

The ARM architecture

Matteo SONZA REORDA
Dip. Automatica e Informatica
Politecnico di Torino



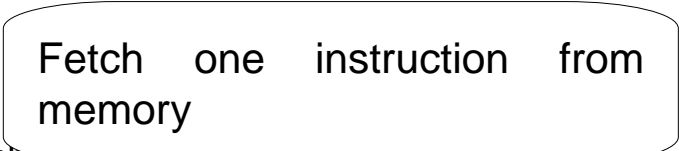
The ARM architecture

- Several ARM processors have been developed and sold.

<u>Core</u>	<u>Architecture</u>
ARM1	v1
ARM2	v2
ARM2aS, ARM3	v2a
ARM6, ARM600, ARM610	v3
ARM7, ARM700, ARM710	v3
ARM7TDMI, ARM710T, ARM720T, ARM740T	v4T
StrongARM, ARM8, ARM810	v4
ARM9TDMI, ARM920T, ARM940T	v4T
ARM9ES	v5TE
ARM10TDMI, ARM1020E	v5TE

3-stage ARM

- This architecture was the one originally developed by Acorn, and employed up to ARM7
- The 3 stages are
 - Fetch
 - Decode
 - Execute
- Some instructions (e.g., those accessing the memory) require more than 3 clock cycles to be executed



Fetch one instruction from memory

- This architecture was the one originally developed by Acorn, and employed up to ARM7
- The 3 stages are
 - Fetch
 - Decode
 - Execute
- Some instructions (e.g., those accessing the memory) require more than 3 clock cycles to be executed

Decode one instruction and generates the required control signals

- This architecture was developed by Acorn and employed up to ARM7
- The 3 stages are
 - Fetch
 - Decode
 - Execute
- Some instructions (e.g., those accessing the memory) require more than 3 clock cycles to be executed

3-stage ARM

- This architecture was the one originally developed by Acorn, and employed up to ARM7
- The 3 stages are
 - Fetch
 - Decode
 - Execute
- Some instructions (e.g., those accessing the memory) require more than 3 clock cycles to be executed

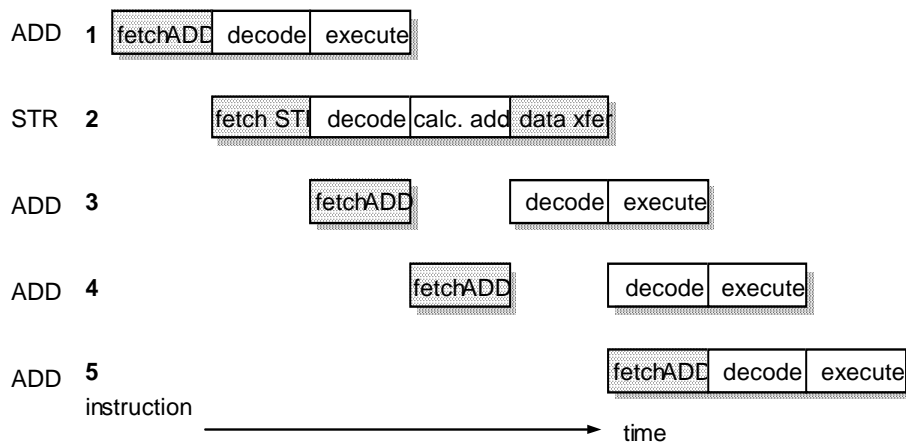
Execute one instruction:

- The operands are read from registers
- One operand is possibly shifted
- The ALU generates the result
- The result is written in the destination register

- This architecture developed by
- The 3 stages
 - Fetch
 - Decode
 - Execute
- Some instructions (e.g., those accessing the memory) require more than 3 clock cycles to be executed

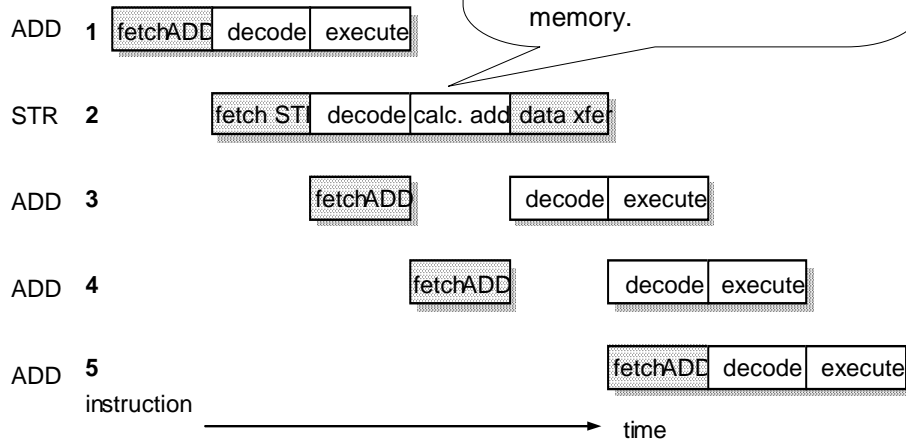
For example, LOAD and STORE instructions require 2 clock cycles for execution (one for address computation, the other for memory access)

Pipeline behavior



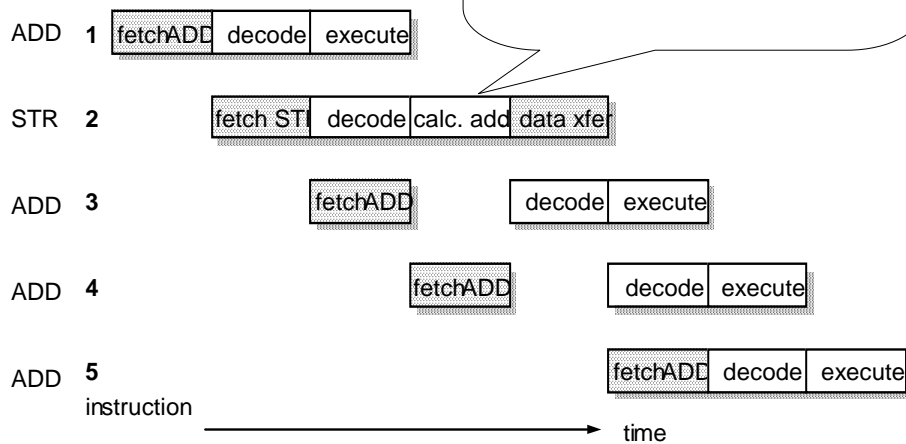
Pipeline

- The STR instruction requires:
- 1 clock cycle to be fetched
 - 1 clock cycle to be decoded
 - 1 clock cycle to compute the memory address
 - 1 clock cycle to access the memory.



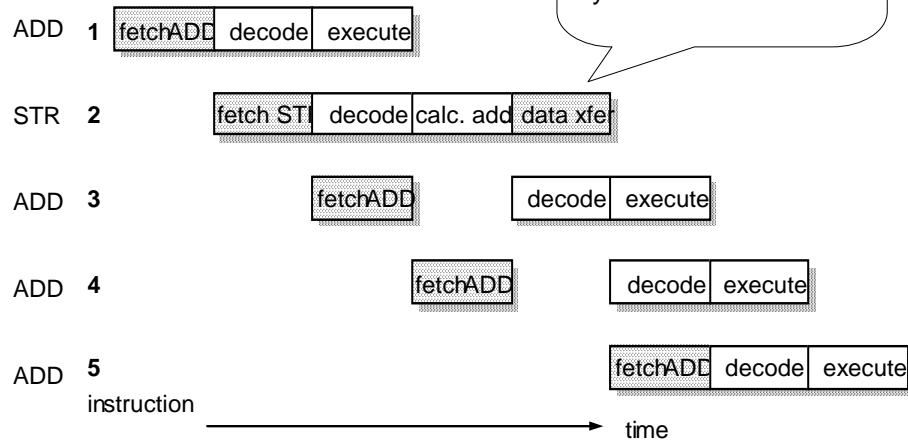
Pipeline

In this clock cycle the decode logic is still busy generating the control signals for accessing the memory in the following cycle.



Pipeline beh

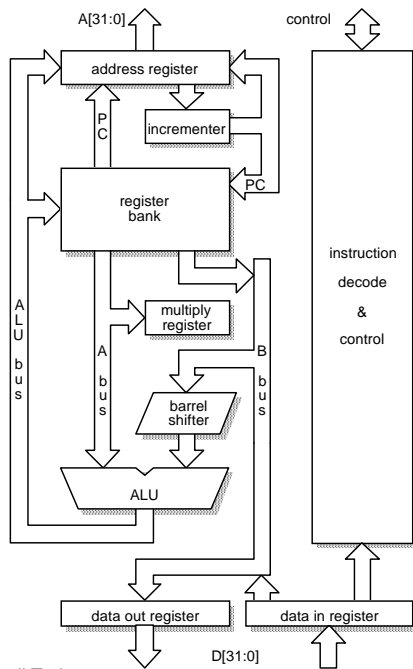
Only one memory access can be executed at any given clock cycle.



Branch instructions

- They always flush and refill the pipeline
- No delayed branch mechanism is supported

Architecture

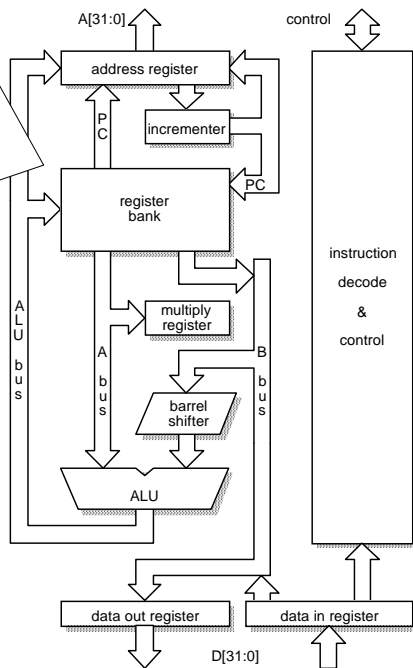


Matteo SONZA REORDA

Politecnico di Torino

13

It has two read ports and one write port. One additional read port and one additional write port are reserved for r15.



Matteo SONZA REORDA

Politecnico di Torino

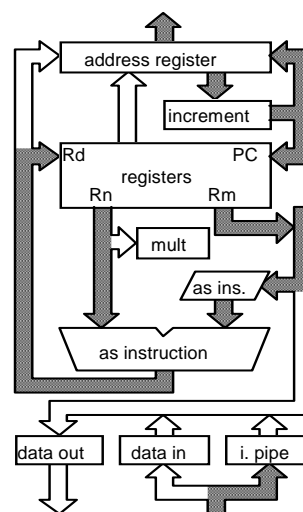
14

PC access

- If an instruction accesses the PC during the Execute stage, it reads a value which is incremented by 8 with respect to its proper address

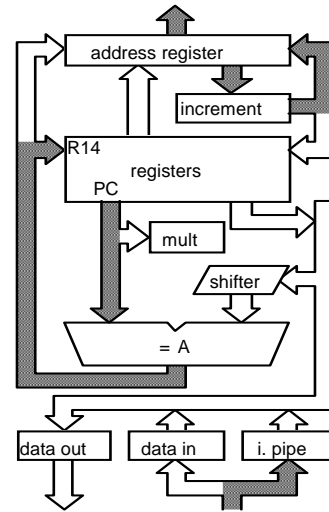
Data processing reg-reg instruction execution

- Instruction *i* is executed:
 - Two operands are read from registers *Rn* and *Rm*
 - One operand is possibly rotated
 - The ALU generates the result
 - The result is written to register *Rd*
 - A further instruction is fetched from memory
 - The PC is updated



Branch and link instructions

- In this case, a further clock cycle is required (while the pipeline is refilled) to save the return address in r14



(b) 2nd cycle - save return addr e21

Matteo SONZA REORDA

Politecnico di Torino

Branch and link instructions

- They require a third clock cycle to adjust the saved address, so that it holds the address of the following instruction

Matteo SONZA REORDA

Politecnico di Torino

22

Instruction speed

- Most instructions do require 3 clock cycles to execute
- Some instructions do require additional clock cycles
- The additional clock cycles may depend on specific values of the operands

Instruction speed

Instruction	Cycle count	Additional
Data Processing	1S	+ 1I for SHIFT(Rs) + 1S + 1N if R15 written
MSR, MRS	1S	
LDR	1S + 1N + 1I	+ 1S + 1N if R15 loaded
STR	2N	
LDM	nS + 1N + 1I	+ 1S + 1N if R15 loaded
STM	(n-1)S + 2N	
SWP	1S + 2N + 1I	
B,BL	2S + 1N	
SWI, trap	2S + 1N	
MUL,MLA	1S + mI	
CDP	1S + bI	
LDC,STC	(n-1)S + 2N + bI	
MCR	1N + bI + 1C	
MRC	1S + (b+1)I + 1C	

5-stage ARM

- The new architecture was adopted starting from ARM9 (1995)
- It uses separate data and code memories (i.e., caches)
- The 5 stages are
 - Fetch
 - Decode
 - Execute
 - Buffer/data
 - Write-back
- The higher number of stages allows for a faster clock

Matteo SONZA REORDA

Politecnico di Torino

25

Fetch one instruction from memory

- The new architecture was adopted starting from ARM9
- It uses separate data and code memories (i.e., caches)
- The 5 stages are
 - Fetch
 - Decode
 - Execute
 - Buffer/data
 - Write-back
- The higher number of stages allows for a faster clock

Matteo SONZA REORDA

Politecnico di Torino

26

- The new ARM9
- It uses separate data and code memories (i.e., caches)
- The 5 stages are
 - Fetch
 - Decode
 - Execute
 - Buffer/data
 - Write-back
- The higher number of stages allows for a faster clock

Decode the instruction and read the operands (up to 3) from registers

- The new ARM9
- It uses separate data and code memories (i.e., caches)
- The 5 stages are
 - Fetch
 - Decode
 - Execute
 - Buffer/data
 - Write-back
- The higher number of stages allows for a faster clock

- One operand is possibly shifted
- The ALU generates the result (or the address if a load/store instruction is considered)

5-stage ARM

- The new ARM9
- It uses separate caches)
- The 5 stages are
 - Fetch
 - Decode
 - Execute
 - Buffer/data
 - Write-back
- The higher number of stages allows for a faster clock

Memory is accessed.
Only load/store instructions perform useful operations in this stage.

5-stage ARM

- The new ARM9
- It uses separate caches)
- The 5 stages are
 - Fetch
 - Decode
 - Execute
 - Buffer/data
 - Write-back
- The higher number of stages allows for a faster clock

Results (including values from memory) are written to the result register.

PC value

- Thanks to suitable connections, each instruction still reads a value from r15 which is incremented by 8 with respect to its address.

Program compatibility

- 5-stage processors can execute the same binary code executed by 3-stage processors
- The same code can cause different stall situations in the 3-stage or 5-stage architectures.

The ARM coprocessors

- The ARM instruction set can be extended by adding external coprocessors (up to 16)
- If coprocessors are absent, the corresponding instructions can be emulated in software through undefined instruction traps.

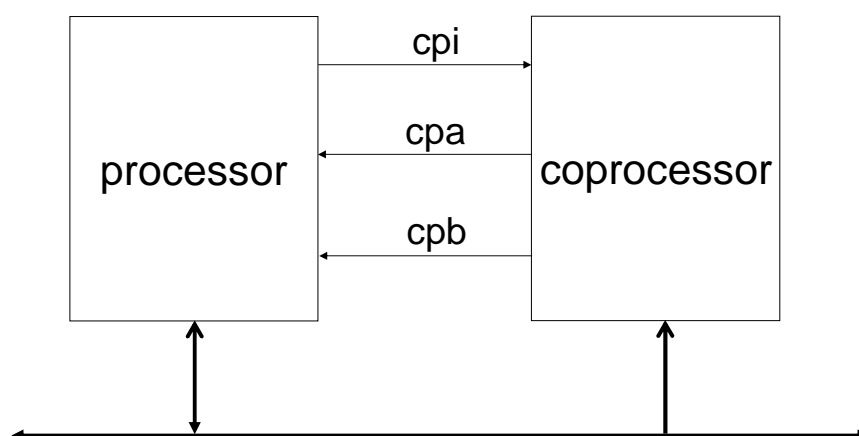
Coprocessor architecture

- Each coprocessor
 - May have up to 16 registers of any size
 - Has a load-store architecture, including instructions to move data
 - From one internal register to another
 - From one internal register to a ARM one, and viceversa
 - From one internal register to memory, and viceversa.

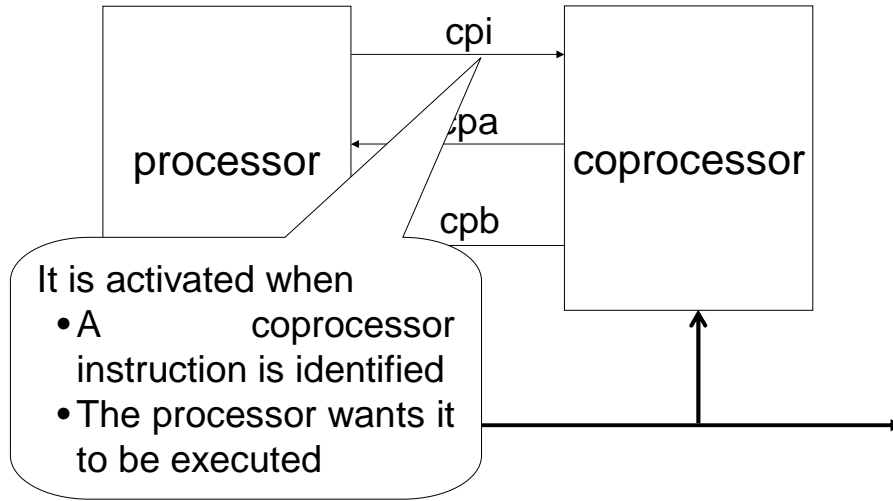
The ARM7TDMI coprocessor

- It is connected to the same data bus of the processor
- It continuously monitors the content of the bus, and fetches every instruction
- It contains an internal pipeline that mimics that in the processor.

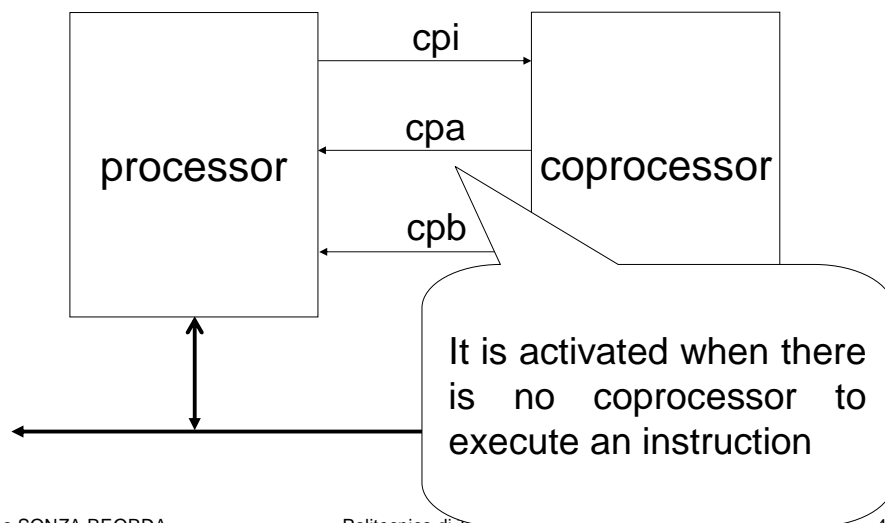
Processor-coprocessor interface

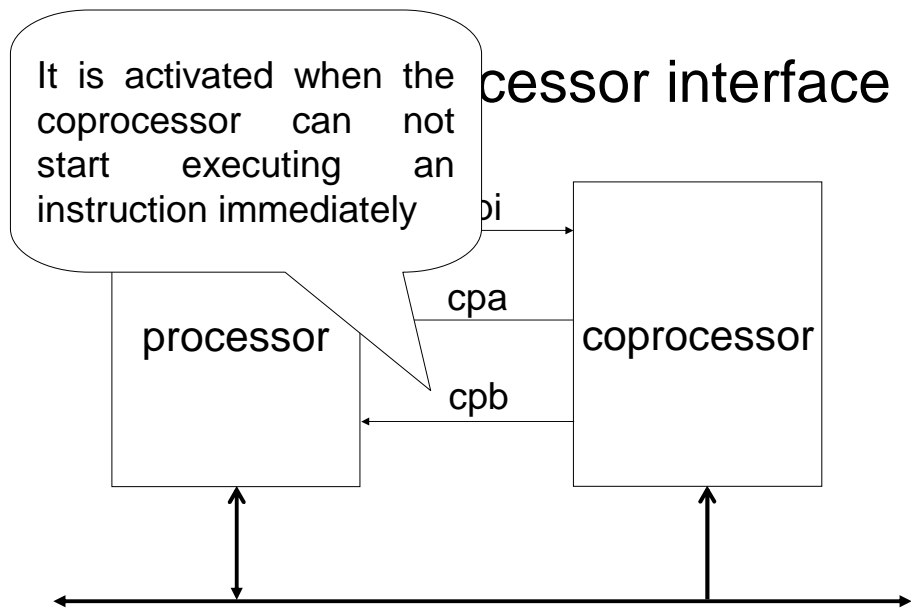


Processor-coprocessor interface



Processor-coprocessor interface





Processor-coprocessor handshaking

- Once an instruction has entered the processor and coprocessor pipelines, one out of the four following possibilities may arise:
 - The ARM may decide not to execute it (e.g., because it is conditioned, and the condition was false). *cpi* is not activated. Both the processor and the coprocessor will discard the instruction.

Processor-coprocessor handshaking

- Once an instruction has entered the processor and coprocessor pipelines, one out of the four following possibilities may arise:
 - The ARM may decide to execute it (*cpi* is activated), but the corresponding coprocessor does not exist (*cpa* remains active). The undefined instruction trap is triggered, and the instruction is possibly emulated.

Processor-coprocessor handshaking

- Once an instruction has entered the processor and coprocessor pipelines, one out of the four following possibilities may arise:
 - The ARM decides to execute it (*cpi* is activated), the corresponding coprocessor exists (*cpa* is deactivated), but it is not ready to start execution (*cpb* is active). The processor starts waiting, stalling the instruction stream. Only interrupt requests are served.

Processor-coprocessor handshaking

- Once an instruction has entered the processor and coprocessor pipelines, one out of the four following possibilities may arise:
 - The ARM decides to execute it (*cpi* is activated), the corresponding coprocessor exists (*cpa* is deactivated), and it is ready to immediately start execution (*cpb* is deactivated). The instruction starts its execution.

Data transfers

- Since coprocessors are not connected to the address bus, when memory must be accessed by a coprocessor instruction, the processor generates addresses.

The Thumb Instruction Set

- Some of the ARM processors (those with a T in the acronym) support the Thumb instruction set (together with the standard ARM instruction set)
- In the Thumb instruction set
 - Instructions are encoded on 16 bits
 - Instructions are less powerful
 - Instructions are less.

Thumb instruction usage

- Encoding an algorithm in Thumb instructions
 - Requires more instructions, but less code memory
 - Results in slower execution, but requires less power.
- Thumb instructions are therefore used for low-cost, low performance applications.

Thumb-ARM differences

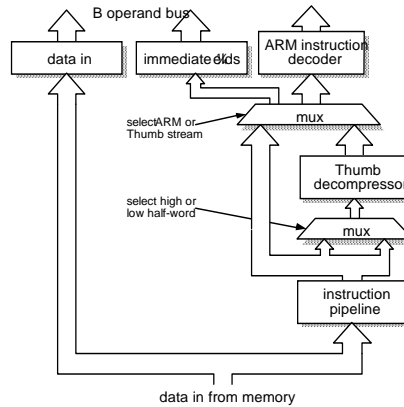
- Most Thumb instructions are executed unconditionally
- Many Thumb data processing instructions use a 2-address format (the destination register is the same as one of the source registers)
- Thumb instruction formats are less regular than ARM instruction formats.

The T bit

- The mechanism to switch to/from Thumb instructions is driven by the T bit in the CPSR:
 - If T=1, the processor interprets the fetched code as a sequence of Thumb instructions
 - If T=0, the processor interprets the fetched code as a sequence of usual ARM instructions.
- The value of T can be changed via software.

Thumb implementation

- The Thumb instruction set requires some additional logic to translate Thumb instructions into ARM instructions.
- This operation is performed in the decode stage, without significant effects on performance.



Privileged modes

- ARM processors may work
 - In the user mode
 - In a privileged mode
- Privileged modes are used to handle exceptions and supervisor calls (i.e., software interrupts)
- The current operating mode is defined by the lower 5 bits of the CPSR
- When a privileged mode is entered, the current value of the CPSR is saved in the SPSR (Saved Program Status Register)

Operating modes

CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

Exceptions

- Exceptions include interrupts (from the outside), traps and supervisor calls
- They may be categorized in 3 groups:
 - Exceptions having a direct effect of an instruction:
 - Software interrupts
 - Undefined instructions
 - Prefetch abort (i.e., memory fault during fetch)
 - Exceptions that are a side-effect of an instruction
 - Data aborts (i.e., memory fault during a load/store data access)
 - Exceptions generated externally
 - Reset
 - IRQ
 - FIQ

Exceptions management

- When an exception arises, the processor
 - Completes the current instruction (as best as it can)
 - Changes the operating mode to the appropriate one
 - Saves the address of the following instruction to r14
 - Saves the previous value of CPSR in SPSR
 - Disables interrupts on IRQs by setting bit 7 of CSR
 - Disables fast interrupts on FIQ by setting bit 6 of CSR, if the exception is a fast interrupt
 - Forces the PC to a value between 00_{16} and $1C_{16}$, depending on the exception type

The vector address

- Locations from 00_{16} to $1C_{16}$ are called *vector address*, and usually contain branches to exception handlers
- The FIQ code, being the last, may start immediately

Exception vector addresses

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Privileged mode registers

- Each privileged mode owns two special registers, which are used to
 - Save the return address
 - Holds a stack pointer

Return from exceptions

- When the exception procedure ends, the handler code should restore the user state exactly as it was when the exception arose
- This means
 - Restoring the modified user registers
 - Restoring the CPSR from the appropriate SPSR
 - Restoring the PC to the next instruction in the user stream

Re To speed-up service routine activation, FIQ mode has additional private registers.

- When the handler code should restore the user state exactly as it was when the exception arose
- This means
 - Restoring the modified user registers
 - Restoring the CPSR from the appropriate SPSR
 - Restoring the PC to the next instruction in the user stream

Return from exceptions

- When the exception procedure ends, the handler must be performed exactly in the reverse order of the operations that were performed during the exception. The last two operations must be performed at the same time
- This means:
 - Restoring the modified user registers
 - Restoring the CPSR from the appropriate SPSR
 - Restoring the PC to the next instruction in the user stream

Return from exceptions

- In case the return address is in r14, the following instructions can be used
 - To return from a SWI or undefined instruction trap:
`MOVS pc, r14`
 - To return from an IRQ, FIQ, or prefetch abort:
`SUBS pc, r14, #4`
 - To return from a data abort to retry the data access:
`SUBS pc, r14, #8`

Return

The S suffix identifies a special form of the instruction, which restores the CPSR, too

- In case the return instruction is used following instructions that use the PC, the following instructions should be used:
 - To return from a SWI or undefined instruction trap:
`MOVS r14, pc`
 - To return from an IRQ, FIQ, or prefetch abort:
`SUBS r14, pc, #4`
 - To return from a data abort to retry the data access:
`SUBS r14, pc, #8`

Return from exceptions

- In case the return instruction is used following instructions that use the PC, the following instructions should be used:
 - To return from a SWI or undefined instruction trap:
`MOVS r14, pc`
 - To return from an IRQ, FIQ, or prefetch abort:
`SUBS r14, pc, #4`
 - To return from a data abort to retry the data access:
`SUBS r14, pc, #8`

Return must be performed one instruction early

Return from exceptions

- In case the return address is in the stack, the following instructions can be used
 - To return from a trap:


```
MOVS    pc, r14
```
 - To return from an IRQ, or prefetch abort:


```
SUBS    pc, r14, #4
```
 - To return from a data abort to retry the data access:


```
SUBS    pc, r14, #8
```

Return must be performed two instructions early

Return from exceptions

- In case the return address is in the stack, the following instruction can be used

```
LDMFD  r13!, {r0-r3,pc}^
```

This character identifies a special form of the instruction, which restores the CPSR, too

Exception priorities

- If multiple exceptions arise at the same time, the following priorities are used
 - Reset (highest priority)
 - Data abort
 - FIQ
 - IRQ
 - Prefetch abort
 - SWI and undefined instruction