

# r3k.vhdl - MIPS R3000 auf einem FPGA

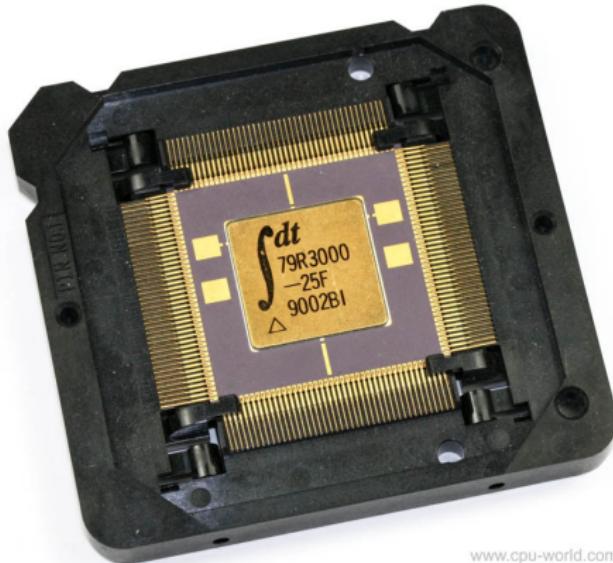
Basispraktikum Technische Informatik

Aicha Ben Chaouacha, Ahmad Fatoum, Niklas Fuhrberg | 05.06.2017

LEHRSTUHL SYSTEMARCHITEKTUR

# Der MIPS R3000

- 1988 auf dem Markt gekommen
- 32-bit, RISC Architektur
- 32 Register
- 5-Stufige Pipeline



[www.cpu-world.com](http://www.cpu-world.com)

# Was wird weg gelassen ?

- Traps
- Interrupts
- Overflow
- Syscall
- Betriebsmodi (Kernel/User)
- MMU
- Caching

# MIPS Befehlssatz

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands; exception possible
	subtract	sub \$1,\$2,\$3	\$1 = \$2 - \$3	3 operands; exception possible
	add immediate	addi \$1,\$2,100	\$1 = \$2 + 100	+ constant; exception possible
	add unsigned	addi \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands; exception possible
	subtract unsigned	subi \$1,\$2,\$3	\$1 = \$2 - \$3	3 operands; exception possible
	add immediate unsigned	addi \$1,\$2,100	\$1 = \$2 + 100	+ constant; exception possible
	Move from coprocessor register	mfc0 \$1,\$epc	\$1 = \$epc	Used to get of Exception PC
Logical	and	and \$1,\$2,\$3	\$1 = \$2 & \$3	3 register operands; Logical AND
	or	or \$1,\$2,\$3	\$1 = \$2   \$3	3 register operands; Logical OR
	and immediate	and \$1,\$2,100	\$1 = \$2 & 100	Logical AND register, constant
	or immediate	or \$1,\$2,100	\$1 = \$2   100	Logical OR register, constant
	shift left logical	sll \$1,\$2,10	\$1 = \$2 << 10	Shift left by constant
	shift right logical	srl \$1,\$2,10	\$1 = \$2 >> 10	Shift right by constant
Data transfer	load word	lw \$1,(100)\$2	\$1 = Memory[\$2+100]	Data from memory to register
	store word	sw \$1,(100)\$2	Memory[\$2+100] = \$1	Data from memory to register
	load upper immediate	lui \$1,100	\$1 = 100 * 2 <sup>16</sup>	Load constant in upper 16bits
Conditional branch	branch on equal	beq \$1,\$2,100	if (\$1 == \$2) go to PC+4+100	Equal test; PC relative branch
	branch on not equal	bne \$1,\$2,100	if (\$1 != \$2) go to PC+4+100	Not equal test; PC relative
	set on less than	slt \$1,\$2,\$3	if (\$2 < \$3) \$1 = 1; else \$1 = 0	Compare less than; 2's complement
	set less than immediate	slti \$1,\$2,100	if (\$2 < 100) \$1 = 1; else \$1 = 0	Compare < constant; 2's complement
	set less than unsigned	sltu \$1,\$2,\$3	if (\$2 < \$3) \$1 = 1; else \$1 = 0	Compare less than; natural number
	set less than immediate unsigned	sltiu \$1,\$2,100	if (\$2 < 100) \$1 = 1; else \$1 = 0	Compare constant; natural number
	jump	j 10000	goto 10000	Jump to target address
Unconditional jump	jump register	j \$31	goto \$31	For switch, procedure return
	jump and link	jal 10000	\$31 = PC + 4; go to 10000	For procedure call

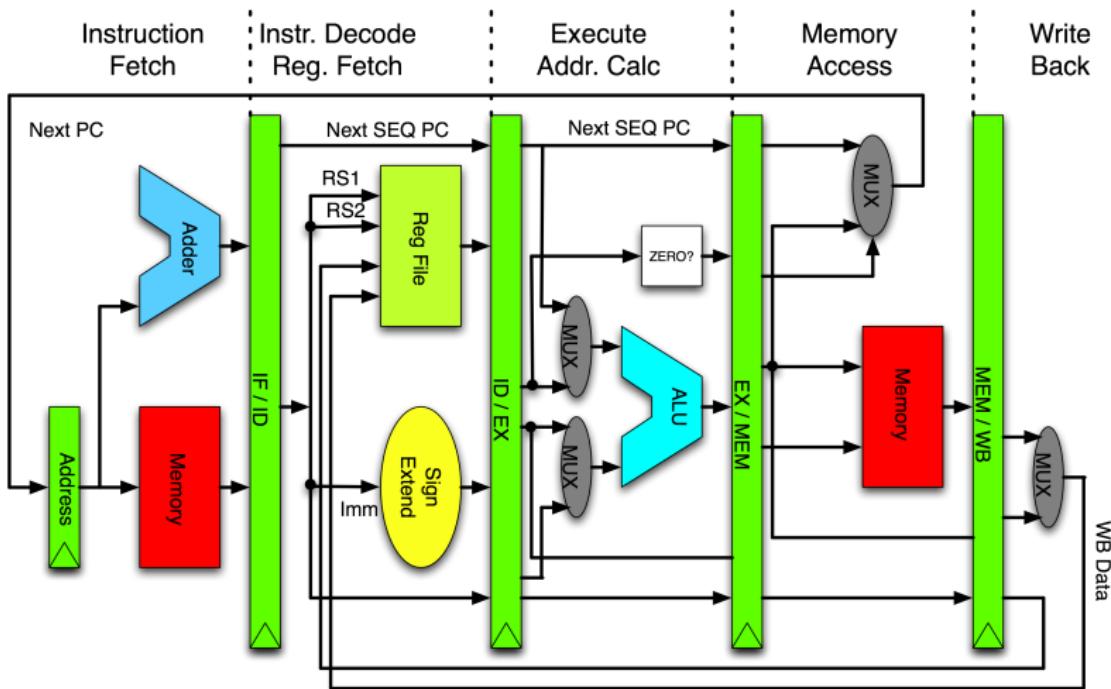
# MIPS Befehlsformate

- R-Type – Arithmetik and Logik
- I-Type – Laden/Speichern, Verzweigen, Direktwert
- J-Format – Jump

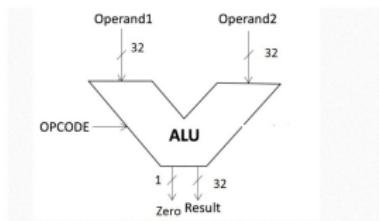
## BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	
	31 26 25	21 20	16 15	11 10	6 5		0
I	opcode	rs	rt		immediate		
	31 26 25	21 20	16 15				0
J	opcode			address			
	31 26 25						0

# MIPS Pipeline



# Wie werden wir das Ganze implementieren? Bsp.: ALU



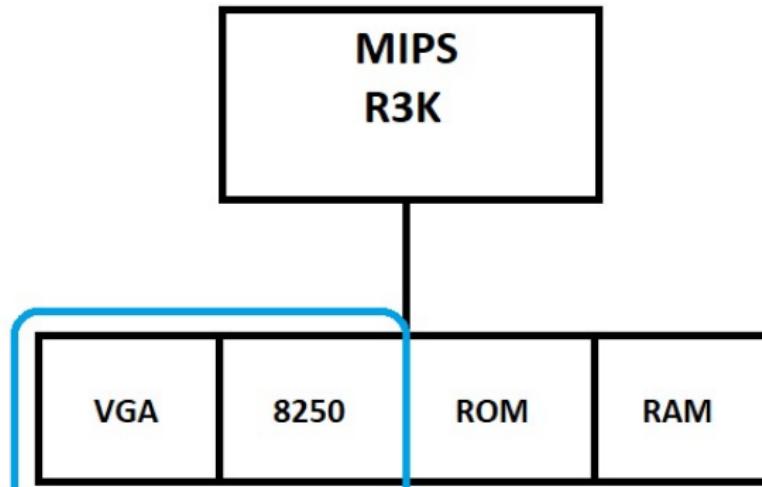
```
entity ALU is
port (
    operand1, operand2 : in std_logic_vector(31 downto 0);
    opcode:in opcode ; --enum
    result : out std_logic_vector(31 downto 0);
    zero : out std_logic;
);
end ALU ;

architecture behv of ALU is
begin
    process (operand1, operand2, opcode)
    begin
        case opcode is
            when alu_and => result <= operand1 and operand2;
            when alu_or => result <= operand1 or operand2;
            when alu_nor => result <= operand1 nor operand2;
            when others => result <= (others => 'X');
        end case;
    end process;

```

# Hardware-Komponenten

- MIPS R3000
- 32bit



# UART

- Universal Asynchronous Receiver Transmitter
- NS8250 Class
- Adresse: 0x1fd00x3f8
- Eingabe
- Priorität -> Erlaubt debugging
- keine interrupts

# VGA Frame Puffer

- Adresse: 0x1fd0????
- C Programm schreibt schreibt in Array
- Array wird aus Speicher gelesen
- VGA Clock

# VGA

- 255 Farben
- Größe: 160\*120 -> 19k byte
- klein anfangen, feststellen ob der Speicher reicht

# Strom da. Was nun?

Reset Vektor wird angesprungen. Assembler Code verantwortlich für

- Globale Variablen (.BSS) nullen
- Daten von ROM in den RAM kopieren
- Stack initialisieren
- main() im C Code anspringen

# BIOS (Basic Input/Output System)

Verantwortlich für

- Initialisiert UART
- Startet Kommandozeilen Interpreter auf UART
- Startet Applikationen
- Interrupts und Traps?

# C-Toolchain

- GCC 6.2.0 MIPS Barebones Cross-Compiler
- -ffreestanding -nostdlib -nostartfiles
- Wie kommt das Programm in den Speicher?
  - Linker Script: Spezifiziert Addressvergabe an Symbole
  - objcopy: Extrahiert rohen Maschinen Code
  - iMPACT: Spielt extrahierte Binärdatei auf ROM



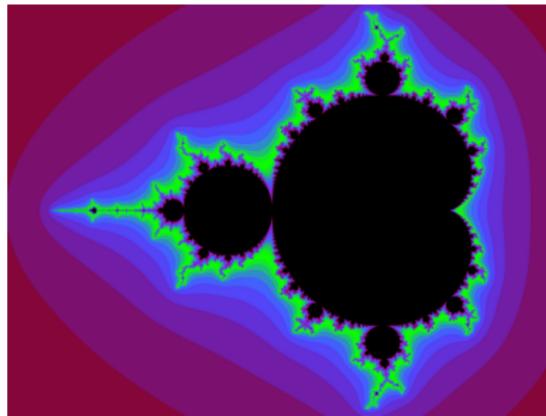
# Komplexität im Griff behalten?

- Emulation:
  - QEMU mit R3000, UART und VGA
  - Code lokal Testbar vor Auspielen auf den FPGA
- Simulation: Testbenches: z.B. ALU und Instruktionsdecoder.  
Alleinstehend, gut testbar.



# Applikationen

- Mandelbrot-Fraktal
- Vllt. DOS Demos portieren



# Doom?



# Aufgaben

- ALU (Aicha)
- Instruction Decoder (Ahmad)
- UART (Niklas)
- Address decoder
- ROM
- RAM
- VGA frame buffer
- Register File
- Pipeline (Data path)

# Fragen? Anregungen?

