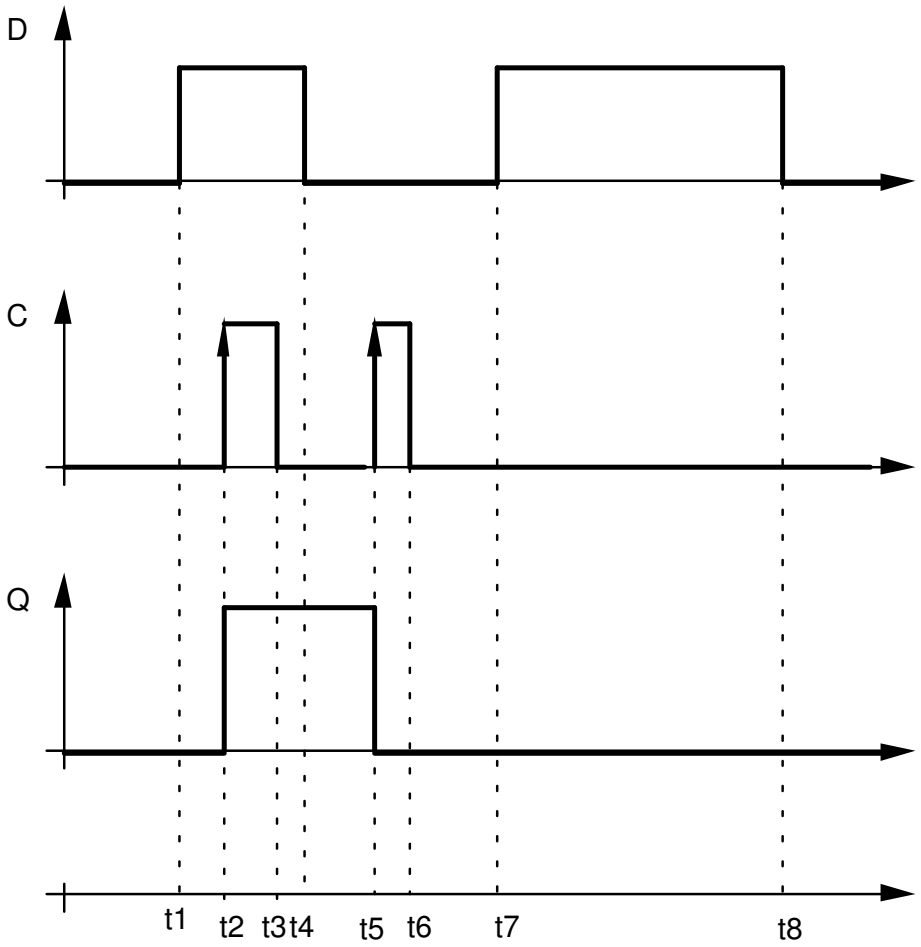


Figura 6: Esempio di evoluzione temporale dell'uscita del flip-flop di tipo D commutante sul fronte di salita del segnale di abilitazione

In Figura 6 è rappresentata una possibile evoluzione temporale di un flip-flop di tipo D commutante sul fronte di salita del segnale di abilitazione. Notare come l'uscita Q rimane a livello logico 0 da t5 in poi, anche se tra t7 e t8 l'ingresso D è pari ad 1. Ciò è dovuto al fatto che in tale intervallo non è presente la commutazione del segnale di abilitazione.

Domanda 3

Qual'è il livello logico dell'uscita Q all'istante t8 se ci fosse un impulso dell'ingresso C tra l'istante t6 e t8?

Poichè la commutazione degli *n* flip-flop deve avvenire contemporaneamente in modo sincrono, il segnale di abilitazione di questi flip-flop deve essere unico, come schematizzato in Figura 7.

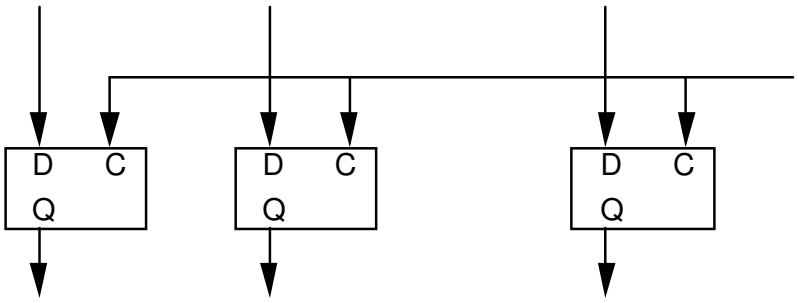


Figura 7: Schema di registro

Esercizio 1

Schematizzare un registro usando flip-flop di tipo SR.

2.3. Organizzazione della struttura di interconnessione

In questo Paragrafo si faranno vedere separatamente alcune possibili implementazioni di strutture di interconnesione per il trasferimento dei dati tra registri interni e da/a registri interni a/da i circuiti di calcolo. Dopodichè si farà vedere come le diverse scelte potranno essere fuse utilizzando un'unica struttura di interconnessione. Infine si farà vedere come questa struttura di interconnessione possa essere interfacciata con la memoria di lavoro e con i dispositivi di ingresso ed uscita.

2.3.1 Operazioni di trasferimento dei dati tra registri

Indicando con R_S (registro sorgente) e con R_d (registro destinazione) due generici registri interni del processore (quindi non solo i registri R0-R7, ma anche gli altri registri come TEMP1,TEMP2, PC, MAR, I/OAR etc.), il trasferimento del contenuto del dato memorizzato in R_S nel registro R_d, indicato con:

$$(R_S) \rightarrow R_d$$

viene coordinato dal SCO. Per far ciò il SCO abilita la scrittura del contenuto del registro sorgente nel registo destinatario. Al termine dell'operazione il contenuto del registro R_d sarà uguale a quello di R_S, mentre il valore precedentemente memorizzato in R_d viene perduto.

Per poter effettuare questa operazione, che è proprio una operazione di ricopiatura, nel caso che si disponga di registri implementati con flip-flop di tipo D, i due registri in questione possono essere collegati come in Figura 8; in questo caso il trasferimento dei dati viene effettuato in un singolo passo (cioè il SCO deve solo generare il segnale di controllo Wd).

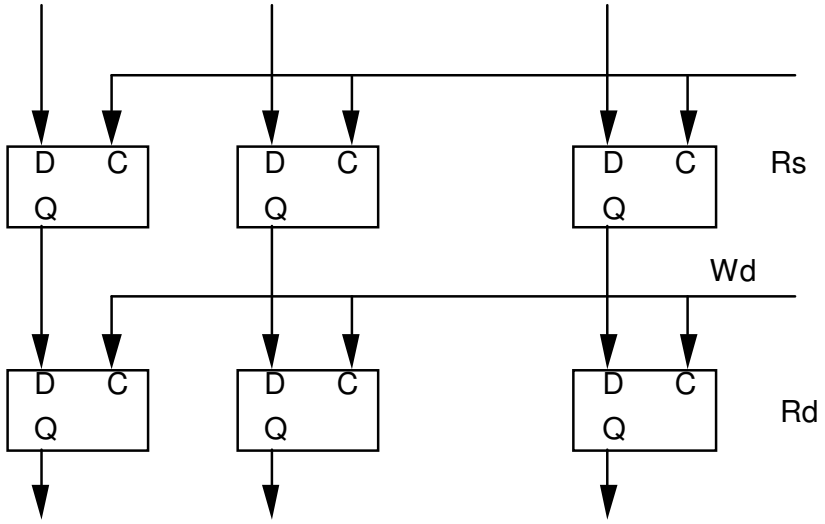


Figura 8: Interconnessione tra due registri

Esercizio 2

Schematizzare un'interconnessione tra due registri ipotizzando di usare registri con flip-flop SR.

All'interno del processore c'è una molteplicità di registri, poichè in generale qualunque registro dovrebbe essere connesso ad un qualunque altro registro è necessario prevedere una struttura di interconnessione che permetta questa possibilità. Vi sono diverse strutture che permettono ciò. Di seguito verranno schematizzate le due soluzioni estreme. Queste soluzioni massimizzano e minimizzano il numero contemporaneo di trasferimenti possibili, quindi se *n* è il numero di registri da connettere, nel primo caso si potranno avere contemporaneamene *n/2* trasferimenti, mentre nel secondo caso solo *uno*. La prima soluzione necessita di un collegamento diretto tra ogni coppia di registri. Questa soluzione può essere implementata tramite l'utilizzazione di tanti multiplexer per quanti sono i registri, come schematizzato in Figura 9.

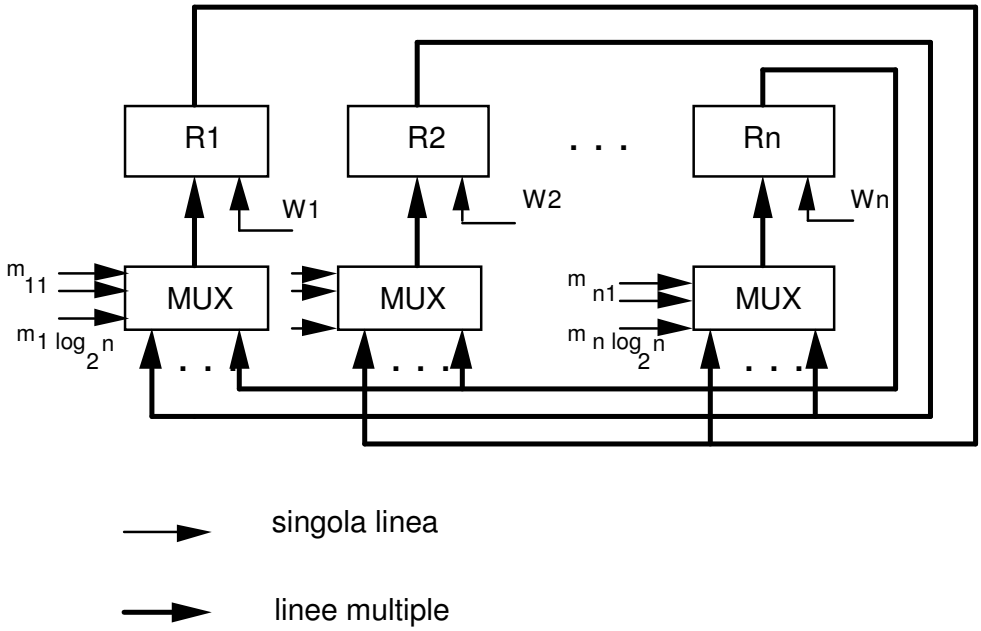


Figura 9: Struttura di interconnessione con multiplexer.

I segnali di controllo m_{11}, . . ., $m_{1\log_2 n}$, m_{n1}, . . ., $m_{n\log_2 n}$ e W_1, W_2, . . ., W_n sono generati dal SCO e servono, rispettivamente, per la selezione dei registri sorgente e l'abilitazione della scrittura dei dati sui registri destinatari. Quindi il numero degli ingressi di ogni singolo multiplexer è paria $log_2 n$ più *n-1* per il numero di bit dei registri, di conseguenza la complessità della rete di interconnessione è dell'ordine di *n(n-1)*, che è una complessità notevole.

Domanda 4

Dato lo schema di Figura 9, quanti registri possono memorizzare contemporaneamente il contenuto di un registro?

La soluzione opposta, che minimizza il numero di trasferimenti, minimizza anche la complessità circuitale. Infatti, se si prevede un solo trasferimento alla volta, si può pensare di utilizzare un unico mezzo trasmissivo in cui di volta in volta il registro abilitato alla scrittura può inviare i propri dati. Questa soluzione può essere implementata utilizzando un unico set di linee trasmissive (pari al numero dei bit di ogni singolo registro) a cui ogni registro è interfacciato con un buffer three state, come schematizzato in Figura 10, a questa struttura si da il nome di bus. Per comodità del lettore si ricorda che i buffer three-state sono dispositivi che hanno la peculiarità di presentare in uscita o un livello ad alta impedenza (che equivale alla

sconnessione tra ciò che sta a monte del buffer three-state da ciò che sta a valle) o un livello a bassa impedenza riportando in uscita il valore logico presente in ingresso (corrispondente ad uno 0 oppure un 1 logico). Il livello alto o basso dell'impedenza di uscita è controllato da un segnale di controllo.

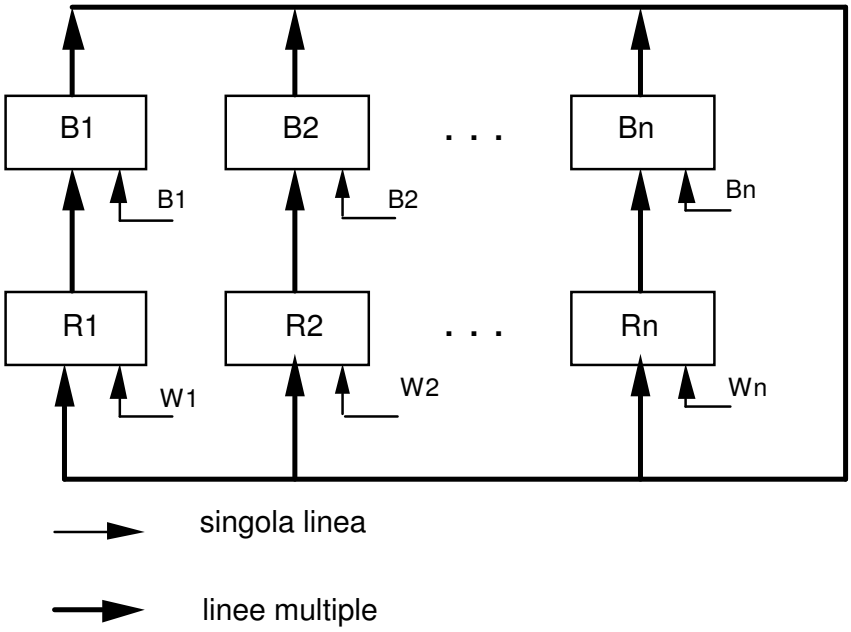


Figura 10: Struttura di interconnessione a bus.

I segnali di controllo B_1,. . .,B_n abilitano il trasferimento dei dati da ogni singolo registro sulle linee del bus. Di conseguenza, per evitare fenomeni elettrici non desiderati (quali cortocircuiti), è necessario abilitare un solo registro alla volta. I segnali di controllo W_1,. . .,W_n sono necessari per abilitare la memorizzazione del dato presente sulle linee del bus nei registri.

Ovviamente tra queste due soluzioni estreme vi sono innumerevoli soluzioni intermedie utilizzanti più bus su cui sono attestati differenti insiemi di registri.

Domanda 5

Dato lo schema di Figura 10, quanti registri possono memorizzare contemporaneamente il contenuto di un registro?

Esercizio 3

Descrivere una possibile soluzione in cui è possibile il trasferimento contemporaneo del contenuto di due registri.

2.3.2 Operazioni di trasferimento dei dati verso/da i circuiti di calcolo

Alcune istruzioni aritmetiche (come ADD, SUB) ed alcune istruzioni logiche (come AND, OR) del set di istruzione del PD32 operano su due operandi; altre istruzioni aritmetiche e logiche (come NEG, NOT) e le istruzioni di rotazione e spostamento operano su un unico operando. Di conseguenza, mentre l'ALU deve essere messa in condizione di poter ottenere dati da due registri contemporaneamente, lo shifter necessita di essere connesso con un solo registro alla volta. Dopodichè una volta effettuata l'operazione richiesta il risultato va memorizzato in un registro.

Supponiamo come nella Sezione precedente che il processore sia corredato da un certo numero di registri e che questi siano operativamente equivalenti per l'ALU e lo shifter (ovvero ogni operando dei circuiti di calcolo può essere memorizzato in uno qualunque dei suddetti registri). Di conseguenza, indicando con R_{S1} (registro sorgente 1), con R_{S2} (registro sorgente 2) e con R_d (registro destinazione) tre generici registri interni del processore, l'esecuzione di una operazione a due operandi, indicata con:

$$R_{S1} \text{ op } R_{S2} \rightarrow R_d$$

viene coordinata dal SCO, che abilita il prelievo dei dati memorizzati nei registri sorgenti, abilita l'operazione richiesta ed infine abilita la scrittura del risultato nel registo destinatario.

Mentre, indicando con R_S (registro sorgente) e con R_d (registro destinazione) due generici registri interni del processore, l'esecuzione di una operazione ad un solo operando, indicata con:

$$\text{op } R_S \rightarrow R_d$$

viene coordinata dal SCO, che abilita il prelievo dei dati memorizzati nel registro sorgente, abilita l'operazione richiesta ed infine abilita la scrittura del risultato nel registo destinatario.

Poichè in generale, qualunque registro potrebbe essere connesso ad entrambi i circuiti di calcolo è necessario prevedere una struttura di interconnessione che permetta ciò. Vi sono diverse strutture di interconnessione. Di seguito verranno schematizzate due soluzioni estreme. La prima minimizza il tempo di esecuzione delle operazioni a due operandi e permette anche l'esecuzione in parallelo di operazioni comportanti l'attività di entrambi i circuiti di calcolo. La seconda invece minimizza l'uso di componenti hardware, con una conseguente degradazione delle prestazioni.

La prima soluzione necessita di un collegamento diretto di ogni registro con entrambi i circuiti di calcolo e viceversa. Questa soluzione può essere implementata (vedi Figura 11) tramite l'utilizzazione di tanti multiplexer quanti sono i registri, per connettere le uscite dell'ALU e dello shifter con gli ingressi dei registri, di due multiplexer per connettere le uscite dei registri con

i due ingressi dell'ALU, e di un multiplexer per connettere le uscite dei registri con l'ingresso dello shifter.

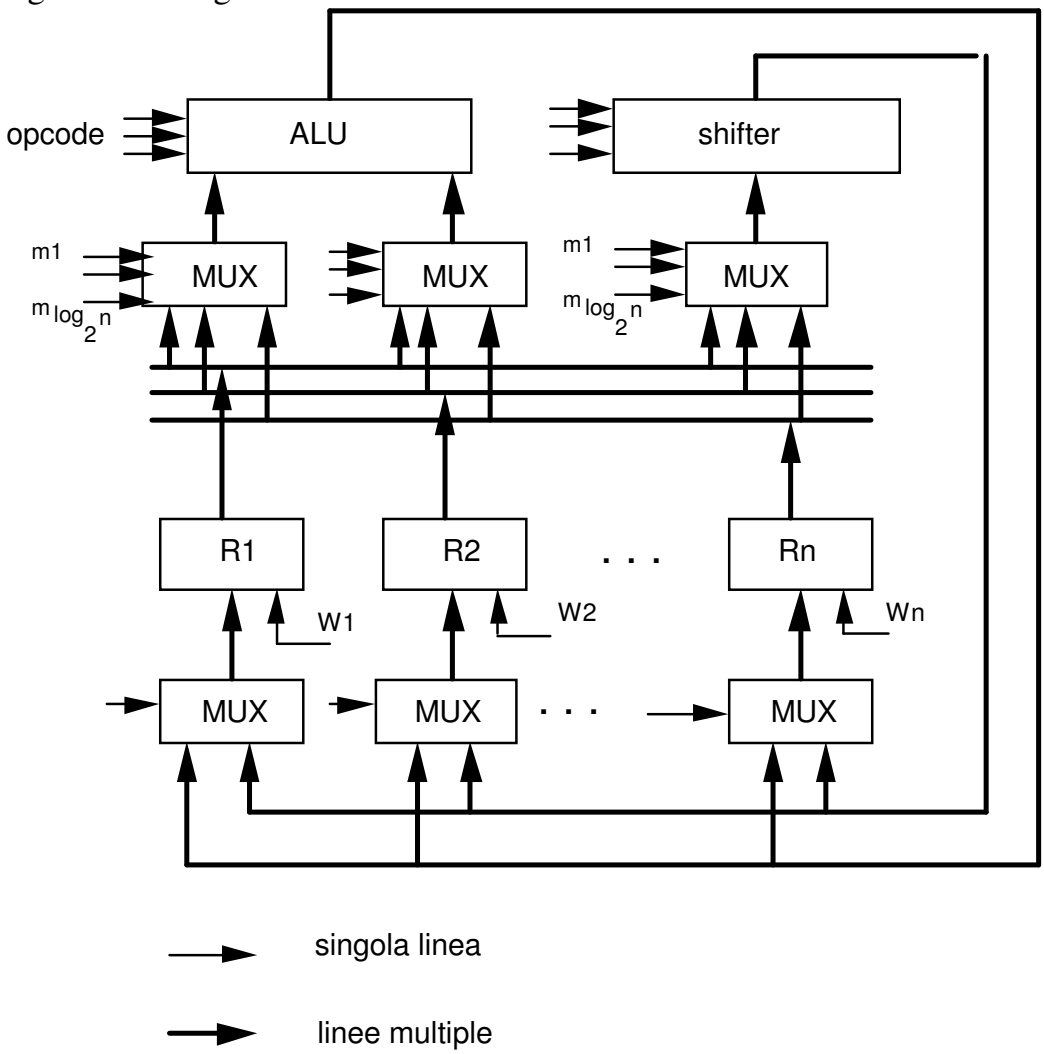


Figura 11: Struttura di interconnessione con multiplexer.

Da notare che l'esecuzione di un operazione a due operandi prevede quindi:

- la connessione di due registri con l'ALU (che viene comandata dal SCO abilitando opportunamente i multiplexer in ingresso all'ALU);

- la specificazione del tipo di operazione da effettuare (che viene fatta dal SCO comandando opportunamente l'ALU);

- la scrittura del risultato nel registro destinatario (che viene effettuata dal SCO comandando uno dei multiplexer in ingresso ai registri).

Domanda 6

Quali sono i passi elementari necessari per eseguire una operazione ad un solo operando?

Da notare che poichè esiste una interconnessione diretta tra ogni registro ed ogni singolo circuito di calcolo, e viceversa, i due circuiti di calcolo potrebbero lavorare in parallelo se il registro destinatario delle operazioni eseguibili in parallelo è diverso.

Si può notare che questa struttura potrebbe essere utilizzata anche per effettuare il trasferimento del contenuto di un registro in un altro. In questo caso è sufficiente prevedere che tra le funzionalità dello shifter ci sia quella di non operatività, in modo da presentare in uscita ciò che è presente al suo ingresso.

L'altra soluzione è simile alla seconda utilizzata per la struttura di interconnessione tra registri interni e prevede l'utilizzazione di un unico bus.

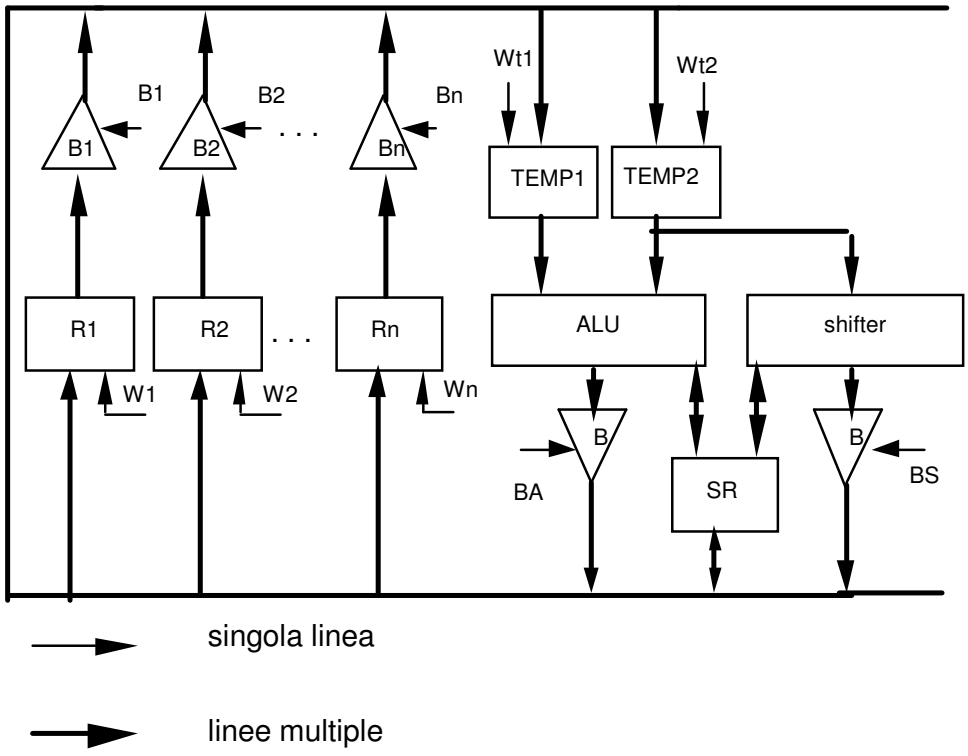


Figura 12: Struttura di interconnessione con bus.

Anche in questo caso, poichè vi è una sola risorsa trasmissiva (le linee di comunicazione del bus) è necessario che ogni singolo dispositivo (registro o circuito di calcolo) che può inviare dati sul bus sia provvisto di un buffer three state. Inoltre, poichè un solo dispositivo alla volta è abilitato ad inviare i dati sul bus, è necessario disporre di due registri tampone o temporanei (da cui la necessità di TEMP1 e di TEMP2 nel SCA del PD32) che si interfaccino

direttamente con l'ALU e di un registro tampone che si interfacci direttamente con lo shifter. Per risparmiare hardware, dato il principio di funzionamento sequenziale della macchina di Von Neumann (quindi se è attivo lo shifter non lo può essere l'ALU), è possibile utilizzare come ingresso dello shifter uno dei due registri tampone visti precedentemente. In questo modo quindi non ci potrà essere interferenza nè logica nè elettrica nel passaggio dei dati dai registri sorgenti ai circuiti di calcolo e da questi al registro destinazione.

L'esecuzione di un operazione a due operandi prevede quindi:

- la scrittura del primo operando su TEMP1 (che viene comandata dal SCO abilitando opportunamente il buffer three state relativo al primo registro sorgente e abilitando in scrittura il registro TEMP1);

- la scrittura del secondo operando su TEMP2 (che viene comandata dal SCO abilitando opportunamente il buffer three state relativo al secondo registro sorgente e abilitando in scrittura il registro TEMP2);

- la specificazione del tipo di operazione da effettuare (che viene fatta dal SCO comandando opportunamente l'ALU);

- la scrittura del risultato nel registro destinatario (che viene effettuata dal SCO abilitando opportunamente il buffer three state in uscita dell'ALU e abilitando in scrittura il registro destinatario).

In questo caso, contrariamente al caso precedente, l'esecuzione temporale delle micro-operazioni deve essere rigidamente controllata, dovendosi serializzare l'accesso sull'unica risorsa in condivisione (che è il bus). Notare come, rispetto alla prima soluzione, ad una riduzione della complessità dell'hardware corrisponde un aumento della complessità temporale per l'esecuzione della stessa operazione elementare a due operandi.

Domanda 7

Quali sono i passi elementari necessari per eseguire una operazione ad un solo operando?

Si può notare che rispetto alla soluzione a bus utilizzata per effettuare il trasferimento dati tra registri interni, le interfaccie e i comandi in lettura e scrittura dei registri non sono cambiati, quindi la struttura che abbiamo determinato permette anche il trasferimento diretto di dati tra registri e questo può essere effettuato ad un singolo passo (ovvero con l'esecuzione di una sola microistruzione del SCO).

Ovviamente tra queste due soluzioni estreme vi sono innumerevoli soluzioni intermedie utilizzanti più bus su cui sono attestati differenti insiemi di registri.

2.3.3 Implementazione della struttura di interconnessione interna del PD32

Poichè il set delle istruzioni del PD32 prevede il trasferimento di al più un dato alla volta, per la struttura di interconnessione interna del processore è sufficiente utilizzare un unico bus (che come abbiamo visto è anche la soluzione più vantagiosa dal punto di vista della complessità hardware).

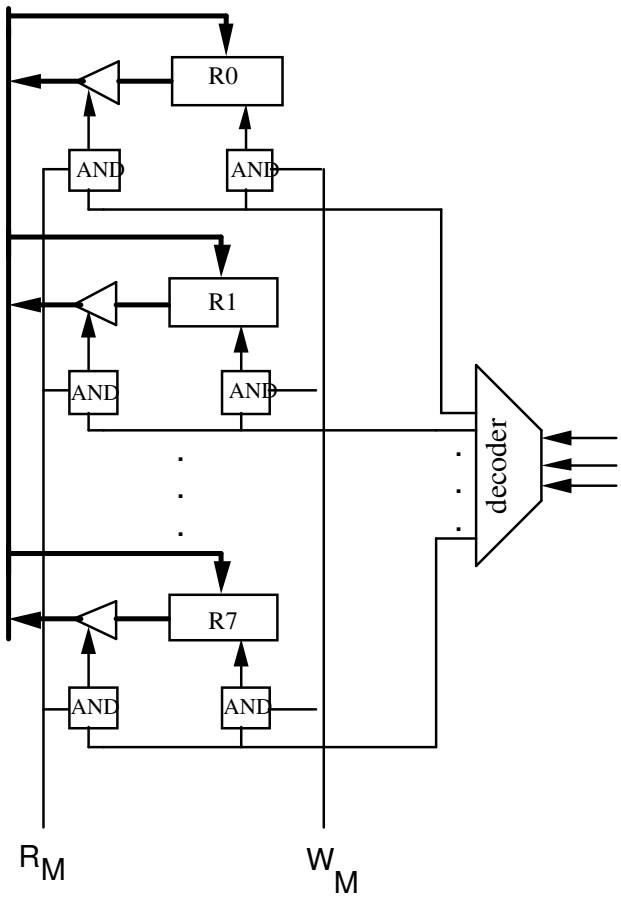


Figura 13: Organizzazione a memoria dei registri R0-R7.

La struttura di interconnessione potrebbe essere ricavata immediatamente, specializzando quella indicata in Figura 12, dove al posto dei generici registri vengono specificati i registri visibili (come R0-R7) e non visibili (come PC, IR) da parte dell'utente. Si può notare, comunque, che limitando la possibilità dei trasferimenti diretti tra i registri R0-R7 (pur permettendo un trasferimento indiretto utilizzando un registro temporaneo) si

può ulteriormente ridurre la complessità del SCA e del SCO del processore. In particolare essendo 8 il numero di tali registri e potendo utilizzarne solo uno alla volta (in lettura o in scrittura) questi registri potrebbero essere organizzati a schiera come una piccola memoria. In questo caso (vedi Figura 13) vi è una sola uscita (ovviamente controllata da un buffer three-state) ed un solo ingresso verso il bus. Per poter selezionare uno dei registri si fa uso di un decoder e per abilitare la scrittura e la lettura si utilizzano due segnali di controllo(W_M e R_M).

Rispetto alla soluzione precedente vi è un risparmio nel numero di buffer three-state ed anche un risparmio del numero dei segnali di controllo che debbono essere generati dal SCO. Infatti da *2n* segnali di controllo di cui *n* per la lettura ed *n* per la scrittura (*n = 8*, in questo caso), si potrebbe passare a $2+log_2n$ segnali di controllo (uno per la lettura, uno per la scrittura e log_2n per la selezione di uno dei registri). Però, ricordando il formato dell'istruzione, si può notare che la codifica dell'indirizzo del singolo registro interessato al trasferimento è memorizzata nel registro istruzione (IR). Quindi il SCO, invece di generare i segnali di controllo relativi alla codifica dell'indirizzo del suddetto registro, può generare solo un segnale di controllo che permette il trasferimento dei bit *0-2* o dei bit *6-8* del IR sul decoder. Per poter effettuare questo trasferimento c'è necessità di un multiplexer a due ingressi di tre bit ciascuno e di un segnale di controllo per la selezione di uno dei due ingressi. In questo caso il numero dei segnali di controllo che deve generare il SCO è pari a *3* (uno per la lettura, uno per la scrittura e uno per la selezione di uno dei due ingressi del multiplexer) e la parte del SCA relativa alla selezione di uno dei registri del banco di memoria interna diviene come quella schematizzata in Figura 14.

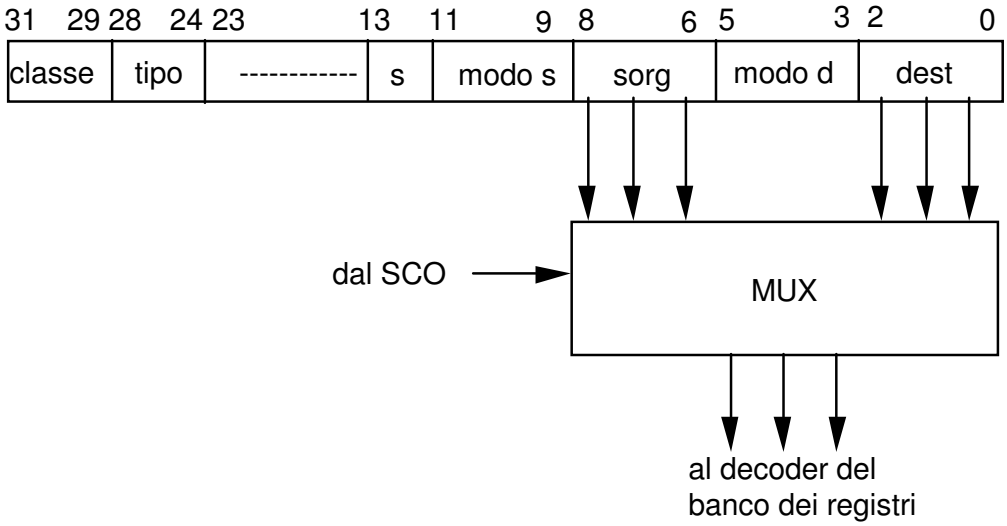


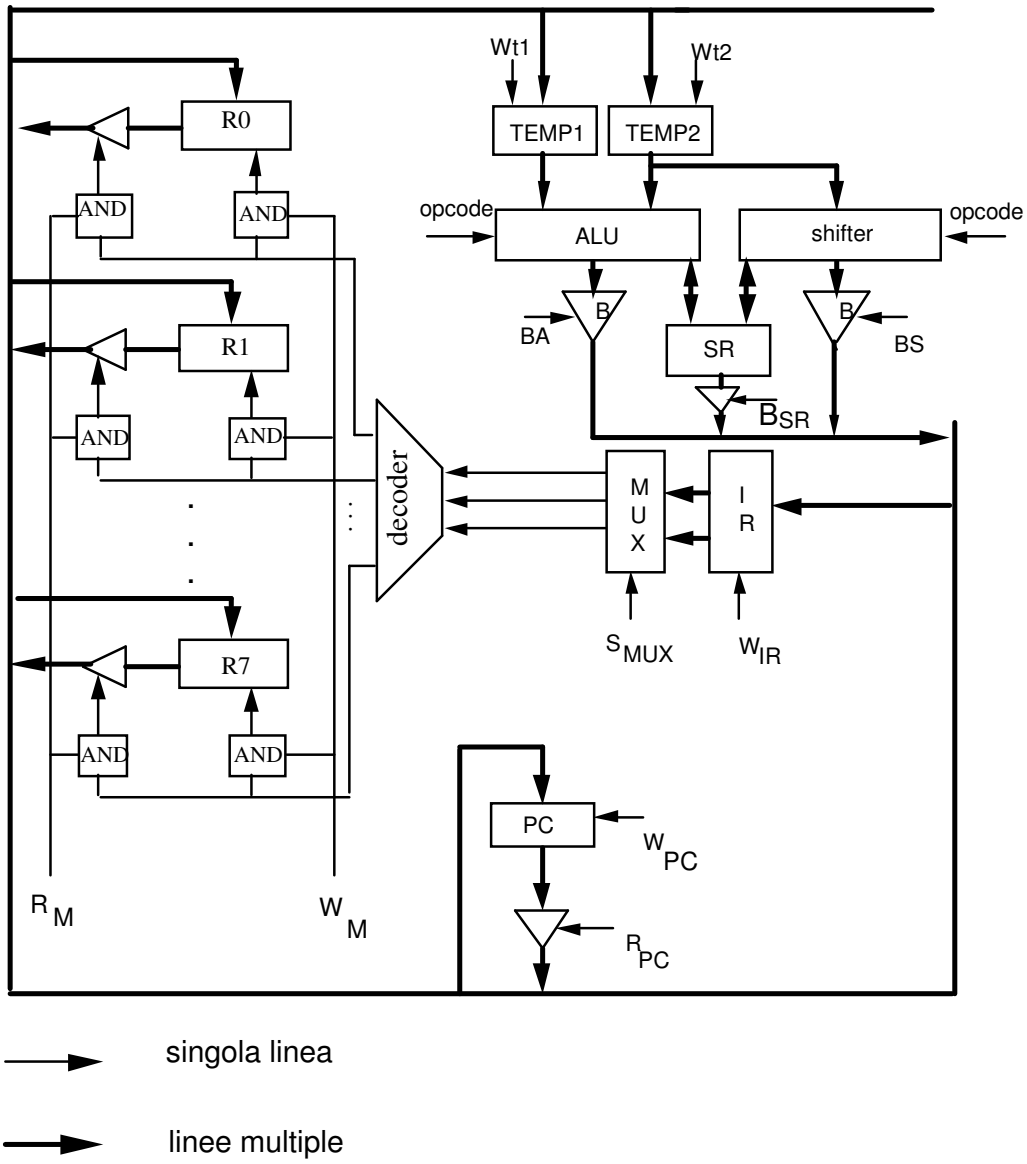
Figura 14: Circuito di selezione del registro sorgente o destinatario

Infine, per permettere il trasferimento indiretto dei dati tra due dei registri R0,. . .,R7 si potrebbe utilizzarne uno tra TEMP1 e TEMP2 come registro tampone. Dalla Figura 12 si può notare che tali registri hanno solo capacità di lettura dal bus e non di scrittura, quindi per poter implementare lo spostamento di dati tra gli otto registri è necessario prevedere che il contenuto di uno dei due registri temporanei possa fluire sul bus. A tal fine o si connette in uscita uno dei due registri temporanei sul bus interno del PD32, oppure si prevede che uno dei due circuiti di calcolo (l'ALU o lo shifter) metta in comunicazione uno dei due suddetti registri con il bus. Poichè tra le funzionalità proprie degli shifter è prevista quella di non operatività (vedi Paragrafo 2.4), operazione che permette il passaggio di ciò che sta a monte dello shifter con quello che vi sta a valle senza modificarne il contenuto, la seconda soluzione è quella a costo inferiore ed è quella normalmente implementata nei processori. Di conseguenza si implementerà nel PD32 la seconda soluzione. Poichè lo shifter può ricevere dati da TEMP2, questo sarà il registro che si userà per effettuare le operazioni di trasferimento dati tra i registri R0,. . .,R7**.**

Quindi l'architettura complessiva del SCA senza considerare le interfacce verso la memoria e i dispositivi di ingresso ed uscita è quella schematizzata in Figura 15.

Esercizio 4

Data l'architettura del SCA specificata in Figura 15, elencare le micro operazioni che deve emettere il SCO per effettuare il trasferimento dati da R1 a R5.



N.B. non sono evidenziate le variabili di condizione che da SR e IR vanno al SCO

Figura 15: Architettura SCA del PD32 senza interfaccie esterne

2.3.4 Interfacciamento del bus interno del PD32 con le unità esterne

2.3.4.1. Interfacciamento con la memoria di lavoro

La memoria di lavoro è organizzata logicamente come un vettore di celle ed ogni cella è costituita da un singolo byte. Le celle sono numerate in sequenza, da *0* a 2^n *-1*, essendo *n* il numero di bit utilizzati per definire un

indirizzo di memoria. Nella memoria sono immagazzinati sia i dati che le istruzioni del programma. Ogni istruzione è costituita da *4* byte, mentre i dati possono essere memorizzati con uno, due o quattro byte. Per questo motivo le celle di memoria possono essere indirizzate (ovvero lette o scritte) singolarmente (byte), due alla volta (parola), quattro alla volta (parola lunga o doppia parola). Gli indirizzi dei dati costituiti da uno, due o quattro byte e quelli delle istruzioni possono essere qualunque.

Avendo a disposizione moduli di memoria RAM organizzati a byte, e volendo poter indirizzare uno, due o quattro byte, è necessario organizzare l'indirizzamento della memoria fisica in modo da poter selettivamente selezionare uno, due o quattro moduli contemporaneamente (ovvero si vuole realizzare una memoria a *parallelismo quattro* utilizzando moduli di memoria di un byte). In Figura 16 è schematizzata con un vettore l'organizzazione logica di una memoria con sedici celle di memoria (ciascuna cella delle dimensioni di un byte), mentre in Figura 17 è schematizzata una possibile organizzazione della stessa memoria di lavoro avendo a disposizione quattro moduli di memoria (ciascuna cella delle dimensioni di un byte). In entrambe le figure sono evidenziati gli indirizzi di ogni singola cella.

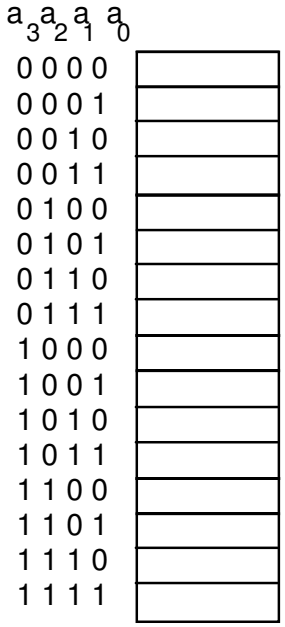


Figura 16: Organizzazione logica a vettore di 16 celle di memoria

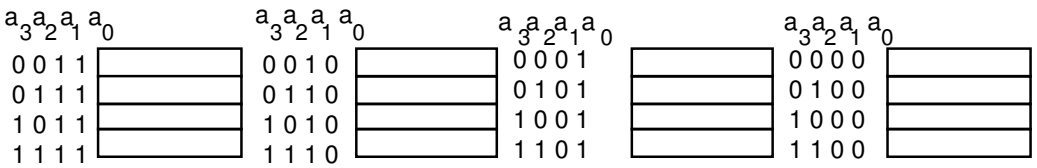


Figura 17: Un'organizzazione a 4 moduli di 16 celle di memoria

Dalla Figura 17 si può notare che le celle di memoria di ogni singolo modulo hanno un indirizzo la cui parte meno significativa è uguale per tutte le celle di memoria del modulo (per esempio il primo modulo ha celle di memoria caratterizzate da un indirizzo pari a --00, dove - indica don't care conditions); ciò sta a significare che esiste un *indirizzo di modulo* che è rappresentato dai due bit meno significativi. Inoltre si può notare che le celle i-esime di ogni singolo modulo hanno i primi due bit identici e pari alla codifica in binario del valore di i - 1 (*indirizzo di riga*). Per esempio l'indirizzo della seconda cella di ogni singolo modulo è pari a 01- -, mentre l'indirizzo della quarta cella è pari a 11--. Ed è proprio questo tipo di organizzazione degli indirizzi che permette di indirizzare contemporaneamente i quattro moduli di memoria.

Domanda 8

Nel caso di una memoria logica costituita da 32 celle di memoria organizzata con 4 moduli di memoria da un byte, di quanti bit è l'indirizzo di modulo?

I moduli di memoria RAM statica sono normalente costituiti da un certo numero di matrici di celle che contengono un bit di informazione (quindi ci saranno 8 matrici se si desidera avere una memoria a byte), da due decoder di selezione per accedere alle celle e da un amplificatore di ingresso ed uscita per adattare le tensioni interne con quelle esterne. Per poter funzionare tali moduli necessitano (vedi Figura 18):

- di linee di ingresso in cui specificare l'indirizzo della cella interessata al trasferimento, il cui numero dipende dalle dimensioni del singolo modulo;

- di linee su cui trasmettere/ricevere il dato da scrivere/leggere nella/dalla cella di memoria indirizzata (8 linee nel caso di memoria a byte);

- un segnale di controllo di ingresso per la lettura (RD);

- un segnale di controllo di ingresso per la scrittura (WR).

Oltre agli ingressi di cui sopra, ingressi che sono sufficienti per una corretta utilizzazione dei moduli di memoria, normalmente è presente un altro ingresso che abilita il funzionamento dei moduli di memoria (Chip Select, CS) e che permette una utilizzazione più flessibile degli stessi.

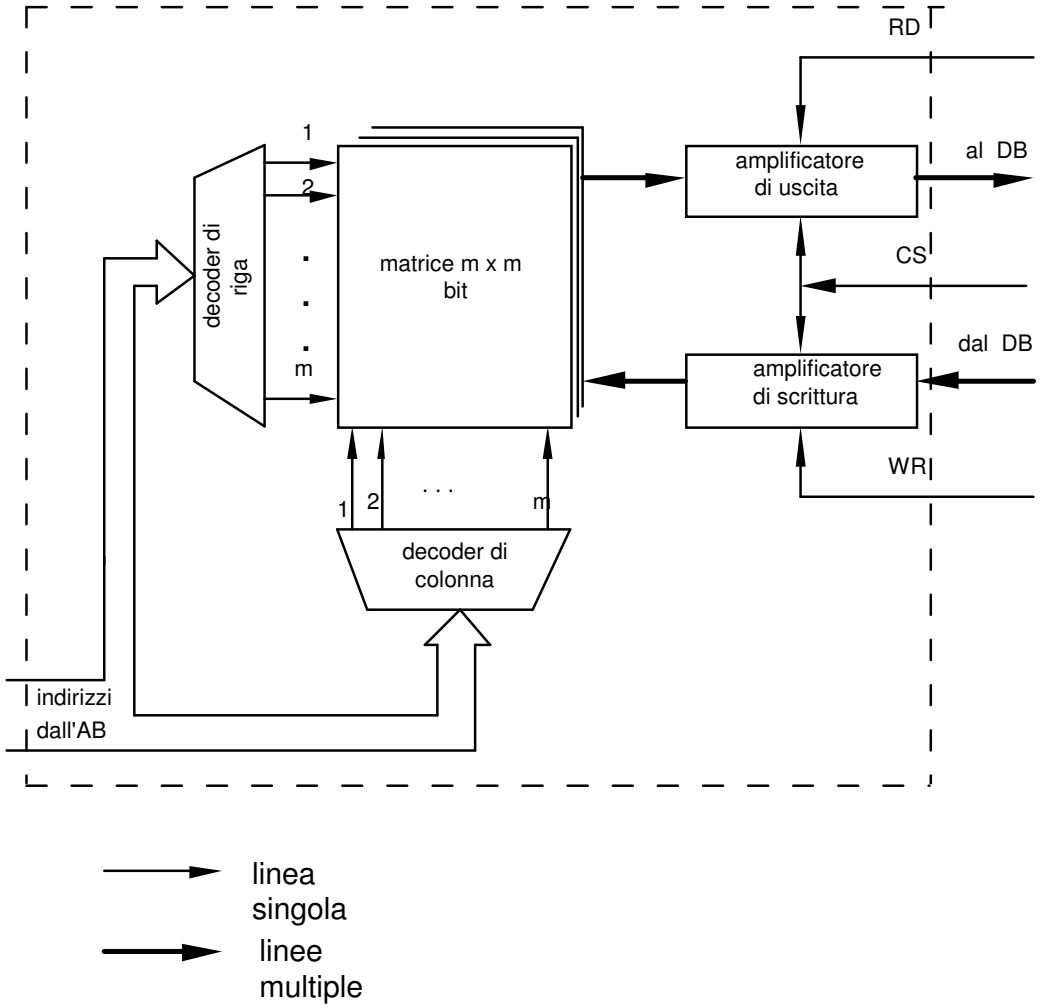


Figura 18: Modulo di memoria RAM statica ad un byte.

Per utilizzare questi moduli (che sono entità passive, cioè asservite all'utilizzatore) è necessario che il SCO del PD32 generi dei segnali di controllo che abilitano la lettura o la scrittura dei dati (MRD e MWR) (segnali che vanno connessi agli ingressi RD e WR del modulo), che si utilizzino un insieme di linee per il trasferimento dei dati dal bus interno del PD32 ad uno dei registri della memoria e viceversa (Data Bus, DB) e un insieme di linee dal processore al modulo di memoria per selezionare una delle celle della memoria (Address Bus, AB).

Ipotizzando, quindi, la presenza di un Data Bus e di un Address Bus l'organizzazione del banco della memoria in cui ci sono quattro moduli è schematizzata in Figura 19.

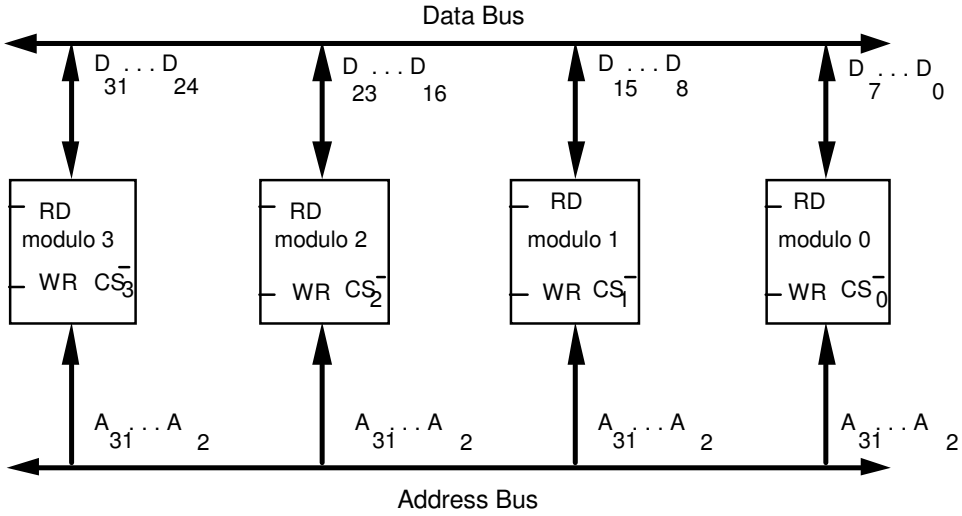


Figura 19: Organizzazione del banco di memoria con quattro moduli.

Da notare che ogni singolo modulo di memoria è organizzato a byte, mentre i registri interni del PD32 sono da 32 bit, quindi per consentire a quattro moduli di memoria di leggere/scrivere contemporaeamente dati sul DB è necessario che questi siano connessi a quattro insiemi distinti di linee del DB. Per esempio il modulo 0 è connesso alle linee 0-7 del DB, mentre il modulo 3 è connesso alle linee 24-31.

Inoltre è da notare che nell'indirizzare un singolo byte è necessario selezionare il modulo di memoria relativo. A tal fine i chip select (CS) potrebbero essere generati usando i bit meno significativi dell'Address Bus (A_1, A_0). Però un chip select ricavato solo dai bit meno significativi dell'AB può creare dei problemi nel caso di indirizzamento di una parola o di una doppia parola. Infatti osservando la Figura 17 può accadere che una parola o una doppia parola possono essere memorizzate in byte con indirizzo di riga diverso. Per esempio in Figura 20 è evidenziata una parola di indirizzo 0011 che è costituita da due byte di cui il primo di indirizzo 0011 e il secondo 0100 (nell'indirizzo 0011 vi è il byte meno significativo del dato, mentre nell'indirizzo 0100 il byte più significativo del dato). Mentre in Figura 21 è evidenziata una doppia parola di indirizzo 0010, costituita da quattro byte di indirizzo 0010, 0011, 0100 e 0101 (nell'indirizzo 0010 vi è il byte meno significativo del dato, mentre nell'indirizzo 0101 il byte più significativo del dato).

Per accedere alla parola di figura 20 è necessario poter indirizzare sia il byte di indirizzo 0011 che quello di indirizzo 0100. Però data l'organizzazione della memoria, così come schematizzata in Figura 19, in parallelo si può accedere solo byte con lo stesso indirizzo di riga. Quindi per poter accedere ai due suddetti byte il SCO del PD32 deve generare due indirizzi di riga successivi e leggere/scrivere dal/sul DB solo il byte di interesse (la prima volta il byte 0011 e la seconda volta il byte 0100). Dopodichè, nel caso di

lettura della parola dalla memoria è necessario che il SCO ricomponga opportunamente i due byte per formare la parola prima di poterla utilizzare. La stessa problematica si solleva nel caso di indirizzamento di una doppia parola memorizzata come in Figura 21.

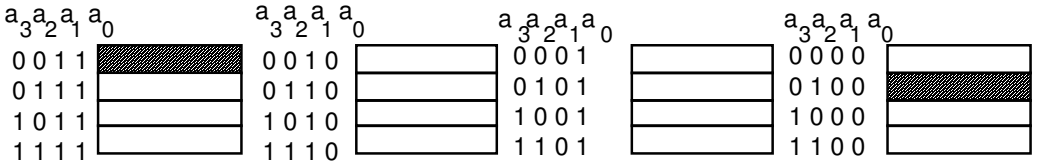


Figura 20: Esempio di parola con byte di indirizzo di riga diverso.

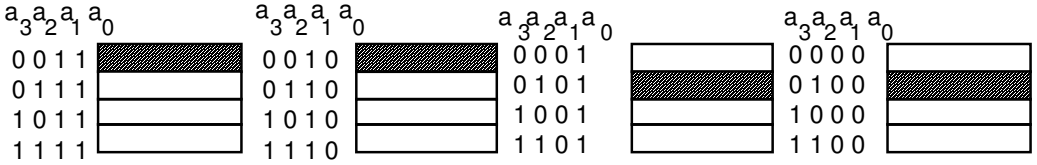


Figura 21: Esempio di parola doppia con byte di indirizzo di riga diverso.

Per poter selezionare opportunamente i moduli di memoria relativi ai byte con lo stesso indirizzo di riga si sono previsti quattro segnali di controllo: Mb_0, Mb_1, Mb_2, Mb_3. Questi segnali di controllo vengono connessi con i chip select (CS) dei relativi moduli di memoria.

Per esempio, per accedere alla parola di figura 20, la prima volta il SCO del PD32 abilita la scrittura sull'AB dell'indirizzo di riga contenuto sul MAR (che nel caso particolare sarà pari a 00, ovvero $A_3 = 0$ e $A_2 = 0$) e genera il segnale Mb_3, mentre la seconda volta il SCO del PD32 abilita sempre la scrittura sull'AB dell'indirizzo di riga contenuto sul MAR (che questa volta sarà pari a 01, ovvero $A_3 = 0$ e $A_2 = 1$) e genera il segnale di controllo Mb_0.

Da notare, comunque, che nel caso di accesso di quattro byte allineati sullo stesso indirizzo di riga (relativi o ad un dato o ad un'istruzione) il SCO del PD32 deve generare un solo indirizzo ed abilitare contemporaneamente i quattro moduli di memoria. In Figura 22 è evidenziata un'informazione di quattro byte di indirizzo 1000.

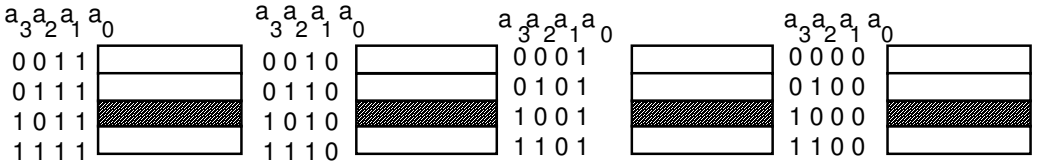


Figura 22: Esempio di memorizzazione di una informazione di quattro byte allineati sullo stesso indirizzo di riga.

Domanda 9

Dato lo schema di Figura 19, quanti byte possono essere indirizzati?

Per semplicità di esposizione sono stati previste due linee di controllo indipendenti per comandare la lettura e la scrittura, quando invece sarebbe sufficiente un solo segnale di controllo; infatti, è da notare che un singolo modulo di memoria o è in lettura o è in scrittura quindi si potrebbe usare un segnale che in forma diretta abilita un tipo di operazione mentre in forma complementata ne abilita l'altro. I segnali di controllo MRD e MWR vengono generati dal SCO ed indicano se si deve eseguire una istruzione di lettura o di scrittura in memoria. L'indirizzo della cella (delle celle) di memoria a cui si vuole accedere è un operando dell'istruzione stessa, di conseguenza è necessario prevedere un registro interno al SCA del PD32 in cui memorizzare la codifica dell'indirizzo di tale cella di memoria. Questo registro, anch'esso a 32 bit, e denominato Memory Address Register (MAR) viene usato per pilotare le linee dell'AB. Inoltre per permettere il trasferimento dati da/a uno dei registri interni del processore a/da un registro di memoria indirizzato si è previsto di utilizzare un registro tampone (MDR). Per consentire il trasferimento dei dati tra l'MDR ed un registro di memoria è necessario connetterli fisicamente; ciò viene fatto tramite le linee del DB. Per pilotare la direzione del trasferimento e per separare elettricamente la parte del SCA interna al PD32 dal mondo esterno (a tal fine vedi la gestione delle richieste di rilascio dei bus esterni) si fa uso di un buffer three-state bidirezionale. Da notare che anche l'uscita del MAR potrebbe essere messa ad alta impedenza e quindi anche la sua uscita verso l'AB è pilotata da un buffer three-state.

Quindi l'interfaccia del SCA del PD32 con la memoria può essere schematizzata come in Figura 23.

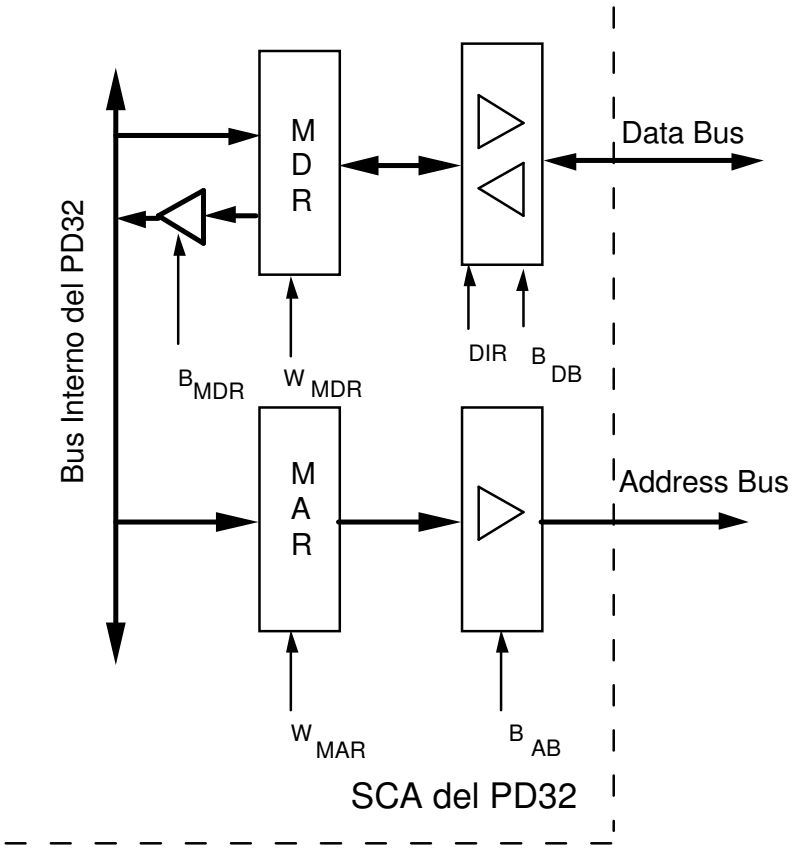


Figura 23: Interfaccia della SCA interna del PD32 con la memoria

E' da notare che, nel caso di lettura/scrittura di dati dalla/sulla memoria di lavoro della dimensione di uno o due byte, i byte prima di essere utilizzati devono essere opportunamente manipolati. Per esempio i dati da un byte sono memorizzati nei registri interni del PD32 nei bit meno significativi (0-7), mentre i rimanenti bit contengono il valore dell'ottavo bit (l'estensione del segno viene fatto per facilitare le operazioni aritmetiche). La cella di memoria interessata al trasferimento può, invece, essere localizzata in uno qualunque dei moduli di memoria, i quali sono interfacciati su quattro differenti insiemi di linee del data bus. Quindi, per esempio nel caso di lettura di un byte memorizzato sul quarto modulo di memoria i dati da leggere sono disponibile nei bit (D_{24}-D_{31}) del Data Bus, mentre vanno memorizzati nella parte meno significativa di uno dei registri interni. Per il riallineamento dei bit e l'estensione del segno si usa lo shifter (vedi Sezione 2.4).

Infine è da ricordare che nel caso di lettura/scrittura dalla/sulla memoria di lavoro di parole o doppie parole non allineate sullo stesso indirizzo di riga il SCO del PD32 deve effettuare due accessi sequenziali alla memoria di lavoro.

Esercizio 5

Indicare una possibile organizzazione di una memoria di lavoro costituita da quattro banchi di memoria da un byte ciascuno ed in cui è possibile prelevare un solo byte alla volta.

2.3.4.2 Interfacciamento con le porte di ingresso ed uscita

Le porte di ingresso ed uscita sono viste dal PD32 come dei singoli registri indirizzabili individualmente. Di conseguenza per poter effettuare un trasferimento dati dal PD32 ad una porta esterna e viceversa è necessario connettere fisicamente un registro interno del PD32 con quello della porta esterna selezionata ed abilitare il trasferimento dei dati. Quindi l'interfacciamento tra le porte di ingresso ed uscita e la parte del SCA interna al PD32 è del tutto simile a quella che abbiamo visto precedentemente tra la memoria e il PD32. La parte di SCA del PD32 relativa all'interfacciamento con le porte esterne è schematizzata in Figura 24.

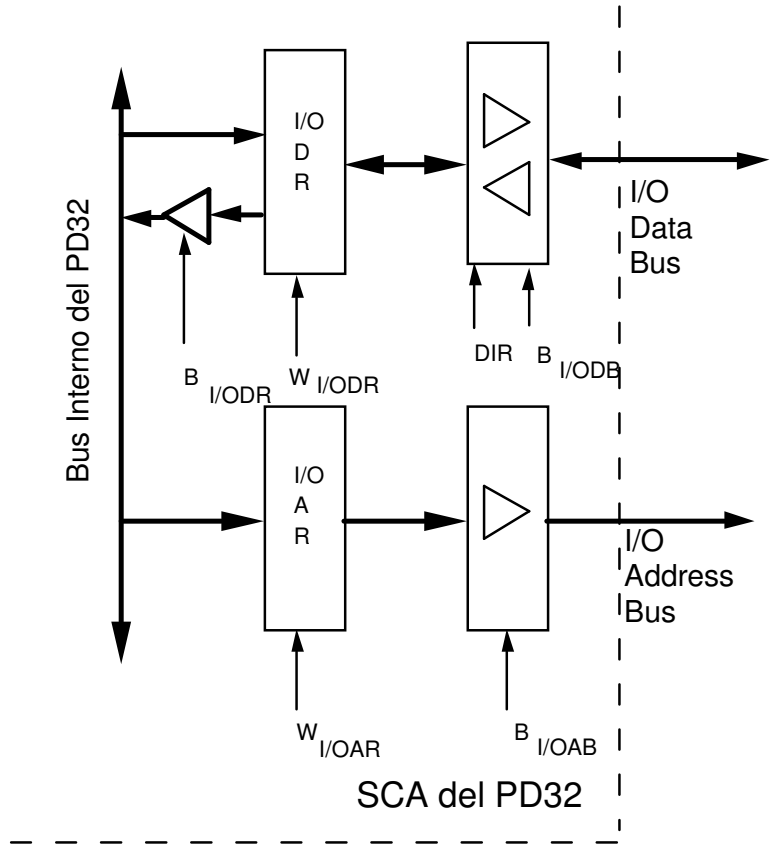


Figura 24: Interfaccia della SCA interna del PD32 con le porte di I/O.

In Figura 24 il registro I/OAR è quello adibito alla memorizzazione della codifica dell'indirizzo della porta interessata al trasferimento (indirizzo che è l'operando dell'istruzione macchina di I/O da interpretare). Poichè nella codifica delle istruzioni si è previsto che il numero di porte indirizzabili in ingresso ed in uscita è pari a 256 il registro I/OAR è di 8 bit. Il registro I/ODR è il registro tampone con funzioni di interfaccia verso il bus interno del PD32. Anche in questo caso per consentire il trasferimento dei dati tra l'I/ODR ed il registro della porta specificata è necessario connetterli fisicamente; ciò viene fatto tramite le linee del I/ODB. Per pilotare la direzione del trasferimento e per separare elettricamente la parte del SCA interna al PD32 dal mondo esterno si fa uso di un buffer three-state bidirezionale. Da notare che anche l'uscita del I/OAR potrebbe essere messa ad alta impedenza e quindi anche la sua uscita verso l'I/OAB è pilotata da un buffer three-state.

Da notare che il segnale di controllo che il SCO deve generare per effettuare una lettura di un dato da una porta di ingresso è I/ORD, mentre il segnale di controllo per effettuare una scrittura di un dato su una porta di uscita è I/OWR. Per quanto riguarda i protocolli e le velocità di trasferimento dei dati si rimanda al successivo Paragrafo 3.4.

Domanda 10

Se il registro I/OAR fosse di 16 bit, quante periferiche si potrebbero indirizzare?

Domanda 11

Si potrebbe eliminare il registro I/ODR?

2.4 Circuiti di calcolo

2.4.1. Shifter

Di seguito verrà presentato uno "shifter" (conosciuto come *barrel shifter*) che permette di eseguire operazioni di rotazione e scorrimento di *k* bit (scorrimento che può essere logico o aritmetico), dove *k* può assumere il valore compreso tra *0* ed *n -1* (dove *n* è il numero di bit). Da notare che per *k = 0* non si prevede alcun spostamento di bit (questa possibilità permette tra l'altro il passaggio di dati tra i registri interni del PD32 R0,. . .,R7, vedi Paragrafo 2.3.3).

Lo shifter è un modulo con *n* bit in ingresso (i bit da manipolare, a_n,. . .,a_1) ed n bit di uscita (b_n,. . .,b_1, i bit manipolati dall'operazione richiesta allo shifter), di seguito *n* sarà posto pari a *32*. Per specificare allo shifter il tipo di operazione da eseguire e il numero di posizioni dello spostamento si useranno dei segnali di controllo. Quindi necessitano $log_2 n$ bit per specificare il numero di spostamenti richiesti; *2* segnali di controllo per specificare il tipo di operazione (rotazione, scorrimento logico o aritmetico) ed *1* per specificare se i bit debbono essere spostati a destra o a sinistra. E' da ricordare che le operazioni di rotazione e scorrimento fanno uso del flag di carry (C) dello Status Register (SR), per questo motivo vi deve essere una interconnessione tra lo "shifter" e il suddetto flag .

Di seguito per dare un'idea dell'organizzazione, del numero di porte logiche elementari e del tempo di esecuzione delle operazioni si descriverà una rete combinatoria che implementa un'operazione di *spostamento logico a destra o a sinistra di k posti*. Per esempio, nel caso di spostamento a destra, questa operazione prevede:

$$b_{n-i} = 0 \quad (per\ 0 \le i < k)$$
$$b_i = a_{i+k}\ (per\ 1 \le i \le n - k)$$
$$C = a_k$$

operazione che può essere specificata come in Figura 25.

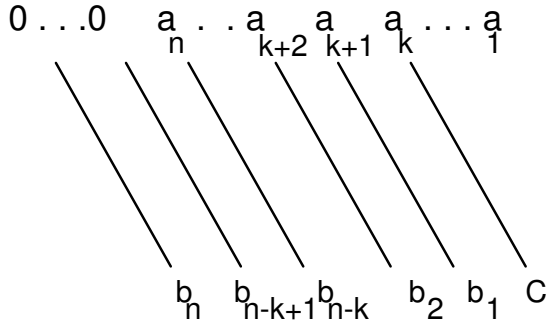


Figura 25: Spostamento logico a destra di k posti.

In Figura 26 è schematizzata una rete logica che permette ogni spostamento logico a destra/sinistra utilizzando $log_2\ n$ stadi. Il numero di porte per ogni stadio è propozionale ad n . Oltre agli ingressi a_i *(i=1,. . .,n)*, sono presenti $log_2 n$ ingressi di controllo che codificano il numero (*k*) degli spostamenti ($c_1,. . .,c_{log_2 n}$) ed un bit *d* di direzione che specifica spostamento a destra o a sinistra (d=0 a destra, d=1 a sinistra).

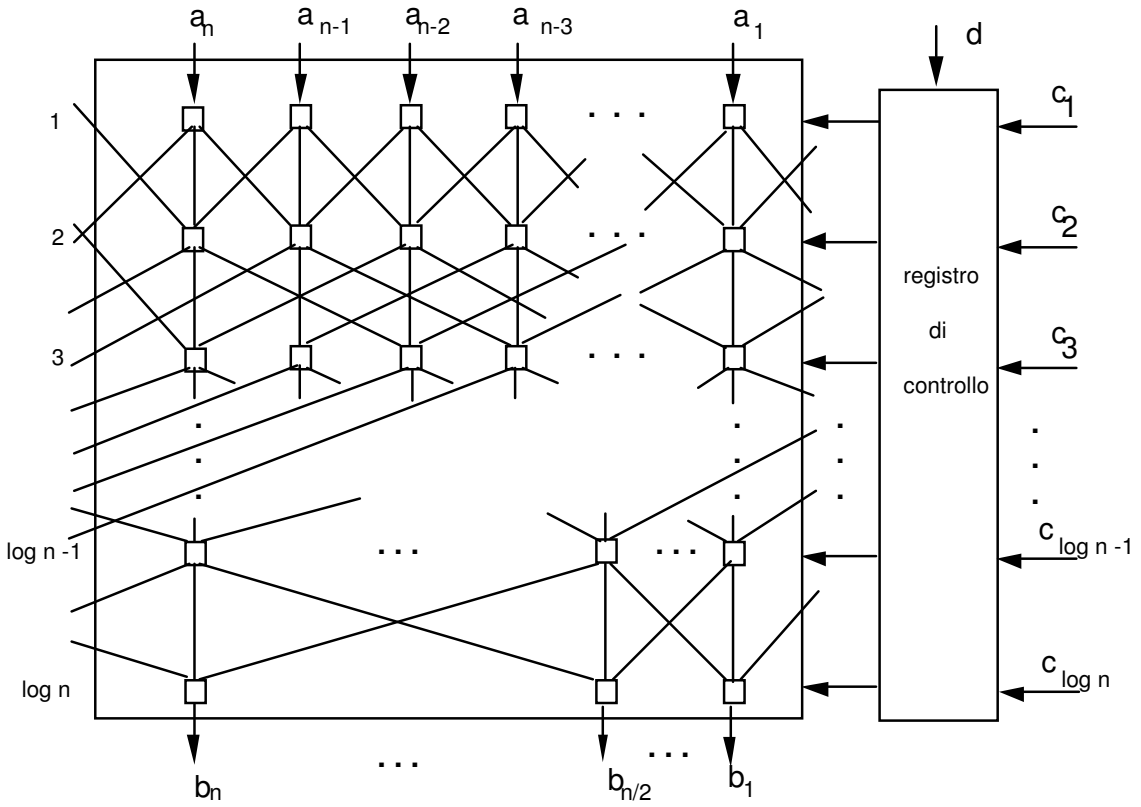


Figura 26: Schema di principio di un "barrell shifter".

I bit di ingresso al primo stadio possono essere spostati di zero od una posizione (secondo se $c_1 = 0$ od *1*), le uscite del primo stadio sono gli ingressi al secondo stadio. In questo stadio, però a differenza del primo, i bit di ingresso possono essere sposati di zero o di due posizioni (secondo se $c_2 = 0$ od *1*), e così via per gli stadi successivi. In particolare nello stadio *i-esimo* i bit possono essere spostati di *0* o 2^{i-1} posizioni (secondo se $c_i = 0$ od *1*). Quindi nello stadio $log_2 n$ i bit possono essere spostati di *0* o $2^{log_2 n - 1} = n/2$ posizioni.

Ogni cella della rete può essere implementata con una porta OR e tre porte AND come specificato in Figura 27.

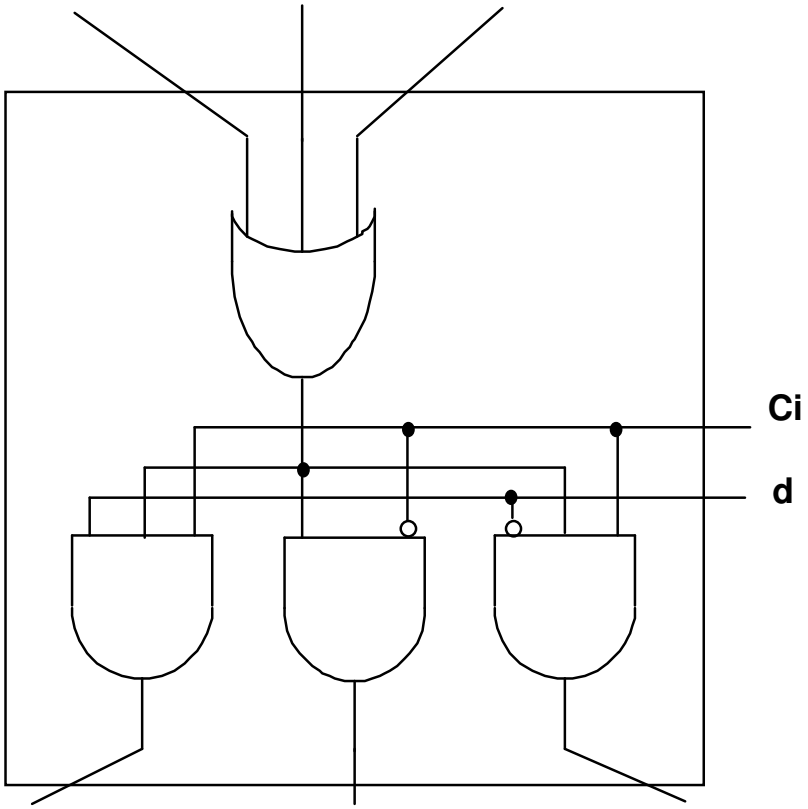


Figura 27: Schema logico della cella elementare dello shifter

Da notare che essendovi per ogni stadio due elementi di ritardo, l'intera operazione di spostamento richiede $2log_2 n$ ritardi, indipendentemente dal valore di k.

2.4.2 Unità Logico Aritmetica (ALU)

Come dice il suo nome l'unità logico aritmetica è il circuito di calcolo che effettua le operazioni di tipo logico ed aritmetico (vedi il set delle istruzioni). Le operazioni logico/aritmetiche che l'ALU del PD32 può effettuare sono la somma (ADD), la somma con carry (ADC), il confronto (CMP),la negazione aritmetica (NEG), la sottrazione (SUB), la sottrazione con riporto (SBB), l'and logico (AND), l'or logico (OR), lo xor (XOR) e la negazione logica (NOT).

Le istruzioni logiche/aritmetiche operano su uno o due operandi. Poichè per la struttura di interconnessione interna del PD32 si è optato per un

singolo bus, nel Paragrafo 2.3.2 si è visto che per poter effettuare tali operazioni è necessario utilizzare due registri temporanei (TEMP1 e TEMP2).

L'esecuzione di un operazione a due operandi prevede:

- la scrittura del primo operando su TEMP1 (che viene comandata dal SCO abilitando opportunamente il buffer three state relativo al primo registro sorgente e abilitando in scrittura il registroTEMP1);

- la scrittura del secondo operando su TEMP2 (che viene comandata dal SCO abilitando opportunamente il buffer three state relativo al secondo registro sorgente e abilitando in scrittura il registro TEMP2);

- la specificazione del tipo di operazione da effettuare (che viene fatta dal SCO comandando opportunamente l'ALU);

- la scrittura del risultato nel registro destinatario (che viene effettuata dal SCO abilitando opportunamente il buffer three state in uscita dell'ALU e abilitando in scrittura il registro destinatario).

Ovviamente l'esecuzione di ogni singola istruzione può comportare l'utilizzazione e la modifica di uno o più flag di stato, per questo motivo l'ALU deve essere connessa in lettura ed in scrittura con i flag del registro di stato (SR).

3. IMPLEMENTAZIONE DEL SOTTOSISTEMA DI CONTROLLO

3.1 Richiamo alla microprogrammazione

Come visto in Sezione 1 la struttura del SCO dipende fondamentalmente dall'organizzazione del SCA. Ciò è dovuto al fatto che l'architettura del SCA influenza sia il numero di segnali di controllo, sia la loro sequenza temporale che il numero di operazioni eseguibili in parallelo. Poichè il modello di riferimento scelto per il processore è quello proposto da Von Neumann, si può eseguire una istruzione alla volta, ciò ha comportato la definizione di una SCA con un basso investimento hardware (per esempio per la struttura di interconnessione interna al PD32 si è optato per un singolo bus). Come vedremo queste scelte influenzeranno notevolmente l'organizzazione del SCO e le sue modalità di funzionamento.

Innanzittutto data la quantità delle operazioni da effettuare e delle relative sequenze temporali (che sono proporzionali al numero di stati) è opportuno progettare il SCO tramite la *microprogrammazione*.

Per eseguire un programma memorizzato nella memoria di lavoro un processore, che segue il modello di Von Neumann, deve far corrispondere ad ogni singola istruzione macchina l'esecuzione di un *microprogramma* del SCO (microprogramma memorizzato nella memoria del SCO). Il processore quindi è una *struttura logica* che ha la funzione di *tradurre* programmi scritti in linguaggio macchina in programmi scritti in linguaggio microprogrammativo e questa traduzione è di tipo i*nterpretativo*. Interpretativo perchè ad ogni istruzione macchina si fà corrispondere una serie di attività del SCA.

Quindi per ogni istruzione da interpretare il SCO esegue un *microprogramma*; microprogramma che è costituito da un insieme di *microistruzioni* il cui numero dipende del numero di *microoperazioni* che debbono essere effettuate e dal loro sequenziamento. Ed è proprio il numero delle microoperazioni possibili e il numero massimo di microoperazioni che debbono essere eseguite in parallelo che determina la possibilità di optare per una organizzazione cosiddetta *orizzontale, verticale* o *diagonale* del *microcodice*.

Infatti la scelta della struttura del microcodice è solitamente una soluzione di compromesso tra due esigenze contrastanti: buona flessibilità e potenza operativa del microcodice e contenuta occupazione della memoria dei microprogrammi. E' possibile quindi immaginare strutture del microcodice che privilegino l'uno o l'altro dei requisiti di compattezza e potenza operativa del microcodice: se si privilegia unicamente la compattezza del microcodice, allora le microistruzioni vengono fortemente codificate e limitate nelle capacità di diramazione, e si parla di struttura *verticale* del microcodice; se si massimizza la potenza espressiva del microcodice, ovvero si tende alla

massima parallelizzazione delle microoperazioni e massima flessibilità nel sequenziamento degli indirizzi, allora si parla di struttura *orizzontale* del microcodice. Soluzioni intermedie vengono dette *diagonali*.

E' ovvio che una impostazione orizzontale è giustificata dall'adozione di un SCA ad elevato parallelismo operativo; in caso contrario le potenzialità di una microistruzione a formato orizzontale saranno sottoutilizzate.

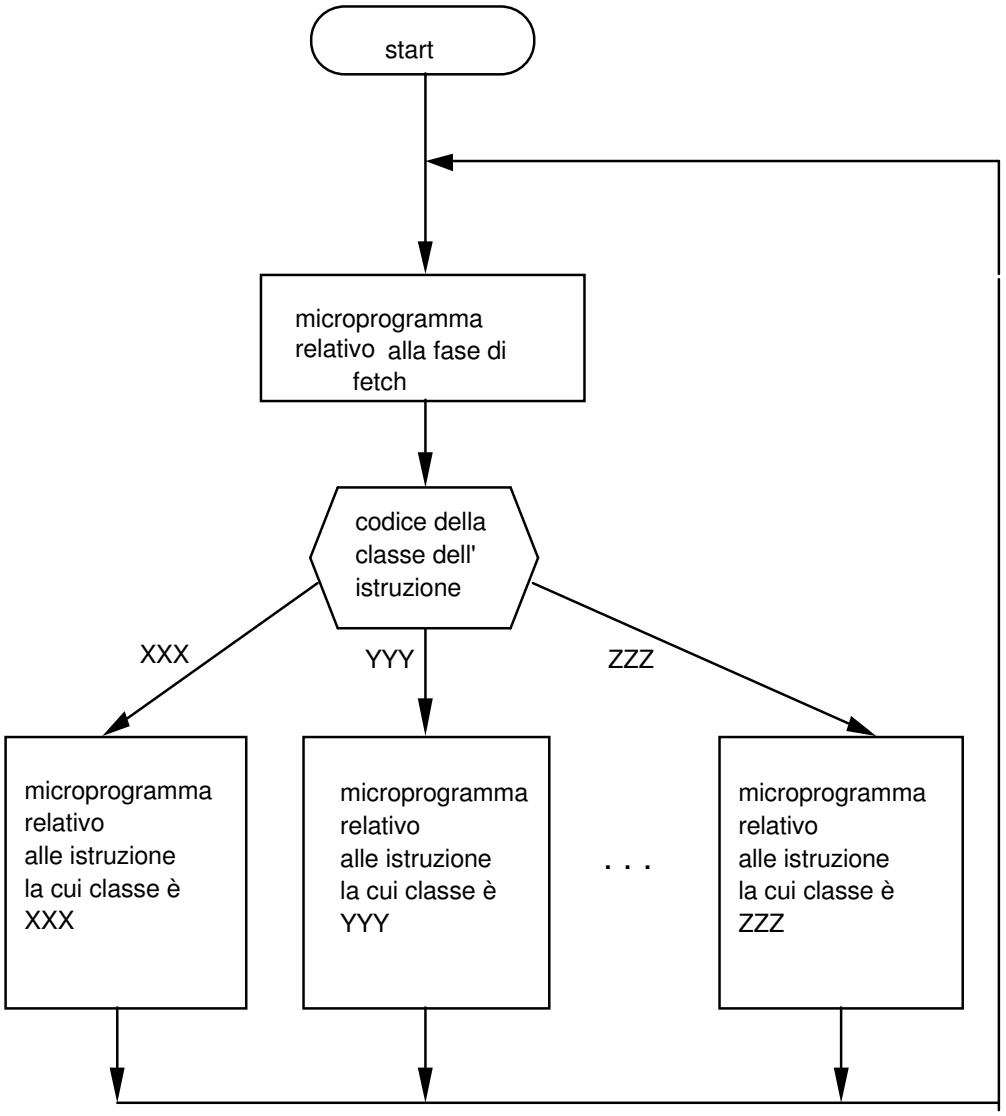


Figura 28: Una possibile organizzazione del micro codice del SCO.

Per poter interpretare un qualsiasi programma il SCO deve essere strutturato in modo da far corrispondere ad ogni istruzione macchina una o più procedure sequenziali (cioè più microprogrammi distinti). A tal fine il SCO può essere implementato con un'unica memoria ROM considerando i vari microprogrammi come tante sottomacchine di un'unica macchina

sequenziale. La scelta del microprogramma da eseguire dipende dalla classe dell'istruzione da eseguire. Considerando che una istruzione è memorizzata nella memoria di lavoro, prima di intraprendere le attività connesse all'interpretazione dell'istruzione stessa è necessario prelevare il codice dell'istruzione e metterla in un registro interno del processore e rendere disponibile il codice dell'istruzione stessa al SCO (questa fase si chiama *fetch* dell'istruzione). Il registro abilitato a tali funzioni è l'IR (Instruction Register). Da notare che ogni singola istruzione macchina è caratterizzata da un codice di classe, ed è proprio questo codice che può essere utilizzato per selezionare l'indirizzo del microprogramma da eseguire. Nel caso in cui non si debba prevedere la gestione di eventi asincroni (Paragrafo 3.4) il codice può quindi essere organizzato come in Figura 28.

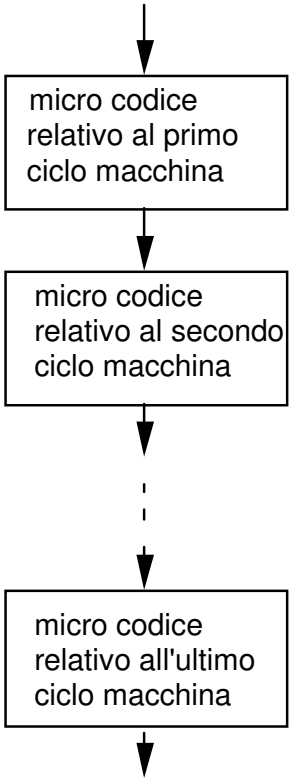


Figura 29: Organizzazione del microprogramma relativo ad una generica istruzione macchina.

L'esecuzione del microprogramma relativa ad una istruzione viene detta *ciclo istruzione*. Ogni ciclo istruzione è composto da una o più fasi elementari che comporteranno l'attivazione di comandi (*micro operazioni*) relativi ad unità interne al processore (registri, ALU, shift register) e/o relative ad unità esterne allo stesso (memoria, unità periferiche). Le fasi elementari che comportano interazione con le unità esterne vengono anche dette *cicli macchina*. Ogni istruzione è caratterizzata dal relativo numero dei

cicli macchina, e il micro codice relativo all'interpretazione dell'istruzione può essere schematizzato come in Figura 29.

Per permettere l'esecuzione del microprogramma relativo al codice dell'istruzione macchina è necessario utilizzare il codice della classe dell'istruzione come ingresso del SCO.

Quindi il SCO avrà tanti ingressi quante sono :

- le variabili di condizione provenienti dalla parte di SCA esterna al processore;

- le variabili di condizione provenienti dalla parte di SCA interna al processore, quali il contenuto dei flag di stato (memorizzati nel registro SR), la classe ed il tipo dell'istruzione oltre che le modalità di indirizzamento degli operandi dell'istruzione stessa (che sono memorizzati nell'IR).

Il SCO avrà tante uscite per quanti sono i segnali di controllo necessari per comandare la parte di SCA interna ed esterna al PD32.

L'organizzazione del SCO dipenderà dal costo implementativo e dalle prestazioni che si vogliono ottenere, oltre che dal modello di macchina sequenziale scelta (di tipo Mealy o Moore).

Se non si vuole minimizzare il costo si possono adottare le organizzazioni schematizzate nelle dispense sulla microprogrammazione, in cui il dimensionamento della ROM è fatto tenendo in conto di tutte le variabili di condizione (ingresso), di tutti i segnali di controllo (uscite) e utilizzando un numero di variabili di stato sufficiente a codificare tutti i possibili stati. Nel caso più semplice, in cui si voglia evitare di utilizzare del codice rientrante, il numero di stati del SCO sarà pari al numero di stati della fase di fetch più il numero di stati delle sottomacchine relative alle operazioni macchina da interpretare. Quindi, indicando con N_0 il numero di stati della fase di fetch e con N_i il numero massimo di stati relativi all'*i-esima* classe delle istruzioni macchina (per $i = 1, \ldots, M$, se M è il numero delle classi), il numero di variabili di stato (k) è pari a:

$$k = \left\lceil \log_2 \sum_{0}^{M} N_i \right\rceil$$

Quindi indicando con *m* il numero delle variabili di ingresso e con *r* il numero di variabili di uscita, nel caso di SCO implementata secondo il modello di Mealy le dimensioni della ROM sono pari a 2^{m+k} parole ognuna di $k+r$ bit; mentre nel caso di SCO implementata secondo il modello di Moore vi avrà necessita di una ROM di 2^k parole ognuna di $k2^m+r$ bit.

Si può notare che, essendo il numero delle variabili di condizione, di controllo e di stato dell'ordine delle decine, le dimensioni delle ROM in entrambi i casi sono notevoli e quindi di costo proibitivo. Per questo motivo c'è la necessità di andare ad una riorganizzazione del SCO, evitando di non penalizzarne le prestazioni. Di seguito si farà vedere come si può avere un risparmio riorganizzando opportunamente gli ingressi e le uscite della ROM. Infatti ad una riduzione del numero degli ingressi corrisponde una riduzione del numero delle parole (ovvero del numero delle microistruzioni) della ROM, mentre ad una riduzione del numero delle uscite corrisponde una riduzione delle dimensioni di ogni singola parola della ROM.

Un primo risparmio può essere ottenuto notando che le microistruzioni relative all'interpretazione di una istruzione macchina possono essere allocate fisicamente in modo adiacente. Quindi si potrebbe utilizzare il codice della classe dell'istruzione (che è un sottoinsieme delle variabili di condizione) come indirizzo base del blocco di microistruzioni. Di conseguenza per ogni microistruzione invece di tenere codificato tutto l'indirizzo della microistruzione successiva (delle microistruzioni successive, nel caso di utilizzazione del modello di Moore), è sufficiente tenere codificata solo la porzione relativo allo spiazzamento. In questo caso è importante notare che il numero di bit necessari allo spiazzamento deve essere dimensionato sul microprogramma, tra tutti quelli relativi ai codici delle istruzione macchina e della fase di fetch, di dimensione massima.

Il numero di microprogrammi distinti è pari al numero delle differenti classi di istruzioni macchina più uno (quello relativo alla fase di fetch). Quindi se *M* è il numero dei differenti classi di istruzioni macchina il numero di bit del registro base è pari a:

$$\lceil \log_2(M+1) \rceil$$

mentre il numero di bit del registro spiazzamento è dato da

$$\max \lceil \log_2 N_i \rceil \text{ per } i = 0, \ldots, M$$

Un secondo risparmio si può ottenere riducendo il numero di variabili di condizione che possono essere esaminate in ogni singolo stato. Infatti, dati

i compiti del processore (che deve seguire il comportamento del modello di Von Nemann) in ogni stato il SCO può esaminare solo un sottoinsieme delle variabili di condizione. Per esempio negli stati relativi alla fase di fetch, il SCO, interagendo solo con la memoria, non deve considerare le variabili di condizione provenienti dallo Status Register (SR). Questo può essere fatto mascherando gli ingressi, ovviamente nella parola della ROM si dovrà prevedere un campo per la selezione della o delle variabili di ingresso (campo SEL.

Due strutture che permetteno di implementare le suddette varianti sono schematizzate in Figura 30 e 31. La prima è relativa ad un modello di Mealy e la seconda ad un modello di Moore.

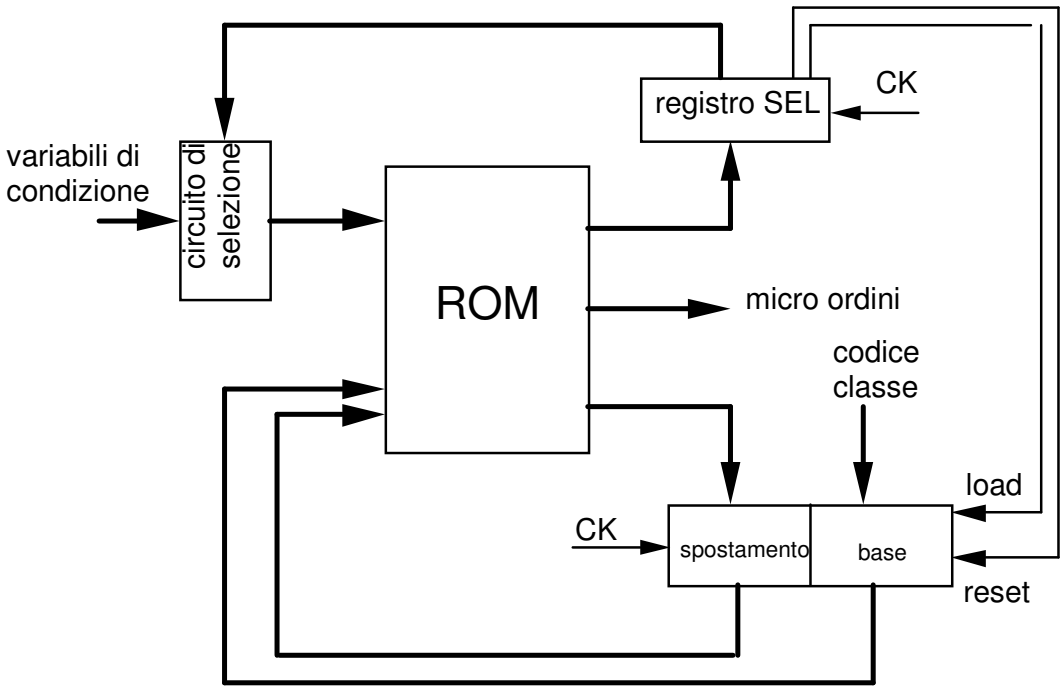


Figura 30: Schema di SCO secondo modello di Mealy.

Altri risparmi potrebbero essere ottenuti riducendo il numero di segnali di controllo (di uscita) generati dal SCO. Infatti si può notare che possono essere raggruppati logicamente quei segnali di controllo che sono mutuamente esclusivi. In questo caso è possibile utilizzare una codifica binaria per identificare nel gruppo di ogni specifico insieme quello che deve essere attivato. Per esempio nel caso dei segnali di pilotaggio dell'ALU, invece di utilizzare un numero di segnali di controllo pari al numero delle operazioni che questa unità può effettuare (*10*), dato che queste operazioni possono essere attivate una alla volta, si possono utilizzare solo *4* bit (numero intero superiore di $log_2 10$). Sarà poi compito dell'ALU, dato il codice dell'operazione, identificare l'operazione richiesta.

Infine altri risparmi si potrebbero ottenere prevedendo di strutturare il microprogramma con codice rientrante. Questa possibilità può essere sfruttata in quanto l'interpretazione di differenti istruzioni macchina comporta l'esecuzione di parti di microcodice identico. Per poter sfruttare questa possibilità è necessario prevedere una struttura del SCO che permetta il salvataggio degli indirizzi in una memoria gestita a stack a livello di microcodice. Dato il carattere introduttivo di queste dispense questa possibilità di risparmio non viene trattata in dettaglio.

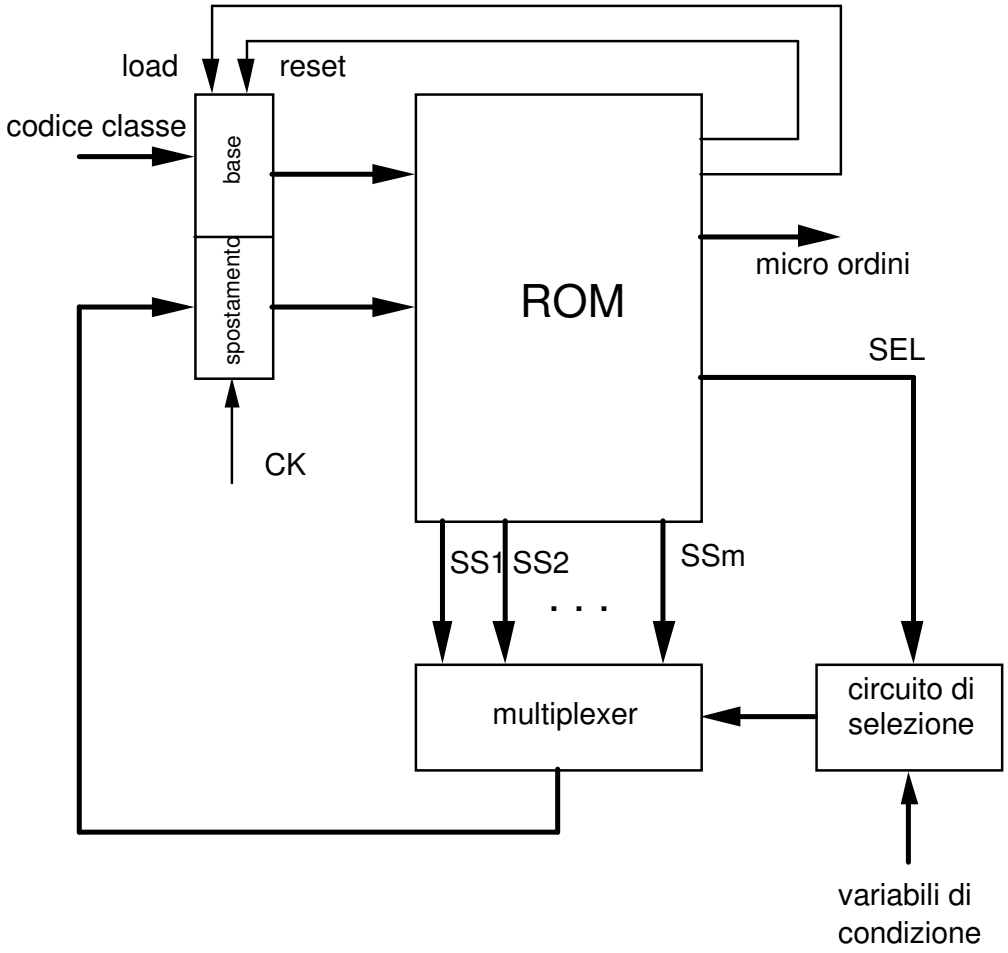


Figura 31: Schema di SCO secondo modello di Moore.

La velocità di funzionamento del SCO, e quindi il dimensionamento del clock, è funzione del tipo di macchine utilizzate per progettare il SCO e il SCA (Mealy o Moore) e dalle relative velocità di calcolo. Mentre per il SCO la velocità di calcolo è indipendente dallo stato in cui si trova, per il SCA, invece, dipende da quelle delle unità funzionali interessate all'esecuzione di un'attività. Di consequenza a parità di velocità di funzionamento del SCO più

sono veloci i componenti del SCA più piccolo è il periodo del clock di sistema. Poichè il dimensionamento della frequenza di funzionamento dipende dal più lento componente della SCA, è necessario progettare adeguatamente questi componenti in modo da non avere una grossa disparità di velocità di funzionamento tra di essi.

Domanda 12

Il campo SEL memorizzato nel registro omonimo di Figura 30 si riferisce alla microistruzione corrente o alla successiva?

Domanda 13

A che cosa serve il clock (CK) nello schema di SCO di Figura 30 e 31?

Domanda 14

A cosa serveno i segnali di load e di reset indicati negli schemi di SCO di Figura 30 e 31?

3.2 Comunicazioni con la memoria di lavoro

Di seguito si descriveranno le temporizzazioni tra il SCO e la memoria. Ovvero la sequenza temporali dei comandi che il SCO deve generare per poter effettuare un trasferimento dati da e verso la memoria di lavoro. Per semplicità di esposizione in queste temporizzazioni si ipotizza che il tempo di operazione dei moduli di memoria sia inferiore al tempo di un ciclo macchina. In caso di tempi di accesso più lunghi si rimanda al Paragrafo successivo.

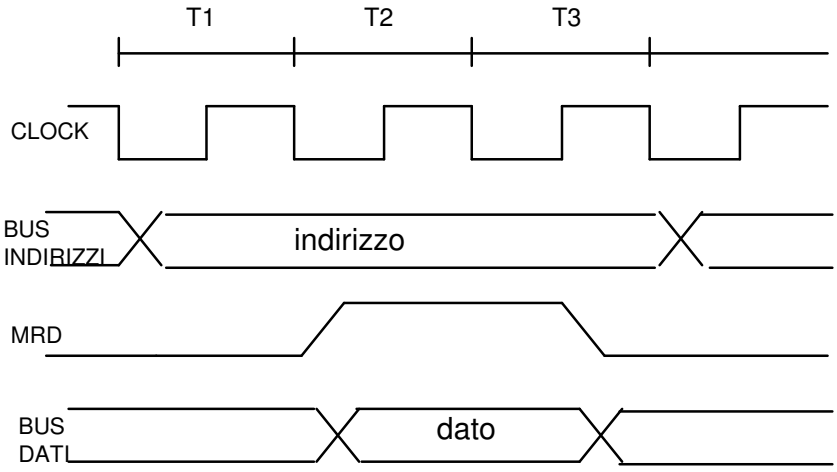


Figura 32: Ciclo di lettura in memoria

In Figura 32 è rappresentato il ciclo di lettura in memoria del PD32. Durante il primo periodo di clock del SCO, l'indirizzo della cella di memoria da leggere viene messo sul bus indirizzi. Nel secondo periodo di clock viene generato il segnale di comando MRD, dopo un intervallo di tempo sul bus dei dati si presenta stabile il dato da leggere; nel terzo periodo il dato viene memorizzato nel registro tampone MDR e il segnale MRD viene azzerato. Nei successivi periodi di clock il dato letto può essere trasferito e/o manipolato all'interno del SCA del PD32.

Domanda 15

Se il segnale MRD fosse generato nel primo periodo di clock il dato potrebbe essere letto correttamente?

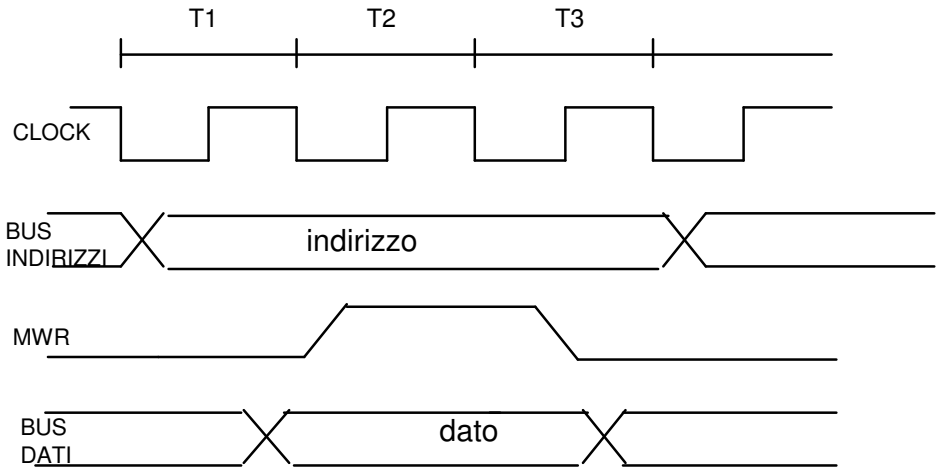


Figura 33: Ciclo di scrittura in memoria

In Figura 33 è rappresentato il ciclo di scrittura in memoria del PD32. Durante il primo periodo di clock del SCO, l'indirizzo della cella di memoria dove scrivere il dato viene messo sul bus indirizzi, mentre sul bus dati viene messo il dato. Nel secondo periodo di clock viene generato il segnale di comando MWR per la scrittura in memoria del dato presente sul bus dati; nel terzo periodo il segnale MWR viene azzerato. Per una corretta memorizzazione del dato è necessario che il tempo di scrittura dei dati in memoria sia inferiore del tempo in cui è alto il segnale di controllo MWR.

Domanda 16

Cosa potrebbe accadere se il dato venisse trasmesso alla memoria dopo la generazione del segnale MWR?

3.3 Comunicazione con i dispositivi di ingresso ed uscita

In questo Paragrafo si descriveranno le temporizzazioni tra il SCO e le porte di I/O. Ovvero la sequenza temporale dei comandi che il SCO deve generare per poter effettuare un trasferimento dati da una porta di ingresso o verso una porta di uscita.

Le tecniche di trasferimento dati tra CPU e dispositivi di ingresso/uscita dipendono dalle caratteristiche dei dispositivi e dalle relative velocità di funzionamento. In generale i dispositivi di ingresso/uscita sono delle "entità" con un proprio SCO. Quindi il trasferimento dati tra la CPU ed un dispositivo esterno deve essere effettuato in modo coordinato tra il SCO della CPU e quello del dispositivo esterno. Ovvero il SCO del dispositivo deve essere in grado di reagire quando il SCO della CPU vuole effettuare un trasferimento e viceversa; ed inoltre il trasferimento può essere effettuato solo quando entrambi i SCO lo consentono. Questo coordinamento deve avvenire seguendo un insieme di regole prestabilite (protocollo) che definisce le possibili sequenze di interazione. Se il trasferimento è coordinato unicamente dal SCO del processore il trasferimento si dice *sincrono* (sincrono con la velocità di funzionamento del SCO del processore), altrimenti *semi-sincrono* o *asincrono*.

Nel primo caso il dispositivo indirizzato (cioè quello con cui la CPU necessita di interagire) deve leggere e/o scrivere il dato nell'intervallo di tempo previsto dal SCO del PD32 per la lettura e/o la scrittura. Invece, se la velocità di lettura/scrittura del dispositivo è inferiore a quella richiesta è necessario "rallentare" il SCO del PD32 nell'interazione con il dispositivo stesso. A tal fine è possibile usare un segnale di controllo $\overline{\text{WAIT}}$ che viene usato dal SCO del processore per individuare se la periferica ha completato o meno il trasferimento richiesto. La generazione di questo segnale può essere controllata dal SCO del dispositivo interagente o nel caso di asservimento completo del dispositivo al processore (ovvero il dispositivo è visto come un componente del SCA coordinato dal SCO del processore) questo segnale può essere generato indirettamente dal SCO del processore stesso. Notare come quest'ultima tecnica può essere utilizzata anche per effettuare il trasferimento dati utilizzando memorie con velocità di funzionamento inferiore a quella dei processori che vi accedono. Questo tipo di trasferimento si dice *semi-sincrono* in quanto il trasferimento dati è effettuato come variante del *sincrono*, salvaguardandone il principio di funzionamento. Notare come la velocità di trasferimento dei dati tra il registro interno del processore e quello del dispositivo esterno, sia nel caso sincrono che semi-sincrono, è funzione della frequenza del clock interno del processore. Con il protocollo *asincrono*, invece, la velocità di trasferimento dati è funzione delle velocità di entrambi i dispositivi interagenti piuttosto che di una frequenza fissa di un unico clock. Ci sono diverse tecniche di trasferimento dei dati asincrone, alcune

implementabili completamente via software, altre utilizzanti hardware ed software.

Nel Paragrafo successivo si descriverà la gestione degli eventi asincroni, tra questi c'è la gestione della richiesta di interruzione che è fondamentale per implementare i protocolli asincroni di trasferimento.

Di seguito si descriveranno i cicli di lettura e di scrittura verso periferica del PD32 in cui è previsto il trasferimento semi-sincrono.

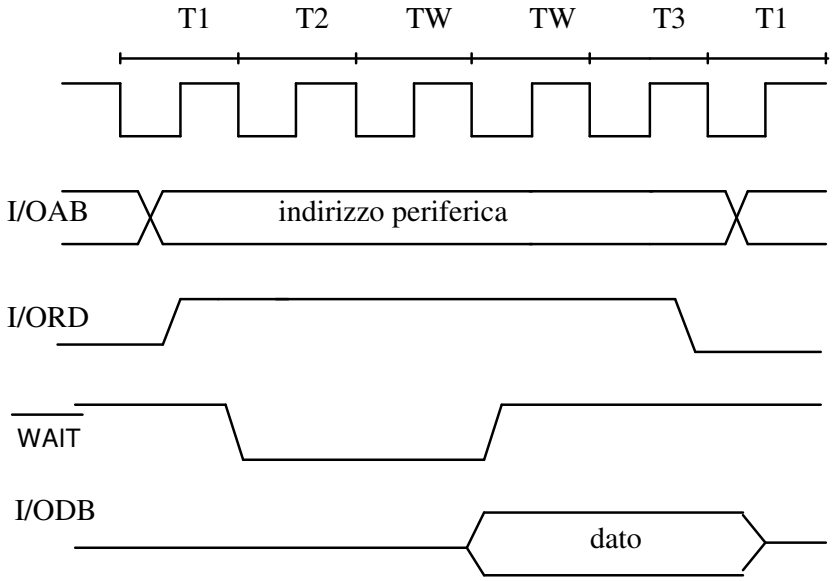


Figura 34: Ciclo di lettura da periferica.

In Figura 34 è rappresentato il ciclo di lettura da periferica del PD32. Durante il primo periodo di clock del SCO, viene generato il segnale di comando I/ORD e l'indirizzo della periferica da cui bisogna leggere il dato viene messo sul bus indirizzi delle periferiche (I/OAB). Nel periodo successivo il SCO verifica se la variabile di condizione $\overline{\text{WAIT}}$ è alta o bassa; nel caso di $\overline{\text{WAIT}}$ bassa il SCO non effettua alcuna operazione ed attende che questa variabile divenga alta (da ricordare che quando è alta il dispositivo esterno ha messo sull'I/ODB il dato richiesto). Quando verifica che è alta il SCO memorizza nel registro tampone I/ODR il valore presente sull'I/ODB e il segnale I/ORD viene azzerato. Nei successivi periodi di clock il dato letto può essere trasferito e/o manipolato all'interno del SCA del PD32. In Figura 34 i periodi in cui il SCO è in attesa che il segnale $\overline{\text{WAIT}}$ diventi alto sono indicati con TW (W da wait).

In Figura 35 è rappresentato il ciclo di scrittura in periferica del PD32. Durante il primo periodo di clock del SCO, l'indirizzo della periferica su cui

bisogna scrivere il dato viene messo sul bus indirizzi delle periferiche (I/OAB), il dato da scrivere viene posto sull'I/ODB e viene generato il segnale di comando I/OWR. All'inizio del periodo successivo il SCO verifica se la variabile di condizione $\overline{\text{WAIT}}$ è alta o bassa; nel caso di $\overline{\text{WAIT}}$ basso il SCO non effettua alcuna operazione ed attende che questa variabile divenga alta (quando è alta il dispositivo esterno ha letto dall'I/ODB il dato). Quando verifica che è alta il SCO abbassa il segnale di controllo I/OWR e entra in un ciclo macchina successivo.

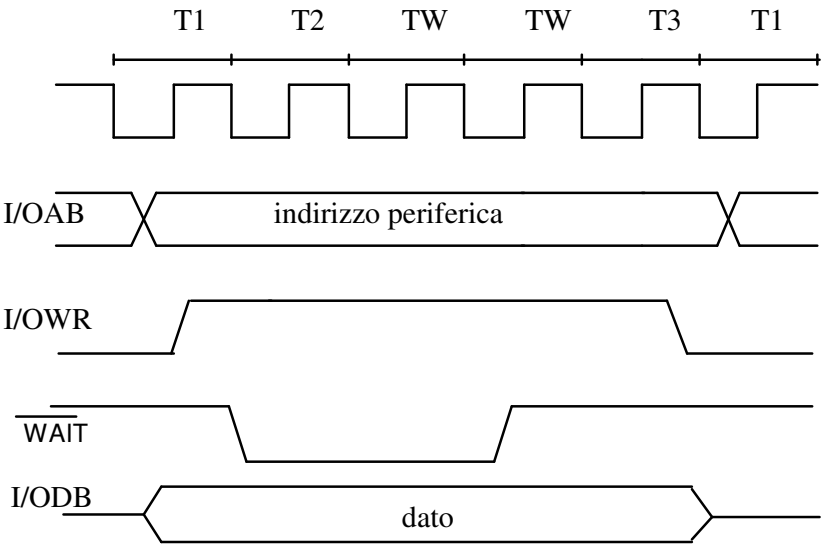


Figura 35: Ciclo di scrittura in periferica.

3.4 Gestione eventi asincroni

Come abbiamo visto nel Paragrafo precedente il trasferimento dati tra processore e dispositivi di ingresso ed uscita può essere effettuato in modo asincrono. Tale modalità può essere implementata tramite la tecnica delle interruzioni. Questa tecnica può essere utilizzata anche per la segnalazione di eventi non prevedibili a priori, come un abbassamento di tensione elettrica, in cui è necessario prevedere un intervento "immediato" del processore, che dovrebbe quindi sospendere (interrompere, da cui il nome interruzione) l'esecuzione del programma corrente per intraprendere le attività per cui è stato interrotto.

Il trasferimento di un blocco di dati tra due periferiche o tra due moduli di memoria o tra una periferica ed un modulo di memoria può essere effettuato sotto il controllo del processore, tramite l'esecuzione di un programma. In questo caso il trasferimento di ogni singolo dato (byte, parola o doppia parola) può essere fatto utilizzando un registro interno del

processore per memorizzare temporaneamente il dato che deve essere trasferito. Ciò comporta che il trasferimento di ogni singolo dato necessita almeno dell'esecuzione di un'istruzione per prelevare il dato dalla sorgente, di un'istruzione per la scrittura del dato nel dispositivo destinatario, di un'istruzione di incremento o decremento di un contatore (necessario per tenere in conto del numero di trasferimenti effettuati) e di un'istruzione per effettuare il controllo se si è completato il trasferimento del blocco di dati. Considerando che l'esecuzione di ogni singola istruzione necessita di un ciclo macchina per il fetch e di eventuali altri cicli macchina per leggere/scrivere dati e/o operandi, si comprende come questa tecnica sia relativamente lenta. Per incrementare la velocità di trasferimento dei dati, poichè il relativo programma di controllo può essere scritto in modo parametrico (dove i parametri sono: identificazione sorgente, identificazione destinazione, lunghezza del blocco da trasferire), si può evitare di prelevare le istruzioni del programma dalla memoria (evitando così la fase di fetch di ogni singola istruzione) specializzando un opportuno dispositivo per il trasferimento di blocchi di dati: questo dispositivo è denominato DMAC da Direct Memory Access Controller. Questo dispositivo per poter effettuare il trasferimento dei dati deve emulare il comportamento della CPU. Per far ciò è necessario che generi gli stessi segnali di controllo e di indirizzamento della CPU e i dispositivi e/o i blocchi di memoria si devono comportare come se il trasferimento fosse sotto il controllo della CPU. Di conseguenza tra la CPU e il DMAC vi è una concorrenza nell'utilizzazione dei bus. Affinchè il DMAC possa utilizzare correttamente queste risorse è necessario che la CPU non interferisca elettricamente su di esse, e viceversa. Poichè la richiesta di utilizzare i bus (e quindi che essi vengano rilasciati dalla CPU) può sorgere indipendentemente dalle attività del processore (ovvero in modo asincrono) è necessario utilizzare un segnale di controllo verso il processore (*Memory Bus Request*, $\overline{\text{MBR}}$) per avvertire il suo SCO della presenza di questo evento. Una volta riconosciuta tale richiesta il SCO del PD32 mette ad alta impedenza le uscite del processore verso i suddetti bus. Il DMAC viene avvertito del rilascio dei bus tramite un segnale di controllo specifico (*Memory Bus Grant*, MBG).

Di seguito viene descritto come il SCO del PD32 gestisce le richieste di interruzione e le richieste di utilizzare i bus (e quindi che essi vengano rilasciati dalla CPU).

Domanda 17

Se la velocità di trasferimento di un dispositivo esterno con un proprio SCO è maggiore di quella del processore è sempre possibile usare la tecnica di trasferimento sincono?

3.4.1 Gestione delle richieste di interruzione

Scopo di una interruzione è quello di permettere ad un dispositivo esterno di far sospendere l'esecuzione del corrente programma e di obbligare la CPU ad iniziare un programma di servizio relativo alla richiesta stessa. Per poter far ciò è necessario:

- che la CPU sia disponibile ad essere interrotta (interruzioni abilitate);

- che l'interruzione del programma corrente non comporti un'interferenza sulla correttezza della sua esecuzione;

- che la CPU identifichi chi ha mandato l'interruzione.

Di seguito faremo vedere come il PD32 gestisce l'abilitazione delle interruzioni, garantisca che l'esecuzione di un programma di servizio non interferisca con quello corrente e quale tecnica utilizza per identificare la sorgente dell'interruzione.

Abilitazione/disabilitazione delle interruzioni

Per garantire la corretta esecuzione di un segmento di programma o per garantirne la sua esecuzione in un fissato intervallo di tempo alcune volte è necessario che la sua intepretazione da parte della CPU venga fatta in modo non interrompibile. Per memorizzare l'informazione che le interruzioni siano o meno abilitate si fa uso di un flip-flop (denominato *I* e contenuto nel registro SR). Il contenuto di questo flip-flop può essere manipolato dalle istruzioni macchina CLRI e SETI e può essere usato nelle istruzioni di salto condizionato.

Verifica richiesta delle interruzioni

La richiesta di un'interruzione avviene in modo asincrono rispetto alle attività del processore e quindi del suo SCO. Le attività del SCO sono scandite in modo sincrono da un clock e si è visto nel Paragrafo precedente che il controllo del SCO una volta eseguito un microprogramma relativo ad una istruzione macchina ritorna al microprogramma relativo alla fase di fetch (l'insieme fetch ed esecuzione di una istruzione viene anche detto *ciclo istruzione*). Per non complicare ulteriomente l'organizzazione del SCO, se le attività correnti sono interrompibili, la verifica della presenza della richiesta di un'interruzione viene fatta alla fine di ogni ciclo istruzione, questo perchè in questo modo è necessario salvare solo lo stato del programma in

esecuzione (corrente) codificato in linguaggio macchina, altrimenti sarebbe stato necessario salvare quello del microprogramma.

Salvataggio dello stato

Prima di eseguire il programma di servizio la CPU deve mettersi nelle condizioni di poter riprendere le attività interrotte, una volta eseguito il programma di servizio. Questo modo di operare è simile a quello visto in precedenza nella gestione dei sottoprogrammi, quindi prima che la CPU possa iniziare l'esecuzione del programma di servizio il SCO deve salvare il contenuto del PC e il contenuto dello SR in una zona di memoria predefinita. Poichè si prevede che la gestione di una interruzione possa essere a sua volta interrotta da un'altra interruzione è necessario prevedere che la gestione della memoria usata per salvare lo stato della CPU venga fatta come uno stack. Il salvataggio del contenuto del PC e del SR è effettuato dal SCO nelle locazioni di memoria in cima allo stack. Per motivi di efficienza il puntatore dello stack deve essere memorizzato in un registro interno della CPU. Il registro R7 viene specializzato per questa finalità.

E' da notare che il contenuto del PC deve essere salvato necessariamente dal SCO, in quanto per poter attivare il programma di servizio richiesto dall'interruzione è necessario caricare l'indirizzo della sua prima istruzione nel PC, e quindi se non la si salva precedentemente questa informazione viene persa. Il contenuto dello SR potrebbe anche essere salvato via software, ma poichè, in generale, l'esecuzione di un programma di servizio comporta la modifica di questo registro tanto vale salvarlo via SCO guadagnandone in velocità. Eventualmente potrebbero essere salvati anche i contenuti di tutti i registri interni visibili dall'esterno via SCO (gestione salvataggio via *hardware*), ma in generale i programmi di servizio ne usano solo un sottoinsieme, quindi conviene farlo effettuare di volta in volta dai programmi di servizio (gestione salvataggio via *software*).

Identificazione sorgente dell'interruzione

L'identificazione del dispositivo esterno che ha effettuato la richiesta dell'interruzione può essere fatta via software o via hardware. Nel PD32 si è optato per il secondo tipo di identificazione.

Una volta riconosciuta la presenza di un'interruzione il SCO genera un segnale di controllo, *Interrupt Acknowledgement* (IACK), per avvertire il dispositivo esterno che è in grado di ricevere sul I/O Data Bus (I/ODB) l'identificazione del dispositivo esterno. Notare che in questo caso ci potrebbero essere più dispositivi che hanno fatto una richiesta di interruzione e quindi ci potrebbe essere un'interferenza in scrittura sul bus dell'identificazione della sorgente. Quindi c'è la necessità di un modulo di interfaccia tra la CPU e i dispositivi esterni che raccolga le varie richieste di

interruzioni e che le serializzi o in ordine temporale di arrivo o seguendo un qualunque criterio di priorità.

L'identificazione del dispositivo esterno è utilizzata per indirizzare il programma di servizio. Sull'I/O Data Bus abbiamo a disposizione 8 bit e questi devono essere convertiti in 32 bit per poter identificare l'indirizzo iniziale del programma di servizio. A tal fine dalla locazione di memoria $(0000)_H$ è memorizzato il cosidetto vettore delle interruzioni ("interrupt vector table"), in cui ogni singolo componente è costituito di quattro byte; al componente *i-esimo* è associato l'indirizzo iniziale del programma di servizio relativo al dispositivo *i*. Per identificare la posizione del vettore in cui è memorizzato l'indirizzo del programma di servizio si moltiplica per quattro l'identificatore del dispositivo esterno. Il numero massimo di programmi di servizio è pari a $2^8 = 256$. E' ovvio che è compito del programmatore di sistema associare ad ogni locazione del vettore delle interruzioni l'indirizzo del relativo programma di servizio.

Esecuzione del programma di servizio

Per garantire che l'interruzione dell'esecuzione del programma corrente non comporti un'interferenza sulla sua evoluzione è necessario che, oltre alle informazioni salvate dal SCO (contenuto del PC e SR), nello stack vengano anche memorizzati i contenuti di quei registri che il programma di servizio modificherà. Questo salvataggio (tramite l'esecuzione di una serie di istruzioni PUSH) deve essere effettuato dal programma di servizio prima di iniziare ad eseguire quelle istruzioni che possono modificarne il contenuto, quindi è buona norma mettere all'inizio del programma di servizio le istruzioni di salvataggio del contenuto dei registri. Dopodichè si potranno eseguire le attività connesse alla richiesta di servizio.

Ripristino dello stato e ripresa dell'ultimo programma interrotto

Le ultime attività del programma di servizio debbono essere quelle di ripristino dello stato del programma interrotto salvato via software (tramite una serie di POP i cui operandi saranno in ordine inverso di quello delle corrispondenti PUSH) e quindi si dovrà ristabilire il contenuto del SR e del PC. Questo viene fatto con l'esecuzione dell'istruzione RTI, che dovrà quindi essere l'ultima del programma di servizio.

Domanda 18

Cosa si deve fare per non interrompere un programma di servizio?

3.4.2 Gestione delle richieste di accesso ai bus

Abbiamo visto precedentemente che la richiesta di accesso ai bus (e, quindi, del loro rilascio da parte della CPU) può sorgere in modo asincrono rispetto alle attività del processore e che questa richiesta viene trasmessa alla CPU tramite il segnale $\overline{\text{MBR}}$. A differenza della gestione dell'interruzione, in caso di richiesta di rilascio del bus il processore non deve eseguir alcun programma di servizio, ma deve solo mettere ad alta impedenza le proprie uscite verso i bus esterni. La verifica della presenza della richiesta di rilascio del bus quindi non deve necessariamente essere effettuata alla fine del *ciclo istruzione* ma è possibile verificarla alla fine di ogni singolo *ciclo macchina*. Una volta riconosciuta tale richiesta il SCO del PD32 mette ad alta impedenza le uscite del PD32 verso i suddetti bus. Il richiedente viene avvertito del rilascio dei bus tramite un segnale di controllo specifico (Memory Bus Grant, MBG). Il richiedente avvertirà il SCO del rilascio dei bus condivisi annullando la richiesta (ovvero mettendo alto il segnale $\overline{\text{MBR}}$).

3.4.3 Diagramma di flusso della gestione degli eventi asincroni

In Figura 36 viene ripresentato il diagramma di flusso del microprogramma relativo ad una generica istruzione macchina in cui è prevista anche la gestione delle richieste di interruzione e della richiesta del rilascio dei bus. Da notare che mentre la verifica della richiesta delle interruzioni viene fatta a fine ciclo istruzione, quella del rilascio del bus viene effettuata alla fine di ogni ciclo macchina.

Domanda 19

Perchè il riconoscimento della presenza delle interruzioni è effettuata alla fine del ciclo istruzione?

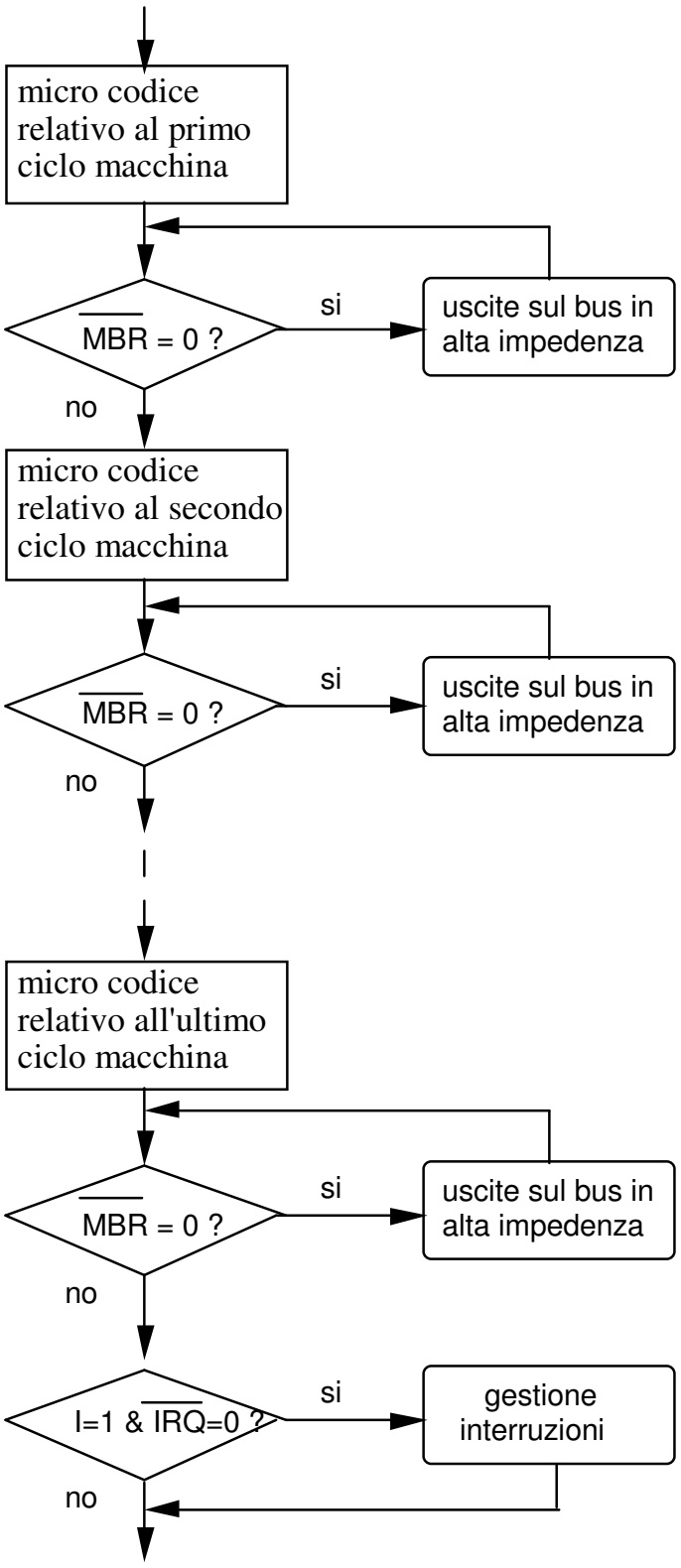


Figura 36: Gestione eventi asincroni.

3.5 Segnali di controllo del PD32

Di seguito, per comodità del lettore, prima di elencare i cicli macchina (o cicli di bus) del PD32 e descriverne alcuni in dettaglio, vengono elencati integralmente i segnali di controllo in ingresso al PD32 (variabili di condizione del SCO) e i segnali di controllo in uscita dal PD32 (generati dal SCO) necessari per interagire con le unità esterne. Alcuni segnali di controllo sono stati introdotti nelle Sezioni precedenti, altri invece saranno descritti in dettaglio successivamente. Il riepilogo viene fatto separando i segnali di ingresso da quelli in uscita. I segnali di ingresso ed uscita corrispondenti sono disposti sulla stessa riga.

USCITE del SCO **- segnali di controllo -**	**INGRESSI del SCO** **- variabili di condizione -**
Mb_i (i= 0, 1, 2, 3)	
MRD (Memory Read)	
MWR (Memory Write)	
I/ORD (I/O Read)	
I/OWR (I/O Write)	
IACK (Interrupt Acknowledgment)	$\overline{IRQ}$ (Interrupt Request)
MBG (Memory Bus Grant)	$\overline{MBR}$ (Memory Bus Request)
	$\overline{WAIT}$
	$\overline{RESET}$
	$\overline{READY}$
START	
CLEAR	
SETIM (Set Interrupt Mask)	
CLRIM (Clear Interrupt Mask)	

$\overline{\text{IMS}}$ (Interrupt Mask Status)

Di seguito viene data una descrizione delle attività collegate ai segnali di controllo.

Mb_i (i= 0, 1, 2, 3)

Segnale di controllo generato dal SCO all'atto dell'interpretazione di un'istruzione macchina di tipo trasferimento dati da/verso la memoria. Questo segnale di controllo abilita l'accesso al modulo di memoria i-esimo.

MRD (Memory Read)

Segnale di controllo generato dal SCO all'atto dell'interpretazione di un'istruzione macchina di tipo trasferimento dati da memoria. Questo segnale di controllo abilita la lettura del dato memorizzato nella cella (o nelle celle se si tratta di dati a due o quattro byte) di memoria specificata nell'AB e il suo trasferimento sul DB.

MWR (Memory Write)

Segnale di controllo generato dal SCO all'atto dell'interpretazione di un'istruzione macchina di tipo trasferimento dati a memoria. Questo segnale di controllo abilita la scrittura del dato sul DB nella cella (o nelle celle se si tratta di dati a due o quattro byte) di memoria specificata dall'AB.

I/ORD (I/O Read)

Segnale di controllo generato dal SCO all'atto dell'interpretazione di un'istruzione macchina di tipo trasferimento dati da una porta di ingresso. Questo segnale di controllo abilita la lettura del dato memorizzato nel registro della porta di ingresso specificata nell'I/OAB e il suo trasferimento sull'I/ODB.

I/OWR (I/O Write)

Segnale di controllo generato dal SCO all'atto dell'interpretazione di un'istruzione macchina di tipo trasferimento dati a porta di uscita. Questo segnale di controllo abilita la scrittura del dato sull'I/ODB nel registro della porta di uscita specificata dall'I/OAB.

$\overline{\text{IRQ}}$ (Interrupt Request)

Segnale di controllo generato da un dispositivo esterno per sospendere l'attività corrente del PD32 e fargli eseguire un programma di servizio.

IACK (Interrupt Acknowledgment)

Segnale di controllo generato dal SCO del PD32 per avvertire il dispositivo generante la richiesta dell'interruzione che ha sospeso la precedente attività e che attende l'identificazione del richiedente.

$\overline{\text{MBR}}$ (Memory Bus Request)

Segnale di controllo generato da un dispositivo esterno per richiedere al SCO di mettere le proprie uscite sui bus esterni in alta impedenza. Questo segnale di controllo deve essere mantenuto fino a che il dispositivo esterno non ha finito di utilizzare i suddetti bus.

MBG (Memory Bus Grant)

Segnale di controllo generato dal SCO del PD32 per avvertire il dispositivo generante la richiesta di rilascio dei bus che ha effettivamente messo in alta impedenza le proprie uscite e quelle del SCA interno del processore verso i bus esterni.

$\overline{\text{WAIT}}$

Segnale di controllo generato da un dispositivo esterno per "rallentare" il SCO del PD32 nell'interazione con il dispositivo stesso. Questo segnale deve rimanere basso fino a che il dispositivo esterno non ha completato il trasferimento richiesto.

$\overline{\text{RESET}}$

Segnale di controllo generato da un dispositivo esterno per forzare il PD32 ad eseguire un programma da un indirizzo prefissato. All'atto della ricezione di questo segnale il SCO prende il contenuto informativo presente sull'I/ODB e lo pone sul PC, pone a zero i flip-flop del registro SR (quindi disabilita le interruzioni).

START

Segnale di controllo generato dal SCO all'atto dell'interpretazione dell'istruzione macchina START per azzerare il flip-flop di stato STATUS del dispositivo indirizzato dall'I/OAB e per avviare l'operazione di I/O.

CLEAR

Segnale di controllo generato dal SCO all'atto dell'interpretazione dell'istruzione macchina CLEAR per azzerare il flip-flop di stato STATUS del dispositivo indirizzato dall'I/OAB.

$\overline{\text{READY}}$

Segnale di controllo generato da un dispositivo esterno per segnalare lo stato del proprio flip-flop STATUS. Questo segnale viene usato dal SCO del processore all'atto dell'interpretazione delle istruzioni macchina JR e JNR.

SETIM (Set Interrupt Mask)

Segnale di controllo generato dal SCO all'atto dell'interpretazione dell'istruzione macchina SETIM per abilitare il dispositivo indirizzato dall' I/OAB (viene posto ad 1 il flip-flop IM) ad inviare interruzioni.

CLRIM (Clear Interrupt Mask)

Segnale di controllo generato dal SCO all'atto dell'interpretazione dell'istruzione macchina CLRIM per disabilitare il dispositivo indirizzato dall' I/OAB (viene posto a O il flip-flop IM) ad inviare interruzioni.

$\overline{\text{IMS}}$(Interrupt Mask Status)

Segnale di controllo generato dal dispositivo indirizzato dall'I/OAB per segnalare lo stato del proprio flip-flop IM. E' usato dal SCO all'atto dell'interpretazione delle istruzioni macchina JIM e JNIM.

Domanda 20

Per scrivere o leggere un dato dalla memoria sono necessari due segnali (come MRD e MWR) o è possibile usare un solo segnale di controllo?

3.6 Passi elementari dell'interpretazione di una istruzione

Come visto l'interpretazione di ogni istruzione del PD32 comporta l'esecuzione di un microprogramma da parte del SCO. L'esecuzione del microprogramma relativa ad una istruzione viene detta *ciclo istruzione*. Ogni ciclo istruzione è composto da una o più fasi elementari che comporteranno l'attivazione di comandi (*micro operazioni*) relativi ad unità interne al processore (registri, ALU, shift register) e/o relative ad unità esterne allo stesso (memoria, unità periferiche). Le fasi che comportano interazione con le unità esterne vengono anche dette *cicli macchina.* A sua volta ogni ciclo macchina può essere costituito da uno o due *cicli di bus*; per esempio la lettura di una parola memorizzata su due byte non allineati sullo stesso indirizzo di riga necessita di due accessi in memoria (cioè di due cicli di bus).

I cicli macchina possibili per il PD32 sono:

- fetch dell'istruzione;

- lettura da memoria;
- scrittura in memoria;
- lettura da memoria tramite stack pointer;
- scrittura in memoria tramite stack pointer;
- lettura da periferica;
- scrittura in periferica;
- riconoscimento di interruzione;
- riconoscimento di interruzione nello stato di halt;
- riconoscimento di $\overline{\text{RESET}}$;
- inizializzazione di periferica;
- azzeramento del flip-flop di stato STATUS di periferica;
- lettura da periferica dello stato del flip-flop STATUS;
- abilitazione di periferica ad inviare interruzioni;
- disabilitazione di periferica ad inviare interruzioni;
- lettura da periferica dello stato di abilitazione delle interruzioni.

Ogni ciclo istruzione è composto da almeno un ciclo macchina (uguale per tutti i cicli istruzioni) per l'estrazione dalla memoria dell'istruzione da eseguire (fetch dell'istruzione), da una o più attività interne ed eventualmente da altri cicli macchina se la semantica dell'istruzione comporta ulteriori interazioni con le unità esterne.

I cicli macchina sono costituiti da una sequenza di microistruzioni (o stati operativi) il cui numero dipende dal tipo di ciclo.

Di seguito si faranno vedere le attività del PD32 relativamente ad alcuni cicli macchina.

3.6.1 Ciclo di fetch

In questo ciclo si legge dalla memoria la prossima istruzione da eseguire e si predispone il Program Counter (PC) a puntare alla successiva istruzione. Poichè il Program Counter (PC) non è connesso al bus esterno degli indirizzi in questa fase il contenuto del PC è messo sul Memory Address Register (MAR), che ha compiti di interfaccia tra la CPU e il bus degli indirizzi. L'istruzione da eseguire deve essere caricata sull'Instruction Register (IR) per essere interpretata dal SCO.

```
PC -> MAR;                  /* trasferimento del contenuto del PC sul
                               MAR */
(MAR) -> IR31-0;/* trasferimento dell'istruzione da eseguire
                               sull'IR */
```

```
PC + 4 -> PC ;          /* predisposizione esecuzione istruzione
                           successiva */
```

Da notare che questo ciclo macchina può essere costituito da uno o due cicli di bus a secondo se il contenuto di PC è multiplo o meno di quattro (vedi organizzazione della memoria Sezione 2.3.4).

3.6.2 Ciclo di riconoscimento di interruzione

Questo ciclo viene eseguito se le interruzioni sono abilitate ed esiste una richiesta di interruzione. La richiesta di interruzione viene verificata alla fine di ogni ciclo istruzione. Questo ciclo serve a dichiarare al PD32 l'indirizzo iniziale del sottoprogramma di servizio associato alla richiesta di interruzione. A tal fine il SCO si aspetta che un dispositivo esterno invii sull'I/ODB un byte che una volta moltiplicato per quattro, fornisce l'indirizzo dell'elemento del vettore delle interruzioni contenente l'indirizzo iniziale del programma di servizio. Questo ciclo è seguito da due cicli di scrittura in memoria tramite stack pointer per il salvataggio del PC e dello SR, dopodichè inizierà l'esecuzione del programma di servizio.

```
/*ciclo di riconoscimento di interrupt */

1 -> IACK;          /*generazione segnale di
                        riconoscimento di interruzione */
I/ODB -> I/ODR;  /*prelievo identificatore dispositivo*/
I/ODR* 4 -> MAR;    /*individuazione della locazione del vettore
                    di interruzione in cui è memorizzato
                    l'indirizzo iniziale del programma di
                    servizio richiesto*/
(MAR) -> TEMP2;     /*caricamento dell'indirizzo iniziale del
                    programma di servizio in un registro
                  temporaneo*/

/*primo ciclo di scrittura in memoria tramite stack pointer
per salvare PC del programma corrente */

R7 - 4 -> R7;       /*decremento stack pointer*/
R7 -> MAR;
PC -> MDR;
MDR -> (MAR);       /*memorizzazione del contenuto del PC
                    relativo al programma corrente nello
                    stack*/
```

/* secondo *ciclo di scrittura in memoria tramite stack pointer* per salvare SR del programma corrente*/

```
R7 - 4 -> R7;          /* decremento stack pointer*/
R7 -> MAR;
SR -> MDR;
MDR -> (MAR);          /*memorizzazione del contenuto del SR
                       relativo al programma corrente nello
                       stack*/
```

/**attività interne* per la disabilitazione delle interruzioni e la memorizzazione nel PC dell'indirizzo iniziale del programma di servizio*/

```
0 -> I;
TEMP2 -> PC;
```

/* *ciclo di fetch* per prelevare la prima istruzione del programma di servizio*/

<vedi ciclo di fetch>

Da notare che il ciclo di scrittura in memoria tramite stack pointer può essere costituito da uno o due cicli di bus a secondo se il contenuto di R7 è multiplo o meno di quattro (vedi organizzazione della memoria Sezione 2.3.4).

Esercizio 6

Descrivere il ciclo di scrittura in periferica relativo all'istruzione OUTB R1,<dev>.

4. CONCLUSIONI

4.1 Sommario

In questa dispensa si è progettato il PD32 con reti logiche combinatorie e sequenziali.

Per progettare questo processore, data la complessità delle funzioni che il PD32 deve effettuare, si è scomposto il processore in due entità interagenti: un Sottosistema di COntrollo (SCO), con funzioni di coordinamento, ed un Sottosistema di CAlcolo (SCA) asservito a quello di controllo.

In particolare, dato il numero delle funzioni di controllo, si è implementato il SCO tramite la microprogrammazione. Quindi il problema della progettazione del SCO è divenuto quello della determinazione dell'algoritmo che implementa le funzioni di controllo richieste e la sua organizzazione che ne minimizza il costo. Per il SCA, invece, la difficoltà della progettazione è risieduta nella identificazione, tra le possibili alternative, della soluzione tenendo in conto del costo e delle ripercussioni sulla complessità dell'algoritmo del SCO. Il modello di riferimento scelto per il processore è quello proposto da Von Neumann, di conseguenza, poichè il PD32 può eseguire una istruzione alla volta, abbiamo definito un SCA con un basso investimento hardware. Infatti la struttura di interconnessione che si è scelta è quella basata sul bus. Durante la progettazione del SCA si sono fatte vedere alcune possibili alternative e si è sempre discusso criticamente le loro ripercussioni in termini di *costo* e di *prestazioni*. Inoltre, nell'esplicitare le funzioni e le possibili alternative architetturali sono state motivate le necessità della presenza dei registri interni al PD32, così come quali ripercussioni si sono avute in termini di costo e prestazioni nell'organizzare i registri interni visibili dall'utente in un banco di memoria.

Dopodichè si sono viste alcune tecniche di interfacciamento del PD32 con la memoria di lavoro e con i dispositivi di ingresso ed uscita. Ed anche in questo caso si sono fatte vedere le necessità architetturali per poter implementare tali collegamenti, sia a livello di SCA che di SCO. Particolare attenzione si è dedicata alla definizione delle tecniche di comunicazione asincrone con i dispositivi esterni e si è fatto vedere come il PD32 gestisce la richiesta delle interruzioni e la richiesta del rilascio dei bus.

4.2 Risposta alle domande

Domanda 1

Codici delle istruzioni macchina, dati ed indirizzi di memoria.

Domanda 2

Poichè 10 sono le operazioni che deve eseguire l'ALU, se il codice dell'operazione è codificato in binario sono necessari 4 segnali di controllo.

Domanda 3

0 se il fronte di salita dell'impulso si presenta tra t6 e t7, 1 se si presenta tra t7 e t8.

Domanda 4

n-1.

Domanda 5

n-1.

Domanda 6

- Connessione di un registro con l'ALU (che viene comandata dal SCO abilitando opportunamente uno dei multiplexer in ingresso all'ALU);

- specificazione del tipo di operazione da effettuare (che viene fatta dal SCO comandando opportunamente l'ALU);

- scrittura del risultato nel registro destinatario (che viene effettuata dal SCO comandando uno dei multiplexer in ingresso ai registri).

Domanda 7

- Scrittura dell'operando su TEMP1 o su TEMP2 (che viene comandata dal SCO abilitando opportunamente il buffer three state relativo al primo registro sorgente e abilitando in scrittura il registro temporaneo);

- specificazione del tipo di operazione da effettuare (che viene fatta dal SCO comandando opportunamente l'ALU);

- scrittura del risultato nel registro destinatario (che viene effettuata dal SCO abilitando opportunamente il buffer three state in uscita dell'ALU e abilitando in scrittura il registro destinatario).

Domanda 8

2.

Domanda 9

2^{32}.

Domanda 10

2^{16}.

Domanda 11

Si, perchè ha solo funzioni di memorizzazione temporanea tra un registro interno del PD32 visibile dalle istruzioni macchina ed il registro di una periferica, e viceversa. Notare, comunque, la necessità del buffer three state bidirezionale che non può essere eliminato.

Domanda 12

Alla successiva.

Domanda 13

Serve ad abilitare la memorizzazione dei dati in ingresso ai registri (spostamento e campo SEL in Figura 30 e spostamento in Figura 31).

Domanda 14

Il primo serve a caricare il codice della classe, mentre il secondo ad azzerare il contenuto del registro spostamento.

Domanda 15

Si, purchè rimanga alto fino al terzo periodo di clock.

Domanda 16

Si potrebbe avere la memorizzazione di un dato diverso da quello desiderato.

Domanda 17

No, perchè la presenza di un SCO nel dispositivo esterno indica la possibilità che il dispositivo stia effettuando altre operazioni diverse da quelle richieste dal processore.

Domanda 18

Si devono disabilitare le interruzioni.

Domanda 19

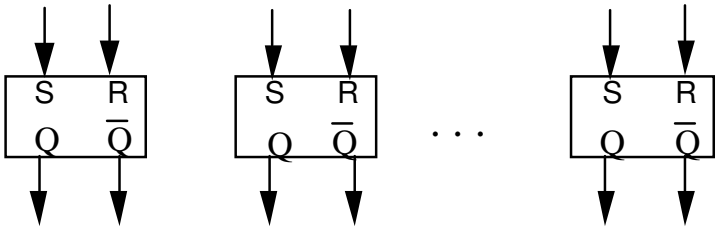
Perchè in questo modo è necessario salvare solo lo stato del programma in esecuzione (corrente) scritto in linguaggio macchina, altrimenti sarebbe stato necessario salvare quello del microprogramma.

Domanda 20

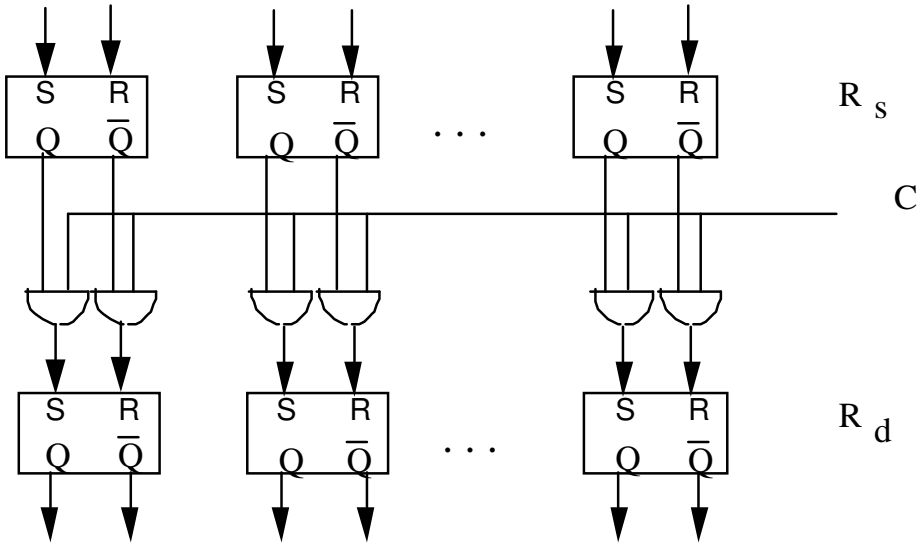
Data la struttura delle memorie, queste una volta indirizzate o effettuano un'operazione di lettura o alternativamente un'operazione di scrittura. Quindi può essere usato un solo segnale di controllo.

4.3 Soluzione degli esercizi

Esercizio 1

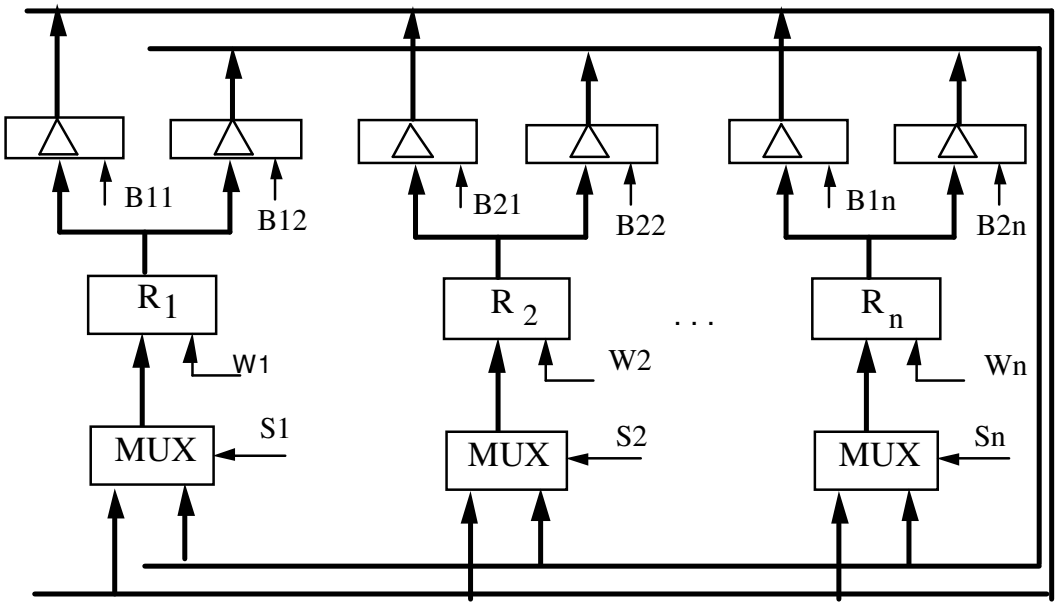


Esercizio 2



Il segnale C serve per abilitare la scittura del contenuto di R_s in R_d.

Esercizio 3



Esercizio 4

- Scrittura del contenuto del registro sorgente, R1, nel registro tampone TEMP2:

SMUX = 1 /*i bit 6-8 dell'IR vengono trasferiti al decoder*/
$R_M = 1$ /* scrittura del contenuto di R1 sul bus*/
$W_{t2} = 1$ /*scrittura del contenuto del bus su TEMP2*/

- specificazione del tipo di operazione da effettuare che lo shifter deve fare (cioè nessuna, dato che deve far passare solo il contenuto dei TEMP2 sul bus):

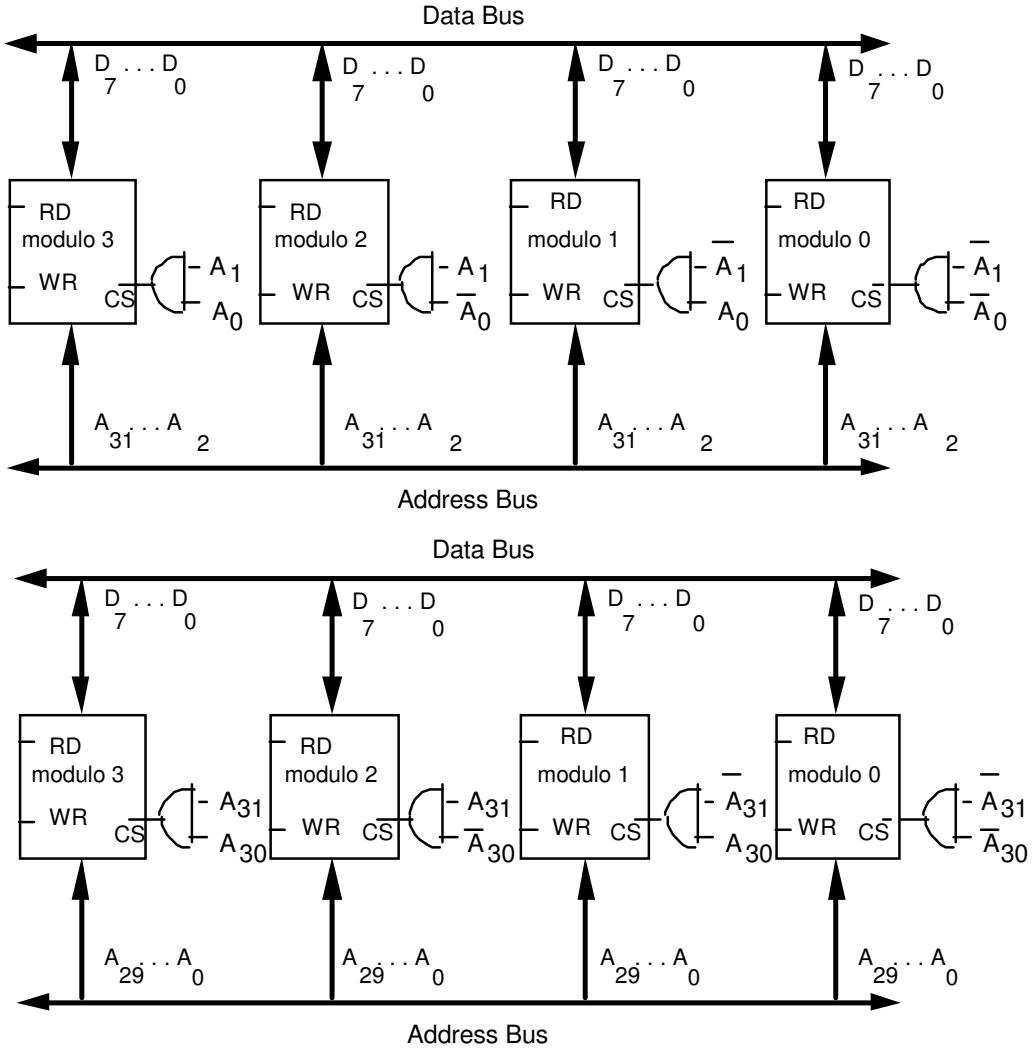
$opcode_{shifter}$ = nop

- scrittura del valore in uscita dello shifter sul registro destinatario, R5:

SMUX = 0 /*i bit 0-2 dell'IR vengono trasferiti al decoder*/
BS = 1 /* scrittura del valore in uscita dello shifter sul bus*/
$W_M = 1$ /*scrittura del contenuto del bus su R5*/

Esercizio 5

Di seguito sono presentate due soluzioni, la prima usa i bit meno significativi per identificare i moduli di memoria, mentre la seconda i bit più significativi.



Esercizio 6

/ ciclo di scrittura in periferica*/*

IR16-23 ->I/OAR;/*memorizzazione identificatore periferica sul registro tampone I/OAR*/

R1 -> I/ODR; /*memorizzazione dato da scrivere sulla periferica nel registro tampone I/ODR*/

I/ODR -> (I/OAR); /*memorizzazione del contenuto dell' I/ODR nella periferica puntata da I/OAR*/