ARM Processor

An INTRO…

Launch Of ARM

- ❖ Founded in November 1990
  - – Spun **out** of Acorn Computers

- ❖ Designs the ARM range of RISC processor cores
- ❖ Licenses ARM core designs to semiconductor partners who fabricate and sell to their customers.
  - ARM does not fabricate silicon itself
- ❖ Also develop technologies to assist with the design-in of the ARM architecture
  - – Software tools, boards, debug hardware, application software, bus architectures, peripherals etc
- ❖ *Used in computationally more involved applications*
- ❖ *Low power and cost sensitive embedded application*

ARM Partnership Model

ARM Powered Products

RISC v/s CISC

RISC

- Simple instructions, few in number
- Fixed length instructions
- Complexity in compiler
- Only **`LOAD/STORE`** instructions access memory
- Few addressing modes

CISC

- Many complex instructions
- Variable length instructions
- Complexity in microcode
- Many instructions can access memory
- Many addressing modes

ARM Features

The ARM is a *32-bit* architecture.

When used in relation to the ARM:

- **Byte** means 8 bits
- **Half** means 16 bits (two bytes)
- **Word** means 32 bits (four bytes)

Most ARM's implement two instruction sets

- 32-bit ARM Instruction Set
- 16-bit Thumb Instruction Set

Jazelle cores can also execute Java bytecode

ARM Features

Based upon RISC Architecture with enhancements to meet requirements of embedded applications

- ❖ Load-store architecture, where data processing operations operate on register contents only.
- ❖ A large uniform register file
- ❖ Uniform and fixed length instructions only
- ❖ 32 bit processor
- ❖ instructions are 32 bit long
- ❖ Good speed / power consumption
- ❖ High code density

ARM Architecture Versions

Version 1

26 bit addressing , no multiply or coprocessor.

Version2

Includes 32 bit result multiply coprocessor.

Version3

32 bit addressing.

Version4

add signed, unsigned half-word and signed byte load and store instructions

Version4T

16 bit Thumb compressed form of instruction introduced

ARM Architecture Versions

Version5T

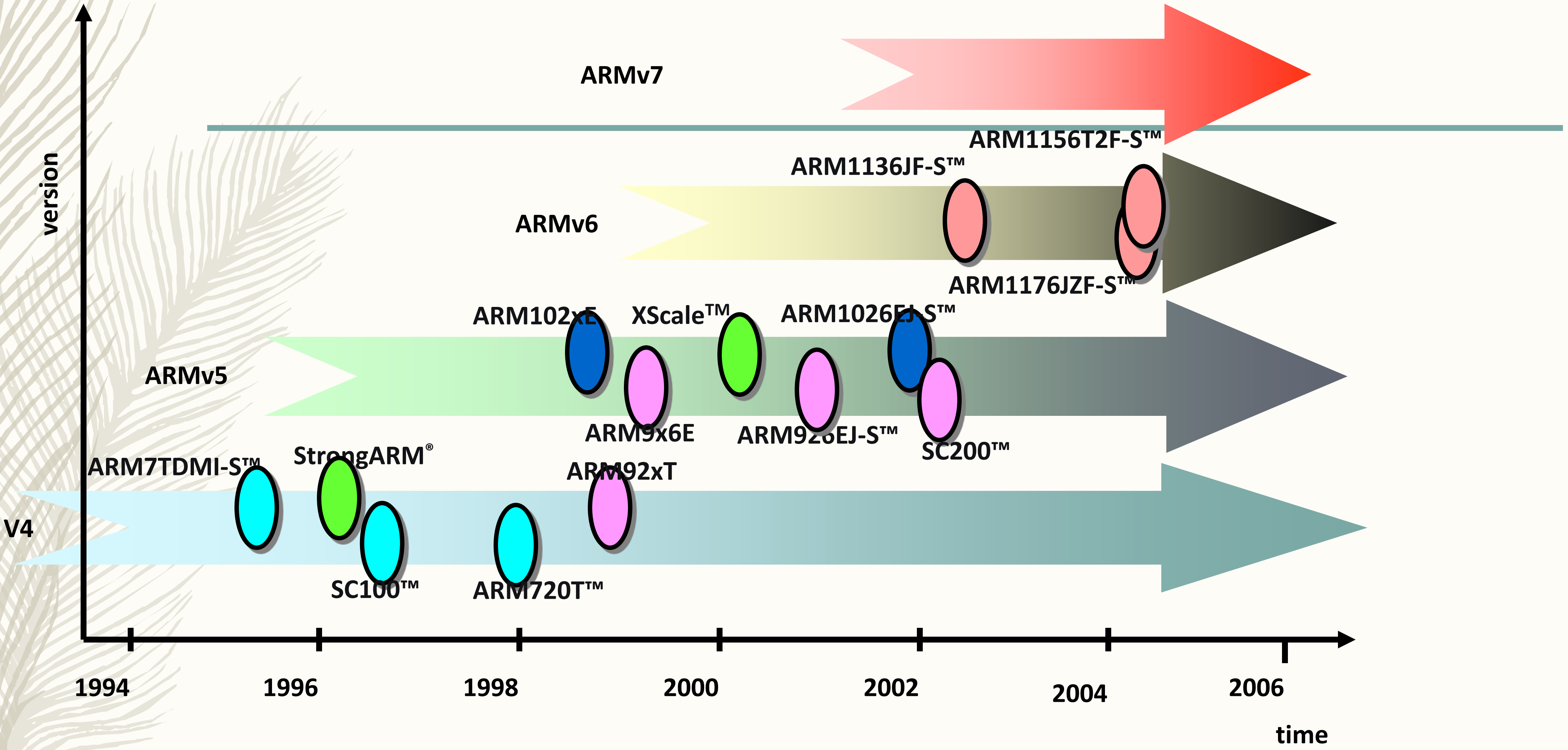
Superset of 4T adding new instructio

Version 5TE

Add signal processing signal extension

Version	Family
ARMv1	ARM1
ARMv2	ARM2, ARM3
ARMv3	ARM6, ARM7
ARMv4	Strong ARM, ARM7TDMI, ARM9TDMI
ARMv5	ARM7EJ, ARM9E, ARM10XE
ARMv6	ARM11
ARMv7	Cortex

Architecture Revisions



XScale is a trademark of Intel Corporation

Highlights of ARM Development

- ❖ The introduction of the novel compressed instruction format called **'Thumb'** which reduces cost and power dissipation in small systems;
- ❖ Significant steps upwards in performance with the ARM9, ARM 10 and 'Strong-ARM' processor families;
  - ✓ A state-of-the-art software development and debugging environment;
  - ✓ A very wide range of embedded applications based around ARM processor cores.
  - ✓ Harvard Architecture

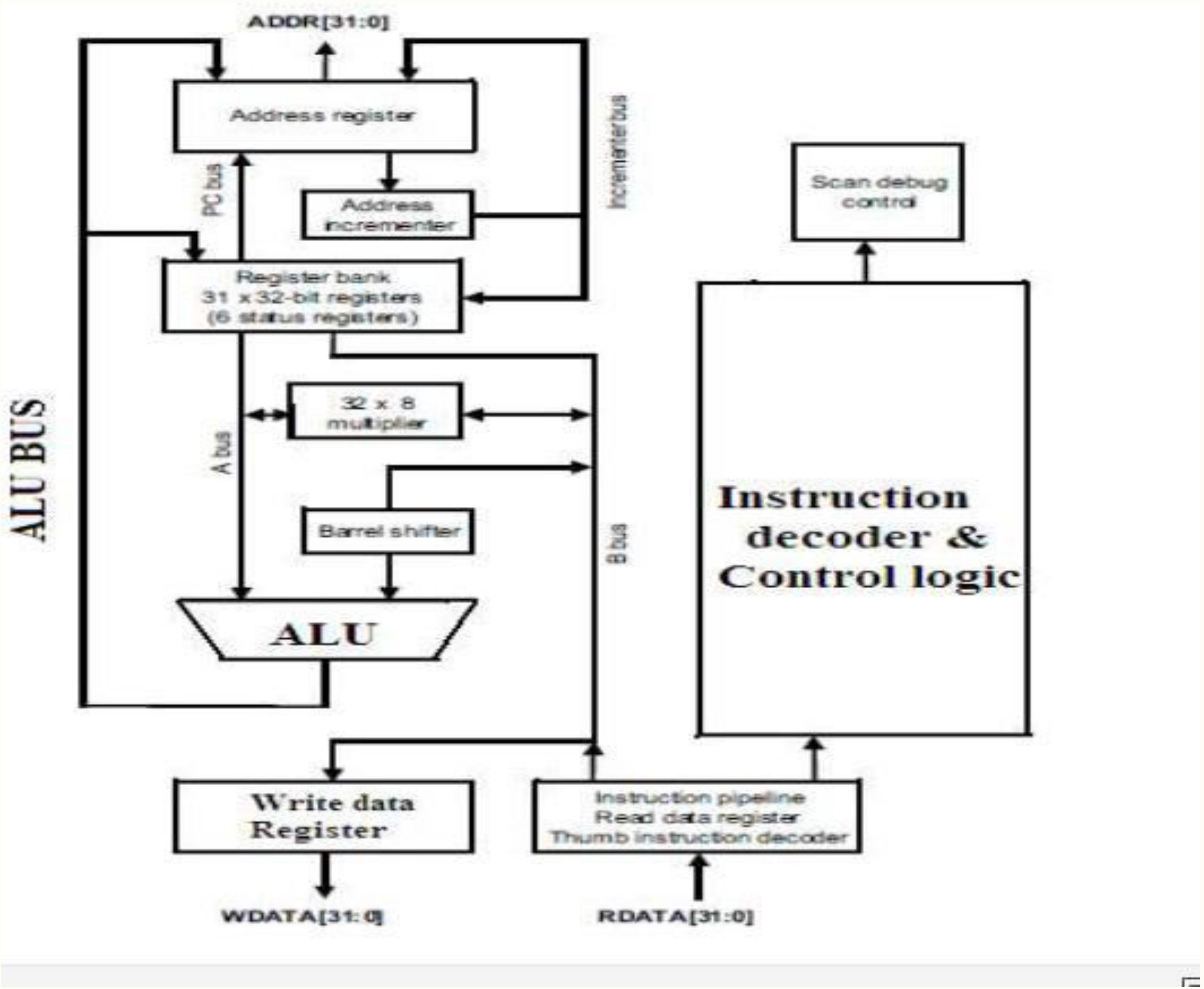
Enhanced RISC Features

- ❖ Control over ALU and shifter for every data processing operations to maximize their usage.
- ❖ Auto increment and decrement addressing modes to optimize program loops
- ❖ Load and store multiple instructions to maximize data throughput.
- ❖ Conditional execution of instruction to maximize execution throughput.

ARM7TDMI

- ❖ Belongs to ARM family of general-purpose 32-bit microprocessors.
- ❖ Von Neumann architecture, with a single 32-bit data bus carrying both instructions and data.
- ❖ ARM uses the Advanced Microcontroller Bus Architecture (AMBA) bus architecture.
  - ✓ This AMBA include two system buses: the AMBA High-Speed Bus (AHB) or the Advanced System Bus (ASB), and the Advanced Peripheral Bus (APB)

ARM7TDMI Block Diagram



Block Diagram

- The ARM processor consists of :
  - Arithmetic Logic Unit (32-bit)
  - One Booth multiplier(32-bit)
  - One Barrel shifter
  - One Control unit
  - Register file of 37 registers each of 32 bits

Overview : Core Data Path

- ❖ Data Items are placed in Register File
- ❖ No Data Processing Instructions directly manipulate Data in memory
- ❖ Instructions Typically use two Source Registers & single result or Destination Registers
- ❖ A Barrel Shifter on the data path can pre-process data before it enters into ALU
- ❖ Increment/Decrement Logic can update the Register Content for Sequential access independent of ALU

Memory

- Viewed as a large, single-dimension array, with an address.
- A memory address is an index into the array.
- "Byte addressing" means that the index points to a byte of memory.

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data
.	. . .
.	. . .

Memory Organization

- Bytes are nice, but most data items use larger "words"
- For ARM, a word is 32 bits or 4 bytes.

- 2^{32} bytes with byte addresses from 0 to $2^{32} - 1$
- 2^{30} words with byte addresses 0, 4, 8, ... $2^{32} - 4$
- Words are aligned

  i.e., what are the least 2 significant bits of a word address?
- Standard little-endian organization

0	32 bits of data
4	32 bits of data
8	32 bits of data
12	32 bits of data
.	...
.	...

Load-Store Architecture

- ❖ Instruction set will operate only on registers.

- ❖ Only memory access:
  - ✓ Copy memory values to registers (load) read
  - ✓ Copy register values to memory (store) write

- ❖ Unlike in CISC processors, memory-to-memory operations are not supported.

Instruction categories

- ❖ Data processing instructions
  - ✓ *Use and change register values.*
- ❖ Data transfer instructions
  - ✓ *Load and store*
- ❖ Control flow instructions
  - ✓ *Execution switching*

Registers

- ARM has 37 registers all of which are 32-bits long.
  - ✓ 1 dedicated program counter
  - ✓ 1 dedicated current program status register
  - ✓ 5 dedicated saved program status registers
  - ✓ 30 general purpose registers

Registers(1)

- The current processor mode governs which of several banks is accessible. Each mode can access
  - ✓ a particular set of r0-r12 registers
  - ✓ a particular r13 (the stack pointer, sp) and r14 (the link register, lr)
  - ✓ the program counter, r15 (pc)
  - ✓ the current program status register, cpsr

- Privileged modes (except System) can also access
  - ✓ a particular spsr (saved program status register)

Registers(2)

- General purpose registers hold either data or address.
- All registers are of 32 bit
- In user mode, 16 data registers and 2 status registers are visible.
- Data registers :r0 to r15

Registers(3)

- Depending upon context, r13 and r14 can also be used as GPR.
- Any instruction which use r0 can be used with any other GPR(r1-r13).
- PC (r15) value is stored in bits [31:2] with [1:0] bits undefined

Special function registers

- **PC** (R15): Program Counter. Any instruction with PC as its destination register is a program branch
- **LR** (R14): Link Register. Saves a copy of PC when executing the BL instruction (subroutine call) or when jumping to an exception or interrupt routine

  - It is copied back to PC on the return from those routines
- **SP** (R13): Stack Pointer. There is **no stack** in the ARM architecture. Even so, R13 is usually reserved as a pointer for the program-managed stack

Special function registers(2)

- **CPSR** : Current Program Status Register. Holds the visible status register
- **SPSR** : Saved Program Status Register. Holds a copy of the previous status register while executing exception or interrupt routines

  - It is copied back to CPSR on the return from the exception or interrupt

  - No SPSR available in User or System modes

Program counter (r15)

- When the processor is executing in ***ARM state***:
  - ✓ All instructions are 32 bits wide
  - ✓ All instructions must be word aligned

- When the processor is executing in ***Thumb state***:
  - ✓ All instructions are 16 bits wide
  - ✓ All instructions must be halfword aligned

- When the processor is executing in ***Jazelle state***:
  - ✓ All instructions are 8 bits wide
  - ✓ Processor performs a word access to read 4 instructions at once

Processor Modes

The ARM has seven basic operating modes:

- ❖ User: unprivileged mode under which most tasks run
- ❖ FIQ: entered when a high priority (fast) interrupt is raised
- ❖ IRQ: entered when a low priority (normal) interrupt is raised
- ❖ Supervisor : entered on reset and when a Software Interrupt instruction is executed
- ❖ Abort: used to handle memory access violations
- ❖ Undef : used to handle undefined instructions
- ❖ System: privileged mode using the same registers as user mode

Processor Modes

- ❖ Processor Modes determines
  - ✓ Which registers are Active
  - ✓ Access Rights to CPSR Register itself
- ❖ Each Processor Mode is either
  - ✓ **Privileged :**
    - Full Read-Write access to the CPSR
  - ✓ **Non-Privileged :**
    - Only Read access to the Control Field of
    - CPSR but Read-Write access to the Condition Flags

Processor Modes (2)

ARM has Seven Modes

- ❖ **Privileged :**
  - Abort, Fast Interrupt Request (FIQ), Interrupt Request (IRQ), Supervisor, System & Undefined
- ❖ **Non-Privileged :**
  - User

    User Mode is used for Programs and Applications

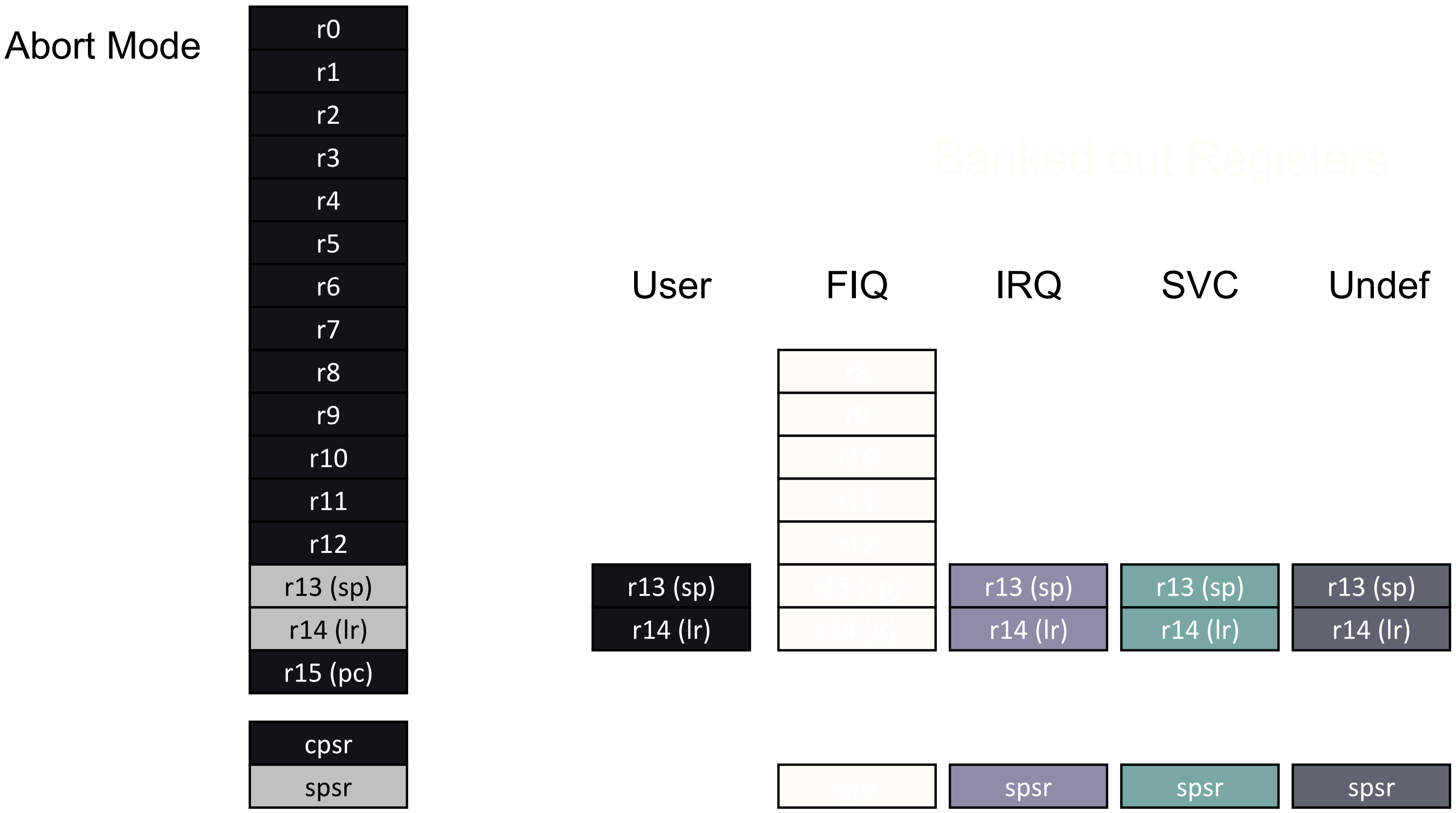
Privileged Modes

- **Abort :**
  - When there is a failed attempt to access memory
- **Fast Interrupt Request (FIQ) & Interrupt Request :**
  - Correspond to Interrupt levels available on ARM
- **Supervisor Mode :**
  - State after Reset and generally the mode in which OS kernel executes

Privileged Modes (2)

- **System Mode :**
  - Special Version of User Mode that allows Full Read Write access of CPSR
- **Undefined :**
  - When the Processor encounters and Undefined Instruction

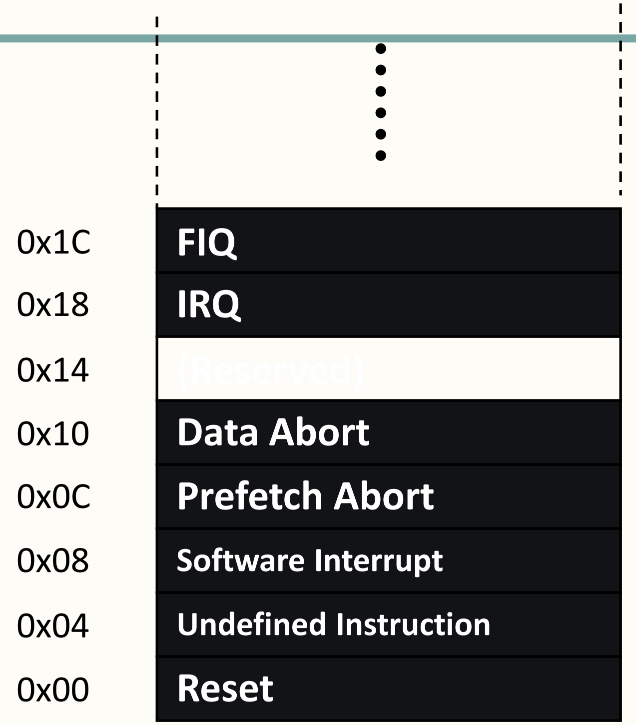
The ARM Register Set



Exception Handling

- ❖ When an exception occurs, the ARM:
  - ✓ Copies CPSR into SPSR_<mode>
  - ✓ Sets appropriate CPSR bits
    - *Change to ARM state*
    - *Change to exception mode*
    - *Disable interrupts (if appropriate)*
  - ✓ Stores the return address in LR_<mode>
  - ✓ Sets PC to vector address
- ❖ To return, exception handler needs to:
  - ✓ Restore CPSR from SPSR_<mode>
  - ✓ Restore PC from LR_<mode>

This can only be done in ARM state.



Vector table can be at `0xFFFF0000` on ARM720T and on ARM9/10 family devices

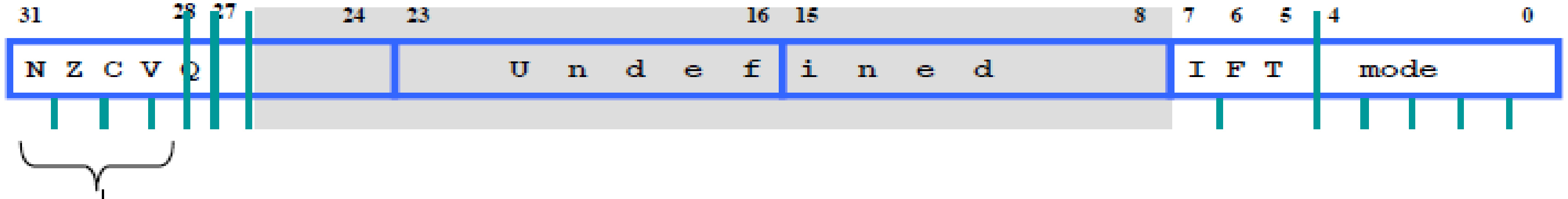
Program Status Registers

31	28	27	24	23	16	15	8	7	6	5	4	0
N Z C V		Q	J	n d e f i		n e d		I	F	T	mode	

- Condition code flags
  - N = **N**egative result from ALU
  - Z = **Z**ero result from ALU
  - C = ALU operation arried out
  - V = ALU operation o erflowed
- Sticky Overflow flag - Q flag
  - Architecture 5TE/J only
  - Indicates if saturation has occurred
- J bit
  - Architecture 5TEJ only
  - J = 1: Processor in Jazelle state
- Interrupt Disable bits.
  - I = 1: Disables the IRQ.
  - F = 1: Disables the FIQ.
- T Bit
  - Architecture xT only
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- Mode bits
  - Specify the processor mode

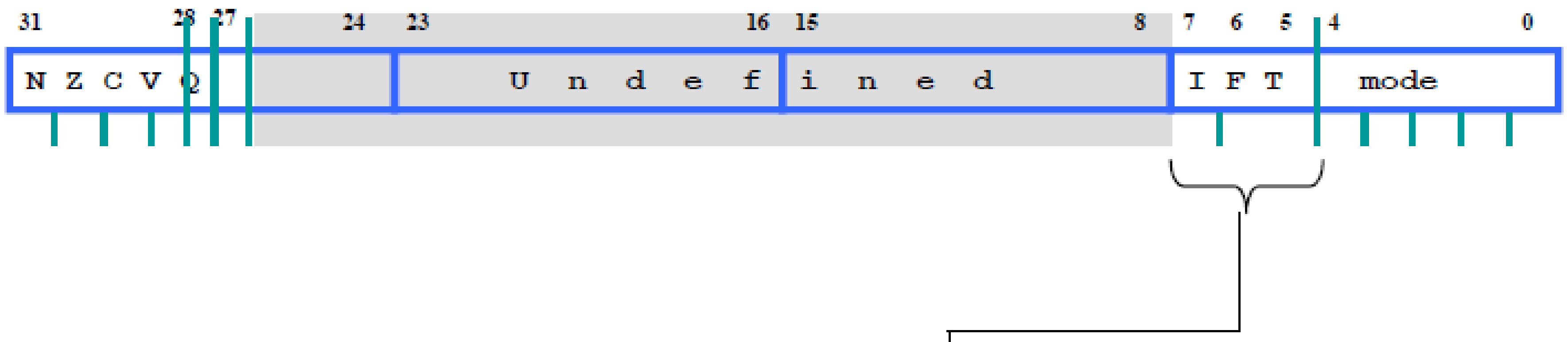
Program Status Register

CPSR : Monitors & Control Internal Operations



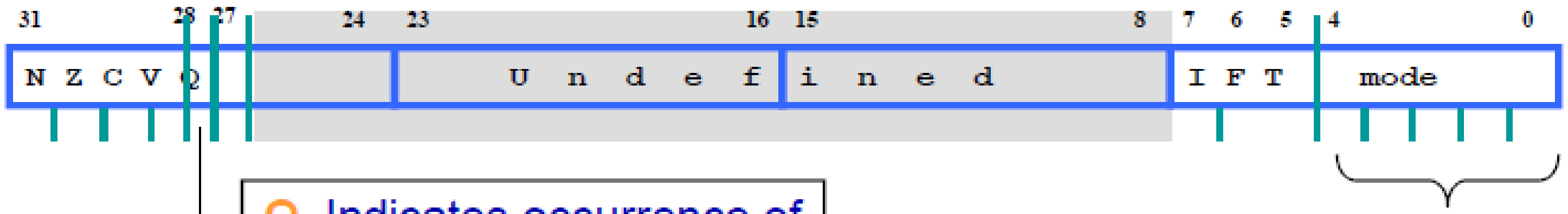
Condition Code Flags	
N	Set to 1 when result is negative
Z	Set to 1 when result is zero
C	Set to 1 on carry or borrow generation and on shift operations
V	Set to 1 if signed overflow occurs
Q	Set to 1 if Saturation occurs

Program Status Register



I	Disables IRQ interrupts when set
F	Disables FIQ interrupts when set
T	T=0 indicates ARM execution
	T=1 indicates Thumb execution

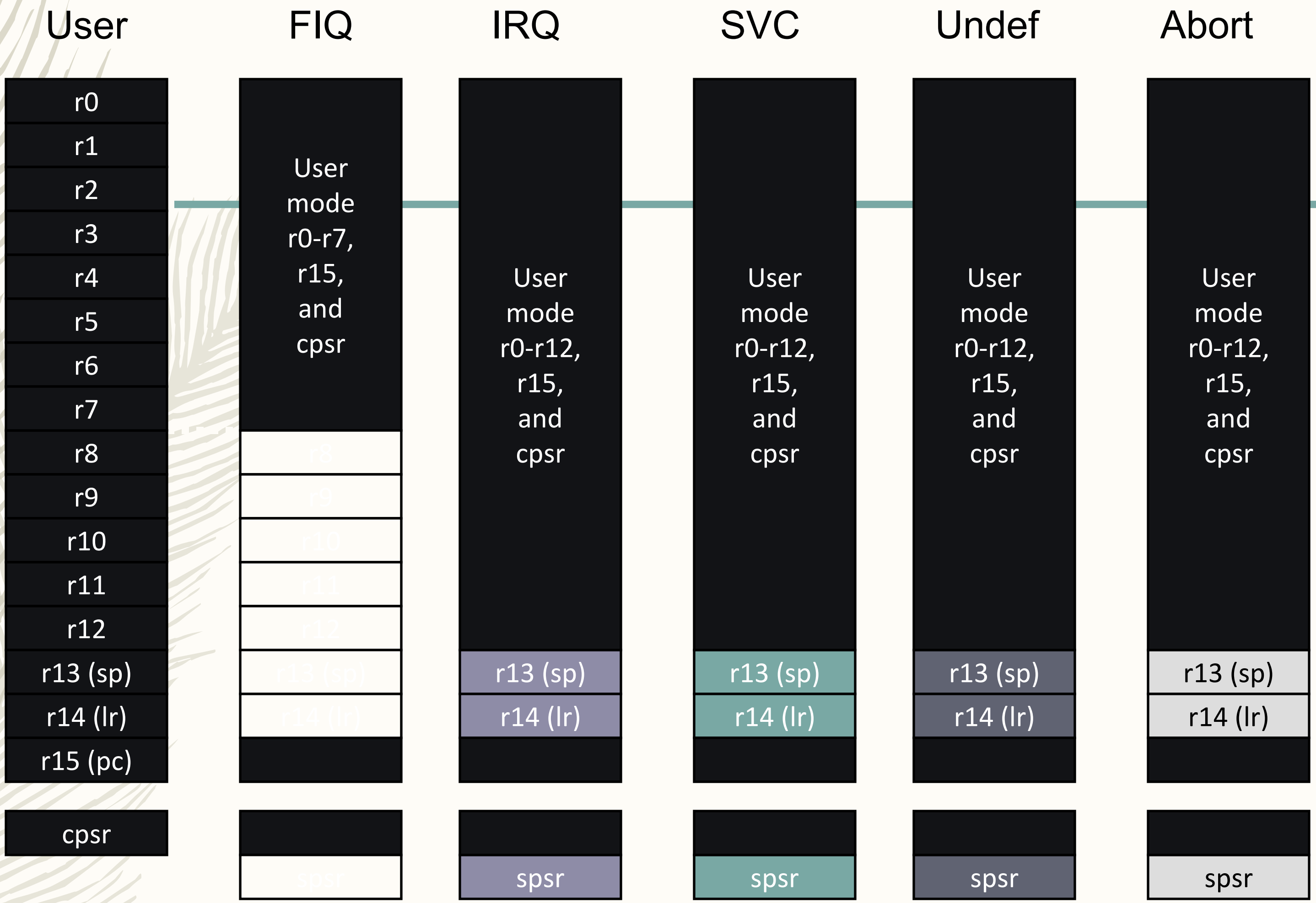
Program Status Register



Q Indicates occurrence of overflow and/or saturation

M[4:0]	Mode
0b10000	User
0b10001	FIQ
0b10010	IRQ
0b10011	Supervisor
0b10111	Abort
0b11011	Undefined
0b11111	System

Register Organization Summary



Note: System mode uses the User mode register set

ARM Architecture

In General

ARM Architecture

Typical RISC architecture:

- ❖ *Large uniform register file*
- ❖ *Load/store architecture*
- ❖ *Simple addressing modes*
- ❖ *Uniform and fixed-length instruction fields*

ARM Architecture (2)

Enhancements:

- *Each instruction controls the ALU and shifter*
- *Auto-increment and auto-decrement addressing modes*
- *Multiple Load/Store*
- *Conditional execution*

ARM Architecture (3)

Results:

- *High performance*
- *Low code size*
- *Low power consumption*
- *Low silicon area*

Pipeline

In General

Pipeline

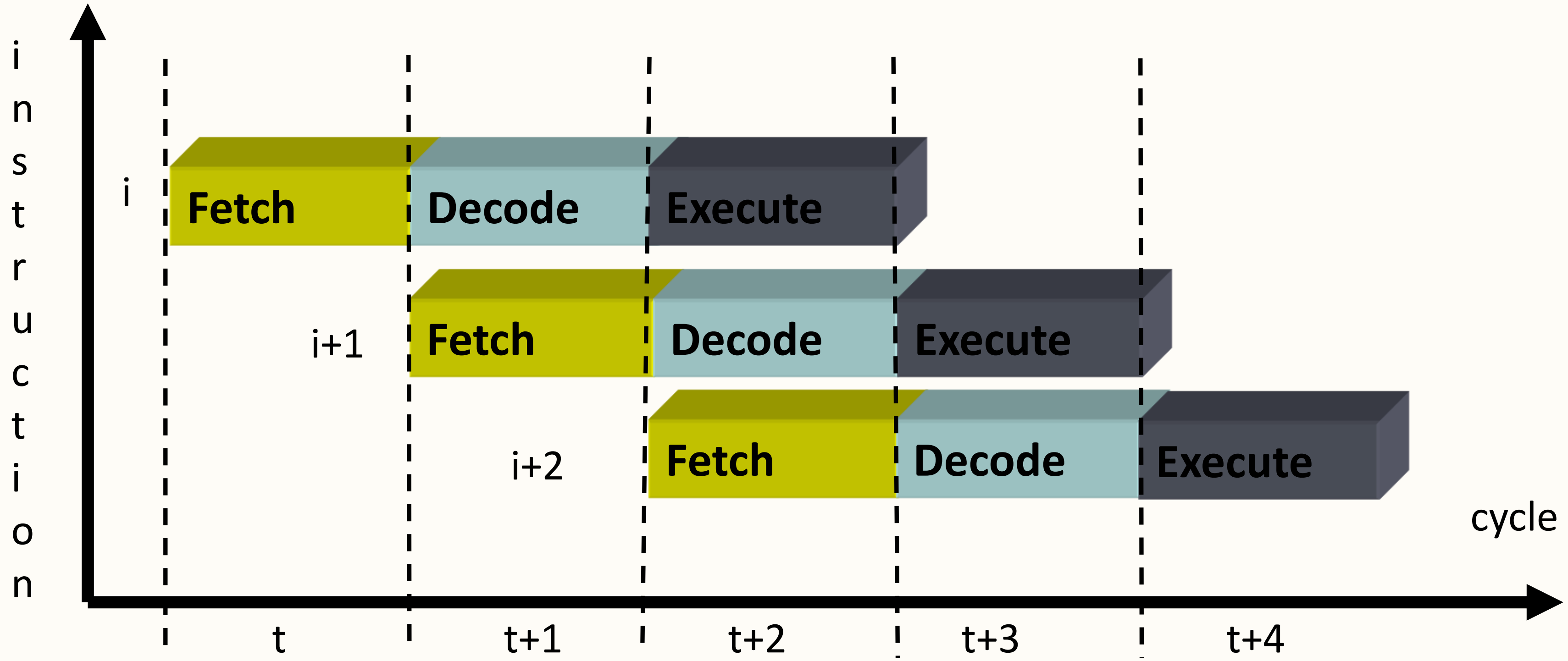
- Modern CPUs are designed as **pipelined** machines in which several instructions are executed in parallel.
  - It increases the efficiency of the CPU.
- A pipeline is the mechanism a RISC processor uses to execute instructions.
  - Pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed.

Pipeline Organization

- ❖ Increases speed :
  - ✓ Most instructions executed in single cycle
- ❖ Versions:
  - ✓ 3-stage (ARM7TDMI and earlier)
  - ✓ 5-stage (ARMS, ARM9TDMI)
  - ✓ 6-stage (ARM10TDMI)

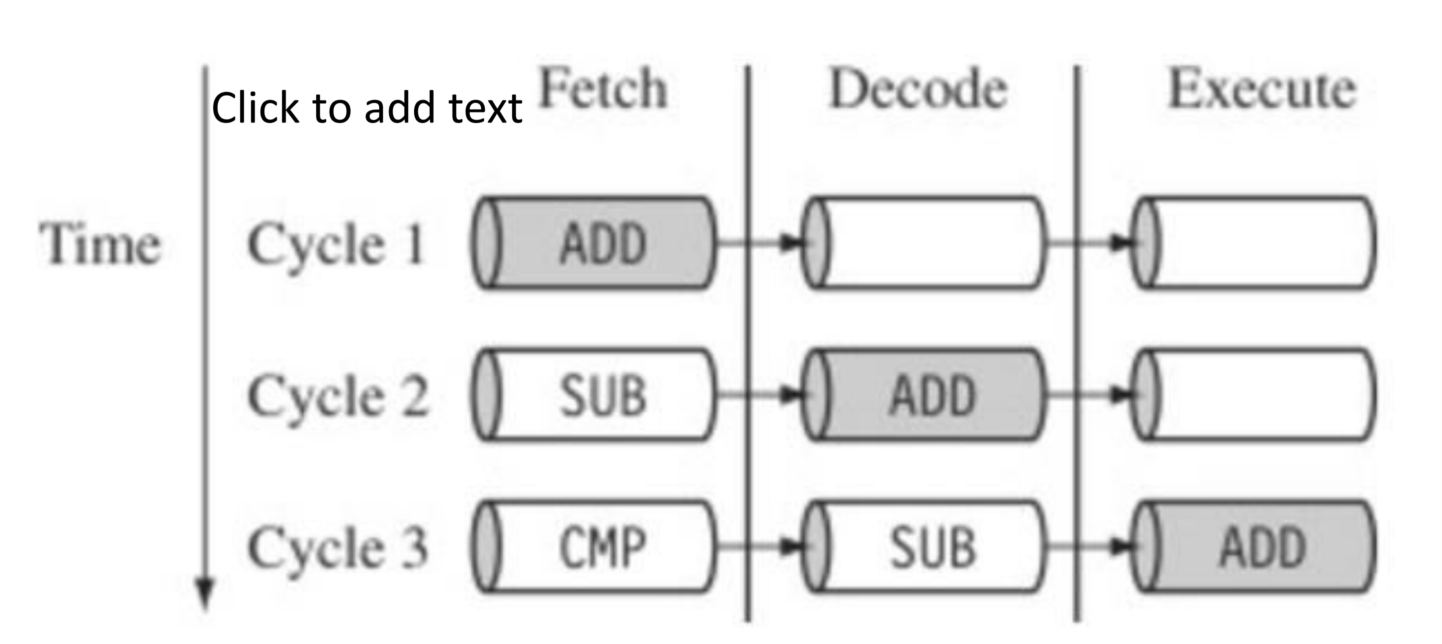
Pipeline Organization (2)

- 3-stage pipeline: Fetch – Decode - Execute
- Three-cycle latency, one instruction per cycle throughput



3-stage pipeline : Example

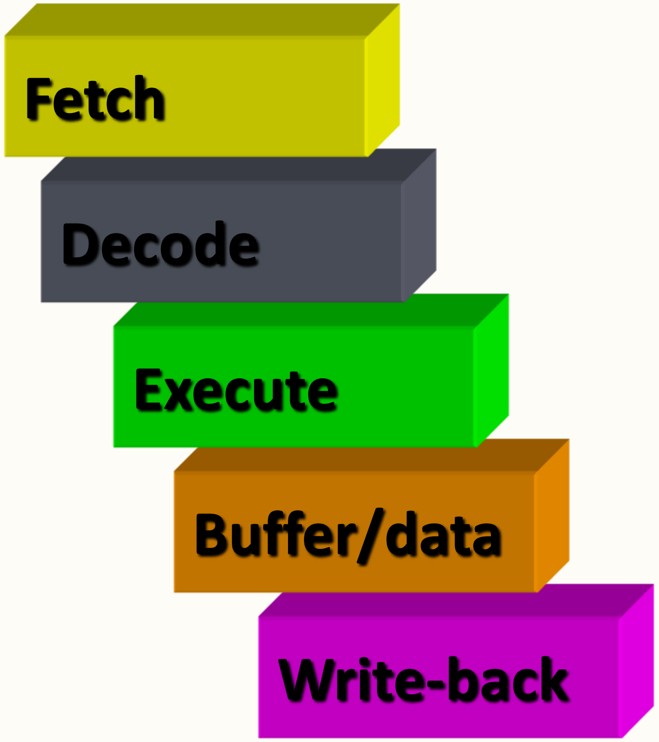
Click to add text



Pipeline Organization (3)

- **5-stage pipeline:**
  - *Reduces work per cycle* => allows higher clock frequency
  - *Separates data and instruction memory* => reduction of CPI (average number of clock Cycles Per Instruction)

Stages:



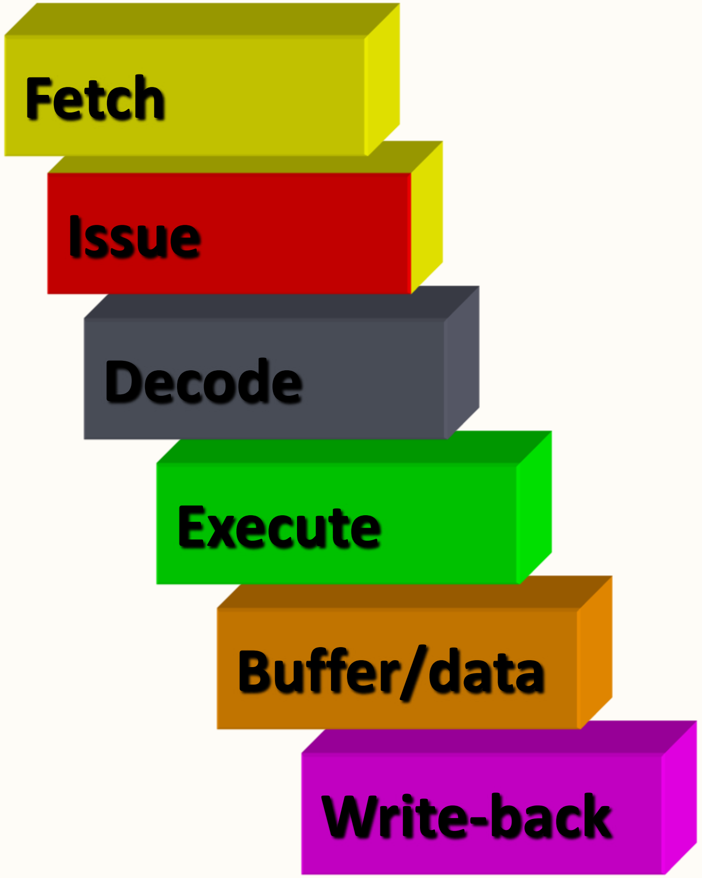
Pipeline Organization (4)

❖ **6-stage pipeline:**

❖ *Reduces work per cycle =>* allows higher clock frequency

❖ *Uses in ARM10*

Stages:

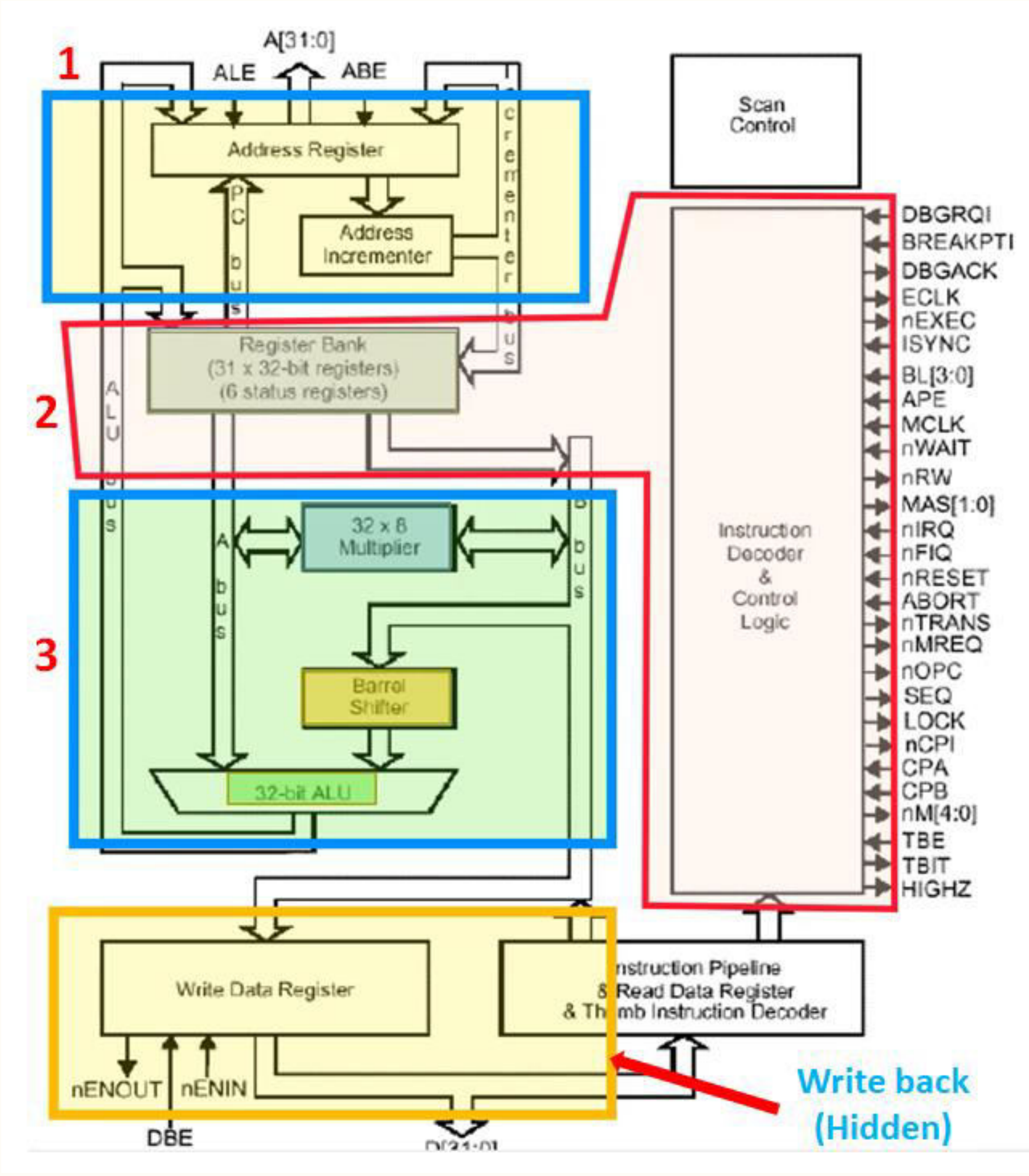


ARM Pipeline Characteristics

- The ARM pipeline doesn't process an instruction until it passes completely through the execution stage.
- In the execution stage, the PC always points to the instruction address + 8 bytes.
- When the processor is in thumb state, PC always points to the instruction address + 4 bytes.
- While executing branch instructions or branching by direct modification of PC causes the ARM core to **flush** it's pipeline.
- As instruction in the execution stage will complete its execution even though an interrupt has been raised.

Pipeline Organization (5)

- ❖ Pipeline flushed and refilled on branch,
  - ✓ *causing execution to slow down*
- ❖ Special features in instruction set
  - ✓ *eliminate small jumps in code*
  - ✓ *to obtain the best flow through pipeline*



3-Stage Pipeline in ARM7

1) **Fetch**

- The instruction is fetched from memory and placed in the instruction pipeline

2) **Decode**

- The instruction is decoded and the data path control signals prepared for the next cycle

3) **Execute**

- The register bank is read, an operand shifted, the ALU result generated and written back into destination register

Thank You

Thank You!