B3

APPENDIX

Instruction Cycle Timings

Andrew Sloss,
ARM; *Dominic Symes,* ARM;
Chris Wright,
Ultimodule Inc.

This appendix lists the instruction cycle timings for some common ARM implementions. Timings can vary between different revisions of an implementation and are also affected by external events such as interrupts, memory speed, and cache misses. You should treat these numbers as a guide only and verify performance measurements on real hardware. Refer to the manufacturer's data sheets for the latest timing information.

ARM cores use pipelined implementations. The number of cycles that an instruction takes may depend on the previous and following instructions. When you optimize code, you need to be aware of these interactions, described in the "Notes" column of the timing tables.

B3.1 Using the Instruction Cycle Timing Tables

Use the following steps to calculate the number of cycles taken by an instruction:

- Find the table in this appendix for the ARM core you are using. Note, ARM7xx parts usually contain an ARM7TDMI core; ARM9xx parts, an ARM9TDMI core; and ARM9xxE, parts an ARM9E core.
- Find the relevant instruction class in the left-hand column of the table. The class "ALU" is shorthand for all of the arithmetic and logical instructions:

TABLE B3.1 Standard cycle abbreviations.

Abbreviation	Meaning
B	The number of busy-wait cycles issued by a coprocessor. This depends on the coprocessor design.
M	The number of multiplier iteration cycles. This depends on the value in register *Rs*. Each implementation section contains a table showing how to calculate *M* from *Rs* for that implementation.
N	The number of words to transfer in a load or store multiple. This includes *pc* if it is in the register list. *N* must be at least one.

ADD, ADC, SUB, RSB, SBC, RSC, AND, ORR, BIC, EOR, CMP, CMN, TEQ, TST, MOV, MVN, CLZ.

- Read the value in the "Cycles" column. This is the number of cycles the instruction usually takes, assuming the instruction passes its condition codes and there are no interactions with other instructions. The cycle count may depend on one of the abbreviations in Table B3.1.
- If the "Notes" column contains any notes of the form $+k$ *if condition*, then add on to your cycle count all the additions that apply.
- Look for interlock conditions that will cause the processor to stall. These are occasions where an instruction attempts to use the result of a previous instruction before it is ready. Unless otherwise stated, input registers are required on the first cycle of the instruction and output results are available at the end of the last cycle of the instruction. However, implementations with multiple execute stage pipelines can require input operands early and produce output operands later. Table B3.2 defines the statements we use in the "Notes" sections to describe this.
- If your instruction fails its condition codes, then it is not executed. Usually this costs one cycle. However, on some implementations, instructions may cost multiple cycles even if they are not executed. Look for a note of the form "[*k* cycles if not executed]."

TABLE B3.2 Pipeline behavior statements.

Statement	Meaning
Rd is not available for *k* cycles.	The result register *Rd* of the instruction is not available as the input to another instruction for *k* cycles after the end of the instruction. If you attempt to use *Rd* earlier, then the core will stall until the *k* cycles have elapsed.
Rn is required *k* cycles early.	The input register *Rn* of the instruction must be available *k* cycles before the start of the instruction. If it was the result of a later operation, then the core will stall until this condition is met.
Rn is not required until the *k*th cycle.	The input register *Rn* is not read on the first cycle of the instruction. Instead it is read on the *k*th cycle of the instruction. Therefore the core will not stall if *Rn* is available by this point.
You cannot start a type *X* instruction for *k* cycles.	The instruction uses a resource also used by type X instructions. Moreover the instruction continues to use this resource for *k* cycles after the last cycle of the instruction. If you attempt to execute a type *X* instruction before *k* cycles have elapsed, then the core will stall until *k* cycles have elapsed.

B3.2 ARM7TDMI Instruction Cycle Timings

The ARM7TDMI core is based on a three-stage pipeline with a single execute stage. The number of cycles an instruction takes does not usually depend on preceding or following instructions. The multiplier circuit uses a 32-bit by 8-bit multiplier array with early termination. The number of multiply iteration cycles M depends on the value of register Rs according to Table B3.3. Table B3.4 gives the ARM7TDMI instruction cycle timings.

TABLE B3.3 ARM7TDMI multiplier early termination.

M	Rs range (use the first applicable range)	Rs bitmap s = sign bit x = wildcard-bit			
1	$-2^8 \le \times < 2^8$	ssssssss	ssssssss	ssssssss	xxxxxxxx
2	$-2^{16} \le \times < 2^{16}$	ssssssss	ssssssss	xxxxxxxx	xxxxxxxx
3	$-2^{24} \le \times < 2^{24}$	ssssssss	xxxxxxxx	xxxxxxxx	xxxxxxxx
4	remaining $\times$	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx

TABLE B3.4 ARM7TDMI (ARMv4T) instruction cycle timings.

Instruction class	Cycles	Notes
ALU	1	+1 if you use a register-specified shift *Rs*.
		+ 2 if *Rd* is *pc*.
B, BL, BX	3	
CDP	1 + *B*	
LDC	1 + *B* + *N*	
LDR/B/H/SB/SH	3	+2 if *Rd* is *pc*.
LDM	2 + *N*	+2 if *pc* is in the register list.
MCR	2 + *B*	
MLA	2 + *M*	
xMLAL	3 + *M*	
MRC	3 + *B*	
MRS, MSR	1	
MUL	1 + *M*	
xMULL	2 + *M*	
STC	1 + *B* + *N*	
STR/B/H	2	
STM	1 + *N*	
SWI	3	
SWP/B	4	

B3.3 ARM9TDMI Instruction Cycle Timings

The ARM9TDMI core is based on a five-stage pipeline with a single execute stage and two memory fetch stages. There is usually a one- or two-cycle delay following a load instruction before you can use the data. Using data immediately after a load will add interlock cycles. The multiplier circuit uses a 32-bit by 8-bit multiplier array with early termination. The number of multiply iteration cycles *M* depends on the value of register *Rs* according to Table B3.5. Table B3.6 gives the ARM9TDMI instruction cycle timings.

TABLE B3.5 ARM9TDMI multiplier early termination.

M	*Rs* range (use the first applicable range)	Rs bitmap s = sign bit x = wildcard-bit			
1	$-2^8 \leq \times < 2^8$	ssssssss	ssssssss	ssssssss	xxxxxxxx
2	$-2^{16} \leq \times < 2^{16}$	ssssssss	ssssssss	xxxxxxxx	xxxxxxxx
3	$-2^{24} \leq \times < 2^{24}$	ssssssss	xxxxxxxx	xxxxxxxx	xxxxxxxx
4	remaining $\times$	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx

TABLE B3.6 ARM9TDMI (ARMv4T) instruction cycle timings.

Instruction class	Cycles	Notes
ALU	1	+1 if a register-specified shift *Rs* is used.
		+2 if *Rd* is *pc*.
B, BL, BX	3	
CDP	1 + *B*	
LDC	*B* + *N*	
LDRB/H/SB/SH	1	*Rd* is not available for two cycles.
LDR *Rd* not *pc*	1	*Rd* is not available for one cycle.
LDR *Rd* is *pc*	5	
LDM not loading *pc*	*N*	+1 if *N* = 1or the last loaded register used in the next cycle.
LDM loading *pc*	*N* + 4	
MCR	1 + *B*	
MRC *Rd* not *pc*	1 + *B*	*Rd* is not available for one cycle.
MRC *Rd* is *pc*	3 + *B*	
MRS	1	
MSR	1	+2 if any of the *csx* fields are updated.
MUL, MLA	2 + *M*	
xMULL, xMLAL	3 + *M*	
STC	*B* + *N*	
STR/B/H	1	
STM	*N*	+1 if *N* = 1.
SWI	3	
SWP/B	2	*Rd* is not available for one cycle.

B3.4 StrongARM1 Instruction Cycle Timings

The StrongARM1 core is based on a five-stage pipeline. There is usually a one-cycle delay following a load or multiply instruction before you can use the data. Additionally, there is often a one-cycle delay if you start a new multiply instruction immediately following a previous multiply instruction. The multiplier circuit uses a 32-bit by 12-bit multiplier array with early termination. The number of multiply iteration cycles *M* depends on the value of register *Rs* according to Table B3.7. Table B3.8 gives the StrongARM1 instruction cycle timings.

TABLE B3.7 StrongARM1 multiplier early termination.

M	*Rs* range (use the first applicable range)	Rs bitmap s = sign bit x = wildcard bit			
1	$-2^{11} \leq \times < 2^{11}$	ssssssss	ssssssss	sssssxxx	xxxxxxxx
2	$-2^{23} \leq \times < 2^{23}$	sssssssss	xxxxxxx	xxxxxxxx	xxxxxxxx
3	remaining $\times$	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx

TABLE B3.8 StrongARM1 (ARMv4) instruction cycle timings.

Instruction class	Cycles	Notes
ALU	1	+1 if a register-specified shift is used [even if the instruction is not executed].
		+2 if *Rd* is *pc* [only if executed].
B, BL	2	
LDR/B/H *Rd* not *pc*	1	*Rd* is not available for one cycle.
LDRSB/SH *Rd* not *pc*	2	*Rd* is not available for one cycle.
LDR *Rd* is *pc*	4	
LDM $N = 1$, not *pc*	2	[2 cycles if not executed.]
LDM $N > 1$, not *pc*	N	The last loaded value is not available for one cycle.
		[N cycles if not executed.]
LDM loading *pc*	$N + 3$	[max(N ,2) if not executed.]
MRS	1	*Rd* is not available for one cycle.
MSR to *cpsr*	3	+1 if any of the *csx* fields are updated.
MSR to *spsr*	1	
MUL, MLA	*M*	*Rd* is not available for one cycle. You cannot start another multiply on the next cycle.
MULS, MLAS	4	
xMULL, xMLAL	$1 + M$	*RdHi* is not available for one cycle. You cannot start a multiply on the next cycle. [2 if instruction not executed.]
xMULLS, xMLALS	5	[2 if instruction not executed.]
STR/B/H	1	
STM	*N*	+1 if $N = 1$.
		[Same number of cycles if not executed.]
SWP/B	2	[2 if instruction not executed.]

B3.5 ARM9E Instruction Cycle Timings

The ARM9E core is based on a five-stage pipeline. There is usually a one- or two-cycle delay following a load or multiply instruction before you can use the data. The multiplier circuit uses a 32-bit by 16-bit multiplier array. The multiplier does not terminate early. Table B3.9 gives the ARM9E instruction cycle timings.

TABLE B3.9 ARM9Erev2 (ARMv5TE) instruction cycle timings.

Instruction Class	Cycles	Notes
ALU *Rd* not *pc*	1	+1 if a register-specified shift is used.
ALU *Rd* is *pc*	3	+1 if the operation is logical or any shift is used.
B, BL, BX, BLX	3	
CDP	1 + *B*	
LDC	*B* + *N*	
LDRB/H/SB/SH	1	*Rd* is not available for two cycles.
		+1 if the load offset is shifted.
LDR *Rd* not *pc*	1	*Rd* is not available for one cycle.
		+1 if the load offset is shifted.
LDR *Rd* is *pc*	5	+1 if the load offset is shifted.
LDRD	2	*R*(*d* + 1) is not available for one cycle.
LDM not loading *pc*	*N*	+ 1 if *N* = 1 or the last loaded register used in the next cycle.
LDM loading *pc*	*N* + 4	
MCR	1 + *B*	
MCRR	2 + *B*	
MRC *Rd* not *pc*	1 + *B*	*Rd* is not available for one cycle.
MRC *Rd* is *pc*	4 + *B*	
MRRC	2 + *B*	*Rn* is not available for one cycle.
MRS	2	
MSR	1	+2 if any of the *csx* fields are updated.
MUL, MLA	2	*Rd* is not available for one cycle, except as an accumulator input for a multiply accumulate.
MULS, MLAS	4	
xMULL, xMLAL	3	*Rd Hi* is not available for one cycle, except as an accumulator input for a multiply accumulate.
xMULLS, xMLALS	5	
PLD	1	
QxADD, QxSUB	1	*Rd* is not available for one cycle.
SMULxy, SMLAxy, SMULWx, SMLAWx	1	*Rd* is not available for one cycle, except as an accumulator input for a multiply accumulate.
SMLALxy	2	*RdHi* is not available for one cycle, except as an accumulator input for a multiply accumulate.
STC	*B* + *N*	
STR/B/H	1	+1 if a shifted offset is used.
STRD	2	
STM	*N*	+1 if *N* = 1.
SWI	3	
SWP/B	2	*Rd* is not available for one cycle.

B3.6 ARM10E Instruction Cycle Timings

The ARM10E core is based on a five-stage pipeline with branch prediction. There is usually a one-cycle delay following a load or multiply instruction before you can use the data. The ARM10E uses a 64-bit-wide data bus, so load and store instructions can transfer 64 bits per cycle. The multiplier does not use early termination. Table B3.10 gives the ARM10E instruction cycle timings.

TABLE B3.10 ARM10E (ARMv5TE) instruction cycle timings.

Instruction class	Cycles	Notes
ALU	1	+1 if a register-specified shift, or RRX, is used.
		+4 if *Rd* is *pc*.
		An exception is MOV *pc*, *Rn*. This takes 4 cycles.
B, BX	0-2	+4 if the branch is mispredicted.
BL, BLX	1-2	+4 if the branch is mispredicted.
CDP	1	
LDC	1	Data availability depends on the coprocessor.
LDR/B/H/SB/SH	1	*Rd* is not available for one cycle.
Rd not *pc*		+1 if the addressing mode is register preindexed with the option of a (constant) shift.
LDR *Rd* is *pc*	6	+1 if the offset (pre- or postindex) is a shifted register.
		[2 cycles if not executed].
LDRD	1	*Rd* and *R*(d + 1) are not available for one cycle.
LDM not loading *pc*	1	The first data item is not available for one cycle. Once the address is 8-byte aligned, data items are loaded in pairs, at two per cycle. Therefore the *k*th data item will be available after (k + *a* + 1)/2 cycles, where *a* is bit 2 of the base address. You cannot start another load or store until this one has finished.
LDM loading *pc*	*L* + 6	L = (N + a) / 2, and *a* is bit 2 of the base address.
MCR, MCCR	1	
MR{R}C *Rd* not *pc*	1	*Rd* is not available for one cycle.
MRC *Rd* is *pc*	2	
MRS	1	
MSR to *cpsr*	1	+3 if any of the *csx* fields are updated.
MSR to *spsr*	3	[2 if the instruction is not executed.]
MUL, MLA	2	*Rd* is not available for one cycle.

TABLE B3.10 ARM10E (ARMv5TE) instruction cycle timings. (*Continued*.)

Instruction class	Cycles	Notes
MULS, MLAS	4	
xMULL, xMLAL	3	*RdHi* is not available for one cycle.
xMULLS, xMLALS	5	
PLD	1	+1 if a shifted register offset is used.
QxADD, QxSUB	1	*Rd* is not available for one cycle.
SMULxy, SMULWx	1	*Rd* is not available for one cycle.
SMLAxy, SMLAWx	2	
SMLALxy	2	*RdHi* is not available for one cycle.
STC	1	
STR/B/H	1	+1 if a preindexed shifted register offset is used.
STRD	1	
STM	1	Registers are stored two per cycle once the address is 8-byte aligned. You cannot write a register in the register list until its value has been stored. You cannot start another load or store until this one is complete.
SWP/B	2	

B3.7 Intel XScale Instruction Cycle Timings

The Intel XScale is based on a seven-stage pipeline. There is usually a two-cycle delay following a load instruction before you can use the data. Multiply instructions usually issue in a fixed number of cycles, but then the result is not available for a variable number of cycles, depending on the value of *Rs*. Table B3.11 shows how the number of multiply iteration cycles *M* depends on the value of Rs. Table B3.12 gives the Intel XScale instruction cycle timings.

TABLE B3.11 Intel XScale multiplier early termination.

M	*Rs* range (use the first applicable range)	Rs bitmap s = sign bit x = wildcard bit			
1	$-2^{15} \le x < 2^{15}$	ssssssss	ssssssss	sxxxxxxx	xxxxxxxx
2	$-2^{27} \le x < 2^{27}$	sssssxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
3	remaining x	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx

TABLE B3.12 Intel XScale (ARMv5TE) instruction cycle timings.

Instruction class	Cycles	Notes
ALU	1	+1 if a register-specified shift, or RRX, is used.
		+4 if *Rd* is *pc*.
B, BL	1	+4 if the branch is mispredicted.
BX, BLX	5	[1 cycle if not executed.]
LDR/B/H/SB/SH *Rd* not *pc*	1	*Rd* is not available for two cycles.
LDR *Rd* is *pc*	8	[2 cycles if not executed.]
LDRD	1	*Rd* is not available for two cycles. *R*(d + 1) is not available for three cycles.
		+1 if *Rd* is *r 12*.
LDM not loading *pc*	2 + *N*	The last value loaded is not available for two cycles. The value previous to that is not available for one cycle.
LDM loading *pc*	7 + *N*	Increase to 10 cycles if $N < 3$.
		[3 + *N* cycles if not executed.]
MCR to copro 15	2	
MRC from copro 15	4	
MRS	1	*Rd* is not available for one cycle.
MSR	2	+4 if any of the *csx* fields are updated.
MUL, MLA	1	*Rd* is not available for *M* cycles. You cannot start another multiply in the next *M* −1 cycles.
MULS, MLAS	1 + M	
xMULL	1	*RdHi* is not available for *M* +1 cycles. *RdLo* is not available for *M* cycles. You cannot start another multiply in the next *M* cycles.
xMLAL	2	*RdHi* is not available for *M* cycles. *RdLo* is not available for *M* −1 cycles. You cannot start another multiply in the next *M* −1 cycles.
xMULLS, xMLALS	2 + *M*	
PLD	1	
QxADD, QxSUB	1	*Rd* is not available for one cycle.
SMULxy, SMLAxy	1	*Rd* is not available for one cycle.
SMULWx, SMLAWx	1	*Rd* is not available for two cycles. You cannot start another multiply for one cycle.
SMLALxy	2	*RdHi* is not available for one cycle.
STR/B/H	1	
STRD	2	
STM	2 + *N*	
SWI	6	
SWP/B	5	

B3.8 ARM11 Cycle Timings

The ARM11 core uses an eight-stage pipeline with three execute stages. There is usually a two-cycle delay following a load instruction before you can use the data. Some operations such as shift, multiply, and address calculations require their input registers a cycle early.

For example, the following code sequence will stall the core for three cycles because the result of the load is not available for two cycles, and the input to the shift is required one cycle early:

```
LDR  r0, [r1]      ; r0 not available for 2 cycles
MOV  r2, r0, ASR#3 ; r0 required one cycle early
```

The ARM11 core has a separate address generation unit that can calculate simple addresses in one cycle. More complicated addresses take two cycles. Table B3.13 defines the number of address calculation cycles *A* for each addressing mode.

TABLE B3.13 ARM11 address calculation cycles.

A	Addressing modes
1	[*Rn, # <signed-offset>*]{!}
	[*Rn*], *# <signed-offset>*
	[*Rn, Rm* {, *LSL #2*}]{!}
	[*Rn*], *Rm* {, *LSL #2*}
2	[*Rn, - Rm*] {!}
	[*Rn*], *- Rm*
	[*Rn*, {-} *<shifted_Rm>*]{!} where shift is not *LSL #0 or LSL #2*
	[*Rn*], {-} *<shifted_Rm>* where shift is not *LSL #0 or LSL #2*

The ARM11 core uses prediction to minimize the number of cycles caused by a change in program flow. To enable prediction, set bit 11 of CP15 register c1. There are three branch predictors.

A *static predictor* predicts relative branches that are not recorded in the branch prediction cache. This is the case the first time the processor sees a given branch. The static predictor predicts forward conditional branches as taken and backward conditional branches as not taken.

A *dynamic predictor* predicts relative branches that are recorded in the branch prediction cache. The branch prediction cache has 128 entries based on the branch instruction address. Each cache entry predicts the branch destination and if the branch is taken. A cache entry has four states: strongly not taken, weakly not taken, weakly taken, strongly taken. Each time the branch is taken, the state moves one to the right in this list (if it can), and each time the branch is not taken, the state moves one to the left in this list (if it can).

A *return stack* predicts unconditional subroutine return instructions. The stack has three entries storing the return address from the three deepest BL, BLX subroutine calls.

Table B3.14 gives the ARM11 instruction cycle timings.

TABLE B3.14 ARM11 (ARMv6) instruction cycle timings.

Instruction class	Cycles	Notes
ALU operations except a MOV to *pc* (for MOV to *pc*, see BX)	1	*Rm* is required one cycle early if shifted by a constant shift. +1 if a register-specified shift is used. In this case *Rs* is required one cycle early and *Rn* is not required until the second cycle.
		+6 if *Rd* is *pc*.
B <*immed*> BL <*immed*> BLX <*immed*>	1	Assumes successful dynamic prediction. Some dynamically predicted branches may be folded, to be zero cycles. +3 for successful static prediction. +4 for unsuccessful static or dynamic prediction. In this case the flags are required two cycles early.
BX *lr*	4	+1 if unconditional and return stack is empty.
MOV *pc*, *lr*		+3 if unconditional and return stack mispredicts.
BX *Rm* (not *lr*)		+1 if conditional. In this case the flags are required two cycles early.
BLX *Rm*	5	If no shift on MOV and conditional, the flags are required two cycles early.
MOV *pc*, *Rm* (not *lr*)		+1 if a constant shift is used for MOV. In this case *Rm* is required one cycle early. If conditional, then the flags are required one cycle early. +2 if a register-specified shift is used for MOV. In this case *Rs* is required one cycle early, and *Rn* is not used until the second cycle.
CPS	1	+1 if a mode change occurs.

TABLE B3.14 ARM11 (ARMv6) instruction cycle timings. (*Continued*.)

Instruction class	Cycles	Notes
LDR/B/H/SB/SH/D *Rd* not *pc*	A	*Rd* is not available for two cycles. *R*(*d* + 1) is not available for two cycles for LDRD.
		If the load is potentially unaligned (base or offset unaligned), then you cannot start another memory access on the next cycle.
		If the load is unaligned, then *Rd* is not available for three cycles for LDR/H/SH. For LDRD *Rd* is not available for two cycles and *R*(*d* + 1) for three cycles.
LDR *pc*, [*sp*, # *off*] {!}	4	+4 if unconditional and return stack is empty.
LDR *pc*, [*sp*], # *off*		+5 if unconditional and return stack mispredicts
		+4 if conditional.
LDR *pc* not using a constant stack offset	A + 7	
LDM not loading *pc*	1	You cannot start another memory access for the next (*N* + *a* − 1)/2 cycles, where *a* is bit 2 of the address.
		The *k*th register in the list not available for (*k* + *a* + 3)/2 cycles.
LDM *sp*{!} loading *pc*	4	+5 if conditional or return stack empty or return stack mispredicts. You cannot start another memory access for (*N* + *a*)/2 cycles. The *k*th register in the list not available for (*k* + *a* + 5)/2 cycles.
LDM loading *pc* not from the stack	8	You cannot start another memory access for (N + a)/2
		cycles. The *k*th register in the list not available for (*k* + *a* + 5)/2 cycles.
MCR/MCRR	1	This counts as a memory access.
MRC/MRRC	1	This counts as a memory access. The result registers are not available for two cycles.
MRS	1	
MSR to *cpsr*	1	+3 if any of the *csx* fields are updated.
MSR to *spsr*	5	
MUL, MLA	2	*Rd* is not available for two cycles, except as an accumulator input for another multiply accumulate when it is not available for one cycle.
		Rm and *Rs* are required one cycle early. *Rn* is not required until the second cycle for MLA.
MULS, MLAS	5	*Rm* and *Rs* are required one cycle early. *Rn* is not required until the second cycle for MLAS.
xMULL, xMLAL	3	*RdLo* is not available for one cycle. *RdHi* is not available for two cycles. Reduce these latencies by one if these registers are used as accumulator inputs for another multiply accumulate.

TABLE B3.14 ARM11 (ARMv6) instruction cycle timings. (*Continued*.)

Instruction class	Cycles	Notes
		Rm and *Rs* are required one cycle early. *RdLo* is not required until the second cycle for MLAL.
xMULLS, xMLALS	6	*Rm* and *Rs* are required one cycle early. *RdLo* is not required until the second cycle for MLAL.
PKHBT, PKHTB	1	*Rm* is required one cycle early.
PLD	A	
QxADD, QxSUB	1	*Rd* is not available for one cycle. *Rn* is required one cycle early for QDADD and QDSUB.
REV, REV16, REVSH	1	*Rm* is required one cycle early.
{S,SH,Q,U,UH,UQ} ADD16, ADDSUBX, SUBADDX, SUB16, ADD8, SUB8	1	*Rd* is not available for one cycle for saturating or halving operations (SH, Q, UH, UQ prefix).
SEL	1	
SETEND	1	
SMULxy, SMLAxy, SMULWy, SMLAWy SMUAD, SMLAD, SMUSD, SMLSD	1	*Rd* is not available for two cycles, except as an accumulator input for another multiply accumulate when it is not available for one cycle. *Rm* and *Rs* are required one cycle early.
SMLALxy, SMLALD{X}, SMLSLD{X}	2	*RdLo* is not available for one cycle. *RdHi* is not available for two cycles. Reduce these latencies by one if these registers are used as accumulator inputs for another multiply accumulate.
		Rm and *Rs* are required one cycle early. *RdHi* is not required until the second cycle.
SMMUL{R}, SMMLA{R}, SMMLS{R}	2	*Rd* is not available for two cycles, except as an accumulator input for another multiply accumulate when it is not available for one cycle.
		Rm and *Rs* are required one cycle early. *Rn* is not required until the second cycle.
SSAT, USAT, SSAT16, USAT16	1	*Rd* is not available for one cycle. *Rm* is required one cycle early for SSAT and USAT.
STR/B/H/D	A	If the store is potentially unaligned (base or offset unaligned), then you cannot start a memory access on the next cycle.
		For STRD you cannot start another instruction that writes to R (d + 1) for one cycle.
STM	1	You cannot start another memory access for the next (N + a − 1)/2 cycles, where *a* is bit 2 of the address.
		You cannot start an instruction that writes to the *k*th register in the list for *k*/2 cycles.
SWI	8	
SWP/B	2	*Rd* is not available for one cycle.
SXT, UXT	1	*Rm* is required one cycle early.

TABLE B3.14 ARM11 (ARMv6) instruction cycle timings. (*Continued*.)

Instruction class	Cycles	Notes
UMAAL	3	*RdLo* is not available for one cycle. *RdHi* is not available for two cycles. These latencies are reduced by one for another accumulate.
		Rm and *Rs* are required one cycle early. *RdLo* is not required until the second cycle.
USAD8, USADA8	1	*Rd* is not available for two cycles, with the exception that the result of USAD8 is available as the accumulator for USADA8 after one cycle.
		Rm and *Rs* are required one cycle early.