COMP25212

Two hours

No special instructions.

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

System Architecture

Date

Time

Please answer any THREE Questions from the FOUR questions provided

Use a SEPARATE answerbook for each SECTION

For full marks your answers should be concise as well as accurate.
Marks will be awarded for reasoning and method as well as being correct

---

The use of electronic calculators is permitted provided they
are not programmable and do not store text.

[PTO]

Section A

1. **Caches**

a) Explain why CPU caches are important in the design of modern computer systems. (2 marks)

CPU clock frequencies are much faster than main memory access times (1) Caches can be used to hold subsets of main memory, but the majority of program working sets, to lower effective memory access times to close to the clock time. (1)

b) Explain the differences between fully associative, direct mapped and set associative caches. (6 marks)

In a fully associative cache, any memory location can appear in any cache location (1) and every cache location needs to be searched on every access to see if a hit occurs. (1)
In a direct mapped cache, each memory location can occur at only one location in the cache (1) since the cache is indexed by the low-order bits of the memory address. This means that only one cache location needs to be searched on every memory access (1)
In a set associative cache, each memory location can occur in only a small subset of cache locations (1). A set associative cache can be described as multiple direct mapped caches operating in parallel. (1)

c) Explain how cache misses can be categorized as Compulsory, Capacity and Conflict misses. (6 marks)

Compulsory: a miss on a memory location that could not possibly have been in the cache (1), for example following a cache flush or the first instruction fetch after loading a program into main memory (1)
A Capacity miss occurs when the cache is full of other items (2)
A Conflict miss occurs when two or more locations in main memory compete for the same locations in a cache, for example in a direct-mapped or set associative cache. (2)

A typical program accesses a number of different memory regions, one, for example containing its instructions, another containing local variables in the current stack frame, and a third containing data stored on a heap. Explain how such a program might cause Conflict cache misses in

[Note: this part of Q1 asks the students to problem-solve – it requires them to think about concepts discussed in the lectures, but to make their own deductions and projections.]

d) a direct-mapped cache (1 mark)

If the addresses of the program instructions and its local variables map to the same locations in a direct-mapped cache, they will both suffer conflict misses (1)

e) a 2-way set-associative cache (2 marks)

If the instructions, local variables and heap data all map to the same index value in a set-associative cache, conflict misses will occur in a 2-way cache. (2)

f) a 4-way set-associative cache (2 marks)

We need 5 memory areas to cause conflicts giving misses in a 4-way cache – so we need (eg) program instructions, local variables, and heap accesses to variables in at least 3 areas all with the same cache index value to cause this. (2)

g) a fully-associative cache (1 mark)

A fully-associative cache doesn't suffer Conflict misses (1)

2. **Virtualization**

a) What is "System Virtualization"? (1 mark)

A means of running complete operating systems, library and application stacks under the control of a layer of software, known as a Virtual Machine Monitor or Hypervisor. (1)

b) What are the major goals of System Virtualization? (2 marks)

Isolation of the guest running operating system from the exact details of the hardware system that it is running on, (1) for example the exact CPU type, the memory configuration, and the detailed nature of its peripheral devices. (1)

Why are these goals not achievable within existing software components? (1 mark)

Although these goals are shared with the original goals of Operating Systems, current designs of these create too much internal state related to the hardware configuration of their underlying hardware to be able to (eg) migrate a running operating system easily. (1)

What do the following terms mean, and how can they be implemented using System Virtualization?

c) Rapid provisioning (4 marks)

The deployment of an operating system image, including libraries and application, that has been previously configured and saved, perhaps in an archive of virtual machines, onto a hardware system (2)
By copying the file containing a "frozen" image of the required virtual machine to the host system and resuming it under control of the Virtual Machine Monitor/Hypervisor. (2)

d) Checkpointing and restoring (4 marks)

Every time the VMM/Hypervisor pauses a Virtual Machine (for example to time slice operation of a different VM), enough state is saved to be able to resume the VM later. (1) Checkpointing saves this state in a file, alongside an image of the "physical" memory of the VM and its configuration, for later usage. (1) Restoring this checkpoint is a matter of moving the memory image to the correct place, reloading relevant hypervisor state from that stored earlier, and resuming execution. (1) Credit for noting issues (without solutions) for network connections, permanent storage, etc. (1)

e) Live migration (4 marks)

Live migration is moving a VM from one hardware host system to another without significant downtime (1) It is achieved by first copying all memory pages from the source VM to the destination, clearing the VMM's idea of a "dirty" for each page as it is copied. (1) Since the source VM continues to run, it will re-dirty pages that have already been copied, so these

need to be detected in a second scan and re-copied. (1) When the set of "dirty" pages gets sufficiently small, the source VM is suspended, all remaining dirty pages and VM metadata is copied to the destination VMM, and the VM resumed there. (1)

A "Reverse debugger" allows a program that is being debugged to run backwards, so that a programmer can, for example, step backwards from a program crash to examine how the program reached that point. Reverse debuggers are typically implemented by storing a trace of every instruction executed and its results, so that most recent instructions can be "undone" to simulate stepping backwards. This is necessarily slow.

f) Outline how a "reverse debugger" could be implemented using, as a basis, one of the terms you have described in parts (c), (d) or (e)? (4 marks)

[NOTE: this is a problem-solving question – we didn't cover this example in the lecture]

Instead of stepping backwards from location n to location n-1, we could step forward from a recent checkpoint c by (n-c)-1 steps. (2) This suggests that we can implement a reverse debugger using checkpointing and restoring, (d) above. (1) If we need to step backwards through a checkpoint c, we could fall back to the previous checkpoint b and step forwards (c-b)-1 steps. (1)

Section B

3. **Pipelining**

a) Explain the concept of pipelining in the context of processor design and the benefits its utilisation provides. (4 marks)

It is a technique in which the instructions are split into different stages in a way that multiple instructions can overlap their execution.
It provides a more efficient utilisation of the processor resources while at the same time allows increasing clock frequency. Instruction throughput is increased.

b) Illustrate the operation of a classic 5-stage pipeline with the following example. Assume all memory accesses hit the cache and that all types of forwarding are implemented. (4 marks)

```
ldr r1 x
ldr r2 y
add r4 r1 r2
ldr r3 z
sub r5 r4 r3
str r5 a
```

Solution:

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11
ldr r1 x	IF	ID	EX	MEM	WB							
ldr r2 y		IF	ID	EX	MEM	WB						
add r4 r1 r2			IF	ID	stall	EX	MEM	WB				
ldr r3 z				IF	stall	ID	EX	MEM	WB			
sub r5 r4 r3						IF	ID	stall	EX	MEM	WB	
str r5 a							IF	stall	ID	EX	MEM	WB

Applied penalties:
Non-straight stalls (-1 mark)
Instructions overpassing each others (-3 marks)
Incorrect forwarding (-1 mark)
Data dependencies not observed (-4 marks)

c) Rearrange the instructions in question b) to accelerate the execution of the code and show how the pipeline would behave with your new arrangement. (6 marks)

Solution:

[NOTE: this is a problem-solving question]

Penalties: same as above

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9
ldr r1 x	IF	ID	EX	MEM	WB					
ldr r2 y		IF	ID	EX	MEM	WB				
ldr r3 z			IF	ID	EX	MEM	WB			
add r4 r1 r2				IF	ID	EX	MEM	WB		
sub r5 r4 r3					IF	ID	EX	MEM	WB	
str r5 a						IF	ID	EX	MEM	WB

d) Describe the problems that can arise with the use of pipelining and how they can be solved / alleviated. (6 marks)

Data Hazard: An instruction needs to read from a register that is written by a previous instruction before it is stored in the register bank which will require to stall the pipeline until the data is available.
To reduce the impact of data hazards, forwarding/bypassing can be used. It consists in adding connections from the output of the ALU (stage 3) and from the output of the Memory Access (stage 4) to the input of the ALU. The reordering of instructions during compilation can help to reduce the number of pipeline stalls by separating enough those instructions that have data dependencies.

Control Hazard: When executing a branch, the following instruction is not decided until stage 2 if it is unconditional or stage 3 if it is conditional, so the instruction fetched after a branch may not be the one that needs to be executed after the branch instruction.
The solution is either to stall the pipeline once a branch instruction is decoded, or to instrument the compiler to add NOP instructions after each branch instruction. A more advanced solution is to use branch prediction and perform speculative execution, but a roll-back mechanism needs to be implemented to undo any changes made if prediction fails.

Incorrect hazards or solutions penalise the total mark

4. **Multithreading / Multicore**

a) Explain the differences between superscalar, multithreading and multicore. Are these techniques compatible with each other? (7 marks)

A superscalar processor replicates execution logic to allow issuing several instructions per cycle from a single instruction flow (thread).
With multithreading the control logic is replicated so that instructions from several instruction flows (threads) can be issued but the execution logic is shared among the different threads.
Finally, with multicore all the processing logic (control + execution) is replicated so that the different instruction flows are executed independently.
All of them can be implemented in the same processor and indeed most modern processors implement all of them together. (e.g. Intel core i7 'Sandy Bridge' architecture has up to 4 cores with 2-way hyperthreading and 4-way superscalar).

b) Imagine you need to design a system which will be used to compute 4 different outputs from a single stream of input data, e.g. an embedded system to control a plantation which takes input from a distributed sensor network and decides in real-time when to perform processes such as fertilization, irrigation, fumigation or application of herbicides. If the computation of the 4 outputs features low ILP, which of the 3 previous approaches seems to be the most efficient and why? (4 marks)

[NOTE: this is a problem-solving question which requires a global knowledge of the subject – not covered in the lectures]

Let's consider the three alternatives.
As the 4 operations have low ILP implementing a superscalar processor would not improve the performance whereas it increases complexity.
Implementing a multicore processor will increase complexity greatly as a cache coherency protocol will be needed which may reduce the performance due to a slower memory access due to the coherence overhead resulting of data sharing.
A (fine grained) multithreading configuration will slightly increase complexity but, as there is data shared between the 4 operations, it will be able to exploit cache locality (once one of the threads bring the data, the rest have it available, i.e. fewer cache misses) and will allow the low-ILP threads to increase the utilisation of the pipeline by reducing the number of stalls due to dependences.
Therefore multithreading is the best solution.
Other answers with well-founded reasoning may get 1 or 2 marks.

c) Enumerate and describe the different types of multithreading. (3 marks)

Coarse-grain multithreading: Instructions are issued from a single thread until there is an expensive operation (a cache miss) in which case the instruction flow is switched to a different thread.
Fine-grain multithreading: Instructions from several threads are interleaved executing an instruction from each thread unless it is stalled.

Simultaneous multithreading: This technique allows to exploit a superscalar architecture by allowing to issue several (independent) instructions from any ready-marked thread.

d) Explain what is meant by the Memory Wall (2 marks)

The memory wall is the disparity between the speeds of the processor (much faster) and the memory (much slower). Memory latency is a huge bottleneck for the performance of computing systems.

e) Explain the concept of cache coherence. (2 marks)

In multicore/multiprocessor environments the main memory is shared by the different cores. However, if a cache hierarchy is implemented each core will have copies of main memory's data in their local caches. Keeping all these copies up-to-date is what is known as cache coherence.

f) Why is cache coherence important in multicore processing systems? (2 marks)

Cache coherency is important because if no precaution is taken, two cores could see different values when accessing the same variable which is against the semantics/principles of shared memory systems which may lead to unstable or erroneous execution.

END OF EXAMINATION