MANCHEstER 1824

The University of Manchester

Support for memory hierarchy

- Outline:
  - the ARM system control coprocessor
  - the ARM MMU architecture
  - the ARM memory protection unit (MPU)
  - the Level-2 cache controller
  - lockdown

  ☞ hands-on: memory protection

MANCHEstER 1824

The University of Manchester

Support for memory hierarchy

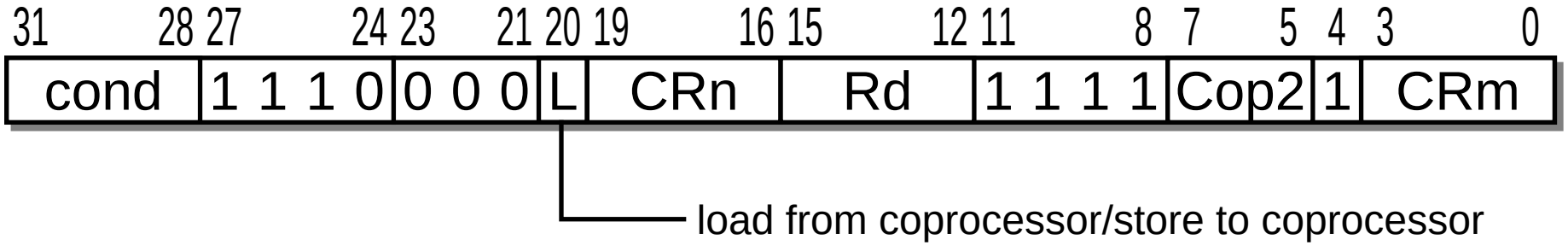
- Outline:
  - ➜ **the ARM system control coprocessor**
  - the ARM MMU architecture
  - the ARM memory protection unit (MPU)
  - the Level-2 cache controller
  - lockdown

  - ☞ hands-on: memory protection

The ARM system control coprocessor, CP15

- When an ARM system includes a cache and/or an MMU, these must be configured and controlled
  - this is done via the system control coprocessor, CP15
    - avoiding any pollution of the ARM address space
  - the architecture definition standardizes CP15 as far as possible

CP15 register transfer instructions



- only MCR and MRC instructions are supported

```
MCR    p15, 0, Rd, CRn, CR0, <Cop2>
MRC    p15, 0, Rd, CRn, CR0, <Cop2>
```

- these must execute in a privileged (non-user) mode
  - else the instruction is 'undefined'
- data is written from Rd to CRn, or vice versa

MANCHEstER 1824

The University of Manchester

Support for memory hierarchy

- Outline:
  - the ARM system control coprocessor
  - ➜ **the ARM MMU architecture**
  - the ARM memory protection unit (MPU)
  - the Level-2 cache controller
  - lockdown

  - ☞ hands-on: memory protection

MANCHEstER 1824

The University of Manchester

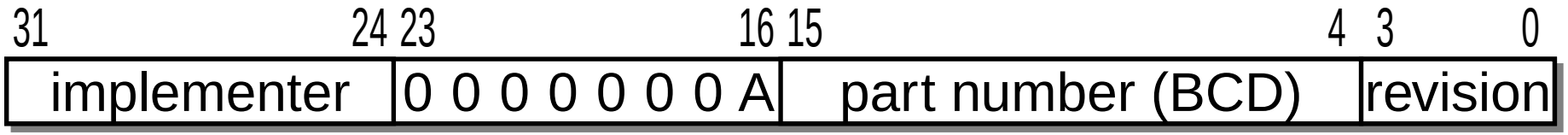
CP15 register structure - MMUs

Register	Purpose
0†	ID Register
1	Control
2	Translation Table Base
3	Domain Access Control
4	UNUSED
5†	Fault Status (data/instruction)
6	Fault Address
7	Cache Operations
8	TLB Operations
9†	Cache lockdown/TCM region
10	TLB lockdown
11-12	Reserved
13†	PID
14	Reserved
15	Test

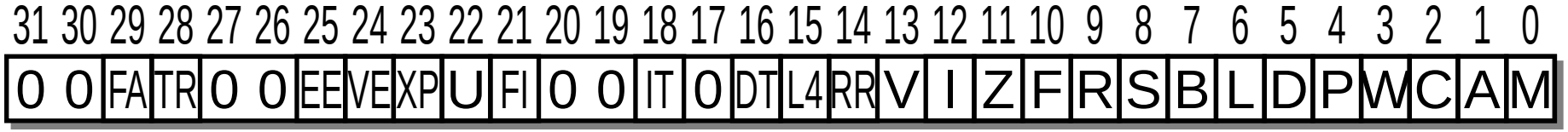
† some register addresses are subdivided using other address fields

CP15 registers - MMUs

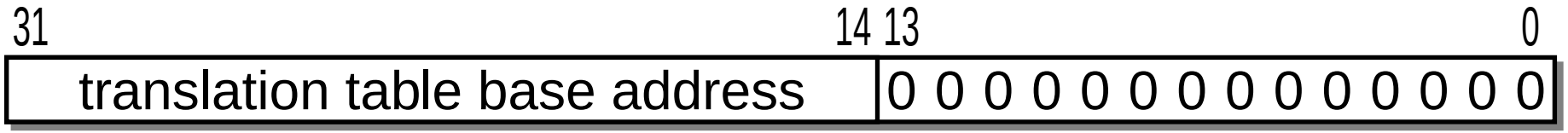
- Register 0



- Register 1



- Register 2 (Translation table base)



CP15 registers - MMUs

- Register 1 bits

Bit	Name	Function
0	M	MMU enable
1	A	Alignment fault check enable
2	C	(Unified) Cache enable
3	W	Write buffer Enable
4	P	PROG32 <effectively obsolete (= 1) >
5	D	DATA32 <effectively obsolete (= 1) >
6	L	Late abort <effectively obsolete (= 1) >
7	B	Big endian
8	S	System protection
9	R	ROM protection
10	F	*implementation defined*
11	Z	Branch prediction enable
12	I	Instruction cache enable
13	V	High Vectors selected
14	RR	Enable predictable (e.g. Round Robin) cache replacement
15	L4	Ignore bit <0> when loading PC (i.e. *don't* switch ARM/Thumb mode)

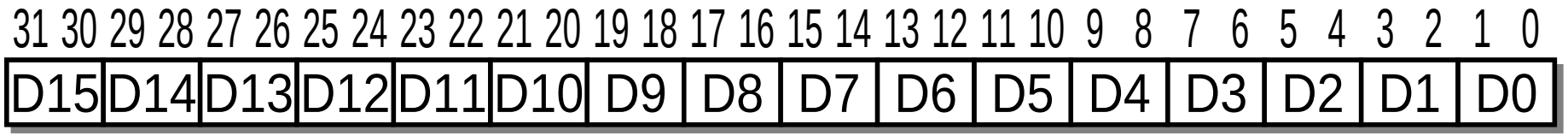
CP15 registers - MMUs

- Register 1 bits (mostly ARM11+ only)

Bit	Name	Function
16	DT	Enable data tightly coupled memory (TCM) <deprecated>
17	–	–
18	IT	Enable instruction tightly coupled memory (TCM) <deprecated>
19,20	–	–
21	FI	Low latency FIQ enable
22	U	Unaligned data access enable
23	XP	Extended page table disable
24	VE	Vectored interrupt controller (VIC) enable
25	EE	Endianness ($CPSR_{[9]}$) on exception
26,27	–	–
28	TR	TEX remap enable
29	FA	Force access permissions
30,31	–	–

CP15 registers - MMUs

- Register 3 (Domain access control)



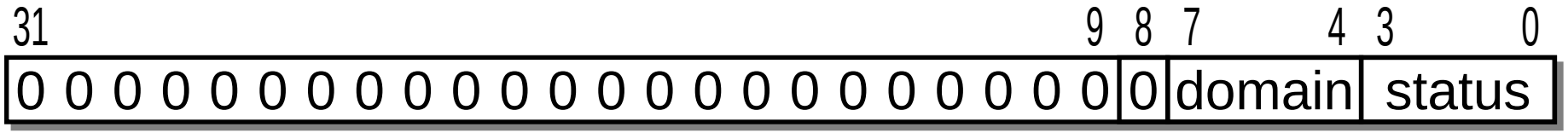
  - domain access control bits:

Value	Status	Description
00	No access	Any access will generate a domain fault
01	Client	Page and section permission bits are checked
10	Reserved	Do not use
11	Manager	Page and section permission bits are not checked

  - More on **domains**, later

CP15 registers - MMUs

- Register 5 (Fault status)



- Register 6 (Fault address)

- Registers 7,8
  - perform various cache and TLB control functions (e.g. flush)
- Register 9 (cache lockdown)
- Register 10 (TLB lockdown)

MANCHEstER 1824

The University of Manchester

The ARM MMU architecture

- The ARM MMU performs:
  - logical to physical address translation
  - memory protection, checking permissions
- The ARM MMU has:
  - a 2-level page table
  - table-walking hardware
  - a TLB
    - separate I-D caches can have separate TLBs.

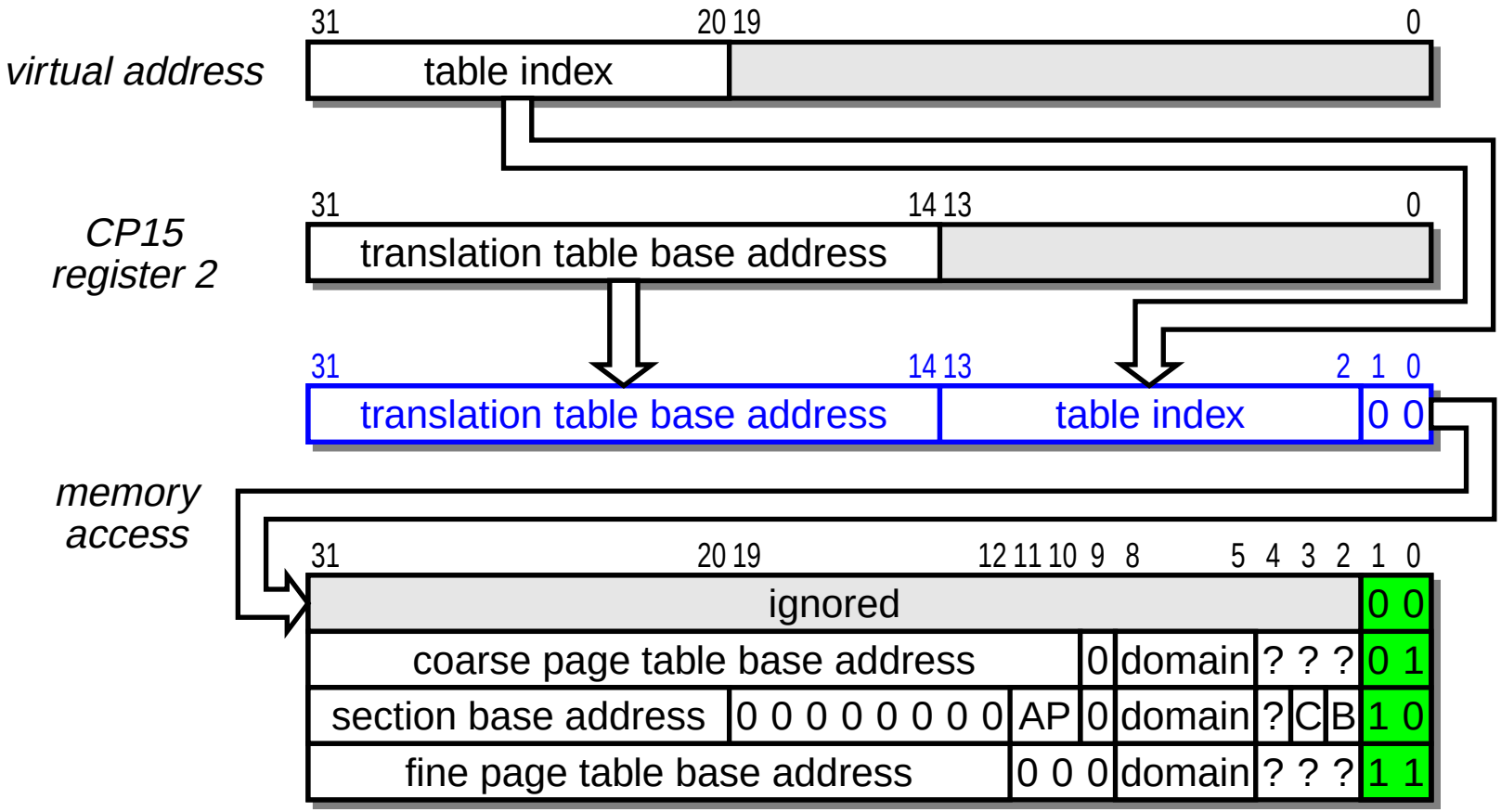
MANCHEstER 1824

The University of Manchester

The ARM MMU architecture

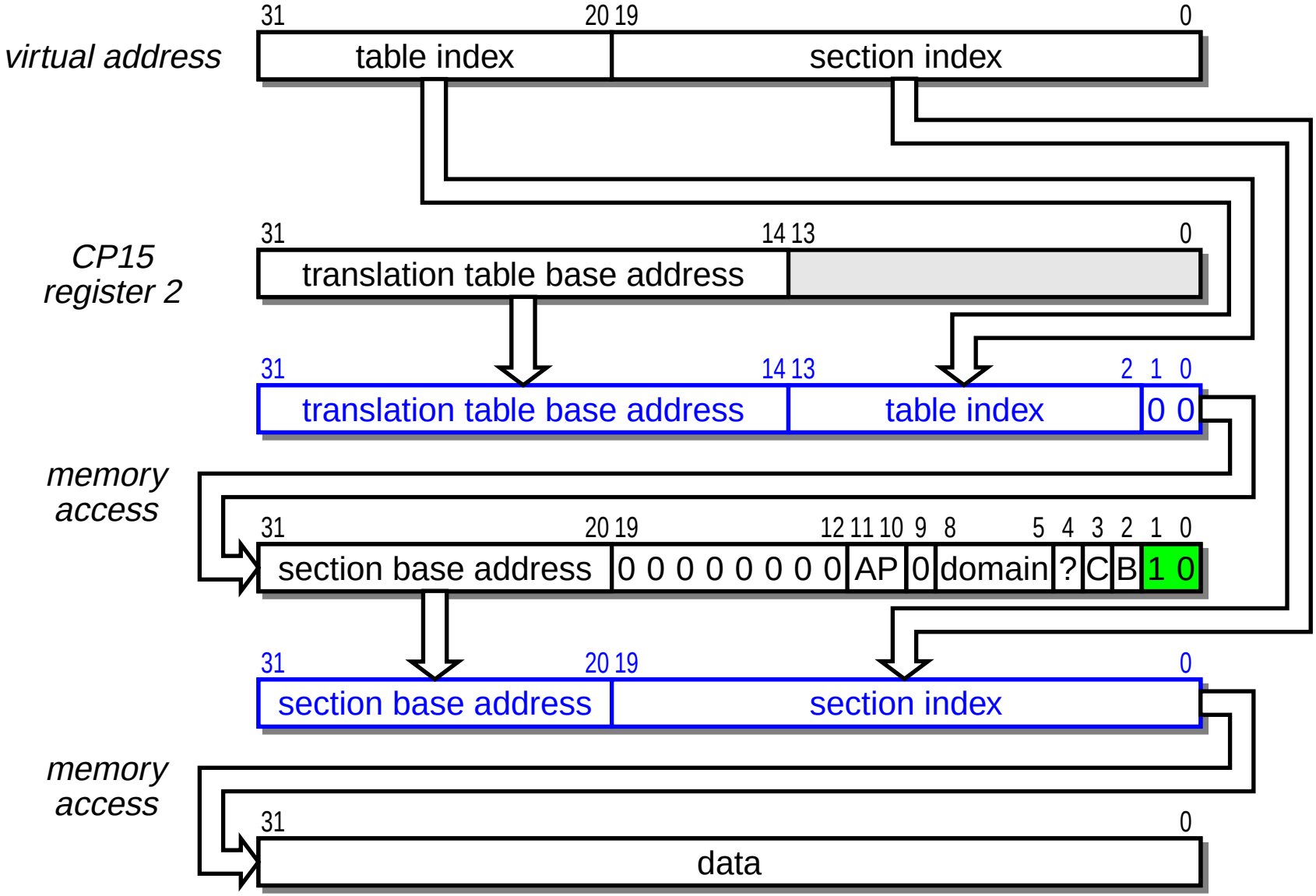
- One basic mechanism can map:
  - *Sections* - 1 Mbyte blocks of memory
  - *Large pages* - 64 Kbyte blocks
    - with access control on 16 Kbyte **subpages**
  - *Small pages* - 4 Kbyte blocks
    - with access control on 1 Kbyte **subpages**
  - *Tiny pages* - 1 Kbyte blocks (from v5)
- the first-level translation fetch defines the size of unit

First-level translation fetch



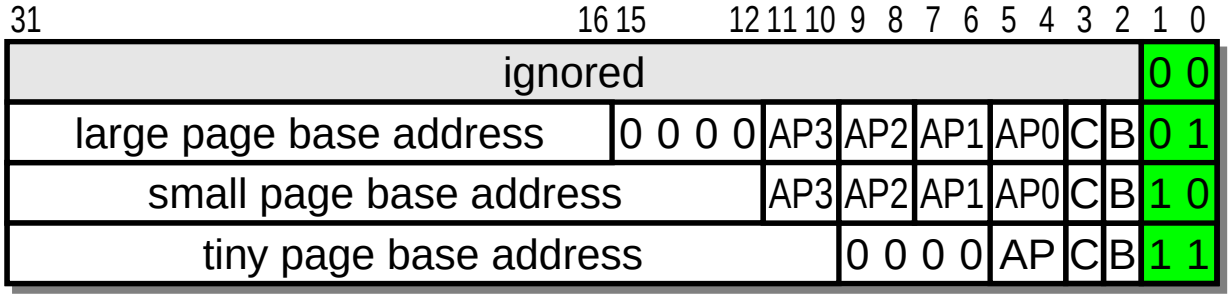
- 12 MSBs of address index into table (on 16KB boundary)
- bits <1, 0> in table entry determine what happens next

Section translation sequence



Page translation sequence

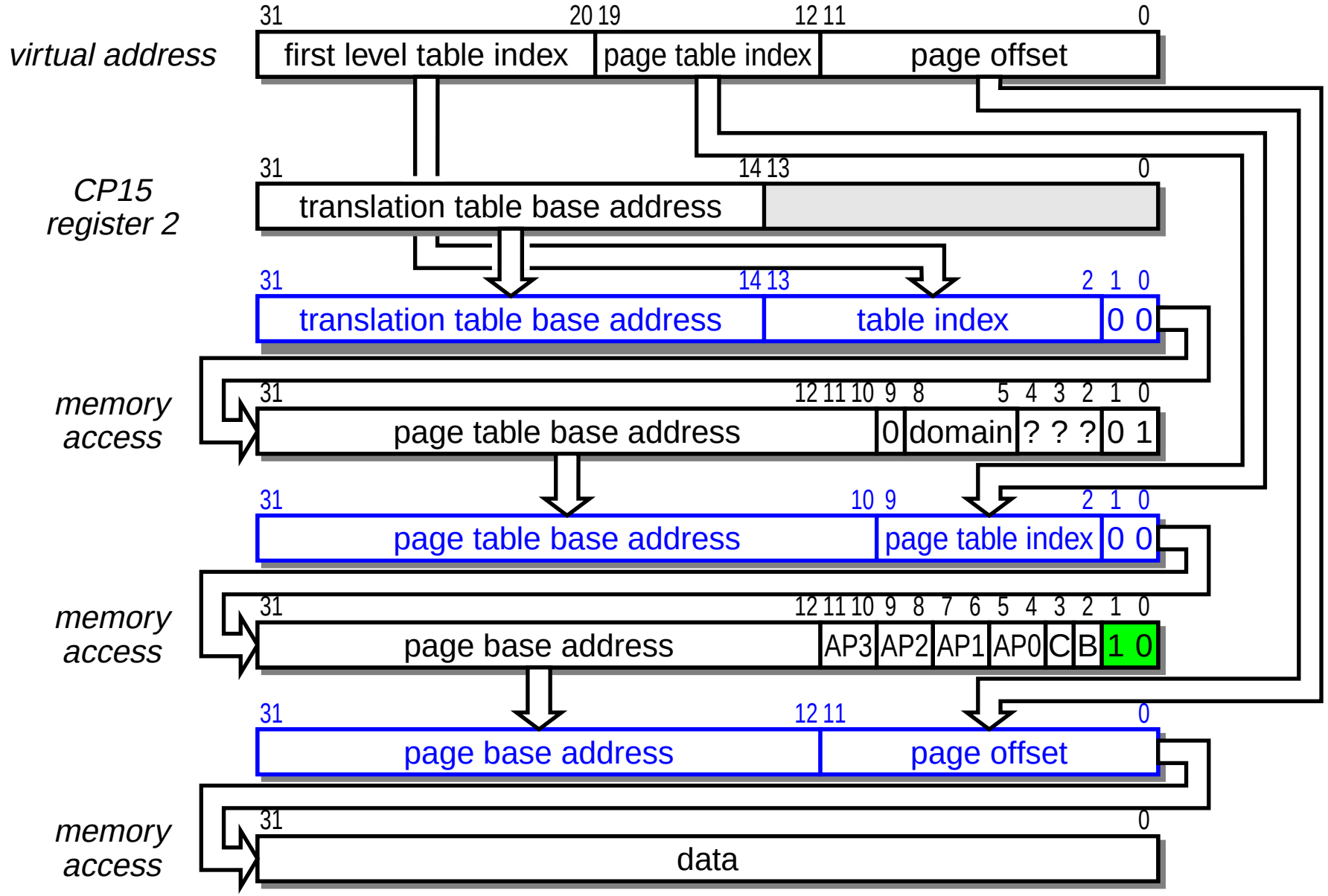
- The page base address & page table index are concatenated
- The 32-bit page table entry is read



- The two LSBs determine the page size

[1:0]	Name	Page size	Notes
00	Unmapped	–	Generate a translation fault Remaining bits can be used by software
01	Large	64 Kbyte	Repeat entry 64/16 times in a fine/coarse page table
10	Small	4 Kbyte	Repeat entry 4 times in a fine page table
11	Tiny	1 Kbyte	Do not use with coarse second level tables

Small page translation sequence



MANCHEstER 1824

The University of Manchester

Domains

- Domains are:
  - groups of sections and/or pages
  - with common access permissions
- They allow:
  - different processes to share translation tables
  - whilst retaining some mutual protection
  - giving lightweight process switching
    - saving significant power

MANCHEstER 1824

The University of Manchester

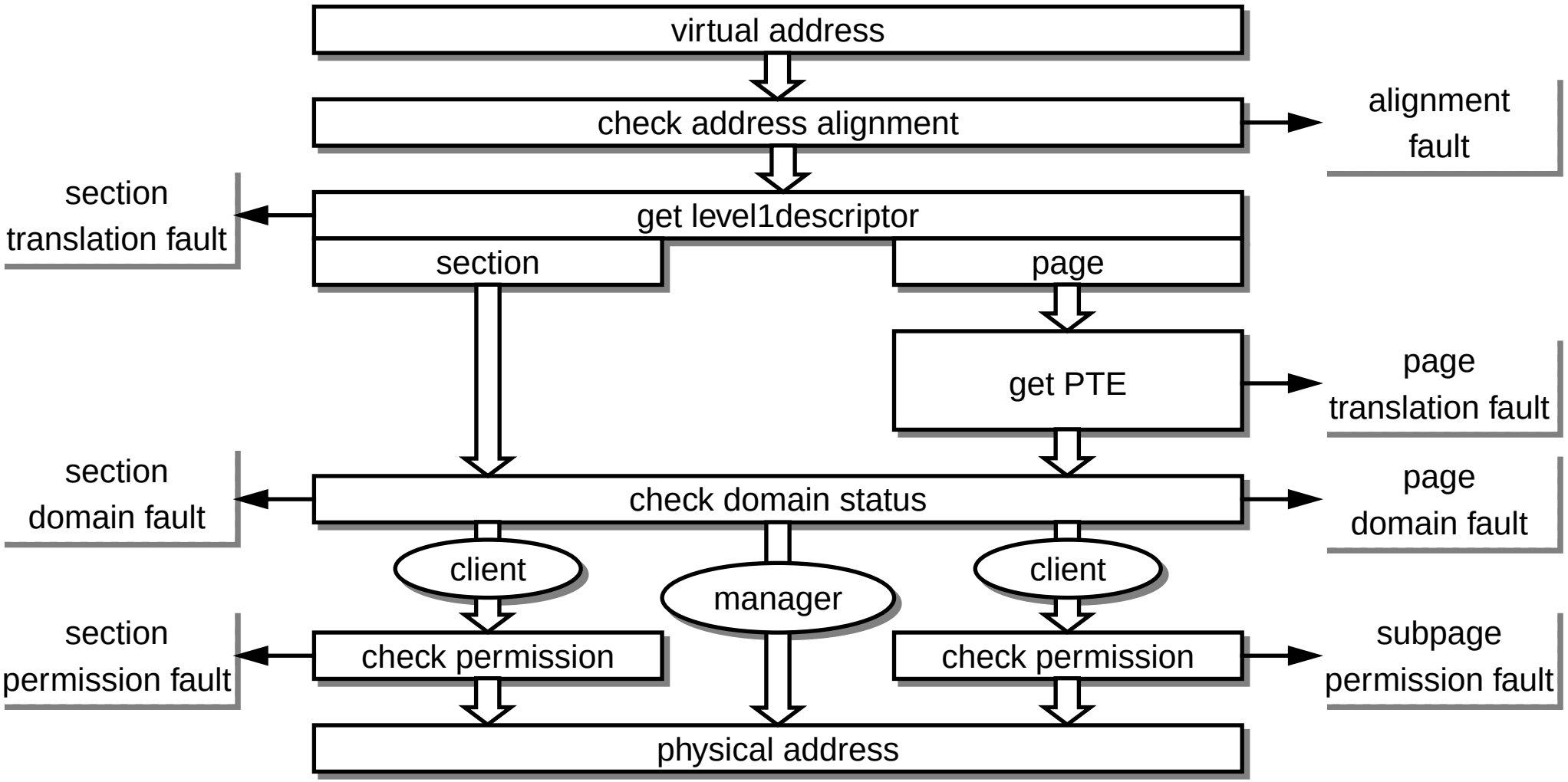
Domains

- A program may be:
  - a *client* of a domain, observing individual page and section access permissions
  - a *manager* of a domain, bypassing the individual access permissions
  - neither - all access is denied
- The access status is determined by:
  - the domain the page/section belongs to
  - the client/manager status in CP15 reg 3

MANCHEstER 1824

The University of Manchester

Access permission checking scheme



Access permissions

- domain information in CP15 reg 3
- domain identifier and AP (Access Permissions) in section/ subpage page table entry
- the S & R bits in CP15 reg 1

AP	S	R	Supervisor	User
00	0	0	No access	No access
00	1	0	Read only	No access
00	0	1	Read only	Read only
00	1	1	Do not use	
01	–	–	Read/write	No access
10	–	–	Read/write	Read only
11	–	–	Read/write	Read/write

MANCHEstER 1824

The University of Manchester

Support for memory hierarchy

- Outline:
  - the ARM system control coprocessor
  - the ARM MMU architecture
  - ➜ **the ARM memory protection unit (MPU)**
  - the Level-2 cache controller
  - lockdown

  - ☞ hands-on: memory protection

MPU function

- Defines eight protection regions of 2^N bytes (N >= 12)
  - appropriately aligned
  - regions may overlap: highest numbered region checked first
- Specify properties of each region {C, B, AP}
- The addresses themselves are not modified

MANCHEstER 1824

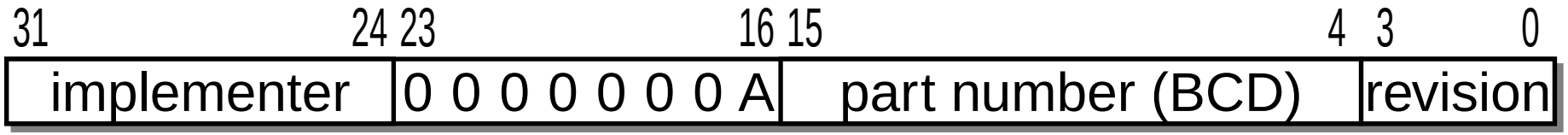
The University of Manchester

CP15 register structure - Protection units

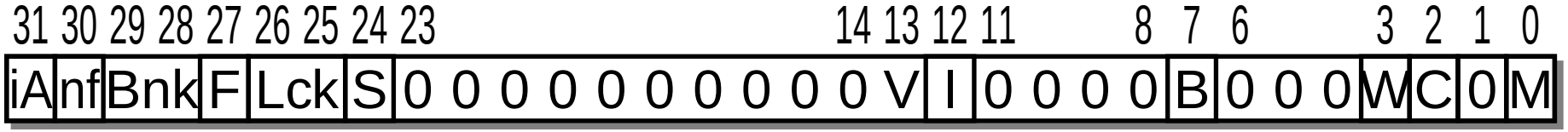
Register	Purpose
0	ID Register
1	Configuration
2	Cache Control
3	Write Buffer Control
4	UNUSED
5	Access Permissions
6	Region Base & Size
7	Cache Operations
8	UNUSED
9	Cache Lock Down
10 to 14	UNUSED
15	Test

CP15 registers - Protection units

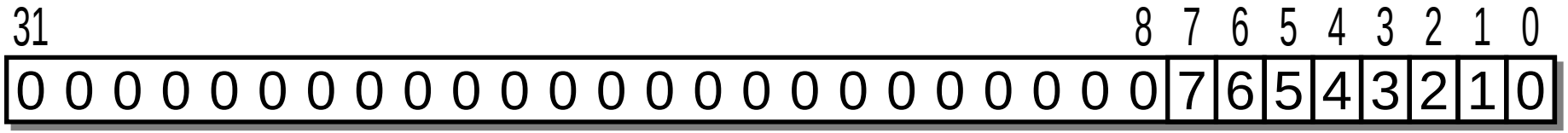
- Register 0



- Register 1

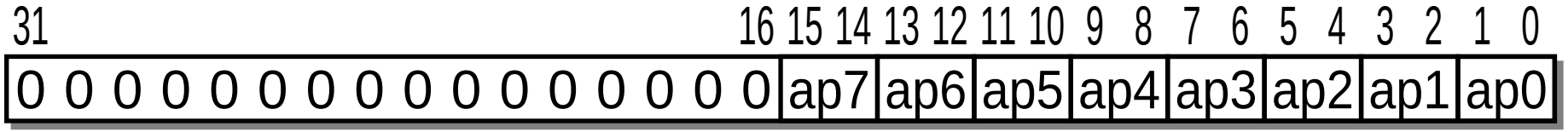


- Register 2, 3 (cacheable, write bufferable)

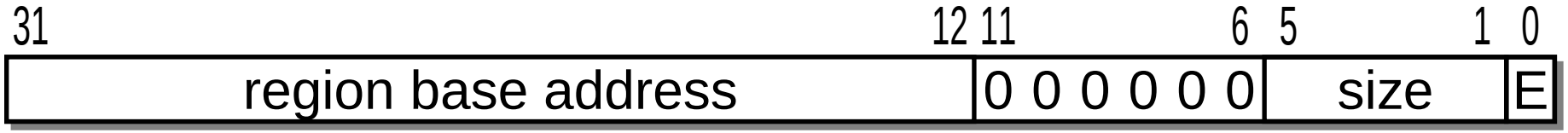


CP15 registers - Protection units

- Register 5 (access permissions)



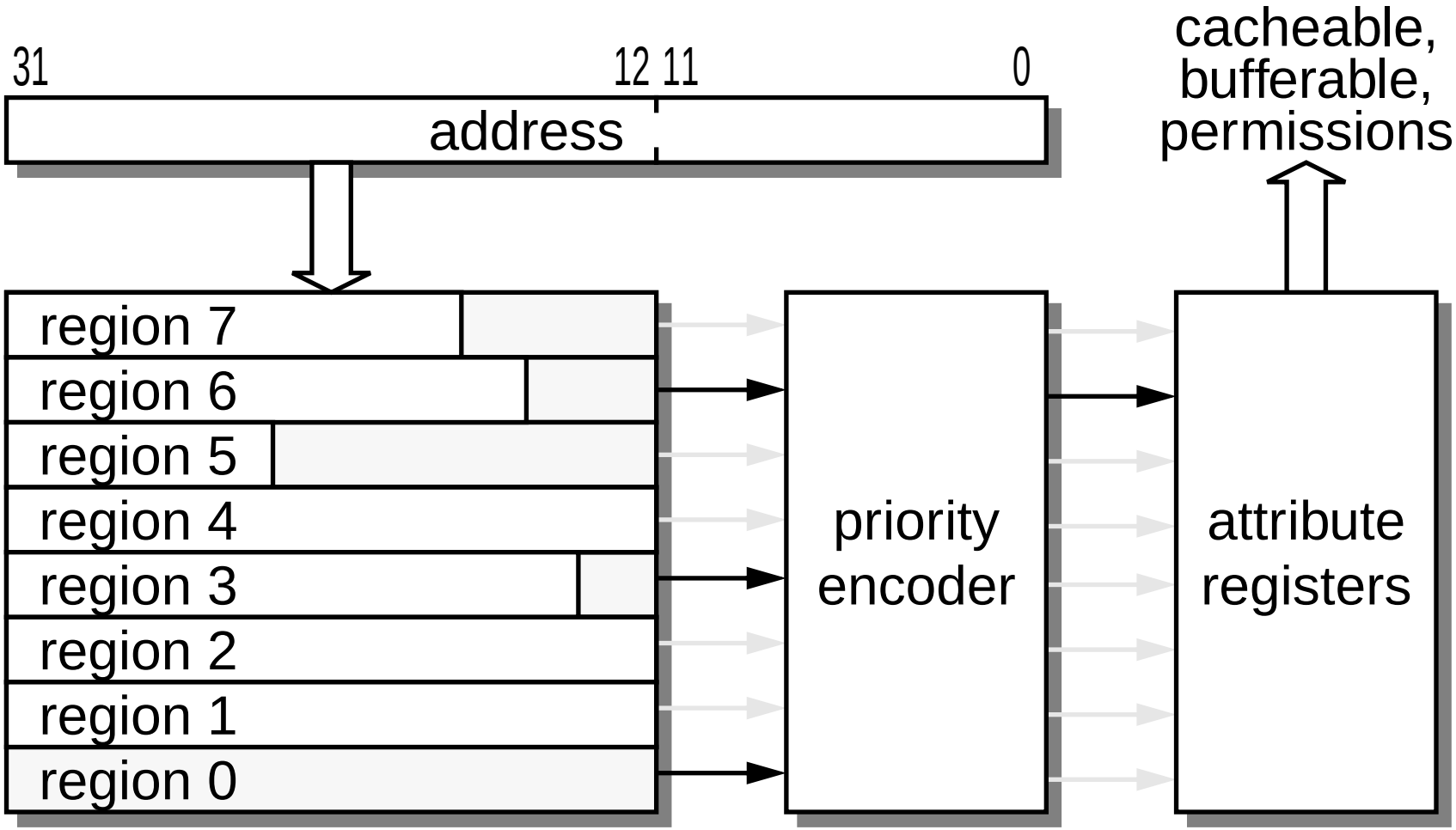
- Register 6 (protection area control)



  - <CRm> is used to specify which region

- Registers 7,9
  - perform various cache control functions, including lock-down

Protection Unit - example



MANCHEstER 1824

The University of Manchester

Support for memory hierarchy

- Outline:
  - the ARM system control coprocessor
  - the ARM MMU architecture
  - the ARM memory protection unit (MPU)
  - ➜ **the Level-2 cache controller**
  - lockdown

  - ☞ hands-on: memory protection

MANCHEstER 1824

The University of Manchester

L220 cache controller

- The L1 cache(s) form part of the CPU macrocell
- The L220 is a **controller** for an **on-chip L2 cache**
  - 16K-2Mbytes (‘bring your own’ RAM)
  - unified
  - physically addressed/tagged
  - 8-way associative (reducable by lock-down)
  - 32 byte lines
  - allocation/writeback configurable
  - no ‘snooping’ (coherency assurance is a software task)

MANCHEstER 1824

The University of Manchester

L220 programmers' model

- Control registers are memory mapped
  - relocatable 4KB 'page'

Register	Offsets	Function
0	0x000-0x0FC	System ID and Cache Type
1	0x100-0x1FC	Control
2	0x200-0x2FC	Interrupt/Counter Control
3-6	–	Reserved
7	0x700-0x7FC	Cache Maintenance Operations
8	0x800-0x8FC	Reserved
9	0x900-0x9FC	Cache Lockdown
10-14	–	Reserved
15	0xF00-0xFFC	Test and Debug

    - note: each "register" can encompass up to 64 32-bit values

MANCHEstER 1824

The University of Manchester

Support for memory hierarchy

- Outline:
  - the ARM system control coprocessor
  - the ARM MMU architecture
  - the ARM memory protection unit (MPU)
  - the Level-2 cache controller
  - ➜ **lockdown**

  - ☞ hands-on: memory protection

MANCHEstER 1824

The University of Manchester

Lockdown

- On cache miss:
  - reject a line, chosen by some allocation policy
  - overwrite that position with a new line
- Rejection policy
  - Round-robin – lines are rejected in turn
  - Pseudo-random
- **Lockdown** prevents some (or all) lines being rejected
  - part of the cache is ‘frozen’

Lockdown

- Different ARMs have
  - different cache architectures
  - therefore different lockdown schemes
    - high associativity caches tend to lock down a *number* of ‘ways’
      - e.g. ARM920T
    - low associativity caches tend to lock down ways *individually*
      - e.g. ARM1176JFZ
- TLBs can also be locked down

Lockdown

- Why?
  - gives better predictability in real-time code
- How?
  - unlock
  - get cache replacement into known state
    - maybe need to specify replacement strategy
  - access required area (loads cache)
  - lock

MANCHEstER 1824

The University of Manchester

Hands-on: memory protection

- Memory protection:
  - set up a memory protection model within ARMulator
  - generate memory protection faults

☞ Follow the ‘Hands-on’ instructions