Architettura RISC-V

I processori RISC-V sono un esempio di architettura RISC. Furono sviluppati nel 2010 alla UC Berkeley. RISC-V è un'architettura molto regolare ed è stata progettata per un'implementazione efficiente della pipeline.

RISC-V a 32 bit

Istruzioni

Tutte le istruzioni hanno la dimensione di 32 bit, le istruzioni che manipolano i dati usano solo i registri. Le uniche operazioni che si fanno nella memoria sono il *load* ed il *store* per fare scambi tra memoria e registri, mai memoria-memoria. Quindi tutte le istruzioni operano sui registri es:

```
    add x1, x2, x3
```

Registri

Ci sono 32 registri da 32 bit si indicano con `x1, x2, x3, ..., x0` quest'ultimo contiene sempre 0.

Dati

I registri possono contenere byte, mezze parole e parole. Visto che i registri hanno capienza di 32 bit, se un dato è "più corto" si riempiono i bit rimanenti con 0 oppure si estende il segno.

Indirizzamento

Si usa l'indirizzamento immediato: `addi x2, x2, 004`
Oppure il displacement: `sw x1 00c(x2)`

Ma anche delle modalità derivabili:

- indiretto registro (displacement a 0): `sw x2, 000(x3)`
- assoluto (registro 0 come registro base): `lw x1, 0c4(x0)`

Formato istruzioni

RISC-V a 32 bit utilizza vari formati di istruzioni, ovviamente tutti a 32 bit:

- **Formato R**: utilizzato per le operazioni registro-registro e istruzioni aritmetiche e logiche (**6 campi** a dimensione fissa). Le istruzioni di tipo R sono: `add` e `sub`.

7 bit	5 bit	5 bit	3 bit	5 bit	7 bit
funz7	rs2	rs1	funz3	rd	codop

    - *funz7* e *funz3*: rappresenta il codice funzione che identifica la variate operativa (somma, sottrazione,...)
    - *rs2* e *rs1*: registri che contengono secondo e primo argomento
    - *rd*: registro destinazione riceve il risultato
    - *codop*: codice operativo identifica l'operazione aritmetica
- **Formato I**: utilizzato per operazioni immediate e caricamenti ad esempio `addi` e `lw`. 5 campi di dimensione costante.

12 bit	5 bit	3 bit	5 bit	7 bit
imm[11:0]	rs1	funz3	rd	codop

    - *imm11:0* : valore immediato di 12 bit

    - **salto incondizionato JARL** (*jump and link register*): utilizza la codifica di tipo I. L'indirizzo di destinazione si ottiene aggiungendo il valore immediato con segno al registro *rs1*, impostando a zero il bit meno significativo del risultato. L'indirizzo dell'istruzione successiva al salto viene scritto nel registro *rd*. Il registro *x0* può essere utilizzato come destinazione se il risultato non è richiesto.
- **Formato S**: utilizzato per le istruzioni di caricamento in memoria (*store*) per esempio `sw`.

7 bit	5 bit	5bit	3 bit	5 bit	7 bit
imm[11:5]	rs2	rs1	funz3	imm[4:0]	codop

    - Spiazzamento: di 12 bit è diviso in *imm11:5* e *imm4:0*

    - **sottoformato B**: tutte le istruzioni di salto condizionale utilizzano questo sottoformato. Il B-immediato a 12 bit codifica gli offset con segno in multipli di 2 e viene aggiunto al PC corrente per fornire l'indirizzo di destinazione.
- **Formato U**: utilizzato per istruzioni di caricamento in memoria (*store*).

20 bit	5 bit	7 bit
imm[31:12]	rd	codop

- LUI (*load upper immediate*): utilizzato per creare costanti a 32 bit, inserisce il valore immediato U nei primi 20 bit del registro di destinazione *rd* riempiendo i 12 bit più bassi con 0
- AUIPC (*add upper immediate to pc*): utilizzato per creare indirizzi relativi al PC, forma un offeset di 32 bit dall'immediato di 20 bit, riempiendo i 12 bit più bassi con 0, aggiunge questo offset al PC ed inserisce il risultato nel *rd*
    - **sottoformato J**: l'istruzione (**JAL**) utilizza il formato UJ, dove l'immediato codifica un offset con segno in multipli di 2 byte. L'offset viene esteso con segno e aggiunto al PC in modo da formare

l'indirizzo di destinazione del salto. JAL memorizza il risultato dell'indirizzo dell'istruzione successiva al salto nel *rd*. Per convenzione si usa `x1` come regisstro dell'indirizzo di ritorno e `x5` come registro di collegamento alternativo.

Ciclo esecutivo di un'istruzione

Il ciclo di esecuzione è suddiviso in 5 fasi, istruzione di esempio `add x8, x18, x8`:

- **IF**: Instruction Fetch
  - IR <-- Mem[PC]
  - PC, NPC (registro temporaneo) <-- PC + 4 byte (1 word)
- **ID**: Instruction Decode
  - A <-- Regs[rs1] = [x18]
  - B <-- Regs[rs2] = [x8]
  - in questo caso si usa il formato R
- **EX**: Execution
  - ALUOutput <-- [A] funct [B]
- **MEM**: Memory access
  - nessuna operazione
- **WB**: Write Back
  - Regs[rd] = x8 <-- [ALUOutput]

Altri esempi di ciclo esecutivo:

1. istruzione `lw x8, 1200(x9)`
   - IF
     - IR <-- Mem[PC]
     - PC, NPC <-- PC + 4
   - ID formato I
     - A <-- Regs[rs1] = [x9]
     - B <-- Regs[rs2] = [x8]
     - Imm <-- 1200
   - EX
     - ALUOutput <-- [A] + [Imm]
   - MEM
     - LMD <-- Mem[ALUOutput]

- WB
  - Regs[rd] = x8 <-- [LMD]
    L'istruzione `lw` dice: Al registro di destinazione `x8` assegno il valore contenuto all'indirizzo di memoria `x9 + 1200`, per questo motivo viene svolta la fase MEM. Il WB si svolge perchè bisogna modificare il contenuto di un registro. Per un'istruzione di `sw` (store word), invece, non serve il WB visto che l'obbiettivo non è modificare un registro ma la RAM, per esempio `sw x8, 1200(x9)` modifica la locazione di memoria all'indirizzo `x9 + 1200` assegnando il valore contenuto nel registro `x8`.

Datapath

Caratterische principali:

- unità di memoria condivisa, singola ALU condivisa tra le istruzioni e le connessioni tra le unità condivise
- unità funzionali condivise che richedono l'aggiunta o l'ampliamento di multiplexor, nuovi registri temporanei che trattengono i dati tra i cicli di clock della stessa istruzione
- registro delle istruzioni (IR), registro dei dati di memoria (MDR), A, B e ALUOut

Pipeline

La pipeline è composta da:

- 1 ciclo di clock per fase/stadio
  - 1 istruzione in 5 cicli di clock
  - 1 ciclo di clock completa 5 fasi di 5 istruzioni diverse
  - ogni ciclo di clock termina un'istruzione se a regime
- ***Pipeline registers*** (*pipeline latches*)
  - i dati utili a fasi successive sono memorizzati nel registro successivo
  - i registri memorizzano sia dati che segnali di controllo
  - in ogni istante registri diversi contengono dati relativi ad istruzioni diverse

Le fasi di IF e ID non dipendono dai valori dei segnali di controllo. Nella fase di ID si possono calcolare i segnali corretti per le fasi successive e propagati attraverso i registri di pipeline.

Dipendenze dei dati (Pipeline)

Nella Pipeline RISC-V è possibile individuare tutte le dipendenze dai dati nella fase di ID:

- Se si rileva una dipendenza dai dati per un’istruzione, questa va in stallo prima di essere rilasciata, ossia quando una passa dalla fase ID a quella EX
- Sempre nella fase ID, è possibile determinare che tipo di data forwarding adottare per evitare lo stallo ed anche predisporre gli opportuni segnali di controllo

- Per semplicità, considereremo solo data forwarding per un’operazione che deve essere eseguita nella fase EX

Situazione	Esempio di codice	Azione

Nessuna dipendenza	LW x1, 45(x2)
	ADD x5, x6, x7
	SUB x8, x6, x7
	OR x9, x6, x7
		Non occorre fare nulla

Dipendenza che richiede uno stallo	LW x1, 45(x2)
	ADD x5, x1, x7
	SUB x8, x6, x7
	OR x9, x6, x7
		Opportuni comparatori rilevano l'uso di x1 in ADD ed evitano il rilascio di ADD

Dipendenza risolvibile con un forwarding	LW x1, 45(x2)
	ADD x5, x6, x7
	SUB x8, x1, x7
	OR x9, x6, x7
		Opportuni comparatori rilevano l'uso di x1 in SUB e inoltrano il risultato della load alla ALU in tempo per la fase EX di SUB

Dipendenza con accessi in ordine	LW x1, 45(x2)
	ADD x5, x6, x7
	SUB x8, x6, x7
	OR x9, x1, x7
		Non occorre fare nulla perchè la lettura di x1 in OR avviene dopo la scrittura del dato caricato

Condizioni per decicere come effettuare il forwarding:

- Tutti i dati provengono:
  - dalla memoria dati
  - dall'output della ALU
  - sono in registri MEM/WB oppure EX/MEM
- Tutti i dati sono diretti verso:
  - l'input della ALU
  - verso registri ID/EX

Quindi occorre confrontare e se sono uguali va fatta la propagazione.