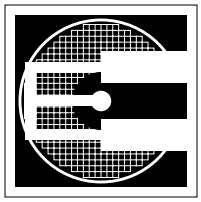


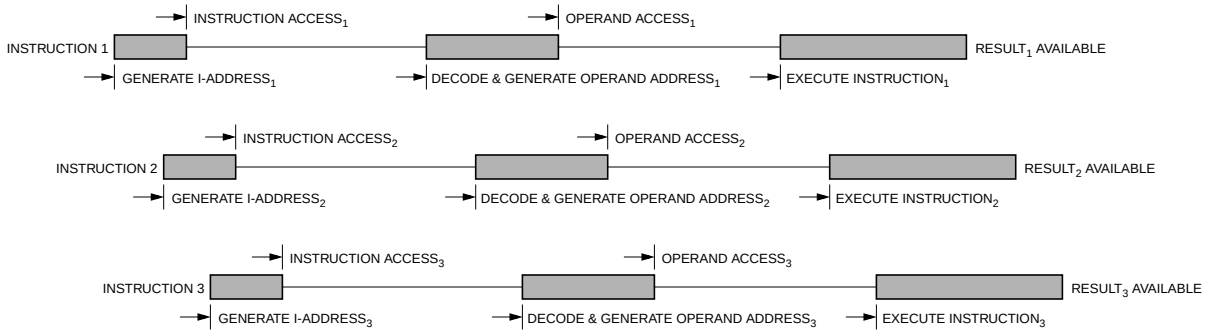
IBM 360/91's Out-of-Order Fixed-Point Pipe

ENEE 446: Digital Computer Design, Fall 2000
Prof. Bruce Jacob

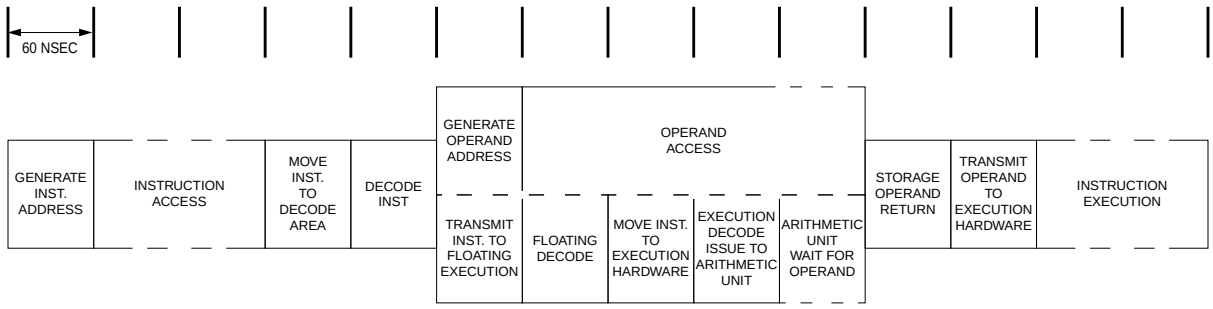
This is a guess (my guess) as to the implementation of the out-of-order instruction issue and commit mechanism in IBM's System/360 Model 91, fixed-point pipeline. The guess is based on the text and figures 2, 3, 6 and 7 in the article "The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling" by Anderson, Sparacio, and Tomasulo.

The fundamental problem is this: how does the system know when a given instruction may write to the general-purpose register file? The pipeline has in-order enqueue, out-of-order execution and completion, and it synchronizes through the register file: instructions reading their operands from the register file do not obtain them from forwarding paths. The pipeline enforces coherent writing to the register file by scheduling when instructions that would otherwise cause a write-after-write hazard are allowed access to the register file. The article mentions that each instruction that writes to a GPR increments a counter associated with that register during decode and decrements that counter at the time of register file update, and the article says that no instruction may read an operand from a GPR unless its associated counter has returned to zero. What is missing is the mechanism by which an instruction knows that it is its turn to write its result into the GPR.

The paper does not give a detailed diagram of the fixed-point pipeline ... the diagrams are a bit simple, but this is probably enough detail (taken from figure 2):

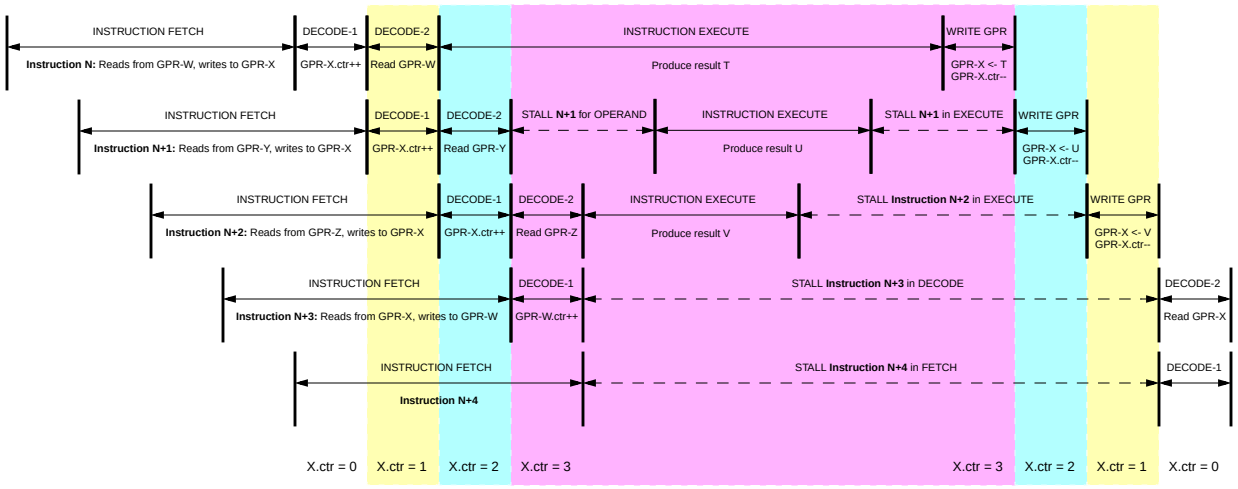


For comparison, here is the floating-point pipeline, taken from figure 3:



We'll assume for the moment that the fixed-point pipeline is similar to the floating-point pipeline, as is suggested by the general flow shown in figure 2. Here is the behavior of four instructions in

the fixed-point pipeline, three of which write to GPR X, the last of which reads from GPR X. This figure is adapted (and extrapolated) from figures 6 and 7:

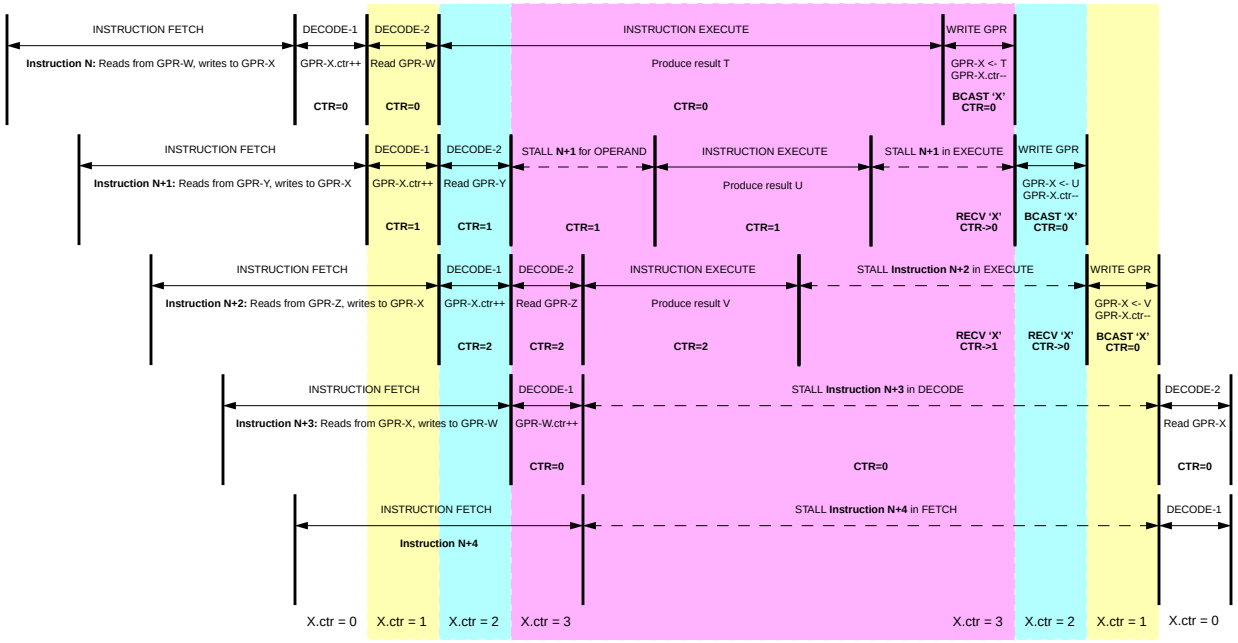


The figure clearly shows that instruction decode (and therefore instruction enqueue) is stalled until the requested read-register is free of hazards—i.e., decode is stalled until it is known that there are no more outstanding instructions that wish to write to the desired register. This is accomplished by the use of a counter associated with each register. The counter is incremented during the DECODE-1 phase whenever an instruction is decoded that wishes to write to the register. An instruction may only read from the register if its associated counter value is zero, or, presumably, if it happened to be the instruction that incremented it to 1. This mechanism is very clear from the article. Moreover, instruction issue/dispatch—the act of sending a ready instruction to a functional unit—can be done out of order. This is also clear from the article (for example, see figure 7). However, what the article does not describe in detail is the commit mechanism—how a given instruction "knows" when it is allowed to write to a GPR. For instance, in the above example, instruction N completes execution after instruction N+1, but instruction N+1 delays writing its result to the register file until N completes. How is this done?

Here is my guess. The counter behaves like a semaphore: when an instruction increments the counter, it also retains the counter's previous value, which is used like a ticket at a deli to impose order on instruction access to the shared resource (the register). The pipeline registers that hold the state for an instruction also hold the previous value of the CTR variable for the GPR that the instruction targets. In the preceding example, instruction N would retain the value "0"; instruction N+1 would retain the value "1"; and instruction N+2 would retain the value "2". In many implementations of semaphores, when the resource is used and then released by a process, the other processes waiting in line for access to the resource are notified, and they decrement their counters. When a process's counter reaches zero, it is allowed access to the resource.

We can imagine a similar scheme here, where the various copies of the X.ctr values are decremented whenever the corresponding register is written during instruction commit. The counter values held by each instruction in the pipeline (e.g. the copies of X.ctr that are stored in each pipeline register) would need to be updated whenever another instruction writes to the same register in the general-purpose register file. For instance, when instruction N writes its result to GPR X, it broadcasts that fact—i.e. the register's identifier: the value "X"—on a result bus. This

indicates to all instructions in the system (some of which might be waiting to write to GPR-X) that the current "lock-holder" for GPR-X has given up the lock. Note that only one result can be written to the register file at a time, and the register number is a small value requiring only a few bits, so this is relatively efficient. Continuing with the example, any instruction that targets the register GPR X would see this broadcast and decrement the local value for CTR that it is holding. Once that value reaches zero, the instruction may write its result back to the register file. This is illustrated below.



The figure shows the local value of CTR for each instruction during each cycle. As indicated, instruction N retains the value "0"; instruction N+1 initially retains the value "1"; and instruction N+2 initially retains the value "2". When it approaches the WRITE-GPR stage, instruction N has the local value "0" in its pipeline register, which indicates that it can write to GPR-X as soon as its result is ready. Instruction N+1 reaches the end of the EXECUTE stage before the previous instruction is finished, however, and because its local copy of CTR has a non-zero value, it stalls waiting for access to the register file. When the previous instruction writes to the register file, it broadcasts the value "X", and this fact is noted by instruction N+1, which decrements its local copy of CTR because its target matches the value on the bus: its target register is GPR-X. When its local copy of CTR is zero, instruction N+1 commits its result to the register file. The behavior of instruction N+2 is similar.

When instruction N+3 enters the DECODE phase, it is noted that it targets a different register (GPR-W), but it reads from GPR-X. Because the global value of X.ctr is non-zero, the instruction stalls in the DECODE phase. Note that when the value "X" is broadcast by other instructions when they write to GPR-X, instruction N+3 does not respond: its local copy of CTR remains at zero. This is because its CTR value corresponds to GPR-W, not GPR-X. Once the global counter X.ctr is zero, instruction N+3 is allowed to read from the register file.