Domande ADC

CODIFICA BINARIA

• **Quali tipi di codifica binaria dei numeri relativi esistono?**

MODULO E SEGNO: (-1)^s \* V con V=modulo e s=segno(0: + / 1: -)

Non canonica perchè: (1) non è chiaro dove posizionare il bit di segno (dx o sx?)

(2) si richiede un passo aggiuntivo per determinare il segno s del modulo v

(3) 2 rappresentazioni diverse per lo zero (zero negativo e positivo)

COMPLEMENTO A 1: operazione di inversione bit a bit (ad es. 89 (001011001) con

bit s = 0 (+) diventa 110100110. Per capire che è -89 inverto (s = 1) e ottengo 89.

COMPLEMENTO A 2: C1 + costante 1 aggiunta → posso semplificare la somma tra

numeri relativi poichè invece che una sottrazione se segni discordi, effettuo una

semplice somma x + (-y). Esiste anche un metodo di calcolo veloce (da dx scrivo i bit

uguali fino al primo 1 compreso, poi inverto i successivi).

ECCESSO 2^m: con m = #cifre - 1 (bit s). Se ad es. [s][][][][] → m=4 → 2^4 (k = 16)

→ aggiungendo k ottengo un range di [-16, 15]: il min ad es. è -16 (che sommato a

16 mi da zero → ho “eliminato i negativi”). Il bit di segno risulta invertito

• **Cos'è l'overflow?**

Un calcolo causante di un eccedenza nella memoria

• **Quali tipi di rappresentazione binaria dei numeri esistono (virgola)?**

FIXED POINT: è solo una convenzione (nulla di pratico); interpreto semplicemente x

bit come frazioni in base 2 (ad es. k=½ ; k=¼ ; ..). Valori non potenze di due vengono

approssimati.Con FLOATING POINT la k diventa una variabile: V = M \* 2^e con M

mantissa ed exp in codifica eccesso 2^m → [s][ exp ][ mantissa ]

• **Definisci il concetto di canonicità e normalizzazione.**

Se voglio ad es. codificare “1” con V=M\*2^e posso usare diversi modi (M=1 - e=0;

M=½ - e = 1; ..) → scelgo una sola interpretazione: la 1a cifra della mantissa deve

assumere valore 1 (NB: n molto piccoli possono essere rappresentati solo in forma

normalizzata; altri (come lo 0) non lo possono essere).

• **Cos'è lo standard IEEE-754 e perchè è stato creato? Descrivilo.**

Uno standard di codifica usato a livello ancora più alto del MIPS che formalizza l’uso

dei Floating Point con singola/doppia precisione. Esso rende implicito il 1o bit dei n.

normalizzati → (-1)^s \* (1+M) \* 2^(exp-polarizz.).

1. INTERVALLO: -126 <= exp <= 127: num normalizzati
2. exp = -127: non normalizzati (molto piccoli)
3. exp = 128: range valori speciali (+- inf, NaN)

[1][1..1][0..0]: - inf / [0][1..1][0..0]: + inf / [y][1..1][x..x]: NaN

• **Definisci il concetto di ridondanza.**

Grado di efficienza spaziale di una rappresentazione. Si definisce il coefficente di

ridondanza R come (#combinazioniPossibili) / (#combinazioniUsate): se 1<R<2 è

minimale; se R>2 RIDONDANTE (posso dimunuire i bit senza perdita di info.)

• **In che modo è possibile ottimizzare le risorse utilizzate nella codifica? (Base del sistema di compressione).**

Come prima ottimizzazione uso solo i valori effettivamente usati; poi codifico al livello

0 i valori + frequenti; e ad ogni livello n espando a n+1 verso i meno frequenti

**• Elenca e riassumi i tre tipi di codifica di lunghezza di un codice (numero).**

FISSA: semplice ma spesso spreco + limite max / VARIABILE: nessun limite ma di

difficile gestione / A ESPANSIONE: bilanciamento dell’efficienza.

LOGICA CIRCUITALE

• **Quali sono i tre connettivi logici principali?**

AND, OR, NOT (logica a 3 livelli)

• **Quali tipi di rappresentazione delle funzioni logiche esistono?**

GRAFICA - ALGEBRICA - TAVOLE DI VERITÀ - MAPPA DI KARNAUGH - BDD

• **Cos'è la distanza di Hamming e in che modo viene utilizzata nelle MDK?**

Il numero di volte in cui una cifra n1 è diversa dalla cifra n2 nella stessa posizione.

Nelle mappe di Karnaugh cambiamo l’ordine di rappresentazione dei bit in modo tale

da ottenere una distanza di Hamming sempre pari a 1 per poter poi semplificare.

• **Commutatore - Multiplexer (MPX): descrivine lo scopo.**

Dispositivo che permette tramite bit di controllo (come un “tasto”) di deviare in uscita

un’entrata x a seconda che il bit C sia premuto oppure no. (es: premo→ a ; sennò b)

• **Commutatore - Multiplexer (MPX): come si realizza?**

Dati i ingressi; log2(i) c controllo; 1 uscita u → si realizza mettendo in AND gli in.

con i c, e successivamente tutto in or per ottenere l’uscita. Si introduce la

rappresentazione grafica compatta per estendere mantenendo chiarezza.

• **Demultiplexer (DPX): descrivine lo scopo.**

Dispositivo inverso all’MPX che, data una conf. di controllo, in un caso porta in uscita

l’unico ingresso, negli altri no. Rappresenta l’inverso poichè se collego l’uscita di uno

all’altro riportando la stessa conf. di controllo, ottengo gli stessi risultati.

• **Demultiplexer (DPX): come si realizza?**

Con una serie di AND (2^n con n = var di controllo) tra l’ingresso e la conf di controllo

• **Decoder (DEC): descrivine lo scopo.**

Dispositivo riconducibile al DPX ma con Input costante 1: rappresenta dunque una

tavola di verità → 2^#c = #righe tav. di verità (con #c = var di controllo)

• **Decoder (DEC): come si realizza?**

Con una serie di AND (2^n con n = var di controllo) tre le conf di controllo

• **Sommatore (Full Adder), descrivilo.**

Invece che un Half Adder, molto complesso da modularizzare ed estendere

mantenendo una rappresentazione minimale; si utilizza il Full Adder, un circuito

completo ottenuto collegando con fili di riporoto (carry) diversi moduli adder.

• **Come si può evitare il ritardo tipico del Full Adder? Cos'è la logica a 3 livelli?**

Si possono introdurre ulteriori circuiti in logica a 3 livelli, i carry-lookahead, che

prevedano il riporto dei moduli precedenti, ottenendo un buon compromesso. Devo

anche tenere conto del t di adattamento alle variazioni. In logica a 3 livelli è uguale a

(#or \* tOr)+(#not \* tNot) + (#and \* tAnd). In full adder quindi #riporti \* tTot (peggiore in

logica a 3 livelli: 3\*t (incontro or, not, and)). Posso anche realizzare in logica NAND.

• **Circuito di cambio segno, descrivilo per le varie rappresentazioni binarie.**

Con modulo e segno inverto con un not il bit di segno; con complemento a 1 inverto

tutti i bit con n not; con C2 posso usare metodi diversi: (1) C1 con somma successiva

(2) utilizzo di diversi moduli adder per ottimizzare (3) usare il “trucco” dello swap

visto manualmente, procedendo all’opposto (mando in uscita il risultato (se il bit bool

“invertito” = 1 lo inverto)) + il bit invertito (init a 0, diventa 1 quando incontro un 1).

Posso realizzare con XOR o con !a\*b + !b\*a in logica a 3 livelli.

• **Dispositivo Comparatore, descrivilo.**

In caso di uguaglianza, potrei semplicemente mandare in uscita 1 se a=b con una

realizzazione in logica a 3 livelli, ottenendo però una crescita exp del FAN-IN.

Posso optare per una realizzazione modulare con crescita lineare del numero di

xnor da utilizzare all’aumentare dei bit (ma anche del fan-in dell’and). In caso di “>”

(modularizzando) faccio confronti cifra x cifra (appena so che una cifra è > di quella

corrispondente: fine) → ho un bit e (trovato) e un bit f (risultato).

• **Dispositivo Moltiplicatore, descrivilo.**

Viene realizzato in modo da “simulare” una classica moltiplicazione in colonna (serie

di and per lo shift e successiva somma (half adder per le prime cifre; poi non basta

per via dei riporti che si accumulano → ci vogliono + fili).

• **Che cos'è l'Unità Aritmetico Logica (ALU)?**

L’ALU è un dispositivo generale controllato da dei bit di controllo (con un MPX) che

permette di eseguire diverse funzioni (aritm-logiche). In uscita ho sempre x bit, ho poi

uscite “extra” (overflow, carry, ..)

• **Cos'è un dispositivo a tre stati?**

Mi permette di selezionare un’operazione x rispetto ad una y “staccando” una conf.

di ingresso rispetto all’altra, permettendo un notevole risparmio di utilizzo di fili. Un

esempio chiarificante è l’utilizzo dei 3-state nelle op. R/W sulla RAM.

• **Come si riconosce un circuito sequenziale?**

Perchè il filo di uscita è collegato all’ingresso (“ciclo”). In alcune conf potrei non saper

definire quale bit verrebbe prodotto in uscita (dipendenza reciproca).

• **Descrivi il Flip-Flop Set-Reset.**

è un tipo di circuito sequenziale che permette la memorizzazione di una conf preced;

In riferimento alla tav. di ver. otteniamo alla conf. 0-0 il riporto della conf. prec (Q-!Q);

sulle conf. discordi otteniamo una commutazione; la conf 1-1 invece viene ignorata

poichè genera una problematica: il passaggio da 11 a 00 genera una oscillazione tra

1 e 0; perchè con entrambi i bit a 0 l’uscita e l’ingresso dipendono l’una dall’altra, e

avendo come conf. precedente 00 si genera una instabilità

• **Descrivi il Flip-Flop di tipo D (data).**

Il FF-D ha una composizione simile al S-R con in aggiunta due and + not (per il bit

strobe) → si evita l’1-1 in entrata al S-R. Possiamo definire intuitivamente lo strobe

con una analogia fotografica: strobe è il tappo della macchina (quando str=0 si ripete

il valore di D in uscita; quando str=1 viene memorizzato l’attuale valore di D (“foto”).

Il problema è la lentezza del processo, che obbliga D a non variare per il tempo in cui

strobe passa da 0 a 1 e poi viceversa (“fa la foto”).

• **Descrivi la configurazione Master-Slave.**

Accodiamo due FF di tipo D, uno in qualità di master e uno di slave: il primo avrà il

controllo e priorità sul secondo, che avrà il compito di rendere trasparente in uscita

il calcolo di un nuovo valore da parte del master (mantenendo il prec.). Il FF MS è di

tipo Edge-Triggered (dipende dalla variazione del clock); o in fronte di salita o di

discesa. Se Ck=0: M insensibile e dà in uscita il prec. (ripetuto dallo slave). Se Ck=1:

M è libero di seguire le variazioni di input senza perdere il prec. → a Ck=0 (in caso di

fronte di discesa) “si scatta la foto” e nel master rimane il valore x.

• **Descrivi la configurazione Master-Slave tipo T e tipo J/K.**

Tipo T: se T=0 “simil” tipo D (master tiene il valore e slave riproduce)

se T=1 “simil” conf 1-1 dell’SR → clock (ad ogni impulso varia commutando il val.)

Tipo JK: il più generico tra tutti. Se segni concordi si comporta come T (0 o 1).

Se segni discordi si comporta come tipo D (il master prende il valore di J)

• **Cosa si intende col termine "registri"?**

Un insieme di flip-flop utilizzati come moduli per memorizzare una sequenza di bit.

• **Definisci il registro di tipo D (data).**

Un reg. di tipo D è una sequenza di Flip Flop M/S per memorizzare una sequenza di

bit, con comportamento edge-triggered su di un clock (mantiene un val. fino al prox

fronte di salita/discesa).

• **Definisci il registro di tipo T (toggle, contatore, commutatore).**

Tramite una serie di FF di tipo T (e degli and alle cifre più significative) si realizza un

contatore (crescente, decrescente, in modulo, ...). Può essere utile aggiungere un bi

di reset comandato da multiplexer (per poter scegliere che op. eseguire)

• **Definisci il registro a scorrimento.**

Permette di eseguire shift (a sx (molt.); a dx (div.); circolari). Utile per trasmettere in

modo sincrono utilizzando un singolo mezzo trasmissivo.

• **Riassumi come è realizzata una RAM statica.**

Composta da una sequenza di registri indicizzati di tipo D, in cui ogni indirizzo

corrisponde ad una cella di memoria; dai fili per la trasmissione dei dati e per gli

indirizzi; dal clock e dai fili di controllo (CS (se =1: operaz. in corso) / R/!W).

Grazie al 3-State le operazioni sono mutuamente esclusive (per risparmiare sui fili).

ARCHITETTURA PDP-11

• **Cos'è la Macchina di Von Neumann (MVN)? Da cosa è composta?**

Possiamo definire la MVN come un tipo di architettura HW, comprendente una CPU,

una RAM (master-slave), dei dispositivi di I/O, e dei fili che permettano la comunicaz.

• **In quante e quali fasi sono gestiti i programmi (e le loro istruzioni) da eseguire?**

FETCH: IR← RAM [ PC ]; PC ← PC + 1: metto in ir l’addr della prox istr. da decodif.

DECODE: decodifica l’istruzione attualmente presente in RAM [ IR ]

EXEC: esegue l’istruzione decodificata al ciclo precedente

• **Cos'è il registro Program Counter (PC) e Instruction Register (IR)? Dove si trovano?**

PC e IR sono due registri speciali che si trovano in CPU: PC punta alla prox istruz.

mentre IR contiene l’addr dell’istruzione che sta venendo eseguita.

• **Cosa sono le istruzioni di controllo di flusso e perchè si utilizzano?**

Se voglio eseguire più volte una stessa istruzione; invece che ripeterla n volte in

RAM, utilizzo una serie di istruzioni che mi permettono di “saltare” da una istr. all’altra

tramite la modifica del PC (program counter).

• **Jump: istruzione di salto, descrivila.**

PC ← IR (assoluto) ; PC ← PC +- 4 (relativo)

• **Conditional Jump: istruzione di salto condizionale, descrivila.**

Come sopra, ma solo se di verifica una determinata condizione.

• **Istruzioni di tipo logico-aritmetico, cosa sono e da cosa vengono implementate?**

Vengono implementate dall’ALU e permettono di manipolare una seq. di bit.

• **Istruzione di tipo Load/Store e l'indirizzamento diretto/indiretto, in cosa consistono?**

Load: da RAM a registro; Store: da registro a RAM

Permettono di estrarre/*intrarre* valori tra ram e registri (LOAD R1← RAM [ 4 ])

• **Istruzione di scorrimento (Shift), in cosa consiste**

Stesso concetto per il reg. di scorrimento. Si aggiunge che uno shift a SX (con

numero con segno) può generare overflow ; DX no perchè normalmente si

re-inserisce la cifra più significativa.

• **Modo di indirizzamento indicizzato e di tipo autoincremento, che differenze?**

Il metodo indicizzato mi pemette di accedere ad un addr in ram sommando un REG

sorgente ad una costante per offset (REG← RAM[R4 + 4]); mentre autoincr. accede

all’indirizzo in un reg (post-autoincr e pre-autodecr; ad es R2←RAM[R4]; R4←R4+1).

• **Con quanti bit si lavora normalmente nella memoria RAM? Come si denominano?**

16 bit (word) - (4\*2) bit (byte)

• **Descrivi i modi di indirizzamento autoincremento indiretto, autodecremento indiretto.**

R2 ← RAM[ RAM[R4] ] ; R4 ← R4 + 1 (oppure viceversa per pre-autodecr)

• **In cosa differisce la tecnica di indirizzamento immediato da quella diretta?**

La diretta prende da un registro; la immediata da costante (un esempio è l’indirizz. immediato che si ottiene applicando al PC l’indirizz. auto-incr.)

• **In cosa consistono le istruzioni MOVE, COMPARE, BIT SET, ADD, SUB, XOR, MUL, ARITHMETIC SHIFT?**

MOVE: equivalente della load/store (copia un val./reg in un reg./ind.)

COMPARE: Viene sottratta ADD1 - ADD2 e il risultato che ci interessa (se <, >, ==)

viene salvato nel reg. di stato tramite bit di flag, mentre il risultato della

sottrazione viene ignorato.

BIT SET: and bit a bit tra src e dest (SRC & DEST) → risultato nel reg. di stato

ADD-SUB: esegue la somma algebrica tra due addendi (a decidere se add o sub

è il primo bit (0: add; 1:sub). → uno degli operandi viene sovrascritto.

XOR: xor bit a bit (istr. ad espansione 7 bit → src deve essere registro generale)

MUL: moltiplicazione tra due valori. Con indirizz. immediato il secondo op.ndo si

trova nella cella subito sottostante a quella dell’istruzione (word 16 bit).

ARITHMETIC SHIFT: shift logico a sx o dx. Il range può andare da -32 a 31

per l’arithmetic shift estesa ad una coppia di registri.

• **Com'è organizzata la memoria RAM (scopi delle varie celle)?**

CODICE - DATI STATICI - STACK - (HEAP)

• **Che funzionalità permette lo STACK? Qual è l'istruzione che "aggiunge" celle occupate? E quella di rimozione? Come si sviluppa lo STACK all'interno della RAM?**

1) fornire una zona per salvare il contenuto dei registri al context switch

2) fornire una zona per il passaggio parametri e per eventuali valori di ritorno

3) fornire una zona per le variabili locali

Le istruzioni PUSH e POP aggiungono e tolgono rispettivamente celle (NB: nella

nostra convenzione SP punta all’ultima cella piena). Lo stack, dipendentemente dalla

convenzione, può svilupparsi dall’alto verso il basso o viceversa.

• **Chiamata a funzione: JSR RTS e salvataggio registri**

JSR implementa la chiamata di una funzione salvando il contenuto di PC nello stack,

e caricando in esso l’indirizzo della prima istruzione della procedura a cui salto.

La RTS (op. inversa) ritorna al chiamante sovrascrivendo il PC col valore precedente

salvato nello stack. Quando chiamiamo una subroutine, potrebbe essere necessario

fare delle push (successive alla jsr) e poi delle contestuali pop (precedenti alla rts) al

fine di salvare dei registri R0, .. Rn usati dal chiamante.

• **In cosa consiste la mappatura in memoria dei registri per dispositivi I/O?**

Consiste nell’associare x indirizzi (per dispositivi I/O) non utilizzati per altro, ai

registri dei dispositivi. Accedendo a questo indirizzo, in realtà opero su di essi.

• **Qual è il problema principale della lettura e della scrittura (intesi come input e output) quando uno dei due dispositivi non è pronto? Cos'è il meccanismo dell'interrupt?**

In caso di dispositivo non pronto ho problemi di consistenza (rischio di perdere dei

caratteri). Piuttosto che effettuare polling (busy waiting) per aspettare un op. da

mandare in esecuzione, si introduce il meccanismo delle interruzioni: segnali inviati

dai dispositivi I/O contestuale all’operazione (read: vuoto (ho letto); write: pieno)

• **A cosa serve il sistema di priorità nelle interrupt? In quale fase la CPU non può rispondere alle interruzioni?**

Serve per discernere tra operazioni più urgenti (ad esempio l’input da tastiera, i cui

caratteri inseriti possono essere persi man mano che l’utente scrive) da altre più

rimandabili (ad es. l’output, che può essere messo in attesa). Si introduce il bit di

maschera per evitare loop dell’interrupt handler. Mask parte a 1; viene chiamato

l’handler e settato mask←0; in caso di ulteriore int. l’and tra il segnale e mask sarà 0

(in caso di più livelli di priorità verrà fatto l’or tra i bit di mask). La CPU non può

rispondere alle interruzioni durante il bootstrap, poichè in attesa delle inizializzazioni.

• **In cosa aiuta la Memory Management Unit (MMU) e la Virtual Memory?**

L’MMU traduce gli indirizzi logici generati dalla CPU in indirizzi fisici e impedisce

tramite trap l’accesso diretto tramite indirizzi fisici. Unica eccezione sarà durante il

bootstrap in mod privilegiata. Con la Virtual Memory razionalizzo la memoria (vedi

segmentation/paging), per facilitare l’astrazione della memoria (alla ram “non importa

avere una separazione logica in segmenti”).

• **Cos'è la tecnica di segmentazione? Cosa si intende per implicita ed esplicita?**

Consiste nel dividere la RAM in blocchi dediti ognuno a determinati compiti. Si ha la

modalità esplicita (specifica nell’addr il #segmento su cui operare) e la implicita: si

assume di conoscere su quale segmento si opererà sulla base del tipo di operazione

• **Cos'è la tecnica di paginazione? Cosa risolve rispetto alla segmentazione? Come vengono organizzate le pagine?**

Consiste nel suddividere la RAM in pagine di ugual misura in modo di poter

tabellare gli spazi inutilizzati all’interno della RAM, così da poter assegnare pagine

vuote ad altri programmi senza riallocazioni→ efficienza.

• **Segmentation + paging**

Posso associare pagine fisiche a logiche: i segmenti devono essere multipli della

dimensione di una pagina. Vengono mappate le pagine libere sugli indirizzi virtuali

consecutivi nel medisimo segmento. S’introduce dunque una tabella con l’associaz.

tra #pagVirtuale e #pagFisica, alla cui corretta posizione accederò tramite una

seconda tabella (Segm. table) contenente la size e l’addr della Page table.

La CPU manda l’ind. virtuale alla MMU che indirizza la segm. table sulla base dei bit

in modalità esplicita, per poi controlalre la size (nel caso TRAP) ed effettuare la

traduzione in ind. fisico accedendo alla specifica page table. L’offset è riportato =.

• **Illustra i motivi che spingono a usare l'impaginazione a richiesta (Demand Paging). Cos'è il working set?**

In principio nessuna pagina è in RAM, e tutte si trovano in Swap Area (nell’HD): i

bit pres sono dunque inizializzati a 0. Al momento di una richiesta di traduzione da

ind. logico a fisico, se la pagina richiesta nella pageTable ha pres=0, viene mandata

una Trap e l’operazione viene interpretata come una richiesta esplicita di aggiungere

la pagina al working set in RAM.

• **In cosa consiste l'algoritmo LRU (Page replacement algorithm)?**

S’introducono i bit used (sono state effettuate operazioni su quella pagina) e dirty

(sono state effettuate operazioni di scrittura su quella pagina → modifcata).

Utilizzando questi bit per seguire delle priorità sulla sostituzione dal working set

delle pagine e sfruttando la proprietà di località nel tempo, l’LRU (approssimato)

azzera allo scadere di un timer i bit used al fine di approssimare la località nel tempo.

Precedenza: U:0 D:0 → U:1 D:0 → U:0 D:1 (solo con LRU) → U:1 D:1

Non è ovviamente un metodo preciso (quale è l’utilizzo di un timestamp, che tuttavia

diventa oneroso per la necessità di mantenerlo per ogni pagina).

• **In generale, quale fase di un programma prende più tempo: l'avvio o l'esecuzione intermedia?**

L’avvio; poichè vanno caricate le pagine

• **Cosa sono le istruzioni privilegiate? In cosa differiscono da quelle normali?**

Le istruzioni privilegiate possono essere eseguite esclusivamente in modalità priv.

bypassando così l’MMU e lavorando direttamente con l’ind. fisico.

• **Cos'è e perchè è importante il Security Kernel?**

Il security kernel gestisce le risorse hw del sistema, e filtra le richieste degli utenti

in modo tale che ognuno di essi possa accedere esclusivamente alle informazioni

ad esso riservate. Il passaggio da modalità privilegiata a non priv non è critico,

mentre il viceversa lo è (manda trap/interrupt al secuirty kernel, che lavorerà a interr.

disabilitate).

• **Come si definisce la virtualizzazione in termini generici?**

L’utilizzo combinato di istruzioni privilegiate, interrupt e trap handlers e mem virtuale

• **Quali sono i quattro principi di Denning? (Sicurezza della macchina)**

Virtualizzazione risorse: mediazione tra utenti e risorse (il sec.ker media)

Minimo privilegio: dare i minimi permessi necessari agli utenti del sistema (integrità)

Economia/ridondanza di controllo: semplifico al max / aggiungo layer di controllo

Anello debole della catena: la sicurezza del sistema dipende dal suo anello debole

• **Cos’è l’ipervisore**

Funziona come nucleo di sicurezza realizzando i principi di Denning. Ne esistono

due tipologie: 1) sostituisce l’S.O installando direttamente sull’hw l’ipervisore: l’S.O

crede di poter gestire direttamente l’HW.

2) ipervisore a livello più alto (applicativo) che virtualizza un set di vm interfacciando

ad esse → la virtualizzazione vera e propria viene fatta dal S.O (il quale a sua volta è

una “virtualizzazione” che implementa il time sharing) e le vm effettueranno system

call mediate dall’ipervisore per le loro istruzioni privilegiate. S’introduce anche il

concetto di context switch: interrompo un programma salvando i suoi registri, e

ripristinando i registri di un programma che era stato sospeso precedentemente.

MEMORIE, VIRTUALIZZAZIONE, PIPELINE

• **Da cosa è composto un dispositivo Cache e perchè si utilizza tale memoria?**

La cache è un dispositivo di memorizzazione statica (composto da registri). Viene

utilizzata poichè più veloce della RAM, che utilizza tecnologia dinamica.

• **Descrivi i principi di località nei programmi, da chi è gestito questo processo e come vengono usate?**

Località nel tempo: dopo aver generato un indirizzo i, è probabile lo rigeneri dopo

Località nello spazio: se è stato generato i, mi aspetto venga generato anche i+1

Un esempio di cache è la TLB (parte dell’MMU) che è il dispositivo che effettua

l’associazione tra il [#segm + #pagine dell’ind. virtuale] (tag) e l’indirizzo fisico.

(l’offset è irrilevante poichè copiato). Nella TLB si sceglie di tenere i nuovi valori e

scartare i vecchi, sulla base delle proprietà discusse sopra → algoritmo LRU

• **Che tipo di memoria è la Cache? Da cosa è costituita una linea di Cache?**

La cache è una memoria associativa (TAG-VALORE).

La cache è organizzata in linee (es: [tag] [ ] [ ]: tag associato a 2 celle contigue)

NB: con linee più “grandi” sfrutto di più la località nello spazio

• **In che modo viene applicato l'algoritmo LRU nella Cache?**

Dovendo operare un confronto “<” non basta uno XOR, dunque si usa LRU appross.

perchè sarebbe dispendioso il contrario (LRU con timestamps)

• **Descrivi le 3 alternative con cui può essere organizzata la Cache**.

COMPLETAMENTE ASSOCIATIVA: 1 comparatore per ogni tag (liv. assoc. max)

CORRISPONDENZA DIRETTA: 1 solo comparatore (liv. assoc. 1)  
 ASSOCIATIVITA’ A INSIEMI: ogni sottoinsieme di n tag ha un comparatore.

→ livello di associatività 1 < n < max

NB: si introducono i livelli di cache (ai primi livelli: + veloce, - grande, vicino alle

caratteristiche della CPU / ai livelli inferiori: - veloce, + grande, simile alla RAM).

La prima cache *potrebbe* essere assoc. completa, l’ultima a corrispond. diretta.

• **In che momento è necessario attuare il protocollo di consistenza?**

La Write richiede il protocollo di consistenza.

• **Descrivi i due protocolli di consistenza.**

Write-through: aggiorno in RAM e in cache.

Vantaggio: mantiene la consistenza / Svantaggio: lentezza

Viene utilizzato dalle prime cache (semplifico sulle prime e velocizzo dopo)

Write-back: non copio sempre in RAM → copio per forza alla sostituzione della

linea di cache per mantenere la consistenza. Viene utilizzato nelle ultime cache

(+ grandi → meno frequente il caso di sostituzione delle linee).

• **Virtualizzazione: quali sono i 4 cardini del concetto di sicurezza?**

Riservatezza: fornire accesso alle risorse solo agli aventi diritto

Affidabilità: avere mezzi per riparare guasti (1 caricabatterie)

Integrità: mantenere la consistenza (attraverso memoria virtuale)

Disponibilità: avere più mezzi per riparare guasti (2 caricabatterie)

• **Cos'è la pipeline, come e perché viene implementata?**

Tramite il pipelining si “aumenta la velocità” della cpu a parità di frequenza di clock.

Viene realizzato tramite l’introduzione di più dispositivi (1 per ogni fase) che

permettono l’esecuzione parallela di più istruzioni. Ad es. posso avere 4 stadi della

pipeline: (1) is int/trap? (2) fetch (3) decode (4) exec.

• **Legato al meccanismo della pipeline, quali sono i due tipi di istruzione di salto?**

JUMP: normale esecuzione dell’istruzione di salto

DJUMP: l’istruzione successiva (è nella pipeline) all’istruzione di salto viene

eseguita comunque. Se essa non è eseguibile (non esiste, altro jump,..) la

sostituisco con un NOP (“non guadagno” un ciclo di ck ma evito errori).

Ciò non viene gestito nel caso di jump condizionale (+2 istruzioni nel delay

perchè se c’è un jump cond. me ne accorgo all’exec e non al decode).

AMBER

• **Che cos'è Amber? In cosa differiscono le versioni 23 e 25?**

è un processore di tipo ARM. La prima cifra (2) indica la versione di arm (ARM 2).

La seconda indica il numero di stadi della pipeline. NB: la versione 25 esegue in un

unico ciclo di pipeline la read e la write.

• **Quanti registri vi sono complessivamente in Amber? Cosa sono i modi di funzionamento e che relazione hanno con questi registri?**

- 16 registri (da 0 a 7 generali; specifici: R8, 9, 10, FP, IP, SP, LP, PC)

- 4 modi: 00: USR 10: interrupt (irq) 01: fast interrupt (firq) 11: supervisor (SVC)

firq ha duplicati i registri dall’8 al 14; dal 13 al 14 sono duplicati per ogni modo

I primi 8 non sono duplicati, così come il 15 (PC)

• **Com'è strutturata un'istruzione di tipo REGOP?**

Il tipo d’istruzione REGOP comprende:

[cond.] [00] [I] [OPcode] [S] [Rn] [Rd] [shifter\_operand]

con I: 1 se immediato / S: 1 modifica il bit di stato / Rn è un operando, sh\_o l’altro

shifter\_operand: permette di effettuare diverse operazioni, tra le quali delle rotazioni

sulla rappresentazione al fine di poter rappresentare parole su più bit di quelli

effettivamente disponibili ( se I = 0; sennò prendo un registro ): lsl, lsr, asr, ror

• **Descrivi l'istruzione di swap.**

Salva il contenuto di un registro in una cella, e nel frattempo mette il contenuto di

un’altra cella (o della stessa) in quel registro.

(Rn: source dalla ram ed Rd: reg. in CPU per lo swap; Rm: cella destinazione RAM)

• **Descrivi i tipi l'istruzione di trans e mtrans.**

Con trans s’intendono le istruzioni load e store; il bit L diversifica load (1) e store(0)

(Tramite un offset negli ultimi 12 bit si può effettuare un indirizzamento indexed;

PU e WL (in generale) servono a differenziare i modi di funzionamento)

• **Descrivi l'istruzione di branch e branch&link.**

Esegue un salto nel PC. Se bl salva in LP il vecchio valore del PC, utile per dei

ritorni al chiamante (su chiamate singole, ad es. in chiamate ricorsive si passa a

salvare in stack).

• **Da cosa è determinato il prefisso "co-" nelle istruzioni codtrans, coregop, cortrans?**

Sono istruzioni delegate dalla CPU a un coprocessore.

• **Cos'è l'istruzione SWI?**

Software Interrupt: interrupt mandate da programmi applicativi per eseguire sys\_call

(provoca cambiamento di stato verso modalità SVC (11))

Es: MODE (bit nel PC) ← PRIV ; LP ← PC ; PC ← K (handler)

BUS DRIVER, DMA, BUFFER

• **Cos'è il bus e qual è il protocollo a cui fa riferimento? In quali tipi di protocolli / circuiti fa si suddivide?**

Il bus è un sistema di comunicazione seriale tra più dispositivi, facente riferimento

al protocollo Master-Slave. In mod async non ha bisogno di un segnale di clock,

mentre in mod sync ha bisogno del ck.

• **Qual è lo svantaggio di un circuito asincrono (nella scrittura)?**

Dovendo considerare l’ack per preparare il bus alla prox operazione, e questo

genera più ritardo (in bus lunghi questo ritardo può determinarmi il max ritardo

possibile). Nella scrittura abbiamo 2 cicli di esecuzione (nella lettura 3).

Fasi Read (async): 1) Master(M) prepara l’addr, R/!W, ... e fa partire (CS=1)

2) Slave (S) esegue il comando ricevuto (scrive) e manda ack = 1

3) M legge ack = 1 e legge i dati. Mette CS a 0 per terminare

4) S stacca il 3-State (data) e riporta ack = 0

Fase Write: 1) M manda i dati e CS=1

2) S legge → effettua l’operazione e alla fine mette ack=1

3) M riceve ack=1 (S ha scritto e ha concluso) e mette di conseguenza CS=0

Il vantaggio è l’indipendenza da ritardi e caratteristiche dei dispositivi.

• **Cos'è l'arbitrazione del bus e in che ambito la si utilizza?**

Si aggiunge una logica di arbitrazione del bus nei casi in cui più master devono

concorrere all’utilizzo del bus (cpu in casi multicore / DMA).

1. M fa richiesta (REQ) al bus 2) con un criterio il bus accetta una req: GRANT

• **Qual è il problema principale degli accessi in scrittura a una risorsa da parte di due utenti/applicazioni differenti? Come viene risolto?**

Il problema principale sono le race conditions, ossia accessi contemporanei e non

gestiti da parte di più utentia ad una stessa risorsa in memoria condivisa. Può essere

risolto con advisory lock, anche se per utilizzarlo tutte le app devono seguire il proto.

• **Cos'è il Buffer relativo ai trasferimenti I/O?**

Il buffer di sistema (e il buffer applicativo).

• **Parla della DMA e dei due “tipi” di DMA**

La DMA è un dispositivo a cui delegare le operazioni I/O da e verso i dispositivi (più

sensato usarla in device che richiedono grossi spostamenti (block devices come il

disco oppure NIC) e non in char devices come mouse e tastiera). La CPU dovrà

programmare la DMA (tipo di op, da dove a dove, quanto scrivere/leggere).

Vantaggi: aumento prestazioni CPU conseguente alla delega di op.

Svantaggi: costi (se ogni dispositivo ha la sua DMA).

La DMA può essere un dispositivo separato, oppure essere inclusa all’interno del

device I/O; in quest’ultimo caso dimezzo gli accessi al bus poichè non sarà più

necessario utilizzarlo per accedere ai dati nel registro I/O.

**• Come si risolve il problema di dispositivi I/O troppo veloci?**

Tramite l’introduzione di buffer, per permettere al device di “scaricare” i dati nel

frattempo che essi vengono letti dalla DMA (nel caso read).

Per comunicazione su sistemi veloci si utilizza la struttura ring-based, comprendente

2 strutture ad anello (una per IN, una per OUT) che possono essere popolate con

indirizzi di buffer aggiunti alla init dalla CPU, la quale comunicherà alla DMA dove

trovare il ring. La DMA dovrà dunque solo accedere con polling e operare sul buffer x

se il bit di occupazione lo permette (es. bit = 1 nell’input→ c’è info da leggere),

non rendendo più necessarie continue riprogrammazioni da parte della cpu. In caso

write a fine op la DMA mette il bit a 0 e manda un interr. per notificare op. avvenuta.

• **Tipi di kernel**

Monolitico: opera su reg/IO/devices/risorse fissi e immutabili → devo ricompilarlo in

caso di variazioni. E’ dunque un sist completo e a sè stante (ridondanza di controllo)

Mircro Ker: al contrario del monolitco effettua solo le operazioni di virtualizzazione,

delegando a dei server (applicazioni non privilegiate ma comunque parte dell’OS) il

resto delle operazioni. Si ha dunque più fault tolerance e scalabilità.

Modulare: via di mezzo tra i due e per questo il più usato nella realtà (passare al

micro ker sarebbe stato difficile): opera con moduli interscambiabili (device driver

MA operanti a livello privilegiato a differenza dei server del micro ker → si mantiene

la struttura USR<-->Intermediario(Ker)<-->risorse; pur con modularità e portabilità.

• **POSIX bytestream: quando viene effettuata la write?**

Viene effettuata o 1) implicitamente (buffer finito) o 2) alla fflush (chiamata esplicitam.

dal programmatore oppure dalle primitive di stampa quando s’incontra “\n”).

• **Lezione del 22/04/21: esempio sulla READ e fasi**