
ARM® Processor Programmer Model *aka* Instruction Set Architecture

**An assembly language programmer's
view of the processor hardware**

1

Instruction Set Architecture

- Programmer's do care about:
 - What happens when a “branch” is executed?
 - What happens when a subroutine is called?
 - What happens when an interrupt occurs?
- Programmers Don't Care so much about:
 - How is the processor control unit structured?
 - What is architecture of the cache memory?
 - What is the memory decoding architecture?
 - What is the I/O bus protocol?
- We care about BOTH since this
“Microcontroller Architecture and Interfacing”²

ISA Versions and Part Numbers

- Confusing since Acorn produced ARM1 and ARM2 then Advanced Risc Machines started with ARM6 Part numbers
- ISA Versions More or Less in Sequential order
- Be careful when referring to the “version” of ARM, make it clear if you mean part numbers or architectures

3

ISA Versions and Part Numbers

ARM Family	ARM Architecture	ARM Core
ARM1	ARMv1	ARM1
ARM2	ARMv2	ARM2
	ARMv2a	ARM250
ARM3	ARMv2a	ARM3
ARM6	ARMv3	ARM60
		ARM600
		ARM610
ARM7	ARMv3	ARM700
		ARM710
		ARM710a

4

ISA Versions and Part Numbers (cont)

ARM Family	ARM Architecture	ARM Core
ARM7TDMI	ARMv4T	ARM7TDMI(-S)
		ARM710T
		ARM720T
		ARM740T
ARM7EJ	ARMv5TEJ	ARM7EJ-S
ARM8	ARMv4	ARM810
StrongARM	ARMv4	SA-1
ARM9TDMI	ARMv4T	ARM9TDMI
		ARM920T
		ARM922T
		ARM940T

5

ISA Versions and Part Numbers (cont)

ARM Family	ARM Architecture	ARM Core
ARM9E	ARMv5TE	ARM946E-S
		ARM966E-S
		ARM968E-S
	ARMv5TEJ	ARM926EJ-S
	ARMv5TE	ARM996HS
ARM10E	ARMv5TE	ARM1020E
		ARM1022E
	ARMv5TEJ	ARM1026EJ-S
XScale	ARMv5TE	XScale
		Bulverde
		Monahans

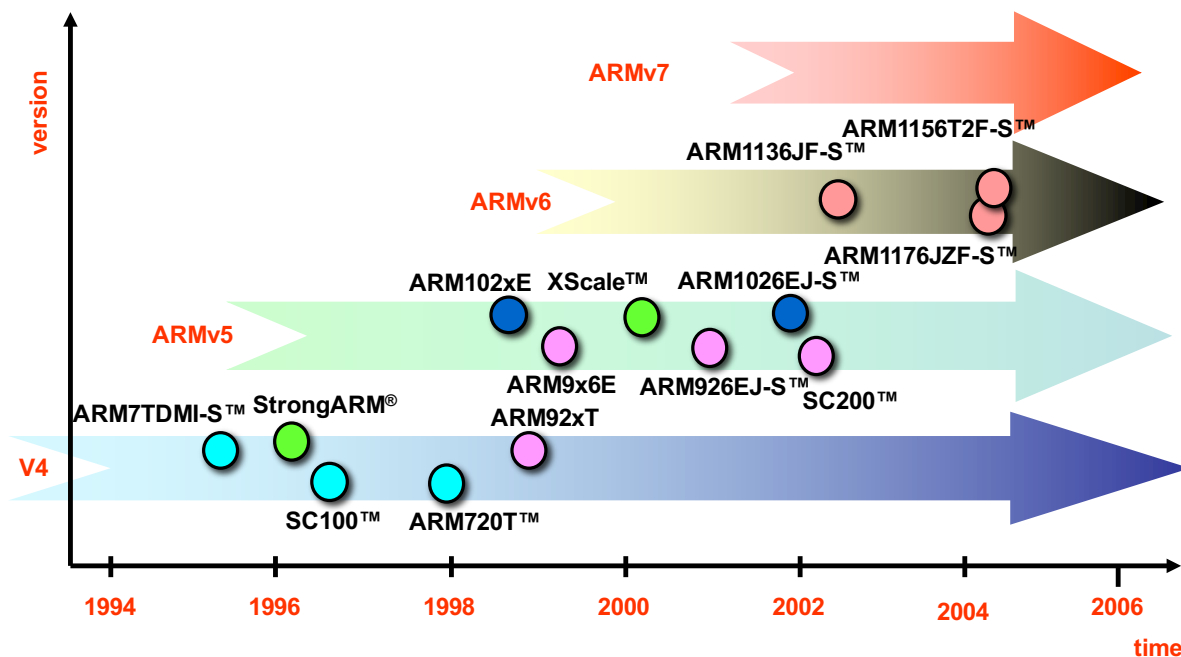
6

ISA Versions and Part Numbers (cont)

ARM Family	ARM Architecture	ARM Core
ARM11	ARMv6	ARM1136J(F)-S
	ARMv6T2	ARM1156T2(F)-S
	ARMv6ZK	ARM1176JZ(F)-S
	ARMv6K	ARM11 MPCore
Cortex-A	ARMv7-A	Cortex-A5
		Cortex-A8
		Cortex-A9 MPCore
		Cortex-A15 MPCore
Cortex-R	ARMv7-R	Cortex-R4(F)
Cortex-M	ARMv6-M	Cortex-M0
		Cortex-M1
	ARMv7-M	Cortex-M3
	ARMv7-ME	Cortex-M4

7

Version Timeline



8

Information on Features/Applications

http://en.wikipedia.org/wiki/ARM_architecture

ARM® Processor Programmer Model

ARM9 Processor

The ARM9™ processor family is built around the ARM9TDMI processor core, which implements the v4T architecture with a 5-stage pipeline, and supports the 16-bit Thumb® instruction set.

ARM920T-based ARM9 processor:

- ARM9TDMI core + Dual Caches (32K) + MMU
- Targeted for OS-based embedded applications

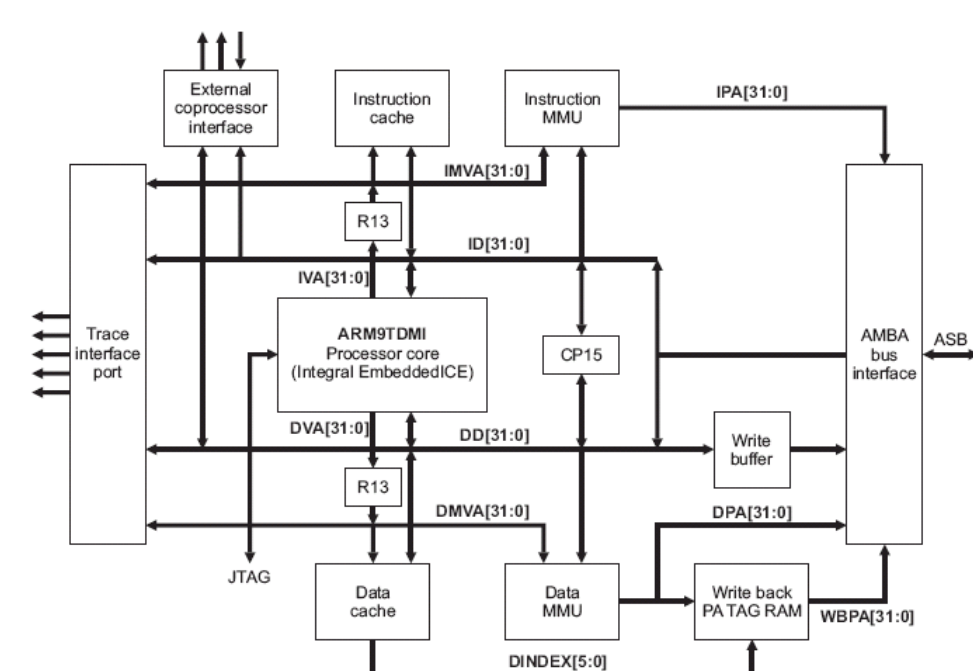
Separate instruction cache and data cache enabled.

Harvard cache operation:

- The AMBA bus interfaces to the rest of the system using a unified address and data buses.

11

ARM920T Block Diagram



12

ARM9TDMI and ARM7TDMI

The ARM9TDMI processor core implements a v4T ARM architecture like the ARM7TDMI

- Executes the ARM 32-bit instruction set and the compressed Thumb 16-bit instruction set

Fully code compatible to the ARM7TDMI, with two differences:

- 1.Better handling of the Data Abort exceptions that occurs during a memory access — Base register is restored to the original value before the exception occurs.
- 2.Fully implements the instruction set extension space — provides the flexibility to emulate additional instruction sets triggered through the Undefined exception.

13

Programmer' s Model

The Programmer' s Model describes the features of the processor available to the programmer:

- What are the registers available
- How codes are stored in the memory
- How different data types are handled
- What instructions are available to manipulate the processing of data

(Usually best understood through a processor instruction set presented at the lowest level, i.e., assembly language)

14

Memory Format

- The ARM is a 32-bit architecture.
- When used in relation to the ARM:
 - **Nybble** means 4 bits (Halfbyte-one Hex digit)
 - **Byte** means 8 bits
 - **Halfword** means 16 bits
 - **Word** means 32 bits (four bytes, Word definition varies for diff. processors, eg. Intel 8088 is 8, 8086 is 16, 80486 is 32, Coldfire is 16)
- Most ARM's implement two instruction sets
 - 32-bit ARM Instruction Set
 - 16-bit Thumb Instruction Set
- Jazelle cores can also execute Java bytecode

15

Memory Format

The 32-bit v4T ARM processors access memory in word aligned format

- Stored in groups of four bytes in an ascending memory order
 - The first data will correspond from byte 0 to byte 3 in the memory
 - The second data will be stored from byte 4 to byte 7

Data can be in the endian format (i.e., bi-endian)

31	24	23	16	15	8	7	0
Word at address A							
Halfword at address A+2				Halfword at address A			
Byte at address A+3		Byte at address A+2		Byte at address A+1		Byte at address A	

Addresses used for a little-endian word

16

Data Types

The 32-bit v4T ARM processors support data types of the following sizes with the proper boundary alignment in the memory:

- 32-bit words
 - Aligned to four byte boundaries, with the lowest two address bits equal to zero (xxxxxx00b)
- 16-bit half-words
 - Aligned to two byte boundaries, with the lowest address bit equal to zero (xxxxxxx0b)
- 8-bit bytes
 - Can be placed on any byte boundary

17

Processor Operating States

The v4T based ARM processor has two operating states (**don't confuse states with modes**):

1.ARM state

- Executes 32-bit word aligned ARM instructions

2.Thumb state

- Executes 16-bit half-word aligned Thumb instructions

The two states can be switched through the BX (Branch and Exchange) instruction.

As all exception handlings are performed in the ARM state, processors in the Thumb state will change to the ARM state when any exception occurs and revert back to the Thumb state on return.

18

Processor Operating Modes

The v4T-based ARM processor has seven operating modes that are further classified into Non-Privileged (1) mode and Privileged mode (6)

1.Non-Privileged mode

–User

2.Privileged modes

–System

–Supervisor

–FIQ

–IRQ

–Abort

–Undefined

19

Processor Operating Modes Explained

- The ARM has seven basic operating modes:
- **User** : unprivileged mode under which most tasks run
- **FIQ** : entered when a high priority (fast) interrupt is raised
(eg. fast response-battery drained-save register content to memory)
- **IRQ** : entered when a low priority (normal) interrupt is raised
(eg. slower response-user has pressed a button)
- **Supervisor** : entered on reset and when a Software Interrupt instruction is executed
- **Abort** : used to handle memory access violations
(violations cause a type of interrupt sometimes called a “trap” or “exception”)
- **Undef** : used to handle undefined instructions
- **System** : privileged mode using the same registers as user mode

20

Processor Operating Modes (cont' d)

1. User mode

- The usual processor mode used when executing a program
- Switch to the one of the other modes when exception occurs. (For example, when an IRQ interrupt occurs)

2. System mode

- A privileged user mode; share the same register set as those of the User mode. But allows, for example, enabling and disabling of FIQ and IRQ interrupt
- Useful for implementing nested priority interrupt system, and executing with an operating system

21

Processor Operating Modes (cont' d)

3. Supervisor (SVC) mode

- The default mode entered on reset and SWI exception
- No restriction to the access of hardware, which is needed during boot up in order to initialize the processor and system
- Normally, this is the mode the processor is in when an operating system is operating in its Protected mode

4. Fast Interrupt (FIQ) mode

- Entered through the nFIQ exception
- Use for fast interrupt response. For example, to support data transfer or channel process

22

Processor Operating Modes (cont' d)

5. Interrupt (IRQ) mode

- Entered through the lower priority interrupt
- Use for general-purpose interrupt handling from peripheral and external signal sources

6. Abort mode

- Entered through either Data Abort or Prefetch Abort exception
- Triggered by invalid memory access violation (data or instruction prefetch access)

7. Undefined mode

- Entered when an undefined instruction is encountered
- Can be useful for the software emulation of hardware

23

Processor Registers

The v4T based processor has a total of 37 registers that are selectively made available depending on the operating modes and states of the processor. These include:

- 30 general-purpose registers
- Six status registers
- One program counter

The registers are arranged in an overlapped bank

- some registers are shared among different operating modes

24

Processor Registers

- General-purpose Registers
 - 32 bits wide (one word)
 - arranged in partially overlapping banks
 - This means, at an instant in time, a programmer sees 15 (r0 through r14), 1 or 2 status registers, and the PC (or r15)
 - Always have the same names (r0-r14, etc) but Physically different depending on the MODE
 - r3 may contain something DIFFERENT after a mode change!!!!
- Why “Swap” or “Bank Out” Registers?

25

ARM Registers by State

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0 1	r0	r0	r0	r0	r0
r1 2	r1	r1	r1	r1	r1
r2 3	r2	r2	r2	r2	r2
r3 4	r3	r3	r3	r3	r3
r4 5	r4	r4	r4	r4	r4
r5 6	r5	r5	r5	r5	r5
r6 7	r6	r6	r6	r6	r6
r7 8	r7	r7	r7	r7	r7
r8 9	r8_fiq 16	r8	r8	r8	r8
r9 10	r9_fiq 17	r9	r9	r9	r9
r10 11	r10_fiq 18	r10	r10	r10	r10
r11 12	r11_fiq 19	r11	r11	r11	r11
r12 13	r12_fiq 20	r12	r12	r12	r12
r13 14	r13_fiq 21	r13_svc 23	r13_abt 25	r13_irq 27	r13_und 29
r14 15	r14_fiq 22	r14_svc 24	r14_abt 26	r14_irq 28	r14_und 30
r15 (PC) 1	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

ARM-state program status registers

CPSR 1	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq 2	SPSR_svc 3	SPSR_abt 4	SPSR_irq 5	SPSR_und 6

 = banked register

26

ARM Register File

Current Visible Registers

Abort Mode

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
spsr

Banked out Registers

User

FIQ

IRQ

SVC

Undef

	r8			
	r9			
	r10			
	r11			
	r12			
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
	spsr	spsr	spsr	spsr

27

Register r0 to r15 and Special Uses

In the ARM state, 16 registers, plus one or two Status Registers, are visible in any of the operating modes.

Registers r0 to r15 are directly accessible in all the modes:

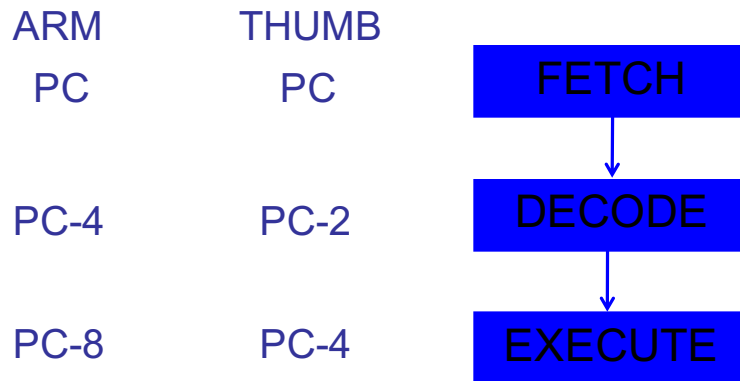
- r0 to r12: used for general purposes, holding either data or addresses
- r13: used as the Stack Pointer (SP)
- r14: typically used as the return address Link Register (LR) in subroutine and branch link operations
- r15: always used as the Program Counter (PC)

NOT FULLY ADHERING TO PURE RISC PHILOSOPHY

28

Program Counter (PC or r15) Register

- Keeps track of current instruction (IP in x86 architecture)
- Can be implemented differently in various architectures
- Illustrated with Fetch/Decode/Execute Pipeline:



29

Program Counter (PC or r15) Register

- **When the processor is executing in ARM state:**
 - All instructions are 32 bits wide
 - All instructions must be word aligned
 - Therefore the **pc** value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be halfword or byte aligned)
- **When the processor is executing in Thumb state:**
 - All instructions are 16 bits wide
 - All instructions must be halfword aligned
 - Therefore the **pc** value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte aligned)
- **When the processor is executing in Jazelle state:**
 - All instructions are 8 bits wide
 - Processor performs a word access to read 4 instructions at once

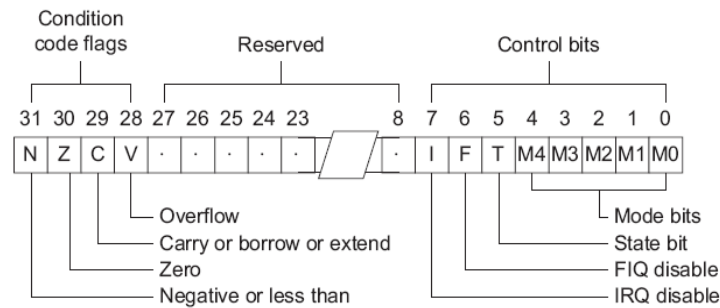
30

Current Program Status Register

Current Program Status Register (CPSR):

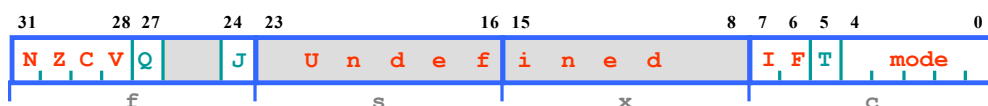
- Contain current state information of processor
- Contains the condition code flags of the most recently performed ALU operation
- Controls (enable/disable) the interrupts
- Controls (set) the processor mode

Shared among all the operating modes



31

Program Status Registers



- **Condition code flags**
 - N = Negative result from ALU
 - Z = Zero result from ALU
 - C = ALU operation Carried out
 - V = ALU operation oVerflowed
- **Sticky Overflow flag - Q flag**
 - Architecture 5TE/J only
 - Indicates if saturation has occurred
- **J bit**
 - Architecture 5TEJ only
 - J = 1: Processor in Jazelle state (processor executes Java Bytecode)
- **Interrupt Disable bits.**
 - I = 1: Disables the IRQ.
 - F = 1: Disables the FIQ.
- **T Bit**
 - Architecture xT only
 - T = 0: Processor in ARM state
 - T = 1: Processor in Thumb state
- **Mode bits**
 - Specify the processor mode

32

CPSR Bit Settings

Mode bits (M4–M0) determines the processor operating modes

- changes automatically when entering the exception privileged modes of different processors
- can be changed through software instruction when in any of the privileged mode (e.g., for nested interrupt support)

Mode Bit settings (M4–M0)

USER mode:	10000b	SYS mode:	11111b
FIQ mode :	10001b	IRQ mode:	10010b
SVC mode:	10011b	ABORT mode:	10111b
UNDEFINE mode:	1100b		

33

CPSR Bit Settings (cont' d)

Conditional code flags (N, Z, C, V)

- usually affected during the arithmetic and logical operations
- can also be changed by MSR and LDM instructions

Control bits F and I enable/disable the FIQ and IRQ

- setting the I bit disables the IRQ interrupt
- setting the F bit disables the FIQ interrupt

State bit T reflects the processor operating state

- when set, it indicates the processor is in the Thumb state
- when clear, it indicates the processor is in the ARM state

34

Vector Table

EXCEPTION VECTORS		
EXCEPTION TYPE	MODE	VECTOR ADDRESS
Reset	SVC	0x00000000
Undefined Instructions	UNDEF	0x00000004
Software Interrupt (SWI)	SVC	0x00000008
Prefetch Abort (instruction fetch memory abort)	ABORT	0x0000000C
Data Abort (data access memory abort)	ABORT	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

35

Saved Program Status Registers

Saved Program Status Register (SPSR):

- Only available when operating in the privileged mode (i.e., except user mode)
- Contains the condition code flags and mode bits that allow entry to the privileged mode (i.e., during exception handling)

There are five SPSRs: one for each of the five privileged operating modes (except the system mode)

- Used to preserve the value of the CPSR when switching modes
- Since User and System mode are not entered during an exception, they have no SPSR
- Attempting to read SPSR while in User/System mode, unpredictable result occurs, writes are ignored

36

What happens during an Exception?

- When an exception occurs, the ARM:

- Copies CPSR into SPSR_<mode>
- Sets appropriate CPSR bits
 - Change to ARM state
 - Change to exception mode
 - Disable interrupts (if appropriate)
- Stores the return address in LR_<mode>
- Sets PC to vector address

- To return, exception handler needs to:

- Restore CPSR from SPSR_<mode>
- Restore PC from LR_<mode>

This can only be done in ARM state.

0x1C

0x18

0x14

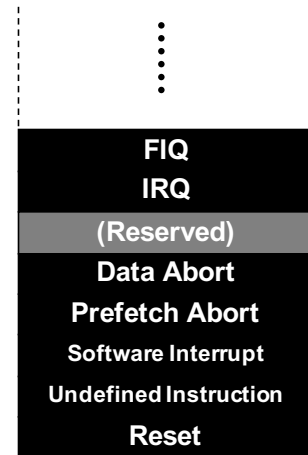
0x10

0x0C

0x08

0x04

0x00



Vector Table

Vector table can be at
0xFFFF0000 on ARM720T
and on ARM9/10 family devices

37

ARM Application Procedure Call Standard (AAPCS)

ARM Application Procedure Call Standard (AAPCS)

- defines a standard way of how registers are used in a typical program that is coded with routines, functions, and procedures

Examples:

- r0 – r3: use to pass arguments and return results between routines call
- r4 – r8, r10 and r11: use for local variables of each routine
- r12: use as scratchpad register

Hence, routines must preserve contents of r4–r8, r10, and r11, usually by saving onto the stack.

38

ARM Application Procedure Call Standard (AAPCS)

Arguments into function Result(s) from function otherwise corruptible (Additional parameters passed on stack)	Register		The compiler has a set of rules known as a Procedure Call Standard that determine how to pass parameters to a function (see AAPCS)
		r0	
		r1	
		r2	
Register variables Must be preserved		r3	CPSR flags may be corrupted by function call. Assembler code which links with compiled code must follow the AAPCS at external interfaces
		r4	
		r5	
		r6	
		r7	
		r8	
		r9/sb	
		r10/sl	
Scratch register (corruptible)		r11	- Stack base - Stack limit if software stack checking selected
		r12	
Stack Pointer Link Register Program Counter		r13/sp	- SP should always be 8-byte (2 word) aligned - R14 can be used as a temporary once value stacked
		r14/lr	
		r15/pc	

39

Banked Registers

Most of the registers in the ARM states are shared across different operating modes

- The same registers are used in all modes.
- The content must be saved first if the value needs to be preserved.

Banked registers are separate physical registers that are swapped in and out when switching modes

- Allow the immediate use of registers without saving the content first.

40

Banked Registers (cont' d)

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
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_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und
r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und
r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

ARM-state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

 = banked register

41

Banked Registers (cont)

Notes:

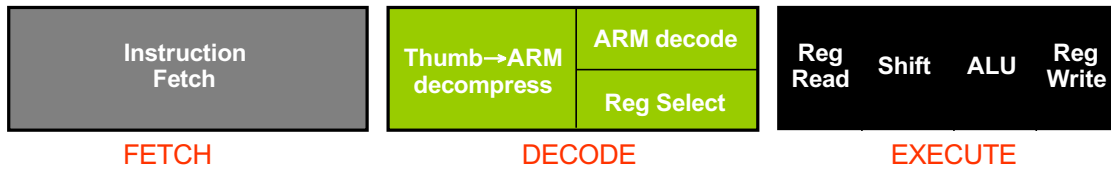
1. Different banks of r13 and r14 registers (used as the Stack pointer and Link register) are always swapped when the operating mode changes.
2. SPSRs are always available as banked registers.
3. FIQ mode (Fast interrupt) contains the largest banked register, which allows immediate access to 'fresh' r8 to r14 when entering this interrupt mode.

(Discussion: What is the advantage?)

42

ARM Core Pipelines

ARM7TDMI



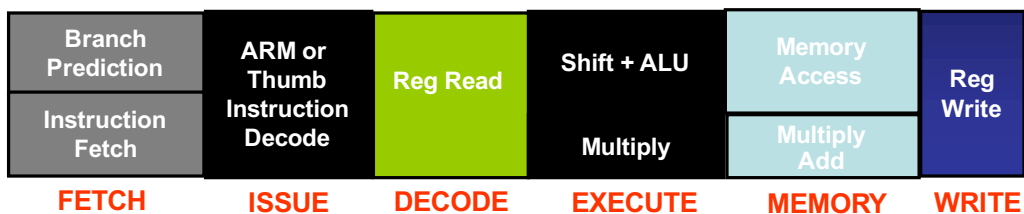
ARM9TDMI



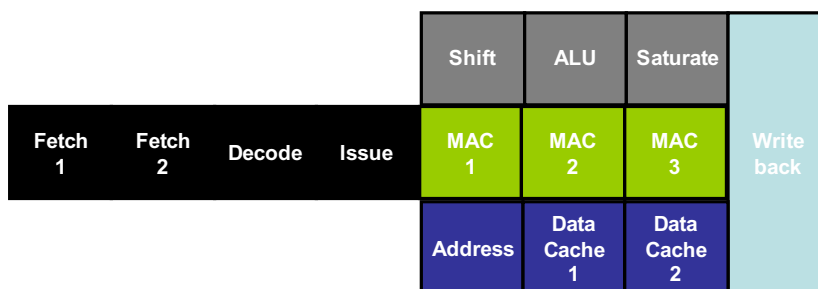
43

ARM Core Pipelines (cont)

ARM10



ARM11



Superscalar Architecture

44

The Thumb Processor State

45

Thumb Instruction Set

To reduce the higher memory footprint generally associated with RISC-based architecture

- ARM also provides 16-bit instructions: the Thumb instruction set

Can pack two Thumb instructions in one 32-bit memory location

- i.e., more instructions can be stored in the same memory device

Can use 16-bit data width memory device

- lowers the component cost

Hence, less memory means lower cost and lower operating power.

46

THUMB Instructions

The Thumb instruction set is a subset of the 32-bit ARM instruction set

- Some of the of 32-bit ARM instructions are not available when operating in THUMB state.

For codes that cannot be implemented with the THUMB instructions

- Two cycles are needed to fetch a single 32-bit instruction.

Additional constraint

- Restrictions in some of the instructions when used in the Thumb state. (e.g., Branches have a shorter range)
- Some of the registers cannot be used

47

Thumb State Registers

The Thumb state provides a subset of the ARM-state registers:

- Eight general purpose registers r0 to r7 (also known as the low registers)
- SP register (banked in all modes)
- LR register (banked in all modes)
- PC register
- CPSR register
- SPSR (banked for the privileged modes)

The high registers r8 to r15 are generally not accessible when in the Thumb state (except for the three instructions MOV, ADD, and CMP).


48

Thumb-State Registers (cont' d)

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
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
SP	SP_fiq	SP_svc	SP_abt	SP_irq	SP_und
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und
PC	PC	PC	PC	PC	PC

Thumb-state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

 = banked register

49

Switching Between States

The ARM processor starts up in ARM state on reset

- Switching between the Thumb state and the ARM state is accomplished through the instruction

BX Rn (Branch and Exchange)

where Rn contains the destination address

When in the Thumb state

- Status bit 'T' of the CPSR will be set to 1

50

Unified Memory Address

The v4T ARM uses a unified memory space with the 32-bit address bus;

- hence, the total addressable memory space:
 $2^{32}-1 = 4 \text{ GB}$

Memory and peripherals of specific functions are mapped in various addresses within the 4 GB address space;

- but certain address ranges are usually not utilized in typical implementation — indicated as RESERVED