# COMPUTER SCIENCE TRIPOS Part IB

Wednesday 4 June 2008 1.30 to 4.30

PAPER 5

*Answer **five** questions.*
*No more than **two** questions from any one section are to be answered.*

*Submit the answers in five **separate** bundles, each with its own cover sheet. On each cover sheet, write the numbers of **all** attempted questions, and circle the number of the question attached.*

**You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator**

STATIONERY REQUIREMENTS
*Script paper*
*Blue cover sheets*
*Tags*

SPECIAL REQUIREMENTS
*None*

## SECTION A

### 1 Software Engineering

How do software engineering tools change as systems scale? Discuss this question with reference to

($a$) a 2000-line device driver for a safety-critical sensor on board an aircraft;

($b$) a 100,000-line engine control unit for a diesel engine that adapts it for use in trucks, generators or irrigation pumps;

($c$) a 1,000,000-line social networking site such as Facebook or MySpace;

($d$) a 50,000,000-line operating system.

[20 marks]

## 2 Computer Design

(*a*) The classic MIPS 5-stage pipeline is depicted below.

| | | | | |
|---|---|---|---|---|
| instruction fetch | decode and register fetch | execute | memory access | write back |

(*i*) With reference to the 5-stage pipeline, what are *data hazards* and how can they be resolved to ensure that the programmer's model of sequential execution is always preserved whilst minimising performance impact? [6 marks]

(*ii*) With reference to the 5-stage pipeline, what are *control hazards* and how can they be resolved? [4 marks]

(*b*) If we wanted the above pipeline to mimic two processors running at half the speed, then we could have two copies (A and B) of the register state and keep the existing pipeline. The instruction–fetch stage would alternate between fetching from threads A and B on alternate clock cycles. As a consequence, if instruction fetch was from thread B, then an instruction from thread A would be in decode, B in execute, A in memory access and B in write back.

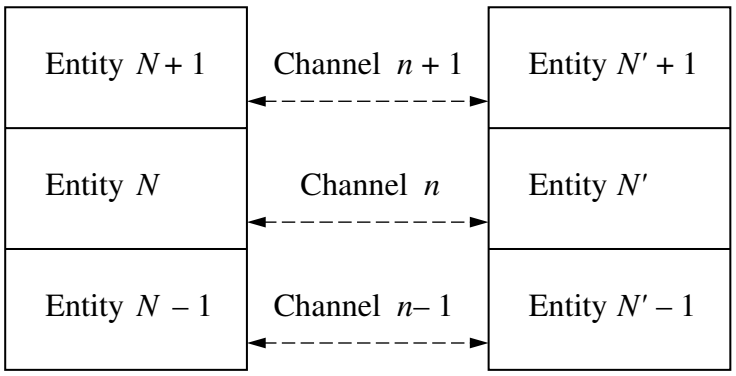
(*i*) For this dual-threaded processor, if branches are performed in the decode stage, does the pipeline exhibit a branch delay slot? [5 marks]

(*ii*) How can the resolution of data hazards be simplified for this dual-threaded processor? [5 marks]

## 3 Digital Communication I

($a$) Describe *five* physical properties of a communications channel. [5 marks]

($b$) Consider the figure below. Entities $N$ and $N'$ use an ARQ system.




($i$) Explain how the latency of channel $n-1$ can have a direct effect on the capacity of channel $n$. [6 marks]

($ii$) Define *windowing* as it relates to an ARQ system and describe how the capacity of the ARQ system may be improved through its use. [4 marks]

($iii$) If an ARQ system is used for an interactive session, the ARQ system can lead to many small packets, each under-full and perhaps sent with significant overhead. Design and describe an algorithm that overcomes the limitation of sending many mostly-empty packets for an interactive session. [5 marks]

## 4 Concurrent Systems and Applications

(*a*) A web server is an application that listens for incoming network connections on TCP port 80. Once a connection is established, the task of processing client requests and sending replies can be handled by an instance of a `Worker` class which you may assume already exists. `Worker` implements the `java.lang.Runnable` interface and has an accessible constructor that takes as argument a `java.net.Socket` object representing the network connection to a client.

Provide the Java code for a webserver which, upon start-up, attempts to listen on TCP port 80 and starts a new `Thread` running a new `Worker` for every connection. Your program should print helpful error messages indicating the likely cause of problems when it is unable to proceed as expected. [10 marks]

(*b*) A busy web server might expect to handle concurrent requests to read and update some shared data and could use Timestamp Ordering (TSO) to enforce isolation between concurrent transactions.

(*i*) Explain how TSO enforces isolation. [5 marks]

(*ii*) Is TSO appropriate for a web server application? Explain your reasoning. [5 marks]

## SECTION B

### 5 Computer Graphics and Image Processing

(*a*) Describe in detail an algorithm that returns the minimum distance from a point to a line segment in two dimensions. Ensure that you include all of your assumptions and all necessary mathematical calculations. [7 marks]

(*b*) A quadratic Bézier curve is defined by three points, $P_1$, $P_2$, $P_3$, and a parameter, $t$:

$$P(t) = (1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3,\ 0 \leq t \leq 1$$

Describe an algorithm that draws the quadratic Bézier curve, using straight lines only, to within a tolerance $\tau$. You may use the algorithm from part (*a*) and you may assume that you already have an algorithm for drawing a straight line. [8 marks]

(*c*) Consider the control of detail in a curve that is represented by a sequence of many straight line segments. Describe how Douglas and Pücker's algorithm can be used to remove superfluous points. You may use the algorithm from part (*a*). [5 marks]

## 6 Compiler Construction

Consider the following grammar for expressions (where `Id` is a terminal symbol representing an identifier resulting from lexical analysis):

```
Expr  ::=  1 | 2 | Id | Expr + Expr | Expr / Expr |
                       Expr ^ Expr | (Expr)
```

(*a*) Explain in what principal respect this grammar is unsatisfactory. [1 mark]

(*b*) Assuming further that `+` is to be left-associative, `^` is to be right-associative and `/` is to be *non-associative* (i.e. `2/2/2` is forbidden but `(2/2)/2` and `2/(2/2)` are allowed), re-write the grammar to reflect this. [4 marks]

(*c*) List the terminal symbols and non-terminal symbols, and count the production rules both in the original grammar and in the grammar in your answer to part (*b*). Indicate the *start symbol* in both grammars. [2 marks]

(*d*) Define a type or types (in C, Java, or ML) suitable for holding an abstract syntax tree resulting from your answer to part (*b*). [2 marks]

(*e*) Give a brief and elementary explanation of the principles of how the grammar resulting from part (*b*) might be used to create a syntax analyser taking a token stream as input (via calls to function `lex()`) and giving as output an abstract syntax tree corresponding to part (*d*). Mention both hand-written and automatically-generated syntax analysers. [8 marks]

(*f*) Summarise any issues related to left- or right-associative operators in the two techniques (in implementing the parser and in constructing the tool) you outlined in part (*e*). [3 marks]

## 7 Concepts in Programming Languages

(*a*) Write a procedure and a call to it in block-structured pseudocode such that the execution of the procedure under *pass-by-reference* and under *pass-by-value/result* yields different outcomes. Justify your answer.

[7 marks]

(*b*) Explain the meaning of *static* (i.e. compile-time) and *dynamic* (i.e. run-time) type checking.

Compare the advantages and disadvantages of these two approaches to type checking from the point of view of the language designer, the language implementer, and the programmer.

[6 marks]

(*c*) Explain how *objects* can be simulated in SML, giving an example.

Does it follow that SML, together with its module system, is an object-oriented programming language? Why?

[7 marks]

## 8 Databases

(*a*) What is the difference between a *key* and a *functional dependency*? [3 marks]

(*b*) The schema $R(A,\ B,\ C,\ D,\ E)$ has the following functional dependencies.

$$\begin{aligned} A,\ B &\rightarrow C \\ B,\ C &\rightarrow D \\ C,\ D &\rightarrow E \\ D,\ E &\rightarrow A \end{aligned}$$

(*i*) What are all of the keys of $R$? [3 marks]

(*ii*) Which functional dependencies violate Boyce–Codd Normal form (BCNF)? [3 marks]

(*iii*) Which functional dependencies violate Third Normal form (3NF)? [3 marks]

(*iv*) Find a lossless-join decomposition of $R$ into BCNF relations. [8 marks]

## SECTION C

### 9 Foundations of Functional Programming

(*a*) Define the translation of the *call-by-name* $\lambda$-calculus into continuation passing style. [9 marks]

(*b*) How does the translation differ for the *call-by-value* $\lambda$-calculus? [2 marks]

(*c*) Now consider extending the *call-by-name* $\lambda$-calculus with exceptions:

$$\begin{array}{rcl} M & ::== & \mathsf{try}\ M\ \mathsf{catch}\ M \\ & | & \mathsf{raise} \\ & | & \lambda x.\, M \\ & | & M\ M \\ & | & x \end{array}$$

where it reduces in the following way:

$$\begin{array}{rcl} \mathsf{try}\ \mathsf{raise}\ \mathsf{catch}\ M & \rightarrow & M \\ \mathsf{try}\ \lambda x.M_1\ \mathsf{catch}\ M_2 & \rightarrow & \lambda x.M_1 \\ \mathsf{raise}\ M & \rightarrow & \mathsf{raise} \end{array}$$

Show how to translate this language into pure $\lambda$-calculus using continuations.

[Hint: Use two continuations: one for the exceptional case, and one for the normal case.]

[9 marks]

## 10 Computation Theory

(*a*) *The Halting Problem for register machines is unsolvable.* State, without proof, a precise form of this result. [3 marks]

(*b*) Let the computation by program $c$ on data $d$ be represented by the natural number $k$ that codes the pair $(c, d)$. By considering the set $H(k)$ of the HALTing computations represented by codes $k' < k$, show that there is an increasing total function $h(k)$ which *grows too fast* to be computable. [6 marks]

(*c*) Given $h : \mathbb{N} \to \mathbb{N}$ with the above property

$$\begin{aligned} \text{let} \quad f(k) &= h(k) + k \\ \text{and} \quad g(x) &= \sup\{k : f(k) \leqslant x\}. \end{aligned}$$

Then $f : \mathbb{N} \to \mathbb{N}$ is strictly increasing, and $g : \mathbb{N} \to \mathbb{N}$ satisfies

$$g(f(k)) = k, \quad g(x) < k \quad \text{for all } x < f(k).$$

Show that $g$ *grows too slowly* to be computable in the following sense:

given $G : \mathbb{N} \to \mathbb{N}$ such that

- (*i*) $\{G(n) : n \in \mathbb{N}\}$ is unbounded
- (*ii*) $G(n) \leqslant g(n)$ for all $n \in \mathbb{N}$

then $G(n)$ is *not* computable. [11 marks]

## 11 Semantics of Programming Languages

Below is a fragment of $L_1$, equipped with a rather strange semantics. Call it $L_?$.

Booleans $b \in \mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$
Integers $n \in \mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$
Locations $\ell \in \mathbb{L} = \{l_0, l_1, l_2, \ldots\}$
Stores $s$, finite partial functions from $\mathbb{L}$ to $\mathbb{Z}$
Operations $op ::= + \mid \geq$
Expressions $e ::= b \mid n \mid !\ell \mid \ell := e \mid e_1 \;\; op \;\; e_2 \mid \mathsf{if} \;\; e \;\; \mathsf{then} \;\; e_1 \;\; \mathsf{else} \;\; e_2$

(op +) $\langle n_1 + n_2, s\rangle \longrightarrow \langle n, s\rangle \quad$ if $n = n_1 + n_2$

(op $\geq$) $\langle n_1 \geq n_2, s\rangle \longrightarrow \langle b, s\rangle \quad$ if $b = (n_1 \geq n_2)$

(opA) $\dfrac{\langle e_1, s\rangle \longrightarrow \langle e_1', s'\rangle}{\langle e_1 \;\; op \;\; e_2, s\rangle \longrightarrow \langle e_1' \;\; op \;\; e_2, s'\rangle}$ (opB) $\dfrac{\langle e_2, s\rangle \longrightarrow \langle e_2', s'\rangle}{\langle e_1 \;\; op \;\; e_2, s\rangle \longrightarrow \langle e_1 \;\; op \;\; e_2', s'\rangle}$

(deref) $\langle !\ell, s\rangle \longrightarrow \langle n, s\rangle \quad$ if $\ell \in \mathrm{dom}(s)$ and $s(\ell) = n$

(assignA) $\langle \ell := n, s\rangle \longrightarrow \langle n, s + \{\ell \mapsto n\}\rangle \quad$ if $\ell \in \mathrm{dom}(s)$

(assignB) $\dfrac{\langle e, s\rangle \longrightarrow \langle e', s'\rangle}{\langle \ell := e, s\rangle \longrightarrow \langle \ell := e', s'\rangle}$

(ifA) $\dfrac{\langle e, s_1\rangle \longrightarrow^* \langle \mathsf{true}, s_2\rangle}{\langle \mathsf{if} \;\; e \;\; \mathsf{then} \;\; e_1 \;\; \mathsf{else} \;\; e_2, s_1\rangle \longrightarrow \langle e_1, s_1\rangle}$

(ifB) $\dfrac{\langle e, s_1\rangle \longrightarrow^* \langle \mathsf{false}, s_2\rangle}{\langle \mathsf{if} \;\; e \;\; \mathsf{then} \;\; e_1 \;\; \mathsf{else} \;\; e_2, s_1\rangle \longrightarrow \langle e_2, s_1\rangle}$

Here $\longrightarrow^*$ is the reflexive transitive closure of $\longrightarrow$, defined by:

(incl) $\dfrac{\langle e, s\rangle \longrightarrow \langle e', s'\rangle}{\langle e, s\rangle \longrightarrow^* \langle e', s'\rangle}$ (tran) $\dfrac{\langle e, s\rangle \longrightarrow^* \langle e', s'\rangle \longrightarrow^* \langle e'', s''\rangle}{\langle e, s\rangle \longrightarrow^* \langle e'', s''\rangle}$

(refl) $\dfrac{}{\langle e, s\rangle \longrightarrow^* \langle e, s\rangle}$

($a$) Give a terminating sequence of reduction steps, with full derivations for each, of the configuration

$$\langle \mathsf{if} \;\; (\ell_0 := 3) \geq 2 \;\; \mathsf{then} \;\; 7 \;\; \mathsf{else} \;\; 8, \{\ell_0 \mapsto 0\}\rangle \qquad \text{[5 marks]}$$

($b$) Describe, with examples and alternative reduction rules, how the behaviour of $L_?$ expressions differs from that in $L_1$: for ($i$) binary operations, ($ii$) store operations, and ($iii$) conditionals. Discuss the effects that these differences could have on programming in the language. [15 marks]

## 12 Complexity Theory

($a$) Give a precise definition of what it means for one decision problem to be polynomial-time reducible to another. [3 marks]

($b$) Consider the following two decision problems:

**HamCycle**: Given a graph $G = (V, E)$ does it contain a cycle that visits every vertex exactly once?

**HamPath**: Given a graph $G = (V, E)$ and two distinguished vertices $s, t \in V$, is there a simple path in $G$ that starts at $s$, ends at $t$ and visits every other vertex exactly once?

Show that **HamCycle** is polynomial-time reducible to **HamPath**. [8 marks]

($c$) The following decision problem is known to be solvable in polynomial time:

**EulerCycle**: Given a graph $G = (V, E)$ does it contain a cycle that visits every edge exactly once?

What can you conclude about the truth of the following statements? Justify your answers.

($i$) **EulerCycle** is polynomial-time reducible to **HamCycle**. [3 marks]

($ii$) **EulerCycle** is polynomial-time reducible to **HamPath**. [3 marks]

($iii$) **HamPath** is polynomial-time reducible to **EulerCycle**. [3 marks]

**END OF PAPER**