# Guida alla preparazione all’esame orale di Embedded System

**Domanda1 : Il Bus I2C e caratteristiche. SLIDE3**

Il bus I2C è un canale di comunicazione seriale sincrono, multi master, multi slave, con comunicazione a pacchetto e terminazione singola.

Ciascun dispositivo connesso al bus è identificato tramite un indirizzo e il clock è scandito dal Master.

È dotato di un meccanismo di controllo delle collisioni e non necessita di un rigoroso bit-rate.

È utilizzato per connessioni tra circuiti integrati a breve distanza e a velocità che vanno dai 400 kbit/s ai 5 Mbit/s.

Il trasporto dei dati avviene tramite due collegamenti: SDA Serial Data e SCL Serial Clock.

Le connessioni OPEN-DRAIN consentono di supportare più master sul canale condiviso e anche l’indirizzamento di più slave oltre allungamento del clock in caso di slave più lenti che tengono giù SCL.

Il canale di comunicazione è bidirezionale e condiviso.

**In trasmissione:** il livello logico basso vene attivato dal pull-down FET cortocircuitato a massa, e per il livello logico alto viene spento il pull-down FET e la linea viene rilasciata flottante sul resistore di pull-up che porta la tensione a salire.

Lo slave non può trasmettere dati a meno che non venga etichettato come master. Ogni driver connesso al bus ha un codice univoco identificativo al proprio interno.

Per l’invio dati il Master invia sul canale la START CONDITION che è una transizione **da alto a basso** sulla linea SDA mentre sulla linea SCL il segnale è alto. Quindi trasmette il dato e completa la trasmissione con la STOP CONDITION ovvero la transizione **da alto a basso** sulla linea SDA mentre sulla line SCL il segnale è alto.

Esiste anche la condizione di START RIPETUTA che è equivalente ad aprire, chiudere e riaprire la connessione, utile quando il Master vuole ritrasmettere senza perdere il controllo del bus.

Un bit di dati viene trasferito ad ogni impulso di clock della SCL. Ogni byte di dati è composto da otto bit trasmessi sulla linea SDA. Vengono trasmessi prima i bit più significativi (MSB).

Tra la Start e la Stop condition si può trasferire qualsiasi numero di byte.

Alla trasmissione di ciascun byte il ricevitore trasmette un ACK non appena il trasmettitore avrà rilasciato la linea SDA. Per fare ***ciò tira giu la linea SDA durante il nono ciclo di clock per confermare la ricezione***. Se invece resta alto equivale al trasmettere un NACK ovvero l’errata ricezione.

Ovviamente sono molteplici le condizioni che portano al NACK come: Il ricevitore non è in grado di ricevere o trasmettere, ll destinatario non può ricevere più byte di dati, il master ha finito di leggere i dati e lo indica allo slave tramite un NACK oppure il ricevitore riceve dati o comandi che non comprende.

In dettaglio per **iniziare a scrivere** sul bus, il master invia una condizione di avvio con l'indirizzo dello slave seguito dal bit R/W impostato a 0; quindi lo slave invia l’ ACK di conferma; Il master invia l'indirizzo di registro del registro a cui desidera scrivere; lo slave conferma; a questo punto inizia la trasmissione dei dati che termina con la condizione di Stop.

**Per leggere i dati sul bus** il master invia la condizione di START seguita dall'indirizzo dello slave con il bit R/W impostato a 0 e dall'indirizzo del registro da cui vuole leggere; se lo slave riconosce l'indirizzo del registro, il master invia nuovamente la Start Condition (START ripetuto), con il bit R/W impostato ad 1; se lo slave conferma la richiesta il master rilascia SDA e continua ad fornire il clock allo slave: a questo punto Il master diventa il master-ricevitore e lo slave lo slave-trasmettitore. Ad ogni byte ricevuto il master invia l’ACK e al termine della ricezione invia un NACK e quindi la Stop condition.

**Domanda2 : Indirizzamento a 10 bit. SLIDE3**

Al fine di evitare conflitti di indirizzi, a causa della gamma limitata degli indirizzi a 7 bit, è stato introdotto un nuovo schema di indirizzamento a 10 bit. Questo miglioramento può essere combinato con l'indirizzamento a 7 bit e aumenta l'intervallo di indirizzi disponibile di circa dieci volte. Dopo la condizione di avvio, un "**11110**" iniziale introduce lo schema di indirizzamento a 10 bit. Gli ultimi due bit di indirizzo del primo byte concatenati con gli otto bit del secondo byte dell'intero indirizzo a 10 bit. I dispositivi che utilizzano solo l'indirizzamento a 7 bit ignorano semplicemente i messaggi con '11110' iniziale.

**Lo Slow Star byte**

I microcontrollori non dotati di circuiti di controllo I2C devono osservare le linee I2C in modo permanente per rilevare una trasmissione I2C

(polling). Per ridurre lo spreco di potenza della CPU, è possibile stabilire un trasferimento I2C con un metodo di arbitrato più lento

**Domanda3 : Processori DSP. SLIDE1**

**I processori DSP sono processori dedicati e ottimizzati per eseguire in maniera estremamente efficiente sequenze di istruzioni ricorrenti e specifiche. Molto usati nelle applicazioni embedded come nell’elaborazione di segnali di campionamento**

**Un set di segnali generato da misurazioni campionate del mondo fisico, generalmente prese a una frequenza regolare chiamata frequenza di campionamento, viene elaborato da questa tipologia di processori. Le frequenze di campionamento vanno dai pochi Hz fino a diversi GHz.**

**Elaborazione di segnali digitali ottenuti dalla conversione di segnali analogici, mediante uso di un convertitore analogico-digitale (ADC). Il segnale ottenuto può successivamente essere riconvertito in analogico usando un convertitore digitale-analogico (DAC).**

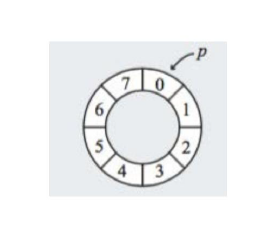
I microprocessori DSP a chip singolo ( SoC ) sono apparsi per la prima volta all'inizio degli anni '80, e le caratteristiche centrali dei DSP includono un'unità hardware ad accumulazione multipla; diverse varianti dell'architettura di Harvard (per supportare più dati simultanei e recuperi di programmi); e modalità di indirizzamento che supportano l'incremento automatico, i buffer circolari e l'indirizzamento con inversione di bit (quest'ultimo per supportare il calcolo FFT).

La maggior parte supporta la precisione dei dati in virgola fissa di 16-24 bit, in genere con accumulatori molto più ampi (40-56 bit) in modo che un gran numero di istruzioni di accumulazione multipla successive possano essere eseguite senza overflow.

I DSP sono difficili da programmare rispetto alle architetture RISC, principalmente a causa di istruzioni specializzate complesse, una pipeline esposta al programmatore e architetture di memoria asimmetriche.

**Domanda3 : Buffer circolare (finire)**

**Per realizzare un filtro FIR (Finite Impulse Response) che richiede una linea temporale, piuttosto che allocare in un array ciascun campione, uno in una nuova cella di memoria, è più conveniente usare un buffer circolare come struttura dati per l’elaborazione di tali campioni.**

**L'implementazione accetta prima un nuovo valore di input x(n) e poi calcola la somma a ritroso, iniziando con il termine i = N-1 termini, dove ad esempio N=8.**

**Quindi il buffer avrà sempre dentro otto campioni.**

**All’arrivo di ogni campione viene incrementato p dove pi ={ 0,… 7}, quindi per il primo input abbiamo x(0) e pi =0. Il** programma scrive il nuovo input *x(n)* nella posizione data da e quindi incrementa, impostando *p = pi +1.* **Così facendo si genera una aritmetica in modulo 8.**

**Ad ogni ciclo, il nuovo campione x[n] è inserito nella locazione relativa all'ultimo campione x[n-7] (quello più vecchio e l'indice di ognuno viene scalato)**

~~Il calcolo del filtro FIR legge quindi x(n-7) dalla posizione~~ *~~p~~* ~~=~~ **~~pi~~** ~~+ 1 e lo moltiplica per a7. Il risultato viene memorizzato in un registro accumulatore (precedentemente azzerato). P viene aumenta di nuovo di uno… e così via..~~

**Continua finché non legge x(n) dalla posizione p = pi + 8 scritto** a stessa posizione dell'ultimo input *x(n) e moltiplica quel valore per a0*

**Ad esempio un filtro FIR sia dotato di campioni alla frequenza di 1 MHz (un milione di campioni al secondo), e che N = 32** e uscite devono essere calcolate a una frequenza di 1 MHz e ciascuna uscita richiede 32 moltiplicazioni e 31 addizioni quindi u**n processore deve essere in grado di sostenere una velocità di calcolo di 63 milioni di operazioni aritmetiche al secondo per implementare questa applicazione**

**Domanda4 : Istruzioni di somma in ARM. SLIDEARM**

**Ogni istruzione Arm esegue un'operazione specifica su uno o due operandi. Il primo operando è sempre nel registro Rn mentre il Secondo operando è inviato all'ALU tramite barrel shifter.**

**Abbiamo 2 istruzioni di somma e sono *ADD Operando1 + Operando2* e *ADC Operando1 + Operando2+Carry***

**La sintassi è <Operazione>{<cond>}{S} Rd, Rn, Operando2 dove il secondo operando può essere pure un Immediato #Imm.**

**Domanda5 : GPIO. SLIDE3**

**I General Purpose I/O (GPIO) consentono il collegamento di dispositivi come microprocessori e microcontrollori con altre periferiche. Un GPIO oltre all'ingresso e all'uscita digitali, può essere commutato anche su funzioni alternative (AFs) o gestire periferiche interne. Il Digital Input/Output è gestito dai bit dei Pin Level Register che sono dei registri che servono a pilotare il comportamento di questi Pin.**

**Solitamente i Microcontrollori SoC embedded hanno moltissimi GPIO.**

**La selezione delle funzioni per i pin GPIO è gestita tramite una matrice di routing configurabile tramite registri. Ogni configurazione di bit di questi registri di selezione abilita rispettivamente l’input, output, o le funzioni alternative: ad esempio il PI ha 6 funzioni alternative per 54 GPIO. Ovviamente alcune GPIO sono riservate per il collegamento del SoC con l’hardware a bordo, mentre i 40 pin dell’Header contengono parecchi GPIO disponibili per il collegamento con hardware esterno.**

**Domanda6 : L'architettura Harvard. SLIDE1**

**L'architettura Harvard, in informatica, è un tipo di architettura hardware per computer digitali in cui vi è separazione tra la memoria contenente i dati e quella contenente le istruzioni. Il termine inizialmente indicava l'architettura del computer Harvard Mark I, un computer basato su relè che memorizzava le istruzioni su un nastro perforato mentre i dati venivano memorizzati in un contatore elettromeccanico a 23 cifre. Questa macchina non era dotata di un'unità di immagazzinamento dei dati: questi erano interamente memorizzati dalla CPU e il loro caricamento e salvataggio era un processo eseguito in modo manuale agendo sui contatori. In un'architettura Harvard le memorie per i dati e per le istruzioni possono essere anche differenti, con tecnologia d'implementazione e timing diversi, in particolare in alcuni sistemi l'ampiezza degli indirizzi o la larghezza di parola delle istruzioni è superiore a quella dei dati, e in altri sistemi i programmi sono memorizzati in una memoria a sola lettura (ROM) mentre i dati sono normalmente in una memoria a lettura-scrittura (RAM).**

**Oggi, la maggior parte dei processori implementa percorsi di segnale separati per motivi di prestazioni, ma in realtà implementa un'architettura Harvard modificata, quindi possono supportare attività come il caricamento di un programma dalla memoria del disco come dati ed eseguirlo.**

**Domanda7 : Architettura RISC SLIDE1**

**Reduced Instruction Set Computer (in acronimo RISC), nell'elettronica digitale, indica un'idea di progettazione di architetture per microprocessori che predilige lo sviluppo di un'architettura semplice e lineare. Questa semplicità di progettazione permette di realizzare microprocessori in grado di eseguire il set di istruzioni in tempi minori rispetto a una architettura CISC. I più comuni processori RISC sono AVR, PIC, ARM, DEC Alpha, PA-RISC, SPARC, MIPS, RISC-V, POWER e PowerPC.**

**Domanda8 : Architettura MIPs SLIDE1**

**L'architettura MIPS (acronimo dell'inglese microprocessor without interlocked pipeline stages) è un'architettura informatica per microprocessori RISC sviluppata dalla MIPS Computer Systems Inc. (oggi MIPS Technologies Inc.). Il disegno dell'architettura e del set di istruzioni è semplice e lineare, spesso utilizzato come caso di studio nei corsi universitari indirizzati allo studio delle architetture dei processori; tale architettura ha influenzato le architetture di molti altri processori RISC tra i quali si segnala la famiglia DEC Alpha: è utilizzata ad es. nel campo dei computer SGI, ed ha trovato grossa diffusione nell'ambito dei sistemi embedded, dei device di Windows CE e nei router di Cisco e anche le console Nintendo 64, Sony PlayStation, PlayStation 2 e PlayStation Portable utilizzano processori MIPS. Nel 1981 il professore John L. Hennessy della Stanford University avviò un gruppo di ricerca sulle architetture RISC. Le ricerche del team di sviluppo portarono allo sviluppo della prima generazione di processori MIPS. Allora era noto che per incrementare le prestazioni dei processori si sarebbe potuto utilizzare la tecnica delle pipeline. Questa tecnica sebbene fosse semplice da ideare non era semplice da implementare. Le pipeline per funzionare correttamente richiedono che le varie unità siano sincronizzate e che i dati delle varie istruzioni non si sovrappongano, quindi all'interno delle pipeline vengono posti dei blocchi (interblocco) che sorvegliano il completamento delle varie istruzioni e fanno procedere la pipeline solamente quando tutti gli stadi sono pronti. Questo meccanismo garantisce la corretta esecuzione del programma ma introduce spesso stalli nella CPU che deprimono le prestazioni.**

**Domanda9 : La pipeline ARM. SLIDEARM (da finire)**

**ARM utilizza una pipeline per aumentare la velocità del flusso d’istruzioni al processore. Consente di eseguire più operazioni contemporaneamente anziché in serie. Il flusso prevede tre passaggi: fase di FETCH, fase di DECODE e fase di EXECUTE.**

**Nella fase di caricamento il PC punta sempre all’istruzione successiva e mai a quella in esecuzione.**

**Nella fase di decodifica viene effettua la decodifica dei registri utilizzati nell'istruzione.**

**Nella fase di esecuzione vengono fatte diverse operazioni: Lettura dei registri coinvolti dalla Register Bank, Barrel Shift e operazione ALU, Riscrittura dei registri nella Register Bank.**

**Le istruzione ARM sono allineate a 4 byte.**

**Domanda10 : Interlocks. SLIDE2**

**DaSlide**

**Le istruzioni ML spesso dipendono dai risultati delle precedenti istruzioni: Se B necessita di un risultato da un registro scritto da A (data hazard) non può procedere lungo la pipeline.** Nelle macchine semplici il programmatore (o il compilatore) deve occuparsi di tali dipendenze (pipeline esplicita). il compilatore inserisce tre istruzioni no-op (che **non fare nulla) tra A e B per garantire che la scrittura avvenga prima della lettura. Queste istruzioni no-op formano una bolla di pipeline che** si propaga lungo la pipeline. Hardware aggiuntivo può gestire dati e controllo **problemi di rischi. Interlock inserisce automaticamente le bolle della** pipeline.

Nelle macchine più sofisticate sempre più complesse vengono adottate tecniche, come la ridenominazione dei registri, l'esecuzione fuori servizio e speculativa, la previsione dei rami, le condutture lunghe e la superscalarità.

**DaInternet**

**Interlocks**

**Il risultato di un'istruzione di caricamento della parola allineata non è disponibile fino al termine della fase di memoria della pipeline. Se la seguente istruzione richiede l'uso di questo risultato, allora deve essere interbloccato in modo che sia disponibile il valore corretto. Questo interblocco è denominato interblocco uso carico a ciclo singolo.**

**L'esempio seguente comporta un interblocco a ciclo singolo:**

**LDR r0, [r1]**

**ADD r2, r0, r3**

**ORR r4, r4, r5**

**L'esempio seguente non prevede un interblocco:**

**LDR r0, [r1]**

**ORR r4, r4, r5**

**ADD r2, r0, r3**

**Le istruzioni di caricamento di parole non allineate, di byte di caricamento (LDRB) e di caricamento di semiparole (LDRH) utilizzano l'unità di rotazione dei byte nella fase di scrittura della pipeline. Ciò introduce un interblocco uso-carico a due cicli, che può influenzare le due istruzioni immediatamente successive all'istruzione di carico.**

**L'esempio seguente prevede un interblocco a due cicli:**

**LDRB r0, [r1, #1]**

**ADD r2, r0, r3**

**ORR r4, r4, r5**

**Domanda11 :** **Branch e Branch with Link. (DA SISTEMARE)**

**B Branch e jal utilizzano rispettivamente un immediato con segno a 13 e 21 bit, per l'offset. I bit più significativi dell'offset sono codificati nei campi immediati a 12 e 20 bit delle istruzioni di tipo B e J. Il bit meno significativo dell'offset è sempre 0, quindi non è codificato nell'istruzione.**

**BL (branch and link) e MOV PC, LR sono le due istruzioni essenziali necessarie per una chiamata e un ritorno di funzione. BL svolge due compiti: memorizza l'indirizzo di ritorno dell'istruzione successiva (l'istruzione dopo BL) nel Link Register (LR), e si dirama all'istruzione di destinazione. Ricorda che PC e LR sono nomi alternativi rispettivamente per R15 e R14. Per ARM il PC fa parte del set di registri, quindi è possibile eseguire un ritorno di funzione con un'istruzione MOV. Molti altri set di istruzioni mantengono il PC in un registro speciale e utilizzano un'istruzione speciale di ritorno o salto per tornare dalle funzioni.**

**i compilatori ARM eseguono una funzione di ritorno utilizzando BX LR . L' istruzione branch e exchange BX è come una branch, ma può anche effettuare la transizione tra il set di istruzioni ARM standard e il set di istruzioni Thumb .**

**Qualsiasi chiamata Arm un Branch ARM con operazione di collegamento richiedono tre cicli:**

**Durante il primo ciclo, un'istruzione branch calcola la destinazione branch durante l'esecuzione di un Prefetch dal PC corrente. Questo Prefetch viene eseguito in ogni caso, perché nel momento in cui è stata presa la decisione di prendere il ramo, è già troppo tardi per impedire il Prefetch. Se l'istruzione precedente ha richiesto un accesso alla memoria dati, i dati vengono trasferiti in questo ciclo.**

***I bit 23-0 contengono il valore di offset (complementato a due) e l'offset è relativo all'indirizzo dell'istruzione branch +2 word***

***l'offset viene spostato di due posizioni a sinistra e aggiunto al PC***

***il bit 24 indica se il valore PC-8 deve essere copiato in LR (Link) prima della diramazione o meno***

**Domanda12 : ARM Data Processing Instructions e Format. SLIDEARM**

**I Processori ARM condividono tutti lo stesso formato di istruzione. Queste istruzioni possono essere: Operazioni Aritmetiche, Confronti, Operazioni Logiche, Spostamento tra registri. Le istruzioni funzionano esclusivamente sui registri e non sulla memoria. Architettura LOAD/STORE. Il primo operando è sempre un registro Rn; il secondo operando è inviato all'ALU tramite barrel shifter (può essere: omesso, un registro, o un immediato)**

**Domanda13 : Rappresentazione formato in virgola fissa.**

**Molti processori embedded forniscono hardware solo per l'aritmetica di interi. Per rappresentare un numero decimale, la strategia è quella di usare in punto binario che è come un punto decimale, tranne per il fatto che separa i bit anziché le cifre del numero. Consideriamo ad esempio un numero intero a 16 bit.**

**Con questa metrica è possibile rappresentare i numeri compresi tra -1 e 1 (approssimativamente) posizionando un punto binario (concettuale) appena sotto il bit di ordine superiore del numero.**

**Senza il punto binario, un numero rappresentato dai 16 bit è un numero intero x { -2^15, …, 2^15 -1} (complemento a due).**

**Invece con il punto binario, interpretiamo i 16 bit per rappresentare un numero y = x/2^15. Quindi l’intervallo sarà y { -1, …, 1 – 2^15}. Questo è noto come numero a virgola fissa. Il formato di questo numero a virgola fissa può essere scritto 1,15 che indica che c'è un bit a sinistra del punto binario e 15 a destra.**

**Quando due di questi numeri vengono moltiplicati alla massima precisione, il risultato è un numero a 32 bit. La posizione del punto binario segue dalla legge di conservazione dei bit (con i formati nm e pq, il risultato ha il formato (n+p).(m+q)).**

**I processori spesso supportano tali moltiplicazioni ad alta precisione, in cui il risultato va in un registro accumulatore che ha almeno il doppio dei bit rispetto ai normali registri di dati.** **Per riscrivere il risultato in un registro dati, però, dobbiamo estrarre 16 bit dai 32 bit del risultato. Esiste la possibilità di overflow, perché stiamo scartando il bit di ordine superiore ( es -1 \*-1).**

C'è una perdita di informazioni qui: se scartiamo semplicemente i 15 bit di ordine inferiore, la strategia è nota come **troncamento;** se invece aggiungiamo un bit alla parte più significativa del risultato a 32 bit, il risultato è noto come **arrotondamento**. L'arrotondamento sceglie il risultato più vicino al risultato di precisione completa, mentre il troncamento sceglie il risultato più vicino che è di grandezza minore.

**I processori DSP in genere eseguono l'estrazione con arrotondamento o troncamento nell'hardware quando i dati vengono spostati da un accumulatore a un registro generico o alla memoria.**

**Domanda14 : Il Frame buffer.**

***(Il display Pi HDMI è generato dal Videocore.)* Un pixel (picture element) si trova attraverso le sue coordinate xy ( da (0,0) a (-1,-1) ). Il colore di un *pixel è controllato da un valore codificato in bit color\_depth*. Quando si utilizza 32 bit per pixel (bpp), il colore del pixel è codificato come valore ARGB : 8 bit per ciascuno dei** **componenti red, green, blu colors 8 bit per il canale alfa utilizzato per gli effetti di trasparenza .**

**I valori dei pixel sono memorizzati in un framebuffer, che è una regione di memoria condivisa dal VC e dalla CPU. Il framebuffer è organizzato come la riga principale della matrice di color\_depth.**

**I valori dei pixel vengono compressi uno dopo l'altro in una matrice lineare di valori di bit di color\_depth. Il primo elemento dell'array controlla il pixel in alto a sinistra. In generale, il colore del pixel x,y è controllato dall'i-esimo valore nell'array lineare: i = width\*y+x.**

**Ad esempio il pijFORTHos-se configura il display HDMI per 1024x768x32bpp all'avvio e il colore del pixel x,y è controllato dal valore ARGB nella parola all'offset del byte 4\*(1024\*y+x) dall'indirizzo di base del framebuffer ( per il primo pixel bianco 00FFFFFF 3E8FA000 ! ).**

**Domanda15 : I registri ARM.**

**ARM ha 37 registri in totale, tutti lunghi 32 bit. 1 è Program Counter Register (r15), 1 è Current Program Status Register, 5 sono lo Saved Program Status Register e 30 General Purpose Register. Questi registri sono organizzati in diversi banchi, l’accessibilità ad ogni banco è regolata dalla modalità processore ( USER, FIQ, IRQ, SUPERVISOR, ABORT, UNDEF, in V4 anche SYSTEM ).**

**Ogni modalità può accedere: ai registri da r0-r12, a un particolare registro r13 (Stack Pointer), r14 (Link Register), al registro r15 (Program Counter) e al CPSR (Current Program Status Register).**

**Le modalità privilegiate possono accedere anche: al SPSR (Saved Program Status Register).**

**Invece tutte le istruzioni possono accedere direttamente a r0-r14. La maggior parte delle istruzioni consente anche l'uso del PC ( r15).**

**Domanda26 : Program Counter.**

**Il Program counter è il registro che, nella fase di fecth della pipeline, memorizza la prossima istruzione da eseguire. Pertanto il valore del PC viene memorizzato nei bit [31:2] con i bit [1:0] uguali a zero (poiché l'istruzione non può essere allineata a metà parola o byte).**

**Domanda20 : Il Link Register. ( da sistemare)**

**Il Link Register è un registro usato per scopi speciali che contiene l'indirizzo di ritorno quando viene completata una chiamata di funzione. Questo è più efficiente dello schema più tradizionale di memorizzazione degli indirizzi di ritorno in uno stack di chiamate, a volte chiamato stack di macchine. Il Link Register non richiede scritture e letture della memoria contenente lo stack che può far risparmiare una notevole percentuale di tempo di esecuzione con ripetute chiamate di piccole subroutine.**

**Domanda18 : Gestione dello Stack.**

**Lo Stack è un'area di memoria che cresce quando i nuovi dati vengono "spinti" sulla "parte superiore" di esso e si restringe quando i dati vengono "spuntati" dall'alto. Due puntatori definiscono i limiti correnti dello stack*: Base pointer* e *lo Stack pointer*. ARM usa lo STMFD / LDMFD: stack discendente completo ovvero lo Stack Pointer punta sempre all'ultimo indirizzo occupato (Full stack).**

**Un uso degli stack è quello di creare un'area di lavoro di registro temporanea per subroutine. Tutti i registri necessari possono essere inseriti nello stack all'inizio della subroutine e estratti di nuovo alla fine in modo da ripristinarli prima di tornare al chiamante: STMFD sp!,{r0-r12, lr} impilare tutti i registri e l 'indirizzo di ritorno. Se l'istruzione pop ha anche il bit 'S' impostato (usando '^'), il trasferimento del PC in una modalità privilegiata causerebbe anche la copia dell'SPSR nel CPSR (vedi modulo di gestione delle eccezioni).**

**Domanda29 : Per codificare un riferimento ad un registro quanti bit servono?**

**Generalmente servono 4 bit. Ma nel caso di una load/store multiple (LDM / STM) basta 1 bit ( da 0 a 15 ).**

**Domanda19 : Processori Embbedded vs General Purpose (tempo medio).**

**Le CPU General Purpose sono in media molto veloci perché sono superpipeline, superscalari e sfruttano ottimizzazioni come l'esecuzione fuori ordine, la previsione del brach supportata da memorie cache veloci. Per fare una valutazione del tempo medio viene misurato il tempo totale trascorso e viene calcolato il tempo per l'elaborazione del campione. In questo aspetto una CPU embedded, avendo hardware dedicato a svolgere l’elaborazione del campione senza fronzoli, è la scelta migliore; inoltre è meno costosa, spesso con un consumo minore di risorse, ed anche il software è più semplice poiché realizzato su hardware che non deve fare diverse operazione, ma sempre le stesse… quindi con istruzioni ben definite allo scopo : per esempio i DSP.**

**I processori embedded utilizzano spesso architetture VLIW anziché architetture superscalari per ottenere tempi più ripetibili e prevedibili. I processori VLIW includono più unità di funzione, come i processori superscalari, ma invece di determinare dinamicamente quali istruzioni possono essere eseguite contemporaneamente, ciascuna istruzione specifica cosa dovrebbe fare ogni unità di funzione in un ciclo particolare. Un set di istruzioni VLIW combina più operazioni indipendenti in un'unica istruzione.**

**Ad esempio processori superscalari presentano uno svantaggio significativo per i sistemi embedded, ovvero che i tempi di esecuzione possono essere estremamente difficili da prevedere e, nel contesto del multitasking (interrupt e thread), potrebbero non essere nemmeno ripetibili.**

**Gestione di sistemi realtime richiedono vincoli temporali. Meccanismi hardware come le pipeline esplicite sono spesso applicate in contesti in cui è essenziale una tempistica precisa;**

**Per i sistemi embedded le vanno privilegiati i meccanismi che rendono le tempistiche precise e prevedibili come:**

**Data and Control Hazard: *L'analisi dei tempi* di un programma può diventare estremamente difficile quando c'è una profonda pipeline, con branch elaborati e speculazioni; *Le pipeline esplicite* sono relativamente comuni nei processori DSP, e sono spesso applicate in contesti in cui è essenziale una tempistica precisa; *Speculative execution* invece sono comuni nei processori General Purpose, dove i tempi sono importanti solo in senso aggregato.**

**Un progettista di sistemi embedded deve comprendere i requisiti dell'applicazione ed evitare processori in cui il livello di precisione temporale richiesto è irraggiungibile.**

**Domanda21 : Salto condizionato.**

**La maggior parte dei set di istruzioni consente l'esecuzione di branch solo in modo condizionale. Invece ARM riutilizzando l'hardware di valutazione delle condizioni aumenta in maniera efficente il numero di istruzioni. Infatti tutte le istruzioni contengono un campo condizionale che determina se la CPU le eseguirà. Le istruzioni non eseguite assorbono comunque 1 ciclo. Ciò elimina la necessità di molte diramazioni, che bloccherebbero la pipeline. La penalità di tempo per non aver eseguito più le istruzioni condizionali è spesso inferiore all'overhead della chiamata a branch o a subroutine che sarebbe altrimenti necessaria.**

**I bit tra 31:28 rappresentano il blocco condizionale (N Z C V). Per eseguire un'istruzione in modo condizionale, è sufficiente postfissare l’istruzione con la condizione adeguata.**

**ADDEQ r0,r1,r2 ; Se il flag zero è impostato, allora... r0= r1 + r2 . Per fare in modo che i flag di condizione vengano aggiornati, è necessario impostare il bit S (bit 20 per ARITMETICA) dell'istruzione.**

**Domanda23 : Timing.**

**Data and Control Hazard: *L'analisi dei tempi* di un programma può diventare estremamente difficile quando c'è una profonda pipeline con branch elaborati e speculazioni; *Le pipeline esplicite* sono relativamente comuni nei processori DSP, e sono spesso applicate in contesti in cui è essenziale una tempistica precisa; *Speculative execution* invece sono comuni nei processori General Purpose, dove i tempi sono importanti solo in senso aggregato.**

**Un progettista di sistemi embedded deve comprendere i requisiti dell'applicazione ed evitare processori in cui il livello di precisione temporale richiesto è irraggiungibile.**

**Domanda24 : Esecuzione vettoriale.**

**Molte applicazioni embedded operano su tipi di dati notevolmente inferiori alla dimensione della parola del processore (es. dati RBG). Un'ampia ALU è suddivisa in sezioni più strette consentendo operazioni aritmetiche o logiche simultanee con parole più piccole.**

**Intel ha introdotto il parallelismo delle subword nel processore Pentium per General Purpose con MMX. Molte architetture di processori progettate per applicazioni embedded, inclusi molti processori DSP supporta anche il parallelismo delle subword.**

**Un processore vettoriale è *quello in cui il set di istruzioni include operazioni su più elementi di dati*** *contemporaneamente*.

**Il parallelismo delle subword è una forma particolare di elaborazione vettoriale.**

**Domanda15 : Barrel Shifter**

**In Arm non esiste un vero e proprio meccanismo di spostamento, ma il Barrel Shift riesce a compensare.**

**Esistono diverse tipologie di SHIFT:**

**Left Shift: produce spostamenti a sinistra per l'immediato specificato (moltiplica per potenze di due) es: LSL #5 = moltiplica per 32.**

**Right Shift: Sposta a destra della quantità specificata (divide per potenze di due) es: LSR #5 = dividere per 32**

**Arithmetic Shift Right: sposta a destra (divide per potenze di due) e conserva il bit del segno, per le operazioni in complemento a 2. per esempio ASR #5 = dividere per 32**

**ROR è simile a un ASR ma i bit ruotati. Quindi partendo da LSB appaiono come MSB. L’ultimo bit ruotato è utilizzato anche come Carry Out. ROR #5**

**Rotazione a destra estesa (RRX): Questa operazione utilizza il flag CPSR C come 33° bit. Ruota a destra di 1 bit. ROR #0**

**Nel Barrel Shift il *secondo operando* può essere: un registro con un valore di spostamento intero senza segno a 5 bit oppure di un valore specificato in un altro registro (byte inferiore); un Immediato (numero da 8 bit) che può essere ruotato a destra di un numero pari di posizione ed è l’assembler a calcolare la rotazione da costante per un numero pari di posizioni.**

**Domanda16 : Bus parallelo.**

**La caratteristica peculiare della trasmissione parallela è che vengono utilizzati più conduttori per trasmettere simultaneamente informazioni: un cavo che effettua una trasmissione parallela a n bit è formato da almeno n conduttori separati. Ogni linea accende o spegne uno specifico bit. Nella realtà il cavo sarà dotato quasi sicuramente di un cavo aggiuntivo per la massa e anche di altri cavi di controllo come quello di clock. Rispetto alla trasmissione seriale la trasmissione parallela risulta avere prestazioni più elevate in termini di velocità di trasmissione a parità di frequenza, ma ovviamente sarà anche più ingombrante e costosa.**

**Domanda17 : Load/Store e Multiple.**

**ARM è un'architettura Load/Store. Non supporta le operazioni di elaborazione dei dati direttamente da memoria a memoria. Sposta i valori dei dati nei registri prima di utilizzarli.**

**1 Caricare i valori dei dati dalla memoria nei registri.**

**2 Elaborare i dati nei registri utilizzando una serie di istruzioni di elaborazione dati che non sono rallentate dall'accesso alla memoria.**

**3 Memorizza i risultati dei registri in memoria.**

**Le istruzioni e gli mnemonici sono: Trasferimento dati a registro singolo (LDR / STR), trasferimento blocco dati (LDM/STM), Swap (SWP).**

***Single Register Data Transfer*: Le istruzioni di LOAD e STORE sono LDR/STR/LDRB/STRB. Nella V4 sono supportate le mezze parole, ed ogni istruzione può essere condizionale.**

**Intanto la posizione di memoria a cui accedere è conservata in un registro base:**

**STR r0, [r1] Memorizzare il contenuto di r0 nella posizione indicata dal contenuto di r1**

**LDR r2, [r1] Carica r2 con il contenuto della posizione di memoria indicato dal contenuto di r1.**

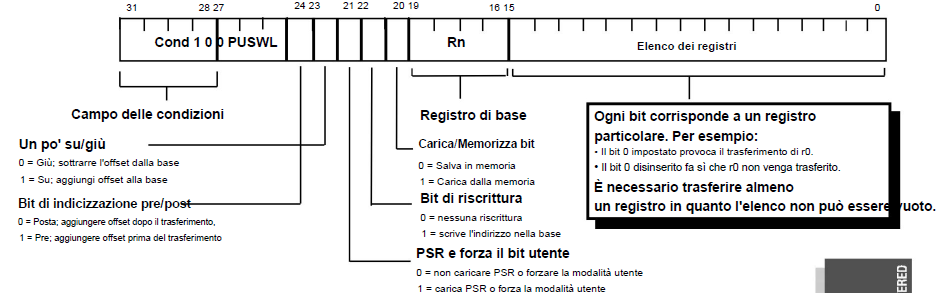
**Oltre ad accedere alla posizione effettiva contenuta nel registro di base, queste istruzioni possono accedere a un offset di posizione dal puntatore del registro di base che può essere un immediato da 12 bit ( 0-4095 ) o un registro opportunamente shiftato ad un valore immediato che può essere aggiunto o sottratto dal registro di base ( base + o base -). Questo offset può essere applicato prima che venga effettuato il trasferimento (Indirizzamento pre-indicizzato) oppure dopo (Indirizzamento post-indicizzato) provocando l'autoincremento del registro di base.**

**Pre-indicizzazione *STR r0, [r1,#-12]* oppure *STR r0, [r1, #12]!* o *STR r0, [r1, r2, LSL #2]* se r2 contiene 3.**

**Post-indicizzazione *STR r0, [r1], #12* oppure *STR r0, [r1], #-12* o *STR r0, [r1], r2, LSL #2* se r2 contiene 3.**

***Block Data Transfer*: Le istruzioni di LOAD e STORE multiple (LDM / STM) sono consentite per i registri dall’ 1 al 16 per il trasferimento dalla memoria, e per la memoria.**

**Ogni bit dall’0 al 15 corrisponde ad un registro. Il bit 0 impostato provoca il trasferimento di r0. È necessario trasferire almeno un registro in quanto l'elenco non può essere vuoto.**

****

**Il Base Register è utilizzato per determinare da dove si dovrebbe accedere alla memoria. Abbiamo 4 diverse modalità di indirizzamento consentite: incrementi e decrementi inclusi o esclusivi del registro di base.**

**Il registro di base può essere aggiornato facoltativamente a seguito del trasferimento ( ! ).**

**Queste istruzioni sono molto efficaci per il salvataggio e ripristino del context e per lo spostamento di grandi blocchi di dati nella memoria.**

**LDM / STM vengono utilizzati anche per implementare gli stack.**

***SWAP*: L’istruzione SWP è una operazione atomica di una lettura seguita da una scrittura, che sposta quantità di byte o parole tra i registri e la memoria.**

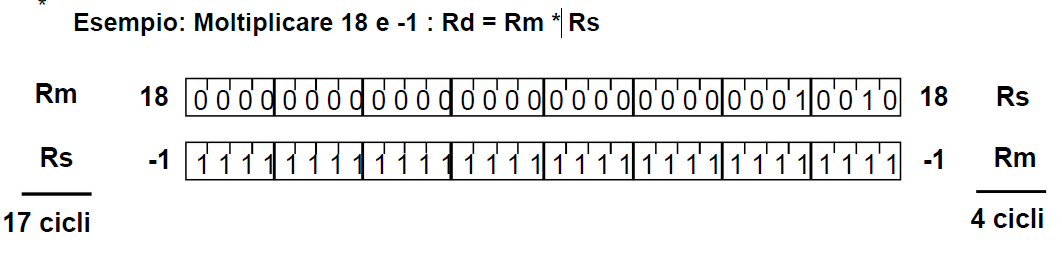
**SWP{<cond>}{B} Rd, Rm, [Rn]**

**Domanda28 : Arm moltiplicazione senza mul.**

**CON MUL**

**ARM fornisce due istruzioni di moltiplicazione MUL e MLA. Rd e Rm non possono essere lo stesso registro e può essere evitato scambiando Rm e Rs. Questo funziona perché la moltiplicazione è commutativa (NO PC).**

**ARM utilizza l'algoritmo di Booth 8 bit per eseguire la moltiplicazione di interi (usando il barrel shift). Per moltiplicare ogni coppia di bit occorre 1 ciclo (più 1 ciclo per cominciare) fino a quando non rimangono più bit 1 nel Rs, che porterà la moltiplicazione a terminare anticipatamente.**

****

**ARM contiene un hardware di moltiplicazione esteso. Ciò fornisce tre miglioramenti: la moltiplicazione viene eseguita più velocemente (il massimo per le istruzioni standard è ora di 5 cicli); usa il metodo di terminazione anticipata migliorato in modo da completare la moltiplicazione quando tutti i set di bit rimanenti contengono tutti zeri o tutti uno a seconda di cosa sia più conveniente;** è **ora possibile produrre risultati a 64 bit da due operandi a 32 bit per maggiore precisione, infatti viene usata una coppia di registri.**

**SENZA MUL si può fare con lo shift e le altre operazioni.**

**Per le moltiplicazioni spesso è possibile trovare una soluzione più ottimale utilizzando una combinazione di MOV, ADD, SUB e RSB con gli shift. Si possono fare moltiplicazioni per una costante uguale a ((potenza di 2)+- 1) in un ciclo.**

**Ad esempio ADD con LSL#2 poichè lo shift a sinistra moltiplica per potenze di 2(comp a 2).**

**Domanda22 : Bus SPI. SLIDE3**

**Il Serial Pheriferal Interconnect è un tipo di collegamento che mette in comunicazione le cpu con altri dispositivi, sia on board che off board, con pochi segnali, anche solo due.**

**I dati sono inviati un bit alla volta da un TX ad un RX. Sono collegamenti molto economici rispetto ai collegamenti paralleli, ma a parità di clock sono più lenti. Ma molto spesso però sono sufficienti a gestire comunicazioni con applicazioni vincolate da fattori esterni (come limite di bitrate o unità di tempo macroscopiche o caratteristiche del canale).**

**I bit sono organizzati in Frame composti anche da singoli byte. Nel frame è possibile aggiungere bit per il controllo errori. Questa tecnica di *framing* semplifica la sincronizzazione. Inoltre hardware specializzato può alleggerire il lavoro della cpu nella trattazione del Frame (assemblaggio, disassemblaggio, crc ..)**

**Le trasmissioni seriali possono essere di due tipi:**

**Asincrone: La trasmissione può iniziare in qualunque momento. TX e RX hanno ognuno il proprio clock che viene preconfigurato per il tick alla stessa velocità ( es 115200 ). È richiesta solo la sincronizzazione all’inizio della trasmissione. La dimensione del frame è nota sia a TX e sia a RX.**

**Sincrone: La trasmissione è sincronizzata da un clock condiviso. Non è necessaria la sincronizzazione iniziale, ma serve una linea in più per il Clock.**

**Trasmissione Asincrona: Es UART (TX e RX-clk) La linea dati viene mantenuta ALTA quando il collegamento è inattivo. Quando TX vuole trasmettere un frame, invia lo Start bit (BASSO). Il SYNC LOGIC in RX riconosce ALTO-BASSO come condizione di inizio e avvia il proprio clock.**

**Ad ogni impulso di TX CLOCK, il *serializzatore* in TX trasmette un bit del frame, contenuto nel** PARALLEL IN REGISTER, attraverso la linea SERIAL DATA. A ciascun fronte di salita di RX CLOCK, il deserializzatore in RX preleva un bit da SERIAL DATA e lo accoda nel registro PARALLEL OUT REGISTER.

Alla fine del frame il TX invia il **bit di Stop (HIGH)** e RX riconosce la fine della trasmissione. La linea SERIAL DATA viene mantenuta ALTA fino al frame successivo.

**Trasmissione Sincrona: Es USART (TX , RX , CLK) Il segnale di Clock è esplicito e condiviso da TX e RX. Il** SERIAL DATA e il CLOCK vengono mantenuti ALTI quando il collegamento è inattivo. **Quando TX vuole trasmettere avvia il proprio clock. Semplicemente ad ogni fronte di discesa del clock il TX trasmette un bit del frame su la SERIAL DATA.**

**Ad ogni fronte di salita del clock, il deserializzatore in RX esegue il campionamento a bit che accoda al PARALLEL OUT REGISTER.** Le linee SERIAL DATA e CLOCK vengono mantenute ALTE fino al frame successivo.

Esempi di SPI: UART(Asincrono), USART(Sincrono), IrDA, Modbus…

Autobus: Modbus (RS-485), Circuito interintegrato bus I²C (Philips,NXP), 1-Wire® (Dallas Semiconductor), CAN (Controller Area Network) per applicazioni automotive.

UART (universale) servono 2 segnali TX ed RX del trasmittente sono collegati a RX e TX del ricevente. Le velocità in bit variano da poche centinaia a pochi milioni al secondo.

Esistono standard elettrici come RS-232 e RS-485. (Tensioni da 3 a 15V) Questi collegamenti sono chiamati Transistor-Transistor Logic TTL

SPI è "full duplex" (invio separato e linee di ricezione) così, facoltativamente, i dati possono essere contemporaneamente trasmessi e ricevuti.

Il campionamento può essere implementato semplicemente in hardware con uno shift register o tramite bit banging.

Il collegamento seriale può essere di tre tipi: a **singolo Slave**, **MultiSlave con lo Slave Select** (SS ogni slave è indirizzato indipendentemente) o **Multi slave con Daisy-Chaning** ( a Margherita ovvero un'unica linea SS è condivisa da tutti gli slave. Una volta inviati tutti i dati, viene sollevata la linea SS, che provoca l'attivazione simultanea di tutti i chip (registri a scorrimento a catena).

**Domanda25 : Caratteristiche del secondo operando in ARM.**

**Siamo nel Barrel Shift!**

**Nelle istruzioni, dove previsto, il secondo operando può essere di 2 tipo:**

***Se il secondo operando è un registro* e il valore dello stesso oltre un immediato 5 bit senza segno, in questo caso non c’è overhead, quindi l’operazione di shift avviene nello stesso ciclo; oppure il secondo operando lo shift può essere specificato nel byte inferiore di un registro successivo (che non sia il PC ) allora servirà un ciclo in più.**

**Se non viene specificato alcun Shift, viene applicato quello predefinito: LSL #0 (nessun effetto sul valore)**

**Per le moltiplicazioni spesso è possibile trovare una soluzione più ottimale utilizzando una combinazione di MOV, ADD, SUB e RSB con gli shift. Si possono fare moltiplicazioni per una costante uguale a ((potenza di 2)+- 1) in un ciclo.**

***Se il secondo operando è un immediato* esiste una singola istruzione che caricherà una costante immediata a 32 bit in un registro senza eseguire un caricamento dati dalla memoria. Il Data Processing instruction ha 12 bit disponibili per il secondo operando.**

**Se usato direttamente questo darebbe solo un intervallo fino da 0 a 4095. Questo viene utilizzato per memorizzare costanti a 8 bit, fornendo un intervallo compreso tra 0 e 255. Questi 8 bit possono quindi essere ruotati fino a un numero pari di posizioni (cioè ROR di 0, 2, 4,..30) e ciò fornisce una gamma molto più ampia di costanti che possono essere caricate direttamente senza accedere alla memoria.**

**0 – 255 [0 - 0xff]**

**256,260,264,..,1020 [0x100-0x3fc, passaggio 4, 0x40-0xff ror 30]**

**1024,1040,1056,..,4080 [0x1000-0x3fc0, passaggio 64, 0x40-0xff ror 26]**

**4096,4160, 4224,..,16320 \* [0x400-0xff0, passaggio 16, 0x40-0xff ror 28]**

**Ad esempio MOV r0, #0x40, 26 ; => MOV r0, #0x1000 (cioè 4096)**

**Per valori superiore ci sono 2 meccanismi: I complementi bit per bit possono essere formati anche utilizzando MVN: es MOV r0, #0xFFFFFFFF ; assembla in MVN r0, #0;**

**Caricamento Di Costanti A 32bit**

**oppure l’assembler fornisce anche un metodo che caricherà QUALSIASI costante a 32 bit LDR rd,=costante numerica e se la costante può essere costruita utilizzando un MOV o MVN l’assembler genera questa in caso contrario, l’assembler produrrà un'istruzione LDR con un indirizzo relativo al PC per leggere la costante da un pool letterale**

**LDR r0,=0x42 ; genera MOV r0,#0x42**

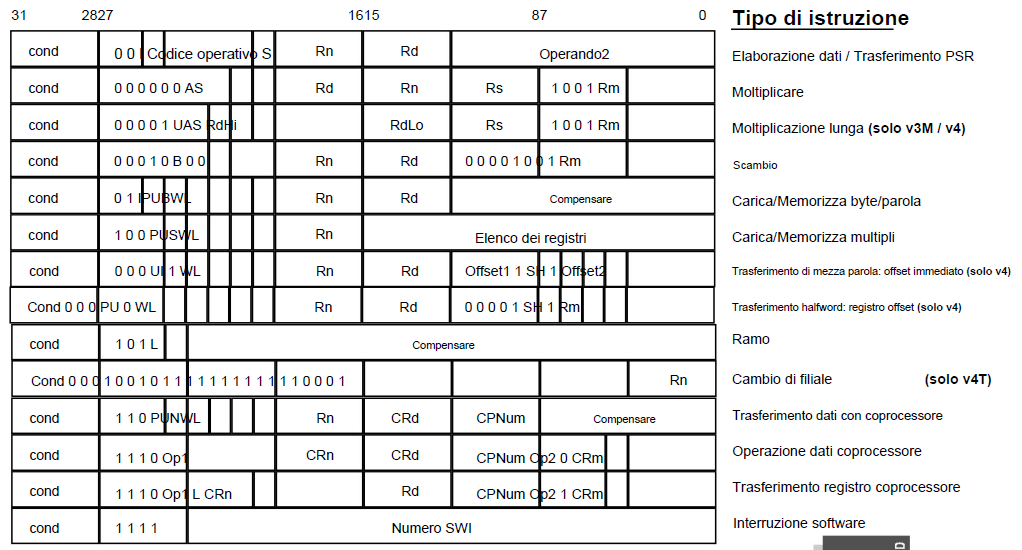
**LDR r0,=0x55555555 ; genera LDR r0,[pc, offset a pool illuminato]**

**Domanda26 : Formato istruzione ARM. SLIDEARM**

**In ARM32 tutte le istruzioni sono lunghe 32 bit. Tutte le istruzioni devono essere allineate a WORD.**

**I tipi di dati sulle quali le istruzioni assembly ARM operano sono:**

**Byte 1 byte, 8 bit; Half word 2 byte, 16 bit; Full word o word 4 byte, 32 bit.**

**Abbiamo visto 14 tipi di istruzioni arm: **

**Per creare una istruzione condizionata è semplicemente necessario postfissare la condizione.**

**Tutte le istruzioni contengono un campo condizione (i bit dal 31 al 28). Le istruzioni non è eseguite assorbono 1 ciclo. Il bit 24 è il Bit di BRANCH ( 0 Branch, 1 Branch con Link ). La differenza tra l'istruzione branch e l'indirizzo target deve essere minimo 8 (per consentire la pipeline). Il bit 26 shiftato a destra di 2 bits.**

**Domanda28 : ISA.**

**L'instruction set, in informatica ed elettronica, è l'insieme di istruzioni macchina che descrive quegli aspetti, visibili a basso livello al programmatore, dell'architettura di un calcolatore, definita in inglese come *instruction set architecture* o in acronimo ISA.**

**Domanda32 : Hazard.**

**Il pipelining è il primo passo nella ricerca di elevate prestazioni di elaborazione. La velocità effettiva è aumentata in media, ma anche la latenza.**

**Al fine di risolvere le bolle nella pipeline esistono dei meccanismi:**

**Out-of-order Execution - Esecuzione fuori ordine: viene fornito hardware che rileva un pericolo, ma invece di ritardare semplicemente l'esecuzione di B; procede a recuperare C e se C non legge i registri scritti da A o B e non scrive i registri letti da B, avverrà l'esecuzione di C prima di B. Ciò riduce ulteriormente il numero di bolle della pipeline.**

**Addressing Control Hazard: *Delay Branch* documenta semplicemente il fatto che il ramo verrà preso un certo numero di cicli dopo che è stato incontrato, e lascia al programmatore (o compilatore) il compito di assicurarsi che le istruzioni che seguono l'istruzione del brach condizionale siano innocue (come no-ops) o facciano qualche operazione utile che non dipende da quel branch;**

***Interlocks*: hardware per inserire le bolle nella pipeline secondo necessità, proprio come il Data Hazard;**

***Speculative Execution* è l’hardware a stimare se è probabile che il branch venga preso e inizia a eseguire le istruzioni che prevede di eseguire. Se la sua aspettativa non viene soddisfatta, annulla tutti gli effetti collaterali (come le scritture dei registri) causati dalle istruzioni quindi *eseguite in modo speculativo*.**

**Data and Control Hazard: *L'analisi dei tempi* di un programma può diventare estremamente difficile quando c'è una profonda pipeline con branch elaborati e speculazioni; *Le pipeline esplicite* sono relativamente comuni nei processori DSP, e sono spesso applicate in contesti in cui è essenziale una tempistica precisa; *Speculative execution* invece sono comuni nei processori General Purpose, dove i tempi sono importanti solo in senso aggregato.**

**Un progettista di sistemi embedded deve comprendere i requisiti dell'applicazione ed evitare processori in cui il livello di precisione temporale richiesto è irraggiungibile.**

**Domanda29 : Interruzione software (SWI)**

**Campo Condizione 1111.**

**In effetti, un SWI è un'istruzione definita dall'utente. Provoca una trap di eccezione al vettore hardware SWI che fa passare alla modalità supervisore, più il salvataggio dello stato associato.**

**Utilizzando il meccanismo SWI, un sistema operativo può implementare un insieme di operazioni privilegiate che le applicazioni in esecuzione in modalità utente possono richiedere.**

**Domanda27 : Pull up e down**

**Entrambe le resistenze di pull-up e pull-down sono usate nei circuiti logici elettronici per garantire che gli ingressi di un sistema logico stabilito siano a livelli logici previsti se i dispositivi esterni sono scollegati o ad alta impedenza.L'idea di una tale resistenza è strettamente correlata al concetto di logica tri-state. Questa debole resistenza "tira" la tensione del filo senza tensione di origine ed è collegata verso il suo livello di tensione di origine (come ad esempio i 5V in caso di un resistore di pull-up, o basso 0V come nel caso della resistenza di pull-down) quando gli altri componenti della linea sono inattivi.**

**Domanda31 : Filtro FIR**

**I DSP sono tipicamente macchine CISC e includono istruzioni che supportano specificamente il filtraggio FIR e spesso altri algoritmi come FFT (fast trasformate di Fourier) e la decodifica di Viterbi. Per qualificarsi come DSP, un processore deve essere in grado di eseguire FIR filtraggio in un ciclo di istruzioni per tap (campione).**

**Ad esempio il DSP TMS320c54x di Texas Instruments è concepito per essere utilizzato in applicazioni embedded con limitazioni di potenza che richiedono elevate prestazioni di elaborazione del segnale.**

**Il ciclo interno di un calcolo FIR è:**

1. **RPT numero di rubinetti – 1 (ciclo zero overhead)**
2. **2. MAC \*AR2+, \*AR3+, A**

**a := a + x \* y**

**I registri AR2 e AR3 possono essere allestiti per implementare buffer circolari.**

**Il processore c54x include una sezione di memoria su chip che supporta due accessi in un unico ciclo, e fintanto che gli indirizzi si riferiscono a questa sezione della memoria, l'istruzione MAC verrà eseguita in un unico ciclo.**

**Ogni ciclo, il processore esegue due recuperi di memoria, una moltiplicazione, un'addizione e due incrementi di indirizzo.**

**Il tempo per eseguire un filtro FIR ora si riduce a 1/2 ciclo per tap!**

**Domanda32 : Architettura multi core.**

**Una macchina multicore è una combinazione di più processori su un singolo chip. Macchine multicore eterogenee combinano una varietà di tipi di processore su un singolo chip, rispetto a più istanze dello stesso tipo di processore.**

**Per le applicazioni embedded, le architetture multicore presentano un potenziale vantaggio significativo su architetture single-core perché le attività in tempo reale e critiche per la sicurezza possono avere una funzione dedicata processore.**

**Questo è il motivo delle architetture eterogenee utilizzate per i telefoni cellulari, poiché le funzioni di elaborazione radio e vocale sono funzioni hard real-time con notevole carico computazionale.**

**Questa architettura può vantare cache multilivello, dove il secondo livello o superiore la cache è condivisa tra i core.**

**Gli FPGA sono chip la cui funzione hardware è programmabile utilizzando strumenti di progettazione hardware. Sfortunatamente, tale condivisione rende molto difficile isolare il comportamento in tempo reale del Architetture multicore.**

**I soft core sono processori implementati su FPGA. Il vantaggio dei soft core è che possono essere accoppiati strettamente all'hardware personalizzato più facilmente rispetto ai processori standard.**

**Domanda33 : Bit Banding.**

**Una regione bit-band mappa ogni parola in una regione alias bit-band su un singolo bit nella bit-band**

**regione. Le regioni a banda di bit occupano l'1 Mbyte più basso della SRAM e delle regioni di memoria periferica.**

**La mappa di memoria ha due regioni alias da 32 Mbyte che mappano a due regioni bit-band da 1 Mbyte.**

**Accede alla mappa della regione alias SRAM da 32 Mbyte alla regione bit-band SRAM da 1 Mbyte.**

**Accede alla mappa della regione alias periferica da 32 MB alla regione bit-band periferica da 1 MB**

**NUCLEO**

**Memoria di programma, memoria dati, registri e porte I/O**

**sono organizzati all'interno dello stesso indirizzo lineare di 4 Gbyte**

**di spazio**

**I byte sono codificati in memoria in formato Little Endian.**

**Il byte con il numero più basso in una parola è considerato il**

**byte meno significativo della parola e il numero più alto**

byte il più significativo.

Lo spazio di memoria indirizzabile è suddiviso in 8 principali

**blocchi, di 512 Mbyte ciascuno.**

**Tutte le aree di memoria che non sono allocate su chip**