Technische Hochschule Rosenheim
Technical University of Applied Sciences

Prof. Dr. Florian Künzner
Technical University of Applied Sciences Rosenheim, Computer Science

CA 4 – Processor 1

The lecture is based on the work and the documents of Prof. Dr. Theodor Tempelmeier

CAMPUS Rosenheim
Computer Science

Overview

What are the properties of a processor?

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

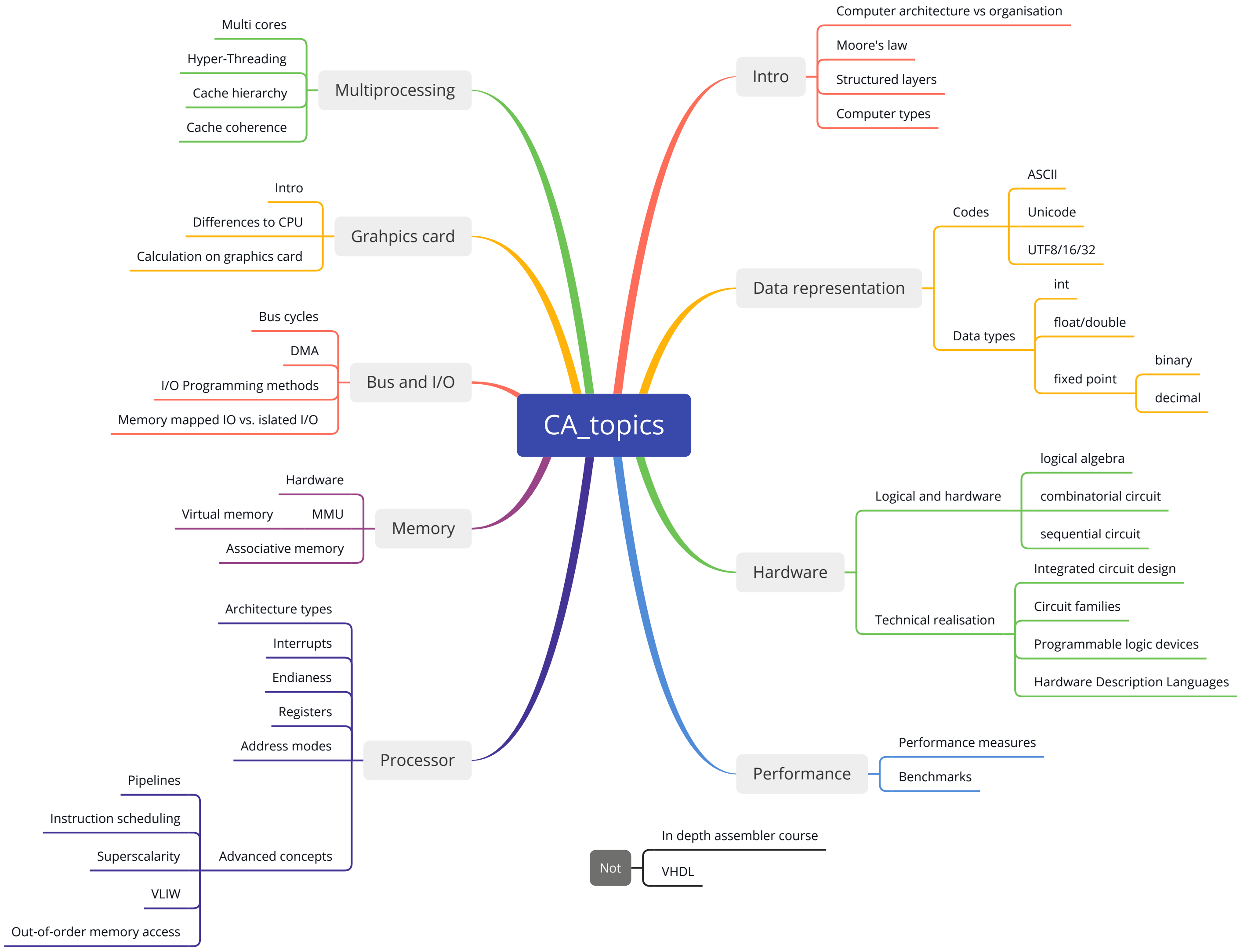
Processor properties

- Processor architecture
- Exception and interrupt handling
- Instruction set architecture
- Memory model (part of memory lectures)
- Endianness
- Registers
- Addressing modes
- Advanced concepts
  - Instruction scheduling
  - Pipelines
  - Superscalarity
  - VLIW
  - Out-of-order memory access

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal



CAMPUS Rosenheim
Computer Science

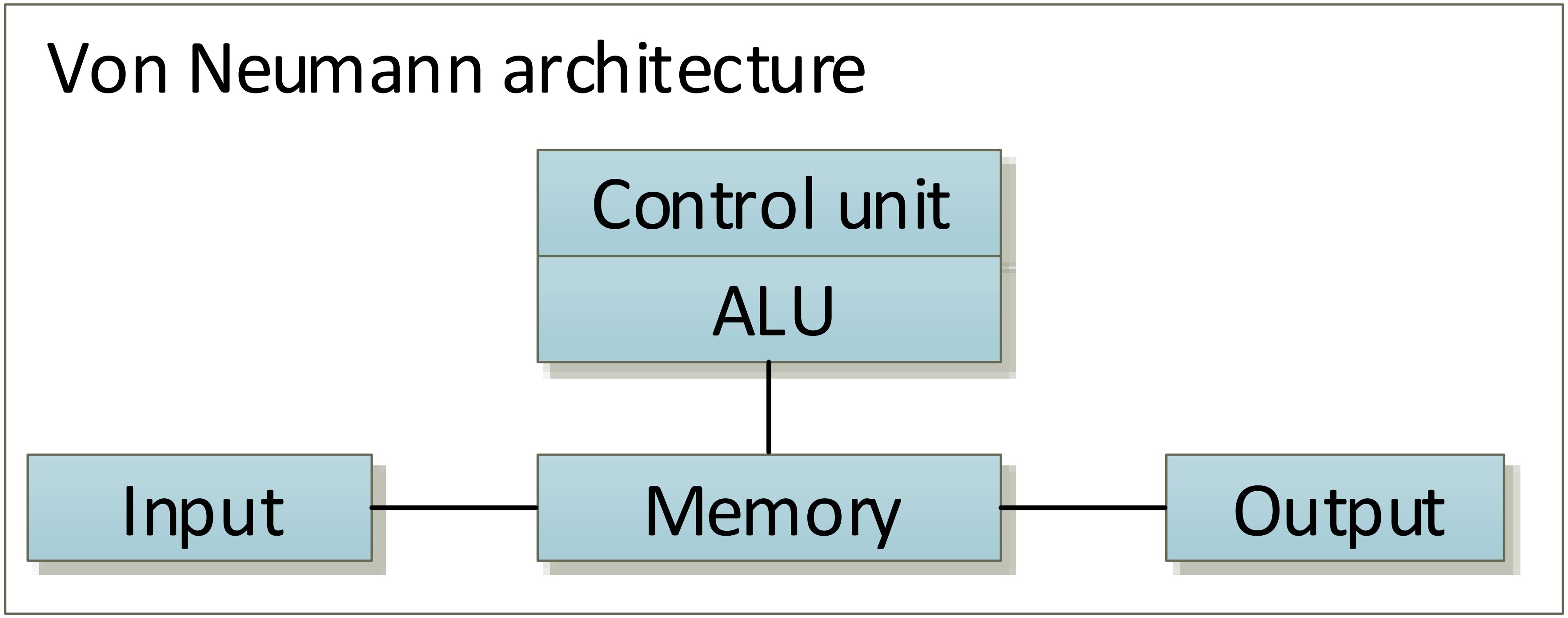
Goal

CA::Processor 1

- Processor architecture
- Exception and interrupt handling
- Instruction set architecture

CAMPUS Rosenheim
Computer Science

Von Neumann architecture



[schematic, simplified view]

CAMPUS Rosenheim
Computer Science

Von Neumann architecture

Properties:

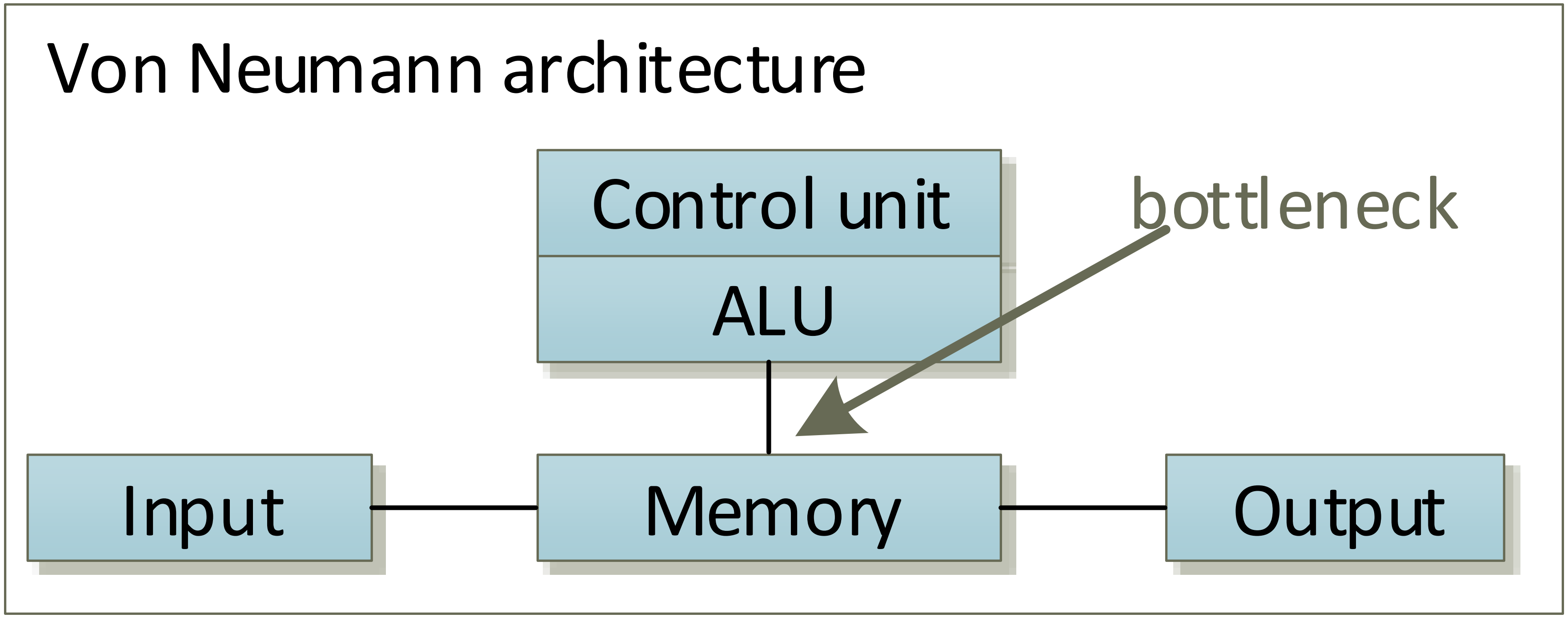
- **Instructions and data** are located in the **same memory** or address space
- Von Neumann – **bottleneck**:
  Instruction execution time $<$ Memory access time

Is the Von Neumann bottleneck still relevant?

In order to avoid and mitigate such problems, various strategies have been developed (e.g. cache memory)–but its still there!

CAMPUS Rosenheim
Computer Science

Von Neumann architecture – bottleneck

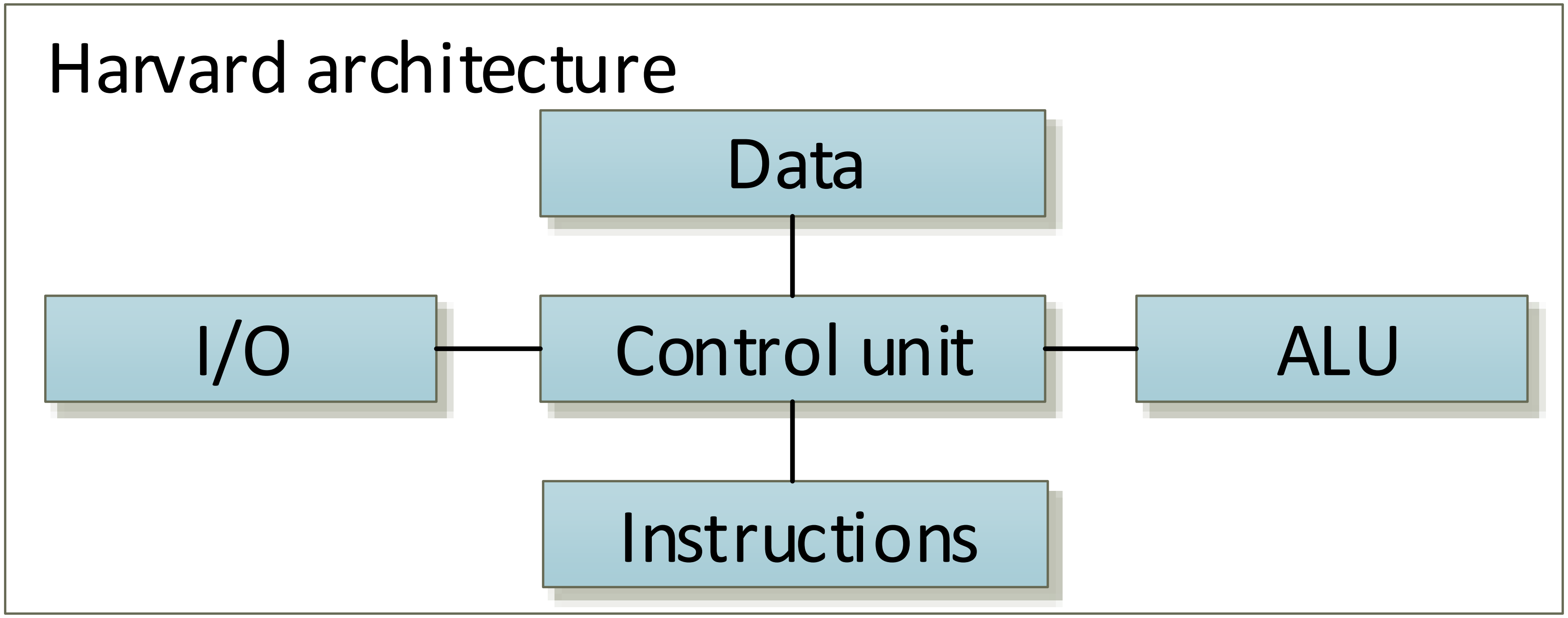


[schematic, simplified view]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Harvard architecture



[schematic, simplified view]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Harvard architecture

Properties:

- **Separate memory** for **data** and **instructions**
- **Data** memory is usually **read- and writeable**
- **Instruction** memory is usually **read-only**. Can't be modified through runtime

CAMPUS Rosenheim
Computer Science

Processor architecture

Discussion

Von Neumann vs Harvard architecture: Does it play a role nowadays?

CAMPUS Rosenheim
Computer Science

Control unit

Pseudo C code of control unit inside the CPU:

```
while(true){
    fetch_next_instruction();
    decode_instruction();
    execute_instruction();
    if(interrupt_is_requested()) {
        save_PC_and_SR();
        load_new_PC();
    }
}
```

Instruction cycle: in principle, it's an **endless loop**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupts

What is an interrupt?

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt handling

An **interrupt request (IRQ)** causes the processor to **stop the current workflow** and to **process with a predefined interrupt service routine (ISR)**.

- **IRQ** -> **Interrupt request**
- **ISR** -> **Interrupt service routine**
- An ISR is a **function** that is called when an interrupt occurs
- It's just a **memory address** where the function (ISR) starts
- There exist **different types** of interrupts
- The ISR addresses are managed within the CPU with the **interrupt vector table**
- The **interrupt vector table** can be **manipulated** (e.g. in supervisor mode by the OS kernel)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt types

Name	Usual name	Reason, cause	Arrival	Comment
Reset	Reset	external	asynchronous	reset potentially at any point in an instruction
Interrupt	Real interrupt, IRQ	external (e.g. I/O)	asynchronous	is usually handled at the end of an instruction
Exception	Internal interrupt	internal (e.g. instruction error)	synchronous	is usually cancelled and may be repeated later
System call	Supervisor call (SVC), Trap, Software interrupt	internal	synchronous	SVC n (n is the number of the SVC)
Timer	Timer, SysTick, Watchdog	external (clock)	asynchronous	is usually handled at the end of an instruction

...

[for example: ARM Cortex-M Exception handlers]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

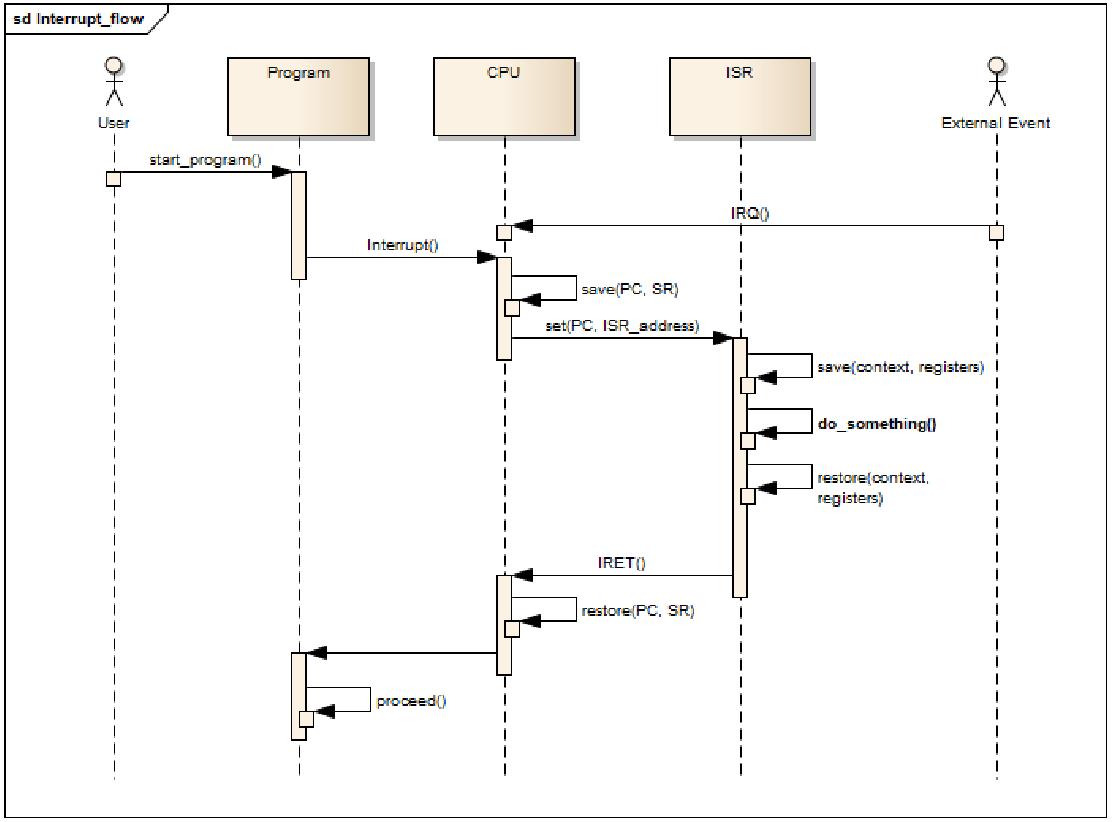
Exceptions

Examples for exceptions:

- Division by zero
- Illegal instruction code
- Load or store to an unaligned address
- Unauthorized memory access

CAMPUS Rosenheim
Computer Science

Interrupt flow



Sequence in the control unit

1. Save the old
   - PC (program counter) and
   - SR (status register)

   (e.g. on the stack)
2. Assign new values to
   - PC (program counter) and
   - SR (status register)

   from a fixed address ("interrupt vector")

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt vector table

Exception number	IRQ number	Vector	Offset
16+n	n	IRQn	0x40+4n
.		.	.
.		.	.
.		.	.
18	2	IRQ2	0x48
17	1	IRQ1	0x44
16	0	IRQ0	0x40
15	-1	SysTick, if implemented	0x3C
14	-2	PendSV	0x38
13		Reserved	
12			
11	-5	SVCall	0x2C
10		Reserved	
9			
8			
7			
6			
5			
4			0x10
3	-13	HardFault	0x0C
2	-14	NMI	0x08
1		Reset	0x04
		Initial SP value	0x00

- Example: **Cortex M0**
- Interrupt request (IRQ)
- For each IRQ one entry
- Each entry contains the address of an ISR

[source: Cortex-M0 Generic User Guide (2009), p. 2-22]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt details

Who saves the registers?

- The CPU saves and restores PC and SR
- The ISR has to save the other registers at the beginning
- The ISR has to restore the other registers at the end
- Usually, the operating system does this (if you have one) in advance

Interruption in the middle of an instruction?

- Usually at the end of a CPU instruction
- But: exceptions can interrupt a running CPU instruction

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

ISA - instruction set architecture

Do you know some common ISAs?

CAMPUS Rosenheim
Computer Science

ISA - instruction set architecture

The interface for low-level programming, very close to the hardware.

Degrees of freedom in instruction set architecture (ISA) design

Operations:	How many? Which? How complex?
Data types:	Which data types can the operations handle?
Instruction format:	Instruction length in bytes? Number of addresses? Size of the address fields?
Endianness:	Byte order within a long word: Big endian vs little endian?
Register:	How many? Usable in which way?
Addressing:	Addressing types for the operands? Can be combined with the operations arbitrary ("orthogonal") or restricted?

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

ISA - instruction set architecture

Instruction formats:

Instruction address	Example	Operands
Zero-address	`ADD`	Operands and result on stack!
One-address	`ADD X`	`A = A + X (A = "accu")`
Two-addresses	`ADD X, Y`	`X = X + Y or Y = X + Y`
Three-addresses	`ADD X, Y, Z`	`X = Y + Z or ...`

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

ISA - special instructions

Synchronisation instructions:

- TAS (test and set) or similar
- Tests a memory cell ($? = 0$) and sets it to one, if the test was successful.
- In the hardware single, atomic (i.e., non-interruptible) operation
- Uses a read-modify-write memory cycle that completes the operation without interruption.

Synchronisation area:

Area	**Protection through**
Critical area in a process	P and V operation
P-Operation in the operating system	TAS instruction (or similar)

In other architectures, partly different, more efficient, much more complicated solutions. Similar: Compare_and_Swap, keyword "Lock-free Programming".

ISA - special instructions: TAS

TAS example for Freescale ColdFire architecture

```
byte LOCK = 0; // =0 -> lock is free; !=0 -> locked
//...
__asm { ;Inline assembly block with assembler instructions
  GetLock:
    TAS.B LOCK    ;Sets the N- or Z-Bit depending on LOCK and
                  ;always sets LOCK = 0x80
    BNE GetLock   ;If LOCK was != 0 then try it again (loop)
                  ;(BNE, branch not equal)
}
// Now, we have the LOCK and we are inside the critical section
// ...
__asm { ;Inline assembly block with assembler instructions
  ReleaseLock:
    CLR.B LOCK     // Sets LOCK = 0
}
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

ISA - CISC vs RISC

CISC - complex instruction set computer

- Instructions should be as close as possible to the high-level languages
- Can take a lot of internal CPU cycles for an instruction
- Support for a lot of addressing modes
- Allows compact encoding of programs (one instruction does a lot)

RISC - reduced instruction set computer

- Less, simple instructions
- One instruction should only take a few internal CPU cycles
- Supports only simple addressing modes (load/store necessary)
- Usually fixed length of opcodes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

ISA - CISC vs RISC

Discussion

CISC vs RISC: Does it play a role nowadays?

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Summary and outlook

Summary

- Processor architecture
- Exception and interrupt handling
- Instruction set architecture

Outlook

- Endianness