key features of RISC and pipelines

- Small and simple instruction set
- Most instructions execute in single clock cycle (no microcode)
- Memory access instructions only transfer data: memory <-> registers
- ALU operations are register to register
- Large number of general purpose registers

- RISC lends itself easily to efficient instruction execution pipeline
  - Pipelines accelerate performance
  - Pipelines are not always used in embedded architectures
    - Hardware overheads are necessary to implement and control a pipeline
    - But many embedded processors are pipelined, eg. ARM Cortex cores
      - ARM Cortex M3 – 3 stage pipeline
      - ARM Cortex A9 – variable length, 9-11 stage pipeline

even simplest microcontrollers use pipelines

- AVR has a two-stage pipeline: parallel instruction fetches and instruction executions
- This is enabled by the Harvard architecture of the AVR and the use of Instruction Register.
- This basic pipelining results in fast execution, single-clock cycle per instruction.

AVR parallel fetch and execution

- Instruction loaded into Instruction Register executes
- At the same time next instruction is fetched from Program Memory
### Pipeline hazards

- Instruction cycle broken into \( n \) phases (one execution stage per phase)
  - e.g. Fetch, Decode, ReadOPs, Execute1, Execute2, WriteBack
- A new instruction is fetched at each phase
- Maximum speed gain is \( n \) times
- Pipeline hazards reduce the ability to achieve a gain of \( n \) times
  - Types of Hazards
    - Resource
      - Hazard occurs when instruction needs a resource being used by another instruction
    - Data
      - RAW (read-after-write hazard: read is requested before write has finished)
      - WAR (write-after-read hazard: write is requested before read is finished)
      - WAW (write-after-write hazard: writes occur in unintended order)
    - Control
      - Hazard occurs when a wrong fetch decision at a branch results in an extra instruction fetch and pipeline flush

### Data hazards

- Dependences between instructions may cause data hazards when \( I_n \) and \( I_m \) are so close that their overlapping within the pipeline would change their access order to Reg.
  - Three types of data hazards:
    - Read After Write (RAW): \( I_n \) tries to read operand before \( I_m \) writes it.
    - Write After Read (WAR): \( I_n \) tries to write operand before \( I_m \) reads it.
    - Write After Write (WAW): \( I_n \) tries to write operand before \( I_m \) writes it.

### Data access hazard - example

![Data access hazard diagram](image)

### Pipeline conflict due to data dependency hazard

![Pipeline conflict diagram](image)
Solutions to data access and data dependency hazards

• Software solutions: (Compiler scheduling):
  - Putting no-op instructions after each instruction that may cause a hazard
  - Instruction scheduling: rearrange code to reduce no-ops

• Hardware solutions:
  - Hazard detection hardware is necessary.
  - Interlocking: stall pipeline for one or more cycles
  - Forwarding: two types of forwarding:
    - The ALU result of Instr₁ in EX stage can immediately be forwarded back to ALU input of EX stage as an operand for Instr₂.
    - The memory load data from MEM stage can be forwarded to ALU input of EX stage.
  - Forwarding with interlocking:
    - Assuming that Instr₂ is data dependent on the load instruction Instr₁ then Instr₂ has to be stalled until the data loaded by Instr₁ becomes available.

Hardware solution to data dependency hazard – interlocking

Solution to data dependency hazard - forwarding

Example of data dependency hazard not resolvable by forwarding
Control hazards

Control hazards occur when a wrong fetch decision results in a new instruction fetch and the pipeline being flushed.

Dealing with control hazards may require significant hardware overheads.

Solutions include:
- Multiple Pipeline streams
- Prefetching the branch target
- Using a Loop Buffer
- Branch Prediction
- Delayed Branch
- Reordering of Instructions
- Multiple Copies of Registers (backups)

3-stage ARM Cortex M3 pipeline

- Each instruction is executed in three stages:
  - Fetch – instruction is fetched from memory and placed in pipeline;
  - Decode – instruction is decoded and data path signals prepared for next cycle;
  - Execute – instruction reads ALU operands from registers and writes result to destination register.

- ARM pipeline is linear; processor throughput is one instruction per clock cycle while individual instruction takes three clock cycles.
  - When a branch instruction is fetched, a pipeline faces difficulties, because wrong instructions may have been fetched into the pipeline. If this occurs, pipeline flushes and has to be refilled. ARM pipeline maintains high performance and predicts branch behaviour because PC address is calculated 2 instructions ahead of current instruction.

MIPS Pipeline

- Pipelined version of MIPS is a good example of a pipelined RISC architecture (relatively simple and real)

- We will first recall the single-cycle (non-pipelined) MIPS architecture and its data path organisation

- And we will look how MIPS pipeline works
MIPS (RISC) Instructions

I-type (immediate):
- Operation: \( a \) ( immediate )
- Target: 5
- Source register specifier: \( s \)
- Immediate value: \( t \)
- Function: ALU ALU function specifier

Floating Point Instruction Formats

<table>
<thead>
<tr>
<th>op</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Single-cycle MIPS architecture

PC increment

Add

PC

Address Instruction memory

Relative branch

Data

Instruction

Register #

Registers

Address

Data memory

Register #

ALU

Address

Data

NB: This diagram does not show J (absolute jump) instruction data path

Single-cycle MIPS with control signals shown

NB: This diagram does not show J (absolute jump) instruction data path
Single-cycle MIPS with Instruction Decoder (Control) shown

MIPS Pipeline Stages

MIPS Pipeline registers and data paths

Single-cycle MIPS with J data path shown
<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>Id</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>(64 bits)</td>
<td>(142 bits)</td>
<td>(107 bits)</td>
<td>(71 bits)</td>
<td></td>
</tr>
</tbody>
</table>

Pipelined MIPS with control signals shown