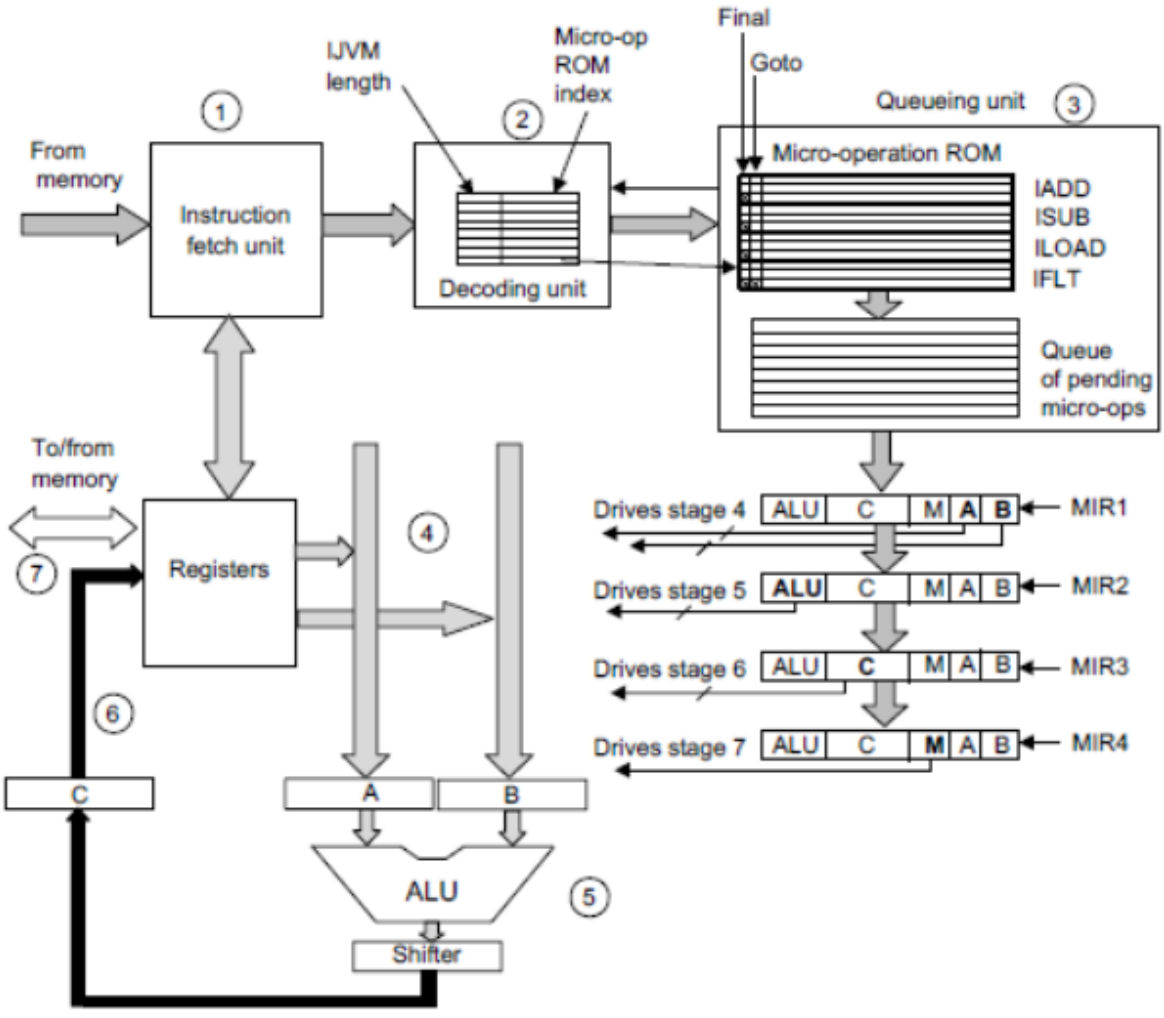
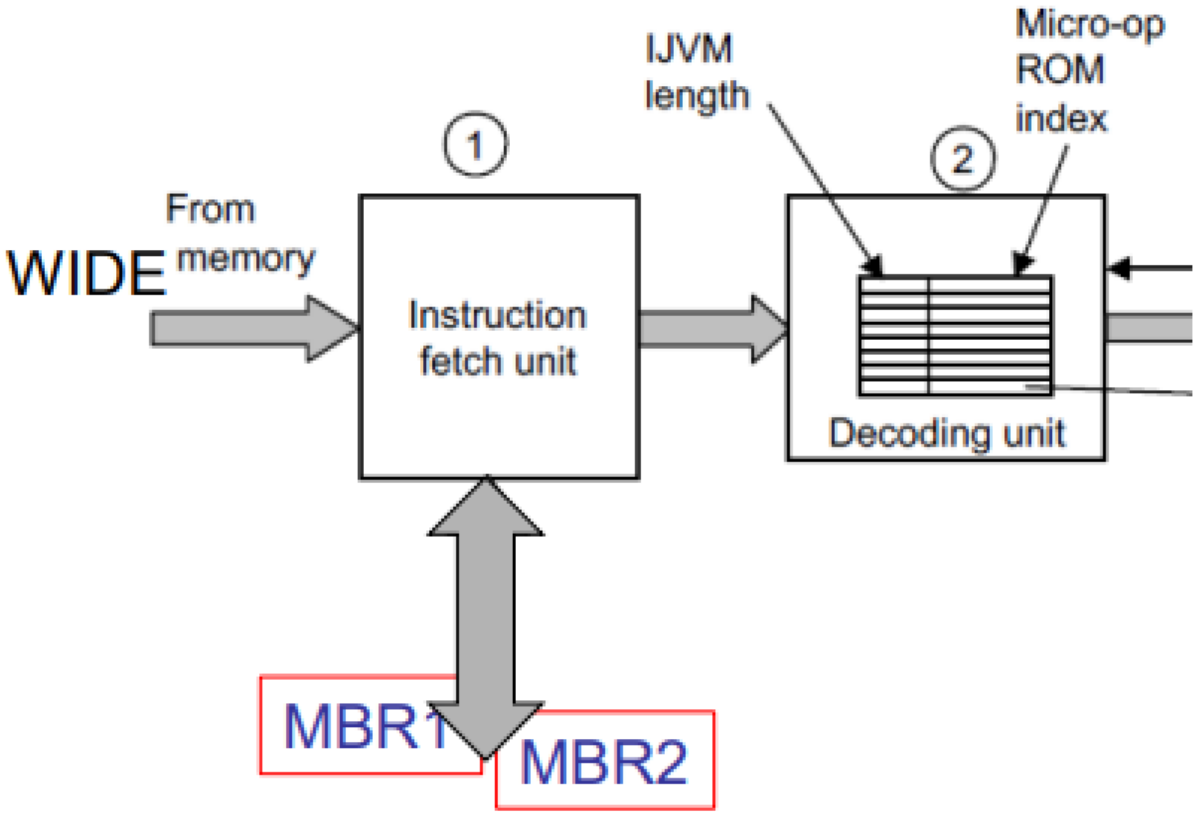
MIC-4

Il MIC-3 migliora notevolmente le prestazioni ma non si sofferma molto sull'istruzione successiva. Le istruzioni che hanno una diramazione (condizione) interrompono la pipeline.

Il compito del MIC-4 serve per mantenere la pipeline con le diramazioni.



IFU come abbiamo già detto gestisce il PC in modo separato, gestisce una ROM indicizzata con il codice delle istruzioni IJVM. Tramite la lunghezza identifica nel flussi di byte con il successivo codice operativo, interpreta il codice WIDE per trasformare il successivo o passa alla componente successiva l'indice relativo alla ROM.

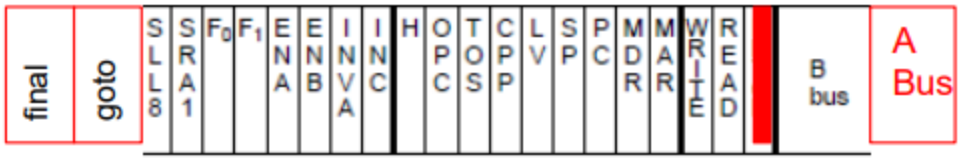


L'unità di accodamento

Gestisce due tabelle: una ROM e una RAM, la prima salva le micro-operazioni per il programma, la seconda è un queue di operazioni, nel primoprogramma le micro-operazioni sono inserite in sequenza: **non è più possibile la condivisione di sottosequenze.**

La micro-istruzioni

Rispetto alle micro-istruzioni del MIC-2, le micro-istruzioni del MIC-4 non hanno il campo di NEXT_ADDRESS e JAM, hanno due campi nuovi Final e Goto.



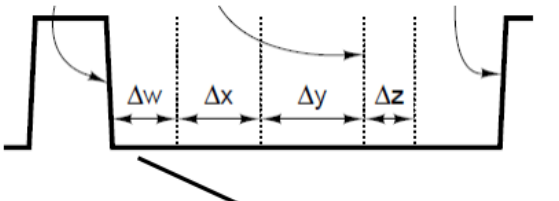
Final vale 1 quando finisce la micro-sitruzione IJVM, Goto vale 1 nelle micro-operazioni che effettuano diramazioni su Z e N.

I registri MIR

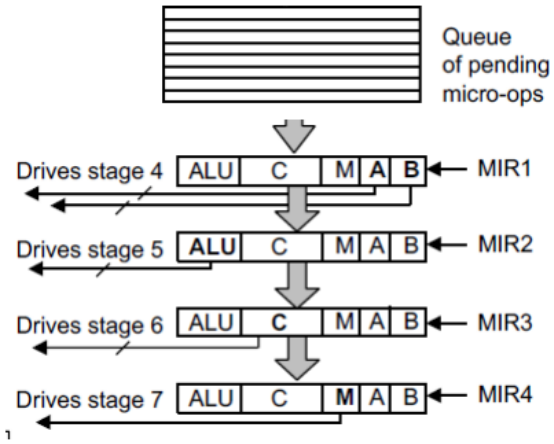
I campi della micro-operazione non sono attivi contemporaneamente, si usano 4 MIR indipendenti:

- MIR-1: controlla il primo ciclo.
- MIR-2: controlla il secondo ciclo.
- MIR-3: controlla il terzo ciclo.
- MIR-4: controlla il quarto ciclo.

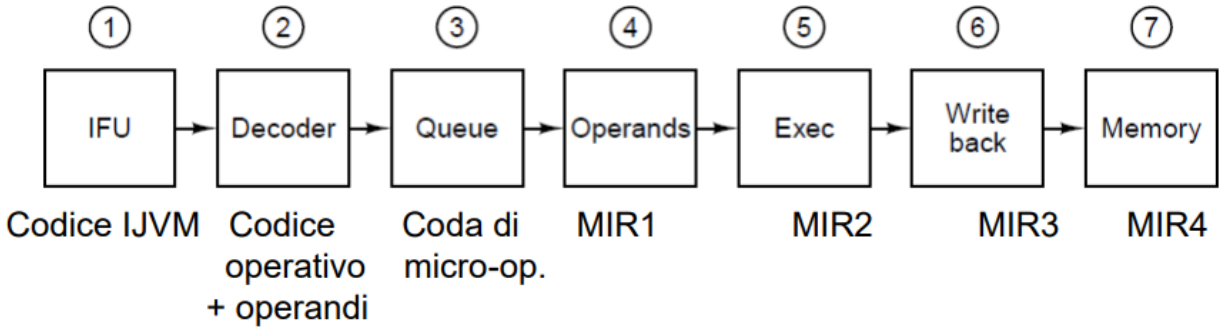
Ogni MIRx controlla una parte del percorso dati.



- All'inizio di ogni ciclo:
  - MIR3 → MIR4
  - MIR2 → MIR3
  - MIR1 → MIR2
  - pop(queue) → MIR1



Ogni passo ha una durata breve, quindi si può avere un'alta frequenza.



I primi tre passi non sono sempre necessari: un istruzione IJVM richiede più cicli di clock per essere eseguita. Le micro-diramazioni hanno un formato specifico:

- i bit a JAM.
- un indice alla ROM per micro-operazioni.
- possono eseguire solo la diramazione e non operazioni aggiuntive.

Quando l'unità di accodamento riceve un segnale di diramazione (Goto = 1) non si invia subito un segnale all'unità di decodifica: se è falsa si invia un segnale all'unità di decodifica di ACK altrimenti se vera si carica nelle coda di sequenze il next_address.

Se ci sono intoppi nella pipeline si creano delle bolle: **salti** (incodizionati o condizionati), **dipendenze** e **tempi di accesso** alla memoria (cache).

La pipeline funziona perfettamente su una sequenza di istruzioni, il problema avviene quando ci sono dei salti. L'unità di prelievo dell'istruzione IFU potrebbe continuare a leggere i byte a partire dall'indirizzo di destinazione del salto. Per farlo dovrebbe interpretare l'istruzione compito dell'unità di decodifica. Il salto incondizionato viene riconosciuto e viene memorizzata l'istruzione successiva.

Per rompere la pipeline si esegue il codice successivo, si crea così una **posizione di ritardo** la quale va riempita con istruzioni non pericolose con i NOP.

I salti condizionati come JAMZ o JAMN invece non hanno chiarezza sulla prossima microistruzione da eseguire per questo si esegue la microistruzione successiva con un ciclo di ritardo.

Per evitare questa perdita di uno o più cicli si può fare una scommessa prevedendo dove andrà il salto.

Le tecniche di branch prediction sono:

- **Tecniche statistiche** (compile time) → suggerimenti del compilatore basati sull'analisi del codice.
- **Tecniche dinamiche** (run time) → mantenere una tabella che tiene traccia degli avvenimenti del passato.

Predizione statistica

Se nei salti condizionali avviene una predizione corretta: esegue l'istruzione successiva senza problema, se la esegue in maniera errare allora si stoppa l'esecuzione successiva con uno **squashing**.

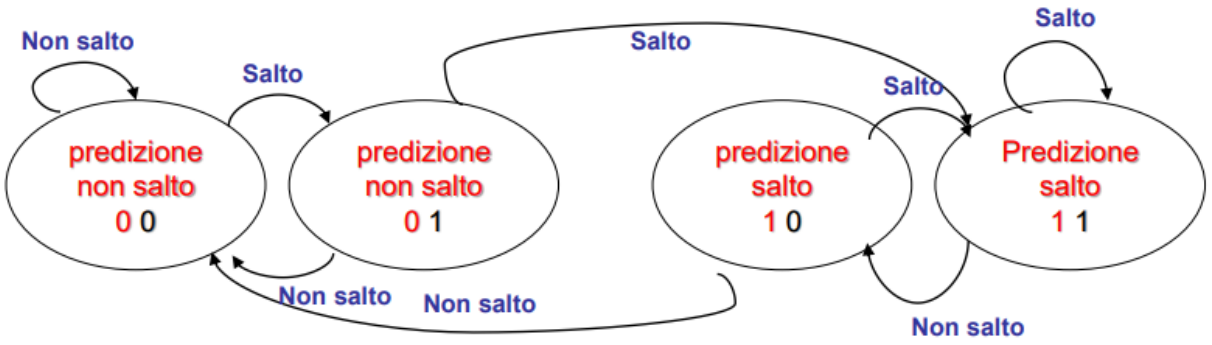
Predizione dinamica

Si utilizza una tabella che segna per ogni indirizzo di salto l'esito dell'ultima esecuzione: 0 se non effettuato, 1 se effettuato.

Se è la prima volta si considera salto avanti falso (0) salto indietro vero (1).

Predizione a due bit

La soluzione a 1 bit presenta molti inconvenienti come la impossibilità di controllare cicli annidati. Quindi si deve fare una predizione su 2 bit.



Per salvare i risultati si usano memorie associative ovvero simili a quelle usate per la cache.

Esecuzione in ordine

					R	R	R	R	R	R	R	R	W	W	W	W	W	W	W	W
Cy	N	Decoded	Iss	Ret	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
1	1 2	R0=R0*R1 R4=R0+R2	1 2		1 2	1	 1									1	 1			
2	3 4	R5=R0+R1 R6=R1+R4	3 -		3	2	1											1		
3					3	2	1									1	1	1		
4				1 2 3	2 1	1 1											1	1 1		
5	 5	 R7=R1*R2	4 5			1 2			1										1	 1
6	6	R1=R0-R2							1										1	1
7				4																1
8				5																
9	 7	 R3=R3*R1	6 -		1		1													
10					1		1													
11				6																
12			7			1		1								1				
13	8	R1=R4+R4	-			1		1								1				
14						1		1								1				
15				7																
16			8						2					1						
17									2					1						
18				8																

Possiamo anche eseguire un’esecuzione fuori ordine tramite **RAW** (Read After Write) ovvero lettura di un operando di una istruzione precedente ancora in corso.

Abbiamo anche quelle in scrittura: **WAR** (Write After Read) ovvero la scrittura di un risultato in un registro usato come operando dalle operazioni precedenti in corso, infinite **WAW** (Write After Write) scrittura di un risultato in un registro usato come destinazione di un risultato da un’operazione in corso precedente.