Solutions to E6 computer architecture questions - Question 1

Pipelining

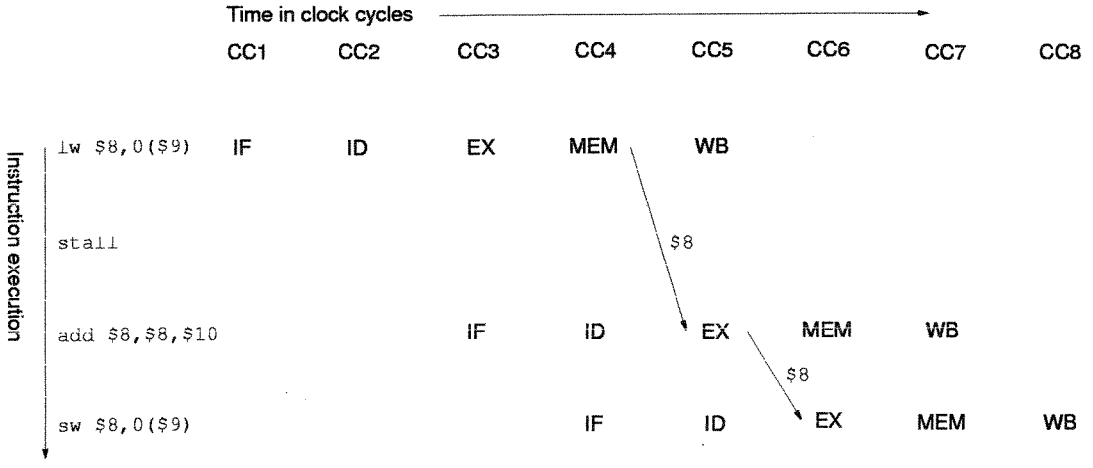
(a) [Bookwork] The term "hazard" is used to describe dependencies between instructions which can disrupt the operation of a pipelined datapath. "Data hazards" occur when an instruction requires data before a previous instruction has written it to the register file (or memory). "Branch hazards" occur when the address of the next instruction is required (for instruction fetching) before an earlier conditional branch instruction has been evaluated.

(b) There are data hazards between the `lw` and `add` instructions, and between the `add` and `sw` instructions. If there is no data forwarding, the pipeline must stall for three clock cycles between these instructions, since data is read from the register file at pipe stage 2 but written at pipe stage 5.

There is a branch hazard associated with the `bne` instruction. If there is no pipeline flushing, the pipeline must stall for three clock cycles after the `bne` instruction is fetched, since the address of the next instruction is not known until pipe stage 4.

There are therefore 9 stalls in total, and 5 instructions, so the code segment takes $14n$ clock cycles to execute.

(c) Data forwarding can be used to pass values from the pipe registers to the ALU inputs. Only four stalls are required with data forwarding: one after the `lw` instruction, and three after the `bne` instruction. The code now takes $9n$ clock cycles to execute.



(d) The unrolled code is shown below (comments are included for explanation — candidates would not be expected to comment every line of code).

```
Loop:   lw $8,0($9)          # $8 loaded with data at address $9+0
        add $8,$8,$10        # $8 loaded with $8+$10
        sw $8,0($9)          # $8 stored at address $9+0
        lw $8,4($9)          # $8 loaded with data at address $9+4
        add $8,$8,$10        # $8 loaded with $8+$10
        sw $8,4($9)          # $8 stored at address $9+4
        addi $9,$9,8         # $9 loaded with $9+8
        bne $9,$11,Loop      # Jump back 7 instructions if $9≠$11
```

The pipeline will stall once after each `lw` instruction and three times after the `bne` instruction. There are therefore 13 clock cycles for each of the $n/2$ loops, so the code segment takes $13n/2$ clock cycles to execute.

(e) The optimized code is shown below (again, comments are included for explanation).

```
Loop:   lw $8,0($9)          # $8 loaded with data at address $9+0
        addi $9,$9,4         # $9 loaded with $9+4
        add $8,$8,$10        # $8 loaded with $8+$10
        sw $8,-4($9)         # $8 stored at address $9-4
        bne $9,$11,Loop      # Jump back 4 instructions if $9≠$11
```

The data hazard between the `lw` and `add` instructions has been removed by scheduling the `addi` instruction between them. The pipeline will stall three times after the `bne` instruction, so the code segment takes $8n$ clock cycles to execute.

(f) Unrolling loops appears to give faster execution but produces longer, less compact machine code. We have assumed that the CPU does not have to stall for cache misses as it executes the code segments. The code in (d) is less "instruction cache friendly" than the code in (e), so might execute slower on a machine with a small instruction cache.

(g) With delayed branches, m instructions following the branch are executed, whether the branch is taken or not. This reduces the number of hardware

stalls by m. The compiler must find appropriate instructions to schedule in the delay slots (using `nop`'s as a last resort). In practice, all machines with delayed branch employ a single instruction delay ($m = 1$). With a single instruction delayed branch, the code could be optimized as follows:

```
Loop:   lw $8,0($9)          # $8 loaded with data at address $9+0
        addi $9,$9,4         # $9 loaded with $9+4
        add $8,$8,$10        # $8 loaded with $8+$10
        bne $9,$11,Loop      # Delayed branch
        sw $8,-4($9)         # $8 stored at address $9-4
```

This reduces the number of stalls to two, so the code executes in $7n$ clock cycles.

Solutions to E6 computer architecture questions - Question 2

Carry lookahead adders and processor-memory buses

(a) [Bookwork] Carry lookahead can be used to determine the carry inputs to each full adder without using ripple carry. For each bit i of the adder, we define two signals, generate gi and propagate pi. Bit i generates a carry if the two bits it's adding are both 1, and propagates a carry if either of the two bits it's adding are 1:

$$\mathsf{gi} = \mathsf{ai.bi} \qquad\qquad \mathsf{pi} = \mathsf{ai} + \mathsf{bi}$$

c1, the carry into bit 1, will be 1 if either bit 0 generates a carry or c0 is 1 and bit 0 propagates a carry:

$$\mathsf{c1} = \mathsf{g0} + \mathsf{p0.c0}$$

Likewise for c2 and c3:

$$\mathsf{c2} = \mathsf{g1} + \mathsf{p1.g0} + \mathsf{p1.p0.c0}$$
$$\mathsf{c3} = \mathsf{g2} + \mathsf{p2.g1} + \mathsf{p2.p1.g0} + \mathsf{p2.p1.p0.c0}$$

It takes time T to evaluate the gi and pi signals, $2T$ to evaluate the sum-of-products expressions for the ci signals, and a further $2T$ to calculate the sums, so the 4-bit add takes $5T$.

For the alternative design, we define the term P0, which is true if the first 2-bit adder propagates a carry. This will occur if each of its constituent bits propagate a carry:

$$\mathsf{P0} = \mathsf{p0.p1}$$

We also define the term G0, which is true if the first 2-bit adder generates a carry. Either its MSB generates a carry, or its LSB generates a carry and its MSB propagates it:

$$\mathsf{G0} = \mathsf{g1} + \mathsf{p1.g0}$$

The carry-in signal to the second 2-bit adder is therefore

$$C1 = G0 + P0.c0$$

The 4-bit add proceeds by evaluating pi and gi, then P0 and G0, then C1, then ci and finally the sums. The total time is therefore $T + 2T + 2T + 2T = 9T$.

Even though the single-level adder appears to be faster, it requires logic gates with a fan-in of 4. Compared with 2-input gates, these may be relatively slow and difficult to implement. On the other hand, the two-level adder requires only 2-input gates. The relative propagation delays through the 2- and 4-input gates will depend on the particular implementation technology, but the two-level adder *will* be faster if the 4-input gates are especially slow.

(b) [Bookwork] Synchronous buses can support fast communication protocols (no handshaking is required). However, they have two major disadvantages. First, every device on the bus must run at the same clock rate. Second, because of clock-skew problems, synchronous buses cannot be long if they are fast. Processor-memory buses are often synchronous because the devices communicating are close, small in number and prepared to run at high clock rates. I/O buses are often asynchronous because they have to be long and accommodate a wide variety of devices running at different speeds.

(i) Memories invariably utilize error correcting codes to improve reliability. These codes have to be reset after memory writes, which consequently take longer than memory reads.

(ii) When a cache miss requires a write-back, the cache block is written to memory, then the new block is read from memory, taking a total of 26 clock cycles. Cache misses which do not require write-back take 12 clock cycles. The number of cycles per instruction spent handling cache misses is therefore $(0.4 \times 26 + 0.6 \times 12) \times 0.05 = 0.88$ cycles.

(iii) The important point is that it does *not* take twice as long to read or write a 16-word block than it does an 8-word block. A significant part of the memory transfer time is overhead. A conservative estimate would be that it takes one clock cycle to transfer each word between processor and memory, with the remaining cycles representing a constant overhead. It would therefore take $14 + 8 = 22$ cycles to write a 16-word block and $12 + 8 = 20$ cycles to read a 16-word block. The number of cycles per instruction spent handling cache misses is now $(0.4 \times 42 + 0.6 \times 20) \times 0.025 = 0.72$ cycles.

Solutions to - Question 3

The main problems with the code shown in Fig. 3 are:

(1) The lack of proper type information associated with the returned data none of which is logically a single byte.

(2) The need to modify various parts of the module (the type sensortype and the case statement) each time a new type of sensor is added to the IO network

(3) The need for read_sensor to return a boolean value to indicate success /failure, a value which may have to be passed back up through several levels of function call to a much higher level coordinating procedure.

(4) The large number of goto statements (albeit all to the same place) do not clarify the semantics.

(1)(2), can be improved by the use of exceptions. These are similar to the Pascal non-local goto except that the point in the program to which control is returned is determined dynamically at run time rather than statically at compile time. Exceptions require

(i) storage in which to record what should be done when the exception is raised, e.g.
VAR thisexception: EXCEPTION;

(ii) a means to specific where in the program to return to and what to do, e.g.
WHEN thisexception DO < statement >;
and then continue exeution with the statement following

(iii) a means to trigger the exception, e.g.
RAISE thisexception;

Note that exception actions are automatically stacked by the compiler so that when a procedure that has executed a WHEN returns, the previous action for the exception is reinstated.

(3)(4) can be improved by the use of object oriented features, specifically the inclusion of methods as part of an object (so that the method belongs to the individual

object) and inheritance whereby base class can be defined and variants of this defined derived from it. In the case of the code in Fig. 3 , the base type would represent a sensor and the derived types the different specific type of sensor. All that the read_sensor functionality needs to know about is base type, leaving it to the methods in the individual derived types to handle the sensor specific details. Derived types can be added (in separate modules if required) at any stage without affecting the code already written.

Note that in the definition of a method, a field name given without the object name refers to the object whose method is being called.

[the above implementations of exceptions and objects follow the syntax and semantics described in lectures]

Putting this into practice:

```
{Definitions imported from byteio module as before but with addition of
VAR byteio_error:EXCEPTION;
and send_byte and read_byte become procedures as the error handling is
done by the exception
}

TYPE sensor = OBJECT
                PUBLIC id:byte;
                PUBLIC onerror:EXCEPTION;
                PUBLIC PROCEDURE read;
                PROCEDURE request;
                PROCEDURE dodata;
              END;
     switch_sensor = OBJECT
                INHERIT sensor;
                PUBLIC val:boolean;
                PROCEDURE dodata;
              END;
     xy_sensor = OBJECT
                INHERIT sensor;
                PUBLIC x,y:position;
                PROCEDURE dodata;
              END;
     temperature_sensor = OBJECT
                INHERIT sensor;
                PUBLIC val:temperature;
                PROCEDURE dodata;
              END;
```

```
VAR sensor_err:EXCEPTION; {for errors at this level}

PROCEDURE sensor.read;
BEGIN
  WHEN err DO BEGIN
    reset_byte_io;
    RAISE onerror
  END;
  byteio_error:= sensor_err;
  request; {same for all current sensors}
  processreply; {different for each sensor type}
END;

PROCEDURE sensor.request;
VAR tmp:byte;
BEGIN
  send_byte(OP_READ);
  send_byte(s.id);
  read_byte(tmp);
  IF tmp <> OP_OK THEN RAISE sensor_err;
  read_byte (tmp);
  IF tmp <> s.id THEN RAISE sensor_err;
END;

PROCEDURE sensor.dodata;
BEGIN
  RAISE sensor_err; {or alternatively do nothing}
END;

PROCEDURE switch_sensor.dodata;
VAR tmp:byte;
BEGIN
  read_byte (tmp);
  val:= tmp <> 0
END;

PROCEDURE xy_sensor.dodata;
VAR tmp:byte;
BEGIN
  read_byte(tmp); x:=tmp;
  read_byte(tmp); x:=x+(tmp*256);
  read_byte(tmp); y:=tmp;
  read_byte(tmp); y:=y+(tmp*256);
END;
```

```
PROCEDURE temperature_sensor.dodata;
VAR tmp:byte;
BEGIN
  read_byte(tmp);val:=tmp;
  read_byte(tmp);val:=val+(tmp*256);
END;
```

This code as it stands will not run properly in a multi-process environment because there is nothing to stop several processes trying to address different sensors simultaneously thus breaking the protocol (send 2 byte request, receive 2 byte header plus data) defined above. We need to ensure that only one such request+response operation can take place at a time. A simple solution would be to use semaphores with a binary semaphore (ionet_access) to protect access to the IO network. This would be used in sensor.read (and any other procedures providing access to the IO network.

```
PROCEDURE sensor.read;
BEGIN
  P(ionet_access);
  WHEN err DO BEGIN
    reset_byte_io;
    RAISE onerror
  END;
  byteio_error:= sensor_err;
  request; {same for all current sensors}
  processreply; {different for each sensor type}
  V(ionet_access)
END;
```

Solutions to - Question 4

(a) The software lifecycle provides a framework for describing the progression of piece of software from an initial concept to completed project. This can be split into the following stages:

(1) Requirements Definition: Initial specification of what the system is intended to do (rather than how it is to do it). This is produced by the software engineering team and the customer in discussion. Since the (generally non-specialist) customer is involved, the notations used tend to be biased towards easy readability rather than formal precision.

The requirements definition can be split down into the main functional requirements plus the constraints placed on the system (or non-functional requirements) and a set of qualitative guidelines (or Goals). In time terms, a typical sequence is : feasibility study; outline definition; preliminary design study; full definition.

(2) Design: At this stage the software engineering team (now without further direct input from the customer) specify how the system is to perform its task. Again the process is one of gradual refinement, typically starting with an overall structure specification, then producing specifications for each component of this and finally the detailed component design. The overall decomposition can either be procedural (do A in terms of B and C, do B in terms of D, E and F, etc) or object oriented (in which the primary decomposition is based on the data in the system and then the operations or methods for each datatype are defined).

The notations used can now be more specialised ranging from: graphical (sturcture diagrams and dataflow diagrams); to informal mixtures of free-form description and programming language (program description languages); to very formal interface specifications and operational specifications which can be used in formal program verification.

(3) Implementation: This stage involves the production of the actual program. It is done by the software engineering team and uses one or more programming languages. Typically members of the team will be responsible for different components (idenified and with interfaces defined at the design stage). As well

as being split up on the basis of procedures and/or objects, a large system may also involve the use of several interacting processes. Typically, the components will, once written, then be individually tested before being assembled into a complete system ready for system testing.

(4) System Testing: The complete system needs to be tested before being put into operation. The nature and duration of this will depend on the amount of testing done in earlier stages (e.g. formal design and program verification, component testing) but will at least in part require empirical testing involving the customer and its use in a real environment.

(5) Operation and Maintenance: Once testing is complete, the system can be used and should (but rarely does!) operate perfectly! Even if there are no flaws in the program itself, it is likely that it will affect the environment in which it runs to such an extent that the customer's long-term requirements change and some modification is required.

All the above stages interact and there is feedback from the later stages into earlier decisions. The aim is to minimise the amount of feedback requried because major changes to the requirements late in the lifecycle will be costly. In order to minimise this risk, prototype systems are often produced at the requirement definition and initial design stages to allow the customer to gain an impression of the eventual system and to allow its interaction with pre-existing systems and components to be tested.

(b) [see attached]

(c)

(i) A convenient way to use a dataflow diagram for procedural design is first to transform it into a Structure Chart. This represents the procedural hierarchy of the system graphically with rectangles representing the procedureal units. The lines joining them denote the procedural relationship and alongside these is shown the flow of data.

[see attached]

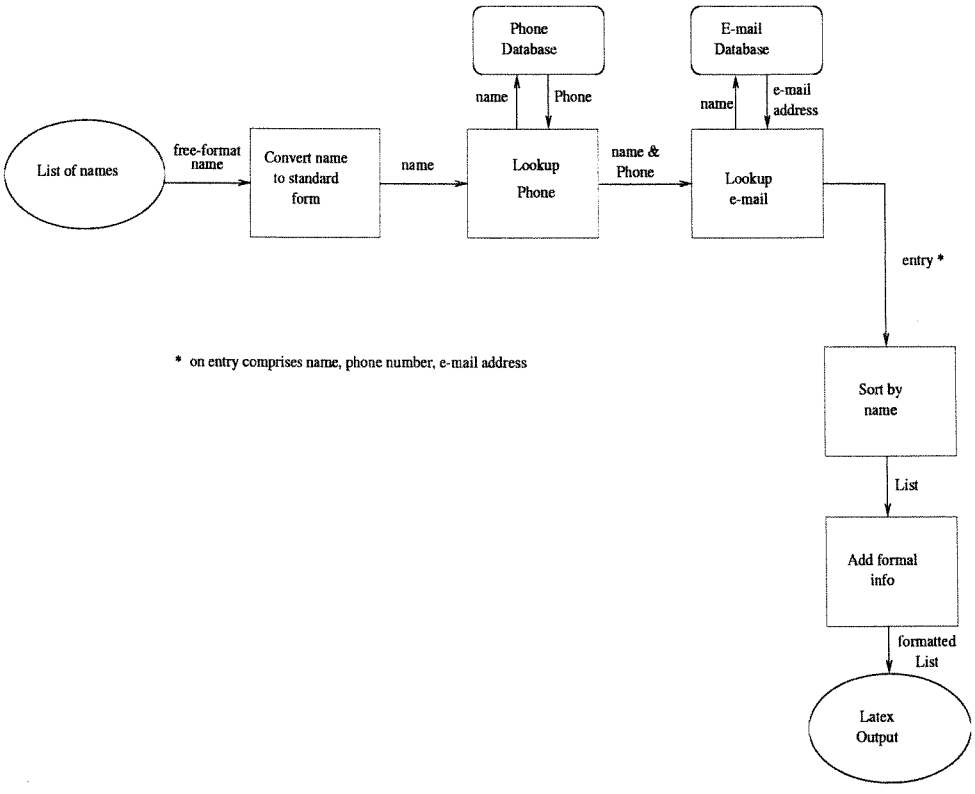
A Structural Chart is derived from a dataflow diagram by first identifying the central transformation in the system in which "input" is converted into "output". Here this is the database lookup.

[see attached]

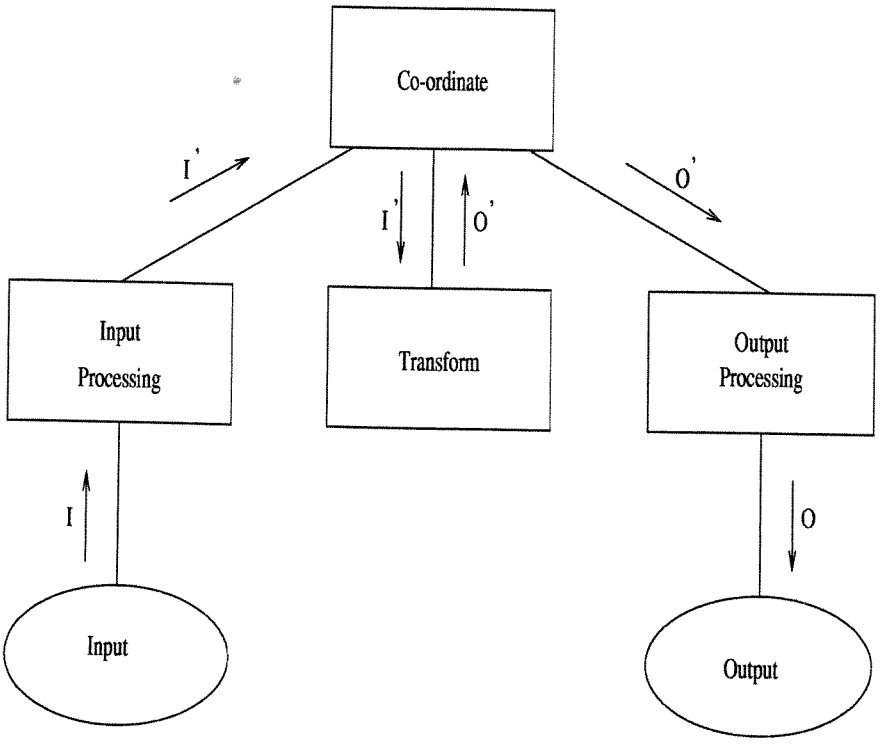
Each part of the chart (input region, transform, output region) can then be successively refined using a similar structure (preprocess, main transform, post-process).

(ii) The dataflow diagram by its nature identifies data items and the transformations performed on these. Data items become objects and these objects include as their methods the transformations performed on them.

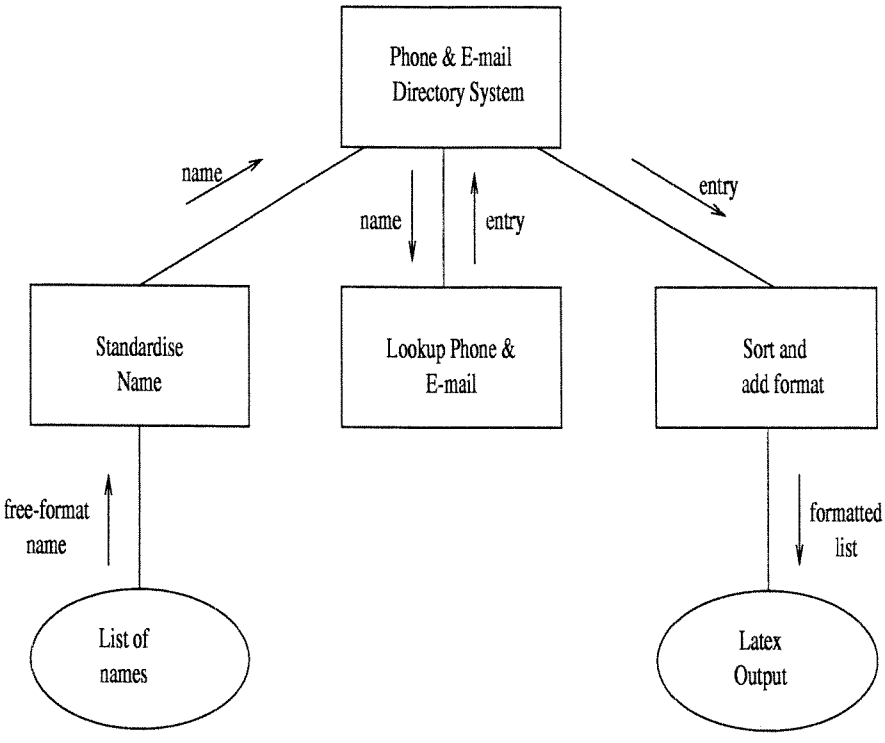
b) Dataflow Diagram



C) Basic Form of Structure Chart



C) Structure Chart (Top Level) For Problem in (b)



SOLUTION - Question 5

Bookwork answer to *Bayes'* minimum error rate classifier. Should show *Bayes'* rule to compute posterior probabilities of class membership, e.g. *Bayes'* rule is

$$P(X|M) = \frac{P(M|X)P(X)}{P(M)}$$

$P(X)$ is the *prior* probability of event X
$P(M|X)$ is the *likelihood* (conditional probability) of the model M given event X
$P(M)$ is the *evidence* (prior) that model M is correct
$P(X|M)$ is the *posterior* (conditional) probability of event given model M

(4 marks)

W_A : Adverse

W_B : Benign

$P(X|W_A) \sim N(1,1); P(X|W_B) \sim N(0,1)$

$N(\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$

Posterior probability of adverse outcome

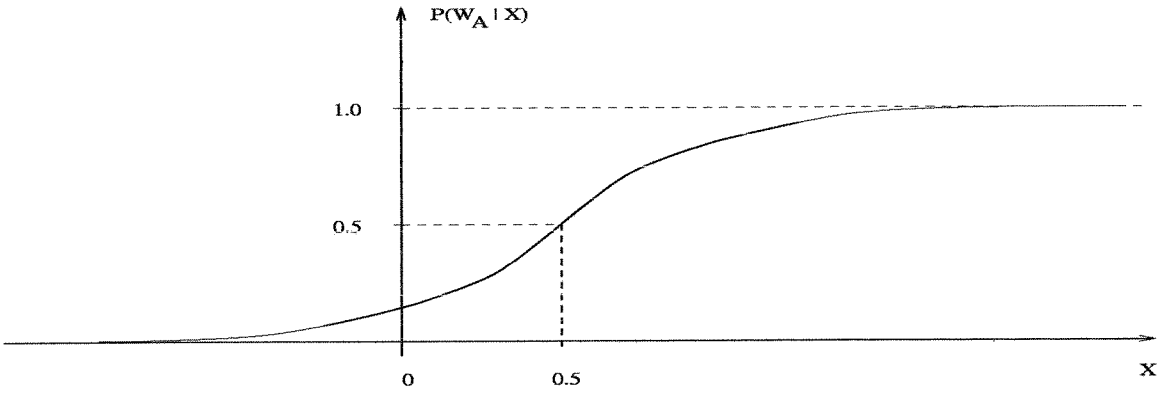
$$P(W_A|X) = \frac{P[W_A].P(X|W_A)}{P[W_A].P(X|W_A) + P[W_B].P(X|W_B)}$$

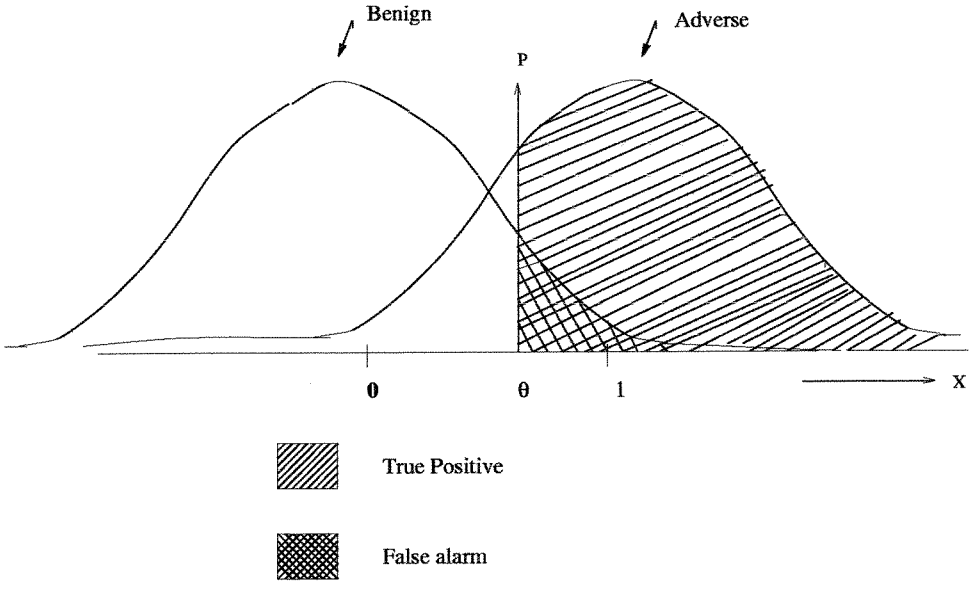
assume the prior probabilities are equal substitute the Gaussian densities

$$\begin{aligned} P(W_A|X) &= \frac{1}{1 + \exp\left\{-\frac{(x-\mu_B)^2}{2\sigma^2} + \frac{(x-\mu_A)^2}{2\sigma^2}\right\}} \\ &= \frac{1}{1 + \exp\left\{\frac{(\mu_B-\mu_A)}{\sigma^2}\, x + \frac{\mu_A^2-\mu_B^2}{2\sigma^2}\right\}} \\ \text{of the form} &= \frac{1}{1 + \exp\{w_0 + w_1 x\}} \\ \text{in fact} & \quad \frac{1}{1+\exp\{\frac{1}{2} - x\}} \end{aligned}$$

X^2 terms cancel out

(5 marks)





(5 marks)

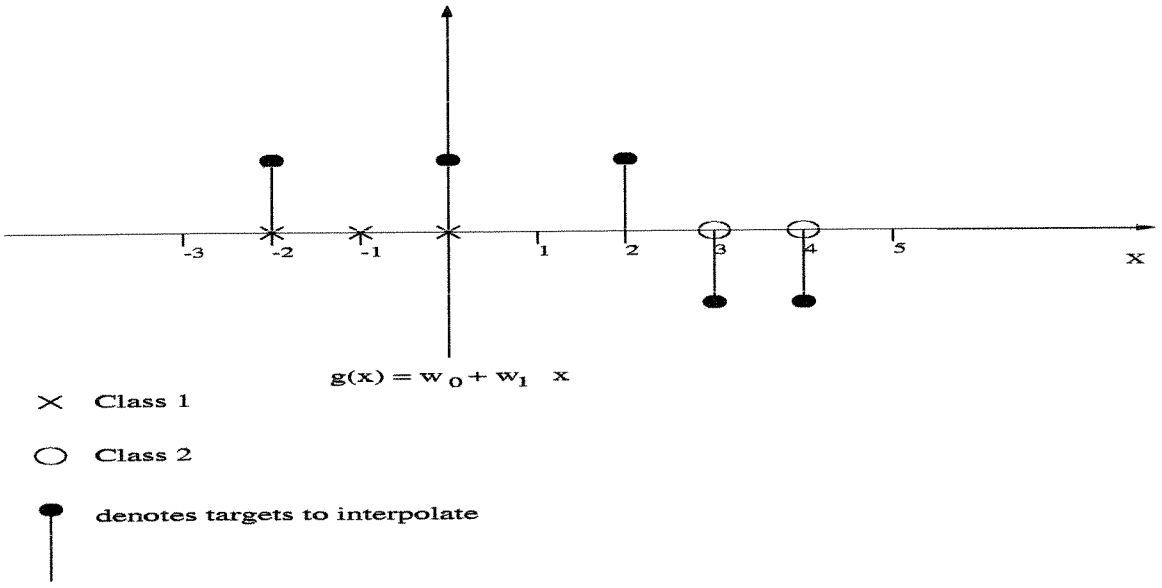
Cumulative probabilities tabulated in maths data book.

Bayes optimal value for θ is 0.5 due to symmetry because we have assumed equal prior probabilities.

θ	F.P.	T.P.
0.2	1 - 0.5793	0.7881
	$\Longrightarrow$ 42.1%	$\Longrightarrow$ 78.8%
0.5	1 - 0.6915	0.6915
	$\Longrightarrow$ 30.8%	$\Longrightarrow$ 69.2%

(6 marks)

SOLUTION - Question 6



$$\text{data matrix y} = \begin{pmatrix} -2 & 1 \\ 0 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{pmatrix}$$

$$\textit{weights } \underline{a} = \begin{pmatrix} w_1 \\ w_0 \end{pmatrix} \qquad \textit{target } \underline{t} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}$$

least squares solution given by the pseudo inverse

$$a = (Y' Y)^{-1} Y' \underline{t}$$

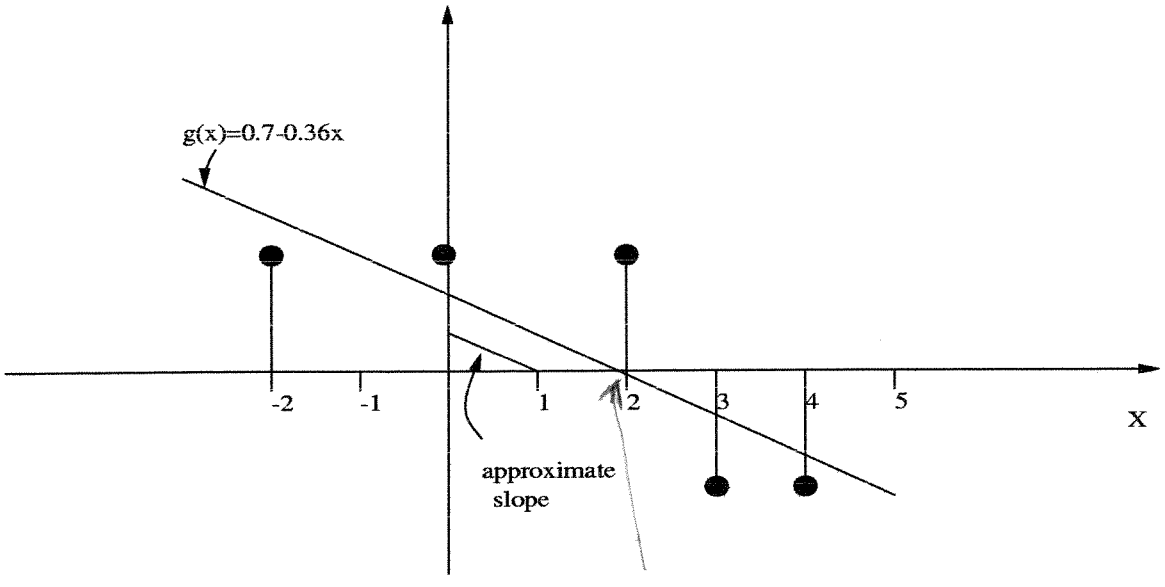
$$= \begin{pmatrix} 33 & 7 \\ 7 & 5 \end{pmatrix}^{-1} \begin{pmatrix} -2 & 0 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}$$

$$\underline{a} = \frac{1}{116} \begin{pmatrix} 5 & -7 \\ -7 & 33 \end{pmatrix} \begin{pmatrix} -7 \\ 1 \end{pmatrix}$$

$$= \frac{1}{116} \begin{pmatrix} -42 \\ 82 \end{pmatrix}$$

$$w_1 = -0.36$$

$$w_0 = 0.7$$



intercept is at 1.95

(10 marks)

Total squared error

$$E = \sum_{n=1}^{N} \{t_n - (w_0 + w_1 x_n)\}^2$$

gradient

$$\frac{\partial E}{\partial w_0} = \sum_{n=1}^{N} 2\{t_n - (w_0 + w_1\ x_n)\}(-1)$$
$$\frac{\partial E}{\partial w_1} = \sum_{n=1}^{N} 2\{t_n - (w_0 + w_1\ x_n)\}(-x_n)$$

Start from some random guess of $\underline{a} = \begin{pmatrix} w_1 \\ w_0 \end{pmatrix}$ we apply

$$\underline{a}^{(n+1)} = \underline{a}^{(n)} - \eta\ \underline{\nabla} E$$

(6 marks)

We see that the least squares solution doesn't produce a suitable classifier. Example at $x = 2$ is classified wrongly.

This is because examples far away from the class boundary in this case some value of x between 2 and 3.

Perception learning rule will find a solution that classifies this problem correctly because it is an error correction procedure. *(4 marks)*

Solutions to - Question 7

(a)(i) The term combinatorial explosion refers to the situation where the time required to search through a space of possibilities grows very rapidly (often exponentially) as the search-space (tree) increases. In practice, this phenomena occurs in all but the most trivial AI problems and limits the application of exhaustive search techniques.

(ii) A blind search is one that proceeds simply by searching the tree until a solution path is found. No knowledge of the problem domain is applied. A heuristically informed search is one that does possess knowledge, in the form of 'heuristics', of the problem domain. By using this prior knowledge, it can search more efficiently, avoiding obvious fruitless paths and therefore, sometimes dramatically, may alleviate the explosive nature of the effort required in blind search.

(iii) For a tree of depth n, there are b^n nodes so the total number is : $\sum_{n=0}^{d} b^n = \frac{b^{d+1}-1}{b-1}$

(b)(i) See attached diagram.

(ii) a) B O C N R S L S R C N L S O C N R L R C L O N

b) B O S C S O R N R R C C L S L C L N R O N N L

(other sequences may be correct depending on how the candidate has drawn their tree)

(iii) The answer is simply N.

(c) The question allows some room for individual interpretation but the key point is that $f*(n)$ allows the incorporation of heuristic knowledge about the problem to be incorporated into the evaluation function. This means that the search need not rely exclusively upon the existence of a sophisticated mathematical algorithm but can embody 'experience' or 'good guesses' about the situation.

(i) The system will find the optimal path.

(ii) The system will find a successful path. The degree of apparent randomness may however increase if h* is much less than the true distance.

(iii) The A* algorithm will not function properly. Again, the search behaviour may appear random.

Give extra credit if the student mentions the 'admissibility' condition.

Again , some latitude for individual student insight but the main point is that the 'cost' of calculating (or executing) $f*(n)$ must not be too high in terms of computational resource or time. Also, the heuristic component of the equation must, for the function to be useful, actually be a fair estimate of what the real situation actually is. The design of a good evaluation function can make a very significant difference to the performance of the search.

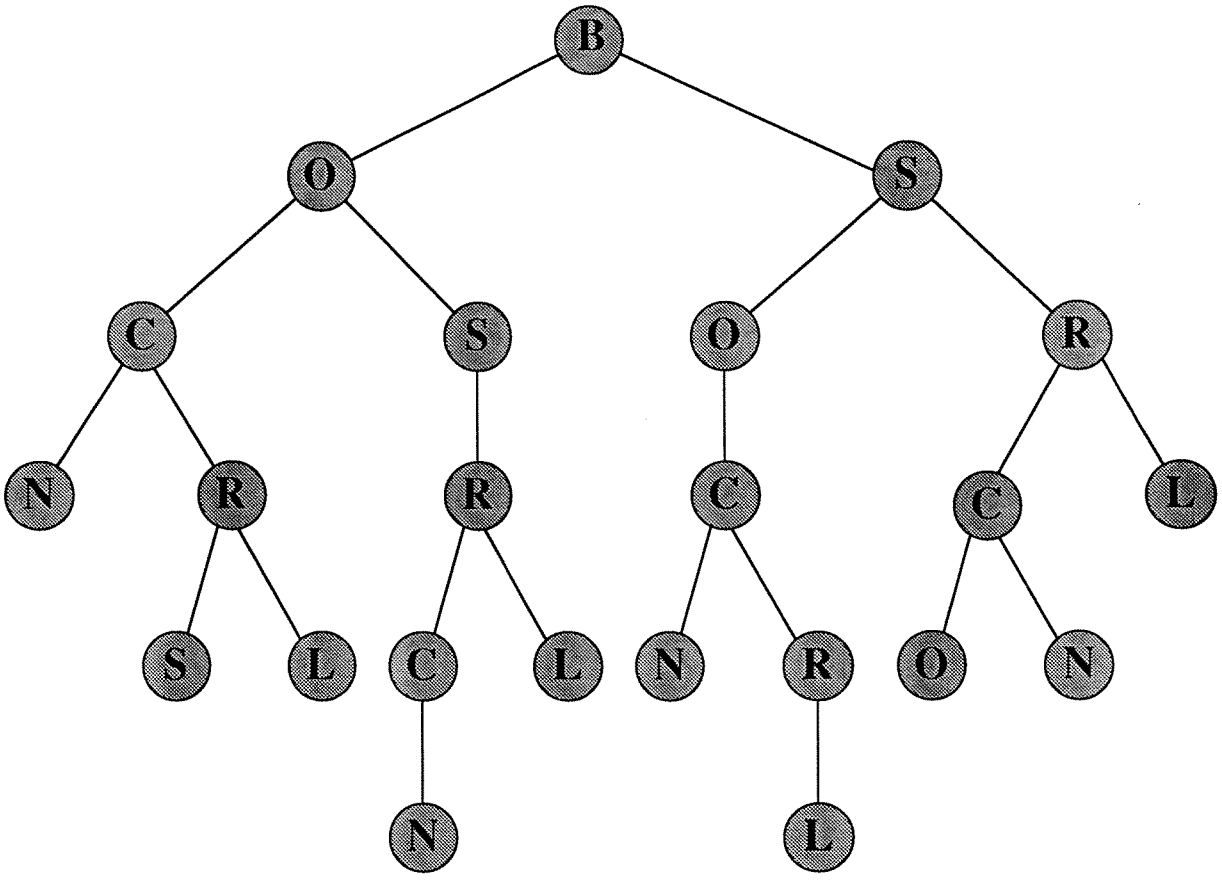


Diagram for answer to Q7

Solutions to - Question 8

(a) The student's diagram should contain the components of: knowledge base, inference engine and user interface. Sensible looking additional items are permissible.

Knowledge base: Contains the knowledge (possibly in the form of rules), facts and other information pertinent to the problem area.

Inference engine: Contains the mechanism wherby the problem solving knowledge (e.g. rules) contained in the knowledge base is applied to the problem in hand. Typically, this mechanism would employ 'forward-' and /or 'backward-' chaining.

User interface: Translates the user's queries into machine readable instructions and vice-versa. May assist in explaining how the computer has reached its decisions.

(b)

(i) Expressing the knowledge given in first-order predicate logic.

- $VxVy[Harder(x,y) => Cuts(x,y)]$
- $VxVyVz[Cuts(x,y)\&Cuts(y,z) => Cuts(x,z)]$
- $Harder(Diamond, Steel)$
- $Harder(Steel, Copper)$

(ii) Converting the clauses into normal form.

$\neg Harder(n,m) V Cuts(n,m)$ $---(1)$

$\neg Cuts(x,y) V \neg Cuts(y,z) V Cuts(x,z)$ $---(2)$

- $Harder(Diamond, Steel)$ $---(3)$

- $Harder(Steel, Copper)$ $---(4)$

(iii) For resolution, one tries to prove the negation of the expression to be proved. If this leads to a clear contradiction, then obviously the negation is false. Hence the original assertion must be true.

(a) Negating the assertion to be proved gives:

$\neg Cuts(Diamond, Copper)$ $---(5)$

(b) From (2), we specialise x to $Diamond, z$ to $Copper$.

$\neg Cuts(Diamond, y)\ V \neg Cuts(y, Copper)\ V Cuts(Diamond, Copper) ---(2a)$

(c) Resolving (2a) and (5) gives:

$\neg Cuts(Diamond, y) V \neg Cuts(y, Copper)$ $---(6)$

(d) In (1) we both replace n with y and specialise m to $Copper$.

$\neg Harder(y, Copper) V Cuts(y, Copper)$ $---(1a)$

(e) Resolving (1a) with (6) gives:

$\neg Harder(y, Copper) V \neg Cuts(Diamond, y)$ $---(7)$

(f) From (1), we both specialise n to $Diamond$ and replace m by y

$\neg Harder(Diamond, y) V Cuts(Diamond, y)$ $---(1b)$

(g) Resolving (1b) and (7)

$\neg Harder(Diamond, y) V \neg Harder(y, Copper)$ $---(8)$

(h) In (8), we specialise y to $Steel$.

$\neg Harder(Diamond, Steel) V \neg Harder(steel, Copper)$ $---(8a)$

(i) Resolving (3) and (8)

$\neg Harder(Steel, Copper)$ $---(9)$

(i) Resolving (4) and (9)

$Harder(Steel, Copper)$ $---(4)$

$\neg Harder(Steel, Copper)$ $---(9)$

Nil $---(10)$

This indicates a contradiction. This indicates that the negation of the original theorem is false, hence the original theorem is proven to be true.

(c)

Some latitude here for answers but the point of this question is to see how well the student appreciates the practicality of this technique for 'real world' (i.e. large) problems. In a real materials selection problem, properties of materials (such as hardness) do not take on discrete values for given families of material but occupy 'regions'. A particular type of steel, for example, might even be less hard than some coppers. The logical expressions above are rather too dogmatic or limiting in their manner of knowledge representation. Some other form of logic, such as fuzzy logic, might be more appropriate. Also, with large material databases, the time taken for the computer to find or resolve a proposition may be unrealistically high.