Engineering Part IIA and EIST Part I 1996
Paper E6 (Computing Systems) Crib

1 Question 1

(a) The hazards are between the **and** and `lw` instructions (**and** needs `$2` before `lw` has written to it) and the **sub** and **and** instructions (**sub** needs `$12` before **and** has written to it).

Without dataforwarding, the register file is written at pipe stage 5 and read at pipe stage 2, so the pipeline has to be stalled for 3 cycles between dependent instructions. The time taken to complete three instructions is therefore $5+2=7$ clock cycles running and $3+3=6$ clock cycles stalled, a total of 13 clock cycles.

With dataforwarding the result of the **and** can be forwarded from the **EX/MEM** pipe register to the **ALU** input, so the **sub** can follow directly, without a stall. The result of the `lw` can be forwarded from the **MEM/WB** pipe register to the **ALU** input, so the **and** can follow after only one stall. The time taken to complete the three instructions is therefore $5+2=7$ clock cycles running and 1 clock cycle stalled, a total of 8 clock cycles.

(b)

(i) Polling

Advantages: Simplest way to handle I/O. No special hardware required.

Disadvantages: Requires constant attention from CPU, even when device is idle.

Used for: Low bandwidth device like mice. Or battery checking.

(ii) Interupt driven I/O

Advantages: No burden on CPU when idle, no special hardware required.

Disadvantages: Would still overlaod the CPU for high bandwidth devices (when not idle).

Used for: many low/medium bandwidth devices like printers, keyboards, floppy disks.

(iii) DMA

Advantages: Relieves CPU of I/O burden for high bandwidth devices.

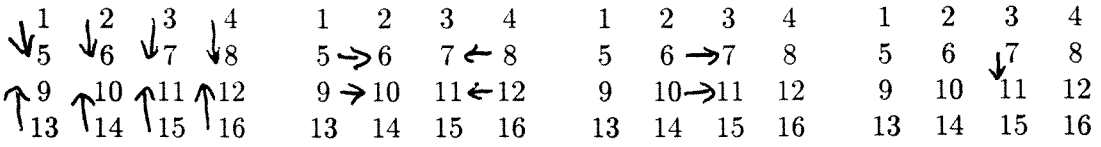
Disadvantages: Requires special controller. Cache coherency complications in memory system.

Used for: High bandwidth devices, like hard disks.

(c) In the first iteration, the partial sums are sent from processors 9-16 to processors 1-8. All the traffic has to travel through the links between processors 5-8 and 9-12, creating a bottleneck. The network performs poorly, running nearer its bisection bandwidth than its total network bandwidth.

The program will probably have to stall while the receiving processors wait for the partial sums to arrive.

A better program would send the partial sums between neighbouring processors. There would then be no bottleneck and the program could proceed with fewer stalls. One possible scheme would accumulate the sums like this:



Iteration 1 Iteration 2 Iteration 3 Iteration 4

2 Question 2

(a) A cache is a small, fast store which holds duplicate copies of some of the data in the slower main store.

(i) There is a cache between the CPU and main memory.

(ii) In a VM hierarchy, main memory acts as a cache to disk.

(iii) The TLB is used to cache part of the page table.

(b) Spatial locality of reference: If one word is accessed, it's quite likely neighbouring words will be needed soon (sequential instructions, arrays of data etc.) The miss penalty increases, since more words have to be replaced on a cache miss.

(c)

Cache A: 16 1-word blocks. Block address is word address. Index is block address module 16

Cache B: 4 4-word blocks. Block address is $\lfloor \frac{\text{word address}}{4} \rfloor$ Index is block address modulo 4.

	A			B		
Word address	Block address	Index	H/M	Block address	Index	H/M
0	0	0	M	0	0	M
4	4	4	M	1	1	M
8	8	8	M	2	2	M
5	5	5	M	1	1	H
4	4	4	H	1	1	H
0	0	0	H	0	0	H
8	8	8	H	2	2	H
20	20	4	M	5	1	M
6	6	6	M	1	1	M
14	14	14	M	3	3	M
Final contents:	14, 6, 20, 8, 0, 5			12-15, 4-7, 8-11, 0-3		

(d) Let n be miss penalty for cache B. B has 6 misses, A has 7. So $6n < 7 \times 8 \Leftrightarrow n \leq 9$, rounding to nearest integer. Since B has to load 4 words to A's one on every cache miss, one would expect the miss penalty for B to be at least 3 clock cycles more than A ie. at least 11 clock cycles. So it is not really reasonable to expect B to out-perform A for this particular string of memory references.

(e) Cache C: 4 4-word blocks grouped into two sets of two blocks. Block address is $\lfloor \frac{word\ address}{4} \rfloor$. Index is block address modulo 2.

Word add.	Block add.	Set index	H/M
0	0	0	M
4	1	1	M
8	2	0	M
5	1	1	H
4	1	1	H
0	0	0	H
8	2	0	H
20	5	1	M
6	1	1	H
14	3	1	M

Final contents: set 0 0-3 8-11
set 1 12-15 4-7

(f) In general, increased associativity reduces the miss rate but increases the miss penalty (since the block replacement strategy takes time). For caching scenarios where the miss penalty is enormous anyhow, it's of paramount importance to reduce the miss rate as much as possible. Hence VM systems use fully associative placement of pages, since a miss entails a very slow disk access. For main memory caches a lower degree of associativity gives better performance.

3 Question 3

In a true abstract datatype the interface to the rest of the software system is provided by a set of operations (or methods) which may be performed on the type. The structure of the data and the implementation of the operations is hidden, either within the module or the object, from the rest of the system.

Simple modules as provided by most versions of Pascal lack some features which are required to make the types they implement truly abstract, eg:

+ The data must be visible to allow other modules to declare items of this type whereas all that is really required is the size of the object.

+ Standard operators (eg + and -) cannot be used with the type nor can standard procedures like read and write.

+ It is not possible to define constants of the type

Relatively minor additions to the language (eg private types, overloading of operator and procedure names) can resolve these problems. It is hard however without explicit programming to ensure that objects are properly initialised before use or disposed of after use, this is a particular problem where the object has a limited lifetime implied by the context of its

declaration, eg local to a procedure in Pascal. Since the procedures implementing the operations are part of the module rather than of each object, they cannot conveniently contain local data whose value is retained between calls.

True objected oriented langauges like C++ solve both these problems and also add the very powerful concept of inheritance. Inheritance is implemented by allowing one type to be derived from another type (a base class). Methods defined in the base class may be redefined in derived classes. A program manipulating the object need only know about the base class, the method appropriate to the actual derived class is automatically selected by the code produced by the compiler. This provides excellent information hiding since one part of the system need only "know" about just as much of the rest as it actually needs to.

The use of object oriented techniques can influence the software lifecycle as a whole. Firstly it encourages a data centred view of the problem, ie first identify the data objects, then think about the operations to be performed on these. By contrast, more conventional top down successive refinement techniques split the problem into sub-problems and then split these and so on. The concept of inheritance can also be very valuable since it provides a convenient mechanism for specifying components of the system which are similar but not quite identical; it avoids replication of detail which may subsequently be altered in some but not all contexts.

Using the notation developed in lectures (ie OBJECT declarations, INHERITance, CREATE methods, and operator definitions).

```
TYPE impedance = complex;
     frequency = real;

     voltage = OBJECT
        mag : complex;
        freq : frequency;
        PUBLIC OPERATOR + (v2 : voltage) : voltage;
        PUBLIC OPERATOR - (v2 : voltage) : voltage;
        PUBLIC OPERATOR / (z : component) : current
     END;

     current = OBJECT
        mag : complex;
        freq : frequency;
        PUBLIC OPERATOR + (i2 : current) : current;
        PUBLIC OPERATOR - (i2 : current) : current;
        PUBLIC OPERATOR * (z : component) : voltage
     END;

     component = OBJECT
        magnitude : complex;
        PUBLIC FUNCTION val (f : frequency) : impedance
     END;

     resistor = OBJECT
        INHERIT component
```

```
    END;

    capacitor = OBJECT
       INHERIT component
       PUBLIC FUNCTION val (f : frequency) : impedance
    END;

    inductor = OBJECT
       INHERIT component;
       PUBLIC FUNCTION val (f : frequency) : impedance
    END;

voltage.+ (v2 : voltage) : voltage;
BEGIN
  + := mag + v2.mag
END;

voltage.- (v2 : voltage) : voltage;
BEGIN
  - := mag - v2.mag
END;

voltage./ (z : component) : current;
BEGIN
  / := mag / z.val(freq)
END;

Similarly current.+ and current.-

current.* (z : component) : voltage;
BEGIN
  * := mag * z.val(freq)
END;

component.val (f : frequency) : impedance;
BEGIN
  val.real := mag; { for things like resistors }
  val.imag := 0
END

capacitor.val (f : frequency) : impedance;
BEGIN
  val.real := 0;
  val.imag := -1 / (mag * f)
END

inductor.val (f : frequency) : impedance;
```

```
BEGIN
  val.real := 0;
  val.imag := mag * f
END
```

Note:
1) The use of a base class, component, to avoid having to deal with each component type when calculating voltages and currents.
2) The redefinition of operators as implied by the wording of the question.

This design can easily be extended to support the drawing of schematics. component is extended

```
component = OBJECT
   magnitude : complex;
   PUBLIC FUNCTION val (f : frequency) : impedance;
   PROCEDURE draw (...)
END;
```

and the base class version of draw draws the typical rectangular symbol for an impedance with the complex value alongside it. Some positional information might be required as an argument. Particular components can redefine the draw method if a specific symbol is required.

4 Question 4

[First part is bookwork - taken largely from the notes]

Languages like Ada use a client-server model in which the client calls an entry point in the server and thus obtains the service provided by this. Communications are via synchronous duplex channels.

An entry point is the server process end of a channel and is the message equivalent of a procedural interface point in a module. It is thus a useful abstraction for the implemention of Remote Procedure Calls.

An entry point could be declared as:

```
ENTRY entry_name(inmsg : intype) : outtype;
```

The parameter `inmsg` is omitted if there is no input message and the returned type `outtype` is omitted if there is no response message.

The calling process initiates the process by calling `entry_name` exactly like a procedure or function. This reduces the changes required to the client code if a module is instead implemented as a separate, possibly distributed, process.

The server process accepts and responds to the message using

```
ACCEPT entry_name(inmsg) DO statement;
```

`SELECT` provides a mechanism whereby communication through a channel will only be attempted if the rendezvous will complete immediately. It takes the form:

```
SELECT
  WHEN cond1 ACCEPT entry1 DO statement1
  WHEN cond2 ACCEPT entry2 DO statement2
  ...
END
```

In the guard 'WHEN cond', if the boolean condition cond is false the rest of the entry is ignored. A missing guard is equivalent to WHEN true. Clauses for which the guard is true are said to be open.

If any of the ACCEPTs in the open clauses can complete immediately it is used and the following statement(s) executed. If more than one can, one is selected at random.

If there are no open clauses which can complete, the process suspends until there is one. The guards are not reevaluated during this.

The special clause

```
ACCEPTWAIT timeval DO statementT
```

limits the time for which the process will suspend. If after timeval there are still no usable accepts, the process resumes and executes statementT.

If timeval is zero, the process will not be suspended.

Calls from the client to the entry point or ACCEPTs in the server cause the process to suspend if the rendezvous cannot complete, ie the other end isn't yet ready. Similarly processes are suspended by clauses in SELECT statements as described above. Processes are restarted when the condition causing them to be suspended is resolved. Suspension and resumption of processes is handled by the scheduler. In languages like Ada which provide direct support for processes, this scheduling must be provided as part of the language's run time support, ie as part of the overall executable program produced by the compiler.

```
PROCESS receive_proc;

EXPORT

  ENTRY receive_data : user_data;

IMPLEMENT

  VAR packet : pkt;
      seqno : sequence_number;  { next data packet wanted }

  BEGIN
    seqno := 0;
    WHILE true DO ACCEPT link_receive DO BEGIN
      pkt := link_receive;
      IF pkt.type = data THEN BEGIN
        IF pkt.seqno = seqno THEN BEGIN
          ACCEPT receive_data DO
```

```
          receive_data := pkt.payload;
        seqno := (seqno + 1) MOD 1024
      END;
      pkt.type := ack;
      IF pkt.seqno > seqno THEN
        pkt.seqno := seqno; { best we can do }
      link_send(pkt)
    END
    { else some error we can't handle }
  END
END.

PROCESS send_proc;

EXPORT

  ENTRY receive_data : user_data;

IMPLEMENT

  VAR packet : pkt;
      seqno : sequence_number;  { of current data packet }

  BEGIN
    seqno := 0;
    WHILE true DO
      ACCEPT send_data(pkt.payload) DO BEGIN
        pkt.type := data;
        pkt.seqno := seqno;
        sent := false;
        link_send(pkt);
        WHILE NOT sent DO
        SELECT
          ACCEPT link_receive DO BEGIN
            ackpkt := link_receive;
            IF ackpkt.type = ack AND ackpkt.seqno = seqno THEN BEGIN
              sent := true;
              seqno := (seqno + 1) MOD 1024
            END
            ELSE
              link_send(pkt)  {may as well try again}
          END;

          ACCEPTWAIT T DO
            link_send(pkt);  {try again}
        END {select}
      END {sent this data}
```

```
END.
```

Key points are:
1) Servicing of entry points
2) use of select including timeout

The solution given takes advantage of the spec to be as simple as possible but does implement the protocol. In practice the assumption that `receive_data` will rendezvous immediately probably isn't valid but to get round this would require a flow control mechanism to be added to the protocol.

5 Question 5

Bookwork part should contian:

$p[\underline{x}|\omega_j]$ is the probability density function of the features; we calculate this as a parametric density (e.g. Gaussian) or nonparamtertic (e.g. Kernel densities).

P_j are prior probabilities. These are estimated either from some higher level knowledge of the problem (e.g. frequecny of characters in a language for a character recognition problem) or from the given data as $\frac{N_1}{N_1+N_2}$ etc.

$P[\omega_1|\underline{x}]$ is the posterior probability; the conditional error probability is related to this. Hence we can design a minimum error rate classifier, etc.

Class boundary required is

$$P[\omega_1|x] = P[\omega_2|x]$$

$$P_1 = P_2$$

The demominators are the same; 2π cancells

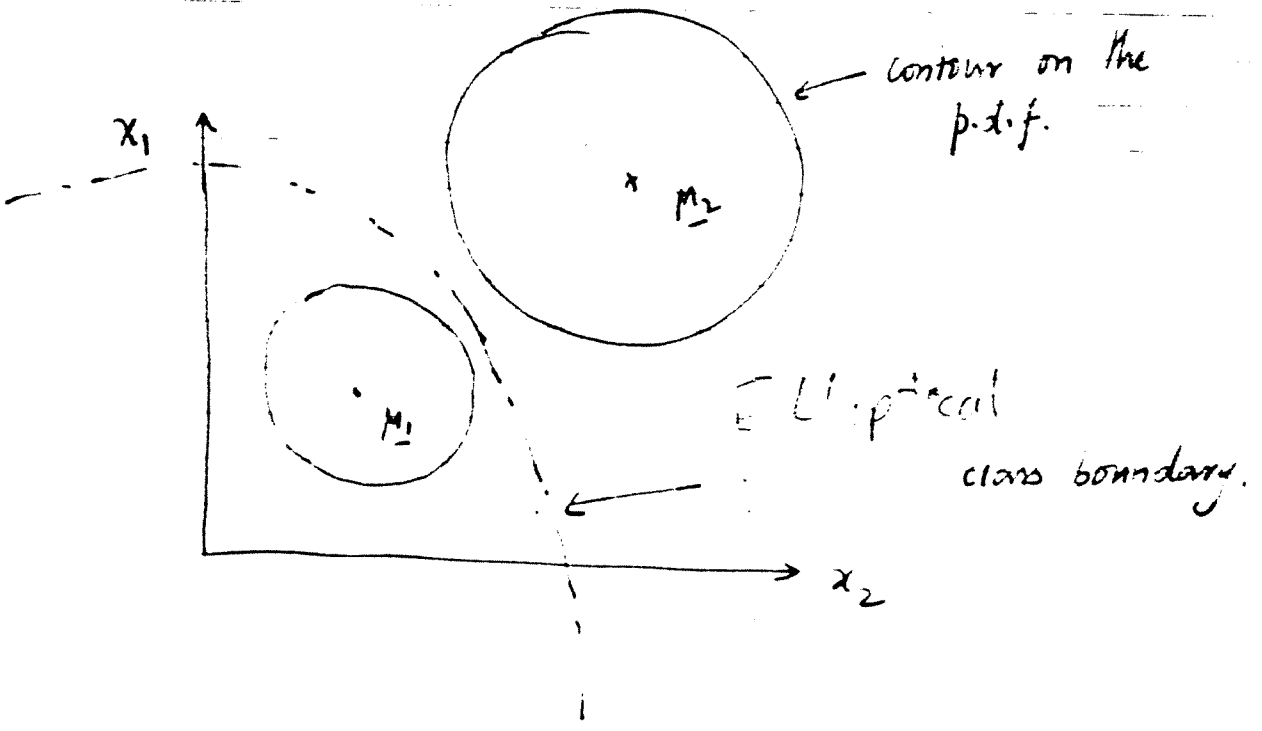
$$\frac{1}{\sigma_1^2}\exp\left(\frac{-1}{2\sigma_1^4}(\underline{x}-\underline{\mu}_1)'(\underline{x}-\underline{\mu}_1)\right) = \frac{1}{\sigma_2^2}\exp\left(\frac{-1}{2\sigma_2^4}(\underline{x}-\underline{\mu}_1)'(\underline{x}-\underline{\mu}_2)\right)$$

$$\frac{1}{2\sigma_1^4}(\underline{x}-\underline{\mu}_1)'(\underline{x}-\underline{\mu}_1) - \frac{-1}{2\sigma_2^4}(\underline{x}-\underline{\mu}_1)'(\underline{x}-\underline{\mu}_2) = \log\left(\frac{\sigma_2}{\sigma_1}\right)^2$$

$$\underline{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad \underline{\mu}_1 = \begin{pmatrix} \mu_{11} \\ \mu_{12} \end{pmatrix} \qquad \underline{\mu}_2 = \begin{pmatrix} \mu_{21} \\ \mu_{22} \end{pmatrix}$$

$$\frac{(x_1-\mu_{11})^2+(x_2-\mu_{12})^2}{2\sigma_1^4} - \frac{(x_1-\mu_{21})^2+(x_2-\mu_{22})^2}{2\sigma_2^4} = \log\left(\frac{\sigma_2}{\sigma_1}\right)^2$$

See figure overleaf.


contour on the p.d.f.
x_1
$\underline{M}_2$
$\underline{M}_1$
Typical class boundary.
x_2

$$\sigma_1 = \sigma_2$$

$$(x_1 - \mu_{11})^2 + (x_2 - \mu_{12})^2 = (x_1 - \mu_{21})^2 + (x_2 - \mu_{22})^2$$

further reduces to a linear class boundary. Also, these two expressions are the same as computing

$$||\underline{x} - \underline{\mu}_1||$$

and

$$||\underline{x} - \underline{\mu}_2||$$

i.e. Bayes' optimal classifier is a *distance to mean classifier.*

6 Question 6

Paramtric vector

$$\underline{a} = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$

Data Class 1:

$$\underline{y}_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \qquad \underline{y}_2 = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

Class 2:

$$\underline{y}_3 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \qquad \underline{y}_4 = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

i) The inequality constraints

$$\underline{a}'\underline{y}_j > 0 \qquad \text{for class 1}$$

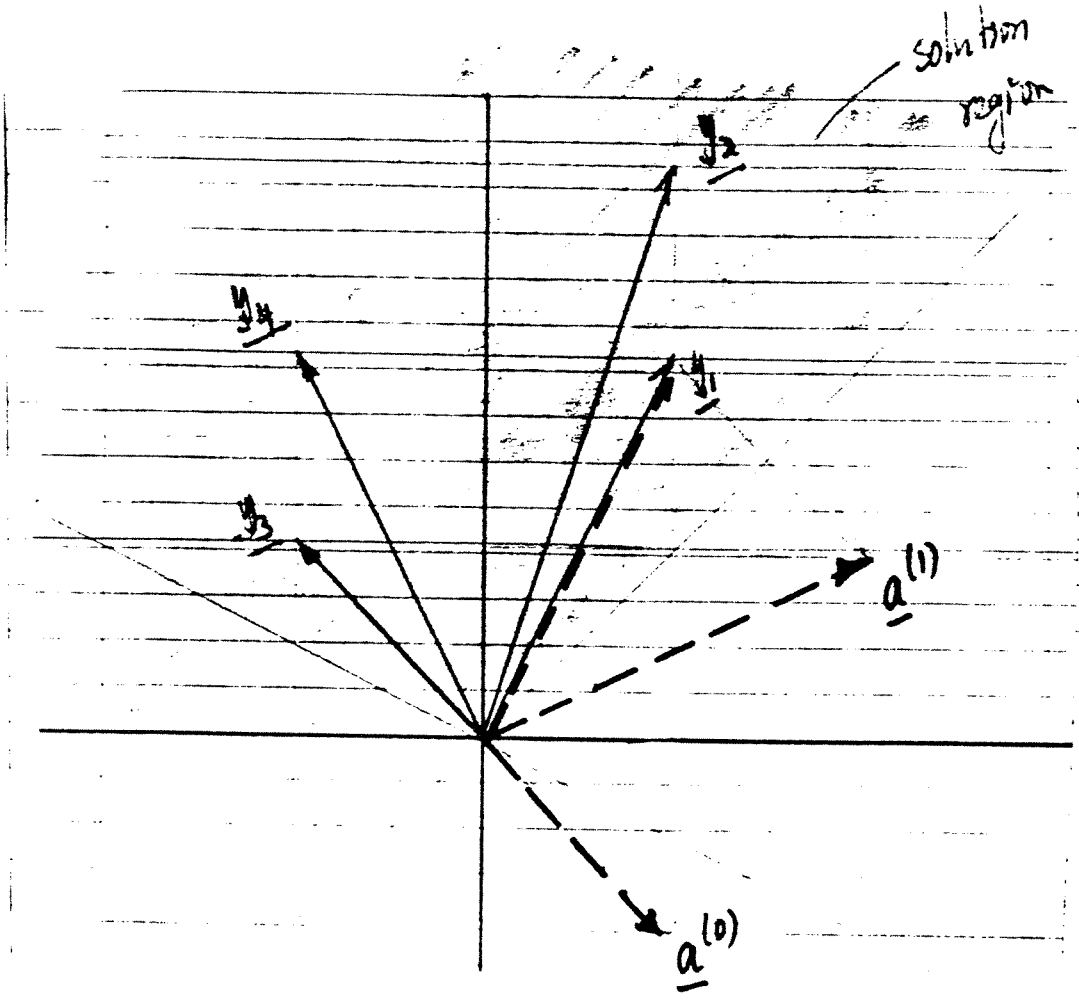
$$\underline{a}'\underline{y}_j < 0 \qquad \text{for class 2}$$

can be written as

$$\underline{a}'\underline{y}_n > 0$$

for all examples by replacing all class 2 examples by $-\underline{y}_j$ [i.e. change the signs]

ii) Dimesionality expanded to dimension 2 by appending a 1 to all x_n; this is a convenient way of dealing with the constant term w_0. See diagram overleaf.


solution region
y_2
y_4
y_1
y_3
$a^{(1)}$
$a^{(0)}$

Percepron performance criterion

$$J = \sum -\underline{a}'\underline{y}_n$$

the summation taken over the set of misclassified examples.

Gradient

$$\nabla J = \sum_{\text{ALL misclassified EXAMPLES}} (-\underline{y})$$

Taking a randomly chosen example a 'noisy' estimate of the true gradient leads to the perceptron correction algorithm.

$$\underline{a}^{(0)} \xrightarrow{\underline{y}_1} \underline{a}^{(1)} \xrightarrow{\underline{y}_2} \underbrace{\underline{a}^{(1)} \xrightarrow{\underline{y}_1} \underline{a}^{(1)}}_{\text{no correction}} \xrightarrow{\underline{y}_3} \underline{a}^{(2)} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Limitations of perceptron
- linear classifier
- algorithm converges only if training data is separable.

7 Question 7

a) i) Depth-first and bredth-first search for *any* root-goal path. They use no information – uninformed search.

ii) Depth first expands deepest node first (i.e. children of first element in queue). For depth d and branching factor b.

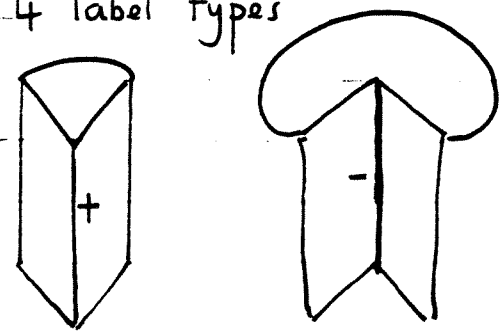
√ complete
X time efficency $O(b^d)$
√ memory efficiency $O(bd)$
X not optimal

iii) Breadth-first expands shallowest node first (i.e. looks at all children and not just the first).

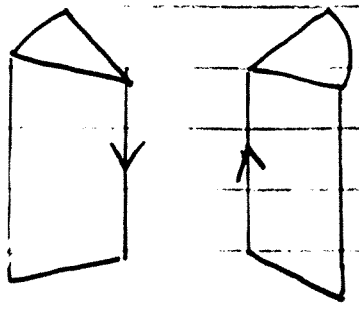
√ complete
X time efficency $O(b^d)$
X memory efficiency $O(b^d)$
X not optimal

iv) For lare b, time and memory requirements dominate and make algorithms impractical. Use "heuristics" to define an *evaluation function* to guide search and to decide which node to expand. (i.e. knowledge in queuing function)., e.g. hill-climbing, beam search, best-first search.

(Q7b). i. 4 label types



convex concave



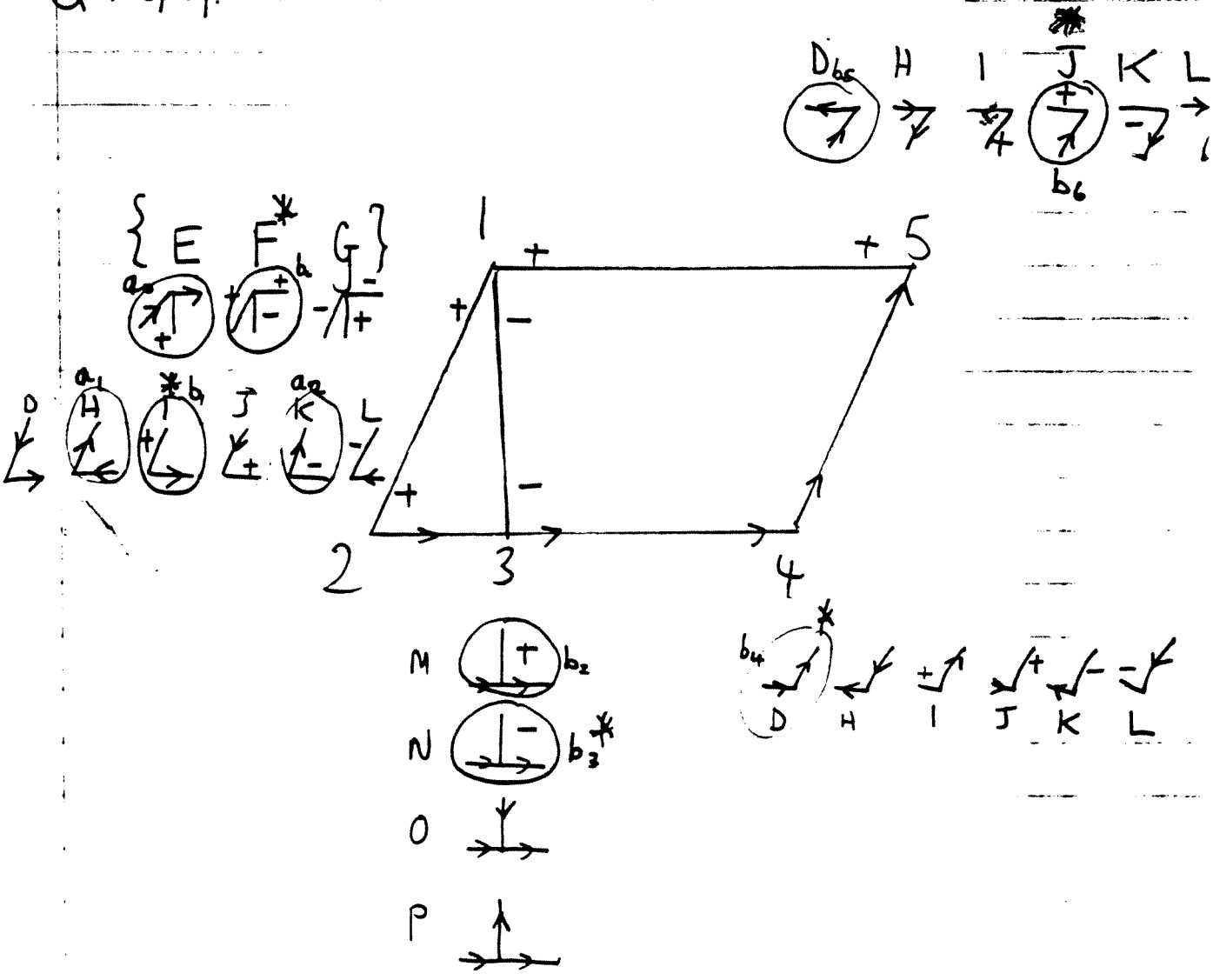
occluding (2 types

4 junction (trihedral) types

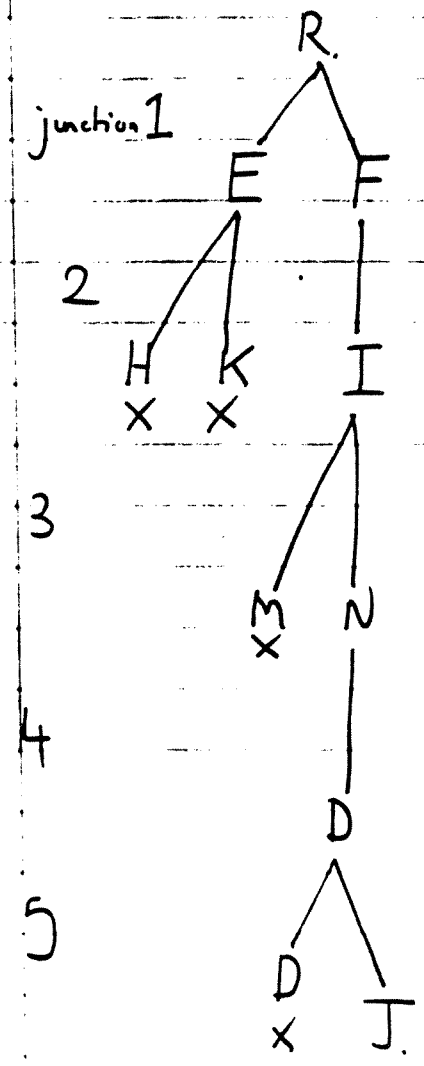
	possible	actual due to visibility
T	64	4
Y	64	3
L	16	6
W or arrow ↓	64	3
	208	→ 16

Only 16 are <u>visible</u> (eg. octant analysis).

Q7 b) ii).



Q7 b. Search tree explored



Compare to tree of depth $d = 5$, average branching factor $= 5$ nodes here plane $O(5^5)$. In above only 10 nodes explored.

8 Question 8

a) An inference rule is sound if the conclusion is true when the premises are true.

The resolution inference rule:

"If (A∨B) and (¬B∨C) then A∨C follows"

premises	A ∨ B
	¬B ∨ C
conclusion	A ∨ C

				premises		conclusions
	A	B	C	A ∨ B	¬ B ∨ C	A ∨ C
1	F	F	F	F	T	F
2	F	F	T	F	T	T
3	F	T	F	T	F	F
4	F	T	T	T	T	T
5	T	F	F	T	T	T
6	T	F	T	T	T	T
7	T	T	F	T	F	T
8	T	T	T	T	T	T

Both premises are true in rows 4, 5, 6, 8. Conclusion is also true and hence rule is a sound rule of inference.

b) In first-order predicate calculus:

i)

$$\forall x \forall y \forall z\, [\text{ son } (x, y) \wedge [\text{ son } (y, z) \vee \text{ daughter } (y, z)] \rightarrow \text{ grandson } (x, z)]$$

Convert to C.N.F. (clause form) and add to 6 existing clauses.

$$\neg \text{son}(x1, y1) \vee \neg \text{son}(y1, z1) \vee \text{grandson}(x1, z1) \qquad (7)$$

$$\neg \text{son}(x2, y2) \vee \neg \text{daughter}(y2, z2) \vee \text{grandson}(x2, z2) \qquad (8)$$

ii) We must prove the following theorem:

$$\exists u \text{ grandson } (u, \text{ George})?$$

Proof by resolution proceeds by

(1) clauses in CNF

(2) add negation of theorem to set of clauses

(3) use resolution/unification of variables to find a contradiction in clauses

(4) if conflict assume conclusion true.

therefore Add $\neg\exists u$ grandson (u, George)

$\equiv \forall u \neg$ grandson (u, George)

$\equiv \neg$grandson ($x3$, George)

1. Son (John, Elizabeth)
2. daughter (Anne, Elizabeth)
3. son (William, George)
4. son (Edward, George)
5. daughter (Elizabeth, George)
6. daughter (Mary, William)
7. $\neg$ son ($x1, y1$) v$\neg$ son ($y1, z1$) v grandson ($x1, z1$)
8. $\neg$ son ($x2, y2$) v $\neg$ daughter ($y2, z2$) v grandson ($x2, z2$)
9. $\neg$ grandson ($x3$, George).

Resolve (9) and (8), unitfy $z2$ \ George. $x3$\ x2.

↓

$\neg$ son ($x2, y2$) v$\neg$ daughter ($y2$, George) (10)

Resolve (10) and (5), unify $y2$ \ Elizabeth

↓

$\neg$ son ($x2$, Elizabeth) (11)

↓

Resolve (11) and (1), unify $x2$ \ John.

↓

Nil

grandson ($x3$, George) is true for $x3$ = John

therefore : grandson (John, George)