

GBANA DESIGN DOC

Version 0.1.0

April 25, 2025

Contents

1. Preface	4
2. Classes	5
3. Block Diagram	6
4. Emulating the Clock & Cycle	7
5. Signals	8
6. Timing	11
6.1. Simple Memory Cycle ¹	11
6.2. N-Cycle ²	12
6.3. S-Cycle ³	13
6.4. I-Cycle ⁴	14
6.5. Merged IS-Cycle	15
6.6. Pipelined Addresses	16
6.7. Depipelined Addresses	16
6.8. Bidirectional Bus Cycle	16
6.9. Data Write Bus Cycle	17
6.10. Halfword Bus Cycle	17
6.11. Byte Bus Cycle	18
6.12. Reset Sequence	18
6.13. General Timing	19
6.14. Address Bus Enable Control	19
6.15. Bidirectional Data Write Cycle	20
6.16. Bidirectional Data Read Cycle	20
6.17. Data Bus Control	21
6.18. Configuration Pin Timing	21
6.19. Exception Timing	21
6.20. Synchronous Interrupt Timing	22
6.21. Memory Clock Timing	22
6.22. Address Latch Enable Control	22
6.23. Address Pipeline Enable Control	22
6.24. General Instruction Cycle	23
6.25. Branch and Branch with Link Instruction Cycle	23
6.26. Thumb Branch with Link Instruction Cycle	23
6.27. Branch and Exchange Instruction Cycle	24
6.28. Data Processing Instruction Cycle	24
6.29. Multiply and Multiply Accumulate Instruction Cycle	25
6.30. Load Register Instruction Cycle	26
6.31. Store Register Instruction Cycle	27
6.32. Load Multiple Register Instruction Cycle	27
6.33. Store Multiple Register Instruction Cycle	28

¹ARM DDI 0029G 3-4

²ARM DDI 0029G 3-5

³ARM DDI 0029G 3-6

⁴ARM DDI 0029G 3-7

6.34. Data Swap Instruction Cycle	29
6.35. Software Interrupt and Exception Instruction Cycle	29
6.36. Undefined Instruction Cycle	30
6.37. Unexecuted Instruction Cycle	30

1. Preface

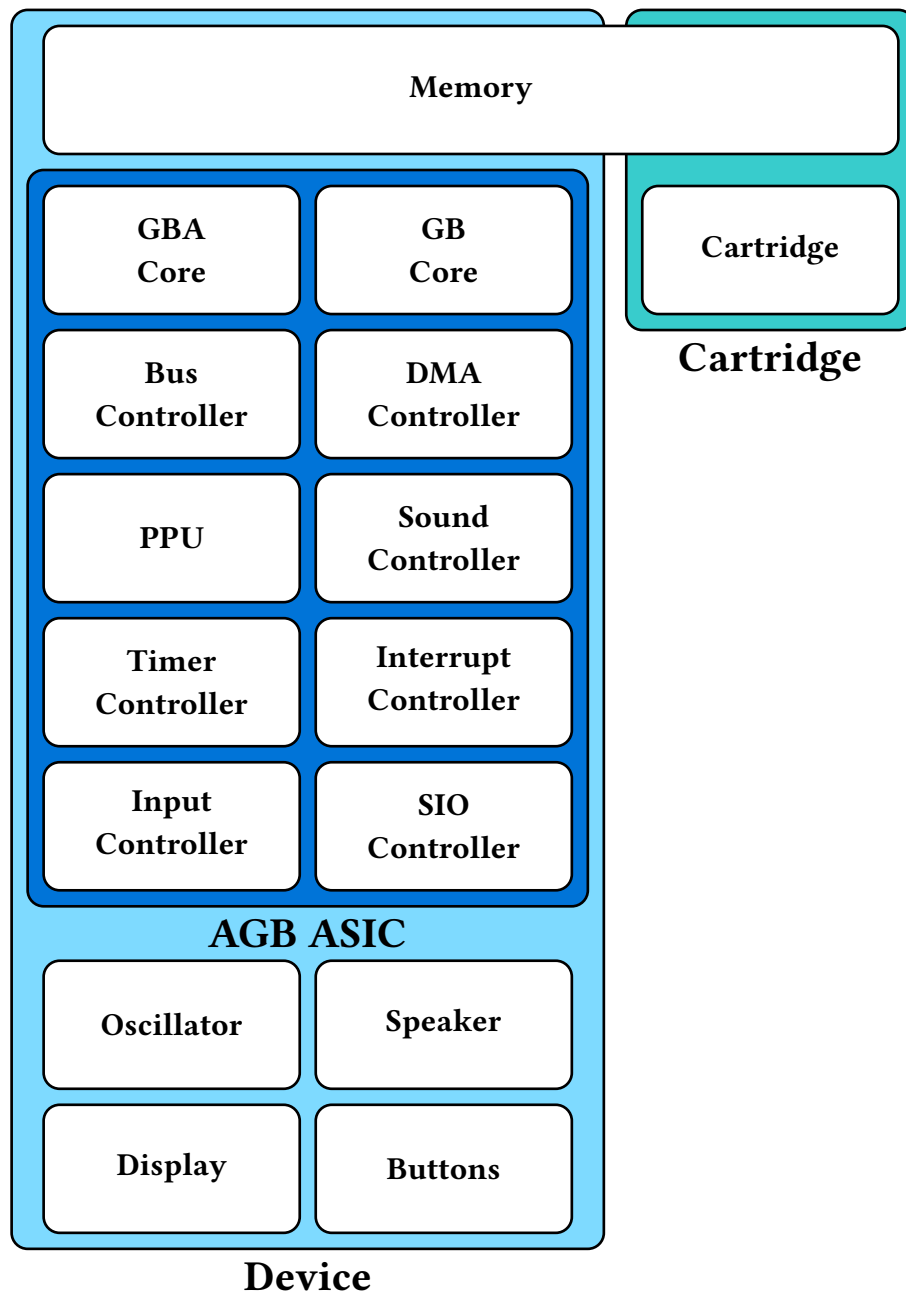
GBANA is a Game Boy Advance emulator. Other GBA emulators exist and they make games look and feel as good as they did on the original GBA. GBANA will make them look and feel *better* than they did on the original GBA.

2. Classes

Each file represents a class. The *singleton classes* correspond to the components in the block diagram on the next page. The *helper classes* contain procedures that are used by the singleton classes. The *instance classes* contain objects that are instantiated and used by the other classes.

NAME	FILE	TYPE	DESCRIPTION
GBA Core	<code>gba_core.odin</code>	Singleton Class	Emulating the ARM7TDMI core inside the AGB chip.
GB Core	<code>gb_core.odin</code>	Singleton Class	Emulating the SM83 core inside the AGB chip.
Bus Controller	<code>bus_controller.odin</code>	Singleton Class	Emulating the bus control logic inside the AGB chip.
DMA Controller	<code>dma_controller.odin</code>	Singleton Class	Emulating the DMA controller inside the AGB chip.
Memory Controller	<code>memory_controller</code>	Singleton Class	Emulating the memory controller inside the AGB chip.
Cpu	<code>cpu.odin</code>	Helper Class	Emulating the shared logic between the components inside the AGB chip.
Memory	<code>memory.odin</code>	Singleton Class	Emulating both the internal and external memory of the gba.
Buttons	<code>buttons.odin</code>	Singleton Class	Emulating the buttons of the GBA.
Cartridge	<code>cartridge.odin</code>	Singleton Class	Emulating a GBA cartridge.
Display	<code>display.odin</code>	Singleton Class	Emulating the display of the GBA.
GBA Isa	<code>gba_isa.odin</code>	Helper Class	Defining the ISA of the GBA core.
GB Isa	<code>gb_isa.odin</code>	Helper Class	Defining the ISA of the GB core.
PPU	<code>ppu.odin</code>	Singleton Class	Emulating the GBA PPU.
Line and Bus	<code>line_and_bus.odin</code>	Instance Class	Emulating the behavior of a line and a bus.
SIO Controller	<code>sio_controller.odin</code>	Singleton Class	Emulating the SIO controller.
Speakers	<code>speakers.odin</code>	Singleton Class	Emulating the speakers.
Util	<code>util.odin</code>	Helper Class	General utilities.

3. Block Diagram

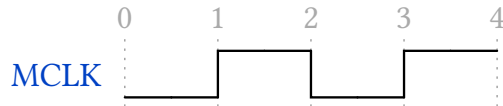


4. Emulating the Clock & Cycle

Related procedures:

`test_main_clock` Test procedure.

GBANA is phase-accurate. Every phase of every cycle is simulated (one tick simulates one phase). Synchronization of events within the phase need not match the real GBA, but at the end of each phase, the correct phase must be produced.



Main clock frequency: 16 MHz, ie approximately 16E6 cycles per second, or 62.5 ns per cycle. This is plenty time to emulate a single cycle on a modern computer. Each cycle has a low phase and a high phase. Each phase is one emulator tick. Each tick has two parts: a *start* part, where all the signals are updated and their callback functions are called, and an *interior* part, where the components execute their logic based on their internal state and the updated signals.

5. Signals

Components communicate by means of two types of interface: lines and buses. A line is just a boolean bus. There are two ways to affect a line/bus: (1) by *putting* data on it, and (2) by *forcing* data on it. Forcing updates the output value immediately. Putting schedules an update to the output value, to occur after a certain number of ticks.

Out of the signals defined in the ARM DDI 0029G, `D` is the only bidirectional signal so I got rid of it, in favor of `DIN` and `DOUT`.

Signal classes:

- clock
- address
- request
- response
- control

NAME	CLASS	COMPONENT	DESCRIPTION
<code>A</code>	Memory Interface	Memory	The 32-bit address bus. The CPU writes an address to this but, for memory access requests.
<code>ABE</code>	Bus Controls	Bus Controller	
<code>ABORT</code>	Memory Management Interface		The memory sets this to <i>high</i> to tell the CPU that the memory access request cannot be fulfilled.
<code>ALE</code>	Bus Controls	Bus Controller	
<code>BIGEND</code>	Bus Controls		
<code>BL</code>	Memory Interface	Memory	Byte latch control. A 4-bit bus where each bit corresponds to one of the bytes in a word. Used to indicate which part of the requested word is to be read/written.
<code>BUSDIS</code>	Bus Controls		
<code>BUSEN</code>	Bus Controls		
<code>DBE</code>	Bus Controls		
<code>DIN</code>	Memory Interface	GBA Core	Unidirectional input data bus.

NAME	CLASS	COMPONENT	DESCRIPTION
DOUT	Memory Interface	Memory	Unidirectional output data bus.
ENIN	Bus Controls	Bus Controller	
ENOUT	Bus Controls	Bus Controller	
FIQ	Interrupts	GBA Core	
ECLK	Clocks and Timing		MCLK exported from the core, for debugging. Has a small latency. Irrelevant for the emulator.
HIGHZ	Bus Controls	Bus Controller	
ISYNC	Interrupts	GBA Core	
IRQ	Interrupts	GBA Core	
LOCK	Memory Interface	Memory	Locks the memory, giving exclusive access to it to the CPU. This is effectively a mutex.
MAS	Memory Interface	Memory	Memory access size.
MCLK	Clocks and Timing		The main clock. Has two phases: a low phase and a high phase. Procedures can be constrained to any combination of these four: (1) the <i>start</i> of the low phase, (2) the <i>interior</i> of the low phase, (3) the <i>start</i> of the high phase, and (4) the <i>interior</i> of the high phase.
M	Processor Mode	GBA Core	
MREQ	Memory Interface	Memory	Set to <i>high</i> to indicate that the next cycle will be used to execute a memory request.
OPC	Memory Interface	Memory	This signal is used to distinguish between next-instruction-fetch and data-read/data-write. Set to <i>high</i> for instruction fetch request, set to <i>low</i> for data read/write requests.
RESET	Bus Controls		Used to start the processor. Must be held <i>high</i> for at least 2 cycles, with WAIT set to <i>low</i> .
RW	Memory Interface	Memory	This signal is used to distinguish between memory read and memory write. Set to <i>high</i> for read requests, set to <i>low</i> for write requests.

NAME	CLASS	COMPONENT	DESCRIPTION
SEQ	Memory Interface	Memory	Set to <i>high</i> to indicate that the address of the next memory request will be in the same word that was accessed in the previous memory access or the word immediately after it. Sequential reads require fewer memory cycles.
TBE	Bus Controls		
TBIT	Processor State	GBA Core	Set to <i>high</i> for Thumb mode, set to <i>low</i> for ARM mode.
TRANS	Memory Management Interface		Irrelevant.
WAIT	Clocks and Timing		This signal is used to insert wait cycles. Different memory regions have different access latency, which determines how many wait cycles need to be inserted.

6. Timing

Types of intervals in a timing diagram⁵:

- *Open Unshaded* - The line/bus is expected to remain stable throughout this interval.
 - *Writing* occurs at the *start* and is prohibited in the *interior*.
 - *Reading* is allowed at the *start* (by signals succeeding it in the tick order) and in the *interior*.
- *Open Shaded* - The line/bus is expected to change at an arbitrary time during this interval.
 - *Writing* is prohibited at the *start* and allowed in the *interior*.
 - *Reading* is allowed at the *start* and prohibited in the *interior*.
- *Closed* - The line/bus is disabled.
 - *Writing* is prohibited at the *start* and prohibited in the *interior*.
 - *Reading* is prohibited at the *start* and in the *interior*.

In request/response contexts, request data is displayed in blue, and response data is displayed in pink.

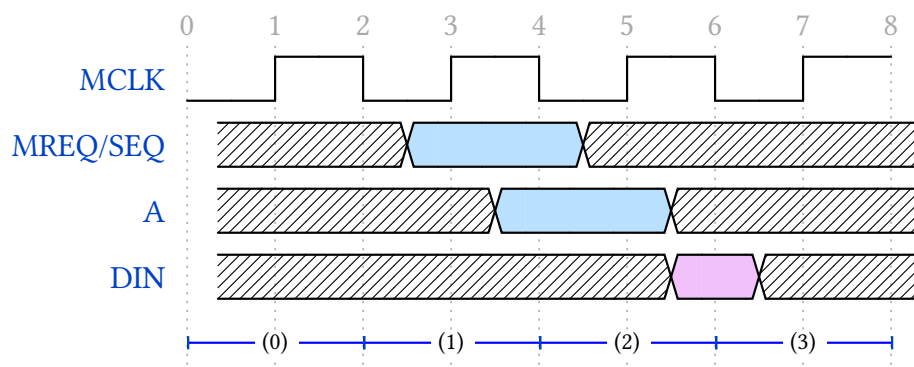
Types of cycles:

- (unidirectional) data read cycle
- (unidirectional) data write cycle
- internal cycle

6.1. Simple Memory Cycle⁶

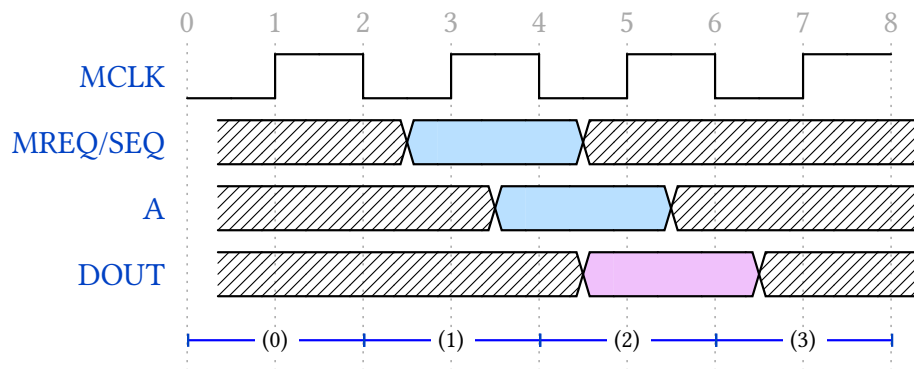
Related variables:

- `memory.memory_request` : This is `MREQ` .
- `memory.sequential_cycle` : This is `SEQ` .
- `memory.address` : This is `A` .
- `gba_core.data_in` : This is `DIN` .
- `memory.data_out` : This is `DOUT` .



⁵ARM DDI 0029G xix

⁶ARM DDI 0029G 3-4



Cycle (0) is the *pre-cycle*, cycle (1) is the *request cycle*, cycle (2) is the *response cycle*, and cycle (3) is the *post-cycle*. The term *memory cycle* refers to cycle (2).

General logic of a memory cycle:

- The CPU must write `MREQ` and `SEQ` during the interior of phase 1 of the request cycle.
- The Memory may read `MREQ` and `SEQ` during phase 2 of the request cycle and/or at the start of phase 1 of the response cycle.
- The CPU must write `A` during the interior of phase 2 of the request cycle.
- The Memory may read `A` during phase 1 of the response cycle and/or at the start of phase 2 of the response cycle.
- The Memory must write `D` during the interior of phase 2 of the response cycle.
- The CPU may read `D` at the start of phase 1 of the post cycle.

6.2. N-Cycle⁷

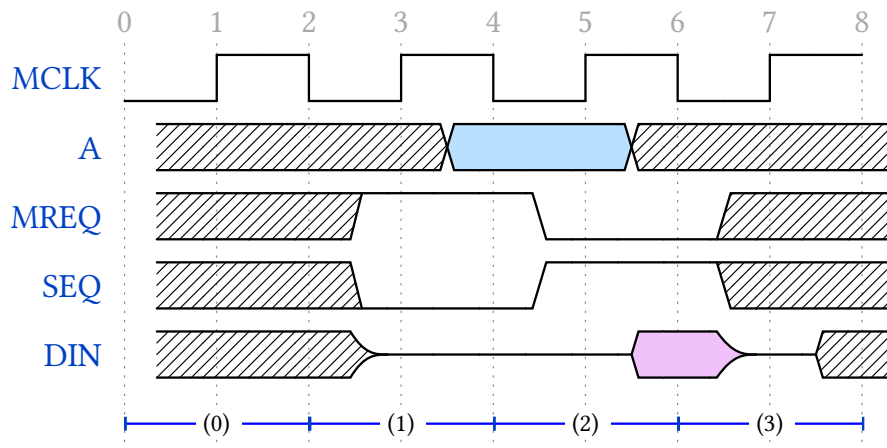
Related variables:

- `memory.memory_request` : This is `MREQ` .
- `memory.sequential_cycle` : This is `SEQ` .
- `memory.address` : This is `A` .
- `gba_core.data_in` : This is `DIN` .

Related procedures:

- `gba_initiate_n_cycle_request` : The **GBA Core** may call during phase 1 to initiate the request cycle.
- `memory_initiate_n_cycle_response` : The **Memory** must call this 2 ticks after to initiate the response cycle.
- `test_n_cycle` : Test procedure.

⁷ARM DDI 0029G 3-5



Cycle (0) is the *pre-cycle*, cycle (1) is the *request cycle*, cycle (2) is the *response cycle*, and cycle (3) is the *post-cycle*. The term *nonsequential memory cycle* refers to cycle (2).

Specific logic of an N-Cycle:

- General memory cycle logic.
- The CPU must set `MREQ` and `SEQ` to low during the interior of phase 1 of the request cycle.
- The Memory may extend phase 1 of the response cycle by setting the `WAIT` signal.

6.3. S-Cycle⁸

Related variables:

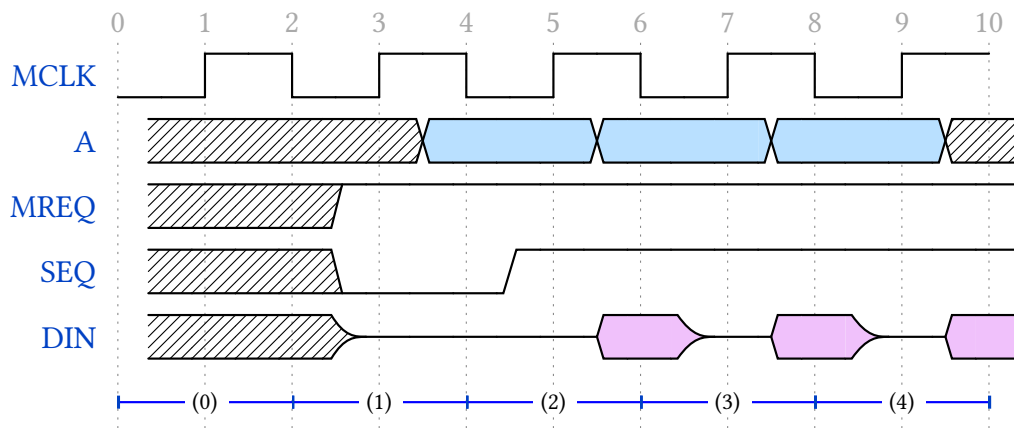
- `memory.memory_request` : This is `MREQ` .
- `memory.sequential_cycle` : This is `SEQ` .
- `memory.address` : This is `A` .
- `gba_core.data_in` : This is `DIN` .

Related procedures:

- `gba_initiate_s_cycle_request` : The **GBA Core** may call during phase 1 to initiate the request cycle.
- `memory_initiate_s_cycle_response` : The **Memory** must call this 2 ticks after to initiate the response cycle.
- `test_s_cycle` : Test procedure.

This is what a Sequential Memory Cycle (S-cycle) looks like:

⁸ARM DDI 0029G 3-6



Cycle (0) is the *pre-cycle*, cycle (1) is the *request cycle*, cycle (2) is the *N-response cycle*, cycle (3) is the 1st *S-response cycle*, cycle (4) is the 2nd *S-response cycle*, etc. The term *sequential memory cycle* refers to cycles (3), (4), etc.

Specific logic of an N-Cycle:

- General memory cycle logic.
- The CPU must set `MREQ` and `SEQ` to low during the interior of phase 1 of the request cycle.
- The CPU must set `SEQ` to high during the interior of phase 1 of the N-response cycle.
- The Memory may extend phase 1 of the response cycle by setting the `WAIT` signal.

6.4. I-Cycle⁹

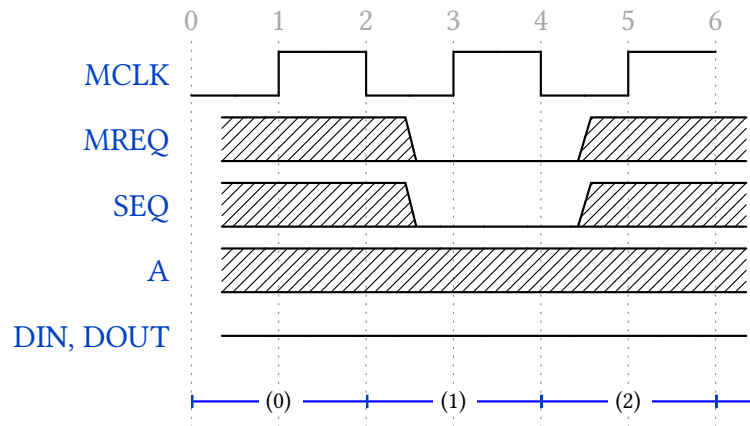
Related variables:

- `memory.memory_request` : This is `MREQ` .
- `memory.sequential_cycle` : This is `SEQ` .
- `memory.address` : This is `A` .
- `gba_core.data_in` : This is `DIN` .
- `gba_core.data_in` : This is `DOUT` .

Related procedures:

- `gba_initiate_i_cycle` : The **GBA Core** may call during phase 1 to initiate an I-cycle.
- `test_n_cycle` : Test procedure.

⁹ARM DDI 0029G 3-7



Cycle (0) is the *pre-cycle*, cycle (1) is the *internal cycle*, and cycle (2) is the *post-cycle*.

Specific logic of an N-Cycle:

- The CPU must set **MREQ** and **SEQ** to low during the interior of phase 1 of the request cycle.
- **D** must remain disabled.

6.5. Merged IS-Cycle

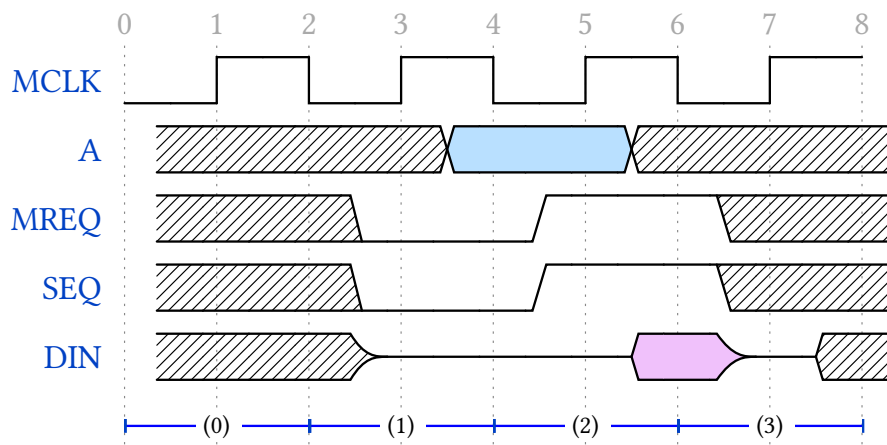
Related variables:

- **memory.memory_request** : This is **MREQ** .
- **memory.sequential_cycle** : This is **SEQ** .
- **memory.address** : This is **A** .
- **gba_core.data_in** : This is **DIN** .

Related procedures:

- **gba_initiate_merged_is_cycle** :
- **test_merged_is_cycle** : Test procedure.

This is what a Merged Internal-Sequential Memory Cycle (merged IS-cycle) looks like:



This looks the same as an N-Cycle, except the request cycle is merged with an I-cycle.

Specific logic of an N-Cycle:

- The CPU may put the address on the bus a cycle earlier, to give more time to the Memory to decode it.

6.6. Pipelined Addresses

This is irrelevant to the emulator, since the GBA uses SRAM and the intended address timing for SRAM is depipelined.

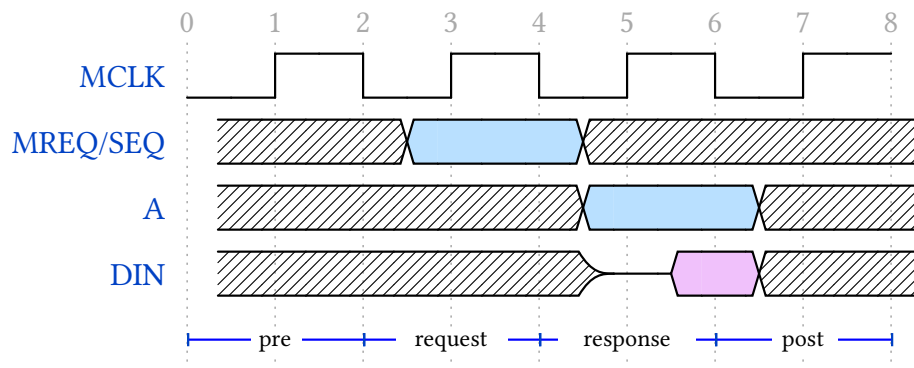
6.7. Depipelined Addresses

Related variables:

- `memory.memory_request` : This is `MREQ` .
- `memory.sequential_cycle` : This is `SEQ` .
- `memory.address` : This is `A` .
- `gba_core.data_in` : This is `DIN` .

Related procedures:

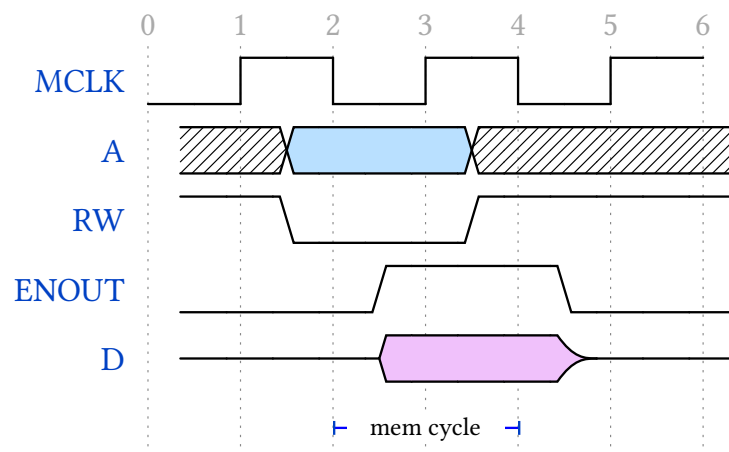
- `test_depipelined_addresses` : Test procedure. Manually initialize the request cycle and verify the response and post cycle.



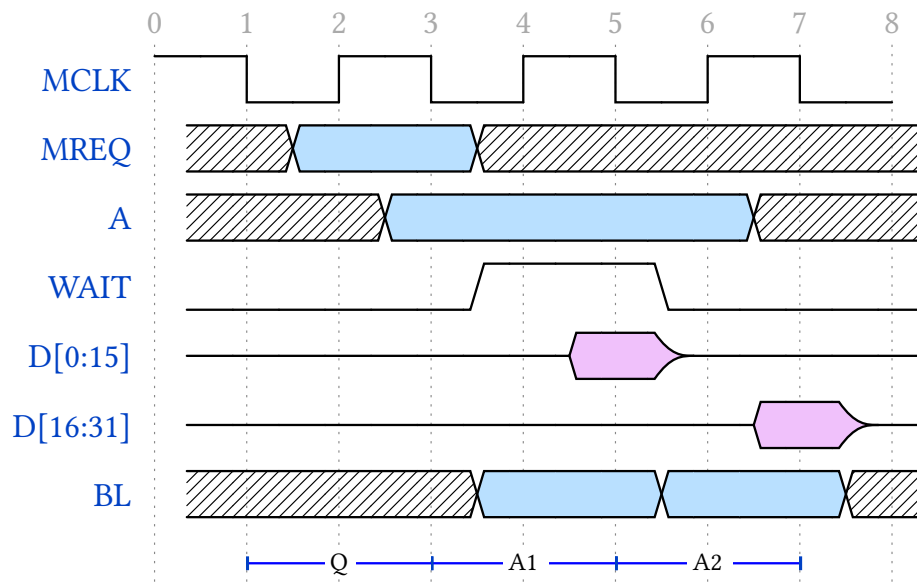
6.8. Bidirectional Bus Cycle

This is irrelevant to the emulator.

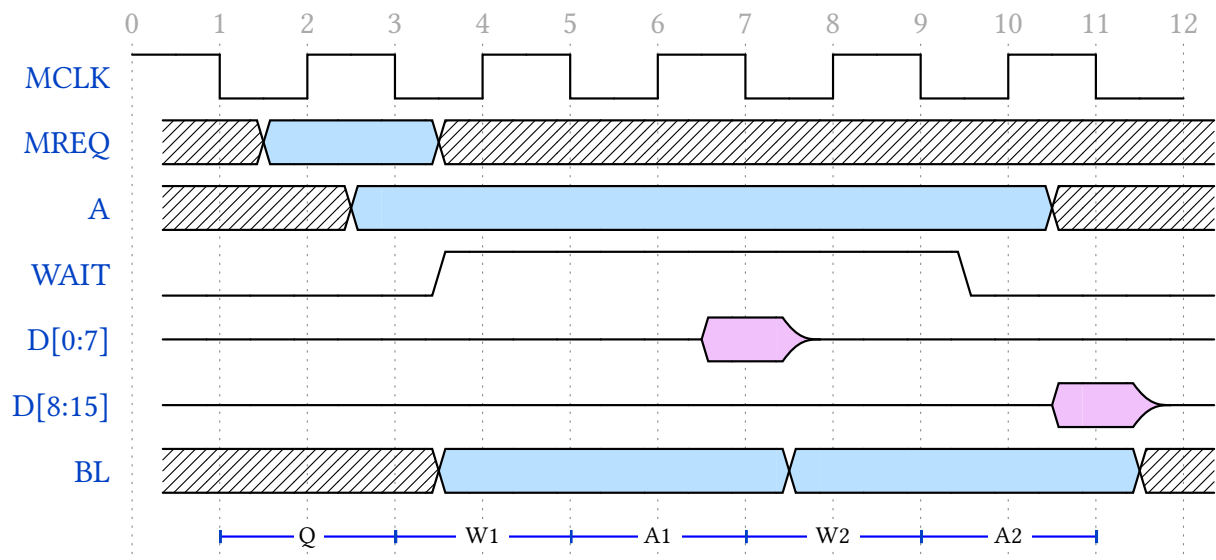
6.9. Data Write Bus Cycle



6.10. Halfword Bus Cycle

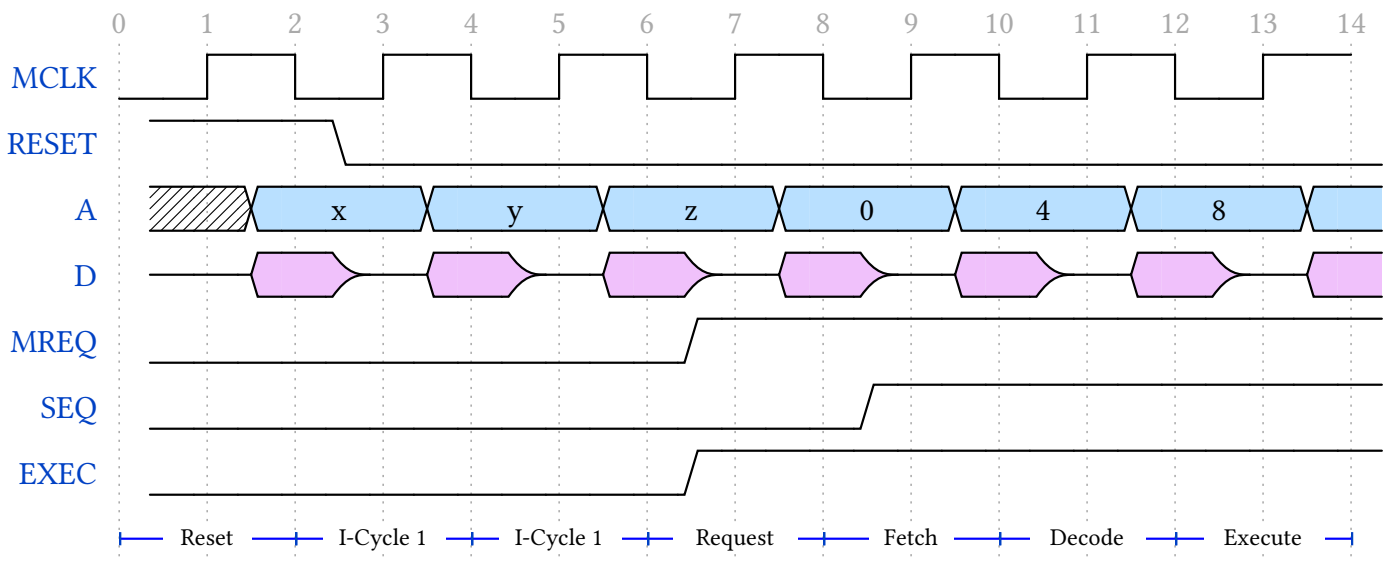


6.11. Byte Bus Cycle

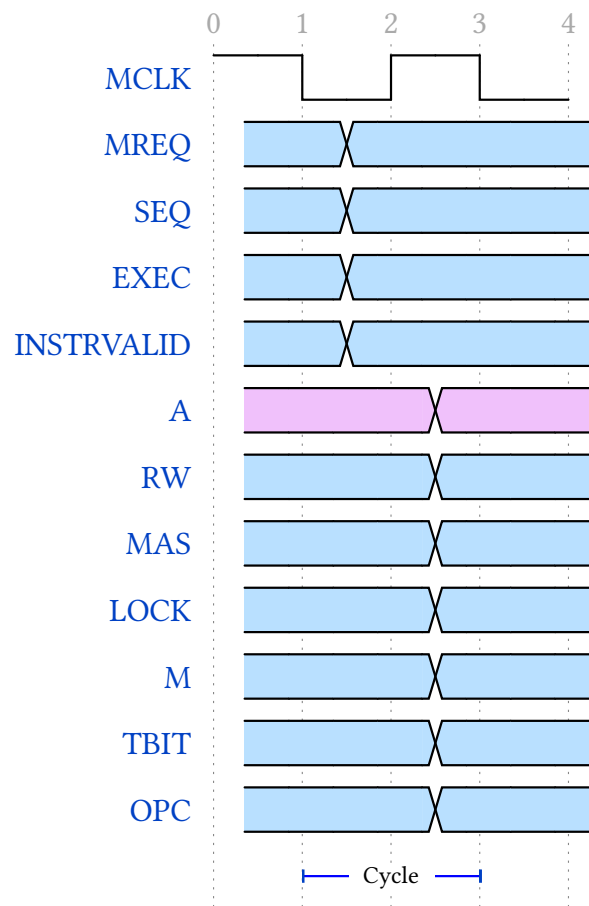


6.12. Reset Sequence

The reset sequence should look like this:

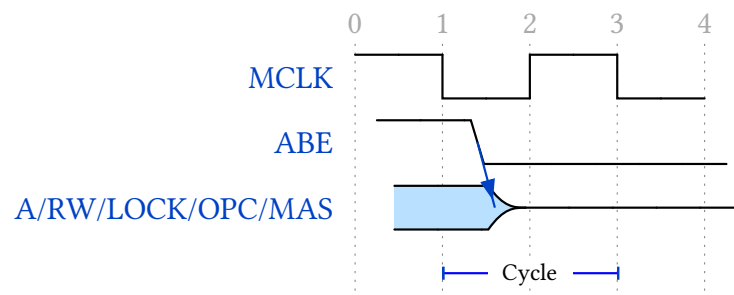


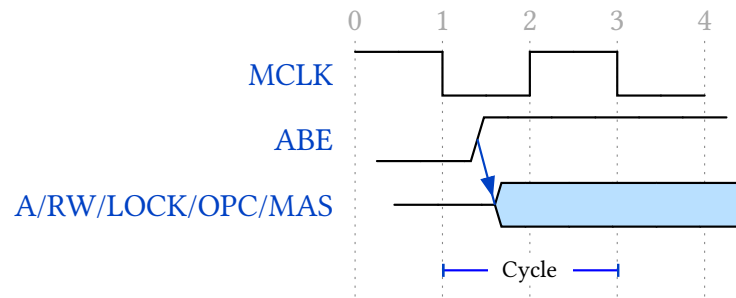
6.13. General Timing



1. **MREQ**, **SEQ**, **EXEC**, and **INSTRVALID** may only be updated at the start of or in the interior of phase 1.
2. **A**, **RW**, **MAS**, **LOCK**, **M**, **TBIT**, and **OPC** may only be updated at the start of or in the interior of phase 2.

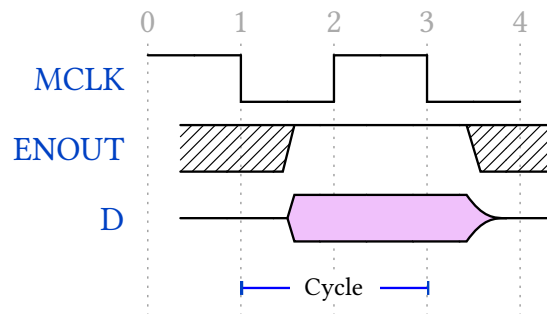
6.14. Address Bus Enable Control





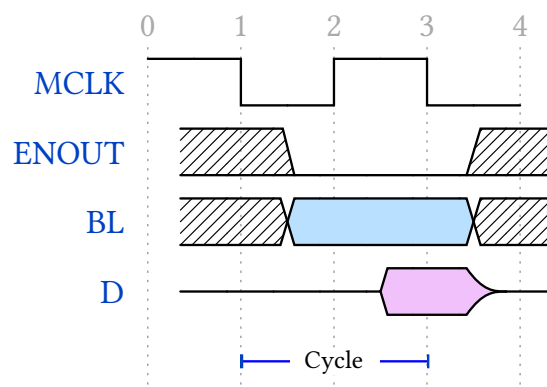
- ABE can change during phase 1.
- A , RW , LOCK , OPC , and MAS are enabled/disabled immediately when ABE switches to high/low.
- A , RW , LOCK , OPC , and MAS must be stable at the starts of both phases.

6.15. Bidirectional Data Write Cycle



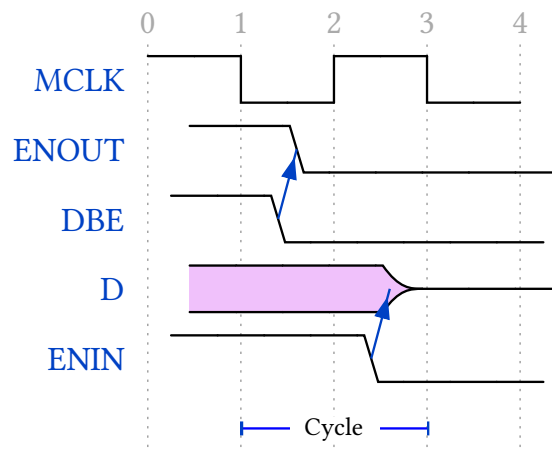
1. The CPU must enable ENOUT during the interior of phase 1.
2. The data must remain stable until the end of phase 1 of the post-cycle.

6.16. Bidirectional Data Read Cycle



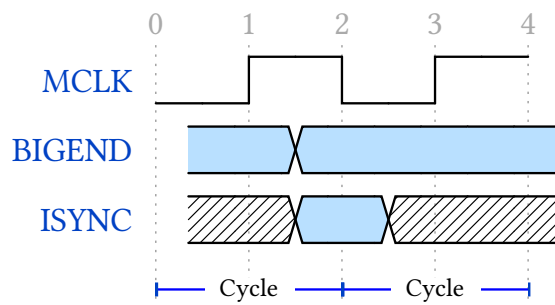
1. The CPU must disable ENOUT during the interior of phase 1.
2. The data must remain stable until the end of phase 1 of the post-cycle.

6.17. Data Bus Control



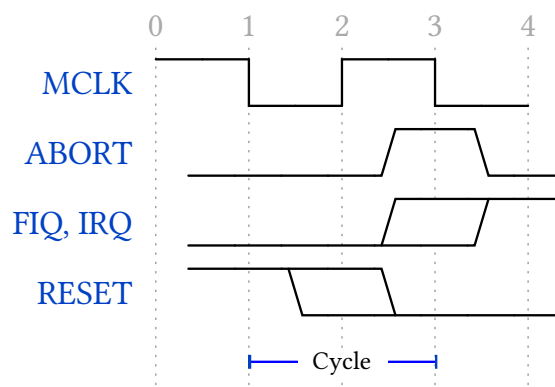
1. ENIN immediately disables D when it goes low.
2. ENOUT doesn't affect D.
3. DBE immediately disables ENOUT when it goes low.

6.18. Configuration Pin Timing

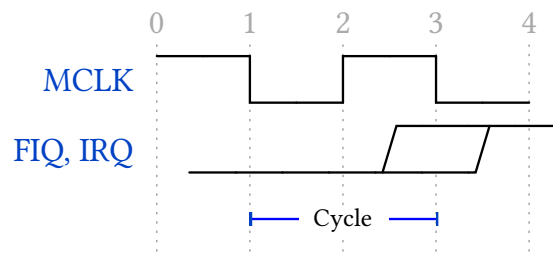


1. BIGEND may be updated during phase 2.
2. ISYNC must be stable at the start of phase 1, it may be written at any other time.

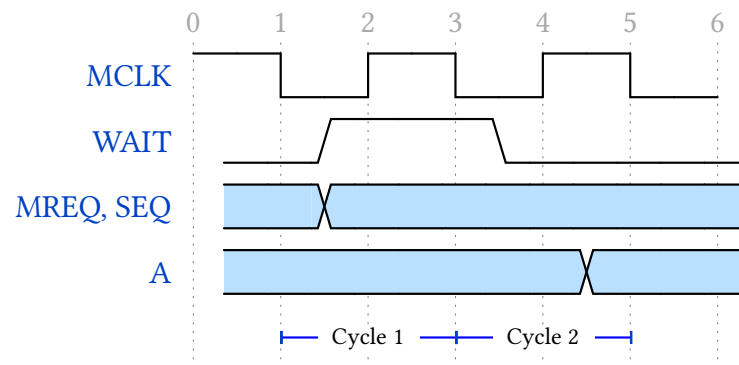
6.19. Exception Timing



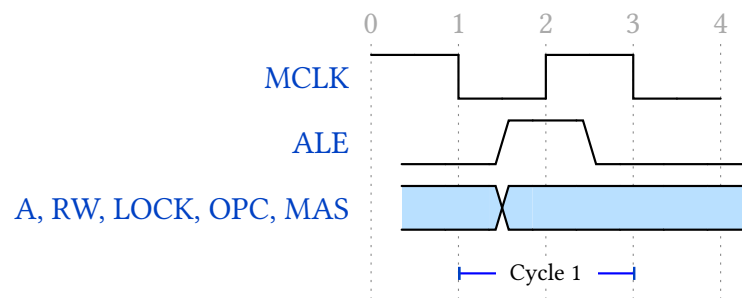
6.20. Synchronous Interrupt Timing



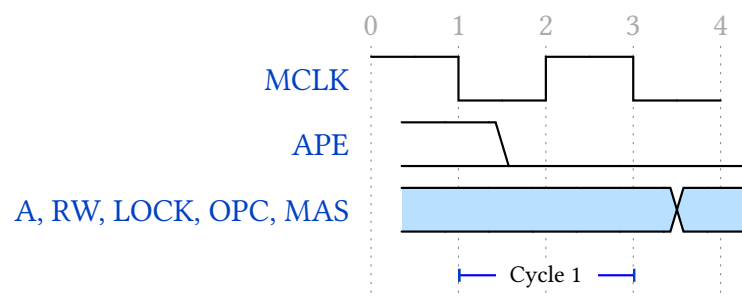
6.21. Memory Clock Timing



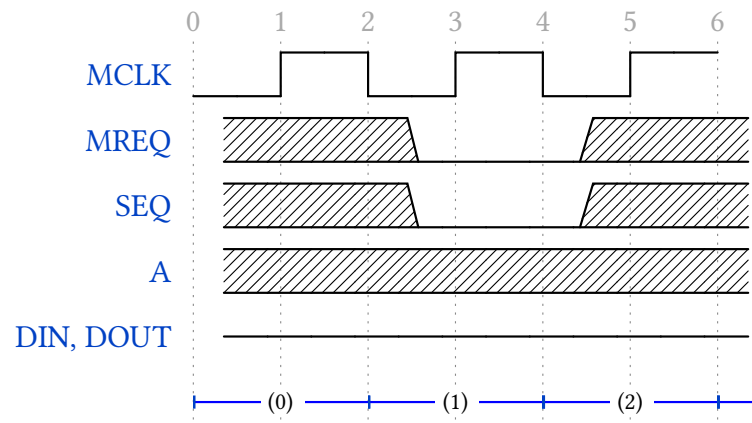
6.22. Address Latch Enable Control



6.23. Address Pipeline Enable Control



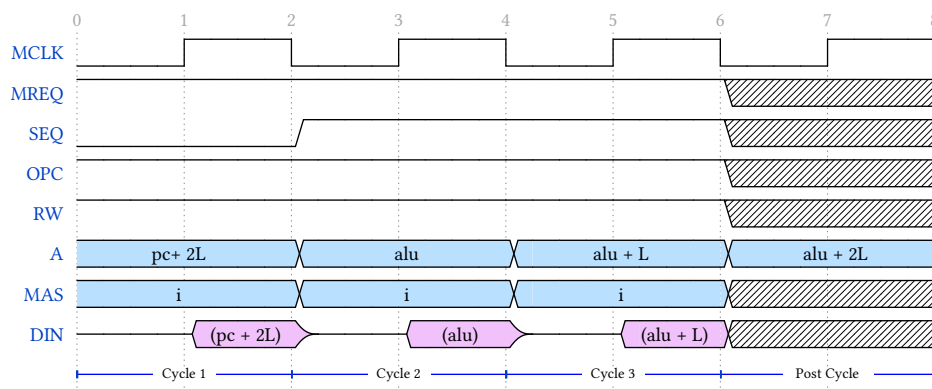
6.24. General Instruction Cycle



- There are two types of request signals: request type signals and request address signals, which are broadcast at least one tick ahead of the response cycle.
- The request type signals (**MREQ** and **SEQ**) are pipelined up to 2 ticks ahead of the cycle to which they apply.
- The request address signals (**A** , **MAS** , **RW** , **OPC** , and **TBIT**) are pipelined up to 1 tick ahead of the cycle to which they apply.
- The instruction cycle is the response cycle.
- When **OPC** is high, the address is incremented each cycle (epistemic status: *guess*).

6.25. Branch and Branch with Link Instruction Cycle

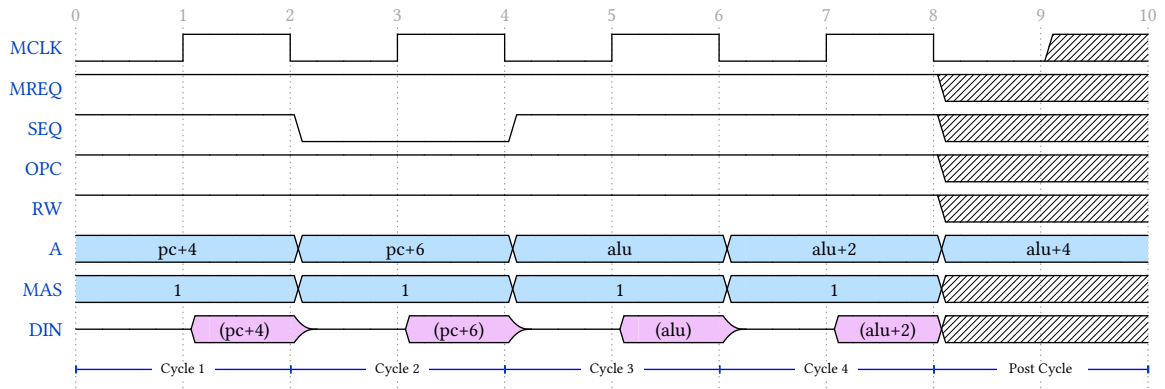
Related instructions: **B** , **BL**



6.26. Thumb Branch with Link Instruction Cycle

Related instructions:

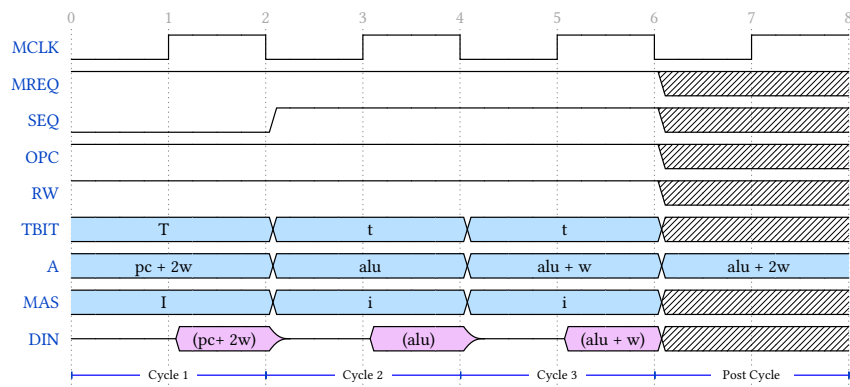
The Thumb instruction set is not needed for executing the BIOS.



6.27. Branch and Exchange Instruction Cycle

Related instructions: BX

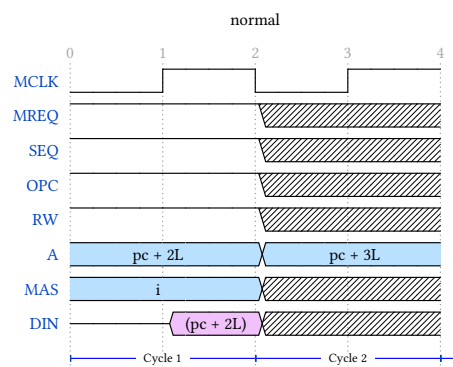
In which phase is TBIT allowed to change? Add a bit set field in the signal struct indicating which phase the signal is allowed to change in and assert during its tick function that this rule is met.

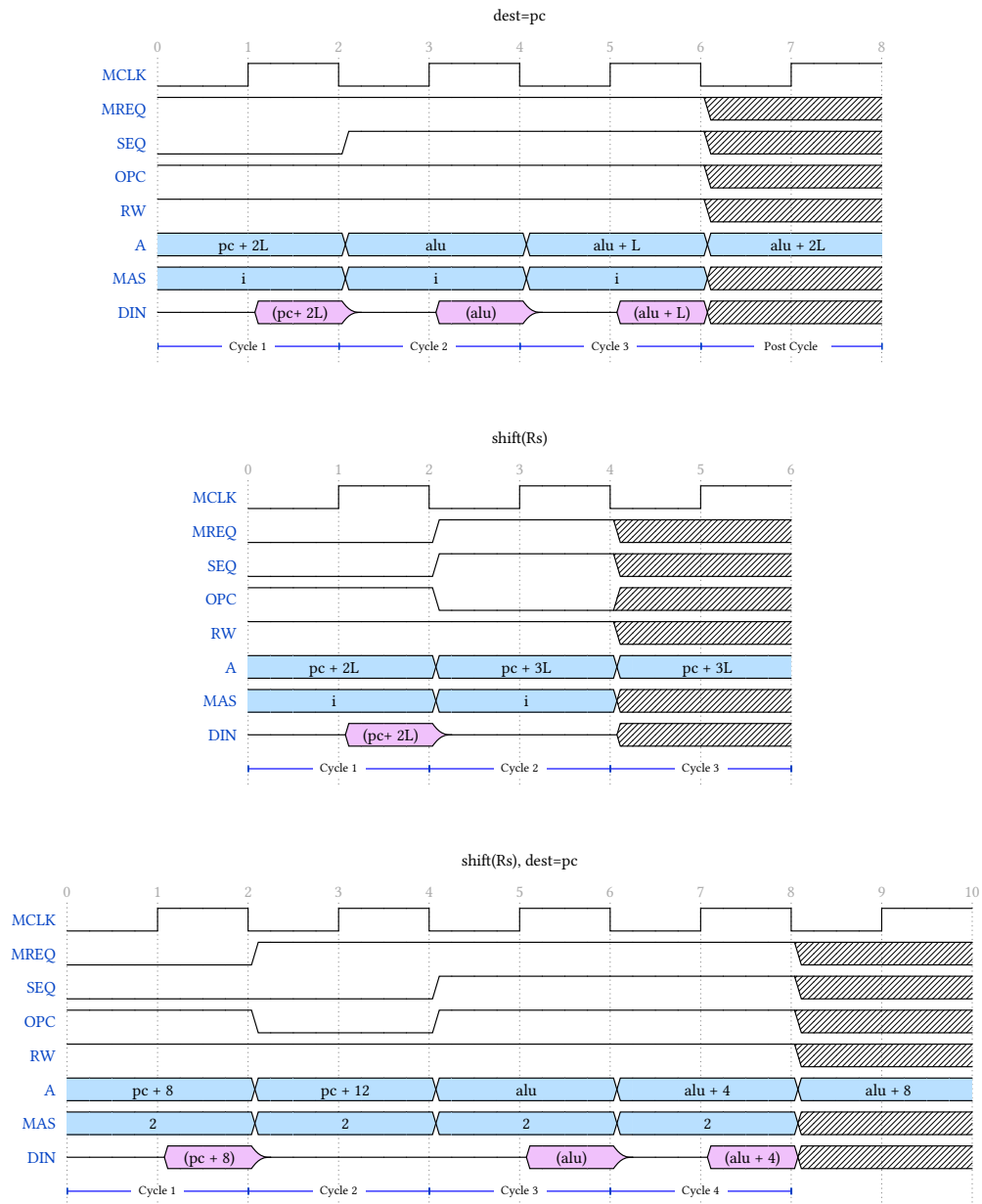


6.28. Data Processing Instruction Cycle

Related instructions: ADC, ADD, AND, BIC, CMN, CMP, EOR, MOV, MRS, MSR, MVN, ORR, RSB, RSC, SBC, SUB, TEQ, TST

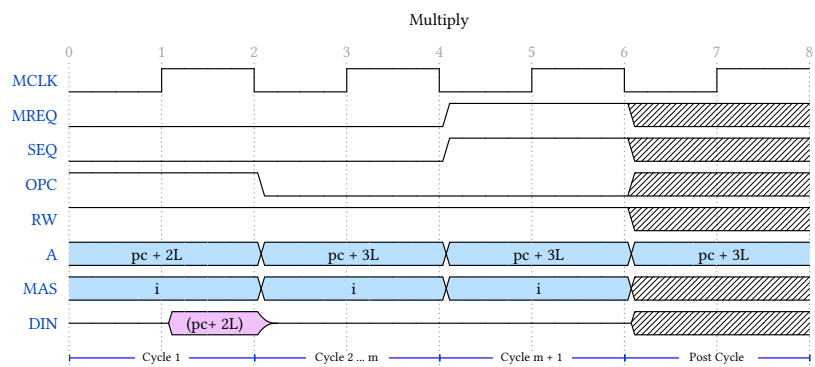
It seems like whenever SEQ is high, the address will always increment in the post cycle.



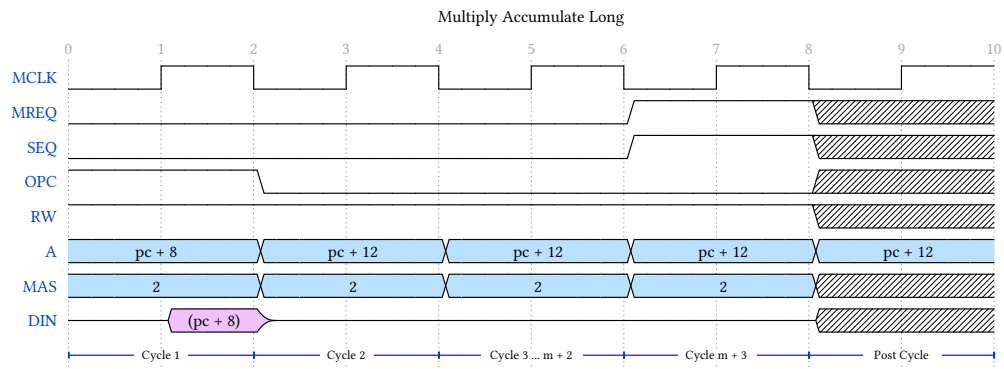
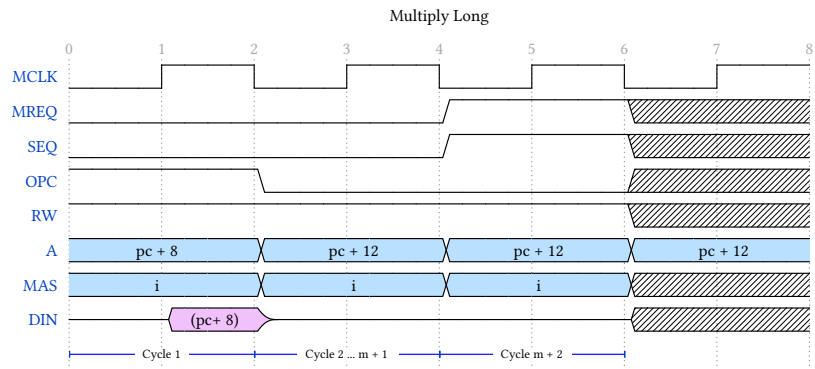
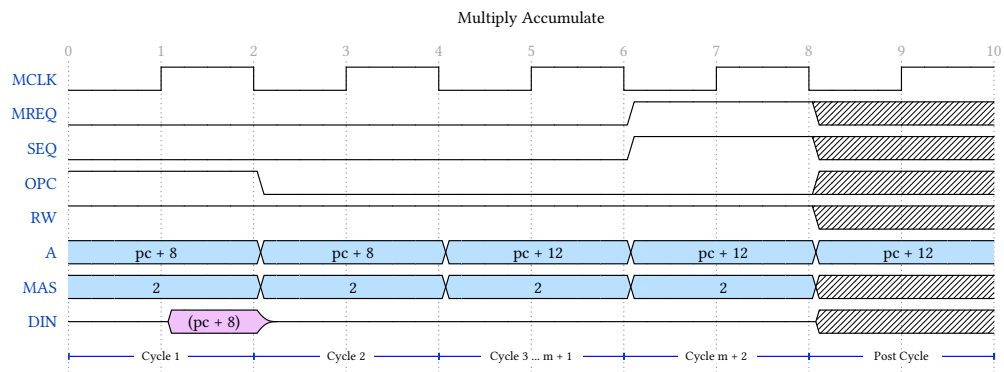


6.29. Multiply and Multiply Accumulate Instruction Cycle

Related instructions: `MLA` , `MUL` , `SMLAL` , `SMULL` , `UMLAL` , `UMULL`

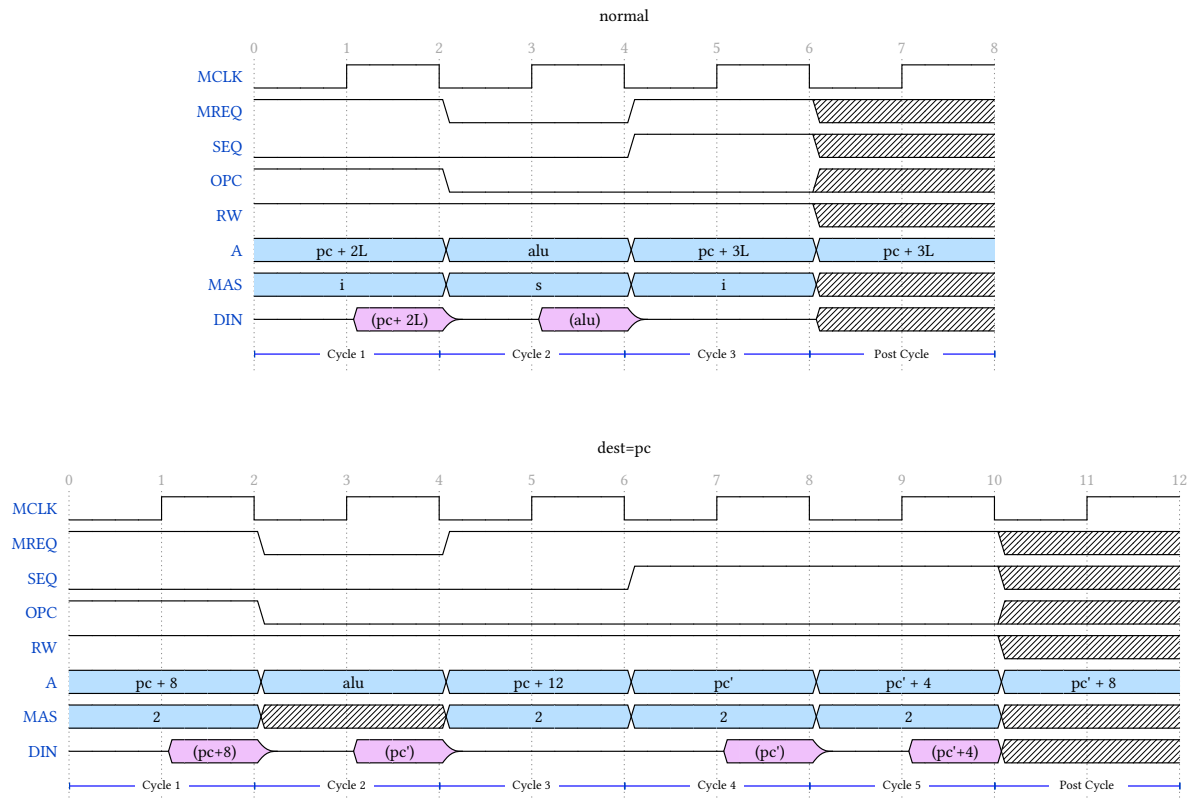


Cycles 2 to m are identical.



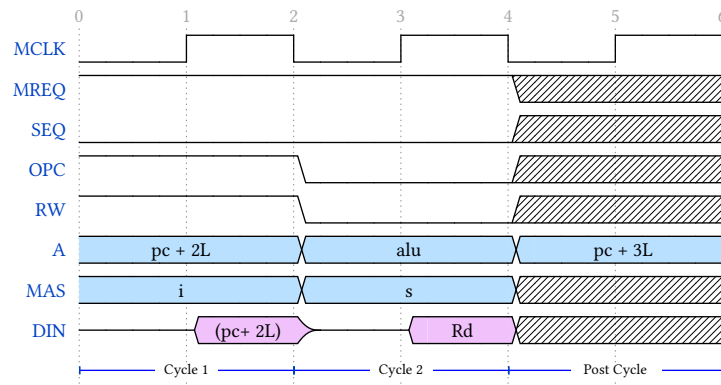
6.30. Load Register Instruction Cycle

Related instructions: LDR , LDRB , LDRBT , LDRH , LDRSB , LDRSH , LDRT



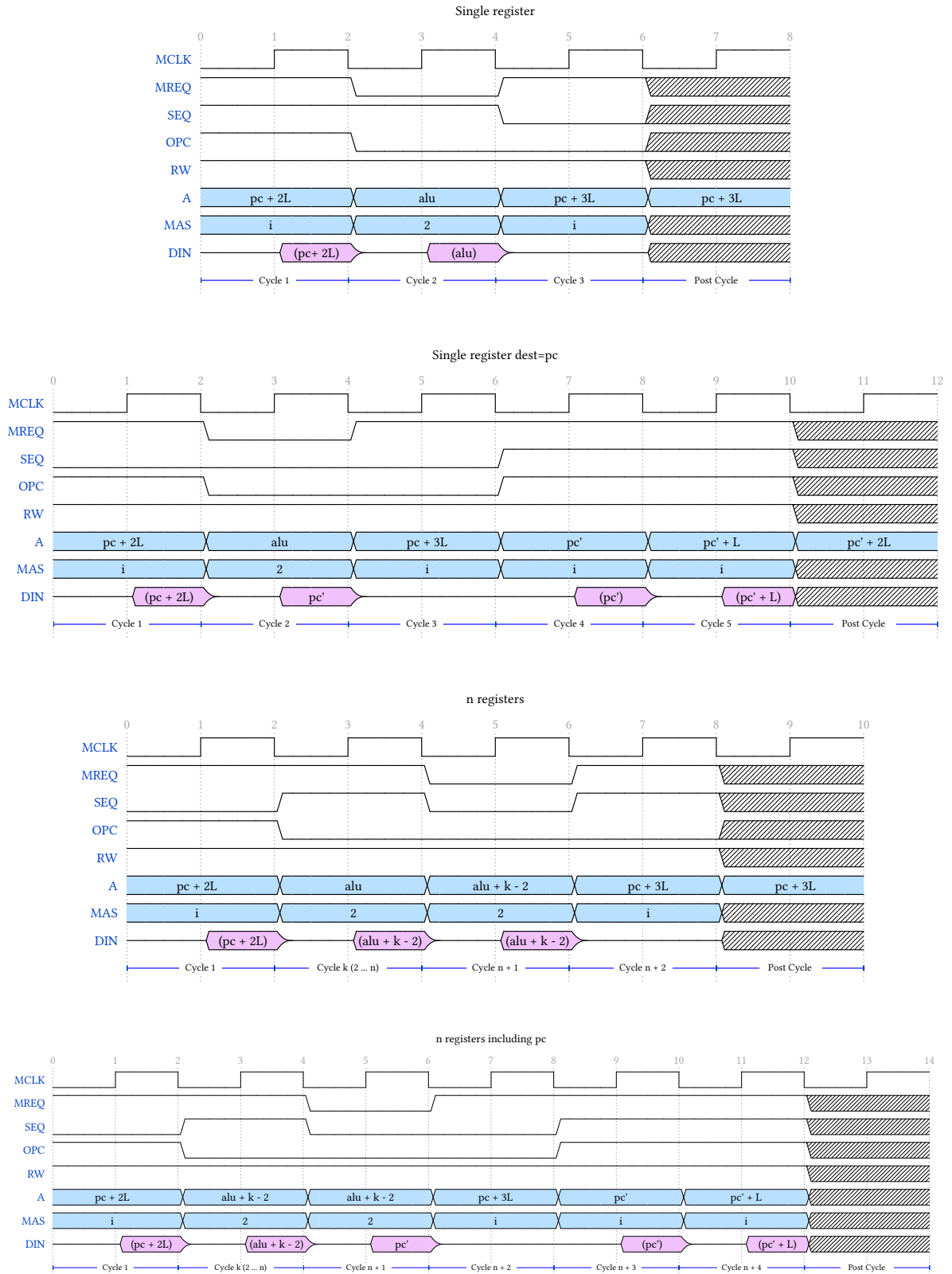
6.31. Store Register Instruction Cycle

Related instructions: STR , STRB , STRBT , STRH , STRT



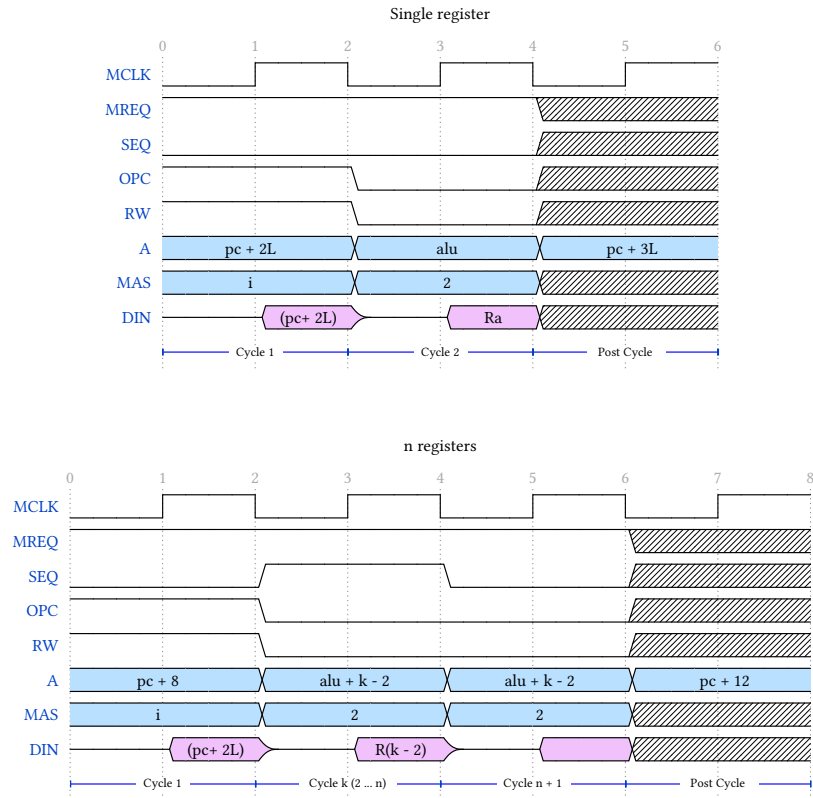
6.32. Load Multiple Register Instruction Cycle

Related instructions: LDM



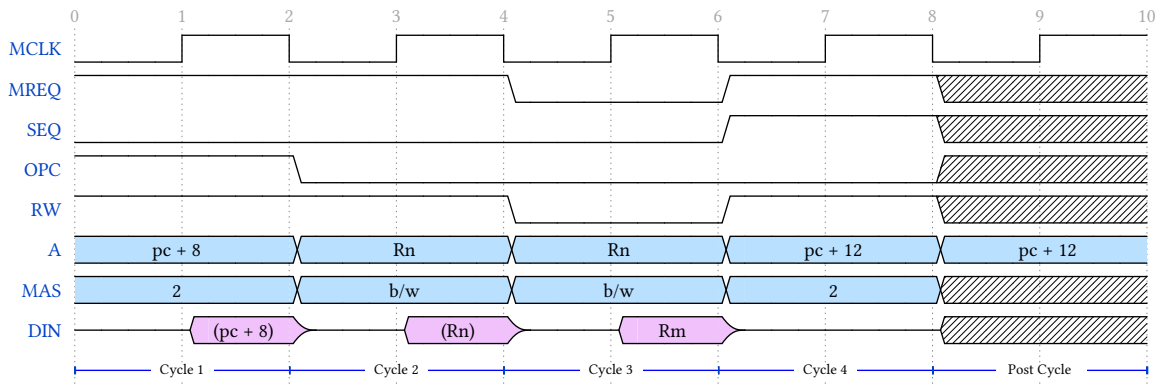
6.33. Store Multiple Register Instruction Cycle

Related instructions: STM



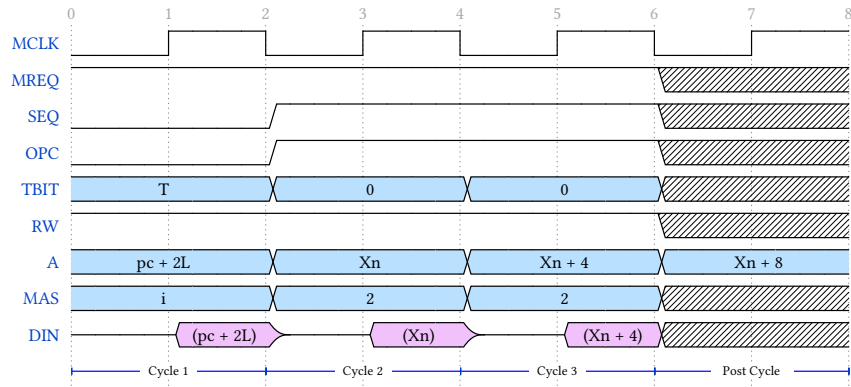
6.34. Data Swap Instruction Cycle

Related instructions: SWP , SWPB



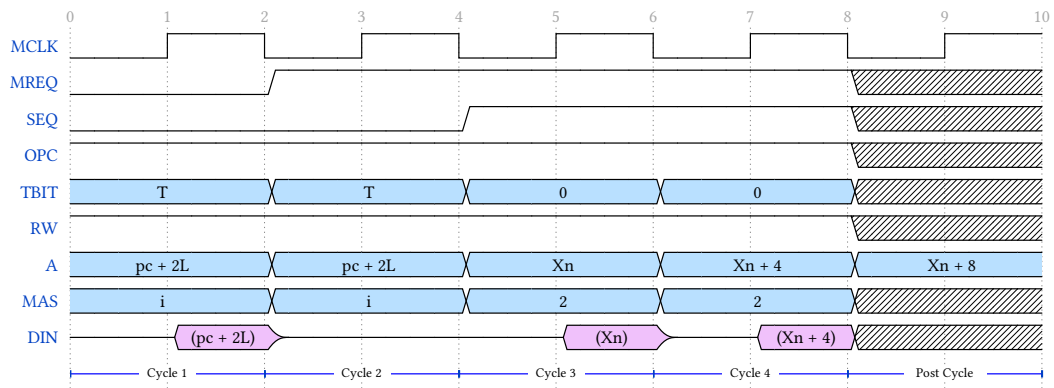
6.35. Software Interrupt and Exception Instruction Cycle

Related instructions: SWI



6.36. Undefined Instruction Cycle

Related instructions:



6.37. Unexecuted Instruction Cycle

Related instructions:

