1.5 Logical Operations

The following are the logical operations performed by the processor:

Logical Operations	MIPS
Shift left	sll
Shift right	srl
Bit by bit AND	and, andi
Bit by bit OR	or, ori
Bit by bit NOT	nor

The first class of such operations is called **shifts.** They move all the bits in a word to the left or right, filling the emptied bits with 0s.

0000 0000 0000 00000 000 0000 0000 0000 $1001_2 = 9_{10}$ After left shifting by four, the new value is 144.

0000 0000 0000 0000 0000 0000 0000 1001 $0000_2 = 144_{10}$

- **Left shift:** Left shifting by i bits is equivalent to multiplying the number by 2^i.
- **Right Shift**: Right shifting by i bits is equivalent to dividing the number by 2^i.
- **AND:** This is used in masking of bits.
- **OR:** It is a bit-by-bit operation that places a 1 in the result if either operand bit is a 1
- **NOT:** A logical bit-by-bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1.
- **NOR:** A logical bit-by-bit operation with two operands that calculates the NOT of the OR of the two operands.

Category	Instruction	Operation
AND	and $s1, $s2, $s3	S1=s2&s3
OR	or $s1, $s2, $s3	S1=s2|s3
NOR	nor $s1, $s2, $s3	S1=~(s2|s3)
NAND	nand $s1, $s2, $s3	S1=~(s2&s3)

AND immediate	andi $s1, $s2, 100	S1=s2&100
OR immediate	Ori $s1, $s2, $s3	S1=s2|100
Shift leftlogical	Sll $s1, $s2, 10	S1=s2<<10
Shift right logical	Srl $s1, $s2, 10	S1=s2>>10

Control Operations

Decision making and branching makes the computers more powerful.

Decision Making:

Decision making in MIPS assembly language includes two decision-making instructions

(conditional branches):

i) Branch if Equal (BEQ):

beq register1, register2, L1

In this instruction, the go to the statement labeled L1 if the value in register1 is equal to the value in register2.

i) Branch if not Equal (BNE): bne register1, register2, L1

In this instruction, the go to the statement labeled L1 if the value in register1 does not equal the value in register2.

Conditional branch is an instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison.

Example:

Consider the following statement,

if (i == j) f = g + h; else f = g – h;

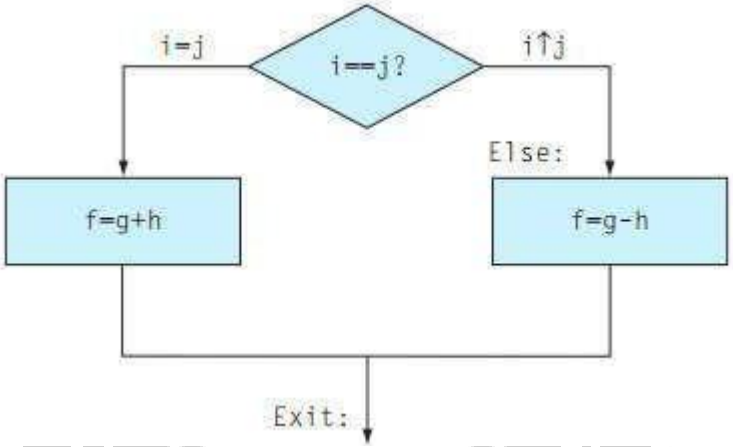

Fig 1 Flowchart for if (i == j) f = g + h; else f = g – h;

The instruction first compares for equality, using beq. In general, the code will be more efficient if we test for the opposite condition to branch over the code that performs the subsequent then part of if (the label Else is defined below):

bne $s3,$s4,Else # go to Else if i =j

The next assignment statement performs a single operation, and if all the operands are allocated to registers, it is just one instruction:

add $s0,$s1,$s2 # f = g + h (skipped if i =j)

This instruction says that the processor always follows the branch. To distinguish between conditional and unconditional branches, the MIPS name for this type of instruction is jump, abbreviated as j (the label Exit is defined below). j Exit # go to Exit

The assignment statement in the else portion of if statement can again be compiled into a single instruction. We just need to append the label Else to this instruction. We also show the label Exit that is after this instruction, showing the end of the if-then-else compiled code:

Else: sub $s0, $ s1, $s2 # f = g – h

(skipped if i = j) Exit:

Compilers create branches and labels wherever necessary for maintaining flow of the program. Also, the assembler calculates the addresses and relieves the compiler and the assembly language programmer.

Looping:

When a set of statements has to be executed more number of times, looping statements are used.

Example:

while (save[i] == k) i += 1;

i and k correspond to registers $s3 and $s5 and the base of the array save is in $s6.

The MIPS instructions are:

- The first step is to load save[i] into a temporary register. This operation needs an address. Multiply the index i by 4 and add i to the base of array to obtain the address.
- Add the label Loop to it to branch back to that instruction at the end of the loop: *Loop: sll $t1,$s3,2 # Temp reg $t1 = 4 * i*
- To get the address of save[i] , add $t1 and the base of save in $s6:
  add $t1,$t1,$s6 # $t1 = address of save[i]
- Use that address to load save[i] into a temporary register:
  lw $t0,0($t1) # Temp reg $t0 = save[i]
- The next instruction performs the loop test, exiting if save[i] k: *bne $t0,$s5, Exit # go to Exit if save[i]* ? *k*
- The next instruction adds 1 to i :
  add $s3,$s3,1 # i = i + 1
- The end of the loop branches back to the while test at the top of the loop. Add the Exit label after it:
  j Loop # go to Loop
  Exit:

Grouping on instructions that makes compiling easy is through partitioning

A sequence of instructions without branches except possibly at the end and without branch targets or branch labels except possibly at the beginning are called basic blocks.

the assembly language instructions into basic blocks.

Case / Switch Statements

These statements allow the programmers to select one among the many options. The simple way is to implement switch is through a sequence of conditional tests using a chain of if- then-else statements. The alternatives are encoded in **jump**

A table of addresses of alternative instruction sequences is maintained in jump address table.

address table. The program needs only to index into the table and then jump to the appropriate sequence.

The jump table is an array of words containing addresses that correspond to labels in the code. MIPS include a **jump register instruction** (jr), to support the unconditional jump to the address specified in a register. The program loads the appropriate entry from the jump table into a register, and then it jumps to the proper address using a jump register.

Category	**Instruction**	**Operation**
Conditional Branch	Beq $s1,$s2,L	If (s1==s2) then goto L (Branch if equal)
	Bne $s1, $s2,L	If (s1 s2) then goto L (Branch if not equal)
	Slt $s1, $s3, $s3	If (s2<s3) set s1=1Else s1=0(set on less than)
	Slt$s1, $s2, 100	If (s2<100) set s1=1Else s1=0(set on less than Immediate)
Un Conditional Branch	J L	Go to L(Jump to target address L)