Op. Codes

Branches

BAL – Branch and Link

0 0 0 0 0 1	0 0 0 0 0	1 0 0 0 1	16'b IMM

BEQ – Branch on Equal

0 0 0 0 1 1	5'b rs	5'b rt	16'b IMM

BGEZ – Branch on Greater or Equal to Zero

0 0 0 0 0 1	5'b rs	0 0 0 1 0	16'b IMM

BGEZAL – Branch on Greater or Equal to Zero and Link

0 0 0 0 0 1	5'b rs	1 0 0 1 0	16'b IMM

BGTZ – Branch on Greater Than Zero

0 0 0 0 0 1	5'b rs	0 0 0 1 1	16'b IMM

BGTZAL – Branch on Greater Than Zero and Link

0 0 0 0 0 1	5'b rs	1 0 0 1 1	16'b IMM

BLTZ – Branch on Less Than Zero

0 0 0 0 0 1	5'b rs	0 0 1 0 0	16'b IMM

BLTZAL – Branch on Less Than Zero and Link

0 0 0 0 0 1	5'b rs	1 0 1 0 0	16'b IMM

BLEZ – Branch on Less Than or Equal to Zero

0 0 0 0 0 1	5'b rs	0 0 1 0 1	16'b IMM

BLEZAL – Branch on Less Than or Equal to Zero and Link

0 0 0 0 0 1	5'b rs	1 0 1 0 1	16'b IMM

These branches test the rs register's value against the instruction condition. If true, the PC will be modified using the equation *PC=PC+IMM<<2*. If it evaluates false, *PC=PC+4*. Links are saved to register 31.

Jumps

J – Jump

0 0 0 1 0 1	26'b IMM

JAL – Jump and Link

0 0 0 1 1 1	26'b IMM

JALR – Jump and Link from Register

0 0 0 0 0 1	5'b rs	1 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

JR – Jump from Register

0 0 0 0 0 1	5'b rs	0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

SJAL – Special Jump and Link

0 0 1 1 1 1	26'b IMM

These Jumps are unconditional GOTOs. The equation *PC={PC[31:28], IMM<<2}* is used to evaluate the new PC. The equation *PC=rs'Data* is used for JALR and JR. Links are saved to register 31. Special Link is saved to register 1.

Arithmetic/Logical Reg

ADD

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 1 0 0 0

ADDU – ADD Unsigned

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 1 0 0 1

AND – Bitwise AND

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 0 0 0 0 0

DIV – Divide

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 0 1 0 0

DIVU – Divide Unsigned

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 0 1 0 1

MOD – Modulo

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 0 1 1 0

MODU – Modulo Unsigned

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 0 1 1 1

MUL – Multiply

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 0 0 1 1 0

MULU – Multiply Unsigned

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 0 0 1 1 1

NAND – Bitwise NAND

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 0 0 0 0 1

NOR – Bitwise NOR

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 1 0 0 0 0

OR – Bitwise OR

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 1 0 0 0 0

SUB – Subtraction

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 1 1 0 0

SUBU – Subtract Unsigned

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 1 1 0 1

SLL – Shift Left Logical

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 0 0 1 0 0

SLT – Store Less Than

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 0 1 1 0 0

Logical less than. *rd=rs<rt.*

SRA – Shift Right Arithmetic

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 0 0 1 1

SRL – Shift Right Logical

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	0 0 0 0 1 0

SLTU – Store Less Than Unsigned

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 0 1 1 0 1

Logical less than for unsigned values. *rd=rs<rt.*

XOR

0 0 0 0 0 0	5'b rs	5'b rt	5'b rd	0 0 0 0 0	1 1 1 0 0 0

These instructions are run through the ALU. The equation format is *rd=rs ? rt*, where *?* is the instruction's defined operation. For example, *rd = rs OR rt*. Regular ADD and SUB throw overflow exceptions.

Arithmetic/Logical Immediate

ADDI

1 0 1 0 0 0	5'b rs	5'b rt	16'b IMM

ADDIU

1 0 1 0 0 1	5'b rs	5'b rt	16'b IMM

ANDI

1 1 0 0 0 0	5'b rs	5'b rt	16'b IMM

NORI

1 1 0 0 0 1	5'b rs	5'b rt	16'b IMM

ORI

1 1 0 0 1 0	5'b rs	5'b rt	16'b IMM

SLTI

1 0 1 1 0 0	5'b rs	5'b rt	16'b IMM

SLTIU

1 0 1 1 0 1	5'b rs	5'b rt	16'b IMM

SLLI

1 1 0 1 0 0	5'b rs	5'b rt	16'b IMM

SRAI

1 1 0 1 0 1	5'b rs	5'b rt	16'b IMM

SRLI

1 1 0 1 1 0	5'b rs	5'b rt	16'b IMM

SUBI

1 0 1 1 1 0	5'b rs	5'b rt	16'b IMM

SUBIU

1 0 1 1 1 1	5'b rs	5'b rt	16'b IMM

XORI

1 1 0 0 1 1	5'b rs	5'b rt	16'b IMM

These instructions are run through the ALU. The equation format is *rt=rs ? IMM*, where *?* is the instruction's defined operation and IMM is a 32bit extended value. For example, *rt = rs OR IMM*. Regular ADDI and SUBI throw overflow exceptions.

Memory Operations

LUI – Load Upper Immediate

0 1 1 0 0 1	0 0 0 0 0	5'b rt	16'b Offset

LUI uses the equation *rt=Offset<<16*.

LW – Load Word

0 1 0 0 0 1	5'b base	5'b rt	16'b Offset

rt=RAM[base'Data + Offset], where base is a register and Offset is a signed extended 16bit immediate.

SW – Store Word

0 1 0 0 1 1	5'b base	5'b rt	16'b Offset

RAM[base'Data + Offset]=rt, where base is a register and Offset is a signed extended 16bit immediate.

These two instructions access the RAM . Load Word reads a 32bit word from memory and Store Word saves a 32bit word into memory.

System Calls

BCPU – Branch CPU

0 0 1 1 0 0	5'b CPU	5'b Run	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Choose which CPU to activate and which register level it should run with. Also, saves usurped PC for the CPU that was just branched.

BCPUJ – Branch CPU Jump

0 0 1 1 0 1	26'b IMM

Should be used immediately before BCPU. Loads the BranchCPU with the desired target address for the CPU you will be activating. *PC=IMM<<2.*

BCPUJR – Branch CPU Jump from Register

0 0 1 1 1 1	5'b rs	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Should be used immediately before BCPU. Loads the BranchCPU with the desired target address for the CPU you will be activating. *PC=rs'Data.*

EXIT – Exit

0 0 1 0 0 1	26'b XXX

Stops the clock running the core indefinitely

SLEEP – Sleep

0 0 1 0 0 0	0 0 0 0 0	5'b DIV	16'b IMM

Stops the clock running the core for a specified amount of clock cycles.