Design Considerations of Embedded Systems

Architecture of ARM Microcontroller

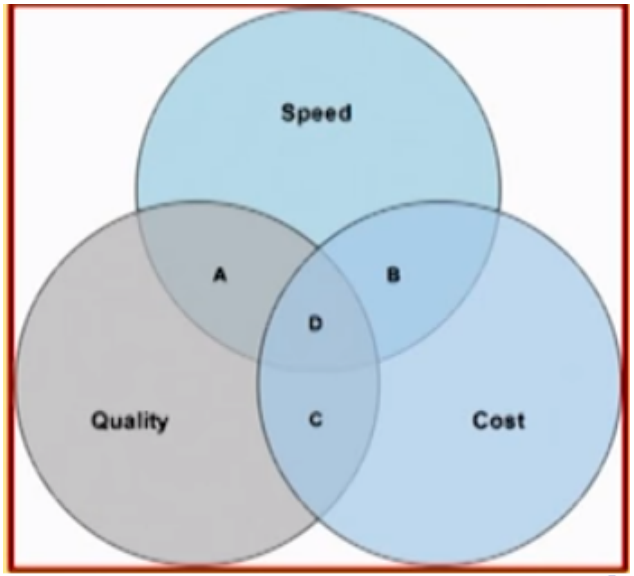
Dr. Munesh Singh

Indian Institute of Information Technology Design and Manufacturing
Kancheepuram, Chennai, Tamil Nadu

January 12, 2020

Design Challenges

- Primary design goal:
  - An implementation that realizes the desired functionality
- The main design challenge is...
  - To simultaneously optimize several design metrics
  - Often mutually conflicting.
- What is a design metric?
  - Some features of an implementation that can be measured and evaluated.



Common Design Metrics

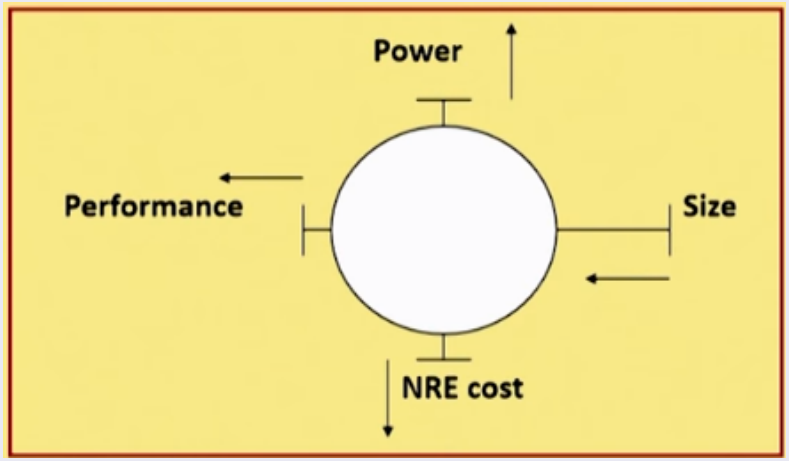
- Non Recurring Engineering (NRE) Cost: One-time initial cost of designing a system.
- Unit cost: The cost of manufacturing each copy of the system, without counting the NRE cost.
- Size: The actual physical space occupied by the system.
- Performance: This is measured in terms of the time taken or throughput.
- Power: The amount of (battery) power consumed by the system.
- Flexibility: The ability to change the functionality of the system

Common Design Metrics

- Maintainability: How easy or difficult it is to modify the design of the system?
- Time-to-prototype: How much time is required to build a working version of the system (i.e a prototype)?
- Time-to-market: How much time is required to develop a system such that it can be released to the market commercially?
- Safety: Are there any adverse effect on the operating environment?

Design Tradeoff

- Many of the design metrics can be mutually conflicting.
  - Improving one may degrade the other (e.g Power, performace, size, and NRE cost, etc.)



Design Tradeoff

- Often required expertise in both hardware and software to take a proper decision.
  - Expertise in hardware may indicate the types of co-processor or I/O interfaces to use for specific applications (e.g analog port, digital ports, PWM ports, etc.)
  - Expertise in software is required to identify parts of the implementation that need to be implemented in software and run on the micro-controller
  - Hardware / Software co-design become important.

Time-to-market Design Metric

- This is a very crucial design metric.
  - Must be strictly followed to make a product commercially viable.
  - Requires exhaustive market study and analysis.
- Starting from the point a product design starts, we can define a Market Window within which it is expected to have high sales.
  - Any delay can result in drastic reductions in sales.

Performance Design Metric

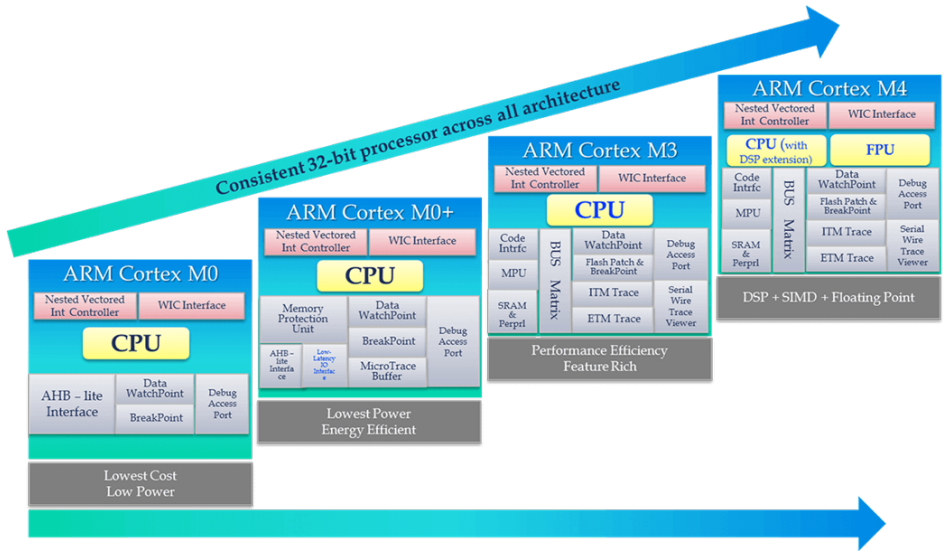
- Most widely used, but can also be most misleading.
  - Must be careful in the evaluation.
- Some of the measures:
  - Clock frequency, MIPS–not very good for comparison
  - Latency (response time)
  - Throughput
- Measure of speedup among design alternatives.

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 1930.
  - 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 processor 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 principle 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.

Popular ARM Architectures

ARM7

- 3 pipeline stages(fetch/decode/execute)
- High code density/low power consumption
- Most widely used for low-end systems

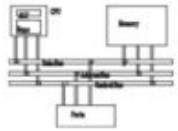
ARM9

- 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 family attribute comparison.

year	1995	1997	1999	2003
	ARM7	ARM9	ARM10	ARM11
Pipeline depth	three-stage	five-stage	six-stage	eight-stage
Typical MHz	80	150	260	335
mW/MHz[a]	0.06 mW/MHz	0.19 mW/MHz (+ cache)	0.5 mW/MHz (+ cache)	0.4 mW/MHz (+ cache)
MIPS[b]/MHz	0.97	1.1	1.3	1.2
Architecture	Von Neumann	Harvard	Harvard	Harvard
Multiplier	8 × 32	8 × 32	16 × 32	16 × 32

[a] Watts/MHz on the same 0.13 micron process.

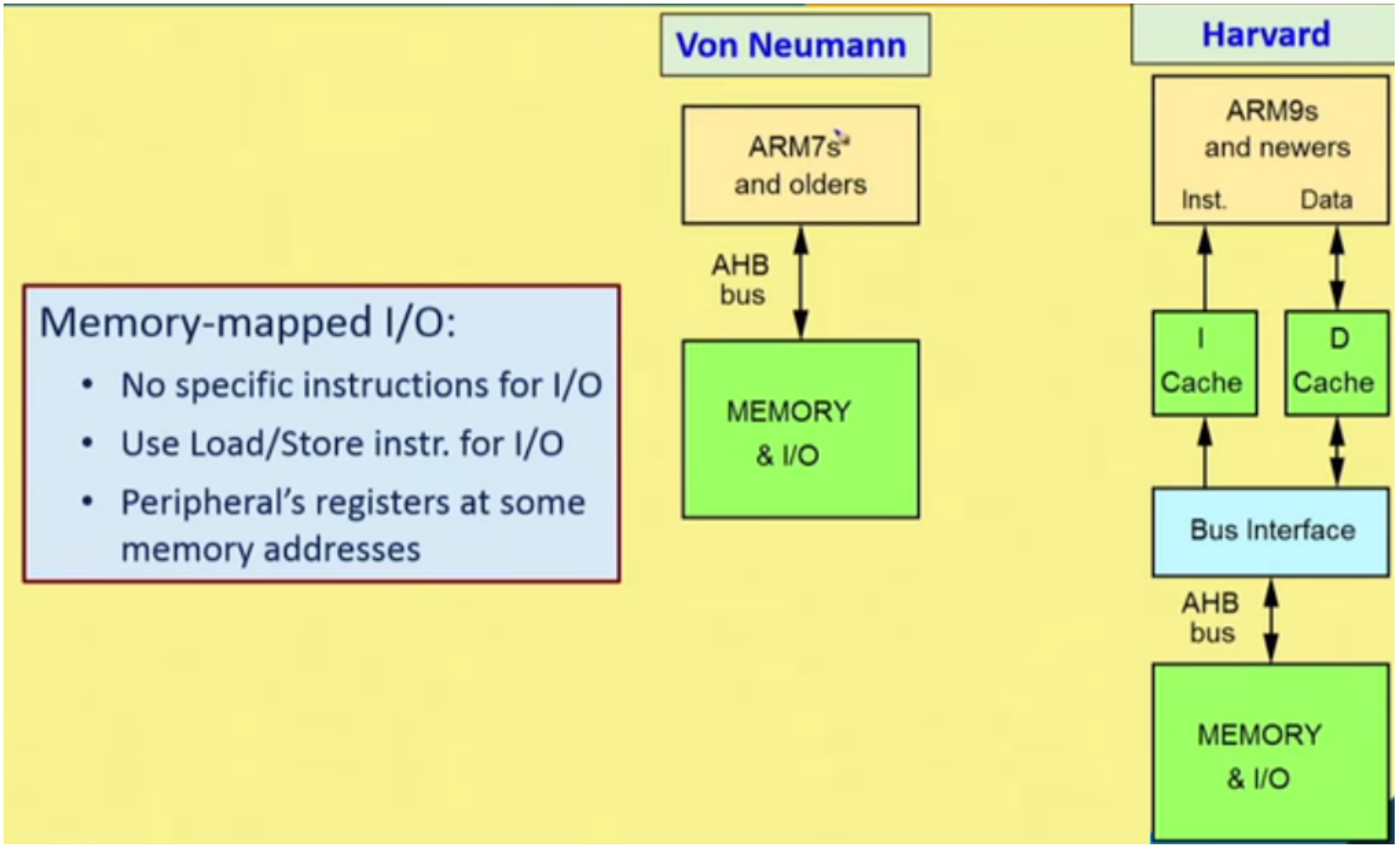
[b] MIPS are Dhrystone VAX MIPS.

ARM is based on RISC Architecture

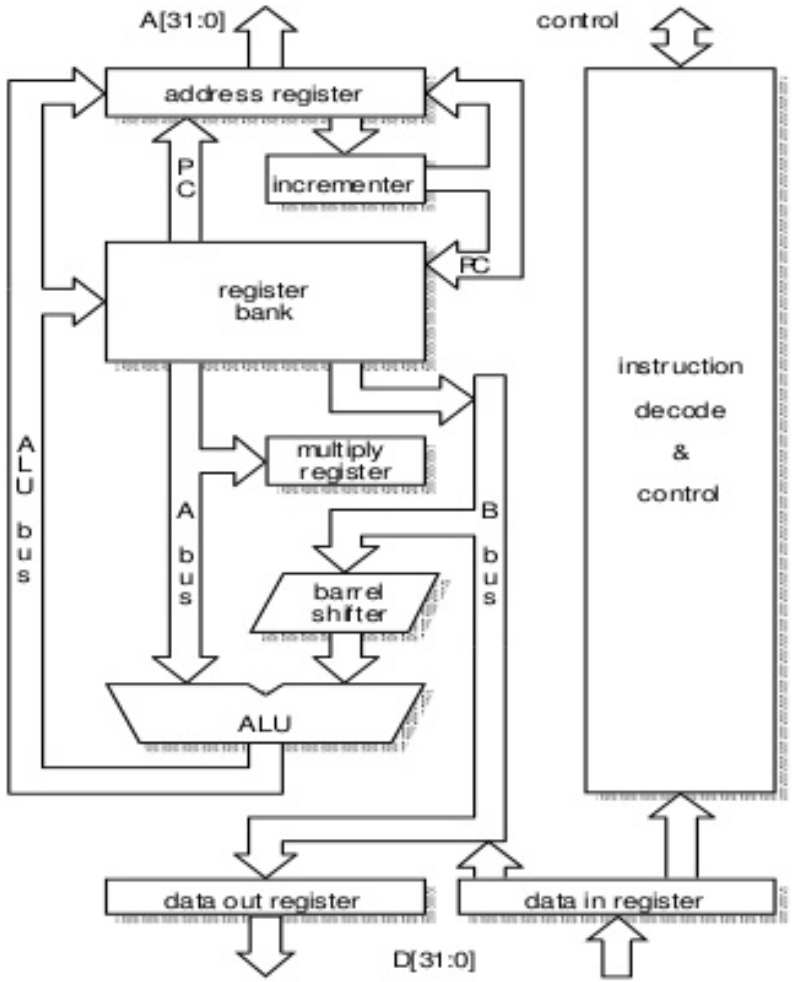
- RISC supports simple but powerful instructions that execute in a single cycle at high clock frequency.
- Major design features:
  - Instructions: reduced set/single cycle/fixed length
  - Pipeline: decode in one stage/no need for microcode
  - Registers: large number of general-purpose registers (GPRs)
  - Load/store Architecture: data processing instructions work on registers only; load/store instruction to trasfer 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 (result in improvement in code density by about 30%)
  - Conditional execution(reduces branch and improve 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
Von Neumann
ARM7s and olders
AHB bus
MEMORY & I/O
Harvard
ARM9s and newers
Inst.
Data
I Cache
D Cache
Bus Interface
AHB bus
MEMORY & I/O

ARM Typical Architecture

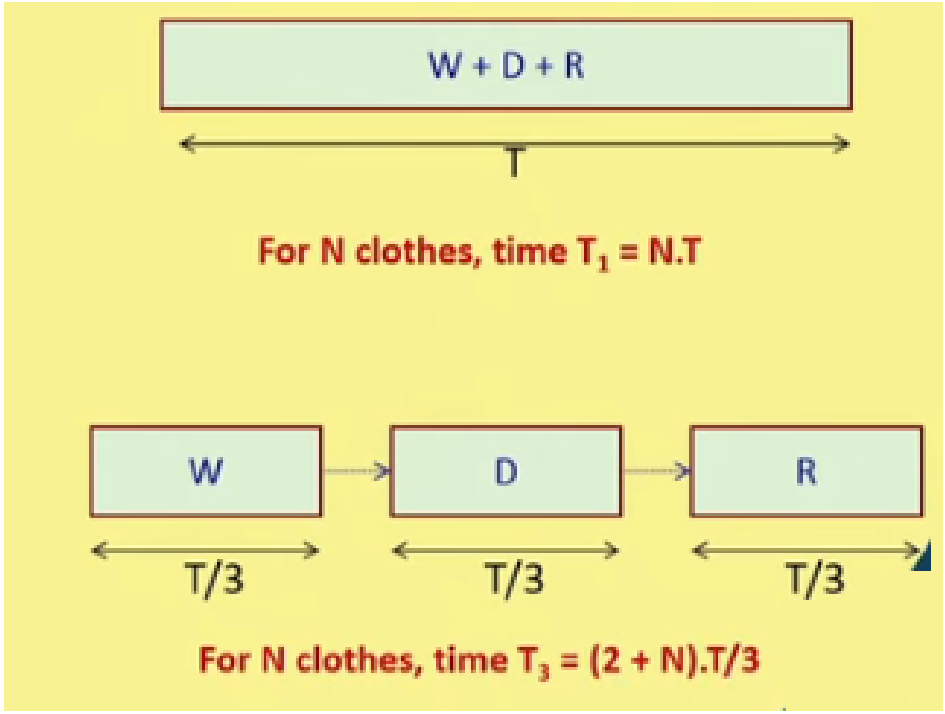


What is Pipelining?

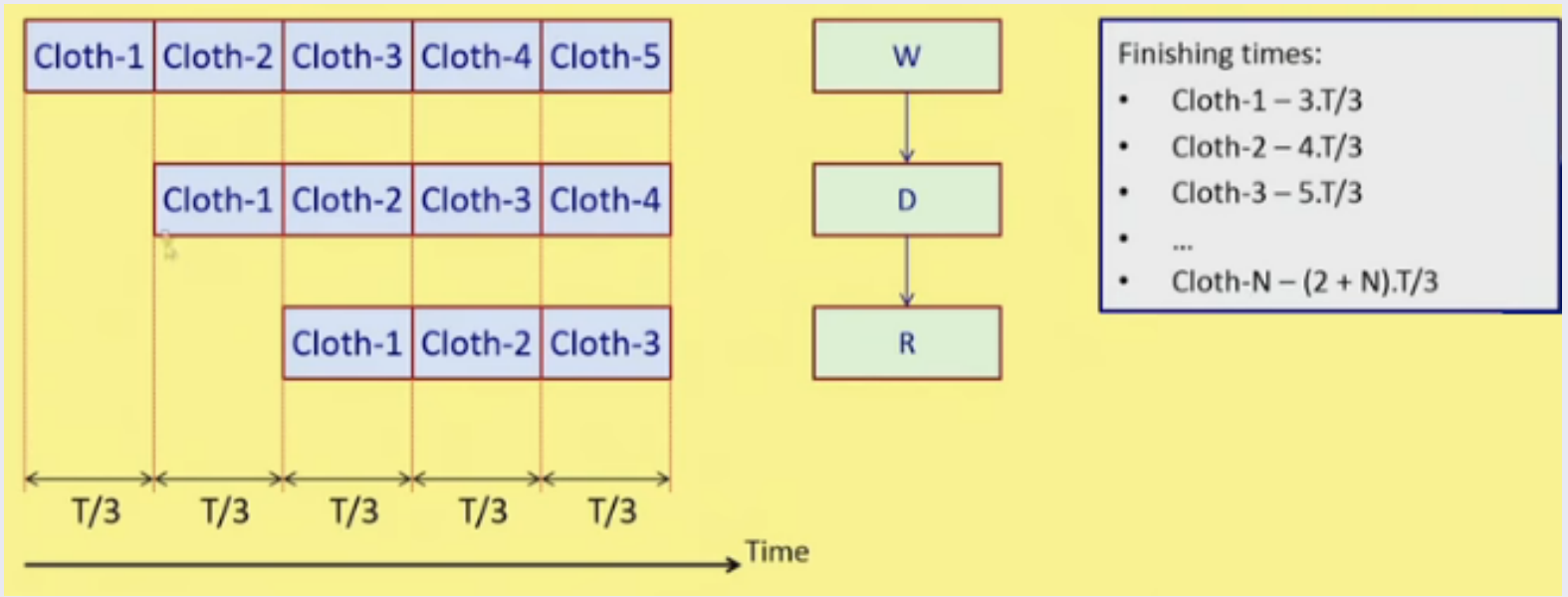
- A mechanism for overlapped execution of several input sets by partitioning some computation into set of k sub-computations (or stages).
  - very nominal increase in the cost of implementation.
  - very significant speedup(ideally, k).
- Where are pipelining used in a computer system?
  - Instruction execution: Several instruction executed in some sequenxce
  - Arithmetic computation:Some 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 $\frac{T}{3}$



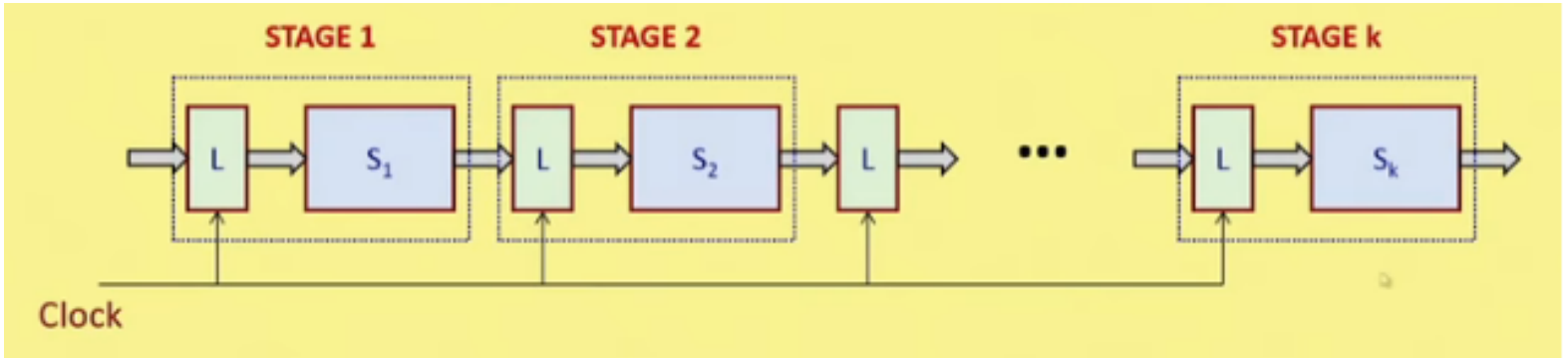
How does the pipeline works?



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 to k times.
  - Alternative 2: Split the computation to K stages $->$ Very nominal cost increase.
- Need for buffering:
  - In the washing examples, we need a tray between machines (W & D, and D & R) to keep cloth temporarily before it is accepting by the next machine.
  - Similarly in hardware pipeline, we need a latch between successive stages to hold the intermediate results temporally.

Model of a Synchronous k-stage pipeline



- The latches are made with master-slave flip-flops, and serve the purpose of isolating inputs from outputs.
- The pipeline stages are typically combinational circuits.
- When clock is applied, all latches transfer data to the next stage simultaneously

Speedup and Efficiency

Some notations:

τ :: *clock period of the pipeline*

t_i :: *time delay of the circuitry in stage* S_i

d_L :: *delay of a latch*

Maximum stage delay

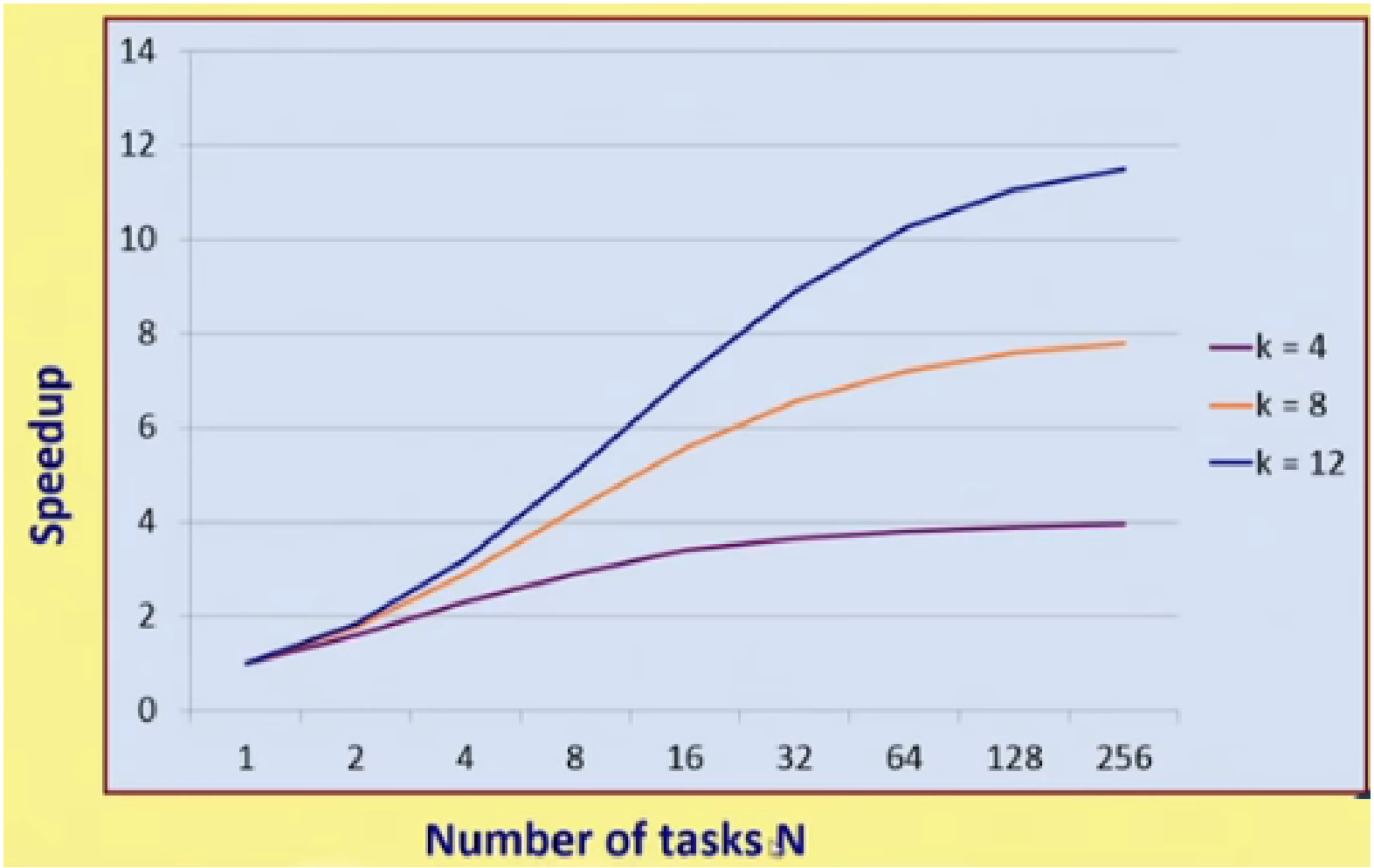
$\tau_m = max\{t_i\}$

Thus, $\tau = \tau_m + d_L$

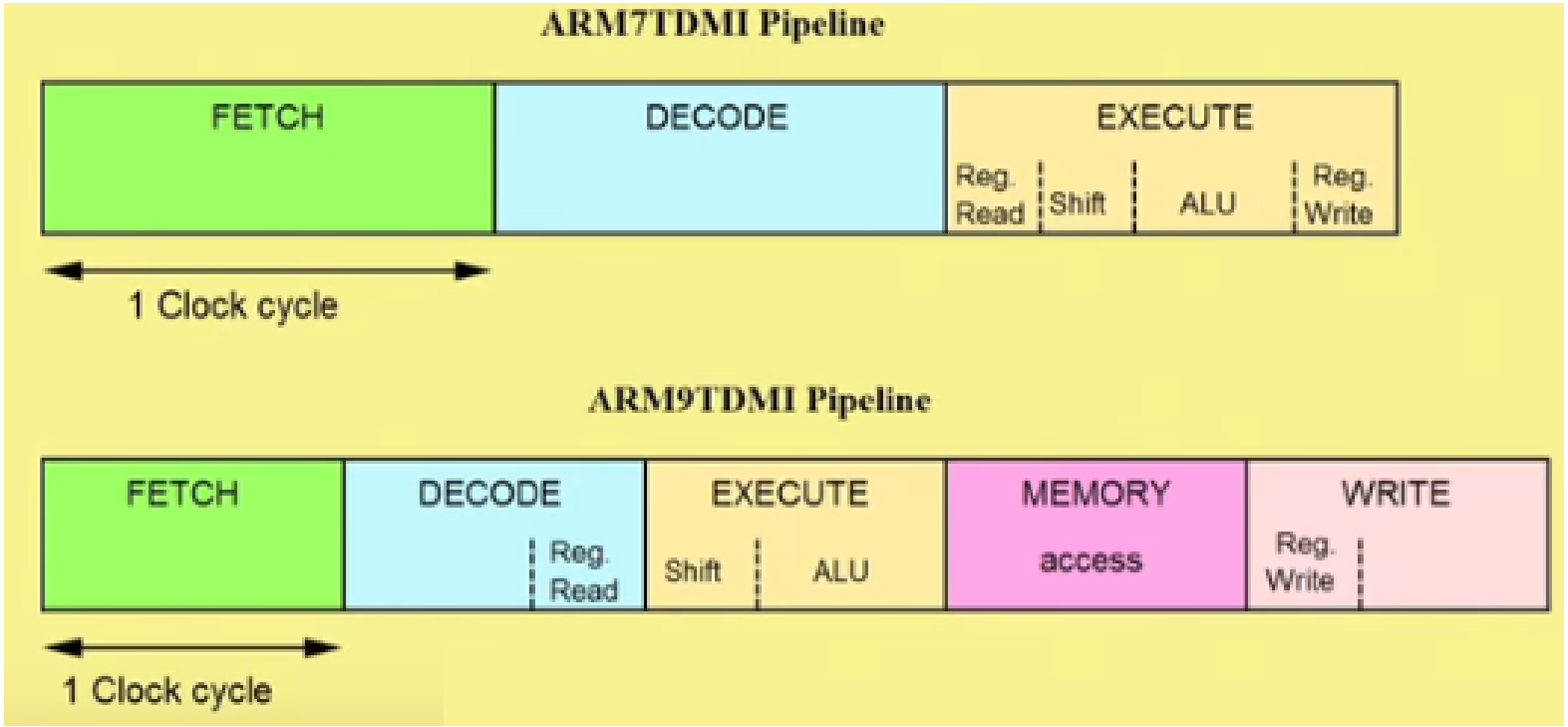
Pipeline frequency $f = 1/\tau$

- *if one result is expected to come out of the pipeline every clock cycle f will represents the maximum through of the pipeline*

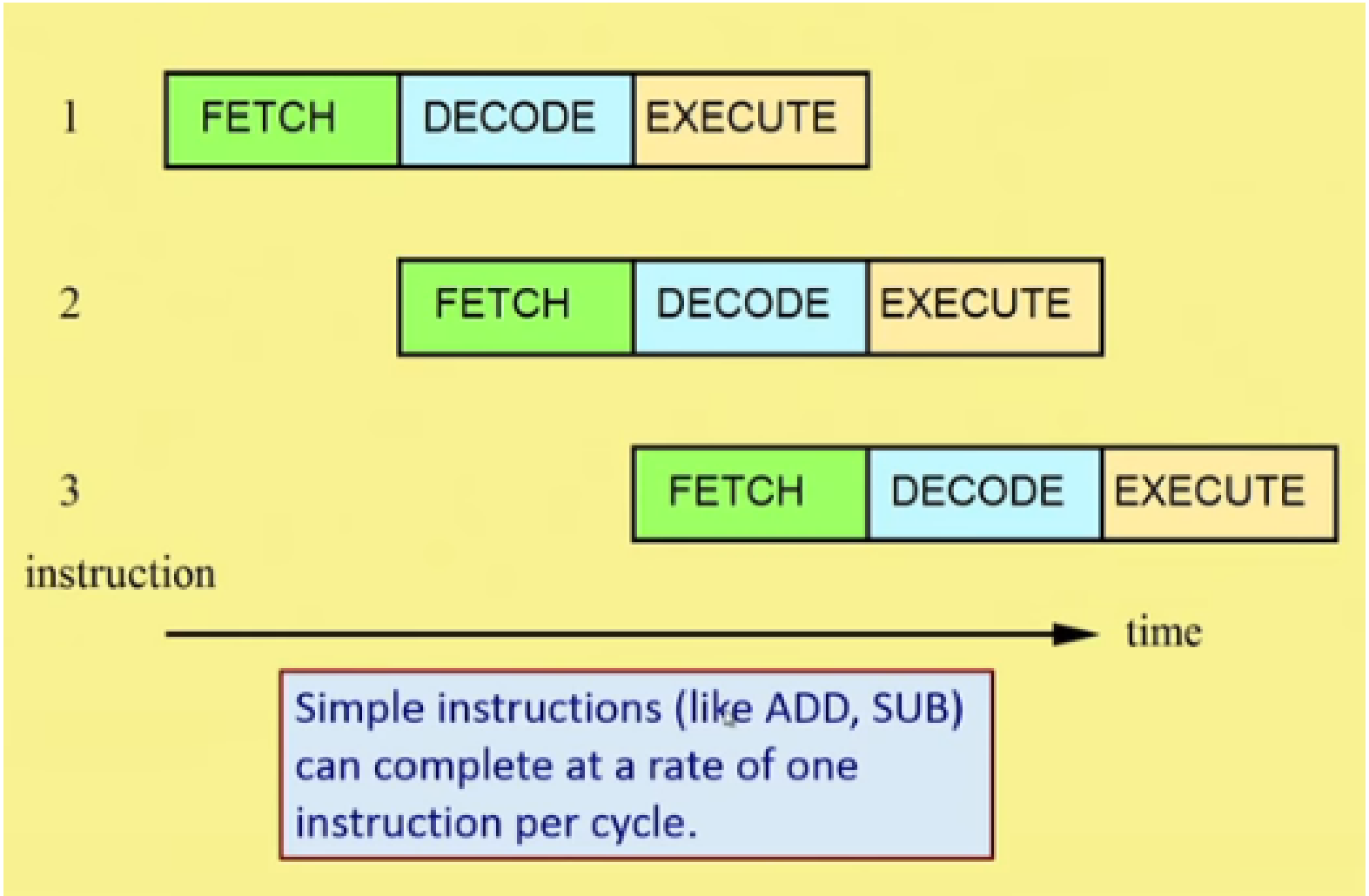
- The total time to process N data sets is given by
  $T_k = [(k-1) + N].\tau$
  $(k-1)\tau$ time required to fill the pipeline
  1 result every τ time after that $->$ total $N.\tau$
- For an equivalent non-pipelined processor (i.e. one stage), the total time is
  $T_1 = N.k.\tau$ ignoring the latch overheads
- Speedup of the k-stage pipeline over equivalent non-pipelined processor:
  $S_k = \frac{T_1}{T_k} = \frac{N.k.\tau}{k.\tau+(N-1).\tau} = \frac{N.k}{k+(N-1)}$
  $As\ N-> \infty, S_k = k$


Speedup
14
12
10
8
6
4
2
0
1
2
4
8
16
32
64
128
256
k = 4
k = 8
k = 12
Number of tasks N

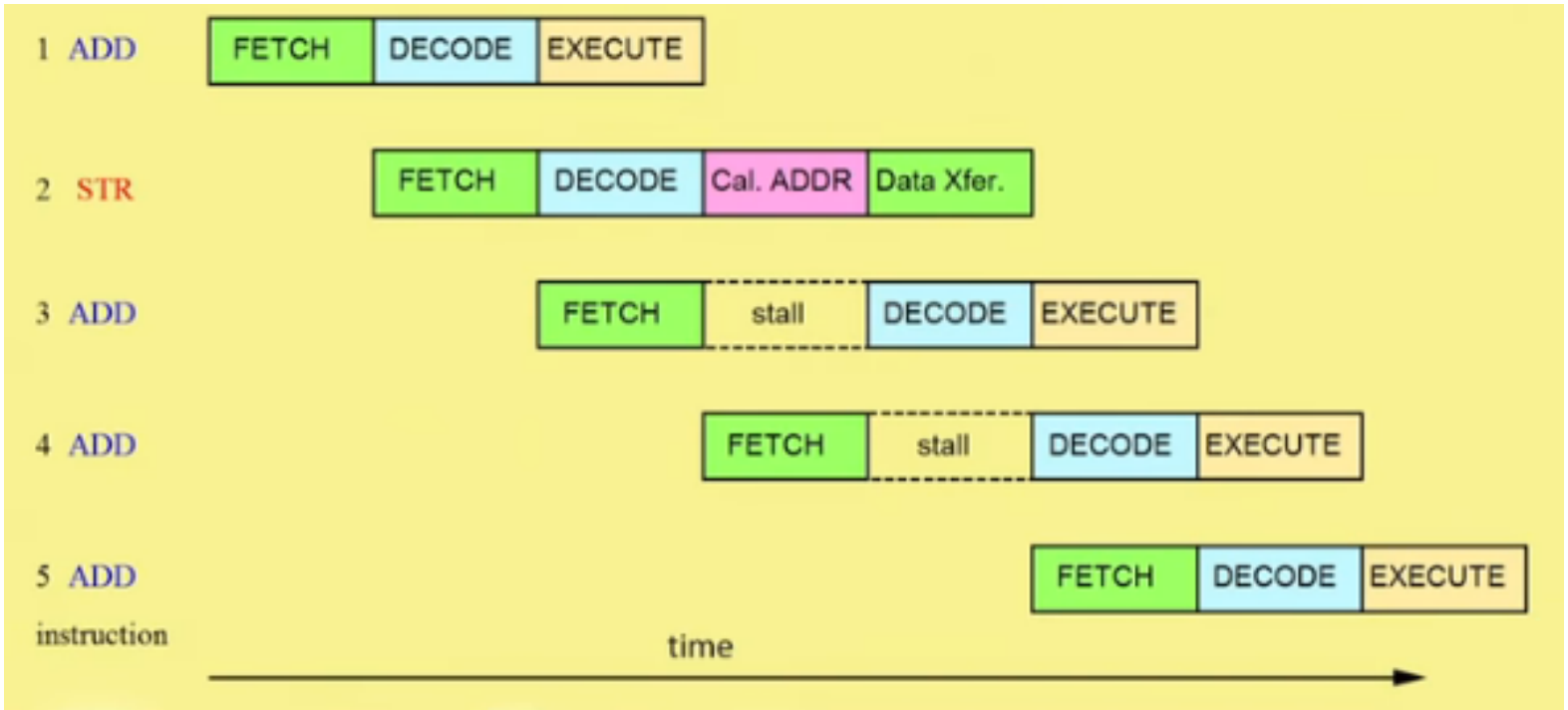
ARM Pipelining Examples



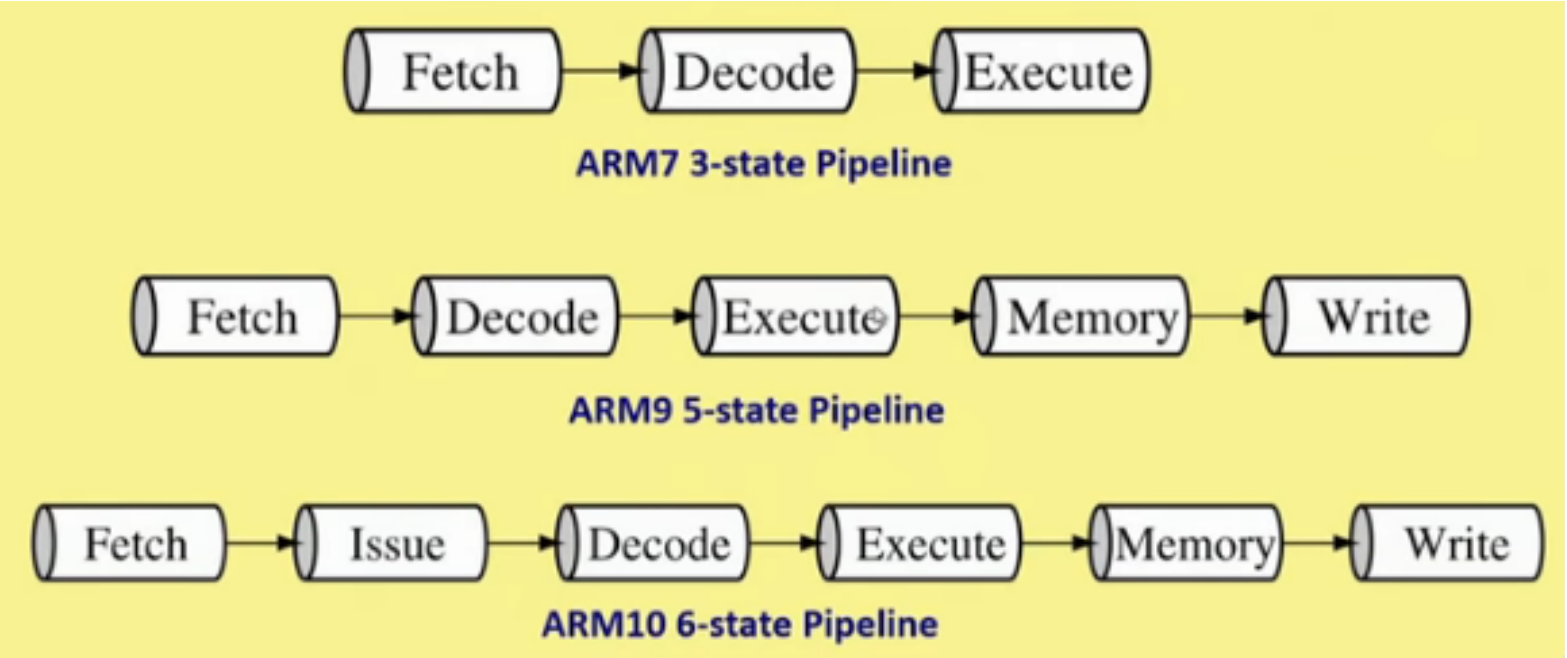
ARM Pipelining Examples



With more complex instruction stall cycles possible

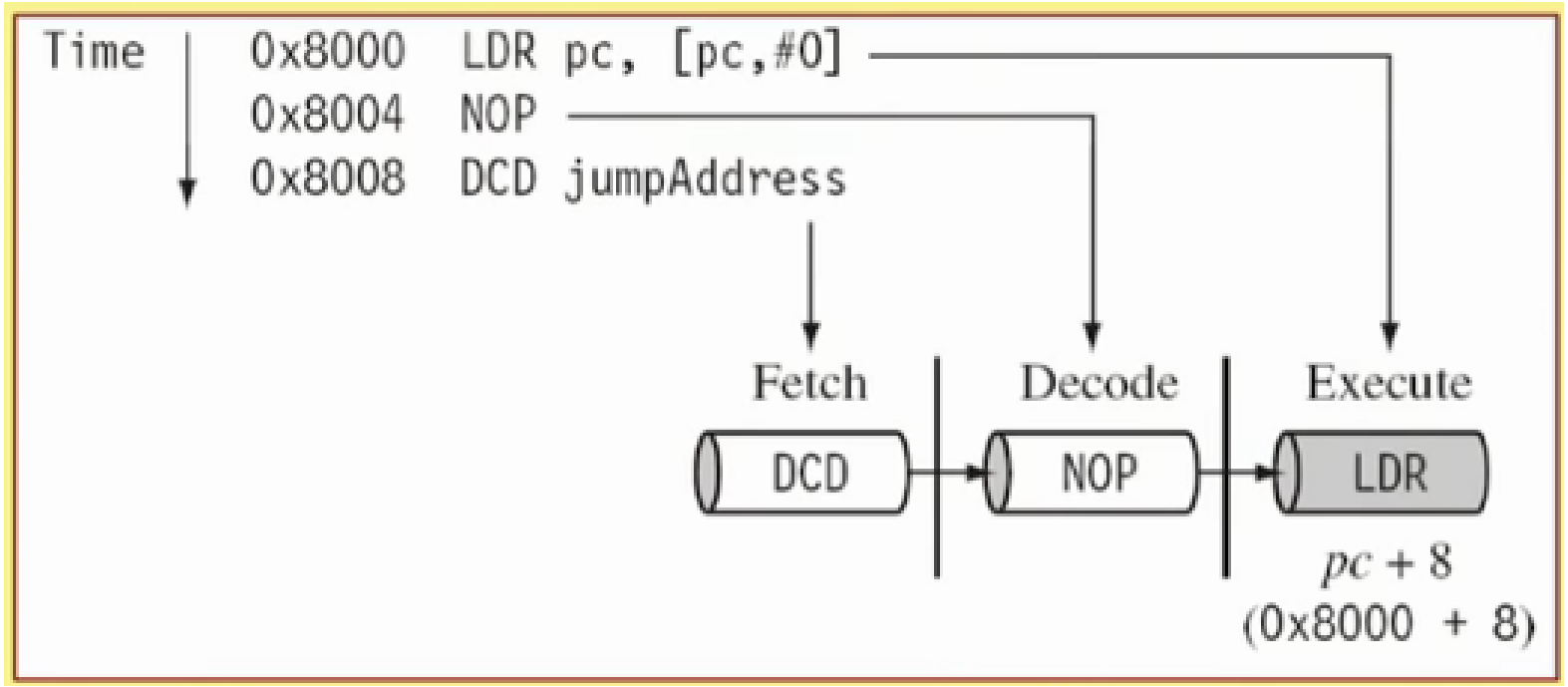


Bird Eye View



- ARM10 has added **Issue** stage in pipeline to prevent pipeline hazards
- Pipeline hazards are situation that prevents the next instruction stream to execute on its designated clock cycle.
- Three types of pipeline hazards:
  - Structural hazards
  - Data hazards
  - Control hazards

ARM Instruction and Memory



- In ARM instruction execution, the program counter (PC) is always 8 bytes ahead
- ARM instruction is of 32 bit, it meas 4 bytes
- ARM Memory design architecture is byte addressable.
  - Structural hazards
  - Data hazards
  - Control hazards

ARM Processor Mode

Processor Modes	Code	Descriptions
User	usr	Normal program execution mode
FIQ	fiq	Enter when a high priority(fast) interrupt is raised
IRQ	irq	Enter when a low priority(low) interrupt is raised
Supervisor	svc	A protected mode for the operating system (entered on reset and when software instruction is executed)
Abort	abt	Used to handle memory access violations
Undefined	und	Used to handle undefined instructions
System	sys	Runs privileged operating system task

ARM Register Sets

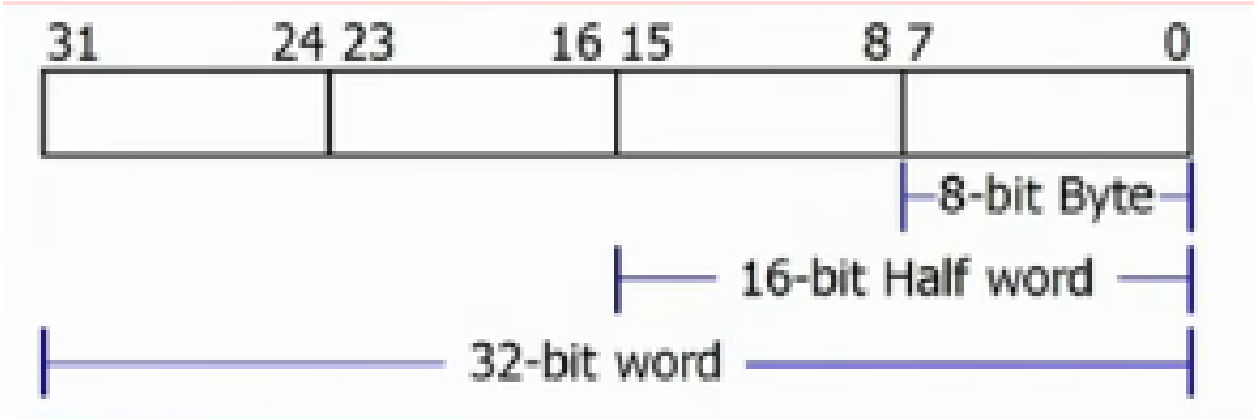
- ARM has 37 registers all of which are 32 bits long
- These registers are:
  - 1 dedicated program counter (PC)
  - 1 dedicated current program status register (CPSR)
  - 5 dedicated saved program status register (SPSR)
  - 30 general-purpose register (GPR)

Registers (contd.)

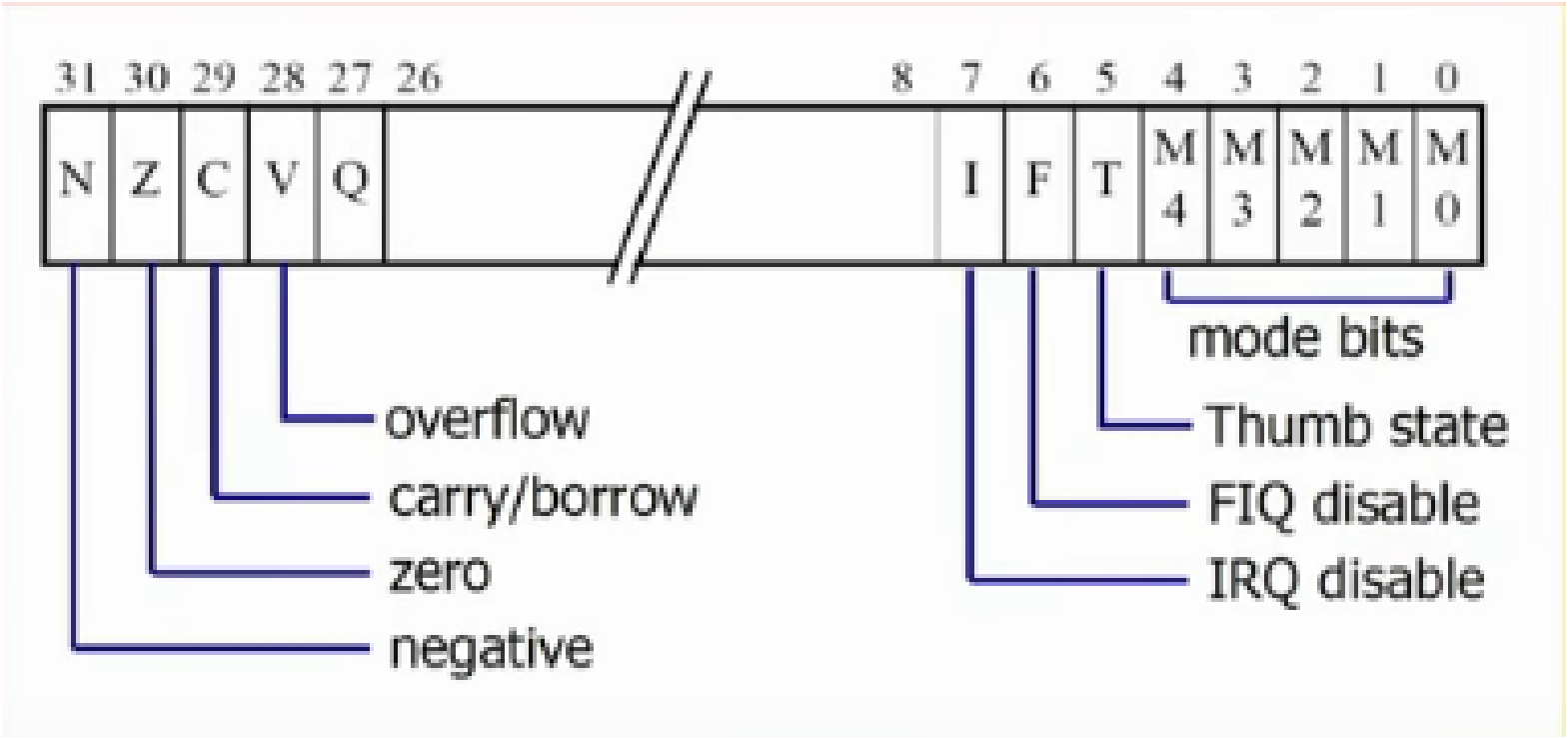
- The current processor mode governs which of several register sets is accessible.
- Only 16 registers are visible to a specific mode of operation.
- Each mode can access:
  - A particular set of registers r0-r12
  - r13 (SP, stack pointer)
  - r14 (LR, link register)
  - r15 (PC, program counter)
  - Current program status register (CPSR)
- Privilege modes (except system) can also access a particular SPSR.

General-purpose Registers

- 6 data types are supported(signed/unsigned)
  - 8 bit byte, 16-bit half-word, 32 bit word
- All ARM operations are 32-bit
  - Shorter data types are only supported by data transfer operations.



Current Program Status Register (CPSR)



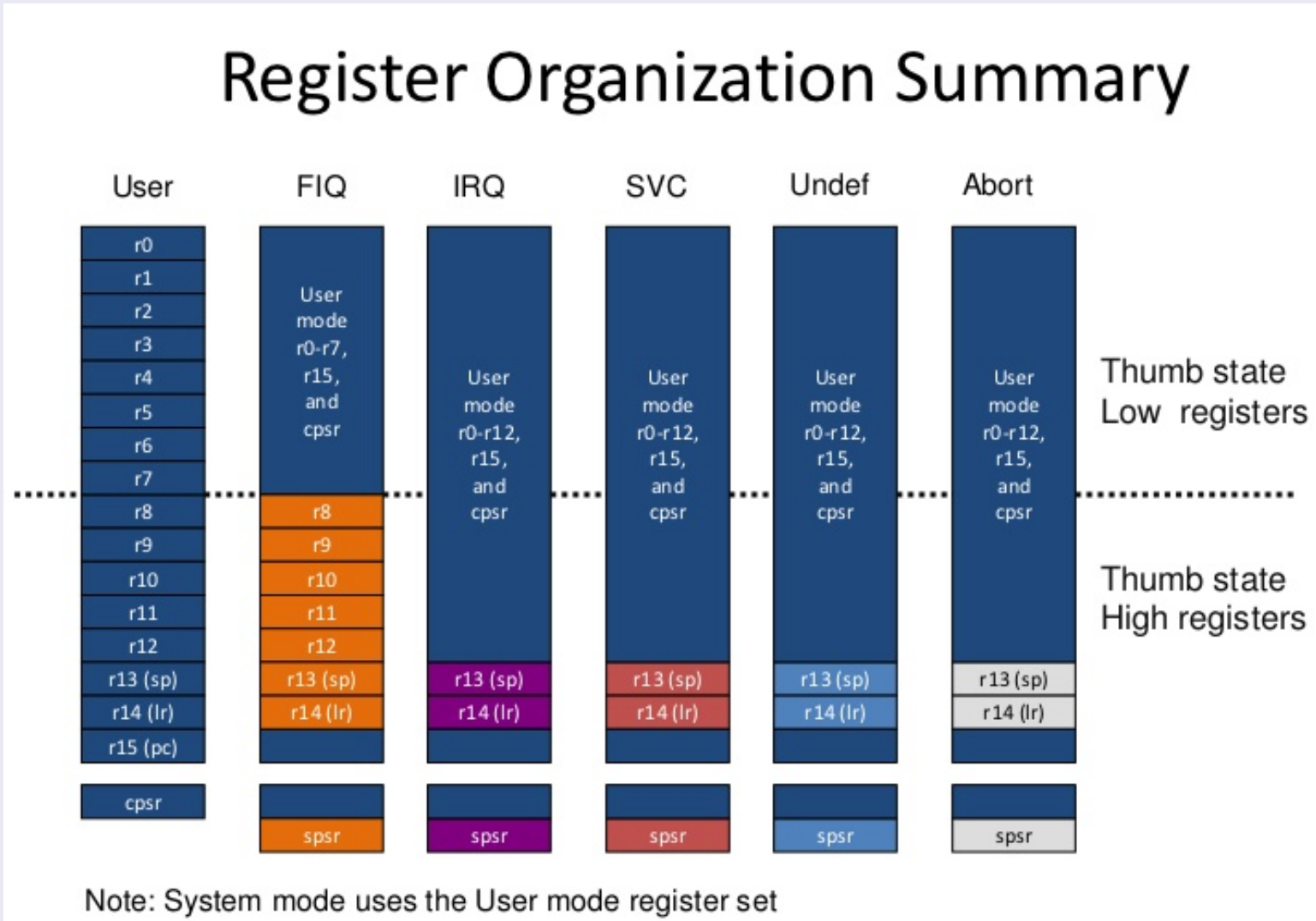
Special Registers

- PC(r15): Any instruction with PC as its destination register is a program branch.
- LR(r14): Saves a copy of PC when executing the BL instruction (subroutine call) or when jumping to an exception or an interrupt handler.
  - it is copied back to PC on return from those routines.
- SP(r13): There is no stack in the ARM architecture.
  - R13 is reserved as a pointer for the software-managed stack.
- CPSR: Hold the visible status register.
- SPSR: Hold a copy of the previous status register while executing exception or interrupt handler routine
  - Copied back to CPSR register on return from execution or interrupt

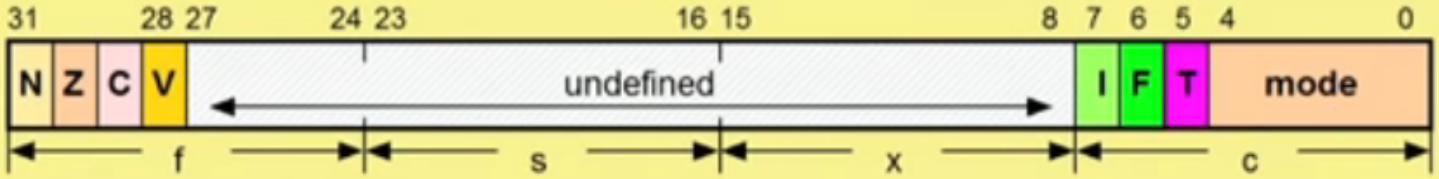
Program Counter

- When the processor is executing in ARM mode:
  - All instructions are 32-bits wide, and must be word aligned.
  - Word aligned: All instruction start from an address that is multiple of 4.
  - The last two bits of PC are zero (i.e. not used).
  - Due to pipelining, PC points 8 bytes ahead of the current instruction, or 12 bytes ahead if the current instruction includes a register-specified shift.
- When the processor is executing in Thumb mode:
  - All instructions are 16-bits wide, and are half-word aligned.
  - The last bit of the PC is zero (i.e. not used).

Register Organization Summary



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

Mode bits

10000	User
10001	FIQ
10010	IRQ
10011	Supervisor
10111	Abort
11011	Undefined
11111	System

Interrupt Disable bits.

I = 1: Disables the IRQ.

F = 1: Disables the FIQ.

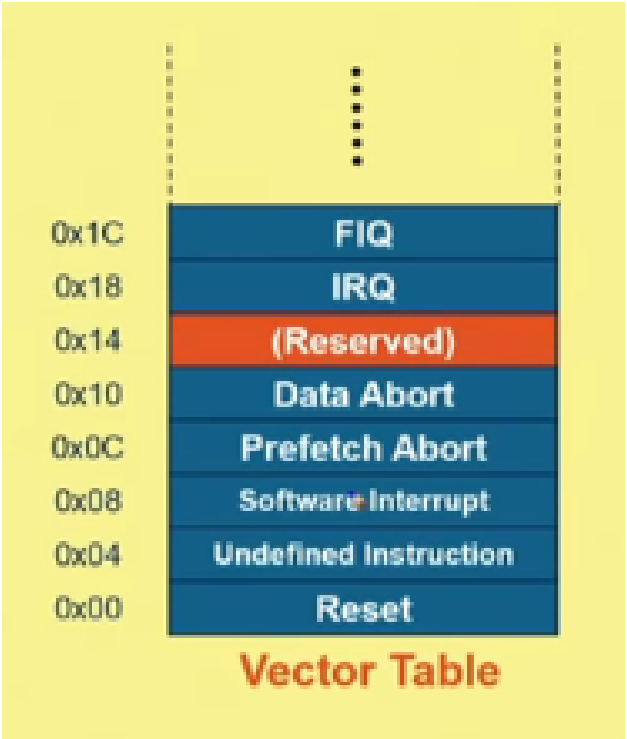
T Bit (Arch. with Thumb mode only)

T = 0: Processor in ARM state

T = 1: Processor in Thumb state

Exception Handling

- When an exception occurs, the processor
  - Copies CPSR into SPSR_< *mode* >
  - Sets appropriate bits in CPSR
    - Changes to ARM state
    - Changes to related mode
    - Disables IRQ, FIQ
  - Stores return address in LR_< *mode* >
  - Sets PC to vector address
- To return, the exception handler needs to:
  - Restore CPSR from SPSR_< *mode* >
  - Restore PC from LR_< *mode* >



- Exception handling in ARM is controlled through an area of memory called vector table.
  - Exists at the bottom of the memory map from 0x00 to 0x1c.
  - Within this table, one word is allocated to each of the various exception types.
  - This word will contain some form of ARM instruction that should perform a branch.
    - Does not contain an address.

ARM and Thumb Instruction Set

- Most ARM implementations provide two instruction sets:
  - 32 bit ARM instruction set
  - 16 bit Thumb instruction set

	ARM (cpsr T = 0)	Thumb (cpsr T = 1)
Instruction size	32-bit	16-bit
Core instructions	58	30
Conditional execution	Most	Only branch instructions
Data processing instructions	Access to barrel shifter and ALU	Separate barrel shifter and ALU instructions
Program status register	Read-write in privileged mode	No direct access
Register usage	15 GPRs + pc	8 GPRs + 7 high registers + pc

What is Conditional Execution?

- It controls whether or not the CPU will execute an instruction.
- Most instructions have a condition attribute that determines whether it will be executed based on the status of the condition flags.
  - Prior to execution, the processor compares the condition attribute with the condition flags in CPSR.
  - If they do not match, the instruction is not executed.
  - Example
    MOVEQ r1,#0 (if zero flag is set, then r1=0)
  - The condition attribute (here EQ) is suffixed to the instruction mnemonic

Thank You