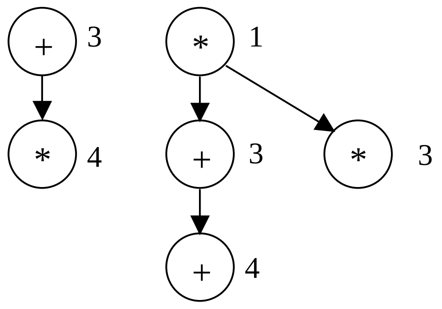
# E4/I4/MSc
# Synthesis of Digital Architectures

## Specimen Paper

7/2/03, gac1@ic.ac.uk

1.

A data-flow graph is shown below. Next to each node is its scheduled start time. Multipliers are assumed to take two cycles and adders to take one cycle.



a) Construct both a multiplier and an adder conflict graph for the scheduled DFG

**[6 marks]**

b) State a property of conflict graphs arising from DFGs that enables them to be coloured optimally in an efficient manner

**[2 marks]**

c) Colour your conflict graphs from part (a) optimally

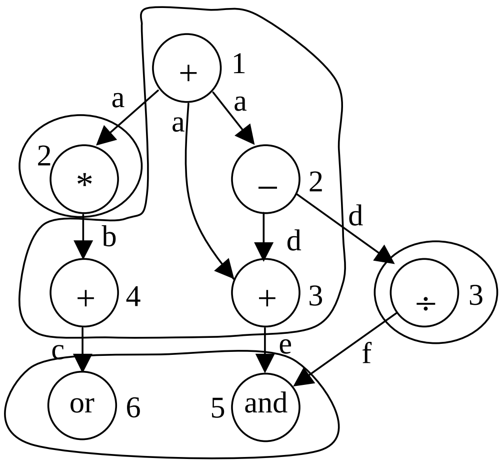
**[6 marks]**

d) State the chromatic number and clique number of each conflict graph

**[6 marks]**

2.

A scheduled and bound DFG is shown below. The number next to each node represents the start time of the operation, the alphabetic label on each edge denotes the variable causing the data-dependency, and hyper-edges denote operations sharing a single resource. Multiplications and divisions take two cycles, all other operations take one cycle. Multi-cycle operations only need valid inputs on the first of its two cycles of operation. The resource set $R = \{*, \div, +/-, \text{and/or}\}$.



a) Design a datapath for this DFG. Your design must include: registers, multiplexers, and computational resources. You may leave external inputs and/or outputs unlabelled. You may ignore the control lines to select between functions performed by a single resource.

**[4 marks]**

b) Design a horizontal microcode controller for your datapath. You may still ignore control lines to select between functions performed by a single resource.

**[3 marks]**

c) Optimize your controller by using a combination of a smaller ROM and $n$-to-$2^n$ decoders

**[3 marks]**

3.

Some pseudo-code used to calculate `y = 5*a`, `z = 3*a` and `q = 10*a` is shown below. Addition and subtraction each take one clock cycle.

```
x1 = a + a;
x2 = x1 + x1;
y = x2 + a;
z = x2 - a;
q = y + y;
```

a) Construct a CDFG for the code shown. Add numerical weights to the CDFG edges representing the number of cycles taken by each operation.

**[3 marks]**

b) Perform an ASAP and ALAP scheduling of the CDFG, given that all operations must complete within a deadline of 4 cycles.

**[3 marks]**

c) State the mobility of each node and identify the critical path.

**[3 marks]**

The code above is repeated below with some lines marked. It is required that the lines of code marked (*) execute during the same clock cycle.

```
x1 = a + a;
x2 = x1 + x1;
y = x2 + a;     (*)
z = x2 - a;     (*)
q = y + y;
```

d) Modify your graph to incorporate this constraint.

**[3 marks]**

e) Perform an ASAP and ALAP scheduling of the modified graph, given that all operations must complete within a deadline of 4 cycles.

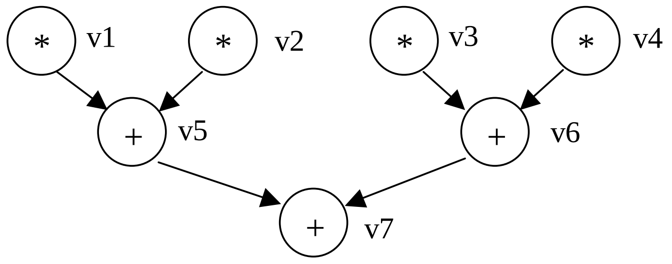
**[5 marks]**

f) State the new mobility of each node and identify the critical path.

**[3 marks]**

4.

A data-flow graph is shown below. Each vertex has been labelled.



a) The behaviour described by the graph is to be implemented using at most two adders and three multipliers. Each multiplier takes two clock cycles and each adder takes one clock cycle.

Name an appropriate algorithm to schedule the DFG

**[2 marks]**

b) Apply the selected algorithm, and state the resulting start cycle for each operation

**[7 marks]**

c) State the overall latency exhibited by your schedule

**[2 marks]**

After presenting your schedule to your customer, she tells you that the overall latency exhibited could be extended by two cycles, if it will result in cheaper hardware.

d) Name an appropriate scheduling algorithm to explore this possibility

**[2 marks]**

e) Apply the selected algorithm, giving the start cycle of each operation, and state how many multipliers and adders you require.

**[7 marks]**

5.

The pseudo-code shown below finds the value $\sqrt{|b^2 - 4ac|}$, used in the solution of quadratic equations.

```
t1 = b*b;           // line 1
t2 = 4*a;           // line 2
t3 = t2*c;          // line 3
t4 = t1 - t3;       // line 4
t5 = (t4 ≥ 0);      // line 5
if t5 then          // line 6
   d = sqrt(t4);    // line 7
else
   t6 = -t4;        // line 8
   d = sqrt(t6);    // line 9
end if;
```

a) Construct a CDFG for this code (you do not need to draw the body of the `sqrt()` function call)

**[6 marks]**

We may assume that the body of the `sqrt()` function call is implemented by a square-root resource, of unbounded latency. All other operations take a single cycle.

The scheduled start times are shown below.

```
Cycle 0: line 1, line 2
Cycle 1: line 3
Cycle 2: line 4
Cycle 3: line 5
Cycle 4: line 6
Cycle 5: line 8
Cycle 6: line 7, line 9
```

b) Draw a conflict graph for each resource type in $R = \{* , - , \text{sqrt}, \geq\}$

**[4 marks]**

c) How many resources of each type are required?

**[2 marks]**

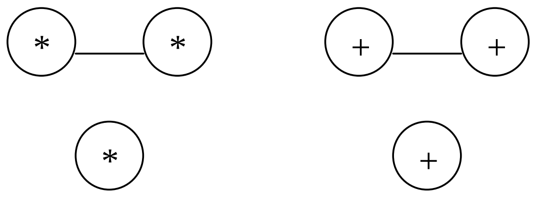
d) Sketch a datapath for this design. (You may leave external inputs unconnected)

**[8 marks]**

# Model Solutions

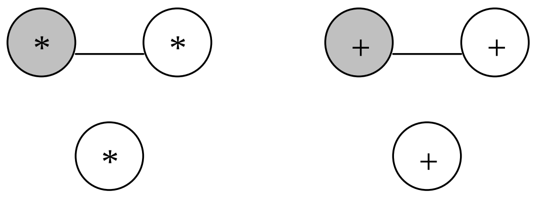
1.

a)



b) Conflict graphs arising from DFGs are interval graphs (c.f. conflict graphs arising from CDFGs)

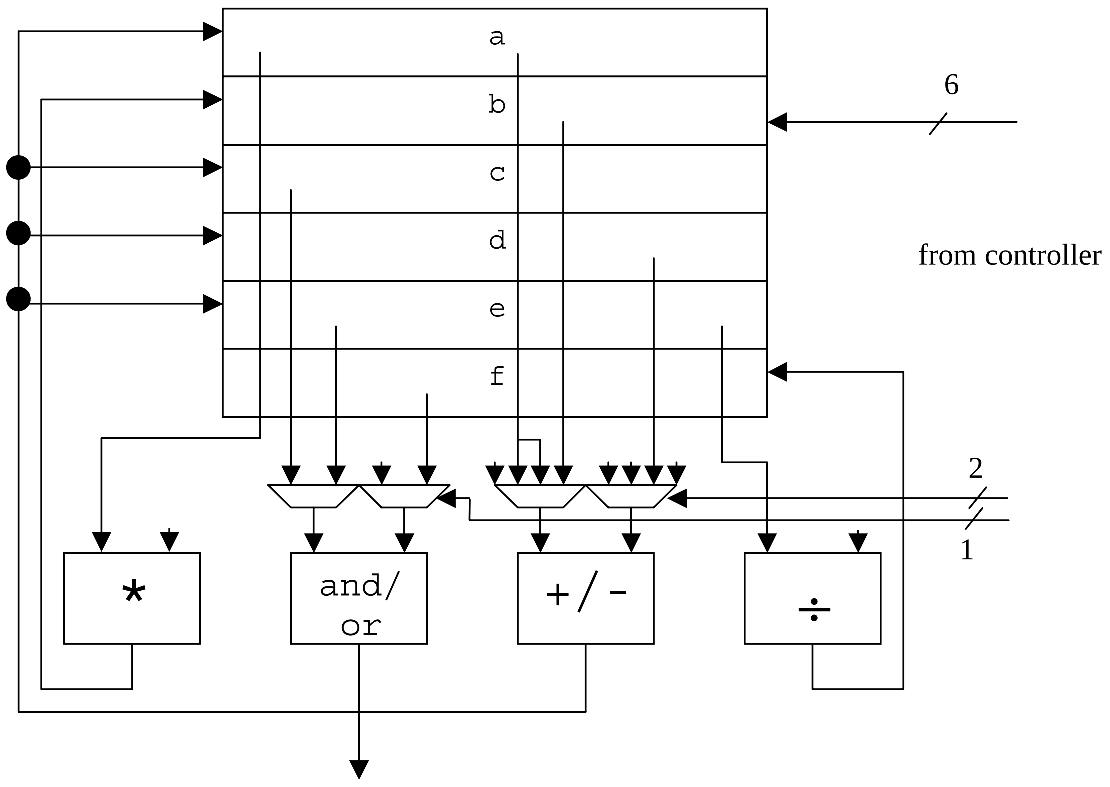
c)



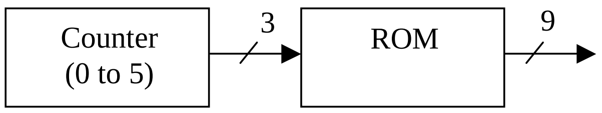
d) In each case, chromatic number is 2, clique number is 2.

2.

a)



b)

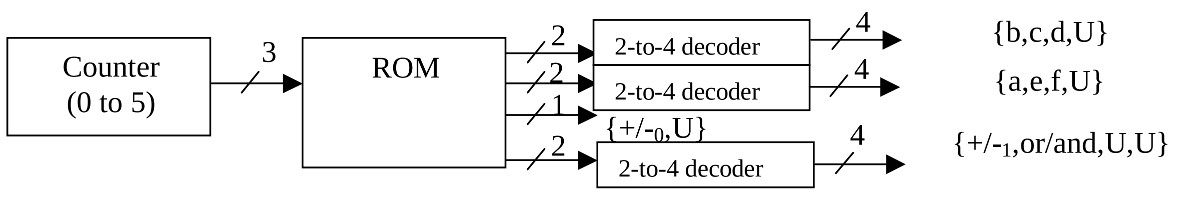


For ROM contents, see table below (d/c = don't care)

| **Cycle** | **enable signals** | | | | | | **select signals** | |
|---|---|---|---|---|---|---|---|---|
| | **a** | **b** | **c** | **d** | **e** | **f** | **or/and** | **+/-** |
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | d/c | 00 |
| **1** | 0 | 0 | 0 | 1 | 0 | 0 | d/c | 01 |
| **2** | 0 | 1 | 0 | 0 | 1 | 0 | d/c | 10 |
| **3** | 0 | 0 | 1 | 0 | 0 | 1 | d/c | 11 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d/c |
| **5** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | d/c |

c) Possible sets: {b,c,d} ; {a,e,f} ; {+/-$_1$,or/and}, {+/-$_0$}

These lead to the following design (U = unused)

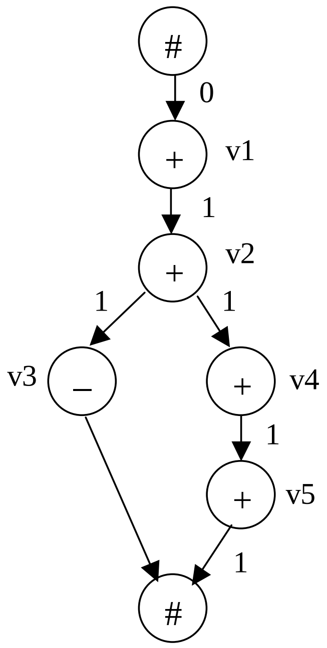


The new ROM contents are given in the table below.

| **Cycle** | **{b,c,d}** | **{a,e,f}** | **{+/-$_1$,or/and}** | **{+/-$_0$}** |
|---|---|---|---|---|
| **0** | 00 | 11 | 00 | 0 |
| **1** | 01 | 00 | 00 | 1 |
| **2** | 11 | 10 | 11 | 0 |
| **3** | 10 | 01 | 11 | 1 |
| **4** | 00 | 00 | 00 | 0 |
| **5** | 00 | 00 | 01 | 0 |

3.

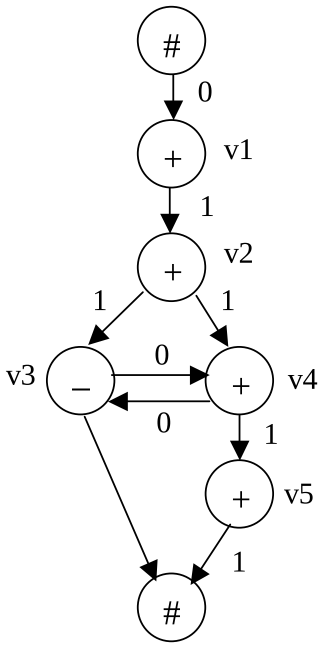
a)



b)  v1: 0 – 0, v2: 1 – 1, v3: 2 – 3, v4: 2 – 2, v5: 3 - 3

c) v1: 0, v2: 0, v3: 1, v4: 0, v5: 0. Critical path = {v1, v2, v4, v5}

d)



e) v1: 0 – 0, v2: 1 – 1, v3: 2 – 2, v4: 2 – 2, v5: 3- 3

f) All have zero mobility. Critical path is all nodes {v1, v2, v3, v4, v5}

4.

a) resource-constrained list scheduling

b) Cycle 0: {v1, v2, v3} ; Cycle 2: {v4, v5} ; Cycle 4: {v6} ; Cycle 5: {v7}

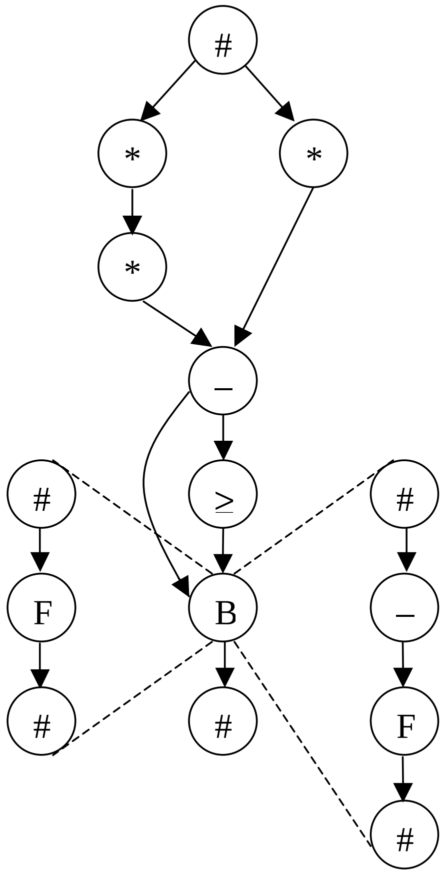
c) 5 cycles

d) latency-constrained list scheduling

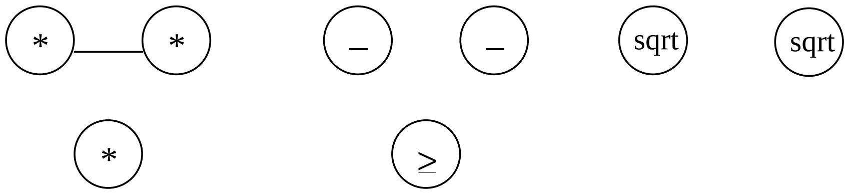
e) Cycle 0: {v1} ; Cycle 2: {v2} ; Cycle 4: {v3, v4, v5} ; Cycle 6: {v6} ; Cycle 7: {v7} (# of mults is expanded from 1 to 2 at cycle 4). This requires 2 mults and 1 add.

5.

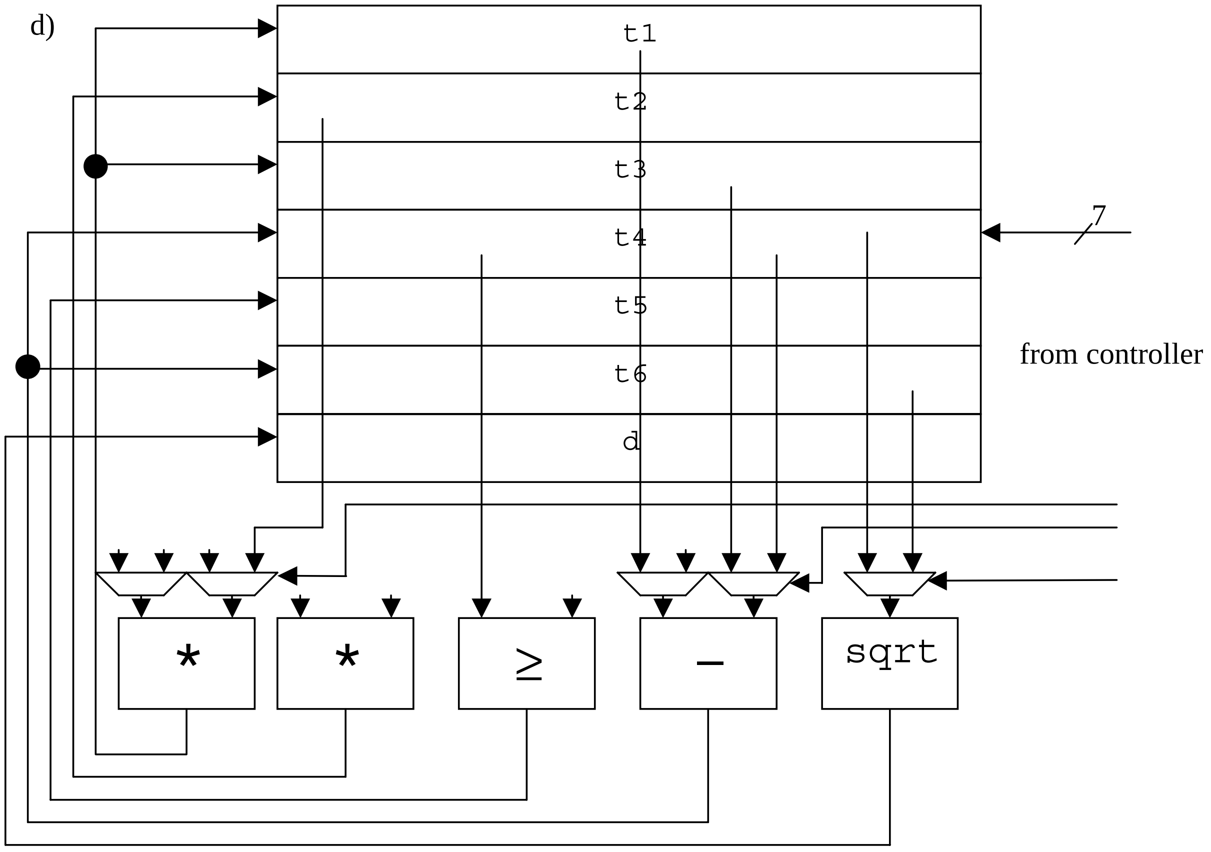
a)



b)



c) Two mults and one of every other resource type d)


t1
t2
t3
t4
t5
t6
d
7
from controller
*
*
≥
−
sqrt