see B2 for how to synchronize CP15 changes with hardware state. Often it's more involved than it may appear.

Chapter B3
The System Control Coprocessor

This chapter describes coprocessor 15, the System Control coprocessor. It contains the following sections:

B3.1 About the System Control coprocessor

All of the standard memory and system facilities are controlled by coprocessor 15 (CP15), which is known as the System Control coprocessor. Some facilities also use other methods of control, and these are described in the chapters relating to those facilities. For example, the Memory Management Unit described in Chapter B4 *Virtual Memory System Architecture* is also controlled by page tables in memory.

ARMv6 systems shall include a System Control Coprocessor, with support for automatic interrogation of cache, tightly coupled memory, and coprocessor provision. It also provides the control mechanism for memory management (MMU and MPU support as applicable).

Prior to ARMv6, CP15 instructions are UNDEFINED when CP15 is not implemented. However, CP15 has become a *de facto* standard for processor ID, cache control, and memory management (MMU and MPU support) in implementations since ARMv4. This manual should be read in conjunction with the relevant implementation reference manual to determine the exact details of CP15 support in a particular part.

This chapter describes the overall design of the System Control coprocessor and how its registers are accessed. Detailed information is given about some of its registers. Other registers are allocated to facilities described in detail in other chapters and are only summarized in this chapter.

B3.2 Registers

The System Control coprocessor can contain up to 16 primary registers, each of which is 32 bits long. Additional fields in the register access instructions are used to further refine the access, increasing the number of physical 32-bit registers in CP15. The 4-bit primary register number is used to identify registers in descriptions of the System Control coprocessor, because it is the primary factor determining the function of the register.

CP15 registers can be read-only, write-only or read/write. The detailed descriptions of the registers specify:

- the types of access that are allowed
- the functionality invoked by each type of access
- whether a primary register identifies more than one physical register, and if so, how they are distinguished
- any other details that are relevant to the use of the register.

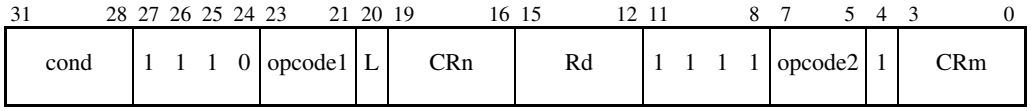
B3.2.1 Register access instructions

The only defined System Control coprocessor instructions are:

- `MCR` instructions to write an ARM® register to a CP15 register
- `MRC` instructions to read the value of a CP15 register into an ARM register
- `MCRR` instructions for range operations introduced in ARMv6, and optional in earlier versions of the architecture.
- `MRRC` optional for IMPLEMENTATION DEFINED features.

All CP15 `CDP`, `CDP2`, `LDC`, `LDC2`, `MCR2`, `MCRR2`, `MRC2`, `MRRC2` , `STC`, and `STC2` instructions are UNDEFINED.

The format of the `MCR`/`MRC` instructions is illustrated below, with bits[11:8](*cp_num*) indicating CP15, and the CRn field indicating the primary register number, with CRm and opcode2 providing additional register decode.



The `MCR` and `MRC` instructions to access the CP15 registers use the generic syntax for those instructions:

```
MCR{<cond>}  p15, 0, <Rd>, <CRn>, <CRm>{, <opcode2>}     (L = 0)
MRC{<cond>}  p15, 0, <Rd>, <CRn>, <CRm>{, <opcode2>}     (L = 1)
```

where:

`<cond>` This is the condition under which the instruction is executed. The conditions are defined in *The condition field* on page A3-3. If `<cond>` is omitted, the `AL` (always) condition is used.

Bits[23:21] These bits of the instruction, which are the <opcode1> field in generic MRC and MCR instructions, are generally 0b000 in valid CP15 instructions. However, <opcode1> == 1 is being used for level 2 cache support and considered for some other specialist tasks. Unassigned values are UNPREDICTABLE.

<Rd> This is the ARM register involved in the transfer (the source register for MCR and the destination register for MRC). This register must not be R15, even though MRC instructions normally allow it to be R15. If R15 is specified for <Rd> in a CP15 MRC or MCR instruction, the instruction is UNPREDICTABLE.

<CRn> This is the primary CP15 register involved in the transfer (the destination register for MCR and the source register for MRC). The standard generic coprocessor register names are c0, c1, ..., c15.

<CRm> This is an additional coprocessor register name which is used for accesses to some primary registers to specify additional information about the version of the register and/or the type of access.

When the description of a primary register does not specify <CRm>, c0 must be specified. If another register is specified, the instruction is UNPREDICTABLE.

<opcode2> This is an optional 3-bit number which is used for accesses to some primary registers to specify additional information about the version of the register and/or the type of access. If it is omitted, 0 is used.

When the description of a primary register does not specify <opcode2>, it must be omitted or 0 must be specified. If another value is specified, the instruction is UNPREDICTABLE.

The MCRR format (see *MCRR* on page A4-64) has less scope for decode. The primary register is implied (no CRn field), and the CRm and opcode fields are used to decode the correct function.

Prior to ARMv6, MCR and MRC instructions can only be used when the processor is in a privileged mode. If they are executed when the processor is in User mode, an Undefined Instruction exception occurs.

ARMv6 introduced user access of the following commands:

- Prefetch flush
- Data synchronization barrier
- Data memory barrier
- Clean and prefetch range operations.

Note

If access to privileged System Control coprocessor functionality by User mode programs is required, the usual solution is that the operating system defines one or more SWIs to supply it. As the precise set of memory and system facilities available on different processors can vary considerably, it is recommended that all such SWIs are implemented in an easily replaceable module and that the SWI interface of this module is defined to be as independent of processor details as possible.

B3.2.2 Primary register allocation

Table B3-1 shows the allocation of the primary registers of the System Control coprocessor.

Table B3-1 Primary register allocation

Reg	Generic use	Specific uses	Details in
0	ID codes (read-only)	Processor ID, Cache, Tightly-coupled Memory and TLB type	*Register 0: ID codes* on page B3-7
1	Control bits (read/write)	System Configuration Bits	*Control register* on page B3-12, and *Register 1: Control register* on page B4-40
2	Memory protection and control	Page Table Control	*Register 2: Translation table base* on page B4-41
3	Memory protection and control	Domain Access Control	*Register 3: Domain access control* on page B4-42
4	Memory protection and control	Reserved	None. This is a reserved register.
5	Memory protection and control	Fault status	*Fault Address and Fault Status registers* on page B4-19, and *Register 5: Fault status* on page B4-43
6	Memory protection and control	Fault address	*Fault Address and Fault Status registers* on page B4-19, and *Register 6: Fault Address register* on page B4-44
7	Cache and write buffer	Cache/write buffer control	*Register 7: cache management functions* on page B6-19
8	Memory protection and control	TLB control	*Register 8: TLB functions* on page B4-45
9	Cache and write buffer	Cache lockdown	*Register 9: cache lockdown functions* on page B6-31
10	Memory protection and control	TLB lockdown	*Register 10: TLB lockdown* on page B4-47
11	Tightly-coupled Memory Control	DMA Control	*L1 DMA control using CP15 Register 11* on page B7-9
12	Reserved	Reserved	None. This is a reserved register.

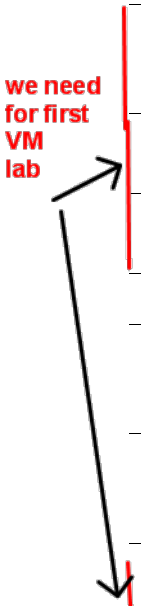


Table B3-1 Primary register allocation

Reg	Generic use	Specific uses	Details in
13	Process ID	Process ID	*Register 13: Process ID* on page B4-52, and *Register 13: FCSE PID* on page B8-7
14	Reserved	-	-
15	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	Implementation documents

B3.3 Register 0: ID codes

CP15 register 0 contains one or more identification codes for the ARM and system implementation. When this register is read, the opcode2 field of the MRC instruction selects which identification code is wanted, as shown in Table B3-2, and the CRm field must be specified as c0 (if it is not, the instruction is UNPREDICTABLE). Writing to CP15 register 0 is UNPREDICTABLE.

Table B3-2 System Control coprocessor ID registers

opcode2	Register	Details in
0b000	Main ID register	*Main ID register*
0b001	Cache type register	*Cache type register* on page B3-10
0b010	*Tightly Coupled Memory* (TCM) type register	*TCM type register* on page B3-10
0b011	TLB type register	
0b100	MPU type register (PMSAv6)	
other	Reserved (see main text)	-

If an <opcode2> value corresponding to an unimplemented or reserved ID register is encountered, the System Control coprocessor returns the value of the main ID register.

ID registers other than the main ID register are defined so that when implemented, their value cannot be equal to that of the main ID register. Software can therefore determine whether they exist by reading both the main ID register and the desired register and comparing their values. If the two values are not equal, the desired register exists.

B3.3.1 Main ID register

When CP15 register 0 is read with <opcode2> == 0, an identification code is returned from which, among other things, the ARM architecture version number can be determined, as well as whether or not the Thumb® instruction set has been implemented.

Note

Only some of the fields in CP15 register 0 are architecturally defined. The rest are IMPLEMENTATION DEFINED and provide more detailed information about the exact processor variant. Consult individual datasheets for the precise identification codes used for each processor.

Implementor code

Bits[31:24] of the main ID register contain an implementor code.

The following codes are defined (all other values of the architecture code are reserved by ARM Limited.):

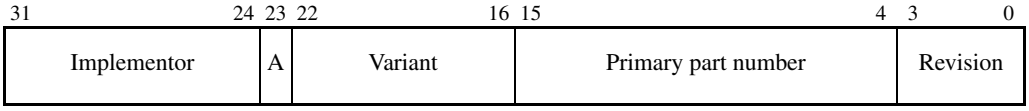
`0x41`	A (ARM Limited)
`0x44`	D (Digital Equipment Corporation)
`0x4D`	M (Motorola - Freescale Semiconductor Inc.)
`0x56`	V (Marvell Semiconductor Inc.)
`0x69`	i (Intel Corporation)

ARM processor implementation IDs

For historical reasons, there are a variety of ways in which the CP15 register 0 ID code might need to be interpreted. If bit[19] is zero, bits[15:12] should be interpreted as follows:

- if they are `0x0`, this indicates an OBSOLETE part (pre-ARMv4 architecture)
- if they are `0x7`, this indicates that the processor is in the ARM7 family
- if > `0x7`, a more recent processor family than ARM7 is involved.

ARM7 processor IDs are interpreted as follows:



Bits[3:0] Contain the IMPLEMENTATION DEFINED revision number for the processor.

Bits[15:4] Contain the IMPLEMENTATION DEFINED representation of the primary part number for the processor. The top four bits of this number are `0x7`.

Bits[22:16] Contain an IMPLEMENTATION DEFINED variant number.

Bit[23] Indicates which of the two possible architectures for an ARM7-based process is involved:

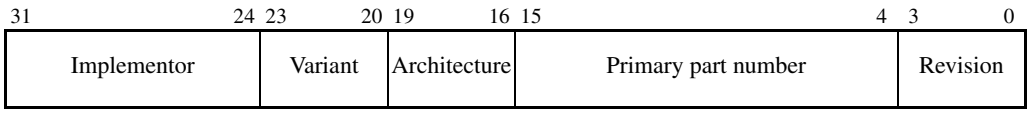
`0`	Architecture 3 (OBSOLETE part)
1	Architecture 4T.

Bits[31:24] `0x41` = A (ARM Limited) implementation code.

Processor implementations since ARM7 have a general format of bits[23:0] which are common across implementations from ARM and architecture licensees. Two general formats are defined, dependent on the value of bit[19]. They are described in the following sections.

Post-ARM7 processors

If bits[15:12] of the ID code are neither `0x0` nor `0x7`, the ID code is interpreted as follows:



Bits[3:0] Contain the IMPLEMENTATION DEFINED revision number for the processor.

Bits[15:4] Contain an IMPLEMENTATION DEFINED representation of the primary part number for the processor. The top four bits of this number are not allowed to be `0x0` or `0x7`.

Bits[19:16] Contain an architecture code. The following architecture codes are defined:

- `0x1` ARM architecture v4
- `0x2` ARM architecture v4T
- `0x3` ARM architecture v5
- `0x4` ARM architecture v5T
- `0x5` ARM architecture v5TE
- `0x6` ARM architecture v5TEJ
- `0x7` ARM architecture v6
- `0xF` Revised CPUID format. Details available from ARM.

All other values of the architecture code are reserved by ARM Limited

Bits[23:20] Contain an IMPLEMENTATION DEFINED variant number. This is typically used to distinguish two variants of the same primary part, for example, two different cache size variants.

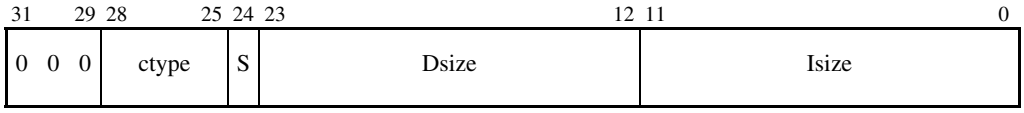
Bits[31:24] Contain an implementor code. See *Implementor code* on page B3-8.

B3.3.2 Cache type register

The Cache type register supplies the following details about the cache:

- whether it is a unified cache or separate instruction and data caches
- its size, line length and associativity
- whether it is a write-through cache or a write-back cache
- cache cleaning and lockdown capabilities.

The format of the Cache type register is:



ctype Specifies details of the cache not specified by the S bit and the Dsize and Isize fields. All values not specified in the table are reserved for future expansion.

S bit Specifies whether the cache is a unified cache (S == 0), or separate instruction and data caches (S == 1). If S == 0, the Isize and Dsize fields both describe the unified cache, and must be identical.

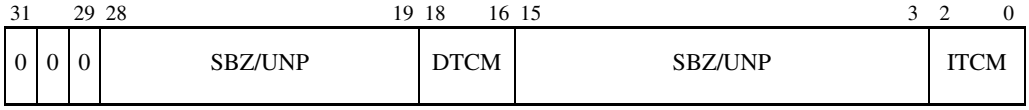
Dsize Specifies the size, line length and associativity of the data cache, or of the unified cache if S == 0.

Isize Specifies the size, line length and associativity of the instruction cache, or of the unified cache if S == 0.

A detailed discussion on caches is provided in Chapter B6 *Caches and Write Buffers*. See *Cache Type register* on page B6-14 for the encoding of the cache type register fields.

B3.3.3 TCM type register

The format of the Tightly-Coupled Memory (TCM) type register is:



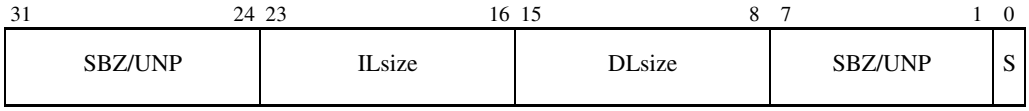
ITCM (Bits[2:0]) Indicate the number of Instruction (or Unified) Tightly-Coupled Memories implemented. This value lies in the range 0-4, all other values are reserved. All Instruction TCMs must be accessible to both instruction and data sides.

DTCM (Bits[18:16]) Indicate the number of Data Tightly-Coupled Memories implemented. This value lies in the range 0-4, all other values are reserved.

A detailed discussion of tightly coupled memory is provided in chapter Chapter B7 *Tightly Coupled Memory*.

B3.3.4 TLB type register

The format of the TLB type register is:



S-bit Specifies whether the TLB is a unified TLB (S == 0), or separate instruction and data TLBs (S == 1).

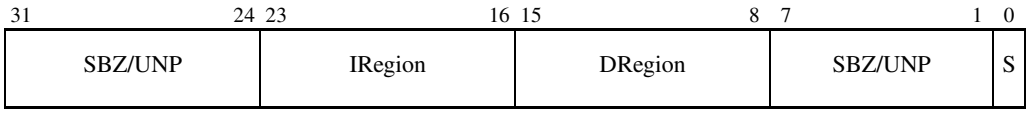
DLsize Specifies the number of lockable entries in the data TLB if S ==1, or the unified TLB if S == 0.

ILsize Specifies the number of lockable entries in the instruction TLB, if S == 1, otherwise SBZ.

A detailed description of the virtual memory system architecture is provided in Chapter B4 *Virtual Memory System Architecture*.

B3.3.5 MPU type register

The format of the Memory Protection Unit (MPU) type register is:



S-bit Specifies whether the MPU is a unified MPU (S == 0), or separate instruction and data MPUs (S == 1).

DRegion Specifies the number of protected regions in the data MPU if S ==1, or the unified MPU if S == 0.

IRegion Specifies the number of protected regions in the instruction MPU, if S == 1, otherwise SBZ.

A detailed description of the protected memory system architecture is provided in Chapter B5 *Protected Memory System Architecture*.

Note

The MPU type register is introduced with PMSAv6.

B3.4 Register 1: Control registers

CP15 register 1 contains configuration control bits for the ARM processor. It contains 3 registers selected by the opcode_2 field. When opcode_2 is 0 the architecturally specified control register is selected. When opcode_2 is 1 an IMPLEMENTATION DEFINED control register is selected.

Table B3-3 System Control coprocessor Control registers

opcode2	Register
0b000	Control register
0b001	Auxiliary control register (format IMPLEMENTATION DEFINED)
0b010	Coprocessor access control register
other	RESERVED

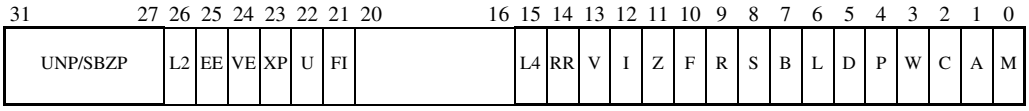
B3.4.1 Control register

This register contains:

- Enable/disable bits for the caches, MMUs, and other memory system blocks that are primarily controlled by other CP15 registers. This allows these memory system blocks to be programmed correctly before they are enabled.
- Various configuration bits for memory system blocks and for the ARM processor itself.

Note

Extra bits of both varieties might be added in the future. Because of this, this register should normally be updated using read/modify/write techniques, to ensure that currently unallocated bits are not needlessly modified. Failure to observe this rule might result in code which has unexpected side effects on future processors.



When a control bit in CP15 register 1 is not applicable to a particular implementation, it reads as the value that most closely reflects that implementation, and ignores writes. (Specific examples of this general rule are documented in the individual bit descriptions below.) Apart from bits that read as 1 according to this rule, all bits in CP15 register 1 are set to 0 on reset.

M (bit[0]) This is the enable/disable bit for the MMU or Protection Unit:

0 = MMU or Protection Unit disabled

1 = MMU or Protection Unit enabled.

On systems without an MMU, this bit reads as 0 and ignores writes.

A (bit[1]) In ARM architecture v6, this controls strict alignment:

0 = Alignment not strict

1 = Strict alignment. If a data access is not aligned to the width of the accessed data item, a Data Abort exception is generated.

In architectures before v6, for memory systems which optionally allow the alignment of data memory accesses to be checked, this bit enables and disables alignment fault checking:

0 = Alignment fault checking disabled

1 = Alignment fault checking enabled.

For other memory systems, this bit ignores writes, and reads as 1 or 0 according to whether the memory system does or does not check the alignment of data memory accesses.

C (bit[2]) If a L1 unified cache is used, this is the enable/disable bit for the unified cache. If separate L1 caches are used, this is the enable/disable bit for the data cache. In either case:

0 = L1 unified/data cache disabled

1 = L1 unified/data cache enabled.

If the L1 cache is not implemented, this bit reads as 0 and ignores writes. If the L1 cache cannot be disabled, this bit reads as 1 and ignores writes.

The state of this bit does not affect other levels of cache in the system.

W (bit[3]) This is the enable/disable bit for the write buffer:

0 = Write buffer disabled

1 = Write buffer enabled.

If the write buffer is not implemented, this bit reads as zero (RAZ) and ignores writes. If the write buffer cannot be disabled, this bit reads as one and ignores writes.

SBO (bits[4:6])

These bits read as 1 and ignore writes.

B (bit[7]) This bit is used to configure the ARM processor to the endianness of the memory system.

ARM processors which support both little-endian and big-endian word-invariant memory systems use this bit to configure the ARM processor to rename the four byte addresses within a 32-bit word.

In V6 this becomes the mechanism by which legacy big-endian operating systems and applications can be supported.

don't flip by accident!

0 = configured little-endian memory system (LE)

1 = configured big-endian word-invariant memory system (BE-32)

Two configuration bits CFGEND[1:0] define the endian model at reset as described in Table A2-7 on page A2-35. (Previous architectures allowed an IMPLEMENTATION DEFINED configuration option to pre-set or reset this bit externally, depending on the external memory subsystem).

S (bit[8]) System protection bit, supported for backwards compatibility. The effect of this bit is described in *Access permissions* on page B4-8. The functionality is deprecated in ARMv6.

R (bit[9]) ROM protection bit, supported for backwards compatibility. The effect of this bit is described in *Access permissions* on page B4-8. The functionality is deprecated in ARMv6.

F (bit[10]) The meaning of this bit is IMPLEMENTATION DEFINED.

Z (bit[11]) On ARM processors which support branch prediction, this is the enable/disable bit for branch prediction:

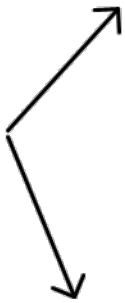
0 = Program flow prediction disabled

1 = Program flow prediction enabled.

If program flow prediction cannot be disabled, this bit reads as 1 and ignores writes. Program flow prediction includes all possible forms of speculative change of instruction stream prediction. Examples include static prediction, dynamic prediction, and return stacks.

On ARM processors that do not support branch prediction, this bit reads as 0 and ignores writes.

speed



I (bit[12]) If separate L1 caches are used, this is the enable/disable bit for the L1 instruction cache:

0 = L1 instruction cache disabled

1 = L1 instruction cache enabled.

If an L1 unified cache is used or the L1 instruction cache is not implemented, this bit reads as 0 and ignores writes. If the L1 instruction cache cannot be disabled, this bit reads as 1 and ignores writes.

The state of this bit does not affect further levels of cache in the system.

V (bit[13]) This bit is used to select the location of the exception vectors:

possibly useful.

0 = Normal exception vectors selected (address range `0x00000000-0x0000001C`)

1 = High exception vectors selected (address range `0xFFFF0000-0xFFFF001C`).

An implementation can provide an input signal that determines the state of this bit after reset.

RR (bit[14]) If the cache allows an alternative replacement strategy to be used that has a more predictable performance, this bit selects it:

for pathological conflict cases.

0 = Normal replacement strategy (for example, random replacement)

1 = Predictable strategy (for example, round-robin replacement).

L4 (bit[15]) This bit inhibits ARMv5T Thumb interworking behavior when set. It stops bit[0] updating the CPSR T-bit. The disable feature is deprecated in ARMv6

The instructions affected by this are:

- *LDM (1)* on page A4-36
- *LDR* on page A4-43
- *POP* on page A7-82.

DT (bit[16]) SBO. "should be 1"

SBZ (bit[17]) This bit reads as 0 and ignores writes.

IT (bit[18]) SBO.

SBZ (bit[19]) This bit reads as 0 and ignores writes.

ST (bit[20]) SBZ/UNP.

FI (bit[21]) Configure Fast Interrupt configuration. This bit may be used to reduce interrupt latency in an implementation by disabling IMPLEMENTATION DEFINED performance features:

0 = All performance features enabled

1 = Low interrupt latency configuration enabled.

U(bit[22])) This bit enables unaligned data access operation, including support for mixed little-endian and big-endian data.

0 = unaligned loads are treated as rotated aligned data accesses (legacy code behavior).

1 = unaligned loads and stores are permitted and mixed-endian data support enabled.

XP(bit[23]) Extended page table configure. This bit configures the hardware page table translation mechanism:

we use this.

0 = Subpage AP bits enabled.

1 = Subpage AP bits disabled. In this case, hardware translation tables support additional features.

VE(bit[24]) Configure vectored interrupts. Enables use of an IMPLEMENTATION DEFINED hardware mechanism to determine the interrupt vectors:

0 = Interrupt vectors are fixed:

- IRQ at `0x00000018` if V bit == 0, IRQ at `0xFFFF0018` if V bit == 1
- FIQ at `0x0000001C` if V bit == 0, FIQ at `0xFFFF001C` if V bit == 1

1 = Interrupt vectors are defined by an IMPLEMENTATION DEFINED hardware mechanism.

EE Bit[25] Mixed Endian exception entry. The EE bit is used to define the value of the CPSR E-bit on entry to an exception vector, including reset. The value is also used to indicate the endianness of page table data for page table lookups. This bit may be preset by CFGEND[1:0] pins on system reset. See *Endian configuration and control* on page A2-34 for more details.

enable L2

L2 Bit[26] L2 unified cache enable.

Bits[31:26] RESERVED. These bits are normally updated using read/modify/write techniques, to ensure that currently unallocated bits are not needlessly modified. Failure to observe this rule might result in code which has unexpected side effects on future processors. One exception that might be useful in some circumstances is that 0 can be written to these bits to restore them to their reset state.

make sure read-mod-write

B3.4.2 Auxiliary control register

The contents of this register are IMPLEMENTATION DEFINED. The register is guaranteed to be privileged read/write accessible, even if an implementation has not created any control bits within this register.

B3.4.3 Coprocessor access register

This register controls accesses to all coprocessors other than CP15 and CP14.

A typical use for this register is to enable an operating system to control coprocessor resource sharing among applications. Initially all applications are denied access to the shared resources. When an application attempts to use that resource it results in an Undefined Instruction exception. The Undefined Instruction handler can then grant access to that resource by setting the appropriate bits in the coprocessor access register.

Sharing resources among applications requires a state saving mechanism. Two possibilities are:

- the operating system, during a context switch, saves the state of the coprocessor if the last executing process had access rights to a coprocessor
- the operating system, after a request for access to a coprocessor, saves off the old coprocessor state with the last process to have access to it.

31 29	27	25	23	21	19	17	15	13	11	9	7	5	3	0
UNP/SBZP	cp13	cp12	cp11	cp10	cp9	cp8	cp7	cp6	cp5	cp4	cp3	cp2	cp1	cp0

Coprocessor access rights

Each pair of bits corresponds to the access rights for each coprocessor:

00 Access denied. Attempts to access corresponding coprocessor generates an undefined exception.

01 Privileged access only. Attempts to access corresponding coprocessor in user mode generates an undefined exception.

10 RESERVED (UNPREDICTABLE)

11 Full access (as defined by the relevant coprocessor).

After updating this register a PrefetchFlush instruction should be executed before the effect of the change to the coprocessor access register can be guaranteed to be visible. None of the instructions executed after changing this register and before the PrefetchFlush should be coprocessor instructions affected by the change in coprocessor access privileges.

After a system reset all coprocessor access rights are set to *Access denied*.

Any unimplemented coprocessors shall result in the associated bit field read-as-zero (RAZ). This allows system software to write all-1's to the coprocessor access register, then read back the result to determine which coprocessors are present, as part of an auto-configuration sequence.

If more than one coprocessor is used for a set of functionality (for example in the case with VFP, where CP10 and CP11 are used) then having different values in the fields of the coprocessor access register for those coprocessors can lead to UNPREDICTABLE behavior.

B3.5 Registers 2 to 15

System Control coprocessor registers other than registers 0 and 1 are allocated to specific areas as follows:

- CP15 registers 2 to 6, 8, 10, and 13 are allocated to the memory protection system. See Chapter B4 *Virtual Memory System Architecture*, Chapter B5 *Protected Memory System Architecture*, and Chapter B8 *Fast Context Switch Extension* for details of these registers.
- CP15 registers 7 and 9 are allocated to the control of caches, and write buffers. See Chapter B6 *Caches and Write Buffers* for details of these registers.
- CP15 register 11 is allocated to the level 1 memory DMA support. See Chapter B7 *Tightly Coupled Memory* for details.
- CP15 register 15 is reserved for IMPLEMENTATION DEFINED purposes. See the technical reference manual for the implementation or other implementation-specific documentation for details of the facilities available through this register.
- CP15 registers 12 and 14 are reserved for future expansion. Accessing (reading or writing) any of these registers is UNPREDICTABLE, and UNDEFINED from ARMv6.

