ARM10 Thumb® Family

Product Overview

ARM®

ARM DVI 0014A

ARM10 Thumb Family
Product Overview

Proprietary Notice

1 Applications and benefits

300MHz low-power system-on-a-chip processor solutions delivered at a consumer price point.

Applications

- Next-generation hand-held products:
  - Communicators
  - Smartphones
  - Subnotebook computers
- Digital consumer appliances featuring:
  - 3D graphics
  - Web content
  - Voice recognition and synthesis
  - Digital video
  - High-speed connectivity

Benefits

- Multi-sourced high-performance, low-power processor macrocells
- High-performance vector floating point delivers 3D graphics and floating point DSP
- Access to existing ARM architecture, tools, OS, and code-base
- Low system cost via excellent code density
- High performance allows cost saving via migration of hardware features to software implementations
- System-on-a-chip ready allowing rapid integration with short time to market
- Designed to run sophisticated OS such as Linux, EPOC, and WindowsCE

2 The ARM10 Thumb Family

The ARM10 Thumb Family of processors will deliver 400 Dhrystone 2.1 MIPS at 300MHz, and 600 MFLOPS for 3D graphics and floating point DSP. Process portable to high performance 0.25 micron and 0.18 micron CMOS fabrication processes, the ARM10 processor units will be licensed to multiple semiconductor partners, offering OEMs guaranteed continuity of supply. The ARM10 Thumb Family maintains traditional ARM values of low system cost, low power consumption, and use within larger system-on-chip designs. The Thumb 16-bit compressed instruction set gives a reduction in the required memory size and bandwidth, which directly reduces system cost. The ARM10TDMI™ integer unit features the ARM 32-bit RISC instruction set, and Thumb compressed 16-bit instruction set. The ARM10TDMI unit employs parallel instruction execution, branch prediction, and a non-blocking data cache interface to achieve high performance on real applications. The ARM1020T™ cached processor macrocell is built around the ARM10TDMI unit, and also features large on-chip instruction and data caches, an MMU with demand paged virtual memory support, a write buffer, and a new high-bandwidth AMBA™ system-on-a-chip bus interface.

The optional VFP10™ Vector Floating-point coprocessor offers high performance single and double precision IEEE floating-point in a small die size by adopting the RISC approach of implementing simple, common operations in silicon and allowing software to handle rare exceptional cases.

2.1 System-on-a-chip Ready

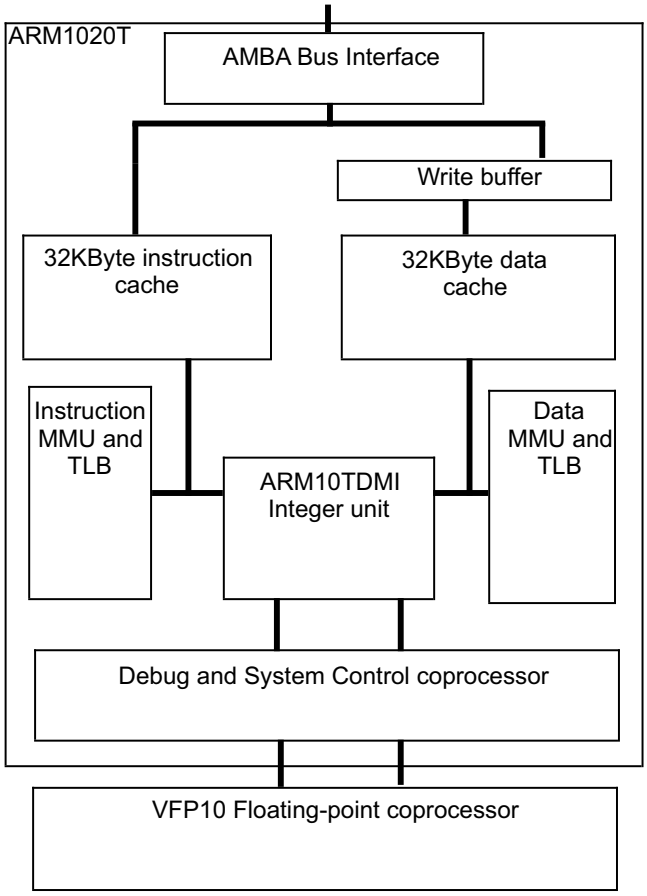
The ARM10 processors feature EmbeddedICE™ JTAG software debug, and the AMBA multimaster on-chip bus architecture that provides for peripheral design reuse and efficient production test. ARM and its partners provide ASIC simulation models, and cosimulation tools to enable the design process.

2.2 Compatible with ARM7™, ARM9™ and StrongARM®

The ARM10 Thumb Processor Family is backwards compatible with the ARM7 Thumb Family, the ARM9 Thumb Family, and StrongARM processor families, giving designers software-compatible processors with a range of price/performance points from 60 MIPS to 400 MIPS. Support for the ARM architecture today includes the WindowsCE, EPOC, JavaOS, and Linux operating systems, more than 25 Real Time Operating Systems, Cosimulation tools from leading EDA vendors, and a variety of software development tools.

3 ARM1020T

The ARM1020T includes cache and memory management functions to support a full demand-paged virtual memory operating system and support for real-time embedded operating systems.



3.1 MMUs

Twin 64-entry *Translation Lookaside Buffers* (TLBs) provide fast access to the most recent address translations. ARM1020T also provides TLB lock-down. This allows critical translations to remain in the TLB to ensure predictable access to real-time code.

3.2 Caches

Two 32KB caches are implemented, one for instructions, the other for data, both with an eight-word line size. These caches connect to the integer unit via 64-bit buses, to allow two instructions to be passed into the instruction prefetch unit every cycle, and to allow load and store multiple instructions to transfer two 32-bit registers every cycle.

3.3 Cache lock-down

Cache lock-down is provided to allow critical code sequences to be locked into the cache to ensure predictability for real-time code. The cache replacement policy can be selected by the operating system as either fully random or round-robin. Both caches are 64-way set-associative.

3.4 Data cache features

The data cache supports nonblocking hit-under-miss operation. Nonblocking operation allows instructions that occur after a data cache miss to continue execution before the data is returned. The hit-under-miss operation allows subsequent load or store instructions after a cache miss to access the data cache. Together these mechanisms can provide significantly higher performance for applications that incur high data cache miss rates.

3.5 Write buffer

ARM1020T also incorporates a double word 8-entry write buffer, to avoid stalling the processor when writes to external memory are performed.

4 The ARMv5T Architecture

4.1 ARM10TDMI integer unit

The ARM10TDMI integer unit is an implementation of the ARM Architecture Version 5T, the latest implementation of the ARM Architecture. ARMv5T is a superset of the ARMv4 ISA implemented by the StrongARM processors and the ARMv4T ISA implemented by the ARM7 Thumb and ARM9 Thumb Family processors.

4.2 Performance and code density

ARM10TDMI executes two instruction sets, the 32-bit ARM instruction set, and the 16-bit Thumb instruction set. The ARM instruction set allows a program to achieve maximum performance with the minimum number of instructions. The simpler Thumb instruction set offers much increased code density for code that does not require maximum performance. Code can switch between the ARM and Thumb instruction sets on any procedure call.

4.3 Registers

The Integer Unit consists of a 32-bit datapath and associated control logic. The datapath contains 31 general-purpose registers, coupled to a full shifter, Arithmetic Logic Unit, and multiplier. At any one time 16 registers are visible to the user. The remainder are synonyms used to speed up exception processing. Register 15 is the *Program Counter* (PC) and can be used in all instructions to reference data relative to the current instruction. R14 holds the return address after a subroutine call. R13 is used (by software convention) as a stack pointer.

Table 1 Modes and Registers

User and System mode	Supervisor mode	Abort mode	Undefined mode	Interrupt mode	Fast Interrupt mode
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12_FIQ
R13	R13_SVC	R13_ABORT	R13_UNDEF	R13_IRQ	R13_FIQ
R14	R14_SVC	R14_ABORT	R14_UNDEF	R14_IRQ	R14_FIQ

Table 1 Modes and Registers (continued)

User and System mode	Supervisor mode	Abort mode	Undefined mode	Interrupt mode	Fast Interrupt mode
PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
-	SPSR_SVC	SPSR_ABORT	SPSR_UNDEF	SPSR_IRQ	SPSR_FIQ

Mode-specific banked registers

4.4 Modes and exception handling

All exceptions have banked registers for R14 and R13. After an exception R14 holds the return address for exception processing. This address is used both to return after the exception is processed and to address the instruction that caused the exception. R13 is banked across exception modes to provide each exception handler with a private stack pointer. The fast interrupt mode also banks registers 8 to 12 so that interrupt processing can begin without the need to save or restore these registers. A seventh processing mode, System mode, does not have any banked registers. It uses the User mode registers. System mode runs tasks that require a privileged processor mode and allows them to invoke all classes of exceptions.

4.5 Status registers

All other processor states are held in status registers. The current operating processor status is in the *Current Program Status Register* (CPSR). The CPSR holds 4 ALU flags (Negative, Zero, Carry and Overflow), two interrupt disable bits (one for each type of interrupt), a bit to indicate ARM or Thumb execution, and 5 bits to encode the current processor mode. All 5 exception modes also have a *Saved Program Status Register* (SPSR) which holds the CPSR of the task immediately before the exception occurred.

4.6 Exception types

ARM10TDMI supports 5 types of exception, and a privileged processing mode for each type. The 5 types of exceptions are:

- fast interrupt (FIQ)
- normal interrupt (IRQ)
- memory aborts (used to implement memory protection or virtual memory)
- attempted execution of an undefined instruction
- software interrupts (SWIs).

4.7 Conditional execution

All ARM instructions (with the exception of `BLX`) are conditionally executed. Instructions optionally update the four condition code flags (Negative, Zero, Carry and Overflow) according to their result. Subsequent instructions are conditionally executed according to the status of flags. Fifteen conditions are implemented.

4.8 4 classes of instructions

The ARM and Thumb instruction sets can be divided into four broad classes of instruction:

- data processing instructions
- load and store instructions
- branch instructions
- coprocessor instructions.

4.9 Data processing

The data processing instructions operate on data held in general purpose registers. Of the two source operands, one is always a register. The other has two basic forms, an immediate value, or a register value optionally shifted. If the operand is a shifted register the shift amount may have an immediate value or the value of another register. Four types of shift can be specified. Most data processing instructions can perform a shift followed by a logical or arithmetic operation. Multiply instructions come in two classes, (normal) 32-bit result and (long) 64-bit result variants. Both types of multiply instruction can optionally perform an accumulate operation.

4.10 Load and store

The second class of instruction is load and store instructions. These instructions come in two main types:

- load or store the value of a single register
- load and store multiple register values.

Load and store single register instructions can transfer a 32-bit word, a 16-bit halfword and an 8-bit byte between memory and a register. Byte and halfword loads can be automatically zero or sign extended as they are loaded. Swap instructions perform an atomic load and store as a synchronization primitive.

4.11 Addressing modes

Load and store instructions have three primary addressing modes:

- offset
- pre-indexed
- post-indexed.

They are formed by adding or subtracting an immediate or register based offset to or from a base register. Register based offsets can also be scaled with shift operations. Pre-indexed and post-indexed addressing modes update the base register with the base plus offset calculation. As the PC is a general purpose register, a 32-bit value can be loaded directly into the PC to perform a jump to any address in the 4Gigabyte memory space.

4.12 Block transfers

Load and store multiple instructions perform a block transfer of any number of the general purpose registers to or from memory. Four addressing modes are provided:

- pre-increment addressing
- post-increment addressing
- pre-decrement addressing
- post-decrement addressing.

The base address is specified by a register value (which may be optionally updated after the transfer). As the subroutine return address and the PC values are in general purpose registers, very efficient subroutine calls can be constructed.

4.13 Branch

The third class of instructions is branch instructions. As well as allowing any data processing or load instruction to change control flow (by writing the Program Counter) a standard branch instruction is provided with 24-bit signed offset, allowing forward and backward branches of up to 32Megabytes.

4.14 Branch with Link

There is a Branch with Link (`BL`) which allows efficient subroutine calls. `BL` preserves the address of the instruction after the branch in R14 (the Link Register or LR). This allows a move instruction to put the LR in to the PC and return to the instruction after the branch.

The third type of branch (`BX` and `BLX`) is used to switch between ARM and Thumb instruction sets optionally with the return address preserving link option.

4.15 Coprocessor

The fourth class of instruction is coprocessor instructions. There are three types of coprocessor instructions:

- coprocessor data processing instructions

  These are used to invoke a coprocessor specific internal operation.

- coprocessor register transfer instructions

  These allow a coprocessor value to be transferred to or from an ARM register.

- coprocessor data transfer instructions.

  These transfer coprocessor data to or from memory, where the ARM calculates the address of the transfer.

4.16 ARM instruction set

Table 2 The ARM Instruction Set

Mnemonic	Operation	Mnemonic	Operation
MOV	Move	MVN	Move Not
ADD	Add	ADC	Add with Carry
SUB	Subtract	SBC	Subtract with Carry
RSB	Reverse Subtract	RSC	Reverse Subtract with Carry
CMP	Compare	CMN	Compare Negated
TST	Test	TEQ	Test Equivalence
AND	Logical AND	BIC	Bit Clear
EOR	Logical Exclusive OR	ORR	Logical (inclusive) OR
MUL	Multiply	MLA	Multiply Accumulate
SMULL	Sign Long Multiply	SMLAL	Signed Long Multiply Accumulate
UMULL	Unsigned Long Multiply	UMLAL	Unsigned Long Multiply Accumulate
CLZ	Count Leading Zeroes	BKPT	Breakpoint
MRS	Move From Status Register	MSR	Move to Status Register
B	Branch		
BL	Branch and Link	BLX	Branch and Link and Exchange
BX	Branch and Exchange	SWI	Software Interrupt
LDR	Load Word	STR	Store Word
LDRH	Load Halfword	STRH	Store Halfword
LDRB	Load Byte	STRB	Store Byte
LDRSH	Load Signed Halfword	LDRSB	Load Signed Byte
LDMIA	Load Multiple	STMIA	Store Multiple
SWP	Swap Word	SWPB	Swap Byte
CDP	Coprocessor Data Processing		
MRC	Move From Coprocessor	MCR	Move to Coprocessor
LDC	Load To Coprocessor	STC	Store From Coprocessor

4.17 Thumb instruction set

Table 3 The Thumb Instruction Set

Mnemonic	Operation	Mnemonic	Operation
MOV	Move	MVN	Move Not
ADD	Add	ADC	Add with Carry
SUB	Subtract	SBC	Subtract with Carry
RSB	Reverse Subtract	RSC	Reverse Subtract with Carry
CMP	Compare	CMN	Compare Negated
TST	Test	NEG	Negate
AND	Logical AND	BIC	Bit Clear
EOR	Logical Exclusive OR	ORR	Logical (inclusive) OR
LSL	Logical Shift Left	LSR	Logical Shift Right
ASR	Arithmetic Shift Right	ROR	Rotate Right
MUL	Multiply	BKPT	Breakpoint
B	Unconditional Branch	Bcc	Conditional Branch
BL	Branch and Link	BLX	Branch and Link and Exchange
BX	Branch and Exchange	SWI	Software Interrupt
LDR	Load Word	STR	Store Word
LDRH	Load Halfword	STRH	Store Halfword
LDRB	Load Byte	STRB	Store Byte
LDRSH	Load Signed Halfword	LDRSB	Load Signed Byte
LDMIA	Load Multiple	STMIA	Store Multiple
PUSH	Push Registers to stack	POP	Pop Registers from stack

4.18 ARM instruction set opcode map

	31 30 29 28	27 26 25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5	4	3 2 1 0
Data processing immediate shift	cond	0 0 0	opcode				S	Rn	Rd	shift amount	shift	0	Rm
Move status register to register	cond	0 0 0 1 0	R	0	0		0	SBO	Rd	SBZ	0 0 0 0		SBZ
Move register to status register	cond	0 0 0 1 0	R	1	0		0	mask	SBO	SBZ	0 0 0 0		Rm
Data processing register shift	cond	0 0 0	opcode				S	Rn	Rd	Rs	0	shift	1	Rm
Branch/exchange instruction set	cond	0 0 0 1 0	0	1	0		0	SBO	SBO	SBO	0 0	L	1	Rm
Software breakpoint	cond	0 0 0 1 0	0	1	0		0	immed			0 1 1 1			immed
Count leading zeros	cond	0 0 0 1 0	1	1	0		0	SBO	Rd	SBO	0 0 0 1			Rm
Multiply (-accumulate)	cond	0 0 0 0	0	0	A		S	Rd	Rn	Rs	1 0 0 1			Rm
Multiply (-accumulate) long	cond	0 0 0 0	1	Un	A		S	RdHi	RdLo	Rs	1 0 0 1			Rm
Swap/swap byte	cond	0 0 0 1	0	B	0	0		Rn	Rd	SBZ	1 0 0 1			Rm
Load/store halfword register offset	cond	0 0 0	P	U	0	W	L	Rn	Rd	SBZ	1 0 1 1			Rm
Load/store halfword immediate offset	cond	0 0 0	P	U	1	W	L	Rn	Rd	Hi Offset	1 0 1 1			Lo Offset
Load signed halfword/byte register offset	cond	0 0 0	P	U	0	W	1	Rn	Rd	SBZ	1 1	H	1	Rm
Load signed halfword/byte immediate offset	cond	0 0 0	P	U	1	W	1	Rn	Rd	Hi Offset	1 1	H	1	Lo Offset
Data processing immediate	cond	0 0 1	opcode				S	Rn	Rd	rotate	immediate			
Move immediate to status register	cond	0 0 1 1 0	R	1	0			Mask	SBO	rotate	immediate			
Load/store immediate offset	cond	0 1 0	P	U	B	W	L	Rn	Rd	immediate				
Load/store register offset	cond	0 1 1	P	U	B	W	L	Rn	Rd	shift amount	shift	0	Rm	
Load/store multiple	cond	1 0 0	P	U	S	W	L	Rn	register list					
Branch and branch with link	cond	1 0 1	L	24-bit offset										
Branch with link/change to Thumb	1 1 1 1	1 0 1	H	24-bit offset										
Coprocessor load and store	cond	1 1 0	P	U	N	W	L	Rn	CRd	cp_num	8-bit offset			
Coprocessor data processing	cond	1 1 1 0	opcode1					CRn	CRd	cp_num	opcode2	0	CRm	
Coprocessor register transfers	cond	1 1 1 0	opcode1			L		CRn	Rd	cp_num	opcode2	1	CRm	
Software interrupt	cond	1 1 1 1	swi number											

4.19 Thumb instruction set opcode map

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Shift by immediate	0	0	0	opcode		immediate					Rm			Rd		
Add/subtract register	0	0	0	1	1	0	opc	Rm			Rn			Rd		
Add/subtract immediate	0	0	0	1	1	1	opc	immediate			Rn			Rd		
Add/subtract/move/compare immediate	0	0	1	opcode		Rd / Rn		immediate								
Data-processing register	0	1	0	0	0	0	opcode				Rm / Rs			Rd / Rn		
Special data processing	0	1	0	0	0	1	opcode		H1	H2	Rm			Rd / Rn		
Branch/exchange instruction set	0	1	0	0	0	1	1	1	L	H2	Rm			SBZ		
Load from literal pool	0	1	0	0	1	Rd			PC-relative offset							
Load/store register offset	0	1	0	1	opcode		Rm				Rn			Rd		
Load/store word/byte immediate offset	0	1	1	B	L	offset					Rn			Rd		
Load/store halfword immediate offset	1	0	0	0	L	offset					Rn			Rd		
Load/store from/to stack	1	0	0	1	L	Rd			SP-relative offset							
Add to SP or PC	1	0	1	0	SP	Rd			immediate							
Adjust stack pointer	1	0	1	1	0	0	0	0	opc	immediate						
Push/pop register list	1	0	1	1	L	1	0	R	register list							
Software breakpoint	1	0	1	1	1	1	1	0	immediate							
Load/store Multiple	1	1	0	0	L	Rn			register list							
Conditional branch	1	1	0	1	cond				offset							
Software interrupt	1	1	0	1	1	1	1	1	immediate							
Unconditional branch	1	1	1	0	0	offset										
BLX suffix	1	1	1	0	1	offset										0
BL/BLX prefix	1	1	1	1	0	offset										
BL suffix	1	1	1	1	1	offset										

4.20 ARM10TDMI integer pipeline stages

The integer pipeline consists of 6 stages to maximize instruction throughput on ARM10TDMI:

F Instruction Fetch and Branch Prediction

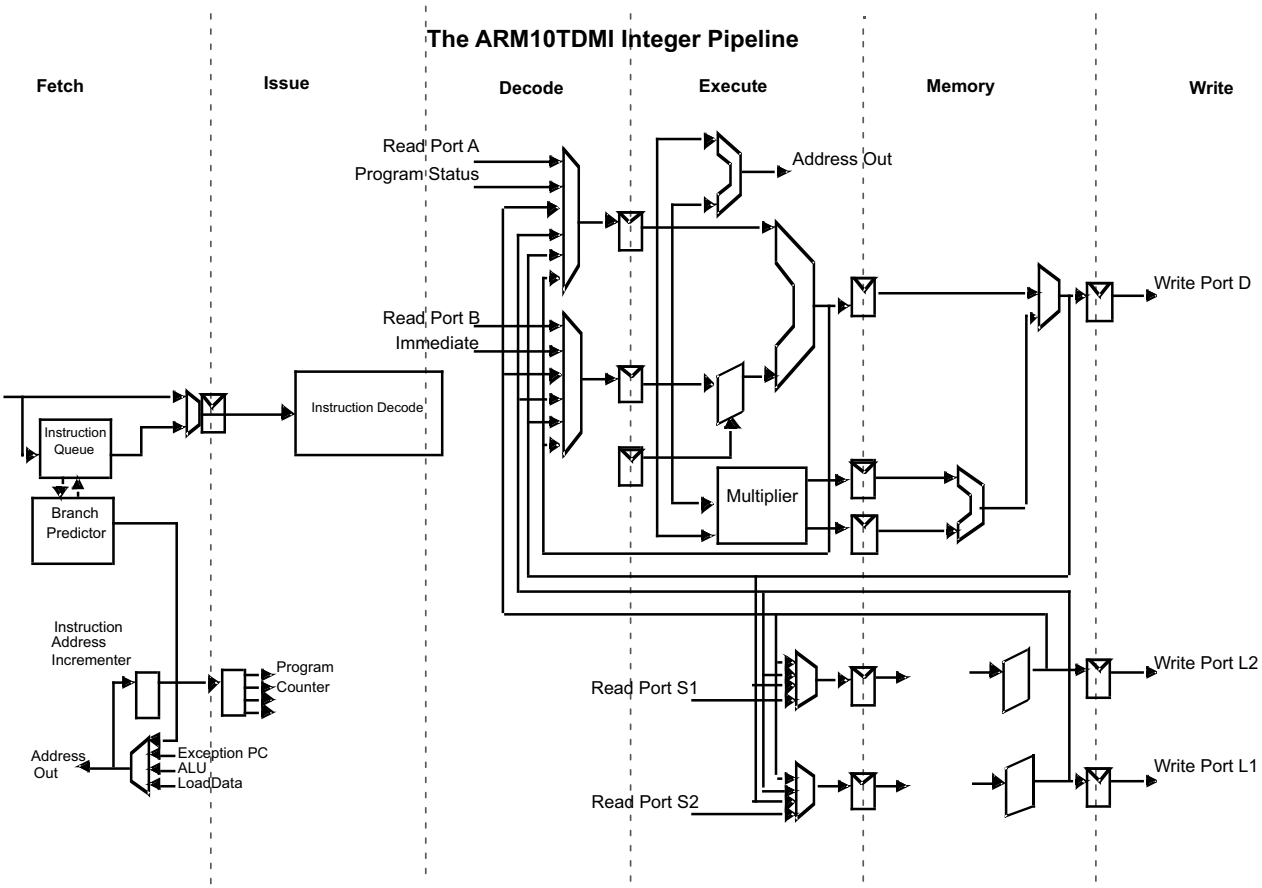
I Instruction issue

D Instruction Decode and Register Read

E
- Execute
- Shift and ALU, or
- Address Calculate, or
- Multiply

M Memory Access and Multiply

W Register Write



4.21 Pipelining

By overlapping the various stages of execution, ARM10TDMI maximizes the clock rate achievable to execute each instruction. It delivers a throughput approaching one instruction per cycle. Furthermore, due to multiple execution units ARM10TDMI allows multiple instructions to exist in the same pipeline stage, allowing simultaneous execution of some instructions. The Fetch stage can hold up to three instructions, where branch prediction is performed on instructions ahead of execution of earlier instructions. The Issue and Decode stage can contain any instruction in parallel with a predicted branch. The Execute, Memory and Write stages can contain a predicted branch, an ALU or Multiply instruction, a load or store multiple instruction and a coprocessor instruction in parallel execution.

4.22 64-bit data buses

ARM10TDMI provides 64-bit data buses between the processor integer unit and the instruction and data caches, and between coprocessors and the integer unit. These 64-bit paths allow two instructions to be loaded into the branch prediction unit, so that branches are predicted before they are executed, and so that load and store multiple instructions can transfer 64 bits (two ARM registers) every cycle. This allows ARM10TDMI to achieve very high performance on many code sequences, especially those that require data movement in parallel with data processing.

4.23 Coprocessors and pipelines

The ARM10TDMI coprocessor interface allows full independent processing in both the ARM execution pipeline and the pipelines of up to 4 independent coprocessors.

4.24 Branch prediction

The branch prediction unit can often completely resolve branches, effectively removing them from the instruction stream. The Load-Store unit can sustain load and store multiple transfers in parallel with data processing instructions. The Branch Prediction Unit works by prefetching instructions beyond the fetch stage, decoding branch instructions, calculating branch target addresses, and fetching the target instruction.

4.25 Benefits of speculative branch prediction

Under normal operation, branch prediction can be completed before the fetch stage requests the branch instruction, so that the instruction at the target of the branch can be speculatively given to the integer unit instead of the branch. This reduces the execution time impact of the branch to zero. The branch prediction scheme is static. Backward branches are assumed taken as these are usually loops. Forward branches are assumed untaken. If the prediction is wrong (a branch mis-predict), the integer unit takes three cycles to resume execution. On average this scheme correctly predicts 80% of branch destinations.

4.26 Debug features

The integer unit also incorporates a sophisticated debug unit to allow both software tasks or external debug hardware to perform hardware and software breakpoint, single stepping, register and memory access. This functionality is made available to software as a coprocessor and is accessible from hardware via the JTAG port. Full speed, real time execution of the processor is maintained until a breakpoint is hit, at which point control is either passed to a software handler, or to JTAG control.

4.27 ARM10TDMI instruction execution timing

Table 4 ARM10TDMI instruction execution timing

Instruction class	Issue Cycles	Result delay
Condition failed	1	NA
Branch Predict	0,1	NA
Branch Mispredict	3	NA
`ALU` instruction	1	0
`ALU` instruction with register shift	2	0
`MOV PC, Rx`	3	NA
`ALU` instruction dest = PC	4	0
`MUL`	1..3	1..3
`MSR` (flags only)	1	0
`MSR` (mode change)	3	NA
`MRS`	1	0
`LDR` (base register value)	1	0
`LDR` (loaded value)	1	1
`LDR` with shifted offset	+1	0
`STR`	1	NA
`STR` with shifted offset	2	NA
`LDM`	1	Position in list / 2 + 1
`STM`	1	NA
`SWP`	2	1
`CDP`	1	NA
`MRC`	1	1
`MCR`	1	NA
`LDC`	1	Number of words / 2
`STC`	1	NA

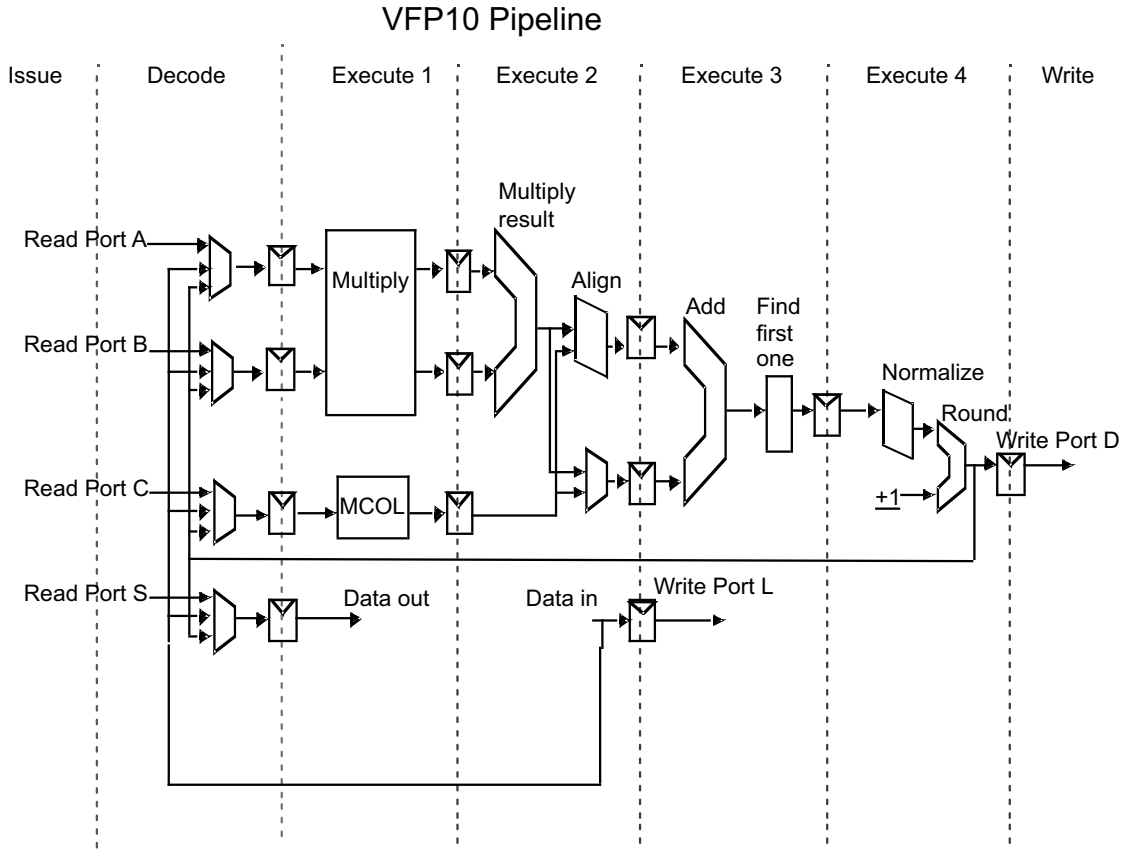
5 VFP10 - Vector Floating-point Unit

5.1 VFP10 floating point unit

The VFP10 Floating-point Unit is the first implementation of the Vector Floating-point architecture (VFP). VFP is designed to provide high-performance, low-cost *floating-point* (FP) computation for a wide spectrum of applications. VFP uses a register bank consisting of 32 single precision values or 16 double precision values. The individual elements of the register bank can be used as a vector of data, allowing a single instruction to operate on multiple data values. In vector mode, the 32 single precision registers are used to provide 8 scalar values and (most commonly) either 6 vectors each containing 4 elements, or 3 vectors each containing 8 elements.

5.2 The VFP10 pipeline

VFP data processing instructions use a multiple-add pipeline. Fundamental operations include multiply-add, negated multiply-add, multiply-subtract, multiply, add, subtract, and compare. Divide, remainder and square root are implemented as iterative processes that use the multiply-accumulate pipeline. Instructions are also provided for data movement of integer values between the VFP register and ARM registers, and conversion between integer values and floating-point values.



5.3 Single instructions and multiple data

The vector nature of the VFP architecture allows a single instruction to specify an operation on multiple data items. This allows multiple instructions to be in execution at once, greatly increasing the performance of FP-intensive applications.

5.4 IEEE 754 compatibility

VFP is fully IEEE 754 compatible. To allow faster execution for some algorithms VFP can optionally avoid the overhead necessary to perform IEEE gradual underflow, instead rounding to zero when the floating point exponent underflows such that the mantissa can no longer remain normalized. This option can be enabled from software via a control register configuration bit.

5.5 Load and store instructions

VFP provides both single and vector load and store instructions, which perform a transfer between memory and the VFP registers. Both single and double precision values are supported. The transfer address is specified in an ARM register. If a multiple transfer is performed the instruction specifies both the first register to transfer, and the number of registers to transfer. After the transfer the base register may be updated for auto indexing for array stack access.

5.6 Branch instructions

VFP does not provide branch instructions. Instead the result of an FP compare instruction can be stored in the ARM condition code flags. This allows the ARM branch instruction to be used for executing conditional FP code.

5.7 VFP10 floating-point pipeline

VFP10 uses two pipelines, a five stage pipe for load and store instructions, and a 7-stage pipe for arithmetic instructions. These two pipes share the first two stages, and can issue one instruction per cycle. The vector nature of the VFP architecture allows a vector arithmetic instruction to execute in parallel with a vector load or store instruction or an integer instruction.

5.8 The 5-stage pipeline

The five stage Load and Store tracks the final 5 stages of the ARM pipeline. If a multiple transfer is being performed, the memory stage of the pipeline is repeatedly used for each data item. Two single precision values or one double precision value can be transferred every cycle. Load and store instructions, and data transfers, stay in lock step with the integer unit to transfer data when the VFP instruction owns the memory stage.

5.9 The 7-stage arithmetic pipeline

VFP10 data processing instructions use a 7-stage pipeline. The first two stages match the issue and decode stages of the integer unit, followed by four stages that perform the actual floating point arithmetic, and the seventh and final stage is for register write. The four arithmetic stages are broken into two parts, multiply and round, and add and round, each part taking two cycles.

5.10 Ensuring IEEE 754 accuracy

To ensure full IEEE 754 accuracy, the pipeline forms a complete result between the multiply and accumulate portions of a multiply-accumulate instruction. All add, subtract and compare operations align the smaller value to the larger value to maintain maximum precision.

5.11 Performance

The VFP design uses a deep pipeline to achieve a high clock frequency. A single precision multiply-add can be issued every cycle, with a result delay of three cycles. For many common algorithms the result delay has little impact on achieved performance as several operations can be started before the first result is required. FIR filters and array multiplies are examples. Coding these algorithms using vector instructions boosts performance further by allowing parallel execution with load or store or integer instructions.

5.12 VFP10 instruction execution timing

Table 5 VFP10 instruction execution timing

Instruction class	Issue Cycles	Result delay
Multiply (Single Precision)	1	3
Multiply (Double Precision)	2	3
Add/ Subtract/Compare/Move	1	3
Multiply Accumulate (Single Precision)	1	3
Multiply Accumulate (Double Precision)	2	3
Divide/Square Root (Single Precision)	19	N/A
Divide/Square Root (Double Precision)	33	N/A
Load	1	1
Store	1	NA
Convert	1	3

5.13 VFP instruction set

Table 6 VFP Instruction Set

Mnemonic	Operation	Mnemonic	Operation
FADD	Add	FCPY	Copy (Move)
FSUB	Subtract		
FMUL	Multiply	FNMUL	Negated multiply
FMAC	Multiply-Accumulate	FNMAC	Negated Multiply-Accumulate
FMSC	Multiply-Subtract	FNMSC	Negated Multiply-Subtract
FDIV	Divide	FSQRT	Square Root
FABS	Absolute Value	FNEG	Negate
FCMP	Compare Register with Register	FCMPZ	Compare with Zero
FCVTDS	Convert Double to Single	FCVTSD	Convert Single to Double
FITOF	Convert Integer to Float	FFTOI	Convert Float to Integer
FLDR	Load Single Value	FSTR	Store Single Value
FLDM	Load Multiple Values (Vector)	FSTM	Store Multiple Values (Vector)

5.14 VFP instruction set opcode map

	31 30 29 28	27	26	25	24	23	22	21	20	19 18 17 16	15 14 13 12	11	10	9	8	7	6	5	4	3 2 1 0
Data processing immediate	cond	1	1	1	0	D	E	F	G	Fn	Fd	1	0	1	S	N	H	M	0	Fm
Move to FP register	cond	1	1	1	0	opcode			0	Fn	Rd	1	0	1	S	N	R	R	1	Reserved
Move to ARM register	cond	1	1	1	0	opcode			1	Fn	Rd	1	0	1	S	N	R	R	1	Reserved
Load	cond	1	1	0	P	U	D	W	1	Rd	Rn	1	0	1	S	Offset or Transfer Length				
Store	cond	1	1	0	P	U	D	W	0	Rn	Fd	1	0	1	S	Offset or Transfer Length				

6 System issues and Third Party Support

6.1 AMBA Bus Architecture

The ARM10 Thumb Family processors are designed for use with the AMBA multi-master on-chip bus architecture. AMBA includes an advanced high performance bus (AHB) connecting processors and high-bandwidth peripherals and memory interfaces, and a low-power peripheral bus allowing a large number of low-bandwidth peripherals. The AHB bus is re-used to allow efficient production test of the ARM1020T processor macrocell and VFP10 coprocessor.

The ARM1020T AHB implementation provides a 32-bit address bus and a 64-bit data bus for high-bandwidth data transfers made possible by on-chip memory and modern SDRAM and RAMBUS memories.

6.2 Everything you need

ARM provides a wide range of products and services to support its processor families, including software development tools, development boards, models, applications software, training, and consulting services.

The ARM Architecture today enjoys broad 3rd party support. The ARM10 Thumb Family processors' strong software compatibility with existing ARM families will ensure that its users benefit immediately from this existing support. ARM is working with its software, EDA, and semiconductor partners to extend this support to use new ARM10 Family features.

6.3 Current support

Support for the ARM Architecture today includes:

- ARM SDT Software Development Toolkit
  - — Integrated development environment
  - — C, C++, assembler, simulators and windowing source level debugger
  - — Available on Windows95, WindowsNT, and Unix
- ARM Multi-ICE™ JTAG interface
  - — allows debug of ARM processor systems through JTAG interface
  - — integrates with the ARM SDT
- ARMulator instruction accurate software simulator
- Development boards
- Design Simulation Models provide signoff quality ASIC simulation
- Software toolkits available from ARM, Cygnus/GNU, Greenhills, JavaSoft, MetaWare, Microtec, and Windriver allowing software development in C, C++, Java, FORTRAN, Pascal, Ada, and assembler.
- 20+ Real Time Operating Systems including Windriver VxWorks, Sun Microsystems Chorus and JavaOS, Microtec VRTX, JMI, Embedded SystemProducts RTXC, and Integrated Systems pSOS.
- Major OS including Microsoft WindowsCE, PSION EPOC, NetBSD and Linux UNIX, Geoworks

- Application software components: DSP, Speech and image compression, software modems, Chinese character input, network protocols, and so on.
- Hardware/Software Cosimulation tools from leading EDA Vendors.

For more information, see www.arm.com

7 Contacting ARM

Addresses

America

ARM INC.
750 University Avenue
Suite 150
Los Gatos
California 95032
USA

Tel: +1-408-579-2200
Fax:+1-408-579-1205
Email: info@arm.com

Austin Design Center
ARM
1250 Capital of Texas Highway
Building 3, Suite 560
Austin
Texas 78746
USA

Tel: +1-512-327-9249
Fax:+1-512-314-1078
Email: info@arm.com

Seattle
ARM
10900 N.E. 8th Street
Suite 920
Bellevue
Washington 98004
USA

Tel: +1-425-688-3061
Fax:+1-425-454-4383
Email: info@arm.com

Boston
ARM
300 West Main St
Suite 215
Northborough
MA 01532
USA

Tel: +1-508-351-1670
Fax:+1-508-351- 1668
Email: info@arm.com

England

ARM Ltd
48-49 Bateman Street
Cambridge
Cambridgeshire
CB2 1LR
England

Tel: +44 1223 400500
Fax:+44 1223 400408
Email: info@arm.com

France

ARM France
12, Avenue des Prés
BL 204 Montigny le Bretonneux
78059 Saint Quentin en Yvelines
Cedex
Paris
France

Tel: +33 1 30 79 05 10
Fax: +33 1 30 79 05 11
Email: info@arm.com

Germany
ARM
Otto Hahn Str. 13B
85521 Ottobrunn-Riemerling
Munich 85521
Germany

Tel: +49 89 608 75545
Fax:+49 89 608 75599
Email: info@arm.com

Japan
ARM K.K.
Plustaria Building 4F
3-1-4 Shin-Yokohama
Kohoku-ku,
Yokohama-shi
Kanagawa 222-0033

Tel: +81 45 477 5260
Fax: +81 45 477 5261
Email: info-armkk@arm.com

Korea
ARM
Room #1115
Hyundai Building
9-4 Soonae-Dong
Boondang-Ku, Sungam
Kyunggi-Do 463-020
Korea

Tel: +82-342-712-8234
Fax: +82-342-713-8225
Email:info@arm.com