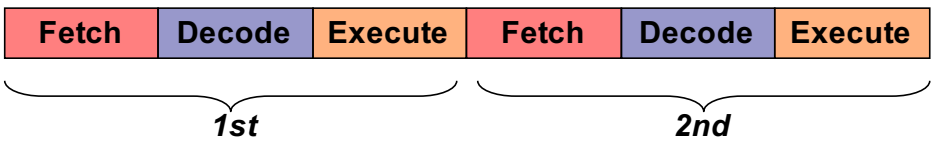
ARM® Memory System

ARM Architecture Design

ARM7 RISC architecture:

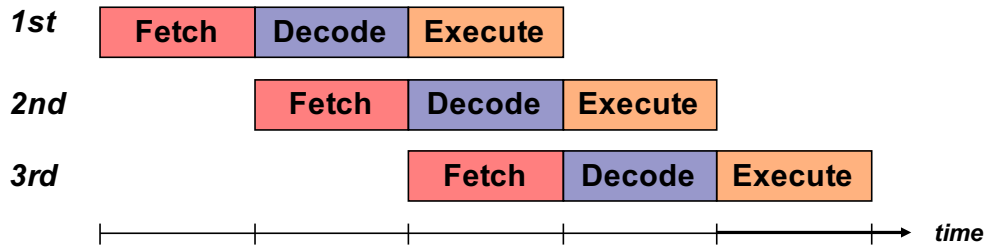
- 32-bit data, but data can be accessed as 8-bit byte, 16-bit half-word, or 32-bit word
- Only the *load*, *store*, and *swap* instructions can be used to access data from the memory
- Each instruction has an execution latency of three clock cycles, i.e., one instruction per three clock cycles



ARM7 Pipeline

Uses the 3-stage pipeline for instruction executions.

- Typical pipeline stages:
  Fetch → Decode → Execute
- Pipeline design allows effective throughput increase to one instruction per clock cycle
- Allows the next instruction to be fetched while still decoding or executing the previous instructions



Pipeline Memory Access

With pipeline:

- Memory access can occur at every clock cycle
  - i.e., each stage can be completed in one processor clock cycle (denoted as MCLK for the ARM processor)
- Fetch instruction opcodes (for example, from Flash)
- Read or write data to the memory (SRAM or DRAM)

Pipeline Memory Access (cont'd)

Implications:

- Memory (and peripherals) must have access time that is compatible to the clock cycle of the processor (MCLK)
- A moderate MCLK of 50 MHz requires memory access time in the order of 20 nsec!
- Therefore, either use a very fast memory device (i.e., a very fast SRAM),
- or operate the processor at a low frequency

ARM7 Processor Clock

All state changes within the ARM7 are controlled by two signals:

- MCLK memory clock signal
- nWAIT control signal

Logical AND of these two signals:

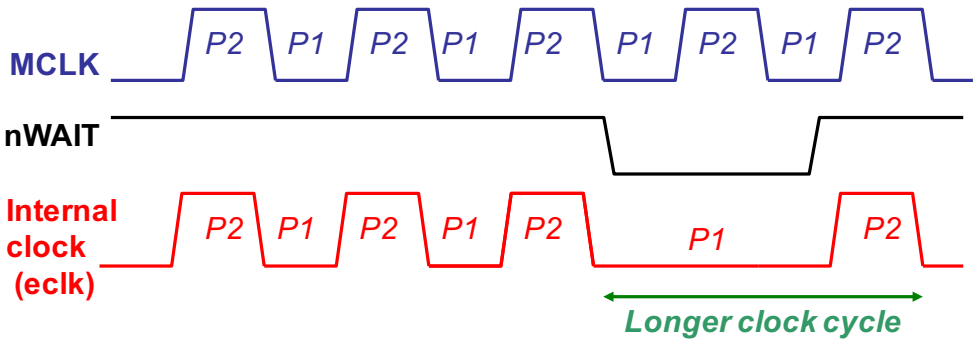
- produces the internal clock cycle that 'powers up' the core.
- allows the processor to 'slow down' (momentarily) without using a lower processor clock signal
  - by skipping clock cycle(s) (for example, in order to match the slower memory access speed).
  - skipping clock cycles sometmes known as "wait states"

ARM7 Clock Timing

ARM design allows the internal processor clock to be varied during operations.

• Typically, P1 of the internal clock cycle can be 'stretched' to make the processor skip one or more of the external MCLK clock cycles (for example, to 'wait' for a slower peripheral access).

Logical AND of MCLK and WAIT#



Memory Interface Signals

Memory interface signals of the ARM7 core:

• A[31:0]: 32-bit address bus

• D[31:0]: 32-bit bidirectional data bus
Dout[31:0]: for separate data out bus
Din[31:0]: for separate data in bus

• r#/w: Read (active low)/Write control signal

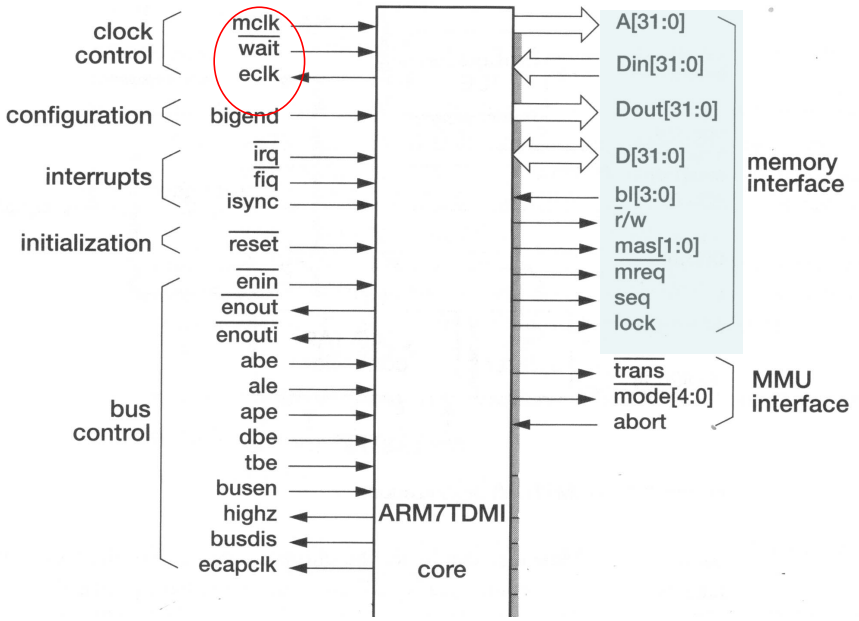
• mas[1:0]: Memory Access Size
  – 00 = Byte; 01 = Half-word; 10 = Word; 11 = Reserved

Memory Interface Signals (cont'd)

- mreq#: Memory request
  - Indicates that the next instruction cycle involves a memory access
- seq: Sequential Addressed Access
  - Indicates that the address used in the next cycle will be either the same or one operand (i.e., word) greater than the current address
  - this is a form of pipelining (memory pipelining)
  - some architectures also refer to this as a "burst mode" such as Intel Pentium

Memory Interface Signals (cont'd)

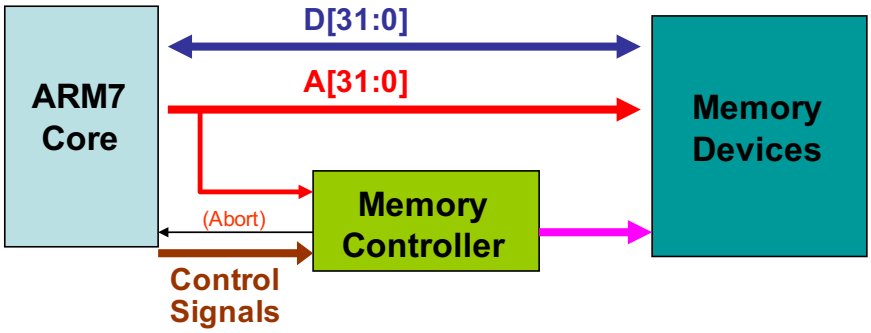
ARM7 memory interface signals



Memory Controller

ARM7 core:

- Provides various control signals that can be used for memory interface
- Needs a separate memory controller to perform the actual memory access control functions
  - For example, address decoding, wait state generation, DRAM refresh cycle, etc.

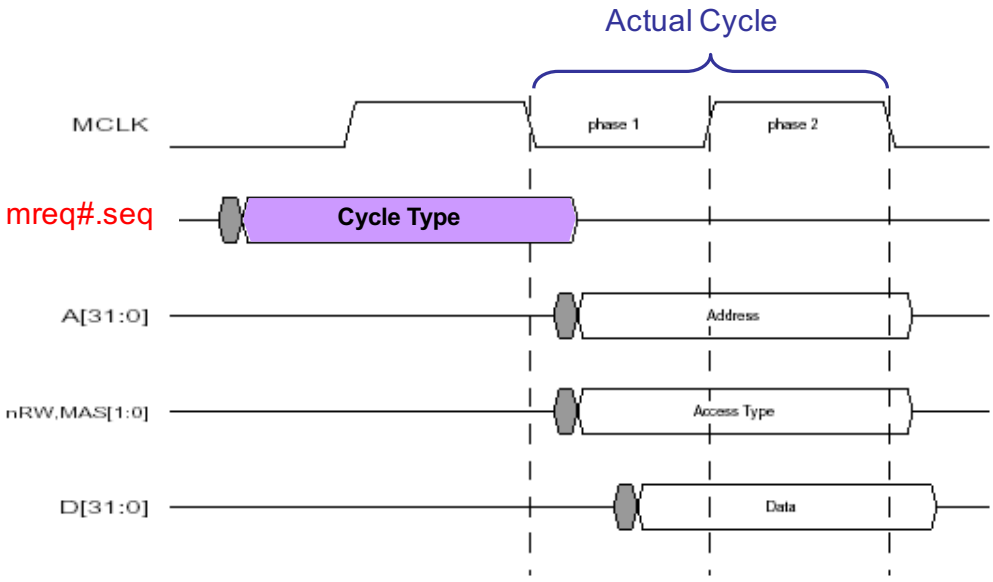


mreq# and seq Timing

Memory controller can also make use of the mreq# and seq signals.

- To decide the best method to handle the memory access in the next cycle
- These signals are issued more than half a cycle before the actual cycle
- Hence, the memory controller can start the memory access 'preparation' before the actual cycle commences

mreq# and seq Timing (cont'd)



Bus Cycle Types (N, S, I, C)

Four possible bus cycle types:

mreq#	seq	Cycle	Type
0	0	N	Nonsequential Memory Access
0	1	S	Sequential Memory Access
1	0	I	Internal Cycle (Bus & Memory inactive)
1	1	C	Coprocessor register transfer (Memory inactive)

Nonsequential Cycle (N-Cycle)

Nonsequential cycle:

- The simplest form of bus cycle
- Occurs when the processor requests a transfer to or from an address that is unrelated to the address used in the preceding cycle
- The memory controller will initiate a memory access to satisfy this request

Sequential Cycle (S-Cycle)

The sequential cycle is indicated by the seq signal.

During a sequential cycle:

• the ARM7 processor will request for a memory location that is part of a sequential burst.

The first address can be the same if the previous cycle is the internal cycle.

• Otherwise, the address is incremented from the previous cycle:

- For a burst of word accesses, the address is incremented by four bytes.
- For a burst of half-word accesses, the address is incremented by two bytes.

S-Cycle and DRAM

The sequential cycles are used to perform burst transfers on the bus.

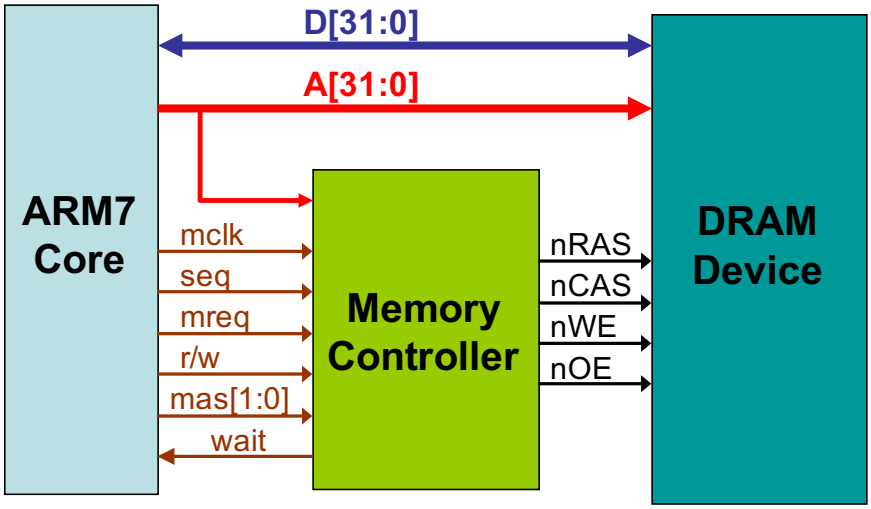
- Can be used to optimize the design of a memory controller interfacing to a burst-optimized memory device, like SDRAM.

SDRAM (Synchronous DRAM):

- Don't Confuse this with SRAM (Static RAM)
- Specially designed to respond faster to a sequential access
- Requires a shorter access time compared to a random (i.e., nonsequential) access

Memory Controller

Memory controller for DRAM interfacing:



Typical S-Cycle Usage

If an S-cycle is to follow an N-cycle:

⇒next address = current address + four bytes

- The memory controller can prepare the memory for fast access.
- For example, to check if the current address is at the end of the row of the DRAM.
- If not, issue Page mode access to the DRAM (which is found to occur in 75% of the memory access).

Typical S-Cycle Usage (cont'd)

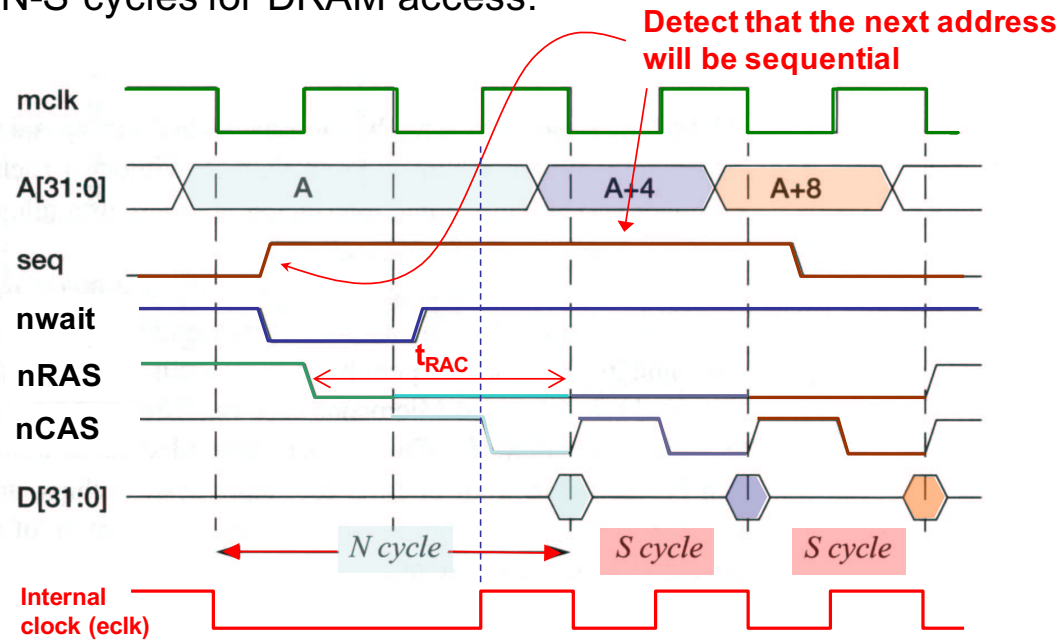
If an S-cycle follows an I-cycle or C-cycle:

The next address is already the current address on the memory bus (because the I-cycle and C-cycle do not use the memory bus).

- Hence, the memory controller can start the memory access immediately.

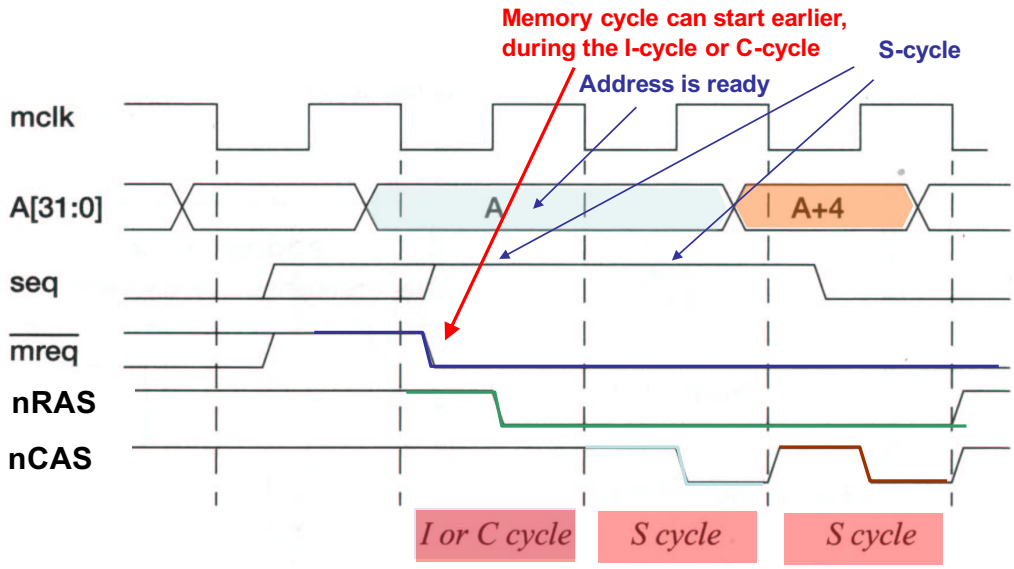
N-S Cycles

N-S cycles for DRAM access:



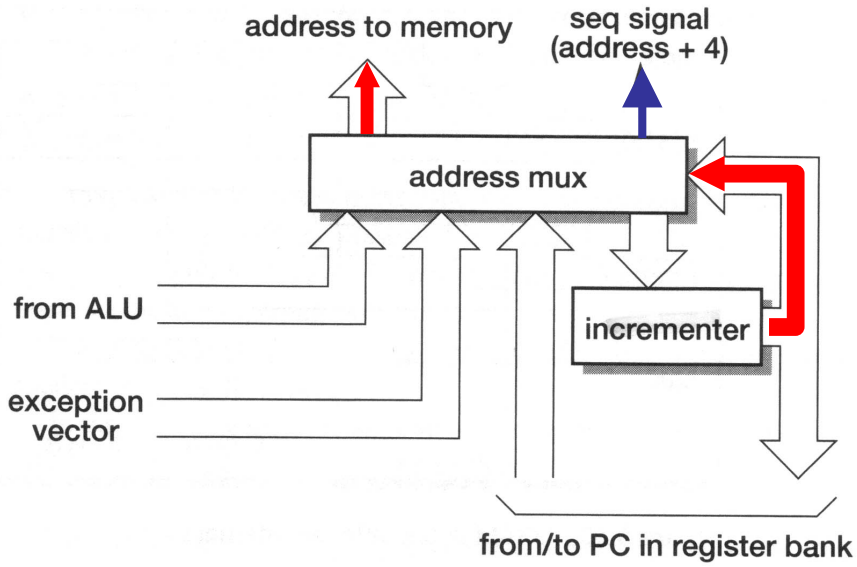
I-S Cycles

I-S cycles for the DRAM access:



Generation of the seq Signal

The seq signal is automatically asserted whenever a memory address is obtained from the incrementer.



Basic ARM Memory System

Typical Memory system Configuration:

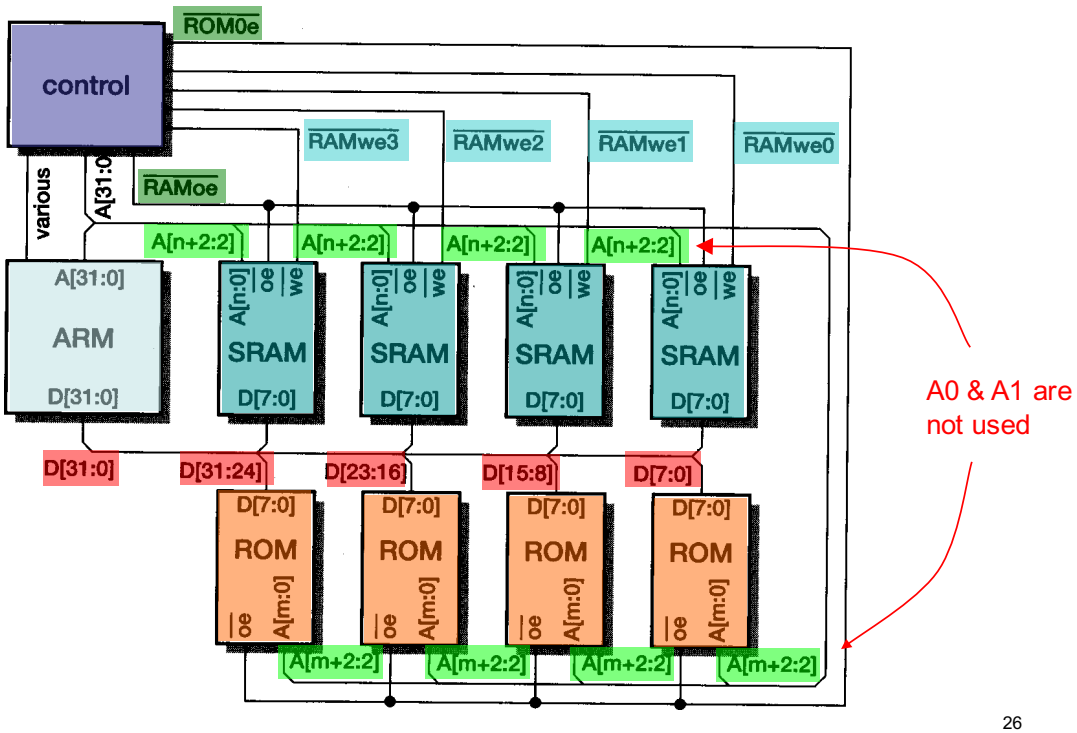
- ROM (Flash, EEPROM, or EPROM) to store the firmware (needed for boot-up and subsequent execution)
- SRAM/SDRAM for program execution and data storage (for example, shadowing of the ROM after boot-up)
- Four standard byte-wide Flash memory devices are used to form a 32-bit bank
  - Data is always accessed in 32-bit word
  - m bits Address lines Ρεσυλτσ ιν Address space = 2^m

Basic ARM Memory System (cont'd)

Memory system specifications for the ARM system:

- Four byte-wide SRAM/SDRAM devices are used to form a 32-bit bank
  - Data can be read in 32-bit word size
  - Data must be able to be written in 8-bit byte size, 16-bit half-word, or 32-bit word as required
  - i.e., a low-order interleaved memory design
  - n bits Address lines Ρεσυλτσ ιν Address space = 2^n

A Simple Memory System



Memory Interface

Note the lowest two bits of the Address Bus A[0] and A[1]

- Not connected to either the ROM or RAM address lines
  - But is used for the SRAM/SDRAM byte selection
- SRAM can be write-access as byte, half-word, or word
  - Controlled by the individual RAMwe# signals
  - Generated based on the values of A[1] and A[0] bits
- SRAM is also always read-access in 32-bit word
  - Controlled by the RAMoe# signal
- ROM is always accessed (read-only) in 32-bit word
  - Controlled by the ROMoe# signal

ROM and RAM Address Space

Since ARM has a flat memory:

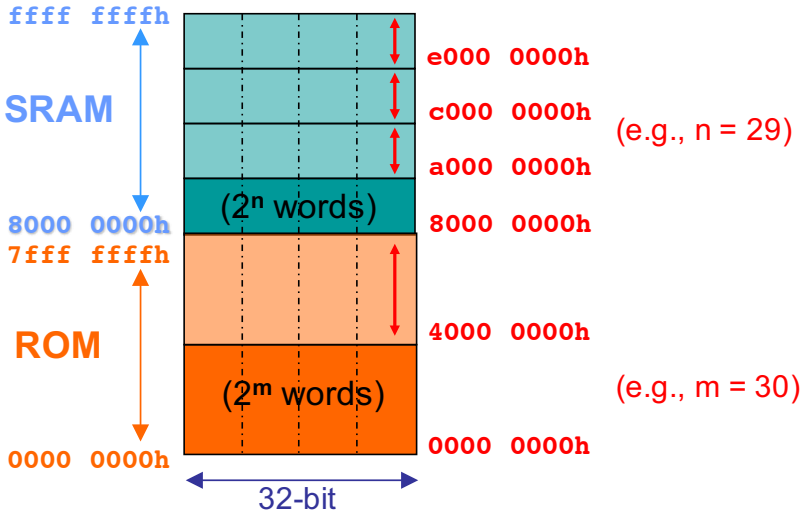
- The ROM and SRAM reside in the same address space
- Actual size depends on the size of the address bus (values of 'm' and 'n') of the ROM and RAM memory devices
- For example;
  - m = 20 Ρεσυλτσ ιν 2^{20} = 1 M of the ROM address space
  - n = 19 Ρεσυλτσ ιν 2^{19} = 512 k of the RAM address space
- The ROM must be in the lower half of the address space
  - Reset vector is at 00000000h
- A simple design will use A[31] to select between the ROM and RAM
- A[1:0] will be used to enable the byte access of the SRAM

Address Space

Partial decoding based on A[31]

•Use only bit 31 to select either ROM or SRAM

•Can cause memory duplication between SRAM and ROM



Memory Controller Logic

Input signals needed by the memory controller:

- A[31] to select between the ROM and SRAM
- mas[0] and mas[1] to indicate the data size to be accessed
- A[0] and A[1] to select the byte(s) within a word, based on the values of mas[1] and mas[0]
- r#/w to indicate a read or write access
- MCLK clock to synchronize the generation of the control signals with the processor clock

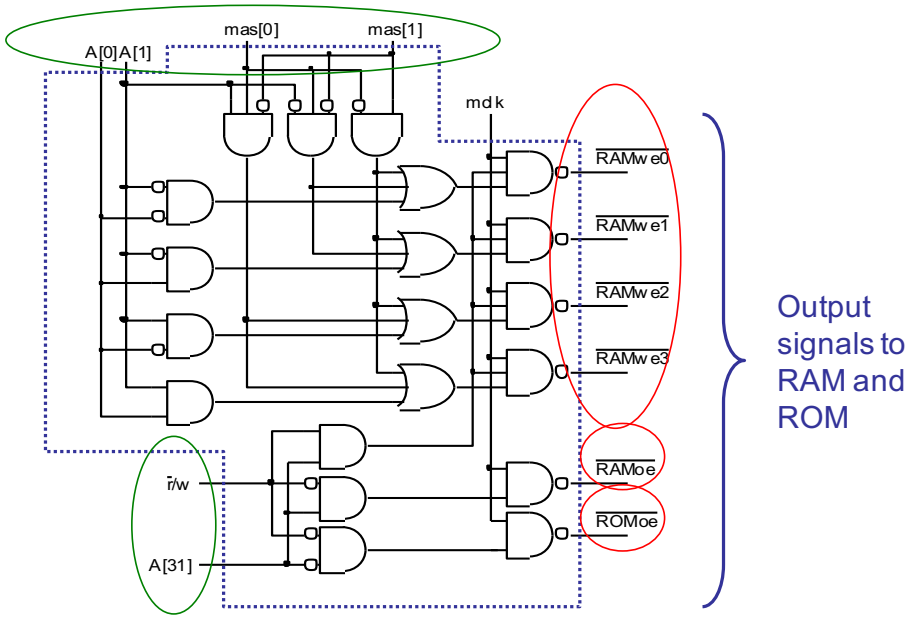
Memory Controller Logic (cont'd)

Output signals generated by the memory controller:

- RAM and ROM enable signals: RAMoe and ROMoe#
- Four RAM-byte write enables signals: RAMwe*#

Memory Controller Logic (cont'd)

Implementation example:



Access Speed of Memory Devices

Apart from generating the control signals,

- signal timings also need to be considered, i.e., largely depend on the access speed of the memory devices used
- 'Simplistic' approach:
  I. Use memory devices with very fast access speed
  II. Reduce the MCLK clock frequency
- In practice, ROM will be much slower than RAM which normally has the access time of high tens of nsec
- Reducing the MCLK frequency will reduce the overall performance of the entire system

Use of Wait States

ARM has the provision to temporarily skip a few of the MCLK clock cycles.

•When accessing slower peripherals

•Can be achieved by asserting the WAIT# input signal of the processor core

•i.e., 'injecting' Wait states into the processor cycles to slow down the processor

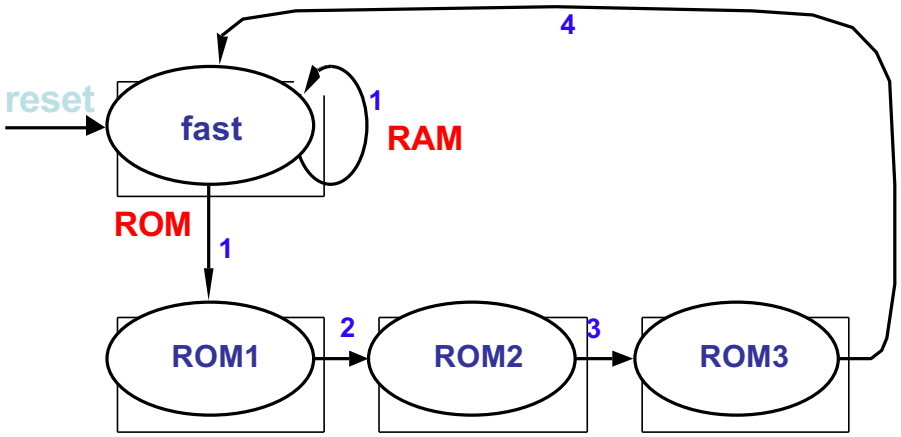
•As ROM is typically much slower than SRAM
- Use SRAM with an access speed that is compatible with the processor clock cycles
- Introduce Wait states when the system is trying to access the ROM

ROM Wait State Transition

Transition diagram for the ARM memory controller:

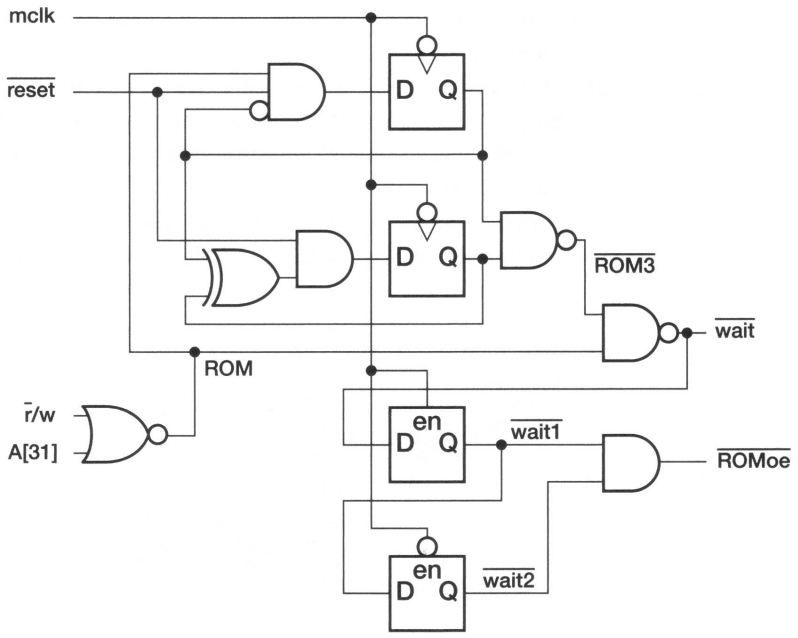
•Assume that the ROM access requires the introduction of four wait states to the processor cycles.

•The RAM access can be carried out in one MCLK cycle.



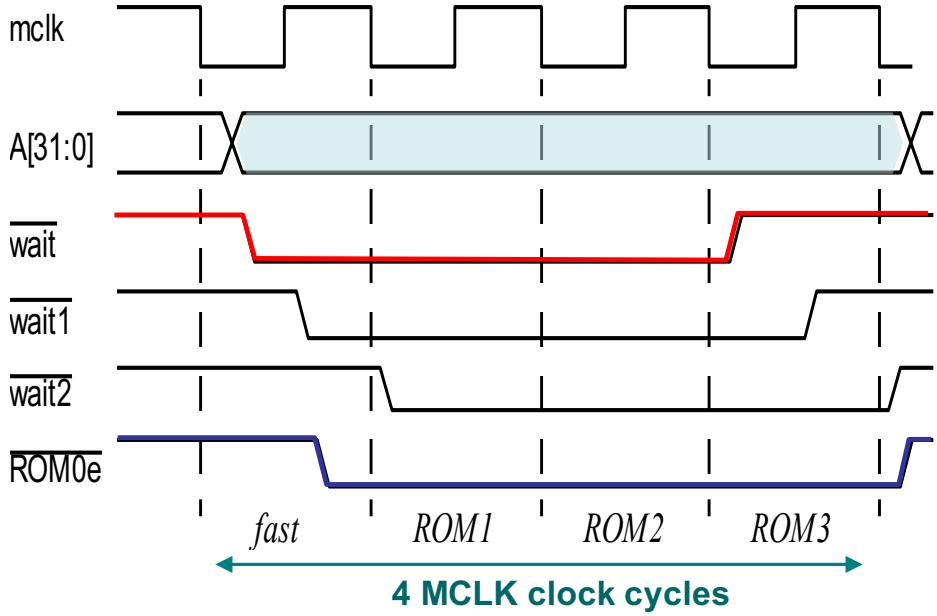
Circuit Implementation

One possible implementation of the Wait state generation:



ROM Wait State Timing

Timing Diagram with Wait states during ROM access:



Practical Memory System

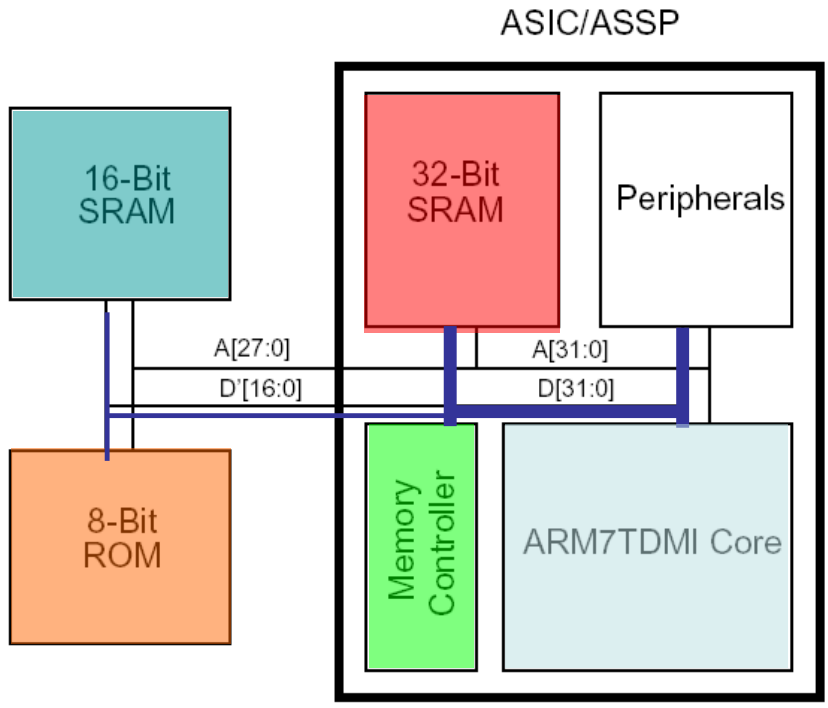
More practical memory system design for ARM system:

- An 8-bit ROM (Flash) to store the firmware for boot-up
- A 16-bit off-chip SRAM to execute Thumb® instructions (i.e., the firmware is transferred from the ROM to a 16-bit RAM after boot-up for faster execution)
- An on-chip 32-bit SRAM to execute full 32-bit ARM instructions

Reference for more details:

ARM Application Note 29 – Interfacing a Memory System to the ARM7TDM Without Using AMBA

Practical Memory System (cont'd)



Summary

The ARM architecture design:

- The pipeline design increases the effective throughput.
- Use wait signals to extend the processor cycle when interfacing with slower memory devices and peripherals.
- Use seq and mreq# signals to provide bus cycle information.
  - Allow more sophisticated operations that permit fast sequential memory access.