ATTENZIONE!
SI PREGA DI NON COPIARE LE RISPOSTE UGUALI A COME SONO SCRITTE QUI DI SEGUITO, MA CAMBIARE ALCUNE PAROLE IN MODO TALE CHE NON SI CAPISCA. GRAZIE.

Capitolo 1: Introduzione al calcolatore

[1] 1.1 Algoritmi di calcolo manuale

D: Con l'avvento delle moderne macchine calcolatrici, si può pensare che algoritmi di esecuzione manuale delle operazioni aritmetiche non abbiano piu alcuna utilità pratica. Esporre e motivare la propria opinione a tal riguardo.

R: A mio parere gli algoritmi di esecuzione manuale delle operazioni aritmetiche hanno ancora un'utilità pratica per eventuali controlli del risultato calcolato dalle macchine calcolatrici. Inoltre conoscere le procedure manuali ci potrebbe aiutare nella creazione di nuovi algoritmi.

[2] 1.2 Precursori dei calcolatori moderni

D: Individuare caratteristiche di calcolatori moderni, ovvero costruiti dopo la meta del XX secolo, che sono riscontrabili anche in macchine o modelli di calcolo antecedenti tale periodo, e indicare in quali.

R: L'architettura dei calcolatori fornita da Von Neumann, che puo essere considerata lo sviluppo dell'idea della macchina di Turing universale, ovvero una macchina dotata di codifica non solo di dati, ma anche di istruzioni.
La codifica delle informazioni, che puo essere ricondotta agli studi di Leibniz sulla formalizzazione della logica nei numeri.
L'idea di una macchina programmabile, che si puo ricondurre alle schede perforate usate nei telai di Jaquard.

[3] 1.3 Grandi creatori dei calcolatori moderni

D: Indicare le tre persone che si ritiene abbiano avuto la maggiore influenza nell'evoluzione dei calcolatori moderni, e descriverne brevemente i contributi piu rilevanti.

R: Pascal: che costrui un calcolatore meccanico in grado di eseguire semplici operazioni.
Babbage: ideo l' Analytical Engine, un calcolatore meccanico programmabile.
Von Neumann: invento l' EDSAC, una macchina a valvole dotata di CPU e capace di memorizzare un programma in memoria.

[4] 1.4 Durata della legge di Moore

D: Fornire una ragione per la quale è lecito considerare inverosimile che la legge di Moore resti valida ancora a lungo.

R: La legge di Moore enuncia che le prestazioni dei processori e il numero dei transistor che lo compongono raddoppiano ogni 18 mesi. E' lecito considerare che sia inverosimile in quanto presto l'elevato ridimensionamento dei transistor farebbe entrare in gioco le leggi della meccanica quantistica.

[5] 1.5 Applicazioni di tipi di calcolatori

D: Indicare quattro esempi di applicazioni per le quali siano rispettivamente adatti quattro tipi diversi di calcolatori.

R: Grid: elaborazione di grandi quantita di dati per la ricerca scientifica.
Workstation: produzione audio/video professionale.
Server: servizi di hosting.
Embedded: elettrodomestici.

[6] 1.6 Esempi di microprocessori incorporati

D: Indicare tre elettrodomestici che potrebbero funzionare con microcontrollore incorporato, e spiegare la funzione di quest'ultimo.

R: Lavatrice: il microcontrollore si occupa della gestione dei programmi di lavaggio.
Asciugacapelli: il microcontrollore si occupa della gestione della temperatura e della potenza del getto d' aria.
Forno a Microonde: gestione dei programmi di cottura.

[7] 1.7 Dispositivi di I/O e accessibilità

D: Indicare tre dispositivi di I/O utili a sopperire a limitazioni delle capacita fisiche di persone diversamente abili, e in qual modo si realizzi tale aspetto della loro utilita.

R: Puntatori oculari: permettono la comunicazione e la gestione del pc attraverso il movimento della pulilla.
Terminale braille: permette agli utenti cechi di interagire con un pc.
Interfaccia neurale: consente la comunicazione tra il cervello ed un dispositivo quale ad esempio il computer.

[8] 1.8 Utilità della memoria cache

D: A cosa si deve il guadagno di prestazioni che si ha con l'impiego di memorie cache nell'architettura di un calcolatore?

R: Si deve all' elevata velocita della memoria cache, che spesso risiede nello stesso circuito integrato del processore, proprio per aumentarne al massimo le prestazioni. Inoltre quando delle istruzioni vengono eseguite ripetutamente, queste vengono prelevate velocemente dalla loro copia presente nella memoria cache.

[9] 1.9 Molteplicità dei livelli di cache

D: Perchè è conveniente avere piu livelli di memoria cache nell'architettura di un calcolatore?

R: E conveniente avere piu livelli di cache perche il dato richiesto viene cercato a partire dalla cache di livello piu basso (piu veloce) a quella di livello piu alto (piu lenta), e tutto cio che si trova in una cache deve trovarsi anche nelle cache di livello superiore. Cio porta ad un incremento delle prestazioni.

[10] 1.10 Località degli operandi della ALU

D: Quale ragione giustifica il vincolo, imposto dall'architettura di un calcolatore, che la ALU possa operare solo su registri interni al processore, e non dunque anche su operandi in locazioni di memoria?

R: La ALU può operare solo su registri interni al processore perche questi operano ad una velocita elevatissima. Operare su operandi in locazione di memoria porterebbe ad un sostanziale rallentamento delle operazioni.

[11] 1.11 Interfaccia processore-memoria

D: La struttura interna del processore schematizzata nella Figura 1.2 del testo mostra un'interfaccia processore-memoria, a supporto del flusso di informazioni tra locazioni di memoria e registri del processore. Perche conviene disporre di tale componente nell'organizzazione del processore, piuttosto che di connessioni dirette fra memoria e registri?

R: L'interfaccia processore memoria serve a controllare il flusso di informazione e a garantire protezione, rilocazione e condivisione, caratteristiche che non si potrebbero avere con delle connessioni dirette fra memoria e registri.

[12] 1.12 Aggiornamento automatico del registro PC

D: Perche conviene che, dopo il prelievo di un'istruzione, il contenuto del registro PC sia aggiornato con l'indirizzo dell'istruzione successiva in memoria piuttosto che con quello dell'istruzione prelevata?

R: Perche dopo il prelievo di un' istruzione, questa viene eseguita, quindi il PC punta subito all' istruzione successiva in memoria, non ha senso che il PC punti di nuovo l' istruzione precedentemente prelevata, in quanto e gia stata messa in esecuzione.

[13]1.13 Modifica del contenuto del registro PC

D:Quali tipi di istruzioni possono modificare il contenuto del registro PC per effetto della loro esecuzione (non del mero prelievo)?

R: Le istruzioni di jump, ad esempio quelle dei cicli, quando il blocco di istruzione viene terminato non si passa subito alla prossima istruzione, ma viene ripetuto il ciclo.

[14] 1.14 Modifica del contenuto del registro IR

D: Ha senso considerare un tipo di istruzioni la cui esecuzione modifichi il contenuto del registro IR? Se si, a qual fine? Se no, perche?

R: Non credo abbia senso considerare istruzioni che modificano il contenuto di IR in quanto questo registro e nato appositamente per mantenere stabile l' istruzione prelevata in memoria. Un'istruzione che modifichi l'IR comporterebbe un arresto anomalo della normale routine di esecuzione delle istruzioni.

[15]1.15 Passi operativi elementari nell'esecuzione di un'istruzione Store

D:Elencare i passi necessari a eseguire l'istruzione macchina:

Store R4, LOC

in termini di interazioni tra i componenti mostrati nella Figura 1.2 del testo. Assumere che l'indirizzo della locazione di memoria contenente questa istruzione sia inizialmente nel registro PC.

R: I passi necessari per eseguire l'istruzione macchina

Store R4, LOC

sono i seguenti:

1. Mandare l'indirizzo della parola dell'istruzione dal registro PC alla memoria ed emettere un comando di controllo Read.
2. Attendere che la parola richiesta sia stata prelevata dalla memoria, poi caricarla nel registro IR, dove viene interpretata (decodificata) dal circuito di controllo per determinare l'operazione da eseguire.
3. Incrementare il contenuto del registro PC per puntare alla prossima istruzione in memoria.
4. Mandare il valore dell'indirizzo LOC dall'istruzione nel registro IR, insieme col contenuto del registro R4, alla memoria ed emettere un comando di controllo Write.
5. Attendere che il contenuto del registro R4 sia stato scritto in memoria.

[16]1.16 Passi operativi elementari nell'esecuzione di un'istruzione Add

D: Elencare i passi necessari a eseguire l'istruzione macchina:

Add R4, R2, R3

in termini di interazioni tra i componenti mostrati nella Figura 1.2 del testo. Assumere che l'indirizzo della locazione di memoria contenente questa istruzione sia inizialmente nel registro PC.

R: I passi necessari per eseguire l'istruzione macchina

Add R4, R2, R3

sono i seguenti:

1. Mandare l'indirizzo della parola dell'istruzione dal registro PC alla memoria ed emettere un comando di controllo Read.

2.Attendere che la parola richiesta sia stata prelevata dalla memoria, poi caricarla nel registro IR, dove viene interpretata (decodificata) dal circuito di controllo per determinare l'operazione da eseguire.

3. Incrementare il contenuto del registro PC per puntare alla prossima istruzione in memoria.

4. Mandare il contenuto dei registri R2 e R3 alla ALU ed emettere il comando Add alla ALU.

5. Mandare la somma dall'uscita della ALU al registro R4.

[17] 1.17 Salvataggio di registri per il servizio di interruzioni

D: Considerando i tre registri PC, IR, Ri (un registro di uso generale), per un corretto servizio di una richiesta di interruzione accade che uno di questi registri sia sempre da salvare, un altro lo sia talvolta ma non sempre, un altro mai. Dire quale caso si applica a ciascuno dei suddetti registri e perche.

R: Il PC va sempre salvato per poter, alla fine dell'interruzione, riprendere dall'indirizzo in cui ci si trovava.
Il registro generico Ri potrebbe essere salvato se ci si aspetta che l'interruzione possa sovrascriverne i dati.
Il registro IR non va mai salvato perche il suo contenuto viene caricato sempre all'inizio del ciclo, con l'istruzione puntata dal registro PC prima che questo venga incrementato.

[18] 1.18 Nidificazione di interruzioni

D: Si supponga che una richiesta di interruzione venga generata da un dispositivo di I/O mentre il processore sta eseguendo una routine di servizio di (un'altra) interruzione. Ha senso permettere che questa esecuzione
venga essa stessa sospesa per servire la nuova richiesta? Se si, a qual fine? Se no, perche?
R: Ha senso nel caso in cui l'interruzione generata dal dispositivo I/O abbia una priorita maggiore rispetto a quella in esecuzione.

[19] 1.19 Pipelining del ciclo prelievo-esecuzione

D: Nel pipelining del ciclo prelievo-esecuzione si possono condurre in parallelo l'esecuzione dell'istruzione corrente e il prelievo della successiva. Cosa accade di un'istruzione prelevata se l'istruzione corrente, eseguita in parallelo con tale prelievo, modifica il contenuto del registro PC?
R: Il prelievo di un'istruzione è un momento in cui la cpu non esegue delle istruzioni.
Parallelizzando l'esecuzione di un'istruzione ed il prelievo della successiva, l'esecuzione del programma verr à completata più velocemente perchè le istruzioni verranno eseguite consecutivamente senza pause tra loro.
Questo è il vantaggio che porta ad un miglioramento delle prestazioni.

[20] 1.20 Conversione dalla rappresentazione binaria naturale a quella decimale

D: Calcolare il valore dei seguenti numeri in rappresentazione binaria naturale, esprimendolo nella usuale rappresentazione decimale: 1011010, 110110, 111011, 10001011, 1010111.
R: Rispettivamente: 90, 54, 59, 139, 87.

[21] 1.21 Conversione dalla rappresentazione decimale a quella binaria naturale

D: Fornire la rappresentazione binaria naturale dei seguenti numeri, espressi nella usuale rappresentazione decimale: 45, 89, 160.
R: Rispettivamente: 101101, 1011001, 10100000.

[22] 1.22 Addizione nella rappresentazione binaria naturale

D: Eseguire le seguenti addizioni nella rappresentazione binaria naturale a 8 bit: 01011010 + 00110110, 00111011 + 10001011, 01010111 + 10100110.
R: Rispettivamente: 10010000, 11000110, 11111101.

[23] 1.23 Sottrazione nella rappresentazione binaria naturale

D: Eseguire le seguenti sottrazioni nella rappresentazione binaria naturale a 8 bit: 01011010 - 00110110, 10001011 - 00111011, 10100110 – 01010111.
R: Rispettivamente: 00100100, 01010000, 01001111.

[24] 1.24 Conversione dalla rappresentazione decimale a quella in complemento a due

D: Fornire la rappresentazione binaria a 8 bit in complemento a due dei seguenti numeri interi negativi, espressi nella usuale rappresentazione decimale: -45, -89, -128.
R: Da decimale a complemento a 2
● Se numero >= 0 allora:
0| binario puro
● Se numero < 0 allora:
scrivere numero in binario puro
complementare bit a bit, segno compreso
aggiungere +1
Rispettivamente: 11010011, 10100111, 10000000.

[25] 1.25 Conversione dalla rappresentazione in complemento a due a quella decimale

D: Calcolare il valore dei seguenti numeri interi nella rappresentazione binaria a 8 bit in complemento a due, esprimendolo nella usuale rappresentazione decimale: 11111111, 10000011, 11110000.
R: Da complemento a 2 a decimale
● Se bit segno = 0
binario puro da convertire in decimale
● Se bit segno = 1
complementare bit a bit segno compreso

sommare +1
convertire in decimale
aggiungere il segno -
Rispettivamente: -1, -125, -16.

[26] 1.26 Somma algebrica in complemento a due

D: Eseguire le seguenti somme algebriche nella rappresentazione binaria a 8 bit in complemento a due, e indicare in ciascun caso se si verifica un trabocco: 10100011 + 00101100, 00110111 + 01011001, 11111111 +10000001.

R: Una somma di due numeri di n cifre in complemento a 2 da (errore di) overflow se e solo se i riporti in colonna n e n +1 sono diversi.
Rispettivamente: 11001111, 10010000 (dove e presente un overflow, in quanto i riporti delle colonne n ed n+1 differiscono), 10000000.

[27] 1.27 Trabocco nella sottrazione in complemento a due

D: Specificare le regole di rilevazione del trabocco nella sottrazione in complemento a due (fornite nel testo per l'addizione), esprimendole in termini di concordanza di segno di minuendo, sottraendo e risultato.

R: Avremo un trabocco se i segni del minuendo e del sottraendo sono discordi e il bitsign del risultato non sarà concorde al bitsign del minuendo.

[28] 1.28 Sottrazione in complemento a due

D: Eseguire le seguenti sottrazioni nella rappresentazione binaria a 8 bit in complemento a due, e indicare in ciascun caso se si verifica un trabocco: 10101011 - 00101100, 00110111 - 01011001, 11111111 - 10000001.

R: I risultati sono rispettivamente :
01111111 (overflow), 11011110, 01111110.

[29]1.29 Rappresentazione dei razionali in virgola mobile

D:Si consideri una rappresentazione a 32 bit dei razionali in virgola mobile (con base 2) in cui si dedicano 8 bit alla rappresentazione dell'esponente con segno e si conviene che la mantissa rappresenti la parte frazionaria di un numero razionale nell'intervallo [1,2[(la cui parte intera è dunque implicitamente 1). Qual è l'intervallo di razionali rappresentabili? Qual è la precisione *relativa* di tale rappresentazione?

R: 1) un bit per il segno
2) 8 bit per l'esponente
3) 23 bit per la mantissa

fanno si che siano 32 bit totali

la mantissa è compresa fra [1,2[
l'esponente è compreso tra [-128,127]
il numero massimo rappresentabile con la mantissa è 2^127
il più piccolo invece *2^-128
la virgola è utile per la precisione del numero

[30] 1.30 Codici correttori

D: Come si spiega che un codice di correzione di errore con distanza di Hamming *n* possa rivelare fino a *n* - 1 errori e ne possa correggere fino a (*n* -1)/2 ?

R: Il codice correttore può rivelare fino a n-1 errori perche con un numero maggiore di errori la sequenza può non essere riconosciuta come la corruzione di una sequenza.
Il codice può correggere fino a (n-1)/2 errori perche e possibile determinare con precisione quale fosse la sequenza (più vicina) originaria.

[31] 1.31 Posizione dei bit ridondanti nei codici di controllo di errore

D: Si supponga di voler realizzare un codice di controllo di errore modificando un codice di lunghezza fissa dato, per esempio ASCII per la rappresentazione di caratteri, estendendolo con un numero fisso di bit ridondanti, i cui valori vengono opportunamente definiti in modo che il codice cosi ottenuto abbia una certa distanza di Hamming specificata. E importante scegliere opportunamente anche la posizione dei bit ridondanti? Se si, a qual fine? Se no, perche?

R: Si, la posizione dei bit ridondanti e importante. Ad esempio nell'algoritmo di Hamming
i bit ridondanti sono posizionati nelle potenze di 2. Questo perche l' obiettivo dei bit di controllo e quello di individuare la posizione dell'errore attraverso un confronto XOR tra i bit di controllo ricevuti e i bit di controllo calcolati dalla sequenza ricevuta.

[32] 1.32 Capacità di un archivio di immagini

D: Si abbia un archivio di immagini fotografiche digitali bitmap da 16 milioni di pixel ciascuna, a 3 byte/pixel di profondità di colore. Quante ne può ospitare una memoria da 2 GB senza compressione? (1 GB = 1024 x 1024 x 1024 byte).

R: Calcolo il numero di immagini ospitabili nell'archivio con la seguente formula:
n=(capacita archivio)/(peso immagine)
esprimendo tutto nell'unita di misura del byte.
Sostituisco quindi i valori:
n=[2(1024)^3]/[(16*10^6)3]= 44,74.
Concludo dicendo che l' archivio puo contenere 44 immagini di quella tipologia.

[33] 1.33 Vantaggi della rappresentazione SVG per la grafica vettoriale

D: Quali sono i principali vantaggi della rappresentazione SVG rispetto a quella bitmap per la rappresentazione di immagini quali diagrammi o caratteri tipografici?
R: I principali vantaggi sono:
1- possibilità di ingrandire l'immagine arbitrariamente senza che questa ne perdi in definizione;
2- possibilità di occupare meno spazio per dati equivalenti al bitmap. (Ad esempio, occupo meno dati a scrivere l'equazione per tracciare una retta che per disegnare una retta in bitmap)

[34] 1.34 Limite inerente della compressione generica senza perdita

D: Perché è impossibile che un tecnica di compressione generica senza perdita abbia un rapporto di compressione maggiore di 1 per qualsiasi sequenza binaria da comprimere?
R: Avendo:
O. File originale
C. File compresso
E posto che C<O (per stessa definizione di compressione), il rapporto di compressione sappiamo essere C/O, che per la condizione prima imposta implica che C/O sia minore di uno. Non si può dunque parlare di compressione se C/O>1, perché C sarebbe maggiore di O e significherebbe inoltre che abbiamo aggiunto dati al compresso, alterando il file originale.

[35] 1.35 Impiego della codifica relativa nella compressione di sequenze video

D: Quale aspetto tipico delle sequenze video rende vantaggioso l'impiego di una codifica relativa per la loro compressione?
R: L' aspetto che rende vantaggioso l'impiego di una codifica per la compressione delle sequenze video, e il fatto che immagini contigue nella sequenza differiscono di poco (tranne nel caso poco frequente del cambio di scena).

Integrazione 1: Aritmetica Maya

[36] I1.1 Vantaggi della rappresentazione Maya delle cifre per l'addizione e la sottrazione

D: Descrivere brevemente gli eventuali vantaggi della rappresentazione Maya delle cifre rispetto a quella comune con cifre arabe, con riferimento alle operazioni aritmetiche additive (addizione e sottrazione).
R: I vantaggi della rappresentazione maya sono:
- la rappresentazione dei numeri attraverso 3 soli simboli (caracol,frijolito,palito) rispetto ai dieci della rappresentazione araba.
- Il vantaggio nelle operazioni aritmetiche nell'eliminare i calcoli complessi delle operazioni aritmetiche arabe.

[37] I1.2 Rappresentazioni esadecimale e ottale con cifre Maya

D: Come si puo modificare la rappresentazione Maya dei numeri, usando gli stessi oggetti per formare le cifre, per avere una rappresentazione esadecimale (cioe in base 16)? Come cambiano le regole di equivalenza di gruppi di oggetti sull'abaco? Risolvere il problema analogo per la rappresentazione ottale (in base 8).
R: Per la rappresentazione esadecimale:
1 palito=4 unita
4 palito=1 unita nella posizione adiacente piu significativa.
Per la rappresentazione ottale:
1 palito=2 unita
4 palito=1 unita nella posizione adiacente piu significativa

[38]1.3 Correttezza dell'algoritmo R

D:Perché occorre fu il contenuto di una casella dell'abaco, nell'algoritmo R di conversione, in ciascun passo di discesa lungo la diagonale secondaria?
R: Poichè i Maya contavano in base 10 in orizzontale, dove la cifra più significativa stava a sinistra e quella meno significativa a destra, mentre contavano in base 20 in verticale, dove la cifra più significativa stava in alto e quella meno significativa in basso. Ne segue che in ciascun passo di discesa lungo la diagonale secondaria, dovendo passare da base 20 a base 10, il contenuto della casella dell'abaco deve essere duplicato.

[39]1.4 Applicabilità dell'algoritmo R per la conversione esadecimale-ottale

D: Si consideri la conversione dei numeri in rappresentazione posizionale fra base 16 e base 8. Adattando opportunamente a entrambi i casi la rappresentazione Maya delle cifre, e applicabile l'algoritmo R di conversione, eventualmente con modifiche? Se si, con quali modifiche? Se no, perche?
R: L'algoritmo R di conversione e applicabile in questo caso a patto di alcune modifiche:
Invece di scrivere la cifra finale in fondo all' abaco in modulo 10, la si deve scrivere in modulo 8. Inoltre il relativo quoziente della divisione per 8 deve essere riportato nel riquadro a sinistra.

[40]1.5 Scalabilità dell'algoritmo R

D: Nella sua formulazione più semplice, l'algoritmo R per la conversione vigesimale-decimale prevede la duplicazione del contenuto delle caselle in ciascun passo di discesa lungo la diagonale secondaria, per applicare poi le regole di equivalenza della rappresentazione decimale alla configurazione di arrivo. Così formulato l'algoritmo non è scalabile, poiché comporta una crescita esponenziale del contenuto delle caselle lungo la discesa. Come lo si può modificare per evitare questo inconveniente e ottenere quindi un algoritmo scalabile?

R: Per evitare questo inconveniente e ottenere quindi un algoritmo scalabile, potremmo considerare le celle divise lungo la diagonale secondaria, esattamente come nelle operazioni di moltiplicazione e divisione.
A questo punto, ad ogni passo di discesa lungo la diagonale secondaria, dopo aver duplicato il contenuto delle celle e averlo inserito nella parte inferiore delle celle poste lungo la diagonale secondaria, potremmo applicare opportune regole di equivalenza mettendo 1 maisito nella parte superiore della cella, per ogni 4 palitos presenti nella parte inferiore della cella, ed eliminando questi ultimi.
In questo modo ridurremmo la crescita esponenziale del contenuto delle caselle lungo la discesa. Alla fine basterà semplicemente riconvertire il valore degli elementi contenuti nella parte superiore di ogni cella e applicare le relative regole di conversione in base 10.

[41]1.6 Uso dell'abaco Maya per la moltiplicazione "alla musulmana"

D: Nell'esempio di moltiplicazione "alla musulmana" con cifre Maya decimali illustrato nel testo della lezione, la significativita delle posizioni cresce progressivamente verso l'alto e verso sinistra, gli operandi sono rispettivamente disposti sul lato orizzontale in alto e su quello verticale a destra, e il risultato e ottenuto lungo gli altri due lati, ovviamente contigui, dove la posizione successiva a quella piu a sinistra del lato orizzontale e quella piu in basso del lato verticale. Rispettando la stessa convenzione sull'ordine di significativita delle posizioni, nonche il requisito che il risultato debba ottenersi lungo due lati contigui, e possibile disporre diversamente operandi e risultato lungo i quattro lati dell'abaco? Se si, in quanti e quali altri modi? Se no, perche?

R: Essendo dato dal testo che la posizione successiva a quella piu a sinistra del lato orizzontale del risultato e quella piu in basso del lato verticale, si puo notare che il risultato puo essere messo solo nel lato verticale a sinistra e orizzontale in basso. Quindi possiamo disporre diversamente solo gli operandi. Avendo il prodotto la proprieta commutativa, scambiando gli operandi di posto non andiamo a commettere errori.

[42]1.7 Parallelismo di passi elementari nella moltiplicazione "alla musulmana" sull'abaco Maya

D: Con riferimento all'esecuzione "alla musulmana" della moltiplicazione sull'abaco con rappresentazione Maya delle cifre, di quali operazioni piu elementari si compone? Quali di queste possono essere eseguite in parallelo da piu agenti di calcolo? Quali dipendono da (quali) altre?

R: La moltiplicazione alla musulmana si compone di due operazioni elementari:
- La moltiplicazione di ogni cifra della riga per ogni cifra della colonna.
- La somma delle diagonali.

Entrambe le operazioni possono essere eseguite in parallelo ma, la somma dipende dalla moltiplicazione.

[43]1.8 Moltiplicazione "alla musulmana" sull'abaco nelle rappresentazioni binaria e ternaria

D: Si consideri l'esecuzione "alla musulmana" della moltiplicazione nella rappresentazione binaria naturale. E ancora necessario suddividere in due parti le caselle dell'abaco o e superfluo? Perche? E in una rappresentazione posizionale ternaria (in base 3)?

R: Nella rappresentazione binaria non credo sia necessario suddividere in due parti le caselle perche in nessun caso abbiamo un riporto significativo.
Per quanto riguarda la rappresentazione posizionale ternaria non possiamo non considerare il riporto, quindi e necessario suddividere le caselle dell' abaco in due parti.

[44]1.9 Coordinamento dell'esecuzione parallela dell'addizione sull'abaco Maya

D: Si supponga di disporre di $n+1$ agenti di calcolo per l'esecuzione parallela dell'addizione di due numeri da n cifre in simboli Maya. Ciascun agente è associato a un'unica posizione delle cifre di addendi e risultato, E può comunicare con gli agenti associati alle posizioni adiacenti. La comunicazione consiste nello scambio di simboli Maya e/o di eventuali altri segnali. Escogitare delle regole di comunicazione fra agenti associati a posizioni adiacenti perché si produca un risultato finale corretto assieme a un segnale di fine dell'esecuzione.

R: Supponiamo di scrivere i due addendi in verticale, in due colonne, cosi che le cifre significative stiano più in alto e quelle meno significative più in basso.

Utilizzeremo n agenti per la gestione delle somme in parallelo. L'agente "libero" verrà utilizzato in caso di un eventuale riporto e per terminare l'esecuzione. Il risultato dell'operazione verrà sovrascritto sulla colonna a sinistra, ovvero la colonna del primo addendo.

Gli agenti della colonna destra sono in grado di leggere il numero e inviare messaggi.

Quelli della colonna sinistra possono sia leggere messaggi che inviarli. Inoltre devono essere in grado di ridefinire i numeri e scorporare eventuali riporti (cosi da non rallentare ulteriormente il calcolo).
L'agente "libero" deve poter leggere messaggi e forzare la fine dell'esecuzione (o comunque notificarla inviando un messaggio).

La somma è cosi gestita:

1) Gli agenti del secondo addendo (colonna di destra) invieranno all'agente alla loro sinistra un messaggio contenente il valore della cifra che loro stanno gestendo. In tal maniera saremo sicuri che la somma avviene rispettando l'ordine corretto delle cifre e la loro significatività;

2) Gli agenti del primo addendo ricevono il messaggio, sommano il loro numero con quello indicato sul messaggio, salvano momentaneamente il risultato e inviano un messaggio all'agente inferiore per indicare che hanno completato l'operazione e che sono in attesa di un eventuale riporto (se esiste). Si mettono quindi in attesa;

3) Non appena l'agente che sta più in basso in colonna (ovvero quello addetto alla cifra meno significativa del risultato) riceve il messaggio di completamento dell'operazione dall'agente che si trova sopra di lui, ridefinisce la somma che aveva calcolato precedentemente separando (se esiste) il riporto, stampa il numero che ne ricava ed invia quindi un messaggio all'agente sopra di lui (che gestisce la cifra di 1 posto a sinistra) indicando un eventuale riporto e un messaggio di stato per indicargli di continuare l'operazione;

4) L'agente in posizione superiore riceve il messaggio da quello in posizione inferiore. Se non ha ancora ricevuto il messaggio dall'agente sopra di lui che gli diceva di essere in attesa del riporto, aspetta questo messaggio. Altrimenti, se lo ha già ricevuto, svolge gli stessi passaggi già indicati. Si continua quindi così finchè non si arriva all'ultimo agente più in alto che invierà questo stesso messaggio all'agente che abbiamo definito "libero";

5) L'agente "libero" ha come compiti il trascrivere un eventuale riporto (se esiste) e sarà lui ad indicare che l'operazione è stata completata;

[45]1.10 Algoritmi per l'estrazione di radice quadrata con simboli Maya

D: Si consideri l'operazione di estrazione di radice quadrata di un numero intero, con risultato intero approssimato per difetto. L'algoritmo per realizzarla sull'abaco Maya, con simboli Maya nella rappresentazione in base dieci, realizzato nel simulatore presentato in aula, procede per successive approssimazioni per difetto, sfruttando:

1. una certa analogia con l'operazione di divisione (anche il quoziente è approssimato per difetto),
2. la formula del quadrato del binomio,
3. la posizionalità della rappresentazione, tenendo in conto che $\sqrt{(xb^{2n})} = (\sqrt{x})b^n$.

L'algoritmo si estende facilmente ai numeri razionali (in modo analogo a quello della divisione). Considerando altri algoritmi, per esempio l'algoritmo babilonese, ci si può chiedere se siano più o meno semplici da realizzare sull'abaco Maya. Analizzare il problema ed esporre le proprie considerazioni in merito.

R:

Integrazione 2: Strutture algebriche, algebre di Boole

[46]2.1 Semigruppi aritmetici

D: Si considerino le algebre costituite dall'insieme N dei numeri naturali e da una delle operazioni aritmetiche binarie di: addizione, moltiplicazione, elevamento a potenza. Sono tutte dei semigruppi? Se no, perche?

R: Abbiamo un semigruppo solo nel caso delle operazioni binarie di addizione e moltiplicazione. Con l'elevamento a potenza invece no, in quanto non e un operazione binaria associativa.

[47]2.2 Monoide di autofunzioni

D: Si consideri il semigruppo che ha per sostegno l'insieme delle autofunzioni $f : S \rightarrow S$ su un insieme S, e la composizione di funzioni quale operazione binaria. Quale elemento del sostegno permette di considerare quest'algebra come un monoide? E possibile considerarla come un gruppo? Se si, qual e l'inverso di un dato elemento del sostegno? Se no, perche?

R: L'identita su S (ovvero Ids) e l'elemento neutro:
f o IDs = IDs o f = f con f,IDs appartenenti al sostegno;
Quindi l'algebra ha un elemento neutro e puo essere considerata un monoide.

La funzione inversa di f (ovvero f^-1) e l'elemento inverso:
f o f^-1 = f^-1 o f = IDs
Quindi il monoide ha un elemento inverso e puo considerarsi gruppo.

[48]2.3 Gruppo commutativo di quattro elementi

D: Verificare che l'algebra avente come sostegno l'insieme dei numeri naturali dispari minori di 8, e la moltiplicazione modulo 8 come operazione binaria, costituisce un gruppo commutativo. Qual e l'elemento neutro? E l'inverso di ciascun elemento?

R: Per verificare che tale algebra costituisce un gruppo commutativo bisogna dimostrare che possiede sia un elemento neutro sia l'inverso di ogni elemento. Avra come elemento neutro 1 poiche ogni numero appartenente all'insieme moltiplicato per esso modulo 8 da come risultato il numero stesso. L'inverso di ogni elemento, invece, e l'elemento stesso, in quanto se dal prodotto dell'elemento per se stesso sottraiamo 1 avremo sempre un multiplo di 8. L' algebra in questione e quindi un gruppo, e, essendo il prodotto un' operazione che gode della proprieta commutativa, la nostra algebra e un gruppo commutativo.

[49]2.4 Semianello di linguaggi

D: Si consideri un'algebra dei linguaggi L su un dato alfabeto A, dove il sostegno dell'algebra e costituito dagli insiemi di stringhe finite di simboli in A (i linguaggi su A), con le operazioni di unione e concatenazione di linguaggi, e con due costanti: il linguaggio vuoto e il linguaggio (non vuoto) che contiene solo la stringa vuota. Dimostrare che tale algebra e un semianello. E un semianello commutativo?

R: Un semianello e una struttura algebrica formata da un insieme (In questo caso i linguaggi L definiti sull'alfabeto A) e munito di due operazioni binarie(Unione e Concatenazione), vediamo se la nostra algebra verifica le seguenti proprieta dell'Unione e della Concatenazione:
-Unione e Concatenazione sono operazioni associative: si ha cioe (L(a) U L(b)) U L(c) = L(a) U (L(b) U L(c)) e (L(a) o L(b)) o L(c) = L(a) o (L(b) o L(c)) per ogni terna di insiemi di stringhe(a,b,c) di caratteri dell'alfabeto A.
-Esiste un (unico) elemento neutro per l'unione vale a dire L(a) U 0(Sarebbe un 0 sbarrato) = 0(Sarebbe un 0 sbarrato)) U L(a) = L(a).
-La concatenazione e distributiva rispetto all'Unione, vale a dire (L(a) U L(b)) o L(c) = L(a) o L(c) U L(b) o L(c) e L(a) o (L(b) U L(c)) = L(a) o L(b) U L(a) o L(c) per ogni terna di insiemi di stringhe(a,b,c) di caratteri dell'alfabeto A.
- Per ogni carattere in A si ha che L(a) o 0(sarebbe un 0 sbarrato) = 0(sarebbere uno 0 sbarrato) o L(a) = 0(sarebbe uno 0 sbarrato).
Valendo Unione e Concatenazione possiamo allora dire che la nostra algebra e un semianello.
La concatenazione di linguaggi non gode della proprieta commutativa:
L(a) o L(b) != L(b) o L(a) (ab e diverso da ba).
Per tale ragione il semianello non si puo definire commutativo.

[50]2.5 Reticolo sui multinsiemi? (ESATTA MA DA MODIFICARE NEL COMPITO)

D: Un multinsieme M = (S, μ) e costituito da un insieme S e da una funzione di molteplicita μ : S → N, dove μx e il numero di copie di x in M. Si possono estendere ai multinsiemi le operazioni di unione e intersezione, in modo che l'algebra cosi ottenuta sia un reticolo? Se si, come? Se no, perche?

R: Sia M'= (S',μ') e M'' = (S'',μ''), possiamo definire la loro unione come il multinsieme M=(S,μ) tale che S = S' U S'' e μx = max(μ'x,μ''x) per ogni x appartenente a S. In tale modo l'algebra formata dai multinsiemi e dall'operazione di unione potra essere considerata un semireticolo, in quanto vale la legge di idempotenza M U M = M.
Discorso analogo per l'intersezione tra M' e M'', che possiamo definire come il multinsieme M=(S,μ) tale che S= S' n S'' e μx = min(μ'x,μ''x) per ogni x appartenente a S. Anche quest'algebra e da considerarsi un semireticolo, in quanto vale la legge di idempotenza M n M = M.
Infine possiamo quindi definire l'algebra formata dai multinsiemi come sostegno e dalle operazioni di unione ed intersezione come reticolo, in quanto entrambe le sue ridotte sono semireticoli e valgono gli assiomi di assorbimento M' U (M' n M'') = M' e M' n (M' U M'') = M'.

[51]2.6 Reticolo sulle coppie di numeri naturali

D: Si consideri l'insieme N2 delle coppie (ordinate) di numeri naturali, parzialmente ordinato dalla relazione ≤ definita dalla condizione: (m, n) ≤ (p, q) se m ≤ p e n ≤ q. Verificare che tale ordinamento parziale costituisce un reticolo. Come sono definite le sue due operazioni?

R: Essendo l'insieme parzialmente ordinato per ogni coppia di elementi di N^2 potremo identificare un estremo superiore ed un estremo inferiore e cio comporta che l'algebra forma un reticolo. Avremo quindi due operazioni /\ e \/ sull'insieme sostegno tali che presa una coppia (x,y) appartenente a N^2, se x ≤y allora x /\ y = x e x \/ y = y.

[52]2.7 Base assiomatica equazionale dei reticoli

D: Dimostrare che ciascuno degli assiomi di idempotenza delle operazioni di reticolo e deducibile dagli altri sei assiomi equazionali di reticolo.

R: Le leggi necessarie alla dimostrazione sono quelle di assorbimento, ovvero:
x ∨ (x ∧ y) = x e x ∧ (x ∨ y) = x
x ∧ x = x
Dimostrazione 1

Poniamo y = x; avremo per la legge dell'assorbimento: x ∨ (x ∧ x) = x --> x ∧ x = x ∧(x ∨ (x ∧x)) = x
x ∨ x = x
Dimostrazione 2
Poniamo y = x; avremo per la legge dell'assorbimento: x ∧ (x ∨ x) = x --> x ∨ x = x ∨ (x ∧ (x ∨ x)) = x

[53]2.8 Definizione equazionale dell'ordinamento parziale nei reticoli algebrici (MOD.NEL.COMP.)

D: Dimostrare che da ciascuna delle due definizioni di ordinamento in un reticolo come abbreviazione equazionale conseguono riflessivita, antisimmetria e transitivita della relazione definita.

R: Un reticolo dotato di una relazione d'ordine possiede le due operazioni binarie ∨ e ∧
La prima definizione di ordinamento dice che: x<=y = x∨y = y
Mentre, equivalentemente, la seconda dice che: x<=y = x∧y = x
Dalle leggi di assorbimento possiamo notare che esse siano effettivamente equivalenti, infatti
a (a∧b)=a a∧(a b)=a
Dunque, sapendo che la relazione d'ordine <= e una relazione binaria tra elementi di un insieme A, potremo attribuire ad essa le seguenti proprieta:
x<=x V**x€**A riflessivita
x<=y e y<=x implica che x=y V**x,y €**A antisimmetria
x<=y e y<=z implica che x<=z V**x,y,z €**A transitivita

[54]2.9 Non-distributività dei reticoli M3 e N5 (MOD.NEL.COMP.)

D: Mostrare che i reticoli rappresentati dai diagrammi di Hasse M3 e N5 non sono distributivi (si suggerisce di numerare i vertici dei diagrammi).

R:

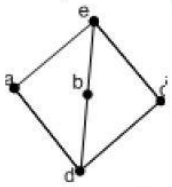
Un reticolo si dice distributivo se la relazione gode della distributività.
Definiamo dunque un reticolo distributivo se:

1) per ogni x,y,z appartenenti ad A $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

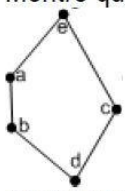
2) per ogni x,y,z appartenentdi ad A $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

ovvero bisogna provare che il reticolo gode della proprietà distributiva (sia dell'intersezione rispetto all'unione, sia dell'unione rispetto all'intersezione). Se esistono almeno 3 elmenti che non verificano la 1 o la 2 il reticolo non è distributivo.

Dunque, il diagramma di M3 (trirettangolo o diamente) è



In esso la distributività non si verifiica fra i 3 elmenti confrontabili, ovvero a,b,c.
Mentre quello di N5 (pentagono) è



Anche in esso la distributività non si verifica fra i 3 elmenti confrontabili a,b,c.

[55]2.10 Completezza dei reticoli finiti

D:Perché qualsiasi reticolo con sostegno di cardinalità finita è completo?

R: Ogni reticolo è completo quando ha cardinalità finita perché, prima di tutto, è possibile stabilire un'ordine(cosa necessaria in un reticolo) e secondariamente, se il sostegno del reticolo è un intervallo limitato, significa che il reticolo conterrà al suo interno un elemento minimo maggiorante ed un elemento massimo minorante, quindi è possibile stabilire la totalità del sostegno con l'operazione in esso definita(anch'essa totale), quindi ogni suo sottinsieme diverso da Ø avrà un minimo ed un massimo(insieme ben ordinato).

[56] 2.11 Algebra di Lindenbaum-Tarski (MOD.NEL.COMP.)

D: Le operazioni dell'algebra di Lindenbaum-Tarski sono definite mediante i rappresentanti delle classi di equivalenza argomento: perche tale definizione non dipende dai rappresentanti?

R: se ∧L interpreta ∧ in tale algebra, e [φ]L e la classe di equivalenza della proposizione φ, allora [φ]L ∧L [ψ]L =[φ ∧ ψ]L.

[57]2.12 Minimalità dell'algebra booleana minimale

D: L'algebra booleana minimale non e quella che ha la minima cardinalita del sostegno. Perche?

R: Perche l'algebra booleana minimale, unica a meno di isomorfismi, e detta minimale in quanto tale e l'insieme di equazioni che soddisfa, ovvero ogni equazione booleana valida in quest'algebra lo e in qualsiasi algebra booleana.

[58]2.13 Completezza funzionale di insiemi di operatori booleani

D: L'insieme degli operatori booleani {∨, ∧} non e funzionalmente completo. Perche?

R: L'inseme degli operatori booleani {∨, ∧} non e funzionalmente completo perche, mancando l'operatore di negazione NOT, non e possibile costruire qualsiasi funzione logica comunque complessa a partire dagli operatori che lo costituiscono. Tale insieme non soddisfa quindi la definizione di insieme booleano funzionalmente completo.

[59]2.14 Aciclicità delle reti combinatorie

D: Ogni rete combinatoria realizza una funzione booleana. Perche si classificano come reti combinatorie solo i circuiti logici privi di cicli?

R: Si classificano come reti combinatorie solo i circuiti privi di cicli poiché, non essendovi la ciclicità, ma svolgendo solo funzioni booleane dirette (le uscite in ogni istante sono funzione esclusivamente dei valori degli ingressi in quello stesso istante), non hanno bisogno di una memoria.
Circuiti logici con cicli sono invece quelli sequenziali, solitamente usati per la memoria di un calcolatore.

[60]2.15 Numerosità di insiemi di funzioni booleane

D: Quante sono le funzioni booleane in due variabili? E in tre variabili? E in *n* variabili?

R: Le funzioni Booleane in 2 variabili sono (2^2)^2=16
in 3 variabili sono (2^2)^3
in n variabili sono (2^2)^n

[61]2.16 Numerosità delle funzioni booleane commutative in due variabili

D: Quante sono le funzioni booleane commutative in due variabili?

R: Sono due, OR e AND, poiche:
OR = a+b = b+a
AND = a*b = b*a

[62]2.17 Numerosità delle funzioni binarie sui byte (MOD NEL COMP)

D: Quante sono le funzioni binarie che hanno due byte quali argomenti e producono un byte quale risultato?

R:

Notando che l'informazione rappresentabile tramite 1 byte è di 2^8 elementi, si deduce che il dominio di una ipotetica funzione binaria qualsiasi ha cardinalità 2^{16}e il codominio 2^8;
sapendo questi dati, tutte le possibili funzioni sono: $(2^8)^{(2^{16})} = 2^{(8*(2^{16}))} = 2^{(2^{19})}$

[63]2.18 Numerosità delle funzioni binarie commutative sui byte (MOD NEL COMP)

D: Quante sono le funzioni binarie commutative che hanno due byte quali argomenti e producono un byte quale risultato?

R: Le tabelle di verita di una funzione di due byte=16 bit ha 2^16 righe, dunque tutte le funzioni di due byte che hanno un byte come risultato sono (28)^2^16 (2 elevato all'ottava elevato a 2 alla sedicesima). Le funzioni in questione sono commutative se e solo se soddisfano f(x,y)=f(y,x), dunque se ne possono scegliere liberamente i valori (le immagini) solo sulla diagonale (x=y) e sulla meta delle altre righe, per un totale di 2^8+((2^16-2^8)/2) righe =(2^16+2^8)/2=2^15+2^7 righe. Quindi le funzioni commutative in questione sono (2^8)(2^15+2^7)

Appendice A: Elementi di logica della commutazione

[64] A.1 Forma SOP della funzione booleana di eguaglianza

D: Esprimere in forma di somma di prodotti (SOP) la funzione booleana binaria EQ, definita come complemento di XOR.

R: $\bar{x1}\ \bar{x2} + x1\ x2$

[65] A.2 Universalità delle porte logiche NAND e NOR (MOD.NEL.COMP.)

D: Quali regole dell'algebra di Boole giustificano la dimostrazione equazionale dell'universalita delle porte logiche NAND e NOR?

R: Per la costruzione della porta NOT: idempotenza; inoltre, per la costruzione delle porte AND e OR: teorema di De Morgan e doppia negazione.

De Morgan $\overline{x+y} = \overline{x} \cdot \overline{y}$ $\overline{x \cdot y} = \overline{x} + \overline{y}$

È sufficiente dimostrare che è possibile costruire le porte logiche AND, OR e NOT utilizzando solo porte NAND (NOR), in quanto ogni formula ammette le forme canoniche congiuntive e disgiuntive che fanno uso soltanto di porte AND, OR e NOT.

Dimostrazione per la porta NAND:

NOT:	$x = x \cdot x = x \uparrow x$
OR:	$x + y = \overline{\overline{x+y}} = \overline{x} \cdot \overline{y} = \overline{x} \uparrow \overline{y} = (x \uparrow x) \uparrow (y \uparrow y)$
AND:	$x \cdot y = \overline{\overline{x \cdot y}} = \overline{x \uparrow y} = (x \uparrow y) \uparrow (x \uparrow y)$

Dimostrazione per la porta NOR:

NOT:	$\overline{x} = \overline{x+x} = x \downarrow x$
OR:	$x + y = \overline{\overline{x+y}} = \overline{x \downarrow y} = (x \downarrow y) \downarrow (x \downarrow y)$
AND:	$x \cdot y = \overline{\overline{x \cdot y}} = \overline{\overline{x}+\overline{y}} = \overline{x} \downarrow \overline{y} = (x \downarrow x) \downarrow (y \downarrow y)$

Da SOP a NAND: $a \cdot b + c \cdot d = (a \uparrow b) \uparrow (c \uparrow d)$

Da POS a NOR: $(a + b) \cdot (c + d) = (a \downarrow b) \downarrow (c \downarrow d)$

[66] A.3 Validità di un'equazione nell'algebra di Boole (1) (MOD.NEL.COMP.)

D: Dimostrare la validita della seguente equazione booleana, usando le regole dell'algebra di Boole e/o le tabelle di verita: $\overline{x}+\overline{y}+z = \overline{x}\,\overline{y}\,\overline{z} + xyz + \overline{x}yz + x\overline{y}z$.

R:

$$\begin{aligned} \overline{(x \oplus y)} \oplus z &= \overline{(x \oplus y)}\,\overline{z} + (x \oplus y)z \\ &= \overline{x}\,\overline{y}\,\overline{z} + x\,y\,\overline{z} + \overline{x}\,y\,z + x\,\overline{y}\,z \end{aligned}$$

[67] A.4 Validità di un'equazione nell'algebra di Boole (2) (MOD.NEL.COMP.)

D: Dimostrare la validita della seguente equazione booleana, usando le regole dell'algebra di Boole e/o le tabelle di verita: $x + y\overline{x} = x + y$.

R:

$$\begin{aligned} x + y\,\overline{x} &= (x+y)(x+\overline{x}) \\ &= x + y \end{aligned}$$

[68] A.5 Validità di un'equazione nell'algebra di Boole (3) (MOD.NEL.COMP.)

D: Dimostrare la validita della seguente equazione booleana, usando le regole dell'algebra di Boole e/o le tabelle di verita: $x1x\_2 + x\_2x3 + x3x\_1 = x1x\_2 + x3x\_1$.

R:

$$\begin{aligned} x_1\,\overline{x}_2 + \overline{x}_2\,x_3 + x_3\,\overline{x}_1 &= x_1\,\overline{x}_2 + \overline{x}_2\,x_3\,(x_1 + \overline{x}_1) + x_3\,\overline{x}_1 \\ &= x_1\,\overline{x}_2 + x_1\,\overline{x}_2\,x_3 + x_3\,\overline{x}_1\,\overline{x}_2 + x_3\,\overline{x}_1 \\ &= x_1\,\overline{x}_2 + x_3\,\overline{x}_1 \end{aligned}$$

[69] A.6 Non-associatività di NAND

D: Dimostrare che l'operatore booleano NAND non e associativo.

R: Basta considerare una NAND a 3 ingressi. Se fosse costruita col metodo a cascata (associativo) avremmo un risultato sbagliato. Basterebbe pero negare il risultato della prima porta, che fa la NAND tra i primi 2 ingressi, per poi fare la NAND con il terzo ingresso.

[70] A.7 Sintesi di rete combinatoria di costo minimo

D: Sintetizzare una rete combinatoria di costo minimo, secondo il criterio di costo dei letterali, della funzione booleana $f(x1, x2, x3, x4)$, dove $f = 1$ se almeno una e non piu di due delle variabili d'ingresso hanno valore logico 1, altrimenti $f = 0$.

R:

Un'espressione SOP di costo minimo è:

$$f = x_1\,\overline{x}_2\,\overline{x}_3 + x_2\,\overline{x}_3\,\overline{x}_4 + \overline{x}_1\,\overline{x}_3\,x_4 + \overline{x}_1\,\overline{x}_2\,x_3 + \overline{x}_1\,x_3\,\overline{x}_4 + x_1\,\overline{x}_2\,\overline{x}_4$$

L"espressione POS di costo minimo è:

$$f = (x_1 + x_2 + x_3 + x_4)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_4)(\overline{x}_1 + \overline{x}_3 + \overline{x}_4)(\overline{x}_2 + \overline{x}_3 + \overline{x}_4)$$

Il costo dell'espressione SOP è di 18 letterali. Il costo dell'espressione POS è di 16 letterali. Dunque l'espressione POS dà la realizzazione di costo minimo.

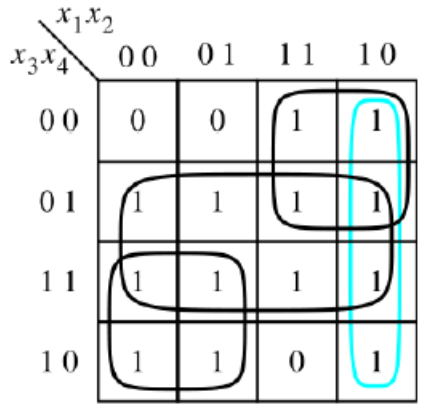
[71] A.8 Sintesi di costo minimo mediante mappe di Karnaugh (MOD.NEL.COMP.)

D: Si consideri la funzione booleana definita dalla seguente espressione:

$f(x1, x2, x3, x4) = (x1 \oplus x3) + (x1\,x3 + \overline{x}1\,\overline{x}3)\,x4 + x1\,\overline{x}2$.

Determinarne una forma SOP di costo minimo mediante una mappa di Karnaugh. Trovare poi, con lo stesso metodo, una forma SOP di costo minimo per la sua complementare, ovvero la funzione $\overline{f}$ *che vale 1 se, e solo se, f vale 0. Prendere quindi il comp* _ che vale 1 se, e solo se, *f* vale 0. Prendere quindi il complemento di tale espressione, usando la regola di De Morgan, per determinare una nuova espressione per *f* . Quest'ultima sara in forma POS (prodotto di somme). Confrontarne il costo (con il criterio dei letterali) con quello della sua forma SOP determinata in precedenza. Si possono trarre conclusioni generali da questo risultato?

R: La mappa di Karnaugh è:



L'espressione SOP di costo minimo è:

$$f = x_4 + \overline{x}_1\,x_3 + x_1\,\overline{x}_3 + x_1\,\overline{x}_2$$

L'espressione SOP di costo minimo per il complemento di f è:

$$\overline{f} = \overline{x}_1\,\overline{x}_3\,\overline{x}_4 + x_1\,x_2\,x_3\,\overline{x}_4$$

Complementando questa espressione e usando la regola di De Morgan si ha:

$$f = (x_1 + x_3 + x_4)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + x_4)$$

Il costo di questa espressione con il criterio dei letterali è 7, uguale a quello dell'espressione SOP. L'uguaglianza
vale in questo caso, ma non in generale, poichè per alcune funzioni la realizzazione POS è meno costosa della realizzazione SOP e viceversa.

[72] A.9 Sintesi in forma SOP di costo minimo con condizioni di indifferenza (MOD.NEL.COMP.)

D: Determinare forme SOP di costo minimo (secondo il criterio dei letterali) per ciascuna delle quattro funzioni booleane in tre variabili specificate dalle seguenti tabelle di verita. Si noti che due delle dette funzioni sono parziali, come indicato dalle condizioni di indifferenza caratterizzate da "x" nella tabella. Qualcuna delle quattro funzioni ha piu di una forma SOP di costo minimo? Se si, determinarle tutte.

x_1	x_2	x_3	f_1	f_2	f_3	f_4
0	0	0	1	1	x	0
0	0	1	1	1	1	1
0	1	0	0	1	0	1
0	1	1	0	1	1	x
1	0	0	1	0	x	x
1	0	1	0	0	0	x
1	1	0	1	0	1	1
1	1	1	1	1	1	0

R: Un’espressione di costo minimo per f1 è:

$$f_1 = \overline{x}_1\ \overline{x}_2 + x_1\ x_2 + x_1\ \overline{x}_3$$

Un’altra espressione che ha lo stesso costo è:

$$f_1 = \overline{x}_1\ \overline{x}_2 + x_1\ x_2 + \overline{x}_2\ \overline{x}_3$$

L’espressione di costo minimo per f2 è:

$$f_2 = \overline{x}_1 + x_2\ x_3$$

L’espressione di costo minimo per f3 è:

$$f_3 = x_1\ x_2 + \overline{x}_1\ x_3$$

Un’espressione di costo minimo per f4 è:

$$f_4 = x_2\ \overline{x}_3 + \overline{x}_2\ x_3$$

Un’altra espressione che ha lo stesso costo è:

$$f_4 = x_2\ \overline{x}_3 + \overline{x}_1\ x_3$$

[73] A.10 Sintesi in forma SOP di costo minimo con condizione di indifferenza specificata da un termine booleano

D: Determinare una forma SOP di costo minimo (secondo il criterio dei letterali) per la funzione booleana parziale *f* in quattro variabili che coincide con quella specificata dal termine
x 1 (*x* 2 *x* _ 3 + *x* 2 *x* 3 + *x* _ 2 *x* _ 3 *x* 4) + *x* 2 *x* _ 4 (*x* _ 3 + *x* 1)
eccetto che *f* e indefinita quando e vera la condizione di indifferenza x specificata dal termine
x = *x* 1 *x* _ 2 (*x* 3 *x* 4 + *x* _ 3 *x* _ 4) + *x* _ 1 *x* _ 3 *x* 4 .

R:

Un’espressione SOP di costo minimo è:

$$f = x_2\ \overline{x}_3 + x_1\ x_2 + x_1\ \overline{x}_3$$

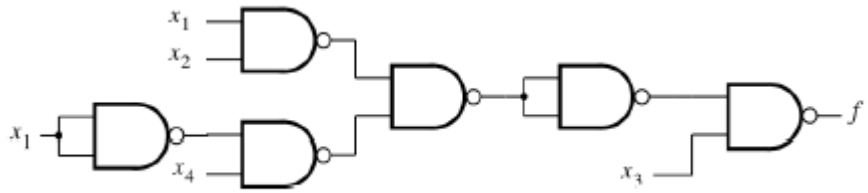
Un’altra espressione che ha lo stesso costo è:

$$f = x_2\ \overline{x}_3 + x_1\ x_2 + x_1\ x_4$$

[74] A.11 Sintesi di costo limitato di rete combinatoria NAND

D: Progettare una rete combinatoria di porte NAND a due ingressi che realizzi la seguente funzione booleana, impiegando non piu di 6 porte NAND: *f* = *x* 1 *x* 2 + *x* _ 3 + *x* _ 1 *x* 4 .

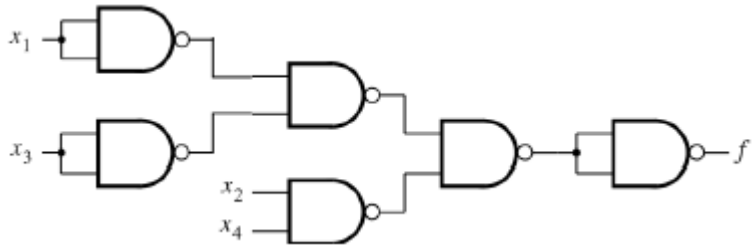
R:



[75] A.12 Sintesi di costo minimo di rete combinatoria NAND

D: Progettare una rete combinatoria di porte NAND a due ingressi che realizzi la seguente funzione booleana nel modo piu economico possibile: *f* = (*x* 1 + *x* 3)(*x* _ 2 + *x* _ 4) .

R:



[76] A.13 Vantaggi della tecnologia CMOS

D: Quali sono i principali vantaggi della tecnologia CMOS rispetto a quella NMOS?

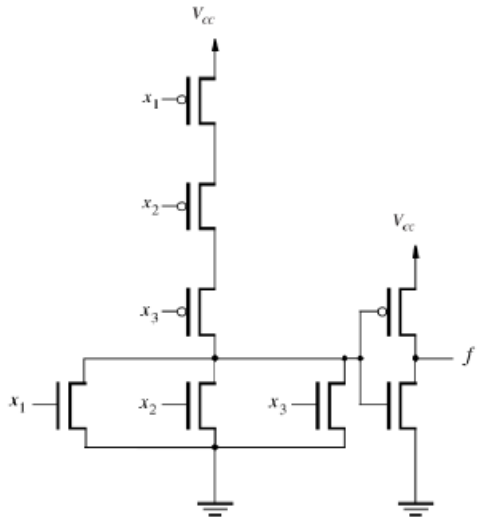
R: I vantaggi della tecnologia CMOS rispetto alla NMOS sono:
- minore consumo energetico
- la bassa sensibilità ai disturbi
- la bassa dissipazione del calore
- alta densità di integrazione

- leggermente più veloce

[77] A.14 Porta OR a tre ingressi in tecnologia CMOS

D: Progettare un circuito CMOS che realizzi la funzione booleana di una porta logica OR a tre ingressi.

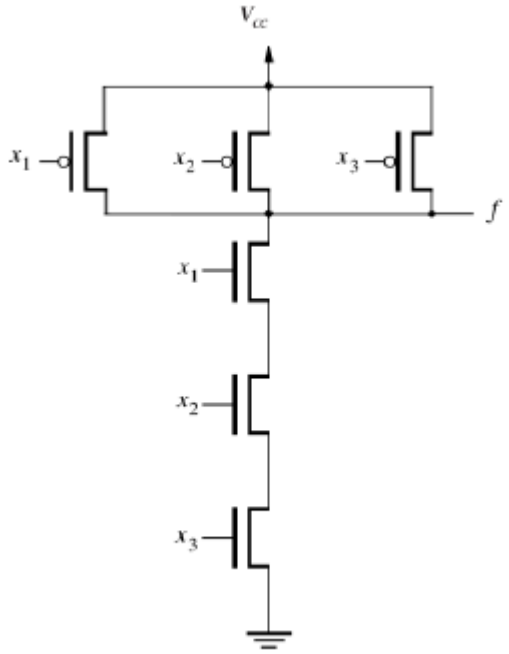
R:



[78] A.15 Porta NAND a tre ingressi in tecnologia CMOS

D: Progettare un circuito CMOS che realizzi la funzione booleana di una porta logica NAND a tre ingressi.

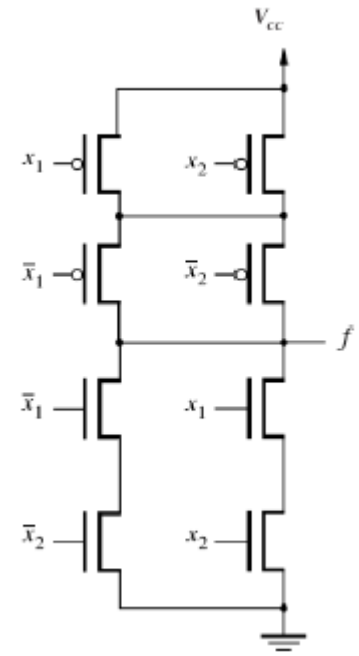
R:



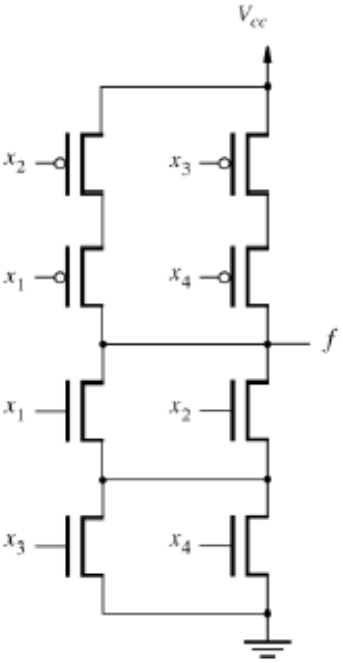
[79] A.16 Porta XOR in tecnologia CMOS

D: Progettare un circuito CMOS che realizzi la funzione booleana di una porta logica XOR.

R:



[80] A.17 Progetto di un circuito CMOS
D: Progettare un circuito CMOS che realizzi la funzione booleana f(x1,x2,x3,x4) = x.1 x.2 + x.3 x.4.
R:



[81] A.18 Bistabili Set-Reset (MOD.NEL.COMP.)
D: Stilare la tabella di transizione del bistabile asincrono di struttura simile a quella del latch SR, ma con porte NAND invece che NOR. Indicare come si puo usare questo circuito per costruire un bistabile sincrono SR.
R:

A	B	Q	$\overline{Q}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0/1	1/0

[82] A.19 Bistabile sincrono di tipo D (MOD.NEL.COMP.)
D: Stilare la tabella di transizione del bistabile sincrono di struttura simile a quella del latch sincrono di tipo D, ma con porte NOR invece che NAND. Si puo usare questo circuito per costruire un bistabile sincrono di tipo D? Se si, come? Se no, perche?
R:

CLK	D	Q(t + 1)
0	0	0
0	1	1
1	x	Q(t)

Dal suo confronto con quella del bistabile sincrono di tipo D appare evidente che anche il circuito con porte NOR realizza un bistabile sincrono di tipo D, ma nel quale il campionamento del dato in ingresso è attivato dal livello 0 del clock.

[83] A.20 Comportamento reale di circuiti logici e flip-flop master-slave

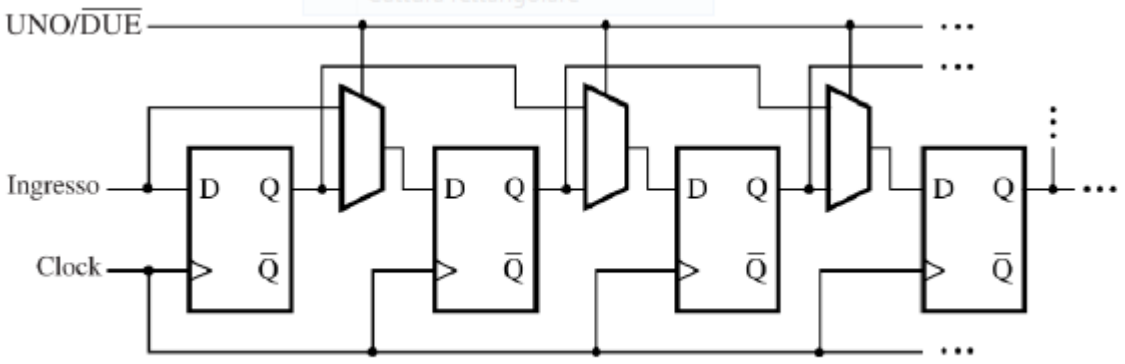
D: Si consideri un circuito sequenziale sincrono in cui il segnale di clock sia simultaneamente inviato a uno degli ingressi di una porta AND e ad una porta NOT, la cui uscita vada all'altro ingresso della porta AND. Da un punto di vista meramente logico, l'uscita della porta AND dovrebbe avere valore logico costante 0. Se pero si tiene conto delle caratteristiche fisiche del circuito, in particolare del ritardo di propagazione attraverso le porte logiche, risulta che il segnale in uscita dalla porta AND ha ben altra forma d'onda. Quale? Potrebbe essere utile combinare questo circuito con quello di un flip-flop di tipo master-slave? Se si, a qual fine? Se no, perche?

R: Dalla forma d'onda del segnale in uscita notiamo che, per effetto del ritardo di propagazione attraverso la porta NOT, il circuito trasforma una forma d'onda quadra, in cui i livelli logici si alternano con semiperiodi di uguale durata, in una forma d'onda di tipo impulsivo, in cui uno dei due livelli logici occupa una porzione molto breve del periodo. Un segnale periodico impulsivo di questo tipo può essere impiegato quale ingresso di clock di un bistabile di tipo D, per realizzare un flip-flop edge-triggered. Si può anche usare il segnale periodico impulsivo per pilotare l'ingresso di clock di un flip-flop di tipo D master-slave, per eliminare la trasparenza nel breve intervallo di campionamento dell'ingresso D.

[84] Registro di scorrimento singolo/doppio

D:La Figura A.33 mostra un registro a scorrimento di tipo semplice, dove i dati scorrono a destra di una posizione per volta sotto il controllo di un segnale di clock. Modificare questo registro per ottenere lo scorrimento a destra di una o due posizioni per volta, controllato dal clock e da un ulteriore segnale di controllo UNO/DUE.

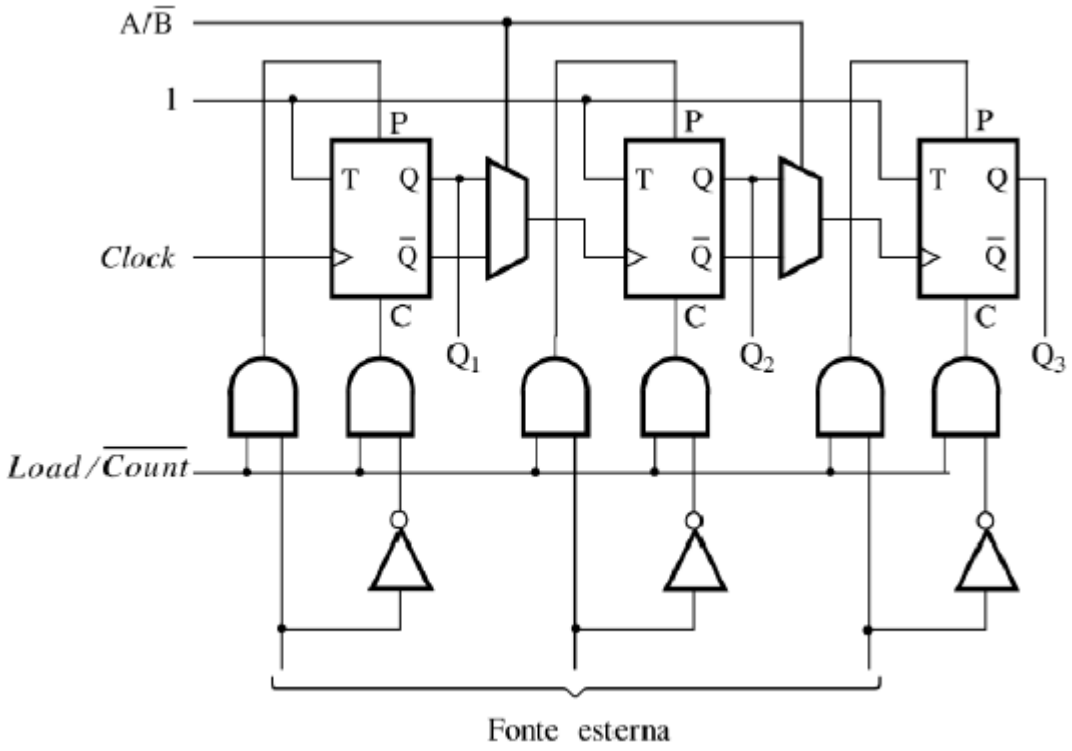
R:



[85] Contatore binario bidirezionale

D:Il contatore binario a 3 bit mostrato in Figura A.35 è detto "verso l'alto" perché il conteggio procede per incrementi, mentre uno in cui il conteggio procede per decrementi è detto "verso il basso". Se il conteggio può procedere in entrambe le direzioni, sotto il controllo di un segnale A/B, il contatore è detto "bidirezionale". Tracciare lo schema logico di un contatore bidirezionale che possa anche essere inizializzato a uno stato qualsiasi mediante scrittura parallela dei suoi flip-flop da una fonte esterna, aggiungendovi un ulteriore segnale di controllo LOAD/COUNT per determinare se il modulo viene inizializzato o funziona da contatore.

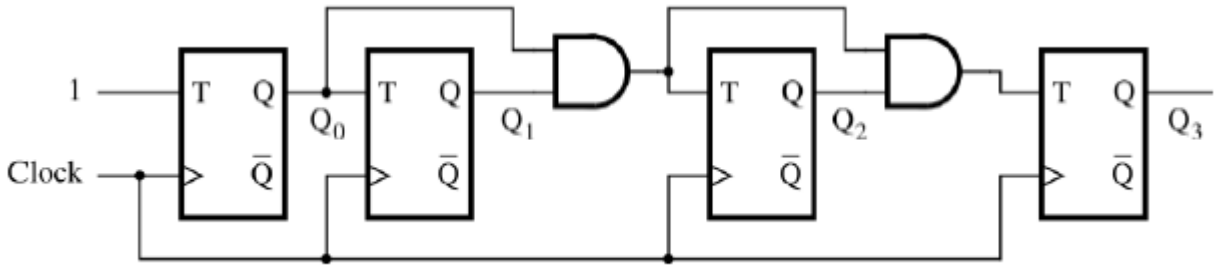
R:



[86] Contatore binario sincrono

D: Il contatore binario a 3 bit mostrato in Figura A.35 è "asincrono" poiché i suoi flip-flop non sono pilotati simultaneamente da uno stesso segnale di clock. A causa del ritardo di propagazione nei flip-flop, si hanno quindi piccoli scostamenti temporali delle loro transizioni di stato rispetto ai multipli del periodo, come indicato in figura. Un contatore binario "sincrono" non presenta questo fenomeno, poiché i suoi flip-flop sono tutti pilotati dallo stesso clock primario. Tracciare lo schema logico di un contatore binario sincrono a 4 bit, mantenendo l'uso di flip-flop di tipo T, i cui ingressi di clock sono però tutti connessi direttamente al clock primario.

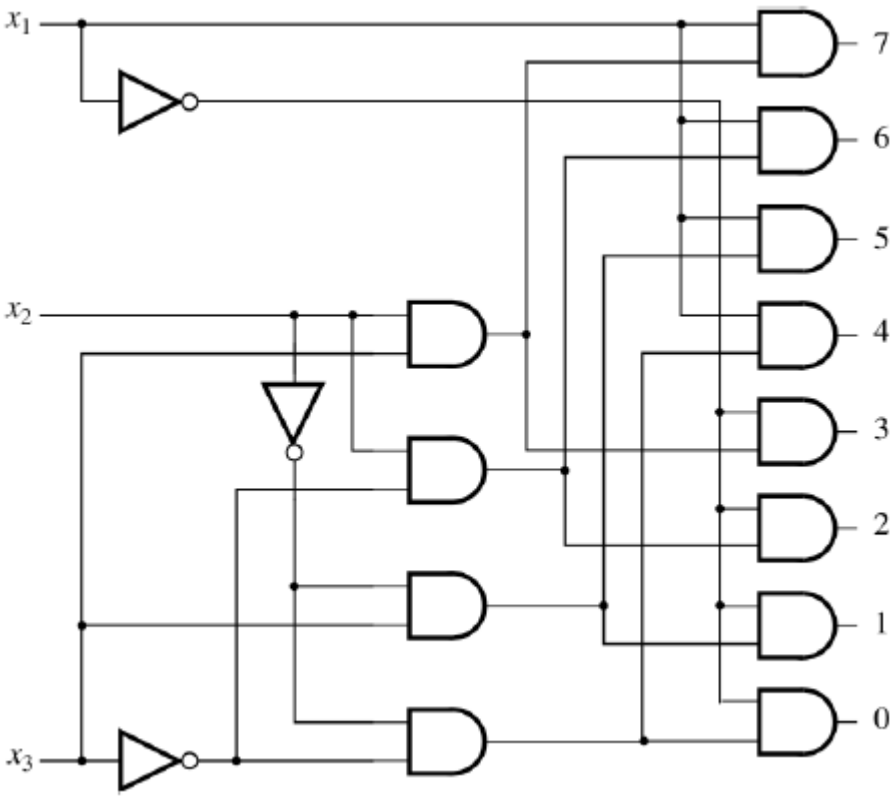
R:



[87] Decodificatore a tre ingressi

D: Tracciare la rete combinatoria di un decodificatore a tre ingressi e otto uscite, facendo uso di porte logiche a non più di due ingressi.

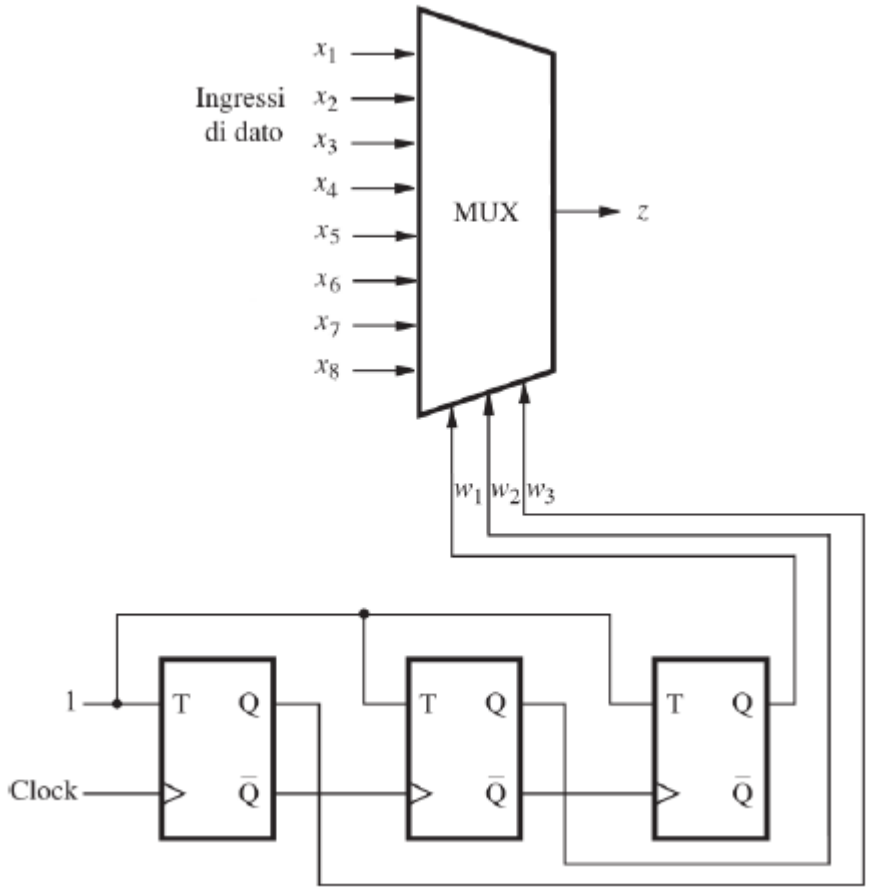
R:



[88] Convertitore parallelo-seriale

D:Un registro seriale-parallelo come quello mostrato in Figura A.34 può essere impiegato per la memorizzazione di dati provenienti da una linea di comunicazione seriale. Il problema converso, ovvero quello della trasmissione di dati, memorizzati su registri a scrittura parallela, su una linea di uscita seriale può essere risolto mediante un multiplatore opportunamente combinato con un contatore binario. Tracciare lo schema logico di un modulo che realizzi tale funzione.

R:



[89] Sintesi di funzione logica mediante multiplatore (1)

D: Specificare una realizzazione della funzione logica $f(x_1,x_2,x_3,x_4) = x_1x_3\overline{x}_4 + \overline{x}_1\overline{x}_3x_4 + \overline{x}_2\overline{x}_3\overline{x}_4$ mediante un multiplatore a tre ingressi di selezione, sintetizzandone la tabella di verità (come mostrato per esempio in Figura A.39). Si può realizzare la stessa funzione mediante un multiplatore a due ingressi di selezione? Se sì, mostrare come.

R:

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	x_4
0	1	1	0
1	0	0	$\overline{x}_4$
1	0	1	$\overline{x}_4$
1	1	0	0
1	1	1	$\overline{x}_4$

[90] Sintesi di funzione logica mediante multiplatore (2)

D:Specificare una realizzazione della funzione logica $f(x_1,x_2,x_3,x_4) = x_1\overline{x}_2x_3 + x_2x_3x_4 + \overline{x}_1\overline{x}_4$ mediante un multiplatore a tre ingressi di selezione, sintetizzandone la tabella di verità (come mostrato per esempio in Figura A.39). Si può realizzare la stessa funzione mediante un multiplatore a due ingressi di selezione? Se sì, mostrare come.

R:

x_3	x_4	f
0	0	$\overline{x}_2$
0	1	$\overline{x}_1$
1	0	x_1
1	1	0

Capitolo 9: Aritmetica

[91] 9.1 Equivalenza di reti combinatorie per il riporto nell'addizione binaria

D: La rete combinatoria in forma SOP per il calcolo del riporto in uscita in Figura 9.2a, che realizza la funzione booleana cn+1 = xnyn + xncn + yncn, è diversa da quella impiegata per lo stesso calcolo nella ALU a 1 bit, che realizza la funzione booleana cn+1 = xnyn + (xn yn)cn. Dimostrare che ⊕ le due reti sono equivalenti.

R:

Le due funzioni booleane sono equivalenti se **apbp + aprp + bprp=apbp + (ap ⊕ bp)rp** e quindi le relative tabelle di verità producono gli stessi risultati.

apbp	AND	aprp	AND	bprp	AND	Or totale
10	0	10	0	00	0	0
01	0	01	0	11	1	1
11	1	11	1	11	1	1
00	0	00	0	00	0	0

apbp	AND	aprp	AND	bprp	AND	aprp⊕bprp	apbp+(aprp⊕bprp)
10	0	10	0	00	0	0	0
01	0	01	0	11	1	1	1
11	1	11	1	11	1	0	1
00	0	00	0	00	0	0	0

Seleziona colonna

Le due tabelle della verità producono lo stesso risultato

IMPORTANTE: sostituire le lettere con:
r=c
p=n
a=x
b=y

SOLUZIONE ALTERNATIVA:
a=x; b=y;r=c; p=not
apbp+aprp+bprp=apbp+(ap⊕bp)rp

x	y	c	notx	noty	notc	(notx∧noty)=A	(notx∧notc)=B	(noty∧noty)=C	A∨B∨C
0	0	0	1	1	1	1	1	1	1
0	0	1	1	1	0	1	0	0	1
0	1	0	1	0	1	0	1	0	1
0	1	1	1	0	0	0	0	0	0
1	0	0	0	1	1	0	0	1	1
1	0	1	0	1	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0

Per la verifica dell'uguaglianza aggiungiamo le seguenti colonne:

(B XOR C)=E	A∨E
0	1
0	1
1	1
0	0
1	1
0	0
0	0
0	0

[92] 9.2 Correttezza della segnalazione di trabocco nell'addizionatore algebrico

D: Il bit di trabocco nell'addizionatore algebrico a n bit in Figura 9.3 è calcolato come XOR del bit di riporto in uscita c n (riporto oltre il bit di segno) e del bit di riporto c n −1 (riporto sul bit di segno). La regola per i l trabocco nella somma algebrica, formulata nel Paragrafo 1.4 del testo appare diversa, poiché stabilisce che si ha trabocco se e solo se gli addendi sono concordi e il segno del risultato è diverso da quello (comune) degli addendi. Dimostrare che il calcolo del trabocco nell'addizionatore algebrico è corretto rispetto a questa regola.

R: Per verificare l'esistenza di un trabocco(overflow) possiamo implementare la seguente espressione logica
Overflow= x(pedice n-1)y(pedice n-1)s(segnato)(pedice n-1)+x(segnato)(pedice n-1)y(segnato)(pedice n-1)s(pedice n-1)
che dimostra che il trabocco avviene quanto il riporto del bit rn e del bit rn-1 sono differenti.
Per implementare questa espressione logica usiamo la porta logica xor.
Esempio :
proviamo che l'utilizzo della porta logica xor vale per la seguente somma e per l'identificazione dell'overflow
proviamo a sommare +7 e -3 con n=4bit;
1 1 1 1 catena di propogazione del riporto
0 1 1 1 + primo addendo=+7 in comp a 2
1 1 0 1 = secondo addendo=-3 in comp a 2
1 0 1 0 0 somma completa a 5 bit

(il riporto in uscita è da trascurare)
il risultato corretto è +4.
la porta xor ha funzionato in quanto
rn=1
rn-1=1
rn e rn-1 sono i riporti della catena di propagazione del riporto
precisamente gli ultimi 2
1(rn) 1(rn-1) 1 1 catena di propogazione del riporto
0 1 1 1 + primo addendo=+7 in comp a 2
1 1 0 1 = secondo addendo=-3 in comp a 2
1 0 1 0 0 somma completa a 5 bit
---
1 xor 1 = 0 (false)
è possibile fare la somma in quanto i bit rn e rn-1 sono differenti, e lo xor restituisce falso in quanto non c'è overflow!!

[93] 9.3 Funzioni aritmetiche di una ALU

D: La ALU a 8 bit illustrata nei simulatori indicati a lezione, con input INC al riporto in ingresso a destra, può eseguire addizioni e sottrazioni a 8 bit in complemento a 2. Spiegare come.

R: Si immagini una ALU con 8 full adder. L'input INC fornisce in ingresso all'adder del bit meno significativo un 1. In questo modo si può procedere con somme e sottrazioni a complemento a due. per sottrarre m da n basta sommare -m a n. Per ricavare -m basta invertire ogni singolo bit di m e aggiungere 1 al risultato (grazie ad INC), ignorando poi un eventuale riporto di 1 finale che causerebbe un overflow. La validità dell'operazione e l'appartenenza del risultato all'intervallo dei numeri rappresentabili è verificabile attraverso gli ultimi due bit di riporto ottenuti: bisogna verificare se il riporto è stato eseguito sul bit del segno (il bit più a sinistra, 0 per i positivi e 1 per i negativi) ma non è stato portato fuori, o viceversa; più semplicemente, se i due bit più a sinistra sulla riga dei riporti non sono entrambi 0 o 1.

[94] 9.4 Uso di una ALU per una funzione costante

D: Descrivere due modi diversi per avere in output da una ALU a 8 bit, quale quella illustrata nei simulatori indcati a lezione, il valore costante -1 (in complemento a due), specificando in ciascun caso i valori dei sei segnali di controllo F1, F2, ENA, ENB, INVA e INC.

R: Un primo modo potrebbe essere di assegnare 0 ad F1, 1 ad F2 (01:OR), 0 ad entrambi ENA e ENB, 0 all'input INC 0 e 1 all'input INVA. Seguendo lo stesso approccio del primo modo si può ottenere il valore costante -1 assegnando 1 ad F1, 0 ad F2, 0 (10:NOT) ad entrambi ENA e ENB, 0 all'input INC 0 e 1 all'input INVA.

[95] 9.5 Estensioni della ALU per la segnalazione di trabocco (NON CAPISCO I TRATT.)

D: La ALU a n bit con i sei segnali di controllo F1, F2, ENA, ENB, INVA e INC, di cui le ALU a 8 bit illustrate nei simulatori indicati a lezione sono realizzazioni particolari, può eseguire addizioni e sottrazioni a n bit in complemento a 2, e dà anche in uscita il bit di riporto oltre il segno, ma non produce in uscita una segnalazione dell'eventuale trabocco. Come si potrebbe modificarla per avere anche una tale uscita?

R:
Viene utilizzata una ALU a 32 bit per la sua funzione; viene determinata dalle 6 linee di controllo: F1, F2, ENA, ENB, INVA, INC. Non tutte le 2^6 = 64 combinazioni sono utili.
F1 ed F2 definiscono la funzione, ENA ed ENB abilitano A e B rispettivamente, INVA nega l'input in A; infine INC forza un riporto nel

bit meno significativo.

F1	F2	ENA	ENB	INVA	INC	FUNCTION
0	1	1	0	0	0	A
0	1	0	1	0	0	B
$\overline{0}$	1	1	0	1	$\overline{1}$	A
$\overline{1}$	0	1	1	0	$\overline{1}$	B

Capitolo 2: Istruzioni macchina

[96] 2.1 Rappresentazione binaria di istruzioni e di dati numerici

D: Data una sequenza binaria in una locazione di memoria, è possibile dire se questa sequenza rappresenti un'istruzione macchina o un numero?

R: No; ogni sequenza binaria si può interpretare come un numero o come un'istruzione.

[97] 2.2 Indirizzamento crescente dei byte

D: Considerare un calcolatore che ha una memoria indirizzabile per byte organizzata in parole da 32 bit in base allo schema di indirizzamento crescente dei byte. Un programma legge i caratteri ASCII digitati su una tastiera e li immagazzina in byte di locazioni consecutive, a partire dalla locazione 1000. Mostrare il contenuto delle due parole di memoria alle locazioni 1000 e 1004 dopo che sia stata digitata la parola "Computer".

R:

1000 COMP
1004 UTER

[98] 2.3 Indirizzamento decrescente dei byte

D: Considerare un calcolatore che ha una memoria indirizzabile per byte organizzata in parole da 32 bit in base allo schema di indirizzamento decrescente dei byte. Un programma legge i caratteri ASCII digitati su una tastiera e li immagazzina in byte di locazioni consecutive, a partire dalla locazione 1000. Mostrare il contenuto delle due parole di memoria alle locazioni 1000 e 1004 dopo che sia stata digitata la parola "Computer".

R:

1000 PMOC
1004 RETU

[99] 2.4 Calcolo in stile RISC di un'espressione (1)

D: Scrivere un programma in stile RISC che calcoli l'espressione SOMMA = 580 + 68400 + 80000.

R:

```
                Move        R2, #NUMERI     Ottieni l'indirizzo dei numeri
                Load        R3, (R2)        Carica 580
                Load        R4, 4(R2)       Carica 68400
                Add         R3, R3, R4      Genera 580 + 68400
                Load        R4, 8(R2)       Carica 80000
                Add         R3, R3, R4      Genera la somma finale
                Store       R3, 12(R2)      Memorizza la somma
prossima istruzione
                ORIGIN      0x500
NUMERI:         DATAWORD    580, 68400, 80000 Numeri da sommare
SOMMA:          RESERVE     4               Spazio per la somma
```

[100] 2.5 Calcolo in stile CISC di un'espressione (1)

D: Scrivere un programma in stile CISC che calcoli l'espressione SOMMA = 580 + 68400 + 80000.

R:

```
                Move        R2, #NUMERI     Ottieni l'indirizzo dei numeri
                Move        R3, (R2)+       Carica 580
                Add         R3, (R2)+       Genera 580 + 68400
                Add         R3, (R2)        Genera la somma finale
                Move        SOMMA, R3       Memorizza la somma
prossima istruzione
                ORIGIN      0x500
NUMERI:         DATAWORD    580, 68400, 80000 Numeri da sommare
SOMMA:          RESERVE     4               Spazio per la somma
```

[101] 2.6 Calcolo in stile RISC di un'espressione (2)

D: Scrivere un programma in stile RISC che calcoli l'espressione RISPOSTA = A · B + C · D.

R:

```
                Move        R2, #A          Ottieni l'indirizzo di A
                Load        R3, (R2)        Carica l'operando A
                Move        R2, #B          Ottieni l'indirizzo di B
```

```
                Load        R4, (R2)        Carica l'operando B
                Multiply    R5, R3, R4      Genera A B
                Move        R2, #C          Ottieni l'indirizzo di C
                Load        R3, (R2)        Carica l'operando C
                Move        R2, #D          Ottieni l'indirizzo di D
                Load        R4, (R2)        Carica l'operando D
                Multiply    R6, R3, R4      Genera C D
                Add         R7, R5, R6      Calcola la risposta finale
                Move        R2, #RISPOSTA   Ottieni l'indirizzo e
                Store       R7, (R2)        memorizza la risposta
prossima istruzione
                ORIGIN      0x500
A:              DATAWORD    100             Dati di prova
B:              DATAWORD    50
C:              DATAWORD    20
D:              DATAWORD    400
RISPOSTA:       RESERVE     4               Spazio per la risposta
```

[102] 2.7 Calcolo in stile CISC di un'espressione (2)

D: Scrivere un programma in stile CISC che calcoli l'espressione RISPOSTA = A · B + C · D.

R:

```
                Move        R2, A           Carica l'operando A
                Multiply    R2, B           Genera A B
                Move        R3, C           Carica l'operando C
                Multiply    R3, D           Genera C D
                Add         R3, R2          Calcola la risposta finale
                Move        RISPOSTA, R3    Memorizza la risposta
prossima istruzione
                ORIGIN      0x500
A:              DATAWORD    100             Dati di prova
B:              DATAWORD    50
C:              DATAWORD    20
D:              DATAWORD    400
RISPOSTA:       RESERVE     4               Spazio per la risposta
```

[103] 2.8 Calcolo dell'indirizzo effettivo (EA)

D: I registri R4 e R5 contengono i numeri decimali 2000 e 3000 prima che i seguenti modi di indirizzamento siano usati per accedere a un operando in memoria. Qual è l'indirizzo effettivo (EA) in ciascun caso?

(a) 12(R4)
(b) (R4,R5)
(c) 28(R4,R5)
(d) (R4)+
(e) -(R4)

R: a) 2000+12 =2012
b) 2000+3000 =5000
c) 28 +2000+3000 =5028
d) =2000 (rimane 2000 perché l'operando è postfisso)
e) 2000-4 =1996

[104] 2.9 Uso dell'indirizzamento indiretto in un programma RISC

D: Riscrivere il ciclo di addizione nella Figura 2.8 in modo da accedere ai numeri della lista in ordine inverso; ovvero il primo numero a cui si accede è l'ultimo nella lista, e l'ultimo numero a cui si accede è alla locazione di memoria NUM1. Cercare di ottenere il modo più efficiente per determinare la terminazione del ciclo. Il nuovo ciclo eseguirà più velocemente del ciclo in Figura 2.8?

R:

```
                    Load        R2, N           Carica la dimensione della lista
                    LShiftL     R2, R2, #2      Moltiplica per 4
                    Clear       R3              Inizializza la somma a 0
                    Move        R4, #NUM1       Ottieni l'indirizzo del primo numero
                    Add         R2, R2, R4      Indirizzo successivo all'ultimo dato
CICLO:              Subtract    R2, R2, #4      Decrementa il puntatore alla lista
                    Load        R5, (R2)        Preleva il prossimo numero
                    Add         R3, R3, R5      Addiziona il numero alla somma
          Branch if [R4]<[R2]   CICLO           Salta indietro se non hai finito
                    Store       R3, SOMMA       Immagazzina la somma finale
```

[105] 2.10 Uso dell'indirizzamento con indice in un programma RISC

D: La lista di voti per gli studenti mostrata in Figura 2.10 viene modificata per contenere j voti per altrettante prove per ciascuno studente. Assumere che ci siano n studenti. Scrivere un programma in stile RISC per

calcolare le somme dei voti di tutti gli studenti per ciascuna prova e immagazzinare queste somme nelle parole di locazioni di memoria agli indirizzi SOMMA, SOMMA+4, SOMMA+8, ... Il numero di prove, j , è più grande del numero di registri del processore, quindi il tipo di programma mostrato in Figura 2.11 per il caso con 3 prove non può essere usato. Usare due cicli annidati. Il ciclo più interno dovrebbe accumulare la somma per una certa prova, e il ciclo più esterno dovrebbe essere iterato per scorrere il numero di prove, j . Assumere che l'area di memoria usata per immagazzinare le somme sia stata inizialmente azzerata.

R:

```
                Move            R2, #J          Calcola e poni in R2
                Load            R2, (R2)        Passo = 4(j + 1)
                Add             R2, R2, #1
                LShiftL         R2, R2, #2
                Move            R3, #LISTA      Poni in R3 la locazione del voto
                Add             R3, R3, #4          della prova 1 per studente 1
                Move            R4, #SOMMA      Poni in R4 la locazione della
                                                    somma per la prova 1
                Move            R5, #J          Poni j nel contatore R5 del
                Load            R5, (R5)            ciclo esterno
ESTERNO:        Move            R6, #N          Poni n nel contatore R6 del
                Load            R6, (R6)            ciclo interno
                Move            R8, R0          Azzera il registro somma R8
                Move            R9, R3          Usa R9 come registro indice
INTERNO:        Load            R10, (R9)       Accumula la somma dei voti
                Add             R8, R8, R10         della prova
                Add             R9, R9, R2      Incrementa di Passo il registro indice
                Subtract        R6, R6, #1      Controlla se tutti i voti per la prova
        Branch if [R6]>[R0]     INTERNO             corrente siano stati accumulati
                Store           R8, (R4)        Salva la somma dei voti e incrementa
                Add             R4, R4, #4          il puntatore alla locazione somma
                Add             R3, R3, #4      Incrementa R3 per puntare al
                                                    prossimo voto per lo studente 1
                Subtract        R5, R5, #1      Controlla se le somme per tutte
        Branch if [R5]>[R0]     ESTERNO             le prove siano state calcolate
prossima istruzione
```

[106] 2.11 Direttive di assemblatore (1)

D: Scrivere un programma in stile RISC che trovi il numero di occorrenze di interi negativi in una lista di n interi a 32 bit e immagazzini il conteggio nella locazione NUMNEG. Il valore n è immagazzinato nella locazione di memoria N e il primo intero della lista è memorizzato nella locazione NUMERI. Includere le direttive di assemblatore necessarie e una lista di esempio che contenga sei numeri, alcuni dei quali negativi.

R:

```
                Move            R2, #N          Ottieni l'indirizzo N
                Load            R2, (R2)        Carica la dimensione della lista
                Move            R3, R0          Azzera il contatore
                Move            R4, #NUMERI     Carica l'indirizzo del primo numero
CICLO:          Load            R5, (R4)        Preleva il prossimo numero
        Branch if [R5]≥[R0]     PROSSIMO        Controlla se il numero `e negativo
                Add             R3, R3, #1      Incrementa il contatore
PROSSIMO:       Add             R4, R4, #4      Incrementa il puntatore alla lista
                Subtract        R2, R2, #1      Decrementa il contatore della lista
        Branch if [R2]>[R0]     CICLO           Torna indietro se non hai finito
                Move            R6, #NUMNEG     Ottieni l'indirizzo di NUMNEG
                Store           R3, (R6)        Immagazzina il risultato
prossima istruzione
                ORIGIN          0x500
NUMNEG:         RESERVE         4               Spazio per il risultato
N:              DATAWORD        6               Dimensione della lista
NUMERI:         DATAWORD        23, -5, -128    Dati di prova
                DATAWORD        44, -23, -9
```

[107] 2.12 Inizializzazione di dati in memoria

D: Entrambi i seguenti frammenti di codice fanno sì che il valore 300 sia immagazzinato nella locazione 1000, ma in momenti diversi. Spiegare la differenza.

```
                ORIGIN      1000
                DATAWORD    300
```

e

```
                Move        R2, #1000
```

```
                    Move    R3, #300
                    Store   R3, (R2)
```

R: Le direttive di assemblatore ORIGIN e DATAWORD fanno s`ı che l'immagine della memoria del programma oggetto costruita dall'assemblatore indichi che 300 `e da inserire nella parola di memoria alla locazione 1000 al momento in cui si carica il programma in memoria prima dell'esecuzione.
Le istruzioni Move e Store inseriscono 300 nella parola di memoria alla locazione 1000 quando queste istruzioni vengono eseguite come parte di un programma.

[108] 2.13 Direttive di assemblatore (2)

D: Scrivere un programma assemblativo nello stile della Figura 2.13 per il programma in Figura 2.11. Assumere lo schema di dati della Figura 2.10.

R:

```
                    ORIGIN  100
                    MOV     R2, #LISTA      Ottieni l'indirizzo di LISTA
                    CLR     R3
                    CLR     R4
                    CLR     R5
                    LD      R6, N           Carica il valore n
CICLO:              LD      R7, 4(R2)       Aggiungi il voto della prova 1 del
                    ADD     R3, R3, R7      prossimo studente alla somma parziale
                    LD      R7, 8(R2)       Aggiungi il voto della prova 2 dello
                    ADD     R4, R4, R7      studente alla somma parziale
                    LD      R7, 12(R2)      Aggiungi il voto della prova 3 dello
                    ADD     R5, R5, R7      studente alla somma parziale
                    ADD     R2, R2, #16     Incrementa il puntatore
                    SUB     R6, R6, #1      Decrementa il contatore
                    BGT     R6, R0, CICLO   Salta indietro se non hai finito
                    ST      R3, SOMMA1      Immagazzina il totale per la prova 1
                    ST      R4, SOMMA2      Immagazzina il totale per prova 2
                    ST      R5, SOMMA3      Immagazzina il totale per la prova 3
prossima istruzione
                    ORIGIN  300
SOMMA1:             RESERVE 4
SOMMA2:             RESERVE 4
SOMMA3:             RESERVE 4
N:                  DATAWORD 50
LISTA:              RESERVE 800
                    END
```

[109] 2.14 Istruzioni assemblative per operazioni sulla pila

D: In un programma il registro R5 è usato per puntare alla cima della pila contenente numeri a 32 bit. Scrivere una sequenza di istruzioni usando i modi di indirizzamento con indice e spiazzamento, con autoincremento e con autodecremento per effettuare ciascuno dei seguenti compiti:
(a) Spilare i due elementi in cima alla pila, sommarli, e quindi impilare il risultato.
(b) Copiare il quinto elemento dalla cima nel registro R3.
(c) Spilare dieci elementi dalla pila.
Per ciascun caso assumere che la pila contenga dieci o più elementi.

R:

(a)

```
Move R2, (R5)+
Add R2, (R5)+
Move -(R5), R2
```

(b)

```
Move R3, 16(R5)
```

(c)

```
Add R5, #40
```

[110] 2.15 Passaggio di parametri nella pila

D: Mostrare il contenuto della pila del processore e il contenuto del puntatore alla pila, SP, subito dopo che ognuna delle seguenti istruzioni nel programma in Figura 2.18 sia stata eseguita. Assumere che [SP] = 1000 al Livello 1, prima che inizi l'esecuzione del programma chiamante.
(a) La seconda istruzione Store nel sottoprogramma
(b) L'ultima istruzione Load nel sottoprogramma
(c) L'ultima istruzione Store nel programma chiamante

R:

(a) La pila conterrà i primi 4 elementi mostrati in Figura 2.19 (R5, R4, R3, R2). Tuttavia, il puntatore alla pila punterà all'indirizzo 976, poiché è già stato adattato a questo valore dalla prima istruzione Subtract nel sottoprogramma.

(b) Il puntatore alla pila avrà il valore 976. Il contenuto della pila sarà lo stesso mostrato in Figura 2.19, eccetto che NUM1 sarà stato sostituito dalla somma.
(c) Il puntatore alla pila avrà il valore 992. Ci saranno 2 elementi nella pila: n e la somma.

[111] 2.16 Opzioni per il salvataggio dell'indirizzo di rientro

D: Considerare le seguenti possibilità per conservare l'indirizzo di rientro da un sottoprogramma:
(a) In un registro del processore
(b) In una locazione di memoria associata alla chiamata, così che una locazione diversa verrà usata quando un sottoprogramma sarà chiamato da posti diversi
(c) Su una pila
Quale di queste possibilità supporta annidamento di sottoprogrammi e quale supporta ricorsione dei sottoprogrammi (ovvero, un sottoprogramma che chiama sé stesso)?

R:
(a) N´e l'annidamento, n è la ricorsione sono supportati.
(b) L'annidamento è supportato, poichè istruzioni Call differenti salveranno l'indirizzo di rientro in locazioni di memoria differenti. La ricorsione non è supportata.
(c) Sia l'annidamento che la ricorsione sono supportati.

[112] 2.17 Sottoprogrammi RISC per operazioni su pila limitata

D: In aggiunta alla pila del processore, in alcuni programmi può essere conveniente usare un'altra pila. Alla seconda pila è generalmente allocato un ammontare fissato di spazio di memoria. In questo caso è importante evitare di impilare un elemento quando la pila ha raggiunto la sua dimensione massima. Inoltre, è importante evitare di tentare di spilare un elemento da una pila vuota, a seguito di un errore di programmazione. Scrivere due brevi sottoprogrammi in stile RISC, chiamati IMPILASICURA e SPILASICURA, per impilare e spilare controllando le apposite condizioni per evitare questi due possibili errori. Assumere che l'elemento da impilare/spilare sia posizionato nel registro R2, e che il registro R5 serva da puntatore alla pila per questa pila dell'utente. La pila è piena se il suo elemento in cima è memorizzato nella locazione CIMA, ed è vuota se l'ultimo elemento spilato era memorizzato nella locazione FONDO. I sottoprogrammi dovrebbero saltare a ERROREPIENO e ERROREVUOTO, rispettivamente, se si verificano i suddetti errori. Tutti gli elementi hanno come dimensione una parola, e la pila cresce verso locazioni con indirizzi più bassi.

R: Il contenuto del registro R2 può essere impilato nella seconda pila o spilato da essa, con controllo di errore, chiamando i seguenti sottoprogrammi in stile RISC:

```
IMPILASICURA: Subtract    SP, SP, #4          Salva registro R3 sulla
              Store       R3, (SP)            pila del processore
              Move        R3, #CIMA
              Branch if [R5]≤[R3]  ERROREPIENO
              Subtract    R5, R5, #4
              Store       R2, (R5)
              Load        R3, (SP)            Ripristina registro R3
              Add         SP, SP, #4
              Return
SPILASICURA:  Subtract    SP, SP, #4
              Store       R3, (SP)      Salva registro R3 sulla pila del processore
              Move        R3, #FONDO
              Branch if [R5]>[R3]  ERROREVUOTO
              Load        R2, (R5)
              Add         R5, R5, #4
              Load R3, (SP)  Ripristina registro R3
              Add         SP, SP, #4
              Return
```

[113] 2.18 Sottoprogrammi CISC per operazioni su pila limitata

D: In aggiunta alla pila del processore, in alcuni programmi può essere conveniente usare un'altra pila. Alla seconda pila è generalmente allocato un ammontare fissato di spazio di memoria. In questo caso è importante evitare di impilare un elemento quando la pila ha raggiunto la sua dimensione massima. Inoltre, è importante evitare di tentare di spilare un elemento da una pila vuota, a seguito di un errore di programmazione. Scrivere due brevi sottoprogrammi in stile CISC, che possono usare i modi di indirizzamento con autoincremento e con autodecremento, chiamati IMPILASICURA e SPILASICURA, per impilare e spilare controllando le apposite condizioni per evitare questi due possibili errori. Assumere che l'elemento da impilare/spilare sia posizionato nel registro R2, e che il registro R5 serva da puntatore alla pila per questa pila dell'utente. La pila è piena se il suo elemento in cima è memorizzato nella locazione CIMA, ed è vuota se l'ultimo elemento spilato era memorizzato nella locazione FONDO. I sottoprogrammi dovrebbero saltare a ERROREPIENO e ERROREVUOTO, rispettivamente, se si verificano i suddetti errori. Tutti gli elementi hanno come dimensione una parola, e la pila cresce verso locazioni con indirizzi più bassi.

R: Il contenuto del registro R2 pu`o essere impilato nella seconda pila o spilato da essa, con controllo di errore, come segue:

```
IMPILASICURA:  Compare    R5, #CIMA     Se R5 ha un valore uguale o minore
```

```
                Branch<=0       ERROREPIENO     di CIMA, la pila e' piena
                Move            -(R5), R2       Altrimenti, inserisci il nuovo elemento
SPILASICURA:    Compare         R5, #FONDO      Se R5 ha un valore uguale o maggiore
                Branch>=0       ERROREVUOTO     di FONDO, la pila e' vuota
                Move            R2, (R5)+       Altrimenti, estrai l'elemento dalla pila
```

[114] 2.19 Programma assemblativo RISC generico per l'elaborazione di un segnale

D: Il calcolo del prodotto scalare è stato discusso nel Paragrafo 2.12.1. Questo tipo di calcolo può essere usato nel seguente compito di elaborazione di un segnale. Una sequenza temporale di segnali in ingresso IN(0), IN(1), IN(2), IN(3), ... è elaborata mediante un vettore peso a tre elementi (WT(0), WT(1), WT(2)) = (1/8, 1/4, 1/2) per produrre una sequenza temporale del segnale in uscita OUT(0), OUT(1), OUT(2), OUT(3), ... come segue:

$OUT(0) = WT(0) \cdot IN(0) + WT(1) \cdot IN(1) + WT(2) \cdot IN(2)$
$OUT(1) = WT(0) \cdot IN(1) + WT(1) \cdot IN(2) + WT(2) \cdot IN(3)$
$OUT(2) = WT(0) \cdot IN(2) + WT(1) \cdot IN(3) + WT(2) \cdot IN(4)$
$OUT(3) = WT(0) \cdot IN(3) + WT(1) \cdot IN(4) + WT(2) \cdot IN(5)$
...

Tutti i valori dei segnali e dei (reciproci dei) pesi sono numeri a 32 bit con segno. Pesi, valori in ingresso e in uscita sono immagazzinati nella memoria a partire dalle locazioni WT, IN e OUT, rispettivamente. Scrivere un programma in stile RISC per calcolare e memorizzare i primi n valori in uscita, dove n è memorizzato alla locazione N.

Suggerimento: Per fare le moltiplicazioni possono essere usati gli scorrimenti aritmetici verso destra.

R:

```
                Move            R2, #N          Carica il numero di valori, n, che
                Load            R2, (R2)            devono essere generati
                Move            R3, #IN         Puntatore alla lista IN
                Move            R4, #OUT        Puntatore alla lista OUT
CICLO:          Load            R5, (R3)        Carica il valore IN(k) e
                AShiftR         R5, R5, #3          dividilo per 8
                Load            R6, 4(R3)       Carica il valore IN(k+1) e
                AShiftR         R6, R6, #2          dividilo per 4
                Add             R5, R5, R6
                Load            R6, 8(R3)           Carica il valore IN(k+2) e
                AShiftR         R6, R6, #1              dividilo per 2
                Add             R5, R5, R6          Calcola la somma e
                Store           R5, (R4)             immagazzinala nella lista OUT
                Add             R3, R3, #4          Incrementa i puntatori
                Add             R4, R4, #4              alle liste IN e OUT
                Subtract        R2, R2, #1          Continua fino a generare tutti
            Branch if [R2]>[R0] CICLO                   i valori nella lista OUT
prossima istruzione
```

[115] 2.20 Sottoprogramma per la copia in memoria di una sequenza di byte (MOD.NEL.COMP.)

D: Scrivere un sottoprogramma assemblativo MEMCOPY per copiare una sequenza di byte da un'area di memoria a un'altra. Il sottoprogramma dovrebbe accettare tre parametri di input mediante i registri per rappresentare l'indirizzo da , l'indirizzo a , e la lunghezza della sequenza da copiare. Le due aree possono sovrapporsi. In tutti i casi tranne uno, il sottoprogramma dovrebbe copiare i byte secondo l'ordine crescente degli indirizzi. Nel caso in cui l'indirizzo a cada dentro la sequenza dei byte da copiare, ovvero quando l'indirizzo a sia tra l'indirizzo da e da+lunghezza-1 , il sottoprogramma deve copiare i byte secondo l'ordine decrescente degli indirizzi partendo dalla fine della sequenza di byte da copiare, per evitare di sovrascrivere i byte non ancora copiati.

R:

```
                Move            R2, #N          Carica il parametro lunghezza
                Load            R2, (R2)                in R2
                Move            R3, #DA         Puntatore alla lista da
                Move            R4, #A          Puntatore alla lista a
                Call            MEMCOPY
                prossima istruzione
MEMCOPY:        Subtract        SP, SP, #12     Salva i registri
                Store           R5, 8(SP)
                Store           R6, 4(SP)
                Store           R7, (SP)
                Add             R5, R3, R2      Calcola l'indirizzo dell'ultimo
                Subtract        R5, R5, #1          elemento della lista da
            Branch if [R4]>[R5] CRESCE          Scansione decrescente se la lista a
            Branch if [R4]<[R3] CRESCE              inizia dentro la lista da
```

```
                    Add             R6, R4, R2          Calcola il puntatore alla lista a
                    Subtract        R6, R6, #1                per la scansione decrescente
DECRESCE:           LoadByte        R7, (R5)            Trasferisci un byte e
                    StoreByte       R7, (R6)
                    Subtract        R5, R5, #1          decrementa i puntatori
                    Subtract        R6, R6, #1
                Branch if [R5]≥[R3] DECRESCE
                    Branch          FATTO
CRESCE:             LoadByte        R7, (R3)            Trasferisci un byte e
                    StoreByte       R7, (R4)
                    Add             R3, R3, #1            incrementa i puntatori
                    Add             R4, R4, #1
                Branch if [R3]≤[R5] CRESCE
FATTO:              Load            R7, (SP)            Ripristina i registri
                    Load            R6, 4(SP)
                    Load            R5, 8(SP)
                    Add             SP, SP, #12
                    Return
```

[116] 2.21 Sottoprogramma per il conteggio delle differenze fra due sequenze di byte(SBAGLIATO)

D: Scrivere un sottoprogramma assemblativo MEMCOMP per effettuare un confronto byte per byte di due sequenze di byte nella memoria principale. Il sottoprogramma dovrebbe accettare tre parametri di input inseriti nei registri per rappresentare il primo indirizzo, il secondo indirizzo e la lunghezza delle sequenze da confrontare. Dovrebbe usare un registro per fornire quale valore di ritorno il conteggio del numero di confronti non corrispondenti.

R:

```
                    Move            R2, #N              Carica il parametro lunghezza
                    Load            R2, (R2) in R2
                    Move            R3, #PRIMO          Puntatore alla prima lista
                    Move            R4, #SECONDO        Puntatore alla seconda lista
                    Call            MEMCOMP
prossima istruzione
MEMCOMP:            Subtract        SP, SP, #12         Salva i registri
                    Store           R5, 8(SP)
                    Store           R6, 4(SP)
                    Store           R7, (SP)
                    Move            R5, R0              Azzera il contatore
CICLO:              LoadByte        R6, (R3)            Carica i byte che devono
                    LoadByte        R7, (R4)                  essere confrontati
                Branch if [R6]=[R7] PROSSIMO
                    Add             R5, R5, #1          Incrementa il contatore
PROSSIMO:           Add             R3, R3, #1          Incrementa i puntatori
                    Add             R4, R4, #1                alle liste
                    Subtract        R2, R2, #1          Reitera se la fine delle liste
                Branch if [R2]>[R0] CICLO                     non è stata raggiunta
                    Move            R2, R5              Restituisci il risultato in R2
                    Load            R7, (SP)            Ripristina i registri
                    Load            R6, 4(SP)
                    Load            R5, 8(SP)
                    Add             SP, SP, #12
                    Return
```

[117] 2.22 Sottoprogramma per la conversione di caratteri in una stringa ASCII (1)

D: Scrivere un sottoprogramma assemblativo chiamato ESCLAMA che accetta un singolo parametro in un registro che rappresenta l'indirizzo iniziale STRINGA nella memoria principale per una stringa di caratteri ASCII in byte consecutivi che rappresenta un insieme di frasi, con il carattere di controllo NUL (valore 0) alla fine della stringa. Il sottoprogramma dovrebbe scandire la stringa che inizia all'indirizzo STRINGA e sostituire ogni occorrenza di un punto (".") con un punto esclamativo ("!").

R:

```
                    Move            R2, #STRINGA        Puntatore alla stringa
                    Call            ESCLAMA
prossima istruzione
ESCLAMA:            Subtract        SP, SP, #12         Salva i registri
                    Store           R3, 8(SP)
                    Store           R4, 4(SP)
                    Store           R5, (SP)
                    Move            R3, #0x2E           Codice ASCII del punto
```

```
                Move            R4, #0x21       Codice ASCII del punto esclamativo
CICLO:          LoadByte        R5, (R2)
                Branch if [R5]=[R0]  FATTO      Controlla se NULL
                Branch if [R5]6=[R3] PROSSIMO   Se punto, allora sostituisci
                StoreByte       R4, (R2)            con punto esclamativo
PROSSIMO:       Add             R2, R2, #1
                Branch          CICLO
FATTO:          Load            R5, (SP)        Ripristina i registri
                Load            R4, 4(SP)
                Load            R3, 8(SP)
                Add             SP, SP, #12
                Return
```

[118] 2.23 Sottoprogramma per la conversione di caratteri in una stringa ASCII (2) (MOD.NEL.COMP.)

D: Scrivere un sottoprogramma assemblativo chiamato MAIUSCOLE che accetta un parametro in un registro che rappresenta l'indirizzo iniziale STRINGA nella memoria principale per una stringa di caratteri ASCII in byte consecutivi, con il carattere di controllo NUL (valore 0) alla fine della stringa. Il sottoprogramma dovrebbe scandire la stringa che inizia all'indirizzo STRINGA e sostituire ogni occorrenza di una lettera minuscola ("a"-"z") con la corrispondente lettera maiuscola ("A"-"Z").

R:

I codici ASCII per le lettere minuscole sono nell'intervallo esadecimale da 61 a 7A. Quando si trova un carattere in questo intervallo, esso può essere convertito in maiuscolo azzerando il bit 5. Un possibile programma `e :

```
                Move R2,#STRINGA            Puntatore alla stringa
                Call MAIUSCOLE
MAIUSCOLE:      Subtract SP, SP, #12        Salva i registri
                Store R3, 8(SP)
                Store R4, 4(SP)
                Store R5, (SP)
                Move R3, #0x61              Codice ASCII di a
                Move R4, #0x7a              Codice ASCII di z
CICLO:          LoadByte R5, (R2)
                Branch if [R5]=[R0] FATTO   Controlla se NUL
                Branch if [R5]<[R3] PROSSIMO Controlla se nell'intervallo da a a z
                Branch if [R5]>[R4] PROSSIMO
                And R5, R5, #0xDF           Crea l'ASCII della lettera maiuscola
                StoreByte R5, (R2)          Immagazzina la lettera maiuscola
PROSSIMO:       Add R2, R2, #1              Passa al prossimo carattere
                Branch CICLO
FATTO:          Load R5, (SP)               Ripristina i registri
                Load R4, 4(SP)
                Load R3, 8(SP)
                Add SP, SP, #12
                Return
```

[119] 2.24 Sottoprogramma per il conteggio delle parole in una stringa ASCII

D: Scrivere un sottoprogramma assemblativo chiamato PAROLE che accetta un parametro in un registro che rappresenta l'indirizzo iniziale STRINGA nella memoria principale per una stringa di caratteri ASCII in byte consecutivi, con il carattere di controllo NUL (valore 0) alla fine della stringa. La stringa rappresenta testo in italiano con il carattere spazio tra le parole. Il sottoprogramma deve determinare il numero di parole nella stringa (escludendo i caratteri di punteggiatura). Il risultato deve essere restituito al programma chiamante in un registro.

R: Le parole possono essere contate rilevando il carattere SPAZIO. Supponendo che le parole siano separate da caratteri SPAZIO singoli, un possibile programma `e il seguente:

```
                Move R2, #STRINGA Puntatore alla stringa
                Call PAROLE
prossima istruzione
PAROLE:         Subtract SP, SP, #12 Salva i registri
                Store R3, 8(SP)
                Store R4, 4(SP)
                Store R5, (SP)
                Move R3, #0x20 Codice ASCII per SPAZIO
                Move R4, R0 Azzera il contatore di parole
CICLO:          LoadByte R5, (R2)
                Branch if [R5]=[R0] FATTO Controlla se NUL
                Branch if [R5]6=[R3] PROSSIMO Controlla se SPAZIO
                Add R4, R4, #1 Incrementa il contatore di parole
```

```
PROSSIMO: Add R2, R2, #1 Passa al prossimo carattere
          Branch CICLO
FATTO:    Move R2, R4 Passa il risultato in R2
          Load R5, (SP) Ripristina i registri
          Load R4, 4(SP)
          Load R3, 8(SP)
          Add SP, SP, #12
          Return
```

[120] 2.25 Stima della lunghezza di programmi assemblativi RISC reali(DUBBIOSA)(MOD.NEL.COMP.)

D: Si consideri il programma assemblativo generico in stile RISC, per la somma di una lista di numeri, proposto in Figura 2.8. Tenendo conto del confronto condotto nell'esempio dell'Approfondimento applicativo 2.6 fra caratteristiche delle ISA dei processori NIOS II e ARM, produrre una stima approssimativa del numero di istruzioni di programmi assemblativi per tali processori che concretamente realizzino il suddetto programma assemblativo generico, motivando la risposta.

R:

Si può effettuare una rapida stima del numero di istruzioni occorrenti per riformulare i due suddetti programmi nei linguaggi assemblativi dei quattro processori in questione contando quelle occorrenti nei programmi mostrati nell'Approfondimento applicativo 2.6 e tenendo conto delle semplificazioni indicate sopra, dovute alla maggior semplicità dell'algoritmo di somma di una lista di numeri. Per i processori CISC va anche notato che uno degli operandi dell'operazione di somma può trovarsi in memoria, ovvero non `e necessario caricarlo prima in un registro. La seguente tabella elenca, per ciascuno dei quattro processori, il numero p di istruzioni del relativo programma nell'esempio di calcolo del prodotto scalare e la relativa stima del numero s di istruzioni di un programma per la somma di una lista di numeri.

Processore	p	s
NIOS II	15	11
ARM	10	8
ColdFire	10	7
IA-32	11	8

[121] 2.26 Stima della lunghezza di programmi assemblativi CISC reali(MOD.NEL.COMP.)

D: Si consideri il programma assemblativo generico in stile CISC, per la somma di una lista di numeri, proposto in Figura 2.26. Tenendo conto del confronto condotto nell'esempio dell'Approfondimento applicativo 2.6 fra caratteristiche delle ISA dei processori ColdFire e IA-32, produrre una stima approssimativa del numero di istruzioni di programmi assemblativi per tali processori che concretamente realizzino il suddetto programma assemblativo generico, motivando la risposta.

R: Il numero di istruzioni nei 2 processori è circa equivalente. Nel coldfire si risparmiano istruzioni grazie al modo di indirizzamento con autoincremento mentre nel IA32 grazie all'autodecremento del contatore N. Quest'ultimo però impiega 2 istruzioni invece di una per l'inizializzazione e l'incremento dell'indice.

[122] 2.27 Programma assemblativo CISC generico con direttive di assemblatore

D: Si consideri il programma mostrato in Figura P2.2, quale soluzione al Problema risolto 2.2 nell'Eserciziario. Produrre un programma assemblativo generico che lo risolva in stile CISC, con il vincolo che non più di un operando di ciascuna istruzione sia collocato in memoria.

R:

```
LISTA     EQU        1000          Indirizzo iniziale della lista
          ORIGIN     400
          Move R2, #LISTA R2       punta all'inizio della lista
          Move R3, 4(R2) R3        è un contatore, inizializzato a n
          Move R4, R2
          Add R4, #8               R4 punta al primo numero
          Move R5, (R4) R5         contiene il pi`u piccolo numero trovato finora
CICLO:    Subtract R3, #1          Decrementa il contatore
          Branch=0 FATTO           Finito se R3 `e uguale a 0
          Compare R5, (R4)+
          Branch0 CICLO            Controlla se hai trovato un numero pi`u piccolo
          Move R5, 4(R4)           Aggiorna il numero più piccolo trovato finora
          Branch CICLO
FATTO:    Move (R2), R5            Immagazzina il numero pi`u piccolo in MINIMO

          ORIGIN 1000
MINIMO:   RESERVE 4                Spazio per il numero pi`u piccolo trovato
N:        DATAWORD 7               Numero di elementi nella lista
ELEMENTI: DATAWORD 4,5,3,6,1,8,2   Elementi nella lista
END
```

Capitolo 3: Operazioni di ingresso e uscita

[123] 3.1 Controllo del bit di stato in un circuito di interfaccia di I/O

D: Il bit dello stato di ingresso in un circuito di interfaccia è posto a zero quando il registro dei dati di ingresso viene letto. Perché questo è importante?

R: Per assicurare che i dati siano letti solo una volta.

[124] 3.2 Controllo di I/O per la visualizzazione esadecimale di una sequenza di byte

D: Scrivere un programma assemblativo che visualizzi in formato esadecimale il contenuto di dieci byte in memoria centrale su una linea di uno schermo. I dieci byte sono in memoria a partire dalla locazione LOC e si mostrano due caratteri esadecimali per ciascun byte. I contenuti di byte successivi dovrebbero essere separati da uno spazio quando vengono mostrati sullo schermo.

R: Assumendo i registri dell'interfaccia dello schermo in Figura 3.3, si può usare il seguente programma:

```
                Move R2, #LOC Ottieni l'indirizzo LOC.
                Move R3, #DISP DATA Ottieni l'indirizzo dello schermo.
                Move R4, #10 Inizializza il contatore di byte.
CICLO:          LoadByte R5, (R2) Carica un byte.
                And R6, R5, #0xF0 Seleziona i 4 bit di ordine alto.
                LShiftR R6, R6, #4 Scorrimento a destra di 4 bit.
                LoadByte R6, TABLE(R6) Carica il carattere per lo schermo.
                Call DISPLAY
                And R6, R5, #0x0F Seleziona i 4 bit di ordine basso.
                LoadByte R6, TABLE(R6) Carica il carattere per lo schermo.
                Call DISPLAY
                Move R6, #0x20 Codice ASCII per SPAZIO.
                Call DISPLAY
                Add R2, R2, #1 Incrementa il puntatore.
                Subtract R4, R4, #1 Decrementa il contatore di byte.
                Branch if [R4]>[R0] CICLO Reitera se non hai finito.
prossima istruzione
DISPLAY:        LoadByte R7, 4(R3)
                And R7, R7, #4 Controlla il bit di stato DOUT.
                Branch if [R7]=[R0] DISPLAY
                StoreByte R6, (R3) Manda il carattere allo schermo.
                Return
                ORIGIN 0x1000
TABLE:          DATABYTE 0x30,0x31,0x32,0x33 Tabella che contiene
                DATABYTE 0x34,0x35,0x36,0x37 i necessari
                DATABYTE 0x38,0x39,0x41,0x42 caratteri ASCII.
                DATABYTE 0x43,0x44,0x45,0x46
```

[125] 3.3 Sottoprogrammi e ISR

D: Qual è la differenza tra un sottoprogramma e una routine di servizio di interruzione?

R: Un sottoprogramma è chiamato da un'istruzione di programma al fine di eseguire una funzione necessaria al programma chiamante. Una routine di servizio delle interruzioni è avviata da un evento come un'operazione
di ingresso o un errore hardware. La funzione che svolge può non essere affatto correlata al programma in esecuzione al momento dell'interruzione. Quindi, non deve incidere su alcun dato o informazione di stato relativi a tale programma.

[126] 3.4 Controllo di bit di stato nell'interfaccia di una tastiera

D: Nella prima istruzione And in Figura 3.4 il valore immediato 2 viene usato quando si controlla la variabile di condizione KIN, ma in Figura 3.5 si usa il valore immediato 1 nella prima istruzione TestBit quando si controlla la stessa variabile di condizione. Spiegare la differenza.

R: Nell'istruzione And il valore immediato 2 rappresenta un valore binario. Nell'istruzione TestBit l'operando immediato specifica uno dei 32 bit che sono identificati come bit da b0 a b31. Pertanto, il valore immediato 1 corrisponde a b1.

[127] 3.5 Programma RISC con controllo di I/O per visualizzare una sequenza di bit

D: Assumere che la locazione di memoria BINARIO contenga una sequenza di 16 bit. Si desidera visualizzare questi bit come una stringa di 0 e 1 su uno schermo che ha l'interfaccia mostrata in Figura 3.3. Scrivere un programma in stile RISC che realizzi questo compito.

R: Assumendo i registri d'interfaccia dello schermo di Figura 3.3, un possibile programma in stile RISC è:

```
                Move            R2, #BINARIO        Ottieni l'indirizzo BINARIO.
                LoadHalfword    R2, (R2)            Carica la sequenza di 16 bit.
                Move            R3, #DISP DATA      Ottieni l'indirizzo dello schermo.
                Move            R4, #16             Inizializza il contatore dei bit.
                Or              R5, R0, #0x8000     Imposta a 1 il bit 15.
CICLO:          And             R6, R2, R5          Valuta un bit.
        Branch if [R6]>[R0]     UNO                 Salta se il bit è 1.
```

```
                    Move            R7, #0x30           Codice ASCII di 0.
                    Branch          BITOUT
UNO:                Move            R7, #0x31           Codice ASCII di 1.
BITOUT:             Call            DISPLAY
                    LShiftR         R5, R5, #1          Scorrimento al prossimo bit.
                    Subtract        R4, R4, #1          Decrementa il contatore dei bit.
                Branch if [R4]>[R0] CICLO               Reitera se non hai finito.
prossima istruzione
DISPLAY:            LoadByte        R8, 4(R3)
                    And             R8, R8, #4          Controlla il bit di stato DOUT.
                Branch if [R8]=[R0] DISPLAY
                    StoreByte       R7, (R3)            Invia il carattere allo schermo.
                    Return
```

[128] 3.6 Programma CISC con controllo di I/O per visualizzare una sequenza di bit

D: Assumere che la locazione di memoria BINARIO contenga una sequenza di 16 bit. Si desidera visualizzare questi bit come una stringa di 0 e 1 su uno schermo che ha l'interfaccia mostrata in Figura 3.3. Scrivere un programma in stile CISC che realizzi questo compito.

R: Assumendo i registri d'interfaccia dello schermo di Figura 3.3, un possibile programma in stile CISC è:

```
                    MoveHalfword    R2, BINARIO         Carica la sequenza di 16 bit.
                    Move            R3, #15             Inizializza il contatore dei bit.
CICLO:              TestBit         R2, R3              Valuta un bit.
                    Branch=/0       ONE                 Salta se il bit è 1.
                    Move            R4, #0x30           Codice ASCII di 0.
                    Branch          BITOUT
ONE:                Move            R4, #0x31           Codice ASCII di 1.
BITOUT:             Call            DISPLAY
                    Subtract        R3, #1              Decrementa il contatore dei bit.
                    Branch>=0       CICLO               Reitera se non hai finito.
prossima istruzione
DISPLAY:            TestBit         DISP STATUS, #2     Attendi che lo schermo sia pronto.
                    Branch=0        DISPLAY
                    MoveByte        DISP DATA, R4       Invia il carattere allo schermo.
                    Return
```

[129] 3.7 Programma RISC con controllo di I/O per determinare se una parola sia palindroma

D: Scrivere un programma in stile RISC che determini se una parola inserita da un utente sulla tastiera sia palindroma, ovvero una parola che è la stessa quando i suoi caratteri sono scritti in ordine normale o inverso. Il programma dovrebbe mostrare un simbolo di richiesta dati e permettere all'utente di inserire i caratteri di una qualsiasi parola da tastiera, seguiti dal ritorno carrello (CR). Il programma dovrebbe leggere i caratteri e immagazzinarli nella memoria centrale. Dovrebbe quindi analizzare la parola per determinare se essa sia palindroma. Infine, dovrebbe mostrare un messaggio per indicare il risultato dell'analisi.

R: Un possible programma in stile RISC è:

```
KBD DATA            EQU             0x4000              Specifica gli indirizzi dei registri dati
DISP DATA           EQU             0x4010                          di tastiera e schermo.
                    Move            R2, #LOC            Locazione dove sarà memorizzata la parola.
                    Move            R3, #KBD DATA R3    punta al registro dati della tastiera.
                    Move            R4, #DISP DATA R4   punta al registro dati dello schermo.
                    Move            R5, #0x0D           Codice ASCII di Ritorno Carrello (CR).
                    Move            R8, #PROMPT
                    Call            DISPLAY             Visualizza il prompt.
/* Leggi la parola. */
LEGGI:              LoadByte        R6, 4(R3)           Leggi il registro di stato della tastiera.
                    And             R6, R6, #2          Controlla il bit di stato KIN.
                Branch if [R6]=[R0] LEGGI
                    LoadByte        R7, (R3)            Leggi un carattere dalla tastiera.
                    StoreByte       R7, (R2)            Scrivi il carattere in memoria principale
                    Add             R2, R2, #1                      e incrementa il puntatore.
ECO:                LoadByte        R6, 4(R4)           Leggi il registro di stato dello schermo.
                    And             R6, R6, #4          Controlla il bit di stato DOUT.
                Branch if [R6]=[R0] ECO
                    StoreByte       R7, (R4)            Invia il carattere allo schermo.
                Branch if [R5]6=[R7] LEGGI              Reitera se il carattere non `e CR.
/* Determina se `e palindroma. */
                    Subtract        R8, R2, #2          Puntatore della scansione a rovescio.
                    Move            R2, #LOC
CICLOD:             LoadByte        R6, (R2)            Scandisci la parola
```

```
                    LoadByte        R7, (R8)            in direzioni opposte e controlla
                Branch if [R6]6=[R7] NONP               se le scansioni coincidono.
                    Add             R2, R2, #1
                    Subtract        R8, R8, #1
                Branch if [R2]<[R8] CICLOD
/* Visualizza il risultato. */
                    Move            R8, #SIPAL          Visualizza che è
                    Call            DISPLAY                     palindroma.
                    Branch          FATTO
NONP:               Move            R8, #NOPAL          Visualizza che non è
                    Call            DISPLAY                     palindroma.
FATTO:          prossima istruzione
DISPLAY:            LoadByte        R6, 4(R4)           Leggi il registro di stato dello schermo.
                    And             R6, R6, #4          Controlla il bit di stato DOUT.
                Branch if [R6]=[R0] DISPLAY
                    LoadByte        R7, (R8)            Reitera l'invio di caratteri
                    StoreByte       R7, (R4)                allo schermo finchè
                    Add             R8, R8, #1              il carattere CR non sia
                Branch if [R7]6=[R5] DISPLAY            stato raggiunto.
                    Return

                    ORIGIN          0x1000
PROMPT:             DATABYTE 0x44, 0x69, 0x67, 0x69, 0x74, 0x61, 0x20, 0x6C, 0x61, 0x20
                    DATABYTE 0x70, 0x61, 0x72, 0x6F, 0x6C, 0x61, 0x0D
SIPAL:              DATABYTE 0x50, 0x61, 0x6C, 0x69, 0x6E, 0x64, 0x72, 0x6F, 0x6D, 0x61, 0x0D
NOPAL:              DATABYTE 0x4E, 0x6F, 0x6E, 0x20
                    DATABYTE 0x70, 0x61, 0x6C, 0x69, 0x6E, 0x64, 0x72, 0x6F, 0x6D, 0x61, 0x0D
```

[130] 3.8 Programma CISC con controllo di I/O per determinare se una parola sia palindroma

D: Scrivere un programma in stile CISC che determini se una parola inserita da un utente sulla tastiera sia palindroma, ovvero una parola che è la stessa quando i suoi caratteri sono scritti in ordine normale o inverso. Il programma dovrebbe mostrare un simbolo di richiesta dati e permettere all'utente di inserire i caratteri di una qualsiasi parola da tastiera, seguiti dal ritorno carrello (CR). Il programma dovrebbe leggere i caratteri e immagazzinarli nella memoria centrale. Dovrebbe quindi analizzare la parola per determinare se essa sia palindroma. Infine, dovrebbe mostrare un messaggio per indicare il risultato dell'analisi.

R: Un possibile programma in stile RISC è:

```
KBD_DATA            EQU         0x4000          Specifica gli indirizzi dei registri dati
DISP_DATA           EQU         0x4010                  di tastiera e schermo.
                    Move        R2, #LOC        Locazione dove sarà memorizzata la parola.
                    Move        R3, #KBD_DATA R3        punta al registro dati della tastiera.
                    Move        R4, #DISP_DATA R4       punta al registro dati dello schermo.
                    Move        R5, #0x0D               Codice ASCII di Ritorno Carrello (CR).
                    Move        R8, #PROMPT
                    Call        DISPLAY         Visualizza il prompt.
/* Leggi la parola. */
LEGGI:              TestBit     4(R3), #1       Attendi l'immissione del carattere.
                    Branch=0    LEGGI
                    MoveByte    (R2), (R3)      Scrivi il carattere in memoria principale.
ECO:                TestBit     4(R4), #2       Attendi che lo schermo sia pronto.
                    Branch=0    ECO
                    MoveByte    (R4), (R2)      Invia il carattere allo schermo.
                    CompareByte R5, (R2)+       Controlla se è CR.
                    Branch6=0   LEGGI           Reitera se il carattere non è CR.
/* Determina se `e palindroma. */
                    Subtract    R2, #1          Puntatore della scansione a rovescio.
                    Move        R7, #LOC
CICLOD:             MoveByte    R6, –(R2)       Scandisci la parola
                    CompareByte R6, (R7)+               in direzioni opposte e controlla
                    Branch6=0   NONP                    se le scansioni coincidono.
                    Compare     R7, R2
                    Branch<0    CICLOD
/* Visualizza il risultato. */
                    Move        R8, #SIPAL      Visualizza che è
                    Call        DISPLAY                 palindroma.
                    Branch      FATTO
NONP:               Move        R8, #NOPAL      Visualizza che non è
```

```
                    Call        DISPLAY                 palindroma.
FATTO:              prossima istruzione
DISPLAY:            TestBit     4(R4), #2       Attendi che lo schermo sia pronto.
                    Branch=0    DISPLAY
                    MoveByte    (R4), (R8)      Invia un carattere allo schermo.
                    CompareByte R5, (R8)+       Controlla se è CR.
                    Branch6=0   DISPLAY         Reitera se non è CR.
                    Return

                   ORIGIN       0x1000
PROMPT:            DATABYTE 0x44, 0x69, 0x67, 0x69, 0x74, 0x61, 0x20, 0x6C, 0x61, 0x20
                    DATABYTE 0x70, 0x61, 0x72, 0x6F, 0x6C, 0x61, 0x0D
SIPAL:             DATABYTE 0x50, 0x61, 0x6C, 0x69, 0x6E, 0x64, 0x72, 0x6F, 0x6D, 0x61, 0x0D
NOPAL:             DATABYTE 0x4E, 0x6F, 0x6E, 0x20
                    DATABYTE 0x70, 0x61, 0x6C, 0x69, 0x6E, 0x64, 0x72, 0x6F, 0x6D, 0x61, 0x0D
```

[131] 3.9 Inizializzazione del gestore delle interruzioni

D: In Figura 3.9, il bit di abilitazione delle interruzioni in PS è assegnato per ultimo nel programma principale. Perché? È importante l'ordine per le operazioni precedenti nel programma principale? Perché o perché no?

R: Le operazioni precedenti all'abilitazione inizializzano l'interrupt Handler e le ISR.
Se l'abilitazione fosse eseguita prima potrebbero aversi interruzioni servite da ISR non inizializzate.

[132] 3.10 Gestore di interruzioni multiple

D: Sebbene possano risultare pendenti più richieste di interruzioni, solo una richiesta verrà gestita per ciascun salto a ILOC in Figura 3.9. Vero o falso? Spiegare.

R: Falso. Tutte le richieste indicate nel registro IPENDING saranno servite.

[133] 3.11 Uso delle eccezioni per il controllo di errore in istruzioni aritmetiche

D: Un programma utente potrebbe controllare se un divisore è zero subito prima di ciascuna operazione di divisione e quindi eseguire azioni opportune senza invocare il sistema operativo. Dare motivazioni per cui questo potrebbe essere preferibile o no, rispetto a permettere la segnalazione di un'interruzione da eccezione su un'effettiva situazione di divisione per zero in un programma dell'utente.

R: Sebbene questo sia fattibile, se ne darebbe senza necessità l'onere al programmatore e si complicherebbe il programma.

[134] 3.12 Uso delle interruzioni per la visualizzazione di una linea di caratteri

D: Il problema di visualizzare su uno schermo una linea di caratteri immessa da tastiera è risolto nell'Esempio 3.2 usando le interruzioni dall'interfaccia della tastiera e il polling dell'interfaccia dello schermo. Si potrebbe risolvere lo stesso problema usando le interruzioni dall'interfaccia dello schermo e il polling dell'interfaccia della tastiera? Se no, perché? Se sì, sarebbe una soluzione più conveniente, o meno, o altrettanto, e perché?

R: Si potrebbe risolvere utilizzando le interruzioni dell'interfaccia dello schermo e il polling dall'interfaccia della tastiera, ma ciò non sarebbe conveniente, poiché l'i/o da tastiera è molto più lento rispetto alla velocità di trasferimento in uscita dal computer al display.

[135] 3.13. Registri di collegamento nei processori NIOS II e ARM

D: L'architettura NIOS II dispone di un registro di collegamento per il rientro da eccezioni, r29 o ea, distinto da quello usato per il rientro da sottoprogramma, r31 o ra. Invece, l'architettura ARM usa in entrambi i casi lo stesso registro, R14 o LR. Quali (differenze di) caratteristiche delle due architetture possono spiegare questa diversità?

R: Semplicemente, un ALTERA NIOS II distingue solo 2 casi di interrupt: hardware interrupt e tutti gli altri, chiamati in seguito "exception". I primi hanno la priorità dinnanzi ai secondi - vengono processati prima; motivo per cui vengono gestite da due coppie di celle diverse. Invece, un ARM definisce ben 7 casi di interrupt, e per ognuno ha una modalità specifica: FIQ, IRQ, SVC(che vale sia per un software interrupt che per un reset), Abort(vale per le violazioni di accesso, sia di dati, che di istruzioni), e Undefined(istruzione non implementata). Ad ognuna viene assegnata una sua priorità; nel caso il processore dovesse entrare in una delle suddette modalita', alcuni dei registri verranno sostituiti con lo stesso numero di registri "banked", un set diverso per ogni tipo di eccezione. Quindi, al posto del registro R14 verrà messo sempre lo stesso registro banked, ogni volta con un suffisso nel nome diverso, variabile in base all'interrupt(R14_modalità). Il contenuto di registri originali rimarrà intatto.

[136] 3.14. Rientro da ISR nei processori NIOS II e ARM

D: Sia nei processori NIOS II che nei processori ARM, prima del rientro da servizio di interruzione di I/O va apportata una correzione (decremento di 4) al contenuto del registro di collegamento. La necessità della correzione è dovuta allo stesso motivo nelle due architetture in questione? Se sì, quale? Se no, a quali (diversi) motivi? La suddetta correzione non va apportata nel caso di rientro da servizio di interruzione software (Trap). Perché?

R: Un processore Altera NIOS II inizia ad eseguire un interrupt (un hardware interrupt) senza completare l'esecuzione dell'istruzione corrente, dunque il suddetto decremento e' necessario al fine di poter rieseguire l'istruzione intermedia; mentre, nel caso di una trap il processore salva l'indirizzo della prossima istruzione, e poi trasferisce il controllo dell'esecuzione all'exception handler; dunque, l'istruzione attuale(cioè la trap) viene

completata. Invece, un ARM opera diversamente: supponiamo che il processore prelevi l'istruzione I1 dall'indirizzo A. Il contenuto del PC diventa A+4, dopo di che inizia l'esecuzione di I1. Prima che essa venga completata, il processore preleva l'istruzione I2 dall'indirizzo A+4, ed incrementa PC, che ora diventa A+8. Se alla fine dell'esecuzione di I1 al processore serve una richiesta di interruzione ordinaria, esso entra una modalità particolare, chiamata IRQ e salva il contenuto di PC, vale a dire A+8; poichè non ha eseguito l'istruzione I2, deve decrementare il valore di 4, al fine di poterne effettuare l'esecuzione correttamente. Nel caso di una trap, invece, l'ARM entra in una modalità diversa, chiamata SWI, dove il valore salvato e' l'indirizzo corretto.

Capitolo 4: Software

[137] 4.1 Espansione di macro nell'assemblaggio in due passi

D:Perché l'espansione di macro viene effettuata nella prima passata dell'assemblaggio in due passi?

R: L'assemblaggio in due passi avviene per risolvere il problema che sorge nel momento in cui un nome appare come operando, prima che il suo valore venga definito (per esempio quando si richiede un indirizzo di un'etichetta definita solo dopo nel programma). Infatti con un solo passo, l'assemblatore non può determinare l'indirizzo della diramazione di destinazione in quanto in valore dell'indirizzo di etichetta non è stato ancora registrato nella tabella simbolo.

[138] 4.2 Simboli esterni e simboli esportabili

D: Nella costruzione modulare di programmi, qual è la differenza tra simboli esterni e simboli esportabili? Che ruolo hanno nel processo di assemblaggio e collegamento di un programma oggetto?

R: Quando viene compilato un file sorgente, vengono generati tanti piccoli file di output; ciascuno di essi non sara' un completo oggetto programma. Ognuno di essi conterra' riferimenti a simboli esterni, indirizzi di variabili,istruzioni (in generale simboli) definite in altri file. Durante il processo, l'assembler li identifica, ne costruisce una lista con i loro nomi, e le relative istruzioni, e ne salva il contenuto nel relativo file. Piu' tardi, un programma linker controlla i riferimenti di ciascun simbolo esterno; esso ha bisogno di posizioni relative degli indirizzi di variabili, cosi' ne puo' ricostruire le posizioni assolute. Le info su di esse, che potrebbero essere menzionati in altri file devono essere esportate da ogni file sorgente. Di solito, e' necessario l'intervento del programmatore per indicare le variabili da esportare; esse sono contenute, come i simboli esterni,(i simboli esportati da un modulo possono essere importati come esterni in un altro modulo) in una lista dentro ogni file generato.

[139] 4.3 Compilatore ottimizzante

D: Un compilatore ottimizzante opera spesso in più fasi. Perché? In cosa può consistere l'ottimizzazione che compie?

R: Questo modo di operare e' dovuto al fatto che un compilatore a più fasi può compiere alcune ottimizzazioni nel codice ad alto livello, in particolare - nei cicli; poiche' molte volte viene utilizzato un contatore numerico, piuttosto che caricarlo e salvarlo in memoria ad ogni incremento, il compilatore ottimizzante lo mantiene in uno dei registri del processore, in modo da fare le operazioni LOAD e STORE una volta soltanto: all'inizio, e alla fine del ciclo.

[140] 4.4 Modalità di debug

D: Le interruzioni software (Trap) possono essere usate da un debugger in due modalità diverse. Quali? In che cosa differiscono?

R: Un debugger e' un programma che permette al programmatore di rilevare errori logici all'interno di un programma, permettendo di fermarne l'esecuzione, generando un interrupt nei punti desiderati, chiamato "Trap", al fine di poter esaminare il contenuto di vari registri del processore e locazioni di memoria.

[141] 4.5 Programma C per la visualizzazione esadecimale di una sequenza di byte

D: Scrivere un programma C che visualizzi in formato esadecimale il contenuto di dieci byte in memoria centrale su una linea di uno schermo. I dieci byte sono in memoria a partire dalla locazione LOC e si mostrano due caratteri esadecimali per ciascun byte. I contenuti di byte successivi dovrebbero essere separati da uno spazio quando vengono mostrati sullo schermo. Si assuma che i registri dell'interfaccia dello schermo abbiano gli indirizzi come da Figura 3.3 e si effettui il controllo di I/O da programma.

R:

```
#define DISP_DATA   (volatile char *) 0x4010
#define DISP_STATUS (volatile char *) 0x4014
#define LOC         (char *) <indirizzo arbitrario>

/* Tabella dizionario per convertire i numeri 0-15 nei caratteri '0'-'F'. */

char tabella[] = {
          0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
          0x38, 0x39, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46
};

void main()
{
   char ch, byte, *byte_ptr;
   int i;

   byte_ptr = LOC;                          /* Imposta posizione iniziale per i byte da visualizzare. */
   for (i = 0; i<10; i++) {                 /* Mostra 10 byte. */
      byte = *byte_ptr;                     /* Leggi byte successivo dalla memoria. */
      ch = tabella [(byte >> 4) & 0xF];     /* Converti cifra esadecimale alta in carattere. */
      while ((*DISP_STATUS & 0x4) == 0);    /* Attendi che lo schermo sia pronto. */
      *DISP_DATA = ch;                      /* Trasferisci il carattere allo schermo. */
      ch = tabella[byte & 0xF];             /* Converti cifra esadecimale bassa in carattere. */
      while ((*DISP_STATUS & 0x4) == 0);    /* Attendi che lo schermo sia pronto. */
      *DISP_DATA = ch;                      /* Trasferisci il carattere allo schermo. */
      ch = 0x20;                            /* Inserisci spazio per separare i byte visualizzati. */
      while ((*DISP_STATUS & 0x4) == 0);    /* Attendi che lo schermo sia pronto. */
      *DISP_DATA = ch;                      /* Trasferisci il carattere allo schermo. */
     ++byte_ptr;                            /* Aggiorna il puntatore al prossimo byte */
   }
}
```

[142] 4.6 Programma C per la visualizzazione di una sequenza di bit

D: Assumere che la locazione di memoria BINARIO contenga una sequenza di 16 bit. Si desidera visualizzare questi bit come una stringa di 0 e 1 su uno schermo che ha l'interfaccia mostrata in Figura 3.3. Scrivere un programma C RISC che realizzi questo compito, con controllo di I/O da programma.

R:

```
#define DISP_DATA (volatile char*) 0x4010
#define DISP_STATUS (volatile char*) 0x4014
void main(){
short int H=0x8000;
volatile short int* BINARIO=(short int*)0x80A3;
int i;cost
for(i=0;i<16;i++){
while((*DISP_STATUS & 0x4)==0);
*DISP_DATA=(((*BINARIO & H)==0)? '0' : '1');
H>>1;
}}
```

Oppure

```
#define DISP_DATA    (volatile char *) 0x4010
#define DISP_STATUS  (volatile char *) 0x4014

/ * Assumi che la quantita' a 16 bit sia immagazzinata in una parola di 32 bit. */
#define BINARY       (unsigned int *) <indirizzo arbitrario>

void main()
{
   char ch;
   unsigned int *binary_ptr, binary_value, bit;

   binary_ptr = BINARY;                     /* Imposta puntatore al dato da visualizzare. */
   binary_value = *binary_ptr;              /* Leggi dato. */
   for (int i = 15; i>= 0; i--) {           /* Visualizza 16 bit. */
      bit = binary_value >> i;              /* Scorrimento a destra per isolare il bit successivo. */
      if (bit % 2 == 0)                         /* Determina il carattere da mostrare in base al bit. */
        ch = '0';
      else
        ch = '1';

      while ((*DISP_STATUS & 0x4) == 0); /* Attendi che lo schermo sia pronto. */
      * DISP_DATA = ch;                     /* Trasferisci il carattere allo schermo. */
   }
}
```

[143] 4.7 Programma C con sottoprogramma assemblativo RISC per la conversione BCD->ASCII di

una sequenza di cifre

D: Scrivere un sottoprogramma assemblativo UNPACK in stile RISC che effettui la conversione di una sequenza di cifre decimali da codice BCD (due cifre per byte) ad ASCII (una cifra per byte). Tre parametri sono passati al sottoprogramma in altrettanti registri: i due indirizzi iniziali in memoria delle sequenze BCD e ASCII e il numero di cifre da convertire. Scrivere poi un programma C, dotato di un array di costanti per la sequenza BCD, che chiami il sottoprogramma e visualizzi quindi la sequenza ASCII che ne ottiene in ritorno.

R:

```
#include<stdio.h>
#define SIZE 3
void UNPACK();
void main(){
int i;
const unsigned char BCD[]={(1<<4)|2,(3<<4)|4,(9<<4)|0};
unsigned char ASCII[2*SIZE];
asm(“move R2,#BCD”);
asm(“move R3,#ASCII”);
asm(“move R4,SIZE”);
UNPACK();
for(i=0;i<(2*SIZE);i++)
printf(“%2hhx”,ASCII[i]);
}
.text
.org
UNPACK:
subi sp,sp,8
stw r5,4(sp)
stw r6,(sp)
Ciclo:
ldb r5,(r2)
srl r6,r5,4
add r6,r6,#Ox30
stb r6,(r3)
add r3,r3,#1
andi r5,r3,0x0F
add r5,r5,#0x30
stb r5,(r3)
add r3,r3,#1
add r2,r2,#1
subtract r4,r4,#1
bgt r4,r0,Ciclo
ldw r6,(sp)
ldw r5,4(sp)
addi sp,sp,8
ret
.global UNPACK
.end
```

Capitolo 5: Struttura di base del processore

[144] 5.1 Periodo di clock minimo di un circuito combinatorio di elaborazione dati

D: Il ritardo di propagazione attraverso il circuito combinatorio in Figura 5.2 è di 600 ps (picosecondi). I registri richiedono un tempo di impostazione (setup time, v. Appendice A) di 50 ps, e il ritardo di propagazione massimo dall'ingresso del clock all'uscita Q è di 70 ps.

(a) Qual è il periodo di clock minimo richiesto per il corretto funzionamento del circuito?

(b) Assumere che il circuito sia riorganizzato in tre stadi come in Figura 5.3, così che il circuito combinatorio in ciascuno stadio abbia un ritardo di 200 ps. Qual è il periodo di clock minimo in questo caso?

R:

a. Minimo periodo di clock = 70 + 600 + 50 = 720 ps.

b. Minimo periodo di clock = 70 + 200 + 50 = 320 ps.

[145] 5.2 Contenuti dei registri interstadi durante l'esecuzione di un'istruzione Load

D: Al momento in cui l'istruzione

Load R6, 1000(R9)

viene prelevata, R6 e R9 contengono i valori 4200 e 85320, rispettivamente. La locazione di memoria 86320 contiene 75900. Mostrare i contenuti dei registri interstadi in Figura 5.8 durante ognuno dei 5 passi di esecuzione di questa istruzione.

R:

I registri sono sempre abilitati per far passare i dati da uno stadio a quello successivo. L’informazione

disponibile all'ingresso di un registro `e caricata nel registro alla fine del ciclo di clock. Quando una nuova istruzione viene caricata nel registro d'istruzione al passo 1, i contenuti dei registri selezionati dai bit $IR_{31-27}$ e $IR_{26-22}$ vengono caricati nei registri RA e RB, rispettivamente, alla fine del passo 2. Prima di ci`o, i contenuti di questi registri sono determinati dall'istruzione precedente.

Passo	RA	RB	RZ	RM	RY
1	*	*	*	*	*
2	*	*	*	*	*
3	85320	4200	*	*	*
4	85320	4200	86320	4200	*
5	85320	4200	86320	4200	75900

* Questi valori sono determinati dall'istruzione precedente.

[146] 5.3 Contenuti dei registri interstadi durante l'esecuzione di un'istruzione Store

D: Al momento in cui l'istruzione
Store R6, 1000(R9)
viene prelevata, R6 e R9 contengono i valori 4200 e 85320, rispettivamente. La locazione di memoria 86320 contiene 75900. Mostrare i contenuti dei registri interstadi in Figura 5.8 durante ognuno dei 5 passi di esecuzione di questa istruzione.

R:
Store R6,1000(R9)
R6: 4200
R9: 85320
86320: 75900
passo 2:
RA: 85320
RB: 4200
passo 3:
RZ: 86320 (RA+1000(immediate value))
RM: 4200
passo 4:
memory address <-- RZ
memory data <-- RM
write memory (data RM memorizzata in address RZ)
nel passo 5 non avviene alcuna azione

[147] 5.4 Contenuti dei registri interstadi durante l'esecuzione di un'istruzione logica (1)

D: Al momento in cui l'istruzione
And R5, R6, R9
viene prelevata, R6 e R9 contengono i valori 0xFFFFA007 e 0x0000FFFF, rispettivamente. Mostrare i contenuti dei registri interstadi in Figura 5.8 durante ognuno dei 5 passi di esecuzione di questa istruzione.

R:
And R5,R6,R9
R6: 0xFFFFA007
R9: 0x0000FFFF
passo 2:
RA: 0xFFFFA007
RB: 0x0000FFFF
passo 3:
RZ: 0x0000A007 (RA AND RB)
RM non utilizzato
passo 4:
RY: 0x0000A007 (RZ)
passo 5:
R5: 0x000A007 (RY)

[148] 5.5 Campi dei registri nella codifica RISC delle istruzioni

D: La Figura 5.12 mostra i campi di bit assegnati agli indirizzi dei registri per gruppi diversi di istruzioni. Perché è importante usare le stesse posizioni dei campi per tutte le istruzioni?

R: Sapere dove sono i campi dei registri nelle istruzioni rende possibile all'hardware del processore leggere i registri sorgente prima di decodificare l'istruzione.

[149] 5.6 Contenuti dei registri interstadi durante l'esecuzione di un'istruzione aritmetica

D: A un certo punto nell'esecuzione di un programma, i registri R4, R6 e R7 contengono i valori 1000, 7500 e 2500, rispettivamente. Mostrare i contenuti dei registri RA, RB, RZ, RY e R6 nella Figura 5.8 durante i passi da 3 a 5 di esecuzione dell'istruzione
Subtract R6, R4, R7
e anche durante il passo 1 dell'istruzione che viene successivamente prelevata.

R: I contenuti dei registri sono letti al passo 2 e caricati nei registri interstadi al termine di tale periodo di clock.

Passo	RA	RB	RZ	RY	R6
3	1000	2500	*	*	7500
4	1000	2500	-1500	*	7500
5	1000	2500	-1500	-1500	7500
1	1000	2500	-1500	-1500	-1500

* Questi valori sono determinati dall'istruzione precedente.

[150] 5.7 Contenuti dei registri interstadi durante l'esecuzione di un'istruzione logica (2)

D: L'istruzione

And R4, R4, R8

è immagazzinata nella locazione 0x37C00 in memoria. Nel momento in cui quest'istruzione viene prelevata, i registri R4 e R8 contengono i valori 0x1000 e 0xB2500, rispettivamente. Determinare i valori nei registri PC, R4, RA, RM, RZ e RY delle Figure 5.8 e 5.10 in ciascun ciclo di clock durante l'esecuzione dell'istruzione, e anche nel primo ciclo di clock della successiva istruzione.

R: Il risultato dell'operazione viene caricato in R4 alla fine del passo 5.

Passo	PC	R4	RA	RM	RZ	RY
1	37C00	1000	*	*	*	*
2	37C04	1000	*	*	*	*
3	37C04	1000	1000	*	*	*
4	37C04	1000	1000	B2500	0	*
5	37C04	1000	1000	B2500	0	0
1	37C04	0	1000	B2500	0	0

* Questi valori sono determinati dall'istruzione precedente.

[151] 5.8 Codifica del registro di collegamento nelle istruzioni di rientro da sottoprogramma

D: Le istruzioni di chiamata a sottoprogramma descritte nel Capitolo 2 usano sempre lo stesso registro di uso generale, LINK, per immagazzinare l'indirizzo di rientro. Perciò l'indirizzo del registro di rientro non viene incluso nell'istruzione. Tuttavia l'indirizzo LINK viene incluso nei bit IR_{31-27} delle istruzioni di rientro da sottoprogramma (vedere Paragrafo 5.4.1 ed Esempio 5.4). Perché le due istruzioni vengono gestite differentemente?

R: Il registro LINK deve essere letto nel passo 2 di un'istruzione di rientro da sottoprogramma, prima che l'istruzione sia decodificata. L'indirizzo LINK viene posto nei bit IR_{31-27} poichè il processore legge sempre il registro identificato da questi bit nel passo 2, senza attendere la fine della decodifica dell'istruzione.

[152] 5.9 Realizzazione del bocco Immediato (1)

D: Si assuma che gli operandi immediati occupino i bit IR21-6 dell'istruzione. Il valore immediato viene esteso con segno a 32 bit nelle istruzioni aritmetiche, come Add, ed esteso con zeri in istruzioni logiche, come OR. Progettare una realizzazione adatta al blocco Immediato in Figura 5.9.

R: Si assuma che il modo in cui si estende l'operando immediato sia determinato da un ingresso di controllo detto SE. L'operando è esteso con segno quando SE = 1, altrimenti viene esteso con zeri. L'uscita Imm_{31-0} per il blocco Immediato può essere ottenuta come segue.

Imm0 = IR6, Imm1 = IR7, : : : , Imm15 = IR21
Imm16 = SE IR21, Imm17 = SE IR21, : : : , Imm31 = SE IR21

[153] 5.10 Esclusione dei modi di indirizzamento con autoincremento e autodecremento dallo stile RISC

D: Abbiamo visto come tutte le istruzioni in stile RISC possono essere eseguite usando i passi in Figura 5.4 sull'hardware a più stadi di Figura 5.8. I modi di indirizzamento con autoincremento e autodecremento non sono inclusi in un insieme di istruzioni in stile RISC. Spiegare perché l'istruzione

Load R3, (R5)+

non possa essere eseguita sull'hardware in Figura 5.8.

R: Il modo di indirizzamento con autoincremento significa che il contenuto del registro R5 è Incrementato di 4 dopo essere stato usato per leggere l'operando in memoria. Entrambi, il valore aggiornato del registro R5 e l'operando letto dalla memoria, devono essere scritti nelbanco dei registri. Questo non è possibile nell'organizzazione hardware in Figura 5.8.

[154] 5.11 Realizzazione del bocco Immediato (2)

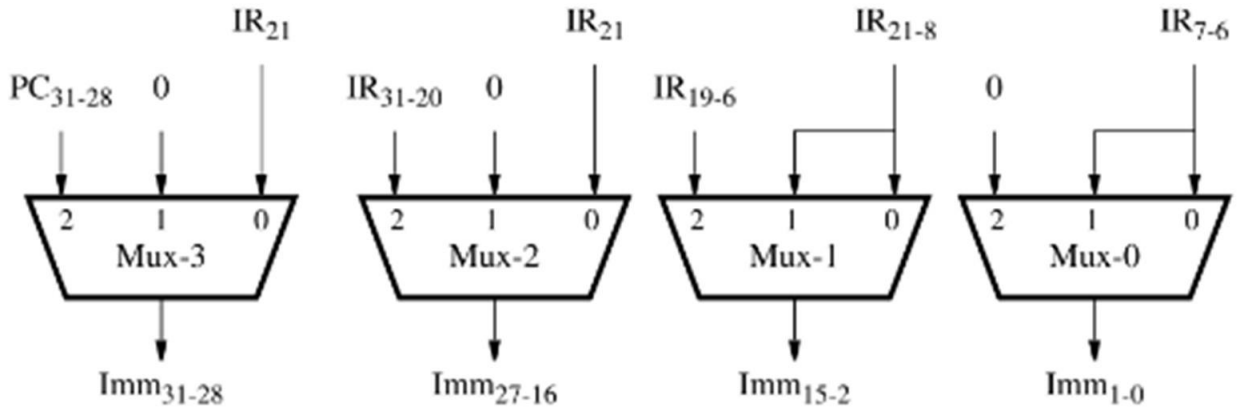
D: Le istruzioni di un calcolatore sono codificate come mostrato in Figura 5.12. Quando un valore immediato è fornito in un'istruzione, esso deve essere esteso a un valore a 32 bit. Si assuma che il valore immediato sia usato in tre modi diversi:

(a) Un valore a 16 bit con segno da usare in operazioni aritmetiche.

(b) Un valore a 16 bit esteso con zeri a sinistra da usare in operazioni logiche.

(c) Un valore a 26 bit esteso con 2 zeri a destra e con i 4 bit di ordine più alto del PC aggiunti a sinistra da usare in istruzioni di chiamata a sottoprogramma.

Mostrare una realizzazione del blocco Immediato nella Figura 5.19 che possa effettuare le estensioni richieste.
R: Valori diversi per l'uscita del blocco Immediato, Imm_{31-0}, vengono selezionati dall'ingresso di controllo Extend in Figura 5.19. Si possono usare dei multiplatori per selezionare i valori di campi di bit differenti come mostrato nella figura sottostante. Le selezioni 0, 1 e 2 corrispondono ai casi a,b,c rispettivamente :



[155] 5.12 Segnale di abilitazione del contatore di passi
D: Perché è necessario includere entrambe WMFC e MFC nell'espressione logica per Contatore_abilita fornita nel Paragrafo 5.6.2?
R: Il segnale WMFC identifica un passo in cui viene lanciato un comando di lettura di memoria o di scrittura in memoria. In questi passi, l'incremento del contatore deve essere differito fino a quando il segnale MFC viene asserito.
[156] 5.13 Segnale di abilitazione alla scrittura del PC
D: Spiegare cosa succederebbe se la variabile MFC venisse omessa dall'espressione per PC_abilita fornita nel Paragrafo 5.6.2.
R: Il PC è abilitato al passo 1, quando si preleva un'istruzione dalla memoria. Se l'istruzione non viene trovata nella cache e deve essere letta dalla memoria principale, questo passo richieder`a parecchi cicli di clock. Senza il segnale MFC, il PC verrebbe erroneamente incrementato a ogni ciclo.
[157] 5.14 Calcolo del tempo di esecuzione di microistruzioni
D: Un processore microprogrammato ha i seguenti parametri. La generazione dell'indirizzo di inizio della microroutine di un'istruzione richiede 2,1 ns e la lettura di una microistruzione dalla memoria di microprogramma richiede 1,5 ns. Lo svolgimento di un'operazione nell'ALU richiede un massimo di 2,2 ns e l'accesso alla memoria cache richiede 1,7 ns. Si assuma che tutte le istruzioni e i dati siano nella cache.
(a) Determinare il tempo minimo necessario per ciascuno dei passi nella Figura 5.26.
(b) Ignorando tutti gli altri ritardi, qual è il ciclo di clock più breve che può essere usato per questo processore?
R:
a. L'indirizzo iniziale di una microroutine viene generato come risultato della decodifica di un'istruzione. I ritardi dovuti ai passi in Figura 5.26 sono riportati nella seguente tabella. Si suppone che l'incremento del PC richieda meno tempo rispetto alla lettura di una parola dalla cache.

Passo	Ritardo componenti	Tempo minimo
1	1,5 + 1,7	3,2
2	1,5 + 2,1	3,6
3	1,5 + 1,7	3,2
4	1,5 + 2,2	3,7
5	1,5 + 1,7	3,2
6	1,5 + 2,2	3,7
7	1,5 + 1,7	3,2

b. Ciclo di clock minimo = 3,7 ns.
[158] 5.15 Prelievo ed esecuzione di un'istruzione Load in un processore CISC
D: Stilare la sequenza di passi necessari per prelevare ed eseguire l'istruzione
Load R3, (R5)+
sul processore di Figura 5.24. Si assuma che gli operandi siano a 32 bit.
R:
Le azioni necessarie dopo la decodifica dell'istruzione sono:
3. Indirizzo di memoria [R5], Lettura dalla memoria, Attesa di MFC, Temp1 ← Dati dalla memoria
4. R5 ← [R5] + 4
5. R3 ← [Temp1]
[159] 5.16 Esecuzione di un'istruzione di chiamata in un processore CISC
D: Si consideri un processore in stile CISC che salva l'indirizzo di rientro da sottoprogramma sulla pila del processore invece che nel registro predefinito LINK. Stilare la sequenza di azioni necessarie per eseguire

un'istruzione Call_Register sul processore di Figura 5.24.
R:
Supporre che l'istruzione
Call-Register R5, R7
chiami una subroutine il cui indirizzo è fornito nel registro R5 e memorizzi l'indirizzo di rientro su una pila puntata da R7. Dopo la lettura e decodifica dell'istruzione, si rendono necessarie le seguenti azioni.
3. Temp1 ← [PC] + 0
4. PC ← [R5] + 0
5. R7 ← [R7] - 4
6. Indirizzo di memoria ← [R7], Dati in memoria ← [Temp1], Scrittura della memoria, Attesa per MFC

Capitolo 6: Introduzione al pipelining

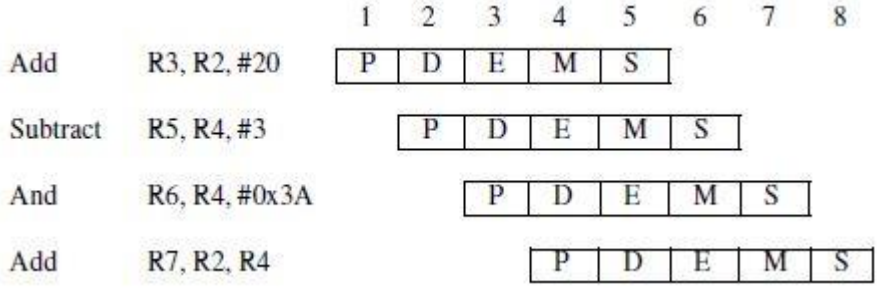
[160] 6.1 Esecuzione in pipeline di una sequenza di istruzioni

D: Considerare le seguenti istruzioni agli indirizzi di memoria indicati:

1000 Add R3, R2, #20
1004 Subtract R5, R4, #3
1008 And R6, R4, #0x3A
1012 Add R7, R2, R4

Inizialmente i registri R2 e R4 contengono 2000 e 50, rispettivamente. Queste istruzioni vengono eseguite in un calcolatore che ha una pipeline a cinque stadi come mostrato in Figura 6.2. La prima istruzione viene prelevata nel ciclo di clock 1, e le rimanenti istruzioni sono prelevate nei cicli successivi.
(a) Disegnare un diagramma simile alla Figura 6.1 che rappresenti il flusso delle istruzioni attraverso la pipeline. Descrivere le operazioni svolte da ciascuno stadio della pipeline durante i cicli di clock da 1 a 8.
(b) Con riferimento alle Figure 5.8 e 5.9, descrivere i contenuti dei registri IR, PC, RA, RB, RY e RZ nella pipeline durante i cicli da 2 a 8.
R:

a. I diagramma di esecuzione in pipeline per il codice fornito viene mostrato sotto.



Si mostra sotto la descrizione delle attivit`a in ogni stadio durante ciascun ciclo. Non si descrivono le attivit`a delle istruzioni precedenti la prima istruzione Add e di quelle dopo la seconda istruzione Add, che comunque hanno luogo come determinato dal tipo di istruzione e dallo stadio della pipeline.

Ciclo	Stadio	Attività
1	P	preleva istruzione Add
2	P	preleva istruzione Subtract
	D	decodifica istruzione Add, legge registro R2 (valore 2000)
3	P	preleva istruzione And
	D	decodifica istruzione Subtract, legge registro R4 (valore 50)
	E	esegue operazione aritmetica 2000+20=2020 per istruzione Add
4	P	preleva istruzione Add
	D	decodifica istruzione And, legge registro R4 (valore 50)
	E	esegue operazione aritmetica 50−3=47 per istruzione Subtract
	M	nessuna operazione per istruzione Add
5	D	decodifica istruzione Add, legge registro R2 (valore 2000) e registro R4 (valore 50)
	E	esegue operazione logica $50 \wedge 3A_{16} = 50$ per istruzione And
	M	nessuna operazione per istruzione Subtract
	S	scrive il risultato 2020 dell'istruzione Add su registro R3
6	E	esegue operazione aritmetica per istruzione Add
	M	nessuna operazione per istruzione And
	S	scrive il risultato 47 dell'istruzione Subtract su registro R5
7	M	nessuna operazione per istruzione Add
	S	scrive il risultato 50 dell'istruzione And su registro R6
8	S	scrive il risultato 2050 dell'istruzione Add su registro R7

b. Il contenuto di ciascun registro durante ciascun ciclo `e descritto nella tabella seguente. Si usa la notazione '−' per indicare che non vi `e informazione sufficiente a determinare il contenuto del registro in quel ciclo. L'istruzione che occupa IR in ogni ciclo `e identificata simbolicamente con un'abbreviazione.

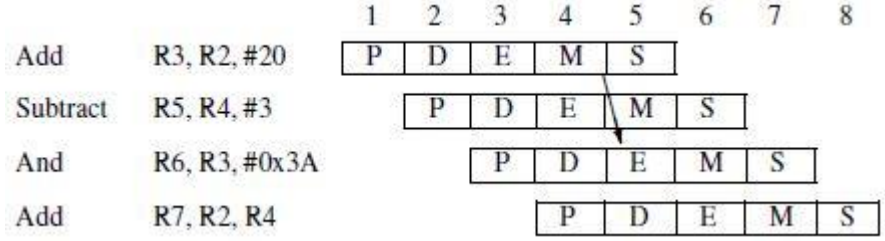
	2	3	4	5	6	7	8
IR	Add	Sub	And	Add	–	–	–
PE	1004	1008	1012	1016	1020	1024	1028
RA	–	2000	50	50	2000	–	–
RB	–	–	–	–	50	–	–
RZ	–	–	2020	47	50	2050	–
RY	–	–	–	2020	47	50	2050

[161] 6.2 Esecuzione in pipeline di una sequenza di istruzioni con inoltro degli operandi

D: Considerare le seguenti istruzioni agli indirizzi di memoria indicati:
1000 Add R3, R2, #20
1004 Subtract R5, R4, #3
1008 And R6, R3, #0x3A
1012 Add R7, R2, R4
Inizialmente i registri R2 e R4 contengono 2000 e 50, rispettivamente. Queste istruzioni vengono eseguite in un calcolatore che ha una pipeline a cinque stadi come mostrato in Figura 6.2. La prima istruzione viene prelevata nel ciclo di clock 1, e le rimanenti istruzioni sono prelevate nei cicli successivi. Si assuma che la pipeline fornisca percorsi per l'inoltro all'ALU dai registri RY e RZ in Figura 5.8 e che il processore usi l'inoltro di operandi.
(a) Disegnare un diagramma simile alla Figura 6.1 che rappresenti il flusso delle istruzioni attraverso la pipeline. Descrivere le operazioni svolte da ciascuno stadio della pipeline durante i cicli di clock da 1 a 8.
(b) Con riferimento alle Figure 5.8 e 5.9, descrivere i contenuti dei registri IR, PC, RA, RB, RY e RZ nella pipeline durante i cicli da 2 a 8.

R:

a. Sotto `e mostrato il diagramma di esecuzione in pipeline per il codice fornito.



La descrizione dell'attivit`a in ogni stadio di ciascun ciclo `e la stessa della parte a del Problema 6.1, eccetto che il precedente contenuto del registro R3 viene letto dall'istruzione And nel ciclo 4, ma lo stadio Elaborazione usa il nuovo valore 2020 calcolato dalla prima istruzione Add. Le differenze con la soluzione della parte a del Problema 6.1 sono evidenziate in corsivo nella tabella parziale che segue.

Ciclo	Stadio	Attività
4	P	preleva istruzione Add
	D	decodifica istruzione And, *legge registro R3 (valore sconosciuto)*
	E	esegue operazione aritmetica 50−3=47 per istruzione Subtract
	M	nessuna operazione per istruzione Add
5	D	decodifica istruzione Add, legge registro R2 (valore 2000) e registro R4 (valore 50)
	E	esegue operazione logica $2020 \wedge 3A_{16} = 32$ per istruzione And
	M	nessuna operazione per istruzione Subtract
	S	scrive il risultato 2020 dell'istruzione Add su registro R3
6	E	esegue operazione aritmetica per istruzione Add
	M	nessuna operazione per istruzione And
	S	scrive il risultato 47 dell'istruzione Subtract su registro R5
7	M	nessuna operazione per istruzione Add
	S	scrive il risultato *32* dell'istruzione And su registro R6

b. Il contenuto di ciascun registro durante ciascun ciclo `e descritto nella tabella seguente. Le differenze con la tabella equivalente nella parte b del Problema 6.1 sono evidenziate in grassetto.

	2	3	4	5	6	7	8
IR	Add	Sub	And	Add	–	–	–
PC	1004	1008	1012	1016	1020	1024	1028
RA	–	2000	50	*prec. R3*	2000	–	–
RB	–	–	–	–	50	–	–
RZ	–	–	2020	47	*32*	2050	–
RY	–	–	–	2020	47	*32*	2050

I contenuti di RZ nel ciclo 4 e RY nel ciclo 5 sono determinati come segue:

$$2020 \wedge 3A_{16} = 7E4_{16} \wedge 3A_{16} = 20_{16} = 32$$

[162] 6.3 Logica di controllo per l'inoltro degli operandi in una pipeline (1)

D: Si richiede della logica di controllo aggiuntiva nella pipeline per inoltrare il valore del registro RZ come mostrato in Figura 6.5. Quali specifiche condizioni deve valutare questa logica aggiuntiva per determinare le impostazioni dei multiplatori che alimentano gli ingressi dell'ALU nello stadio Elaborazione della pipeline?

R: E' necessaria della logica di controllo aggiuntiva per inoltrare il valore di RZ alla ALU. Questa logica seleziona il corretto ingresso di MuxA e MuxB.

Nei buffer interstadi, è necessario avere gli identificatori di registro per gli operandi sorgenti, nonché l'identificatore di registro per la destinazione. Questi identificatori sono introdotti nella pipeline quando l'istruzione viene decodificata e avanzano attraverso gli stadi successivi della pipeline.

E' necessaria della logica per confrontare gli identificatori dei registri. L'identificatore della destinazione nel buffer interstadi che alimenta lo stadio Memoria viene confrontato con ciascuno degli identificatori delle sorgenti nel buffer interstadi che alimenta lo stadio Elaborazione. Se esiste una corrispondenza fra la destinazione e una delle sorgenti, allora vi è una dipendenza potenziale. La sorgente corrispondente determina i segnali di controllo per la multiplazione del valore appropriato al corrispondente ingresso dell'ALU.

Si noti, tuttavia, che il confronto è significativo solo se il segnale di controllo per scrivere il registro di destinazione è impostato nello stadio Memoria. In tal caso il registro di destinazione sarà modificato (nello stadio successivo). Quindi, la corrispondenza dell'identificatore del registro fra la destinazione e una delle sorgenti indica una vera dipendenza ed è necessario l'inoltro, altrimenti non vi è dipendenza.

Vi è anche una considerazione più sottile che coinvolge il registro R0 se è designato quale registro di destinazione in un'istruzione per la quale sia impostato il segnale di scrittura. Per una particolare architettura del repertorio di istruzioni che specifichi che il contenuto del registro R0 sia sempre la costante zero, qualsiasi dipendenza apparente che coinvolga R0 dovrebbe essere ignorata e nessun inoltro di qualsiasi nuovo risultato dovrebbe essere fatto perchè tale risultato non sarà infatti scritto in R0. Pertanto, la logica di controllo deve anche confrontare l'identificatore del registro di destinazione con l'identificatore del registro zero per verificare questa condizione.

[163] 6.4 Logica di controllo per l'inoltro degli operandi in una pipeline (2)

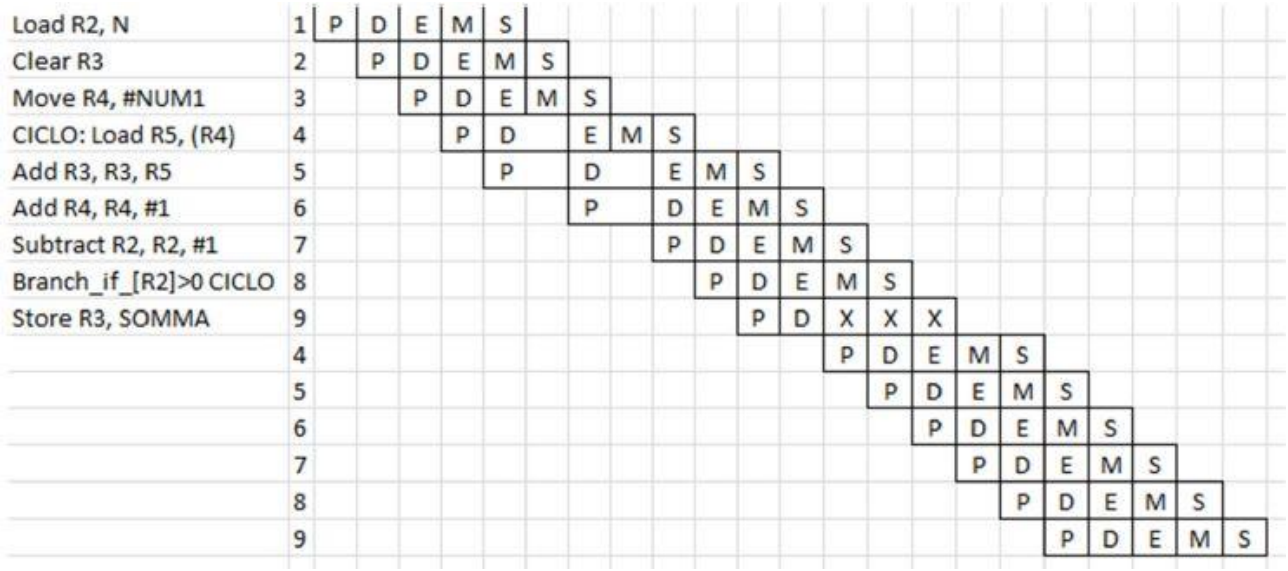
D: Si richiede della logica di controllo aggiuntiva nella pipeline per inoltrare il valore del registro RY in Figura 5.8 ai multiplatori che alimentano gli ingressi dell'ALU nello stadio Elaborazione della pipeline, mostrati in Figura 6.5. Quali specifiche condizioni deve valutare questa logica aggiuntiva per determinare le impostazioni di detti multiplatori?

R: E' necessaria della logica aggiuntiva per trasmettere il valore di RY all'ALU. MuxA e MuxB in Figura 6.5 devono essere ampliati con un ingresso addizionale per il valore di RY. La logica di controllo descritta nella soluzione al Problema 6.10 va estesa per usare MuxA e MuxB al fine di selezionare i corretti ingressi per l'ALU da tutti gli ingressi disponibili. I requisiti per l'invio del valore di RY sono simili a quelli descritti per la soluzione del Problema 6.10, eccetto che si devono considerare l'identificatore del registro di destinazione e il segnale di scrittura per l'istruzione nello stadio Scrittura della pipeline.

[164] 6.5 Esecuzione di un ciclo in pipeline con predizione statica di salto

D: Considerare il ciclo nel programma di Figura 2.8. Assumere che sia eseguito in una pipeline a cinque stadi con percorsi di inoltro all'ALU dai registri RY e RZ in Figura 5.8. Assumere che la pipeline usi la predizione statica di salto con l'assunto che il salto non avvenga. Disegnare un diagramma simile alla Figura 6.1 per l'esecuzione di due iterazioni successive del ciclo.

R:



[165] 6.6 Miglioramento delle prestazioni con il salto differito

D: Si assuma che il 20 percento del conteggio dinamico delle istruzioni eseguite per un programma siano istruzioni di salto. Si usi il salto differito, con un posto del ritardo. Si assuma che non ci siano stalli causati da altri fattori. Dapprima, determinare un'espressione per il tempo di esecuzione espresso in cicli qualora tutti i posti del ritardo siano occupati con istruzioni NOP. Quindi determinare un'altra espressione che rifletta il tempo di esecuzione con il 70 percento di posti di ritardo occupati con istruzioni utili dal compilatore ottimizzante. Da queste espressioni, determinare il contributo del compilatore all'incremento delle prestazioni, espresso come percentuale di accelerazione.

R: I salti costituiscono il 20% del totale di N istruzioni che vengono eseguite. Ogni salto ha un posto di ritardo.

Se tutti i posti di ritardo sono occupati con istruzioni NOP, il numero totale di cicli (trascurando gli ultimi quattro cicli per completare l'esecuzione dell'istruzione finale) è N + 0,2N = 1,2N.

Se è possibile riempire utilmente il 70% dei posti di ritardo con un'altra operazione (lasciando il 30% dei posti di ritardo con istruzioni NOP), allora il numero totale di cicli è N + (0,2 × 0,3)N = 1,06N.

L'accelerazione del secondo caso rispetto al caso iniziale è

$$\left(\frac{1,2}{1,06} - 1\right) \times 100 = 13\%$$

[166] 6.7 Esecuzione superscalare (1)

D: Si consideri un programma che consiste di quattro istruzioni di accesso alla memoria e quattro istruzioni aritmetiche. Si assuma che non ci siano dipendenze di dato tra le istruzioni. Due versioni di questo programma vengono eseguite sul processore superscalare mostrato in Figura 6.13. La prima versione ha le quattro istruzioni di accesso alla memoria in sequenza, seguite dalle quattro istruzioni aritmetiche. La seconda versione ha le istruzioni di accesso alla memoria alternate con le istruzioni aritmetiche. Disegnare due diagrammi simili alla Figura 6.14 per confrontare l'esecuzione di queste due versioni del programma.

R: La sequenza di istruzioni consiste di quattro istruzioni di accesso a memoria e quattro istruzioni aritmetiche.

Con l'ordine originale delle istruzioni, vi sono due casi da considerare per l'invio di coppie di istruzioni nel processore superscalare.

Caso 1: Il processore impone il vincolo che due istruzioni da smistare nello stesso ciclo debbano essere di tipi distinti per ridurre la complessità di ciascuna unità di esecuzione. Nel seguente diagramma di esecuzione in pipeline, M_i denota istruzioni di accesso a memoria e A_j denota istruzioni aritmetiche.

	1	2	3	4	5	6	7	8	9
M_1	P	D	E	M	S				
M_2	P		D	E	M	S			
M_3		P		D	E	M	S		
M_4		P			D	E	M	S	
A_1			P	D	E	S			
A_2			P		D	E	S		
A_3				P		D	E	S	
A_4				P			D	E	S

Caso 2: Il processore permette di smistare due istruzioni dello stesso tipo nello stesso ciclo con buffering più complesso in ciascuna unità di esecuzione. Istruzioni dello stesso tipo smistate simultaneamente sono ancora trattate in pipelining nella loro unità di esecuzione, perciò il completamento di tutte le attività

richiede lo stesso numero di cicli che nel primo caso sopra.

	1	2	3	4	5	6	7	8	9
M_1	P	D	E	M	S				
M_2	P	D		E	M	S			
M_3		P	D		E	M	S		
M_4		P	D			E	M	S	
A_1			P	D	E	S			
A_2			P	D		E	S		
A_3				P	D		E	S	
A_4				P	D			E	S

Ora, invece di preservare l'ordine originale delle istruzioni, è possibile intercalare istruzioni di accesso a memoria e istruzioni aritmetiche in una sequenza di istruzioni diversa. Se si usa tale sequenza, il diagramma di esecuzione in pipeline sotto mostra che occorre un ciclo in meno per completare tutte le attività.

	1	2	3	4	5	6	7	8	9
M_1	P	D	E	M	S				
A_1	P	D	E	S					
M_2		P	D	E	M	S			
A_2		P	D	E	S				
M_3			P	D	E	M	S		
A_3			P	D	E	S			
M_4				P	D	E	M	S	
A_4				P	D	E	S		

[167] 6.8 Esecuzione superscalare (2)

D: Si assuma che un programma non contenga istruzioni di salto. Il programma viene eseguito sul processore superscalare mostrato in Figura 6.13. Qual è il miglior tempo di esecuzione espresso in cicli che si può ottenere se il misto di istruzioni consiste del 75 percento di istruzioni aritmetiche e del 25 percento di istruzioni di accesso alla memoria? Qual è il confronto di questo tempo rispetto al miglior tempo di esecuzione sul processore più semplice in Figura 6.2 con la stessa frequenza di clock?

R: La sequenza di N istruzioni consiste del 75% di istruzioni aritmetiche e 25% di istruzioni di accesso a memoria (Load/Store). Non ci sono istruzioni di salto.

Su un processore con pipeline a cinque stadi il tempo di esecuzione è N cicli (trascurando i quattro cicli finali per le ultime istruzioni).

Su un processore superscalare le istruzioni aritmetiche (la quota maggiore) passano una alla volta attraverso l'unità di esecuzione corrispondente, quindi determinano il tempo di esecuzione come 0; 75N cicli.

L'accelerazione con l'esecuzione superscalare è

$$\left(\frac{1}{0,75} - 1\right) \times 100 = 33\%$$

Se la distribuzione di istruzioni fosse stata del 50% per ciascuna delle categorie aritmetiche e di accesso a memoria, l'accelerazione ideale sarebbe stata un fattore 2, o il 100%. La distribuzione non uniforme fra le due categorie di istruzioni limita l'accelerazione conseguibile.

[168] 6.9 Esecuzione superscalare (3)

D: Si assuma che un programma venga eseguito sul processore superscalare mostrato in Figura 6.13, con un'accuratezza di predizione del 100 percento per tutte le istruzioni di salto. Qual è il miglior tempo di esecuzione espresso in cicli che si può ottenere se il misto di istruzioni consiste per il 15 percento di istruzioni di salto, con salti mai effettuati, per il 65 percento di istruzioni aritmetiche, e per il 20 percento di istruzioni di accesso alla memoria? Qual è il confronto di questo tempo rispetto al miglior tempo di esecuzione sul processore più semplice in Figura 6.2 con la stessa frequenza di clock?

R: Assumendo che *N* (conteggio dinamico delle istruzioni) sia uguale a 4 otteniamo:

Nel primo caso:

$$S = 5*0,25 + 4*0,75 = 4,25$$

$$T = \frac{N*S}{R} = \frac{4*4,25}{R} = \frac{17}{R}$$

Nel secondo caso:

$$S = 5*0,25 + 4*0,25 = 4,75$$

$$T = \frac{N*S}{R} = \frac{4*4,75}{R} = \frac{19}{R}$$

[169] 6.10 Compilazione di cicli con predizione dinamica di salto

D: Si consideri un processore che usi lo schema di predizione di salto rappresentato in Figura 6.12b. L'insieme di istruzioni per il processore viene migliorato con una caratteristica che abilita il compilatore a specificare lo stato iniziale della predizione come PS o PNS per ciascuna istruzione di salto. Questo stato iniziale viene usato dal processore a tempo di esecuzione quando l'informazione sull'istruzione di salto non viene trovata nel buffer di destinazione di salto. Discutere come il compilatore dovrebbe usare questa caratteristica quando genera il codice per i seguenti casi:
(a) Un ciclo con un'istruzione di salto condizionato alla fine per saltare all'inizio del ciclo
(b) Un ciclo con un'istruzione di salto condizionato all'inizio del ciclo per uscire dal ciclo, e un salto incondizionato alla fine del ciclo per saltare all'inizio
R: Per il miglioramento proposto del repertorio di istruzioni, relativo ai salti, si riportano appresso vari suggerimenti di predizione per i casi considerati in questa domanda.
a. Per un ciclo con un salto indietro alla fine (e nessun altro ramo di uscita dal ciclo nel corpo del ciclo), si può supporre che il numero di iterazioni sarà maggiore di uno, quindi si può usare il suggerimento probabilmente salta (PS) per inizializzare la storia delle predizioni.
b. Per un ciclo con un salto condizionato in avanti all'inizio per uscire dal ciclo e un salto incondizionato indietro per la reiterazione, si può usare il suggerimento PS per inizializzare la storia delle predizioni di salto alla fine, proprio come nella parte a. Per il salto in avanti, si può usare il suggerimento opposto probabilmente non salta (PNS) per inizializzare la storia delle predizioni di salto.

[170] 6.11 Compilazione di istruzione condizionale con predizione dinamica di salto

D: Si assuma che un processore usi lo schema di predizione di salto rappresentato in Figura 6.12b, e che l'insieme di istruzioni per il processore abbia una caratteristica che abilita il compilatore a specificare lo stato iniziale della predizione come PS o PNS per ciascuna istruzione di salto. Questo stato iniziale viene usato dal processore a tempo di esecuzione quando l'informazione sull'istruzione di salto non viene trovata nel buffer di destinazione di salto. Si consideri un'espressione della forma
IF A > B THEN A = A + 1 ELSE B = B + 1
(a) Generare il codice in linguaggio assemblativo per l'espressione qui sopra.
(b) In assenza di qualsiasi altra informazione, discutere come il compilatore dovrebbe specificare lo stato iniziale della predizione per istruzioni di salto nel codice assemblativo.
(c) Uno studio del comportamento a tempo di esecuzione del programma contenente l'espressione qui sopra rivela che il valore della variabile A è spesso maggiore del valore della variabile B. Se questa informazione fosse resa disponibile al compilatore, discutere come ne sarebbe influenzata la predizione iniziale per le istruzioni di salto.
R: Si riportano di seguito il programma in linguaggio assemblativo e il suggerimento di predizione per l'istruzione IF-THEN in questa domanda.
a. Un modo conveniente di generare istruzioni in linguaggio assemblativo per un'istruzione IF-THEN è quello di usare l'opposto della condizione di confronto impiegata nell'istruzione del linguaggio di alto livello, per saltare alla sequenza di istruzioni corrispondenti alla clausola ELSE.

```
IF:     Load                 R2, A
        Load                 R3, B
        Branch if [R2][R3]   ELSE
THEN:   Add                  R2, R2, #1
        Store                R2, A
        Branch               CONT
ELSE:   Add                  R3, R3, #1
        Store                R3, B
CONT:   istruzione successiva
```

b. Senza altre informazioni per influenzare la preferenza per il suggerimento di predizione nel salto condizionato, si possono scegliere sia PS che PNS come predizione iniziale per l'istruzione. Per il salto incondizionato, tuttavia, PS dovrebbe essere la previsione iniziale.
c. Se è noto che A > B la maggior parte delle volte, allora si può usare il suggerimento PNS per inizializzare la storia di predizione del salto condizionato in modo che la sequenza di istruzioni che inizia con l'etichetta THEN sia eseguita la prima volta che si preleva l'istruzione di salto. Il salto incondizionato dovrebbe tuttavia usare il suggerimento PS.
suggerimento PS.

[171] 6.12 Compilazione ed esecuzione in pipeline di istruzione condizionale con predizione statica di salto

D: Si consideri un'espressione della forma
IF A > B THEN A = A + 1 ELSE B = B + 1
(a) Si consideri un processore che abbia l'organizzazione in pipeline mostrata in Figura 6.2, con predizione statica di salto e assunto di salto non effettuato. Scrivere il codice in linguaggio assemblativo per l'espressione qui sopra. Disegnare diagrammi simili alla Figura 6.1 per mostrare l'esecuzione in pipeline delle istruzioni per differenti decisioni di salto e determinare i tempi di esecuzione espressi in cicli.
(b) Si assuma adesso che venga usato il salto differito. Scrivere il codice in linguaggio assemblativo per l'espressione qui sopra. Disegnare i diagrammi che mostrano l'esecuzione in pipeline delle istruzioni per differenti decisioni di salto e confrontare i tempi di esecuzione espressi in cicli con i tempi per il caso

precedente.
R: Si discutono nel seguito i due casi per l'analisi di esecuzione descritta in questa domanda.
a. Le istruzioni in linguaggio assemblativo per la soluzione di questo problema sono le stesse della soluzione fornita per il Problema 6.17. In primo luogo, si considera l'esecuzione per la predizione statica con l'assunto di salto non effettuato. Se A > B, allora il salto condizionato non è effettuato, quindi non vi è alcuna penalità di salto. Il salto incondizionato invece causa una penalità di un ciclo. Infine, la dipendenza fra le istruzioni Load e il salto condizionato richiede due cicli di stallo. In totale sono necessari 9 cicli prima che si possa prelevare l'istruzione all'etichetta CONT.
Se A ≤ B, allora il salto condizionato è effettuato e incorre nella penalità di un ciclo. Il salto incondizionato non viene eseguito in questo caso, quindi si preleva un'istruzione in meno rispetto al primo caso sopra. Vi sono ancora i due cicli di stallo dovuti alla dipendenza. Il totale è di 8 cicli prima che si possa prelevare l'istruzione all'etichetta CONT.
b. Ora, si consideri l'esecuzione di salto differito con un posto di ritardo. In questo caso non si può usare la sequenza di istruzioni convenzionale (come mostrata nella soluzione al Problema 6.17): si deve fare qualcosa con il posto di ritardo dopo ogni istruzione di salto. Le istruzioni in linguaggio assemblativo e le dipendenze sono tali che il posto di ritardo dopo il salto condizionato non può essere riempito con un'istruzione precedente, quindi si deve usare una NOP. I due cicli di stallo dovuti alla dipendenza del salto condizionato sono ancora presenti. Per il salto incondizionato, l'istruzione Store precedente può essere ricollocata per riempire quel posto di ritardo.
Per il caso A > B, il totale è 8 cicli prima che si possa prelevare l'istruzione all'etichetta CONT.
Per il caso A ≤ B, la NOP nel posto di ritardo del salto condizionato dà lo stesso contributo al tempo di esecuzione dell'istruzione Add scartata nella parte a. Il salto incondizionato non viene eseguito in questo caso, quindi il totale è ancora di 8 cicli prima che si possa prelevare l'istruzione all'etichetta CONT.

[172] 6.13 Realizzazione combinatoria della predizione dinamica di salto (1)

D: Si progetti una rete combinatoria che realizzi l'automa a 4 stati di predizione dinamica di salto, rappresentato in Figura 6.12b, assieme a un registro a due bit che mantiene la codifica dello stato. La rete ha in ingresso i due bit di uscita del registro (codifica dello stato corrente) e un bit, fornito dallo stadio Decodifica dell'istruzione di salto, che rappresenta l'effettiva decisione di salto. L'uscita della rete è collegata all'ingresso del registro e coincide con la codifica del prossimo stato dell'automa.

R:

[173] 6.14 Realizzazione combinatoria della predizione dinamica di salto (2)

D: Si vuole progettare una rete combinatoria che realizzi l'automa a 4 stati di predizione dinamica di salto, rappresentato in Figura 6.12b, assieme a un registro a due bit che mantiene la codifica dello stato. La rete ha in ingresso i due bit di uscita del registro (codifica dello stato corrente) e un bit, fornito dallo stadio Decodifica dell'istruzione di salto, che rappresenta l'effettiva decisione di salto. L'uscita della rete è collegata all'ingresso del registro e coincide con la codifica del prossimo stato dell'automa. Mediante una scelta opportuna della codifica degli stati, si può realizzare la rete suddetta usando solo un sommatore completo a 1 bit (o anche un circuito più semplice). Mostrare come.
Suggerimento : si codifichi lo stato dell'automa in modo che uno dei due bit rappresenti la predizione di salto nello stato corrente, mentre l'altro coincida, nella codifica del prossimo stato, con l'input fornito dallo stadio Decodifica.

R:

Capitolo 7: Indirizzo di ingresso e uscita

[174] 7.1 Malfunzionamento di dispositivo slave su bus sincroni e asincroni

D: Le Figure 7.4, 7.5 e 7.6 mostrano tre protocolli per il trasferimento dati tra un master e uno slave. Cosa accade in ciascun caso se il dispositivo indirizzato non risponde a causa di un malfunzionamento durante un'operazione Read? Quali problemi causerebbe e quali sono i possibili rimedi?

R: Figura 7.4: Il malfunzionamento non sarebbe riconosciuto. Il master che ha richiesto i dati legge qualunque cosa sia presente sulle linee dei dati e la usa come dato, dando luogo a un errore. Per evitare questo errore, il bus dovrebbe includere un segnale quale la risposta Slave-pronto nei protocolli di Figure 7.5 e 7.6.
Figura 7.5: Il master attende la risposta Slave-pronto per un periodo di tempo prestabilito, quindi pone fine all'operazione. Solitamente questo d`a luogo a un'interruzione, cos`ı che una routine di servizio dell'interruzione possa informare del malfunzionamento l'utente.
Figura 7.6: Come nel caso di Figura 7.5.

[175] 7.2 Temporizzazione su bus sincrono con segnale di risposta dallo slave

D: Nel diagramma di temporizzazione in Figura 7.5, il processore mantiene l'indirizzo sul bus finche non riceve una risposta dal dispositivo. Questo e necessario? Quali aggiunte sono necessarie sul dispositivo se il processore manda un indirizzo per un ciclo soltanto?

R: L'indirizzo è necessario solo per selezionare un registro di un dispositivo di I/O. Una volta che il dispositivo riconosca l'indirizzo di uno dei suoi registri durante il primo ciclo di clock, esso può procedere autonomamente con il resto della transazione sul bus, a condizione che tenga traccia di quale registro sia stato indirizzato. Per esempio, l'interfaccia può includere un flip-flop associato a ciascuno dei suoi registri indirizzabili per individuare il registro che è stato indirizzato.

[176] 7.3 Distanza fra processore e dispositivo di I/O su bus asincroni e sincroni

D: Che effetto ha sul diagramma di temporizzazione in Figura 7.6 la crescita della distanza tra processore e dispositivo di I/O? Come si tiene conto della maggiore distanza nel caso di Figura 7.4?

R: La durata di un'operazione di trasferimento di dati aumenta automaticamente a causa del maggior ritardo di propagazione. Ciascuna delle due parti che scambiano dati usando il protocollo asincrono in Figura 7.6 attende che un segnale arrivi dall'altra, così i dati sono trasferiti correttamente. Nel caso di Figura 7.4, la durata del ciclo di clock deve essere tale da inglobare il ritardo di propagazione più lungo possibile. Una distanza maggiore richiede un ciclo di clock più lungo, altrimenti potrebbero verificarsi errori.

[177] 7.4 Trasferimenti di dati di ingresso in più cicli del bus

D: Si consideri un bus sincrono che operi conformemente al diagramma di sincronizzazione in Figura 7.5. Il bus e il circuito d'interfaccia collegato a esso hanno i seguenti parametri:

Ritardo del pilota del bus	2 ns
Ritardo di propagazione sul bus	da 5 a 10 ns
Ritardo del decodificatore di Indirizzi	6 ns
Tempo di prelievo dei dati richiesti	da 0 a 25 ns
Tempo di impostazione	1,5 ns

(a) Qual e la massima frequenza del clock a cui questo bus può operare?

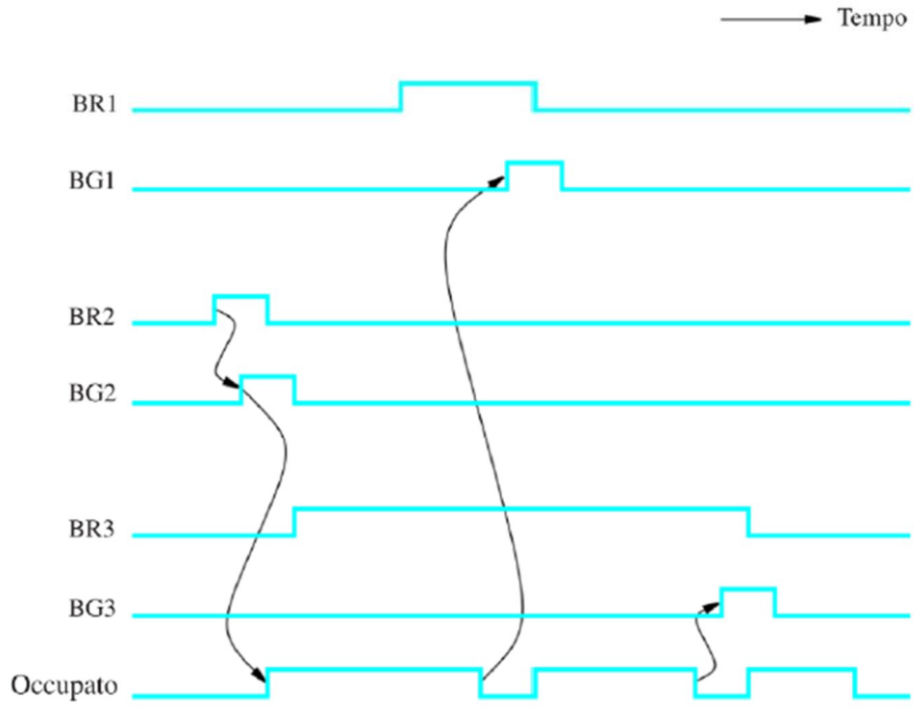
(b) Quanti cicli di clock sono necessari per completare un'operazione di ingresso?

R: Il periodo di clock deve essere tale da accogliere i ritardi del caso peggiore introdotti dai piloti del bus, dalla propagazione sul bus, dalla decodifica dell'indirizzo e dai tempi di impostazione. Si tiene conto della variabilità del tempo di risposta del dispositivo indirizzato usando un numero diverso di cicli di clock.

a. Tempo minimo per l'invio e la decodifica dell'indirizzo = 2 + 10 + 6 = 18 ns.
Tempo minimo di invio dei dati = 2 + 10 + 1,5 = 13,5 ns.
Pertanto, il periodo minimo di un ciclo di clock è 18 ns.

b. Si assuma che si usi il periodo di clock minimo di 18 ns. Il dispositivo indirizzato inizia a prelevare i dati richiesti all'inizio del secondo ciclo di clock. Se i dati richiesti sono disponibili immediatamente, il tempo necessario a inviarli è 13,5 ns, che è inferiore al periodo di clock, quindi il dispositivo può rispondere nel secondo ciclo di clock. Un dispositivo che richieda 25 ns per prelevare i dati li avrà disponibili per l'invio 7 ns dopo l'inizio del terzo ciclo di clock: questo non lascia tempo sufficiente per inviare i dati al processore nello stesso ciclo, quindi risponderà nel quarto ciclo di clock.

La seguente figura mostra un diagramma di temporizzazione per un protocollo che usa un segnale di Occupato.



[178] 7.6 Trasferimento di dati in ingresso su bus sincrono

D: Il protocollo del bus di Figura 7.4 specifica che il dispositivo slave dovrebbe inviare i suoi dati solo nella seconda fase del clock.

(a) E possibile che un dispositivo riconosca il suo indirizzo e sia pronto a mandare dati prima. Perche non gli e permesso farlo? Il processore riceverebbe dati sbagliati?

(b) Ne sorgerebbe qualche altro problema?

R: a. Il processore non carica i dati di ingresso nel proprio registro di ingresso fino al termine del ciclo di clock. Non fa differenza se il dispositivo invia i propri dati in anticipo. I dati saranno comunque disponibili alla fine del ciclo e il processore li riceverà correttamente.
b. L'uscita di un decodificatore di indirizzi può cambiare più volte prima di stabilizzarsi al corretto valore finale. Prima di arrivare a questo valore finale, potrebbe indicare erroneamente l'indirizzamento di un dispositivo. Imporre ai dispositivi di attendere la seconda fase del clock prima di intraprendere qualsiasi azione assicura che solo il dispositivo indirizzato risponderà. In caso contrario un dispositivo potrebbe compiere operazioni errate, come azzerare il suo bit di stato. Inoltre, le uscite di tutti i dispositivi di I/O sono collegate insieme mediante le linee del bus. Se due o più dispositivi abilitano simultaneamente i loro piloti tri-state si potrebbe avere un picco di corrente elettrica, con inutile dissipazione di potenza e possibili danni alle porte.

Capitolo 8: Sistema di memoria

[179] 8.1 Multiplazione e demultiplazione in chip di memoria con indirizzamento di bit singolo

D: Nell'organizzazione di un chip di memoria 1K x 1 in Figura 8.3 compare un blocco che comprende due circuiti: un multiplatore per l'output e un demultiplatore per l'input. In che cosa differiscono le funzioni di questi due circuiti? In che senso il demultiplatore puo essere considerato come una generalizzazione del decodificatore?

R: Il moltiplicatore e usato dalla parte trasmettitrice e il demoltiplicatore e usato dalla parte ricevente, la differenza tra i due circuiti e: il multiplatore ha in ingresso piu segnali(a differenza del demultiplatore che ne ha solo uno) in uscita in base a dei segnali di controllo verra attivata una e una sola porta di output,quindi un solo segnale in uscita. Il demoltiplicatore invece riceve un solo segnale e lo passa a destinazione in base all'ingresso di selezione.
Il demultiplexer ha la funzione esattamente inversa al multiplexer: il multiplexer infatti riunisce piu entrate in un'unica uscita mentre il demultiplexer smista un ingresso in piu uscite.

[180] 8.2 Temporizzazione della lettura a raffica di dati da memoria SDRAM

D: Nel diagramma di temporizzazione in Figura 8.9, relativo a una lettura di dati a raffica in una SDRAM, la lettura dell'indirizzo di colonna avviene con un ritardo di 2 cicli di clock dopo quella dell'indirizzo di riga (e realisticamente tale ritardo e spesso maggiore, tipicamente ammonta a 5-6 cicli), mentre il trasferimento dei dati ha inizio con un solo ciclo di ritardo dopo la lettura dell'indirizzo di colonna. Come si spiega tale differenza?

R: Tale differenza si spiega dal fatto che per trovare la cella interessato in una matrice di una certa grandezza, il decodificatore di indirizzi deve prima individuare la riga(impiegando un certo lasso di tempo) e poi, per selezionare la cella corretta, deve individuare anche la colonna(impiegando un altro lasso di tempo). Dopo tale operazione, per disporre il trasferimento dei dati, è necessario che la cella selezionata venga caricata in lettura e ciò ovviamente impiega meno tempo della precedente operazione.

[181] 8.3 Tempo totale di trasferimento di dati a raffiche e latenza su memoria SDRAM

D: Si consideri una memoria principale costruita con chip SDRAM. I dati sono trasferiti a raffiche, come mostrato in Figura 8.9, eccetto che la lunghezza della raffica e 8. Si assuma che 32 bit di dati siano trasferiti in parallelo. Se si usa un clock a 400 MHz, quanto tempo e necessario per trasferire:
(a) 32 byte di dati
(b) 64 byte di dati
Qual e la latenza in ciascun caso?

R: Ogni campionamento di indirizzo di colonna fa trasferire 8 _ 4 = 32 byte.
a. Latenza = 5 cicli di clock ovvero 12,5 ns.
Tempo totale = 5 + 8 = 13 cicli di clock, o 32,5 ns.
b. Occorre un secondo campionamento di indirizzo di colonna per trasferire la seconda raffica di 32 byte. Pertanto:
Latenza = 5 cicli di clock ovvero 12,5 ns.
Tempo totale = 5 + 8 + 2 + 8 = 23 cicli di clock, o 57,5ns.

[182] 8.4 Organizzazione di un modulo di memoria statica

D: Descrivere una struttura simile a quella in Figura 8.10 per una memoria 16M x 32 usando chip di memoria 1M x 4.

R: Un modulo di 16M può essere organizzato in 16 righe, ciascuna contenente otto chip 1M ×4. Si necessita di un indirizzo da 24 bit. Le linee di indirizzo A_{19-0} vanno collegate a tutti i chip. Le linee di indirizzo A_{23-20} vanno collegate a un decodificatore a 4 bit per selezionare una fra le 16 righe.

[183] 8.5 Fattori influenti sulle prestazioni di un calcolatore

D: Esprimere una valutazione critica della seguente affermazione: "L'uso di un processore piu veloce fornisce un corrispondente incremento delle prestazioni di un calcolatore anche se la velocita della memoria principale rimane la stessa."

R: Un processore più veloce darà una prestazione maggiore. Tuttavia, l'entità dell'aumento non sarà direttamente
proporzionale all'aumento di velocità del processore, perchè la penalità di cache miss rimarrà la stessa se la velocità della memoria principale non migliora.

[184] 8.6 Raffinamento della gerarchia di memoria per processori RISC

D: Nell'organizzazione in pipeline di un processore RISC sono presenti registri interstadi oltre ai registri di uso generale (questi ultimi in un banco di registri). Se si volesse raffinare la gerarchia della memoria, illustrata in Figura 8.14, distinguendo i registri di uso generale dai registri interstadi, quale sarebbe la loro posizione relativa? Quali dei parametri indicati nella figura (dimensione, velocita, costo per bit) potrebbero motivare la risposta?
R: La loro posizione dovrebbe essere subito dopo i registri di uso generale, poichè i registri interstadi, hanno una dimensione maggiore e la loro velocità può variare a seconda dell'operazione che possiede

[185] 8.7 Decomposizione dell'indirizzo per una cache a corrispondenza associativa a gruppi

D: Una cache a corrispondenza associativa a gruppi consiste di un totale di 64 blocchi, divisi in gruppi di 4 blocchi. La memoria principale contiene 4096 blocchi, ciascuno dei quali consta di 32 parole. Assumendo uno spazio degli indirizzi a 32 bit indirizzabile a byte, quanti bit ci sono in ciascuno dei campi Tag, Set (gruppo) e Word (parola)?
R: Ogni blocco contiene 128 byte, quindi si ha un campo Spiazzamento di 7 bit. Vi sono 16 gruppi, occorre quindi un campo Gruppo di 4 bit. I rimanenti 21 bit dell'indirizzo formano il campo Etichetta.

[186] 8.8 Contenuti di una cache a corrispondenza diretta

D: Un calcolatore indirizzabile a byte ha una piccola cache di dati capace di tenere otto parole da 32 bit. Ciascun blocco della cache consiste di una parola da 32 bit. Quando un certo programma viene eseguito, il processore legge i dati sequenzialmente dai seguenti indirizzi esadecimali:
200, 204, 208, 20C, 2F4, 2F0, 200, 204, 218, 21C, 24C, 2F4
Questa sequenza di letture e ripetuta in quattro iterazioni di un ciclo. Si assuma che la cache sia inizialmente vuota. Mostrare i contenuti della cache alla fine di ciascuna iterazione del ciclo, e calcolare il tasso di successo, nell'ipotesi di cache a corrispondenza diretta.
R:

Contenuto della cache dati dopo:

Posizione	Passo 1	Passo 2	Passo 3	Passo 4
0	[200]	[200]	[200]	[200]
1	[204]	[204]	[204]	[204]
2	[208]	[208]	[208]	[208]
3	[24C]	[24C]	[24C]	[24C]
4	[2F0]	[2F0]	[2F0]	[2F0]
5	[2F4]	[2F4]	[2F4]	[2F4]
6	[218]	[218]	[218]	[218]
7	[21C]	[21C]	[21C]	[21C]

Tasso di hit = 33/48 = 0,69

[187] 8.9 Contenuti di una cache a corrispondenza associativa con sostituzione LRU

D: Un calcolatore indirizzabile a byte ha una piccola cache di dati capace di tenere otto parole da 32 bit. Ciascun blocco della cache consiste di una parola da 32 bit. Quando un certo programma viene eseguito, il processore legge i dati sequenzialmente dai seguenti indirizzi esadecimali:
200, 204, 208, 20C, 2F4, 2F0, 200, 204, 218, 21C, 24C, 2F4
Questa sequenza di letture e ripetuta in quattro iterazioni di un ciclo. Si assuma che la cache sia inizialmente vuota. Mostrare i contenuti della cache alla fine di ciascuna iterazione del ciclo, e calcolare il tasso di successo, nell'ipotesi di cache a corrispondenza associativa con algoritmo di sostituzione LRU.
R:

Contenuto della cache dati dopo:

Posizione	Passo 1	Passo 2	Passo 3	Passo 4
0	[200]	[200]	[200]	[200]
1	[204]	[204]	[204]	[204]
2	[24C]	[21C]	[218]	[2F0]
3	[20C]	[24C]	[21C]	[218]
4	[2F4]	[2F4]	[2F4]	[2F4]
5	[2F0]	[20C]	[24C]	[21C]
6	[218]	[2F0]	[20C]	[24C]
7	[21C]	[218]	[2F0]	[20C]

Tasso di hit = 21/48 = 0,44

[188] 8.10 Contenuti di una cache a corrispondenza associativa a gruppi con sostituzione LRU

D: Un calcolatore indirizzabile a byte ha una piccola cache di dati capace di tenere otto parole da 32 bit. Ciascun blocco della cache consiste di una parola da 32 bit. Quando un certo programma viene eseguito, il processore legge i dati sequenzialmente dai seguenti indirizzi esadecimali:
200, 204, 208, 20C, 2F4, 2F0, 200, 204, 218, 21C, 24C, 2F4
Questa sequenza di letture e ripetuta in quattro iterazioni di un ciclo. Si assuma che la cache sia inizialmente vuota. Mostrare i contenuti della cache alla fine di ciascuna iterazione del ciclo, e calcolare il tasso di successo, nell'ipotesi di cache a corrispondenza associativa a gruppi di 4 blocchi, con algoritmo di sostituzione LRU.

R:

Contenuto della cache dati dopo:

	Posizione	Passo 1	Passo 2	Passo 3	Passo 4
Gruppo 0	0	[200]	[200]	[200]	[200]
	1	[208]	[208]	[208]	[208]
	2	[2F0]	[2F0]	[2F0]	[2F0]
	3	[218]	[218]	[218]	[218]
Gruppo 1	4	[204]	[204]	[204]	[204]
	5	[24C]	[21C]	[24C]	[21C]
	6	[2F4]	[2F4]	[2F4]	[2F4]
	7	[21C]	[24C]	[21C]	[24C]

Tasso di hit = 30/48 = 0,63

[189] 8.11 Dimensione del blocco di cache

D: La dimensione del blocco di cache in molti calcolatori va da 32 a 128 byte. A parita di dimensione dalla cache, quali sarebbero i principali vantaggi e svantaggi nello scegliere una dimensione dei blocchi di cache maggiore o minore?

R: Dimensione maggiore:

- meno miss se la maggior parte dei dati nel blocco sono effettivamente usati,
- spreco se molti dati non sono usati prima che il blocco venga rimosso dalla cache,
- penalità di miss più grande.

Dimensione minore:

- più miss,
- penalità di miss più piccola.

[190] 8.12 Analogia per il concetto di cache

D: Considerare la seguente analogia per il concetto di cache. Un tecnico va in una casa per riparare il sistema di riscaldamento. Egli porta una cassetta che contiene un certo numero di utensili che ha usato recentemente in lavori simili. Egli usa questi utensili ripetutamente, finche non raggiunge un punto in cuisono necessari altri utensili. E probabile che abbia gli utensili richiesti nel suo automezzo parcheggiato davanti alla casa. Tuttavia, se gli utensili necessari non sono nell'automezzo, deve andare a prenderli nella sua officina. Supporre che la cassetta, l'automezzo e l'officina corrispondano alla cache L1, alla cache L2 e alla memoria principale di un calcolatore. Quanto e accurata questa analogia? Discuterne le caratteristiche corrette e quelle non corrette.

R: L'analogia è buona per quanto riguarda:

- dimensioni relative di cassetta degli utensili, automezzo e officina rispetto a cache L1, cache L2 e memoria principale,
- tempi di accesso relativi,
- frequenza d'uso relativa di utensili nei tre posti rispetto al tasso di accessi a dati nelle cache e nella memoria principale.

L'analogia fallisce sotto i seguenti aspetti:

- All'inizio di una giornata di lavoro gli utensili posti nell'automezzo e nella cassetta sono preselezionati in base all'esperienza acquisita in precedenti lavori. Tuttavia, quando il sistema operativo seleziona un nuovo programma per l'esecuzione, nessun dato rilevante viene caricato nelle cache prima che inizi l'esecuzione.
- La maggior parte degli utensili nella cassetta e nell'automezzo sono utili in lavori successivi, mentre i dati lasciati nella cache da un programma non sono utili per i programmi successivi.
- Utensili rimossi a causa della necessità di usare altri utensili non sono mai buttati via. Nel caso di una cache, un nuovo blocco sovrascrive un blocco esistente. Il contenuto di un blocco da sostituire è scritto in memoria prima che il blocco venga sostituito nella cache solo se il blocco `e etichettato come modificato.

[191] 8.13 Efficacia di cache a due livelli

D: Lo scopo dell'uso di una cache L2 e di ridurre la penalita di miss della cache L1, e quindi di ridurre il tempo di accesso alla memoria visto dal processore. Un'alternativa e di incrementare la dimensione della cache L1 per incrementare il suo tasso di successo. Cosa limita l'utilita di questo approccio?

R: All'aumentare della dimensione della cache, accadono due cose. Il tasso di hit aumenta, il che significa che meno accessi alla memoria subiranno la penalità di miss. Al contempo, una memoria più grande ha un tempo di accesso più lungo. Ciò significa che tutti gli accessi alla cache saranno serviti in più tempo. Quindi, al crescere della dimensione della cache, l'effettivo tempo di accesso alla memoria visto dal processore può inizialmente migliorare. Tuttavia, a un certo punto il secondo fattore diventa dominante e un ulteriore aumento delle dimensioni della cache diventa controproducente per le prestazioni.

[192] 8.14 Caratteristiche di una unità a disco

D: Un'unita a disco ha 24 superfici di registrazione, un totale di 14.000 cilindri e una media di 400 settori per traccia. Ciascun settore contiene 512 byte di dati.

(a) Qual e il massimo numero di byte che possono essere immagazzinati in questa unita?

(b) Qual e il tasso di trasferimento dati in byte per secondo alla velocita di rotazione di 7200 rpm?

(c) Usando una parola di 32 bit, suggerire uno schema appropriato per specificare l'indirizzo su disco.

R: a. Il numero massimo di byte che si possono immagazzinare sul disco è $24 \times 14000 \times 400 \times 512 = 68{,}8 \times 10^9$ byte.

b. Il tasso di trasferimento dati è $(400 \times 512 \times 7200)/60 = 24{,}58 \times 106$ byte/s.

c. Sono necessari 9 bit per identificare un settore, 14 bit per una traccia e 5 bit per una superficie. Cos`ı, usando una parola a 32 bit $b_{31} \ldots b_0$, un possibile schema `e di usare i bit $b_{8\text{-}0}$ per il settore, $b_{22\text{-}9}$ per la traccia e $b_{27\text{-}23}$ per la superficie. Non si userebbero i bit $b_{31\text{-}28}$.

[193] 8.15 Concorrenza di più unità a disco

D: Il tempo medio di posizionamento e la latenza di rotazione in un sistema a disco sono 6 ms e 3 ms, rispettivamente. Il tasso di trasferimento dati da o verso il disco e 30 MB/s, e tutti gli accessi al disco sono per 8 KB di dati, immagazzinati in settori contigui. I blocchi di dati sono immagazzinati in locazioni casuali sul disco. Il controllore del disco ha un buffer di 8 KB. Il controllore del disco, il processore e la memoria principale sono tutti collegati a un singolo bus. La larghezza del bus dati e di 32 bit e un singolo trasferimento
sul bus da o verso la memoria principale richiede 10 nanosecondi.

(a) Qual e il massimo numero di unita a disco che possono simultaneamente trasferire dati da e verso la memoria principale?

(b) Quale percentuale di accessi alla memoria principale sono usati da una unita a disco, in media, su un lungo periodo di tempo durante il quale viene effettuata una sequenza di trasferimenti indipendenti di 8 KB?

R:

a. I dati sono trasferiti dal buffer del disco alla memoria al massimo tasso di trasferimento dati del bus. Quindi, solo un disco può trasferire dati a ogni dato istante. Dopo il trasferimento dei contenuti del buffer si ha una pausa nel trasferimento di dati mentre si legge il blocco successivo dal disco e lo si memorizza nel buffer. Si sostiene il massimo tasso di trasferimento dei dati quando i blocchi sono immagazzinati in settori contigui sulla stessa traccia. In questo caso il tasso di trasferimento da un qualsiasi disco `e di 30 Mbyte al secondo. Ciò presuppone che l'hardware sia organizzato in modo che possa iniziare a leggere un blocco dal disco mentre il blocco precedente viene inviato alla memoria. Il massimo tasso di trasferimento in memoria è $4/(10 \times 10^{-9}) = 400$ Mbyte al secondo. Pertanto, fino a 13 dischi possono contemporaneamente trasferire dati da/verso la memoria principale.

b. Il tempo medio necessario per trasferire un blocco `e 6+3+8/30 = 9,27 ms, durante il quale vengono trasferiti 8K byte. Questo corrisponde a un tasso di trasferimento di 863K byte/s. Perciò su un lungo periodo di tempo un singolo disco usa $863 \times 10^3 = 400 \times 10^6 = 0{,}22\%$ dei cicli di memoria disponibili.

[194] 8.16 Rappresentazione digitale dell'informazione sui CD

D: Su quale principio è basata la rappresentazione digitale dell'informazione sui CD? A tal riguardo, in che cosa differiscono CD-ROM, CD-R, CD-RW e DVD?

R: I bit sono impressi sotto forma di pit e land; i pit sono le zone stampate nel substrato di plastica del disco mentre i land sono le zone non stampate. Il substrato e posto vicino alla superficie superiore del disco ed e metallizzato al fine di riflettere il raggio laser.

Durante la lettura ogni transizione pit-land e land-pit viene interpretata come un bit 1, mentre le aree piane, che si trovano prima e dopo ogni transizione, sono qualificate come uno o piu bit 0 consecutivi.

- CD-ROM: lettura (i CD-ROM sono supporti di sola lettura);
- CD-R: lettura e scrittura;
- CD-RW: lettura, scrittura, cancellazione e riscrittura;
- DVD : a differenza dei CD, i DVD hanno i settori a lunghezza fissa da 2048 byte, cambia anche la densita di informazione, la tecnologia usata per la realizzazione e la capacita.