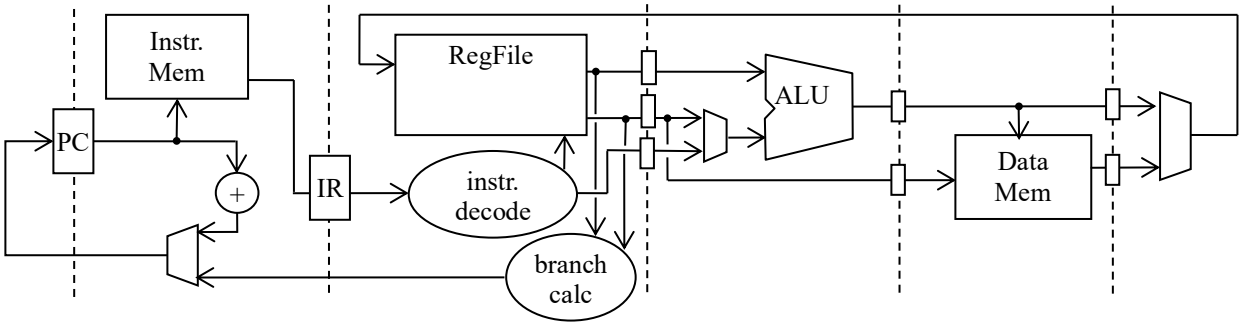
Simplified 5-stage pipelined MIPS datapath. Dashed lines indicate boundaries between pipeline stages.
Note: In this version, PC calculations for jumps and branches are performed during the second stage.



Units and conversions

1 KB = 1024 bytes = 2^{10} bytes ≈ one thousand bytes
1 MB = 1024 KB = 1024 x 1024 bytes = 2^{20} bytes ≈ one million bytes
1 GB = 1024 MB = 1024 x 1024 x 1024 bytes = 2^{30} bytes ≈ one billion bytes

1 GHz = 1,000,000,000 cycles/second = 1 billion cycles / second = 1 cycle / ns
1 MHz = 1,000,000 cycles/second = 1 million cycles / second = 1 cycle / us
1 KHz = 1,000 cycles/second = 1 thousand cycles / second = 1 cycle / ms

1 ps = 1 picosecond = one trillionth of a second
1 ns = 1 nanosecond = one billionth of a second = 1000 ps
1 us = 1 microsecond = one millionth of a second = 1000 ns
1 ms = 1 millisecond = one thousandth of a second = 1000 us

ASCII Table

Dec	Hex	Ascii
0	00	'\0'
1	01	SOH
2	02	STX
3	03	ETX
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	'\a' (bell)
8	08	'\b' (backspace)
9	09	'\t' (tab)
10	0A	'\n' (newline)
11	0B	'\v' (vtab)
12	0C	'\f' (formfeed)
13	0D	'\r'
14	0E	SO
15	0F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US

Dec	Hex	Ascii
32	20	' ' (space)
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	´
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Ascii
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Ascii
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

Num	Name	Description
r0	$zero	zero
r1	$at	assembler temp
r2	$v0	function return values
r3	$v1	
r4	$a0	function arguments (first 4, also space on stack)
r5	$a1	
r6	$a2	
r7	$a3	
r8	$t0	temps (caller save)
r9	$t1	
r10	$t2	
r11	$t3	
r12	$t4	
r13	$t5	
r14	$t6	
r15	$t7	

Num	Name	Description
r16	$s0	saved (callee save)
r17	$s1	
r18	$s2	
r19	$s3	
r20	$s4	
r21	$s5	
r22	$s6	
r23	$s7	
r24	$t8	more temps (caller save)
r25	$t9	
r26	$k0	reserved for OS kernel
r27	$k1	
r28	$gp	global pointer
r29	$sp	stack pointer
r30	$fp/s8	frame pointer
r31	$ra	return address

Num	Name	Description
hi		products & quotients
lo		
pc		program counter
ir		instruction register

Notation	Description
SE(x)	x, sign-extended
ZE(x)	x, zero-extended
LB(x)	low byte of x
LH(x)	low half of x
MEM[x]:n	n bytes at address x
hi:lo	hi and lo, joined
<<	left shift
>>	right shift, arithmetic
>>>	right shift, zero extend

Arithmetic Instructions

Instruction	*Operation*
addu $d, $s, $t	$d = $s + $t
addiu $t, $s, imm	$t = $s + SE(imm)
subu $d, $s, $t	$d = $s - $t
xor $d, $s, $t	$d = $s ^ $t
and $d, $s, $t	$d = $s & $t
andi $t, $s, imm	$t = $s & ZE(imm)
div $s, $t	lo = $s / $t; hi = $s % $t
divu $s, $t	lo = $s / $t; hi = $s % $t
mult $s, $t	hi:lo = $s * $t
multu $s, $t	hi:lo = $s * $t
nor $d, $s, $t	$d = ~($s | $t)
or $d, $s, $t	$d = $s | $t
ori $t, $s, imm	$t = $s | ZE(imm)
sll $d, $t, amt	$d = $t << amt
sllv $d, $t, $s	$d = $t << $s
sra $d, $t, amt	$d = $t >> amt
srav $d, $t, $s	$d = $t >> $s
srl $d, $t, amt	$d = $t >>> amt
srlv $d, $t, $s	$d = $t >>> $s
lui $d, imm	$d = imm << 16

Special Register Instructions

mfhi $d	$d = hi
mflo $d	$d = lo

Pseudo-Instructions

li $d, imm	$d = imm
la $d, label	$d = addr of label
move $d, $t	$d = $t
div $d, $s, $t	$d = $s / $t
mul $d, $s, $t	$d = $s * $t
mod $d, $s, $t	$d = $s % $t
nop	no operation

Comparison Instructions

Instruction	*Operation*
slt $d, $s, $t	$d = ($s < $t)
sltu $d, $s, $t	$d = ($s < $t)
slti $t, $s, imm	$t = ($s < SE(imm))
sltiu $t, $s, imm	$t = ($s < ZE(imm))

Note: Comparisons set destination register to 1 (true) or 0 (false).

Branch Instructions & Pseudo-Instructions

beq $s, $t, label	if ($s==$t) pc+=4+SE(imm)<<2
bne $s, $t, label	if ($s!=$t) pc+=4+SE(imm)<<2
bgt $s, $t, label	if ($s>$t) pc+=4+SE(imm)<<2
blt $s, $t, label	if ($s<$t) pc+=4+SE(imm)<<2
bge $s, $t, label	if ($s>=$t) pc+=4+SE(imm)<<2
ble $s, $t, label	if ($s<=$t) pc+=4+SE(imm)<<2
bgtz $s, label	if ($s>0) pc+=4+SE(imm)<<2
bltz $s, label	if ($s<0) pc+=4+SE(imm)<<2
bgez $s, label	if ($s>=0) pc+=4+SE(imm)<<2
blez $s, label	if ($s<=0) pc+=4+SE(imm)<<2

Jump Instructions

j label	pc = SE(imm) << 2
jal label	$31 = pc; pc = SE(imm)<<2
jalr $s	$31 = pc; pc = $s
jr $s	pc = $s

Note: Jumps with an immediate retain the top 4 bits of pc+4.

Load/Store Instructions

lb $t, imm($s)	$t = SE(MEM[$s + SE(imm)]:1)
lbu $t, imm($s)	$t = ZE(MEM[$s + SE(imm)]:1)
lh $t, imm($s)	$t = SE(MEM[$s + SE(imm)]:2)
lhu $t, imm($s)	$t = ZE(MEM[$s + SE(imm)]:2)
lw $t, imm($s)	$t = MEM[$s + SE(imm)]:4
sb $t, imm($s)	MEM[$s + SE(imm)]:1 = LB($t)
sh $t, imm($s)	MEM[$s + SE(imm)]:2 = LH($t)
sw $t, imm($s)	MEM[$s + SE(imm)]:4 = $t