1.

(a) The critical path is the series of gates that take the longest time for a stable signal to reach the end. If the clock is ticked at a speed the signal in the critical path can't keep up with, its signal will be unreliable and the processor will not work correctly. The clock can therefore not be ticked any faster than the amount of time it takes a signal to traverse the critical path.

(b) Pipelining allows multiple processes to run at once, by staggering them (ie, running a command only after the longest-resolving *component* of the critical path is stable, rather than when the entire path is stable. This can then be passed on to the next component, and each component can process the next command).
Since a pipelined datapath cuts the critical path into smaller pieces, a processor using this layout can clock much faster, since it doesn't need to wait for the entire critical path to resolve.

(c) What is a data hazard and how does a pipelined processor handle them?

A data hazard is when a process pulls in data from the register or main memory, if a command before it that has not resolved would have modified the value.
Pipelined processors can handle this by feeding back values from later in the pipeline, in case they are needed by a later command.

(d) What is a control hazard and how does a pipelined processor handle them?

A control hazard is encountered on branch commands.
When a branch occurs, the IP will either continue on as normal, or be set to a different value. So, unless this is accounted for, no branch statements would work properly (the IP would initially increment as normal and then be set once the branch statement made it through the whole pipeline.
The simplest way to deal with this is just to stall the processor while a branch is being computed, by detecting branches and inserting no-ops until they are resolved.

2.

(a) One of the major restraints on clock speed is accessing data from main memory. It takes orders of magnitude longer to do than any other process that might be carried out.
A hardware cache is a small, fast piece of memory that holds a comparatively small amount of data. If the data that a process needs happens to be on it, main memory can be bypassed and the processor doesn't need to wait nearly as long to continue. As long as we intelligently grab data to put on the hardware cache, we can dramatically lower how often main memory needs to be pulled from, and can speed up the clock.

(b) Data and instructions in programs often tend to be clustered together. If we pull memory in chunks and associate these pieces with different parts of the hardware cache, often the next instruction or piece of data the processor needs will be on the hardware cache.
Because of this, a hardware cache would be pretty useless if it (for example) mapped large sequential sections of main memory to each piece of memory it contains. Very often the

processor would end up using the same spot in the hardware cache over and over, needing to pull in data from main memory each time.

(c) I've attached the picture in the same email as this exam