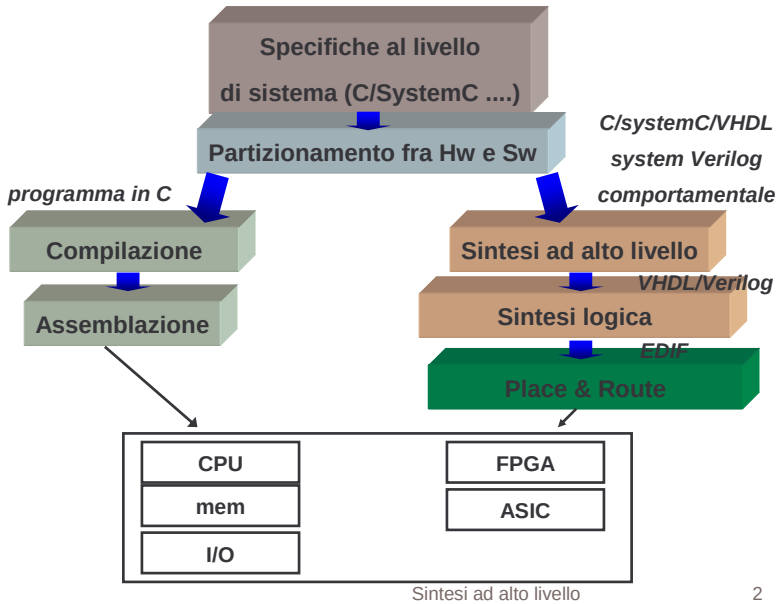
Sintesi ad alto livello di sistemi digitali

Michele Favalli
Universita' di Ferrara

Sintesi di un sistema digitale



Introduzione

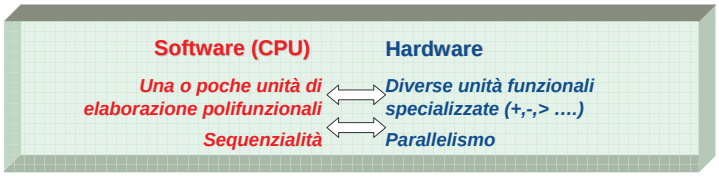
- **Realizzazione hardware di un algoritmo**
  - Motivazioni e applicazioni
  - Parallelo con il software
  - Obbiettivi
- **Ottimizzazione prima della sintesi**
- **Strutture dati per rappresentare un algoritmo**
  - La rappresentazione astratta è data da un Control-Data-Flow-Graph (CDFG)
- **Valutazione di un espressione aritmetica**
  - Data-Flow-Graph (DFG)
  - Costo - Risorse
  - Allocazione e scheduling
  - Prestazioni
  - Controllo
- **Istruzioni di selezione e di ciclo**
- **Flusso di progetto a basso livello**
- **Specializzazione vs. versatilità: il caso delle CPU**

Motivazioni

- **Algoritmo**
  - indipendente dalla realizzazione hardware
    - non specifica le risorse da utilizzare
    - specifica solo in parte l'ordine delle operazioni
- **CPU (architettura di Von Neumann)**
  - versatilità
  - "sequenzialità"
- **Applicazioni (elaborazione dei segnali)**
  - specializzazione
  - specifiche:
    - prestazioni elevate
    - costi ridotti
- **Obbiettivi non raggiungibili con una CPU che:**
  - "spreca" risorse per garantire la versatilità
    - presenta una (o più) unità multifunzionali (ALU)
    - complessa circuiteria di controllo

Motivazioni

- **L'architettura di Von Neumann è un caso particolare di sistema digitale**
- **Cambiamento di prospettiva nella realizzazione di un algoritmo**

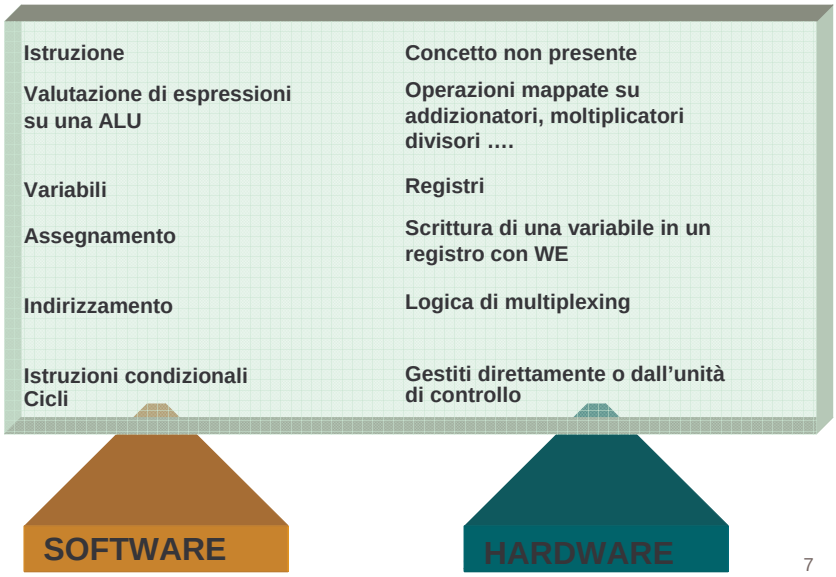


- **Realizzazioni hardware specializzate richiedono la disponibilità di:**
  - tecnologia
  - linguaggi di descrizione degli algoritmi (livello behavioral, HDL)
  - strumenti CAD

Strumenti per la realizzazione di algoritmi al livello hardware

- **Tecnologie microelettroniche**
  - ASICs (Application Specific Integrated Circuits)
    - Gate-Array
    - FPGA
    - Standard-Cells
- **Linguaggi di descrizione hardware consentono di descrivere sia:**
  - algoritmi (livello behavioral)
  - architetture (livello Register Transfer Level - RTL)
- **Strumenti CAD**
  - sintesi
  - analisi (verifica formale, simulazione)

Parallelismo con il software



7

Costi e prestazioni

- **L'ottimizzazione di costi e prestazioni è alla base della realizzazione hardware di algoritmi**
  - controllo (FSM)
  - data-path (registri, interconnessioni e unità funzionali)
- **Sintesi a livello architetturale**
- **Costi e prestazioni**
  - area su silicio (modello approssimato)

$$\sum_{\forall c} A(c)$$

  - prestazioni dinamiche (throughput, latenza o ritardo)
  - consumo di potenza, collaudabilità
  - tutte queste quantità possono essere valutate esatamente solo dopo la sintesi, prima si utilizzano modelli approssimati

Sintesi ad alto livello 8

Specifiche

- **Sono spesso dettate da applicazioni di tipo DSP o real-time e possono essere date sia dal throughput che dal ritardo (latenza)**
  - sistemi che operano su flussi di dati continui
  - sistemi reattivi
- **Oppure possono essere date da un area massima utilizzabile dettata da considerazioni di sistema**
- **Considerazioni di sistema dettano a volte la frequenza di clock**
  - ritardo e throughput possono essere dati in maniera relativa o assoluta
- **Esempi**
  - minimo ritardo per una data area
  - minima area per un certo ritardo

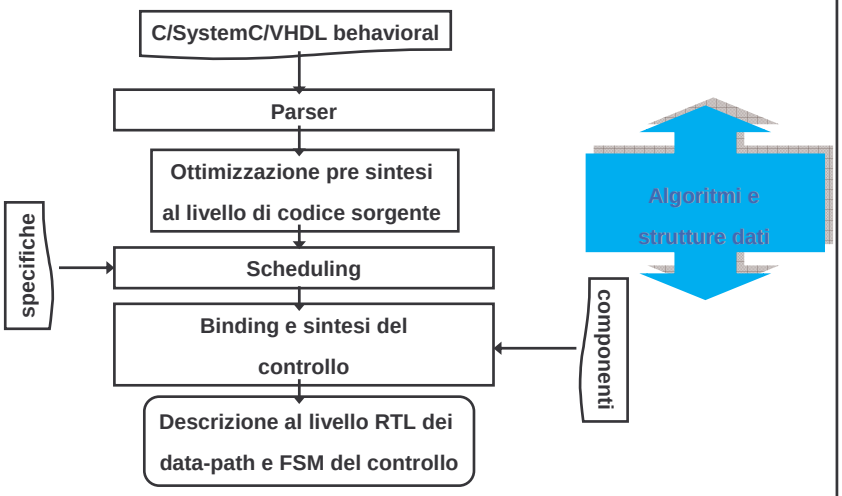
Risorse funzionali

- **Operandi: variabili (registri) e costanti intere (*n* bit)**
- **Operatori: +, -, *, /,>**
- **Blocchi funzionali**
  - Area occupata
  - Ritardo di esecuzione (critical path)
  - si tratta soltanto di approssimazioni in quanto non si è ancora realizzato il layout
- **Registri**
- **Logica di routing**
- **Alcuni operatori possono essere realizzati mediante la stessa risorsa**

Sintesi e ottimizzazione

- **Trovare una soluzione che soddisfi i vincoli di progetto**
- **Spazio di progetto**
- **Non esiste un ottimo assoluto (minima area, prestazioni massime)**
- **Punti di Pareto (soluzioni non dominate)**
- **Soluzioni di compromesso**
  - minima area per un dato ritardo
  - minimo ritardo per una data area
- **Tassonomia di riferimento**
  - rete sincrona
  - con o senza pipelining

Sintesi ad alto livello



Strutture dati per rappresentare un algoritmo

- Semplice algoritmo descritto mediante un comune linguaggio C/JAVA .... (non è l'unica alternativa)
- Interpretazione restrittiva di tipo imperativo e sequenziale riferita ad un architettura di tipo Von Neumann
- In realtà questo codice può avere un interpretazione molto più generale (parallelismo) che è sfruttata sia dalla sintesi dell'hardware che dai compilatori per le CPU attuali
- Sia i compilatori (software) che gli strumenti di sintesi (hardware) necessitano di una struttura dati astratta per rappresentare l'algoritmo

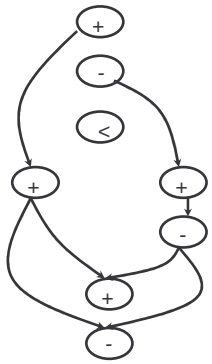
Esempio di algoritmo

```
t=a+b;
u=a-b;
if (a<b)
  v=t+c;
else
  {
    w=u+c;
    v=w-d;
  }
x=v+e;
y=v-e;
```

- Il significato di questo codice dal punto di vista di una CPU scalare è evidente
- In realtà l'algoritmo contiene delle informazioni più generali
- Per esplicitarle è necessario ricorrere a un modello più generale
- Si osserva che esistono istruzioni che si limitano ad alterare il flusso dei dati e altre che alterano il controllo cambiando l'ordine di esecuzione delle operazioni
  - Data-flow
  - Control-flow

Data-flow graph (DFG)

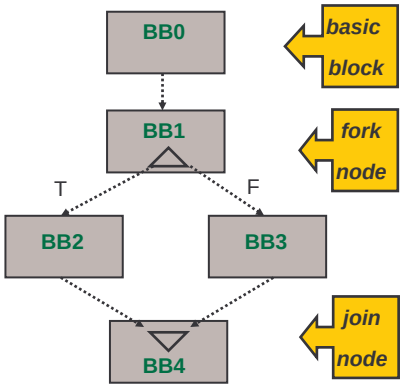
```
1.  t=a+b;
2.  u=a-b;
3.  if (a<b)
4.    v=t+c;
    else
      {
5.      w=u+c;
6.      v=w-d;
      }
7.  x=v+e;
8.  y=v-e;
```



Struttura dati astratta che rappresenta le dipendenze che esistono fra i dati elaborati dalle varie operazioni supponendo di disporre di operatori binari

Control-flow graph (CFG)

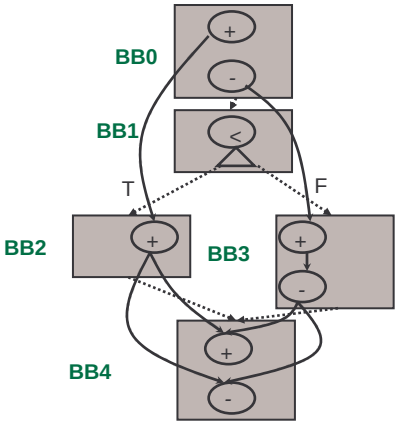
```
1.  t=a+b;          BB0
2.  u=a-b;
3.  if (a<b)        BB1
4.    v=t+c;        BB2
    else
      {
5.      w=u+c;      BB3
6.      v=w-d;
      }
7.  x=v+e;          BB4
8.  y=v-e;
```



Struttura dati astratta che rappresenta le possibile alternative nel flusso di esecuzione del programma

Control-Data-flow graph (CDFG)

```
1.  t=a+b;
2.  u=a-b;
3.  if (a<b)
4.    v=t+c;
else
  {
5.      w=u+c;
6.      v=w-d;
  }
7.  x=v+e;
8.  y=v-e;
```



Esempio: valutazione di un espressione

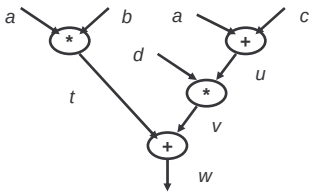
- **Si consideri il seguente frammento di algoritmo e l'espressione aritmetica corrispondente (definita su operandi a *n* bit):**

```
1.  t=a*b;
2.  u=a+c;
3.  v=d*u;
4.  w=t+v;
```

Behavioral

$z=a*b+d*(a+c)$

- **Il Data-Flow-Graph (DFG) associato e' il seguente**



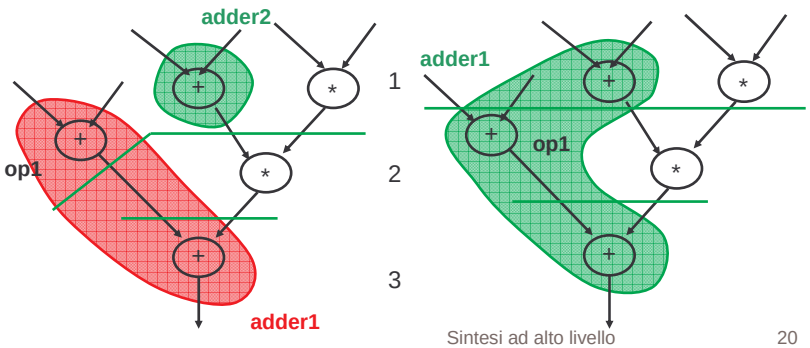
- **Rappresentazione astratta dell'espressione (ipotesi di operatori binari)**
- **E' importante notare che tutti gli ingressi si considerano disponibili contemporaneamente**

Data Flow Graph (I)

- **Il DFG ci dice:**
  - quali tipi di risorse sono necessari
  - l'ordine relativo di priorità fra le operazioni
- **Il DFG non dice:**
  - quali e quante risorse utilizzare (*allocazione*)
  - quale risorsa utilizzare per una data operazione (*binding)*
  - quando effettuare le operazioni (*scheduling*)
- **Allocazione e scheduling sono legate fra loro e determinano costi e prestazioni**
- **Ottimizzazione**

Data Flow Graph (II)

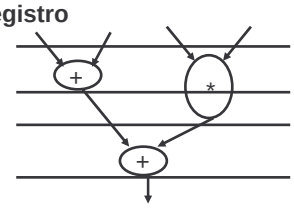
- **Esempio di correlazione fra scheduling e allocazione e binding**
- **L'operazione "op1" può essere eseguita sia nel primo che nel secondo periodo di clock (mobilità)**
- **Nel primo caso sono necessari due addizionatori**
- **Nel secondo caso è sufficiente un singolo addizionatore ⇨ riduzione di area a parità di prestazioni**



Data Flow Graph (III)

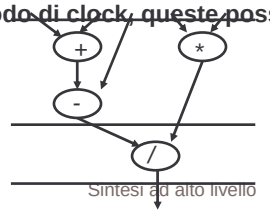
- **Se il ritardo di cammino critico di un componente è maggiore del periodo di clock si può fare un operazione multiciclo agendo sul comando di WE del registro**

multicycling



- **Se invece la somma dei ritardi di cammino critico di due operatori è minore del periodo di clock, queste possono essere concatenate**

chaining

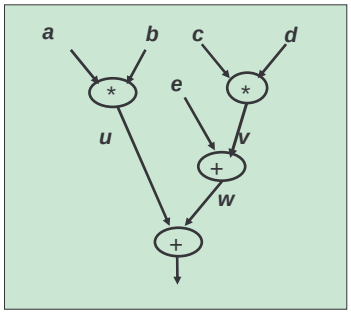


Esempi su chaining e multicycling

- **Algoritmo:**

```
u:=a*b; v:=c*d; w=e+v; z=w+u;
```

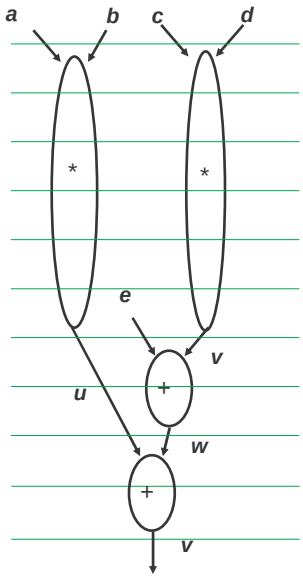
- **DFG**
- **Ritardi**
  - somma 3.8ns
  - moltiplicazione 11ns
  - latenza (combinatoria) 17.6ns
- **Specifiche: latenza minima**



T	latenza
2ns	20ns (10T)
3ns	21ns (7T)
4ns	20ns (5T)
5ns	25ns (5T)
6ns	24ns (4T)
....	
12ns	24ns (2T) (chaining)

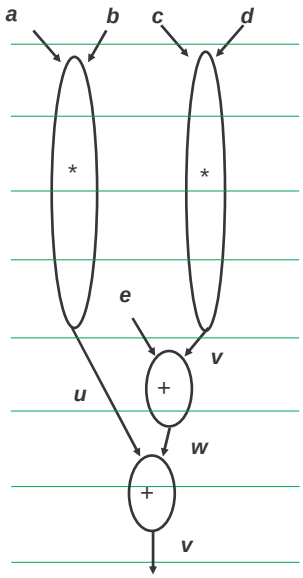
Consumo di potenza?

Multicylcing



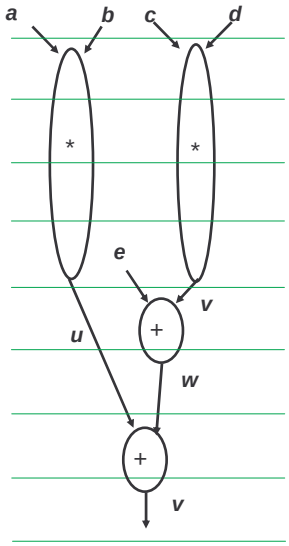
T=2ns latenza=10T
risorse = 2 mult. + 1 adder

Multicylcing



T=3ns latenza=7T
risorse = 2 mult. + 2 adder
nota: i due adder lavorano in modo chained (come un unico blocco combinatorio)

Multicylcing



T=3ns latenza=8T

risorse = 2 mult. + 1 adder

nota: in qusto caso è sufficiente un singolo adder

Esplorazione dello spazio delle possibili soluzioni – I

- **Vediamo ora come sia possibile utilizzare il modello di tipo DFG per descrivere le operazioni svolte dalla sintesi ad alto livello**
- **Focalizziamo l'attenzione su specifiche di costo e di latenza**
  - area minima per una data latenza
  - latenza minima per una data area
- **Per il momento non consideriamo specifiche sulla banda e supponiamo che ciascun operatore svolga le sue operazioni in un singolo ciclo di clock**
- **Il problema è quello di esplorare uno spazio di possibili soluzioni**
- **Questa esplorazione può essere fatta in maniera esatta o nel caso di DFG molto complessi in maniera euristica (il problema ha un costo computazionale che è esponenziale nella complessità del DFG)**

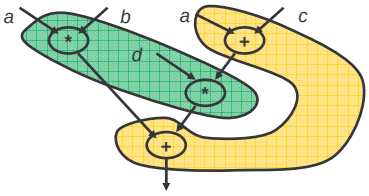
Esplorazione dello spazio delle possibili soluzioni – II

- **In particolare, esistono dei punti di questo spazio che sono particolarmente interessanti:**
  - la soluzione con il minimo assoluto di area (determinato dal minimo numero di operatori in grado di svolgere le operazioni indicate nel DFG)
  - la soluzione con la latenza minima (determinata dal cammino critico del DFG)
- **Prima di descrivere gli algoritmi che svolgono queste operazioni in maniera sistematica, conviene considerare un semplice esempio**

Soluzione a minima area

- **Allocazione: per calcolare l'algoritmo descritto dal DFG sono sufficienti un singolo addizionatore e un singolo moltiplicatore**

*u=a*b*
v=a+c
*w=d*v*
z=u+w

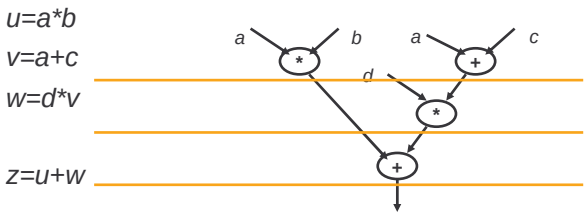


A questi componenti bisogna aggiungere:

- **Registri per memorizzare i risultati intermedi**
- **Logica di multiplexing per selezionare i diversi operandi degli operatori aritmetici**
- **FSM per generare i risultati di controllo di registri e multiplexer**

Scheduling

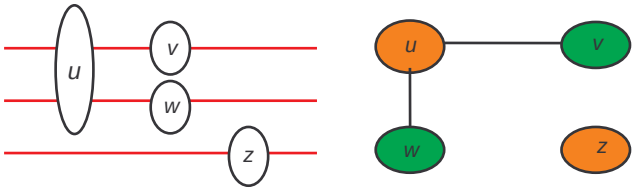
- Nell'esempio si nota che l'utilizzo del minimo numro di risorse impone una sequenza ben definita per le operazioni
- Ad esempio, le due moltiplicazioni non possono essere eseguite contemporaneamente
- Determina in quale periodo di clock si svolgono le operazioni



- Sono possibili scheduling diversi (sotto l'ipotesi di area minima)?
- Si noti che in questo caso la soluzione ad area minima ha anche la latenza minima

Ottimizzazione dei registri

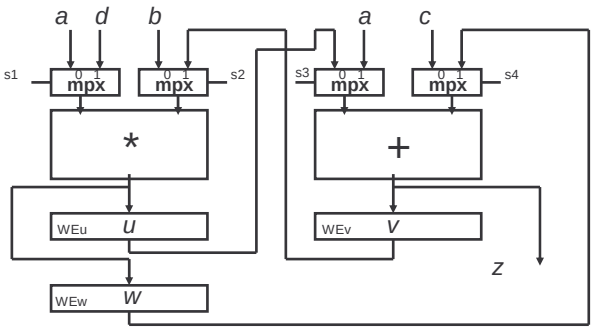
- Dalla descrizione dell'algoritmo sembra che siano necessari 4 registri (uno per ciascuna variabile)
- Sono strettamente necessari?
- Tempo di vita delle variabili ⇨ grafo di incompatibilità



- Il numero di registri necessario si ottiene colorando il grafo di incompatibilità
- La condivisione di registri implica spesso l'utilizzo di multiplexer
- Si può pensare a soluzioni con il minimo numero di multiplexer

Soluzione a minima area

- **Con il minimo numero di multiplexer**
- **Per realizzare l'espressione sono sufficienti un singolo addizionatore e un singolo moltiplicatore**

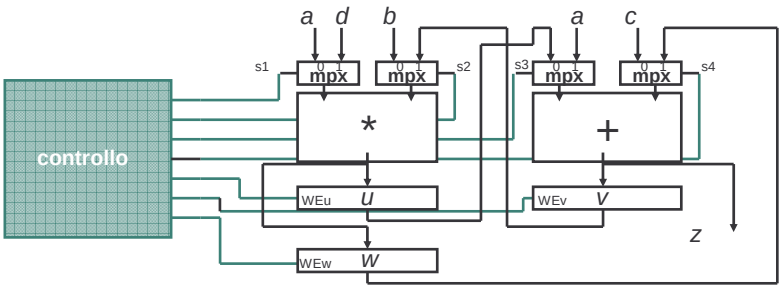


Prestazioni

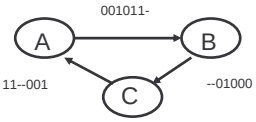
- **La frequenza di clock minima è determinata da t_{MULT} (se non si utilizza multicycle)**
- **Il ritardo (latenza) dopo il quale è disponibile il risultato è pari a 3* t_{MULT}**
- **Queste sono valutazioni approssimate: bisogna considerare il ritardo legato alla logica di multiplexing e quello richiesto dai flip-flop nei registri:**
  - $t=t_{MULT}+t_{MUX}+t_R+t_{SU}$

Controllo

- I segnali di RE e di selezione dei MUX devono essere generati da una FSM

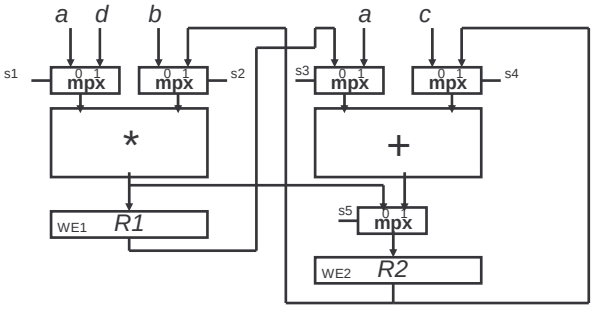


		s1	s2	s3	s4	WEu	WEv	WEw
A	B	0	0	1	0	1	1	-
B	C	1	1	-	-	0	0	1
C	A	-	-	0	1	0	0	0



Soluzione a minima area con registri ottimizzati

- Condivisione di risorse mediante graph coloring



- Descrizione a livello RTL

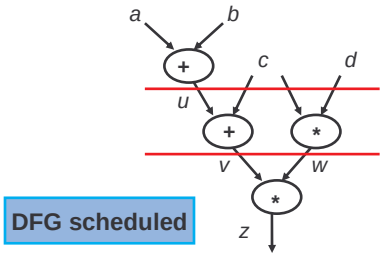
```
R1:=a*b; R2:=a+c;
R2:=R2*d;
R2:=R1+R2;
```

Esercizio

- Si supponga che $t_{MULT}=4*t_{ADD}=8ns$
- Si determinino T, e latenza nel caso precedente
- Si tracci il DFG utilizzando il multicycling supponendo di avere T=2ns
- Si determini un allocazione e uno scheduling ad area minima
- Si calcolino i nuovi valori di throughput e di latenza
- Ccosa cambia nella rete di controllo?

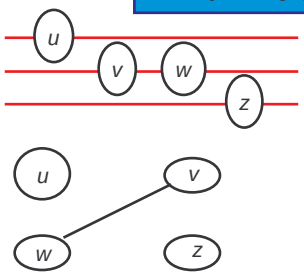
Binding (I)

- Per binding si indica l'assegnamento di un operazione a una particolare risorsa
- Anche questo passo consente di fare ulteriori ottimizzazioni
- Che consentono una volta allocato un certo numero di risorse funzionali e di registri di risparmaire sulla logica steering (multiplexer)
- Si consideri il seguente esempio



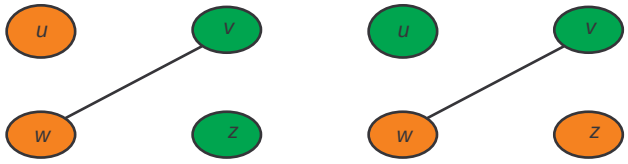
DFG scheduled

incompatibily graph

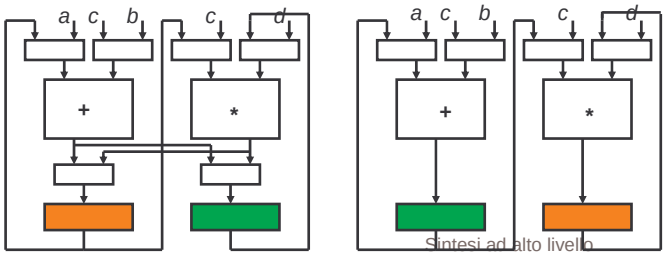


Binding (II)

- Il grafo di incompatibilità può essere colorato con due colori e quindi si possono usare due registri
- Ci sono però due differenti alternative

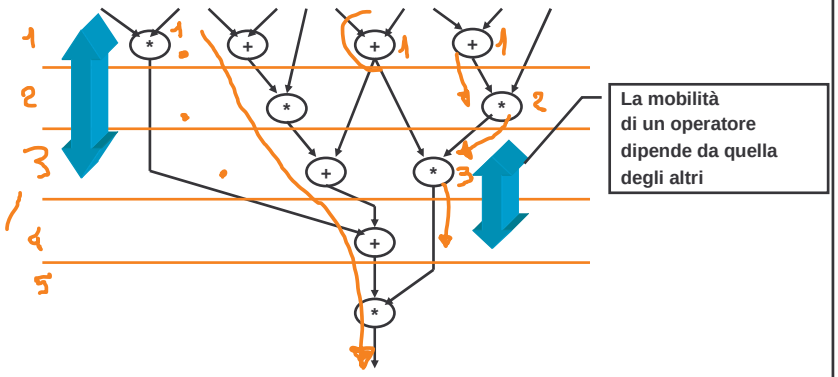


- Queste danno luogo a due diverse realizzazioni



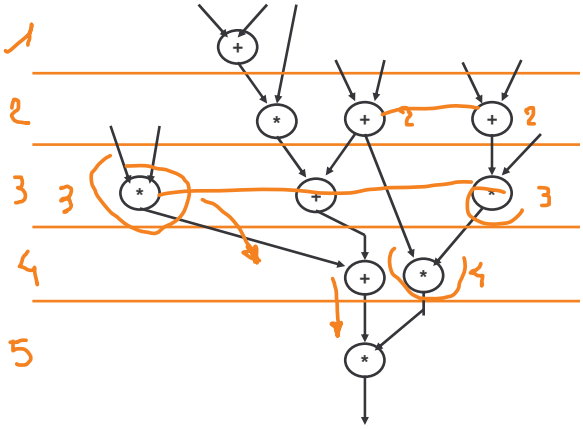
Scheduling (I)

- Mobilità degli operatori (scheduling ASAP)



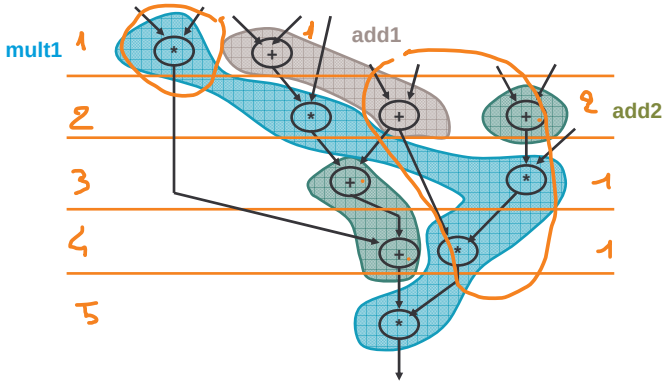
Scheduling (II)

- Scheduling ALAP (As Late As Possible)



Scheduling (III)

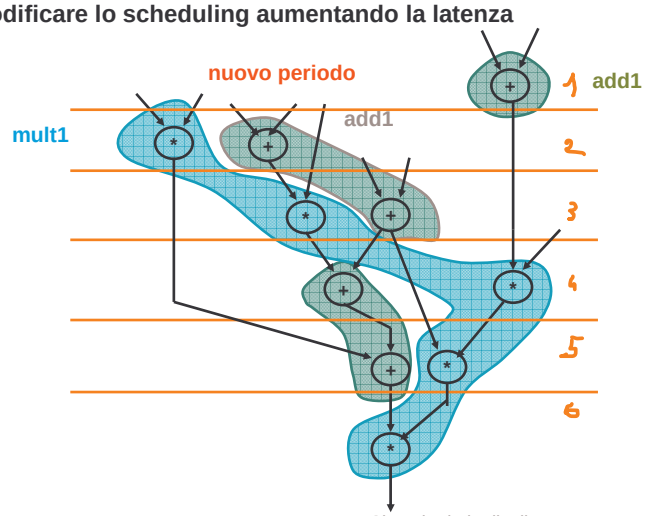
- Sia lo scheduling ALAP che quello ASAP di solito non consentono di minimizzare il numero di risorse utilizzate



- Come si vede, questo scheduling è l'unico che consente di realizzare la rete con un singolo moltiplicatore e con due addizionatori

Scheduling (IV)

- **Se si volesse utilizzare un singolo addizionatore, bisognerebbe modificare lo scheduling aumentando la latenza**

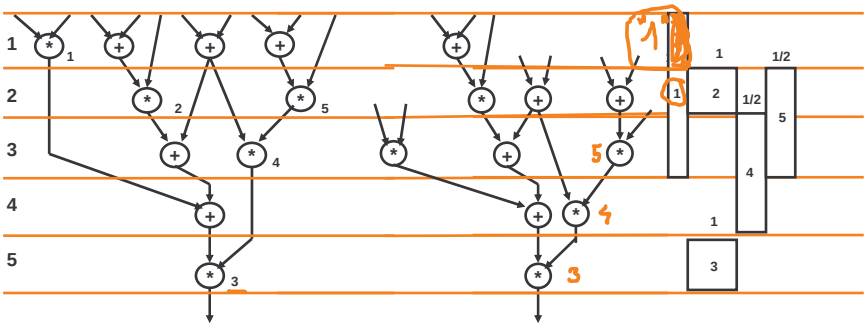


Algoritmi di scheduling

- **Esistono diversi algoritmi di scheduling**
- **Force Directed Scheduling**
  - Algoritmo euristico per minimizzare l'area in presenza di vincoli sulla latenza (tipicamente dati dal cammino critico)
  - In pratica, la densità di operazioni dello stesso tipo in un certo ciclo di controllo viene vista come una forza elastica in grado di attrarre/respingere un operazione
  - Tale forza viene minimizzata per ottenere una distribuzione bilanciata delle risorse
  - L'algoritmo serve sia per unità funzionali, registri e logica di routing

Force directed scheduling (I)

- **Vengono fatti prima uno scheduling ASAP e ALAP**
- **Time frame di un operazione:**
  - intervallo fra il ciclo di clock ASAP e quello ALAP
  - si suppone che la probabilità di programmare l'operazione sia distribuita uniformemente in tale intervallo

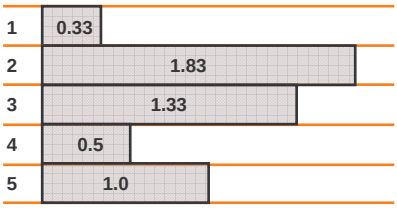


  - Si considera solo il caso delle moltiplicazioni, l'area di ciascun rettangolo è unitaria

Force directed scheduling (II)

- **Grafo di distribuzione delle operazioni**
- **Concorrenza per ogni ciclo di controllo (i) delle operazioni dello stesso tipo**

$$DG(i)=\sum_{opn\ type} Prob(opn,i)$$



Force directed scheduling (III)

- **Per ogni operazione (*op*) e per ogni passo di controllo (*j*) corrispondente a uno scheduling fattibile per tale operazione, si calcola la forza**

$$Force(j)=DG(j) - \sum_{t}^{b} \frac{DG(i)}{b-t+1}$$

- **dove b e t rappresentano lo scheduling dell'operazione nei casi ALAP e ASAP**
- **vediamo il caso della moltiplicazione 1 del DFG**

  Force(1)=0.33-1.16= -0.83

  Force(2)=1.83-1.16= 0.33

  Force(3)=1.33-1.16= 0.17
- **Una forza negativa indica che l'operazione è "attratta", una "positiva" indica che è respinta**
- **Compito dell'algoritmo è scegliere l'operazione (di un certo tipo) caratterizzata dalla minore forza**

Force directed scheduling (IV)

- **Per le operazioni di prodotto 4 e 5 si hanno questi risultati**

  op. 4

  Force(3)=1.83-0.91= 0.92

  Force(4)=0.5-0.91= -0.41

  op. 5

  Force(2)=1.83-1.58= 0.25

  Force(3)=1.33-1.58= -0.25

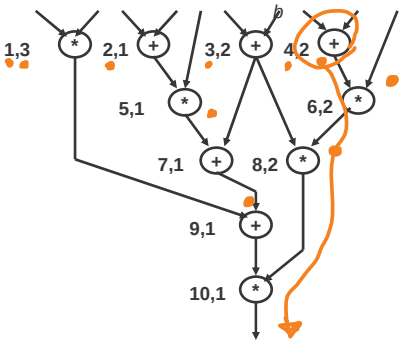
  b
- **L'operazione da selezionare è il prodotto 1 da programmare nel ciclo di controllo 1**
- **In un secondo ciclo, si aggiornano le densità delle operazioni e quindi i valori di DG e si seleziona l'operazione con la minima forza**
- **In realtà, le cose sono più complesse di quanto descritto fino ad ora. Se si programmasee op. 5 al ciclo 4, op. 4 non avrebbe più la stessa mobilità**
- **Alla "*self-force*" si devono aggiungere *predecessor-successor forces***

List based scheduling (I)

- Diversamente dal force based scheduling, il list based scheduling cerca di minimizzare la latenza per un ben definito ammontare delle risorse
- In pratica, è una generalizzazione del metodo ASAP
- Si mantiene una lista di operazioni pronte (i cui predecessori sono già stati programmati)
- A ciascuna operazione pronta si associa una funzione di priorità
  - un esempio di funzione di priorità è l'inverso della mobilità
- A ciascuna iterazione, per un dato tipo di operazione, vengono programmati i nodi con la priorità più alta dilazionando gli altri a ulteriori iterazioni
- Gli operatori a più alta mobilità sono quelli programmati dopo (greedy)
- Si sono usate diverse funzioni di costo

List based scheduling (II)

- Diversamente dal force based scheduling, il list based scheduling cerca di minimizzare la latenza per un ben definito ammontare delle risorse
- Ciascun nodo è annotato dal suo indice e dalla sua mobilità (inverso della priorità)

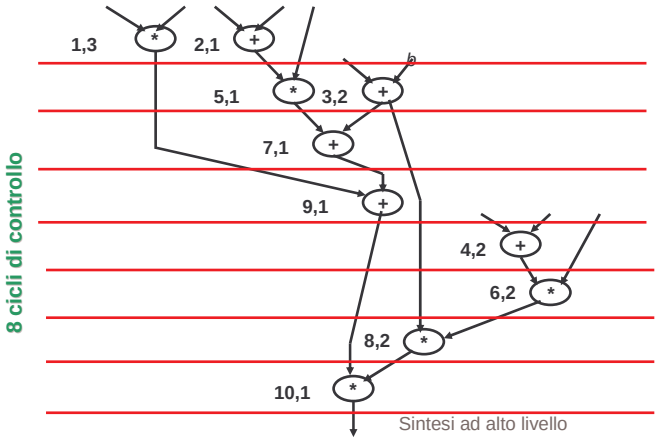


lista di nodi pronti

passo 1: 1,2,3,4
scheduling 1(*),2(+)
passo 2: 3,4,5
scheduling 5(*),3(+)
passo 3: 7,4
scheduling 7(+)
passo 4: 9,4
scheduling 9(+)
passo 5: 4
scheduling 4(+)
passo 6: 6
scheduling 6(*)
passo 7: 8
scheduling 8(*)
passo 8: 10
scheduling 10(*)

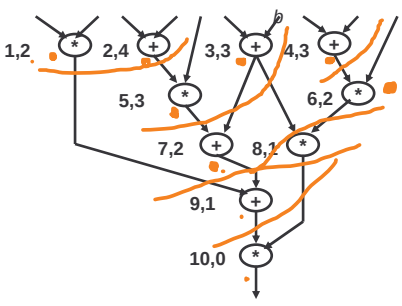
List based scheduling (III)

La scelta di programmare 3 invece di 4 al passo 2 è arbitraria in quanto entrambe le operazioni hanno la stessa priorità. La scelta diversa avrebbe dato luogo a una minore latenza. Si provi a pensare a una versione modificata dell'algoritmo.



List based scheduling critical path based

- **Il critical path di un nodo è il numero di nodi (massimo) da tale nodo all'uscita**
- **La funzione di priorità è ora la lunghezza del cammino critico**
- **Ciascun nodo è annotato dal suo indice e dal suo cammino critico (priorità)**

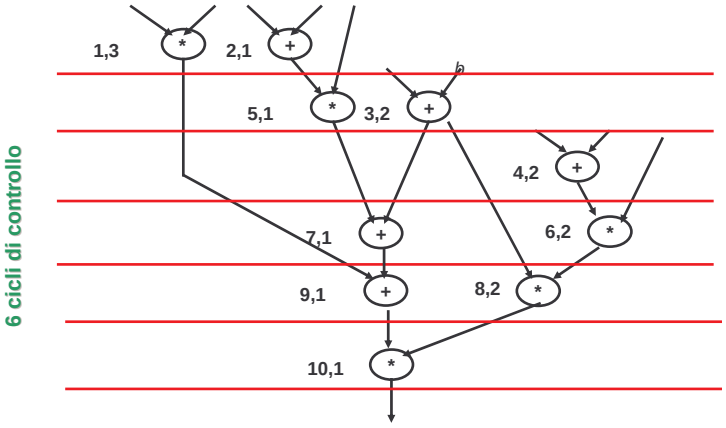


questa volta il nodo 4 è più critico di 7

lista di nodi pronti

passo 1: 1,2,3,4
scheduling 1(*),2(+)
passo 2: 3,4,5
scheduling 5(*),3(+)
passo 3: 7,4
***scheduling* 4(+)**
passo 4: 7,6
***scheduling* 6(*),7(+)**
passo 5: 8,9
***scheduling* 8(*),9(+)**
passo 6: 10
***scheduling* 10(*)**

List based scheduling (critical path)

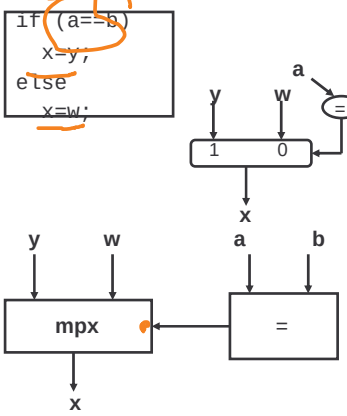


Istruzioni di selezione

- **Si possono inserire sia nel controllo sia, in casi più semplici, nel data-path**

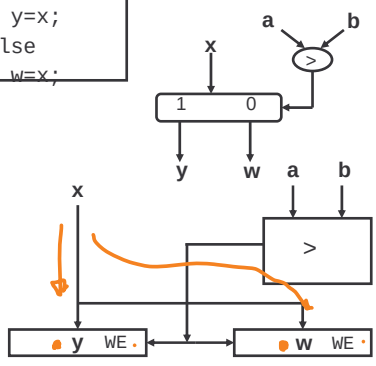
Selezione

```
if (a==b)
  x=y;
else
  x=w;
```



Distribuzione

```
if (a>b)
  y=x;
else
  w=x;
```



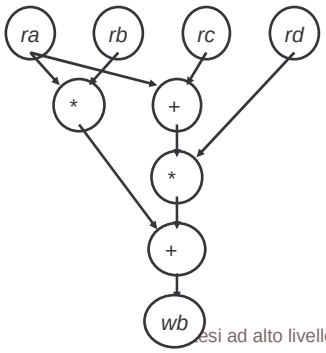
Problemi aperti

- **Per ora si sono visti algoritmi dominati dall'elaborazione dei dati**
- **Esistono algoritmi dominati dal controllo (cicli, diramazioni complesse)**
  - Control Data Flow Graph
  - Algorithmic State Machines (da cui poi estrarre il CDFG)
- **Pipelinining**
- **Elaborazione presintesi**
- **Altri vincoli di progetto: consumo di potenza e collaudabilità**

Bus

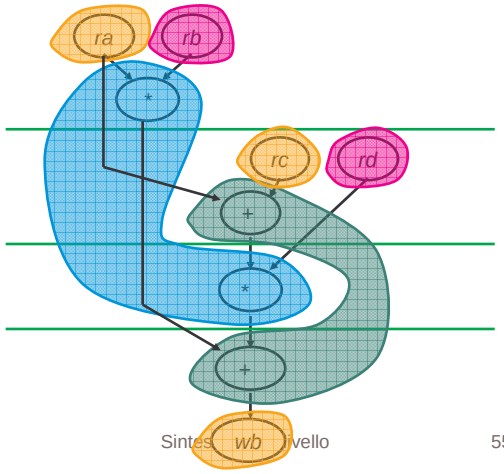
- **I bus possono essere gestiti in maniera del tutto simile ad altre risorse**
- **Si possono inserire nel DFG operazioni di *read* e *write***
- **Tali operazioni possono venire concatenate o meno alle operazioni aritmetiche dipendentemente dalle temporizzazioni dei circuiti**
- **L'algoritmo visto in precedenza diventa**

read a
read b
read c
read d
*u=a*b*
v=a+c
*w=d*v*
z=u+w
write w



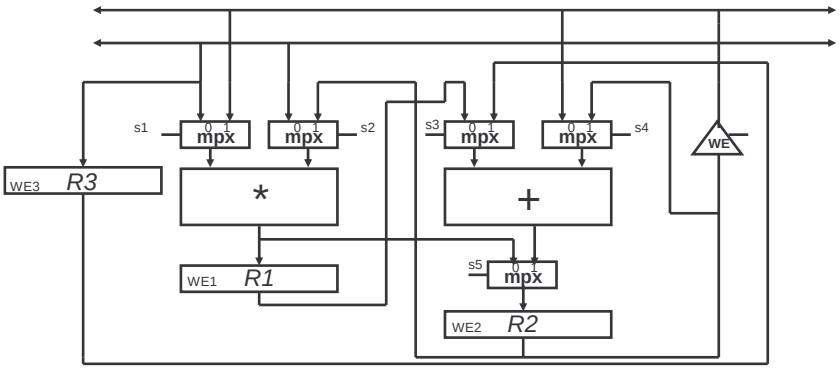
Bus

- Si supponga di avere 2 bus per la lettura e la scrittura
- Con concatenamento (non si ha bisogno di registri a parte quello che contiene a)



bus1
bus2
multiplier
add

Soluzione a minima area con registri ottimizzati



- Descrizione a livello RTL

*R1:=a*b; R3:=a;*
R2:=a+c;
*R2:=R2*d;*
R2:=R1+R2;

Conclusioni

- **Abbiamo visto come si possa sintetizzare un semplice algoritmo ottenendo una realizzazione al livello RTL che soddisfa determinate specifiche su latenza e costo**
- **Rimane da vedere cosa si può fare nel caso in cui le specifiche sono date sul throughput anzichè sulla latenza**
- **In questo caso si devono tipicamente utilizzare sistemi di tipo pipelined**
- **Dopo di questo vedremo alcuni aspetti delle trasformazioni di presintesi che sono particolarmente interessanti in quanto riguardano anche le tecniche di compilazioni per CPU di tipo superscalare e pipelined**