Introduzione

La stragrande maggioranza dei computer in uso è poco visibile. Alcuni esempi:

* far funzionare il motore, i freni, le cinture di sicurezza, l'airbag e l'impianto audio dell'automobile;
* codificare digitalmente la voce e costruire un segnale radio per inviarlo dal telefono cellulare a una stazione base;
* gestire stampanti;
* comandare i robot in una fabbrica, la produzione di energia in una centrale elettrica, i processi in un impianto chimico e i semafori in una città;
* controllare aerei e treni;
* eseguire molti altri compiti "invisibili".

Questi computer meno visibili sono chiamati **sistemi embedded** e il software che eseguono è chiamato **software embedded**.

Sebbene i sistemi embedded siano in uso fin dagli anni '70, per la maggior parte della loro storia sono stati visti semplicemente come piccoli computer. Il problema ingegneristico principale era quello di far fronte a risorse limitate (potenza di elaborazione limitata, fonti di energia limitate, memorie piccole, ecc.). La sfida ingegneristica consisteva nell'ottimizzare i progetti. Poiché tutti i progetti traggono vantaggio dall’ottimizzazione, la disciplina non si distingueva da qualsiasi altro aspetto dell'informatica. Doveva solo essere più aggressiva nell'applicare le stesse tecniche di ottimizzazione.

Di recente, la comunità ha capito che le sfide principali dei sistemi embedded derivano dalla loro interazione con i processi fisici e non dalle loro risorse limitate. Il termine **sistemi cyber-fisici** (CPS) è stato coniato per riferirsi all'integrazione del calcolo con i processi fisici.

Nei CPS, i computer e le reti integrate monitorano e controllano i processi fisici, di solito con anelli di retroazione in cui i processi fisici influenzano i calcoli e viceversa. La progettazione di tali sistemi, quindi, richiede la comprensione delle dinamiche congiunte di computer, software, reti e processi fisici. È proprio lo studio della dinamica congiunta che contraddistingue questa disciplina.

Quando si studiano le CPS, emergono alcuni problemi chiave che sono rari nel cosiddetto **calcolo general-purpose**. Ad esempio, nel software di uso generale, il tempo necessario per eseguire un'attività è un problema di prestazioni, non di correttezza. Non è scorretto impiegare più tempo per svolgere un compito. È semplicemente meno e quindi meno prezioso. In CPS, il tempo necessario per eseguire un'attività può essere critico per il corretto funzionamento del sistema.

I processi fisici sono composizioni di molte cose che avvengono contemporaneamente, a differenza dei processi software, che sono profondamente radicati in passi sequenziali (procedurali). Nel mondo fisico, invece, i processi sono raramente procedurali.

I processi fisici sono composizioni di molti processi paralleli. Misurare e controllare la dinamica di questi processi orchestrando le azioni che li influenzano sono i compiti principali dei sistemi embedded. Di conseguenza, la concorrenza è intrinseca alle CPS. Molte delle sfide tecniche nella progettazione e nell'analisi del software embedded derivano dalla necessità di collegare una semantica intrinsecamente sequenziale con un mondo fisico intrinsecamente concorrente.

I meccanismi con cui il software interagisce con il mondo fisico stanno cambiando rapidamente. Oggi la tendenza è verso sensori e attuatori "intelligenti", dotati di microprocessori, interfacce di rete e software che consentono l'accesso remoto ai dati del sensore e l'attivazione remota dell'attuatore. Chiamato in vari modi, Internet of Things (IoT), Industry 4.0, the Industrial Internet, Machine-to-Machine (M2M), the Internet of Everything, the Smarter Planet, TSensors (Trillion Sensors) or The Fog , la visione è quella di una tecnologia che connette profondamente il nostro mondo fisico con il mondo dell'informazione. Nel mondo IoT, le interfacce tra questi mondi sono ispirate e derivate dall'informatica, in particolare dalla tecnologia web. Le interfacce IoT sono comode, ma non ancora adatte a interazioni strette tra i due mondi, in particolare per il controllo in tempo reale e per i sistemi critici per la sicurezza. Le interazioni strette richiedono ancora una progettazione tecnicamente complessa e di basso livello.

I progettisti di software embedded sono costretti a confrontarsi con controllori di interrupt, architetture di memoria, programmazione a livello assembly (per sfruttare istruzioni specializzate o per controllare con precisione i tempi), progettazione di driver di dispositivi, interfacce di rete e strategie di schedulazione, piuttosto che concentrarsi sulla specificazione del comportamento desiderato.

Nell'informatica general-purpose, la varietà di instruction set architectures è oggi limitata, con l'architettura Intel x86 di Intel che domina su tutte le architetture. Non esiste una tale predominanza nell'informatica embedded.

Quando vengono installati in un prodotto, i processori embedded hanno in genere una funzione dedicata. Non sono chiamati a svolgere funzioni arbitrarie con software definito dall'utente. Di conseguenza, i processori possono essere più specializzati. Vantaggi:

* possono consumare molta meno energia, e di conseguenza essere utilizzabili con piccole batterie per lunghi periodi di tempo;
* possono includere hardware specializzato per eseguire operazioni che sarebbero costose da eseguire su un hardware generico, come ad esempio l'analisi delle immagini.

Quando si valutano i processori, è importante capire la differenza tra:

* **Instruction set architecture** (ISA): le istruzioni che il processore può eseguire e alcuni vincoli strutturali (come la dimensione delle parole). x86 è una ISA.
* **Una realizzazione di processore o un chip**: un pezzo di silicio venduto da un fornitore di semiconduttori. Esistono molte realizzazioni dell'ISA x86.

Un'ISA è un'astrazione condivisa da molte realizzazioni. Una singola ISA può apparire in molti chip diversi, spesso realizzati da produttori diversi e spesso con profili di prestazioni molto diversi.

Il vantaggio di condividere un'ISA in una famiglia di processori è che gli strumenti software, che sono costosi da sviluppare, possono essere condivisi, e (a volte) gli stessi programmi possono essere eseguiti correttamente su più realizzazioni. Quest'ultima proprietà, tuttavia, è piuttosto insidiosa, dal momento che un'ISA non include normalmente alcun vincolo sulla tempistica. Quindi, sebbene un programma possa essere eseguito logicamente allo stesso modo su più chip, il comportamento del sistema può essere radicalmente diverso quando il processore è incorporato in un sistema cyber-fisico.

Data l'enorme varietà di applicazioni embedded, esiste una grande varietà di processori utilizzati. Si va da dispositivi molto piccoli, lenti, economici e a bassa potenza, a dispositivi ad alte prestazioni e per scopi speciali.

Un **microcontrollore** (μC) è un piccolo computer su un singolo circuito integrato che consiste in un'unità di elaborazione centrale (CPU) relativamente semplice combinata con dispositivi periferici, come ad esempio memorie, dispositivi di I/O e timer.

Più della metà di tutte le CPU vendute nel mondo sono microcontrollori, anche se tale affermazione è difficile da dimostrare. I microcontrollori più semplici funzionano con parole a 8 bit e sono adatti per applicazioni che richiedono piccole quantità di memoria e semplici funzioni logiche (rispetto a funzioni aritmetiche ad alte prestazioni). Possono consumare quantità di energia estremamente ridotte e spesso includono una modalità di sospensione che riduce il consumo di energia a nanowatt. È stato dimostrato che esistono nodi di reti di sensori e dispositivi di sorveglianza, in grado di funzionare con una piccola quantità di energia.

I microcontrollori possono essere molto elaborati. L'**Intel Atom**, ad esempio, è una famiglia di CPU x86 utilizzata principalmente nei netbook e in altri piccoli computer portatili. Questi processori sono stati progettati per consumare relativamente poca energia senza perdere troppe prestazioni rispetto ai processori utilizzati in computer di fascia più alta. Sono adatti per alcune applicazioni embedded e per i server in cui il raffreddamento è problematico. **Geode** di AMD è un altro esempio di processore che si trova vicino al labile confine tra i processori di uso generale e i microcontrollori.

Il Motorola 6800 e l'Intel 8080 sono microcontrollori a 8-bit apparsi sul mercato nel 1974. I discendenti di queste architetture sopravvivono oggi, ad esempio sotto forma di Freescale 6811.

Lo Zilog Z80 è un discendente pienamente compatibile dell'8080, che è diventato uno dei microcontrollori più prodotti e utilizzati di tutti i tempi. Un derivato dello Z80 è il Rabbit 2000, progettato da Rabbit Semiconductor.

L'Intel 8051 è un microcontrollore a 8 bit sviluppato da Intel nel 1980. L'ISA dell'8051 è oggi supportato da molti fornitori, tra cui Atmel, Infineon Technologies, Dallas Semiconductor, NXP, ST Microelectronics, Texas Instruments, e Cypress Semiconductor.

Il microcontrollore Atmel AVR a 8 bit, sviluppato da Atmel nel 1996, è stato uno dei primi microcontrollori a utilizzare la memoria flash on-chip per la memorizzazione dei programmi. Sebbene Atmel affermi che AVR non è un acronimo, si ritiene che l'architettura sia stata progettata da due studenti dell'Istituto norvegese di tecnologia, Alf-Egil Bogen e Vegard Wollan, quindi potrebbe essere nata come come RISC di Alf e Vegard.

Molti microcontrollori a 32 bit implementano una qualche variante del set di istruzioni ARM, sviluppato da ARM Limited. ARM era originariamente l'acronimo di Advanced RISC Machine e prima ancora di Acorn RISC Machine ma oggi è semplicemente ARM. I processori che implementano l'ISA ARM sono ampiamente utilizzati nei telefoni cellulari per realizzare le funzioni dell'interfaccia utente e in molti altri sistemi embedded. I produttori di semiconduttori concedono in licenza il set di istruzioni ad ARM Limited e producono i propri chip. Altre architetture di microcontrollori embedded degne di nota sono:

* il Motorola ColdFire (poi Freescale ColdFire)
* Hitachi H8 e SuperH
* il MIPS (originariamente sviluppato da un team guidato da John Hennessy dell'Università di Stanford)
* il PIC (originariamente Programmable Interface Controller, di Microchip Technology)
* il PowerPC (creato nel 1991 da un'alleanza tra Apple, IBM e Motorola)
* RISC-V (Open Source)

**Reduced Instruction Set Computer** (in [acronimo](https://it.wikipedia.org/wiki/Acronimo) **RISC**), nell'[elettronica digitale](https://it.wikipedia.org/wiki/Elettronica_digitale), indica un'idea di [progettazione](https://it.wikipedia.org/wiki/Progettazione) di architetture per [microprocessori](https://it.wikipedia.org/wiki/Microprocessore) 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](https://it.wikipedia.org/wiki/Set_di_istruzioni) in tempi minori rispetto a una architettura [CISC](https://it.wikipedia.org/wiki/CISC).

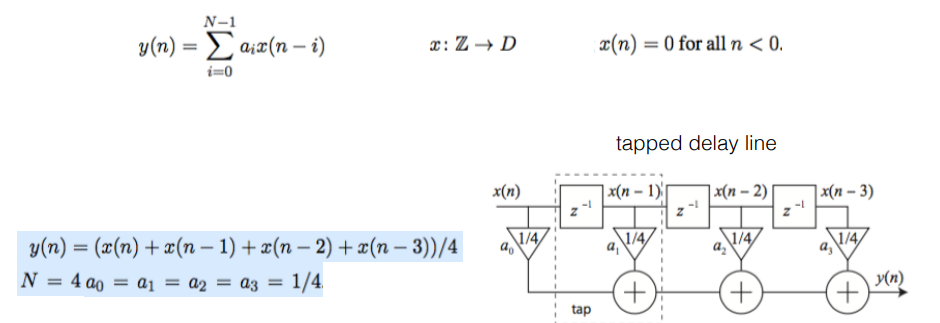
**Complex Instruction Set Computer** (**CISC**), in [elettronica digitale](https://it.wikipedia.org/wiki/Elettronica_digitale), indica un'architettura per [microprocessori](https://it.wikipedia.org/wiki/Microprocessore) formata da un [set di istruzioni](https://it.wikipedia.org/wiki/Set_di_istruzioni) contenente istruzioni in grado di eseguire operazioni complesse come la lettura di un dato in memoria, la sua modifica e il suo salvataggio direttamente in memoria tramite una singola istruzione. Il termine è nato per distinguere l'altro paradigma dominante nei microprocessori, il paradigma [Reduced Instruction Set Computer](https://it.wikipedia.org/wiki/Reduced_Instruction_Set_Computer) (RISC).

**Processori DSP**. Molte applicazioni embedded eseguono una buona parte dell'elaborazione dei segnali. Un segnale è un insieme di misure campionate del mondo fisico, tipicamente prese a una frequenza regolare chiamata frequenza di campionamento. Un'applicazione per il controllo del movimento, ad esempio, può leggere informazioni sulla posizione o sulla localizzazione dai sensori a frequenze di campionamento che vanno da pochi Hertz (Hz o campioni al secondo) a qualche centinaio di Hertz. I segnali audio vengono campionati a frequenze che vanno da 8.000 Hz (o 8 kHz, la frequenza di campionamento utilizzata nella telefonia per la voce) a 44,1 kHz (la frequenza di campionamento dei CD). Le applicazioni a ultrasuoni (come l'imaging medico) e le applicazioni musicali ad alte prestazioni possono campionare i segnali sonori a velocità molto più elevate.

Il video utilizza in genere frequenze di campionamento di 25 o 30 Hz per i dispositivi consumer e frequenze molto più elevate per le applicazioni di misura speciali. Ogni campione, ovviamente contiene un'intera immagine (chiamata fotogramma), che a sua volta ha molti campioni (chiamati pixel) distribuiti nello spazio e non nel tempo. Le applicazioni radio definite dal software hanno frequenze di campionamento che possono variare da centinaia di kHz (per l'elaborazione in banda base) a diversi GHz (miliardi di Hertz). Altre applicazioni embedded che fanno un uso intensivo dell'elaborazione del segnale sono quelle relative a giochi interattivi, radar, sonar e sistemi di imaging LIDAR (Light Detection and Ranging); sistemi di imaging; analisi video (l'estrazione di informazioni dai video, ad esempio per la sorveglianza), sistemi di assistenza alla guida per automobili; elettronica medica e strumentazione scientifica.

Le architetture informatiche specializzate per l'elaborazione dei segnali esistono da tempo. I microprocessori DSP a singolo chip sono comparsi all'inizio degli anni '80. Le prime applicazioni di questi dispositivi comprendevano modem di dati a banda vocale, sintesi vocale, audio per i consumatori, grafica, e controller di unità disco. Le caratteristiche principali dei DSP includono un'unità di multiply-accumulate hardware; diverse varianti dell'architettura Harvard (per supportare più fetch simultanei di dati e programmi) e modalità di indirizzamento che supportano l'incremento automatico, buffer circolari e l'indirizzamento con inversione di bit (quest'ultimo per supportare il calcolo della FFT).

La maggior parte supporta precisioni di dati a virgola fissa di 16-24 bit, in genere con accumulatori molto più ampi (40-56 bit), in modo da poter eseguire un gran numero di istruzioni successive di istruzioni di moltiplicazione-accumulazione successive senza overflow. Alcuni DSP con hardware in virgola mobile non hanno dominato il mercato. I DSP sono difficili da programmare rispetto alle architetture RISC, soprattutto a causa di istruzioni specializzate complesse, pipeline esposta al programmatore e un'architettura di memoria asimmetriche. Fino alla fine degli anni anni '90, questi dispositivi erano quasi sempre programmati in linguaggio assembly. Ancora oggi, i programmi C fanno largo uso di librerie che sono in linguaggio assembly per sfruttare le caratteristiche più esoteriche delle architetture.



Supponiamo che un filtro FIR campioni alla frequenza di 1 MHz (un milione di campioni al secondo) e che N = 32. Le uscite devono essere calcolate a una velocità di 1 MHz e ogni uscita richiede 32 moltiplicazioni e 31 addizioni. Un processore deve essere in grado di sostenere una velocità di calcolo di 63 milioni di operazioni aritmetiche al secondo per implementare questa applicazione. Per sostenere la velocità di calcolo: l'hardware aritmetico deve essere sufficientemente veloce; i meccanismi per l'inserimento e l'estrazione dei dati dalla memoria e per l'inserimento e il disinserimento dei dati dal chip devono essere sufficientemente veloci.

Un filtro FIR necessita di una linea di ritardo. Un'implementazione ingenua prevede l'allocazione di un array in memoria e ogni volta che arriva un campione in ingresso, si sposta ogni elemento dell'array nella posizione superiore per fare spazio per il nuovo elemento nella prima posizione. Ciò comporterebbe un enorme spreco di banda di memoria. Un approccio migliore è quello di utilizzare un buffer circolare.

Immagine che contiene testo, orologio

Descrizione generata automaticamente

* L'implementazione accetta prima un nuovo valore di input e poi calcola la sommatoria a ritroso, iniziando dal termine i=N-1, nell’esempio N=8.
* Quando arriva l’n-esimo input, il valore di p è pi compreso {0,1,…7}.
* Il programma scrive il nuovo input x(n) nella locazione data da p e incrementa p calcolando p = pi+1.
* Tutti i calcoli aritmetici su vengono eseguiti con modulo 8.
* Il calcolo del filtro FIR viene quindi letto dalla posizione x(n-7) dalla locazione p = pi+1 e moltiplicato per a7.
* Il risultato viene memorizzato in un registro di accumulo (precedentemente azzerato). Il registro p si incrementa nuovamente di uno, settando p = pi+2.
* Successivamente legge x(n-6) dalla locazione p = pi+2, la moltiplica per a6, e aggiunge il risultato all'accumulatore.
* Continua fino a leggere x(n) dalla locazione p = pi+8, la stessa in cui è stato scritto l'ultimo ingresso x(n), e moltiplica il valore per a0 , e aggiunge il risultato all'accumulatore.
* Incrementa p, ponendo p = pi+9= pi+1
* Al termine di questa operazione, il valore di p è pi+1 , che indica la posizione in cui deve essere scritto il prossimo ingresso x(n+1).

Un'immagine può essere modellata in modo analogo come una funzione. Un pixel (o elemento dell'immagine) è un campione che rappresenta il colore e l'intensità di un punto in un'immagine. Esistono molti modi per farlo, ma tutti utilizzano uno o più valori numerici per ogni pixel. I set e dipendono dalla risoluzione dell'immagine.

ATSC è un insieme di standard sviluppati dall'Advanced Television Systems Committee. DVB-T è lo standard europeo . L'ATSC supporta una serie di frame rate che vanno da poco meno di 24 Hz a 60 Hz e una serie di risoluzioni. Il video ad alta definizione secondo lo standard ATSC supporta, ad esempio, una risoluzione di 1080 per 1920 pixel a una frequenza di fotogrammi di 30 Hz. Questa risoluzione è chiamata 1080p nel settore. Oggi le apparecchiature video professionali raggiungono una risoluzione quattro volte superiore (4320 x 7680). La frequenza dei fotogrammi può essere molto superiore a 30 Hz. Frequenze di fotogrammi molto elevate sono utili per catturare fenomeni estremamente veloci al rallentatore.

**Programmable Logic Controllers (PLC).** Un controllore logico programmabile (PLC) è una forma specializzata di microcontrollore in un pacchetto robusto con interfacce di I/O adatto al controllo industriale. I PLC sono spesso programmati utilizzando la logica ladder, una notazione grafica originariamente utilizzata per specificare la logica costruita con relè e interruttori. Un relè è un interruttore il cui contatto è controllato da una bobina. Quando una tensione viene applicata alla bobina, il contatto si chiude, consentendo il passaggio di corrente attraverso il relè. Interconnettendo i contatti e le bobine, i relè possono essere utilizzati per costruire controllori digitali che seguono schemi specifici. Nella notazione comune, un contatto è rappresentato da due barre verticali e una bobina da un cerchio.

* Il diagramma è composto da due pioli.
* La bobina del motore sul piolo inferiore accende o spegne un motore.
* I contatti di Start e Stop rappresentano interruttori a pulsante.
* L'avvio è un contatto normalmente aperto.
* Il contatto Stop è normalmente chiuso, indicato dalla barra, cioè diventa aperto quando l'operatore preme il pulsante.
* Quando l'operatore preme Start, la corrente passa alla bobina Run, causando la chiusura di entrambi i contatti Run.
* Il motore funziona anche dopo il rilascio del pulsante di avvio.
* Quando l'operatore preme Stop, la corrente si interrompe ed entrambi i contatti di marcia si aprono, causando l'arresto del motore.
* I contatti cablati in parallelo eseguono una funzione logica OR, mentre i contatti cablati in serie eseguono una funzione logica AND.
* Il piolo superiore è dotato di retroazione; il significato del piolo è una soluzione a punto fisso dell'equazione logica implicita nel motore.

**Graphics Processors.** Un'unità di elaborazione grafica (GPU) è un processore specializzato progettato appositamente per eseguire i calcoli richiesti dal rendering grafico. I processori di questo tipo risalgono agli anni '70, quando venivano utilizzati per il rendering di testo e grafica, per combinare più modelli grafici e per disegnare rettangoli, triangoli, cerchi e archi, Le moderne GPU supportano la grafica 3D, l'ombreggiatura e il video digitale. I fornitori dominanti di GPU sono oggi Intel, NVIDIA e AMD. Alcune applicazioni embedded, in particolare i giochi, si adattano bene alle GPU. Inoltre, le GPU si sono evolute verso modelli di programmazione più generali, e quindi hanno iniziato a comparire in altre applicazioni ad alta intensità di calcolo, come la strumentazione. Le GPU sono in genere piuttosto affamate di energia e quindi oggi non sono adatte alle applicazioni embedded con vincoli energetici.

**The x86 Architecture.** ISA dominante per i computer desktop e portatili. Nasce con l'Intel 8086, un microprocessore a 16 bit progettato da Intel nel 1978. Una variante dell'8086, denominata 8088, è stata utilizzata nel PC IBM originalee da allora questa famiglia di processori ha dominato il mercato dei PC. I processori successivi di questa famiglia sono stati chiamati con nomi che terminano con "86" e in genere hanno mantenuto la retrocompatibilità. L'Intel 80386 è stata la prima versione a 32 bit di questo set di istruzioni, introdotta nel 1985. Oggi, il termine "x86" si riferisce solitamente alla versione a 32 bit, mentre le versioni a 64 bit sono denominate "x86-64". L'Intel Atom, introdotto nel 2008, è un processore x86 con un consumo energetico notevolmente ridotto. Sebbene sia destinato principalmente ai netbook e ad altri piccoli computer mobili, è un'opzione interessante anche per alcuni computer embedded. L'architettura x86 è stata anche introdotta in processori di AMD, Cyrix e di molti altri produttori.

**Parallelism.** La maggior parte dei processori oggi offre varie forme di parallelismo. Questi meccanismi influenzano fortemente la tempistica dell'esecuzione di un programma; quindi, i progettisti di sistemi embedded devono comprenderli.

**Parallelism vs. Concurrency**. La concorrenza è centrale nei sistemi embedded. Un programma è detto concorrente se diverse parti del programma vengono eseguite simultaneamente a livello concettuale. Un programma si dice parallelo se diverse parti del programma vengono eseguite fisicamente in modo simultaneo su hardware distinti (ad esempio su macchine multicore, server in una server farm o microprocessori distinti).

I programmi non concorrenti specificano una sequenza di istruzioni da eseguire. Un linguaggio di programmazione che esprime una computazione come una sequenza di operazioni è chiamato linguaggio imperativo. Il C è un linguaggio imperativo. Quando si usa il C per scrivere programmi concorrenti, bisogna uscire dal linguaggio stesso, tipicamente utilizzando una libreria di thread. Una libreria di thread utilizza strutture non fornite dal C, ma dal sistema operativo e/o dall'hardware. Java è un linguaggio per lo più imperativo esteso con costrutti che supportano direttamente i thread. Pertanto, è possibile scrivere programmi concorrenti in Java senza programmi concomitanti E senza uscire dal linguaggio. Ogni esecuzione (corretta) di un programma in un linguaggio imperativo deve comportarsi come se le istruzioni fossero eseguite esattamente nella sequenza specificata. Tuttavia, spesso è possibile eseguire le istruzioni in parallelo o in un ordine diverso da quello specificato dal programma e ottenere comunque un comportamento corrispondente all'esecuzione sequenziale.

**Dataflow analysis.** Un compilatore può analizzare le dipendenze tra le operazioni in un programma e produrre codice parallelo, se la macchina di destinazione lo supporta. Molti microprocessori oggi supportano l'esecuzione in parallelo, utilizzando flussi di istruzioni o architetture VLIW (very large instruction word). I processori con flussi di istruzioni multi-emissione possono eseguire istruzioni indipendenti simultaneamente. L'hardware analizza le istruzioni al volo per individuare eventuali dipendenze e, se non ci sono dipendenze, esegue più di un'istruzione alla volta.

Le macchine VLIW dispongono di istruzioni a livello ASSEMBLY che specificano più operazioni da eseguire insieme. In questo caso, è necessario che il compilatore produca le istruzioni parallele appropriate. L'analisi delle dipendenze viene effettuata a livello di linguaggio assembly o a livello di singole operazioni, non a livello di righe di C. Una riga di C può specificare più operazioni, o anche operazioni complesse come le chiamate a procedure. In entrambi i casi (multi-issue e VLIW), un programma imperativo viene analizzato per la concorrenza al fine di consentire l'esecuzione parallela. L'obiettivo generale è accelerare l'esecuzione del programma e le prestazioni, dove si presume che finire un compito prima sia sempre meglio che finirlo dopo.

Nel contesto dei sistemi embedded, la concorrenza ha un ruolo molto più importante del semplice miglioramento delle prestazioni. I programmi embedded interagiscono con i processi fisici e nel mondo fisico molte attività si svolgono contemporaneamente. Un programma incorporato deve spesso monitorare e reagire a più fonti simultanee di stimoli e controllare simultaneamente più uscite. I programmi embedded sono quasi sempre programmi concorrenti e la concorrenza è una parte intrinseca della logica dei programmi. Non è solo un modo per migliorare le prestazioni. In effetti, terminare un'attività in anticipo non è necessariamente meglio che terminarla in ritardo. Le azioni eseguite nel mondo fisico spesso devono essere eseguite al momento giusto (né in anticipo né in ritardo). Immaginiamo ad esempio un controllore per un motore a benzina. Accendere le candele prima non è certamente meglio che accenderle più tardi. Devono essere accese al momento giusto.

Proprio come i programmi imperativi possono essere eseguiti in modo sequenziale o in parallelo, i programmi concorrenti possono essere eseguiti in modo sequenziale o parallelo. L'esecuzione sequenziale di un programma concorrente è oggi tipicamente effettuata da un sistema operativo multitasking, che intreccia l'esecuzione di più compiti in un unico flusso sequenziale di istruzioni. Naturalmente, l'hardware può parallelizzare l'esecuzione se il processore è dotato di un'architettura architettura multi-emissione o VLIW. Quindi, un programma concorrente può essere convertito in un flusso sequenziale da un sistema operativo e di nuovo in un programma concorrente da parte dell'hardware. Queste traduzioni multiple complicano notevolmente il problema di garantire che le cose avvengano al momento giusto.

Il parallelismo nell'hardware esiste per migliorare le prestazioni di applicazioni ad alta intensità di calcolo. Dal punto di vista del programmatore, la concomitanza nasce come conseguenza dell'hardware progettato per migliorare le prestazioni, non come conseguenza del problema applicativo da risolvere. L'applicazione non richiede (necessariamente) che più attività procedano simultaneamente, ma richiede solo che le cose siano fatte molto molto velocemente.

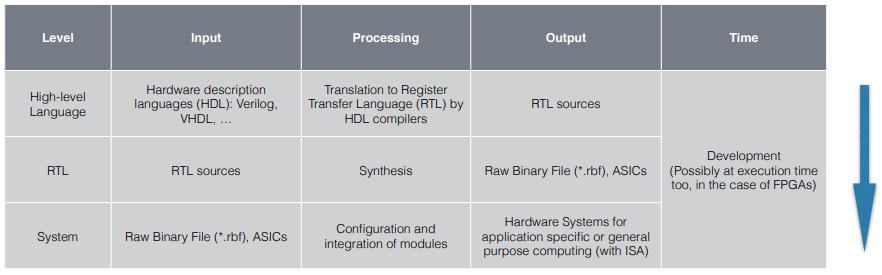
L'esecuzione del software applicativo richiede diverse fasi di elaborazione in **fase di sviluppo** e **di esecuzione**.

Immagine che contiene tavolo

Descrizione generata automaticamente

La logica digitale spesso include più unità funzionali simili (ad esempio ALU, MAC, controllori di memoria) per l'esecuzione in parallelo. Oltre al parallelismo hardware, vengono utilizzate diverse tecniche per migliorare le prestazioni nelle CPU general-purpose: pipeline lunghe, esecuzioni fuori ordine e speculative, predizione dei rami. Queste tecniche sono evitate nei processori embedded (perché?).

Il livello hardware può essere realizzato anche attraverso le specifiche software. La configurazione dell'hardware sta diventando una pratica comune nello sviluppo embedded. Esistono persino CPU di uso generale che includono hardware configurabile (FPGA). Gli HDL includono le nozioni di tempo, eventi e concorrenza nel loro formalismo.



**Pipelining. Una semplice pipeline a cinque stadi per una macchina a 32 bit.**

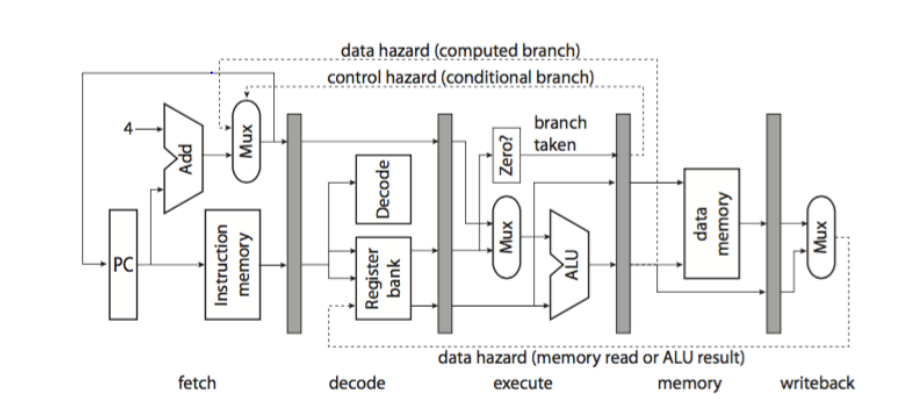


Immagine che contiene testo

Descrizione generata automaticamente

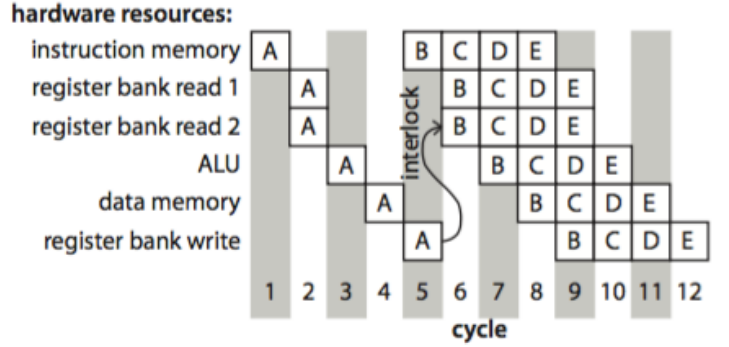
**Addressing data hazards: explicit pipeline**

Per l'esempio in cui B legge un registro scritto da A, il compilatore potrebbe, il compilatore può inserire tre istruzioni no-op (che non fanno nulla) tra A e B per garantire che la scrittura avvenga prima della lettura Queste istruzioni no-op formano una bolla della pipeline che si propaga.

**Addressing data hazards: Interlocks**

L'hardware di decodifica delle istruzioni, quando incontra l'istruzione B che legge un registro scritto da A, rileverà il pericolo e ritarderà l'esecuzione di B finché A non avrà completato la fase di writeback. Per questa pipeline, B dovrebbe essere ritardato di tre cicli di clock per consentire ad A di completare la fase di writeback. Questo può essere ridotto a due cicli se si prevede una logica di inoltro un po' più complessa, in grado di rilevare che A sta scrivendo la stessa posizione che B sta leggendo, e fornisce direttamente i dati invece di richiedere che la scrittura avvenga prima della lettura.

Gli interlocks forniscono quindi in hardware un sistema che inserisce automaticamente le bolle nella pipeline.



**Addressing data hazards: Out-of-order execution.**

Esecuzione fuori ordine: viene fornito un hardware che rileva un pericolo, ma invece di ritardare semplicemente l'esecuzione di B, procede al fetch di C, e se C non legge i registri scritti da A o B, e non scrive registri letti da B, procede all'esecuzione di Cprima di B. Questo riduce ulteriormente il numero di bolle della pipeline.

**Control hazards.**

Un'istruzione di diramazione condizionale cambia il valore del PC se un registro specificato ha valore zero. Il nuovo valore del PC è fornito (facoltativamente) dal risultato di un'operazione ALU. Se A è un'istruzione di diramazione condizionale, allora A deve aver raggiunto lo stadio di memoria prima che il PC possa essere aggiornato. Le istruzioni che seguono A in memoria saranno state recuperate e si troveranno già negli stadi di decodifica e di esecuzione nel momento in cui si determina che tali istruzioni non devono essere eseguite.

**Addressing control hazards:**

* **Delayed branch:** documenta semplicemente il fatto che il ramo sarà preso un certo numero di cicli dopo averla incontrata, e lascia al programmatore (o al compilatore) di assicurarsi che le istruzioni che seguono il ramo di diramazione condizionale siano innocue (come i no-op) o che svolgano un lavoro utile che non dipende dal fatto che il ramo venga eseguito o meno.
* **Interlocks:** hardware per inserire bolle di pipeline secondo le necessità, proprio come nel caso di pericoli dei dati.
* **Speculative execution:** L'hardware valuta se è probabile che il ramo debba essere eseguito e inizia a eseguire le istruzioni che si aspetta di eseguire. Se non viene soddisfatta la condizione, annulla tutti gli effetti collaterali (come la scrittura di registri) che le istruzioni eseguite in modo speculativo hanno causato.

**Data and Control hazards:**

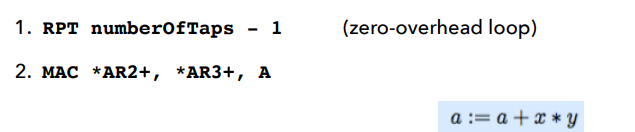
Tranne che per le pipeline esplicite e le ramificazioni ritardate, tutte le tecniche introducono una variabilità nei tempi di esecuzione di una sequenza di istruzioni. L'analisi dei tempi di un programma può diventare estremamente difficile. Le pipeline esplicite sono relativamente comuni nei processori DSP, che sono spesso applicati in contesti in cui la tempistica è essenziale. L'esecuzione fuori ordine e speculativa è comune nei processori general-purpose, dove la tempistica è importante dove la tempistica è importante solo in senso aggregato. Il progettista di un sistema embedded deve comprendere i requisiti dell'applicazione ed evitare i processori in cui il livello di precisione temporale è irraggiungibile.

**Il parallelismo a livello di istruzioni** può essere implementato come:

* ***CISC instructions***
* ***subword parallelism***
* ***superscalar***
* ***VLIW***

Un processore con istruzioni complesse (e tipicamente piuttosto specializzate) è chiamato macchina **CISC** (complex instruction set computer). La filosofia alla base di questi processori è nettamente diversa da quella delle macchine RISC (reduced instruction set computer). I DSP sono tipicamente macchine CISC, e includono istruzioni che supportano specificamente il filtraggio FIR e spesso altri algoritmi come le FFT (fast Fourier) e la decodifica Viterbi. In effetti, per qualificarsi come DSP, un processore deve essere in grado di eseguire il filtraggio FIR in un ciclo di istruzioni per tap.

La famiglia di processori DSP TMS320c54x di Texas Instruments è destinata all'utilizzo in applicazioni embedded a basso consumo energetico che richiedono elevate prestazioni di elaborazione del segnale, come i sistemi di comunicazione wireless e gli assistenti digitali personali (PDA). L'anello interno di un calcolo FIR è :



I registri AR2 e AR3 possono essere impostati per implementare buffer circolari. Il processore c54x include una sezione di memoria on-chip che supporta due accessi in un singolo ciclo e finché gli indirizzi si riferiscono a questa sezione di memoria, l'istruzione MAC verrà eseguita in un singolo ciclo. Ad ogni ciclo, il processore esegue due fetches di memoria, una moltiplicazione, un'addizione e due incrementi di indirizzo (eventualmente modulo).

Se i coefficienti del filtro FIR sono simmetrici:

▸ N è pari e ai = aN-i-1 .

▸ Il numero di moltiplicazioni può essere ridotto riscrivendo la formula come:

Immagine che contiene testo

Descrizione generata automaticamente

Il set di istruzioni TMS320c54x di Texas Instruments include un'istruzione FIRS che funziona in modo simile all'istruzione al MAC, ma con questo calcolo. Questo sfrutta il fatto che il c54x dispone di due ALU e quindi può eseguire il doppio delle addizioni rispetto alle moltiplicazioni. Il tempo di esecuzione di un filtro FIR si riduce ora a 1/2 ciclo per tap.

**Subword parallelism.**