ARM Microprocessor 1

ARM Processors

- ARM 1 in 1985
- By 2001, more than 1 billion ARM processors shipped
- Widely used in many successful 32-bit embedded systems

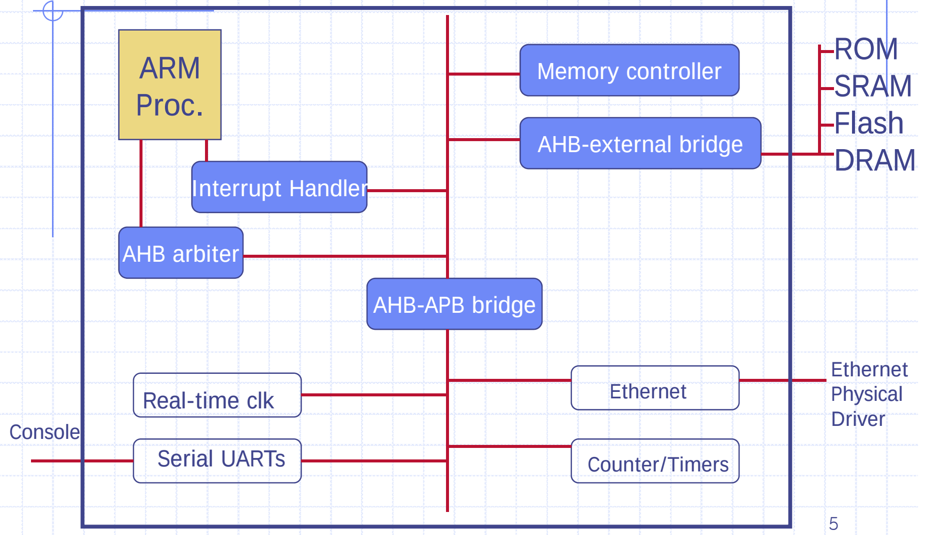
ARM Design Philosophy

- Low power
- High code density
- Low cost
- Small die size (i.e., more for peripherals)
- Hardware debug for the time to market

ARM ISA

- Load-store architecture
- Fixed 32-bit instructions
- Pipelining
- Register Number
- Not a pure RISC!
  - Variable cycle instructions
  - Complex instructions using inline barrel shifter
  - Thumb instruction set
  - Conditional execution
  - Enhanced instructions (DSP)

ARM-based Embedded Device



ARM Bus

- An on-chip bus technology necessary (vs. off-chip bus)
- Advanced Microcontroller Bus Architecture introduced in 1996.
  - ARM System Bus (ASB)
  - ARM Peripheral Bus (APB)
  - ARM High Performance Bus (AHB)
  - Plug-and-Play interface

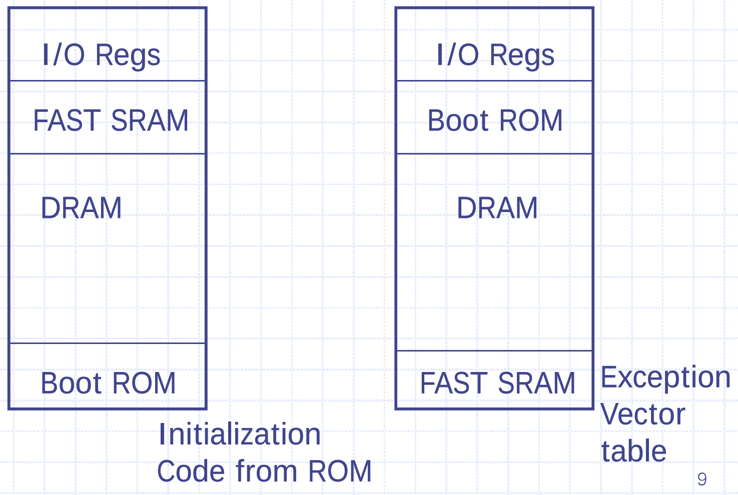
ARM Peripherals

- Memory mapped I/O
- Memory controller
- Interrupt controller
  - Standard interrupt controller
  - Vector interrupt controller

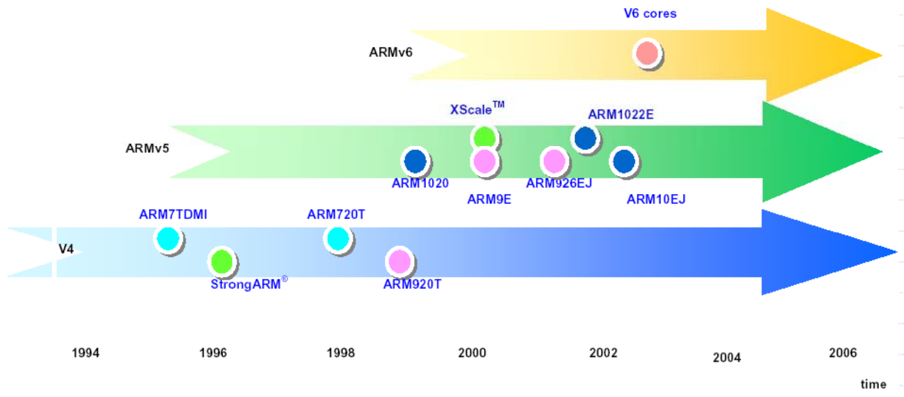
Embedded System Software

- Initialization (Boot) Code
  - Take the processor from the reset state to a state where OS can run
  - Three steps:
    - Initial H/W configuration
      - Memory remapping
    - Diagnostics
    - Booting

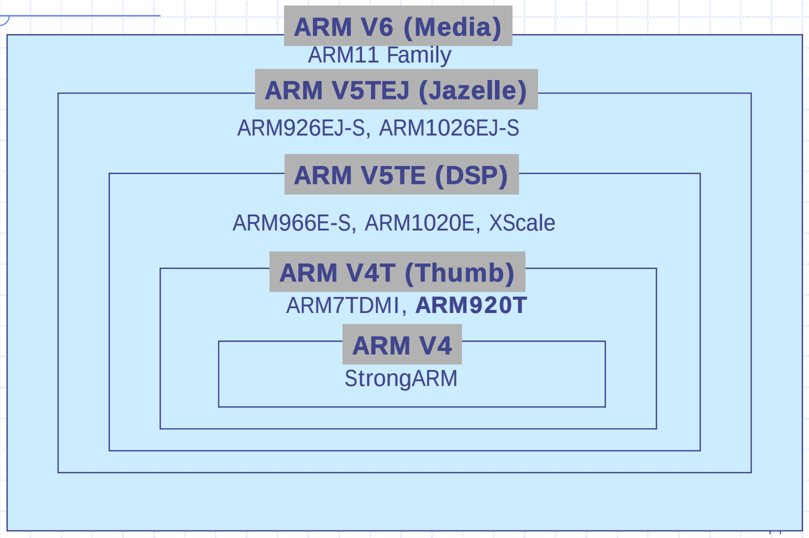
Memory Remapping



ARM Architecture History



ARM Instruction Set



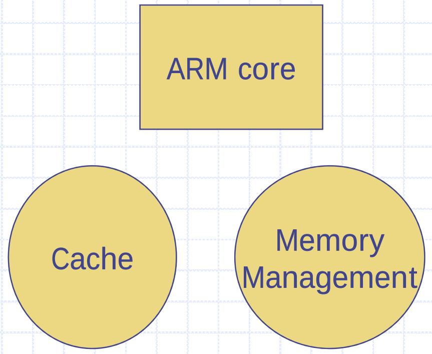
ARM Nomenclature

- ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{-S}
  - x = family, y = MMU/Protection unit, z = cache
  - T = Thumb
  - D = JTAG debug
  - M = fast mpy
  - I = EmbeddedICE macrocell
  - E = enhanced instructions
  - J = Jazelle
  - F = vector FP unit
  - S = synthesizible version

ARM Processor Numbering

x	y	z	Description	Example
7/9/10/11			ARM7/9/10/11 processor core	ARM7TDMI
	2/3/4/6		Cache+MMU/ Cache+MMUw/physical tag Cache+MPU/No cache (WB)	ARM920T ARM946E-S
		0	Standard cache size	ARM920T
		2	Reduced cache size	ARM922T
		6	TCM	ARM946E-S

ARM Processor Organization



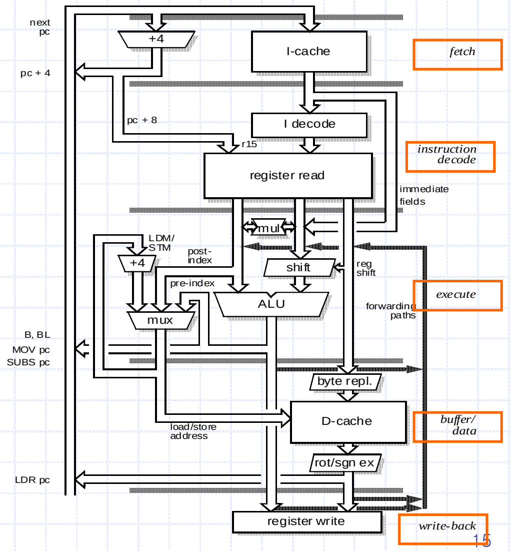
ARM9TDMI Pipeline

ARM7

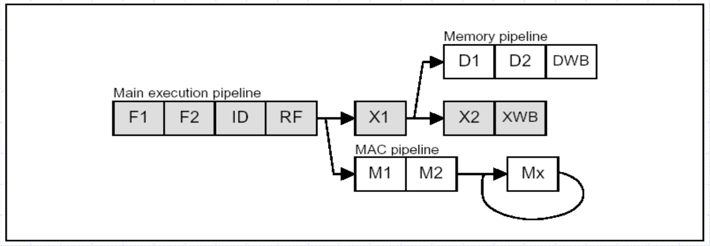
1. Fetch
2. Decode
3. Execute

ARM9

1. Fetch
2. Decode
3. Execute
4. Data/Buffer
5. Write Back



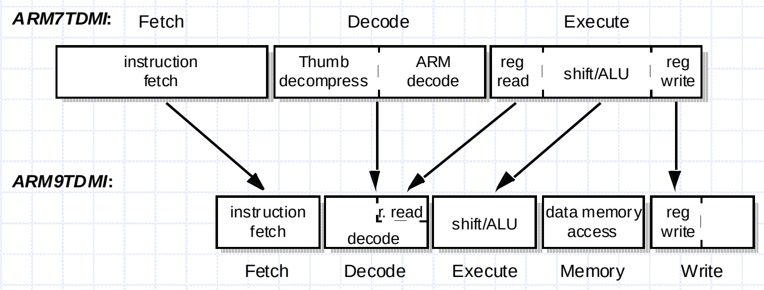
Intel XScale Pipeline



ARM9TDMI

- ARM9TDMI Core
  - 5-stage pipeline
  - Separate instruction/data memory ports
  - 32-bit ARM & 16-bit THUMB instruction set support
  - Coprocessor interfaces for FP and DSP
  - On-chip debug support

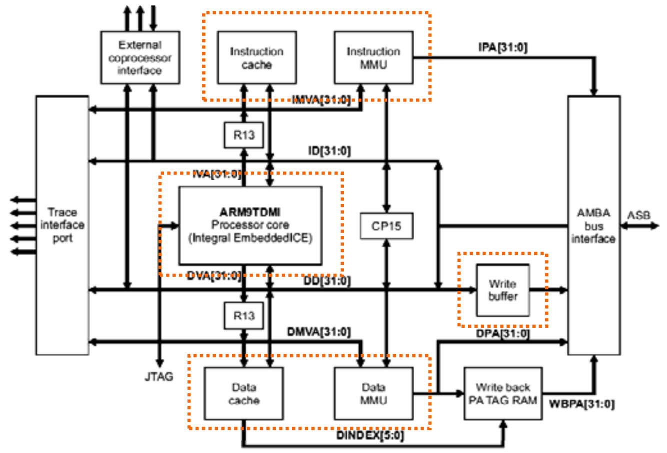
ARM9TDMI Pipeline



ARM 920T

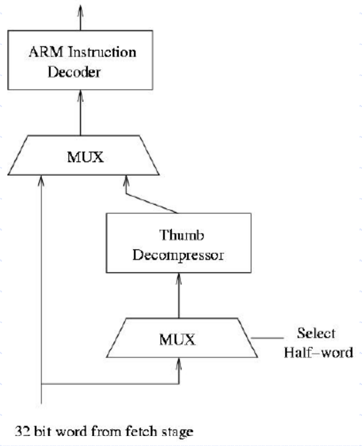
- ARM920T
  - ARM9TDMI core based
  - Instruction/Data Cache
  - MMU, write buffer

ARM920T Functional Block Diagram

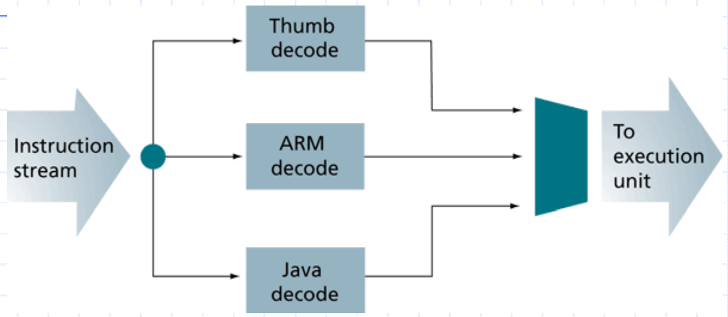


ARM and Thumb

- 32bit ARM Instruction Sets
- 16 bit Thumb Instruction Set
  - Small code size
  - Low instruction cache energy
- All instructions are executed as ARM
- Decompressor converts Thumb to equivalent ARM instruction
- Decompressor present in the decode stage of the pipeline.



Jazelle



- Similar to Thumb extension!
- Java mode like Thumb mode
- CPU core reconfigured to support JVM
  (e.g., Top 4 elements in Java stack mapped to r0 – r3)

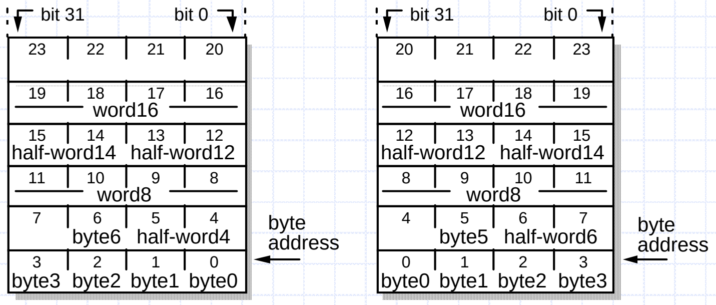
Data Types Supported

- 8-bit signed and unsigned bytes
- 16-bit signed and unsigned half-words
- 32-bit signed and unsigned words

- Alignment Required

Memory organization

- Default: little-endian



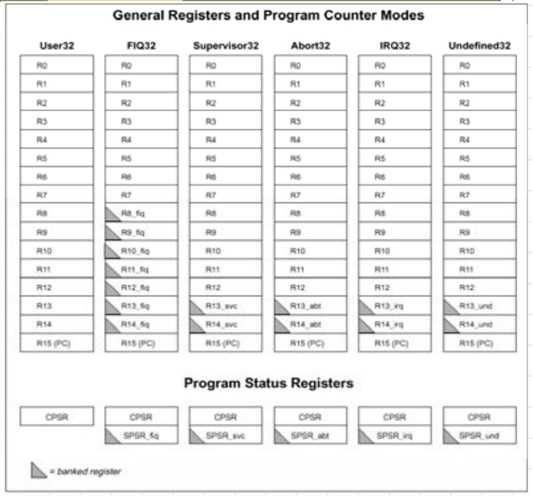
(a) Little-endian memory organization

(b) Big-endian memory organization

Registers

Banked registers
20 registers hidden
Available only in a particular mode

- 31 32-bit registers
  - User mode: R0~R15 (16 reg.)
  - FIQ mode: R8_fiq~R14_fiq (7)
  - 4 other modes : 2 each, R13_xxx~R14_xxx (8)
- 6 Program Status Register
  - PSR 또는 CPSR


General Registers and Program Counter Modes

User32	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
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)

Program Status Registers

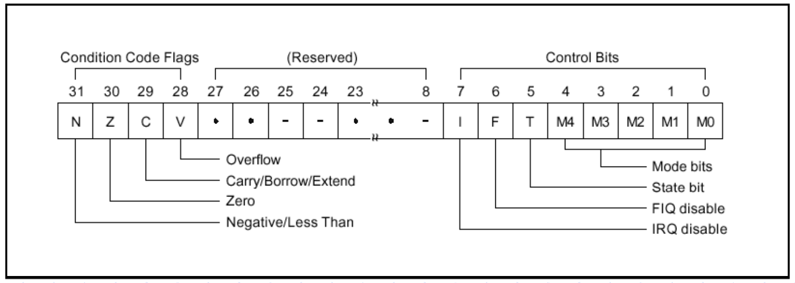
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	◣ SPSR_fiq	◣ SPSR_svc	◣ SPSR_abt	◣ SPSR_irq	◣ SPSR_und

◣ = banked register

Special Register Usage

- Program Counter (r15)
  - Can be used as an operand
  - '**pc**' keyword
- Link Register (r14)
  - Used in BL (branch with link)
  - Move current **pc** to **lr** before jump
  - RET == **mov pc, lr**
- Stack Pointer (r13)
  - Software convention, not hard-wired

ARM Current Status Register



M[4:0]	Mode	Accessible register set	
10000	User	PC, R14..R0	CPSR
10001	FIQ	PC, R14_fiq..R8_fiq, R7..R0	CPSR, SPSR_fiq
10010	IRQ	PC, R14_irq..R13_irq, R12..R0	CPSR, SPSR_irq
10011	Supervisor	PC, R14_svc..R13_svc, R12..R0	CPSR, SPSR_svc
10111	Abort	PC, R14_abt..R13_abt, R12..R0	CPSR, SPSR_abt
11011	Undefined	PC, R14_und..R13_und, R12..R0	CPSR, SPSR_und

Processor Modes

- ARM processor modes in version 4

Processor mode		Description
User	usr	Normal program execution mode
FIQ	fiq	Supports a high-speed data transfer or channel process
IRQ	irq	Used for general-purpose interrupt handling
Supervisor	svc	A protected mode for the operating system
Abort	abt	Implements virtual memory and/or memory protection
Undefined	und	Supports software emulation of hardware coprocessors
System	sys	Runs privileged operating system tasks (ARM architecture version 4 and above)

ARM Exceptions

- IRQ: interrupt request
- FIQ: fast interrupt request
  - Conceptually same as IRQ
  - Support a faster dispatch of ISR
- Software Interrupt
  - Via SWI instruction
- Abort
  - When there is a failed attempt to access memory
  - Prefetch Abort & Data Abort
- Undefined Instruction Trap.

Exceptions & Modes

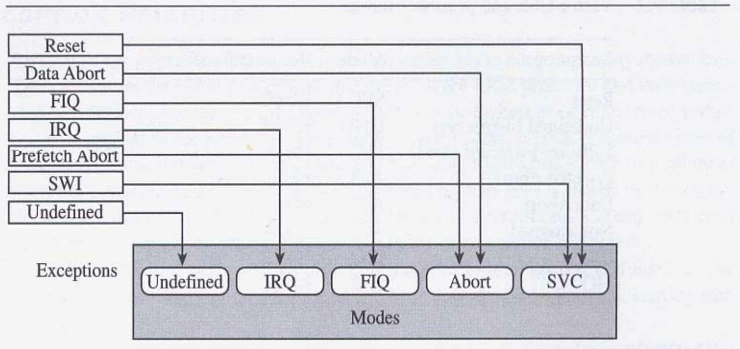


Figure 9.1 Exceptions and associated modes.

Source: ARM System Developer's Guide

ARM Exception Processing

- Vector Table at 0x00
- **Jump to Exception handler** in the vector table entry

Exception Vector Table

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	*Reserved*	*Reserved*
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Exception Vector Table (head.s)

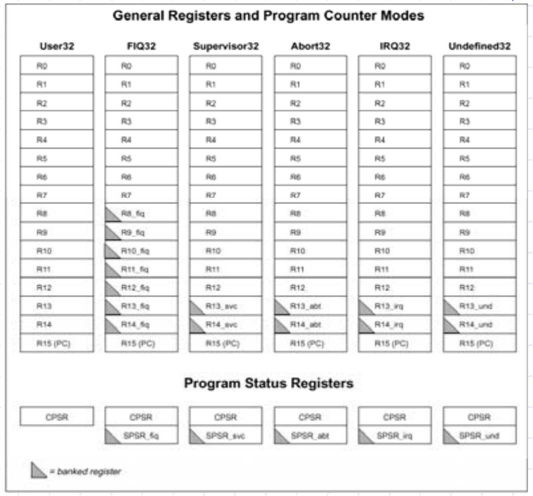
```
0x00:                   b       reset_handler
0x04:                   b       vector_undefinstr
0x08:                   b       vector_swi
0x0c:                   b       vector_prefetch
0x10:                   b       vector_data
0x14:                   b       vector_addrexcptn
0x18:                   b       vector_IRQ
0x1c:                   b       _unexp_fiq
0x20:   reset_handler:  mov     r0, #(MODE_IRQ)
0x24:                   msr     cpsr, r0
0x28:                   ldr     r13, =IRQ_STACK
0x2c:                   mov     r0, #(MODE_SVC|I_BIT|F_BIT)
0x30:                   msr     cpsr, r0
0x34:                   ldr     r13, =SVC_STACK
0x38:                   mov     fp, #0
0x3c:                   b       start_kernel
```

Processor Mode vs. Registers

- Registers are remapped depending on the processor mode
- Reducing the overhead of saving and restoring registers

Registers

- 31 32-bit registers
  - User mode: R0~R15 (16 reg.)
  - FIQ mode: R8_fiq~R14_fiq (7)
  - 4 other modes : 2 each, R13_xxx~R14_xxx (8)
- 6 Program Status Register
  - PSR 또는 CPSR



General Registers and Program Counter Modes

User32	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
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)

Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	◣ SPSR_fiq	◣ SPSR_svc	◣ SPSR_abt	◣ SPSR_irq	◣ SPSR_und

◣ = banked register

Example: IRQ Mode

- In User mode, r0~r15 used
- If there is an interrupt request,
  - Processor mode = IRQ
  - r13_irq & r14_irq available (instead of r13 & r14)
  - A separate stack pointer
  - PC in r14(r14_irq)
  - If necessary, r0~r12 must be saved before used

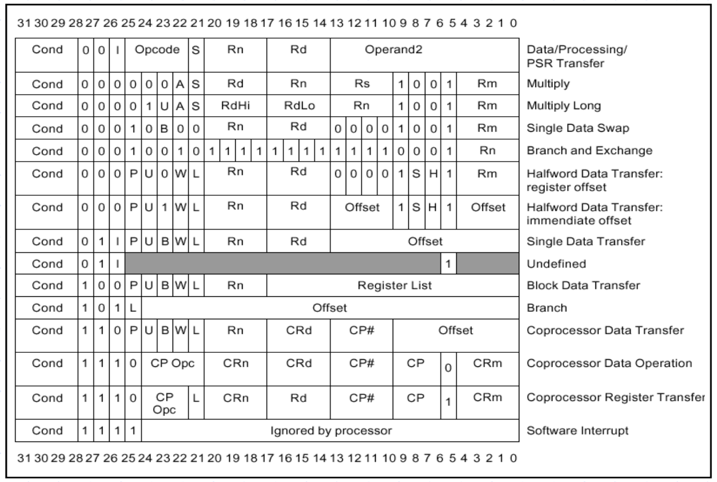
Processor Mode vs. PSR

- CPSR is saved into a corresponding SPSR
- Example: SPSR_irq ← CPSR when IRQ
  - Both CPSR & SPSR_irq accessible in IRQ
  - **MRS r1, cpsr**

Exception Priority

1) Reset (The Highest)
2) Data abort
3) FIQ
4) IRQ
5) Prefetch abort
6) Undefined Instruction, Software Interrupt

ARM Instruction Set: Overview



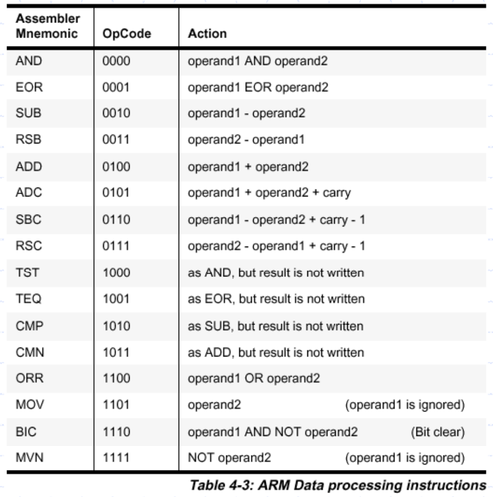
31 30 29 28	27	26	25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6	5	4	3 2 1 0	
Cond	0	0	I	Opcode				S	Rn	Rd	Operand2						Data/Processing/ PSR Transfer
Cond	0	0	0	0	0	0	A	S	Rd	Rn	Rs	1	0	0	1	Rm	Multiply
Cond	0	0	0	0	1	U	A	S	RdHi	RdLo	Rn	1	0	0	1	Rm	Multiply Long
Cond	0	0	0	1	0	B	0	0	Rn	Rd	0 0 0 0	1	0	0	1	Rm	Single Data Swap
Cond	0	0	0	1	0	0	1	0	1 1 1 1	1 1 1 1	1 1 1 1	0	0	0	1	Rn	Branch and Exchange
Cond	0	0	0	P	U	0	W	L	Rn	Rd	0 0 0 0	1	S	H	1	Rm	Halfword Data Transfer: register offset
Cond	0	0	0	P	U	1	W	L	Rn	Rd	Offset	1	S	H	1	Offset	Halfword Data Transfer: immendiate offset
Cond	0	1	I	P	U	B	W	L	Rn	Rd	Offset						Single Data Transfer
Cond	0	1	I	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	1	▨	Undefined
Cond	1	0	0	P	U	B	W	L	Rn	Register List							Block Data Transfer
Cond	1	0	1	L	Offset												Branch
Cond	1	1	0	P	U	B	W	L	Rn	CRd	CP#	Offset					Coprocessor Data Transfer
Cond	1	1	1	0	CP Opc				CRn	CRd	CP#	CP			0	CRm	Coprocessor Data Operation
Cond	1	1	1	0	CP Opc			L	CRn	Rd	CP#	CP			1	CRm	Coprocessor Register Transfer
Cond	1	1	1	1	Ignored by processor												Software Interrupt

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

ARM Instruction Set Overview - Condition Codes

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

ARM Instruction Set Overview - Data processing Instructions



Assembler Mnemonic	OpCode	Action
AND	0000	operand1 AND operand2
EOR	0001	operand1 EOR operand2
SUB	0010	operand1 - operand2
RSB	0011	operand2 - operand1
ADD	0100	operand1 + operand2
ADC	0101	operand1 + operand2 + carry
SBC	0110	operand1 - operand2 + carry - 1
RSC	0111	operand2 - operand1 + carry - 1
TST	1000	as AND, but result is not written
TEQ	1001	as EOR, but result is not written
CMP	1010	as SUB, but result is not written
CMN	1011	as ADD, but result is not written
ORR	1100	operand1 OR operand2
MOV	1101	operand2 (operand1 is ignored)
BIC	1110	operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

Table 4-3: ARM Data processing instructions

ARM instruction

- Conditional instruction
  - **BEQ** **jmp_1** ; Branch to jmp_1 if Z flag set
  - **MOVEQ** **r0,r1** ; r0 := r1 if Z flag set
- S flag: determines if the flag fields are affected or not
  - **SUBS** **r2, r2, #1** ; dec r2 and set cc

ARM instruction examples

- MOV r0, r1 ; r0:=r1
- MOV r0, #0x30 ; r0:=0x30
- MOV r0, r1, LSL #1 ; r0:= r1 <<1, LSL(=logical Shift Left)
- MOV r0, r1, LSR r2 ; r0:=r1 >> r2 , LSR(=Logical Shift Right)
- MOV r0, r0, ASR #24 ; r0:=r0 >> 24, ASR(=Arithmetic Shift Right)
- MOVS r0, r1, LSR #1 ; C(flag) := r1[0]
- MOVCC r0, #10 ; if C=0 then r0:=10
- MOVCS r0, #11 ; if C=1 then r0:=11

Data Processing Instructions

- Data processing instruction types
  - Arithmetic operations
  - Bit-wise logical operations
  - Register-movement operations
  - Comparison operations
  - Multiply instructions
- 3-address format
- 32-bit operands:
  - register
  - constant (immediate) (in the second operand)
  - shifted register
- 32-bit results, stored in a register
  - 64-bit results for long multiples

Data Processing Instructions (cont'd)

Arithmetic Operations

ADD r0, r1, r2	r0 := r1 + r2
ADC r0, r1, r2	r0 := r1 + r2 + C
SUB r0, r1, r2	r0 := r1 - r2
SBC r0, r1, r2	r0 := r1 - r2 + C - 1
RSB r0, r1, r2	r0 := r2 – r1
RSC r0, r1, r2	r0 := r2 – r1 + C - 1

Bit-wise Logical Operations

AND r0, r1, r2	r0 := r1 and r2
ORR r0, r1, r2	r0 := r1 or r2
EOR r0, r1, r2	r0 := r1 xor r2
BIC r0, r1, r2	r0 := r1 and (not) r2

Register Movement

MOV r0, r2	r0 := r2
MVN r0, r2	r0 := not r2

Comparison Operations

CMP r1, r2	set cc on r1 - r2
CMN r1, r2	set cc on r1 + r2
TST r1, r2	set cc on r1 and r2
TEQ r1, r2	set cc on r1 xor r2

Data Processing Instructions (cont'd)

- Immediate operands:

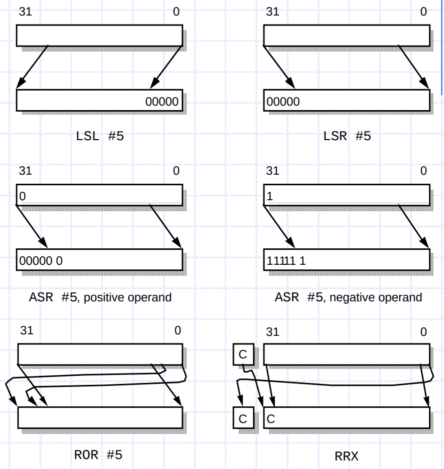
ADD r3, r3, #3	r3 := r3 + 3
AND r8, r7, #&ff	r8 := $r7_{[7:0]}$, & for hex

- Shifted register operands

ADD r3, r2, r1, LSL #3	r3 := r2 + 8 x r1
ADD r5, r5, r3, LSL r2	r5 := r5 + 2^{r2} x r3

ARM shift operations

- LSL – Logical Shift Left
- LSR – Logical Shift Right
- ASR – Arithmetic Shift Right
- ROR – Rotate Right
- RRX – Rotate Right Extended by 1 place



Setting the condition codes

- Data processing instructions can change condition codes (N, Z, V, C) using 'S' suffix
  - Except for comparison instructions
  - Example (r3-r2 := r1-r0 + r3-r2)

```
ADDS r2, r2, r0    ; carry out to C
ADC r3, r3, r1     ; ... add into the high word
```

- Logical & move instructions update, C, N & Z flags.

Multiplies

MUL r4, r3, r2	r4 := [r3 x r2]$_{<31:0>}$
MLA r4, r3, r2, r1	r4 := [r3 x r2 + r1]$_{<31:0>}$

- Example (r0 = r0 x 35)
  - ADD r0, r0, r0, LSL #2 ; r0' = r0 x 5 (=4+1)
    RSB r0, r0, r0, LSL #3 ; r0'' = r0' x 7 (=8-1)

MUL vs. ADD and SUB

- Multiply by 5
  - MOV r2, #5
    MUL r0, r1, r2
  - ADD r0, r1, r1, lsl #2
- Multiply by 105
  - 105 = 128 -16 +3

ADD	r0, r1, r1, lsl #1;	r0 = r1*3
SUB	r0, r0, r1, lsl #4;	r0 = r1*3 – r1*16
ADD	r0, r0, r1, lsl #7;	r0 = r1*3 – r1*16 + r1*128

  - 105 = 15 * 7 = (16 -1) * (8 -1)

RSB	r2, r1, r1, lsl #4;	r2 = r1*15
RSB	r0, r2, r2, lsl #3;	r0 = r2*7

- Multiply by 85

ADD	r0, r1, r1, lsl #4;	r0 = r1*17
ADD	r0, r0, r0, lsl #2;	r0 = r0*5 => *85