MIC-3

Nel MIC-3 viene introdotta una macchina con un maggiore parallelismo, poiché in MIC-2 dopo aver introdotto l'IFU la nostra microarchitettura lavora per la maggior parte in un modo ancora troppo sequenziale. In questa microarchitettura vengono introdotti in mezzo a tutti e 3 i bus dei latch(registri) che verranno chiamati latch A, B e C che dividono il datapath in tre parti distinte che possono funzionare indipendentemente l'una dall'altra. Con questa nuova microarchitettura abbiamo adesso 3 cicli al posto di uno solo: il primo che carica i valori dai registri al bus B e al bus A e più precisamente nei latch A e B, il secondo che carica i valori dai latch all'ALU e per caricare il risultato nel latch C e infine il terzo che carica il valore dal latch C ai registri. Ciascuna di queste parti si chiamerà passo elementare.

Qual'è il vantaggio dei latch?

Il vantaggio dei latch è che adesso possiamo usare tutte le parti del datapath durante ogni ciclo e inoltre possiamo accelerare il clock dato che il ritardo massimo è ora più corto.

Da non sottovalutare che dato che abbiamo assunto che letture e scritture di memoria possono essere soddisfatte dalla cache di primo livello e dato che questa cache è realizzata con lo stesso materiale dei registri, continueremo ad assumere che un'operazione di memoria richieda un solo ciclo. Da notare bene inoltre che spezzando il datapath in 3 parti saremo in grado di usare l'ALU in ogni ciclo.

SWAP

Implementazione swap codice mic-2

	Swap1	Swap2	Swap3	Swap4	Swap5	Swap6
Cy	MAR=SP–1;rd	MAR=SP	H=MDR;wr	MDR=TOS	MAR=SP–1;wr	TOS=H;goto (MBR1)
1	B=SP					
2	C=B–1	B=SP				
3	MAR=C; rd	C=B				
4	MDR=Mem	MAR=C				
5			B=MDR			
6			C=B	B=TOS		
7			H=C; wr	C=B	B=SP	
8			Mem=MDR	MDR=C	C=B–1	B=H
9					MAR=C; wr	C=B
10					Mem=MDR	TOS=C
11						goto (MBR1)

Nel ciclo 1 iniziamo con **Swap1** copiando SP in B(nel latch B). Nel ciclo 2 effettuiamo la sottrazione (B-1). Nel ciclo 3 il risultato viene memorizzato in MAR e alla fine del ciclo inizia l'operazione di lettura. La lettura ci metterà un ciclo per concludersi e sarà completata alla fine del ciclo 4 e quindi il valore in MDR non può essere letto prima del ciclo 5.

Tornando al ciclo 2 si inizia con a **Swap2** con l'assegnamento di SP nel latch B. Nel ciclo 3 il valore di SP verrà passato dentro all'ALU e infine nel ciclo 4 verrà memorizzato nel registro MAR.

Nel ciclo vorremo occuparci dello **Swap3** ma non è possibile poiché perché la sua prima operazione consiste nel far passare MDR nel latch B e questo non si può fare perché MDR sarà disponibile dopo la lettura effettuata nel ciclo 4 dalla **Swap1** e quindi sarà disponibile a partire dal ciclo 5. Finché MDR non sarà disponibile ci troviamo in una situazione di stallo.

Dipendenza effettiva

La situazione in cui un passo elementare non può iniziare, dato che è in attesa di un risultato che non è ancora stato prodotto da un precedente passo elementare si chiama dipendenza.

RAW è una dipendenza(read after write) e indica che un passo elementare vuole leggere un registro che non è ancora stato scritto.

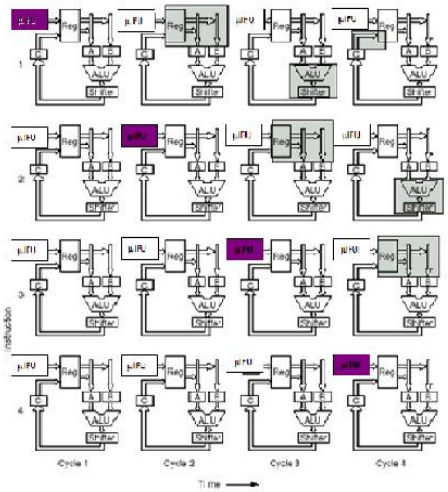
WAR è una dipendenza(write after read) e un indica che un passo elementare vuole scrivere un registro che non è ancora stato letto.

WAW è una dipendenza(write after write) e indica che gli aggiornamenti di un dato registro deve avvenire nell'ordine giusto.

Tempo dei cicli: confronto tra MIC-2 e MIC-3

Il programma MIC-3 richiede più cicli di quello di MIC-2, ma viene eseguito più velocemente. Se indichiamo con Δt ns il tempo di ciclo di MIC-3, allora MIC-3 impiega 11 Δt ns per eseguire Swap. MIC-2 impiega 6 cicli, ciascuno dei quali è lungo $3\Delta t$, per un totale di $18\Delta t$. L'uso della pipeline ha reso la macchina più veloce, anche se è necessario rimanere in stallo per risolvere una dipendenza.

Uso pipeline in MIC-3



Sopra si indica il ciclo che si sta eseguendo mentre a lato indichiamo la microistruzione che si sta effettuando. Se ne indicano 4 perché dopo il 4 ciclo sarebbe stata la stessa cosa con tutte le parti del datapath che funzionano contemporaneamente in modo indipendente una dall'altra. Questa organizzazione rappresenta una pipeline a 4 stadi in cui uno preleva le istruzioni, uno accede agli operandi dai registri ai latch, una che esegue le operazioni della ALU (che è anche la fase più lunga) e uno scrive i risultati nei registri.

La prima colonna schematizza quello che avviene nel primo ciclo la seconda quello che avviene del secondo e cosi via assumendo che non succedano delle dipendenze. La regione viola indicata della prima microistruzione indica che la IFU sta prelevando la prima microistruzione (ad esempio Swap1). Durante il ciclo 2 i registri caricano sui bus A e B(e quindi sui latch) i valori richiesti da quella microistruzione e parallelamente la IFU è impegnata a prelevare la seconda microistruzione(ad esempio Swap2). Durante il ciclo 3 la prima microistruzione usa l'ALU e lo shifter per eseguire la propria operazione mentre la seconda microistruzione sta caricando i valori dai registri ai bus A e B (e quindi sui latch A e B) e allo stesso tempo la terza microistruzione(ad esempio Swap 3) viene

prelevata dall'IFU. Infine nel ciclo 4 ci sono quattro istruzioni in corso d'esecuzione. I risultati della prima microistruzione cominciano ad essere memorizzati nei registri, l'ALU comincia a lavorare per la seconda microistruzione, i registri caricano i valori richiesti dalla terza microistruzione sui bus A e B e infine l'IFU preleva la quarta microistruzione.

MIC-4

L'ultima microarchitettura le cui parti principali sono la IFU, l'unità di decodifica, l'unità di accodamento e abbiamo 4 MIR indipendenti.

IFU

La IFU che preleva dalla memoria parole e mantiene i vari registri MBR, alimenta con il flusso di byte una nuova componente l'unità di decodifica.

Unità di decodifica

Questa unità è dotata di una ROM interna e ogni riga di questa unità ha 2 parti: la lunghezza dell'istruzione IJVM corrispondente e un indice relativo ad un'altra ROM, la ROM delle micro-operazioni.

La lunghezza dell'istruzione IJVM è utilizzata per permettere all'unità di decodifica di analizzare il flusso di byte entrante e suddividerlo in istruzioni, sapendo in ogni momento quali byte rappresentano i codici operativi e quali gli operandi. Ad esempio se l'istruzione è lunga 1 byte(per esempio la POP), allora l'unità di decodifica sa che il byte successivo è un codice operativo. Se invece l'istruzione corrente è lunga 2 byte, l'unità di decodifica sa che il byte successivo è un operando, seguito da altro un codice operativo. Quando invece incontra il prefisso WIDE, l'unità trasforma il byte successivo in uno speciale codice operativo più ampio(ad esempio WIDE+ILOAD diventa WIDE_ILOAD).

L'unita di decodifica in via alla componente successiva(unità di accodamento), l'indice relativo alla ROM delle micro-operazioni che ha trovato nella sua tabella.

Unità di accodamento

L'unità di accodamento riceve dall'unità di decodifica l'indice relativo alla ROM delle micro-operazioni. Questa unità contiene 2 tabelle interne, una in una RAM e l'altra in una ROM. Nella ROM vi è contenuta il microprogramma, in cui ciascuna istruzione IJVM ha un certo numero di elementi consecutivi, chiamati micro-operazioni. Le micro-operazioni sono simili alle microistruzioni tranne per il fatto che sono prive dei campi NEXT-ADDRESS e JAM, ma hanno un nuovo campo per specificare l'input del bus A. Sono presenti 2 nuovi bit: Final e Goto. Il bit Final viene impostato a 1 nell'ultima micro-operazione IJVM di ciascuna sequenza per indicarne la fine. Il bit Goto viene impostato a 1 per indicare le micro-operazioni che effettuano microdiramazioni condizionali. Queste micro-operazioni hanno un formato diverso da quello delle normali micro-operazioni ed è costituito dai bit JAM e da un indice relativo alla ROM delle micro-operazioni.

Funzionamento unità di accodamento

Questa unità riceve dall'unità di decodifica un indice della ROM delle micro-operazioni, cerca la micro-operazione corrispondente e la copia in una coda interna. Successivamente copia nella coda anche le 2 micro-operazioni seguenti. L'unità continua in questo modo finché non incontra una micro-operazione in cui il bit Final vale 1; in questo caso copia e si arresta. L'unità di accodamento, assumendo che non abbia mai incontrato una micro-operazione con il bit Goto attivo e che ci sia ancora molto spazio nella coda, rispedisce all'unità di decodifica un segnale di conferma. Quando vede la conferma, l'unità di decodifica spedisce all'unità di accodamento l'indice della successiva istruzione IJVM. In questo modo la sequenza d'istruzioni IJVM in memoria viene convertita in una sequenza di micro-operazioni in una coda.

Registri MIR

Queste micro-operazioni alimentano i registri MIR, che spediscono i segnali che permettono di controllare il datapath. Esiste tuttavia un altro fattore che dobbiamo considerare: i campi delle micro-operazioni non sono attivi nello stesso momento. Infatti i campi A e B sono attivi nel secondo ciclo, il campo C lo è nel terzo e ogni operazione di memoria avviene si svolge nel quarto ciclo. Per permettere che ciò funzioni correttamente abbiamo 4 registri MIR indipendenti.

All'inizio di ciascun ciclo di clock MIR3 viene copiato MIR4, MIR2 in MIR3, MIR1 in MIR2 e in MIR1 viene caricato con una nuova micro-operazione proveniente dalla coda delle micro-operazioni. Ogni MIR emette i propri segnali, ma non tutti vengono utilizzati. I campi A e B provenienti da MIR1 selezionano i registri che guidano i latch A e B, mentre il campo ALU in MIR1 non viene usato e non è connesso a nessuna componente del datapath. In MIR2 per esempio non vengono usati i campi A,B,C e MEM ma viene usato il campo ALU che viene collegato al datapath. E cosi via per gli altri registri MIR.

Microdiramazioni

Alcune istruzioni IJVM, come IFLT, richiedono di effettuare dei salti condizionati in base, per esempio al bit N. Quando si effettua una microdiramazione la pipeline non può continuare. Per gestire questo caso abbiamo aggiunto alla micro-operazione il bit Goto. Quando incontra una micro-operazione che contiene il bit Goto attivo, mentre la sta copiando all'interno della coda, l'unità di accodamento capisce in anticipo che ci sarà un problema e non spedisce all'unità di decodifica il segnale di conferma. Il risultato è che la macchina rimarrà in stallo finché la microdiramazione non sarà risolta. Probabilmente alcune istruzioni IJVM successive alla diramazione saranno già state portate nell'unità di decodifica(ma non all'unità di accodamento) nel momento in cui si incontra una micro-operazione con il bit Goto attivo e non viene rispedito il segnale di conferma(cioè il permesso di continuare). Per riportare un po' di ordine nella situazione e ritornare sui propri passi è necessario disporre di hardware speciali.

Riassumendo il MIC-4 effettua automaticamente il prefetch di un flusso di byte dalla memoria, li decodifica trasformandoli in istruzioni IJVM che converte in una sequenza di micro-operazioni utilizzando una ROM e infine accoda queste micro-operazioni per poterle utilizzare quando sarà necessario.