MIPS core instructions	Name	gcc	spice	Remaining instructions	Name	gcc	spice
add	add	0%	0%	FP add double	add.d	0%	4%
add immediate	addi	0%	0%	FP substr. Double	sub.d	0%	3%
add unsigned	addu	9%	10%	FP mult. Double	mul.d	0%	5%
add imm. unsigned	addiu	17%	1%	FP divide double	div.d	0%	2%
substract unsigned	subu	0%	1%	load single to FP	l.s	0%	24%
and	and	1%	0%	store single from FP	s.s	0%	9%
and immediate	andi	2%	1%	branch on FP true	bc1t	0%	1%
shift left logical	sll	5%	5%	branch on FP false	bc1f	0%	1%
shift right logical	srl	0%	1%	FP compare double	c.x.d	0%	1%
load upper imm.	lui	2%	6%	move to FP	mtc1	0%	2%
load word	lw	21%	7%	move from FP	mfc1	0%	2%
store word	sw	12%	2%	convert float/integer	cvt	0%	1%
load byte	lb	1%	0%				
store byte	sb	1%	0%	shift right arithmetic	sra	2%	0%
branch on equal	beq	9%	3%	load half	lh	1%	0%
branch on not eq.	sne	8%	2%	branch if < zero	bltz	1%	0%
jump and link	jal	1%	1%	branch if >= zero	bgez	1%	0%
jump register	jr	1%	1%	branch if <= zero	blez	0%	1%
set less than	slt	2%	0%				
set less than imm.	slti	1%	0%				
set less than unsig.	sltu	1%	0%				
s.l.t. imm. unsigned	sltiu	1%	0%				

Figuur 1: Frequentie van MIPS instructies voor gcc en spice

<table>
<tr><th>Categorie</th><th>Voorbeeld</th><th>Betekenis</th><th>Formaat</th></tr>
<tr><td rowspan="16">Rekenkundige/ vergelijkingen/ moves</td><td>add $s1,$s2,$s3</td><td>$s1 = $s2+$s3</td><td>R</td></tr>
<tr><td>sub $s1,$s2,$s3</td><td>$s1 = $s2-$s3</td><td>R</td></tr>
<tr><td>addi $s1,$s2,100</td><td>$s1 = $s2+100</td><td>I</td></tr>
<tr><td>addu $s1,$s2,$s3</td><td>$s1 = $s2+$s3</td><td>R</td></tr>
<tr><td>subu $s1,$s2,$s3</td><td>$s1 = $s2-$s3</td><td>R</td></tr>
<tr><td>addiu $s1,$s2,100</td><td>$s1 = $s2+100</td><td>I</td></tr>
<tr><td>mfc0 $s1,$epc</td><td>$s1 = $epc</td><td>R</td></tr>
<tr><td>lui $s1,100</td><td>$s1 = 100<<16</td><td>I</td></tr>
<tr><td>slt $s1,$s2,$s3</td><td>$s1 = ($s2<$s3)</td><td>R</td></tr>
<tr><td>sltu $s1,$s2,$s3</td><td>$s1 = ($s2<$s3)</td><td>R</td></tr>
<tr><td>slti $s1,$s2,100</td><td>$s1 = ($s2<100)</td><td>I</td></tr>
<tr><td>sltiu $s1,$s2,100</td><td>$s1 = ($s2<100)</td><td>I</td></tr>
<tr><td>mult $s1,$s2</td><td>hi,lo = $s1x$s2</td><td>R</td></tr>
<tr><td>div $s1,$s2</td><td>lo=$s1/$s2;hi=$s1%$s2</td><td>R</td></tr>
<tr><td>mflo $s1</td><td>$s1 = lo</td><td>R</td></tr>
<tr><td>mfhi $s1</td><td>$s1 = hi</td><td>R</td></tr>
<tr><td rowspan="6">Logische</td><td>and $s1,$s2,$s3</td><td>$s1 = $s2&$s3</td><td>R</td></tr>
<tr><td>or $s1,$s2,$s3</td><td>$s1 = $s2|$s3</td><td>R</td></tr>
<tr><td>andi $s1,$s2,100</td><td>$s1 = $s2&100</td><td>I</td></tr>
<tr><td>ori $s1,$s2,100</td><td>$s1 = $s2|100</td><td>I</td></tr>
<tr><td>sll $s1,$s2,4</td><td>$s1 = $s2<<4</td><td>R</td></tr>
<tr><td>srl $s1,$s2,4</td><td>$s1 = $s2>>4</td><td>R</td></tr>
</table>

Categorie	Voorbeeld	Betekenis	Formaat
Branch	`beq $s1,$s2,100` `bne $s1,$s2,100`	`if ($s1==$s2) goto PC+404` `if ($s1!=$s2) goto PC+404`	I I

Jump	`j 2500`
	`jr $ra`
	`jal 2500`
		`goto 10000`
		`goto $ra`
		`$ra = PC+4; goto 10000`
			J
			R
			J

Data transfer	`lw $s1,100($s2)`
	`sw $s1,100($s2)`
	`lb $s1,100($s2)`
	`lbu $s1,100($s2)`
	`sb $s1,100($s2)`
		`$s1 = Memory[$s2+100]`
		`Memory[$s2+100] = $s1`
		`$s1 = Memory[$s2+100]`
		`$s1 = Memory[$s2+100]`
		`Memory[$s2+100] = $s1`
			I
			I
			I
			I
			I

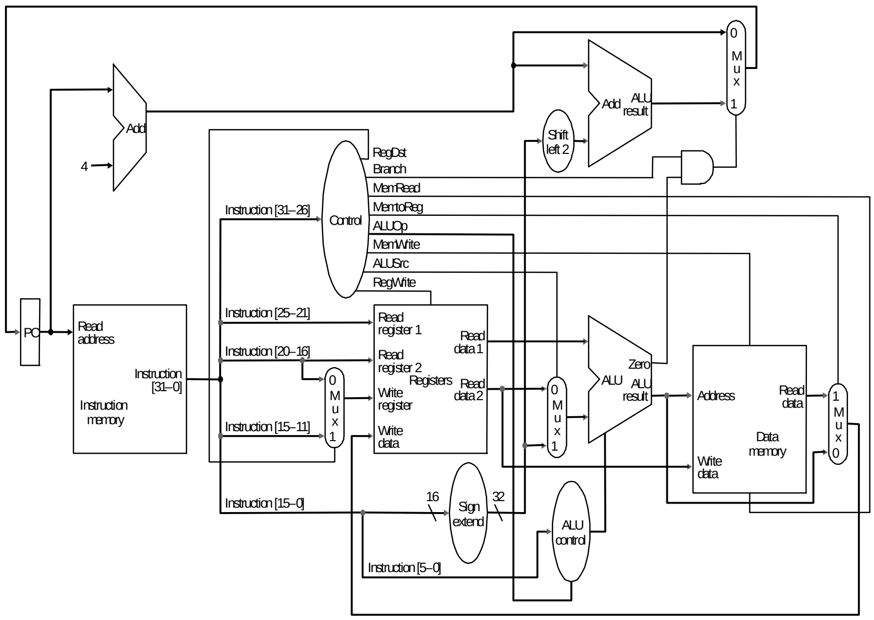
Figuur 2: MIPS instructies

<table>
<tr><td>R</td><td>op</td><td>rs</td><td>rt</td><td>rd</td><td>shamt</td><td>funct</td></tr>
<tr><td>I</td><td>op</td><td>rs</td><td>rt</td><td colspan="3">16-bit immediate</td></tr>
<tr><td>J</td><td>op</td><td colspan="5">26-bit adres</td></tr>
</table>

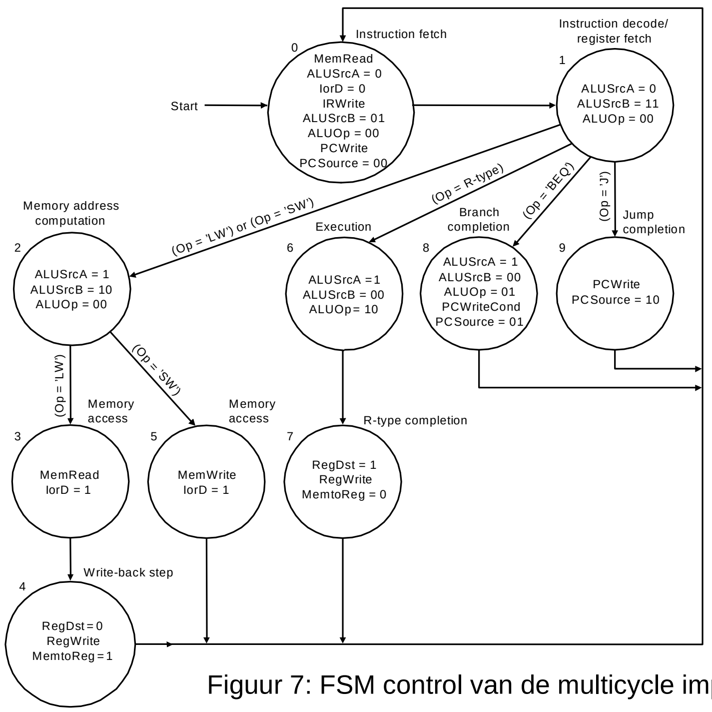
Figuur 3: MIPS instructie-formaten

Name	Register number	Usage
`$zero`	0	the constant value 0
`$v0-$v1`	2-3	values for results and expression evaluation
`$a0-$a3`	4-7	arguments
`$t0-$t7`	8-15	temporaries
`$s0-$s7`	16-23	saved
`$t8-$t9`	24-25	more temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address

Figuur 4: MIPS register conventies



Figuur 5: single-cycle implementatie van de MIPS processor

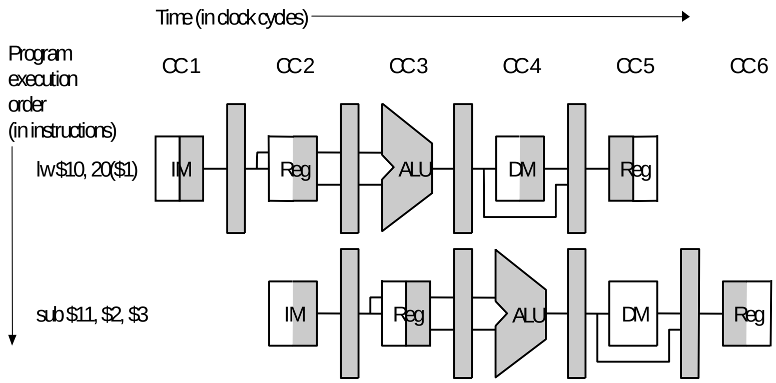


Figuur 7: FSM control van de multicycle implementatie

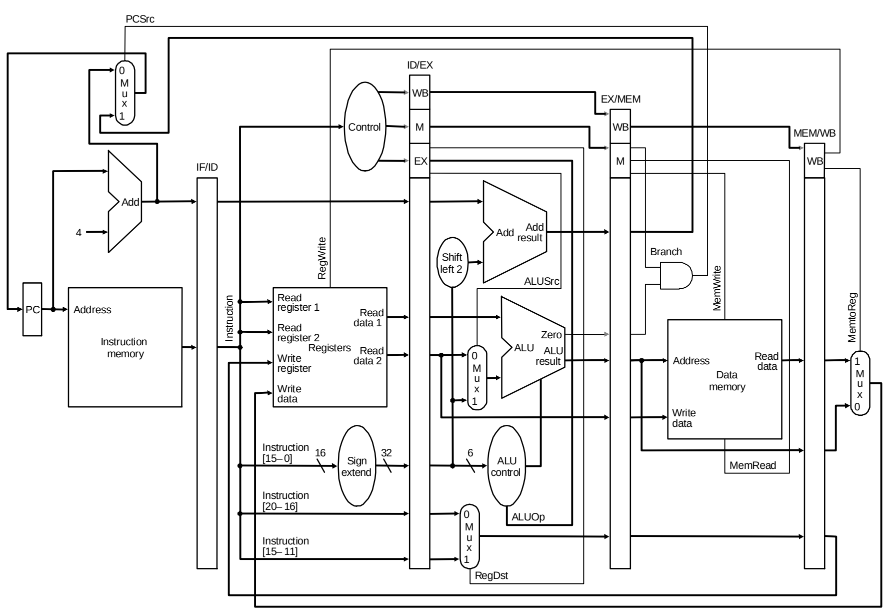
Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
SRC2	B	ALUSrcB = 00	Register B is the second ALU input.
	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshft	ALUSrcB = 11	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read		Read two registers using the rs and rt fields of the IR as the register numbers and putting the data into registers A and B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	MemRead, IorD = 0	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	MemRead, IorD = 1	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	MemWrite, IorD = 1	Write memory using the ALUOut as address, contents of B as the data.
PC write control	ALU	PCSource = 00 PCWrite	Write the output of the ALU into the PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	jump address	PCSource = 10, PCWrite	Write the PC with the jump address from the instruction.
Sequencing	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

Figuur 8: MIPS micro-instructieformaat

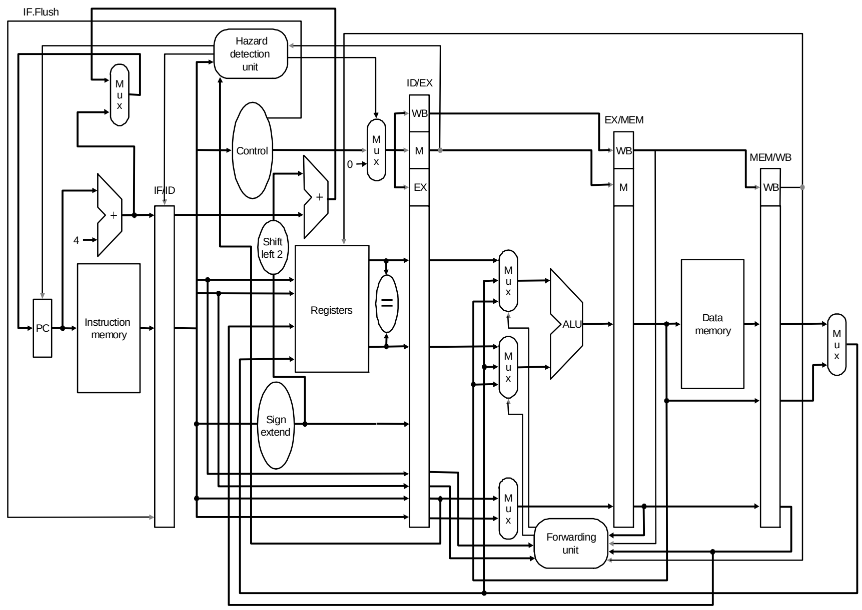
Stage 1 - IF: Instruction Fetch
Stage 2 - ID: Instruction Decode / register file read
Stage 3 - EX: Execute / address calculation
Stage 4 - MEM: Memory acces
Stage 5 - WB: Write Back



Figuur 9: MIPS pipeline



Figuur 10: pipelined implementatie van de MIPS processor zonder *forwarding* en *hazard detection* units



Figuur 11: pipelined datapad van de MIPS processor met *forwarding* en *hazard detection* units en gereduceerde *branch delay*