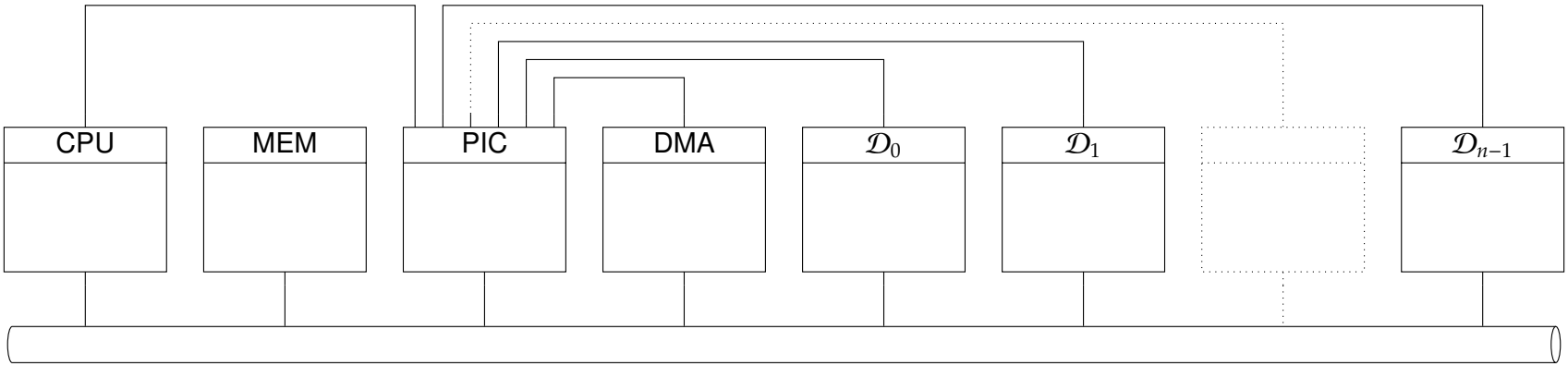
- Problem: our example computer system looks like this



but (so far) we only really know about one component, i.e., the processor.

- Goals: explain
  1. what some of the *other* components are, plus
  2. how the processor communicates with and hence uses them

and thus how **Input/Output (I/O)** functionality is realised.

Definition (**interrupt** [9])

An **interrupt** is an event (or condition) where normal execution of instructions is halted: two major classes exist, namely

- **hardware interrupt** are (typically) generated asynchronously, by an external source, and intentionally (e.g., a hardware device), while
- **software interrupt** are (typically) generated synchronously, by an internal source, and either intentionally (e.g., system call, cf. **trap**), or unintentionally (e.g., divide-by-zero, cf. **exception**).

Definition (**interrupt handler** [9])

Each **Interrupt ReQuest (IRQ)** causes a software **interrupt handler** to be invoked, whose task is to respond. Note that

- **interrupt latency** measures the time between an interrupt being requested and handled, and
- an **interrupt vector table** (located at a known address) allows a specific handler to be invoked for each interrupt type.

Concepts (2)

- The overall process of handling an interrupt is
  1. detect the interrupt,

Concepts (2)

- The overall process of handling an interrupt is
  1. detect the interrupt,
  2. update the processor mode (e.g., to kernel mode),

Concepts (2)

- The overall process of handling an interrupt is
  1. detect the interrupt,
  2. update the processor mode (e.g., to kernel mode),
  3. save the processor state, e.g., using
     - **shadow registers**, or
     - **shadow stacks**

Concepts (2)

- The overall process of handling an interrupt is
  1. detect the interrupt,
  2. update the processor mode (e.g., to kernel mode),
  3. save the processor state, e.g., using
     - **shadow registers**, or
     - **shadow stacks**
  4. execute an (or the) interrupt handler,

Concepts (2)

- The overall process of handling an interrupt is
  1. detect the interrupt,
  2. update the processor mode (e.g., to kernel mode),
  3. save the processor state, e.g., using
     - **shadow registers**, or
     - **shadow stacks**
  4. execute an (or the) interrupt handler,
  5. update the processor mode (e.g., to user mode),

Concepts (2)

- The overall process of handling an interrupt is
  1. detect the interrupt,
  2. update the processor mode (e.g., to kernel mode),
  3. save the processor state, e.g., using
     - **shadow registers**, or
     - **shadow stacks**
  4. execute an (or the) interrupt handler,
  5. update the processor mode (e.g., to user mode),
  6. restart the interrupted instruction.

Definition (**interrupt controller** [9])

External hardware devices are interfaced with the processor via an **interrupt controller**, which

- multiplexes a large(r) number of devices to a small(er) number of interrupt signals (into the processor), and
- offer extended functionality, such as priority levels.

Definition (**(non-)maskable interrupt** [9])

An interrupt may be

- **maskable** if it can be ignored (or disabled) by setting an **interrupt mask** (e.g., within a control register), or
- **non-maskable** otherwise.

Definition (**(im)precise interrupt** [9, 7])

An interrupt is deemed

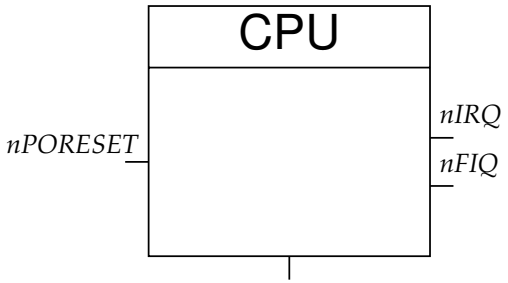
- **precise** if it leaves the processor in a well-defined state, or
- **imprecise** otherwise.

Per [9, 7], well-defined is taken to mean

1. the value of PC is retained somehow,
2. all instructions before PC have completed,
3. no instructions after PC have completed, and
4. the execution state of instruction at PC is known.

Implementation (1) – Cortex-A8: step #1 detect the interrupt

- Interrupt detection is managed automatically by the processor

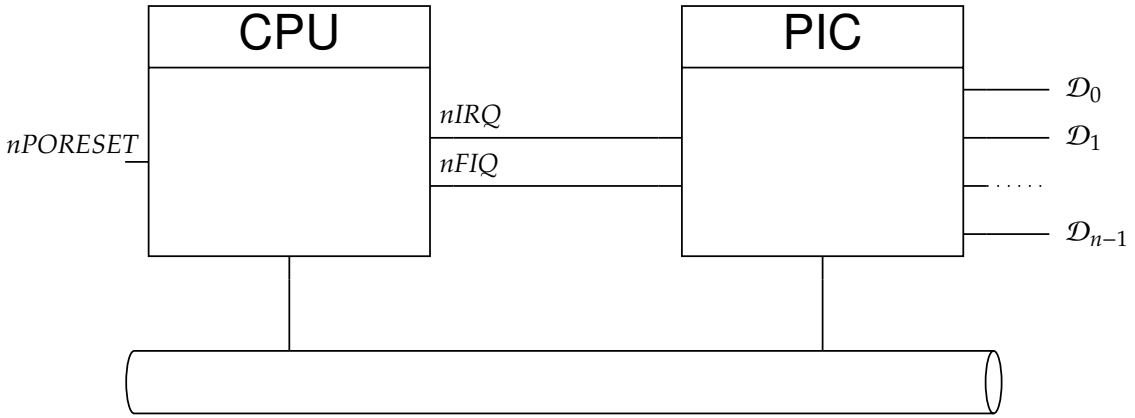


st.

- an interrupt can be requested, e.g., by
  - a software trap,
  - a software exception, or
  - a hardware signal,
- CPSR[F] and CPSR[I] mask FIQ- and IRQ-based interrupts respectively.

Implementation (1) – Cortex-A8: step #1 detect the interrupt

- Interrupt detection is managed automatically by the processor



st.

- an interrupt can be requested, e.g., by
  - a software trap,
  - a software exception, or
  - a hardware signal,
- CPSR[F] and CPSR[I] mask FIQ- and IRQ-based interrupts respectively, and
- features are (optionally) added via a *programmable* interrupt controller (e.g., PL190 [1]).

Implementation (2) – Cortex-A8: step #2 update the processor mode

ARMv7-A processor modes [3, Table B1-1]

Name	Mnemonic	CPSR[M]	Privilege level	Security state
User	USR	$10000_{(2)}$	PL0	Either
Fast interrupt (FIQ)	FIQ	$10001_{(2)}$	PL1	Either
Interrupt (IRQ)	IRQ	$10010_{(2)}$	PL1	Either
Supervisor	SVC	$10011_{(2)}$	PL1	Either
Monitor	MON	$10110_{(2)}$	PL1	Secure
Abort	ABT	$10111_{(2)}$	PL1	Either
Hypervisor	HYP	$11010_{(2)}$	PL2	Non-secure
Undefined	UND	$11011_{(2)}$	PL1	Either
System	SYS	$11111_{(2)}$	PL1	Either

Implementation (3) – Cortex-A8: step #3 save the processor state

privileged modes

USR mode	FIQ mode	IRQ mode	SVC mode	MON mode	ABT mode	HYP mode	UND mode	SYS mode
r0	r0	r0	r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7	r7	r7	r7
r8	r8	r8	r8	r8	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12	r12	r12	r12
r13	r13_fiq	r13_irq	r13_svc	r13_mon	r13_abt	r13_hyp	r13_und	r13
r14	r14_fiq	r14_irq	r14_svc	r14_mon	r14_abt	r14_hyp	r14_und	r14
r15	r15	r15	r15	r15	r15	r15	r15	r15
cpsr	cpsr	cpsr	cpsr	cpsr	cpsr	cpsr	cpsr	cpsr
	spsr_fiq	spsr_irq	spsr_svc	spsr_mon	spsr_abt	spsr_hyp	spsr_und	

Implementation (4) – Cortex-A8: step #4 execute an interrupt handler

ARMv7-A interrupt handling [3, Tables B1-3 + B1-4]

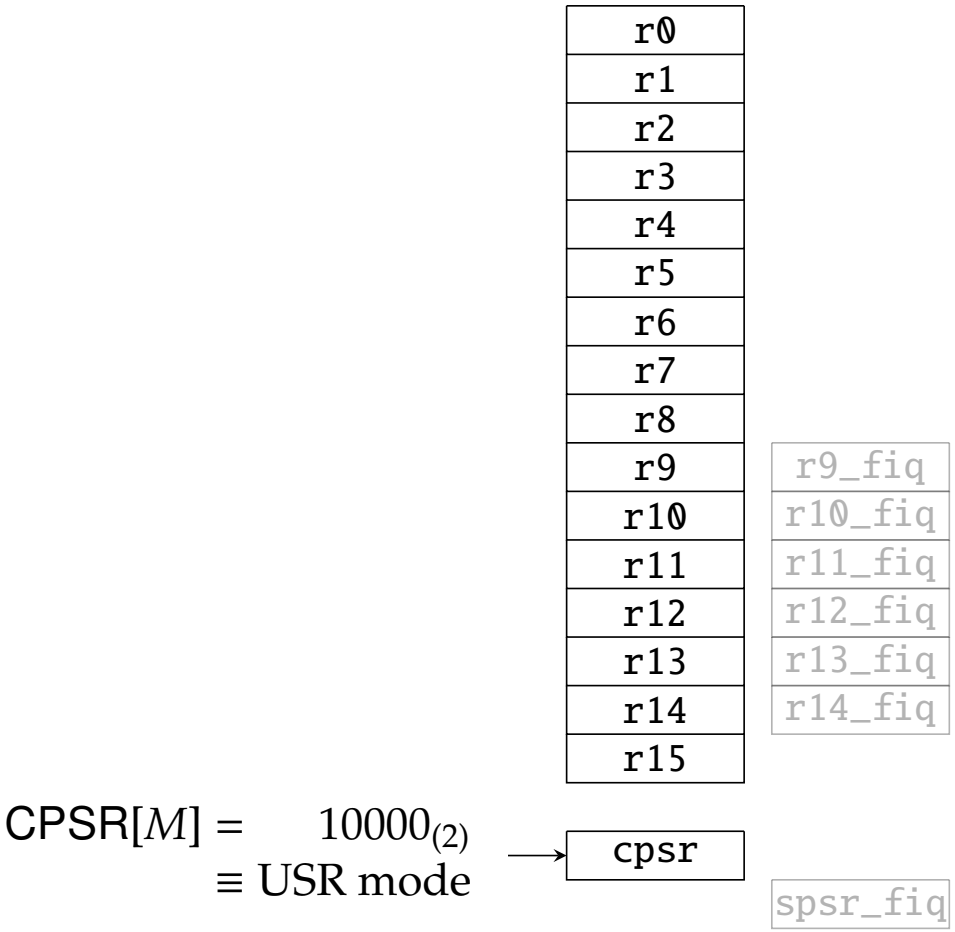
Type	Entry mode	Entry low address	Entry high address
Reset	SVC	$00000000_{(16)}$	$FFFF0000_{(16)}$
Undefined instruction	UND	$00000004_{(16)}$	$FFFF0004_{(16)}$
Software interrupt	SVC	$00000008_{(16)}$	$FFFF0008_{(16)}$
(Pre-)fetch abort	ABT	$0000000C_{(16)}$	$FFFF000C_{(16)}$
Data abort	ABT	$00000010_{(16)}$	$FFFF0010_{(16)}$
		$00000014_{(16)}$	$FFFF0014_{(16)}$
IRQ	IRQ	$00000018_{(16)}$	$FFFF0018_{(16)}$
FIQ	FIQ	$0000001C_{(16)}$	$FFFF001C_{(16)}$

ARMv7-A interrupt handling [2, Table 2-12]

Type	Return offset	Return instruction
Reset	$-0_{(10)}$	
Undefined instruction	$-0_{(10)}$	`movs pc, lr`
Software interrupt	$-0_{(10)}$	`movs pc, lr`
(Pre-)fetch abort	$-4_{(10)}$	`subs pc, lr, #4`
Data abort	$-8_{(10)}$	`subs pc, lr, #8`
IRQ	$-4_{(16)}$	`subs pc, lr, #4`
FIQ	$-4_{(16)}$	`subs pc, lr, #4`

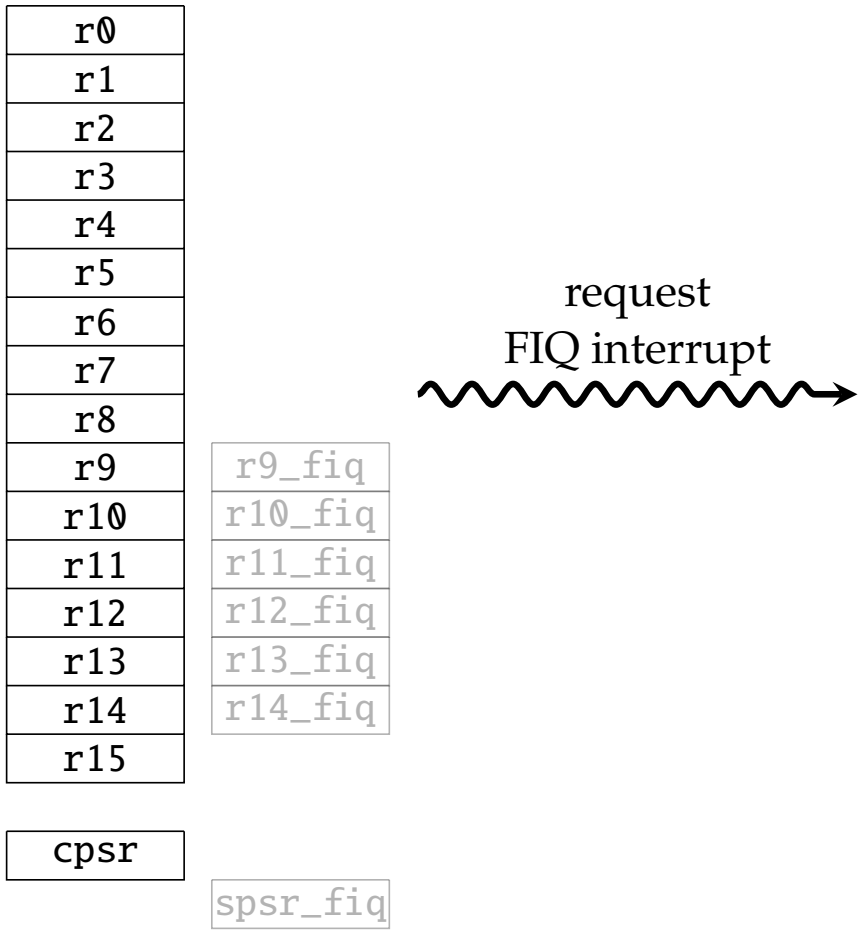
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



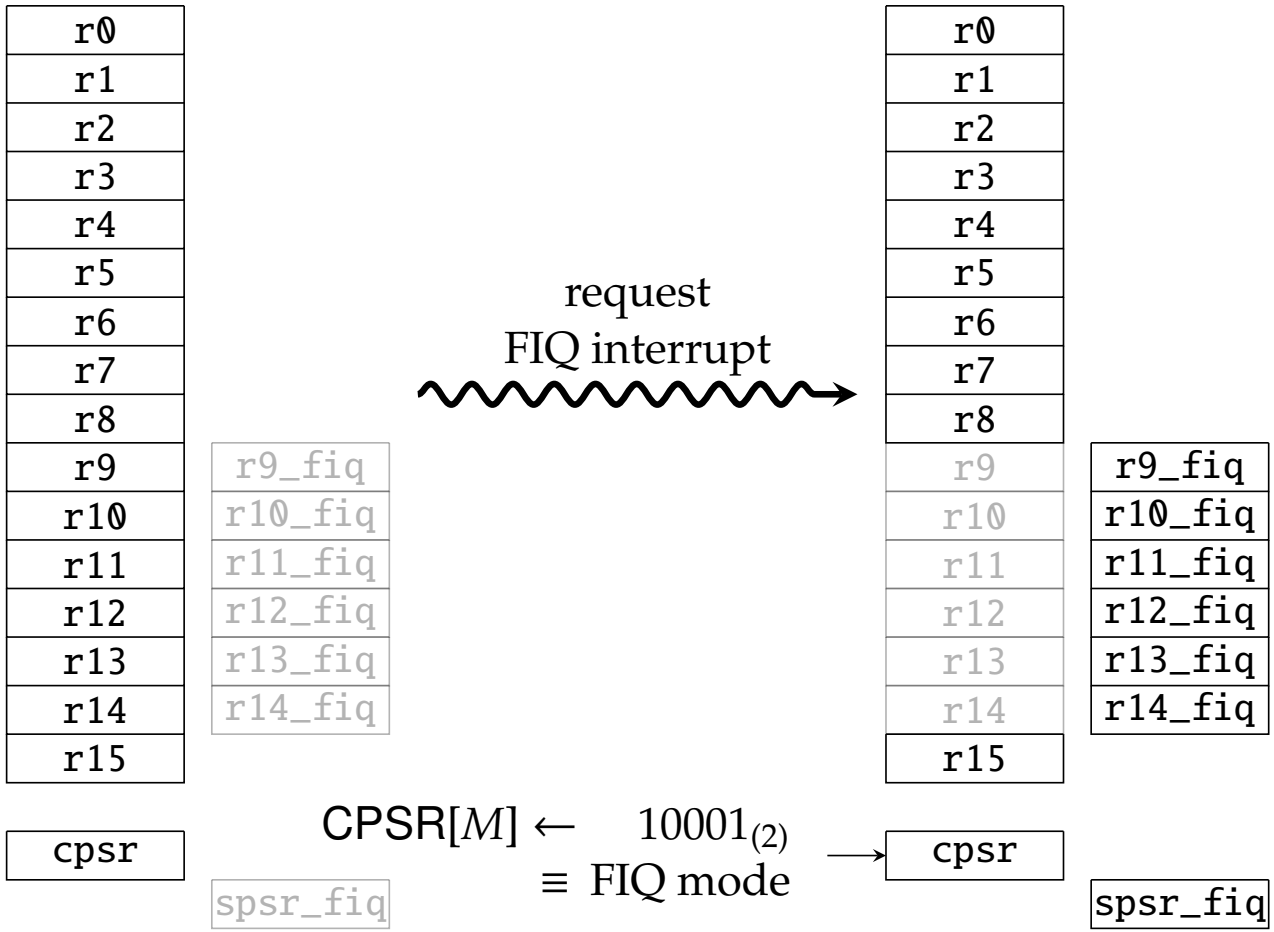
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



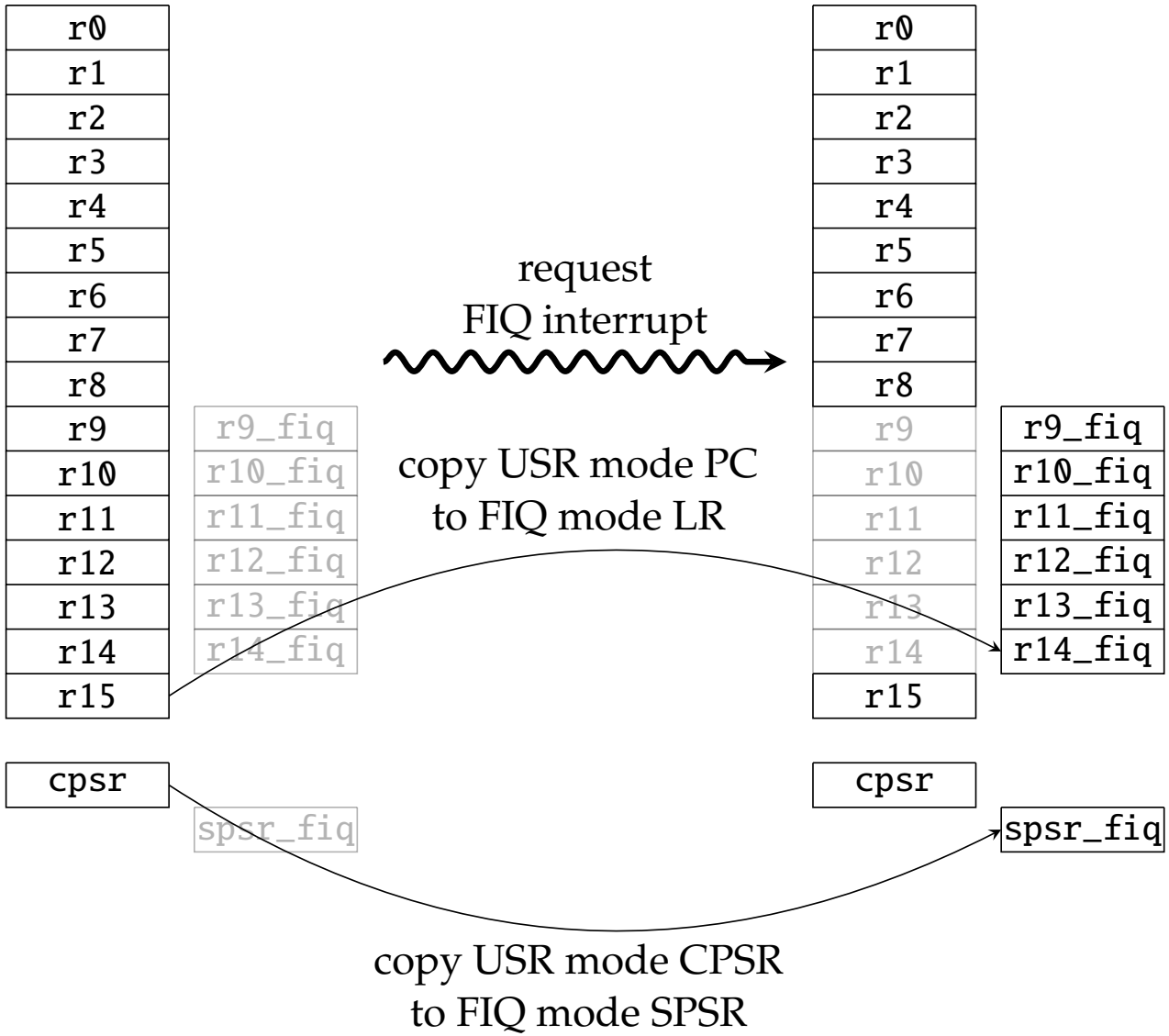
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



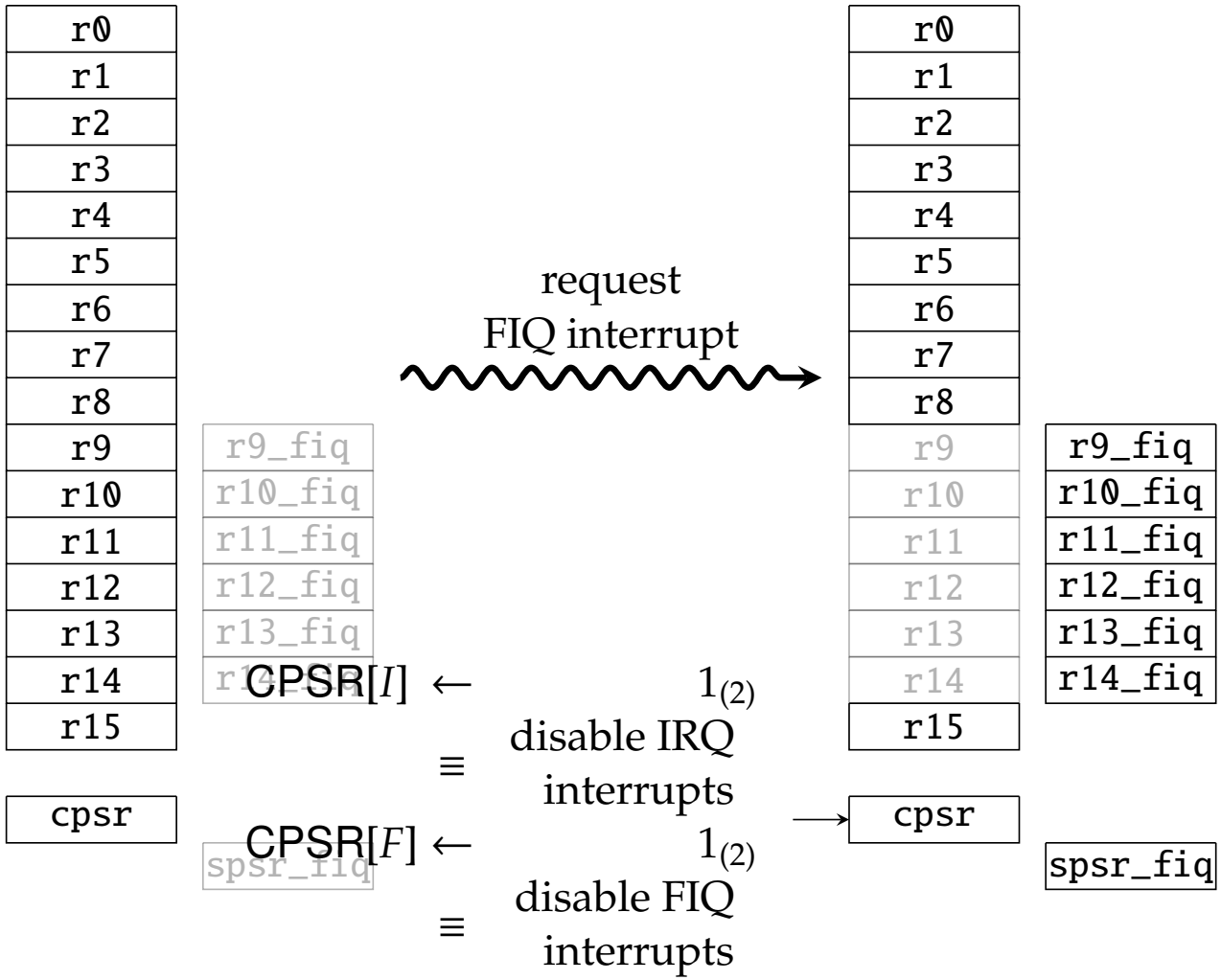
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



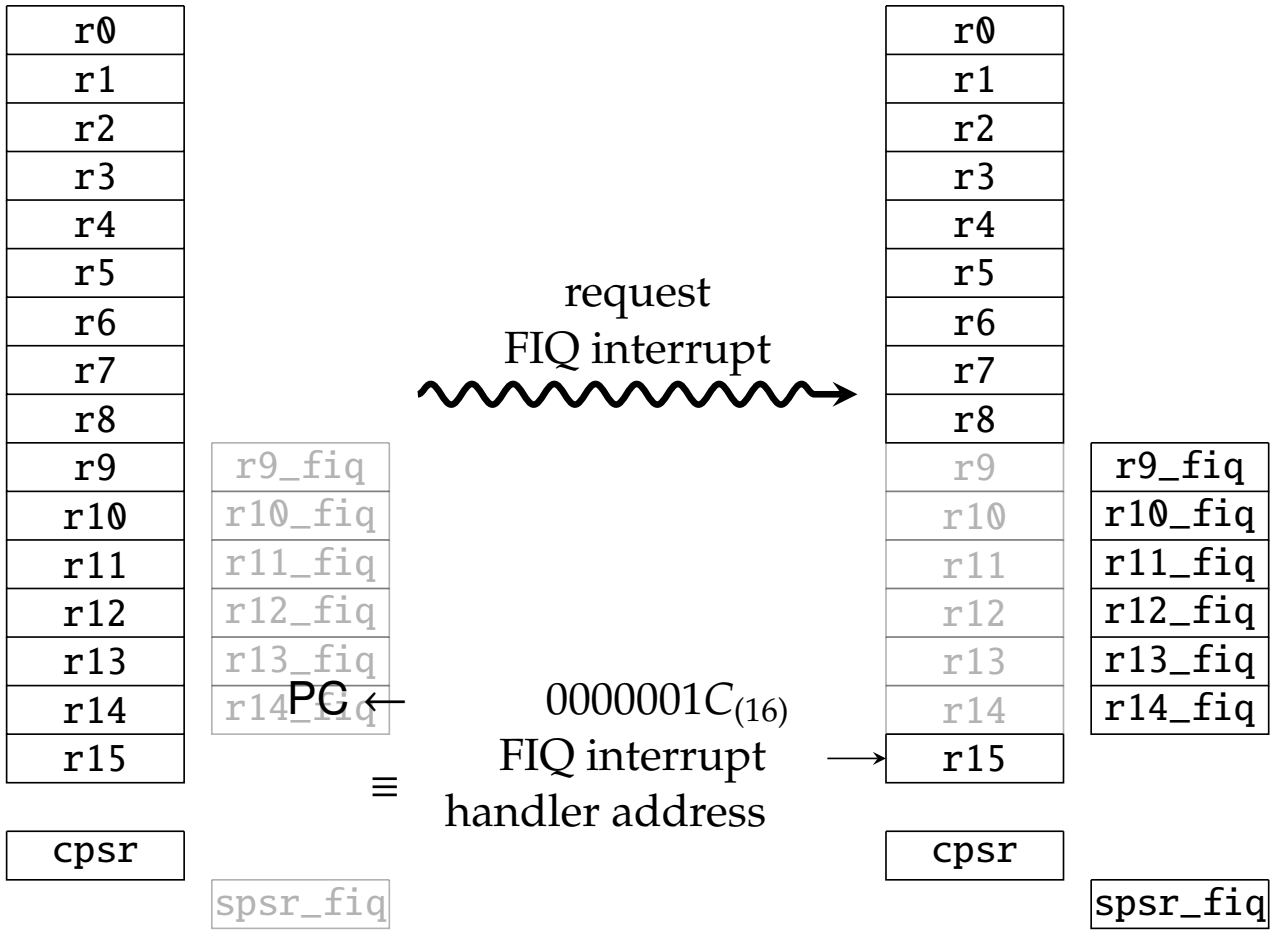
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



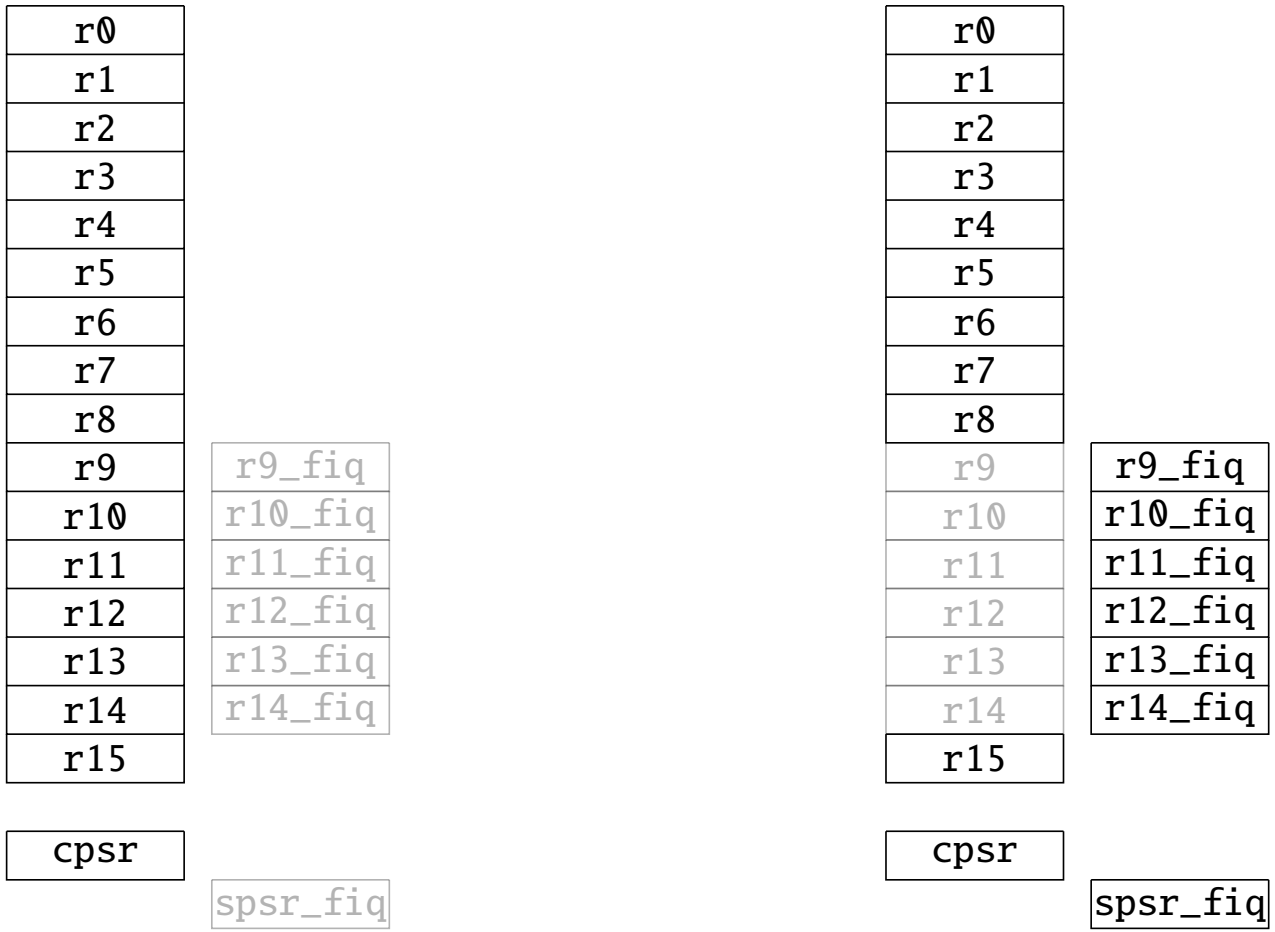
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



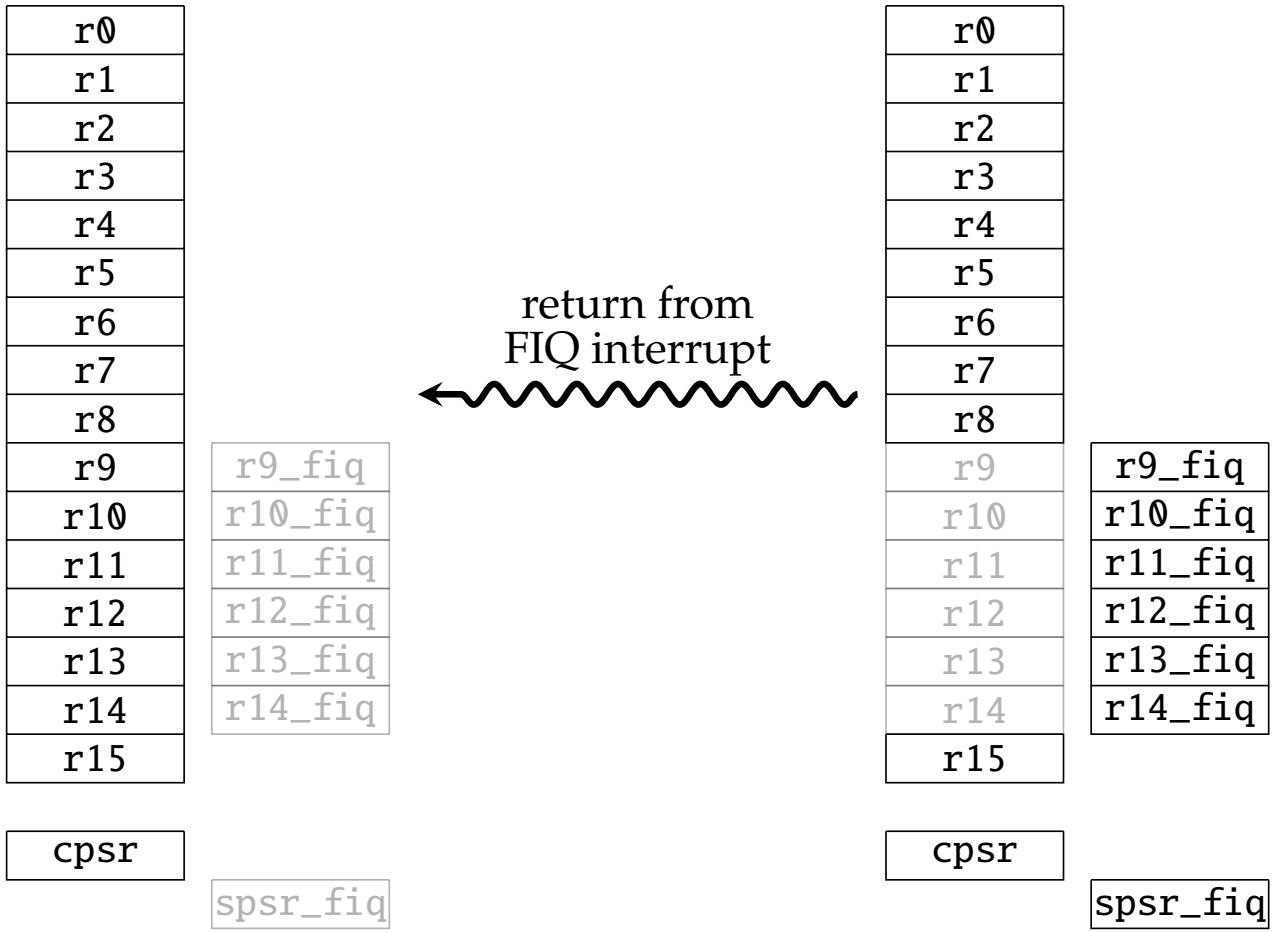
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



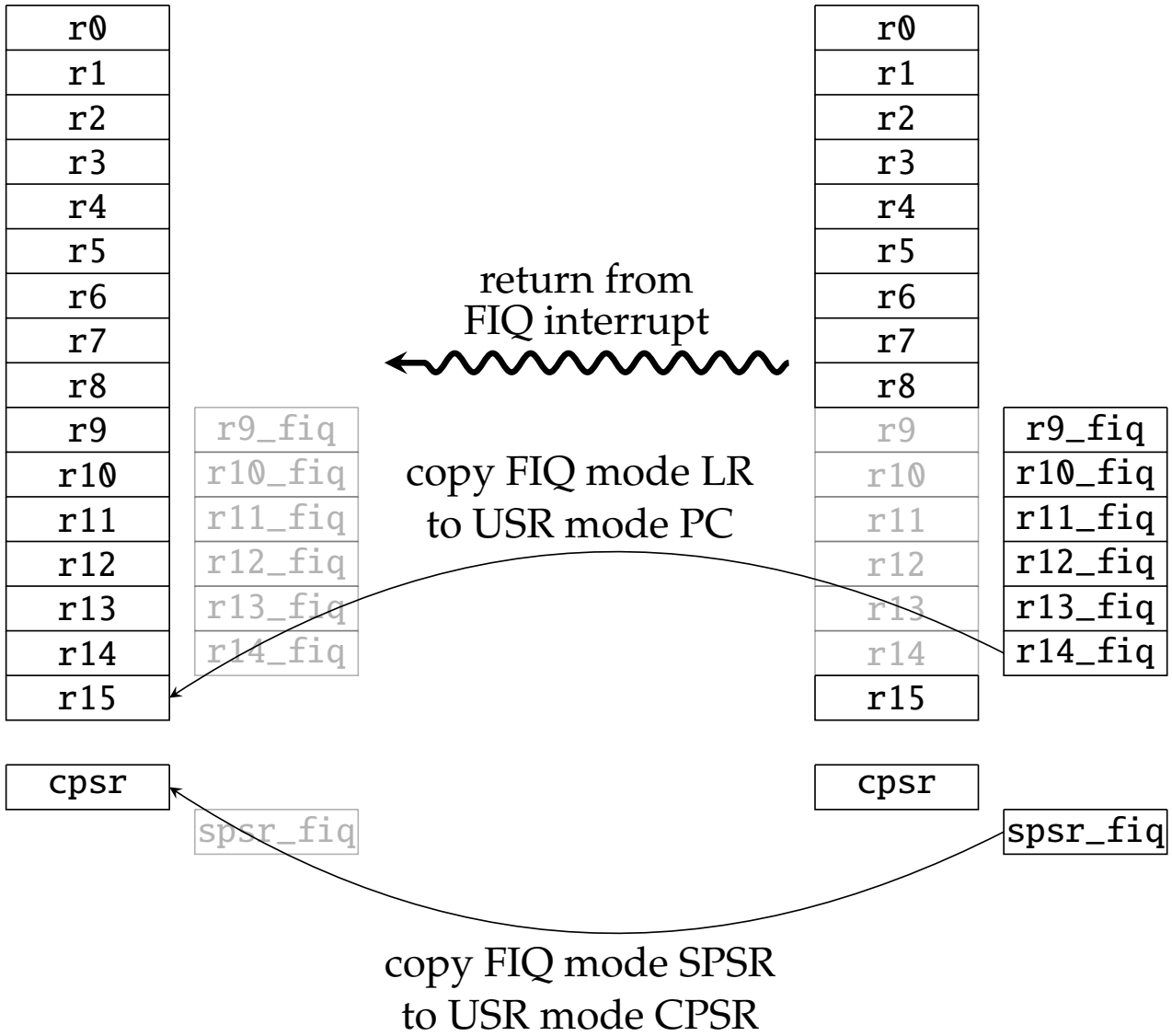
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



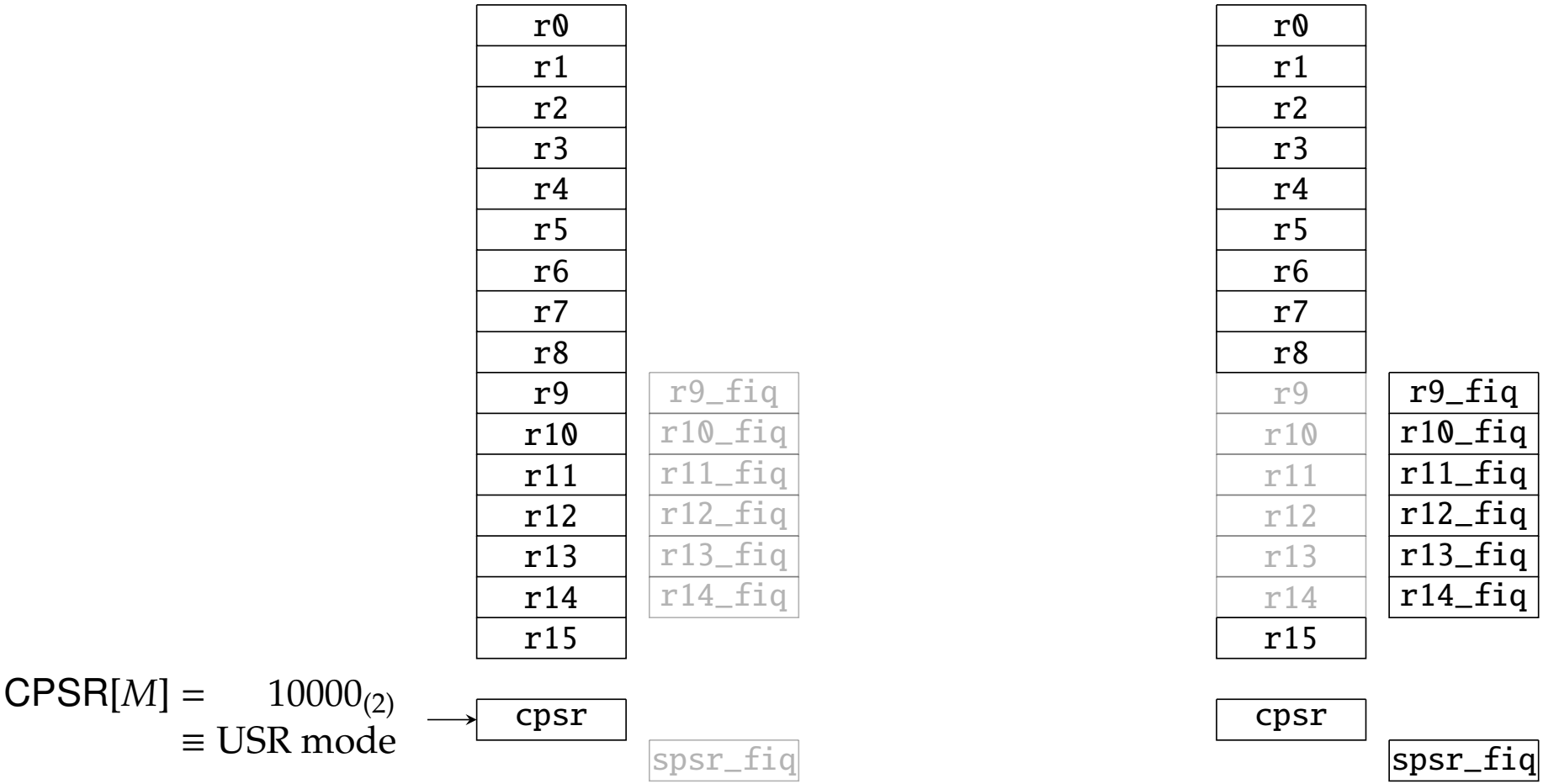
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



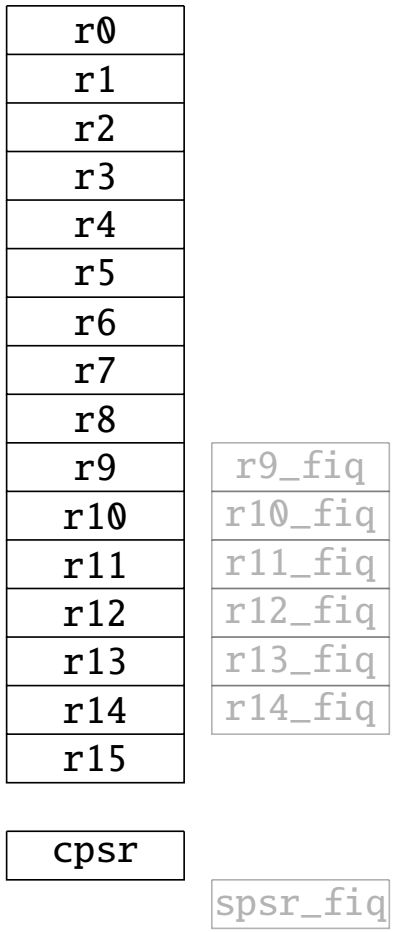
Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

- Example:



Implementation (5) – Cortex-A8: putting it all together [3, Section B1.9.12]

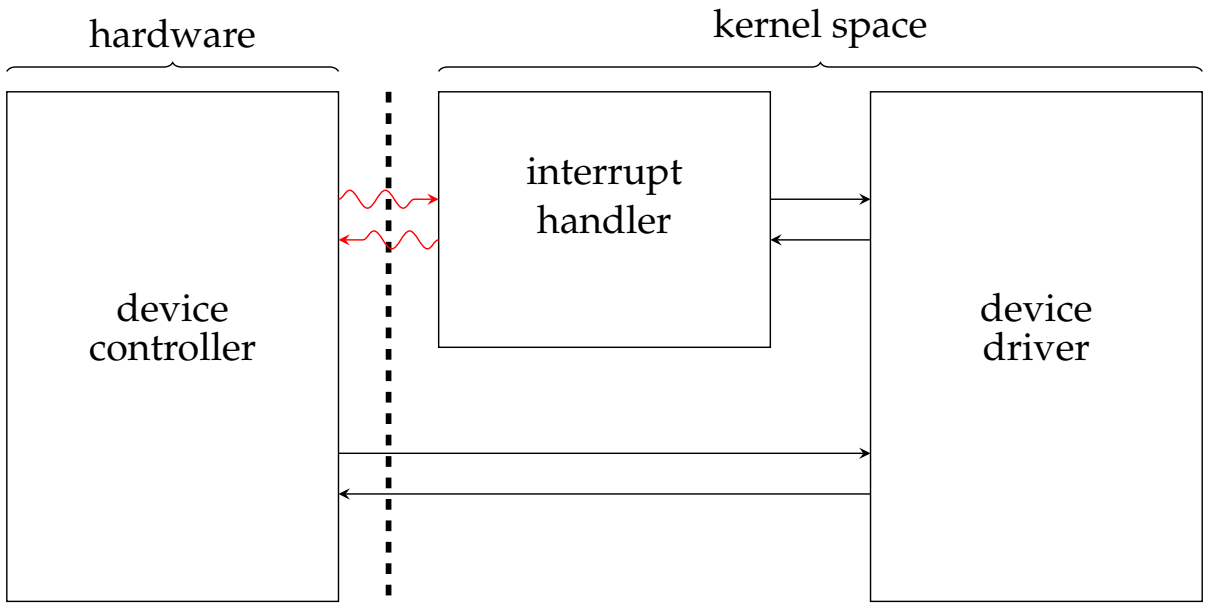
- Example:



Implementation (6) – Cortex-A8: putting it all together [3, Section B1.9.12]

- ... or, more abstractly,
  - a hardware device

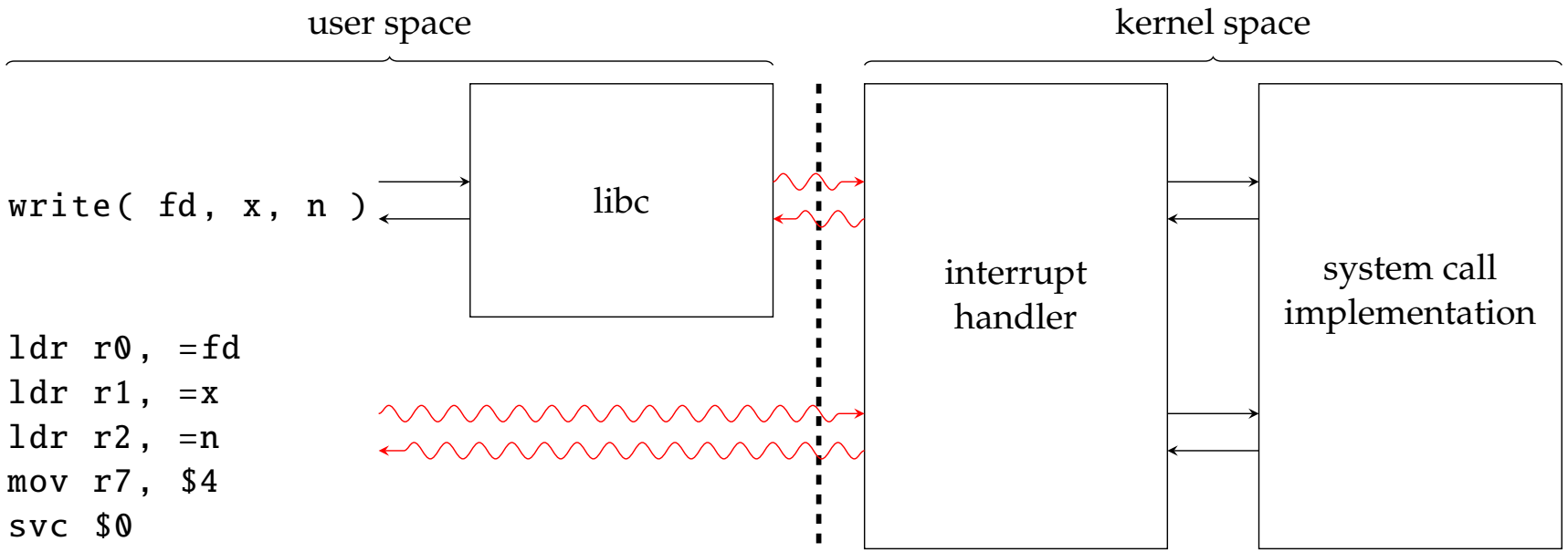
can get attention from (i.e., invoke functionality in) the interrupt-aware kernel, e.g.,



Implementation (6) – Cortex-A8: putting it all together [3, Section B1.9.12]

- ... or, more abstractly,
  - a hardware device, or
  - a user space instruction

can get attention from (i.e., invoke functionality in) the interrupt-aware kernel, e.g.,



the latter case representing a **system call**, which may be

- blocking,
- non-blocking via early abort, or
- non-blocking via asynchronicity.

Continued in next lecture ...

References

[1] ARM Limited.
PrimeCell Vectored Interrupt Controller (PL190) Technical Reference Manual.
Technical Report DDI-0181E, 2004.
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0181e/index.html.

[2] ARM Limited.
Cortex-A8 Technical Reference Manual.
Technical Report DDI-0344K, 2010.
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344k/index.html.

[3] ARM Limited.
ARM Architecture Reference Manual: ARMv7-A and ARMv7-R edition.
Technical Report DDI-0406C, 2014.
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0406c/index.html.

[4] J. Corbet, G. Kroah-Hartman, and A. Rubini.
Chapter 10: Interrupt handling.
In *Linux Device Drivers*. O'Reilly, 3rd edition, 2005.
http://www.makelinux.net/ldd3/.

[5] A. Silberschatz, P.B. Galvin, and G. Gagne.
Chapter 13: I/O systems.
In *Operating System Concepts*. Wiley, 9th edition, 2014.

References

[6] A. N. Sloss, D. Symes, and C. Wright.
Chapter 9: Exception and interrupt handling.
In *ARM System Developer's Guide: Designing and Optimizing System Software*. Elsevier, 2004.

[7] J.E. Smith and A.R. Pleszkun.
Implementing precise interrupts in pipelined processors.
IEEE Transactions On Computers, 37(5):562–573, 1998.

[8] A.S. Tanenbaum and H. Bos.
Chapter 5: Input/output.
In *Modern Operating Systems*. Pearson, 4th edition, 2015.

[9] W. Walker and H.G. Cragon.
Interrupt processing in concurrent processors.
IEEE Computer Journal, 28(6):36–46, 1995.