Delay slot

In computer architecture, a **delay slot** is an instruction slot that gets executed without the effects of a preceding instruction. The most common form is a single arbitrary instruction located immediately after a branch instruction on a RISC or DSP architecture; this instruction will execute even if the preceding branch is taken. Thus, by design, the instructions appear to execute in an illogical or incorrect order. It is typical for assemblers to automatically reorder instructions by default, hiding the awkwardness from assembly developers and compilers.

1 Branch delay slots

When a branch instruction is involved, the location of the following delay slot instruction in the pipeline may be called a **branch delay slot**. Branch delay slots are found mainly in DSP architectures and older RISC architectures. MIPS, PA-RISC, ETRAX CRIS, SuperH, and SPARC are RISC architectures that each have a single branch delay slot; PowerPC, ARM, and the more recently designed Alpha do not have any. DSP architectures that each have a single branch delay slot include the VS DSP, µPD77230 and TMS320C3x. The SHARC DSP and MIPS-X use a double branch delay slot; such a processor will execute a pair of instructions following a branch instruction before the branch takes effect.

The following example shows delayed branches in assembly language for the SHARC DSP. Registers R0 through R9 are cleared to zero in order by number (the register cleared after R6 is R7, not R9). No instruction executes more than once.

R0 = 0; CALL fn (DB); /* call a function, below at label “fn” */ R1 = 0; /* first delay slot */ R2 = 0; /* second delay slot */ /***** discontinuity here (the CALL takes effect) *****/ R6 = 0; /* the CALL/RTS comes back here, not at “R1 = 0” */ JUMP end (DB); R7 = 0; /* first delay slot */ R8 = 0; /* second delay slot */ /***** discontinuity here (the JUMP takes effect) *****/ /* next 4 instructions are called from above, as function “fn” */ fn: R3 = 0; RTS (DB); /* return to caller, past the caller’s delay slots */ R4 = 0; /* first delay slot */ R5 = 0; /* second delay slot */ /***** discontinuity here (the RTS takes effect) *****/ end: R9 = 0;

The goal of a pipelined architecture is to complete an instruction every clock cycle. To maintain this rate, the pipeline must be full of instructions at all times. The branch delay slot is a side effect of pipelined architectures due to the branch hazard, i.e. the fact that the branch would not be resolved until the instruction has worked its way through the pipeline. A simple design would insert stalls into the pipeline after a branch instruction until the new branch target address is computed and loaded into the program counter. Each cycle where a stall is inserted is considered one branch delay slot. A more sophisticated design would execute program instructions which are not dependent on the result of the branch instruction. This optimization can be performed in software at compile time by moving instructions into branch delay slots in the in-memory instruction stream, if the hardware supports this. Another side effect is that special handling is needed when managing breakpoints on instructions as well as stepping while debugging within branch delay slot.

The ideal number of branch delay slots in a particular pipeline implementation is dictated by the number of pipeline stages, the presence of register forwarding, what stage of the pipeline the branch conditions are computed, whether or not a branch target buffer (BTB) is used and many other factors. Software compatibility requirements dictate that an architecture may not change the number of delay slots from one generation to the next. This inevitably requires that newer hardware implementations contain extra hardware to ensure that the architectural behavior is followed despite no longer being relevant.

2 Load delay slot

A load delay slot is an instruction which executes immediately after a load (of a register from memory) but does not see, and need not wait for, the result of the load. Load delay slots are very uncommon because load delays are highly unpredictable on modern hardware. A load may be satisfied from RAM or from a cache, and may be slowed by resource contention. Load delays were seen on very early RISC processor designs. The MIPS I ISA (implemented in the R2000 and R3000 microprocessors) features this behavior.

The following example is MIPS I assembly code, showing both a load delay slot and a branch delay slot.

lw v0,4(v1) # load word from address v1+4 into v0 nop # unused load delay slot jr v0 # jump to the address specified by v0 nop # unused branch delay slot

3 See also

- Control flow
- Bubble (computing)
- Branch predication

4 External links

- Branch Prediction Schemes

5 Text and image sources, contributors, and licenses

5.1 Text

5.2 Images

5.3 Content license