UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2001

BEng Honours Degree in Computing Part III
MEng Honours Degree in Electrical Engineering Part IV
BEng Honours Degree in Information Systems Engineering Part III
MEng Honours Degree in Information Systems Engineering Part III
MSc in Advanced Computing
for Internal Students of the Imperial College of Science, Technology and Medicine

*This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute*

# PAPER C332=I3.26=E4.31

# ADVANCED COMPUTER ARCHITECTURE

Thursday 10 May 2001, 14:30
Duration: 120 minutes

*Answer THREE questions*

Paper contains 4 questions
Calculators not required

1a What are the names of the four cache line states in the Berkeley cache coherency protocol?

b In your answer above, you will have identified two "dirty" states. Under what circumstances would a cache line in a given processor $P_i$ change from one dirty state to the other, and then back again?

c When a processor $P_j$ suffers a read miss, where does it get the data from? There are two cases; explain what distinguishes the two situations.

d "Read snarfing" is a proposed modification to the Berkeley protocol. The idea is that, since each cache controller has to process every read miss request and reply (because they are all broadcast on the bus), it may as well update its cache with the resulting value.

*(i)* Under which precise circumstances might it make sense for a cache controller to do this?

*(ii)* For what kind of application program behaviour might read snarfing be a good idea?

*(iii)* What are the potential disadvantages of read snarfing?

*(The four main parts carry, respectively, 20%, 20%, 20% and 40% of the marks).*

2 In this question, consider a single-issue, dynamically-scheduled processor using Tomasulo's scheme and a Re-order Buffer (ROB) to handle speculative execution.

a Explain in detail the steps involved in issuing a floating-point add instruction

`ADDD F3, F1, F2 (F3=F1+F2)`

To simplify your answer, consider only the case when neither operand is already available in a register; both are the destinations of instructions which have not yet been completed.

b Explain in detail what happens when a functional unit such as the FP adder finishes.

c Consider the following assembler code sequence:

```
     LD   F1,(R1)
     LD   R4,(R2)
     BEQZ R4,L1
     ADDD F2,F2,F1
L1:  LD   F1,F3
```

A timing diagram for execution of this sequence using a particular processor design is given as follows:

| | *Cycle:* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LD F1,(R1) | IF | IS | M | M | M | M | WB | | | | |
| | LD R4,(R2) | | IF | IS | M | M | M | M | WB | | | |
| | BEQZ R4,L1 | | | IF | IS | St | St | St | EX | | | |
| | ADDD F2,F2,F1 | | | | IF | IS | St | EX | EX | WB | | |
| L1: | LD F1,F3 | | | | | IF | IS | WB | | | | |

("`St`" marks a cycle in which the instruction is stalled).

(i) Explain what happens to the result of the `ADDD` instruction if the `BEQZ` instruction is correctly predicted to be not taken.

(ii) Explain what happens to the result of the `LD` instruction labelled `L1` if the `BEQZ` instruction is *incorrectly* predicted to be not taken.

(iii) Draw a similar diagram for this processor design for the following slight variation on this code:

```
     LD   R4,(R2)
     LD   F1,(R1)
     BEQZ R4,L1
     ADDD F2,F2,F1
L1:  LD   F1,F3
```

Label each of the stalls, and explain why each one occurs.

d Under what circumstances might the instruction schedule in part (iii) lead to better performance than the original assembler code sequence?

*(The four main parts carry, respectively, 20%, 20%, 50% and 10% of the marks).*

3a What is a "bimodal" conditional branch?

b Draw a diagram illustrating how a 2-bit bimodal branch predictor is implemented.

c When and how is the branch history table updated?

d Give a high-level-language example of a code sequence in which a correlating predictor would perform better than a bimodal predictor. Explain why.

e Consider an 8-issue processor capable of extensive speculative execution.

*(i)* Explain what happens when a correctly-predicted, taken branch occurs in the middle of an instruction issue packet.

*(ii)* The idea of a "trace cache" is to avoid this problem. When a correctly-predicted, taken branch is completed, a "trace cache fill unit" constructs a packed instruction packet containing 8 instructions from the taken instruction sequence. Draw a diagram showing how the trace cache could be used in combination with branch prediction to fetch the appropriate instruction packet. Explain how it might work.

*(The six parts/subparts carry, respectively, 10%, 20%, 10%, 20%, 10% and 30% of the marks).*

4 In this question, consider the following loop:

```
declare float U[0:M, 0:N]

for t = 1 to M do
  for i = 1 to N-1 do
S:  U[t,i] = (U[t-1,i-1] + U[t-1,i+1]) * 0.5
```

a Suppose `M=4` and `N=6`. Draw the iteration space graph for the loop. Mark on the graph all the dependences present.

b Write down all the dependences present in the loop. For each dependence, indicate whether it is a data-dependence or an anti-dependence, whether it is loop carried, and write down its dependence distance vector.

c Write down a unimodular transformation matrix which represents a valid interchange of the two loops.

d Draw the iteration space for the loop above after this transformation has been applied. Show the dependences.

e Can the transformed loop be tiled? Justify your answer with reference to your iteration space graph.

f Suppose now that `M=4` and `N` is large. Given a fully-associative data cache with enough space for 128 `doubles`, what is the largest tile size which still makes efficient use of the cache?

g Explain, using a diagram, what could go wrong if the cache is direct-mapped instead of fully-associative.

*(The seven parts carry, respectively, 15%, 20%, 10%, 15%, 10%, 20% and 10% of the marks).*