Introduction to ARM-7

The first encounter

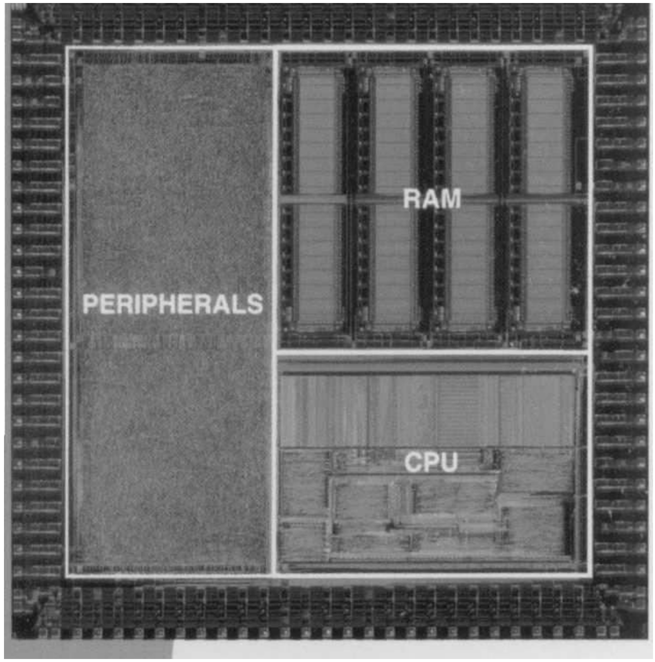
Dr. P. H. Zope
Assistant Professor
SSBT’s COET Bambhori Jalgaon
North Maharashtra University Jalgaon India
phzope@gmail.com
9860631040

What Is ARM?

- Advanced RISC Machine
- First RISC microprocessor for commercial use
- Market-leader for low-power and cost-sensitive embedded applications

ARM7TDMI

TDMI = (?)
– Thumb instruction set
– Debug-interface (JTAG/ICEBreaker)
– Multiplier (hardware)
– Interrupt (fast interrupts)



ARM7/ARM9 Architecture Feature Highlights

- 32/16-bit RISC architecture (ARM v4T)
- 32-bit ARM instruction set for maximum performance and flexibility
- 16-bit Thumb instruction set for increased code density
- Unified bus interface, 32-bit data bus carries both instructions and data
- 8-, 16-, and 32-bit Data Types
- Three-stage pipeline
- 4GBytes Linear Address Space
- 32-bit ALU and high-performance multiplier
- 37 piece of 32 bit register
- Very small die size and low power consumption
- Fully static operation
- Coprocessor interface
- Extensive debug facilities:
  - Embedded ICE-RT real-time debug unit.
  - On-chip JTAG interface unit.
- Interface for direct connection to Embedded Trace Macro cell (ETM).

•Pipelined (ARM7: 3 stages)
• Cached (depending on the implementation)
• Von Neuman-type bus structure (ARM7), Harvard (ARM9)
•7 modes of operation (usr, fiq, irq, svc, abt, sys, und)
• Simple structure -> reasonably good speed / power consumption ratio
•Very Low Power Consumption: Industry-leader in MIPS/Watt.

Differences between RISC and CISC

CISC	RISC
Variable size instructions with many formats	Fixed size instructions (32 bit)with few formats
Multi clock complex instructions.	Single clock reduced instructions.
Memory to memory load andstore instructions	**Register to register load and**store
Small code size, high cycles per second.	Large code size, Low cycles per second.
Emphasis on hardware	Emphasis on software
Increased hardware cost.	Reduced hardware cost.

ARM Powered Products

Pipeline Organization

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

ARM7 Pipeline Model

- **ARM7 → standard 3-stage pipelined architecture**

FETCH → DECODE → EXECUTE

- **Fetch Instruction**
  - Select/Increment PC
  - Read next instruction
- **Related Blocks**
  - Address Selector
  - Address Incrementer
  - Address Register

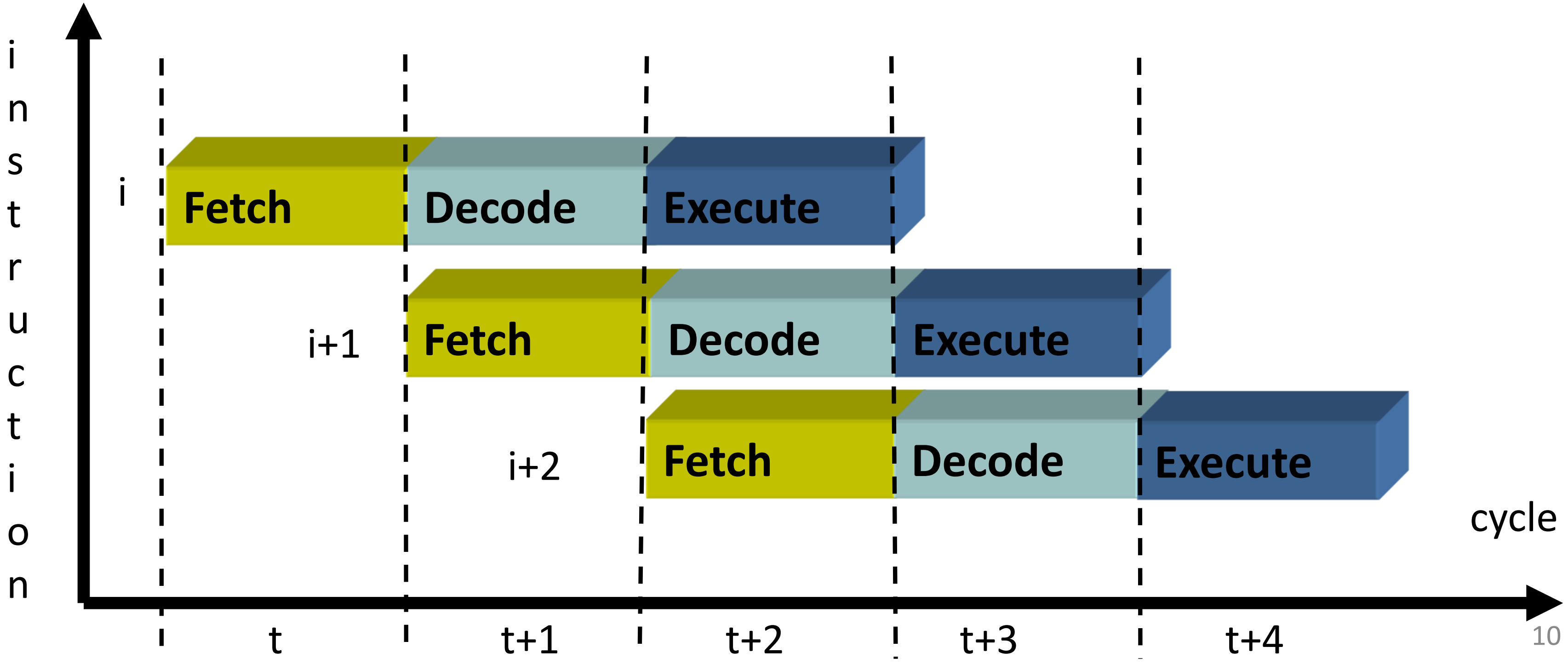
- **Decode Instruction**
  - Generate Ctrl. signals
  - Generate immediate
  - Read from register file
- **Related Blocks**
  - Control Logic (Decoder)
  - Register File

- **Execute Instruction**
  - Arithmetic / Logic
  - Calc. branch addr.
  - Load / Store
- **Related Blocks**
  - Shifter
  - Multiplier
  - ALU

*Register write back (WB) is hidden

Pipeline Organization

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



Pipeline Organization

- 5-stage pipeline:

Reduces work per cycle =>

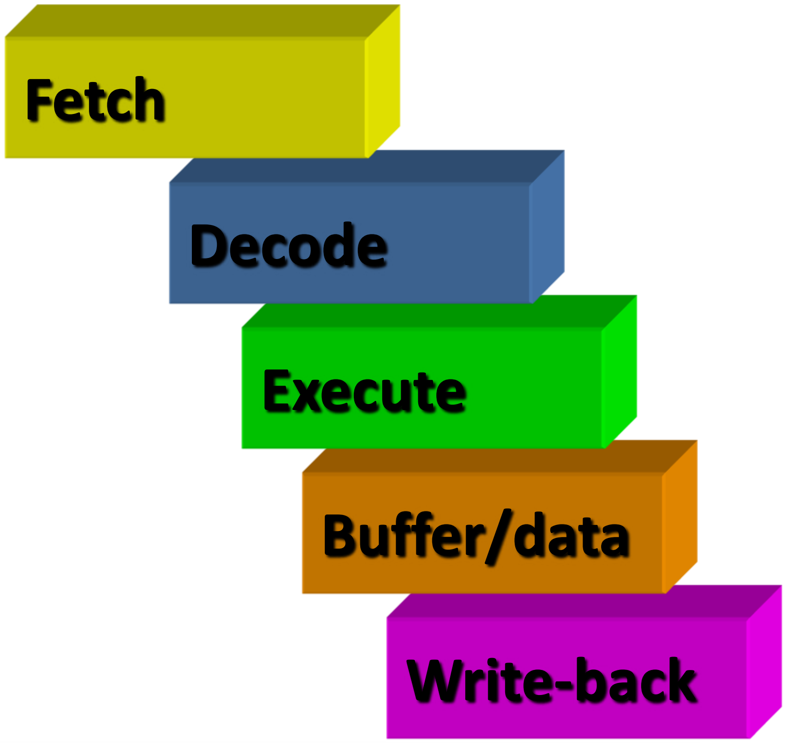
allows higher clock frequency

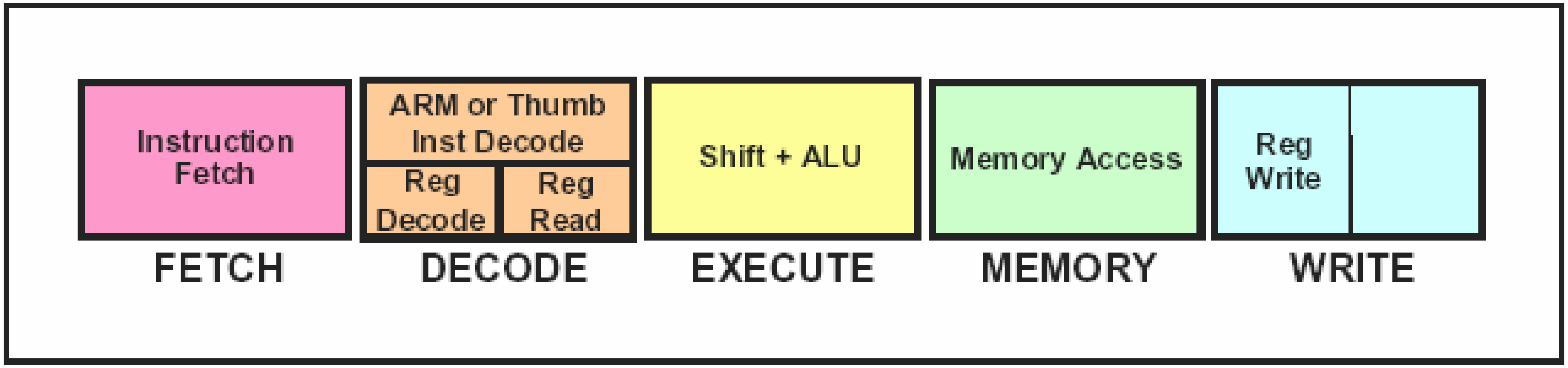
Separates data and instruction memory =>

reduction of CPI

(average number of clock Cycles Per Instruction)

Stages:





Pipeline Organization

- 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

ARM-7 Architecture

ARM Architecture Version Summary

Core	Version	Feature
ARM1	v1	❑26 bit address
ARM2, ARM2as, ARM3	v2	❑32 bit multiply ❑coprocessor
ARM6, ARM60, ARM610, ARM7, ARM710, ARM7D, ARM7DI	v3	❑32 bit addresses ❑Separate PC and PSRs ❑Undefined instruction and Abort modes ❑Fully static ❑Big or little endian
StrongARM, SA-110, SA-1100 ARM8, ARM810	v4	❑Half word and signed halfword/byte support ❑Enhanced multiplier ❑System mode
ARM7TDMI, ARM710T, ARM720T, ARM740T ARM9TDMI, ARM920T, ARM940T	v4T	❑Thumb instruction set

T: Thumb instruction set

D: On-chip Debug

M: enhanced Multiplier

I: Embedded ICE Logic

ARM Architecture Version

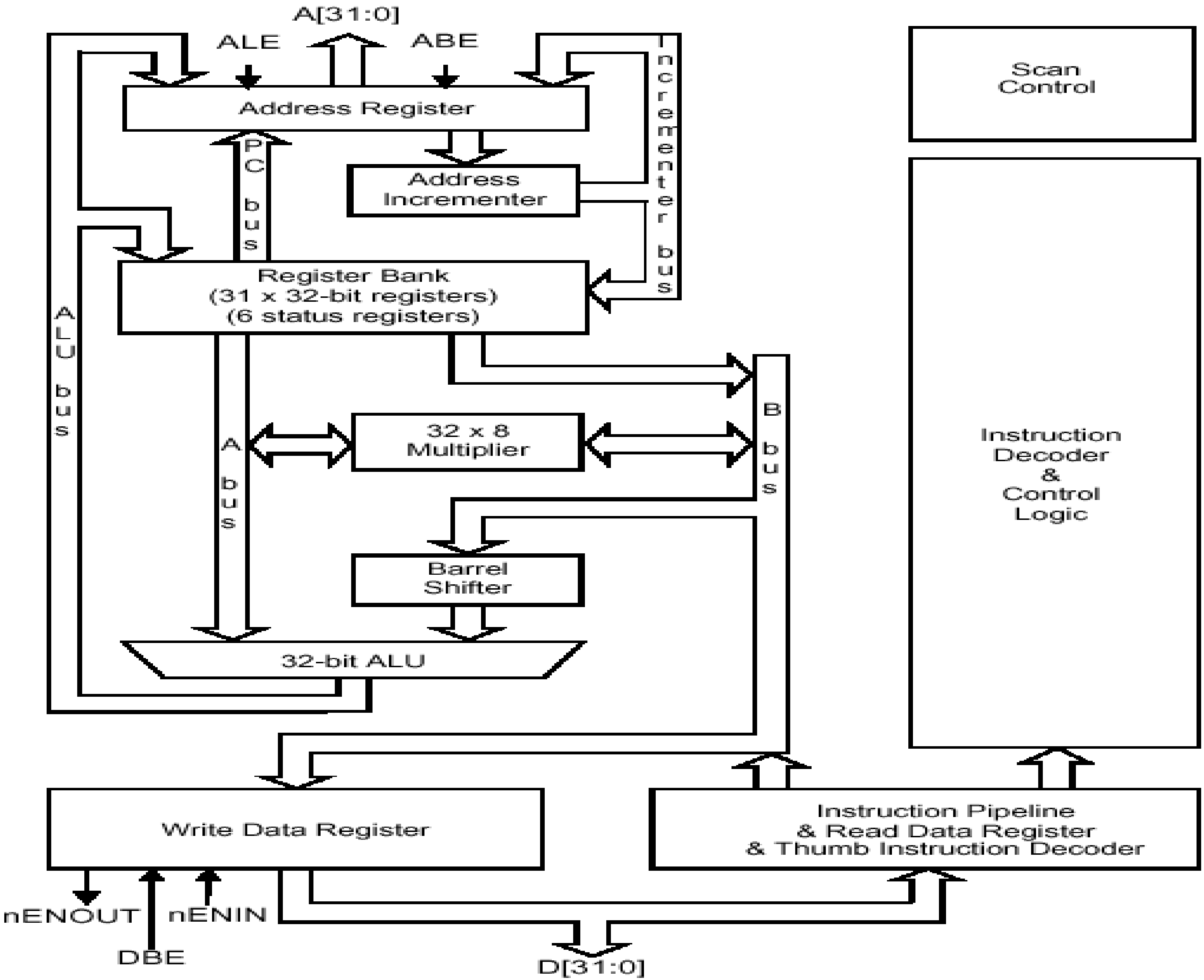
Core	Version	Feature
ARM1020T	v5T	❑Improved ARM/Thumb Interworking ❑CLZ instruction for improved division
ARM9E-S, ARM10TDMI, ARM1020E	v5TE	❑Extended multiplication and saturated maths for DSP-like functionality
ARM7EJ-S, ARM926EJ-S, ARM1026EJ-S	v5TEJ	❑Jazelle Technology for Java acceleration
ARM11, ARM1136J-S,	v6	❑Low power needed ❑SIMD (Single Instruction Multiple Data) media processing extensions

J: Jazelle

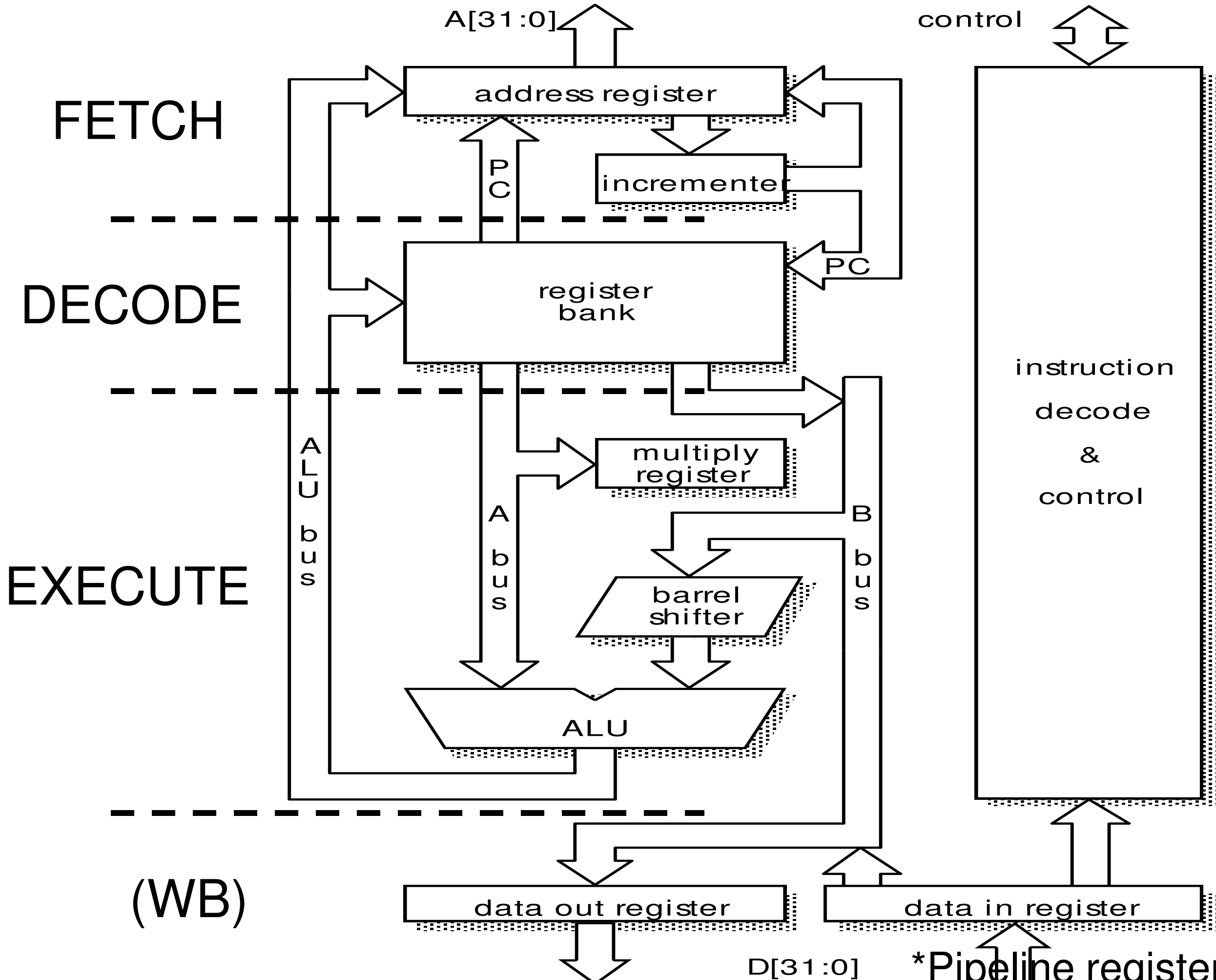
S: Synthesizable

E: Enhanced DSP instruction

F: integral vector floating point unit

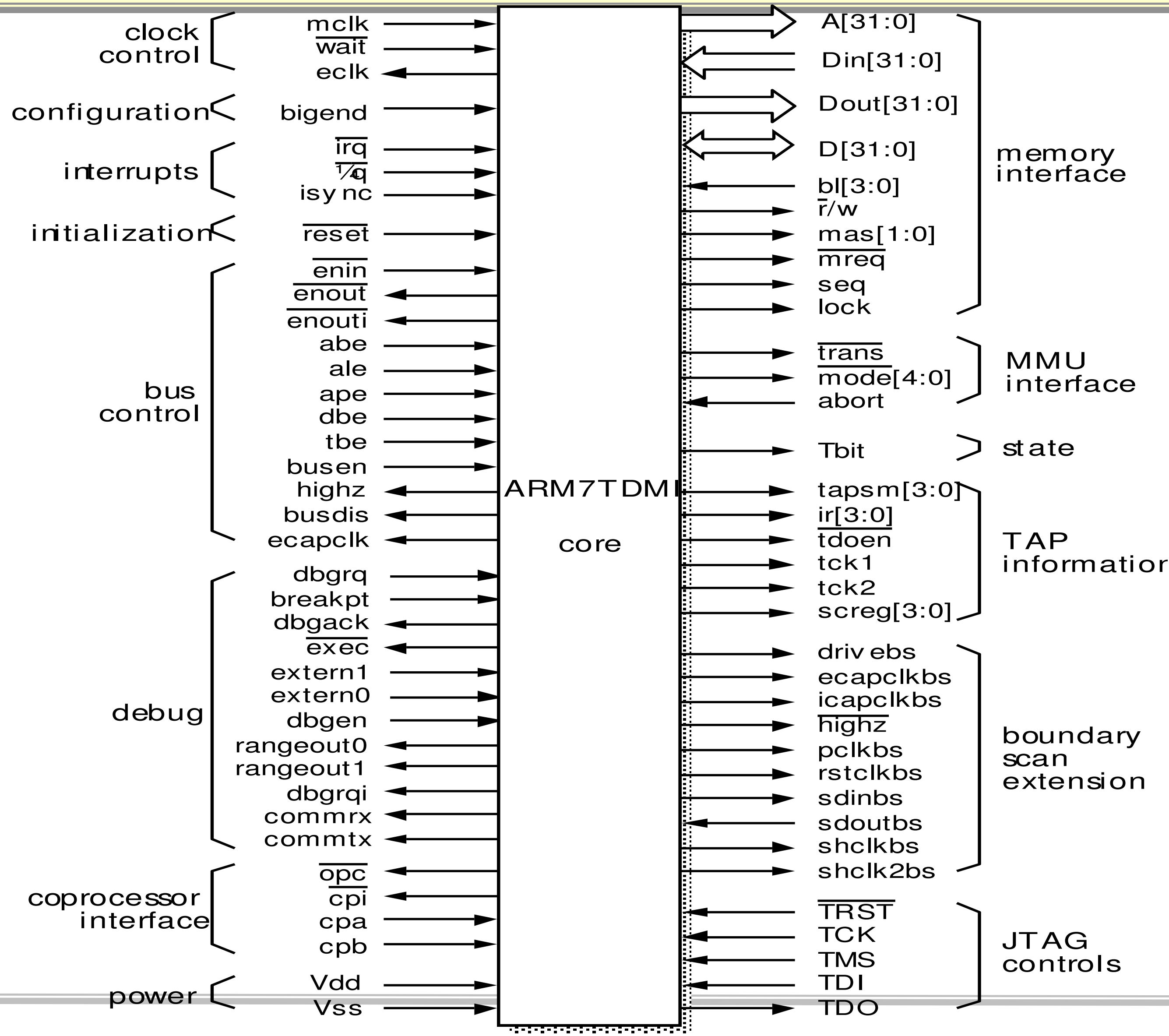

A[31:0]
ALE
ABE
Address Register
PC bus
Address Incrementer
Incrementer bus
Register Bank
(31 x 32-bit registers)
(6 status registers)
ALU bus
A bus
B bus
32 x 8 Multiplier
Barrel Shifter
32-bit ALU
Scan Control
Instruction Decoder & Control Logic
Write Data Register
Instruction Pipeline & Read Data Register & Thumb Instruction Decoder
nENOUT
nENIN
DBE
D[31:0]

ARM7 Datapath Overview



*Pipeline registers are omitted

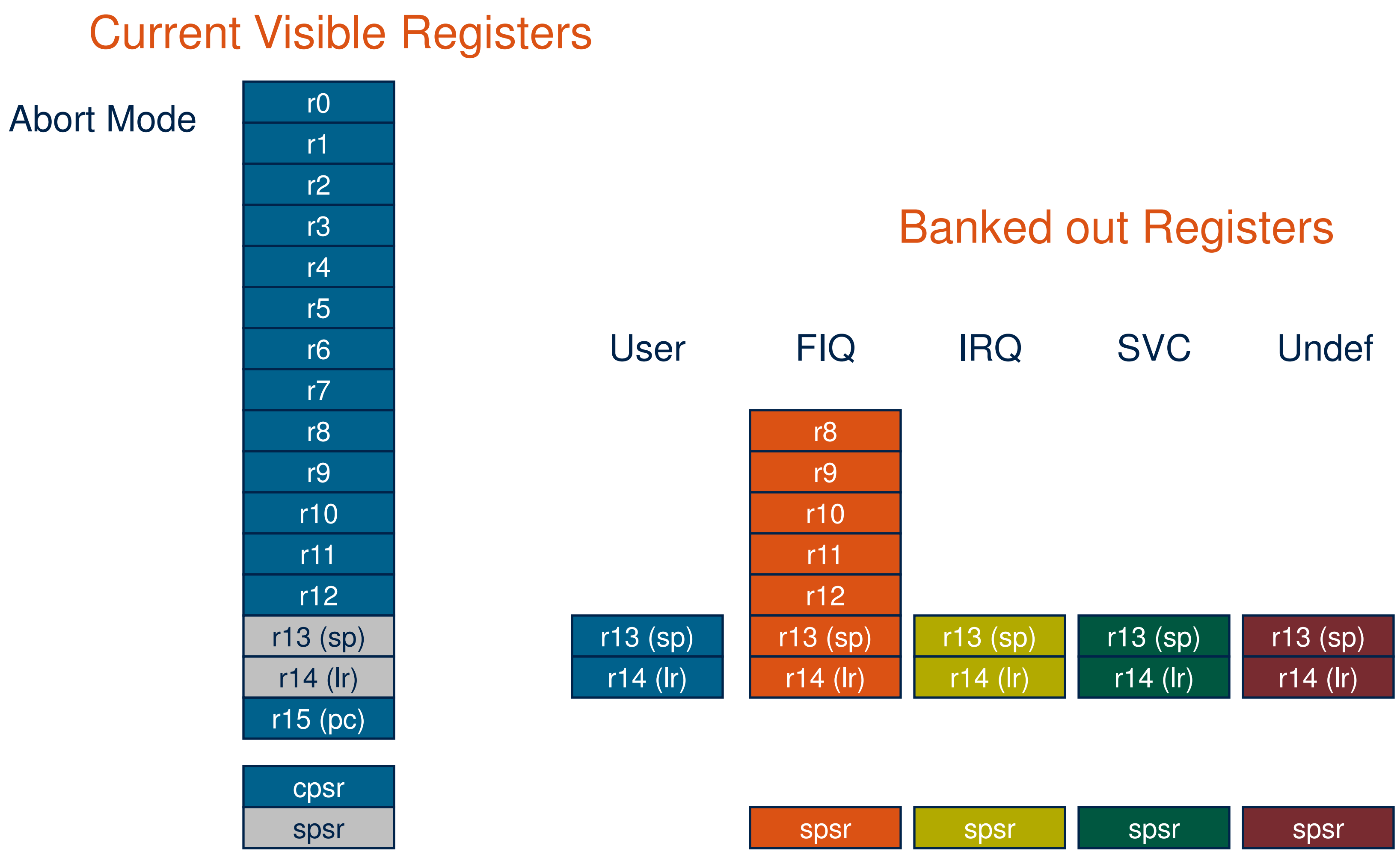
ARM7TDMI Interface Signals (1/4)



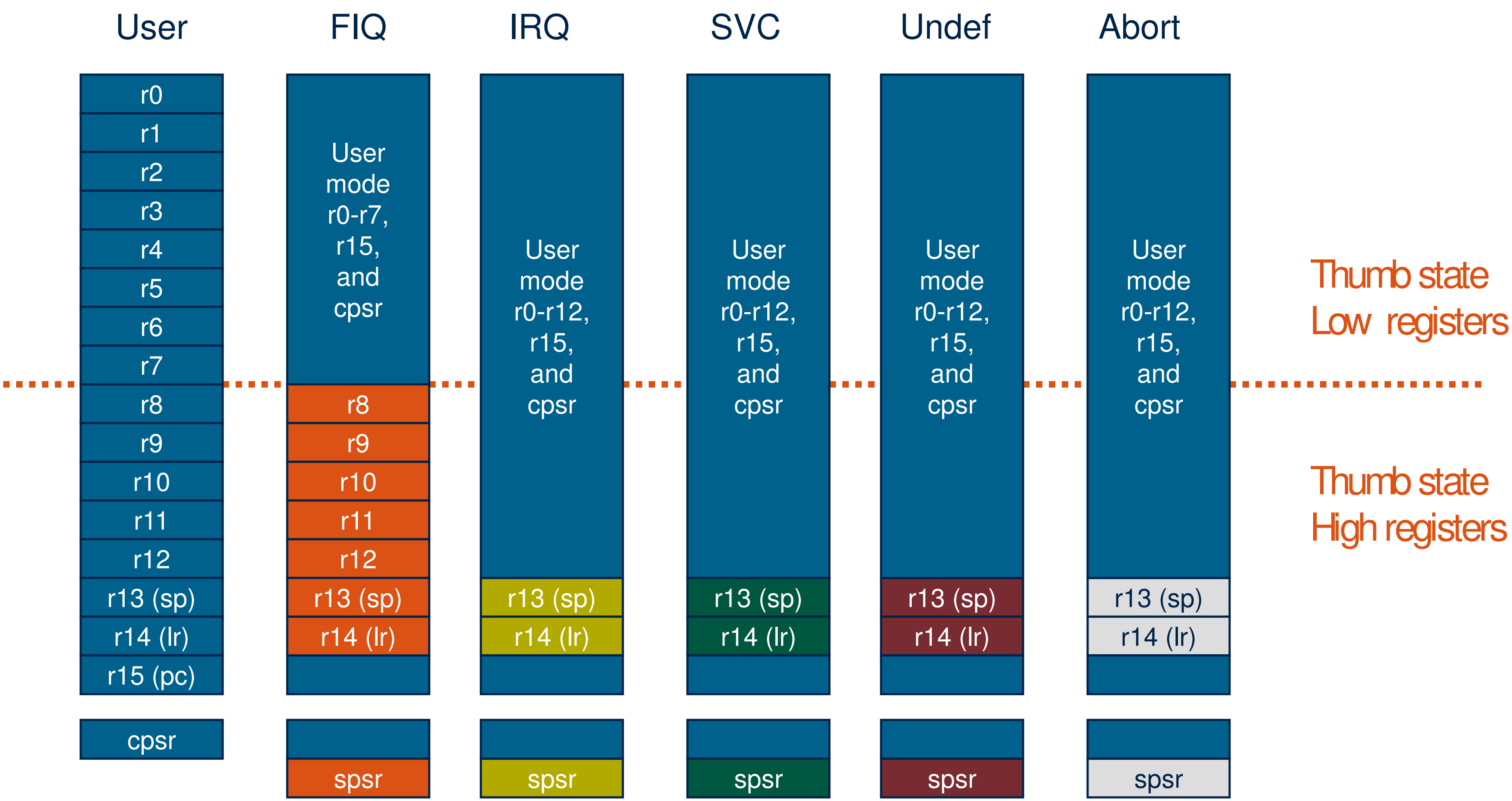
Processor Modes

- **The ARM has seven basic operating modes:**
  1. **User** : unprivileged mode under which most tasks run
  2. **FIQ** : entered when a high priority (fast) interrupt is raised
  3. **IRQ** : entered when a low priority (normal) interrupt is raised
  4. **Supervisor** : entered on reset and when a Software Interrupt
  5. instruction is executed
  6. **Abort** : used to handle memory access violations
  7. **Undef** : used to handle undefined instructions
  8. **System** : privileged mode using the same registers as user mode

The ARM Register Set



Register Organization Summary



Note: System mode uses the User mode register set

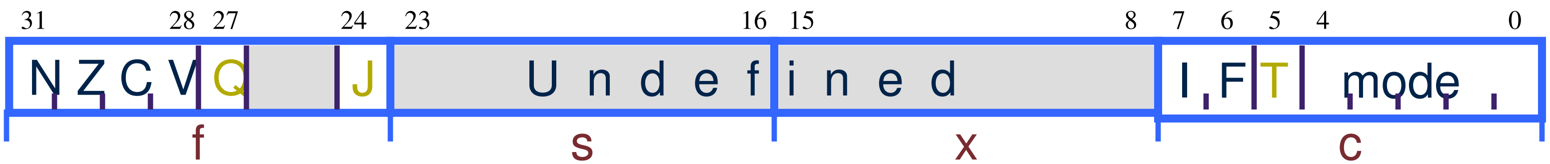
The Registers

- **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

- **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)

Current Program Status Registers



- **Condition code flags**
  - N = **N**egative result from ALU
  - Z = **Z**ero result from ALU
  - C = ALU operation **C**arried out
  - V = ALU operation o**V**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 Counter (r15)

- **When the processor is executing in ARM state:**
  - All instructions are 32 bits wide
  - All instructions must be word aligned
  - Therefore the **pc** value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be halfword or byte aligned).

- **When the processor is executing in Thumb state:**
  - All instructions are 16 bits wide
  - All instructions must be halfword aligned
  - Therefore the **pc** value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte 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

Saved Program Status Register (SPSR)

- Each privileged mode (except system mode) has associated with it a SPSR
- This SPSR is used to save the state of CPSR when the privileged mode is entered in order that the user state can be fully restored when the user process is resumed
- Often the SPSR may be untouched from the time the privileged mode is entered to the time it is used to restore the CPSR
- If the privileged supervisor calls to itself the SPSR must be copied into a general register and saved

Table 2-1. PSR Mode. Bit Values

M[4:0]	Mode	Visible THUMB State Registers	Visible ARM State Registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq..R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc..R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt..R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und..R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

What is Exceptions

- **Exceptions are usually used to handle unexpected events which arise during the execution of a program, such as interrupts or memory faults, also cover software interrupts, undefined instruction traps, and the system reset**
- **Three groups:**
  - Exceptions generated as the direct effect of executing an instruction
    - Software interrupts, undefined instructions, and prefetch abort
  - Exceptions generated as a side effect of an instruction
    - Data aborts
  - Exceptions generated externally
    - Reset, IRQ and FIQ

Exception Entry

- **When an exception arises**
  - ARM completes the current instruction as best it can (except that *reset* exception)
  - handle the exception which starts from a specific location (exception vector).
- **Processor performs the following sequence:**
  - Change to the operating mode corresponding to the particular exception
  - Stores the return address in `LR_`<mode>
  - Copy old CPSR into `SPSR_`<mode>
  - Set appropriate CPSR bits
    - If core currently in Thumb state then ARM state is entered.
    - Disable IRQs by setting bit 7
    - If the exception is a fast interrupt, disable further faster interrupt by setting bit 6 of the CPSR

Exception Entry

- Force PC to relevant vector address

Priority	Exception	Mode	vector address
1	Reset	SVC	0x00000000
2	Data abort (data access memory fault)	Abort	0x00000010
3	FIQ (fast interrupt)	FIQ	0x0000001C
4	IRQ (normal interrupt)	IRQ	0x00000018
5	Prefetch abort (instruction fetch memory fault)	Abort	0c0000000C
6	Undefined instruction	UND	0x00000004
	Software interrupt (SWI)	SVC	0x00000008

- Normally the vector address contains a branch to the relevant routine
- Exception handler use `r13_`<mode> and `r14_`<mode> to hold the stack point and return address

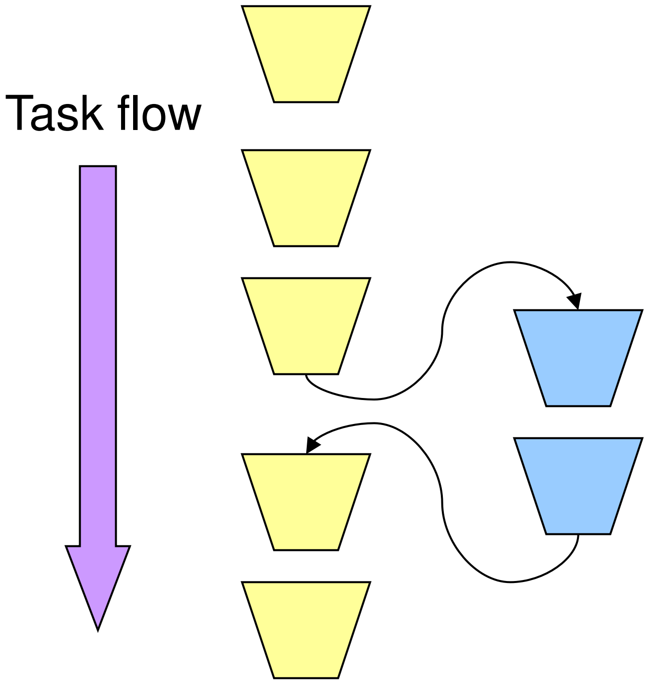
Exception Return

- **Once the exception has been handled, the user task is normally resumed**
- **The sequence is**
  - Any modified user registers must be restored from the handler's stack
  - CPSR must be restored from the appropriate SPSR
  - PC must be changed back to the relevant instruction address
- **The last two steps happen atomically as part of a single instruction**

Exceptions of ARM-7

- **Mode changes can be made under**
  - Software control
  - External interrupts
  - Exception process
- **The modes other than user mode are privileged modes**
  - Have full access to system resources
  - Can change mode freely
- **Exception modes**
  - FIQ
  - IRQ
  - Supervisor mode
  - Abort: data abort and instruction prefetch abort
  - Undefined

Exception



Class	Cause
Interrupt	External stimulus
Fault	Internal cause
Trap	Trap instruction

Exception (cont'd)

ARM7 (ISA v4) Exceptions

Type	Class	Description (Cause)
Reset		Power Up
Undefined Instruction	FAULT	Invalid / coprocessor instruction
Prefetch Abort	FAULT	TLB miss for instruction
Data Abort	FAULT	TLB miss for data access
IRQ	INTERRUPT	Normal interrupt
FIQ	INTERRUPT	Fast Interrupt (no context switch)
SW Interrupt instruction	TRAP	Undefined / coprocessor

Exception (cont'd)

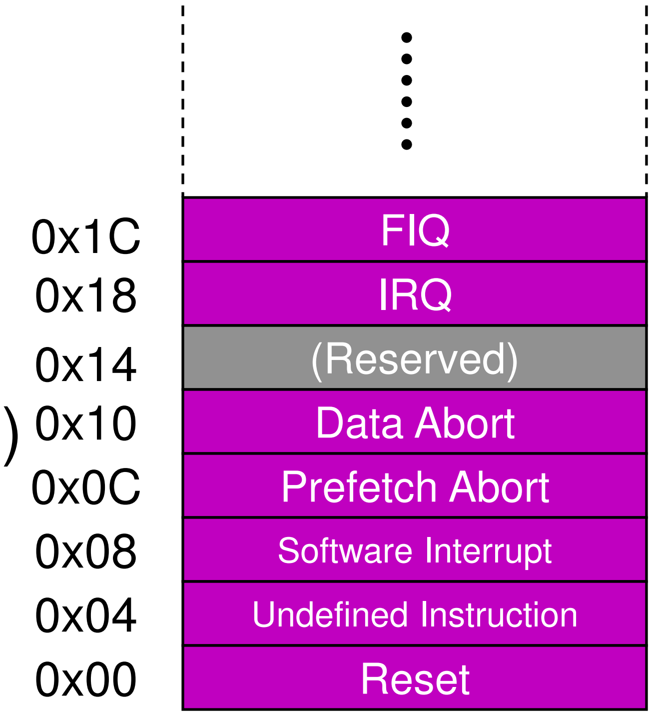
ARM7 (ISA v4) Exception Vectors

Exception	Address	Mode on Entry
Reset	0x00000000	Supervisor
Undefined Instruction	0x00000004	Undefined
SW Interrupt	0x00000008	Supervisor
Prefetch Abort	0x0000000C	Abort
Data Abort	0x00000010	Abort
IRQ	0x00000018	IRQ
FIQ	0x0000001C	FIQ
Reserved	0x00000014	Reserved

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

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

Exception (cont'd) Process

- Current Program Status Register (CPSR)
- Saved Program Status Register (SPSR)
- On exception, entering *mod* mode:
  - (PC + 4) → LR
  - CPSR → SPSR_mod
  - PC ← IV address
  - R13, R14 replaced by R13_mod, R14_mod
  - In case of FIQ mode R7 – R12 also replaced

Exception priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

Highest priority:

1. Reset
2. Data abort
3. FIQ
4. IRQ
5. Prefetch abort

Lowest priority:

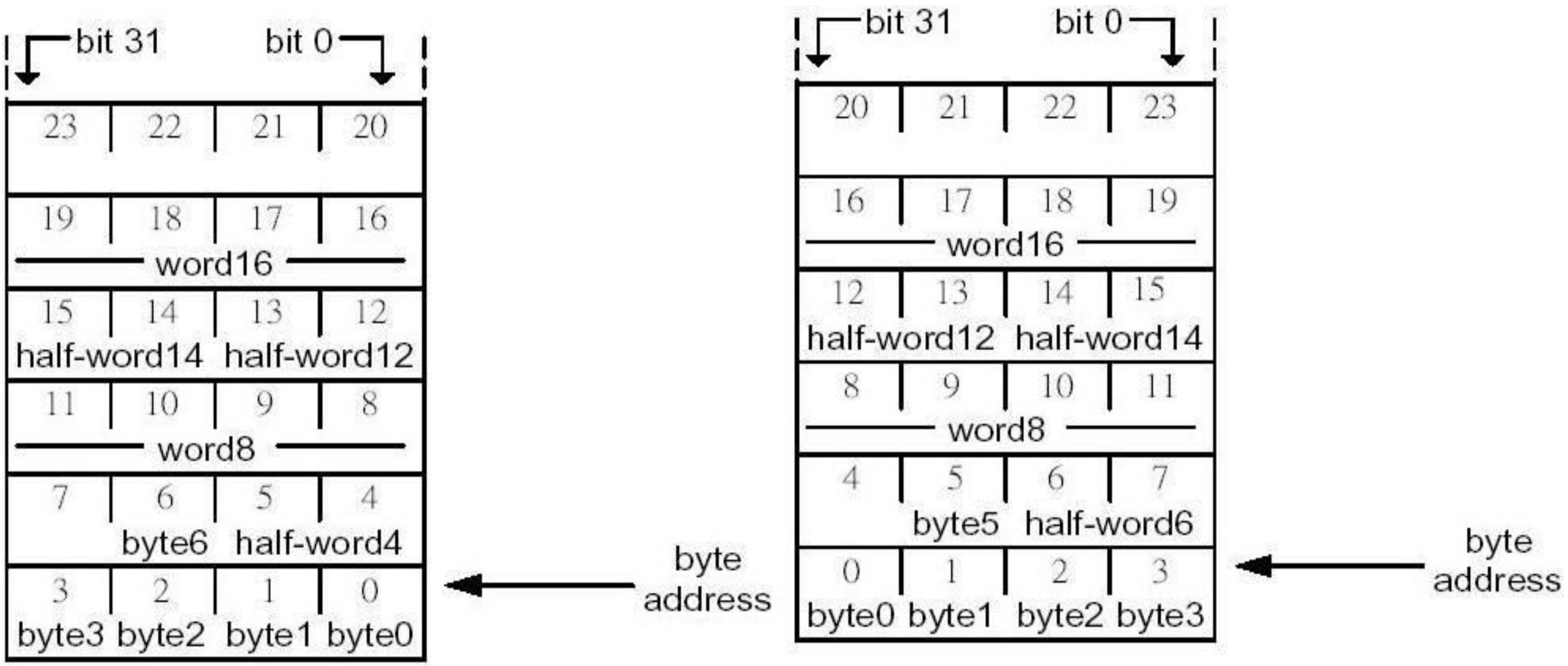
6. Undefined Instruction, Software interrupt.

Memory Organization

There are two ways to store data in memory

1 Little-Endian
2 Big – Endian

Memory Organization



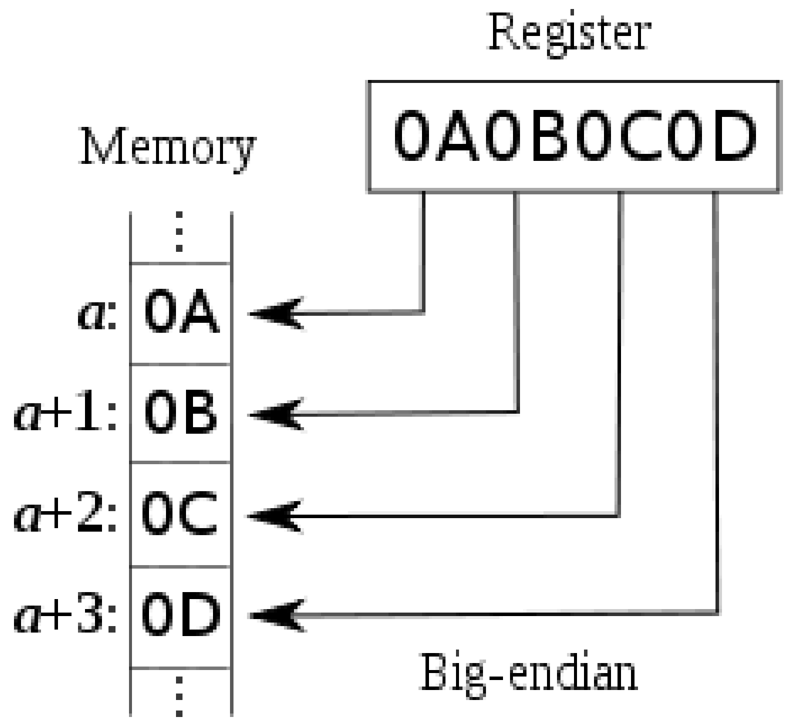
(a) Little-endian memory organization

(b) Big-endian memory organization

- Word, half-word alignment (xxxx00 or xxxxx0)
- ARM can be set up to access data in either *little-endian* or *big-endian* format, through they default to **little-endian**.

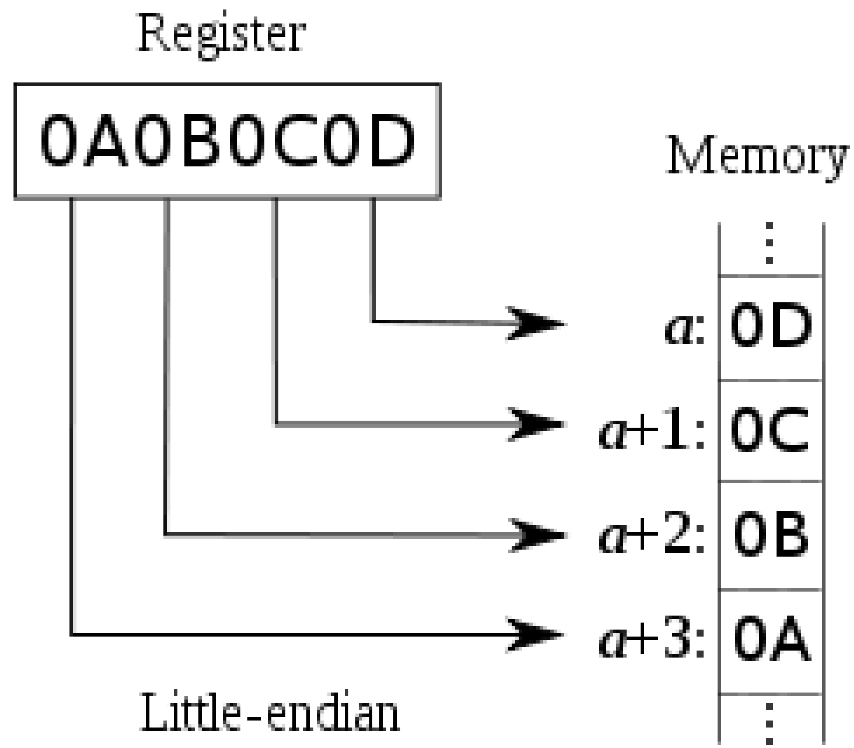
The most significant byte (*MSB*) value, which is $0A_h$ in our example, is stored at the memory location with the lowest address, the next byte value in significance, $0B_h$, is stored at the following memory location and so on. This is akin to Left-to-Right reading in hexadecimal order.

Big-endian

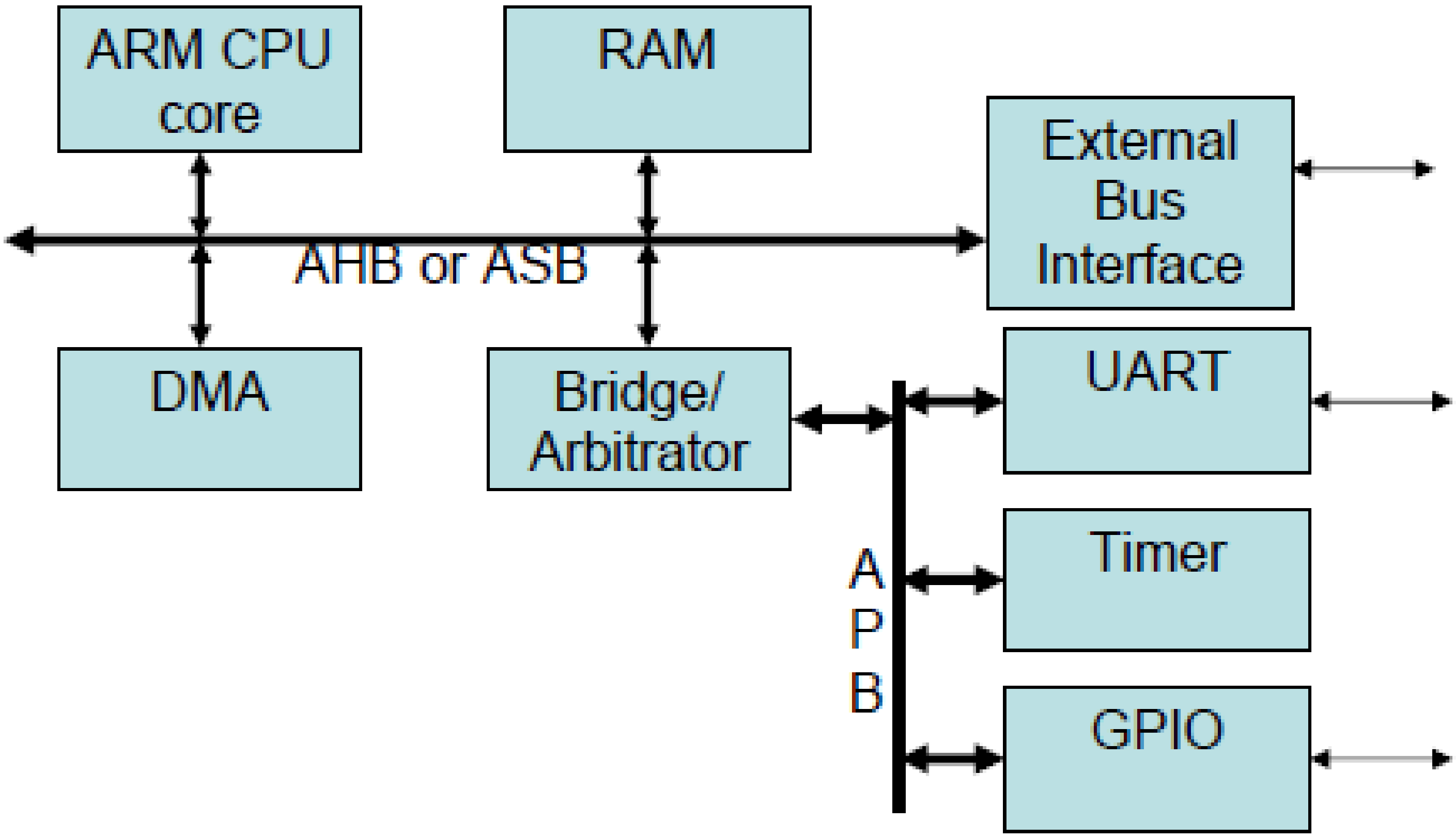


The least significant byte (*LSB*) value, $0D_h$, is at the lowest address. The other bytes follow in increasing order of significance

Little-endian



Advanced Microprocessor Bus Architecture (AMBA)



Advanced Microprocessor Bus Architecture

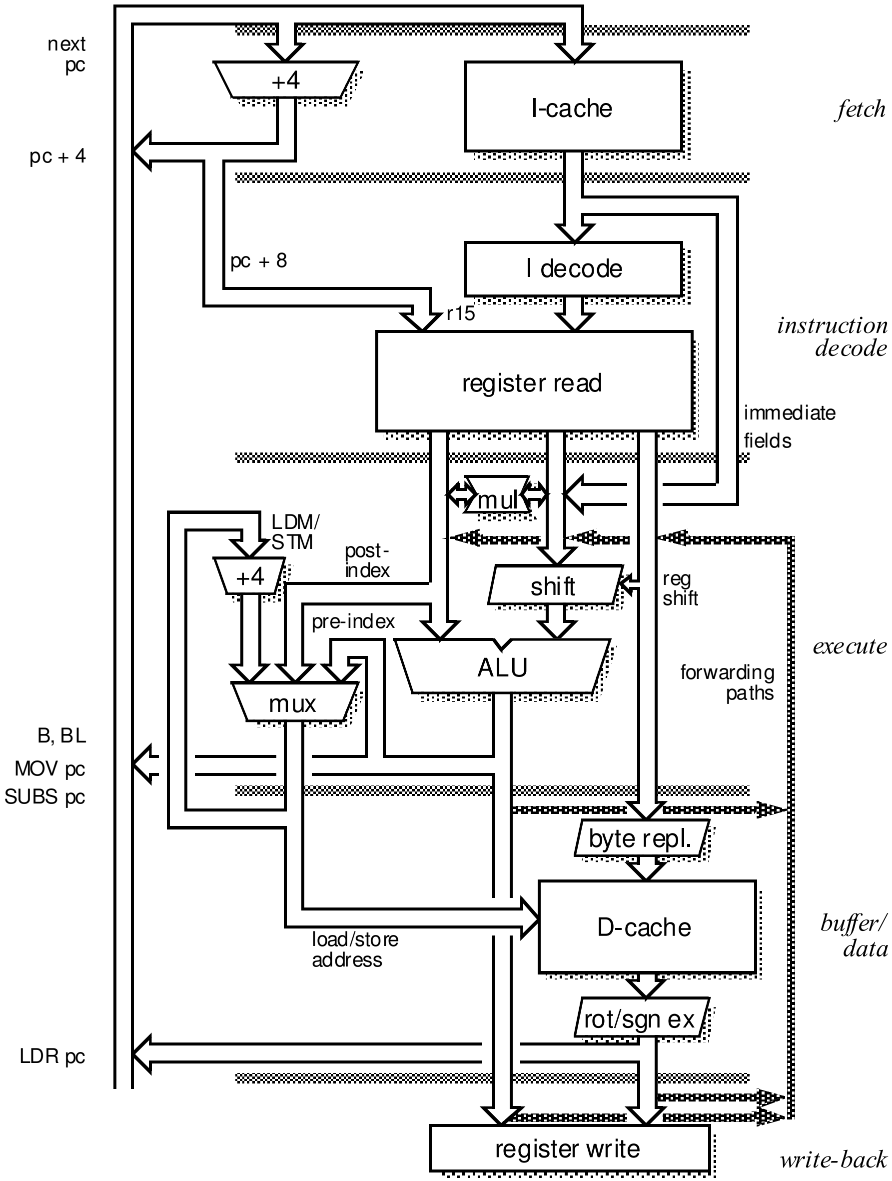
AHB The AMBA AHB is for high performance, high clock frequency system modules. It acts as a high performance system backbone that is capable for doing burst transfer, connecting the CPU and to on chip and off chip memories.

ASB AMBA ASB is an alternative system bus suitable for use where the high performance features of AHB are not required. ASB also supports the efficient connection of CPU, on chip memory and off chip memories.

APB AMBA APB is for low-power peripherals. It is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. APB is connected to CPU via AHB/ASB-APB bridge.

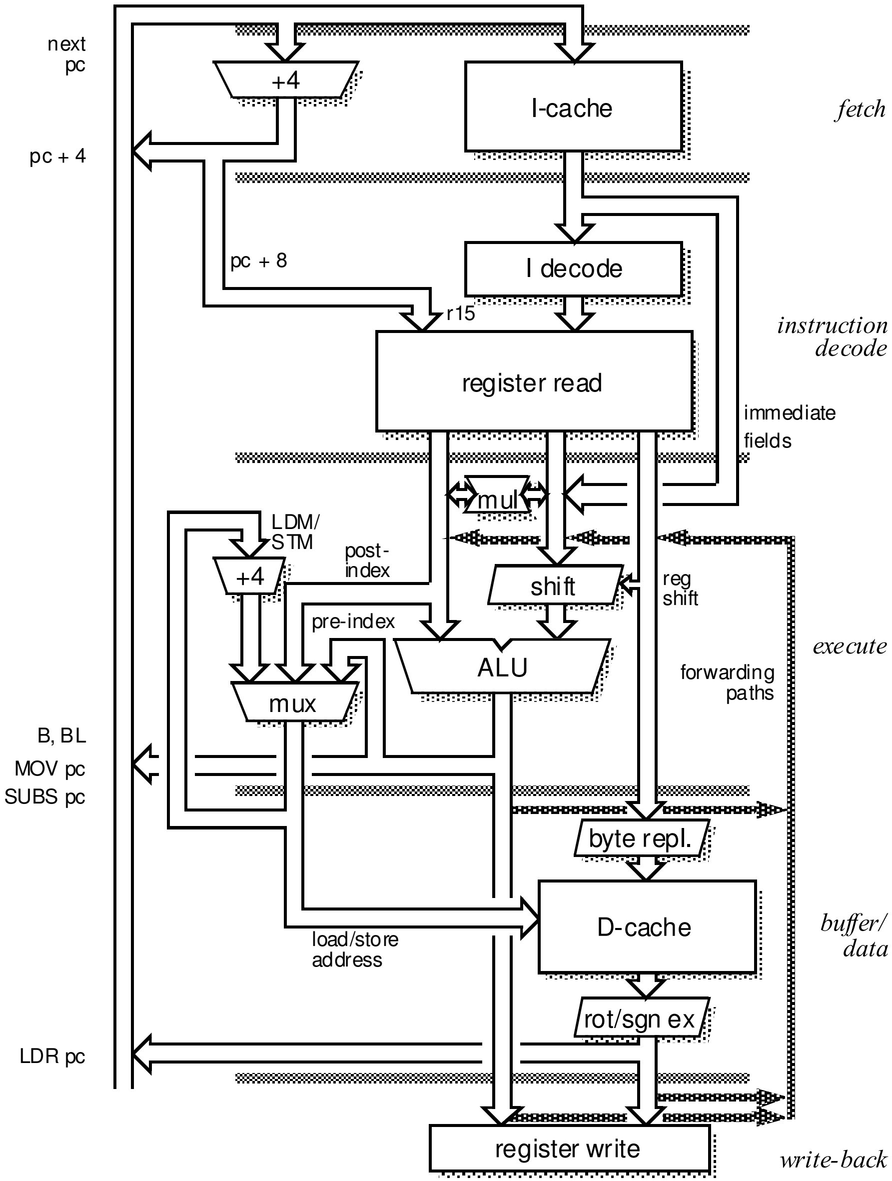
5-Stage Pipeline ARM Organization

5-Stage Pipeline Organization (1/2)



- Fetch
  - The instruction is **fetched** from memory and placed in the instruction pipeline
- Decode
  - The instruction is **decoded** and **register operands read** from the register files. There are 3 operand read ports in the register file so most ARM instructions can source all their operands in one cycle
- Execute
  - An operand is **shifted** and the **ALU result** generated. If the instruction is a load or store, the memory address is computed in the ALU

5-Stage Pipeline Organization (2/2)



- Buffer/Data
  - **Data memory is accessed** if required. Otherwise the ALU result is simply buffered for one cycle
- Write back
  - The result generated by the instruction are **written back to the register file**, including any data loaded from memory