University of Florida EEL 4930/5934 (Reconfigurable Computing)

Midterm 2 SAMPLE QUESTIONS – Fall 2007, Dr. Stitt, Dr. Lam

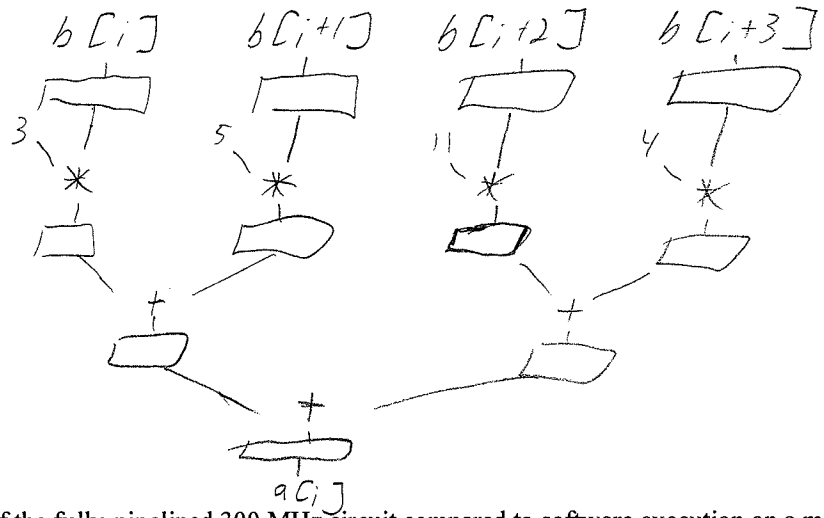
Name: _____ **UFID:**_____

Total Points (100)

1. Systolic arrays

a) Create a fully-pipelined datapath for the following code. Do not perform any optimizations (loop unrolling, etc.).

```
short a[100], b[104];
for (i=0; i < 100; i++)
    a[i] = 3*b[i] + 5*b[i+1] + 11*b[i+2] + 4*b[i+3];
```



b) Calculate speedup of the fully-pipelined 300 MHz circuit compared to software execution on a microprocessor (assuming 20 instructions for each iteration, a CPI of 1.5, and a clock frequency of 3 GHz). Assume memory bandwidth is sufficient for full pipelining.

Latency = 5 cycles, iteration 0 takes 5 cycles, iteration 1-99 takes 1 cycle

Total HW Cycles = 5 + 99 = 104 cycles

Total SW Cycles = total instructions * CPI = (20*100) * 1.5 = 3000 cycles

$$\text{speedup} = \frac{\text{SW time}}{\text{HW time}} = \frac{3000}{104} * \frac{300\ MHz}{3000\ MHz} = 2.8$$

c) Assuming that memory can deliver 128 bits per cycle, how many iterations of the loop can be performed in parallel? (Ignore any creative buffering, i.e. assume that data from previous memory fetches is not reused)

1st iteration requires $16 \times 4 = 64$ bits, each unrolled iteration requires 16 bits

bits available for unrolling = $128 - 64 = 64$ bits

amount of possible unrolling = $\frac{64}{16} = 4$

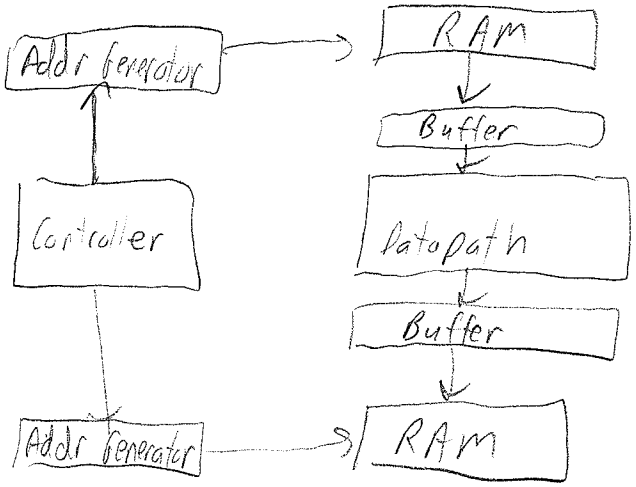
total parallel iterations = $4 + 1 = \boxed{5}$

d) Calculate the speedup of the circuit when utilizing the amount of loop unrolling determined in part c

$$\text{total HW cycles} = 5 + \frac{\text{remaining iterations}}{\text{parallel iterations}} = 5 + \frac{95}{5} = 24 \text{ cycles}$$

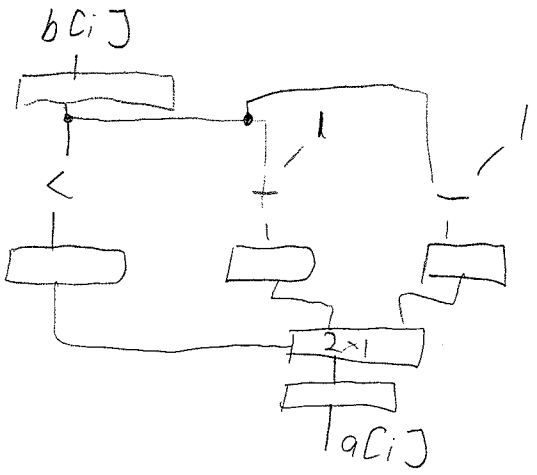
$$\text{speedup} = \frac{3000}{24} \times \frac{300}{3000} = \boxed{12.5}$$

e) Draw a block diagram of the entire circuit (not just the datapath)



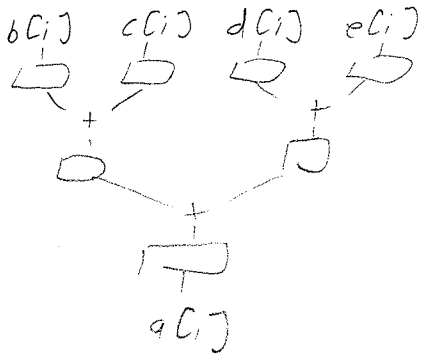
2. Create a fully-pipelined datapath for the following code. Do not unroll the loop.

```
short a[100], b[100];
for (i=0; i < 100; i++){
    if (b[i] < 0)
       a[i] = b[i]+1;
    else
       a[i] = b[i]-1;
}
```



3. Create a fully-pipelined datapath for the following code. Do not perform any optimizations. Explain why without any optimizations, the pipeline may frequently stall. Be specific.

```
short a[100], b[100], c[100], d[100], e[100];
for (i=0; i < 100; i++){
    a[i] = b[i]+c[i]+d[i]+e[i];
}
```



The pipeline would likely stall because the 4 arrays may be stored in different parts of memory.

4. Create a fully-pipelined datapath for the following code. List any optimizations needed to eliminate loop-carried dependencies. Show that such optimizations enable a fully pipelined datapath, assuming memory delivers 128 bits of data every cycle.

```
short a[100], b[104];
for (i=0; i < 100; i++){
    for (j=i; j < i+4; j++) {
       val += b[j];
    }
    a[i] = val;
}
```

val = 0;

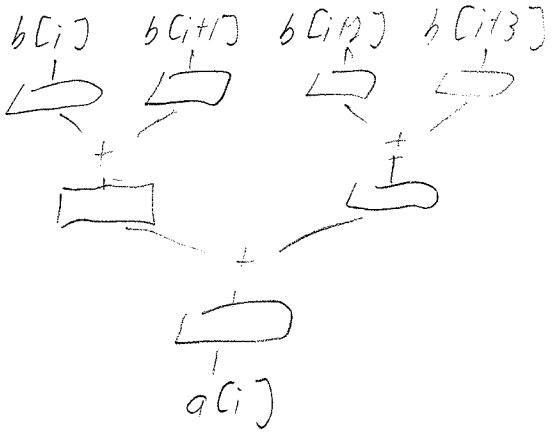
1) unroll inner loop =>

```
val += b[i];
val += b[i+1];
val += b[i+2];
val += b[i+3];
```

2) tree-height reduction =>

```
val = val + b[i] + b[i+1]
    + b[i+2] + b[i+3];
```

3) constant propagation => val = b[i] + b[i+1] + b[i+2] + b[i+3]

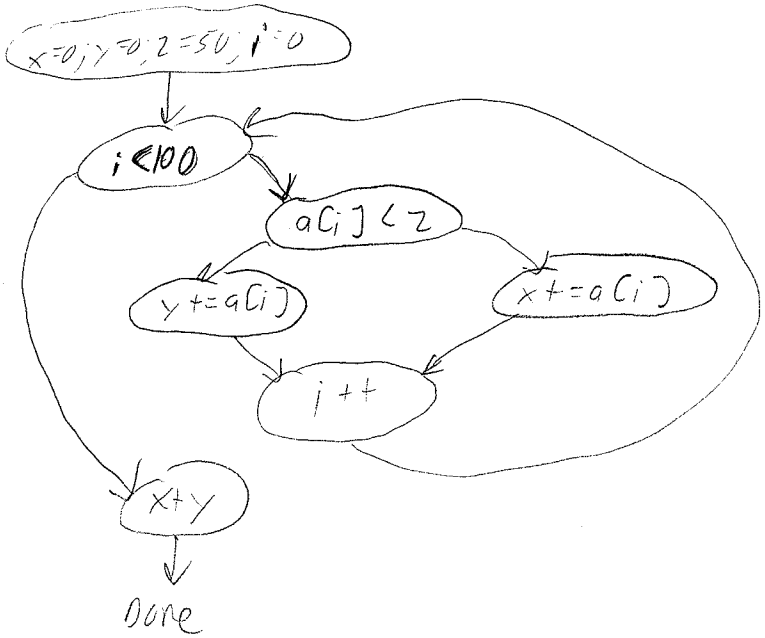


For unrolling to not cause stalls,
circuit requires 4 inputs each cycle
required bandwidth = 4×16 = 64 bits/cycle
actual bandwidth = 128 bits/cycle
unrolling inner loop does not cause stalls

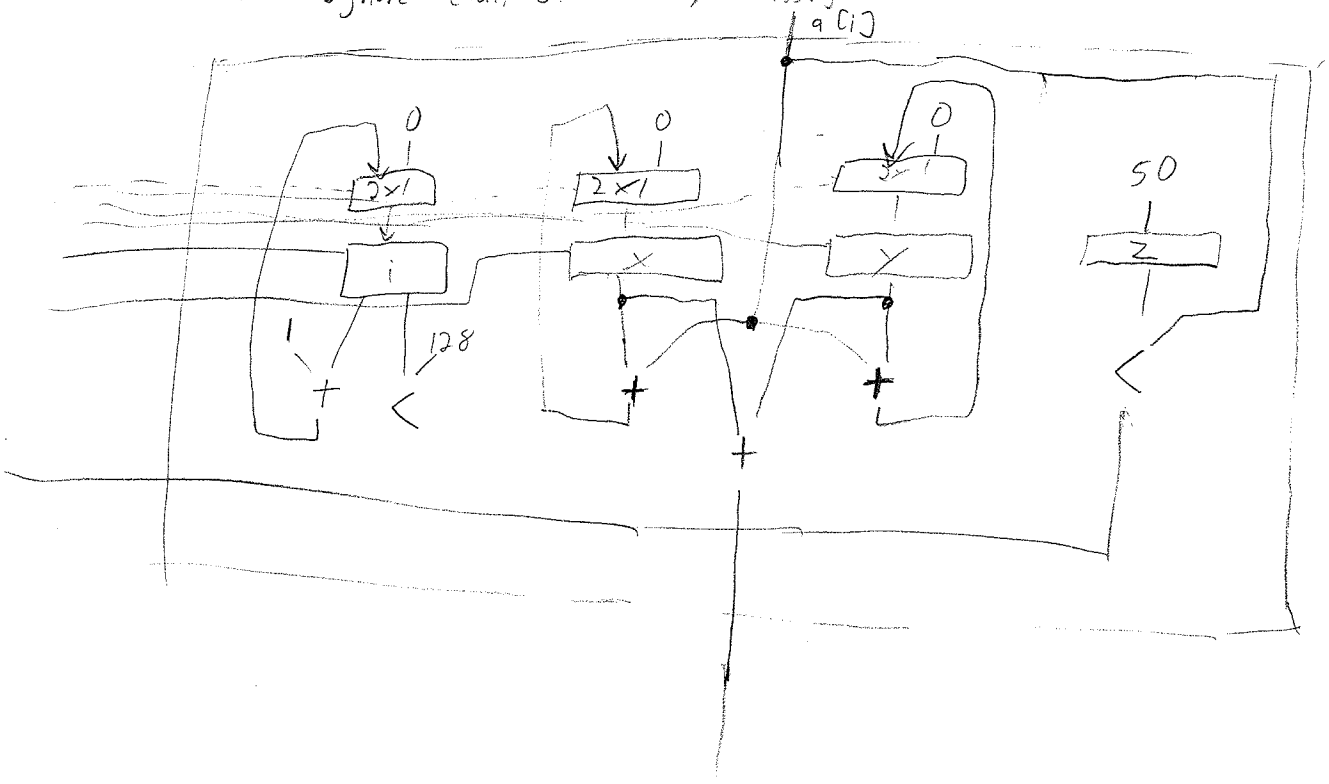
5. For the following code:

```
int f( short a[100] ) {
  int x,y,z;
  x=0; y=0; z=50;
  for (i=0; i < 100; i++){
      if (a[i] < z) {
         x += a[i];
      }
      else {
         y += a[i];
      }
  }
  return x+y;
}
```

(a) Manually create a FSM representing the controller, using the methodology discussed in the lecture slides. (Do not create a systolic array) Ignore details of memory accesses



(b) Manually create a datapath that works with the controller from part a. Show all inputs/ouputs, and control signals. Ignore detail of memory accesses.



(c) Calculate the execution time in terms of cycles. List assumptions

assume 1 cycle per state

1 iteration requires 4 cycles, 100 iterations ⇒ 400 cycles for the loop

total cycles = 400 + 1 + 1 = 402 cycles

(d) Explain why this circuit is likely much slower than a systolic array-based circuit, assuming a systolic array is possible.

not pipelined, each iteration takes 4 cycles

systolic array can do 1 or more iterations per cycle

(e) ~~Explain the difficulty in creating a systolic array-based circuit for this code.~~

6. Optimize the following code, by showing the resulting code after each applied technique.

```
x = 0;
y = a + b;
if (x < 15)
   z = a + b - c;
else
   z = x + 12;
o = z * 12;
```

1) const. prop. =>

```
x=0;
y=a+b;
if(0<15)
   z=a+b-c;
else
   z=12;
o=z*12;
```

2) dead code elim.

```
x=0;
y=a+b;
z=a+b-c;
o=z*12;
```

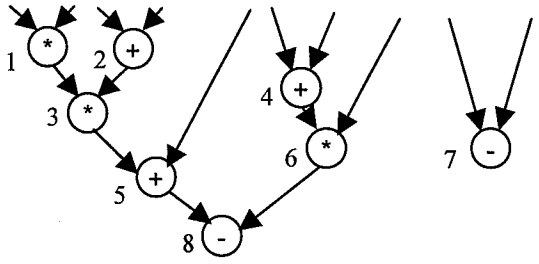
3) common subexpression elim.

```
x=0;
y=a+b;
z=y-c;
o=z*12;
```

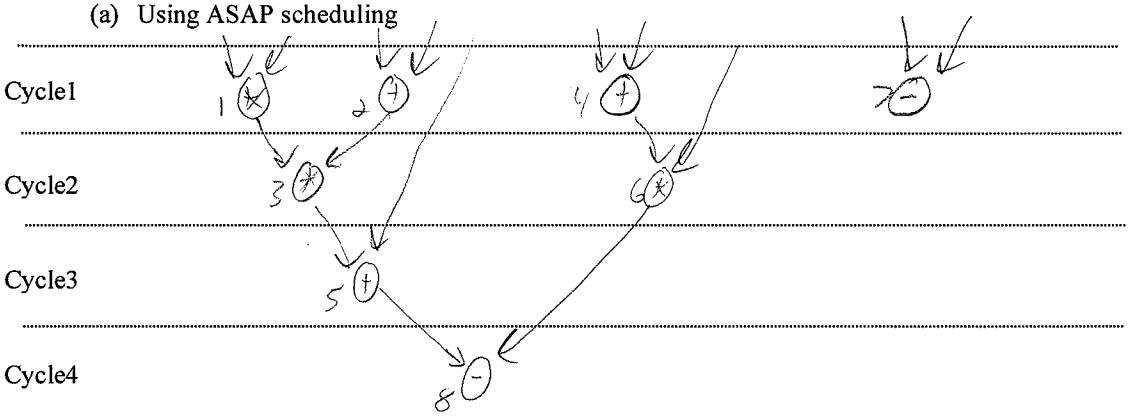
4) strength reduction

```
x=0;
y=a+b;
z=y-c;
o=(z<<3)+(z<<2);
```

7. Schedule the following circuit: (using minimum latency scheduling)

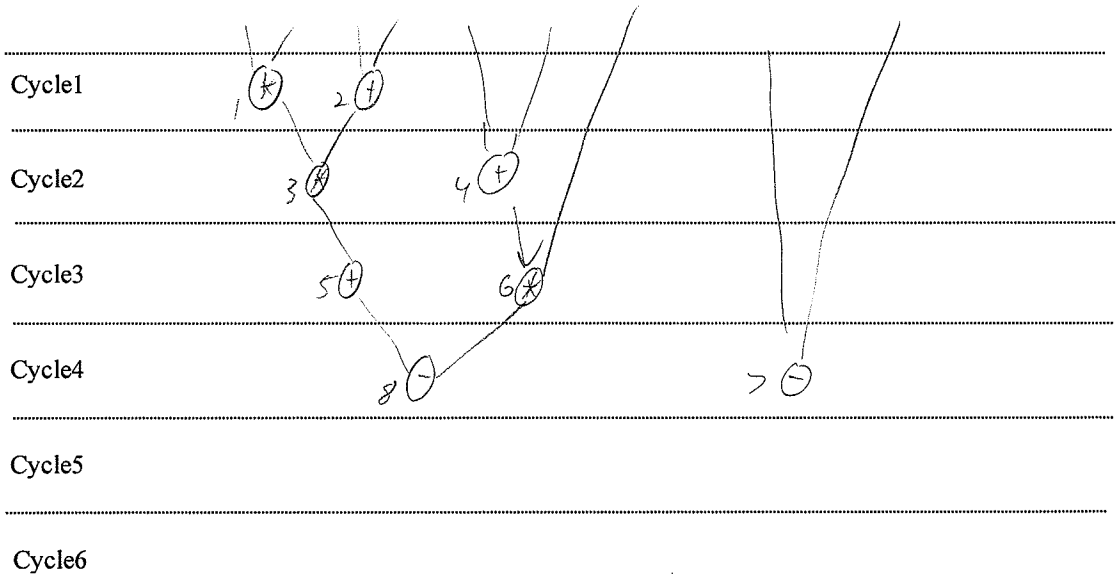


(a) Using ASAP scheduling



minimum latency

(b) Using ALAP scheduling



c) What is the minimum number of resources required for the ASAP schedule? Assuming +/- for ALU, * for Mult

3 ALUs, 2 Mult

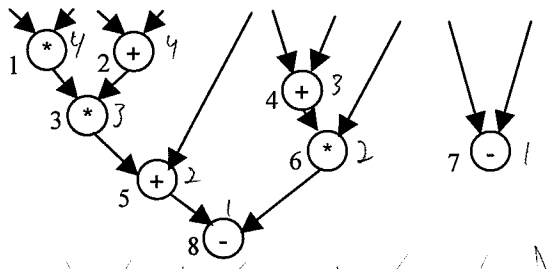
d) What is the minimum number of resources required for the ALAP schedule?

2 ALUs, 1 mult

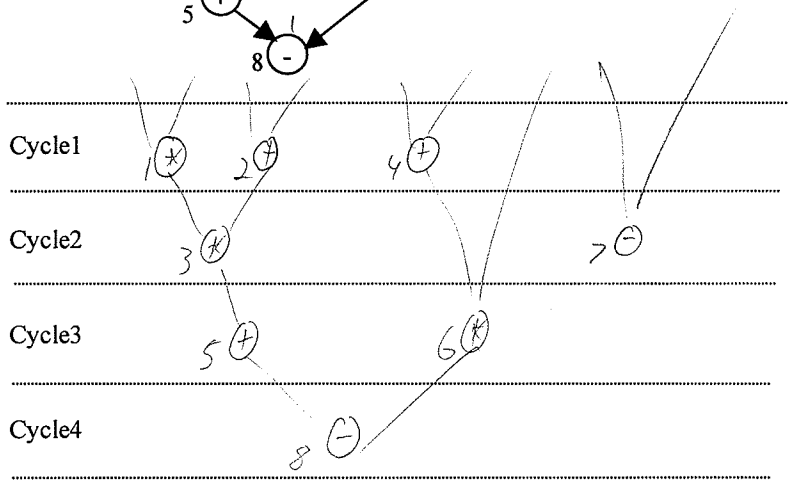
e) How would the resource requirements change for a fully-pipelined implementation?

5 ALUs, 3 mult

8. Trace the steps of ***minimum-latency resource-constrained*** list scheduling on the following DFG. Draw priority next to each node.



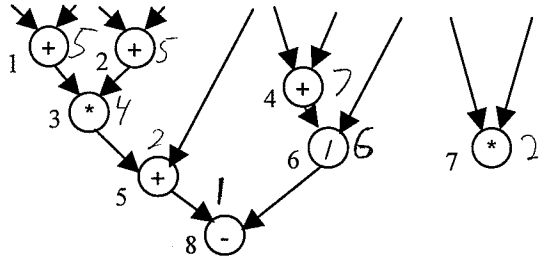
Resource Constraints: 2 ALU(+/-), 1 mult



Candidates

ALUs	Mults
2, 4, 7	1
~~7~~	~~3~~, 6
~~3~~	~~6~~
8	

9. Trace the steps of ***minimum-latency resource-constrained*** list scheduling on the following DFG, assuming that multiplications take 2 cycles and divides take 5 cycles. Draw priority next to each node.

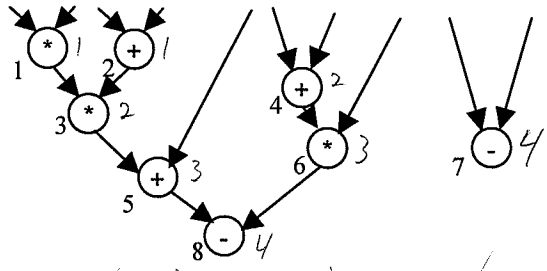


Resource Constraints: 2 ALU(+/-), 2 mults, 1 div

Cycle1
Cycle2
Cycle3
Cycle4
Cycle5
Cycle6
Cycle7
Cycle8
Cycle9
Cycle10

Candidates		
ALUs	Mults	DIV

10. Trace the steps of ***minimum-resource latency-constrained*** list scheduling on the following DFG. Show the last possible cycle next to each node in the original graph. Show the slack for each candidate in parentheses. Show the resource requirements at each step, and the final requirements.



Latency constraint = 4 cycles

Resources

ALUs	Mults
1	1
1	1
1	1
2	1

Cycle1
Cycle2
Cycle3
Cycle4
Cycle5
Cycle6

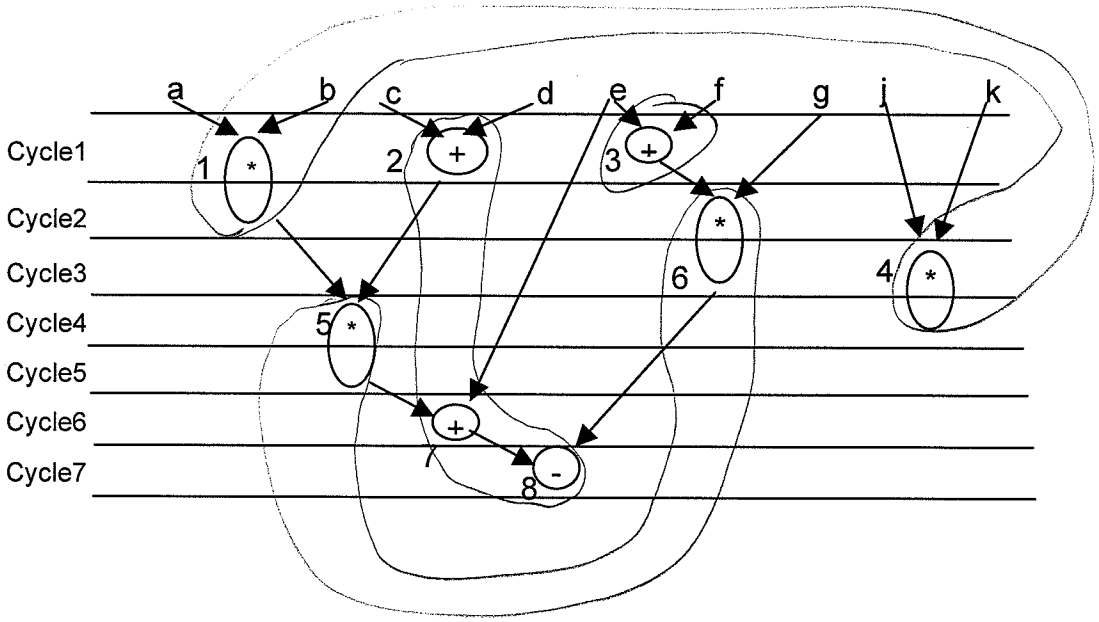
Candidates

ALUs	Mults
1(0), 4(1), 7(3)	1(0)
4(0), 7(2)	3(0)
3(0), 7(1)	6(0)
8(0), 7(0)	

Required resources

2 ALUs, 1 Mult

11. Bind the following scheduled DFG using clique partitioning. Show the compatibility graph and cliques. Assume that the circuit uses 2 multipliers and 2 ALUs.



Clique partitioning

1 — 4

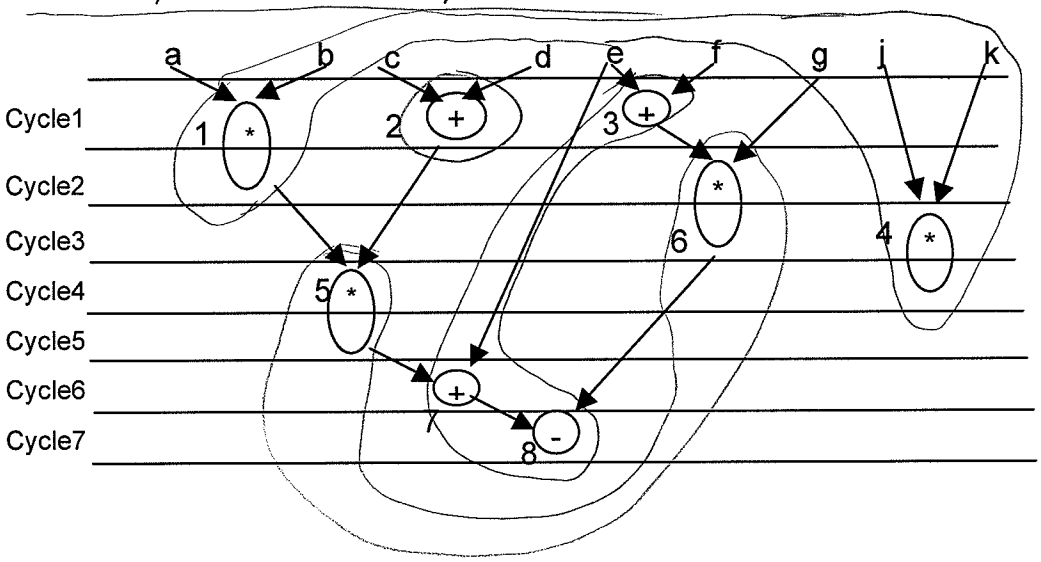
5 — 6

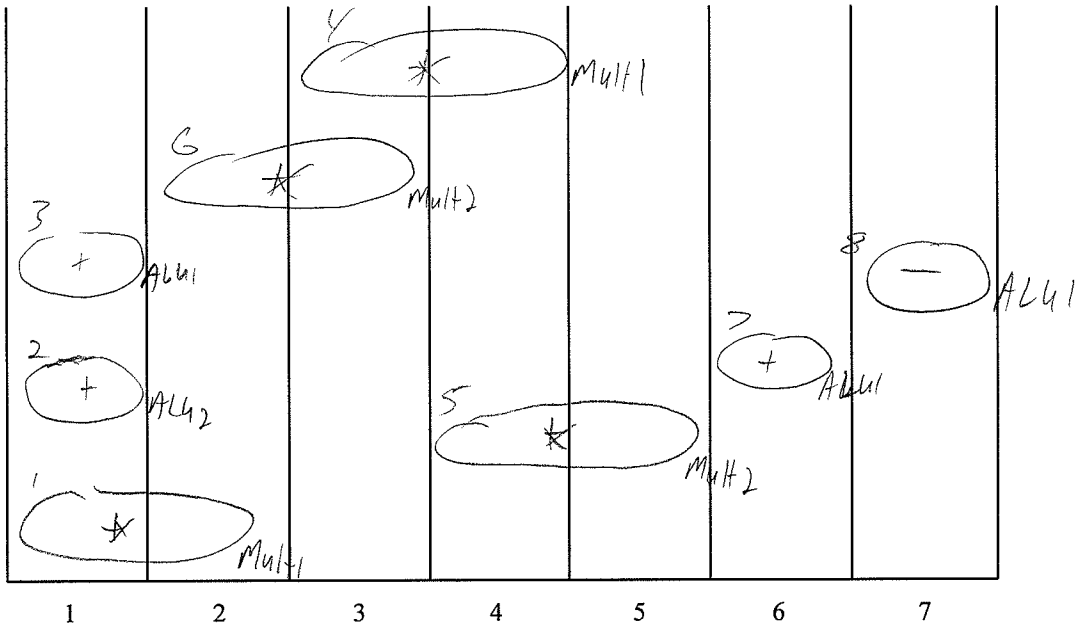
2

3

7 — 8

12. Bind the following scheduled DFG using the left edge algorithm. Assume that the circuit uses 2 multipliers and 2 ALUs. Work top to bottom if multiple candidates.





ALU1 = 3, 7, 8

ALU2 = 2

Mult1 = 1, 4

Mult2 = 6, 5

13. Translate the binding from the previous problem into a datapath.