MicroArchitettura MIC-1

Filippo Pinna

04/04/2019

Chapter 1

Microarchitettura Mic-1 e MAL

1.1 Livello di Microarchitettura

Implementato sopra il livello logico digitale troviamo il livello di microarchitettura. In specifico tratteremo la microarchittettura di processore Mic-1 realizzata da Andrew S. Tanenbaum. Il Mic-1 un unità di controllo caratterizzata da varie componenti logiche il cui compito finale è quello di eseguire un microprogramma. Il microprogramma prima di poter essere eseguito verrà interpretato da un ISA (Instruction Set Architecture) ovvero l'architettura dell'insieme di istruzioni, nel nostro caso tratteremo l'ISA della Java Virtual Machine: l'IJVM. L'IJVM (Integer Java Virtual Machine) come dice il nome è un sottoinsieme di istruzioni della JVM su integer, l'insieme di queste microistuzioni andrà poi a costistuire i microprogrammi. Il Mic-1 per poter consentire la scrittura dell'interprete IJVM ha un suo linguaggio microcodice MAL (Micro-Assembly Language). Il modello di esecuzione si definisce di fetch-decode-execute, si preleva istruzione per istruzione, si decodifica il codice e si manda in esecuzione.

1.1.1 Percorso Dati

Per percorso dati si intende l'insieme dei collegamenti di input/output alla ALU. I collegamenti son formati dai Bus Dati che sono i canali che permettono la comunicazione e trasporto dell'informazione e rappresentano quelli che fisicamente sono i cavi dei circuiti.Esistono due Bus da 32 linee ciascuno, il Bus B collegato all'output dei registri e all'input della ALU, il Bus C collegato all'output dello shifter e all'input dei registri. Nel nostro percorso dati son presenti dei registri a 32 bit che sono selezionati da due linee di controllo, una per abilitare il bus B e l'altra per abilitare il bus C. I dati in uscita vengono eleborati dalla ALU che genera un output a sua volta eleborato dallo shifter. La ALU contiene al suo interno 32 circuiti combinatori. Ogni circuito combinatorio è capace di es-

eguire istruzioni logiche di AND, OR e NOT e istruzioni aritmetiche di somma. In entrata alla ALU vi sono 6 linee di controllo. Le prime due determinano l'operazione della ALU, ENA e ENB abilitano i due input (enable A,B), INVA inverte l'input di sinistra e INC crea un riporto (somma 1 al risultato) nel bit meno significativo. La ALU ha due canali in ingresso A e B. A è collegato al registro H (holding) di mantenimento, B è collegato al Bus B che quindi riceve output da 9 registri. Attraverso lo Shifter il risultato uscente dalla ALU potrebbe non subire nessuna variazione, oppure potrebbe usare uno Shift Left Logical o uno Shift Right Logical. Lo shift a sinistra (SLL8) trasla il valore a sinistra di un byte e imposta gli 8 bit meno significativi a 0; lo shift a destra (SRA1) trasla il valore di 1 bit a destra e lascia inalterati i bit meno significativi.

1.1.2 Ciclo di Clock

All'inizio di ogni ciclo di clock viene generato un breve impulso. Il fronte di discesa dell'impulso è la parte conosciuta e significativa che gestirà le porte logiche. Il fronte di discesa si divide in una serie di intervalli:

1. Δw: Viene selezionato il registro e il suo contenuto viene mandato al bus B
2. Δx: Vengono impostati bus B e successivamente il registro H che manterrà i contenuti (in fase di stabilizzazione) prima che vengano inviati alla ALU.
3. Δy: ALU e Shifter (che son rimasti sempre attivi) ricevono i dati da elaborare.
4. Δz: I risultati passano per il Bus C che li porta ai relativi registri che si aggiorneranno coi nuovi dati/valori.

Al successivo impulso sul fronte iniziale di salita i risultati giungono effettivamente nei registri. Sempre in questo istante il Bus B non riceve più alimentazione dal registro che lo alimentava, per prepararsi al prossimo ciclo. E' importante sottolineare l'importanza di questa rigida e necessaria temporizzazione perchè determina l'intervallo di esecuzione e quindi il costo in tempo. Ai singoli componenti sono estranei i sottocicli di clock, gli stiamo considerando tali perchè rappresentano i limiti in cui i determinati valori possono essere considerati validi. E' compito del progettista assicurarsi che il ciclo di clock avvenga in un tempo sufficientemente esatto,

1.1.3 Memoria

Ci sono diversi modi con cui l'architettura si interfaccia alla memoria: una porta a 32 bit con indirizzi in parole e una porta di 8 bit con indirizzi in byte. I registri MAR (Memory Address Register) e MDR (Memory Data Register) controllano la porta a 32 bit. La porta a 8 bit è controllata dal PC (Program Counter) che legge 1 byte negli 8 bit meno significativi del registro MBR (Memory Buffer Register). Ciascun registro è controllato da uno o due segnali di controllo, tra

questi riconosciamo un segnale di abilitazione output verso il bus B. Il registro MAR non avendo connessione al Bus B e il registro H che è sempre abilitato (essendo l'unica uscita) non hanno questo segnale. L'altro tipo di segnale di controllo per i registri ne determina l'abilitazione a ricevere dati in entrata dal Bus C. Quest'ultimo non è presente nel registro MBR perchè non può esser caricato dal Bus C. Rispettivamente la combinazione di MAR/MDR viene usata per leggere/scrivere dati del livello ISA mentre la combinazione PC/MBR per leggere l'eseguibile a livello ISA. Nell'implementazioni reali esiste un unnica memoria, è necessario quindi consentire a MAR di contare il numero di parole dato che bisogna distinguere dal bus degli indirizzi che specificano byte. I bit di MAR non vengono direttamente mappati sulle 32 linee (da 0 a 31). Si esegue un "shift" per cui ilbit 0 di MAR si collega alla linea 2 del bus indirizzi e i due bit più alti vengono scartati in quanto non necessariamente significativi. I dati letti dalla porta ad 8 bit vengono restituiti a MBR (registro a 8 bit), a sua volta MBR viene copiato nel Bus B in due modi: con o senza segno (determinato da due segnali di controllo a seconda del metodo). Nella parola a 32 bit quando viene richiesto un valore senza segno negli 8 bit meno significativi si memorizzerà il valore di MBR e i restanti 24 verranno posti a 0. Per convertire il registro MBR a 8 in 32 bit si può anche trattarlo come se fosse un valore compreso tra -128 e +127 e convertire quindi questo ultimo valore in una parola da 32 bit. Questa operazione si chiama estensione del segno e si esegue duplicando i bit del segno di MBR nei 24 bit più alti del Bus B. Se il bit più a sinistra di MBR vale 0 i 24 bit più alti saranno tutti a 0, se vale 1 verrano posti a 1.

1.2 MAL (Micro Assembly Language)

Possiamo distinguere 29 segnali divisibili in 5 gruppi per scopo.

- 9 segnali di controllo scrittura del Bus C sui registri
- 9 segnali di abilitazione dei registri sul Bus B
- 8 segnali per funzioni ALU e Shifter
- 2 segnali per indicare alla memoria di leggere/scrivere tramite MAR/MDR
- 1 segnale per indicare il prelievo della memoria tramite PC o MBR

In un ciclo di percorso dati i dati passano dai registri al bus B che li porta sulla ALU che a sua volta rende il risultato ai registri attraverso il Bus C. Un'eventuale lettura dati (su una delle due porte) trasmette dati che non possono essere usati al ciclo successivo ma al ciclo successivo+1. Questo perchè i dati vengono caricati nel MAR a fine ciclo quindi non possiamo aspettarci che siano atomicamente disponibili in MDR all'inizio del ciclo successivo. L'operazione di lettura richiede un ciclo e si possono eseeguire letture in sequenza durante cicli consecutivi. Si possono usare allo stesso tempo le porte di memoria ma leggere e scrivere simultaneamente genera risultati indefiniti. Mentre sul Bus B è abilitato sempre

e solo un registro, su quello C ne possiamo abilitare diversi. L'informazione del Bus B viene codificatain 4 bit e tramite un decodificatore (4 a 16) si generano 16 segnali di controllo (7 inutilizzati). Il percorso dati in definitiva si controlla con 9+4+8+2+1=24 segnali(bit). In aggiunta a questi 24 bit di controllo ciclo sono necessari anche i campi NEXT ADDRESS e JAM per capire cosa dev'essere fatto nel ciclo successivo. Per determinare i collegamenti e incroci tra chip (per ottimizzarli) si è sviluppato un possibile formato di 6 gruppi di un totale di 36 segnali.

- Addr - Contiene indirizzo (address) di una possibile successiva Microistruzione
- JAM - Determina la selezione della successiva Microistruzione
- ALU - Seleziona le funzioni di ALU e Shifter
- C - Seleziona i registri sui quali il Bus C deve scrivere
- Mem - Seleziona la funzione della Memoria
- B - Seleziona codifica sorgente Bus B (0=MDR, 1=PC, 2=MBR, 3=MBRU, 4=SP, 5=LV, 6=CPP, 7=TOS, 8=OPC, 9-15 none)

1.3 Unità di controllo microprogrammata

Nella descrizione della nostra architettura molte volte useremo circuiti standard a scopi didattici non ottimizzati. Introduciamo ora il sequenzializzatore che determina la sequenza di operazioni per eseguire singolarmente le istruzioni ISA. Il sequenzializzatore ad ogni ciclo ci dà lo stato di ogni segnale di controllo del sistema e l'indirizzo della prossima microistruzione da eseguire. Il Mic-1 si può suddividere in due grandi parti: il percorso dati che è stato visto e la sezione di controllo. Il nucleo della sezione di controllo è la memoria di controllo. La memoria di controllo è un circuito (una serie di porte logiche) che memorizza Microistruzioni (da non confondere con istruzioni ISA). Nel Mic-1 la memoria è grande 512 parole (consistono nei 36 bit della Microistruzione). Le sequenze dei microprogrammi tendono ad essere brevi e per avere una flessibilità maggiore ogni microistruzione (solitamente) specifica il suo succes sore. La memoria di controllo ha: un registro degli indirizzi MPC (MicroProgram Counter) ,il cui nome può risultare ambiguo perchè le istruzioni in realtà non hanno un ordine preciso, e un registro dei dati detto MIR (MicroIstruction Register). Il MIR contiene la microistruzione in esecuzione corrente, la quale determinerà il percorso dati. Il percorso totale del Mic-1 si può così descrivere:

- Ad inizio ciclo di clock (fronte di discesa) la parola nella memoria di controllo puntata da MPC viene trasferita in MIR (Δw tempo)
- Negli istanti dopo i segnali si propagano nel bus dati (registri ¿ Bus B ¿ ALU ¿ Bus C) fino ad arrivare al secondo sottociclo

- Dopo un secondo Δy successivo tutto si stabilizza nel circuito e i valori di N e Z (uscite ALU) vengono salvati in una coppia di flip-flop ad 1 bit.
- I bit dei flip-flop vengono salvati in un tempo di fine ciclo (fronte salita clock)
- In un 3 sottociclo si svolgeranno le attività della ALU e dello Shifter (l'output della ALU non si memorizza ma finisce nello Shifter)
- Dopo un ulteriore Δz dallo Shifter si va al Bus C e l'output raggiunge i registri
- Nel quarto sottociclo vengono caricati i registri e i flip-flop N e Z, i risultati delle operazioni di memoria precedenti son quindi disponibili e lo stato di MPC viene aggiornato.

Non necessariamente l'ordine delle istruzioni riflette l'ordine che hanno nella memoria di controllo, è compito del microprogramma determinare l'ordine. Innanzi tutto il MIR deve essere diventato stabile. In un primo momento i 9 bit del campo NEXT ADDRESS vengono copiati in MPC, durante ciò si vede il contenuto del campo JAM e se il suo contenuto è 000 non si esegue altro; a fine copia MPC punta alla microistruzione successiva. Se uno o più bit di JAM valgono 1 (JAMN, JAMZ o JMPC) si eseguono ulteriori azioni. Con JAMN abilitato si calcola l'OR logico con il flip-flop N e si salva il risultato nel bit più significativo di MPC. Si fa lo stesso se è abilitato JAMZ ma l'OR si calcola con il flip-flop Z. Se entrambi sono abilitati si calcola l'OR con entrambe. Sul fronte di salita del clock il Bus B non viene alimentato e gli output della ALU possono non vengono considerati corretti; N e Z ci sono d'aiuto in questo caso perchè salvano i flag della ALU. Per stabilizzare i valori nei flip-flop si usa la funzione nel "Bit Alto": F= (JAMZ AND Z) OR (JAMN AND N) OR NEXT ADDRESS[8]. Il risultato poi va nel MPC che potrà assumere il valore di NEXT ADDRESS oppure il valore di NEXT ADDRESS nel quale il bit più significativo viene calcolato in OR con 1(0x100) nel caso in cui NEXT ADDRESS sia minore o uguale a 0xFF. Se invece JMPC è abilitato (posto ad 1) gli 8 bit di MBR vengono collegati in OR con gli 8 bit meno significativi del campo NEXT ADDRESS della microistruzione precedente e il risultato viene mandato a MPC. Sempre quando JMPC vale 1 viene fatto l'OR di MBR e NEXT ADDRESS in una parte di circuito chiamata "O" ; nel caso in cui valga 0 passa direttamente a MPC. MBR tipicamente contiene codice operativo e JMPC ci da l'indirizzo della prossima istruzione da eseguire (utile per i salti). Vediamo ora le tempistiche in termini di sottociclo:

- Durante il primo sottociclo (primo fronte di discesa del clock) la microistruzione contenuta in MPC viene caricata nel MIR
- Nel secondo sottociclo da MIR i segnali si propagano verso i registri e se ne seleziona uno che poi viene caricato dentro MIR

- Al terzo sottociclo ALU e Shifter possono operare e dare un risultato stabile
- Nel quarto sottociclo Bus C, bus di memoria e valori della ALU diventano stabili.
- Sul fronte di salita il contenuto del Bus C viene caricato nei registri, i flip-flop N e Z vengono caricati, MBR e MDR ottengono eventualmente risultati del ciclo di percorso dati precedente. Quando MBR e MDR sono stabili viene caricato MPC con la prossima microistruzione (da notare che MPC non va caricato sinchè non sono pronti i registri da cui dipende)
- Quando il clock scende nuovamente MPC da inizio ad un nuovo ciclo

MPC può non venire implementato come registro perchè in realtà non c'è bisogno di memorizzare nessun valore al suo interno avendo tutti già tutti i segnali, per questo si può considerare un registro virtuale.

Chapter 2

IJVM

2.1 Concetti Preliminari

Il livello ISA della nostra macchina interpretato dal MAL ed eseguito sarà l'IJVM. Per memorizzare le variabili si usa una parte della memoria chiamata stack al cui interno le variabili non hanno un indirizzo assoluto. Viene impostato un registro LV (Local Variable) che punta alla base delle variabili locali per la procedura attuale. UN altro registro SP (Stack Pointer) punta alla parola in cima allo stack. Per far riferimento alle variabili locali si fornisce il loro offset rispetto a LV. La struttura dati delimitata da LV e Sp viene chiamata: Blocco delle variabili locali. Considerando due procedure A e B (dove A richiama B) nello stack troveremo memorizzato il blocco di A con sopra il blocco di B. Quando verrà richiamata B vedremo LV che punta alla base del blocco B e SP in cima allo stesso blocco. Alla chiusura di B il controllo passa ad A ed LV e Sp punteranno al blocco A. Lo stack può pure memorizzare gli operandi durante un operazioni matematiche e il blocco riservato a questo utilizzo si chiama Stack degli operandi. Durante operazioni matematiche ad SP viene incrementato per puntare all'indirizzo del primo operando sullo stack, LV rimane invariato.

2.2 Modello di Memoria IJVM

La memoria dell'architettura IJVM consiste in un array di 4 Giga Byte o circa 1 miliardo di parole di 4 byte. A livello ISA-IJVM non sono visibili gli indirizzi di memoria assoluti, si usano indirizzi che poi serviranno come base per dei puntatori. Per accedere alla memoria è quindi necessario indicizzarla tramite questi puntatori. La parte di stack della memoria non può superare una certa dimensione stabilita all'inizio al momento di compilazione. Possiamo dividere le parti di memoria IJVM in:

1. Porzione costante di memoria (Costant Pool): viene chiamata costante perchè caricata prima che il programma venga mandato in esecuzione e rimane tale e non modificabile dopo. Contiene costanti, stringhe e puntatori

a memoria. Esiste un registro CPP (Costant Pool Pointer) che contiene l'indirizzo della prima parola della Porzione costante di memoria

2. Blocco delle variabili locali (Local Variable Frame): questa parte di memoria è necessaria alla allocazione di spazio per le chiamate dei metodi. Inizialmente vengono memorizzati i parametri dei metodi. Il registro LV (Local Variable) contiene l'indirizzo della prima locazione del blocco.

3. Stack degli operandi (Operand Stack): Sopra il Blocco delle variabili locali si trova lo stack degli operandi. Esiste un registro SP (Stack Pointer) che contiene l'indirizzo della parola in cima allo stack. E' importante notare che a livello implementativo lo Stack degli operandi fa parte del Blocco delle variabili.

4. Area dei Metodi (Method Area): è l'area dove risiede il programma da eseguire. L'indirizzo dell'istruzione successiva da prelevare (fetching) è contenuto nel registro PC (Program Counter). A differenze delle altri parti quest'area è rappresentata come un vettore di byte.0

I registri CPP, LV ed SP sono tutti puntatori a parole (4 byte) e non a singoli byte, un eventuale incremento di SP corrisponde a prelevare la parola successiva e l'inizio di una lettura. PC invece contiene l'indirizzo espresso in byte, la sua porta infatti è larga 1 byte. Il prelevamento (Fetching) del byte successivo corrisponde ad un incremento di PC e l'inizio di una lettura.

2.3 Set Istruzioni IJVM

Le istruzioni IJVM hanno una codifica esadecimale, sono composte da un codice operativo (assembly) e in alcuni casi un operando, che può essere una costante o uno spiazzamento. Di seguito son qui listate le istruzioni, i campi vuoti negli Operandi indicano la non presenza.

Hex	Codice	Operandi	Funzione
0x10	BIPUSH	byte	Scrive un byte in cima allo stack
0x59	DUP		Legge e inserisce un ulteriore volta la parola in cima allo stack (duplica)
0xA7	GOTO	offset	Salto incondizionato
0x60	IADD		Preleva le prime due parole in cima allo stack e restituisce la loro somma
0x7E	IAND		Preleva le prime due parole in cima allo stack e restituisce il loro AND
0x99	IFEQ	offset	Preleva la parola in cima allo stack e fa un salto se vale 0
0x9B	IFLT	offset	Preleva la parola in cima allo stack e fa un salto se ha valore negativo
0x9F	IF_ICMPEQ	offset	Preleva la parola in cima allo stack e fa un salto se sono uguali
0x84	IINC	varnum const	Somma una costante a una variabile locale
0x15	ILOAD	varnum	Scrive una variabile locale in cima allo stack
0xB6	INVOKEVIRUAL	disp	Invoca un metodo
0x80	IOR		Preleva le prime due parole in cima allo stack e restituisce il loro OR
0xAC	IRETURN		Termina un metodo restituendo un valore intero
0x36	ISTORE	varnum	Preleva una parola in cima allo stack e la memorizza in una variabile locale
0x64	ISUB		Preleva le prime due parole in cima allo stack e restituisce la loro differenza
0x13	LDC_W	index	Scrive in cima allo stack una costante prelevata dalla porzione costante di memoria
0x00	NOP		Nessuna operazione
0x57	POP		Rimuove una parola dalla cima dello stack
0x5F	SWAP		Scambia la posizione delle due parole in cima allo stack
0xC4	WIDE		Istruzione Prefissa: l'istruzione seguente ha un indice a 16 bit