ARM

Nintendo DS Dev. Con. '04

Using the ARM946E-S Processor Core

Agenda

Overview

New Instruction In detail

Pipeline & Avoiding Interlocks

Cache and TCM

Ed’s General Hints and Tips

Internal Processor Features

- **ARM946E-S contains an internal ARM9E-S processor core**
  - Five stage pipeline
  - Harvard bus architecture
    - Separate Instruction and Data Paths
  - Cache
    - Separate Instruction and Data side Cache's
    - With lock down feature
  - Tightly Coupled Memory
    - For both Instruction and Data
  - Architecture v5TE
    - Thumb extensions
    - Saturated maths
  - Extended 32×16 multiplier
    - single cycle 32×16 and 16×16 multiplies

ARM v5TE ISA Features

- **Architecture v5TE ISA contains full v4T ARM and Thumb instruction sets plus:**
  - Improved support for interworking
    - Covered in ARM / Thumb Interworking module
  - Breakpoint instructions (ARM and Thumb)
  - Count Leading Zeros instruction
  - Extended coprocessor instructions - MCR2 etc.
  - Support for saturated mathematics
  - Packed half-word signed multiplication instructions
  - Double-word Load / Store instructions
  - Cache Preload instruction
  - Double-word coprocessor transfer instructions - MCRR/MRRC

Agenda

Overview

New Instruction In detail

Pipeline & Avoid Interlocks

Cache and TCM

Ed's General Hints and Tips

Count Leading Zeros

- `CLZ{cond} Rd, Rm`
  - returns number of binary zero bits before the first binary one bit in a register value
  - source register is scanned from most significant bit to least significant bit
  - executes in 1-cycle (ARM9E-S/ARM102x)
  - result is 32 if no bits set, zero if bit 31 is set
- **Left shift of Rm by Rd will normalize Rm**
- **Signed normalize requires 1 extra cycle**

```
EOR R1, R0, R0, LSL#1
CLZ R1, R1
MOV R0, R0, LSL R1
```

R0 = 0000 0010 1110 1101...0

```
CLZ R1, R0
```

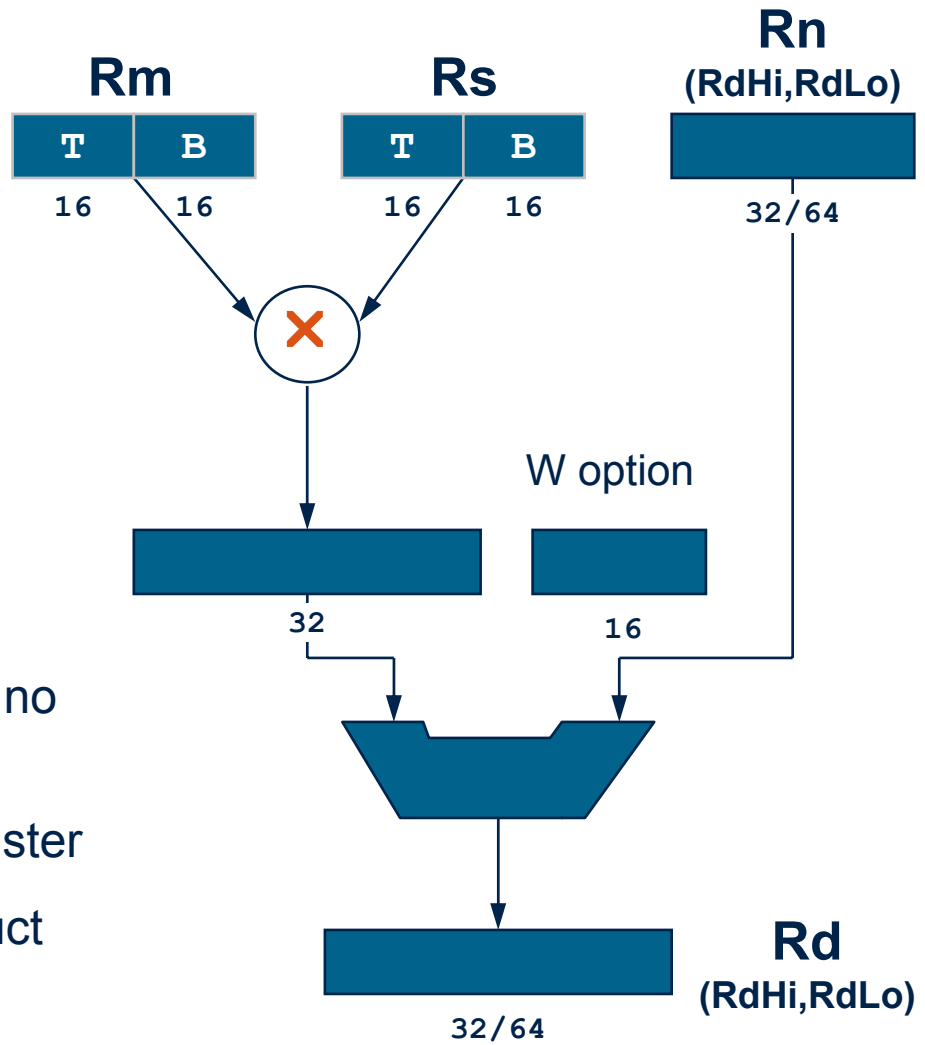
R1 = 0x6

```
MOV R0, R0 LSL R1
```

Rm = 1011 1011 0100 0000...0

New Signed Multiply Operations

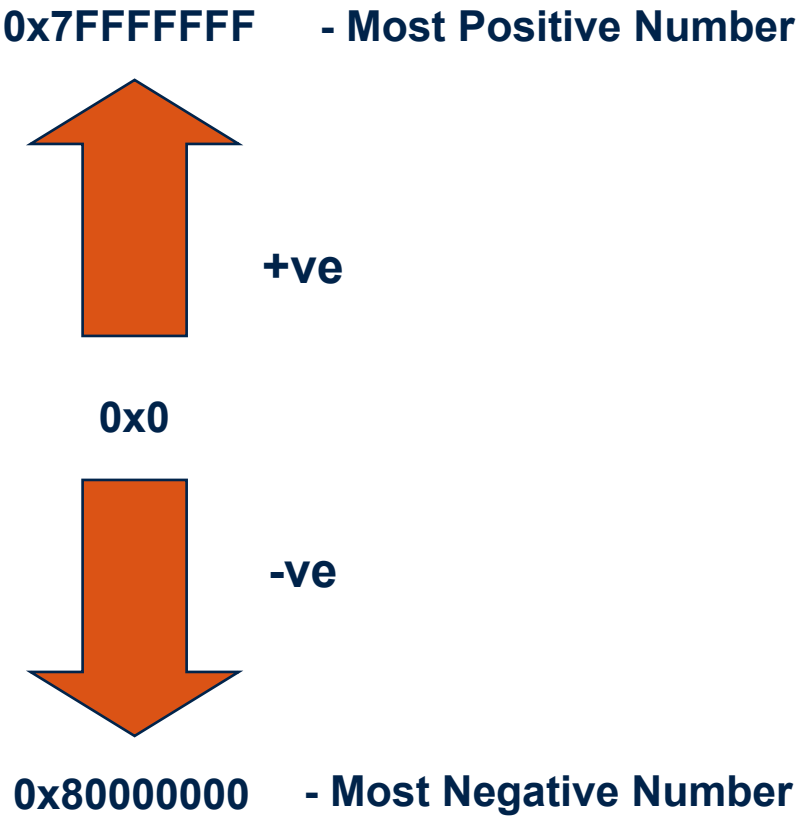
- SMULxy{cond} Rd, Rm, Rs
- SMULWy{cond} Rd, Rm, Rs
- SMLAxy{cond} Rd, Rm, Rs, Rn
- SMLAWy{cond} Rd, Rm, Rs, Rn
- SMLALxy{cond} RdLo, RdHi, Rm, Rs



- Q flag is affected for SMLA instructions (but no saturation)
- x, y selects either Top or Bottom half of register
- W selects the upper 32 bits of a 48-bit product
- Do not affect NZCV ('S' is not allowed)

Saturated Maths

- Adding 1 to `0x7FFFFFFF` causes a transition from a positive value to a negative value
- Subtracting 1 from `0x80000000` causes a transition from a negative value to a positive value
- Saturating instructions recognize this type of event and saturate the result at either the most negative values for subtract, or the most positive value for addition
- Often used to represent 1 to -1
  - “Q31” arithmetic
  - AXD can display Q31 format



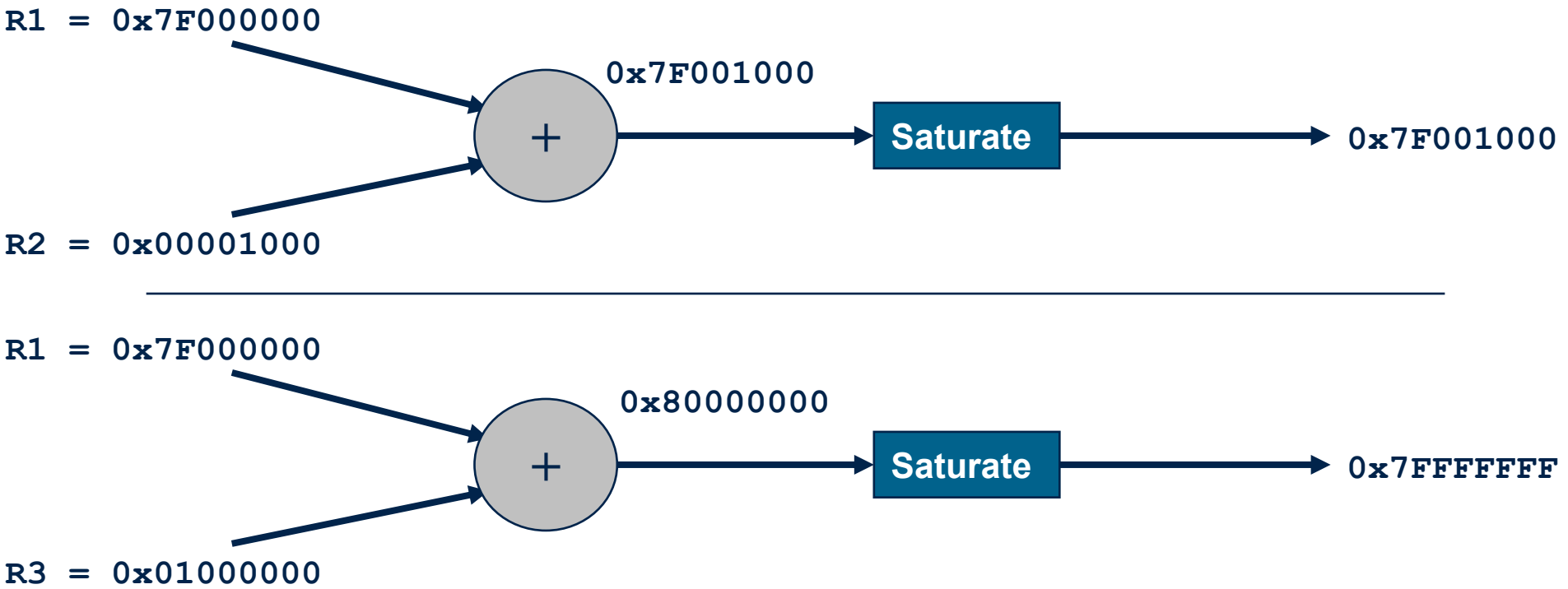
Saturated Maths Instructions

- **Saturation is required by several telecom DSP algorithms**
  - G.723.1 - VoIP
  - AMR - Adaptive MultiRate

`QSUB{cond} Rd, Rm, Rn`	Rd = saturate(Rm - Rn)
`QADD{cond} Rd, Rm, Rn`	Rd = saturate(Rm + Rn)
`QDSUB{cond} Rd, Rm, Rn`	Rd = saturate(Rm - saturate(Rn×2))
`QDADD{cond} Rd, Rm, Rn`	Rd = saturate(Rm + saturate(Rn×2))

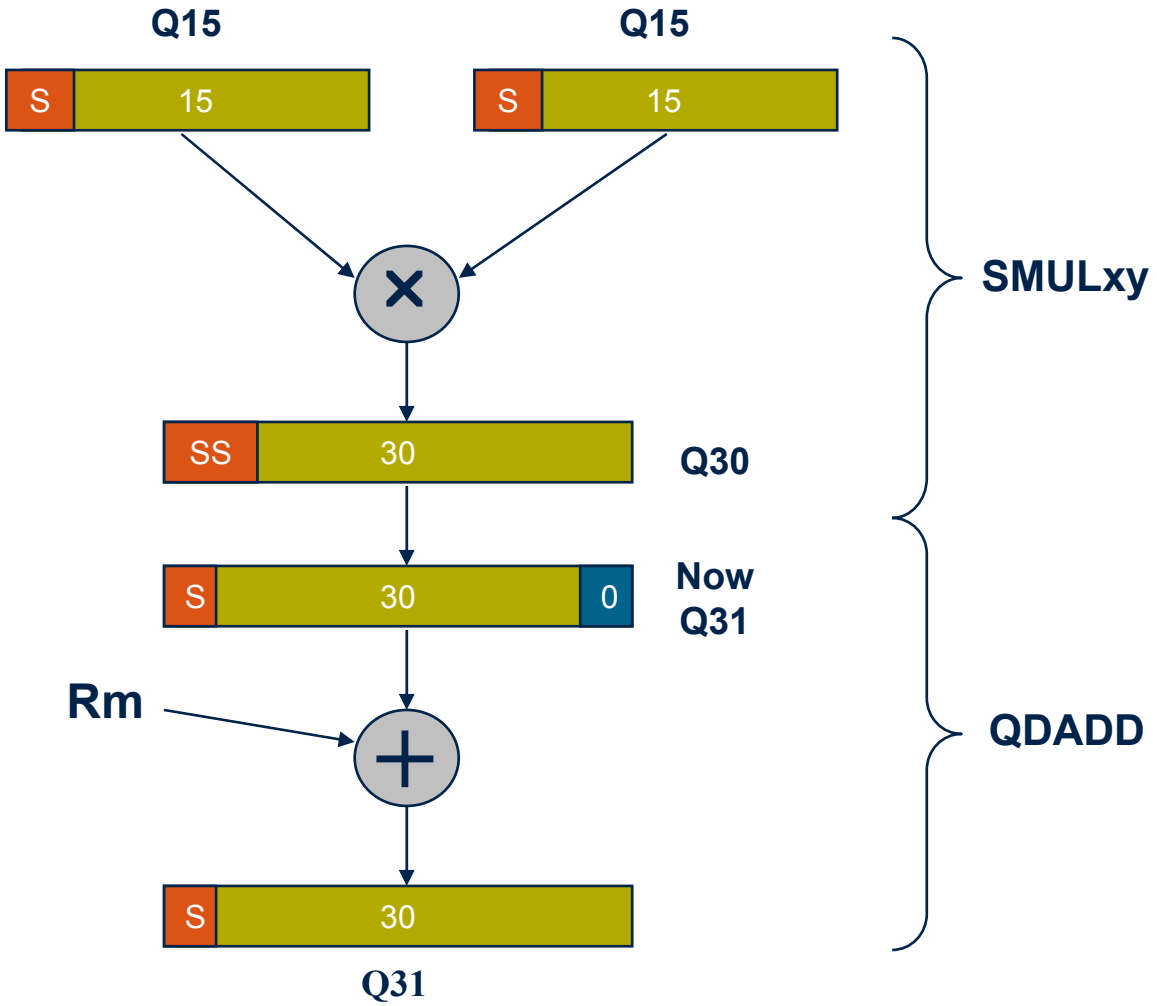
- **Q flag will be set if saturation occurs during these instructions**

QADD Example



- Example 1 does not cross the most positive number boundary and therefore no saturation take place
- Example 2 crosses the most positive number boundary and the result is saturated and the Q flag will be set

QDADD Example



- Inputs into the multiply represent numbers between 1 and -1
- Result of multiply is in Q30 format**
- QDADD converts Rn to Q31 format before performing accumulation

** Note ARM handles the case of -1*-1 correctly

Load / Store Double Registers

`LDR/STR{<cond>}D <Rd>, <addressing_mode>`

- Transfer two adjacent words in memory to / from any of the registers pairs (r0,r1), (r2,r3), (r4,r5), (r6,r7), (r8,r9), (r10,r11) or (r12,r13)
- Rd specifies the even numbered register. The immediately following odd numbered registers is used for the second transfer.
- Use same addressing modes as LDRH/STRH
- Address is that of the lower of the two words loaded by the `LDRD` instruction. The address of the higher word is generated by adding 4 to this address.
- Address must be doubleword (8-byte) aligned.

Cache Preload

```
PLD [Rn,<offset>]
```

- **Offset can be**
  - An unsigned 12-bit immediate value (ie 0 - 4095 bytes).
  - A register, optionally shifted by an immediate value
- **This can be either added or subtracted from the base register:**
  - Prefix the offset with '**+**' (default) or '**-**'.

- **Tells the memory system that an access to the data at a specified address is likely to occur soon.**
- **Memory system can bring the data into cache ready for future accesses.**
- **PLD is a hint instruction. On memory systems that do not support this operations, it will behave as a NOP.**
- **Unconditional**

Agenda

Overview

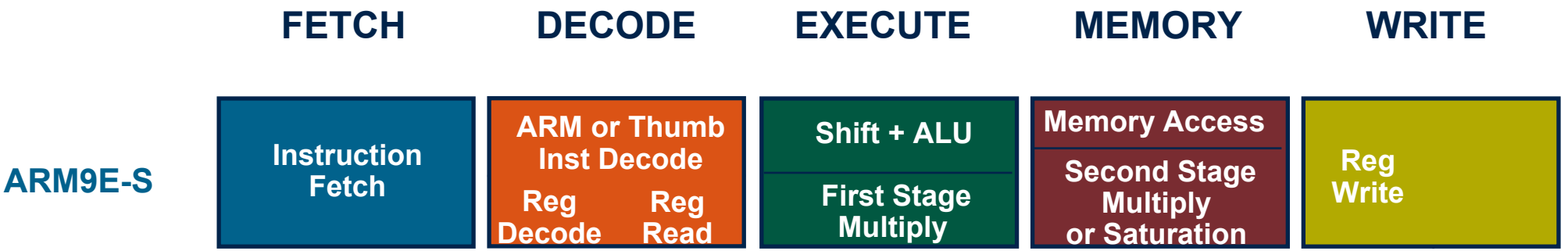
New Instruction In detail

Pipeline & Avoiding Interlocks

Cache and TCM

Ed's General Hints and Tips

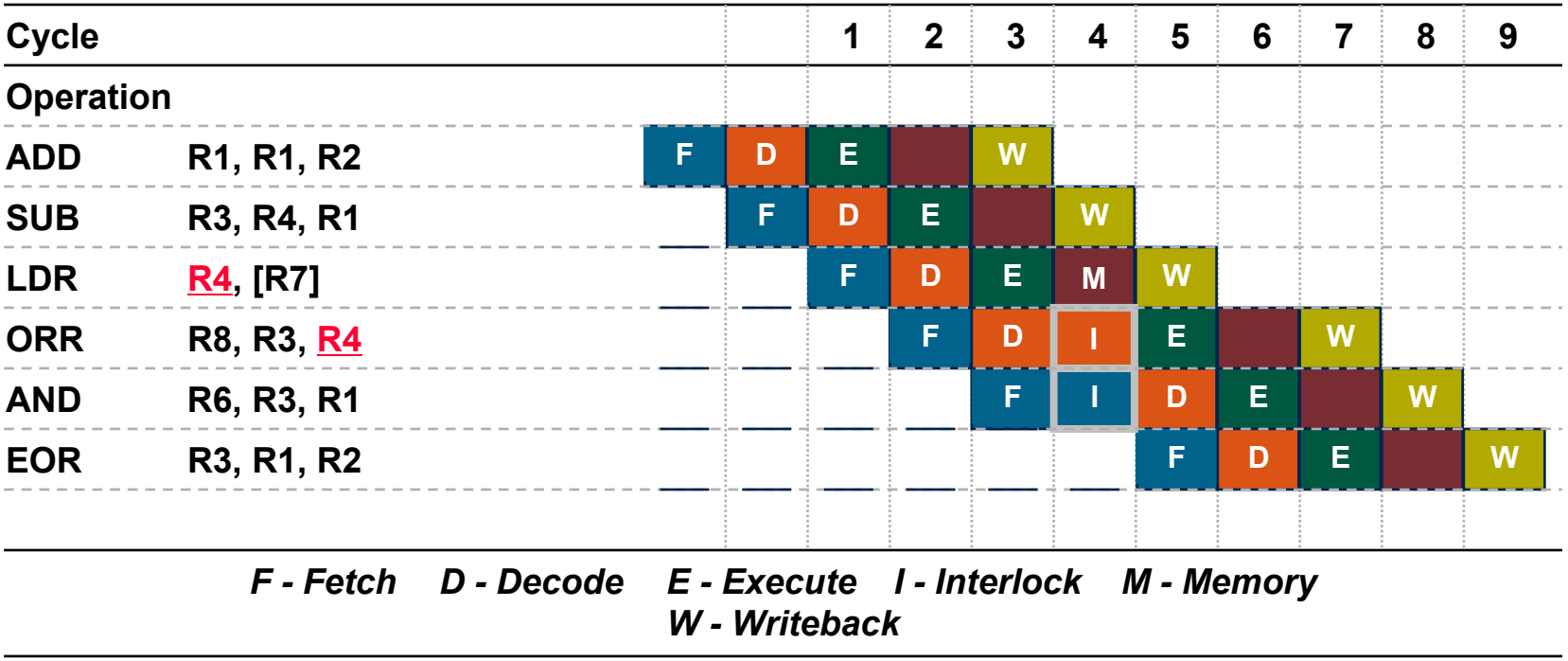
Pipeline changes for ARM9E-S



Pipeline interlocks

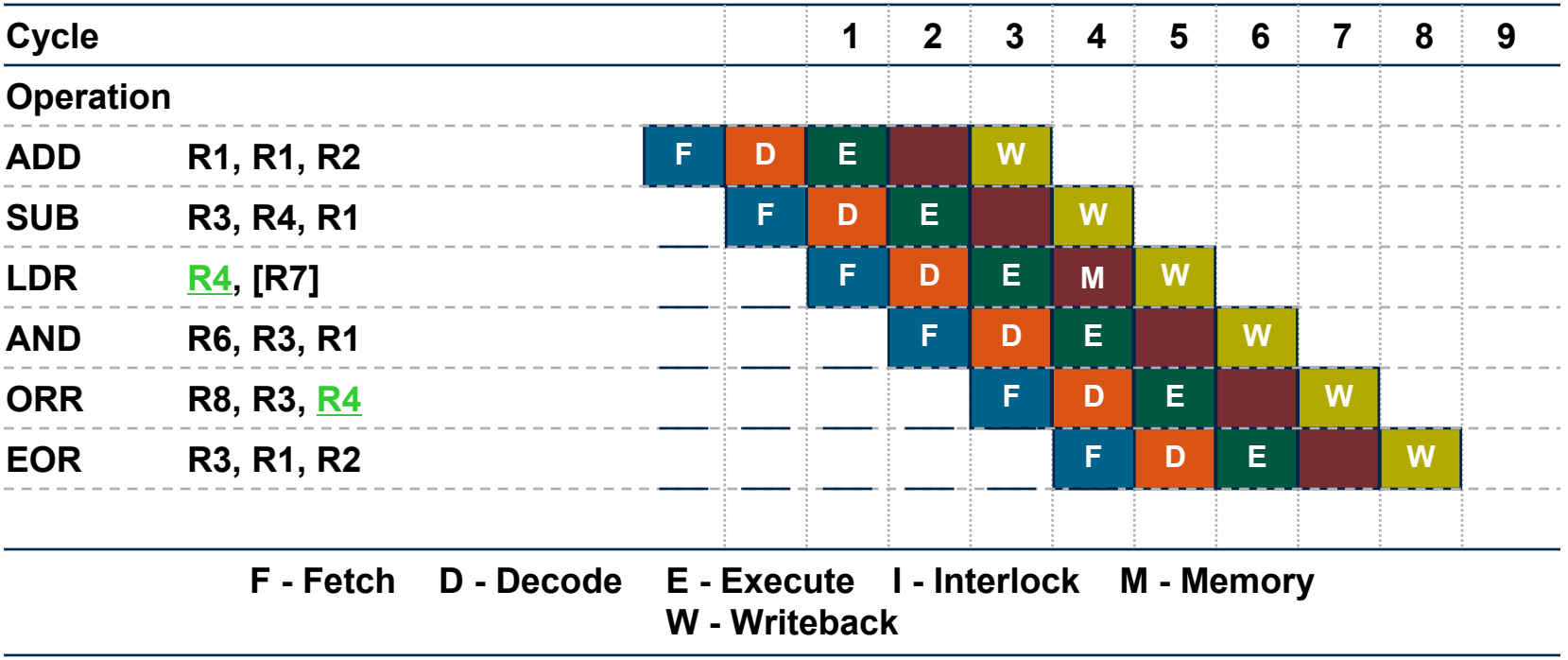
- **Only applicable to ARM9 and later**
  - These cores have longer pipelines which can increase throughput by allowing subsequent instructions to begin execution while earlier instructions are completing
  - For example, following a load instruction, subsequent instructions can execute immediately provided that they do not use the value being loaded
- **Basic types of interlock**
  - LDR
    - Pipeline will interlock for (at least) one cycle if value loaded is used in next instruction
  - LDM
    - Pipeline will interlock for one cycle for every loaded register except the last
    - If the last register loaded is used in the next instruction, there will be one extra cycle of interlock
  - MUL/MLA/QADD etc (9E only)
    - Result is not available until multiply/saturation operation is complete
- **How to avoid interlocks**
  - Avoid using the result immediately if it is delayed

LDR Interlock



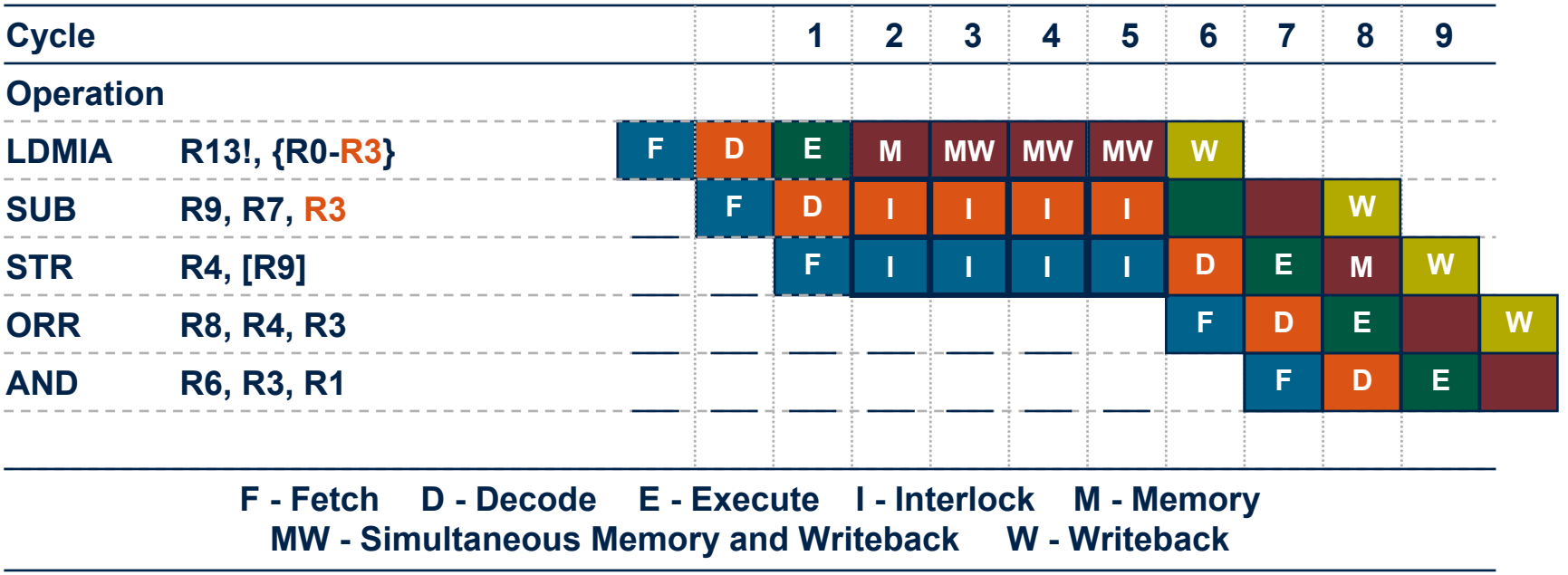
- In this example it takes 7 clock cycles to execute 6 instructions, CPI of 1.2.
- The LDR instruction immediately followed by a data operation using the same register causes an interlock

Optimal Pipelining



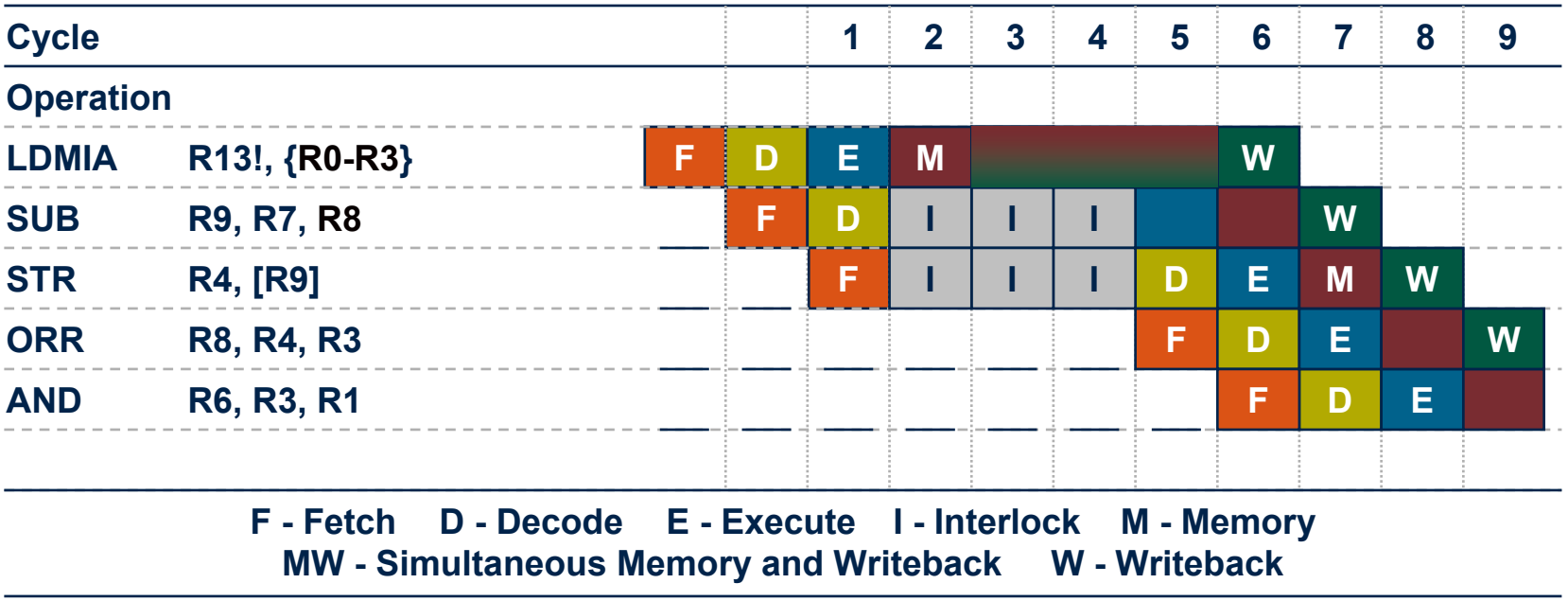
- In this example it takes 6 cycles to execute 6 instructions, CPI of 1.
- The LDR instruction does not cause the pipeline to interlock

LDM Interlock (1)



- In this example it takes 9 clock cycles to execute 5 instructions, CPI of 1.8
- The sub incurs a further cycle of interlock due to it using the highest specified register in the LDM instruction
  - This would occur for any of the LDM variants, e.g. IA, DB, FD, etc.

LDM Interlock (2)



- In this example it takes 8 clock cycles to execute 5 instructions, CPI of 1.6
- During the LDM there are parallel memory and writeback cycles

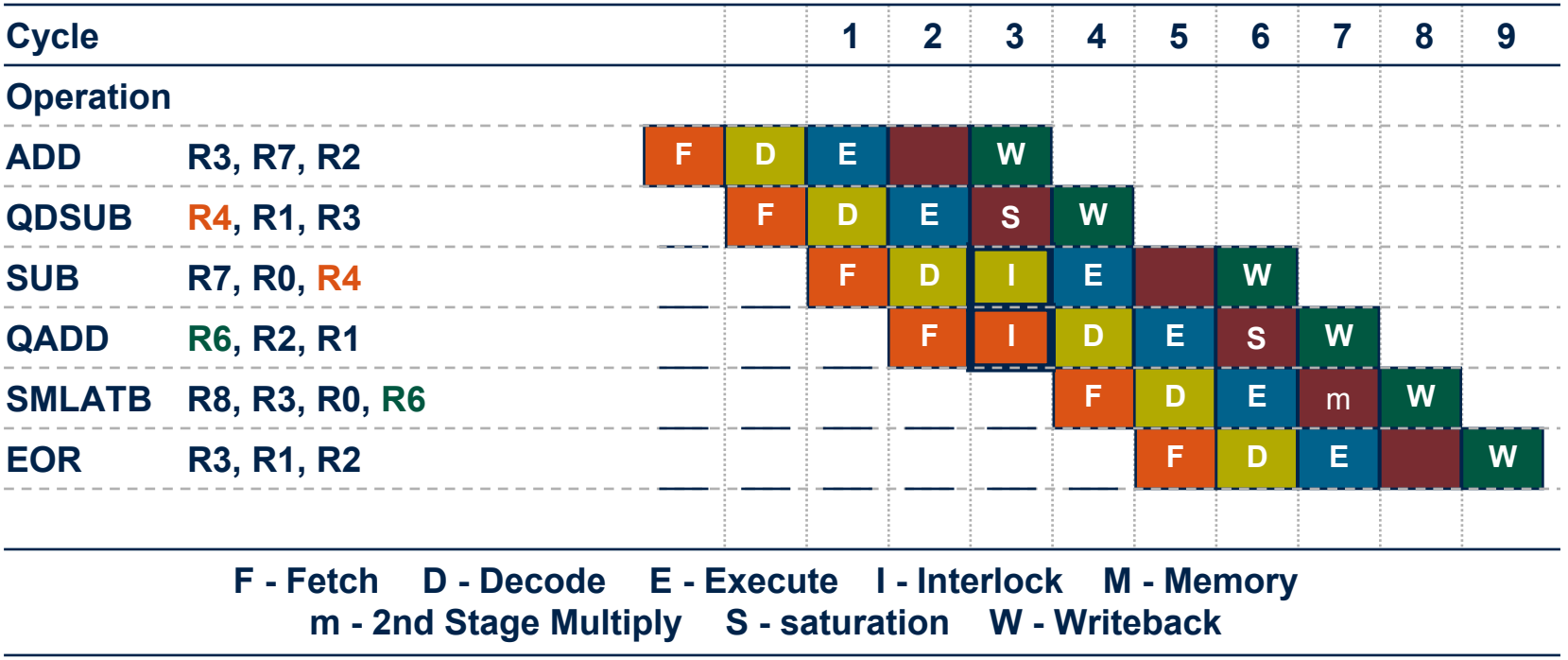
Branching

Cycle			1	2	3	4	5	6	7	8	9
Address	Operation										
0x07328	BL	label (0x03200)	F	D	E	LR		W			
0x0732C	ADD	R0, R2, R8		F	D						
0x07330	MUL	R7, R0, R3			F						
0x03200	STR	R14, [R13, #-4]!				F	D	E	M	W	
0x03204	ORR	R8, R3, R4					F	D	E		W
0x03208	EOR	R3, R1, R2						F	D	E	W

F - Fetch D - Decode E - Execute M - Memory
LR - Link Register Adjust W - Writeback

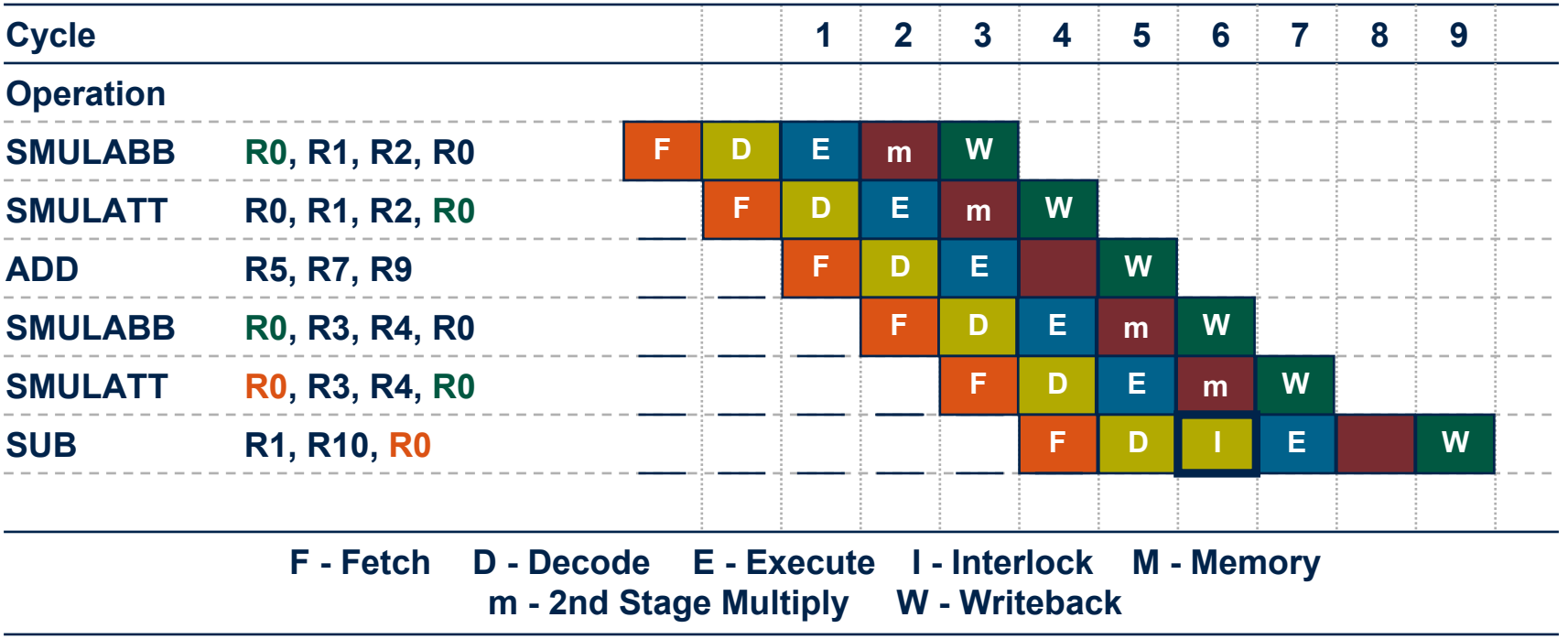
- A branch is executed in 3 cycles
- Only the fetch and decode stages of the pipeline are flushed

Saturated Maths Interlock



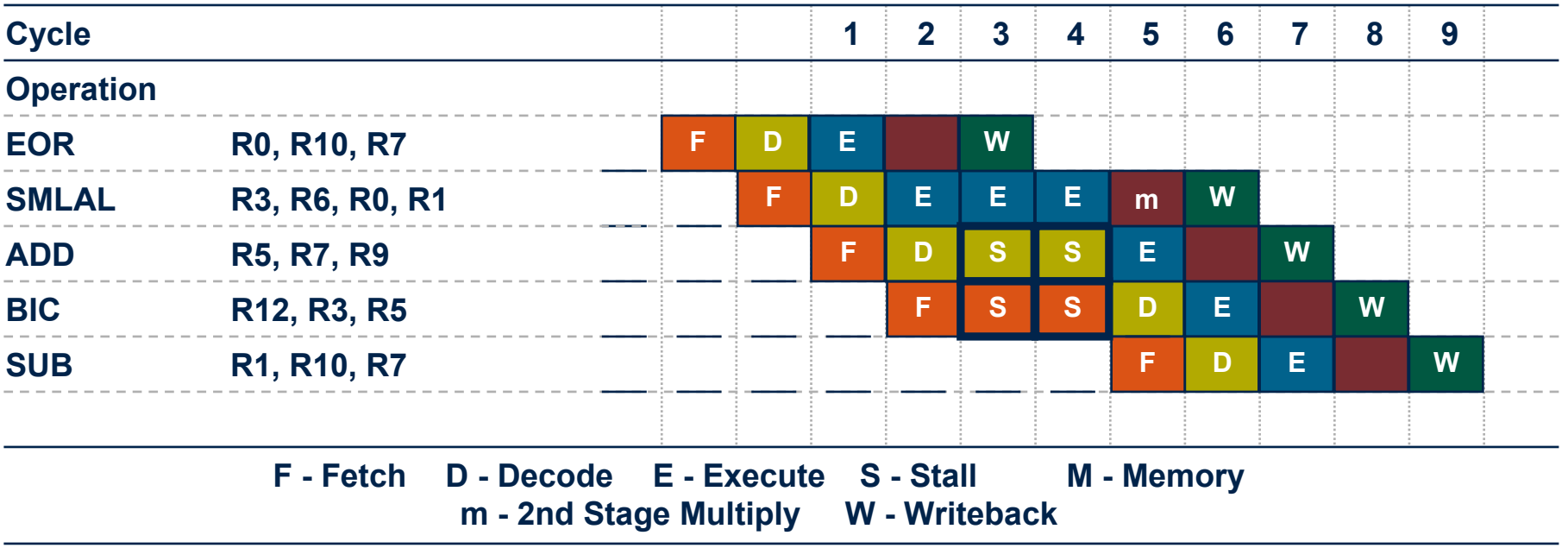
- The SUB is interlocked for one cycle to allow the saturation of R4 from the QDSUB.
- The SMLATB uses R6 for accumulation and therefore doesn't incur an interlock.
- QDSUB R4, R1, R3 ≡ R4 = saturate(R1 - saturate(R3 * 2))
- QADD R6, R2, R1 ≡ R6 = saturate(R2 + R1)

Multiply Interlock



- The SMULATT doesn't incur an interlock as R0 is used for accumulation
- The SUB incurs an interlock due to using R0 as a source operand

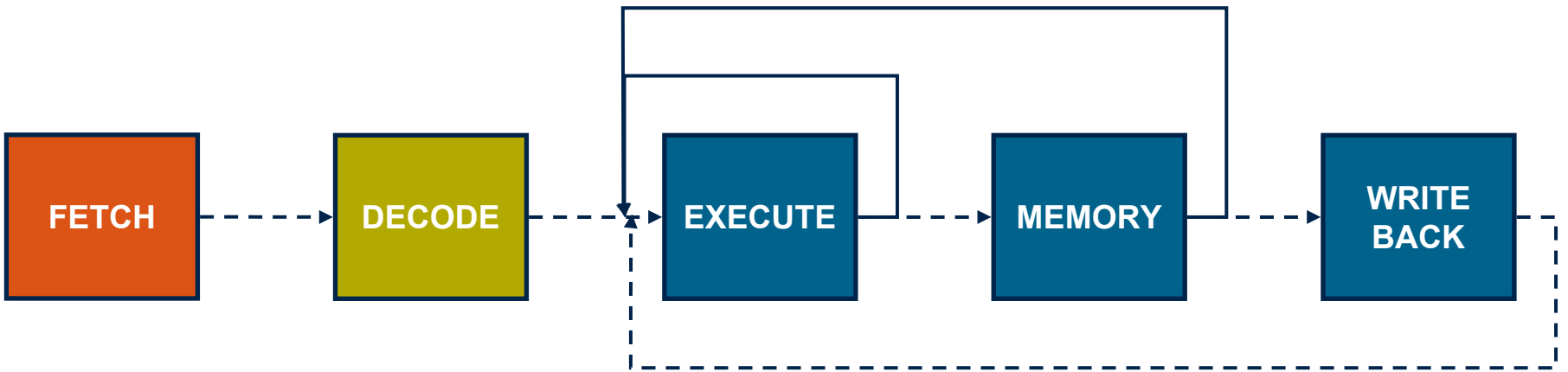
Multiply Stall



- The SMLAL takes 3 cycles to execute
- The ADD instruction is stalled until the multiply completes

So why doesn't ADD interlock?

- **The result of a data processing operation is not written back to the register bank until the end of the Writeback cycle**
  - This is two cycles after the instruction has completed the Execute stage
  - Why is there no interlock?
- **The core implements "data forwarding paths" to allow results to be made available to subsequent instructions before they are written to the register bank**



ARM9E cycle counts (1)

- **ARM9E presents only summary cycle information**
  - Not detailed cycle-by-cycle breakdown
  - Must also present cycle information for **both** memory interfaces
- **Time to execute is not always equal to Result delay**
  - Data memory access instructions “complete” before the loaded value is available to subsequent instructions
- **Below is the cycle information for data processing instructions**

Instruction	Cycles	I bus	D bus	Comment
Data Op	1	1S	1I	Normal case, PC not destination
Data Op	2	1S+1I	2I	Register-controlled shift, PC not destination
Data Op	3	2S+1N	3I	PC destination register
Data Op	4	2S+1N+1I	4I	Register-controlled shift, PC destination

- **These cycle times are constant and determinate**
- **Data dependency issues are not relevant here**

ARM9 cycle counts (2)

- **LDM is similar to the ARM7TDMI case**
  - Delay on next instruction is shorter, though, unless an interlock occurs

Instruction	Cycles	I bus	D bus	Comment
LDM	2	1S+1I	1S+1I	Loading 1 register, not the PC
LDM	n	1S+(n-1)I	1N+(n-1)S	Loading n registers (n>1), not including PC
LDM	n+1	1S+nI	1N+nS+1I	Loading n registers (n>1), not including PC and last value loaded using in next instruction
LDM	n+4	2S+1N+(n+1)I	1N+(n-1)S+4I	Loading n registers, including PC

- **Note that single-register case always incurs a 1-cycle interlock**
- **Normal case takes one cycle per register (two cycles shorter than 7TDMI)**
- **One cycle interlock occurs when last value loaded is used in next instruction**

ARM9E cycle counts (3)

- **Other instructions:**
  - STR - 1 cycle
  - STM - 2 cycles for one register, n cycles for n registers
  - MUL - 3-5 cycles, +1 for long
  - SWP* - 2 cycles
  - SWI - 3 cycles
  - CDP - 1 cycle plus co-processor busy cycles
  - MRC* - 1 cycle if coprocessor not busy
  - MCR - 1 cycle if coprocessor not busy
  - MUL/MLA - 2 cycles (set flags=4)
  - MULL/MLAL - 3 cycles (set flags=5)
  - SMULxy/SMULWx - 1 cycle
  - SMLALxy - 2 cycles
  - QADD* etc - 1 cycle

  - Instructions mark * incur +1 cycle if result is used immediately by next instruction

- **Unexecuted instructions**
  - Instruction which are not executed because the condition code is failed always take 1 cycle.

Agenda

Overview

New Instruction In detail

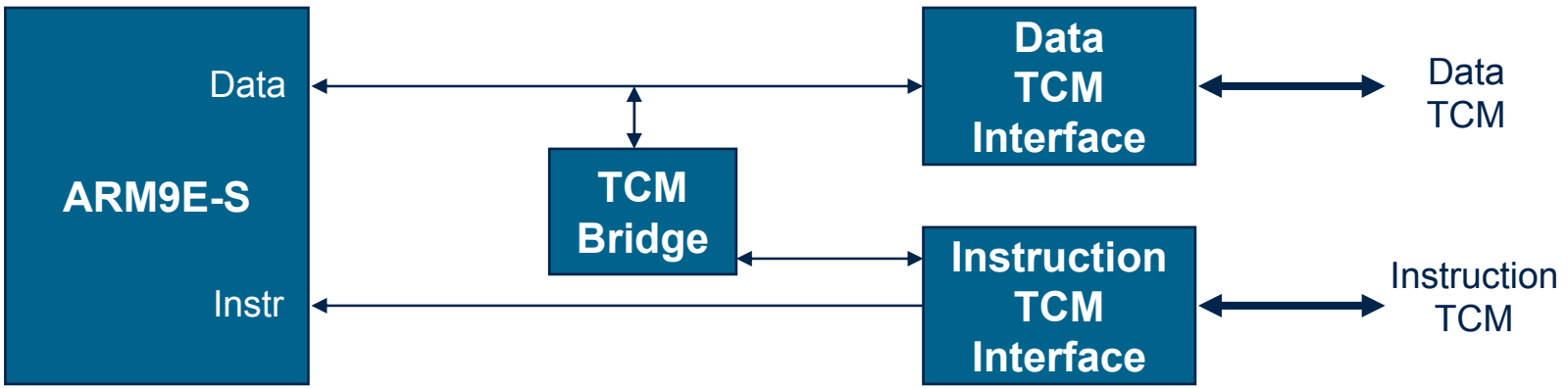
Pipeline & Avoiding Interlocks

Cache and TCM

Ed's General Hints and Tips

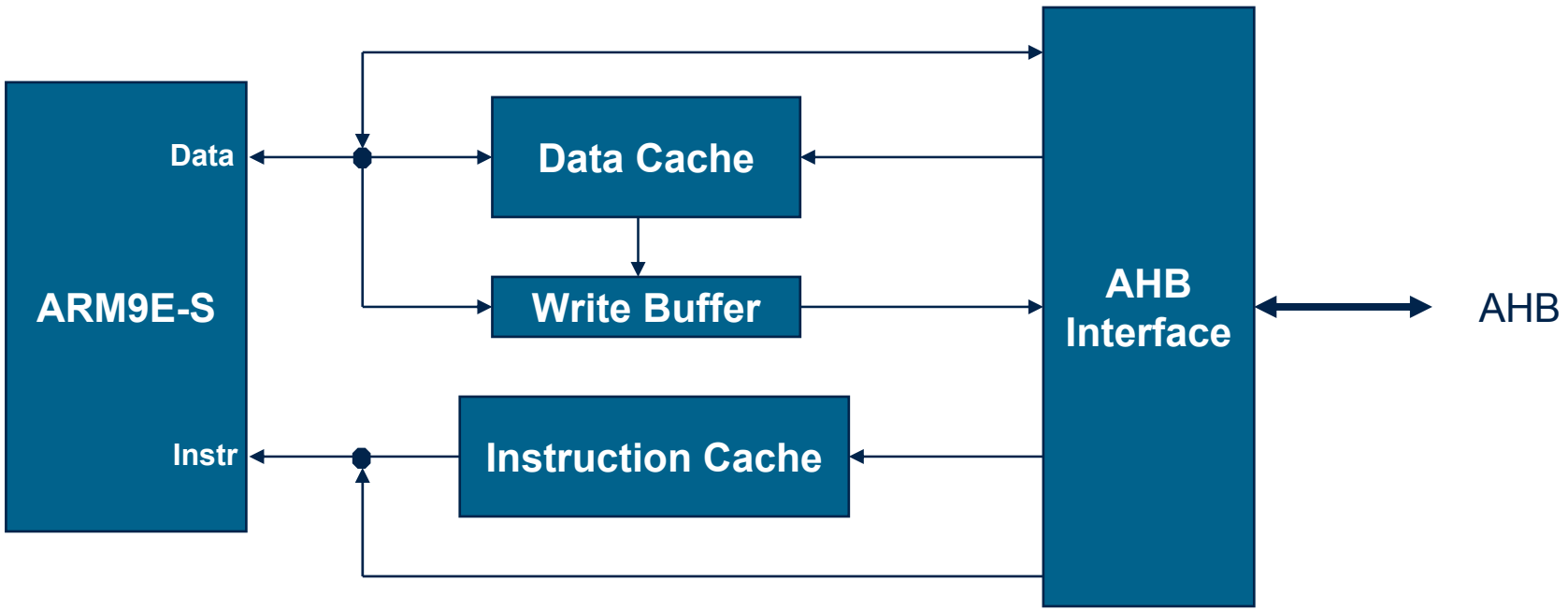
TCM Operations

- **ARM946E-S**
  - I TCM is located at address 0x0
  - D TCM base address is user defined
  - TCM space cannot be cacheable
- **Instruction fetches from D TCM address range access AHB**
  - Data access into I TCM address range allowed



ARM946E-S Cache Operations

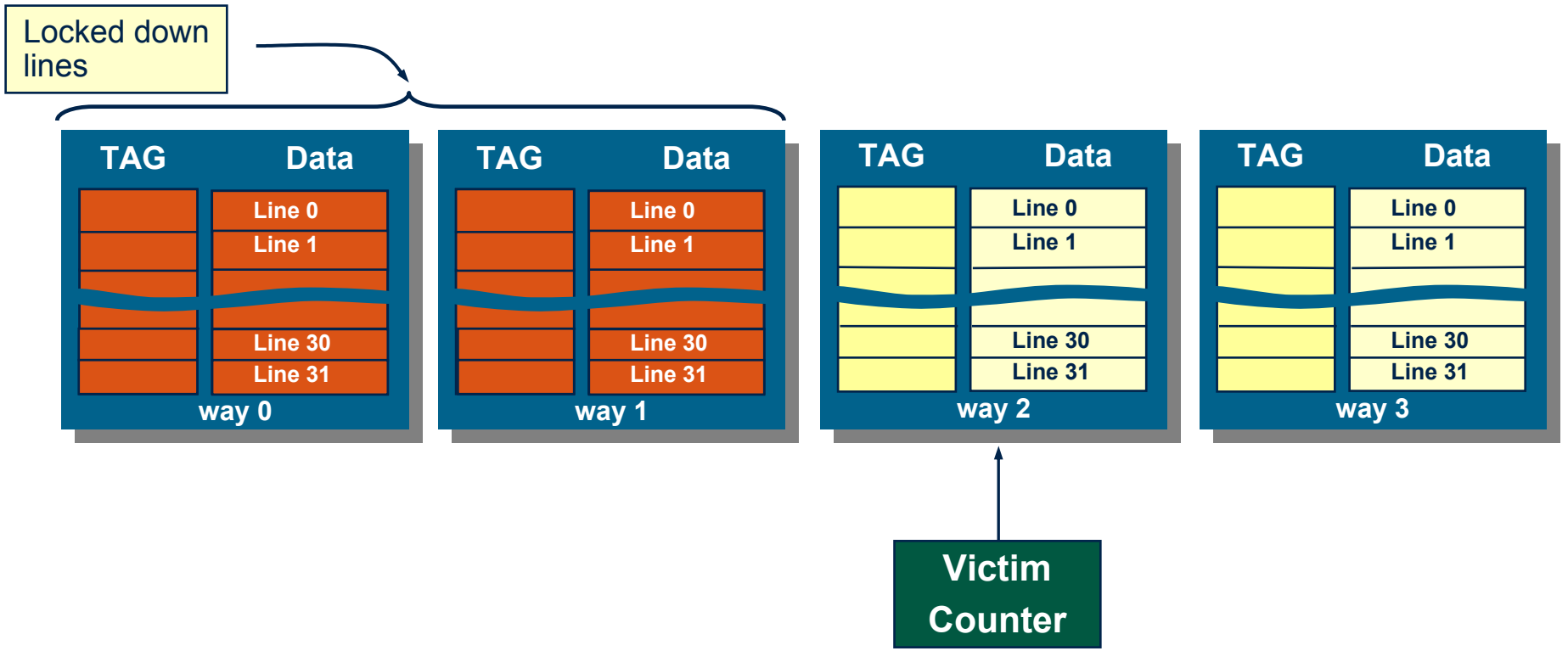
- **Cache supports streaming**
- **Write buffer is drained**
  - before cache line fills
  - when the drain write buffer command is executed
  - Non cacheable, non bufferable accesses bypass write buffer



Cache Lockdown (1)

- **Part of the cache may be locked down to avoid eviction**
  - granularity varies from core to core
- **This might be needed to provide guaranteed real-time performance**
- **Requires short software routine to control linefills**
  - example routines are provided
  - victim counter range is then restricted (cp15 register 9)
- **Locked down lines are immune from replacement**
  - can still be ‘flushed’ - locking mechanism must then be cleared

Cache Lockdown



- This is a 4-way associative cache with 4 ways, each containing 32 lines (sets).
- A single victim counter selects the way in which the replacement takes place.
- Lockdown fixes a base value for victim counter so that ways below this are entirely immune from replacement.
- Lockdown has granularity of a way (1/4 of cache size).

Agenda

Overview

New Instruction In detail

Pipeline & Avoiding Interlocks

Cache and TCM

Ed’s General Hints and Tips

Fixed Point 3D: Geometry

- **Critical Functions are:**
  - Dot Product
  - Cross Product
  - Matrix Vector Product
  - Vector Normalise
  - Reciprocal (1/x)
  - Reciprocal Square Route (1/Sqrt(x))
- **Choose number format carefully for greater efficiency**
  - S15.16 & S1.30 good as general purpose formats
  - Use 16bit or 8bit for object data and rescale
    - Saturated arithmetic and Short (16x16) MUL's. Operations useful
  - Don't be afraid to use long MUL's.
    - SMULL & SMLAL generating intermediate results of 64bits
    - All precision maintain and no fix up required until end of summation.
  - See Dot Product example…

Fixed Point 3D: Dot Product Example

```
;Function: math3dDot
;
;       Purpose : Multiplies two f1616 values as above but hopefully much quicker!
;
;       Parameters :
;         r0                    Pointer to the 1st vector.
;         r1                    Pointer to the 2nd vector.
;
;       Returns :
;         f16.16                The dot product of the two vectors.

math3dDot

        STMFD   sp!, {r4-r7}

        LDMIA   r0, {r2,r3,r4}          ; Get Vector1 (r2=x,r3=y,r4=z)
        LDMIA   r1, {r5,r6,r7}          ; Get Vector2 (r5=x,r6=y,r7=z)

        SMULL   r1, r0, r2, r5          ; (r0,r1) = x1 * x2
        SMLAL   r1, r0, r3, r6          ; (r0,r1) = (r0,r1) + (y1 * y2)
        SMLAL   r1, r0, r4, r7          ; (r0,r1) = (r0,r1) + (z1 * z2)

        MOV     r1, r1, LSR #16         ; Get low half word of result
        ADD     r0, r1, r0, LSL #16     ; Get high halfword

        LDMFD   sp!, {r4-r7}            ; Restore the work registers.
        mov     pc,lr
```

Fixed Point 3D: Geometry

- **Block Float**
  - Dynamic Range without FP overhead
  - Localised 3D data tends toward narrower ranges
    - E.g. Obj and View space data
  - Normalize based on exponent for group of values
    - E.g. object by object basis
    - Gives better dynamic range without overhead of full FP
    - ARM9E has CLZ instruction, very useful here


ARM®
THE ARCHITECTURE
FOR THE DIGITAL WORLD™