ARM Cortex-A9
ARM v7-A

A programmer’s perspective
Part1

ARM: Advanced RISC Machine

First appeared in 1985 as “Acorn RISC Machine” from Acorn Computers in Manchester England

Limited success – outcompeted by x86’s for PCs, so started targeting embedded applications

Interest in using ARM in emerging embedded applications grew (like Apple’s first PDA), which led to the reformation of the company in 1990

“Advanced RISC Machines” took and updated architecture, and began selling IP instead of chips. First IP core shipped in 1991.

In early 90’s ARM abandons all fab, and begins licensing cores to world’s leading silicon companies (Sharp, TI, Philips, Lucent, HP, IBM, QualCom, ST, Agilent, Altera, Samsung, etc.)

By 2002, ARM had over 75% of embedded market, and had sold more than 1 billion cores.

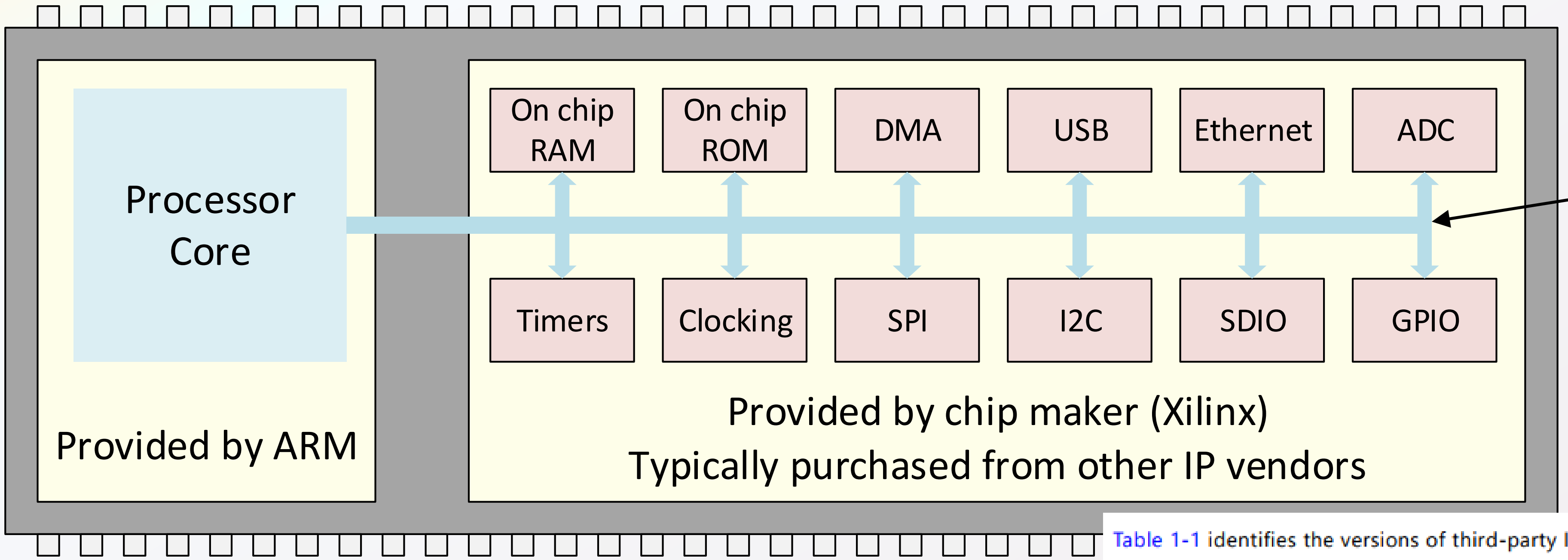
ARM Cortex-A9 launched in 2007, offering scalable performance and low power. More than 10 billion cores sold.

ARM currently has many variants, and is pushing downward into 8-bit market, and upward into high-performance computing. They remain fabless.

Performance and scalability for a diverse range of applications

	Previous	Armv6	Armv7	Armv8	
Cortex-A	Armv5	Armv6	Armv7-A	Armv8-A	
			Cortex-A17 Cortex-A15	Cortex-A73 Cortex-A75 Cortex-A57 Cortex-A72	High performance
		Arm11MPCore Arm1176JZ(F)-S Arm1136J(F)-S	Cortex-A9 Cortex-A8	Cortex-A53 Cortex-A55	High efficiency
	Arm968E-S Arm946E-S Arm926EJ-S		Cortex-A7 Cortex-A5	Cortex-A35 Cortex-A32	Ultra high efficiency
Cortex-R			Armv7-R	Armv8-R	
		Arm1156T2(F)-S	Cortex-R8 Cortex-R7 Cortex-R5 Cortex-R4	Cortex-R52	Real time
Cortex-M	Armv4	Armv6-M	Armv7-M	Armv8-M	
			Cortex-M7		High performance
	Arm7TDMI Arm920T		Cortex-M4 Cortex-M3	Cortex-M33	Performance efficiency
		Cortex-M0+ Cortex-M0		Cortex-M23	Lowest power and area

arm



System bus specified by ARM, implemented by chip maker

The ARM processor core is provided by ARM, and so behaves the same regardless of vendor. But all the peripherals are vendor specific, so they will be different on different “ARM” chips.

Table 1-1 identifies the versions of third-party IP used in the Zynq-7000 AP SoC devices.

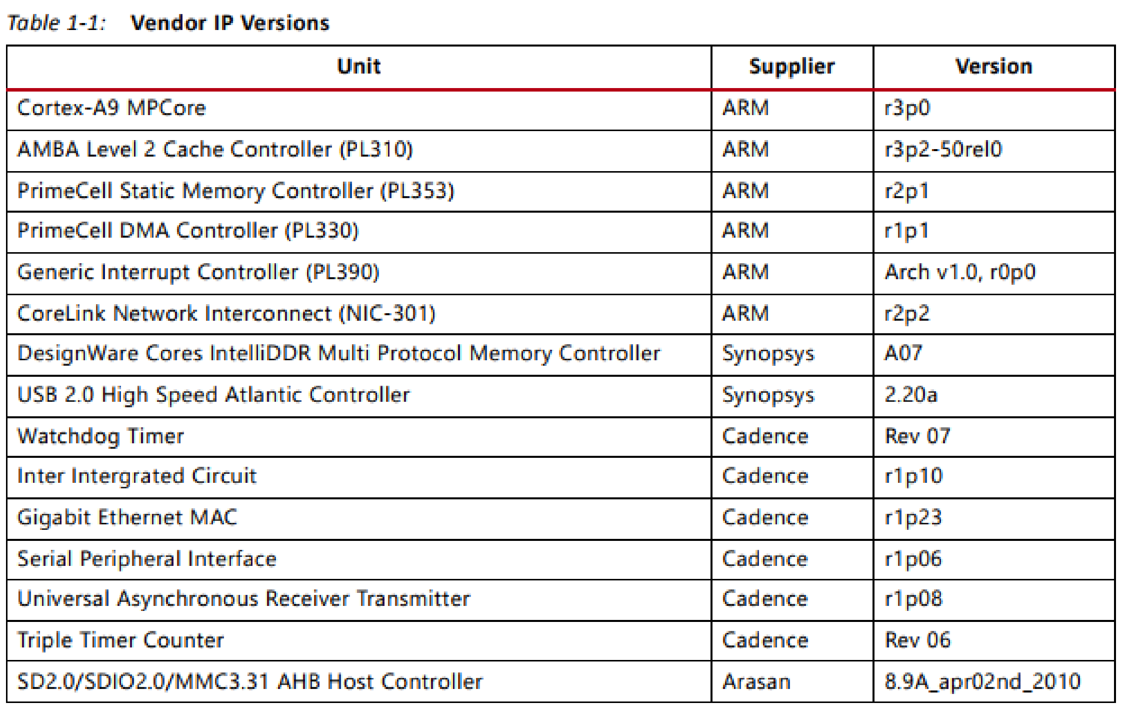
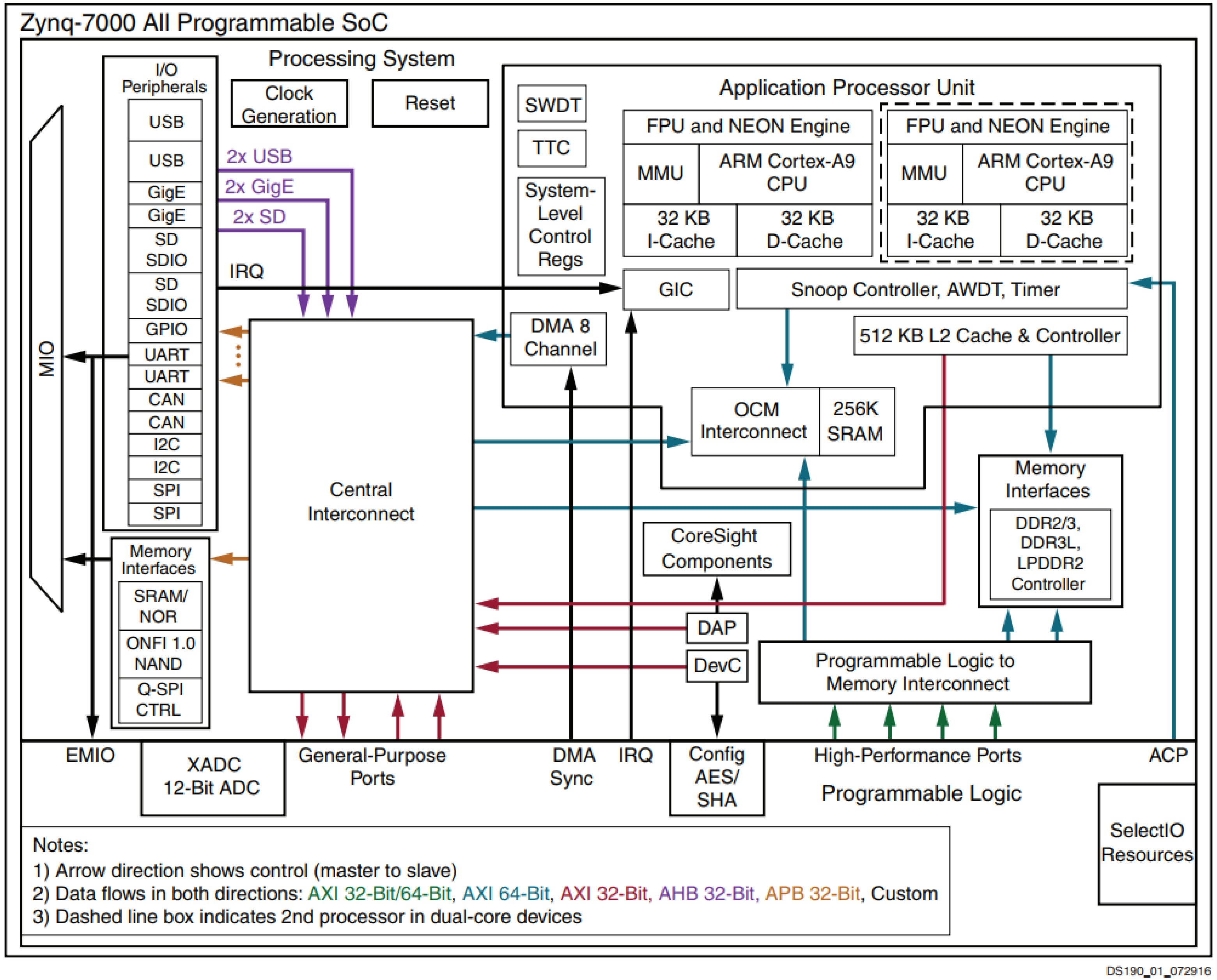


Table 1-1: **Vendor IP Versions**

Unit	Supplier	Version
Cortex-A9 MPCore	ARM	r3p0
AMBA Level 2 Cache Controller (PL310)	ARM	r3p2-50rel0
PrimeCell Static Memory Controller (PL353)	ARM	r2p1
PrimeCell DMA Controller (PL330)	ARM	r1p1
Generic Interrupt Controller (PL390)	ARM	Arch v1.0, r0p0
CoreLink Network Interconnect (NIC-301)	ARM	r2p2
DesignWare Cores IntelliDDR Multi Protocol Memory Controller	Synopsys	A07
USB 2.0 High Speed Atlantic Controller	Synopsys	2.20a
Watchdog Timer	Cadence	Rev 07
Inter Intergrated Circuit	Cadence	r1p10
Gigabit Ethernet MAC	Cadence	r1p23
Serial Peripheral Interface	Cadence	r1p06
Universal Asynchronous Receiver Transmitter	Cadence	r1p08
Triple Timer Counter	Cadence	Rev 06
SD2.0/SDIO2.0/MMC3.31 AHB Host Controller	Arasan	8.9A_apr02nd_2010

ARM on ZYNQ

- Up to 1GHz; 2.5DMIPS/MHz
- TrustZone security
- Thumb-2 instruction set
- Jazelle execution environment
- Neon media processor
- FPU
- Coresight and Program Trace
- Six timers
- 32KByte Level 1 cache
- 512KByte Level 2 cache
- On-chip boot ROM
- 256KByte on-chip RAM
- Memory controller; 1 GByte space
- 8-channel DMA
- Two tri-speed Ethernet MACs
- Two USB 2.0 OTG perupherals
- Two CAN 2.0 bus interfaces
- Two SDIO 2.0 controllers
- Two full-duplex SPI ports
- Two high-speed UARTs
- Two I2C interfaces
- 128 GPIO pins
- 54 MIO pins
- Artix 35T FPGA with AMBA-AXI bus
- Block RAMs
- Two 12-bit ADCs
- Etc…



Notes:
1) Arrow direction shows control (master to slave)
2) Data flows in both directions: AXI 32-Bit/64-Bit, AXI 64-Bit, AXI 32-Bit, AHB 32-Bit, APB 32-Bit, Custom
3) Dashed line box indicates 2nd processor in dual-core devices

Figure 1: **Architectural Overview**

Zynq-7000 All Programmable SoC Data Sheet: Overview: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

ARM Overview

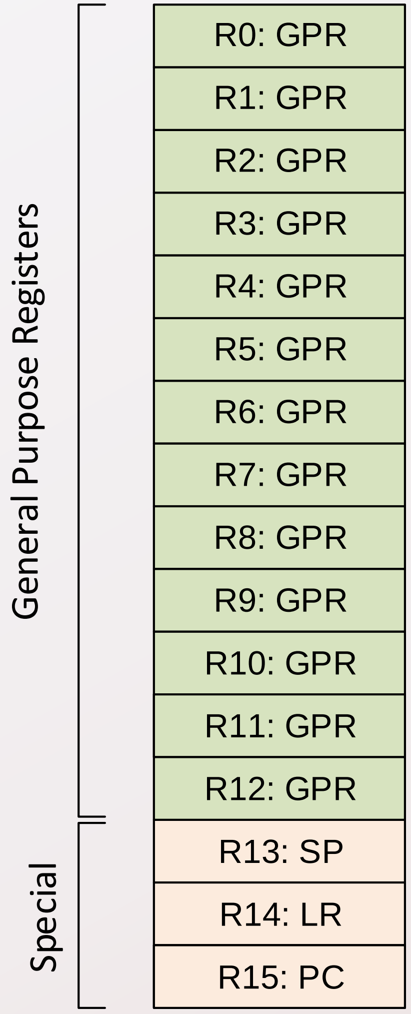
The ARM is a load-store, RISC machine. All inputs to the ALU must come from registers, and the ALU results must be stored in a register: data from memory must be “loaded” into a register before ALU can use it, and results “stored” from a register back into main memory.

ARM uses 16 32-bit registers. 13 are General Purpose Registers (GPRs) that can be used for temporary operand storage. GPR13 is the stack pointer, 14 is the link register, and 15 is the PC. All instructions can access r0-r14 directly, but r13 and r14 should be used with care.

PC is generally not accessible by instructions (some can). Bits 31:2 store PC. Why?

Registers can hold 32-bit pointers, signed or unsigned 32-bit integers, unsigned 16 or 8 bit integers (zero-extended), signed 16 or 8 bit integers (sign extended), two “packed” 16-bit integers, four packed 8-bit integers, or signed or unsigned 64-bit integers held in two consecutive registers.

The ALU status bits are available in the “application program status register” (APSR).



General Purpose Registers

R0: GPR

R1: GPR

R2: GPR

R3: GPR

R4: GPR

R5: GPR

R6: GPR

R7: GPR

R8: GPR

R9: GPR

R10: GPR

R11: GPR

R12: GPR

Special

R13: SP

R14: LR

R15: PC

APSR (CPSR)

	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	
APSR	N	Z	C	V	Q	RESERVED							GE				RESERVED																ALU status

	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	
CPSR	N	Z	C	V	Q	IT		J	RSRV				GE				IT							E	A	I	F	T	M				System status

The Application Program Status Register (APSR) holds the ALU condition flags. Same as CPSR, read by applications

N: Negative flag – set to 1 if ALU result is negative (Bit 31 is set)
Z: Zero flag – set to 1 if ALU result is zero (All bits are zero)
C: Carry flag – set to 1 on carry-out of ALU bit 31
V: Overflow condition flag – set to 1 on overflow or underflow
Q: Saturation flag – set to 1 on “saturation” of certain DSP-related instructions
GE: Greater than or equal flags, set for certain byte or half-word instructions

The Current Program Status Register can be read by “system” to get additional information

IT: If-then execution state bits for Thumb mode
J,T: Execution state bits (ARM, Thumb, Jazelle, or Thumb EE)
E: Endianess – 0 for little endian (default)
A,I,F: Mask/Disable bits for interrupt states
M: Mode (User, FIQ, IRQ, Supervisor, Monitor, Abort)

APSR (CPSR)

	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	
APSR	N	Z	C	V	Q	RESERVED							GE				RESERVED																ALU status

	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	
CPSR	N	Z	C	V	Q	IT		J	RSRV				GE				IT							E	A	I	F	T	M				System status

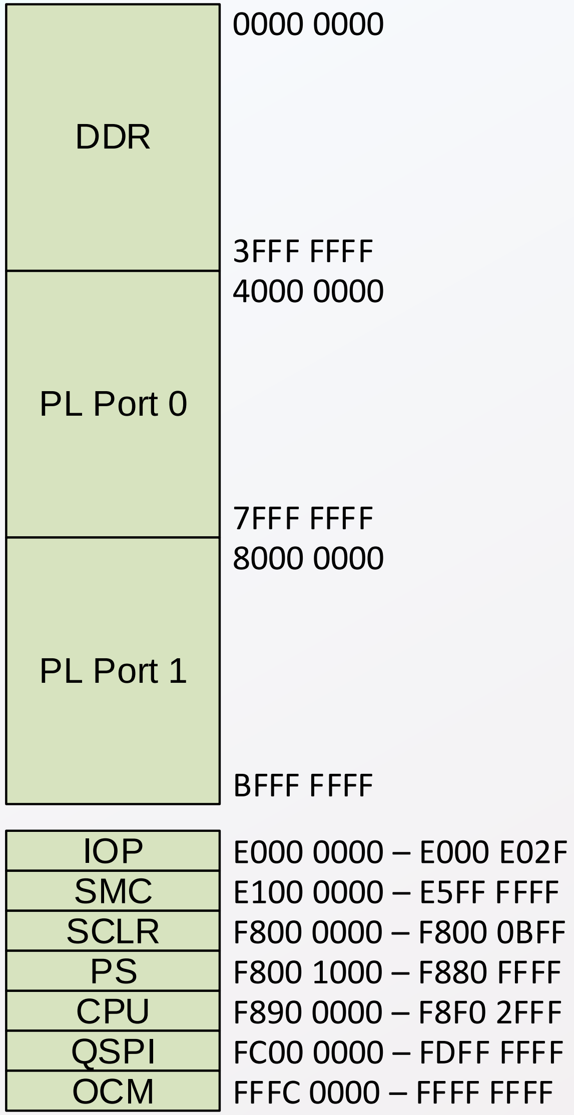
By default, data processing instructions DO NOT affect the condition flags (except for comparisons)

To cause condition flags to be updated, the S bit in the instruction opcode must be set.

```
ADD     R0, R1, R2          @ R0 <- R1 + R2 but CC not affected
ADDS    R0, R1, R2          @ R0 <- R1 + R2 and CC updated
```

Memory Map

The ARM has a 32-bit address space (or up to 40 bits with certain extensions). That’s 4GBytes.



DDR 0000 0000 – 3FFF FFFF

PL Port 0 4000 0000 – 7FFF FFFF

PL Port 1 8000 0000 – BFFF FFFF

IOP E000 0000 – E000 E02F

SMC E100 0000 – E5FF FFFF

SCLR F800 0000 – F800 0BFF

PS F800 1000 – F880 FFFF

CPU F890 0000 – F8F0 2FFF

QSPI FC00 0000 – FDFF FFFF

OCM FFFC 0000 – FFFF FFFF

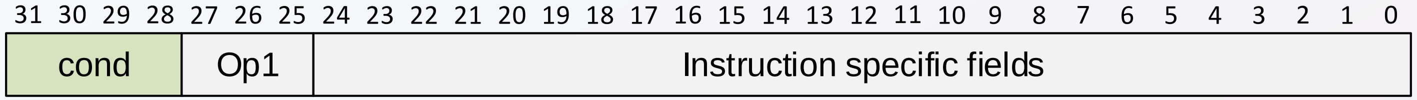
ZYNQ’s ARM assigns the memory space as shown. How much DDR can be used?

FPGA AXI Port 0

FPGA AXI Port 1

IOP: IO peripheral registers (UART, CAN, SPI, I2C, QSPI, Ethernet, SDIO)
SMC: Static Memory Control (NAND/NOR flash)
SCLR: System-level Control Registers (Clk, Rst, APU, MIO pins, etc.)
PS: Processing System Control (Timers/counters, watchdog, OCM, etc.)
CPU: CPU control (Top-level interconnect, Interrupt, cache, etc.)
QSPI: Quad SPI linear address space
OCM: On-chip SRAM

Instruction Formats



The 4-bit condition field is included in every opcode – almost all ARM instructions can be conditional.

The Mnemonic extension can be added to virtually any mnemonic to make the instruction conditional.

ADD r0, r1, r2 //r0 <= r1 + r2 *always*
ADDCS r0, r1, r2 //r0 <= r1 + r2 if C =1
B label1 //PC <=[label1] always
BNE label1 //PC <=[label1] if Z = 0

What's the advantage?

(Non-executed instructions vs. branches)

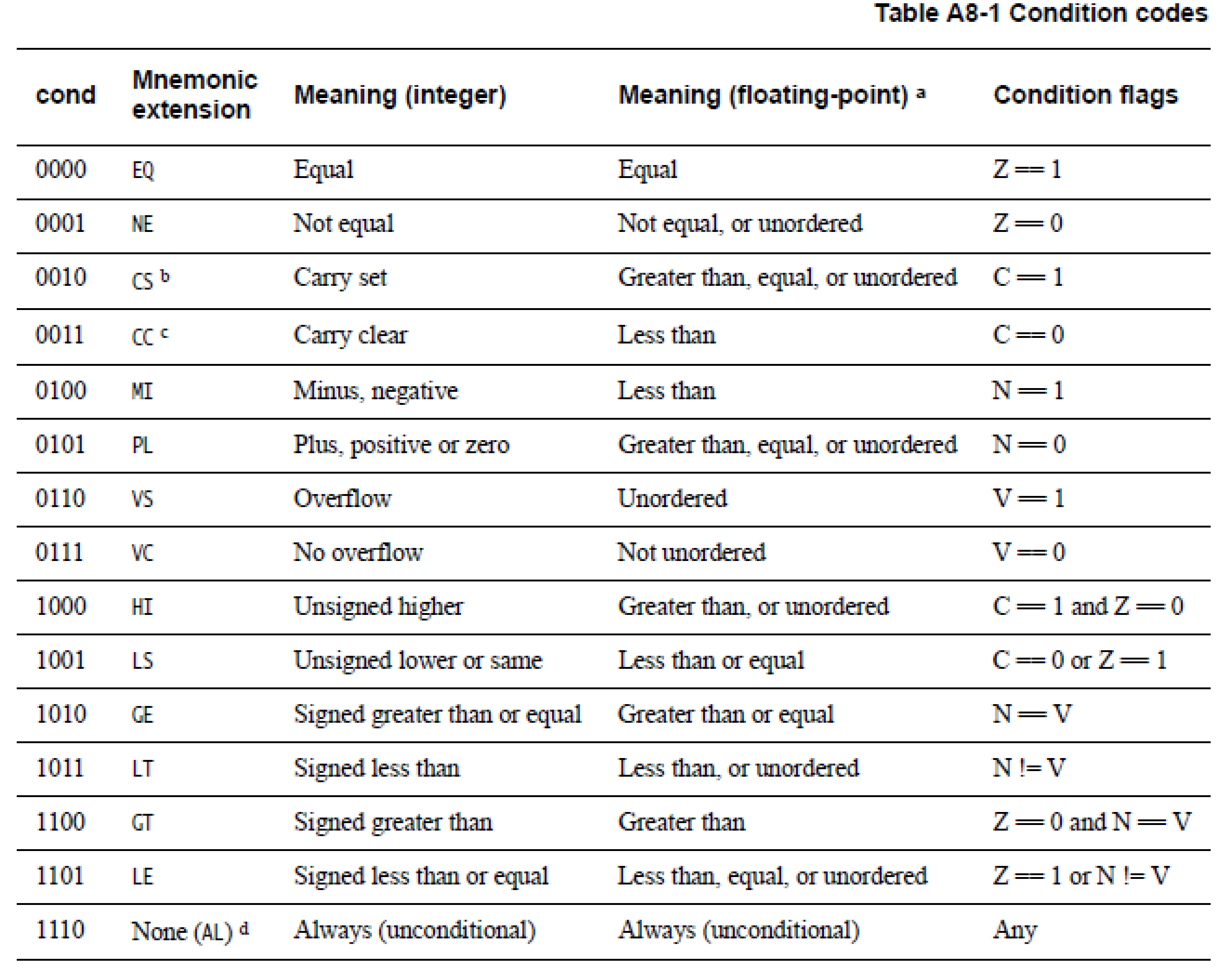


Table A8-1 Condition codes

cond	Mnemonic extension	Meaning (integer)	Meaning (floating-point) [a]	Condition flags
0000	EQ	Equal	Equal	Z == 1
0001	NE	Not equal	Not equal, or unordered	Z == 0
0010	CS [b]	Carry set	Greater than, equal, or unordered	C == 1
0011	CC [c]	Carry clear	Less than	C == 0
0100	MI	Minus, negative	Less than	N == 1
0101	PL	Plus, positive or zero	Greater than, equal, or unordered	N == 0
0110	VS	Overflow	Unordered	V == 1
0111	VC	No overflow	Not unordered	V == 0
1000	HI	Unsigned higher	Greater than, or unordered	C == 1 and Z == 0
1001	LS	Unsigned lower or same	Less than or equal	C == 0 or Z == 1
1010	GE	Signed greater than or equal	Greater than or equal	N == V
1011	LT	Signed less than	Less than, or unordered	N != V
1100	GT	Signed greater than	Greater than	Z == 0 and N == V
1101	LE	Signed less than or equal	Less than, equal, or unordered	Z == 1 or N != V
1110	None (AL) [d]	Always (unconditional)	Always (unconditional)	Any

Instruction Formats

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				Op1			Instruction specific fields																				Op				

The Op1 and Op fields define instruction classes

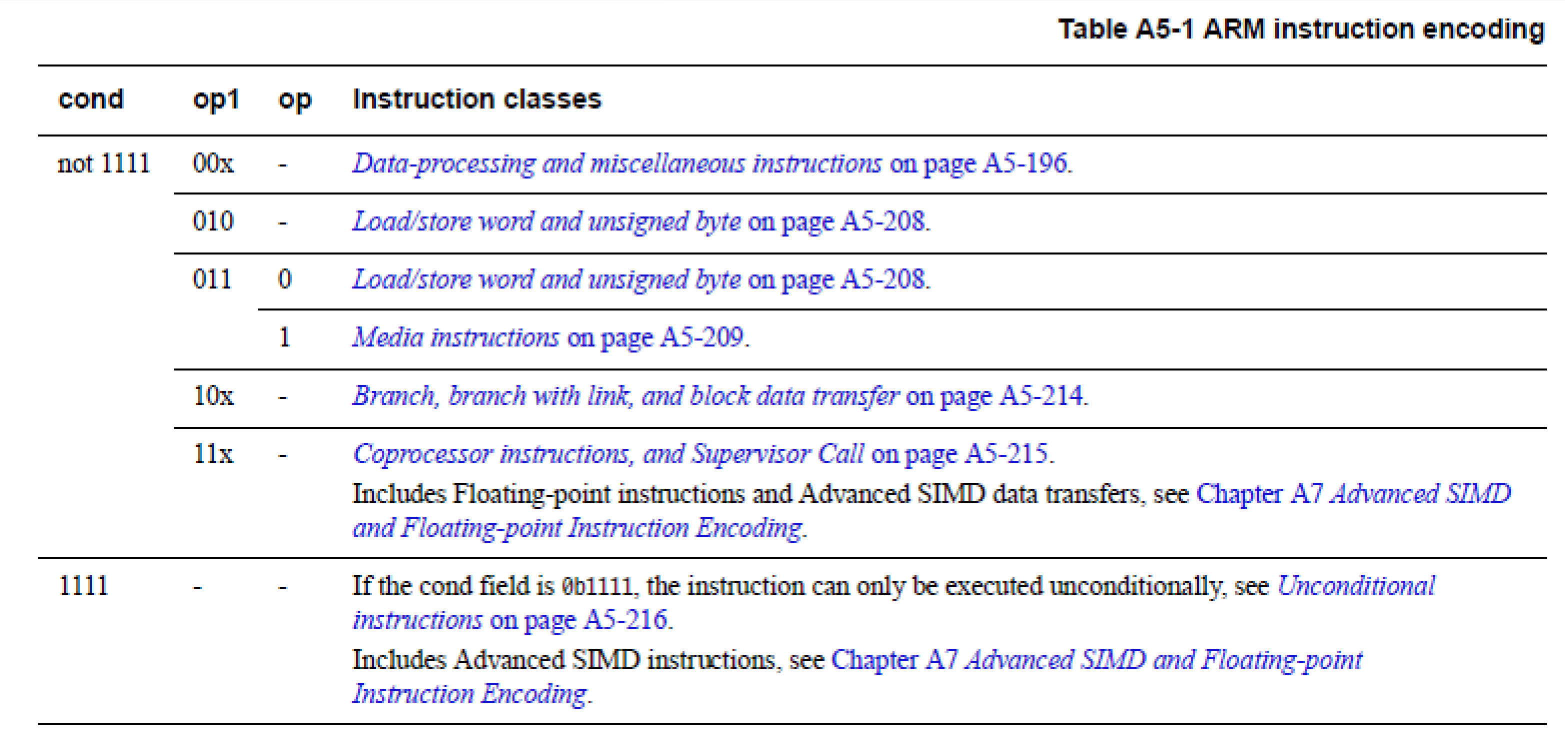
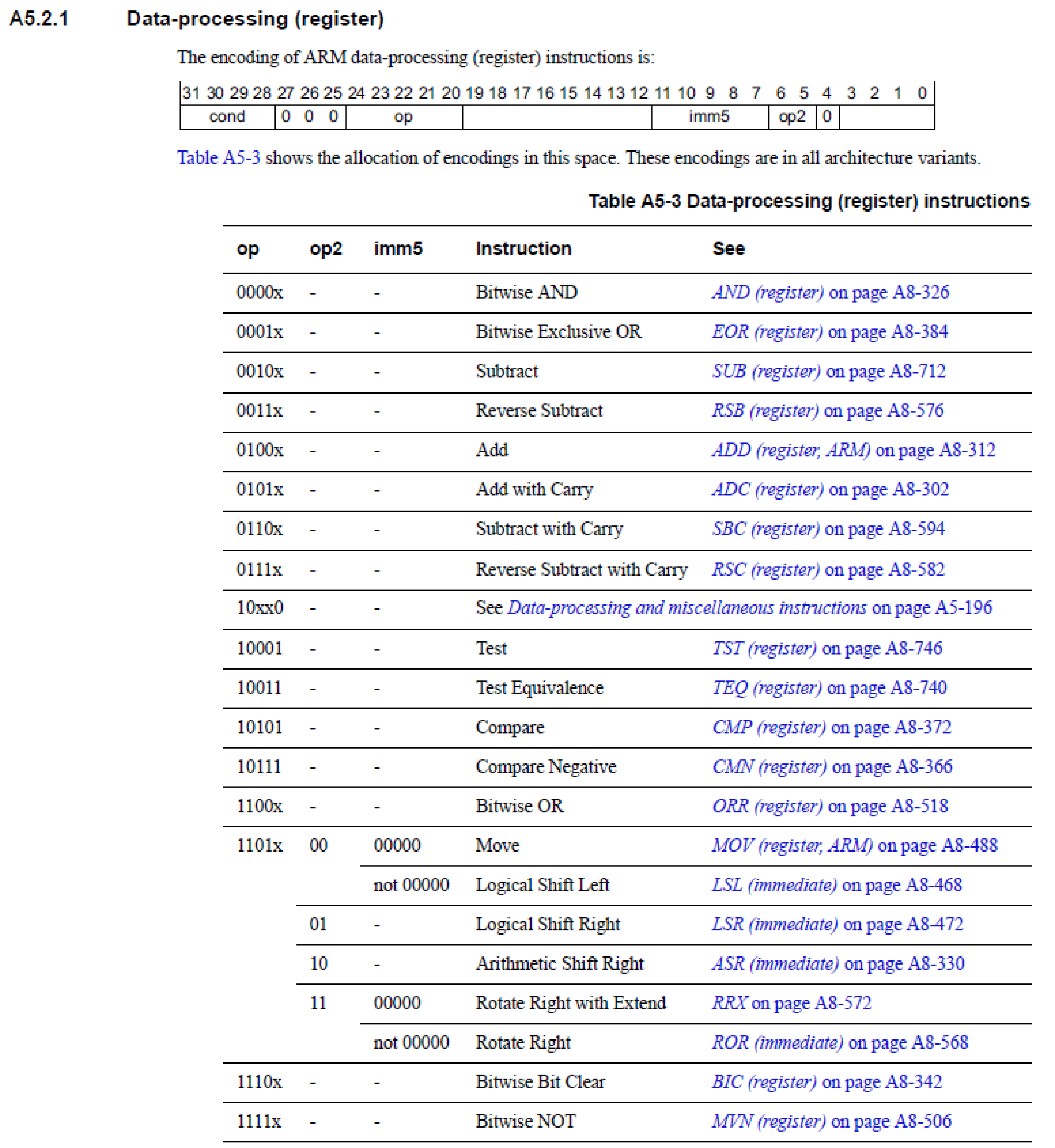


Table A5-1 ARM instruction encoding

cond	op1	op	Instruction classes
not 1111	00x	-	*Data-processing and miscellaneous instructions* on page A5-196.
	010	-	*Load/store word and unsigned byte* on page A5-208.
	011	0	*Load/store word and unsigned byte* on page A5-208.
		1	*Media instructions* on page A5-209.
	10x	-	*Branch, branch with link, and block data transfer* on page A5-214.
	11x	-	*Coprocessor instructions, and Supervisor Call* on page A5-215. Includes Floating-point instructions and Advanced SIMD data transfers, see Chapter A7 *Advanced SIMD and Floating-point Instruction Encoding*.
1111	-	-	If the cond field is `0b1111`, the instruction can only be executed unconditionally, see *Unconditional instructions* on page A5-216. Includes Advanced SIMD instructions, see Chapter A7 *Advanced SIMD and Floating-point Instruction Encoding*.

Data processing instructions: Bits 27-25: 000



A5.2.1 Data-processing (register)

The encoding of ARM data-processing (register) instructions is:

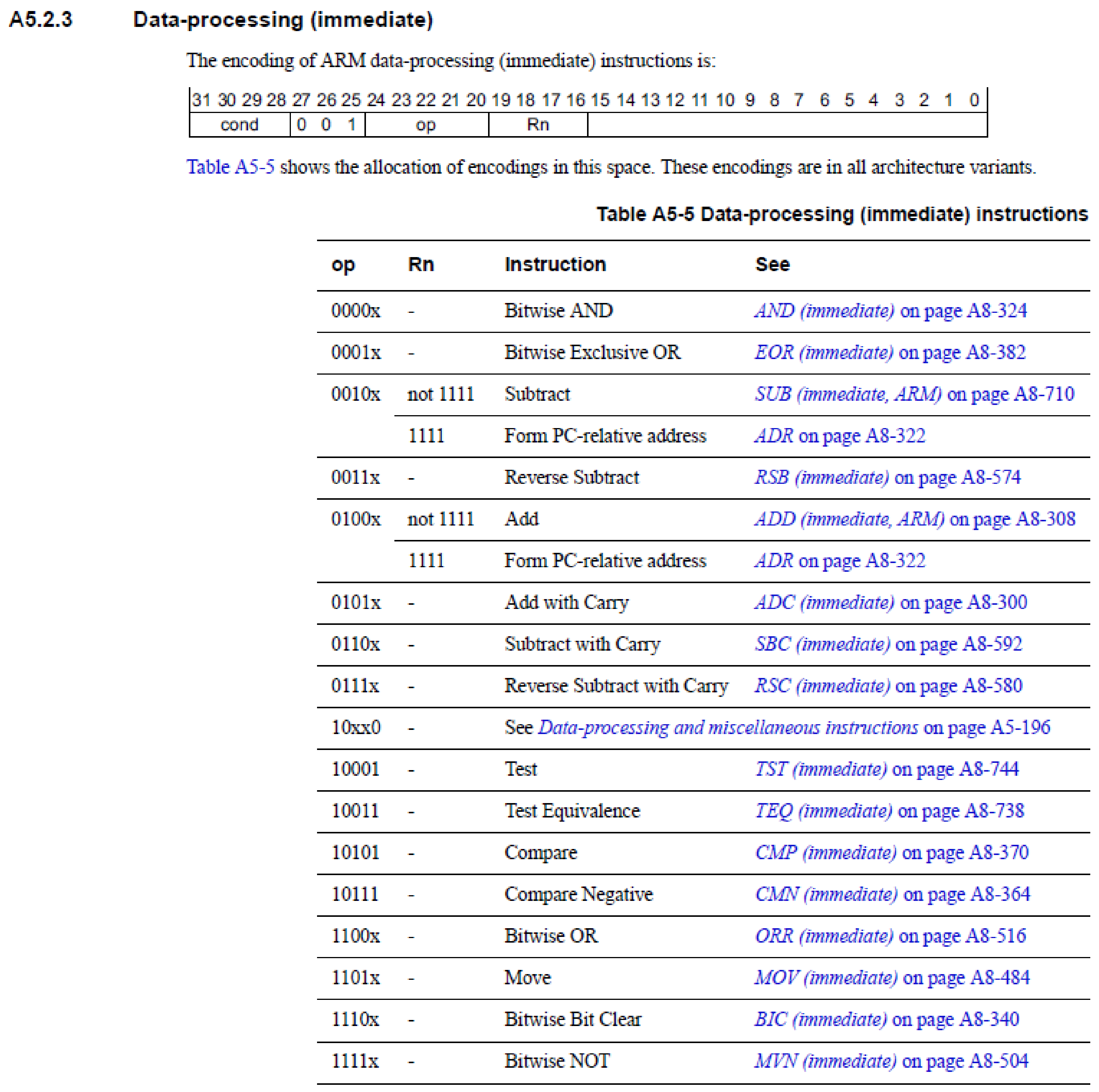
31 30 29 28	27	26	25	24 23 22 21 20	19 18 17 16 15 14 13 12	11 10 9 8 7	6 5	4	3 2 1 0
cond	0	0	0	op		imm5	op2	0	

Table A5-3 shows the allocation of encodings in this space. These encodings are in all architecture variants.

Table A5-3 Data-processing (register) instructions

op	op2	imm5	Instruction	See
0000x	-	-	Bitwise AND	*AND (register)* on page A8-326
0001x	-	-	Bitwise Exclusive OR	*EOR (register)* on page A8-384
0010x	-	-	Subtract	*SUB (register)* on page A8-712
0011x	-	-	Reverse Subtract	*RSB (register)* on page A8-576
0100x	-	-	Add	*ADD (register, ARM)* on page A8-312
0101x	-	-	Add with Carry	*ADC (register)* on page A8-302
0110x	-	-	Subtract with Carry	*SBC (register)* on page A8-594
0111x	-	-	Reverse Subtract with Carry	*RSC (register)* on page A8-582
10xx0	-	-	See *Data-processing and miscellaneous instructions* on page A5-196	
10001	-	-	Test	*TST (register)* on page A8-746
10011	-	-	Test Equivalence	*TEQ (register)* on page A8-740
10101	-	-	Compare	*CMP (register)* on page A8-372
10111	-	-	Compare Negative	*CMN (register)* on page A8-366
1100x	-	-	Bitwise OR	*ORR (register)* on page A8-518
1101x	00	00000	Move	*MOV (register, ARM)* on page A8-488
		not 00000	Logical Shift Left	*LSL (immediate)* on page A8-468
	01	-	Logical Shift Right	*LSR (immediate)* on page A8-472
	10	-	Arithmetic Shift Right	*ASR (immediate)* on page A8-330
	11	00000	Rotate Right with Extend	*RRX* on page A8-572
		not 00000	Rotate Right	*ROR (immediate)* on page A8-568
1110x	-	-	Bitwise Bit Clear	*BIC (register)* on page A8-342
1111x	-	-	Bitwise NOT	*MVN (register)* on page A8-506

Data processing instructions: Bits 27-25: 001



A5.2.3 Data-processing (immediate)

The encoding of ARM data-processing (immediate) instructions is:

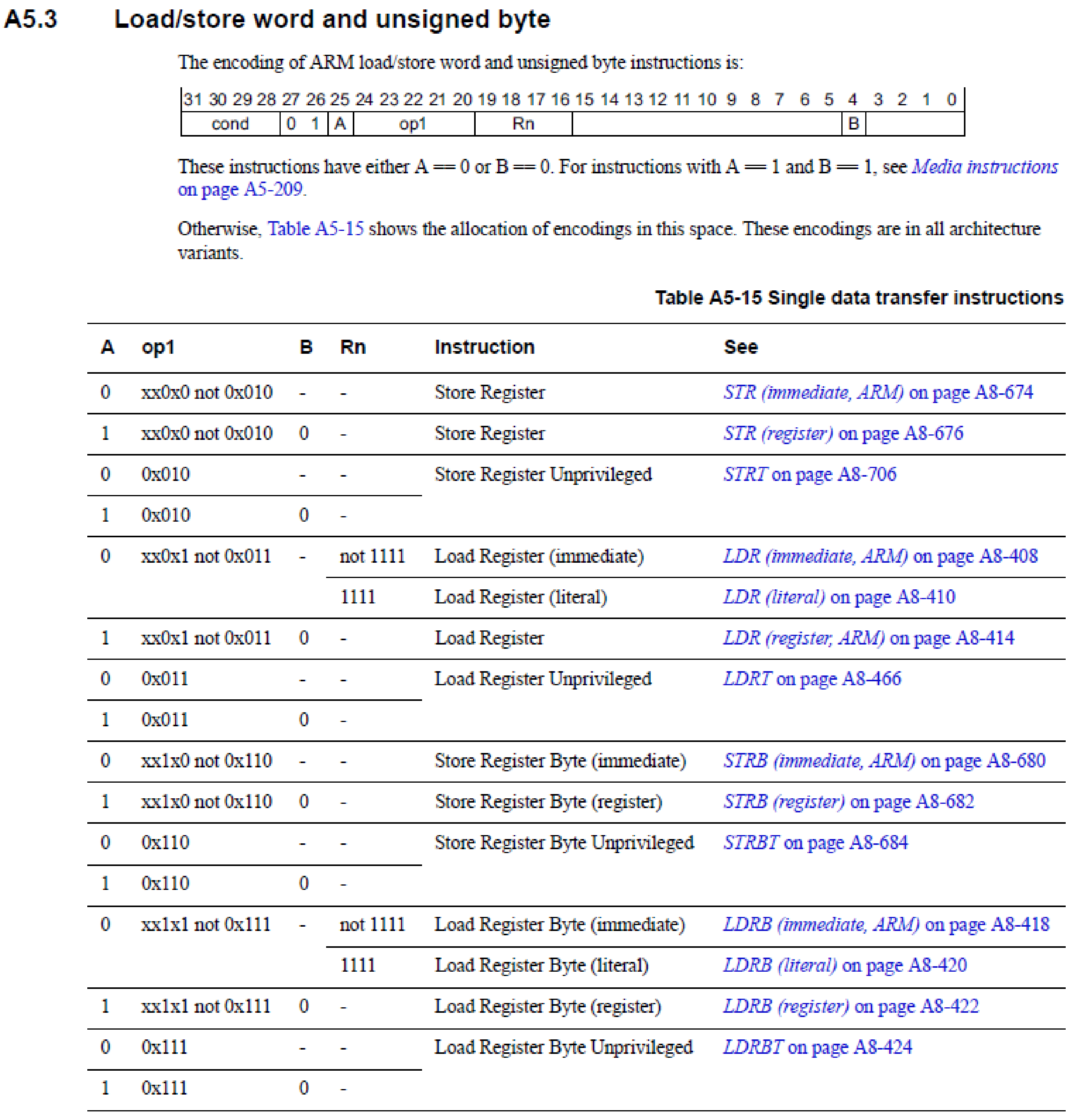
31 30 29 28	27 26 25	24 23 22 21 20	19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond	0 0 1	op	Rn	

Table A5-5 shows the allocation of encodings in this space. These encodings are in all architecture variants.

Table A5-5 Data-processing (immediate) instructions

op	Rn	Instruction	See
0000x	-	Bitwise AND	*AND (immediate)* on page A8-324
0001x	-	Bitwise Exclusive OR	*EOR (immediate)* on page A8-382
0010x	not 1111	Subtract	*SUB (immediate, ARM)* on page A8-710
	1111	Form PC-relative address	*ADR* on page A8-322
0011x	-	Reverse Subtract	*RSB (immediate)* on page A8-574
0100x	not 1111	Add	*ADD (immediate, ARM)* on page A8-308
	1111	Form PC-relative address	*ADR* on page A8-322
0101x	-	Add with Carry	*ADC (immediate)* on page A8-300
0110x	-	Subtract with Carry	*SBC (immediate)* on page A8-592
0111x	-	Reverse Subtract with Carry	*RSC (immediate)* on page A8-580
10xx0	-	See *Data-processing and miscellaneous instructions* on page A5-196	
10001	-	Test	*TST (immediate)* on page A8-744
10011	-	Test Equivalence	*TEQ (immediate)* on page A8-738
10101	-	Compare	*CMP (immediate)* on page A8-370
10111	-	Compare Negative	*CMN (immediate)* on page A8-364
1100x	-	Bitwise OR	*ORR (immediate)* on page A8-516
1101x	-	Move	*MOV (immediate)* on page A8-484
1110x	-	Bitwise Bit Clear	*BIC (immediate)* on page A8-340
1111x	-	Bitwise NOT	*MVN (immediate)* on page A8-504

Load/Store instructions = Bits 27-25: 010 and 011



A5.3 Load/store word and unsigned byte

The encoding of ARM load/store word and unsigned byte instructions is:

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	1	A	op1	Rn		B	

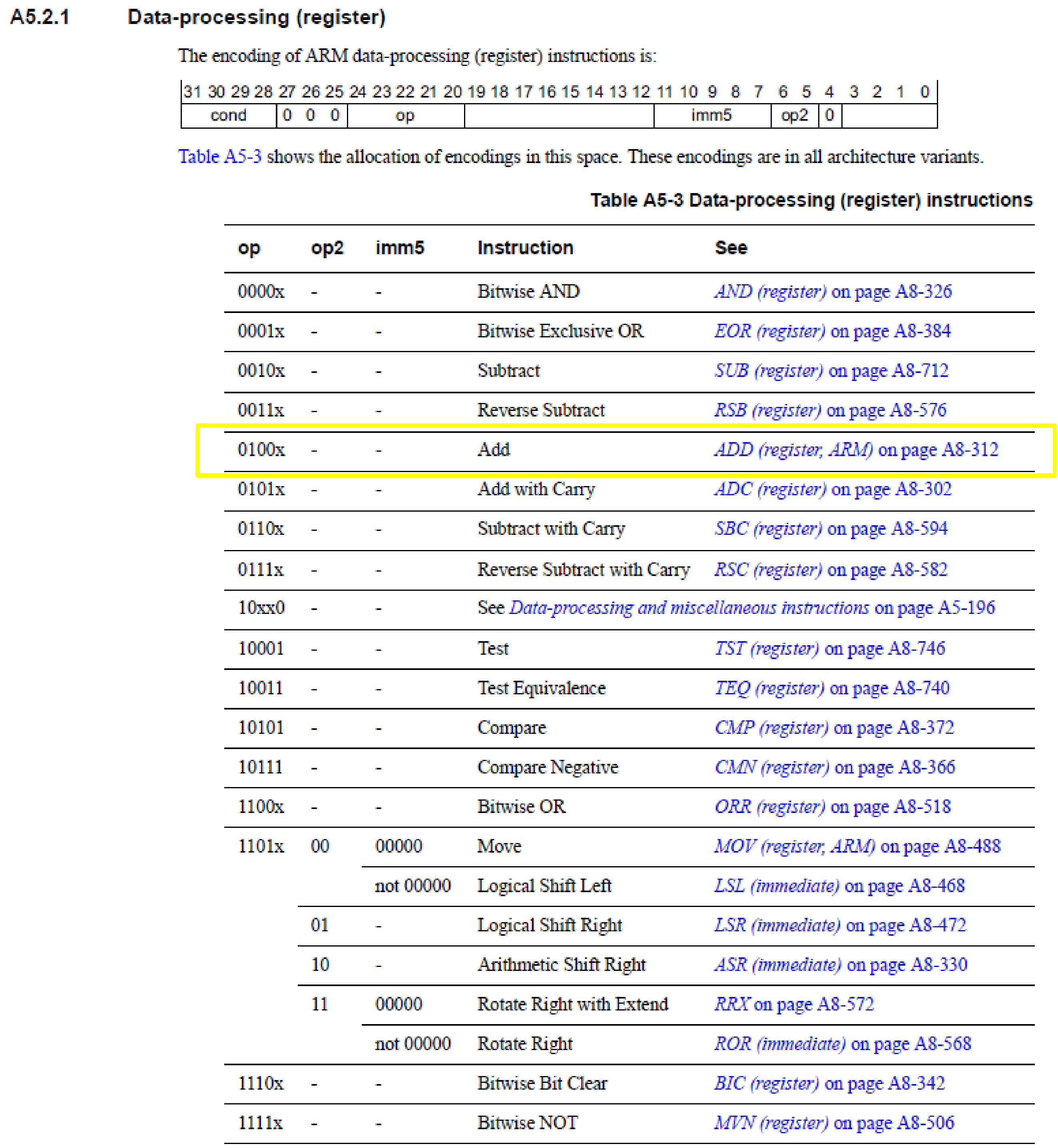
These instructions have either A == 0 or B == 0. For instructions with A == 1 and B == 1, see *Media instructions* on page A5-209.

Otherwise, Table A5-15 shows the allocation of encodings in this space. These encodings are in all architecture variants.

Table A5-15 Single data transfer instructions

A	op1	B	Rn	Instruction	See
0	xx0x0 not 0x010	-	-	Store Register	*STR (immediate, ARM)* on page A8-674
1	xx0x0 not 0x010	0	-	Store Register	*STR (register)* on page A8-676
0	0x010	-	-	Store Register Unprivileged	*STRT* on page A8-706
1	0x010	0	-		
0	xx0x1 not 0x011	-	not 1111	Load Register (immediate)	*LDR (immediate, ARM)* on page A8-408
			1111	Load Register (literal)	*LDR (literal)* on page A8-410
1	xx0x1 not 0x011	0	-	Load Register	*LDR (register, ARM)* on page A8-414
0	0x011	-	-	Load Register Unprivileged	*LDRT* on page A8-466
1	0x011	0	-		
0	xx1x0 not 0x110	-	-	Store Register Byte (immediate)	*STRB (immediate, ARM)* on page A8-680
1	xx1x0 not 0x110	0	-	Store Register Byte (register)	*STRB (register)* on page A8-682
0	0x110	-	-	Store Register Byte Unprivileged	*STRBT* on page A8-684
1	0x110	0	-		
0	xx1x1 not 0x111	-	not 1111	Load Register Byte (immediate)	*LDRB (immediate, ARM)* on page A8-418
			1111	Load Register Byte (literal)	*LDRB (literal)* on page A8-420
1	xx1x1 not 0x111	0	-	Load Register Byte (register)	*LDRB (register)* on page A8-422
0	0x111	-	-	Load Register Byte Unprivileged	*LDRBT* on page A8-424
1	0x111	0	-		

Data processing instructions: Bits 27-25: 000



A5.2.1 Data-processing (register)

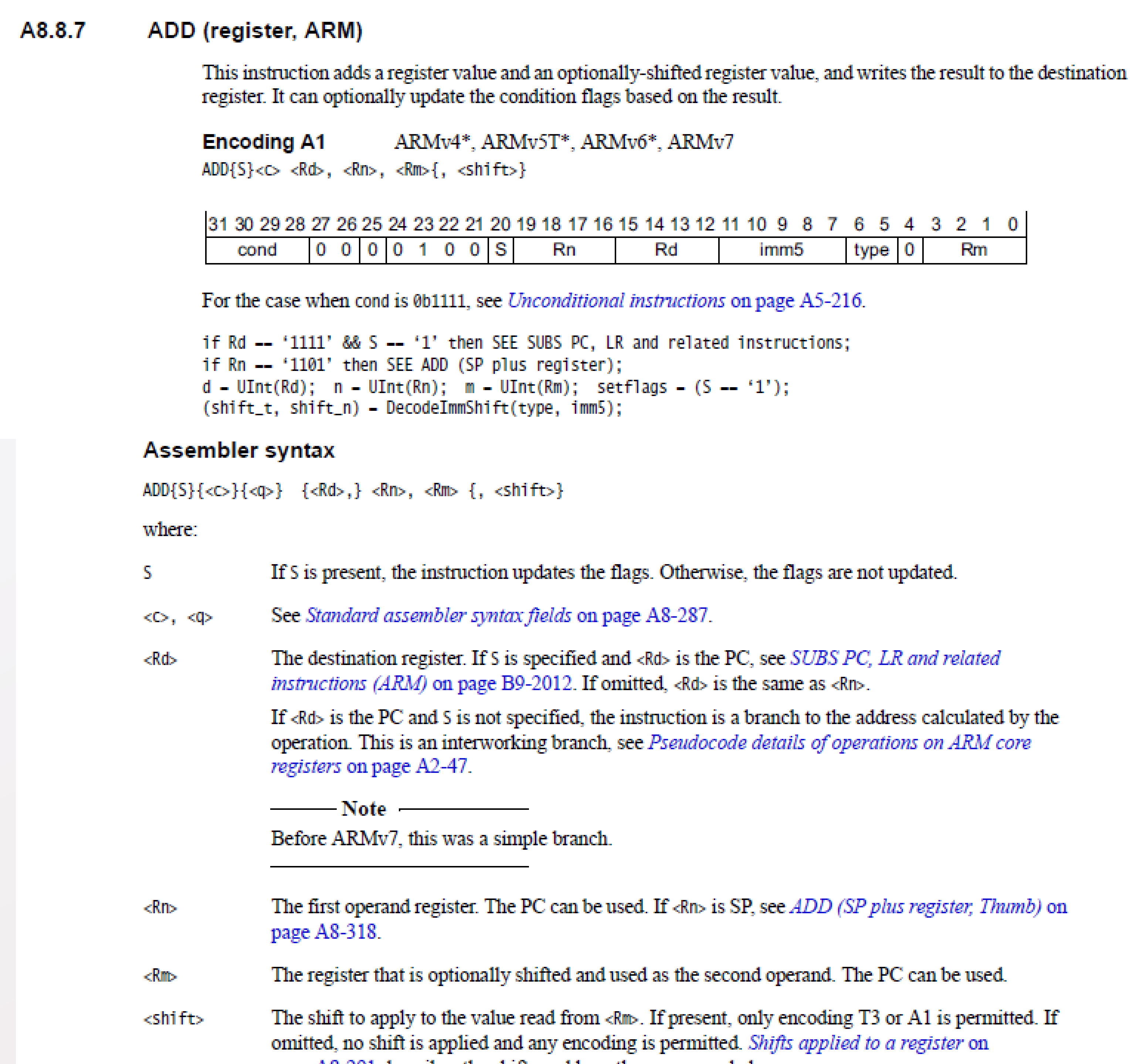
The encoding of ARM data-processing (register) instructions is:

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	0	op		imm5	op2	0	

Table A5-3 shows the allocation of encodings in this space. These encodings are in all architecture variants.

Table A5-3 Data-processing (register) instructions

op	op2	imm5	Instruction	See
0000x	-	-	Bitwise AND	*AND (register)* on page A8-326
0001x	-	-	Bitwise Exclusive OR	*EOR (register)* on page A8-384
0010x	-	-	Subtract	*SUB (register)* on page A8-712
0011x	-	-	Reverse Subtract	*RSB (register)* on page A8-576
0100x	-	-	Add	*ADD (register, ARM)* on page A8-312
0101x	-	-	Add with Carry	*ADC (register)* on page A8-302
0110x	-	-	Subtract with Carry	*SBC (register)* on page A8-594
0111x	-	-	Reverse Subtract with Carry	*RSC (register)* on page A8-582
10xx0	-	-	See *Data-processing and miscellaneous instructions* on page A5-196	
10001	-	-	Test	*TST (register)* on page A8-746
10011	-	-	Test Equivalence	*TEQ (register)* on page A8-740
10101	-	-	Compare	*CMP (register)* on page A8-372
10111	-	-	Compare Negative	*CMN (register)* on page A8-366
1100x	-	-	Bitwise OR	*ORR (register)* on page A8-518
1101x	00	00000	Move	*MOV (register, ARM)* on page A8-488
		not 00000	Logical Shift Left	*LSL (immediate)* on page A8-468
	01	-	Logical Shift Right	*LSR (immediate)* on page A8-472
	10	-	Arithmetic Shift Right	*ASR (immediate)* on page A8-330
	11	00000	Rotate Right with Extend	*RRX* on page A8-572
		not 00000	Rotate Right	*ROR (immediate)* on page A8-568
1110x	-	-	Bitwise Bit Clear	*BIC (register)* on page A8-342
1111x	-	-	Bitwise NOT	*MVN (register)* on page A8-506



A8.8.7 ADD (register, ARM)

This instruction adds a register value and an optionally-shifted register value, and writes the result to the destination register. It can optionally update the condition flags based on the result.

Encoding A1 ARMv4*, ARMv5T*, ARMv6*, ARMv7

`ADD{S}<c> <Rd>, <Rn>, <Rm>{, <shift>}`

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	0	0	1	0	0	S	Rn				Rd				imm5					type		0	Rm			

For the case when cond is 0b1111, see *Unconditional instructions* on page A5-216.

```
if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
if Rn == '1101' then SEE ADD (SP plus register);
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  setflags = (S == '1');
(shift_t, shift_n) = DecodeImmShift(type, imm5);
```

Assembler syntax

`ADD{S}{<c>}{<q>}  {<Rd>,} <Rn>, <Rm> {, <shift>}`

where:

S If S is present, the instruction updates the flags. Otherwise, the flags are not updated.

<c>, <q> See *Standard assembler syntax fields* on page A8-287.

<Rd> The destination register. If S is specified and <Rd> is the PC, see *SUBS PC, LR and related instructions (ARM)* on page B9-2012. If omitted, <Rd> is the same as <Rn>.

If <Rd> is the PC and S is not specified, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see *Pseudocode details of operations on ARM core registers* on page A2-47.

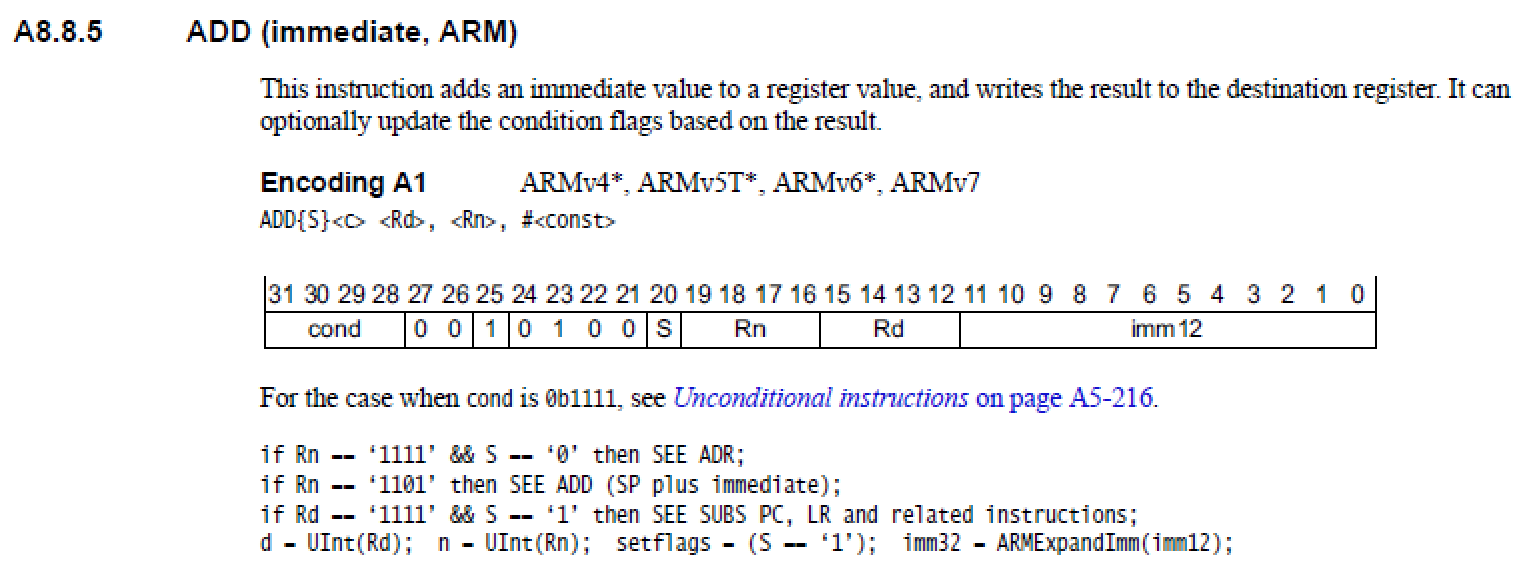
Note

Before ARMv7, this was a simple branch.

<Rn> The first operand register. The PC can be used. If <Rn> is SP, see *ADD (SP plus register, Thumb)* on page A8-318.

<Rm> The register that is optionally shifted and used as the second operand. The PC can be used.

<shift> The shift to apply to the value read from <Rm>. If present, only encoding T3 or A1 is permitted. If omitted, no shift is applied and any encoding is permitted. *Shifts applied to a register* on page A8-291 describes the shifts and how they are encoded.



A8.8.5 ADD (immediate, ARM)

This instruction adds an immediate value to a register value, and writes the result to the destination register. It can optionally update the condition flags based on the result.

Encoding A1 ARMv4*, ARMv5T*, ARMv6*, ARMv7

```
ADD{S}<c> <Rd>, <Rn>, #<const>
```

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	1	0	1	0	0	S	Rn				Rd				imm12											

For the case when cond is 0b1111, see *Unconditional instructions* on page A5-216.

```
if Rn == '1111' && S == '0' then SEE ADR;
if Rn == '1101' then SEE ADD (SP plus immediate);
if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
d = UInt(Rd);  n = UInt(Rn);  setflags = (S == '1');  imm32 = ARMExpandImm(imm12);
```