CORTEX-M4 INSTRUCTION TIMING

The following information was excerpted from the ARM Cortex-M4 Processor Technical Reference Manual (r0p1). Basic instruction cycle counts are found in tables 3.1, 3.2, and 7.1. The footnotes were not part of the source document. Most common situations highlighted in red.

From section 3.3.3 Load/store timings:

Instructions can be optimally paired to achieve more reductions in load and store timings. The following information may help you to achieve further reductions in timing when pairing instructions:

- `STR Rx,[Ry,#imm]` is always one cycle. This is because the address generation is performed in the initial cycle[1], and the data store is performed at the same time as the next instruction is executing. If the store is to the write buffer, and the write buffer is full or not enabled, the next instruction is delayed until the store can complete. If the store is not to the write buffer, for example to the Code segment[2], and that transaction stalls, the impact on timing is only felt if another load or store operation is executed before completion.
- `LDR PC,[any]` is always a blocking operation. This means at least two cycles for the load, and three cycles for the pipeline reload[3]. So this operation takes at least five cycles, or more if stalled on the load or the fetch.
- Any load or store that generates an address dependent on the result of a preceding data processing operation stalls the pipeline for an additional cycle while the register bank is updated. There is no forwarding path[4] for this scenario.
- `LDR Rx,[PC,#imm]` might add a cycle because of contention with the fetch unit.
- `TBB` and `TBH` are also blocking operations. These are at least two cycles for the load, one cycle for the add, and three cycles for the pipeline reload. This means at least six cycles, or more if stalled on the load or the fetch.
- `LDR [any]` are pipelined when possible. This means that if the next instruction is an `LDR` or `STR`, and the destination of the first `LDR` is not used to compute the address for the next instruction, then one cycle is removed from the cost of the next instruction. So, an `LDR` might be followed by an `STR`, so that the `STR` writes out what the `LDR` loaded. More multiple `LDR`s can be pipelined together. Some optimized examples are:
  - `LDR R0,[R1]; LDR R1,[R2]` - normally 3 cycles total.
  - `LDR R0,[R1,R2]; STR R0,[R3,#20]` - normally 3 cycles total.
  - `LDR R0,[R1,R2]; STR R1,[R3,R2]` - normally 3 cycles total.

[1] I believe the "initial cycle" here refers to the 3rd (execute phase) cycle.
[2] Assumes this means that the load map of the linker has placed some code in a read/write portion of memory, which is not common.
[3] Any non-sequential modification of the PC requires a pipeline reload, perhaps reduced by branch prediction.
[4] What is meant by a "forwarding path"?

— `LDR R0,[R1,R5]; LDR R1,[R2]; LDR R2,[R3,#4]` - normally 4 cycles total.

- Other instructions cannot be pipelined after `STR` with register offset. `STR` can only be pipelined when it follows an `LDR`, but nothing can be pipelined after the store. Even a stalled `STR` normally only takes two cycles, because of the write buffer.
- `LDREX` and `STREX` can be pipelined exactly as `LDR`. Because `STREX` is treated more like an `LDR`, it can be pipelined as explained for `LDR`. Equally `LDREX` is treated exactly as an `LDR` and so can be pipelined.
- `LDRD` and `STRD` cannot be pipelined with preceding or following instructions. However, the two words are pipelined together. So, this operation requires three cycles when not stalled.
- `LDM` and `STM` cannot be pipelined with preceding or following instructions. However, all elements after the first are pipelined together. So, a three element `LDM` takes 2+1+1 or 5[5] cycles when not stalled. Similarly, an eight-element store takes nine cycles when not stalled. When interrupted, `LDM` and `STM` instructions continue from where they left off when returned to. The continue operation adds one or two cycles to the first element when started.
- Unaligned word or halfword loads or stores add penalty cycles. A byte aligned halfword load or store adds one extra cycle to perform the operation as two bytes. A halfword aligned word load or store adds one extra cycle to perform the operation as two halfwords. A byte-aligned word load or store adds two extra cycles to perform the operation as a byte, a halfword, and a byte. These numbers increase if the memory stalls. A `STR` or `STRH` cannot delay the processor because of the write buffer.

From section 7.2.3 FPU instruction set table:

- Integer-only instructions following `VDIVR`[6] or `VSQRT` instructions complete out-of-order. `VDIV` and `VSQRT` instructions take one cycle if no more floating-point instructions are executed.
- Floating-point arithmetic data processing instructions, such as add, subtract, multiply, divide, square root, all forms of multiply with accumulate, in addition to conversions of all types take one cycle longer if their result is consumed by the following instruction.
- Both fused and chained multiply with accumulate (MAC) instructions consume their addend one cycle later, so the result of an arithmetic instruction that is followed by a multiply with accumulate instruction is consumed as the addend of the MAC instruction.

[5] Believe this should be 4, not 5.

[6] Believe that the "R" in VDIVR is a typographical error since there seems to be no such instruction.