Sintesi ad alto livello

da scheduling e allocazione al livello RTL

M. Favalli

DE: Department of Engineering Ferrara

Sommario

- Binding
- Ottimizzazione dei registri
- Ottimizzazione dell'I/O

Binding - I

- Il binding consiste nell'associare
  - ogni operazione del DFG a una specifica risorsa
  - ogni variabile dell'algoritmo a uno specifico registro
- Questa parte del processo di sintesi non influisce sul costo delle risorse funzionali, ma sui multiplexer utilizzati per condividere tali risorse e sul numero di registri utilizzati

Binding - II

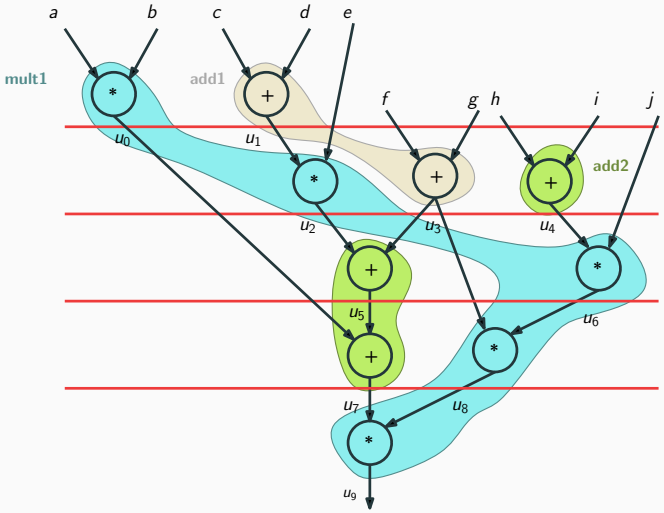
- Anche questi mpx e registri possono essere ottimizzati
- Questo puó essere fatto insieme ad allocazione e scheduling
  - ad esempio il force-directed scheduling puó essere esteso a questo caso
- Oppure dopo tali operazioni
  - qui si vedranno solo alcuni aspetti rilevanti che danno l'idea del tipo di problema

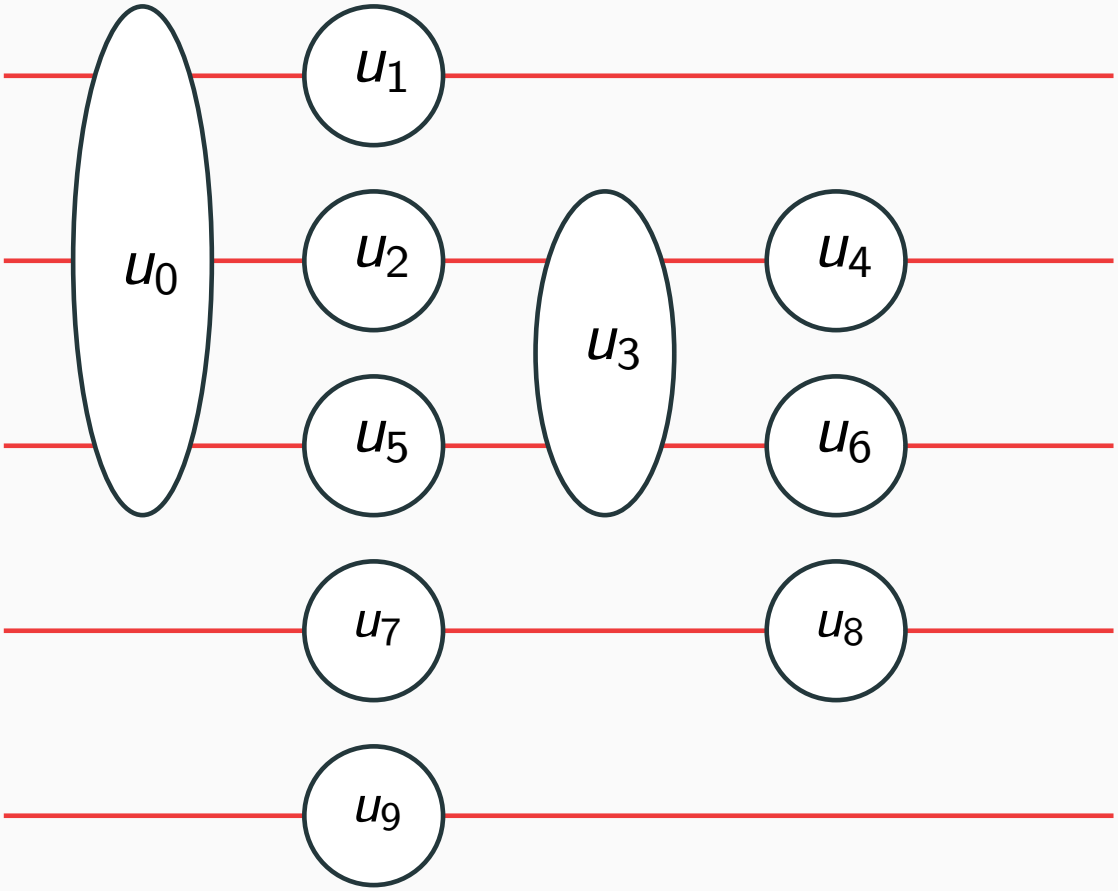
Ottimizzazione dei registri

- Non é necessario utilizzare un registro per ogni variabile dell'algoritmo
- Il numero minimo di registri da utilizzare puó essere determinato contando il numero di archi tagliato da ciascun segmento orizzontale del DFG con scheduling
- Un possibile modo per ottenere il binding delle variabili sui registri richiede di
  1. tracciare il grafo con il tempo di vita delle variabili
  2. tracciare il grafo di incompatibilitá
- Si noti che questa tecnica puó essere utilizzata anche nel software per ottimizzare l'utilizzo dei registri riducendo il numero di accessi alla cache

Tempo di vita delle variabili

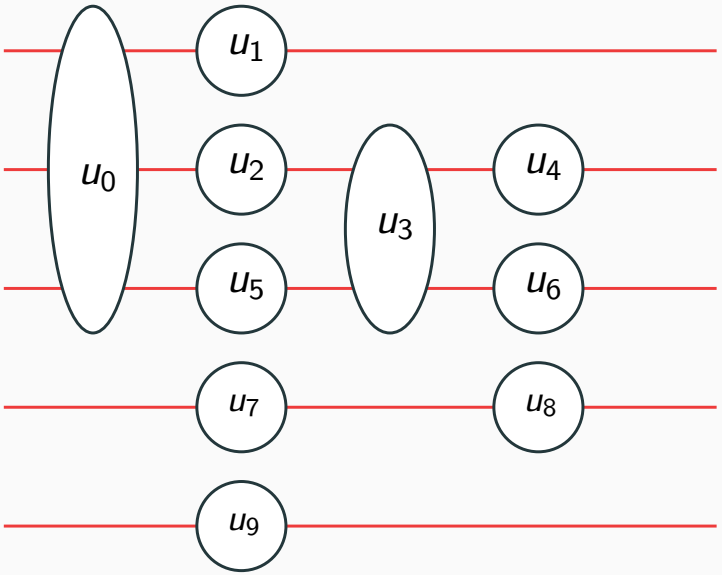
Calcolabile dal DFG con scheduling

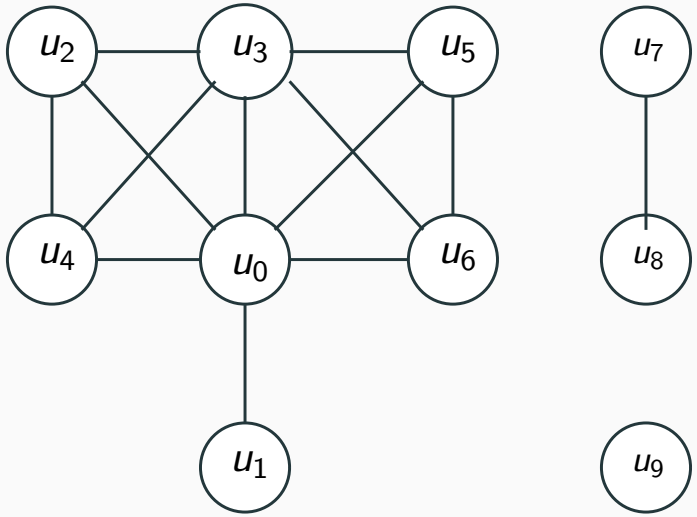




Grafo di incompatibilitá e binding dei registri

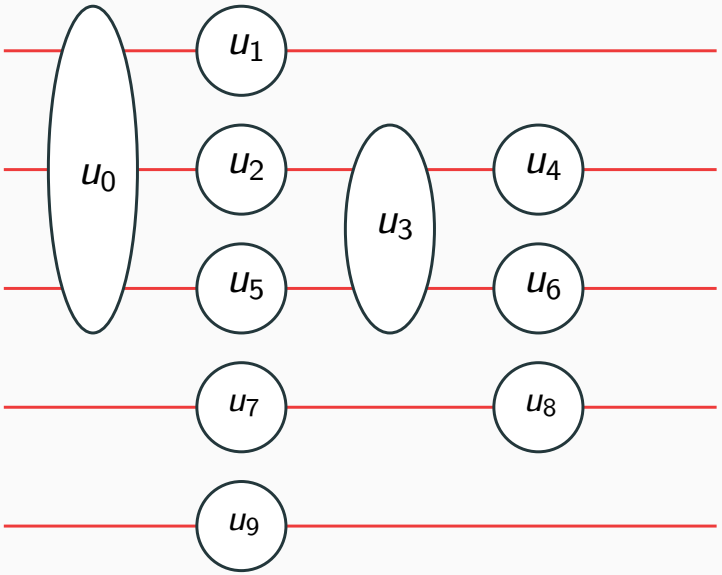
- Grafo con un vertice per ogni variabile e un arco fra due variabili se queste sono attive nello stesso ciclo di clock
- Il binding puó essere fatto utilizzando un algoritmo di graph coloring

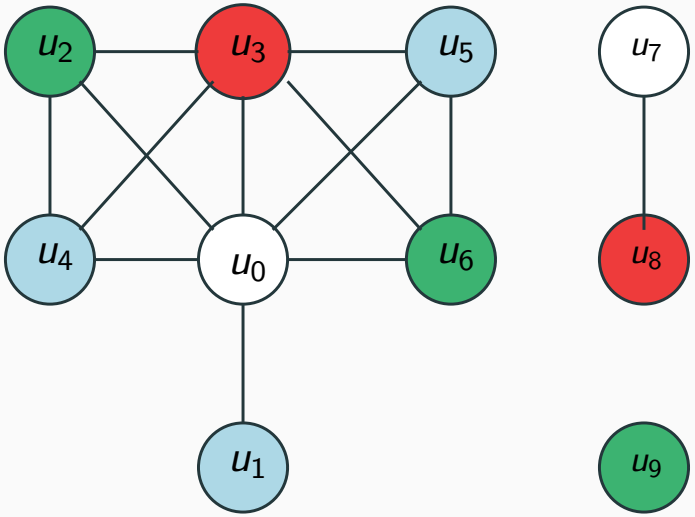




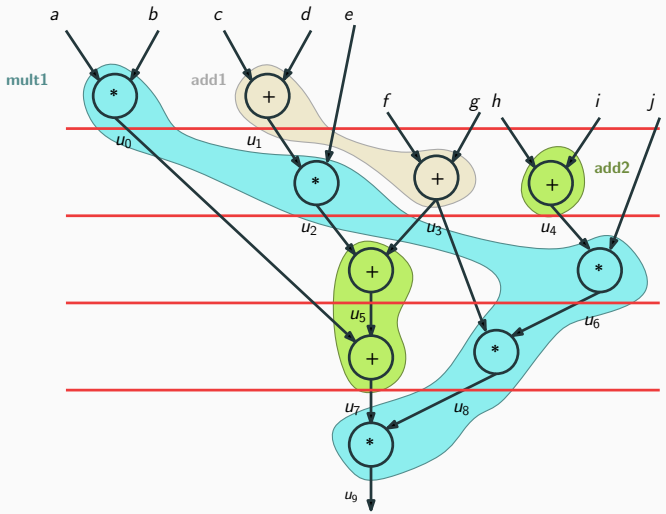
Grafo di incompatibilitá e binding dei registri

- Grafo con un vertice per ogni variabile e un arco fra due variabili se queste sono attive nello stesso ciclo di clock
- Il binding puó essere fatto utilizzando un algoritmo di graph coloring

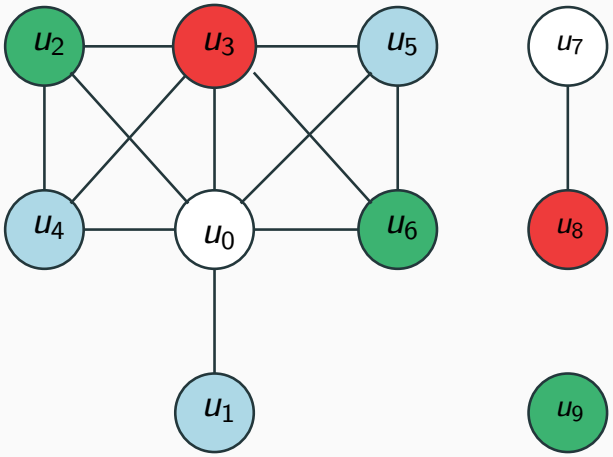




Descrizione RTL



clock	mult1	add1	add2
1	$R_0 = a * b$	$R_2 = c + d$	
2	$R_3 = R_2 * e$	$R_1 = f + g$	$R_2 = h + i$
3	$R_3 = R_2 * j$		$R_2 = R_3 + R_1$
4	$R_1 = R_1 * R_3$		$R_0 = R_0 + R_2$
5	$R_3 = R_0 * R_1$		



colore	registro	variabili
bianco	R_0	u_0, u_7
rosso	R_1	u_3, u_8
azzurro	R_2	u_1, u_4, u_5
verde	R_3	u_2, u_6, u_9

Inferenza dei multiplexer

Dal punto vista del data-path, questa descrizione semplificata che corrisponde a un livello RTL dice tutto meno che i multiplexer

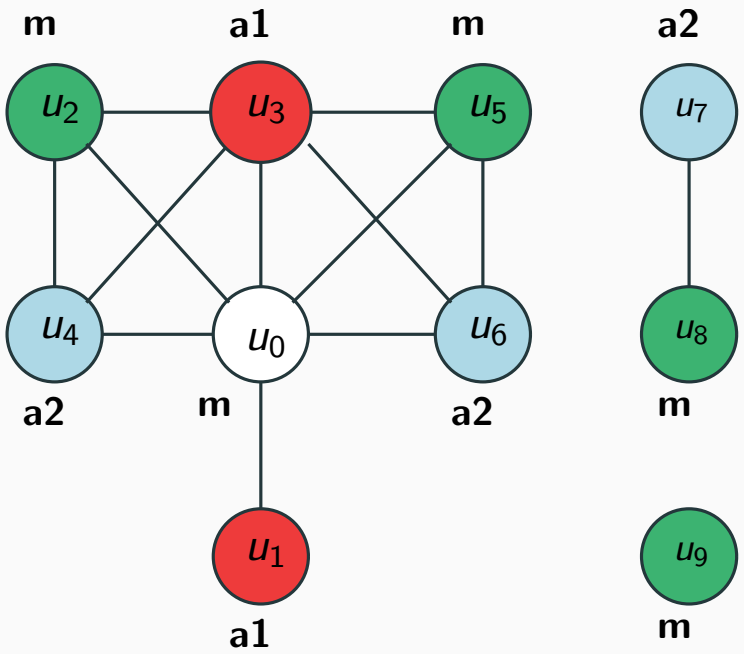
clock	mult1	add1	add2
1	$R_0 = a * b$	$R_2 = c + d$	
2	$R_3 = R_2 * e$	$R_1 = f + g$	$R_2 = h + i$
3	$R_3 = R_2 * j$		$R_2 = R_3 + R_1$
4	$R_1 = R_1 * R_3$		$R_0 = R_0 + R_2$
5	$R_3 = R_0 * R_1$		

blocco	input 1	input 2	mpx1	mpx2
mult1	a, R_0, R_1, R_2	b, e, j, R_3, R_1	4 way	5 way
adder1	c, f	d, g	2 way	2 way
adder2	h, R_0, R_3	i, R_1, R_2	3 way	3 way
R_0	mult1,add2		2 way	
R_1	mult1,add1		2 way	
R_2	add1,add2		2 way	
R_3	mult1,add1		2 way	

Relazione fra mpx e binding dei registri

- Il numero e il tipo di multiplexer utilizzati dipende dal binding delle risorse e dei registri
- Ci limitiamo a fare vedere come si possa ottenere un guadagno sul numero dei mpx anche se é possibile ottimizzare tutto contemporaneamente
- Cerchiamo di limitare il numero di multiplexer in ingresso ai registri utilizzando un euristico
- Nel grafo di incompatibilitá annotiamo ciascuna variabile con il blocco che la produce
- L'idea é quella di assegnare lo stesso colore a variabili prodotte dallo stesso blocco

Descrizione RTL con un minor numero di multiplexer



colore	registro	variabili
bianco	R_0	u_0
rosso	R_1	u_1, u_3
azzurro	R_2	u_4, u_5, u_7
verde	R_3	u_2, u_6, u_8, u_9

clock	mult1	add1	add2
1	$R_0 = a * b$	$R_1 = c+d$	
2	$R_3 = R_1 * e$	$R_1 = f+g$	$R_2 = h+i$
3	$R_3 = R_2 * j$		$R_2 = R_3+R_1$
4	$R_3 = R_1 * R_3$		$R_2 = R_0+R_2$
5	$R_3 = R_2 * R_3$		

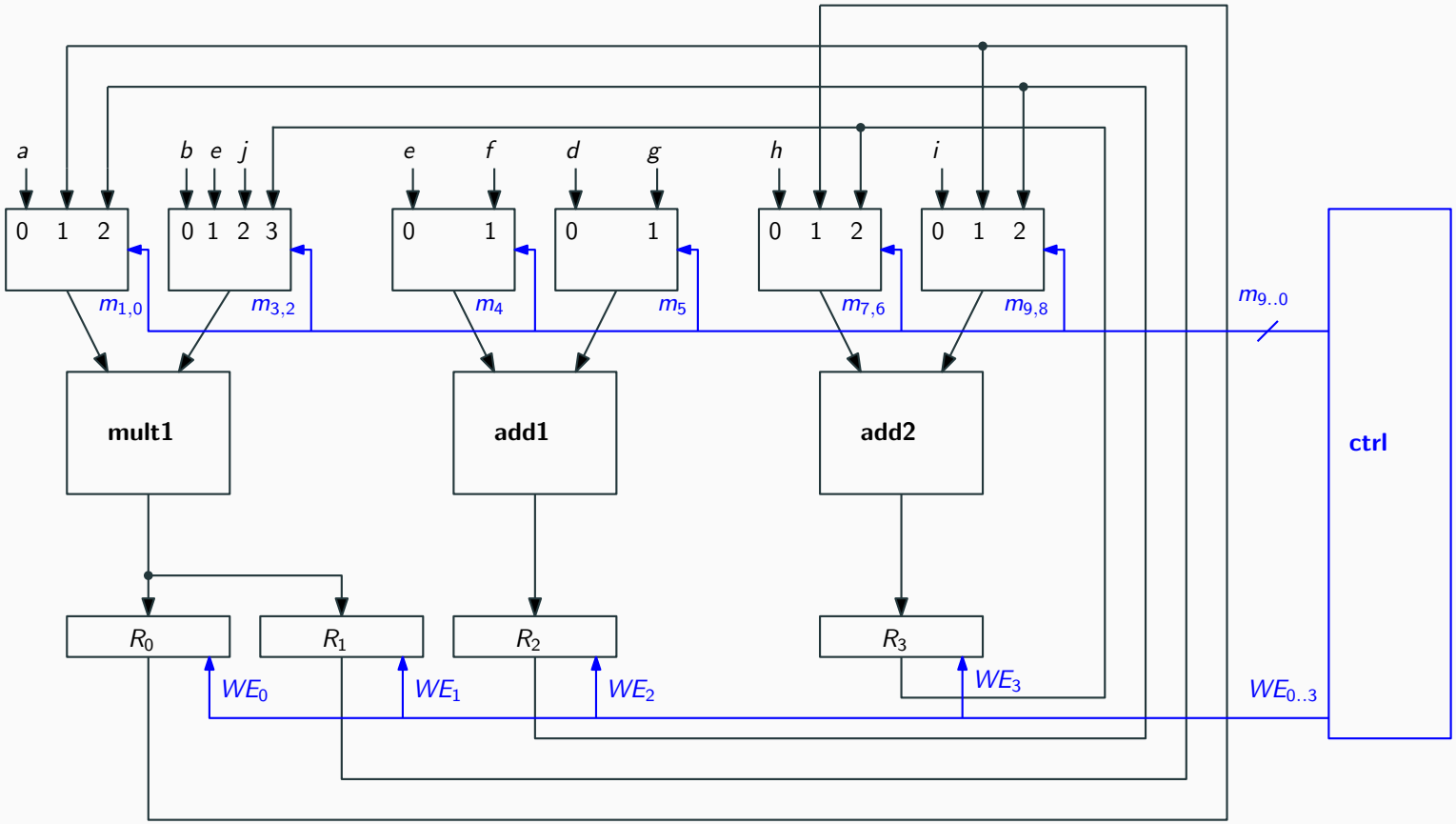
Descrizione RTL e relativi multiplexer

Il costo dei mpx si riduce di un fattore 0.7

clock	mult1	add1	add2
1	$R_0 = a * b$	$R_1 = c + d$	
2	$R_3 = R_1 * e$	$R_1 = f + g$	$R_2 = h + i$
3	$R_3 = R_2 * j$		$R_2 = R_3 + R_1$
4	$R_3 = R_1 * R_3$		$R_2 = R_0 + R_2$
5	$R_3 = R_2 * R_3$		

blocco	input 1	input 2	mpx1	mpx2
mult1	a, R_1, R_2	b, e, j, R_3	3 way	4 way
adder1	c, f	d, g	2 way	2 way
adder2	h, R_0, R_3	i, R_1, R_2	3 way	3 way
R_0	mult1			
R_1	add1			
R_2	add2			
R_3	mult1			

Schema di data-path e controllo



FSM del controllo

- La rete di controllo é realizzata supponendo che i dati in ingresso rimangano stabili per tutti e 5 i cicli di clock
- Questa macchina autonoma ha 5 stati e 14 uscite ($m_{9..0}, W_{0..3}$)
- Tabella di transizione dello stato

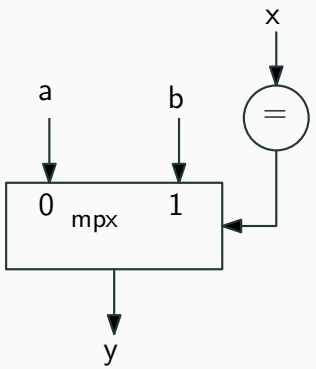
stato presente	stato futuro	$m_{1..0}, m_{3..2}, m_4, m_5, m_{7..6}, m_{9..8}, W_0, W_1, W_2, W_3$
A	B	00,00,0,0,- -,- -,1,1,0,0
B	C	01,01,1,1,00,00,1,1,0,0
C	D	10,10,- ,- ,00,00,0,1,1,1
D	E	01,11,- ,- ,01,01,0,0,1,1
E	A	10,11,- ,- ,10,10,0.0,0,1

Istruzioni di selezione e distribuzione

- Le istruzioni di selezione sono descritte in parte nel DFG (valutazione delle condizioni) e in parte nel CFG (generazione degli opportuni segnali di controllo)
- In casi molto semplici possono essere descritte direttamente nel DFG
- Lo stesso vale per le istruzioni di distribuzione

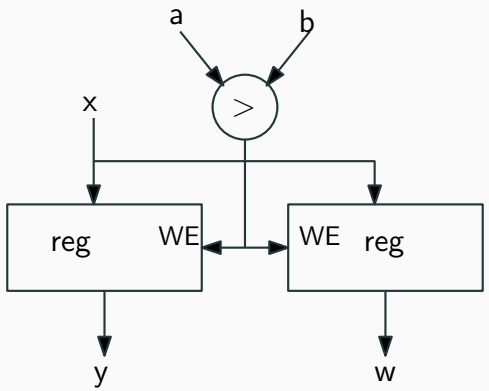
Selezione

```
if (x==0)
  y=a;
else
  y=b;
```



Distribuzione

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

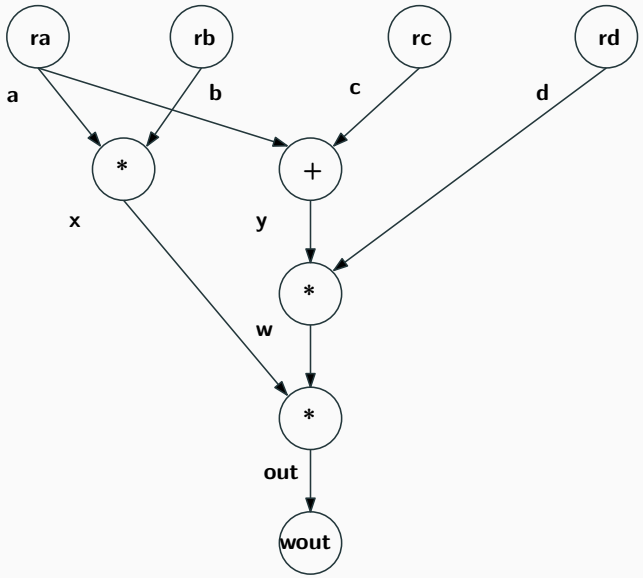


Esempio

Operazioni di lettura e scrittura da bus

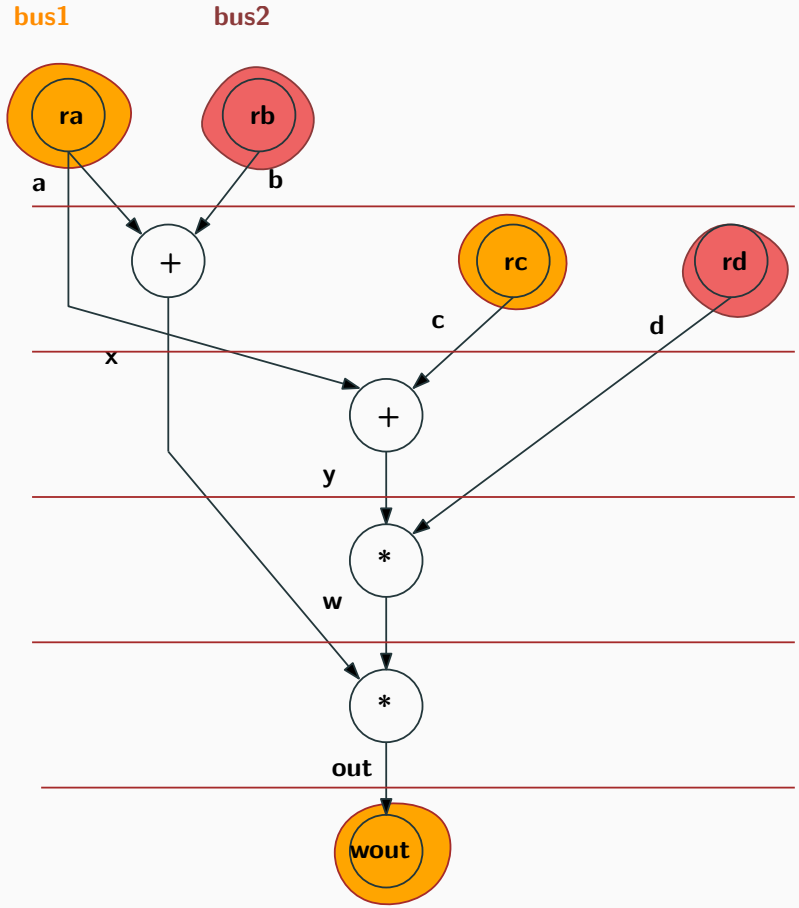
- Nei DFG considerati si é ipotizzato che tutti i dati siano disponibili contemporaneamente
- Questo in generale non é possibile a causa di limitazioni sulle connessioni o per l'accesso a un numero limitato di memorie
- Consideriamo qui un semplice algoritmo con esplicitate le operazioni di lettura e scrittura che si suppone avvengano impegnando come risorse dei bus bidirezionali

```
read a;
read b;
read c;
read d;
x=a*b;
y=a+c;
w=y*d;
out=x*w;
write out;
```



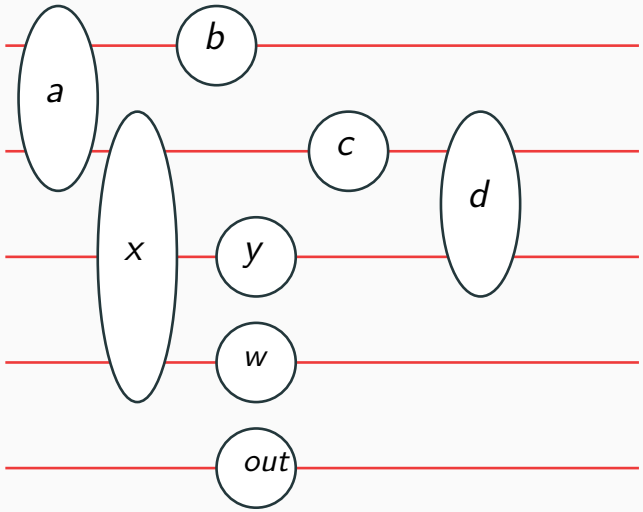
Scheduling con I/O ottimizzato

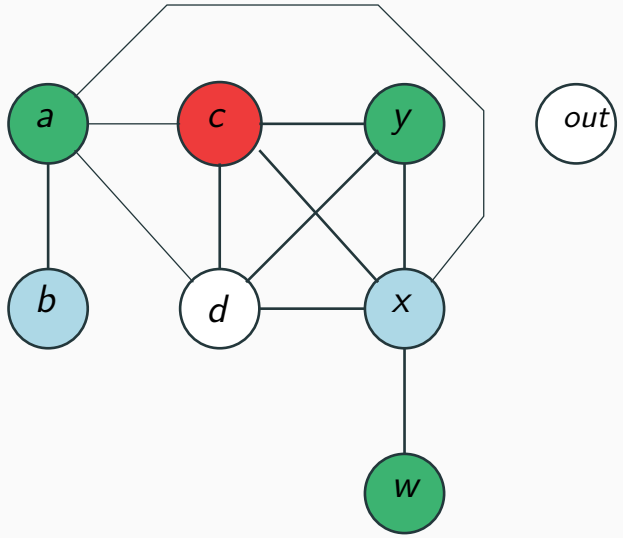
- Si supponga di avere a disposizione due bus oltre a un adder e un moltiplicatore
- Scheduling e binding con la latenza minimizzata



Ottimizzazione dei registri con input da bus

Se tutti gli ingressi non sono disponibili contemporaneamente, é possibile dover utilizzare alcuni registri aggiuntivi





ciclo	bus1	bus2	add	mult
1	$Rv = read\ a$	$Ra = read\ b$		
2	$Rr = read\ c$	$Rb = read\ d$	$Ra = Rv + Ra$	
3			$Rv = Rv + Rr$	
4				$Rv = Ra * Rv$
5				$Rb = Ra * Rv$
6	$write\ Rb$			

Schema del data path con I/O con bus