Mikrorechner Projekt

1 Allgemein

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri.

1.1 Registers

Diese Tabelle beschreibt die Register unseres Prozessors und deren Funktionen. Sie umfasst die Nummerierung der Register, deren Namen sowie eine kurze Beschreibung ihres Zwecks. Register 0, $z, ist ein schreibgeschütztes Nullregister, das spezielle Aufgaben erfüllt. Die Register 1 bis 29, $1 bis $29, sind Generalzweckregister und flexibel einsetzbar. Register 30, $bp, fungiert als Basiszeiger für das Stackframe und erleichtert die Verwaltung von Funktionsaufrufen, während Register 31, $sp, als Stackpointer die Spitze des Stacks anzeigt und für die Speicherverwaltung genutzt wird.

Number	Register	Beschreibung
0	$z	Nullregister (**READ ONLY**)
1 - 29	$1 - $29	Generalzweck Register
30	$bp	Stackframe Basepointer
31	$sp	Stackpointer

1.2 Instruction Formats

Diese Tabelle beschreibt die Instruktionsformate unseres Prozessors, gegliedert nach den Typen R, I und J. Jede Instruktion wird durch die Verteilung ihrer Bits auf bestimmte Felder definiert, die in der Tabelle detailliert dargestellt sind.

Type	Bits					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

1.3 Instruction Reference

Diese Tabelle stellt die Instruktionen unseres Prozessors dar und bietet eine Übersicht über ihre wesentlichen Merkmale. Jede Zeile enthält den Op Code, der den Maschinenbefehl identifiziert, das Format, das die Struktur der Instruktion beschreibt, sowie den Mnemonic, eine lesbare Kurzform für den Befehl. Schließlich wird unter Action die Funktion der jeweiligen Instruktion beschrieben, also die Operation, die der Prozessor ausführt. Diese Darstellung erleichtert die Entwicklung und das Debugging von Programmen für unseren Prozessor. Für alle Instruktionen des Formats R ist der Opcode 0 und die Opcode spalte gibt statdessen die Function an.

Opcode / Funct	Format	Mnemonic	Action
0	R	add $d, $s, $t	$d = $s + $t
1	R	sub $d, $s, $t	$d = $s - $t
2	R	and $d, $s, $t	$d = $s ∧ $t
3	R	or $d, $s, $t	$d = $s ∨ $t
4	R	xor $d, $s, $t	$d = $s ^ $t

Opcode / Funct	Format	Mnemonic	Action
5	R	shl $d, $s, $t	$d = $s ≪ $t
6	R	sal $d, $s, $t	$d = $s ⋘ $t
7	R	shr $d, $s, $t	$d = $s ≫ $t
8	R	sar $d, $s, $t	$d = $s ⋙ $t
9	R	not $d, $s	$d = ~$s
10	R	lts $d, $s, $t	$d = ($s < $t)
11	R	gts $d, $s, $t	$d = ($s > $t)
12	R	ltu $d, $s, $t	$d = ($s < $t)
13	R	gtu $d, $s, $t	$d = ($s > $t)
14	R	eq $d, $s, $t	$d = $s == $t
15	R	ne $d, $s, $t	$d = $s != $t
1	I	lhi $t, i	HH($t) = i
2	I	llo $t, i	LH($t) = i
3	I	lb $t, i($s)	$t = SE(MEM[$s + i]:1)
4	I	lbu $t, i($s)	$t = ZE(MEM[$s + i]:1)
5	I	lh $t, i($s)	$t = SE(MEM[$s + i]:2)
6	I	lhu $t, i($s)	$t = ZE(MEM[$s + i]:2)
7	I	lw $t, i($s)	$t = SE(MEM[$s + i]:4)
8	I	lwu $t, i($s)	$t = ZE(MEM[$s + i]:4)
9	I	sb i($t), $s	MEM[$t + i]:1 = LB($s)
10	I	sh i($t), $s	MEM[$t + i]:2 = LH($s)
11	I	sw i($t), $s	MEM[$t + i]:4 = $s
12	I	br $s, imm	if ($s != 0) pc += imm ≪ 2
13	I	jr $s	pc = $s
14	J	jmp addr	pc += addr ≪ 2
15	I	push $d	MEM[sp] = $d; sp = sp - 4
16	I	pop $d	$d = MEM[sp]; sp = sp + 4
17	J	call addr	MEM[sp] = pc + 4; sp = sp - 4; pc = pc + addr
18	I	callr off($r)	MEM[sp] = pc + 4; sp = sp - 4; pc = $r + off
19	?	ret	pc = MEM[sp]; sp = sp + 4
20	?	trap	Trap the emulator for debugging.
21	?	halt	Halt the CPU.
22	?	nop	Do nothing.

1.4 Architektur