18.1 About instruction timing

The timing information in this chapter covers each instruction in addition to interactions between instructions. It also contains information about factors that influence timings.

When looking at timings, it is important to understand the role that the system architecture plays. Every instruction must be fetched and every load/store must go out to the system. These factors are described along with intended system design, and the implications for timing.

18.2 Processor instruction timings

Table 18-1 shows the Thumb-2 subset supported in the ARMv7-M architecture. It provides cycle information including annotations to explain how instruction stream interactions affect timing. System effects, such as running code from slower memory, are also considered.

Table 18-1 Instruction timings

Instruction type	Size	Cycles count	Description
Data operations	16	1 (+P^a if PC is destination)	ADC, ADD, AND, ASR, BIC, CMN, CMP, CPY, EOR, LSL, LSR, MOV, MUL, MVN, NEG, ORR, ROR, SBC, SUB, TST, REV, REVH, REVSH, SXTB, SXTH, UXTB, and UXTH. MUL is one cycle.
Branches	16	1+P^a	B<cond>, B, BL, BX, and BLX. No BLX with immediate. If branch taken, pipeline reloads (two cycles are added).
Load-store Single	16	2[b] (+P^a if PC is destination)	LDR, LDRB, LDRH, LDRSB, LDRSH, STR, STRB, and STRH, and T variants.
Load-store Multiple	16	1+N^b (+P^a if PC loaded)	LDMIA, POP, PUSH, and STMIA.
Exception generating	16	-	BKPT stops in debug if debug enabled, fault if debug disabled. SVC faults to SVCall handler (see ARMv7-M architecture specification for details).
Data operations with immediate	32	1 (+P^a if PC is destination)	ADC{S}. ADD{S}, CMN, RSB{S}, SBC{S}, SUB{S}, CMP, AND{S}, TST, BIC{S}, EOR{S}, TEQ, ORR{S}, MOV{S}, ORN{S}, and MVN{S}.
Data operations with large immediate	32	1	MOVW, MOVT, ADDW, and SUBW. MOVW and MOVT have a 16-bit immediate (so can replace literal loads from memory). ADDW and SUBW have a 12-bit immediate (so also can replace many from memory literal loads).
Bit-field operations	32	1	BFI, BFC, UBFX, and SBFX. These are bitwise operations that enable control of position and size in bits. These both support C/C++ bit fields (in structs) in addition to many compare and some AND/OR assignment expressions.
Data operations with 3 register	32	1 (+P^a if PC is destination)	ADC{S}. ADD{S}, CMN, RSB{S}, SBC{S}, SUB{S}, CMP, AND{S}, TST, BIC{S}, EOR{S}, TEQ, ORR{S}, MOV{S}, ORN{S}, and MVN{S}. No PKxxx instructions.

Table 18-1 Instruction timings (continued)

Instruction type	Size	Cycles count	Description
Shift operations	32	1	ASR{S}, LSL{S}, LSR{S}, ROR{S}, and RRX{S}.
Miscellaneous	32	1	REV, REVH, REVSH, RBIT, CLZ, SXTB, SXTH, UXTB, and UXTH. Extension instructions same as corresponding ARM v6 16-bit instructions.
Table Branch	16	4+P[a]	Table branches for switch/case use. These are LDR with shifts and then branch.
Multiply	32	1 or 2	MUL, MLA, and MLS. MUL is one cycle and MLA and MLS are two cycles.
Multiply with 64-bit result	32	3-7[c]	UMULL, SMULL, UMLAL, and SMLAL. Cycle count based on input sizes. That is, ABS(inputs) < 64K terminates early.
Load-store addressing	32	-	Supports Format PC+/-imm12, Rbase+imm12, Rbase+/-imm8, and adjusted register including shifts. T variants used when in Privilege mode.
Load-store Single	32	2[b] (+P[a] if PC is destination)	LDR, LDRB, LDRSB, LDRH, LDRSH, STR, STRB, and STRH, and T variants. PLD and PLI are both hints and so act as a NOP.
Load-store Multiple	32	1+N[b] (+P[a] if PC is loaded)	STM, LDM, LDRD, and STRD.
Load-store Special	32	1+N[b]	LDREX, STREX, LDREXB, LDREXH, STREXB, STREXH, CLREX. These fault if no local monitor (is IMP DEF). LDREXD and STREXD are not included in this profile.
Branches	32	1+P[a]	B, BL, and B<cond>. No BLX (1) because it always changes state. No BXJ.
System	32	1-2	MSR(2) and MRS(2) replace MSR/MRS but also do more. These access the other stacks and also the status registers. CPSIE/CPSID 32-bit forms are not supported. No RFE or SRS.
System	16	1-2	CPSIE and CPSID are quick versions of MSR(2) instructions and use the standard Thumb-2 encodings, but only permit use of i and f and not a.
Extended32	32	1	NOP and YIELD (hinted NOP). No MRS (1), MSR (1), or SUBS (PC return link).

Table 18-1 Instruction timings (continued)

Instruction type	Size	Cycles count	Description
Combined Branch	16	1+P[a]	CBZ.
Extended	16	0-1[d]	IT and NOP (includes YIELD).
Divide	32	2-12[e]	SDIV and UDIV. 32/32 divides both signed and unsigned with 32-bit quotient result (no remainder, it can be derived by subtraction). This earlies out when dividend and divisor are close in size.
Sleep	32	1+W[f]	WFI, WFE, and SEV are in the class of hinted NOP instructions that control sleep behavior.
Barriers	16	1+B[g]	ISB, DSB, and DMB are barrier instructions that ensure certain actions have taken place before the next instruction is executed.
Saturation	32	1	SSAT and USAT perform saturation on a register. They perform three tasks. They normalize the value using shift, test for overflow from a selected bit position (the Q value) and set the xPSR Q bit. Saturation refers to the largest unsigned value or the largest/smallest signed value for the size selected.

a. Branches take one cycle for instruction and then pipeline reload for target instruction. Non-taken branches are 1 cycle total. Taken branches with an immediate are normally 1 cycle of pipeline reload (2 cycles total). Taken branches with register operand are normally 2 cycles of pipeline reload (3 cycles total). Pipeline reload is longer when branching to unaligned 32-bit instructions in addition to accesses to slower memory. A branch hint is emitted to the code bus that permits a slower system to pre-load. This can reduce the branch target penalty for slower memory, but never less than shown here.
b. Generally, load-store instructions take two cycles for the first access and one cycle for each additional access. Stores with immediate offsets take one cycle.
c. UMULL/SMULL/UMLAL/SMLAL use early termination depending on the size of source values. These are interruptible (abandoned/restarted), with worst case latency of one cycle. MLAL versions take four to seven cycles and MULL versions take three to five cycles. For MLAL, the signed version is one cycle longer than the unsigned.
d. IT instructions can be folded.
e. DIV timings depend on dividend and divisor. DIV is interruptible (abandoned/restarted), with worst case latency of one cycle. When dividend and divisor are similar in size, divide terminates quickly. Minimum time is for cases of divisor larger than dividend and divisor of zero. A divisor of zero returns zero (not a fault), although a debug trap is available to catch this case.
f. Sleep is one cycle for the instruction plus as many sleep cycles as appropriate. WFE only uses one cycle when event has passed. WFI is normally more than one cycle unless an interrupt happens to pend exactly when entering WFI.
g. ISB takes one cycle (acts as branch). DMB and DSB take one cycle unless data is pending in the write buffer or LSU. If an interrupt comes in during a barrier, it is abandoned/restarted.

Cycle count information:

- P = pipeline reload
- N = count of elements

- W = sleep wait
- B = barrier clearance.

In general, each instruction takes one cycle (one core clock) to start executing as shown in Table 18-1 on page 18-3. Additional cycles can be taken because of fetch stalls.

18.3 Load-store timings

This section describes how best to pair instructions. This achieves more reductions in timing.

- STR Rx,[Ry,#imm] is always one cycle. This is because the address generation is performed in the initial cycle, and the data store is performed at the same time as the next instruction is executing. If the store is to the store buffer, and the store buffer is full, the next instruction is delayed until the store can complete. If the store is not to the store buffer (such as to the Code segment) and that transaction stalls, the impact on timing is only felt if another load or store operation is executed before completion.
- LDR Rx!,[any] is not normally pipelined. That is, base update load is generally at least a two-cycle operation (more if stalled). However, if the next instruction does not require to read from a register, the load is reduced to one cycle. Non register reading instructions include CMP, TST, NOP, and non-taken IT controlled instructions.
- LDR PC,[any] is always a blocking operation. This means minimally two cycles for the load, and three cycles for the pipeline reload. So at least five cycles (more if stalled on the load or the fetch).
- LDR Rx,[PC,#imm] might add a cycle because of contention with the fetch unit.
- TBB and TBH are also blocking operations. These are minimally two cycles for the load, one cycle for the add, and three cycles for the pipeline reload. This means at least six cycles (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 non-base updating 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 LDRs can be pipelined together. Some optimized examples:
  - LDR R0,[R1]; LDR R1,[R2] - normally three cycles total
  - LDR R0,[R1,R2]; STR R0,[R3,#20] - normally three cycles total
  - LDR R0,[R1,R2]; STR R1,[R3,R2] - normally three cycles total
  - LDR R0,[R1,R5]; LDR R1,[R2]; LDR R2,[R3,#4] - normally four cycles total.

- STR with register offset cannot be pipelined after. STR can only be pipelined when after an LDR, but nothing can be pipelined after the store. Even a stalled STR normally only take two cycles, because of the store buffer (bit band, data segment, and unaligned).
- 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, STRD cannot be pipelined with preceding or following instructions. However, the two words are pipelined together. So, three cycles when not stalled.
- LDM, 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 cycles when not stalled. Similarly, an eight element store takes nine cycles when not stalled. When interrupted, LDM and STM instructions continue from where left off when returned to. The continue operation adds one or two cycles to the first element to get 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 store buffer.

