Module -5

ARM microcontrollers

ARM microcontrollers:

- Need for RISC Processor-ARM processor fundamentals, ARM7TDMI Interface signals, Memory Interface, Bus Cycle types, Register set, Operational Modes.
- Instruction Format, ARM 3 stage Pipeline, ARM family attribute comparison. ARM 5 stage Pipeline,
- Data forwarding - a hardware solution, ARM ISA and Processor Variants, Different Types of Instructions, ARM Instruction set

CISC	RISC
The original microprocessor ISA	Redesigned ISA that emerged in the early 1980s
Instructions can take several clock cycles	Single-cycle instructions
Hardware-centric design – the ISA does as much as possible using hardware circuitry	Software-centric design – High-level compilers take on most of the burden of coding many software steps from the programmer
More efficient use of RAM than RISC	Heavy use of RAM (can cause bottlenecks if RAM is limited)
Complex and variable length instructions	Simple, standardized instructions
May support microcode (micro-programming where instructions are treated like small programs)	Only one layer of instructions
Large number of instructions	Small number of fixed-length instructions
Compound addressing modes	Limited addressing modes

ARM Ltd

Founded in 1990

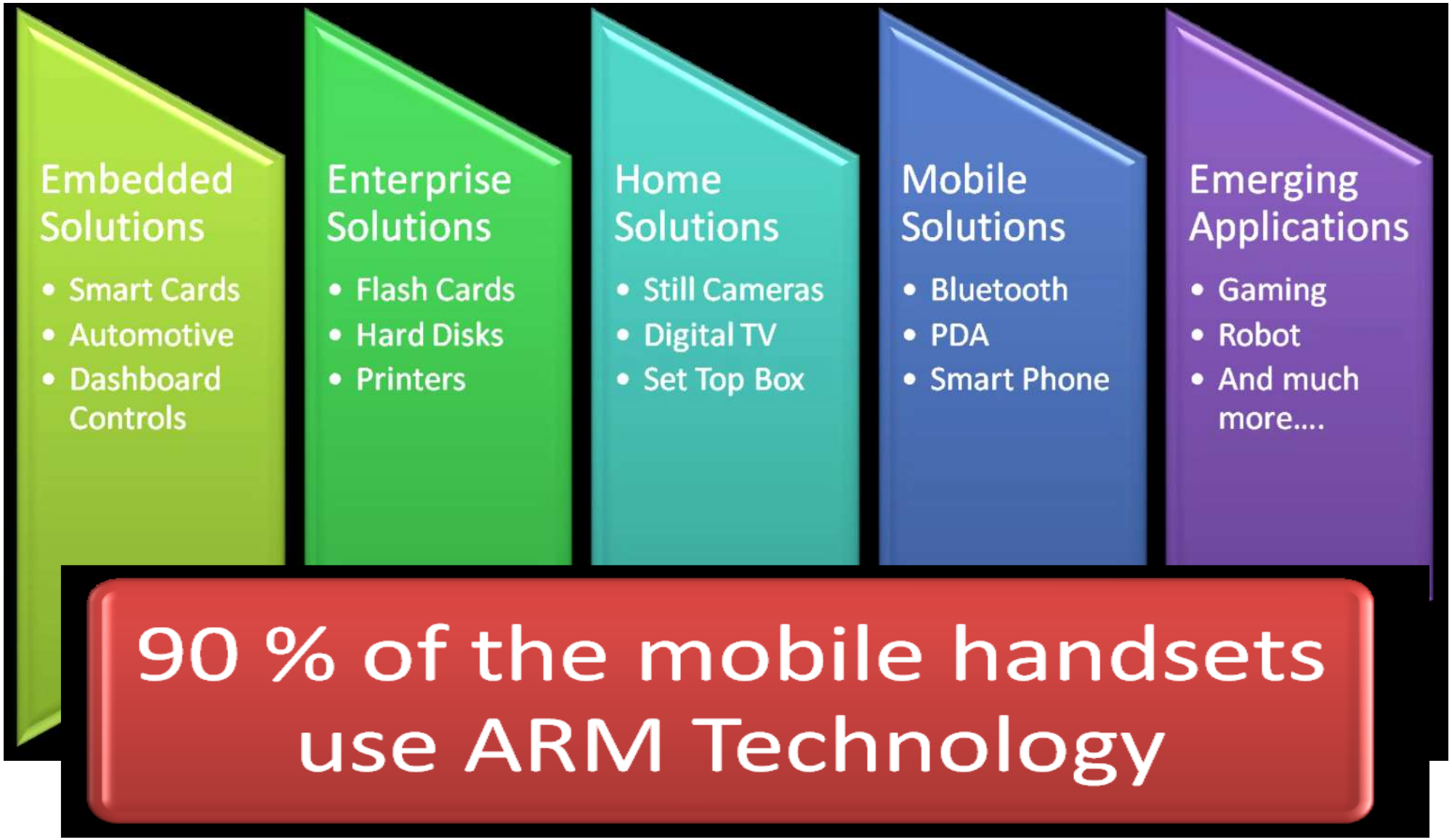
- – Spun out of Acorn Computers

- Designs the ARM range of RISC processor
- Licenses ARM core for Design Partners
  - – ARM does not fabricate Silicon on own
- Develops technology to assist with ARM Arch.
  - – Software Tools
  - – Boards | Debug Hardware | Peripherals
  - – Application Software | Bus Architecture

ARM Ltd

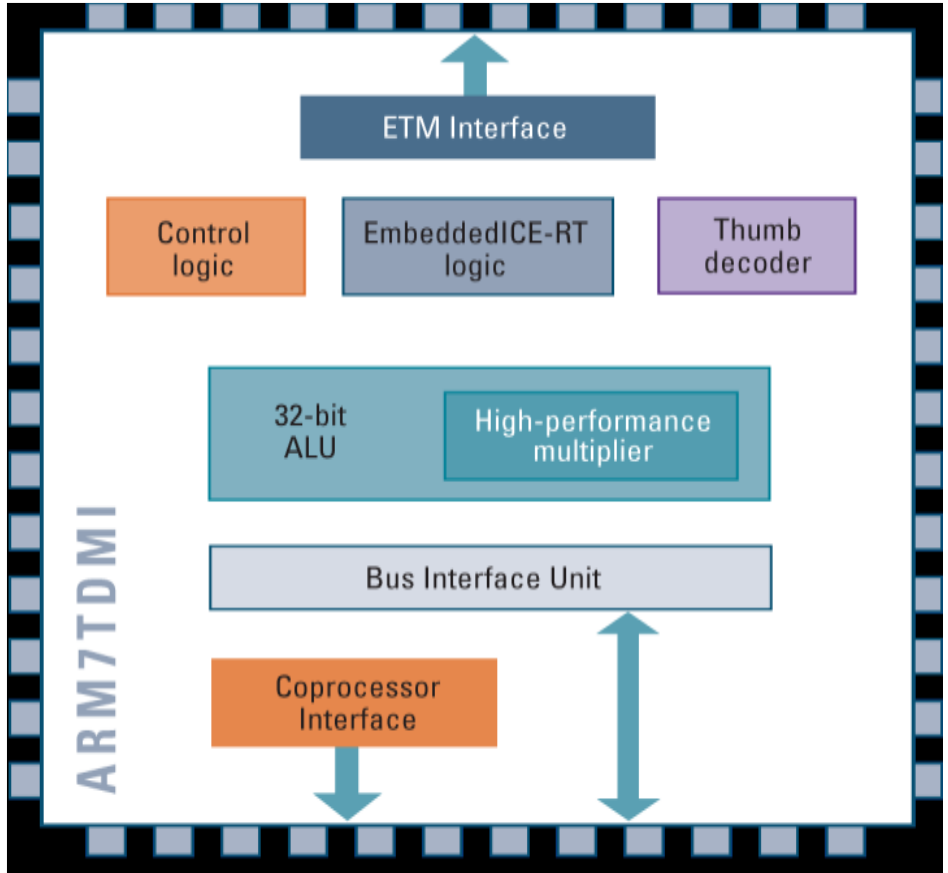
- Shipped in excess of 10 billion units since 1990.
- Over 1300 new chip designs with ARM IP in 2007
- Licensed by majority of world’s leading Semiconductor manufacturers
- At least 90 processors are shipped every second.

ARM key Applications



Why ARM

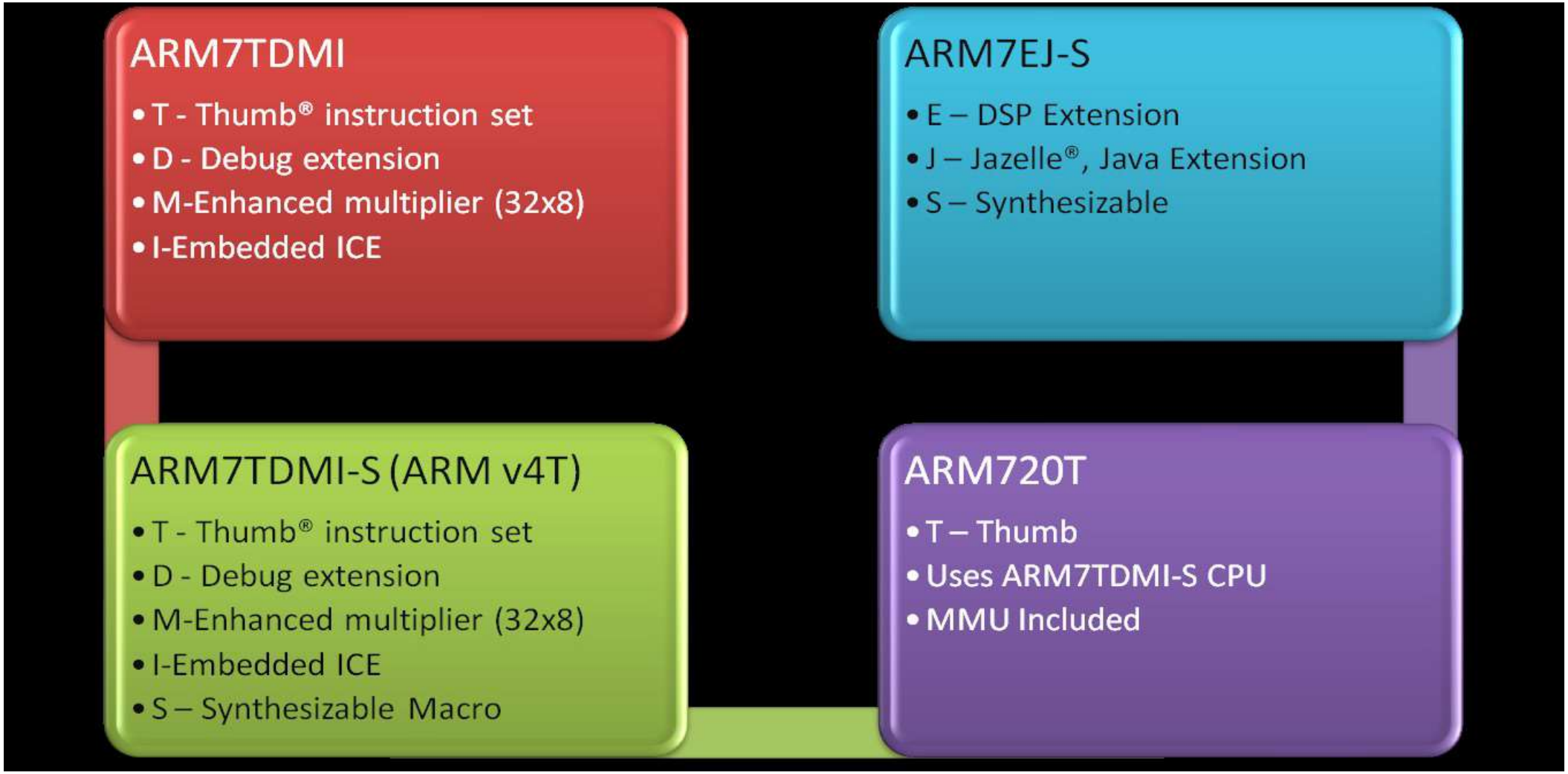
- Built in architecture extensions
  - THUMB®2 - Greatly improved code density
  - DSP - Signal process directly in the RISC core
  - Jazelle® - Java acceleration
  - Trustzone™ - Maximum Security Environment
- Core Performance
- Tools of Choice
- Wide Support
- Low Power Consumption



ARM – Advanced RISC Processor

- Unlike CISC processors, RISC engines generally execute each instruction in a single clock cycle, which typically results in faster execution than on a CISC processor with the same clock speed.
- A RISC processor typically needs more memory than a CISC does to store the same program.
- Except in the most speed-critical of embedded devices, the cost of memory is much more critical than the execution speed of the processor.
- To reduce memory requirements and, thereby, cost, Advanced RISC Machines (ARM) created the Thumb instruction set(16-bit) as an option for their RISC processor cores.

ARM7 Family


ARM7TDMI
• T - Thumb® instruction set
• D - Debug extension
• M-Enhanced multiplier (32x8)
• I-Embedded ICE
ARM7EJ-S
• E – DSP Extension
• J – Jazelle®, Java Extension
• S – Synthesizable
ARM7TDMI-S (ARM v4T)
• T - Thumb® instruction set
• D - Debug extension
• M-Enhanced multiplier (32x8)
• I-Embedded ICE
• S – Synthesizable Macro
ARM720T
• T – Thumb
• Uses ARM7TDMI-S CPU
• MMU Included

ARM7TDMI

ARM7TDMI stands for

- **T**: THUMB
- **D:** for on-chip Debug support, enabling the processor to halt in response to a debug request,
- **M**: enhanced Multiplier, yield a full 64-bit result, high performance
- **I**: Embedded ICE hardware (In Circuit emulator)

Features of ARM7TDMI

- ARM7 is a 32-bit architecture
- Data Paths and Instructions (ARM) are 32 bits wide
- Von-Neumann Architecture
  - Instructions and Data use same bus
- Thumb Mode
  - Subset of 16-bit instructions
  - Code density optimization

Features of ARM7TDMI

- 32-bit microprocessors.
- The ARM family offers high performance for very low-power consumption and gate count.
- The ARM7TDMI processor has a Von Neumann architecture, with a single 32-bit data bus carrying both instructions and data.
- Only load, store, and swap instructions can access data from memory.
- The ARM7TDMI processor uses a three stage pipeline- **FETCH, DECODE, EXECUTE** to increase the speed of the flow of instructions to the processor.
- This enables several operations to take place simultaneously, and the processing, and memory systems to operate continuously. In the three-stage pipeline the instructions are executed in three stages.

Architectural Features of ARM7TDMIS

The typical RISC architectural features of ARM are :

- A large uniform register file
- A load/store architecture, where data-processing operations only operate on register contents, not directly on memory contents
- Simple addressing modes, with all load/store addresses being determined from register contents and instruction fields only uniform and fixed-length instruction fields, to simplify instruction decode.
- Control over both the Arithmetic Logic Unit (ALU) and shifter in most data-processing instructions to maximize the use of an ALU and a shifter
- Auto-increment and auto-decrement addressing modes to optimize program loops
- Load and Store Multiple instructions to maximize data throughput
- Conditional execution of almost all instructions to maximize execution throughput.

Architecture of ARM7

- The ARM 7 processor has a single bus for both data and instructions
- The single bus system decreases the performance of ARM, it is overcome by the pipe line concept.
- 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).
- 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.

Architecture of ARM7

In addition to the 37 registers, ARM also consists of

- one Program status register of 32 bits,
- special registers like:
  - instruction register,
  - memory data read and write register
  - memory address register ,
  - one Priority encoder which is used in the multiple load and store instruction to indicate which register in the register file to be loaded or stored and Multiplexers etc.

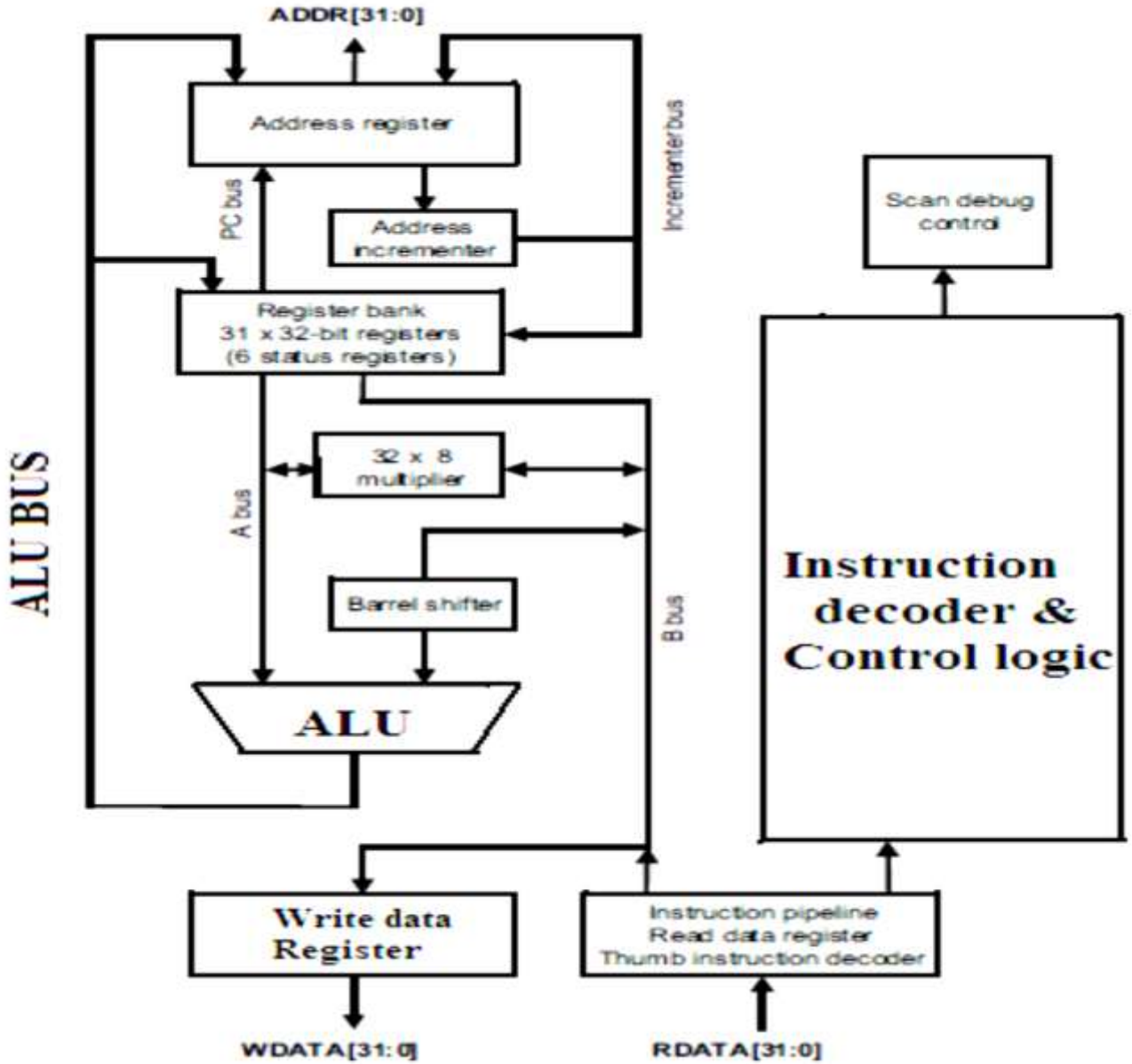
Register in ARM7

- ARM has 37 registers all of which are 32 bits long (31 general Purpose and 6 Status registers)
  - 1 dedicated Program Counter
  - 30 General Purpose Register
  - 1 dedicated Current Program Status Register (CPSR)
  - 5 dedicated Saved Program Status Register (SPSR)

- The current processor mode governs the access of register banks
  - A particular set of r0-r12 registers
  - A particular r13 (stack pointer) and r14 (link register, LR)
  - The program counter (PC)
  - The current program status register (CPSR)

- Privileged Mode can access
  - A particular SPSR (Saved Program Status Register)

Architecture of ARM7



Barrel Shifter

- A barrel shifter is a digital circuit that can shift a data word by a specified number of bits in one clock cycle.
- It can be implemented as a sequence of multiplexers (mux.), and in such an implementation the output of one mux is connected to the input of the next mux in a way that depends on the shift distance.
- A barrel shifter is often implemented as a cascade of parallel 2×1 multiplexers.

Architecture Operation

- Most of the ARM instructions typically have two source registers and a single result or destination register. However, move, comparison and branch instructions do not support such 3-address format.
- The source operands are read from register bank using the internal buses A and B respectively. The second operand, which is passed to the barrel shifter through the internal bus B, is shifted.
- Then the ALU takes the registers values from the internal bus A and barrel shifter and computes the result. The data processing instructions write the result directly to the destination register of the register bank using the ALU bus.
- The load and store instructions use the ALU to generate an address, which is sent to the address register using the ALU bus.
- The PC value in the address register is incremented before the next register value from or to the next memory location is read or written. This process continues until the interrupt or exception occurs

Register in ARM7

- The Arm processor has 37 registers in total. These include a program counter, six program status registers and 30 general-purpose registers.
- At any one time only 18 of these registers: 16 data registers and 2 processor status registers are visible depending upon the processor mode.
- All the registers are 32 bits in size. Each processor mode can use the r0-r12 general-purpose registers, r13: the stack pointer and r14-link register, r15-the program counter and program status register.

ARM Processor Modes

- Seven basic operating modes exist:
  1. User: Unprivileged mode under which most tasks run
  2. FIQ: Entered when a high priority interrupt is raised
  3. IRQ: Entered when a low priority interrupt is raised
  4. Supervisor: Entered on reset and when a software Interrupt instruction is executed
  5. Abort: Used to handle memory access violations
  6. Undef: Used to handle undefined instructions
  7. System: Privileged mode using the same registers as user mode.

ARM Processor Modes

- Unprivileged Mode
- Most tasks run on this mode

- Entered when a high priority Interrupt is occurred

- Entered when a low priority interrupt (normal) is occurred

- Entered on Reset
- Entered on Software Interrupt

- To handle memory access violations

- To handle un-defined instructions

- Privileged mode using the same registers as user mode

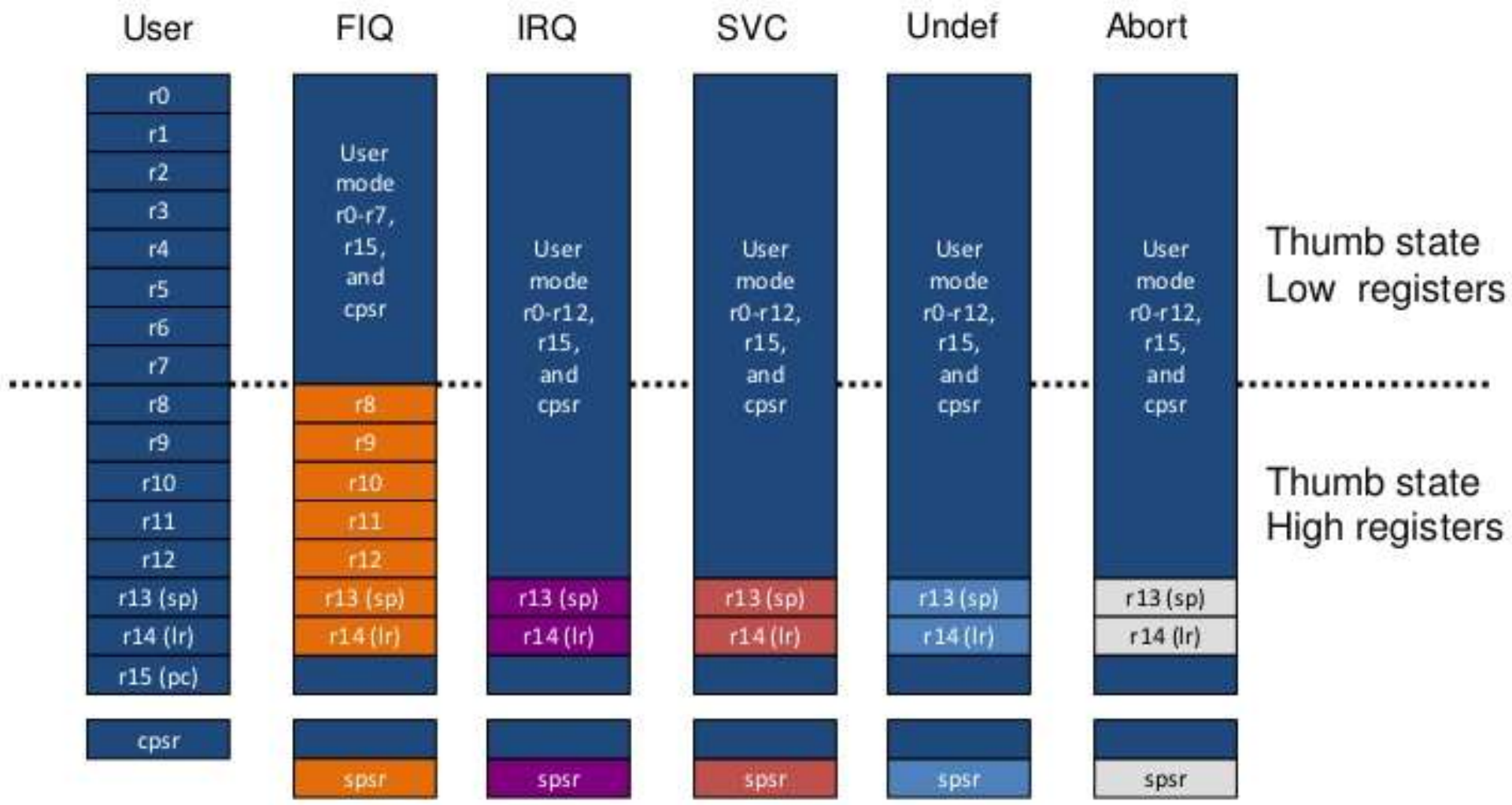
Register in ARM7

- The general-purpose registers r0-r8 are available no matter which mode is the processor.
- A few registers r8-r12 are common to all the processor modes except in the fast interrupt request mode.
- When the processor is in the fast interrupt mode, these registers r8-r12 are replaced with different set of registers r8_q-r12_q.

Register in ARM7

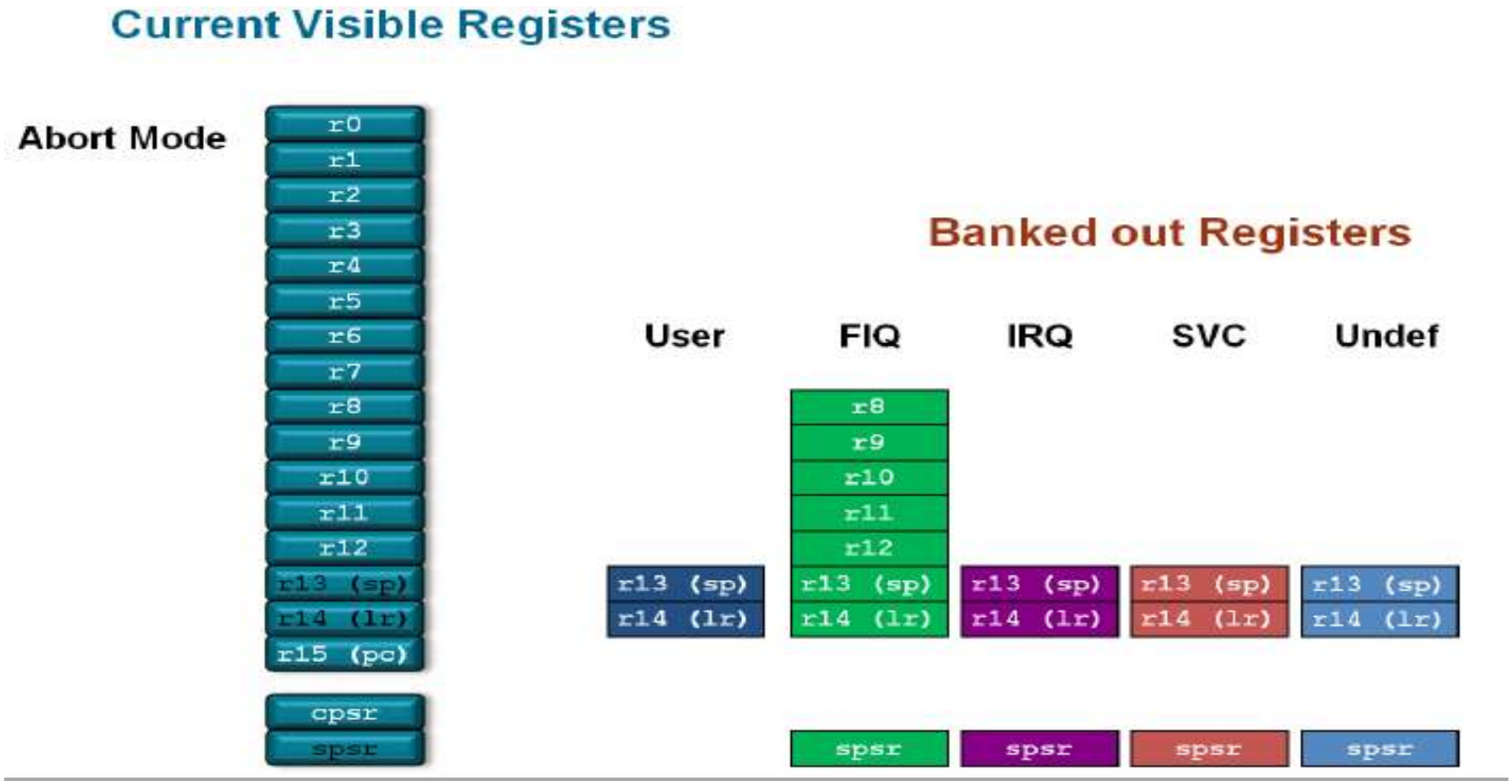
- There are two program status registers: current program status register (CPSR) and saved program status register (SPSR).
- The SPSR can only be modified and read in the privileged mode and the CPSR is common to all the modes.
- However, there is no SPSR available in the user mode.
- The CPSR contains the current status of the processor. It is used to monitor and control internal operations. It is 32-bit register. The CPSR has four different fields with 8 bits width each. They are flags, status, extension and control. The saved program status register contains the copy of CPSR when a processor enters a new privileged mode.

Register Organization for Processor Modes

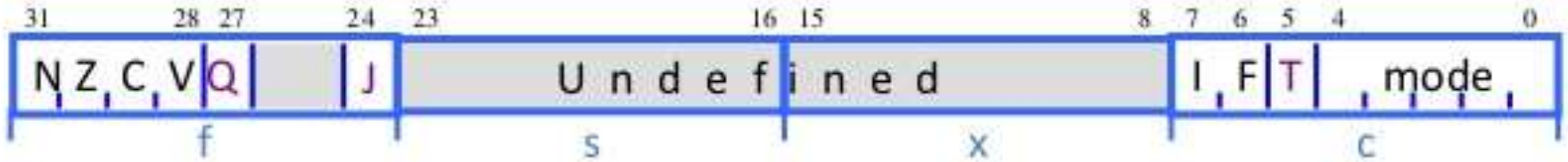


Note: System mode uses the User mode register set

Processor Modes- Register Visible



Program Status Register



- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation overflowed
- 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

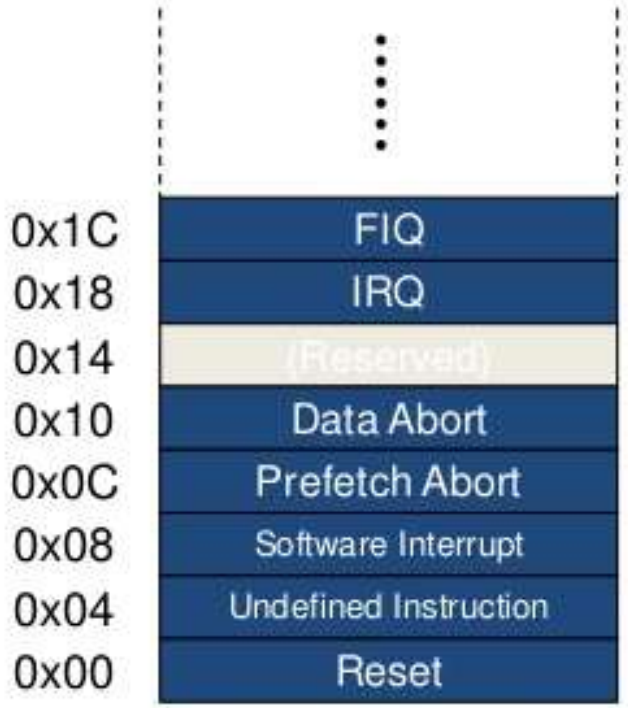
Program Counter (r15)

- 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

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

Features of ARM7TDMI

The ARM7TDMI core uses a pipeline

– Increase the speed of the flow of instructions

– Enables several operations to take place simultaneously

– Program Counter (PC) points to instruction being fetched rather than that being executed

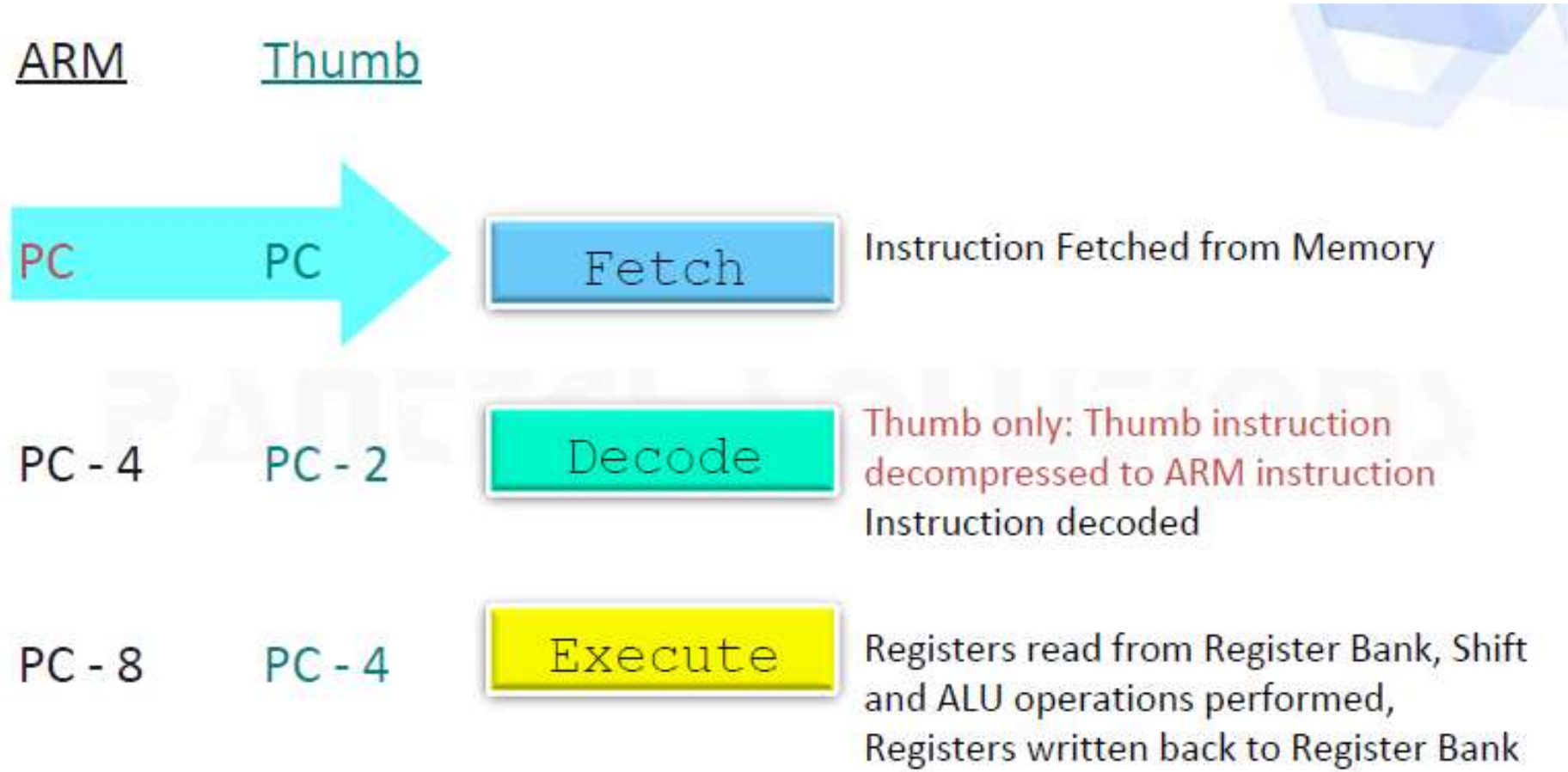
- During normal pipelined operation

– An instruction x is executed

– Instruction (x-1) is being decoded

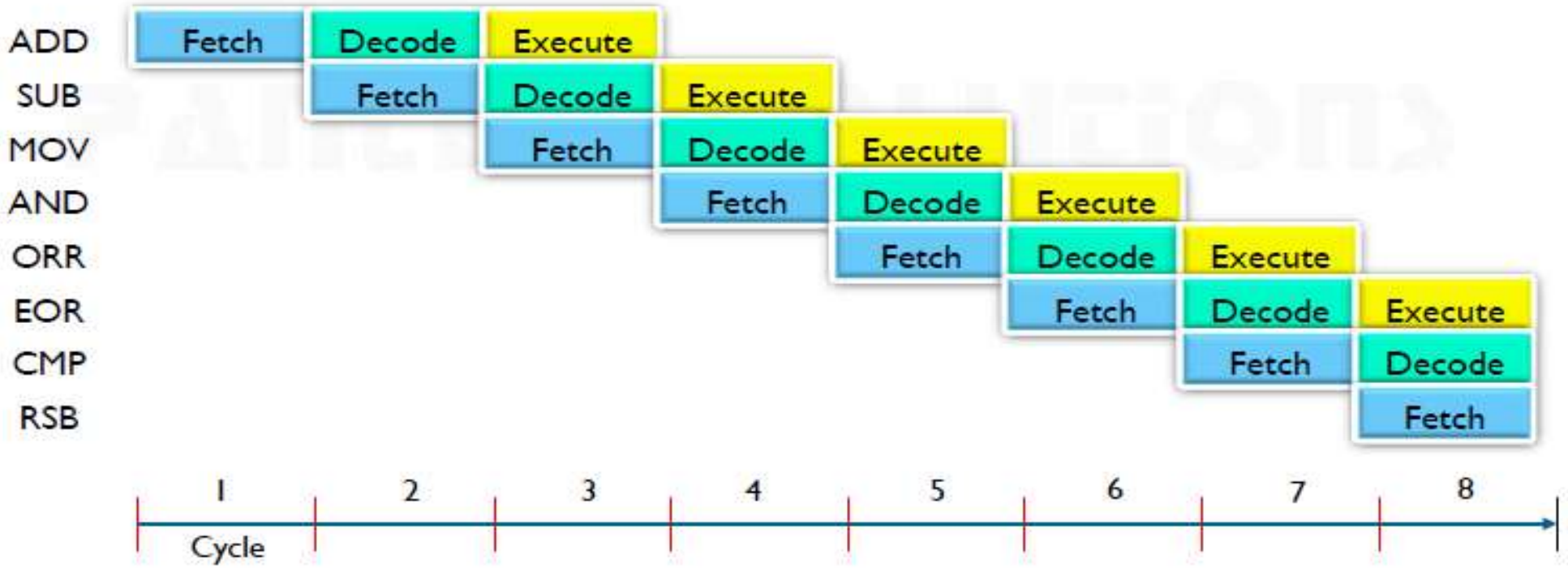
– Instruction (x-2) is being fetched

Pipelining in ARM7TDMIS



Pipelining in ARM7TDMIS

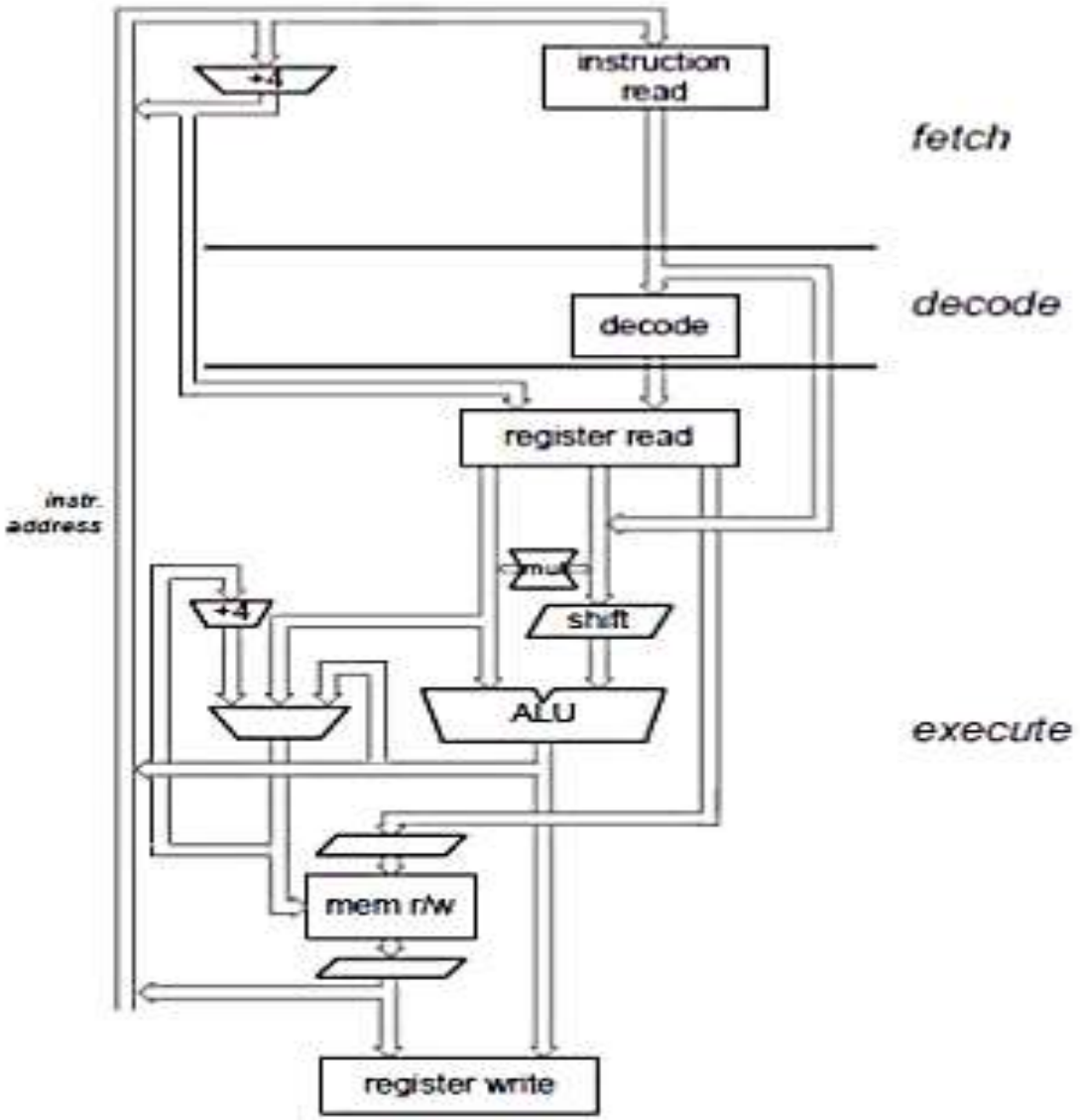
- In this example it takes 6 clock cycles to execute 6 instructions
- All operations are on registers (single cycle instructions)
- Clock cycles per instruction (CPI) = 1



Pipelining in ARM7TDMIS

- 3-stage pipeline organization
  - Principal components
    - The register bank
    - The barrel shifter
      - Can shift or rotate one operand by any number of bits
    - The ALU
    - The address register and incrementer
      - Select and hold all memory addresses and generate sequential addresses
    - The data registers
    - The instruction decoder and associated control logic

Pipelining in ARM7TDMIS



- Fetch - The instruction is fetched from memory and placed in the instruction pipeline
- Decode - The instruction is decoded and the datapath control signals prepared for the next cycle
- Execute - The register bank is read, an operand shifted, the ALU result generated and written back into destination register

Pipelining in ARM7TDMIS

- At any time slice, 3 different instructions may occupy each of these stages, so the hardware in each stage has to be capable of independent operations

- When the processor is executing data processing instructions , the latency = **3** cycles and the throughput = **1** instruction/cycle

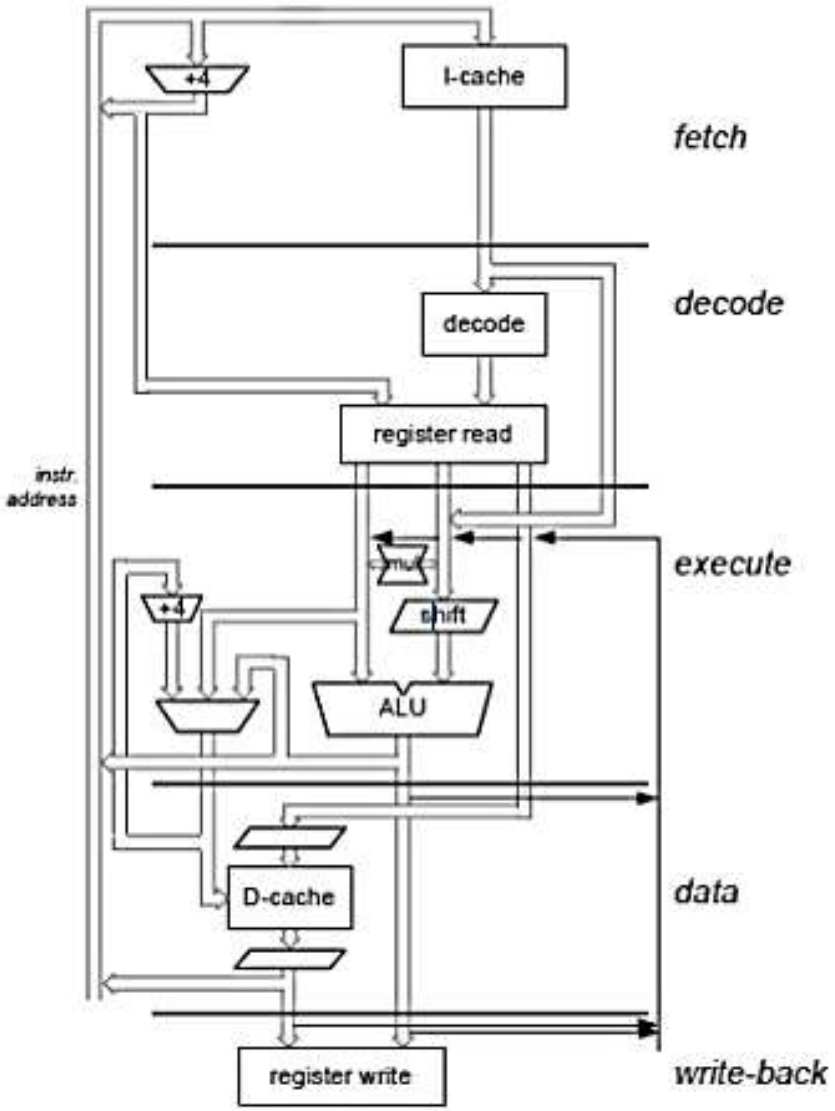
- Drawback: Every data transfer instruction causes a pipeline “stall”. (Single memory for data and instruction- next instruction cannot be fetched while data is being read)

Pipelining in ARM7TDMIS

5-stage Pipeline Organization

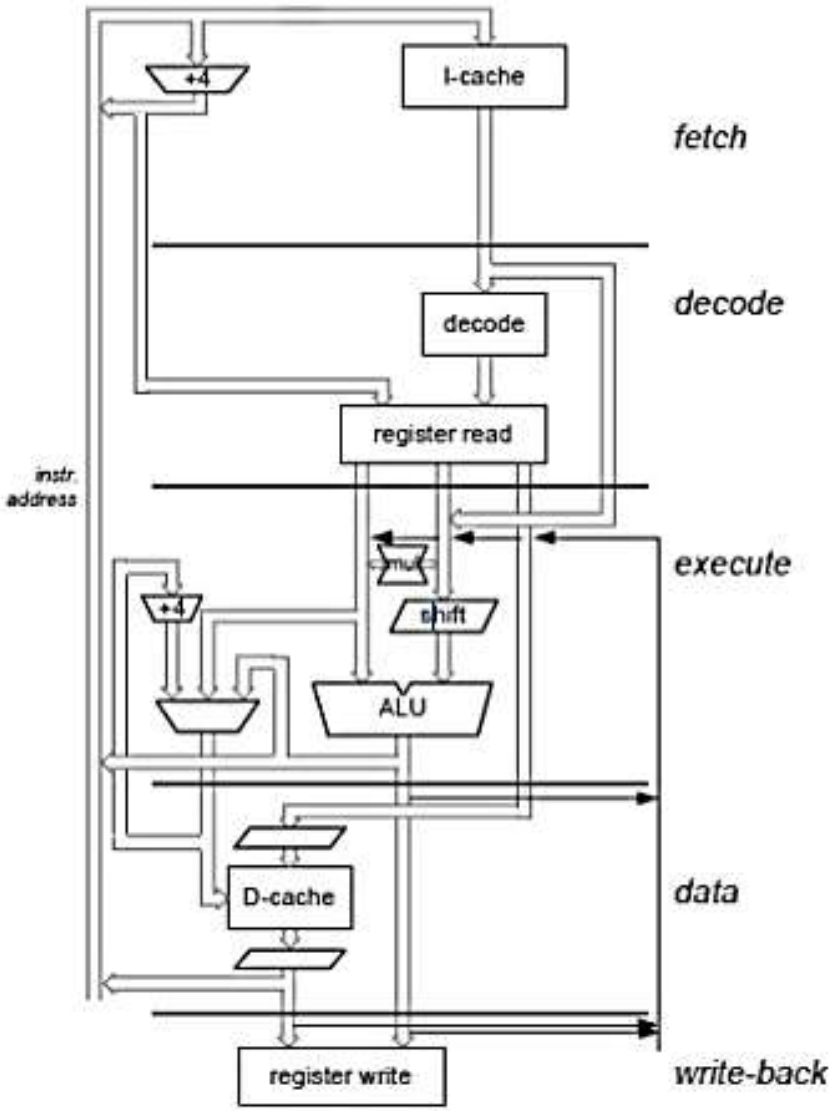
- Implemented in ARM9TDMI
- $T_{prog} = N_{inst} * CPI / f_{clk}$
  - T_{prog}: the time taken to execute a given program
  - N_{inst}: the number of ARM instructions executed in the program (compiler dependent)
  - CPI: average number of clock cycles per instructions => hazard causes pipeline stalls
  - f_{clk}: frequency

Pipelining in ARM7TDMIS



- 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

Pipelining in ARM7TDMIS



- 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.

Pipelining in ARM7TDMIS

- Moved the register read step from the execute stage to the decode stage
- Execute stage was split into 3 stages- ALU, memory access, write back.
- Result: Better balanced pipeline with minimized latencies between stages, which can run at a faster clock speed.

Pipeline Hazards

- There are situations, called hazards, that prevent the next instruction in the instruction stream from being executed during its designated clock cycle. Hazards reduce the performance from the ideal speedup gained by pipelining.
- There are three classes of hazards:
  - Structural Hazards
  - Data Hazards
  - Control Hazards

Pipeline Hazards

Structural Hazards

- When a machine is pipelined, the overlapped execution of instructions requires pipelining of functional units and duplication of resources to allow all possible combinations of instructions in the pipeline.
- If some combination of instructions cannot be accommodated because of a *resource conflict*, the machine is said to have a **structural hazard**.

Pipeline Hazards

- Ex. A machine has shared a **single-memory** pipeline for data and instructions. As a result, when an instruction contains a data-memory reference (load), it will conflict with the instruction reference for a later instruction (instr 3):

Clock cycle number								
instr	1	2	3	4	5	6	7	8
load	IF	ID	EX	**MEM**	WB			
Instr 1		IF	ID	EX	MEM	WB		
Instr 2			IF	ID	EX	MEM	WB	
Instr 3				**IF**	ID	EX	MEM	WB

Pipeline Hazards

- To resolve this, we *stall* the pipeline for one clock cycle when a data-memory access occurs. The effect of the stall is actually to occupy the resources for that instruction slot. The following table shows how the stall is actually implemented.

Clock cycle number									
instr	1	2	3	4	5	6	7	8	9
load	IF	ID	EX	**MEM**	WB				
Instr 1		IF	ID	EX	MEM	WB			
Instr 2			IF	ID	EX	MEM	WB		
Instr 3				**stall**	**IF**	ID	EX	MEM	WB

Pipeline Hazards

- Another solution is to use separate instruction and data memories.
- ARM has moved from the von-Neumann architecture to the Harvard architecture in ARM9.
  - Implemented a 5-stage pipeline and separate data and instruction memory.
  - Doesn't suffer from this hazard.

Pipeline Hazards

Data Hazards

- They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.
- The problem with data hazards can be solved with a hardware technique called data forwarding (by making use of feedback paths).
- Without forwarding, the pipeline would have to be stalled to get the results from the respective registers
- Example:

Clock cycle number								
		1	2	3	4	5	6	7
ADD	R1,R2,R3	IF	ID	EX_{add}	MEM_{add}	WB		
SUB	R4,R5,R1		IF	ID	EX_{sub}	MEM	WB	
AND	R6,R1,R7			IF	ID	EX_{and}	MEM	WB

Pipeline Hazards

Data Hazards

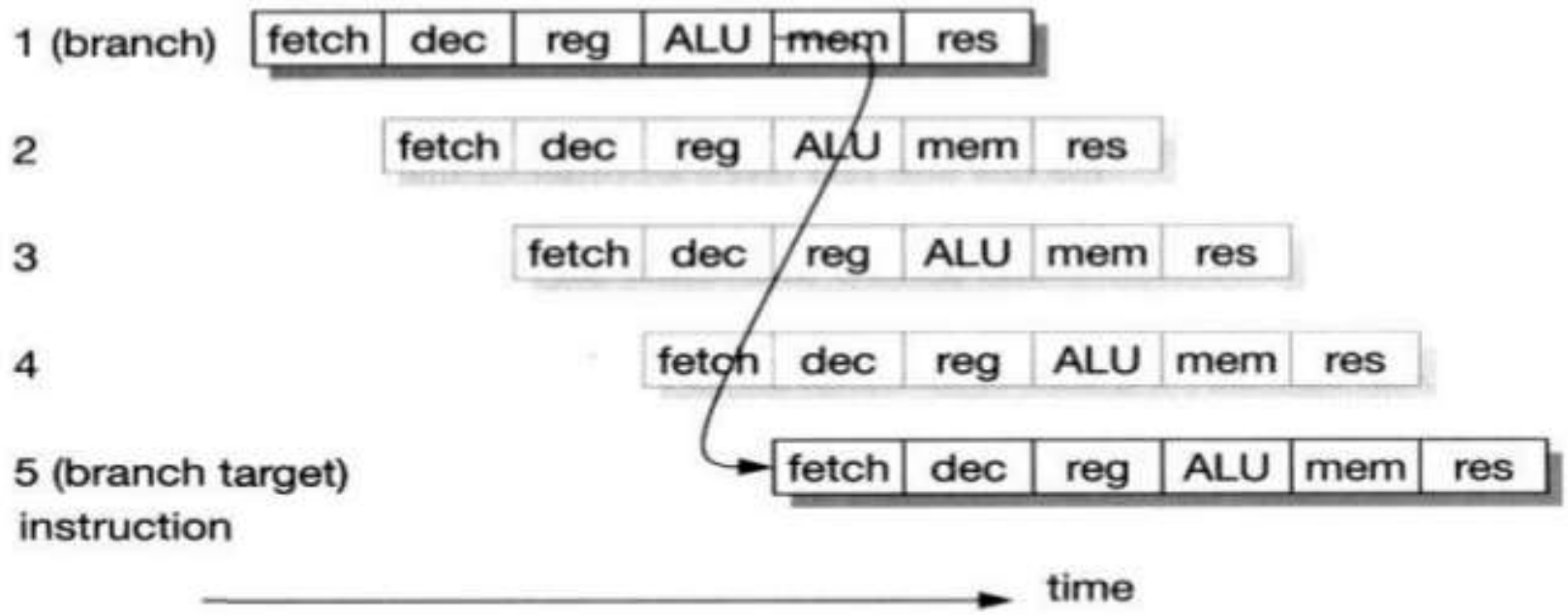
Clock cycle number								
		1	2	3	4	5	6	7
ADD	R1,R2,R3	IF	ID	EX_{add}	MEM_{add}	WB		
SUB	R4,R5,R1		IF	ID	EX_{sub}	MEM	WB	
AND	R6,R1,R7			IF	ID	EX_{and}	MEM	WB

- The first forwarding is for value of **R1** from $\mathbf{EX_{add}}$ to $\mathbf{EX_{sub}}$.
- The second forwarding is also for value of **R1** from $\mathbf{MEM_{add}}$ to $\mathbf{EX_{and}}$.
- This code now can be executed without stalls.
- Forwarding can be generalized to include passing the result directly to the functional unit that requires it: a result is forwarded from the output of one unit to the input of another, rather than just from the result of a unit to the input of the same unit.

Pipeline Hazards

Control Hazards

- They arise from the pipelining of branches and other instructions that change the PC.



Instruction Set of ARM7

- DATA when used in ARM
  - Byte : 8 bits
  - Half Word : 16 bits (2 Bytes)
  - Word : 32 bits (4 Bytes)
- Instruction Set of most ARMs
  - 32 bits ARM Instruction Set
  - 16 bits Thumb Instruction Set
- The Thumb instruction set is a subset of the most commonly used 32-bit ARM instructions.
- Thumb instructions operate with the standard ARM register configurations, enabling excellent interoperability between ARM and Thumb states.
- This Thumb state is nearly 65% of the ARM code and can provide 160% of the performance of ARM code when working on a 16-bit memory system.
- This Thumb mode is used in embedded systems where memory resources are limited.

Features of ARM7 Instruction Set Architecture(ISA)

- All instructions are 32 bits long.
- Most instructions execute in a single cycle.
- Every instruction can be conditionally executed.
- A load/store architecture
  - Data processing instructions act only on registers
    - Three operand format
    - Combined ALU and shifter for high speed bit manipulation
  - Specific memory access instructions with powerful auto-indexing addressing modes.

ARM7 Instruction Set Architecture(ISA)

- The ARM7TDMI processor core implements the ARMv4T *Instruction Set Architecture* (ISA).
- The ARMv4T ISA is a superset of the ARMv4 ISA (implemented by the Strong ARM processor) with additional support for the Thumb 16-bit compressed instruction set.
- The ARMv4T ISA is implemented by the ARM7 Thumb family. This gives full upward compatibility to the ARM9 Thumb family.