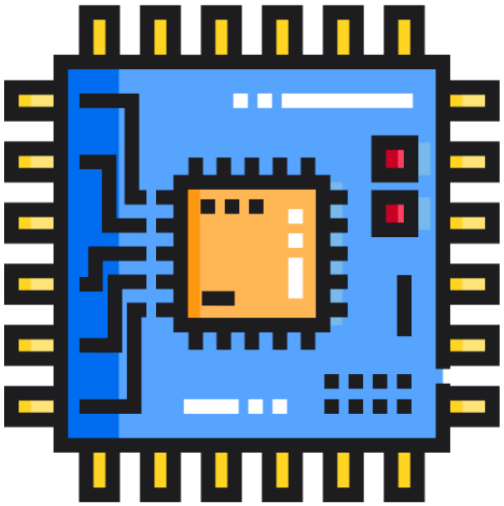
Università di Bologna

Estensioni per la virtualizzazione su RISC-V



Studente :
Vincenzo Salvemini

Docente :
Andrea Bartolini

Indice

Introduzione

La virtualizzazione è una tecnologia chiave per l'ottimizzazione delle risorse hardware, in quanto consente l'esecuzione simultanea di più ambienti operativi isolati su un'unica piattaforma fisica. Questo è reso possibile da software specializzati chiamati **hypervisor**, che si distinguono in due categorie principali: Tipo 1, eseguiti direttamente sull'hardware, e Tipo 2, che operano sopra un sistema operativo. Gli hypervisor permettono una gestione più efficiente, sicura e flessibile delle macchine virtuali.

In questo approfondimento si pone particolare attenzione all'architettura **RISC-V**, uno standard open-source, scalabile e altamente personalizzabile, che ha introdotto specifiche estensioni per la virtualizzazione. Queste estensioni permettono di eseguire più sistemi operativi isolati su un singolo processore, incrementando così la sicurezza e le prestazioni complessive.

Tra i concetti fondamentali trattati vi è la **Two-Stage Address Translation**, una tecnica avanzata di gestione della memoria che consente agli hypervisor di controllare in modo efficiente l'allocazione e la protezione della memoria tra le diverse macchine virtuali. Verranno inoltre analizzati i registri CSR dedicati all'hypervisor (**Hypervisor CSRs**) e le relative **istruzioni specializzate per la virtualizzazione**.

Un focus specifico è riservato al core **CVA6**, un'implementazione RISC-V ottimizzata per ambienti virtualizzati. In particolare, si esaminano le sue caratteristiche architetturali come il **GTLB** e il **Level-2 TLB**, entrambi progettati per ridurre la latenza nella traduzione degli indirizzi e migliorare le prestazioni complessive del sistema.

Infine, viene analizzata l'integrazione dell'hypervisor **Xen** sulla piattaforma RISC-V. Xen è una soluzione open-source per la gestione di sistemi operativi virtualizzati che, una volta portata su RISC-V, consentirà di ottenere una piattaforma completamente open, modulare e scalabile. Il porting di Xen rappresenta un traguardo significativo, soprattutto in ambiti come il **cloud computing** e i **sistemi embedded**, aprendo nuove prospettive per l'adozione di tecnologie completamente libere a livello hardware e software.

1 Virtualizzazione

1.1 Importanza della virtualizzazione

La virtualizzazione è una tecnologia che consente l'astrazione delle risorse hardware, permettendo l'esecuzione di più ambienti operativi isolati su un'unica piattaforma fisica. Attraverso la virtualizzazione, è possibile migliorare l'efficienza dell'hardware, aumentare la sicurezza e semplificare la gestione dei sistemi informatici. Di seguito vediamo alcuni vantaggi della virtualizzazione:

- **Ottimizzazione delle risorse**: Consente un utilizzo più efficiente dell'hardware, riducendo il numero di server fisici necessari.
- **Isolamento e Sicurezza**: Ogni macchina virtuale opera in un ambiente isolato, impedendo che guasti o attacchi su una VM influiscano sulle altre.
- **Portabilità e Disaster recovery**: Le VM possono essere facilmente migrate tra server diversi e ripristinate in caso di guasto hardware.
- **Semplificazione della gestione**: Con strumenti avanzati di orchestrazione e automazione, la gestione dell'infrastruttura virtualizzata diventa più semplice e centralizzata.

1.2 Virtual Machine Monitor

L'implementazione della virtualizzazione richiede un hypervisor, noto anche come **Virtual Machine Monitor**, che gestisce la creazione e l'esecuzione delle VM. Il loro obiettivo è fornire a ciascuna VM un ambiente isolato, eseguendo più sistemi operativi simultaneamente sullo stesso hardware.

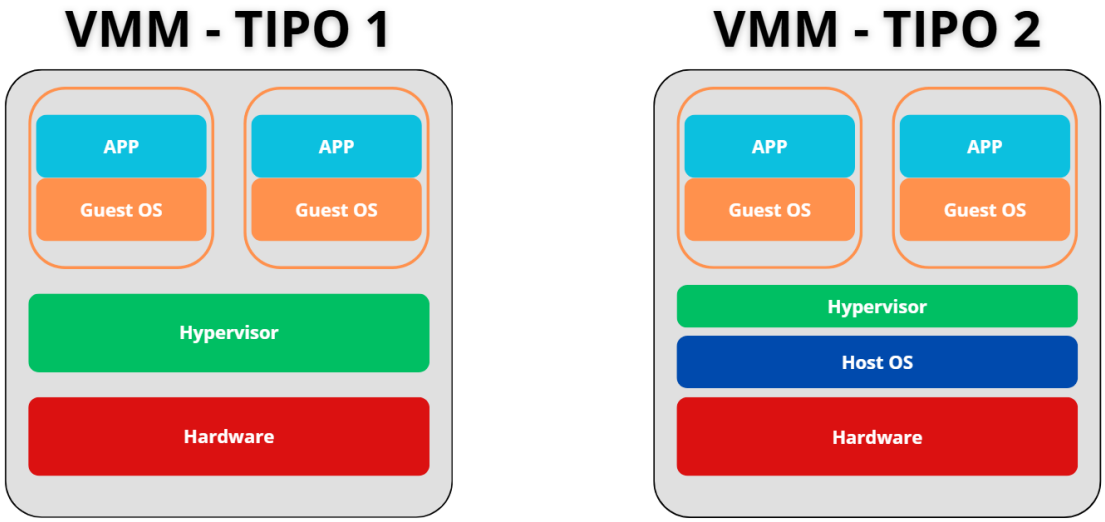


Figura 1: VMM tipo 1 e 2

1.2.1 Hypervisor di Tipo 1 (Bare Metal)

Gli **hypervisor di Tipo 1** operano direttamente sull'hardware fisico del sistema senza necessitare di un sistema operativo host. Questo tipo di hypervisor gestisce direttamente CPU, memoria, dispositivi di I/O e altre risorse, allocandole in modo efficiente alle VM. Alcuni esempi possono essere: Xen, VMware ESXi, Microsoft Hyper-V, KVM.

Vantaggi	Svantaggi
Prestazioni elevate grazie all'accesso diretto alle risorse hardware.	Maggiore complessità di implementazione.
Isolamento e sicurezza migliori tra le VM.	Richiede estensioni di virtualizzazione nella CPU.

Tabella 1: Vantaggi e svantaggi degli hypervisor di Tipo 1

1.2.2 Hypervisor di Tipo 2 (Hosted)

Gli **hypervisor di Tipo 2** vengono eseguiti sopra un sistema operativo host esistente, funzionando come un'applicazione. Il sistema operativo host gestisce direttamente l'hardware e l'hypervisor si affida ai servizi dell'OS per la gestione delle risorse. Alcuni esempi possono essere: VMware Workstation, Oracle.

Vantaggi	Svantaggi
Maggiore flessibilità, può essere installato senza modificare l'hardware o il sistema operativo host.	Prestazioni inferiori rispetto agli hypervisor di Tipo 1, a causa della dipendenza dal sistema operativo host.
Facile da configurare e usare, ottimo per test, sviluppo e ambienti desktop.	Maggiore overhead dovuto all'ulteriore livello di astrazione.

Tabella 2: Vantaggi e svantaggi degli hypervisor di Tipo 2

1.3 RISC-V e virtualizzazione

RISC-V è uno standard di istruzioni aperto e modulare, progettato per essere scalabile ed efficiente in una vasta gamma di applicazioni, dai microcontrollori ai sistemi ad alte prestazioni. Essendo open-source consente ai progettisti di hardware di personalizzare e ottimizzare l'architettura senza vincoli, rendendola una scelta sempre più adottata nel settore dell'informatica embedded, mobile e cloud.
RISC-V ha introdotto estensioni specifiche, come l'**Hypervisor Extension**, che consente l'esecuzione di più sistemi operativi isolati su un'unica CPU. Questa estensione aggiunge livelli di privilegio dedicati agli hypervisor, migliorando la sicurezza e l'efficienza nella gestione delle macchine virtuali.

2 RISC-V

2.1 Introduzione

RISC-V nasce nell' univeristà della California, Berkeley, con l'obiettivo di creare un'architettura libera da licenze proprietarie. Questo permette a ricercatori, sviluppatori e aziende di utilizzare, modificare e implementare il set di istruzioni senza dover affrontare costi di licenza o restrizioni legali. La struttura di RISC-V è divisa in due parti principali:

- **Base ISA**: Un insieme minimale di istruzioni obbligatorie, che costituisce il nucleo fondamentale dell'architettura.
- **Estensioni opzionali** : Funzionalità aggiuntive che possono essere implementate in base alle necessità specifiche dell'applicazione. Tra queste troviamo estensioni per il supporto alle operazioni in virgola mobile, le operazioni vettoriali e, in particolare, la virtualizzazione.

2.2 Design architetturale

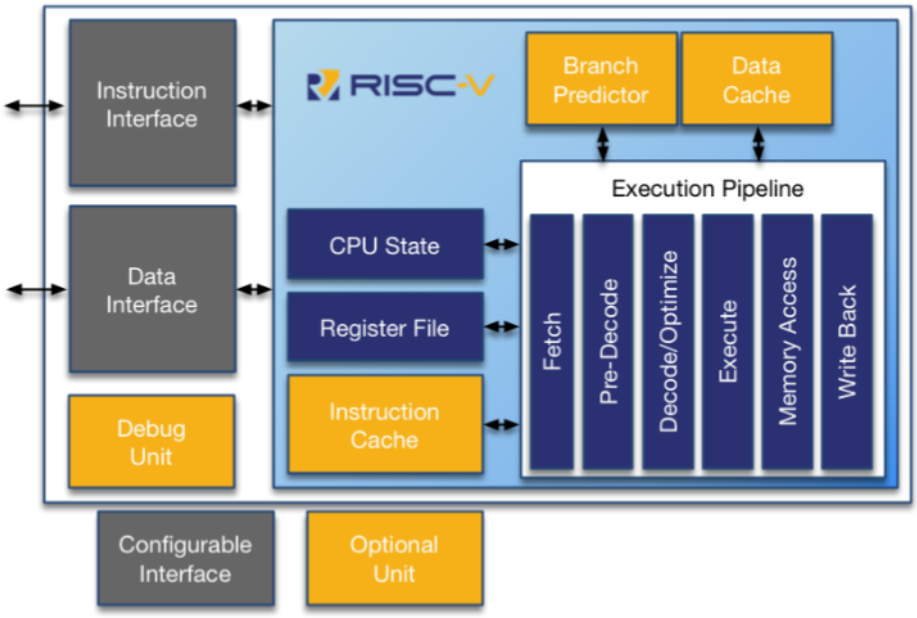


Figura 2: RISC-V schema

RISC-V è una tipica architettura **RISC (Reduced Instruction Set Computer)** che utilizza il modello load–store. Questo significa che le operazioni aritmetiche e logiche vengono eseguite esclusivamente sui dati contenuti nei registri[1]. Per interagire con la memoria, si usano istruzioni specifiche per il caricamento e il salvataggio dei dati. Questa

[1]Nella versione standard di RISC-V ci sono 32 registri interi (mentre nelle versioni per sistemi embedded, ottimizzate per ridurre l'hardware, si utilizzano 16 registri). Se si implementa l'estensione per il supporto in virgola mobile, vengono aggiunti altri 32 registri dedicati esclusivamente ai calcoli in virgola mobile.

separazione semplifica l'implementazione del processore e favorisce una pipeline di esecuzione più efficiente. Come molte architetture RISC, RISC-V sfrutta una pipeline di esecuzione suddivisa in fasi (fetch, decode, execute, memory access e write-back), questo approccio consente di aumentare il throughput del processore, permettendo l'esecuzione parallela di più istruzioni in fasi diverse della pipeline.

2.3 Modello a Livelli di Privilegio in RISC-V

L'architettura RISC-V definisce diversi livelli di privilegio[2] per gestire le operazioni del sistema in modo sicuro e strutturato. Ogni livello ha un diverso grado di accesso alle risorse hardware e ai registri di controllo della CPU.

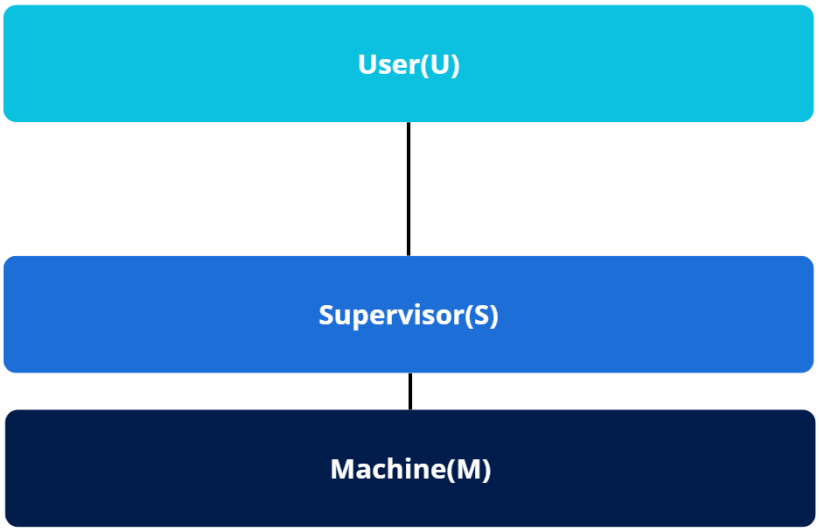


Figura 3: Livelli di privilegio in RISC-V

2.3.1 Machine Mode (M-mode)

È il livello più alto di privilegio, obbligatorio in tutte le implementazioni RISC-V, ed è generalmente utilizzato dal firmware e dal bootloader per inizializzare il sistema. In questo livello risiede il **SBI (Supervisor Binary Interface)**, un'astrazione software che fornisce servizi ai livelli inferiori. Questo livello ha il controllo totale dell'hardware e può accedere a tutti i registri e gestire le interruzioni.

2.3.2 Supervisor Mode (S-mode)

Questo livello è usato dai sistemi operativi per gestire la memoria, i processi e le periferiche. Abilita la gestione della memoria virtuale (tramite un'unità MMU o PMP per protezione della memoria), e a differenza della Machine Mode non ha accesso diretto all'hardware, infatti alcuni servizi vengono richiesti tramite l' SBI.

2.3.3 User Mode (U-mode)

È il livello meno privilegiato, in cui girano le applicazioni. Non può accedere direttamente all'hardware o alla memoria di altri processi: deve passare per il sistema operativo (chiamate di sistema/syscall). Questo livello garantisce un'esecuzione sicura e isolata delle applicazioni.

2.4 Estensioni e Virtualizzazione

La modularità di RISC-V permette di aggiungere diverse estensioni per arricchire il set di istruzioni di base. Tra le più comuni ci sono:

- **Estensioni per operazioni in virgola mobile**: Per il calcolo in singola e doppia precisione.
- **Estensione vettoriale**: Per applicazioni che richiedono elaborazioni parallele, come l'intelligenza artificiale e il machine learning
- **Hypervisor Extension**: Questa estensione è fondamentale per la virtualizzazione. Essa introduce un livello di privilegio aggiuntivo, specifico per l'hypervisor, che consente di gestire più macchine virtuali in maniera isolata e sicura. In pratica, il processore RISC-V con supporto alla virtualizzazione può eseguire contemporaneamente diversi sistemi operativi o ambienti applicativi, ognuno in un contesto isolato, migliorando così la sicurezza e l'efficienza nella gestione delle risorse.

3 Estensioni per la virtualizzazione su RISC-V

3.1 Hypervisor Extension: Nuove modalità di esecuzione

L'estensione per la virtualizzazione di RISC-V introduce una nuova modalità di esecuzione chiamata **HS-mode (Hypervisor Supervisor Mode)**, che si affianca alle modalità esistenti. Questa nuova modalità consente l'esecuzione diretta degli hypervisor e l'aggiunta di due nuovi livelli virtuali:

- **HS-mode**: Un'estensione della Supervisor Mode, che permette agli hypervisor di controllare le VM senza perdere prestazioni.
- **VS-mode (Virtual Supervisor Mode)**: Una modalità di supervisione virtualizzata, in cui il guest OS viene eseguito come se fosse in modalità S reale. Questo permette agli OS guest di funzionare senza modifiche.
- **VU-mode (Virtual User Mode)**: La versione virtualizzata della User Mode, in cui le applicazioni utente dei sistemi guest vengono eseguite come se fossero in modalità U normale.

Questa separazione consente una gestione chiara dei privilegi e riduce il bisogno di interventi software pesanti.

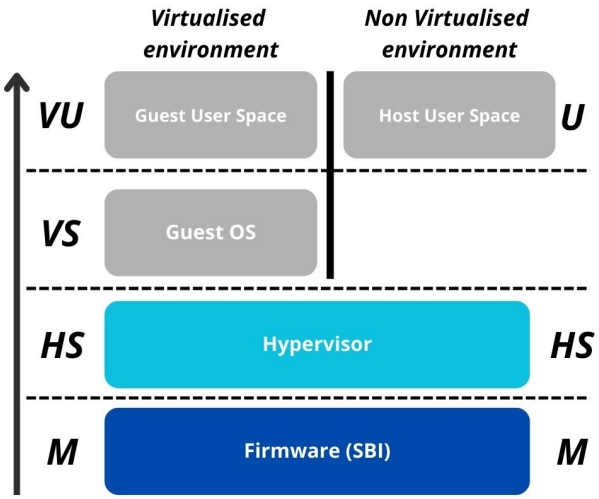


Figura 4: Nuove modalità di esecuzione

3.2 Two-Stage Address Translation

La **Memory Management Unit (MMU)** è un componente hardware che si occupa di tradurre gli indirizzi di memoria virtuale in indirizzi fisici e applicare i permessi di accesso (lettura, scrittura, esecuzione) per garantire la sicurezza. Uno dei problemi principali nella virtualizzazione è la gestione degli indirizzi di memoria. Un sistema operativo guest, infatti, crede di avere accesso diretto alla memoria fisica, ma in realtà la sua memoria è virtualizzata. A questo proposito RISC-V introduce la **Two-Stage Address Translation**[3], che aggiunge un secondo livello di traduzione degli indirizzi chiamato **G-Stage**.

1. Il Guest OS genera un **Guest Virtual Address (GVA)**, che viene tradotto in un **Guest Physical Address (GPA)** tramite la traduzione di primo livello gestita dal guest OS.

2. L'hypervisor traduce il Guest Physical Address (GPA) in un **Host Physical Address (HPA)** usando il secondo livello di traduzione (G-stage).

Questa doppia traduzione permette all'hypervisor di controllare e proteggere l'accesso alla memoria, impedendo a un guest di accedere o modificare dati di altri guest o del sistema host.

3.2.1 Registro vsatp

Il registro **vsatp** (Virtual Supervisor Address Translation and Protection) è utilizzato dal guest OS per la traduzione degli indirizzi virtuali del guest in indirizzi fisici guest. Il registro vsatp contiene tre campi principali:

- **MODE**: Indica lo schema di traduzione (Sv39, Sv48, ecc.).

- **ASID (Address Space Identifier)**: Un identificatore per distinguere i diversi spazi di indirizzi dei processi in esecuzione. Evita la necessità di svuotare completamente la TLB ogni volta che cambia il contesto di un processo.
- **PPN (Physical Page Number)**: Contiene il puntatore alla root della VS-stage page table, che traduce gli indirizzi virtuali guest in indirizzi fisici guest.

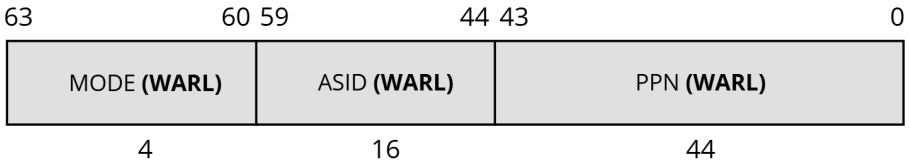


Figura 5: Registro vsatp

3.2.2 Registro hgatp

Il registro **hgatp** è utilizzato dall'hypervisor per gestire la traduzione degli indirizzi fisici guest in indirizzi fisici reali dell'host. Il registro hgatp contiene tre campi principali:

- **MODE**: Specifica lo schema di traduzione della memoria virtuale (es. Sv39, Sv48, ecc..).
- **VMID (Virtual Machine Identifier)**: Un identificatore univoco per distinguere le diverse VM in esecuzione, utile per evitare conflitti nelle TLB.
- **PPN (Physical Page Number)**: Contiene l'indirizzo fisico della G-stage root page table, ovvero la tabella delle pagine che mappa gli indirizzi fisici guest in quelli reali.

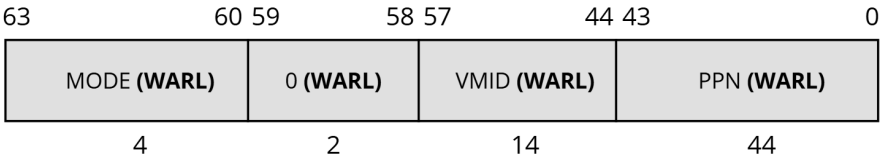


Figura 6: Registro hgapt

3.2.3 Modelli di Traduzione degli Indirizzi in RISC-V

La traduzione degli indirizzi in RISC-V utilizza un **radix tree multilivello**[2] per convertire un indirizzo virtuale in un indirizzo fisico. Questo metodo è usato nei sistemi con

[2]Un radix tree multilivello è una struttura dati gerarchica utilizzata per rappresentare in modo efficiente insiemi di chiavi numeriche o indirizzi. Ogni livello dell'albero suddivide l'indirizzo o la chiave in porzioni di bit fissi, riducendo lo spazio di ricerca e migliorando l'efficienza nella traduzione degli indirizzi

memoria virtuale per consentire la gestione efficiente degli spazi di indirizzamento e per proteggere i processi dall'accesso non autorizzato alla memoria. I modelli di traduzione supportati da RISC-V sono:

- **Sv32**: 2 livelli, per architetture a 32-bit
- **Sv39**: 3 livelli, per architetture a 64-bit
- **Sv48**: 4 livelli, per architetture a 64-bit avanzate
- **Sv57**: 5 livelli, per spazi di indirizzamento ancora più grandi

3.2.4 Struttura della tabella delle pagine

L'indirizzo guest virtuale viene suddiviso in più parti, ciascuna delle quali viene usata per accedere a un livello della tabella delle pagine. Ogni **Table Entry (PTE)** può contenere:

- Un puntatore alla tabella successiva (se la traduzione non è ancora completa).
- Un indirizzo fisico (se è l'ultima entry).
- Permessi di accesso (R=Read, W=Write, X=Execute, V=Valid).

3.2.5 Formato di un indirizzo virtuale in RISC-V

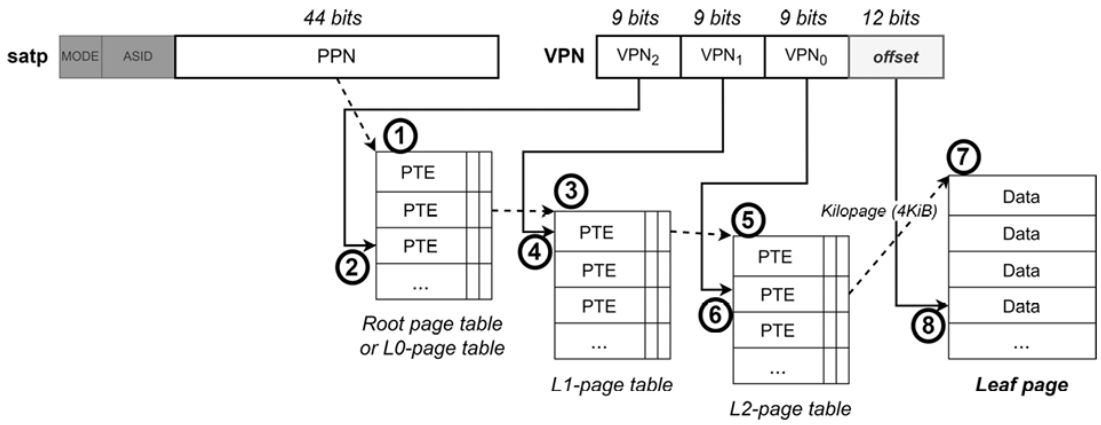


Figura 7: Two stage address translation

L'indirizzo guest virtuale è diviso in tre, quattro o cinque parti, a seconda del modello di traduzione. Se si utilizza il modello Sv39 avremo che:

Bit	Descrizione
38-30	Indice per il livello 1 (PTE1)
29-21	Indice per il livello 2 (PTE2)
20-12	Indice per il livello 3 (PTE3)
11-0	Offset nella pagina (non cambia)

Tabella 3: Modello Sv39

3.3 Hypervisor Control and Status Registers (CSRs)

I **Control and Status Registers (CSRs)** in RISC-V sono registri speciali utilizzati per il controllo e la gestione del processore. Questi registri contengono informazioni critiche sullo stato del sistema, sulle configurazioni di privilegi, sulle eccezioni e sugli interrupt, oltre a fornire meccanismi per l'ottimizzazione delle prestazioni e la virtualizzazione. Quando un sistema è virtualizzato, ogni VM in VS-mode ha bisogno di una propria versione di questi registri, perché ogni sistema operativo guest deve credere di avere il pieno controllo della CPU. RISC-V introduce registri "ombra" per la modalità VS, che sono copie separate dei normali registri S-mode.

CSR	Descrizione
`mstatus`	Stato del processore in Machine Mode, controlla privilegi, interrupt e floating-point.
`sstatus`	Stato del processore in Supervisor Mode, versione ridotta di `mstatus`.
`misa`	Definisce l'architettura ISA supportata (es. RV32, RV64, estensioni MAFD).
`mtvec`	Contiene l'indirizzo base dell'interrupt vector table in Machine Mode.
`stvec`	Contiene l'indirizzo base dell'interrupt vector table in Supervisor Mode.

Tabella 4: CSRs Generali per il controllo del sistema

CSR	Descrizione
`satp`	Gestisce la tabella delle pagine per la memoria virtuale (es. Sv32, Sv39, Sv48).
`vsatp`	Versione virtualizzata di `satp`, utilizzata per il Guest OS.
`hgatp`	Gestisce la tabella delle pagine G-stage per la virtualizzazione a due livelli.

Tabella 5: CSRs per la gestione della memoria e virtualizzazione

CSR	Descrizione
`hstatus`	Stato dell'hypervisor (simile a `mstatus`).
`hedeleg`	Delegazione delle eccezioni dalla modalità H-mode alla VS-mode.
`hideleg`	Delegazione delle interruzioni dalla modalità H-mode alla VS-mode.
`htimedelta`	Contiene la differenza temporale tra hypervisor e guest.

Tabella 6: CSRs per la virtualizzazione e hypervisor

3.3.1 V-bit (Virtualization Mode Flag)

Un aspetto chiave della virtualizzazione RISC-V è il **V-bit**, un flag che indica se il processore sta eseguendo codice in una macchina virtuale. Se V = 1, gli accessi ai registri di S-mode in realtà a avvengono nei CSRs di VS-level , e la traduzione G-stage della memoria è attiva, invece se V = 0, il sistema opera in S-mode normale, senza virtualizzazione.

3.3.2 Gestione delle eccezioni nella virtualizzazione

Quando un sistema operativo guest esegue istruzioni privilegiate o accede a risorse protette, possono verificarsi eccezioni. Nella virtualizzazione, è fondamentale distinguere le eccezioni generate dal guest da quelle generate dall'hypervisor. RISC-V introduce eccezioni specifiche per il livello VS, che vengono gestite separatamente dall'hypervisor. Alcuni esempi:

- **VS-level Page Faults**: Il guest OS potrebbe accedere a una pagina di memoria non valida o non mappata.
- **Illegal Instructions in VS-mode**: Se il guest OS esegue un'istruzione non consentita.

3.4 Hypervisor instructions

Quando un hypervisor virtualizza un sistema operativo guest, può essere necessario accedere alla memoria della VM per diverse operazioni, come il salvataggio dello stato della macchina per snapshot e debugging, la gestione dei dispositivi emulati o il trattamento di fault di memoria generati dal guest. Tuttavia questo accesso non è diretto, poiché la memoria del guest utilizza una traduzione degli indirizzi a due livelli, di conseguenza l'hypervisor non può semplicemente usare le normali istruzioni di caricamento e salvataggio, perché il contesto della memoria del guest segue una traduzione separata, e l'accesso deve rispettare i permessi di isolamento per garantire la sicurezza del sistema.
Per risolvere questo problema, l'architettura RISC-V introduce istruzioni specializzate, come **HLV**, **HSV** e **HLVX**, che consentono all'hypervisor di leggere e scrivere nella memoria del guest in modo sicuro e controllato.

3.4.1 Istruzioni di Load/Store per l'hypervisor

Consentono all'hypervisor di leggere/scrivere direttamente nella memoria del guest. La traduzione degli indirizzi segue gli stessi permessi e privilegi del guest, inoltre garantiscono operazioni efficienti senza bisogno di emulazione complessa.

- **HLV (Hypervisor Load Virtual) / HSV (Hypervisor Store Virtual)**: Consentono di caricare o memorizzare dati nella memoria virtuale del guest rispettando i permessi impostati dal guest stesso. L'hypervisor può accedere alla memoria della VM senza alterare i permessi del guest.

3.4.2 Istruzioni Fence per l'hypervisor

Sincronizzano e aggiornano le strutture di traduzione degli indirizzi, come la TLB (Translation Lookaside Buffer). Sono fondamentali nei cambi di contesto tra VM per evitare

che rimangano dati obsoleti nella cache, e assicurano che le traduzioni della memoria del guest siano sempre coerenti.

- **HFENCE.GVMA (Hypervisor Fence Guest Virtual Memory Address)**: Invalida le TLB del guest, assicurando che le traduzioni obsolete non vengano utilizzate. Utilizzato nei cambi di contesto tra VM o durante le modifiche alle tabelle delle pagine.
- **HINVAL.VVMA (Hypervisor Invalidate Virtual Virtual Memory Address)**: Invalida specifiche entry nelle TLB relative alla memoria virtuale della VM.

3.5 RISC-V "stimecmp/vstimecmp" Extension

Normalmente, in RISC-V, gli interrupt dei timer sono gestiti dalla M-mode, che è la modalità con il massimo privilegio nel sistema. Questo significa che:

- Il timer genera un interrupt, ma questo arriva sempre alla M-mode.
- Il sistema operativo in S-mode non può gestire direttamente il timer, quindi deve fare una chiamata alla M-mode per leggere o aggiornare i registri del timer.
- Questo causa overhead (perdita di tempo e risorse), soprattutto in un ambiente virtualizzato, dove un guest OS deve passare attraverso l'hypervisor in HS-mode.

L'estensione Sstc risolve questo problema permettendo alla S-mode (e alla VS-mode per i guest) di gestire direttamente i timer, senza bisogno di interventi dalla M-mode.

3.5.1 Registro stimecp

Usato dal sistema operativo in S-mode (o in HS-mode per l'hypervisor). Quando il valore del contatore del tempo supera stimecmp, si genera un interrupt di timer direttamente per il supervisor, eliminando la necessità di passare per la M-mode per gestire i timer.

3.5.2 Registro vstimecmp

Usato dal guest OS in VS-mode, e permette a un sistema operativo guest di gestire il timer senza dover chiedere all'hypervisor di intervenire. Migliora l'efficienza della virtualizzazione riducendo il numero di switch di contesto tra il guest e l'hypervisor.

4 CVA6 Core

In questa sezione verranno trattati alcuni aspetti descritti nel paper ***"CVA6 RISC-V Virtualization: Architecture, Microarchitecture, and Design Space Exploration"***[1] che analizza le modifiche apportate al design del CVA6 per supportare la virtualizzazione, esplorando sia gli aspetti architetturali che micro-architetturali.
Vedremo come il design originale del CVA6 è stato modificato per supportare la virtualizzazione, introducendo estensioni mirate a ridurre l'overhead. Tra queste, spicca l'implementazione del **GTLB (G-Stage TLB)**, un secondo livello di TLB integrato con il Page

Table Walker, che ottimizza la traduzione degli indirizzi in ambienti virtualizzati. Inoltre, è stato aggiunto un **L2 TLB**, migliorando ulteriormente le prestazioni complessive del sistema.

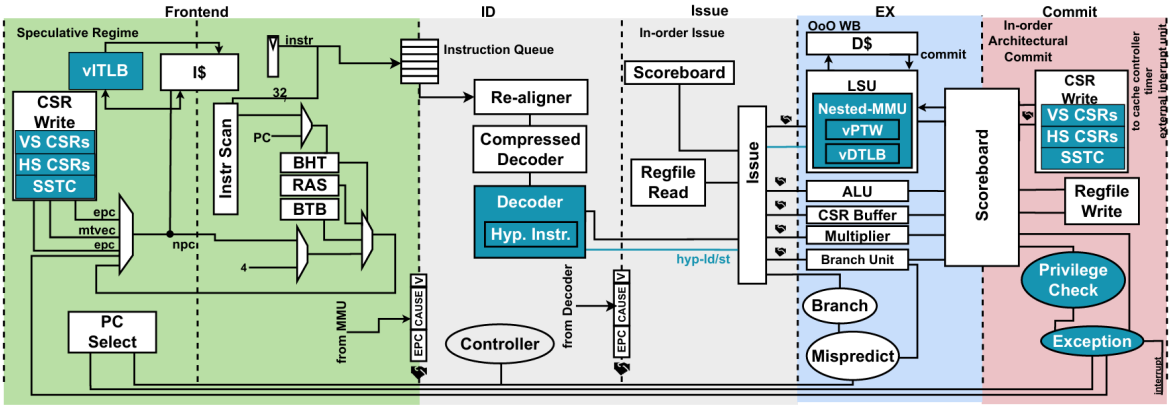


Figura 8: Micro-architettura Core CVA6

4.1 Caratteristiche di CVA6

Il **CVA6** è un core RISC-V di classe applicativa che supporta sia l'ISA RV64 che l'ISA RV32, rendendolo versatile per applicazioni che spaziano dai sistemi embedded ai dispositivi ad alte prestazioni. Una delle sue caratteristiche principali è il supporto completo ai tre livelli di privilegio definiti dall'architettura RISC-V (Machine Mode, Supervisor Mode e User Mode), il che lo rende compatibile con sistemi operativi complessi come Linux. Il core implementa una Memory Management Unit che gestisce la traduzione degli indirizzi virtuali in indirizzi fisici, supportando i meccanismi di paginazione **Sv39** per sistemi a 64 bit e **Sv32** per sistemi a 32 bit. Per ottimizzare le prestazioni, dispone di TLB separati per dati e istruzioni, oltre a un Page Table Walker per navigare autonomamente nelle tabelle delle pagine.

4.1.1 Microarchitettura

Dal punto di vista microarchitetturale, il CVA6 utilizza una **pipeline a 6 stadi** con un'unità di esecuzione **out-of-order**, capace di migliorare l'efficienza e ridurre le dipendenze tra istruzioni. È un core **single-issue**, il che significa che può emettere una sola istruzione per ciclo di clock, garantendo un design semplice ma efficace, inoltre integra 8 entry di **Physical Memory Protection**, fondamentali per il controllo degli accessi alla memoria e per garantire sicurezza nei sistemi multi-tenant. Un'ulteriore espansione delle capacità di calcolo è fornita dall'**unità vettoriale** recentemente introdotta, ottimizzata per il calcolo parallelo e il supporto alle RISC-V Vector Extensions, migliorando così le prestazioni in applicazioni di machine learning, elaborazione multimediale e calcoli scientifici.

4.2 Supporto alla virtuailzzazione

Il supporto alla virtualizzazione nel core CVA6 ha richiesto un'estensione delle sue funzionalità fondamentali, con particolare attenzione alla gestione dei registri CSR, delle

eccezioni e delle interruzioni, nonché alla traduzione degli indirizzi in un contesto di Two-Stage Address Translation. Per garantire un accesso sicuro e controllato ai registri nei nuovi livelli di esecuzione HS-mode e VS-mode, è stato introdotto un meccanismo di verifica dei permessi per le operazioni di lettura e scrittura, inoltre l'hypervisor ha ora la possibilità di intercettare e gestire le interruzioni, delegandole eventualmente al sistema operativo guest, evitando accessi non autorizzati e mantenendo l'isolamento tra le VM.
Un aspetto fondamentale dell'aggiornamento del core è la gestione avanzata della memoria attraverso la **Nested-MMU Translation Logic**, che consente la traduzione degli indirizzi in due fasi:

1. VS-Stage: Guest Virtual Address → Guest Physical Address
2. G-Stage: Guest Physical Address → Host Physical Address

Questo meccanismo permette di mantenere un controllo più preciso sull'accesso alla memoria da parte delle VM, migliorando la sicurezza e l'efficienza della virtualizzazione.
Una modifica che è stata fatta sul core CVA6 riguarda l'aggiornamento della Finite State Machine responsabile della traduzione degli indirizzi. In particolare, è stato aggiunto un nuovo stato di controllo, denominato **G-Stage Intermed**, per gestire la transizione tra la VS-Stage e la G-Stage. Questo nuovo stato consente di tradurre correttamente tutti gli accessi alla memoria che si verificano mentre il Page Table Walker opera nella VS-Stage. Infatti, se le tabelle delle pagine del guest sono memorizzate in indirizzi fisici guest, questi devono prima essere tradotti in indirizzi fisici host per poter essere utilizzati correttamente.

4.3 Estensioni micro-architetturali

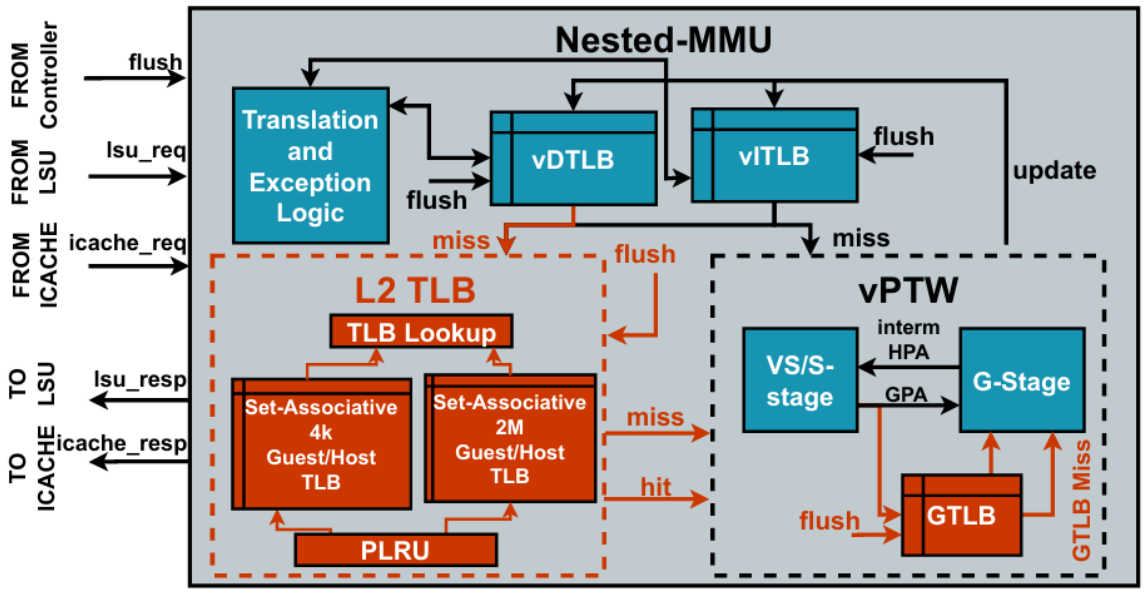


Figura 9: Modifiche micro-architetturali alla MMU

4.3.1 GTLB

Nei sistemi virtualizzati, la traduzione degli indirizzi segue un processo annidato composto da due fasi: il VS-stage, che converte un indirizzo virtuale guest in un indirizzo fisico guest, e il G-stage, che traduce quest'ultimo in un indirizzo fisico host.
Questo meccanismo, in particolare con lo schema Sv39, può risultare oneroso in termini di accessi alla memoria. Un numero elevato di accessi aumenta la latenza in caso di miss nel TLB, generando un overhead che incide sulle prestazioni complessive del sistema.
Per ridurre questo impatto, è stato introdotto il **Guest TLB (GTLB)** all'interno del modulo Page Table Walker annidato. Il GTLB memorizza traduzioni intermedie tra indirizzi fisici guest e indirizzi fisici host, diminuendo il numero di traduzioni ripetute e migliorando l'efficienza della memoria.
L'efficacia di questo approccio è particolarmente evidente con pagine di piccole dimensioni (4 KiB), dove il numero di traduzioni è più elevato, mentre il beneficio risulta meno significativo con superpagine (2 MiB o 1 GiB).
Dal punto di vista microarchitetturale, il GTLB è **completamente associativo**, permettendo una ricerca rapida grazie al confronto parallelo delle voci memorizzate. Supporta tutte le dimensioni di pagina previste dal formato Sv39 e utilizza un algoritmo di sostituzione **pseudo-LRU**, già impiegato nei TLB L1 del CVA6, con una capacità compresa tra 8 e 16 voci.
Infine il GTLB integra un circuito di flush compatibile con l'istruzione HFENCE.GVMA, che consente la rimozione selettiva delle voci in base al Virtual Machine Identifier e all'indirizzo virtuale, evitando così lo svuotamento completo della cache.

4.4 L2 TLB in CVA6

La Memory Management Unit del CVA6, nella sua configurazione base, utilizza piccoli TLB di primo livello separati per istruzioni e dati. In questa architettura, il TLB unifica le traduzioni dei due livelli di virtualizzazione (VS-stage e G-stage) in un'unica voce, scegliendo la dimensione minima tra le due pagine. Questo approccio semplifica l'hardware e riduce il numero di ricerche necessarie, evitando la complessità di gestire TLB separati per ciascun livello di traduzione. Tuttavia, questa soluzione presenta una limitazione: se l'hypervisor e il guest utilizzano superpagine di dimensioni diverse, il TLB non può sfruttarle a pieno, di conseguenza si riduce la copertura delle traduzioni e aumenta il numero di TLB miss, causando più page table walks con conseguente impatto negativo sulle prestazioni complessive del sistema.
Per risolvere questo problema, è stato introdotto il **L2 TLB**, un nuovo TLB di secondo livello di tipo **set-associative**. Questo miglioramento aumenta la copertura e riduce gli svantaggi causati dai TLB miss, inoltre la sua architettura consente di gestire separatamente le traduzioni per pagine di dimensioni diverse, semplificando la logica di ricerca.
Il L2 TLB utilizza una **memoria SRAM** per archiviare le traduzioni e impiega la stessa politica di sostituzione pseudo-LRU già adottata nel GTLB.
Un aspetto chiave del L2 TLB è la sua capacità di eseguire in **parallelo** sia la ricerca nel TLB sia la traduzione tramite il Page Table Walker, evitando così un aumento della latenza in caso di miss nel L1 TLB. Inoltre è stato progettato per eseguire ricerche parallele su pagine di diverse dimensioni (4 KiB e 2 MiB) grazie a strutture hardware indipendenti con unità di controllo dedicate.
Per quanto riguarda la gestione delle voci memorizzate, L2 TLB supporta le istruzioni di

flush (SFENCE e HFENCE). Tuttavia, quando viene attivato il segnale di flush, tutte le voci vengono eliminate in blocco, senza la possibilità di filtrare in base a VMID o ASID. Il controllo del L2 TLB è gestito da una Finite State Machine (FSM) con quattro stati principali:

1. Flush state → svuota tutte le voci del TLB.
2. Idle state → attende nuove richieste dal modulo PTW.
3. Read state → cerca una voce nel TLB: se la trova (hit), segnala il risultato al PTW; in caso contrario (miss), torna nello stato Idle.
4. Update state → aggiorna il TLB con una nuova traduzione ricevuta dal PTW.

4.5 Estensione Sstc

L'estensione Sstc nei processori RISC-V introduce una gestione avanzata dei timer, basata su registri CSR specifici per ogni core del processore. A differenza del metodo tradizionale, che si affida esclusivamente ai timer MMIO come mtime, questa soluzione riduce la latenza e migliora l'efficienza.
I nuovi registri stimecmp e vstimecmp permettono al sistema operativo (modalità S) e alle macchine virtuali (modalità VS) di gestire direttamente i timer, evitando il passaggio attraverso componenti esterni come il CLINT.
Un'importante modifica riguarda l'accesso a questi registri, regolato dai bit **STCE** nei registri CSR menvcfg e henvcfg. Se il bit corrispondente è impostato a 0, qualsiasi tentativo di accesso genera:

- Una illegal exception in modalità S.
- Una virtual illegal exception in modalità VS.

Questo meccanismo fornisce un controllo granulare sull'uso dell'estensione, impedendone l'accesso non autorizzato. Allo stesso tempo, mantiene la compatibilità con il software legacy, che può continuare a configurare i timer tramite l'interfaccia standard SBI utilizzando mtimecmp.

5 XEN su RISC-V

5.1 XEN: Architettura

Xen[4] è un hypervisor open-source di tipo bare-metal che permette di eseguire più sistemi operativi in parallelo su un'unica macchina fisica. Funziona come uno strato software tra l'hardware e le macchine virtuali, garantendo alte prestazioni e un efficace isolamento tra i sistemi ospitati.
La sua architettura è basata su un microkernel, che delega la gestione dell'hardware e delle VM a un sistema operativo privilegiato chiamato **Dom0** , mentre i sistemi guest vengono eseguiti in **DomU**.
Xen supporta sia la paravirtualizzazione, che ottimizza le prestazioni dei sistemi guest modificati, sia la virtualizzazione completa, sfruttando le estensioni hardware delle CPU

moderne (Intel VT-x, AMD-V, ARM Virtualization Extensions) per eseguire sistemi operativi non modificati.
Grazie alla sua compatibilità con architetture x86, ARM, e con sistemi operativi come Linux, Windows e FreeBSD, Xen è ampiamente utilizzato in ambienti cloud, data center e scenari che richiedono elevata sicurezza e isolamento.

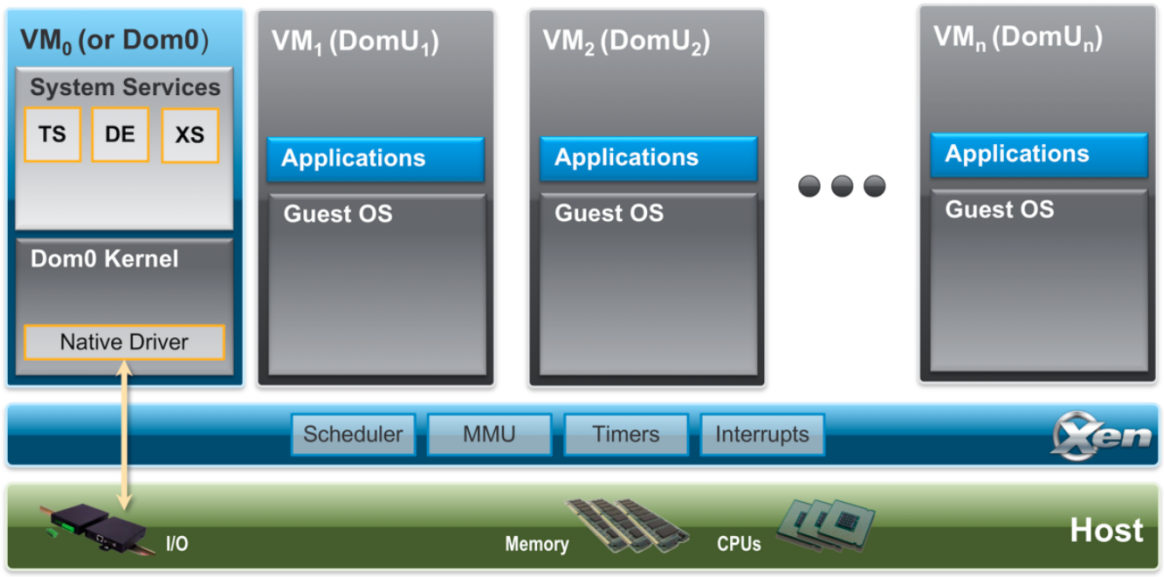


Figura 10: Architettura di Xen

5.1.1 Xen Hypervisor

È il livello più basso del sistema, eseguito direttamente sull'hardware. Si occupa di:

- Gestire la CPU e la memoria delle macchine virtuali.
- Controllare i timer e gli interrupt.
- Non si occupa direttamente di I/O (rete, disco, periferiche).

5.1.2 Domini e Macchine Virtuali

- **Domain 0**: è la macchina virtuale con privilegi speciali che gestisce l'hardware e il controllo del sistema.
- **Domain U**: sono le macchine virtuali standard, completamente isolate dall'hardware e senza accesso diretto alle periferiche.

5.1.3 Dom0 e i suoi componenti

Il Dom0 è essenziale per il funzionamento di Xen e contiene diversi componenti:

- **XenStore**: è una struttura di dati centralizzata utilizzata per memorizzare informazioni di configurazione e stato. Viene utilizzata dal sistema Dom0 per gestire le risorse e dalla macchina virtuale DomU per scambiare informazioni.

- **XenBus**: è un meccanismo di comunicazione che consente alle macchine virtuali di comunicare tra loro e con il sistema host (Dom0).
- **Toolstack (TS)**: permette di creare, avviare, arrestare e configurare le VM.
- **Device Emulation (DE)**: basata su QEMU, fornisce supporto per periferiche virtuali.
- **Driver nativi**: Dom0 contiene i driver fisici per le periferiche hardware.
- **Driver virtuali (backend)**: permettono alle VM di comunicare con l'hardware attraverso Dom0.

5.1.4 Paravirtualized guests

La paravirtualizzazione è una tecnica di virtualizzazione che prevede modifiche al sistema operativo guest per ottimizzare le operazioni che altrimenti sarebbero eseguite in modalità di emulazione lenta. In una virtualizzazione completa, le istruzioni privilegiate (come accessi ai registri di controllo della CPU) vengono intercettate e emulate dall'hypervisor, con un costo prestazionale elevato. Nella paravirtualizzazione, il sistema operativo guest è consapevole di essere in esecuzione in un ambiente virtuale e usa delle API speciali (Hypercalls) per interagire direttamente con l'hypervisor, riducendo il bisogno di emulazione. In Xen, la paravirtualizzazione è realizzata mediante la creazione di interfacce software tra l'hypervisor e i guest OS, che sono progettati per utilizzare queste interfacce ottimizzate. Affinché un sistema operativo possa essere eseguito in modalità paravirtualizzata su Xen, deve essere modificato per essere Xen-aware.

5.2 XEN: Traduzione indirizzi virtuali

Xen ottimizza la traduzione degli indirizzi virtuali riducendo l'overhead e evitando le shadow page tables usate da altri hypervisor. Invece di mantenere copie separate delle tabelle delle pagine, registra direttamente quelle del guest OS nella MMU, limitandone l'accesso in sola lettura per garantire sicurezza. Quando il guest deve aggiornarle, invia una hypercalls a Xen, che verifica e applica le modifiche in modo controllato. Per prevenire modifiche non autorizzate, ogni frame di memoria ha un tipo specifico e un contatore di riferimenti, impedendo ai guest di alterare direttamente la propria gestione della memoria. Xen migliora anche le prestazioni consentendo ai guest di accumulare più aggiornamenti e applicarli in una sola operazione, riducendo il numero di hypercalls. Infine, per garantire la correttezza delle traduzioni, sfrutta il flush della TLB, evitando mappature obsolete e correggendo eventuali errori attraverso la gestione dei page fault.

5.3 XEN: Memoria fisica

Xen assegna a ogni dominio una quantità fissa di memoria al momento della creazione, garantendo isolamento e stabilità. Tuttavia, i domini possono richiedere più memoria fino a un limite massimo predefinito o restituire memoria inutilizzata per ottimizzare le risorse complessive del sistema. Questo è reso possibile dal **balloon driver**, un meccanismo di gestione dinamica della memoria che permette ai guest OS di adattare il loro utilizzo senza modifiche dirette al sistema operativo.

A differenza di altri hypervisor, Xen non assegna blocchi di memoria contigui ai domini. Per evitare problemi di frammentazione, il guest OS crea l'illusione di una memoria continua e gestisce autonomamente la mappatura tra gli indirizzi fisici (quelli usati dal sistema operativo) e gli indirizzi hardware (quelli della memoria fisica reale). Per garantire che ogni dominio acceda solo alla memoria a lui assegnata, Xen fornisce una **tabella di traduzione condivisa** che i guest OS possono consultare per gestire correttamente i propri spazi di indirizzamento.

5.4 XCP-ng

XCP-ng (Xen Cloud Platform - Next Generation)[5] è una piattaforma di virtualizzazione open-source basata su Xen, progettata per offrire un'alternativa flessibile alle soluzioni commerciali (VMware vSphere ecc..). Questa piattaforma consente l'integrazione con strumenti avanzati come Xen-Orchestra[3], fornendo un'interfaccia intuitiva per la gestione di macchine virtuali, snapshot, migrazioni live e monitoraggio delle risorse. La sua architettura è progettata per offrire elevate prestazioni, scalabilità e sicurezza, rendendolo adatto per ambienti di produzione, cloud privati e infrastrutture aziendali.
Uno dei punti di forza di XCP-ng è la sua compatibilità con un'ampia gamma di hardware, oltre alla capacità di gestire carichi di lavoro complessi con supporto per tecnologie GPU passthrough e clustering. Infine, grazie al supporto nativo per API RESTful, è facilmente integrabile con strumenti di automazione e orchestrazione.

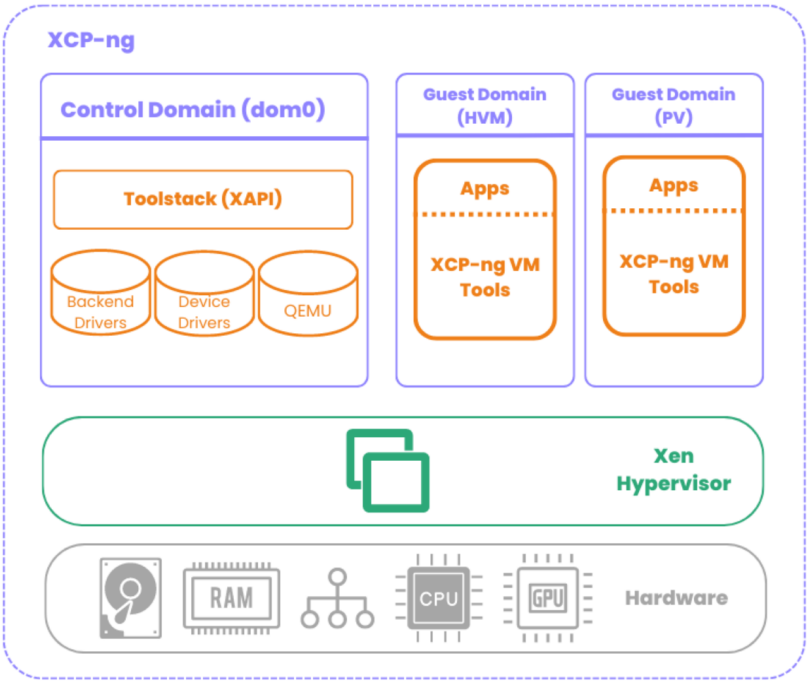


Figura 11: Architettura di XCP-ng

[3]Xen Orchestra è una piattaforma di gestione per XCP-ng che offre un'interfaccia web completa per il controllo e il monitoraggio delle infrastrutture virtualizzate. Sviluppata per semplificare l'amministrazione di ambienti basati su Xen, permette agli utenti di gestire macchine virtuali, archiviazione, networking e sicurezza in modo centralizzato, senza la necessità di strumenti proprietari.

5.5 Porting di Xen su RISC-V

Sia XCP-ng che RISC-V condividono una filosofia open-source. RISC-V, con la sua architettura aperta e modulare, consente ai produttori di personalizzare l'hardware senza dover pagare licenze, rendendolo una scelta ideale per ambienti di ricerca, cloud computing e dispositivi embedded. L'integrazione di un hypervisor come Xen su RISC-V offrirebbe una soluzione completamente libera da vincoli commerciali.
Questa soluzione permetterebbe di eseguire macchine virtuali su hardware a basso costo, riducendo così la dipendenza da processori x86 e ARM. Inoltre, poiché molti dispositivi IoT e sistemi embedded stanno adottando RISC-V per la sua efficienza energetica e flessibilità, un hypervisor come Xen consentirebbe di eseguire macchine virtuali leggere su dispositivi embedded, migliorando l'isolamento delle applicazioni e la sicurezza, senza necessità di hardware proprietario.
Il porting di Xen su RISC-V non è stato ancora del tutto completato, ma ci sono stati degli aggiornamenti negli utlimi quattro anni. Ne elencheremo alcuni nelle sezioni seguenti.

5.5.1 Ottimizzazione del codice per RISC-V

Uno degli obiettivi principali del porting di Xen[6] su RISC-V è stato ridurre il codice specifico per questa architettura e sfruttare il codice comune già esistente. Un passo importante in questa direzione è stata l'integrazione delle macro **BUG()**[4] e **WARN()**[5], già utilizzate su x86, evitando duplicazioni di codice. Inoltre, è stata definita la macro **BUG_INSN**[6] per specificare l'istruzione che causa un'eccezione fatale su RISC-V. Un'altra ottimizzazione significativa è stata la sostituzione della funzione early_printk() con **printk()**[7], standardizzando la gestione della stampa dei messaggi di debug tra le diverse architetture.

5.5.2 Miglioramenti nei test e nell'inizializzazione

Per garantire stabilità, sono stati ottimizzati i test di base (**smoke tests**[8]), rendendoli più automatici e meno dipendenti da modifiche manuali. Inoltre, è stata introdotta l'inizializzazione della sezione **.bss**[9], assicurando che tutte le variabili dichiarate ma non inizializzate partano da valore zero, evitando comportamenti imprevedibili. Sono state anche migliorate la gestione del **hart_id**[10] e del **Device Tree Blob**[11], fondamentali per la configurazione dell'hardware.

[4]Viene chiamata quando il sistema rileva una condizione fatale da cui non può riprendersi. Causa un crash immediato della macchina virtuale o dell'intero sistema.

[5]Registra un avviso nel log di sistema ma permette al codice di continuare l'esecuzione. Utile per errori meno gravi che non devono bloccare il sistema.

[6]Nel codice di basso livello, quando viene chiamata BUG(), il processore deve eseguire un'istruzione speciale per fermarsi correttamente. Questa istruzione varia da architettura ad architettura. Per RISC-V, è stata definita la macro BUG_INSN, che indica al linker quale istruzione usare quando viene chiamata BUG().

[7]Funzione di debug già parte del codice comune di Xen e compatibile con tutte le architetture.

[8]Test preliminari eseguiti per verificare che il sistema non abbia errori critici dopo una modifica.

[9].bss è una sezione della memoria che contiene variabili dichiarate ma non inizializzate. Secondo lo standard del linguaggio C tutte le variabili nella .bss devono essere azzerate prima dell'uso.

[10]L'hart_id è un numero univoco assegnato a ogni core della CPU.

[11]Il DTB è una struttura dati che contiene informazioni sull'hardware del sistema (es. memoria, periferiche, CPU). Viene passato dal bootloader al sistema operativo per permettergli di configurare correttamente l'hardware.

5.5.3 Introduzione di header specifici per RISC-V

Sono stati creati file di intestazione specifici per RISC-V, includendo operazioni essenziali come gestione della memoria (**Fence**[12]), operazioni atomiche (**compare-and-exchange**[13]), manipolazione dei bit e gestione dell'I/O. Per ridurre la duplicazione del codice tra le architetture, sono stati integrati gli header **asm-generic**[14], già presenti in altre versioni di Xen.

5.5.4 Funzionalità in sviluppo

- Gestione degli interrupt
- Mappatura del Device Tree, per assegnare le risorse hardware correttamente.
- Gestione delle pagine di memoria, migliorando l'allocazione delle risorse.
- Ottimizzazione della sincronizzazione tra CPU, tramite l'SBI RFENCE Extension.

5.5.5 Funzionalità Future

- **Dom0less**, che consente di eseguire Xen senza un sistema operativo principale, utile per sistemi embedded.
- **Device Passthrough**, che permette alle VM di accedere direttamente all'hardware, migliorando le prestazioni.
- Supporto all'**AIA**[15] per una migliore gestione degli interrupt.
- Integrazione dell'**IOMMU**[16], per una gestione avanzata della memoria virtualizzata.

5.6 Altri hypervisors

1. **Rvirt**: hypervisor trap-and-emulate per RISC-V che funziona senza KVM o Linux. Supporta QEMU's virt machine e HiFive Unleashed, permettendo l'esecuzione di più istanze di Linux su CPU RISC-V a 64 bit con MMU.
2. **Xvisor**: hypervisor Type-1 open-source leggero e portabile, compatibile con RISC-V, ARM, x86_64 e altre architetture. Supporta full virtualization per eseguire OS non modificati e paravirtualization tramite VirtIO per una maggiore efficienza.

[12]Le istruzioni Fence assicurano che le operazioni di lettura e scrittura siano completate prima di proseguire con altre istruzioni, evitando problemi di incoerenza dei dati tra i core della CPU

[13]Verifica se un valore di una variabile corrisponde a un valore atteso e, in tal caso, lo aggiorna.

[14]Nel codice di Xen, esistono già file di intestazione generici che possono essere utilizzati su più architetture. Questi file si trovano nella directory asm-generic/ e contengono definizioni comuni che possono essere riutilizzate senza dover riscrivere codice per ogni architettura.

[15]L'AIA (Advanced Interrupt Architecture) è un'estensione dell'architettura RISC-V progettata per migliorare la gestione degli interrupt, in particolare per sistemi multiprocessore e virtualizzati.

[16]L'IOMMU (Input-Output Memory Management Unit) è un componente hardware o software che gestisce la traduzione degli indirizzi di memoria per i dispositivi di I/O. Migliorara la sicurezza, l'efficienza e la gestione della memoria per i dispositivi di I/O, soprattutto in sistemi virtualizzati o con periferiche ad alte prestazioni.

3. **KVM per RISC-V**: hypervisor Type-2 che trasforma Linux in un hypervisor. Si basa su un modulo kernel KVM per virtualizzare CPU, memoria e interrupt, e su un tool user-space per gestire le macchine virtuali.

Hypervisor	Tipo	Supporto per RISC-V	Performance	Implementazione	Migliore per
Xen	Bare-metal	In sviluppo, parziale	Alta	Complesso	Ambienti enterprise e alta performance
RVirt	Trap-and-emulate	Completamente supportato	Alta	Facile, ma con limitazioni hardware	Scopi di ricerca e sviluppo
Xvisor	Bare-metal	Supporto parziale	Alta	Complesso	Supporto su più architetture, performance
KVM	Type-2 (Linux)	In fase di sviluppo	Buona	Facile (dipende da Linux)	Ambienti Linux-based su RISC-V

Figura 12: Confronto tra hypervisor su RISC-V

Riferimenti bibliografici

[1] Bruno Sá, *"CVA6 RISC-V Virtualization: Architecture, Microarchitecture, and Design Space Exploration"*.

[2] Zhiyuan Liang, Tianzheng Li, Enfang Cui, *"RISC-V Virtualization: Exploring Virtualization in an Open Instruction Set Architecture"*.

[3] `https://github.com/riscv/riscv-isa-manual`, *"RISC-V Instruction Set Manual"*

[4] `https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview`, *"Xen Project Software Overview"*

[5] `https://docs.xcp-ng.org`, *"XCP-ng Documentation"*

[6] `https://xcp-ng.org/blog/2024/09/23/advancing-xen-on-risc-v-key-updates/`, *"Xen porting on RISC-V"*