*Explain each stage of a 5 stage pipeline*

1

*Briefly explain what pipelining is*

2

*What is a control hazard?*

3

*What are two ways of preventing control hazards?*

4

*Briefly explain what branch prediction is*

5

*What is used to implement branch prediction and what does it do?*

6

*What is a data hazard?*

7

*What is forwarding?*

8

*IF - Fetch instruction from memory*
*ID - Decode instruction; select registers*
*EX - Perform an operation or calculate an address*
*MEM - Access an operand in memory*
*WB - Write to registers*

1

*Where we get all the components of the CPU working at the same time, with buffers that are flushed every clock cycle inbetween each stage, so that we can overlap the execution of instructions to increase overall clock speed.*

2

*If we have a branch at the ID stage, then the fetched instruction at the IF stage will have to be ignored all the way down the pipeline, wasting one full cycle and causing a bubble.*

3

*Pipeline bubbling and branch prediction*

4

*If we can remember what address a branch directed us to fetch next from what it did when we executed that branch previously, then we can pre-emptively load that instruction in the IF stage instead of fetching the instruction at the PC.*

5

*A branch target buffer which maps the virtual address of one branch instruction onto the virtual address of the instruction that is branched to*

6

*This is where we execute instructions in parallel that depend on each other*

7

*Where we add extra paths to the architecture to pass updated register values back to previous stages of the pipeline*

8

*How can we exploit instruction level parallelism?*

9

*What does VLIW stand for?*

10

*How can we implement an out of order processor?*

11

*What does a fully associative cache store?*

12

*How does the CPU locate an item in a fully associative cache?*

13

*What is temporal locality?*

14

*What is spatial locality?*

15

*What are the three common cache replacement algorithms?*

16

*Very Long Instruction Word*

10

*Fetch multiple instructions per cycle*
*Have multiple ALU's to execute instructions in parallel*
*Have common registers and caches, since the instructions are operating on the same data*

9

*Addresses and their corresponding data*

12

*Have a buffer that instructions are fetched into*
*A scheduler to choose which instructions to execute at what times*
*A cache to store memory and register accesses until all instructions have finished so that the application can execute normally as though instructions were executed in parallel*

11

*The principle that if you use an address once, you may use it again soon e.g. loops*

14

*Hardware compares the input address with all stored addresses (in parallel)*
*If we get a match we have a hit*
*If no match we must go to main memory*

13

*Least Recently Used (LRU)*
*Round Robin*
*Random*

16

*The principle that if you use an address once, you are also likely to use addresses nearby e.e. arrays*

15

*Explain the write-through cache write strategy.*

17

*Explain the copy-back cache write strategy.*

18

*Why does a direct mapped cache usually use static RAM?*

19

*Briefly explain what a set associative cache consists of.*

20

*What is the advantage of using a set associative cache?*

21

*What two control bits are usually used in cache entries?*

22

*Explain what a compulsory cache miss is?*

23

*Explain what a capacity cache miss is?*

24

*When a cache line is replaced, if the dirty bit is set, the modified value is written to main memory*

18

*Whenever a write is done to the cache, the write is also done to main memory*

17

*A number of directly mapped caches operating in parallel*

20

*It is a lot faster than dynamic RAM*

19

*Valid bit and dirty bit*

22

*We have more flexible cache replacement strategies as we could choose any one of the caches to replace from*

21

*Since the cache is limited in size, we cant contain all of the pages for a program, so some misses will occur.*

24

*When we first start the computer, the cache is empty, so until the cache is populated, we're going to have a lot of misses*

23

*Explain what a conflict cache miss is?*

25

*Name some stuff that the operating system must load and store on a context switch.*

26

*How is a multi-threaded processor most commonly presented to the operating system?*

27

*What are the three types of hardware multithreading?*

28

*Briefly describe coarse grain multithreading.*

29

*What extras does coarse grain multithreading require from the processor?*

30

*What can a context switch do to the cache?*

31

*Briefly describe fine grained multithreading.*

32

- *Process ID*
- *Program Counter*
- *Stack Pointer*
- *General registers*
- *Memory management information*
- *Open file list (and positions)*
- *Network connections*

26

*In a direct mapped or set associative cache, there is competition between memory locations for places in the cache. If the cache was fully associative, then misses due to this wouldnt occur.*

25

- *Coarse grain*
- *Fine grain*
- *Simultaneous MultiThreading (SMT)*

28

*As a processor with multiple cores (even though only one multithreaded core may be in the processor).*

27

*You don't need to change much in the processor, just make it abort instructions after a cache miss and have it store (and later load) the state of the thread.*

30

*The processor switches threads (a context switch) whenever an expensive operation is started (such as a memory load).*

29

*This involves interleaving the instructions of several threads. When memory is accessed, instructions from other threads will be executed to ensure stalls are brief. The aim is to reduce the cost of switching CPU threads to almost nothing.*

32

*A context switch can trash the cache since the new thread may access different regions of memory and therefore all the memory reads will be misses. New values will be loaded into the cache which will destroy its previous data.*

31

*Briefly describe SMT.*

33

*What are the motivations that are driving us towards multi core systems?*

34

*What do different cores on a processor **not** share?*

35

*What is (my definition$^{TM}$ of) consistency?*

36

*What are the three special instructions used to guarantee out of order processors maintain consistency?*

37

*What is transactional memory?*

38

*What are the two most simple snooping protocols?*

39

*Describe 'Write update' (the snooping protocol).*

40

- *So many transistors per unit area, cooling is a massive issue*
- *Small transistors have unpredictable characteristics*
- *Architecture of processors is becoming too complex to reason about*
- *Exponentially more complex hardware gives sublinear performance gains*
- *Have multiple but more simple cores instead*

34

*We have instructions from multiple threads in the pipeline at the same time. This requires significant hardware overhead, although gives you more freedom for instruction scheduling (since instructions in different threads are rarely interdependent so you can interleave them).*

33

*The programmer's view of the system. For example, they expect that if a memory location is updated in one thread, then the change will be visible across all threads, not just the threads that are running on the core that has the new value in its L1D cache.*

36

*An L1 cache (split into data and instruction caches) and sometimes an L2 cache. They also have their own registers obviously*

35

*Memory that supports transactions (yeah, duh). You can read and write to it however you like, but when you're finished, you have to commit, when your transaction is checked for conflicts and rolled back if it does conflict.*

38

- *A **fence** makes sure each memory access before the fence is comp*
- *A **barrier** makes threads wait until they have all reached the bar*
- *A **lock** makes sure that only one thread enters a critical section*

37

*When a core writes a value to memory, the value is updated in its L1 cache, the cache then broadcasts the address on the bus, and the snooping caches update their copy.*

40

*Write update and write invalidate.*

39

*Describe 'Write invalidate' (the snooping protocol).*

41

*Why is write invalidate better than write update for things like loops or writes to different words of the same cache line?*

42

*What does MESI stand for?*

43

*What is a directory based protocol with reference to multi core systems?*

44

*What are the concerns about a NoC (Network on a Chip)?*

45

*Buses are* [redacted] *at any one time and are controlled by a* [redacted] *that divides its use into* [redacted]*. You can* [redacted] *in one* [redacted] *and* [redacted] *in a future one.*

46

*Name five NoC architectures.*

47

*Describe the three types of NoC routing.*

48

*If a value is being frequently updated, then write invalidate needs to happen once, but write update needs to happen on every update, which wastes power and can saturate the bus.*

42

*When a core writes a value to memory, the value is updated in its L1 cache, but sends a write invalidate message to the other caches which then invalidate the updated cache line in their copies.*

41

*A protocol where there is a directory that holds information on what each L1 cache holds. This lets cores talk on a P2P basis rather than all using one bus.*

44

*Modified, Exclusive, Shared, Invalid.*

43

*Buses are single usage at any one time and are controlled by a clock that divides its use into time slots. You can send in one slot and receive in a future one.*

46

- *Bandwidth*
- *Latency*
- *Fault tolerance*
- *Area*
- *Power dissipation*

45

| | |
|---|---|
| *Minimal* | *Always select the shortest path towards the destination* |
| *Oblivious* | *Take a fixed path every time (really simple!)* |
| *Adaptive* | *Take the least congested route, complex though and uses lots of power so its rarely used.* |

48

- *Crossbar*
- *Ring*
- *Tree*
- *Fat Tree*
- *Mesh*

47

*What are the two types of packet switching in NoC's?*

49

*What are the two types of virtualisation?*

50

*What are the three main advantages of virtualisation?*

51

*A hypervisor runs in* [redacted] *mode, and can run virtual machines in a* [redacted] *mode, having* [redacted] *for when the guest OS does something that requires* [redacted].

52

*What happens when you start a VM?*

53

*When is it best to stop a VM?*

54

*What is retained when a VM is stopped/paused?*

55

*What operations can we do on a VM?*

56

*System virtualisation (run whole OS inside software e.g. VMware) and process virtualisation (run a process under a control layer of software, e.g. JVM).*

50

*Store and forward (wait for all flits before sending) and wormhole (send flits as soon as the head flit arrives).*

49

*A hypervisor runs in privileged mode, and can run virtual machines in a unprivileged mode, having traps for when the guest OS does something that requires system privileges.*

52

*Translation (between instruction sets, system API's etc), abstraction (providing garbage collection, debugging etc), or multiplexing (e.g. RAID or emulating CD drives).*

51

*When the VM's IO is quiescent (i.e. not doing anything).*

54

- *Save the current registers*
- *Load the VM registers*
- *Move the PC to the start address of the VM*

53

- *Move VM's between machines (live migration)*
- *Take a snapshot of a VM*
- *Restore a VM from a snapshot (quickly)*
- *Load balancing using live migration*

56

*Memory, IO state, CPU registers, open files, network connections etc*

55

*What are the two phases in live migration?*

*The warm up phase and the stop and copy phase.*