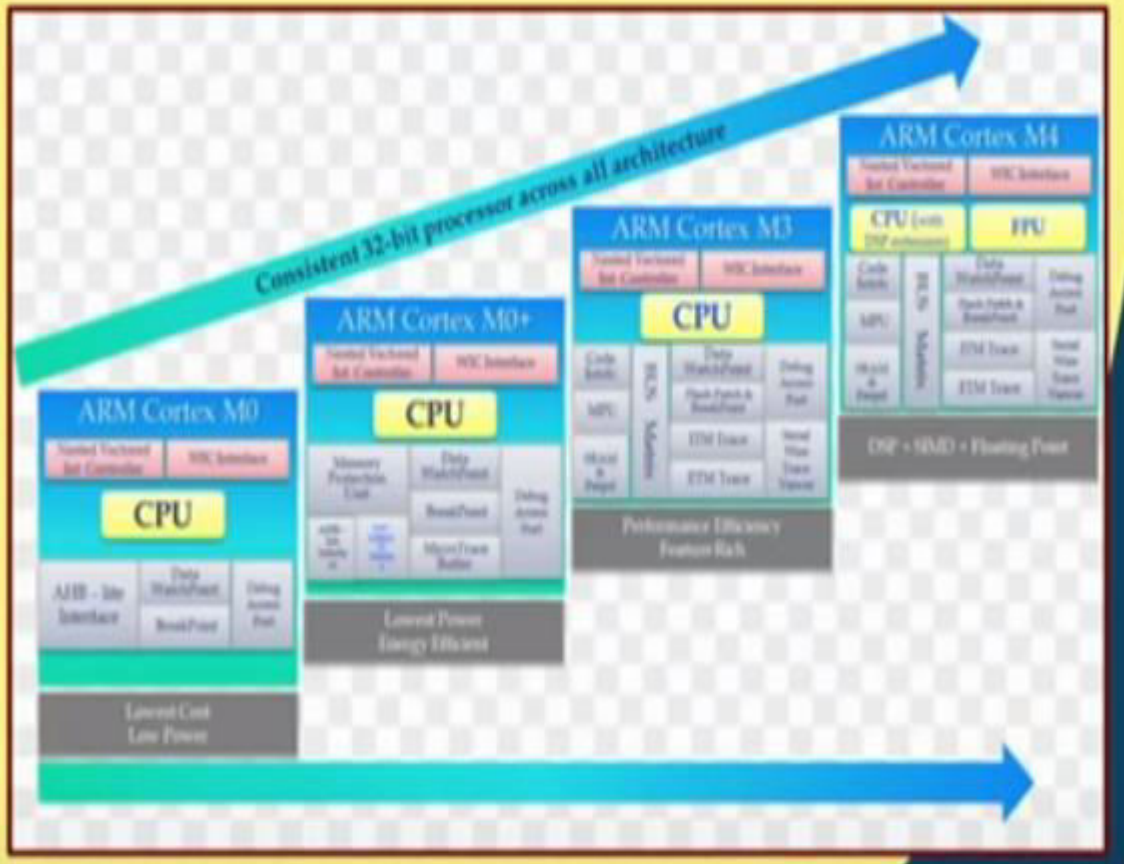
ARM Organization & Implementation

History of ARM Series of Microcontrollers

- Architectural ideas developed in 1983 by Acorn Computers.
  - To replace the 8-bit 6502 microprocessor in BBC computers.
  - The first commercial RISC implementation.
- The company founded in 1990.
  - Advanced RISC Machine (ARM).
  - Initially owned by Acorn, Apple and VLSI.

Why do we talk about ARM?

- One of the most widely used processor cores.
- Some application examples:
  - ARM7: iPod
  - ARM9: BenQ, Sony Ericsson
  - ARM11: Apple iPhone, Nokia N93, N100
  - 90% of 32-bit embedded RISC processors till 2010.
- Mainly used in battery-operated devices:
  - Due to low power consumption and reasonably good performance.



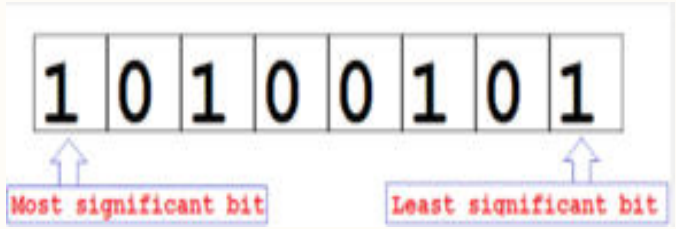
About ARM Processors

- A simple RISC-based architecture with powerful design.
- A whole family of ARM processors exist.
  - Share similar design principles and a common instruction set.
- Design philosophy:
  - Small processor for lower power consumption (for embedded system applications).
  - High code density for limited memory and physical size restrictions.
  - Can interface with slow and low-cost memory systems.
  - Reduced die size for processor to accommodate more peripherals.

Byte Ordering

- Byte ordering, or *endianness*, is another major architectural consideration.
- If we have a two-byte integer, the integer may be stored so that the least significant byte is followed by the most significant byte or vice versa.
  - In *little endian* machines, the least significant byte is followed by the most significant byte.
  - *Big endian* machines store the most significant byte first (at the lower address).

Address	Example Address	Value
Base + 0	1001	..
Base + 1	1002	..
Base + 2	1003	..
Base + ..	..	..



- Ex. Represent the String

APPLE

Byte-Order		Address				
		Base + 0	Base + 1	Base + 2	Base + 3	Base + 4
	Little Endian	E	L	P	P	A
	Big Endian	A	P	P	L	E

Address →	00	01	10	11
Big Endian	12	34	56	78
Little Endian	78	56	34	12

Contd.,

- Why is endianness so important?
  - Suppose you are storing int values to a file, then you send the file to a machine which uses the opposite endianness and read in the value.
  - You'll run into problems because of endianness. You'll read in reversed values that won't make sense.
- Endianness is also a big issue when sending numbers over the network.
  - Again, if you send a value from a machine of one endianness to a machine of the opposite endianness, you'll have problems.
  - This is even worse over the network, because you might not be able to determine the endianness of the machine that sent you the data.

Contd.,

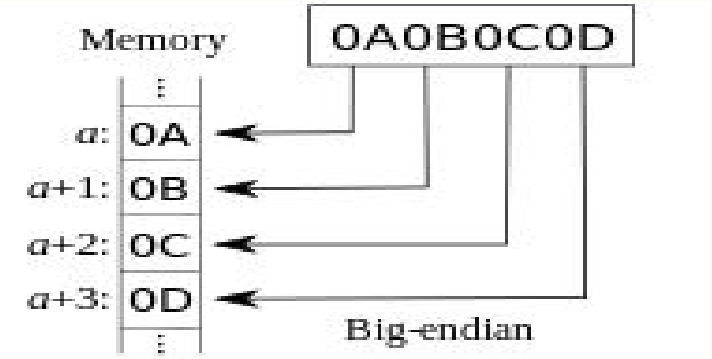
- Where does this term "endian" come from? Jonathan Swift was a satirist (he poked fun at society through his writings).
- His most famous book is "Gulliver's Travels", and he talks about how certain people prefer to eat their hard boiled eggs from the little end first (thus, little endian), while others prefer to eat from the big end (thus, big endians) and how this lead to various wars.

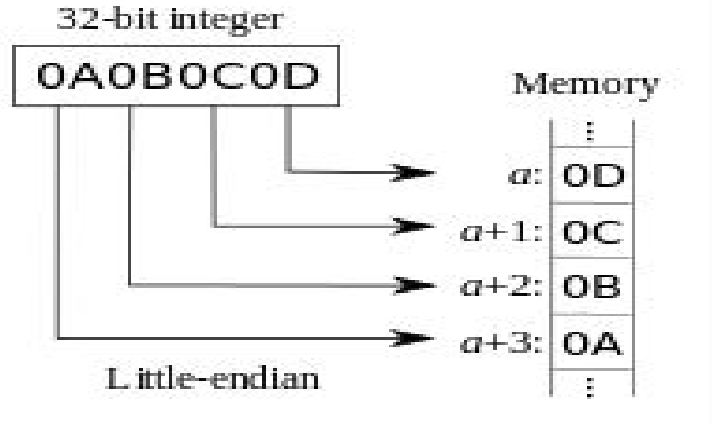
Contd.,

- Endianness only makes sense when you want to break a large value (such as a word) into several small ones.
  - You must decide on an order to place it in memory. However, if you have a 32 bit register storing a 32 bit value, it makes no sense to talk about endianness.
  - The register is *neither* big endian *nor* little endian. It's just a register holding a 32 bit value.
  - The rightmost bit is the least significant bit, and the leftmost bit is the most significant bit.

Contd.,

- Big endian:
  - Is more natural.
  - The sign of the number can be determined by looking at the byte at address offset 0.
  - Strings and integers are stored in the same order.
- Little endian:
  - Conversion from a 16-bit integer address to a 32-bit integer address does not require any arithmetic.





Contd.,

- Examples of **Little Endian**
  - BMP
  - RTF
  - MSPaint
- Examples of **Big Endian**
  - JPEG
  - Adobe Photoshop
  - MacPaint

Organization Vs. Architecture

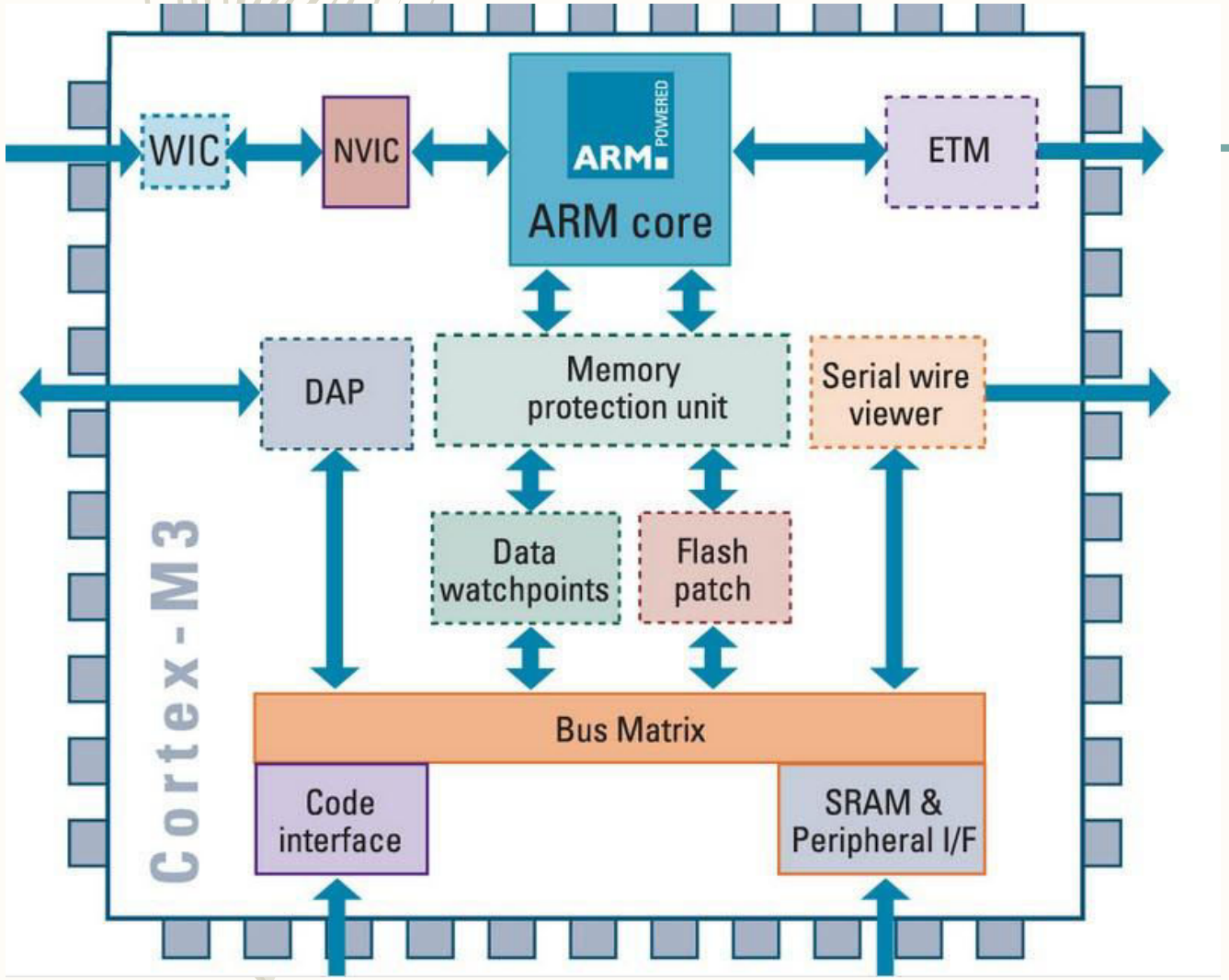
- Architecture
  - how various components in a computer are organized & the design aspect of a computer.
  - Describes users view of the computer
  - Eg: instruction set, visible registers, memory management table structures.
- Computer Organization
  - is how features are implemented, typically hidden from the programmer. E.g. control signals, interfaces, memory technology etc.
  - I.e., describes the user-invisible implementation of architecture.
  - It is basically concerned with the way the hardware components operate and the way they are connected together to form the computer system.

Organization Vs. Architecture

- Architecture refers to how various components in a computer are organized & the design aspect of a computer.

- Computer Organization is how features are implemented, typically hidden from the programmer. E.g. control signals, interfaces, memory technology etc.
  - It is basically concerned with the way the hardware components operate and the way they are connected together to form the computer system.

ARM Architecture



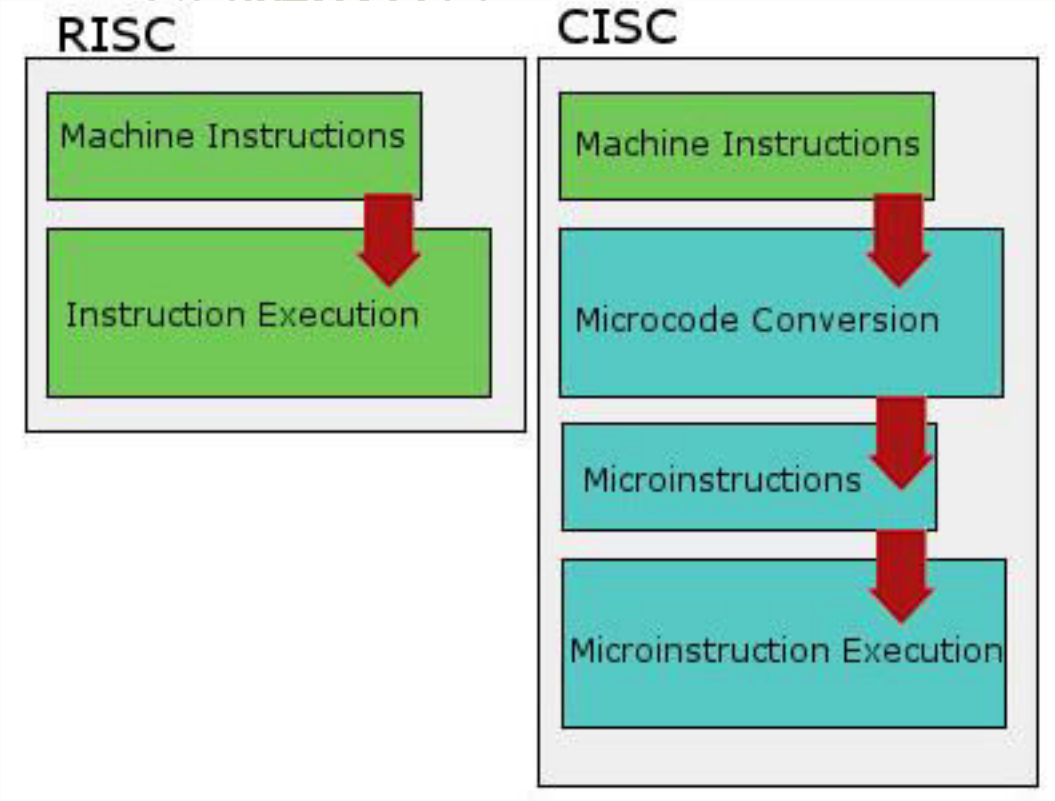
CORE== processing unit or computing engine --this aspect is decided by the architecture, which represents basic design of processor.

Unique feature of ARM as a company == it designs the core and licenses this IP to others

This means that company does not 'fabricate' the chip, but sells only the design.
This design is taken by licensee, who may or maynot add more features to the design.

Different Ways of Selling IP: SOFT IP or Hard IP

The Key To ARM Is RISC



ARM is an acronym which stands for Advanced RISC Machine. In the name is another acronym (yes, nested acronyms!) which stands for Reduced Instruction Set Computing.

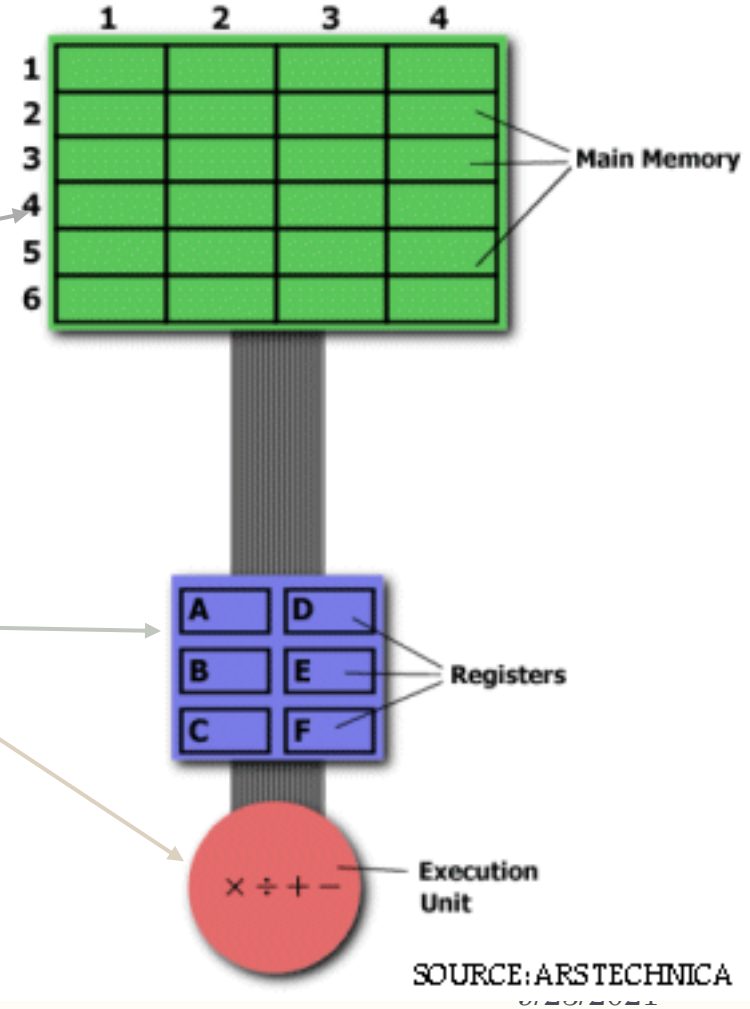
RISC is, in its broadest form, a design philosophy for processors. It stems from a belief that a processor with a relatively simple instruction set will be more efficient than one which is more complex.

[The term originally came into use back in the 1980s with a research project called Berkeley RISC that investigated the possibilities of this approach to design and then created processors based on it].

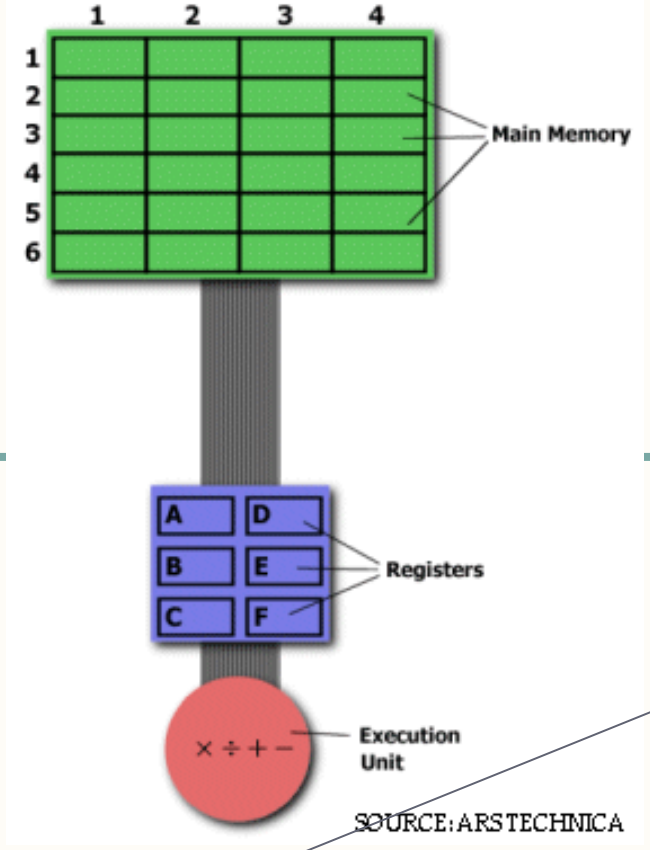
All ARM processors are considered RISC designs, but this doesn't mean much because RISC itself is simply an approach to design rather than a technological standard or processor architecture. Still, a basic understanding of RISC properly frames ARM.

Contd.,
An Example..
Multiplying two numbers in memory

- On the right is a diagram representing the storage scheme for a generic computer.
- The main memory is divided into locations numbered from (row) 1: (column) 1 to (row) 6: (column) 4.
- The execution unit is responsible for carrying out all computations.
- However, the execution unit can only operate on data that has been loaded into one of the six registers (A, B, C, D, E, or F).
- **Let's say we want to find the product of two numbers - one stored in location 2:3 and another stored in location 5:2 - and then store the product back in the location 2:3.**



Contd.,
THE CISC APPROACH



- The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible.
- This is achieved by building processor hardware that is capable of understanding and executing a series of operations. For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "MULT")
- When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers can be completed with one instruction:

```
MULT 2:3, 5:2
```

- MULT is what is known as a "complex instruction."
- It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions.
- It closely resembles a command in a higher level language

Advantages:

- **Because the length of the code is relatively short, very little RAM is required to store instructions.**
- **The emphasis is put on building complex instructions directly into the hardware**

Summarize…

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

ARM's Business Is Different

- ARM refers to itself as an architecture, which can cause a misunderstanding when compared to Intel.
- Intel gives every new chip design its own unique code and talks about each as a new architecture – even when there's often many similarities and they all use the same instruction set (x86)
- ARM, on the other hand, treats its designs as an unbroken family. Updates are still a part of the ARM architecture. They're just given a new version number.
- The trait that's most relevant to consumers is not the micro-architecture (the physical design of the chip) but instead the instruction set.

- The instruction set
  - is the basic set of capabilities and features a processor makes available to software.
  - It determines what arithmetic can be used, how cache should be allocated and the order in which instructions should be executed.
  - Software designed for one instruction set can't be used on another unless it's revised.
- Micro-architectures and instruction sets
  - can't be separated because the architecture is a physical expression of the instruction set.
- This is why ARM-based processors tend to be small, efficient and relatively slow.
- The simple instruction set requires a small, simple design with fewer transistors.
- Transistors consume power and increase die size (which increases production cost), so having as few as possible is ideal when selecting a processor for a smart phone or tablet.

What is a computer system?

Specification

compute the fibonacci sequence

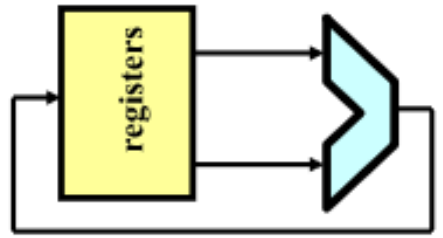
Program

```
for(i=2; i<100; i++) {
  a[i] = a[i-1]+a[i-2];}
```

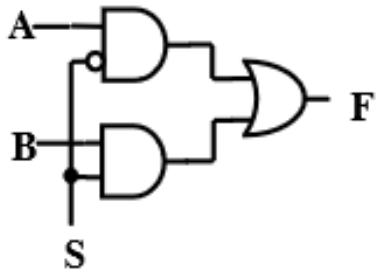
ISA (Instruction Set Architecture)

```
load r1, a[i];
add  r2, r2, r1;
```

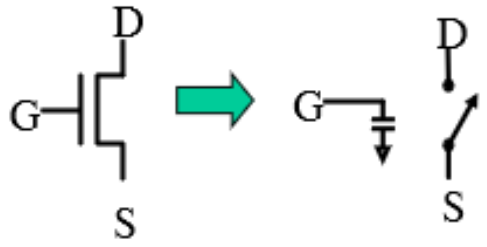
Microarchitecture



Logic



Transistors

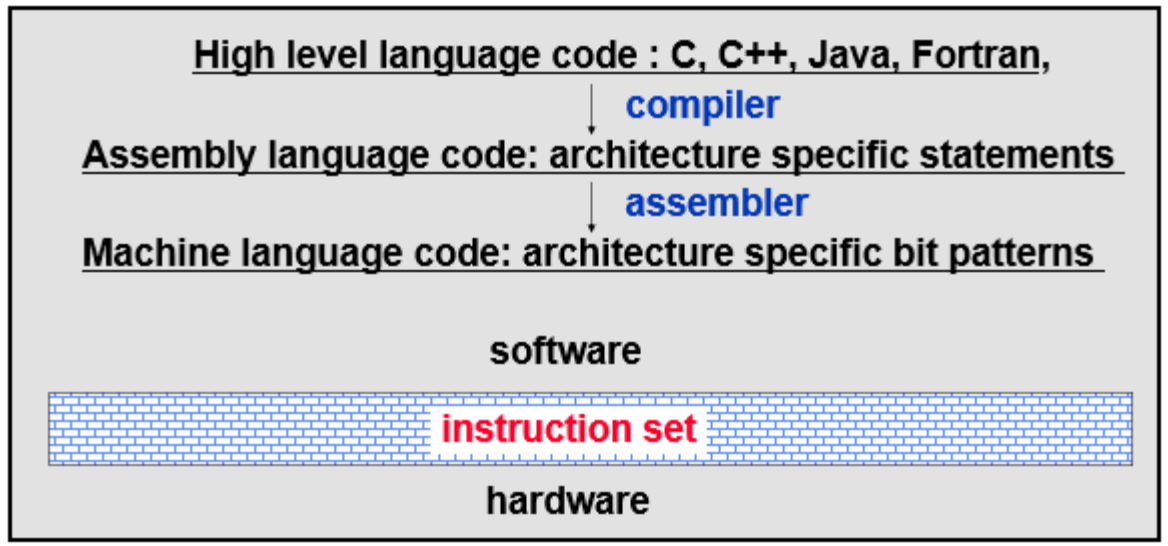


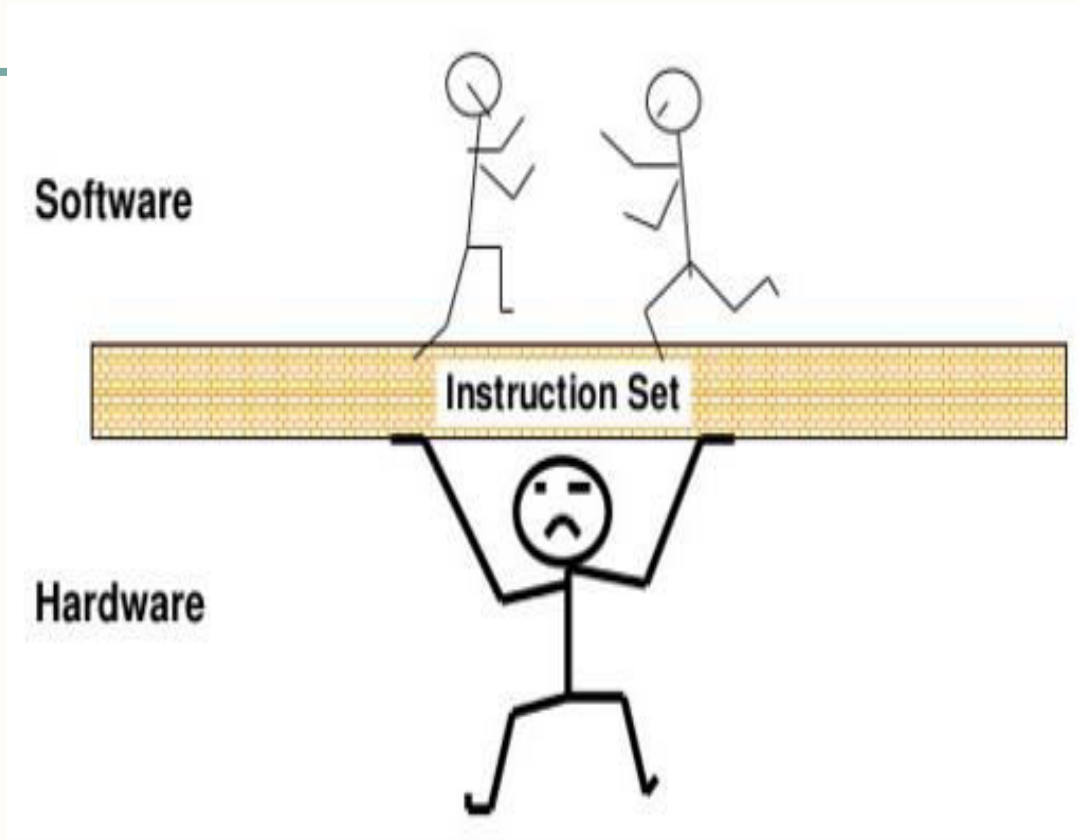
Physics/Chemistry

Instruction Set Architecture

•Serves as an interface between software and hardware.

•Provides a mechanism by which the software tells the hardware what should be done (i.e. instructs the hardware to do the right thing)





What is an instruction?

- ADD A, B
- Here we tell the microprocessor to add the contents of two registers A and B. That is it! This forms the baseline!
- It is like you giving an order to your subordinate! It has to be obeyed!

What do instructions provide?

- Which operation to perform – add r0, r1, r3 // operation code
- Where to find the operand or operands – add r0, r1, r3 // source operands
- Place to store result – add r0, r1, r3 // destination
- Location of next instruction – add r0, r1, r3 br endloop

Popular ARM Architectures

- ARM7
  - 3 pipeline stages (fetch/decode/execute)
  - High code density / low power consumption
  - Most widely used for low-end systems
- ARM9
  - Compatible with ARM7
  - 5 stages (fetch/decode/execute/memory/write)
  - Separate instruction and data cache
- ARM10
  - 6-stages (fetch/issue/decode/execute/memory/write)

ARM Family Comparison

	ARM 7 (1995)	ARM9 (1997)	ARM10 (1999)	ARM11 (2003)
Pipeline depth	3-stage	5-stage	6-stage	8-stage
Typical clock frequency (MHz)	80	150	260	335
Power (mW/MHz)	0.06	0.19	0.50	0.40
Throughput (MIPS/MHz)	0.97	1.1	1.3	1.2
Architecture	Non Neumann	Harvard	Harvard	Harvard
Multiplier	8 x 32	8 x 32	16 x 32	16 x 32

ARM is based on RISC Architecture

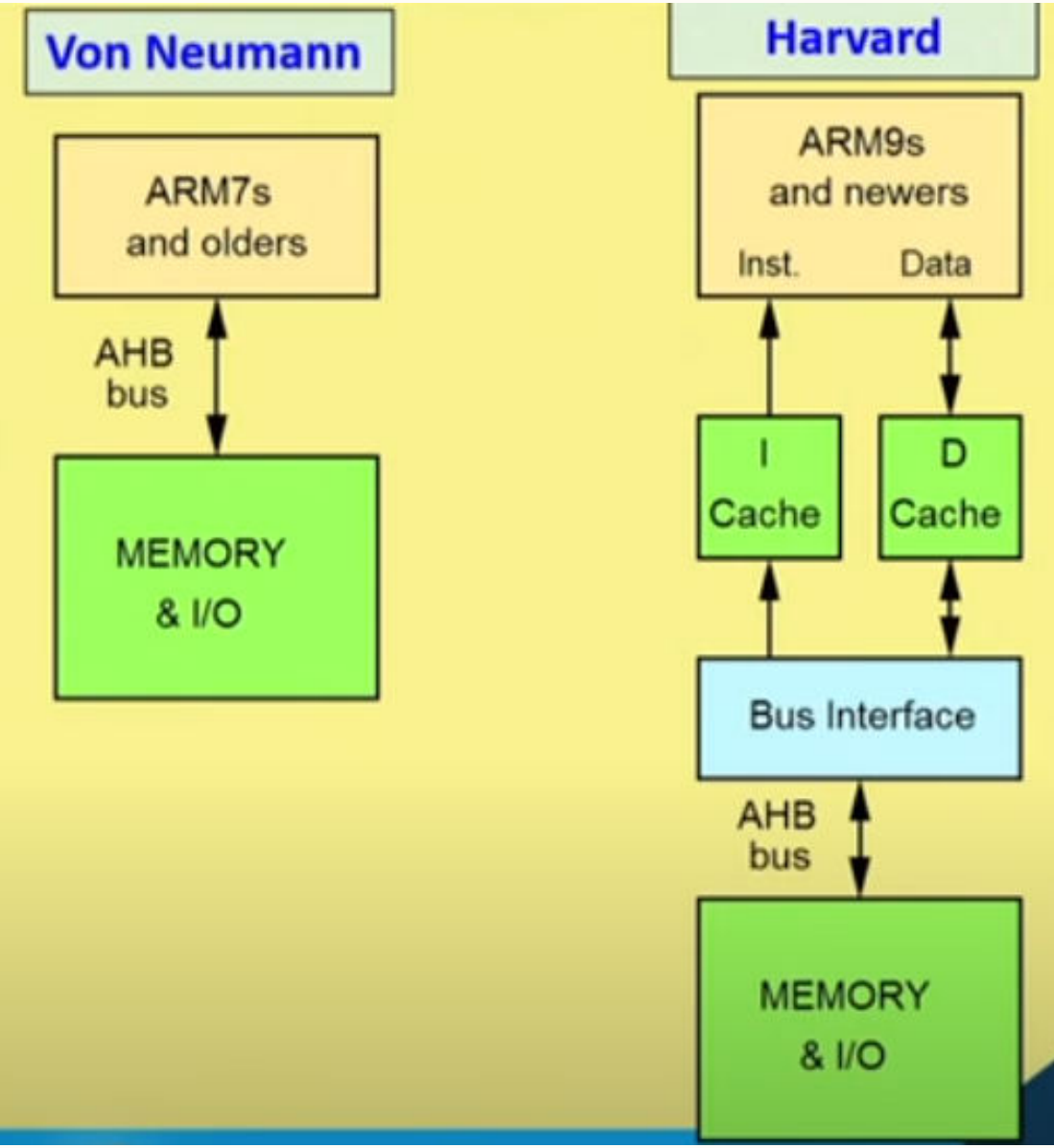
- RISC supports simple but powerful instructions that execute in a single cycle at high clock frequency.
- Major design features:
  - **Instructions**: reduced set / single cycle / fixed length
  - **Pipeline**: decode in one stage / no need for microcode
  - **Registers**: large number of general-purpose registers (GPRs)
  - **Load/Store Architecture**: data processing instructions work on registers only; load/store instructions to transfer data from/to memory.
- Now-a-days *CISC machines also implement RISC concepts*.

ARM Features

- ARM architecture is different from pure RISC:
  - Variable cycle execution for certain instructions (multiple-register load/store for higher code density).
  - In-line barrel shifter results in more complex instructions (improves performance and code density).
  - Thumb 16-bit instruction set (results in improvement in code density by about 30%).
  - Conditional execution (reduces branch and improves performance).
  - Enhanced instructions (some DSP instructions are present).

Memory-mapped I/O:

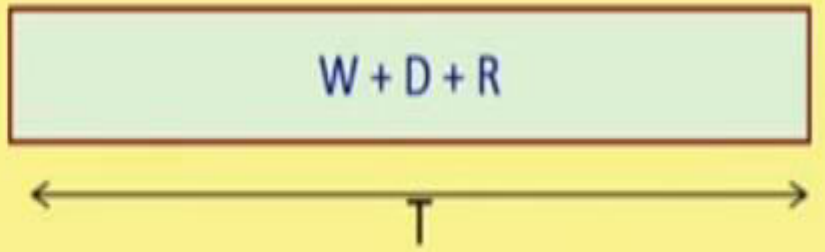
- No specific instructions for I/O
- Use Load/Store instr. for I/O
- Peripheral's registers at some memory addresses



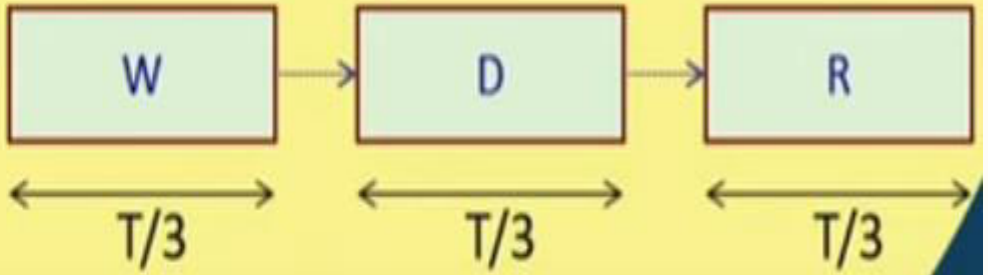
- A mechanism for overlapped execution of several input sets by partitioning some computation into a set of *k* sub-computations (or stages).
  - Very nominal increase in the cost of implementation.
  - Very significant speedup (ideally, *k*).
- Where are pipelining used in a computer system?
  - **Instruction execution**: Several instructions executed in some sequence.
  - **Arithmetic computation**: Same operation carried out on several data sets.
  - **Memory access**: Several memory accesses to consecutive locations are made.

A Real-life Example

- Suppose you have built a machine *M* that can wash (*W*), dry (*D*), and iron (*R*) clothes, one cloth at a time.
  - Total time required is *T*.
- As an alternative, we split the machine into three smaller machines M_W, M_D and M_R, which can perform the specific task only.
  - Time required by each of the smaller machines is *T/3* (say).



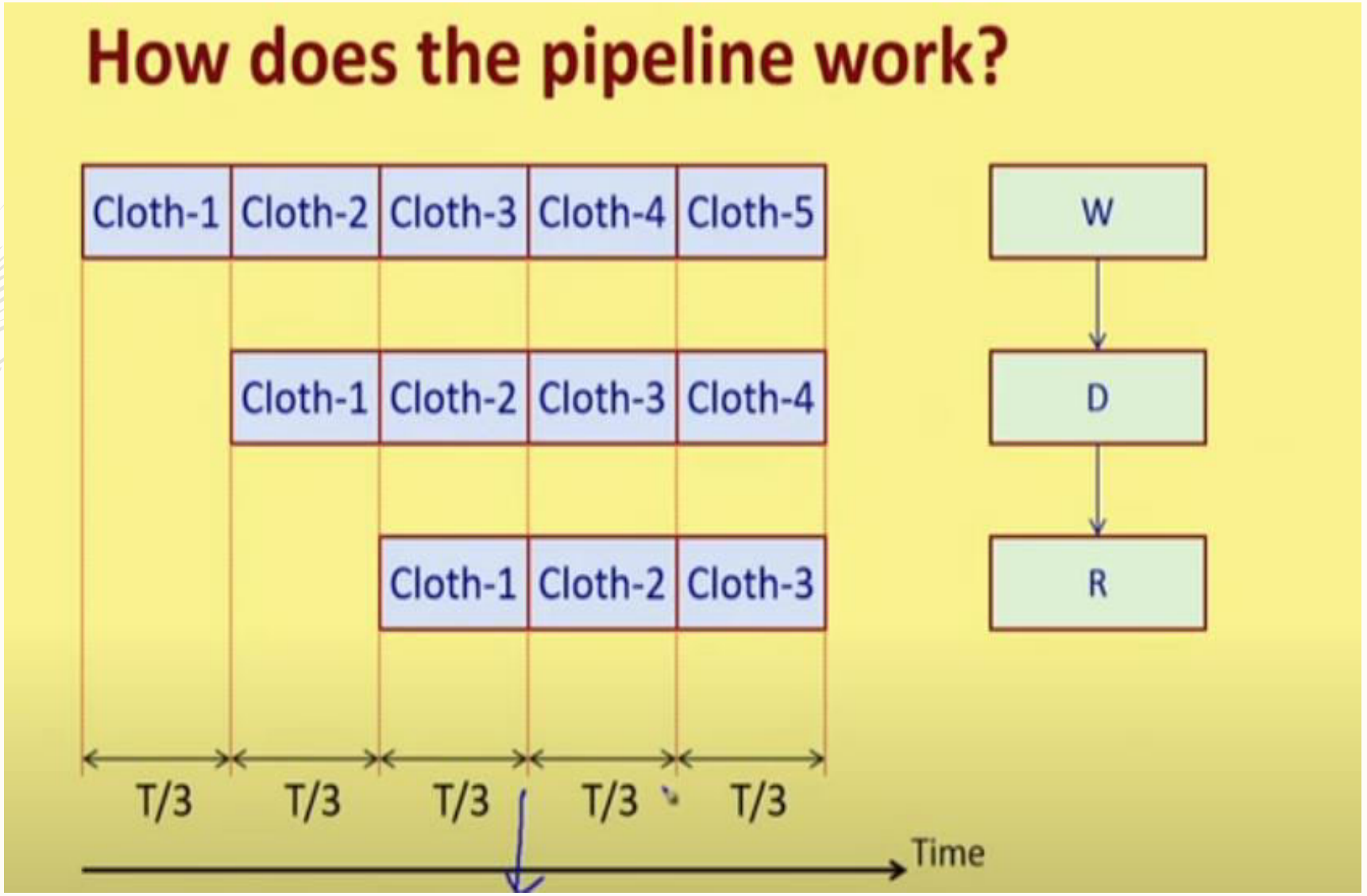
For N clothes, time $T_1 = N.T$



For N clothes, time $T_3 = (2 + N).T/3$

Finishing times:

- Cloth-1 – 3.T/3
- Cloth-2 – 4.T/3
- Cloth-3 – 5.T/3
- ...
- Cloth-N – (2 + N).T/3



Extending the Concept to Processor Pipeline

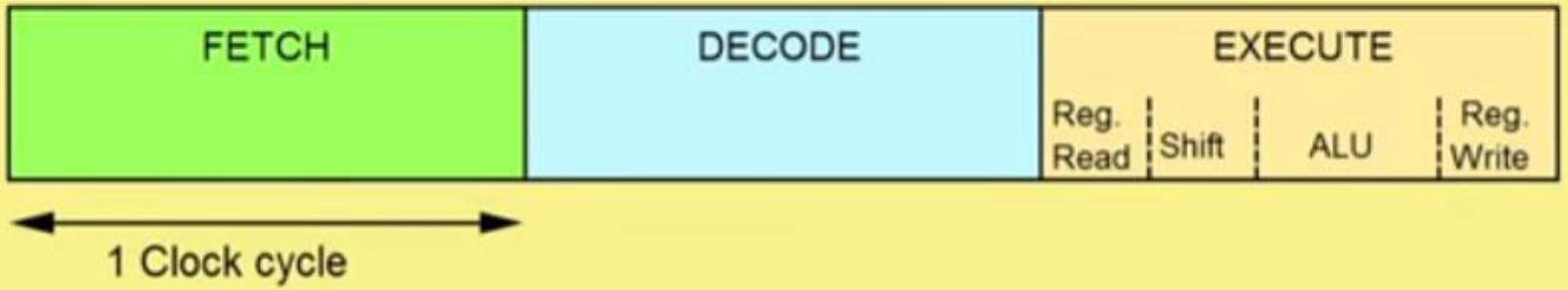
- The same concept can be extended to hardware pipelines.
- Suppose we want to attain *k* times speedup for some computation.
  - **Alternative 1**: Replicate the hardware *k* times → cost also goes up *k* times.
  - **Alternative 2**: Split the computation into *k* stages → very nominal cost increase.
- Need for buffering:
  - In the washing example, we need a tray between machines (W & D, and D & R) to keep the temporarily before it is accepted by the next machine.
  - Similarly in hardware pipeline, we need a *latch* between successive stages to hold the intermediate results temporarily.

Pipeline Organization

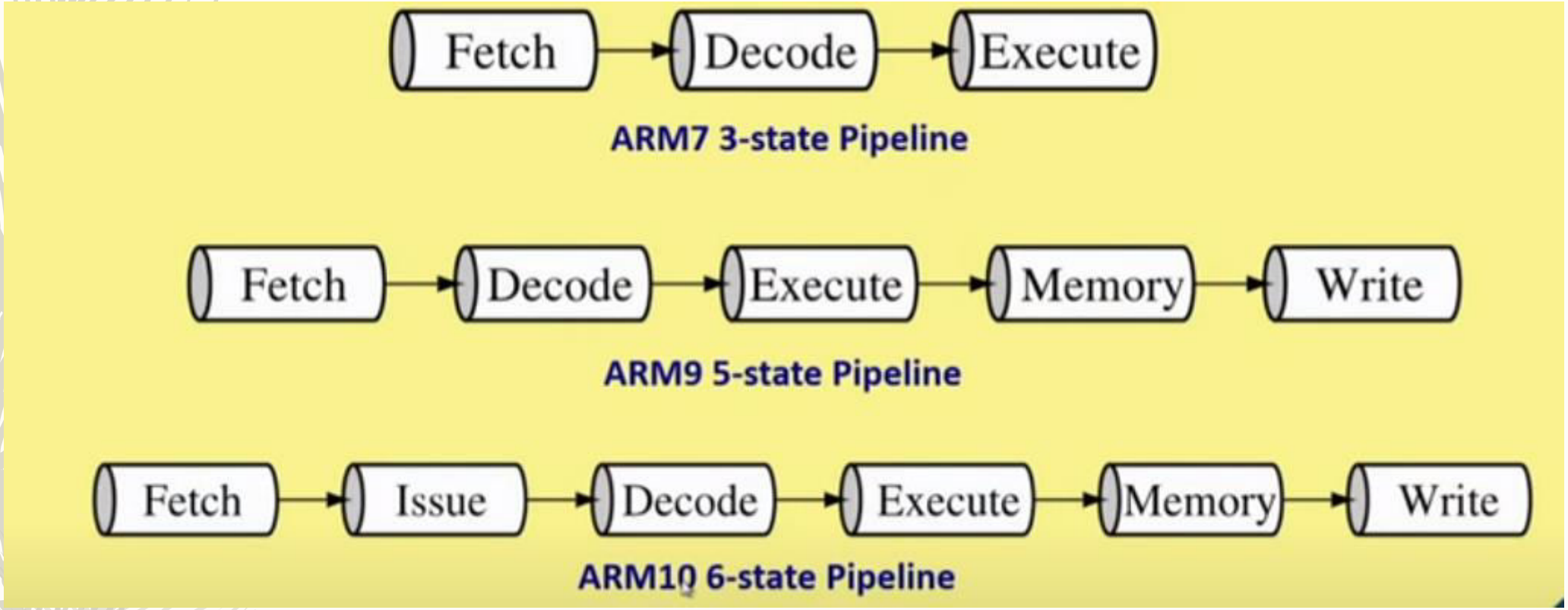
- Increases speed :
  - ✓ Most instructions executed in single cycle
- Versions:
  - ✓ 3-stage (ARM7TDMI and earlier)
  - ✓ 5-stage (ARMS, ARM9TDMI)
  - ✓ 6-stage (ARM10TDMI)
  - ✓ 8-stage(ARM 11)

ARM Pipelining Examples

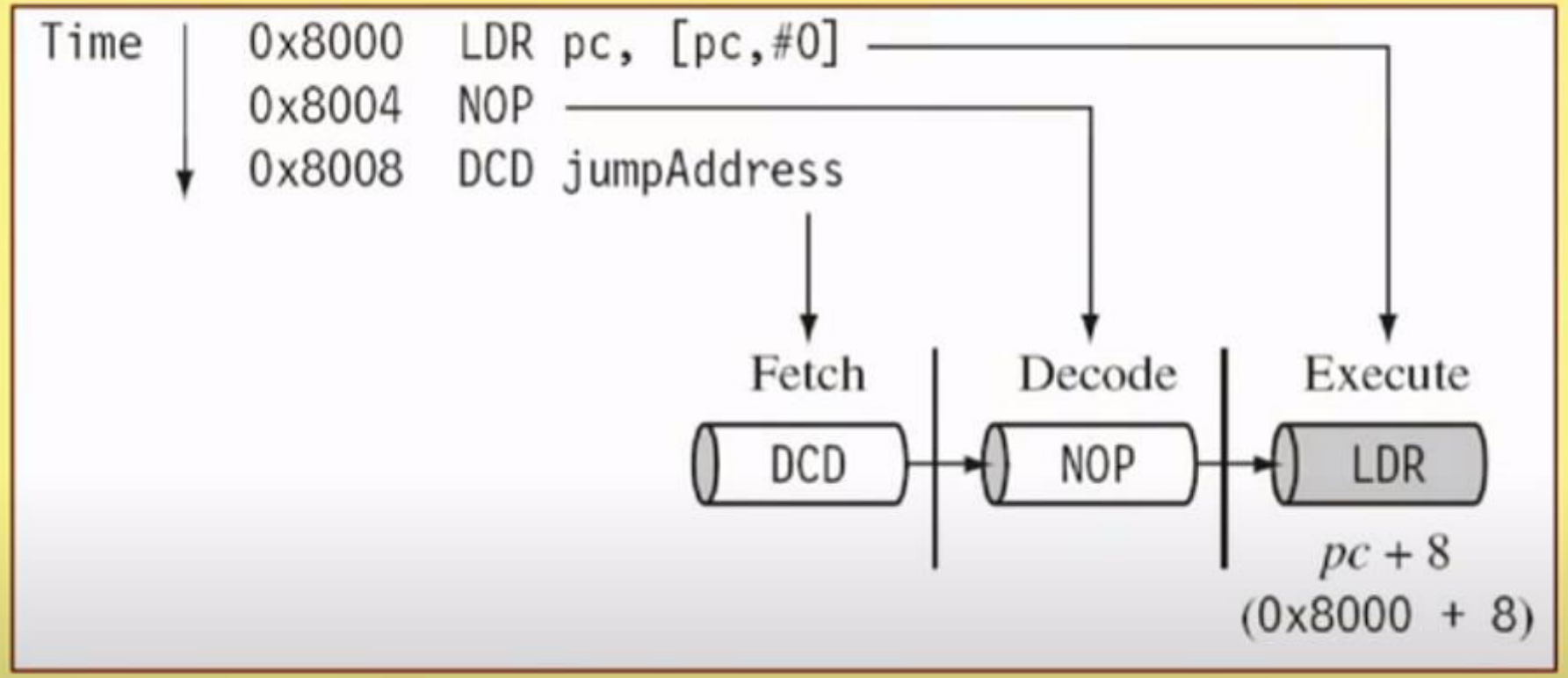
ARM7TDMI Pipeline



ARM9TDMI Pipeline


Fetch
Decode
Execute
ARM7 3-state Pipeline
Fetch
Decode
Execute
Memory
Write
ARM9 5-state Pipeline
Fetch
Issue
Decode
Execute
Memory
Write
ARM10 6-state Pipeline

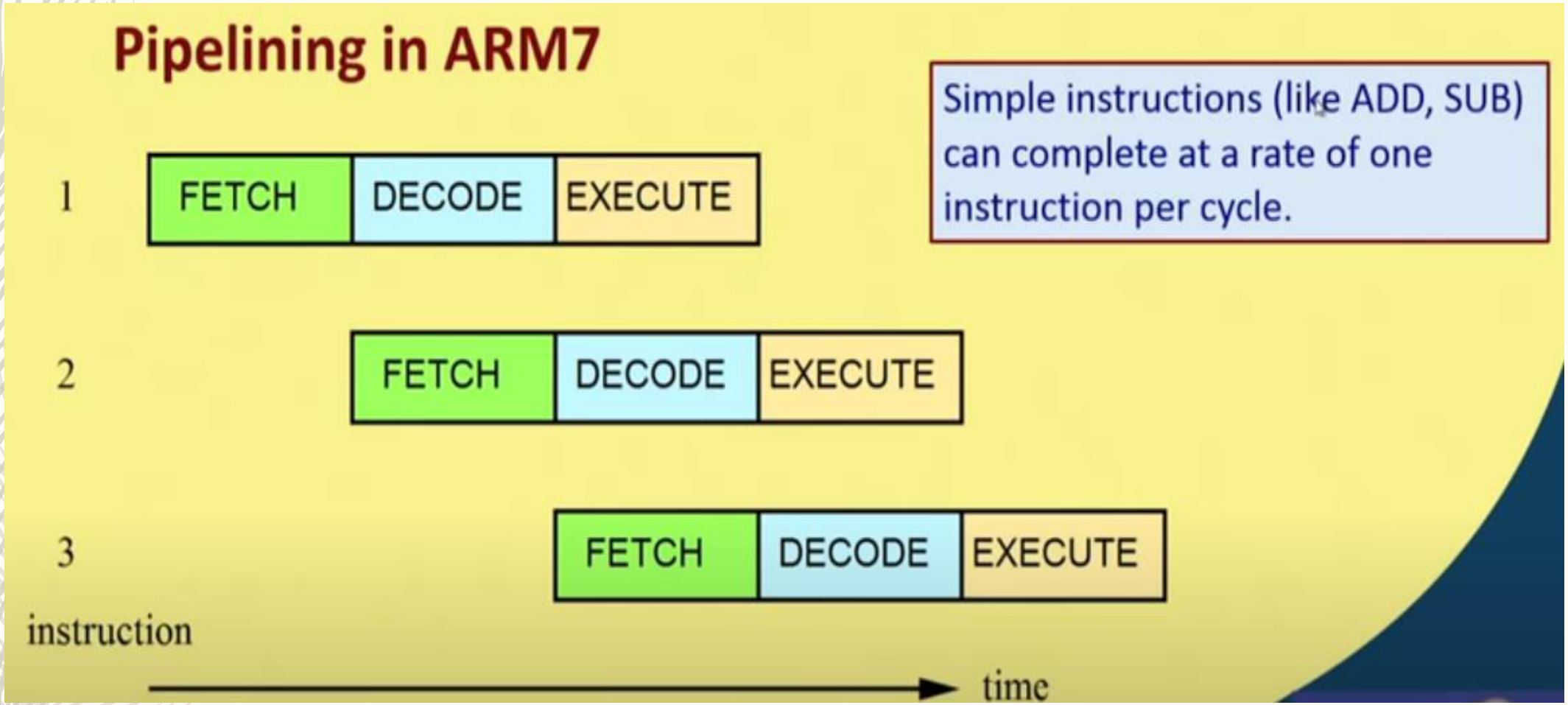
- In execution, the program counter (PC) is always 8 bytes ahead.



3-Stage Pipeline

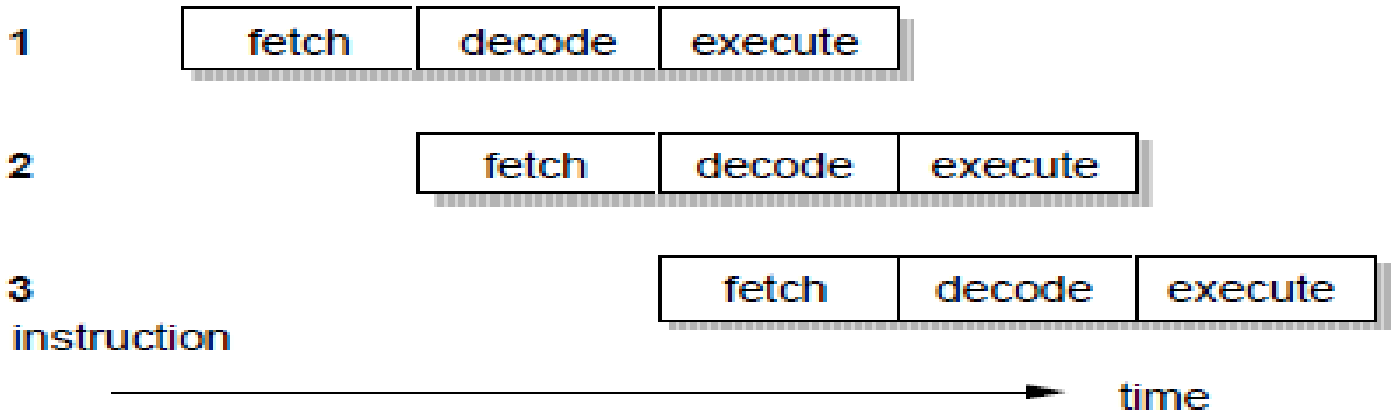
ARM 7

Pipelining in ARM7



Simple instructions (like ADD, SUB) can complete at a rate of one instruction per cycle.

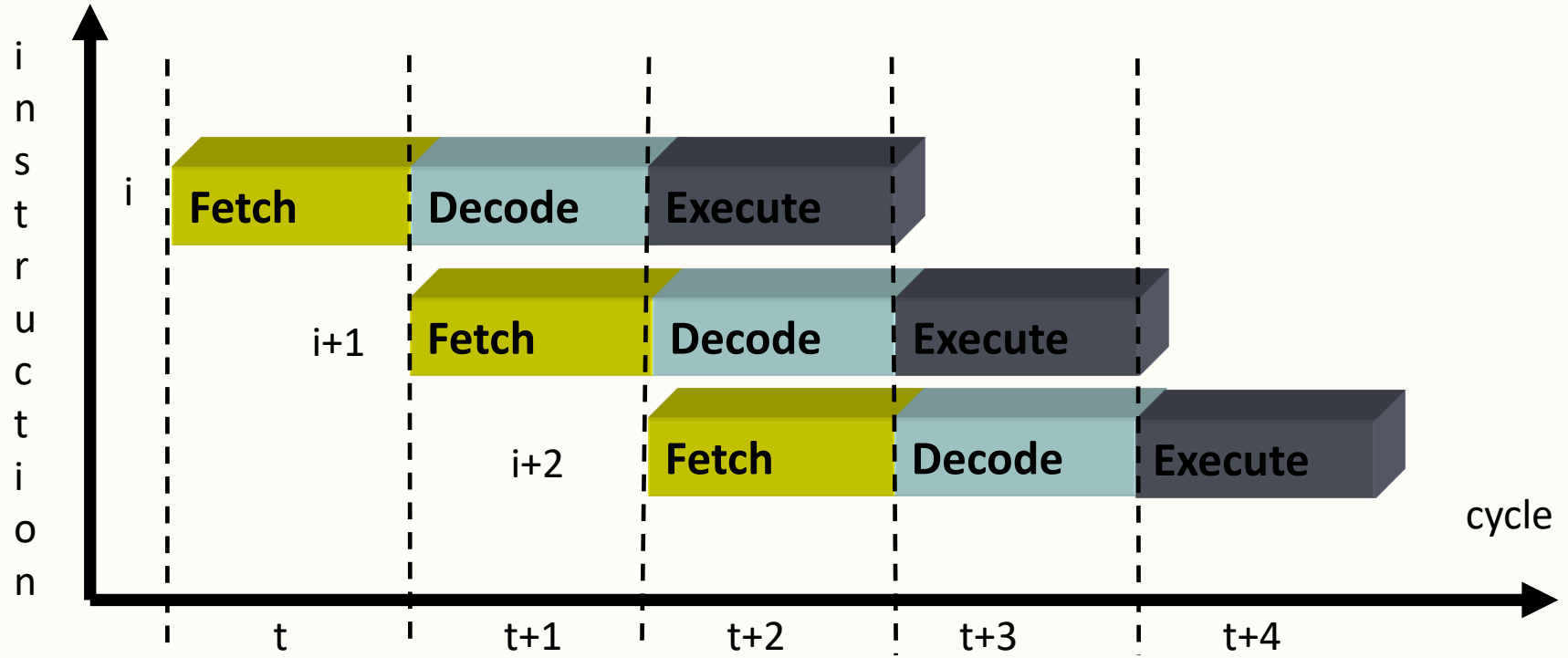
ARM single-cycle instruction pipeline



- *Latency: 3 cycles*
- *Throughput: 1 instruction/cycle*

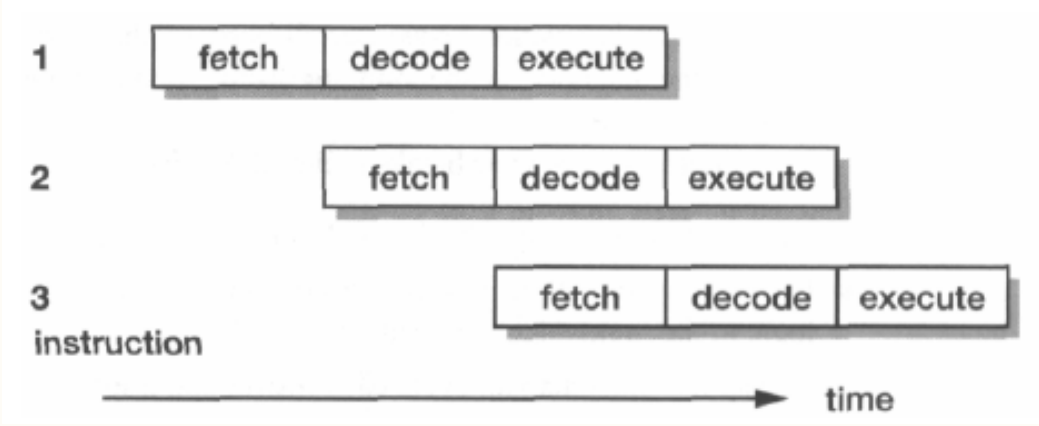
Pipeline Organization

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



STAGES:

- Fetch
  - ✓ *Instruction taken from memory*
- Decode
  - ✓ *Decode the fetched instruction and generate control signals for next cycle*
- Execute
  - ✓ *Read register bank (shift if necessary)*
  - ✓ *ALU result generated*
  - ✓ *Write back to register*



Three-stage pipeline

- 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; in this stage the instruction owns the decode logic but not the datapath
- Execute
  - the instruction owns the datapath; the register bank is read, an operand shifted, the ALU register generated and written back into a destination register

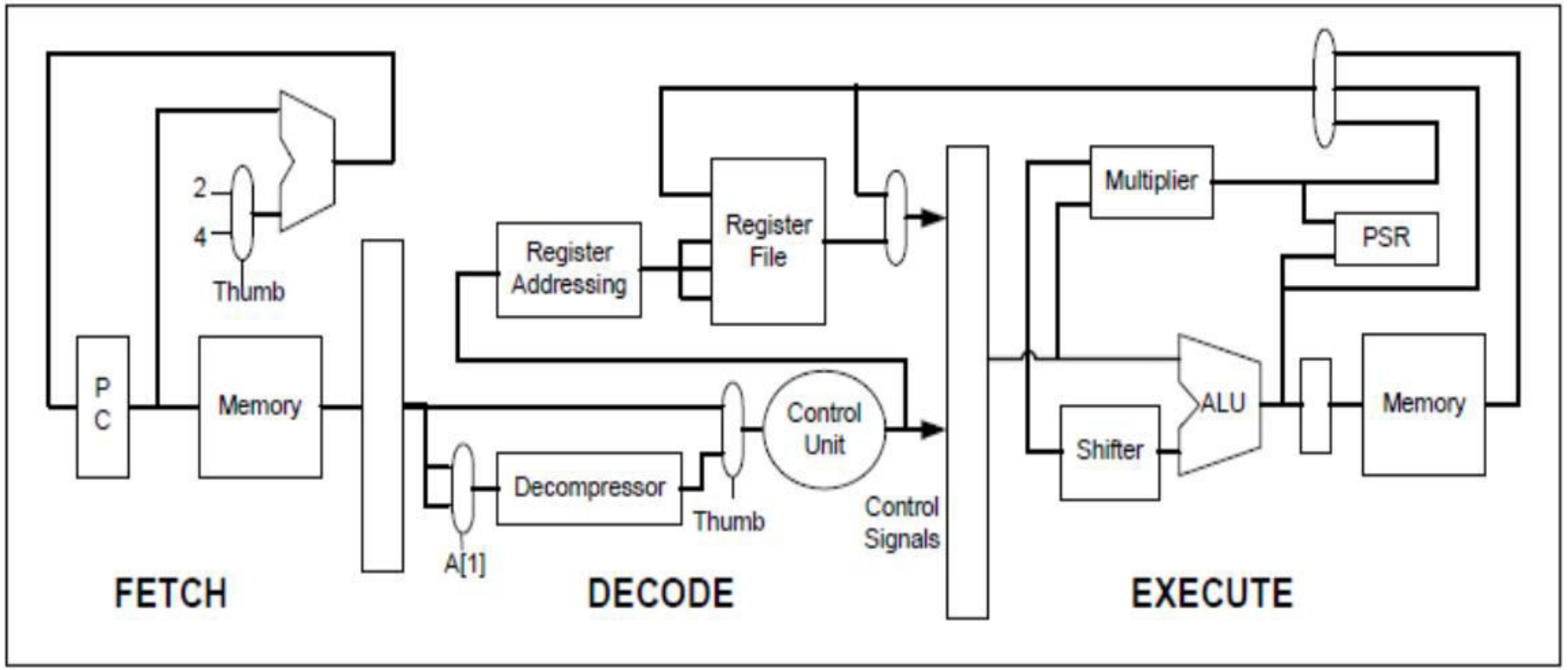
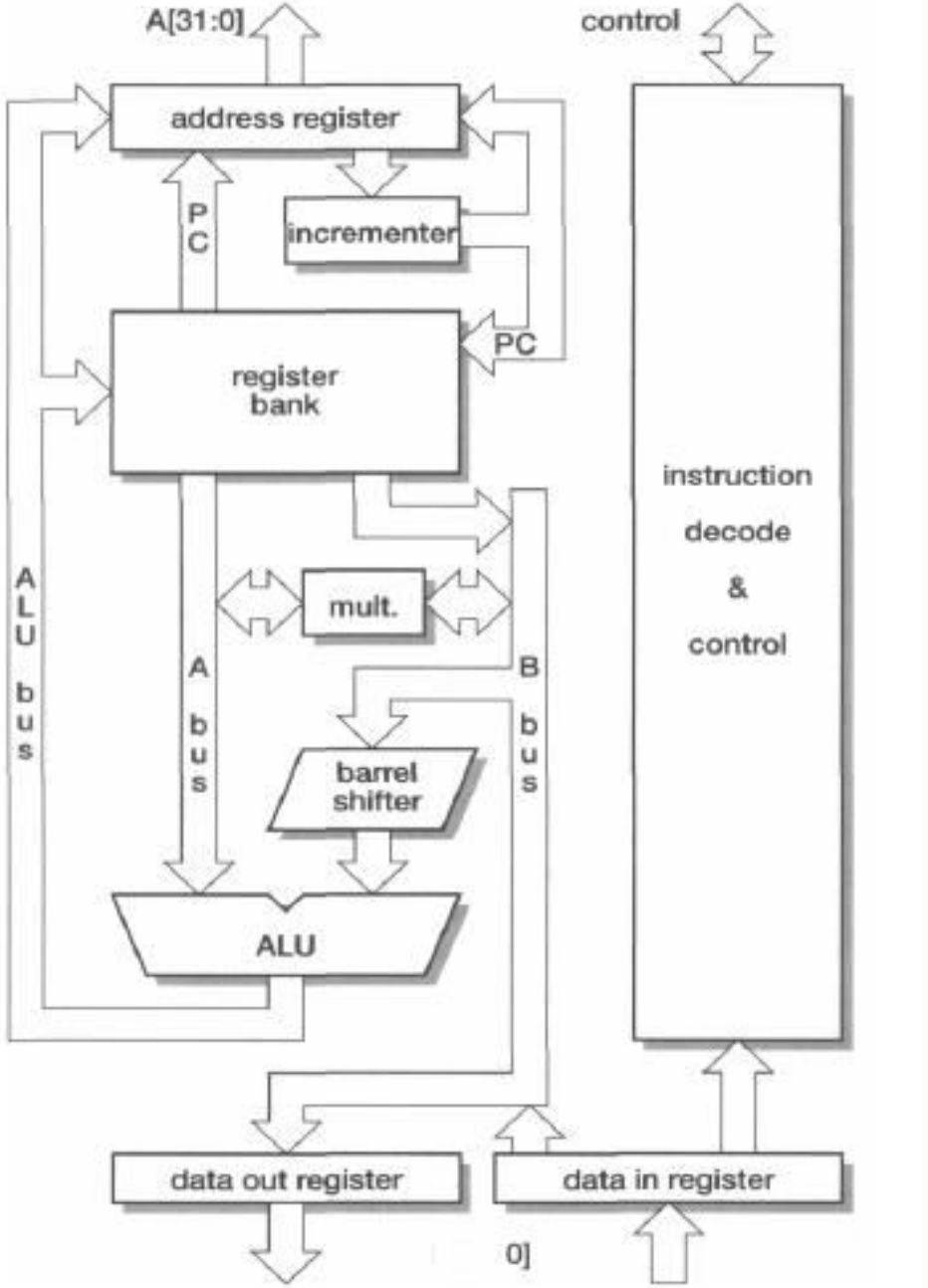


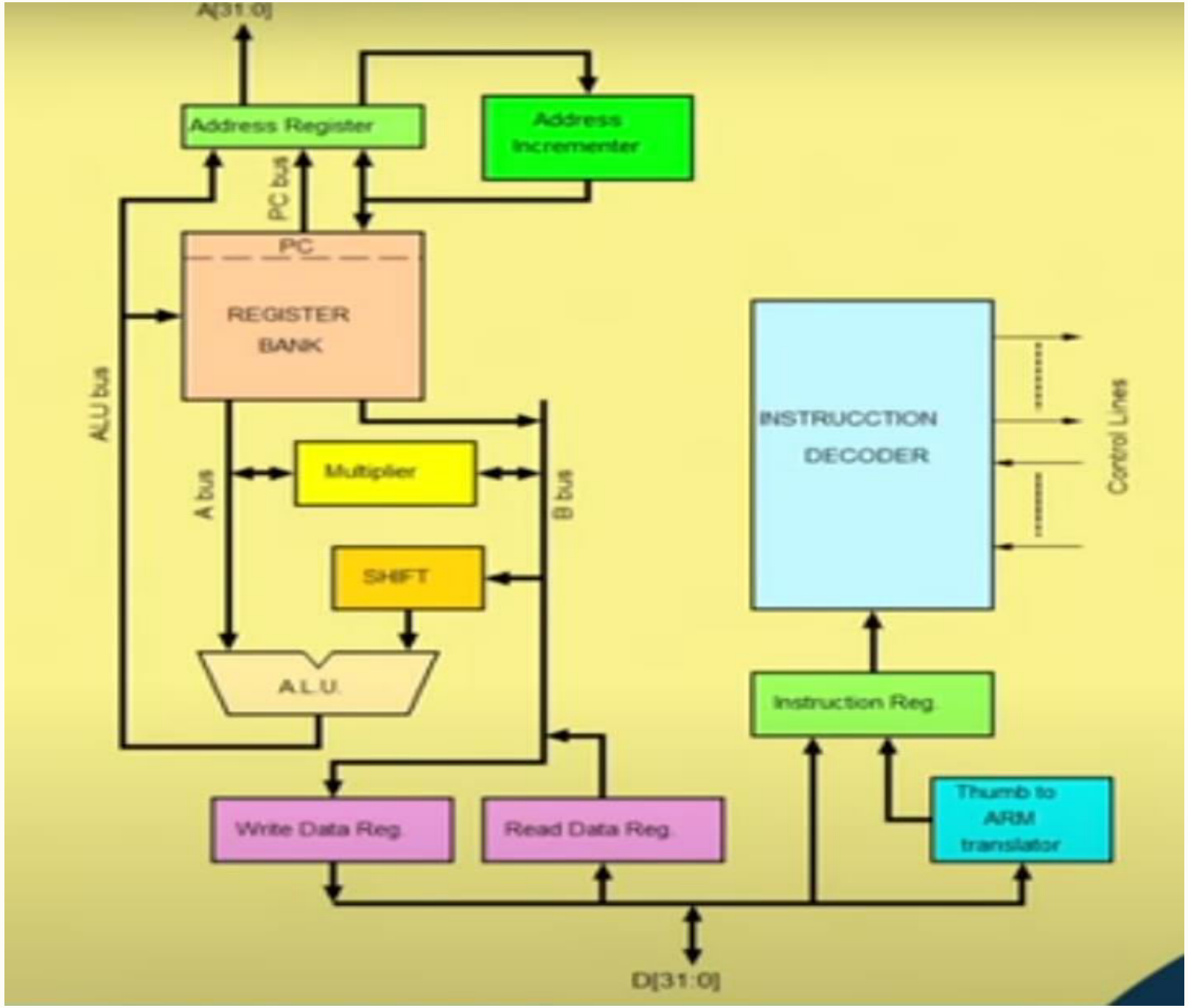
Fig: ARM 7 Core 3-Stage Pipe Lining

3-stage Pipeline ARM

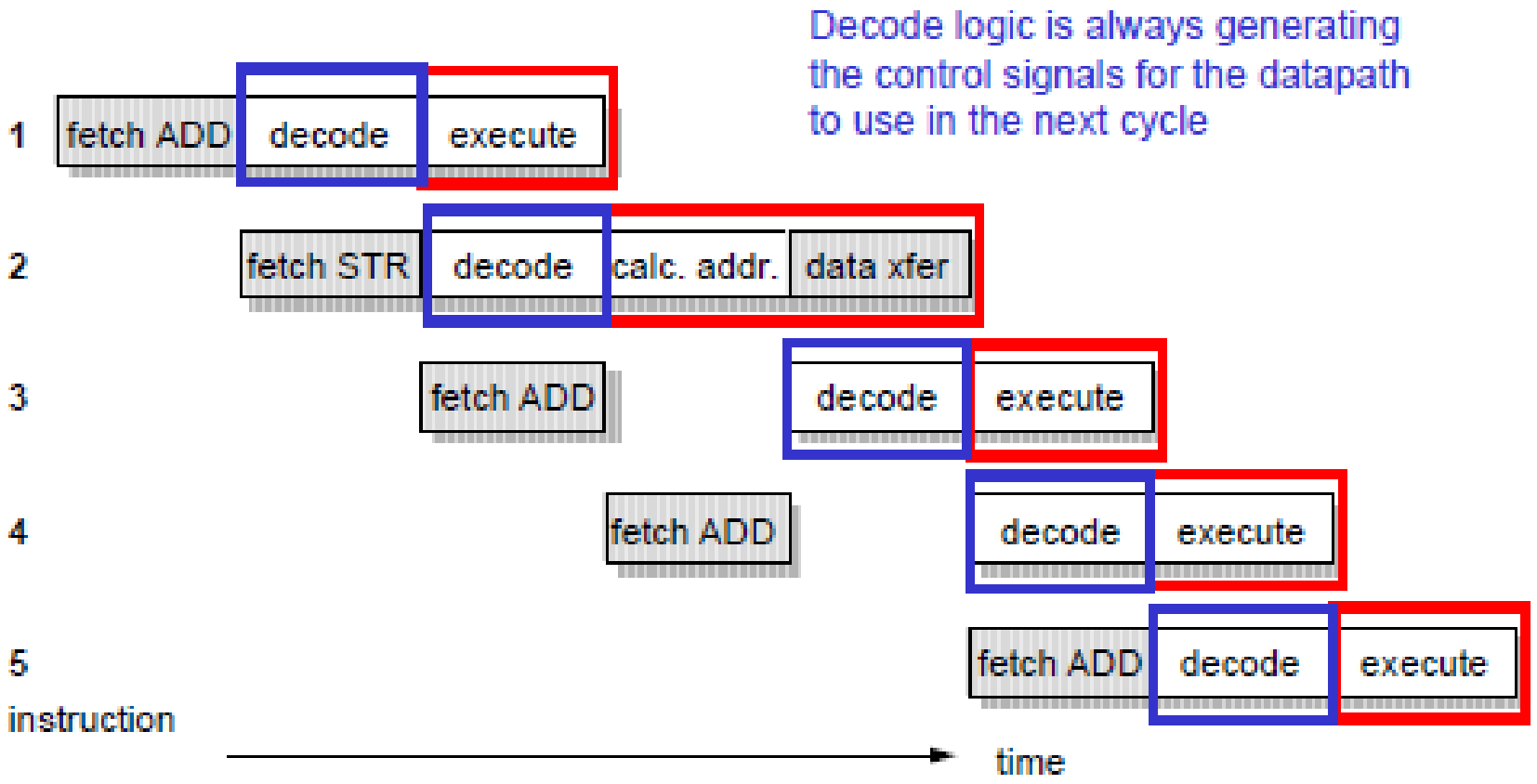
- Register bank
  - ✓ 2 data write
  - ✓ 1 data read
  - ✓ 1 PC write
  - ✓ 1 PC read
- Shifter, ALU, Address register, incrementer, data registers, Decode & control logic



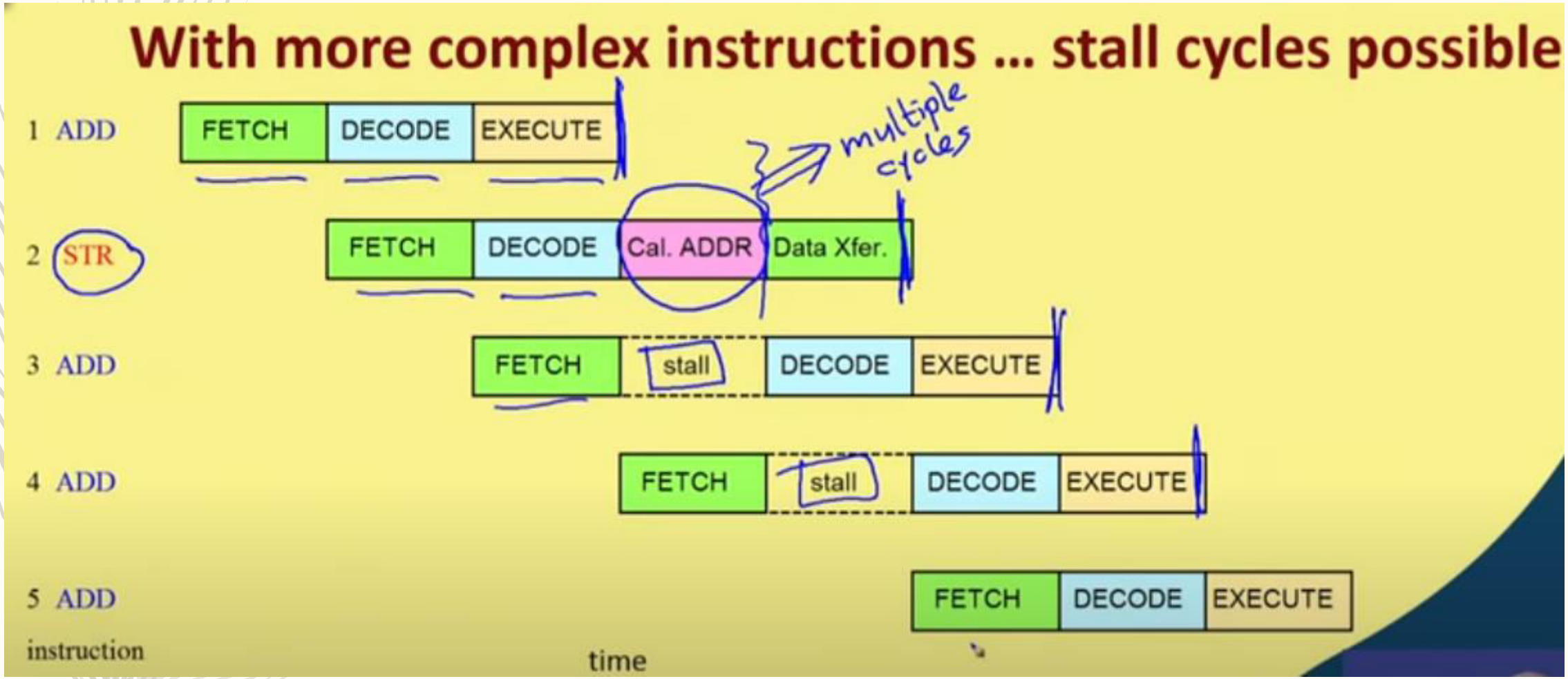
ARM7 Typical Architecture



ARM multi-cycle instruction pipeline



With more complex instructions … stall cycles possible



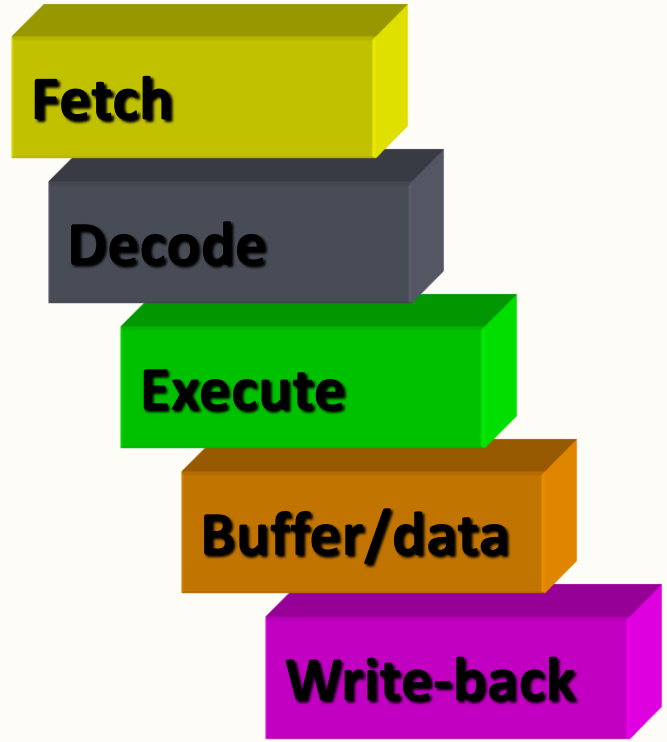
Five stage Pipelining

ARM9TDMI

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*

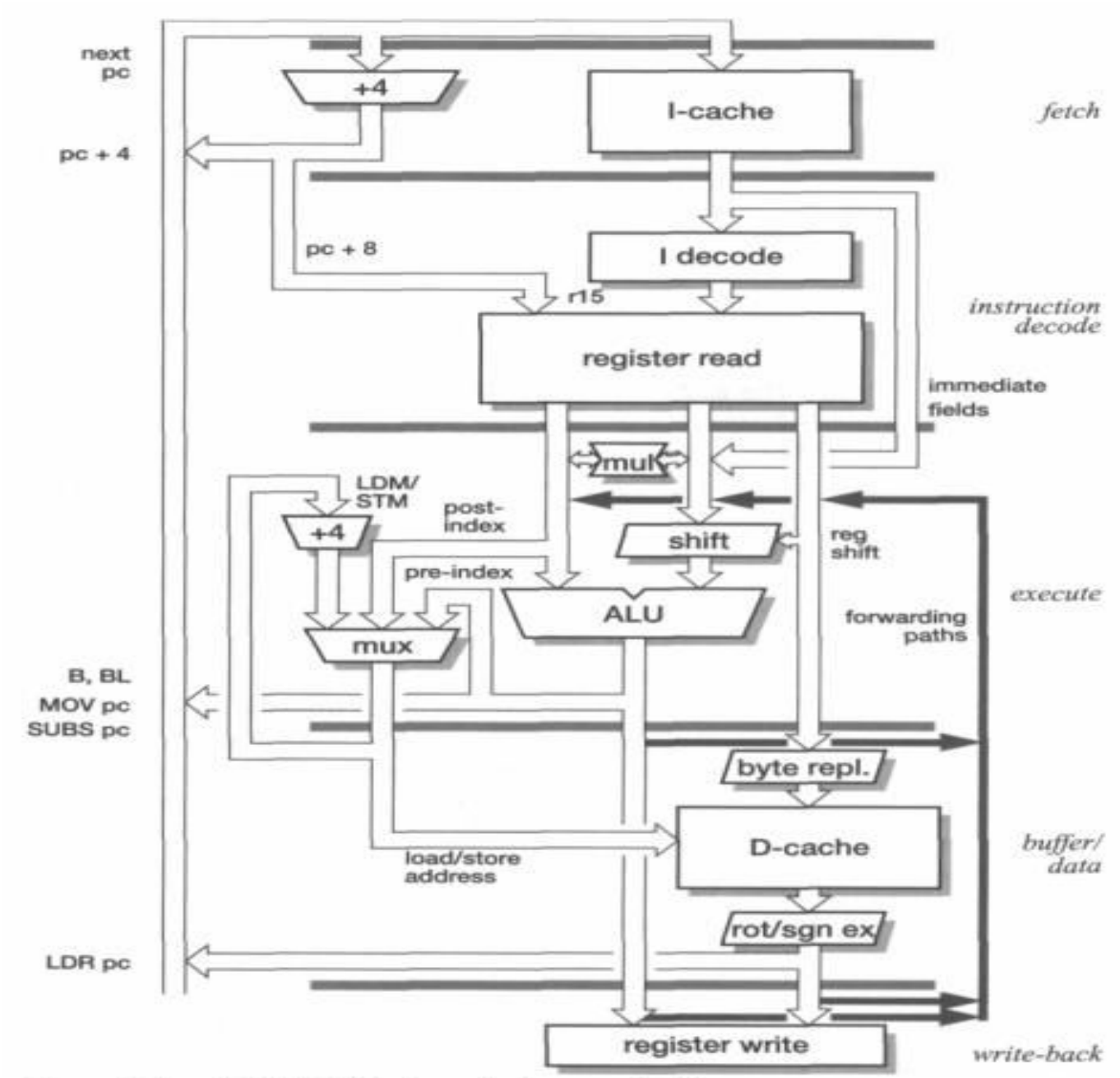


Figure 4.4 ARM9TDMI 5-stage pipeline organization.

Thank You!

References:

1) Text books: Steve Furber, ARM System On Chip Architecture, Addison Wesley, 2000
2) https://youtu.be/3OmyM4zuQw?list=PLbRMhDVUMngcJu5oUhgpgYqtOn7DmSfuU
3) https://youtu.be/CuuIBvHrvtA?list=PLbRMhDVUMngcJu5oUhgpgYqtOn7DmSfuU
4) https://youtu.be/2RDtt8bysZE?list=PLbRMhDVUMngcJu5oUhgpgYqtOn7DmSfuU
5) Lyla B.Das, Embedded Systems an Integrated Approach, Pearson
6) https://www.makeuseof.com/tag/what-is-an-arm-processor/
7) https://en.wikipedia.org/wiki/ARM_architecture
8) https://www.watelectronics.com/arm-processor-architecture-working/