Application Note 18

The ARM6 Family Bus Interface

Document Number: ARM DAI 0018B
Issued: December 1994

ARM
Advanced RISC Machines

Proprietary Notice

Change Log

Issue	Date	By	Change
A (1.0)	Aug. 93		Internal Release
(1.1)	Sep. 93		Initial release
B	Dec. 94	AW	Reformatted into Framemaker

1 Introduction

This application note describes the bus interface for the ARM6 family of microprocessors. Firstly, the general ARM bus interface signals and the minor differences between the products are explained. The ARM6 bus cycles are then described and finally, an example is given showing how code moves up the instruction pipeline, and how the bus behaves as a result.

The ARM6 bus interface is implemented in the ARM6, ARM60 and ARM61 products. There are two variations of the basic interface:

- the ARM7 bus interface as implemented in the ARM7D, ARM7DM and ARM70DM
- the ARM610 bus interface as implemented in the ARM600, ARM610, ARM700 and ARM710.

Both of these variations are covered in separate documents.

2 ARM6 Bus Interface External Signals

2.1 Processor Clocks

The ARM times all its external activity from a single clock, **MCLK**. This is typically a square wave but, due to the static nature of the ARM core, either phase of the clock may be stretched indefinitely. This feature may be exploited by system controllers in order to extend memory accesses to slow peripherals. There is one control signal associated with the clock, **nWAIT**. Inside the ARM, **nWAIT** is ANDed with **MCLK** to make the core's clock. This facility allows cycle stretching in a system with a free running **MCLK**.

This internal version of **MCLK** (gated with **nWAIT**) is divided into two non-overlapping clocks: phase 1 (ph1) when the clock is low, and phase 2 (ph2) when the clock is high. This is shown in *Figure 2-1: The ARM Clock* on page 2. The ARM times all its activity from these internal clock signals. In the ARM documentation, when a signal is said to change in the high phase of **MCLK**, it is assumed that **nWAIT** is also high (ie. when ph2 is high).

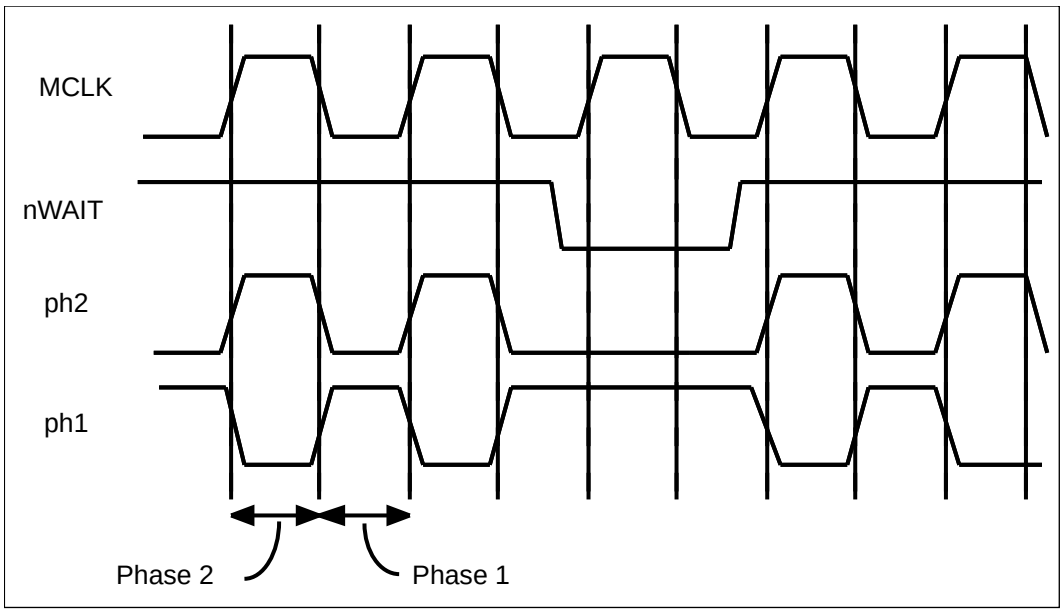

Figure 2-1: The ARM Clock

2.2 Data and Address Buses

With the exception of the ARM6 macrocell, all ARM processors have a 32 bit bidirectional data bus (**D[31:0]**) which is used for instruction fetches and data loads and stores. The ARM6 macrocell has 2 unidirectional buses, one for data in (**DATA[31:0]**), and the other for data out (**DOUT[31:0]**). The ARM can deal with both byte and word (32 bit) quantities. Instructions are always single words. When reading a byte quantity over the data bus, ARM expects it to be presented on the appropriate 8 data lines and ignores the other 24 lines. When writing a byte quantity, the byte appears replicated 4 times over the 32 lines and it is up to the memory system to choose which copy it uses.

The ARM6 core has an address bus that is 32 bits wide (**A[31:0]**), allowing up to 4 gigabytes to be addressed. ARM memory is byte addressed, though most memory systems will be 32 bits wide. The smallest directly addressable unit will be a 32 bit word. For this reason it is often convenient to view **A[31:2]** as a word memory address and **A[1:0]** as a byte selector within that word.

With the exception of the ARM61, all the processors in the family have all 32 address lines externally available. The ARM61 only has 26 address lines (**A[25:0]**) available, allowing up to 64 Megabytes to be addressed.

2.3 Bus Control Signals

There are three main control signals associated with the ARM's buses: Address Latch Enable (**ALE**), Data Bus Enable (**DBE**) and Address Bus Enable (**ABE**). The ARM6 core does not include tristate drivers on the address bus, so there is no **ABE** signal on the ARM6 macrocell. All other processors in the family have the three control signals.

The action of the three control signals is shown in *Figure 2-2: The ARM Bus Control Signals.* Note that the use of **DBE** shown in the figure is unlikely in a real system. It is more likely that **DBE** would be asserted for the whole cycle. A description of each control signal is given below.

Address Latch Enable (ALE)

ALE is used to control transparent latches on the address outputs of the ARM. When **ALE** is high, the latches are transparent and taking **ALE** low will latch the current value. This mechanism can be used to de-pipeline the address when interfacing to memory systems such as ROMs and Static RAMs which require a stable address for the entire cycle.

Data Bus Enable (DBE)

DBE enables the data bus outputs when it is high. When it is low, these outputs are put into a high impedance state. The outputs are normally only enabled during write cycles and so the **DBE** signal will only be needed in systems which require the data bus for DMA or similar shared bus operations.

Address Bus Enable (ABE)

ABE enables the address bus outputs when it is high. When it is low, these outputs are put into a high impedance state. Note that the ARM6 macrocell does not include tristate drivers on the address bus, and so there is no **ABE** signal.

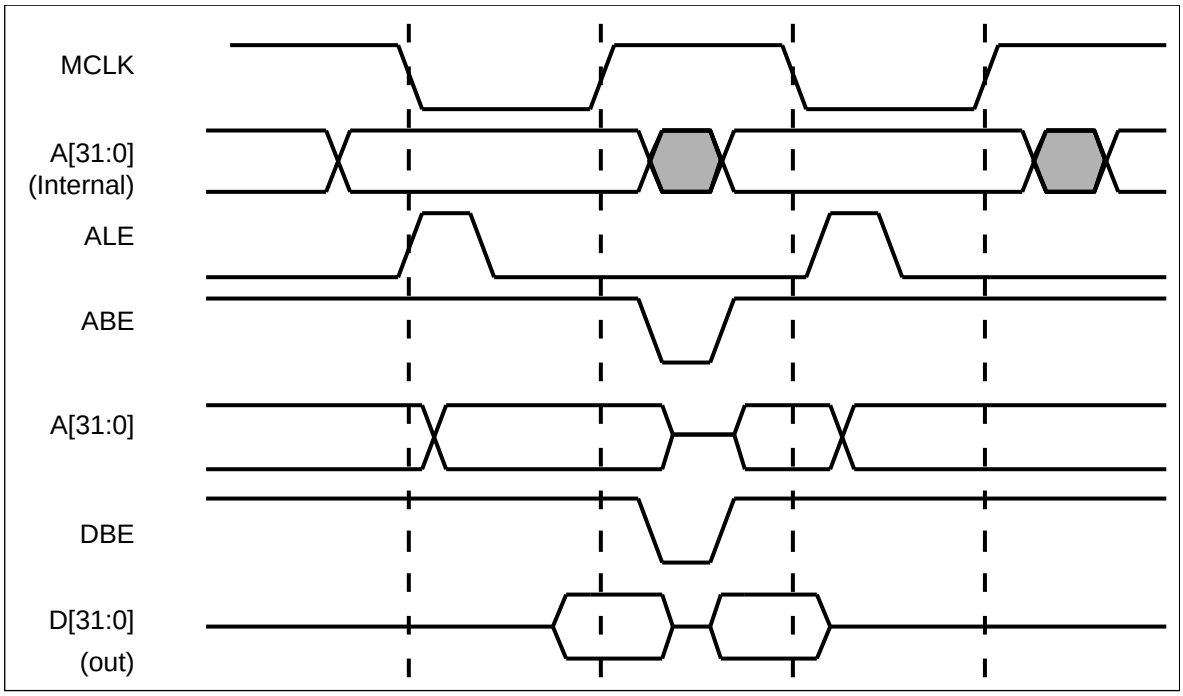

Figure 2-2: The ARM Bus Control Signals

2.4 Memory Control Signals

These signals control access to the memory system. Most signals become valid just before the cycle to which they refer, and remain valid during phase 1 of that cycle. Two signals (**SEQ** and **nMREQ**) have different behaviour, because of the pipeline mechanism, and they are valid in the cycle before the one to which they refer. The pipelining of the bus control signals is described in section 3. The memory signals are shown in *Figure 2-3: The ARM Memory Control Signals* on page 5.

Not Memory Request (nMREQ).

This active-low signal indicates that a memory access will be made in the following cycle. It becomes valid during phase 1 and remains valid throughout phase 2 of the cycle, before the one to which it refers.

Sequential (SEQ).

This active-high signal indicates that the address used in the following cycle is either the same as the last memory address, or is 4 greater (ie. the next word address). It becomes valid during phase 1 and remains valid throughout phase 2 of the cycle before the one to which it refers.

These two signals (**nMREQ** and **SEQ**) indicate burst activity one-cycle in advance.

Address Bus (A[31:0]).

These signals provide the memory address in cycles which access memory. They become valid during phase 2 of the cycle preceding the memory access, and remain valid through phase 1 of the actual memory cycle. This behaviour may be modified by the **ALE** latch.

Not Byte/Word (nBW).

This signal indicates whether a memory cycle will transfer a byte (**nBW** low) or word (**nBW** high) quantity. It becomes valid during phase 2 of the cycle preceding the memory access and remains valid through phase 1 of the actual memory cycle. This signal (**nBW**) allows the system designer to interface 8-bit peripherals and control byte accesses to memory.

Read/Write (nRW).

This signal differentiates between memory cycle reads (**nRW** low) and writes (**nRW** high). It becomes valid during phase 2 of the cycle preceding the memory access and remains valid through phase 1 of the actual memory cycle.

Data Bus (D[31:0]).

These signals are inputs during read cycles and outputs during write cycles. At other times they are high-impedance. During read cycles, data must be valid at the end of phase 2. During write cycles, valid data appears during phase 1 and remains valid throughout phase 2. During byte write cycles the 8-bit data is broadcast across all 4 bytes to simplify 8-bit peripheral connection.

Lock (LOCK).

This signal indicates that a swap instruction (SWP) is in progress and that the two memory cycles used in this instruction should be regarded as indivisible. It becomes valid during phase 2 of the cycle preceding the first memory access and remains valid through phase 1 of the second memory cycle. This signal provides information to bus arbitration logic to indicate 2 bus cycles should not be interrupted to maintain “semaphore” integrity.

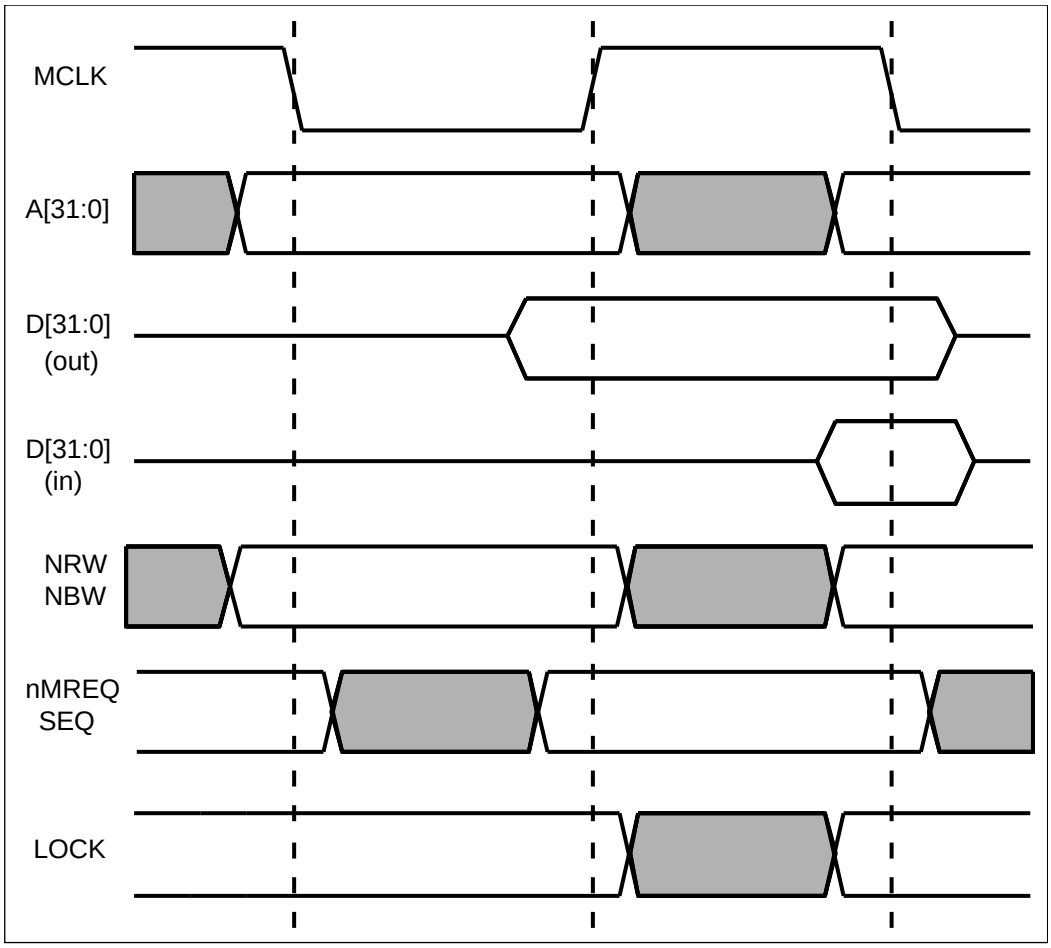

Figure 2-3: The ARM Memory Control Signals

2.5 Memory Management Signals

In addition to the signals described above, ARM processors also have some signals which are of use in systems requiring memory management. They are shown in ◔*Figure 2-4: The ARM Memory Management Signals*.

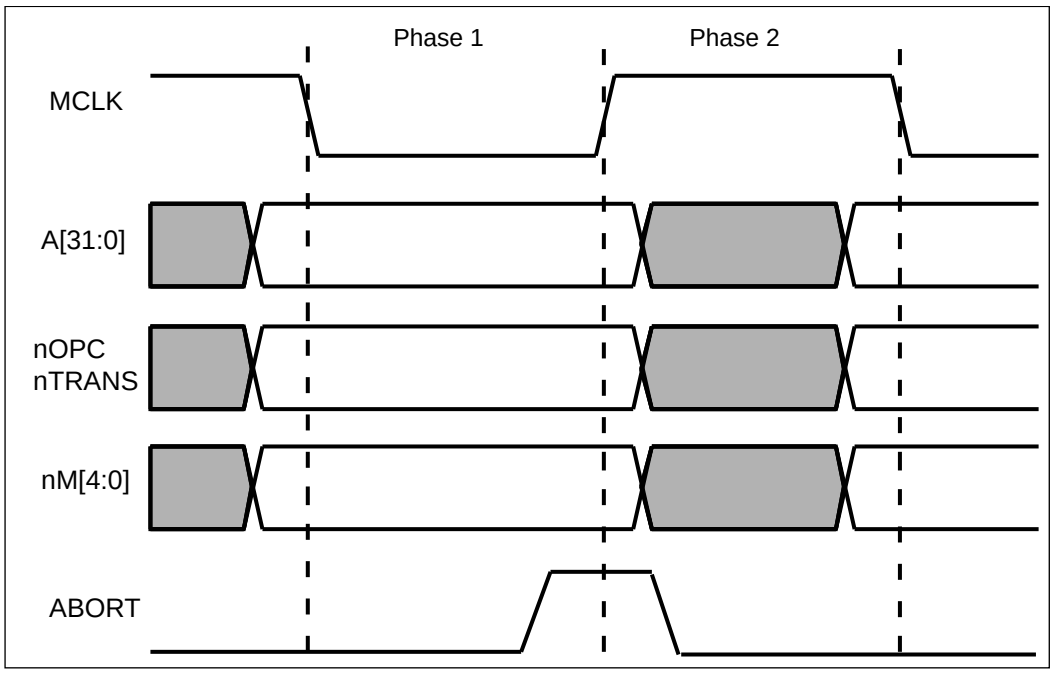

Figure 2-4: The ARM Memory Management Signals

Not Opcode Fetch (nOPC).

This active-low signal indicates that the processor is fetching an instruction from memory. It becomes valid during phase 2 of the cycle preceding the memory access to which it refers, and remains valid through phase 1 of that memory cycle (ie. like the address). **nOPC** may be used to distinguish instruction fetches from data read cycles. This may be used to check for executable permission. **nOPC** may be used by coprocessors for pipeline following.

Not Translate (nTRANS).

This active-low signal indicates that the processor requires the memory manager to perform a translation on the current address. It is driven low during memory accesses while the processor is in User Mode, and becomes valid during phase 2 of the cycle preceding the memory access to which it refers (ie. like the address). **nTRANS** remains valid through phase 1 of the memory cycle.

Processor Mode (nM[4:0]).

These five signals encode the ARM's current mode of operation, and hence the register set in use. They become valid during phase 2 of the cycle preceding the memory access to which they refer, and remain valid through phase 1 of that memory cycle. Note, these signals are not available on the ARM60 and ARM61 processors. For an explanation of the encoding of these signals, please refer to the relevant data sheet.

Memory Abort (ABORT).

This active-high input is used to indicate that a requested access to memory is not allowed. It will normally be generated by the memory management logic. This signal must become valid during phase 1 of the memory cycle, and be held into phase 2. For future compatibility, **ABORT** should be held throughout phase 2 of the memory cycle.

The action taken by the processor after an abort depends on the state of the configuration input, **LATEABT**.

The action of the processor after an aborted memory access is as follows: for an abort on an instruction fetch, if the instruction reaches the execute stage of the pipeline then a branch to the Prefetch-Abort vector is taken. If an abort occurs on a data load, then the destination register of the load is not updated. If this was part of a load multiple (LDM), then all further register updates are also prevented, although the LDM on the bus will continue normally. If base write-back has been set, then processors configured for late-aborts will allow the base write-back. Processors configured for early-aborts will prevent base write back. On completion of the instruction, the Data-Abort vector is taken.

Similarly for aborts on data stores, ARM6 processors configured for early-abort will prevent base write-back. In all systems, it is up to the external memory controller to stop writes on aborted addresses. After an abort part way through a store multiple (STM), the ARM will complete the instruction, continuing the writes on the bus. On completion of the store instruction, the Data-Abort vector is taken.

For more information on the action of the ARM in the case of an abort, please refer to the relevant data sheet. Note that all future ARM processors will implement late-abort timing, where the **ABORT** input must be set up and held around the falling edge of **MCLK** (ie. at the same time as the data). The abort configuration of ARM6 allows users to write ARM7-compatible abort handlers before upgrading the hardware to an ARM7 based processor.

3 The ARM6 Bus Cycles

The ARM6 Bus Interface performs 4 basic types of cycle. These are:

- non-sequential (N) cycles
- sequential (S) cycles
- internal, or idle, (I) cycles
- coprocessor register transfers (CPRT) cycles.

In addition to these there is a fifth special case, the merged IS (MIS) cycle.

The type of memory cycle that the ARM will perform is encoded by the signals **nMREQ** and **SEQ**. To aid memory system performance, these two signals are pipelined ahead by one cycle. Also to aid performance, the address for the memory cycle is pipelined ahead by a phase. This pipelining is shown in ➲*Figure 3-5: The ARM Memory Pipeline*.

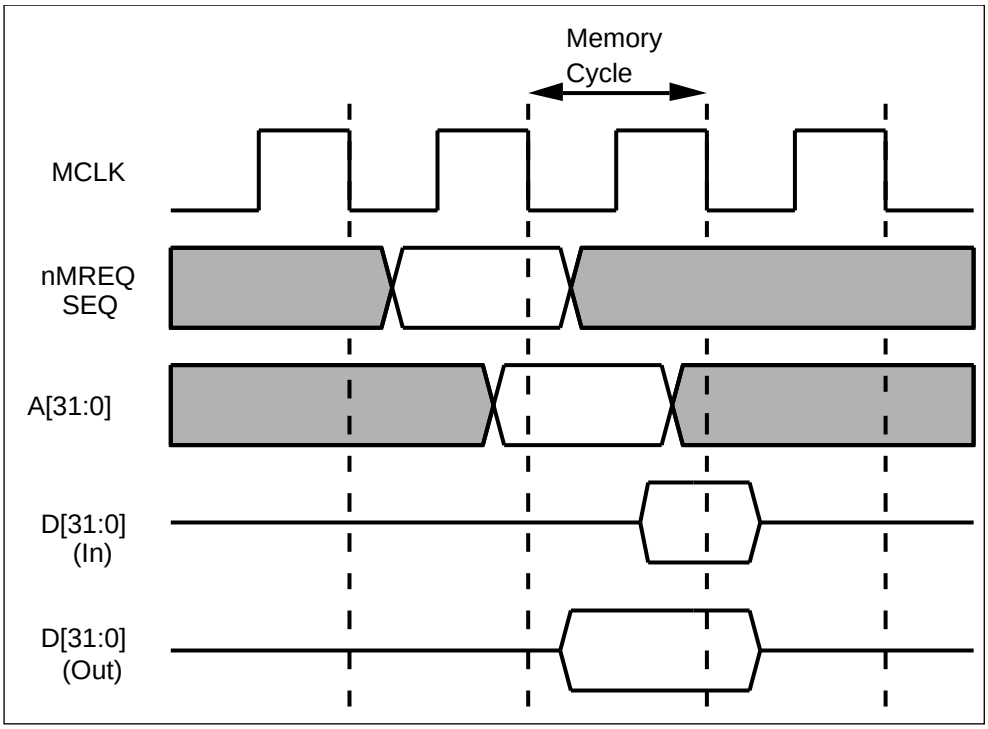

Figure 3-5: The ARM Memory Pipeline

At any time, the type of cycle which will be performed next can be determined by the current state of **nMREQ** and **SEQ**, and the previous state of **nMREQ**. This can be summarised in the following table.

nMREQ	SEQ	Previous	Next Cycle
0	0	X	Non-sequential (N)
0	1	0	Sequential
0	1	1	Merged I-S (MIS)
1	0	X	Internal (I)
1	1	X	CPRT

Table 3-1: The ARM6 Bus Cycle Encoding

The transitions between bus cycles can be summarised by the state diagram shown in ➲*Figure 3-6: The ARM6 Bus State Transitions*.

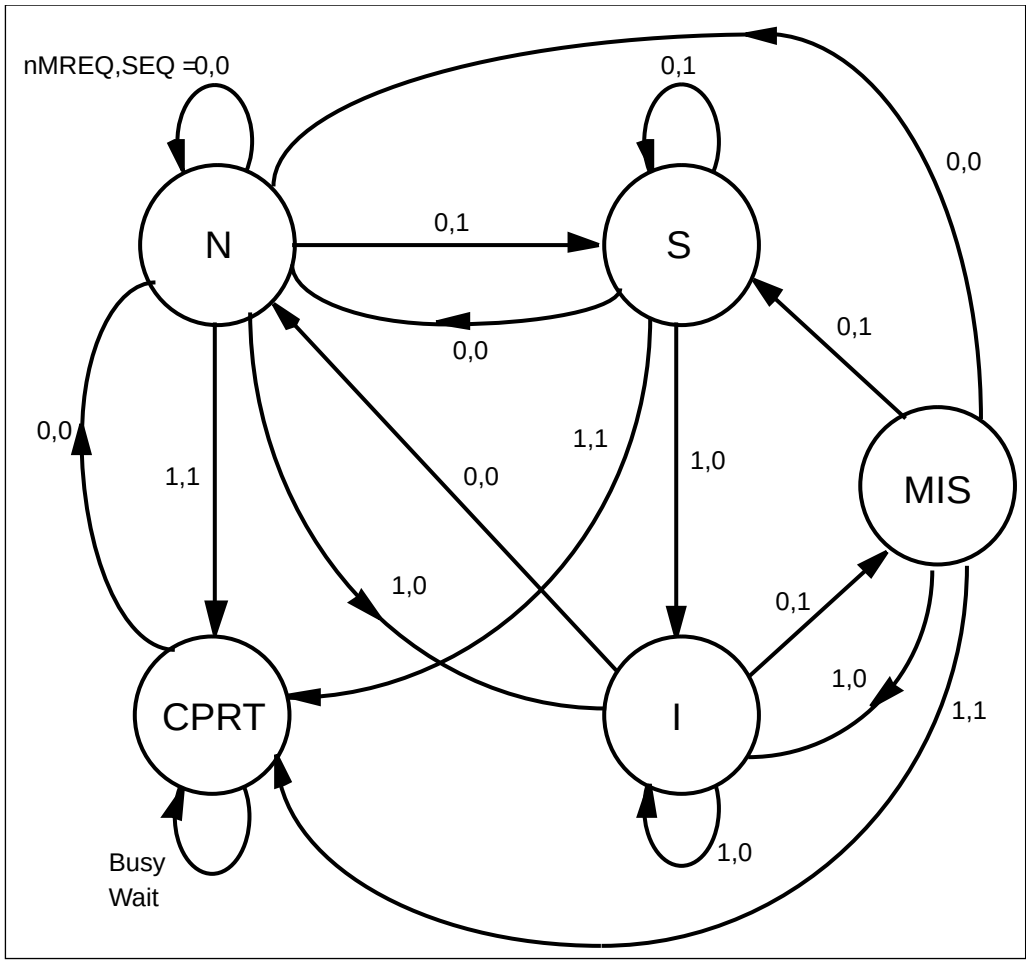

Figure 3-6: The ARM6 Bus State Transitions

Figure 3-7: An Ideal N-Cycle on page 10 shows the bus activity during an N-cycle. During phase 1 of the pervious cycle, **nMREQ** and **SEQ** are driven to indicate that the following cycle will be non-sequential. During phase 2 of the previous cycle, the address becomes valid. Since this cycle is *non-sequential*, the address is unrelated to the previous address. At the end of phase 2 of the actual cycle, the data is valid.

The diagram shows an ideal N-cycle in that it assumes zero wait-state memory. If the system was running from fast ROM or SRAM, then cycles of this speed could be achieved. More common would be a system comprising dynamic RAM (DRAM). In this case, wait states would have to be inserted for the RAS and CAS part of the cycle. *Figure 3-8: A Practical N-Cycle* on page 11 shows how a typical cycle would look using a memory system where non-sequential accesses take twice as long as sequential accesses.

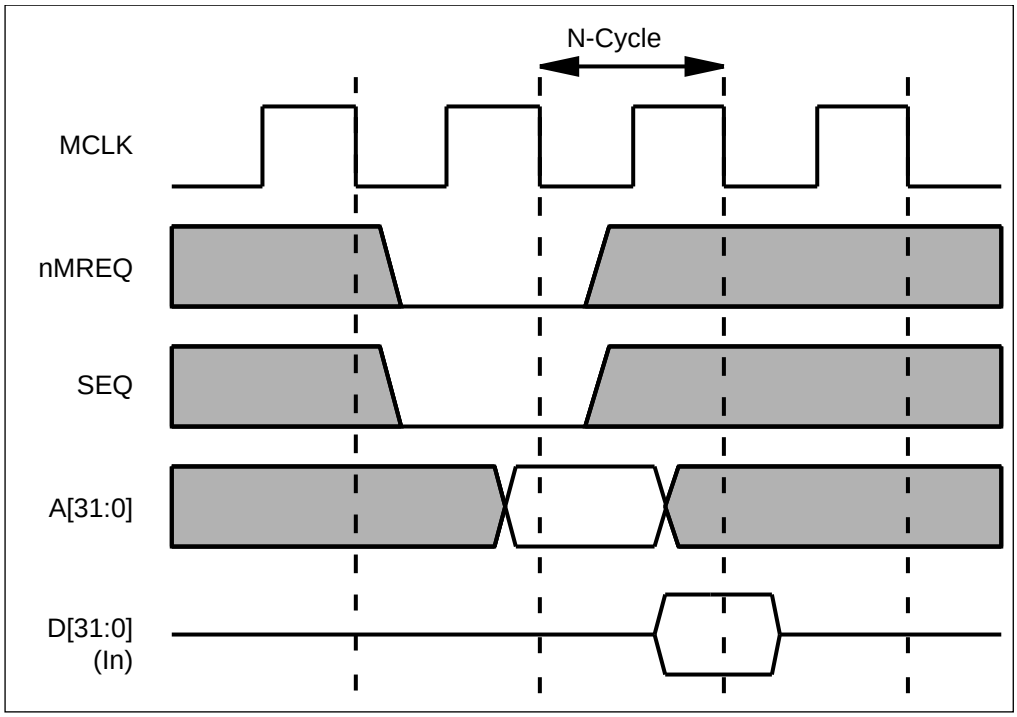

Figure 3-7: An Ideal N-Cycle

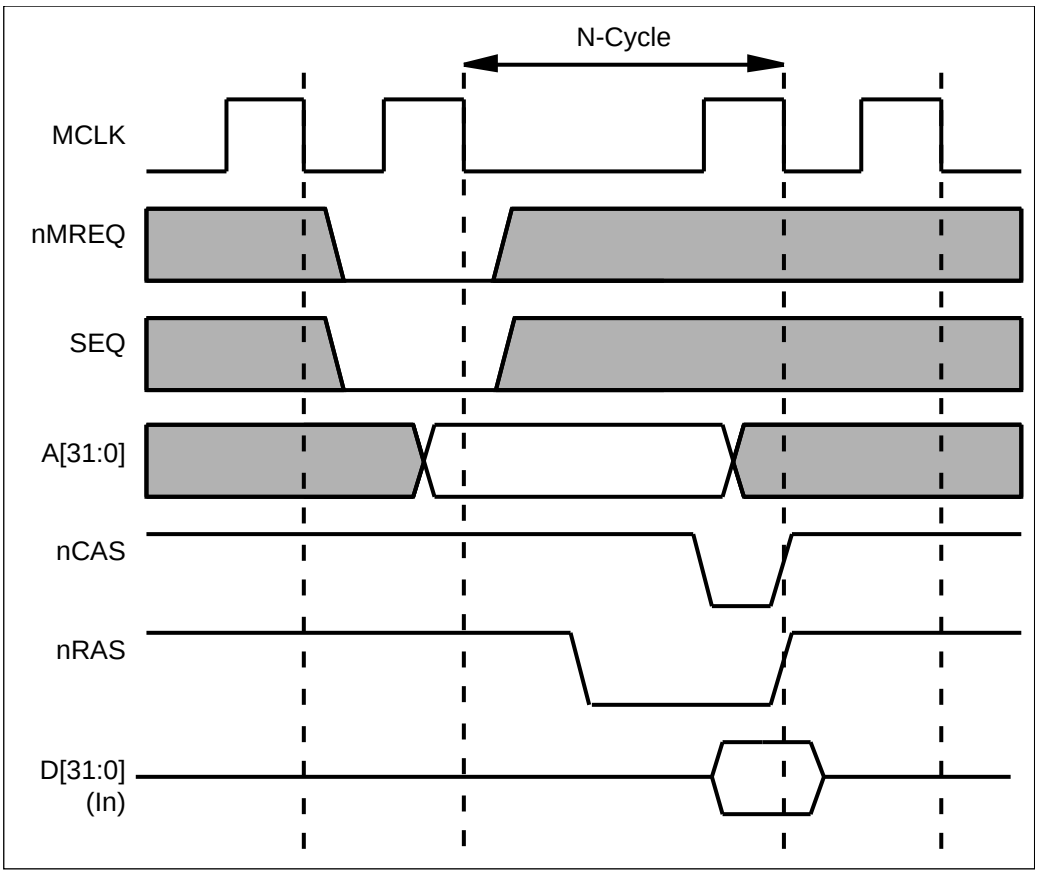

Figure 3-8: A Practical N-Cycle

Note that **MCLK** has been stretched at source, though a system could easily be built with a free running **MCLK** using **nWAIT** to stretch the cycle.

Figure 3-9: Ideal NS-Cycles shows a N-cycle followed by an S-cycle in an ideal memory system. The definition of a sequential cycle is that the address for the access is one word greater than that of the previous cycle. This is particularly of use in a DRAM system since it allows use of burst or page mode memory. *Figure 3-10: Practical NS-Cycles* on page 13 shows this in operation. The memory's RAS input is asserted for the N-cycle, followed by CAS. For the following cycle, RAS is held active, and the sequential access is performed by reasserting CAS. Memory systems built in this way offer both improved performance, and reduced system power consumption.

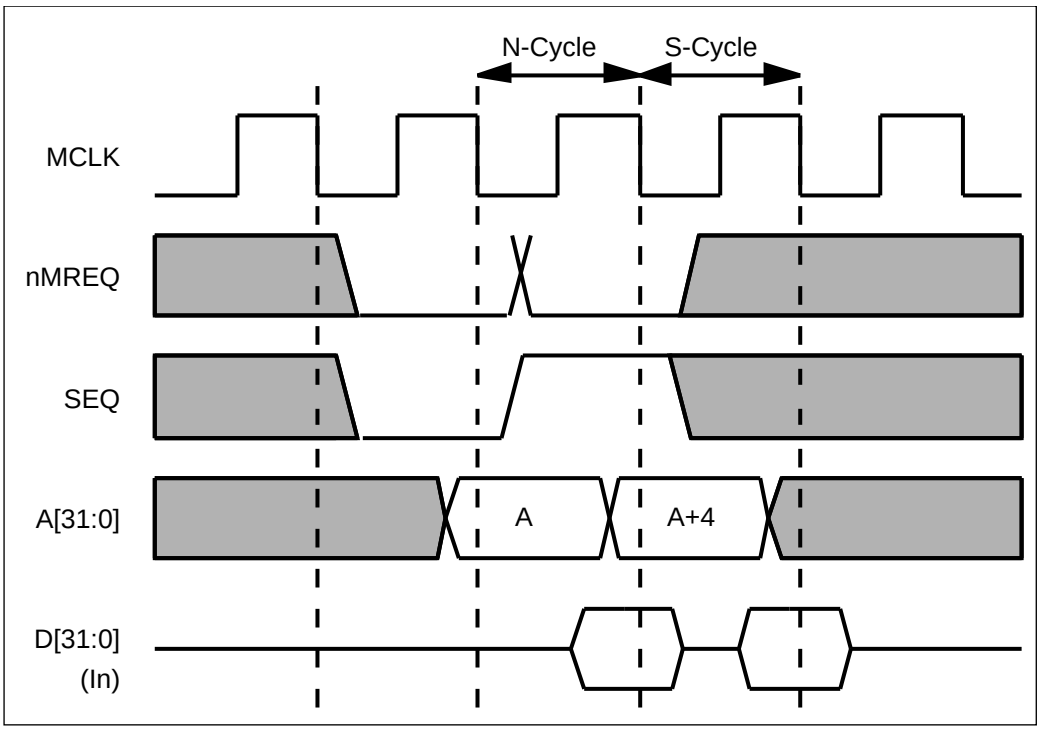

Figure 3-9: Ideal NS-Cycles

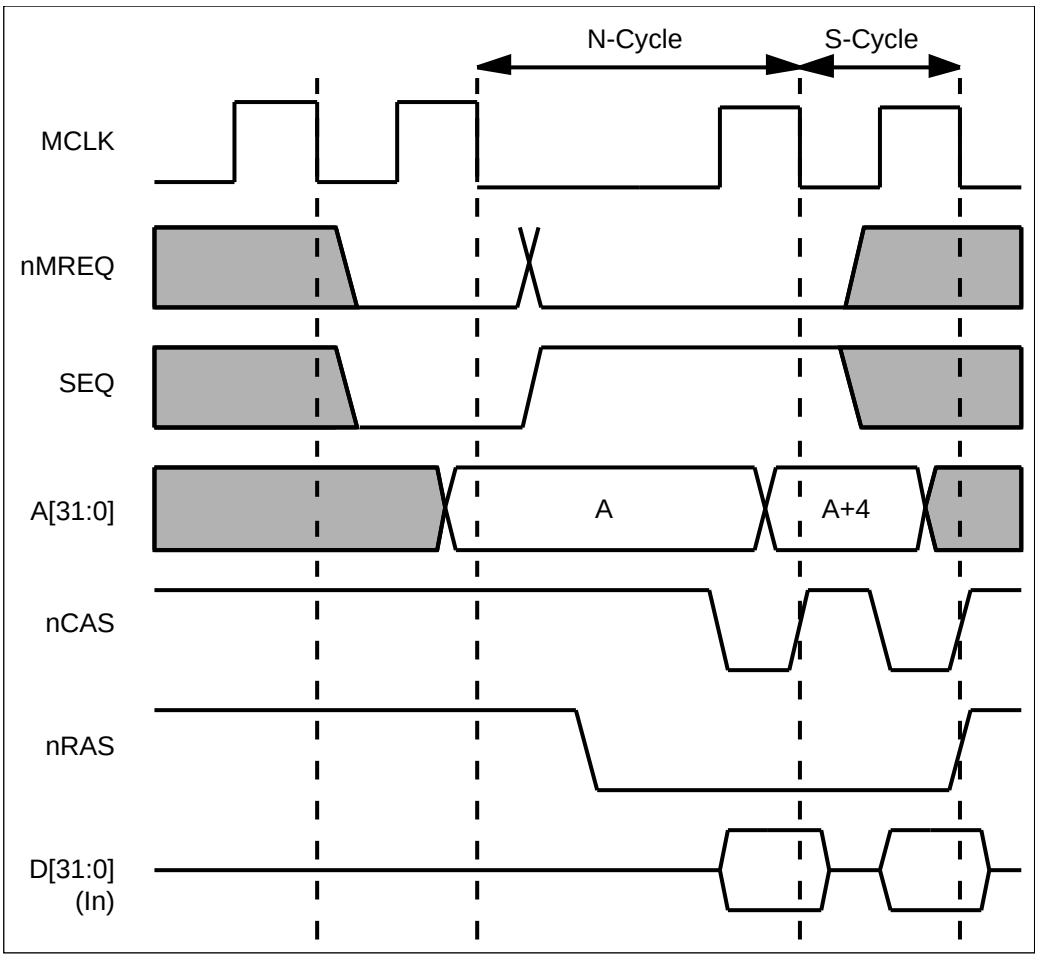

Figure 3-10: Practical NS-Cycles

Figure 3-11: A Merged IS-Cycle(MIS) on page 14 shows a merged IS-cycle. Internal cycles are performed by the ARM whenever it does not need memory. For example, when a multiply is executed, the ARM will perform an instruction fetch, and then perform a number of internal cycles whilst calculating the result of the multiply. Similarly during the final cycle of a load instruction, the ARM will perform an internal cycle whilst writing the loaded data into the destination register. Under most circumstances, after an internal cycle the ARM will perform a sequential cycle, and memory systems can be constructed to exploit this for performance gains.

During the internal cycle, the ARM presents the address for the next instruction fetch on the address bus, although **nMREQ** denotes that memory is not required. During the sequential cycles the address does not change. Thus, during phase 1 of the I-cycle, a memory controller can start decoding the address for the DRAM row access. During phase 2 of the I-cycle, RAS can be asserted once the memory controller has acknowledged that **nMREQ** and **SEQ** have been driven to denote that the following cycle will be sequential. Then, in the S-cycle, CAS can be asserted to complete the memory cycle. Note that if this optimisation is not exploited, and the memory system

does nothing during the I-cycle, then the access will take longer to complete since the S-cycle would have to be treated as an N-cycle. In this case, the access would have the same timing as an IN-cycle.

Merged IS-cycles are not performed when the destination register is the PC. For example, after the internal cycle at the end of a data load, the ARM will normally perform an S-cycle. However, if the load destination was the PC, then an N-cycle will be performed after the I-cycle. This is shown in *Figure 3-12: A Typical IN-Cycle* on page 15.

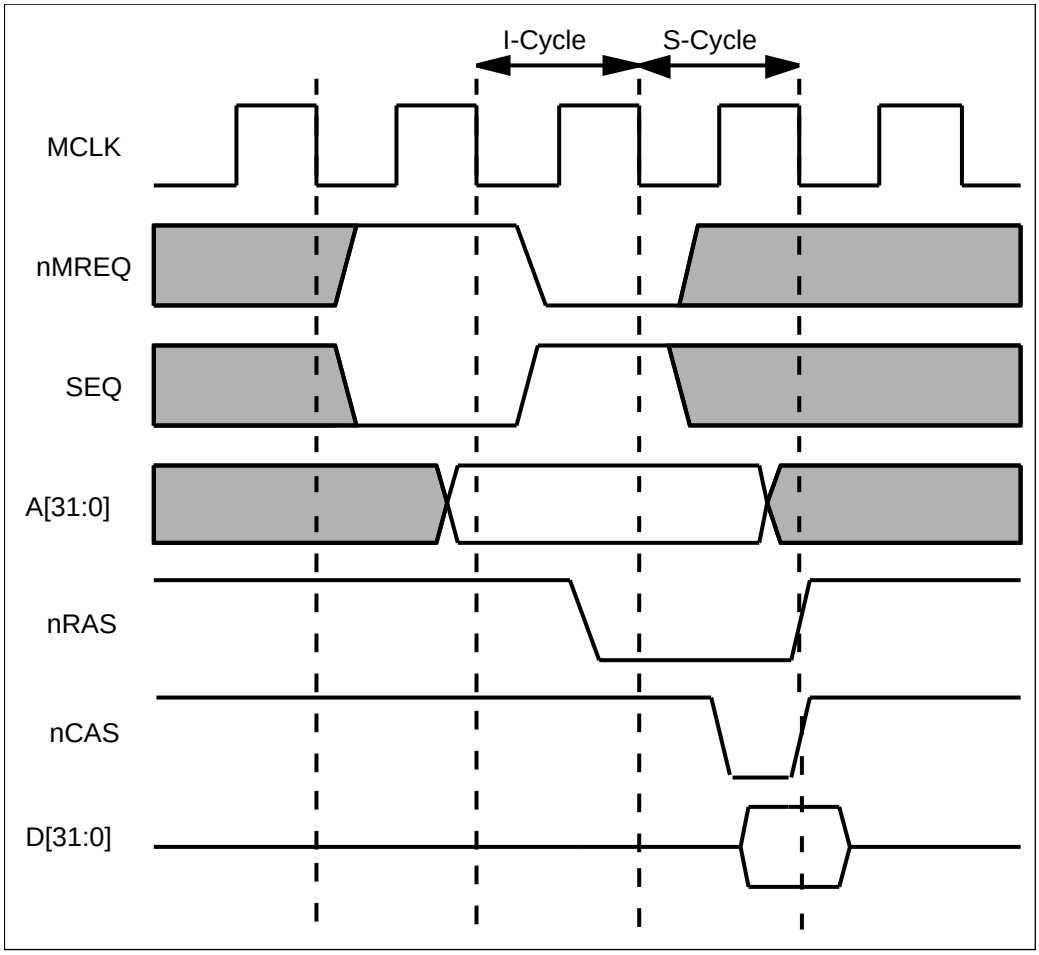

Figure 3-11: A Merged IS-Cycle(MIS)

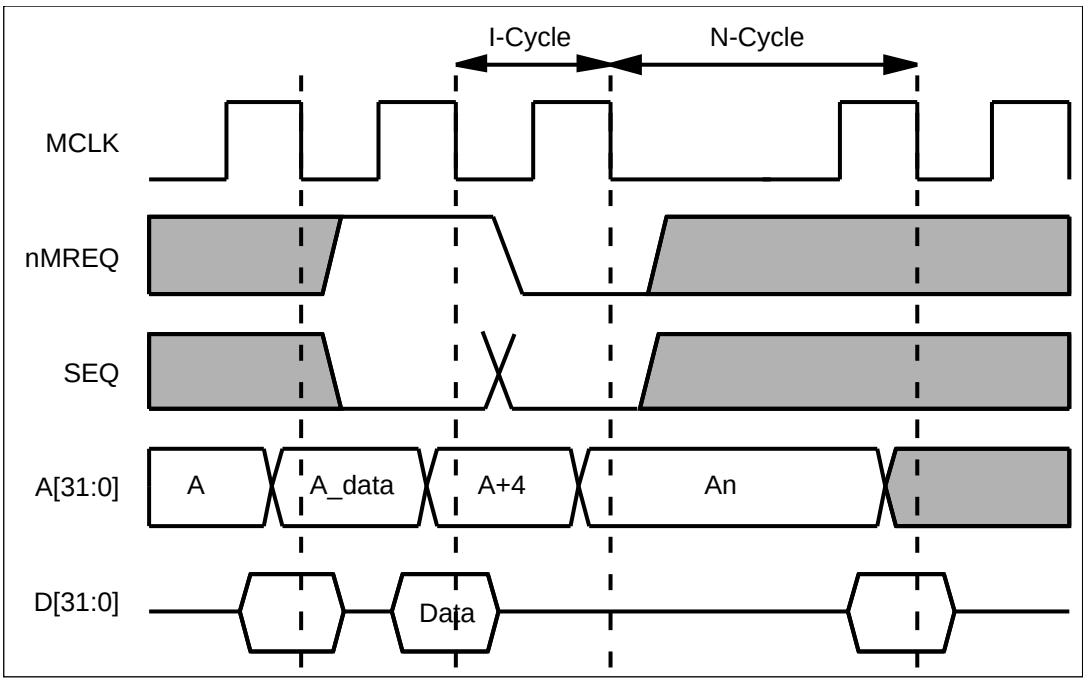

Figure 3-12: A Typical IN-Cycle

The ARM bus performs CPRT cycles whenever it needs to transfer data between itself and a coprocessor. Note that although **nMREQ** is high, denoting that memory is not required, the ARM6,60 and 61 do not have a dedicated coprocessor bus and so use the system data bus to transfer the data. A memory controller must be aware of this and prevent other use of the bus during these cycles. *Figure 3-13: The CPRT Cycle* on page 16 shows a CPRT cycle as executed by an ARM60. The diagram shows both the bus control signals and the coprocessor control signals. For more information on attaching coprocessors to the ARM, please refer to the relevant data sheet.

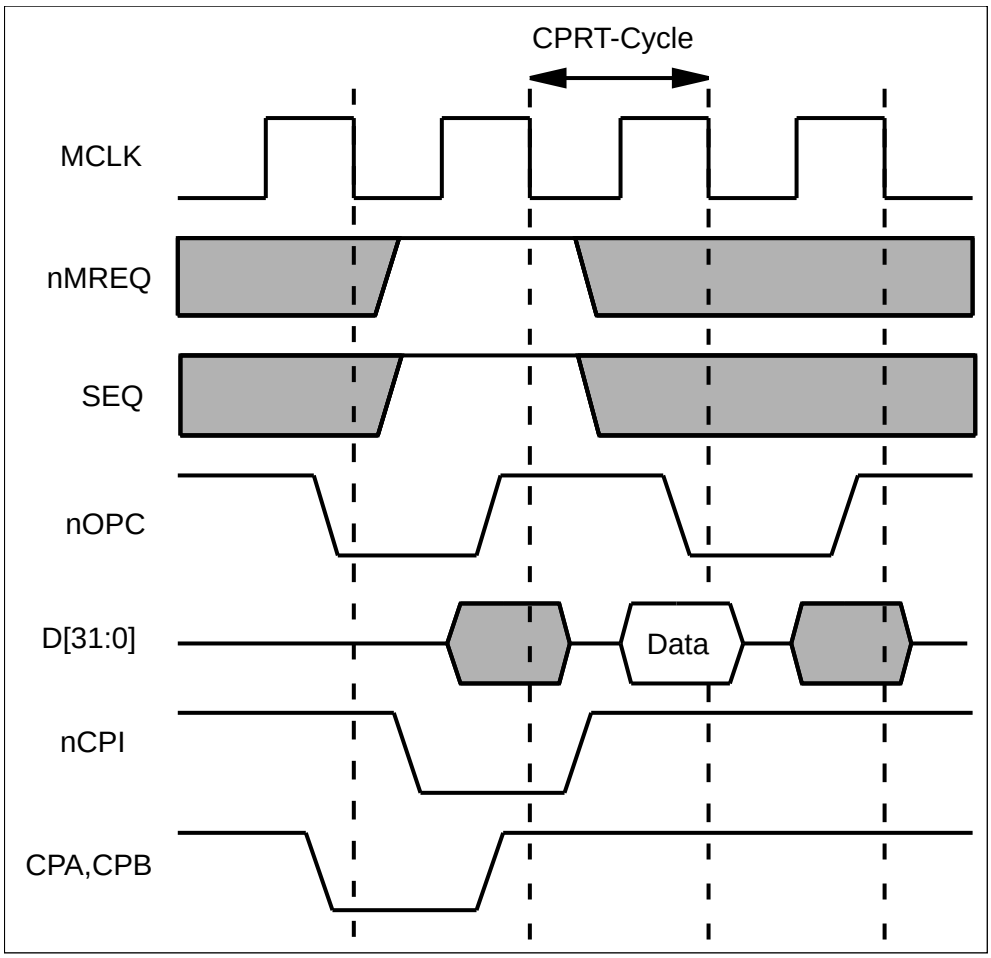

Figure 3-13: The CPRT Cycle

4 Example Code

This section considers typical activity of the ARM6 bus using short sections of code as examples.

4.1 Data Load

The first example shows a branch to a routine which performs a data load.

```
      BL    label
.....

label LDR   R2, [R0]
      ADD   R2, R2, R3
      SUB   R2, R2, R4
      MOV   PC, R14
.....
```

The progress of these instruction through the ARM's Fetch: Decode: Execute pipeline is shown in ➢*Figure 4-14: ARM Instruction Pipeline Activity: Example 1*. Note that after the branch-and-link (BL) instruction, two more instructions are loaded. However, when the instruction gets executed the pipeline is flushed.

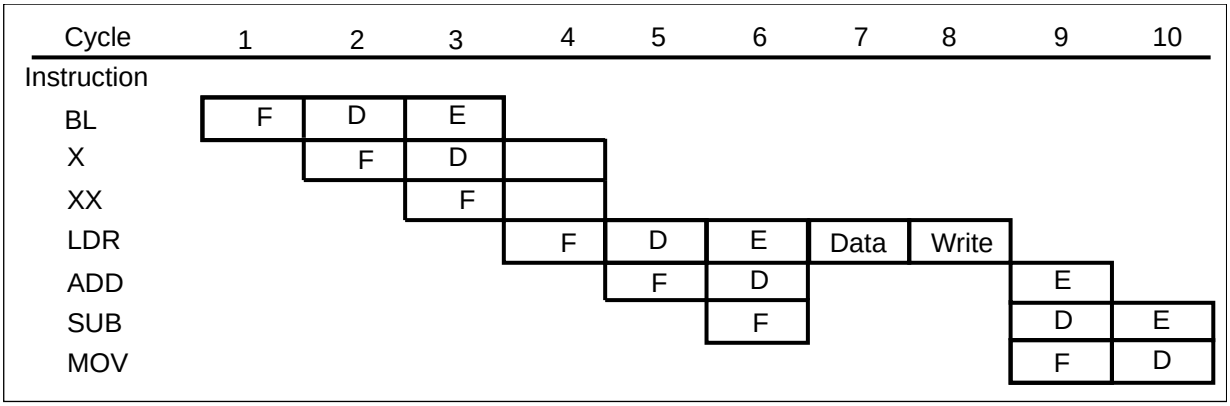

Figure 4-14: ARM Instruction Pipeline Activity: Example 1

➢*Figure 4-15: ARM Bus Activity: Example 1* shows the activity on the ARM's bus between cycles 4 and 9. This shows the full range of bus cycles (except CPRT): a burst of two sequential instruction fetches, a non-sequential data access, followed by a merged IS-cycle.

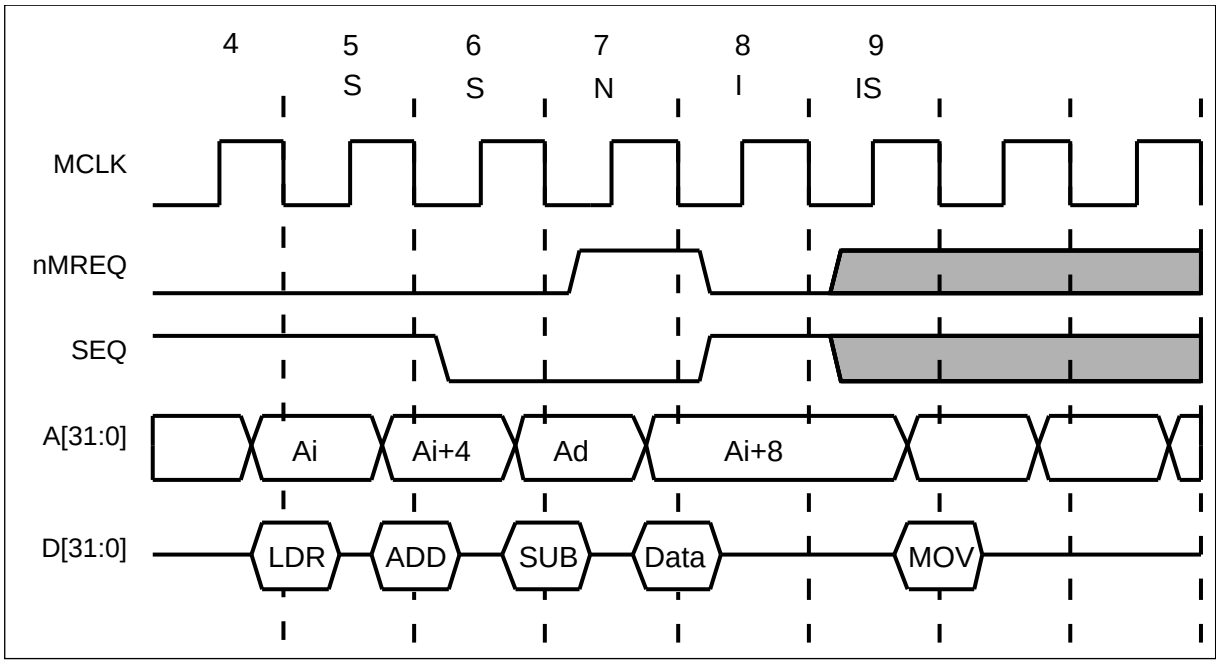

Figure 4-15: ARM Bus Activity: Example 1

4.2 Data Store

The second example shows an instruction sequence containing a data store.

```
......
      ADD   R1, R0, R0
      STR   R0, [R1]
      SUB   R3, R0, R1
      BIC   R4, R3, R2
      AND   R0, R2, R3
      EOR   R7, R3, R2
......
```

Progress through the ARM's instruction pipeline is similar to the previous example, and is shown ➲*Figure 4-16: ARM Instruction Pipeline Activity: Example 2*.

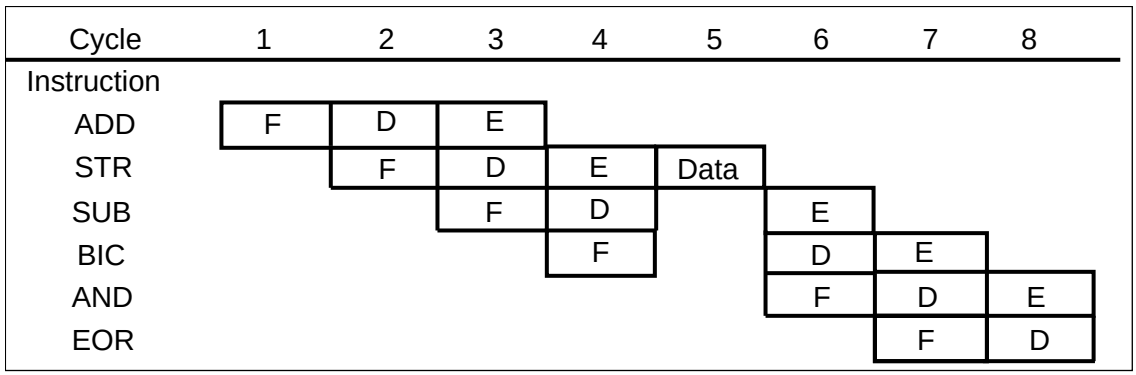

Figure 4-16: ARM Instruction Pipeline Activity: Example 2

The activity on the bus during the execution of cycles 3 to 7 of this section of code is shown in ➲*Figure 4-17: ARM Bus Activity: Example 2*. This shows the sequential instruction fetches, the non-sequential data store followed by a non-sequential instruction fetch, and then further sequential fetches. Note that data stores are always non-sequential, and the instruction fetch after a data store is also always non-sequential. In the case of a store-multiple (STM), the first memory access is always non-sequential, and the following stores are then sequential. The instruction fetch after a store-multiple is always non-sequential.

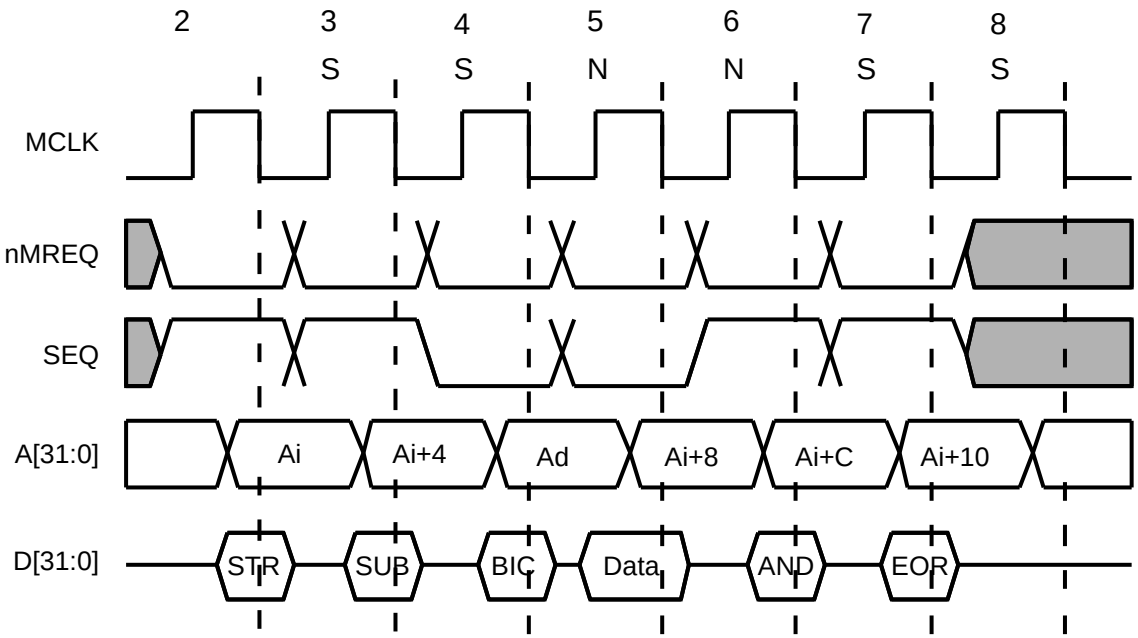

Figure 4-17: ARM Bus Activity: Example 2

ENGLAND
Advanced RISC Machines Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
Telephone: +44 1223 400400
Facsimile: +44 1223 400410
Email: marketing@armltd.co.uk

JAPAN
Advanced RISC Machines
KSP West Bldg, 3F, 3-2-1 Sakado,
Takatsu-ku, Kawasaki-shi
Kanagawa, 213
Telephone: +81 44 850 1301
Facsimile: +81 44 850 1308

USA
ARM USA
Suite 5, 985 University Avenue
Los Gatos
California 95030
Telephone: +1 408 399 5199
Facsimile: +1 408 399 8854
Email: ARMUSA@armltd.co.uk