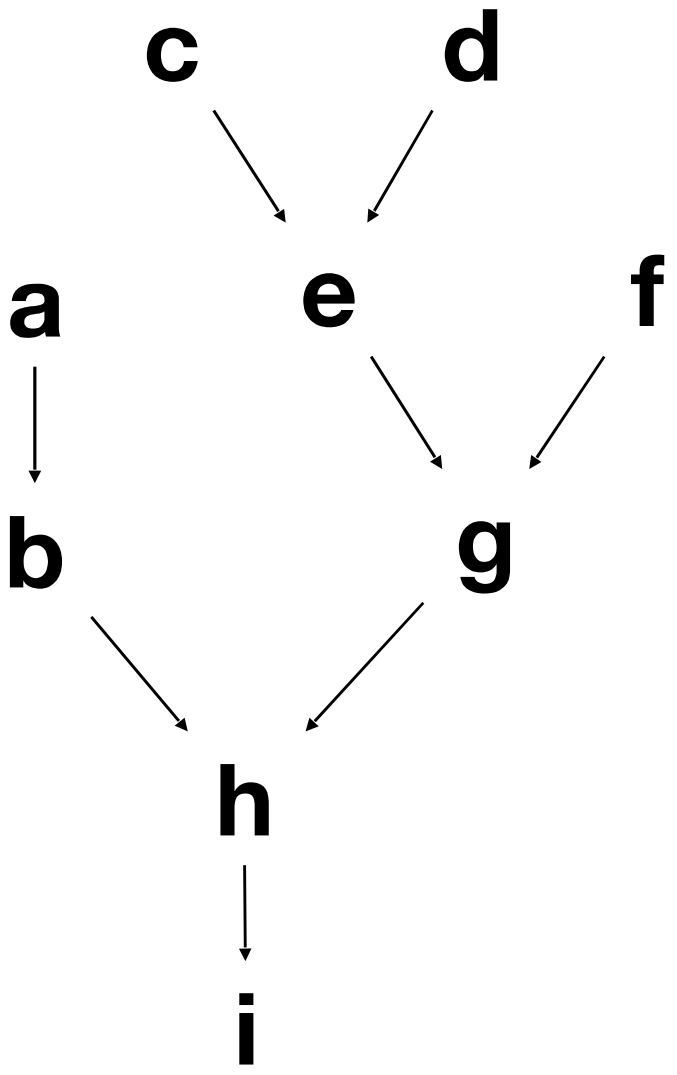
A simple schedule for **w ← x * 2 + 19 * y * z** is included below on the left, along with operation latencies. Produce an optimized schedule with the minimum completion time, using as many registers as you choose.

a	loadAI	r0, @x → r1
b	add	r1, r1 → r1
c	loadI	19 → r2
d	loadAI	r0, @y → r3
e	mult	r2, r3 → r2
f	loadAI	r0, @z → r3
g	mult	r2, r3 → r2
h	add	r1, r2 → r1
i	storeAI	r1 → r0,@w

Instruction	Cycles	Meaning
loadAI rx, c ⇒ rz	3	MEM(rx + c) → rz
storeAI rx ⇒ ry, c	3	rx → MEM(ry + cz)
loadI c ⇒ rx	1	c → rx
add rx, ry ⇒ rz	1	rx + ry → rz
mult rx, ry ⇒ rz	2	rx * ry → rz
Shift rx, c ⇒ ry	1	(rx << c) → ry

Dependence Graph



a	loadAI	r0, @x → r1
b	add	r1, r1 → r1
c	loadI	19 → r2
d	loadAI	r0, @y → r3
e	mult	r2, r3 → r2
f	loadAI	r0, @z → r3
g	mult	r2, r3 → r2
h	add	r1, r2 → r1
i	storeAI	r1 → r0,@w

Instruction	Cycles	Meaning
loadAI rx, c ⇒ rz	3	MEM(rx + c) → rz
storeAI rx ⇒ ry, c	3	rx → MEM(ry + cz)
loadI c ⇒ rx	1	c → rx
add rx, ry ⇒ rz	1	rx + ry → rz
mult rx, ry ⇒ rz	2	rx * ry → rz
Shift rx, c ⇒ ry	1	(rx << c) → ry

```
d: loadAI     r0, @y → r1          latency 3
f: loadAI     r0, @z → r2          latency 3
c: loadI      19 → r3              latency 1
a: loadAI     r0, @x → r4          latency 3
e: mult       r1, r3 → r5          latency 2
     NO-OP (due to use of r5 in g)
g: mult       r2, r5 → r6          latency 2
b: add        r4, r4 → r7          latency 1
h: add        r6, r7 → r8          latency 1
i:  storeAI   r8 → r0,@w           latency 3
     NO-OP
     NO-OP
```

The shortest possible schedule(assuming at most 1 instruction is issued per cycle) is 12 cycles. Attempt to re-order instructions to remove the NO-OP in the middle of the code may remove the NO-OP itself, but results in a new NO-OP due to other data dependencies.