**PART 1**

**Distinguish computer architecture from computer organization. Explain how these concepts play in the notion of a family of computer models offered by a given computer manufacturer.**

Computer architecture refers to attributes of a computer system which are visible to the programmer, i.e., to those attributes which have a direct impact on the logical execution of a program. These attributes include the instruction set, number and size of the processor internal registers.

Computer organization refers to the role of internal operational units and to the ways they interconnect to implement the architectural specification. These include hardware details that are transparent to the programmer such as control signals, interfaces between processor and memory and between the computer system I/O devices and the memory technology used.

Families of computer models offered by a given manufacturer often have the same architecture across the family, but have differences in organization. The different computer models have different implementations of the same architecture, allowing each model to cover different price and performance categories.

**When discussing the subject of computer architecture, one often speaks about structure and function. Distinguish these concepts.**

The structure describes, at each level, how many components there are and the way in which they are connected. For example, at the top level the structure of a computer may be perceived as a processor, main memory, mass memory, I/O and the system interconnection- that ensures the data transfer between the components, usually a bus.

The function is the operation of each individual component as part of the whole. As basic functions, a computer system must be able to: process data, transfer data between itself and the outside world, store data, and finally these need to be controlled to achieve something useful.

**Write the general equation that solves the following question: What is the mean time to failure of a computer module with redundancy 2, MTTF mod redund 2, assuming that mean time to failure and mean time to repair of the module are MTTF mod and MTTR mod and MTTFmod>>MTTRmod?**

MTTF mod redund 2 = (MTTF mod /2)/(MTTR mod/ MTTF mod) = MTTF mod^2 / (2 \* MTTRmod)

Como MTTFmod>>MTTRmod, MTTF mod redund 2 é aproximadamente igual a MTTFmod^2.

**Write the general equation that solves the following question: What is the speed up for the execution of an application in a N processor computer system, where j% of the execution time in a single processor is concurrent and every time the number of processors is doubled, the communication overhead, which is supposed to be non-parallellizable, is increased by a% of the execution time in a single processor? Assume that N is equal to 2^K, where k belongs to the natural numbers.**

Speed up = t /( ( (1-j) + a\*k + (j / N) ) )\*t )

**Take the following memory addresses expressed in decimal: 7836 (10) and 5664 (10). Convert them to hexadecimal assuming a 32-bit address length. Which one can not represent the location of a properly aligned 64-bit operand? Justify your claim in detail.**

7836 / 16 = 489 remain 12 489 / 16 = 30 remain 9 30 / 16 = 1 remain 14 1 / 16 = 0 remain 1

7836 (10) = 1E9C (16)

5664 / 16 = 354 remain 0 354 / 16 = 22 remain 2 22 / 16 = 1 remain 6 1 / 16 = 0 remain 1

5664 (10) = 1620 (16)

To represent the location of a properly aligned 64-bit operand(double-word) the adress 3 least significant bits have to be 0, therefore the address 7836 (10) or 1E9C (16) does not represent the location of a properly aligned 64-bit operand.

Half-words- address has to be even

Words- 2 least significant bits have to be 0 (divisivel por 4)

Double-words- 3 least significant bits have to be 0 (divisivel por 8)

**The Opteron presents a 40-bit memory address. What is the maximum number of bytes and 64-bit words, expressed aproximately as a power of 10, that the memory may have? Justify your claim in detail.**

The maximum number of bytes is 2^40 = 1 T (10^12) and the number of 64 bit words is aproximately (10^12)/8 which is approximately 10^11.

**PART 2**

**Control hazards are a major hurdle on pipeline performance. Explain why it is so by taking as an example the 5-stage classical pipeline organization.**

When a branch is executed, it might or might not change the PC. If a branch instructions changes the PC to its target address it is a taken branch. If it falls through it’s an untaken branch. If an instruction is a taken branch then the PC isn’t changed until the end of the ID stage.

On the next instruction IF it isn’t known if the branch was taken or not. There are several alternatives to solve this, the simplest is to redo the fetch of the instruction following the branch, once the branch is detected, during the ID stage. This makes the first IF cycle essentially a stall. If the branch is untaken there is a penalty since the correct was already fetched.

There are other alternatives, like predicted-untaken that treats every branch as untaken, allowing the hardware to continue as if the branch instruction was not executed. If the branch is taken, then the fetched instruction has to be turned into a no-op, and the fetch has to be restarted at the target address.

**Data hazards are also a major hurdle. However, by using a technique called forwarding some of its effects may be minimized. Explain how it works for the 5-stage classical pipeline organization.**

The technique forwarding uses the ALU result from the pipeline registers EX/MEM nad MEM/WB and always feeds it back to the ALU inputs. If the forwarding hardware detects that a previous operation on the ALU modified the register that corresponds to the source for the current operation on the ALU, then the control logic chooses the result fedback from the pipeline registers referred above as the input for the ALU instead of the value read from the register bank.

This solves most data hazards, one that cannot be solved is after a load intruction, since the data isnt available until the end of clock cycle 4, and the next instruction needs it at the beginning of this cycle.

**What is an exception? Give examples of exceptions that occur within and between instructions?**

An exception is an event that causes the break of the squentiality of an instruction execution, other than branches and jumps.

An event prevents instruction completion by occuring in the middle(within) of execution or it is recognized between instructions.

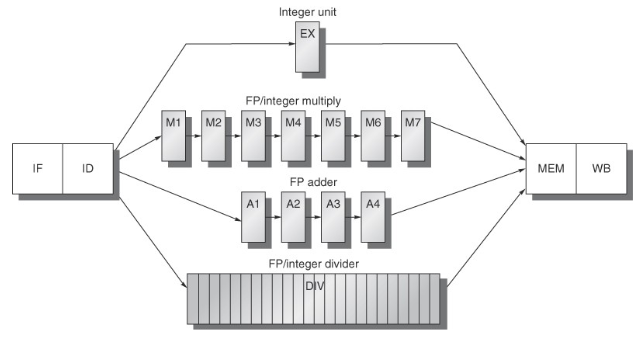
An exception that occurs within instructions can be a power failure or a hardware malfunction and one that occurs between is an I/O device request or an operating system invocation.

**What does one mean by precise exceptions? Why do you think that many recent high performance processors have introduced two modes of operation: one that has precise exceptions and one that has not?**

A precise exception is an exception for which the pipeline can be stopped, so instructions that preceded the faulting instruction can complete, and subsequent instructions can be flushed and redispatched after exception handling has completed.

For other exceptions, like floating-point exceptions, the faulting instruction in some processors writes its result before exception handling. This creates the necessity for the harware to retrieve the source operands, even if the destination is the same as one of the source operands. The reason for the two modes of operation referred in the question is to solve this problem, the one with precise exceptions must be slower to allow less overlapping of floating-point instructions.

**------------------------------------------------------------------------------------------------------------------------------------------**

**Consider the diagram below that depicts a 5-stage classical pipeline with three additional FP functional units, two of them pipelined: FP adder and FP/integer multiply, and one non-pipelined: FP/integer divide.**

**Explain why the instruction execution in this pipeline may be described as in-order execution with eventually out-of-order completion.**

Since different instructions have different exceution times, because of the different functional units, it is possible for an instruction that starts to be executed first, to finish after an instruction that started after it. For example, if a floating point multiplication intruction starts executing first and the next instruction is a floating point addition, then the addition instruction will finish before the division instruction, causing an out-of-order completion with an in-order execution.

**What kind of structural hazards may occur in this pipeline? Justify your claim in detail.**

The main structural hazard is obviously in the FP/integer divider, since it isn’t pipelined if a division instruction is executing and there is another one while the other is executing will provoke a structural hazard.

Another, since different instructions have different execution times it is possible for different instructions to arrive at the MEM stage in the same clock cycle causing an access conflict to the MEM stage creating a structural hazard.

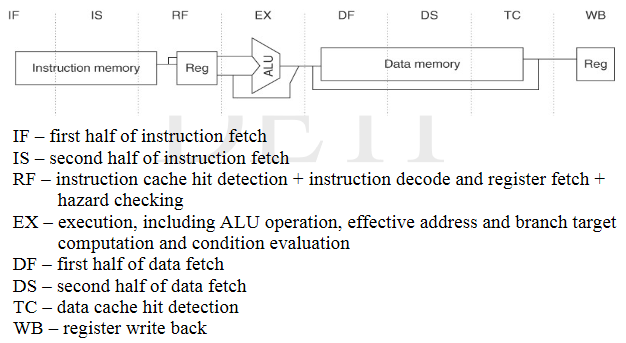
Finally, if the floating point register bank has only one write port, sequences of FP operations and along with FP load instruction can cause access conflicts to the register write port causing a structural hazard.

**What kind of data hazards may occur in this pipeline? Justify your claim in detail.**

The data hazards that occur in this pipeline are due to the different execution time of different operations, that will cause instructions to write a register before it has been written by a previous instruction. Taking the example of the floating point division instruction starting to execute before an addition instruction, if both instructions intend to write to the same register, the first to finish will be the addition and therefore will write to the register before the multiplication, causing a data hazard.

**------------------------------------------------------------------------------------------------------------------------------------------**

**The diagram below depicts the 8-stage pipeline of MIPS R4000.**

****

**Explain why the extra stages were introduced if one compares to the 5-stage classical pipeline.**

The extra stages are dedicated to deal with the memory access problem that has become a bottleneck in the processor execution time. The IF and MEM stage of the classical 5 stage pipeline is decomposed in two stages each (IF split in IF and IS, MEM split in DF and DS) to allow to maintain the rate of data transmission as fast as possible.

**What is the branch delay on this longer pipelining taking into account that the branch condition is computed at the EX stage and that R4000 uses a single-cycle branch delay scheme? Justify your claim in detail.**

Since the R4000 architecture uses a syngle-cycle branch delay scheme with a predicted-untaken strategy, which means that after a branch instruction the next instruction(delay slot) is the instruction that will be executed next if the branch is not taken. Then if the branch is not taken there are no idle cycles produced(branch delay is 0), if the branch is taken produces two idle cycles(branch delay is 2).

**------------------------------------------------------------------------------------------------------------------------------------------**

**PART 3**

**Describe the basic organization of a tournament predictor and explain how it works.**

Tournament predictors use multiple predictors, usually one based on global information and one on local information. A 2-bit hysteresis counter per branch is used to choose between the two different predictors based on which was the most successful in recent predictions. Since a 2-bit hysteresis counter is used it is necessary to have two consecutive mispredictions to change its state.

**What are the key ideas which underlie hardware-based speculation?**

Hardware-based speculation has three key ideas: dynamic branch prediction, speculation and dynamic scheduling. Dynamic branch prediction is used to select the instructions to execute. Speculation allows the processor to execute instructions before the control dependences are resolved, keeping in mind that the incorrect speculated sequence produces effects that can be undone. And finally dinamic scheduling to issue different combinations of basic blocks.

**Out-of-order execution is a feature from dynamic scheduling. In order to implement it, the ID stage of the 5-stage classical pipeline must be split into two sub-stages. What are they and which is the role each playes?**

The sub-stages created to implement out-of-order execution are the issue and read operands stages.

The issue stage decodes the instruction and checks for structural hazards. While the read operands stage waits until all data-hazards are cleared to then read the operands.

The instruction fetch precedes the issue stage, the instruction that was fetched is placed into an instruction register or a queue of pending instructions. When conditions allow instructions are issued from the register or the queue.

All instructions go through the issue stage in-order, but it is possible for some to be stalled and others to overtake them in the read operands stage and enter execution, therefore causing an out-of-order execution.

**The two basic techniques to implement dynamic scheduling are scoreboarding and the Tomasulo’s algorithm. What is the most important difference between them?**

The most important difference between them is that Tomasulo’s algorithm handles antidependences and output dependences by renaming registers dynamically. Register renaming eliminates hazards caused by those dependences by changing the name of all destination registers, including those with pending read or writes from earlier instructions, therefore, the out-of-order write doesn’t affect instructions that depend on the previous value of the operand.