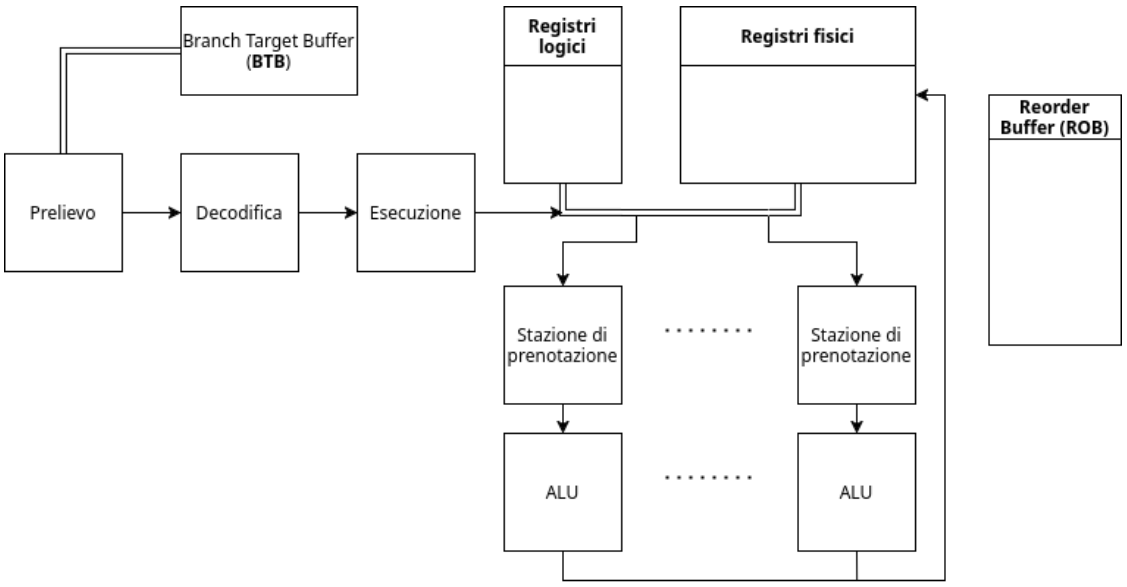
1 Lezione del 09-05-25

Riprendiamo il discorso sull'architettura del processore. Possiamo fare un riassunto dell'architettura descritta finora:



Abbiamo quindi che le istruzioni provengono dal Branch Target Buffer, **BTB**, dalle quali vengono poste in una componente di **prelievo** e una di **decodifica**.

Da qui i poi le istruzioni passano per le stazioni di prenotazione, che poi le inviano alle relative ALU, sulla base delle tabelle associate ai registri fisici.

In fine, in fondo a questo meccanismo si trova il Reorder Buffer, **ROB**, che contiene le istruzioni terminate ma non ancora scritte e permette l'esecuzione speculativa. Il significato del Reorder Buffer è sostanzialmente quello di realizzare un'ulteriore smistamento delle istruzioni a termine della loro esecuzione, che permette o meno il loro write back nei registri (che sappiamo coi registri ridenominati significa semplicemente aggiornare i registri logici) solo sotto le condizioni dell'esecuzione speculativa.

1.0.1 Istruzioni LOAD e STORE

L'esecuzione speculativa vista presenta problemi per quanto riguarda le istruzioni `LOAD` e `STORE`.

Prendiamo ad esempio il framento di codice:

```
1   cmp $1000, %rbx
2   jae fine
3   mov off(%rbx), %rax
4
5 fine:
```

Qui la scrittura va eseguita solo se `%rbx` contiene un valore inferiore a mille. Per le regole dei salti in avanti, il salto in `jae fine` si da per non fatto, e quindi il BTB fornisce l'istruzione successiva, cioè quella di accesso in memoria (che verrebbe tradotta in una `load`). capito 0, per le store come fa?

1.0.2 Istruzioni IN e OUT

Infine, una nota va fatta sulle istruzioni di I/O `in` e `out`: queste infatti hanno conseguenze su dispositivi esterni al processore per cui non ne si può fare l'esecuzione parallela o speculativa.

In fase di emissione di istruzioni di I/o, quindi, il processore si aspetta che tutte le istruzioni attualmente in esecuzione siano state eseguite, e solo dopo procede con l'emissione. e poi?

1.1 Vulnerabilità Meltdown e Spectre

Meltdown e **Spectre** sono state vulnerabilità dell'architettura Intel x86, scoperte nel 2017, che permettevano di superare le limitazioni normalmente imposte al codice in esecuzione in modalità utente.

Abbiamo detto che il programma dovrebbe poter ignorare ciò che accade nel processore a livello **microarchitetturale**, cioè che tutte le operazioni che l'architettura descritta finora compie ai fini di ottimizzare l'esecuzione devono risultare **invisibili** al programma.

Ciò che queste vulnerabilità hanno rivelato è che lo stato microarchitetturale, invece, non è veramente invisibile.

1.1.1 Meltdown

Ad esempio, per quanto riguarda la cache, abbiamo che si può in qualche modo capire se qualcosa è scritto in cache: bastera invalidare una cacheline, far eseguire il processo che vogliamo studiare, e quindi tentare un'accesso alla stessa cacheline: valutando il tempo necessario all'accesso (e magari facendo una media statistica) si potrà capira se quella cacheline è piena oppure no.

Questa era l'approccio adottato dalla vulnerabilità Meltdown.

```
togli_cacheline

mov indirizzo_vietato, %al # qui il programma muore
# da qui in poi si esegue in speculativa
shl $12, %al
mov vettore (%rax), %rbx

# questo si mette nell'handler del segmentation fault
controlla_cacheline
```

L'istruzione illegale `mov indirizzo_vietato, %al` causerà chiaramente l'arresto del programma, ma le due istruzioni successive verranno comunque eseguite dal processore in modalità speculativa, prima che si manifesti il fault.

La `controlla_cacheline`, invece, viene eseguita comunque attraverso una ridefinizione dell'handler di segmentation fault.

A questo punto, si ha quindi che l'indirizzo letto in `%al`, che il processo non sarebbe autorizzato a vedere, viene trasformato in un indice in un vettore, vettore il cui accesso si traduce nel riempimento di una cachline che rileviamo in `controlla_cacheline`.

Iterando questo processo su tutta la memoria kernel si riesce a ricostruirne una copia, e quindi effettivamente leggere tutta la memoria fisica.

Il problema oggi è stato risolto forzando l'invalidazione della cache ad ogni passaggio al kernel.

1.1.2 Spectre

La vulnerabilità Spectre si basa sulla natura stessa del branch prediction: l'utente può sfruttare la struttura del BTB, che abbiamo detto è effettivamente una cache senza controlli sulle collisioni, per "addestrare" il predittore di branch a fare scelte arbitrarie.

In questo modo si potrà in qualche modo "direzionare" l'esecuzione del kernel in luogo dei salti condizionali, e poi usando metodi come Meltdown capire che cosa il kernel ha fatto.

Anche questo problema si può risolvere invalidando una cache in fase di passaggio al kernel, e in particolare invalidando il BTB.