REGISTER RENAMING & RAT

Lesson 6

ILP

Performance improvement can be found by increasing the instruction level parallelism (ILP). To increase ILP false dependencies must be removed. Register renaming is one method of removing false dependencies.

The Execute Stage - forwarding can supply the data in the NEXT cycle, not in the same cycle. So for a RAW dependency, the read instruction must be delayed to allow time for the write.

RAW Dependencies and WAW Dependencies

The CPI is dependent upon RAW dependencies.
WAW dependencies can cause out-of-order instructions.

Removing False Dependencies

RAW dependencies are true dependencies because the program requires a specific order of the instructions.
WAR and WAW are false dependencies because the dependency is caused when the same register is used to hold results.

Duplicating Register Values

Since false dependencies can be caused by two values using the same register, we can eliminate them by altering the use of registers.

Method 1: Instead of overwriting the register, the processor keeps track of the values that are written to a register. Then the processor can determine which value is the last one produced, which may be different than the last one written. This method is complicated and not really possible.
Instead, use register renaming.

Register Renaming

Architectural registers are registers that programmers can access. AND compilers. These are "virtualized"
Physical registers are all the places that a value can be stored.
The processor uses physical registers to rename the registers in the program to eliminate false dependencies. The RAT (Register Allocation Table) is a table that keeps track of which physical register is linked to which architectural register.

RAT

The RAT is a table that stores the location for the data that would go in an architectural register.
This will eliminate name dependencies.

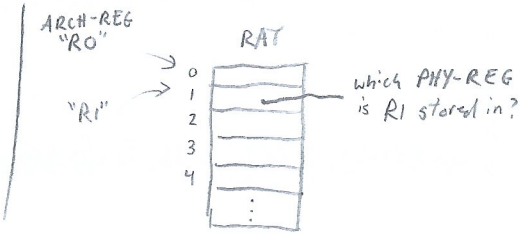
The RAT is a table that says which PHYSICAL REGISTERS has the value for which ARCHITECTURE REGISTER. So for each of the ARCH-REGs, there will be an entry in this table & it will tell us what is the current location in the PHY-REGs where the value for that ARCH-REG can be found.

- For every instruction that produces a result, we'll go to that RAT entry & remap that ARCH-REG to a new PHY-REG where we'll store that new result.
- Reading the RAT is ONLY for when we want to fetch register values, not for when we want to find out where to write.

False Dependencies After Register Renaming

Renaming improves the CPI and IPC.

Removing false dependencies allows us to execute more instructions in parallel, which increases of IPC (which conversely decreases our CPI)

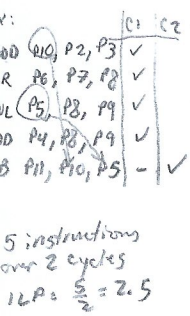


ILP = IPC w/ an "ideal" processor
still needs to follow TRUE (RAW) dependencies

Instruction Level Parallelism

ILP = IPC when the processor can do the entire instruction in 1 cycle, and the processor can do many instructions in same cycle. ILP is the property of a program, it is based on the dependencies of the program.

(so we can ignore the concept of a pipeline (no more stages))



Steps to Get ILP

1. Rename registers as they are used, keeping track of which registers are still free for assignment. (As a shortcut, you don't actually need to rename the registers. You can do this w/ the original program + JUST OBEY the TRUE DEPENDENCIES)
2. "Execute" the program to make sure the false dependencies are removed and determine when the instructions are executed.
3. Then it is the number of instructions / the number of cycles required to execute the instructions.

ILP with Control Dependencies

There are two other types of dependencies, structural and control. Structural dependencies do not apply to ILP because ILP is based on a perfect processor.

(structural dependencies occur when we don't have enough hardware to do things in the same cycle - not applicable to an "ideal" processor)

For ILP and control dependencies, again assume a perfect processor, this time with perfect branch prediction.

(so control dependencies are N/A too)

ILP vs IPC

ILP assumes an ideal out-of-order processor with perfect branch prediction that can execute an infinite number of instructions at a time, while the IPC must consider the limitations of an actual processor.

ILP >= IPC

For IPC:

If the processor is an in-order narrow issue processor, the narrow issue will be more limiting than the fact that it is in-order.

If the processor is in-order and wide issue, the fact that it is in-order will be more limiting. (b/c of "in-order", we cannot find the instructions to keep our wide-issue slots occupied)

Therefore if the processor is wide-issue (capable of issuing 4 or more instructions at a time), it should also be out-of-order. It should also be able to eliminate false dependencies and reorder instructions.

(wide-issue should be able to do out of order so that it is able to find instructions anywhere where they are + not being stopped by the first dependence.)