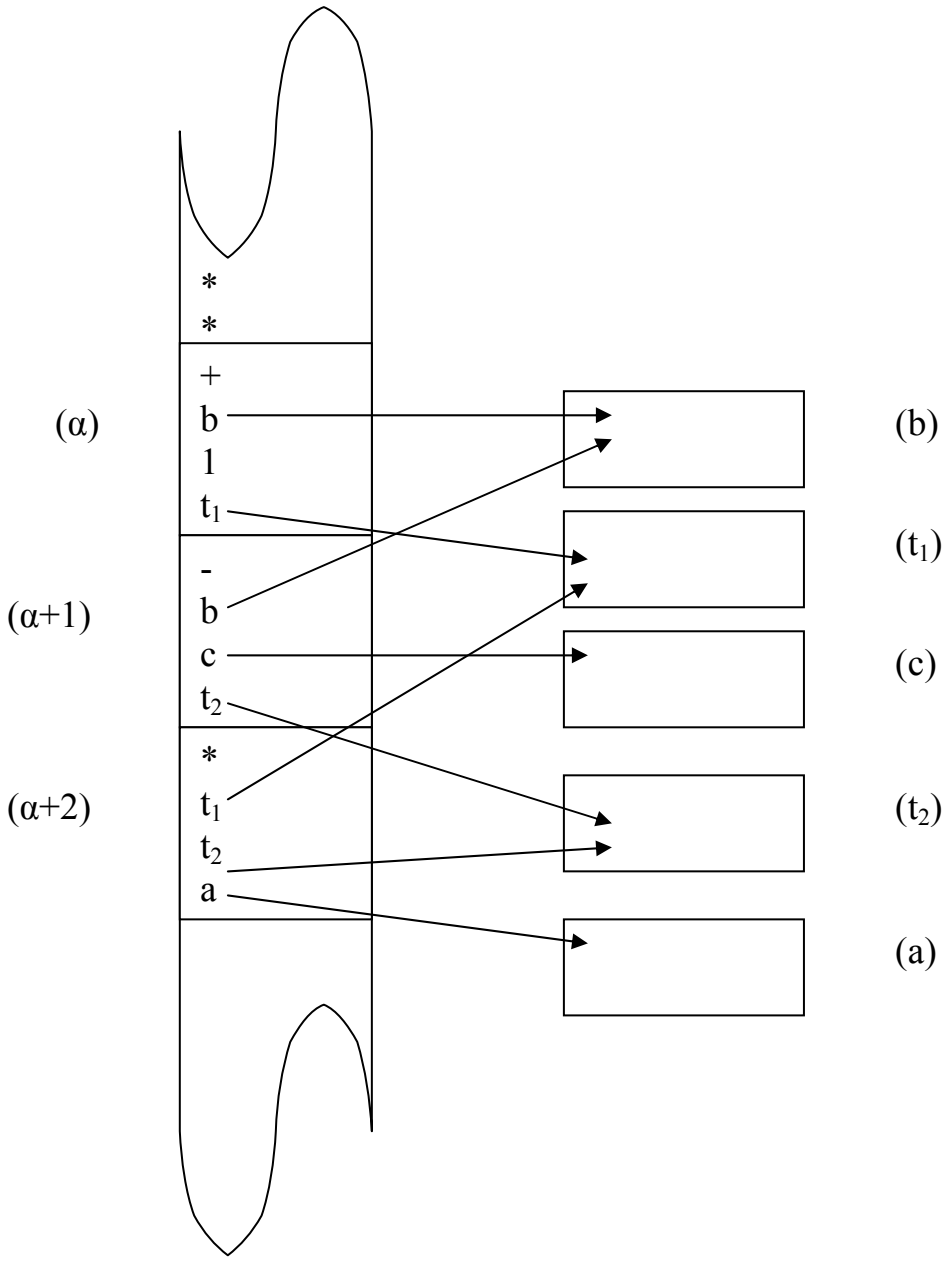
Architetture data-Flow

Le architetture che abbiamo visto finora sono dette architetture control flow. Ciò sta ad indicare che il flusso dell'elaborazione è dettato dall'ordine con cui le varie istruzioni sono registrate in un'area di memoria, ordine secondo il quale esse saranno prelevate ed eseguite grazie al meccanismo implementato dal Program Counter.

Se, ad esempio, vogliamo eseguire la seguente istruzione di alto livello

A=(b+1)*(b-c),

ciò che avviene realmente può essere rappresentato nella seguente figura

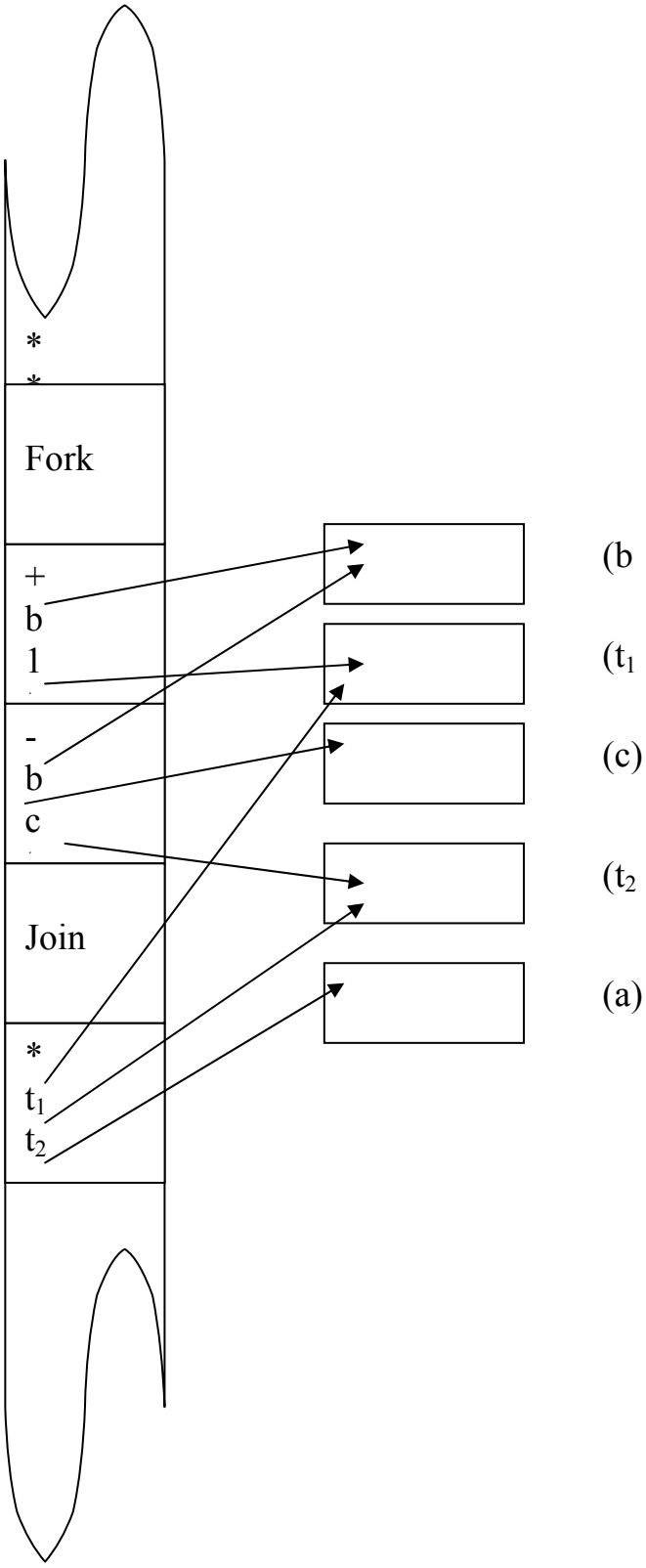


La sequenza di programma che effettua la computazione desiderata inizia con l'istruzione codificata nella cella di memoria di indirizzo (α). Per semplicità supponiamo che ogni istruzione sia codificata in una singola locazione. Tale istruzione farà riferimento alle variabili utilizzando il loro indirizzo di memoria. Successivamente verrà eseguita l'istruzione che si trova all'indirizzo successivo (α+1). Si noti come le due istruzioni hanno come operando la variabile b, il che significa che entrambe accedono alla stessa locazione di memoria per poter manipolare tale operando. Infine verrà eseguita l'istruzione contenuta all'indirizzo (α+2).

Possiamo riassumere dicendo che

- Il mezzo attraverso il quale i dati sono passati da un'istruzione all'altra è quello delle celle di memoria condivise. Il risultato della prima istruzione sarà passato alla terza istruzione perché venga moltiplicato con il risultato della seconda istruzione, perché la prima e la terza istruzione accederanno entrambe alla locazione di memoria che contiene la variabile t_1. Il riferimento agli operandi avviene attraverso i loro indirizzi di memoria
- In un computer tradizionale c'è un singolo flusso di controllo, cioè, ogni volta, l'elaborazione inizia dalla prima istruzione, poi viene prelevata la istruzione successiva e così via. Il processo di esecuzione della sequenza di istruzioni continua finché non si incontra un'istruzione di tipo branch. In questo caso il controllo è trasferito al nuovo indirizzo e l'elaborazione sequenziale continua

Qualcosa di analogo avviene nei processori paralleli


*
Fork
+
b
1
-
b
c
Join
*
t1
t2
(b
(t1
(c)
(t2
(a)

In modelli paralleli control flow, sono utilizzati speciali operatori come FORK e JOIN per specificare il parallelismo, come appare nell'esempio della figura precedente. L'operando di fork fa in modo che il processore principale (il processore che sta eseguendo questo segmento di programma) crei un processore figlio cioè spinga un secondo processore ad eseguire la prima operazione

$t_1=b+1$

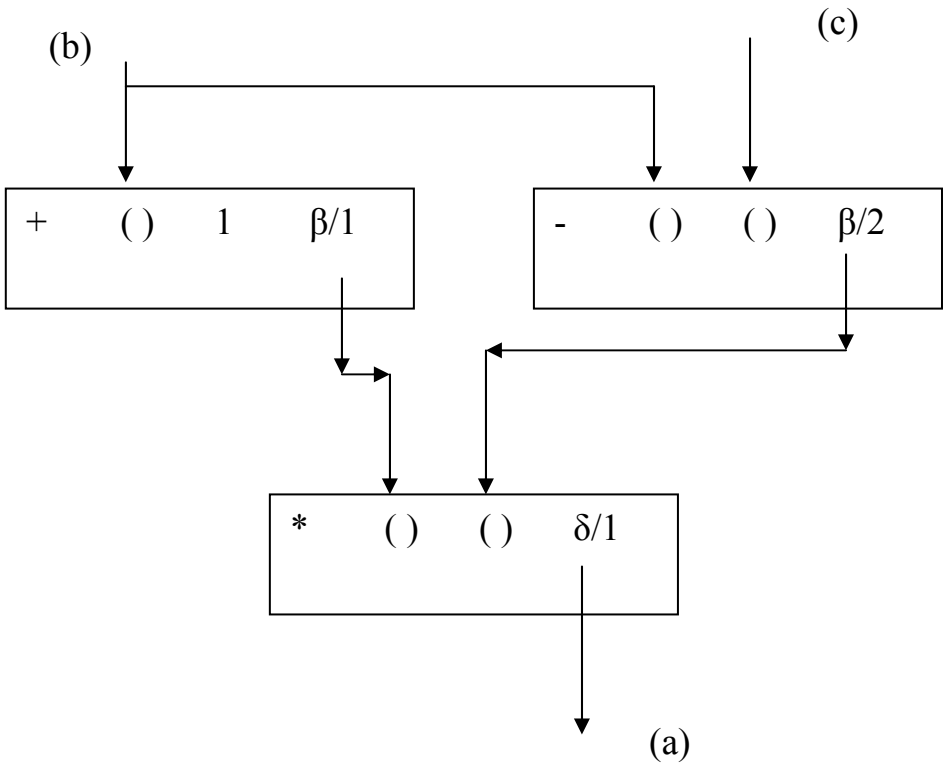
Senza attendere che il processore 1 finisca il suo compito, il processore principale inizia l'esecuzione della successiva istruzione.

$t_2=b-c$

A questo punto vi sono due processori attivi in parallelo, che eseguono simultaneamente le due istruzioni consecutive del codice. L'operando di Join costringe il processore principale ad attendere che il secondo processore abbia completato la sua istruzione. Ciò necessario poiché , nel passo successivo vanno moltiplicati i risultati di entrambi i processori. Anche in questo caso

- I dati sono passati da un'istruzione all'altra facendo riferimento a celle di memoria condivise
- Il flusso di controllo è implicitamente sequenziale, ma speciali operatori di controllo possono essere utilizzati per il parallelismo
- Dei program counter sono usati per sequenziare l'esecuzione delle istruzioni in un ambiente di controllo centralizzato.

In un sistema data flow, invece, le istruzioni non sono eseguite perché si trovano nell'ordine giusto di sequenza dettato da un program counter ma soltanto se sono disponibili i dati su cui operare. Premettiamo che in un sistema data flow ogni istruzione è costituita dall'operatore che indica l'operazione da eseguire, uno o due operandi, e una o più destinazioni (cioè altre istruzioni) verso le quali va inviato il risultato. Allora l'esempio di elaborazione precedente, nel caso di un'architettura data flow diventa il seguente



Le prime due istruzioni verranno eseguite non appena i valori contenuti nelle variabili b e c saranno resi disponibili. La variabile b viene usata contemporaneamente da entrambe le istruzioni perché una copia del suo valore (data token) viene inviata ad esse. Non appena entrambe le istruzioni avranno prodotto in uscita i token risultato, si attiverà l'ultima istruzione.

Allora le caratteristiche fondamentali che si possono elencare per un modello data flow sono

- I risultati intermedi o finali sono passati da un'istruzione all'altra come token o gettoni che attivano l'istruzione seguente
- Non esiste, in questo modello, il concetto di locazioni di memoria condivise e quindi la nozione tradizionale di variabile
- Il sequenziamento del programma è dettato soltanto dalla dipendenza dei dati fra le istruzioni. Nel nostro esempio, l'istruzione (b+1)*(b-c) va eseguita necessariamente dopo le altre due soltanto perché la sua esecuzione dipende dalla disponibilità dei risultati in uscita dalle altre due e non perché scritta in un certo ordine in memoria rispetto ad esse

I computer control flow hanno un'organizzazione guidata dal controllo, cioè il programma ha un completo controllo sul sequenziamento delle istruzioni. Le elaborazioni singole sono realizzate in un computer control flow usando un controllo centralizzato. I computer data flow sono computer guidati dai dati in cui uno stage di esame controlla le istruzioni per verificare se gli

operandi sono disponibili, in tal caso esse sono eseguite immediatamente se vi è un'unità di esecuzione disponibile. Tali architetture sono intrinsecamente asincrone e garantiscono implicitamente un elevato grado di parallelismo. Poiché non si fa uso di celle di memoria condivise, tali architetture sono automaticamente immuni da problemi come la mo0difica involontaria di una variabile da parte di una istruzione, o la modifica di un parametro da parte di un sottoprogramma, ecc.

Un data token è costituito dal valore risultato e dalla sua destinazione.

Per il tipo di filosofia che sottende ad un'architettura data flow i linguaggi imperativi non sono adatti per questi tipi di computer. I linguaggi di programmazione per computer data flow devono avere la caratteristica fondamentale che la sequenza con cui le istruzioni vanno eseguite deve derivare esclusivamente dalla dipendenza dai dati e non dalla sequenza con cui le istruzioni vengono scritte nel programma.

Se consideriamo , ad esempio, il seguente segmento di programma

1. P = X +Y
2. Q =P/Y
3. R = X*P
4. S = R – 0
5. T = R*P
6. U = S/T

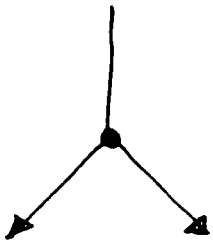
In un sistema control flow, le istruzioni potranno essere eseguite soltanto nell'ordine preimpostato 1, 2, 3, 4, 5 6. In un sistema data flow invece, le istruzioni potranno essere eseguite in un qualsiasi ordine e con qualsiasi parallelismo con le sole limitazioni che

- l'istruzione 2 va eseguita necessariamente dopo l'istruzione 1 perché dipende dal suo output
- La 3 deve essere eseguita necessariamente dopo la 1
- L'istruzione 2 e 3 possono essere eseguite nell'ordine che si vuole o in parallelo perché non c'è dipendenza dai dati per esse
- L'istruzione 4 deve essere eseguita necessariamente dopo la 3
- La 5 deve essere eseguita necessariamente dopo la 3
- La 4 e la 5 possono essere eseguite in qualsiasi ordine o in parallelo
- La 6 deve attendere necessariamente che sia completata l'esecuzione della 4 e della 5

Per tali motivi un algoritmo per macchine data flow viene rappresentato mediante diagrammi di flusso che mostrano le limitazioni nel sequenziamento delle istruzioni derivanti dalle dipendenze fra i dati.

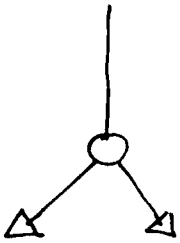
In tali diagrammi abbiamo i seguenti operatori

A) Data link



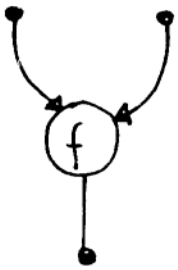
Questo oggetto serve ad indicare che uno stesso token numerico può essere smistato verso due oggetti diversi

B) Boolean link

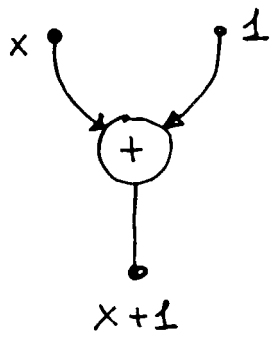


Questo operatore ha la stessa funzione del data link , ma smista dati booleani e non numerici

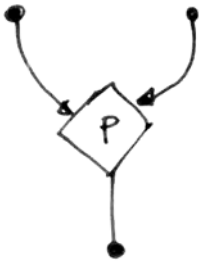
C) Operatore



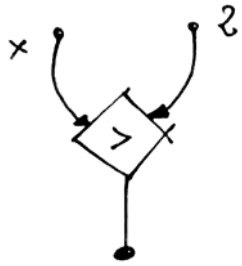
questo operatore da come token in uscita il risultato dell'operazione indicata all'interno dell'operatore e applicata ai token di ingresso. Ad esempio



D) operatore di decisione

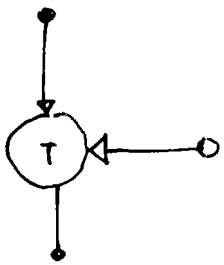


il token di uscita è un valore logico (true/false) risultato della condizione (indicata all'interno dell'operatore) applicata ai due token di ingresso. Ad esempio

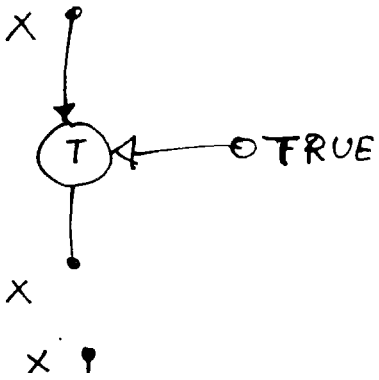


il token di uscita sarà true o false a seconda che x sia maggiore o minore o uguale di 2.

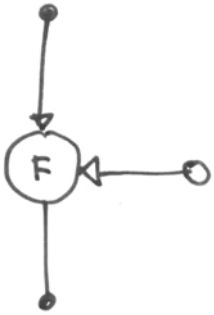
E) porta true



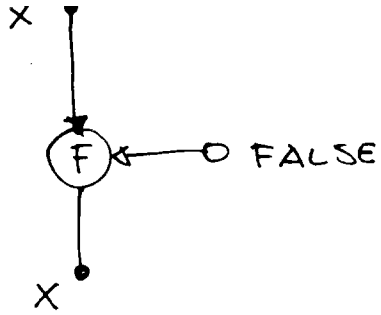
questo operatore consente al token in ingresso di passare in uscita se il token logico di controllo è true



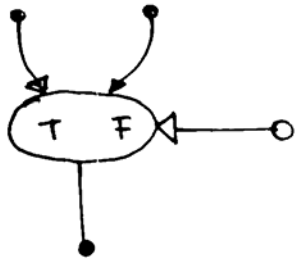
F) porta false



questo operatore consente al token di ingresso di passare in uscita se il token logico di controllo è false



G) operatore merge



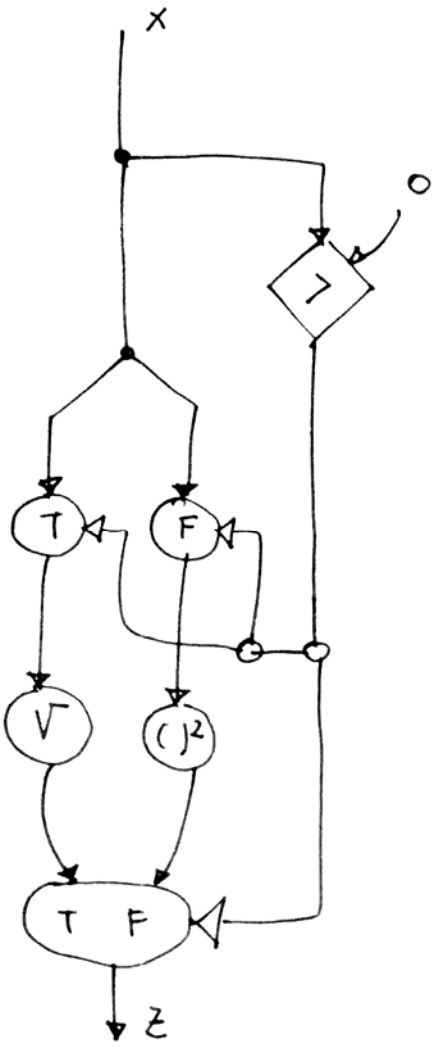
questo operatore smista uno dei due operandi in uscita a seconda del valore del token logico. Se il token logico è true passa in uscita il token d'ingresso a sinistra, altrimenti passa il token d'ingresso di destra.

Esempio 1

Vogliamo rappresentare con un diagramma di flusso il seguente algoritmo

```
If x > 0 then
                y = sqrt(x)
            else
                y = sqr(x)
```

Il corrispondente diagramma di flusso sarà



Esempio 2

Vediamo invece un ciclo while

```
While x> 0 do
       x=x-5
```

che corrisponde al seguente diagramma di flusso