**MIPS Architecture**

Last lecture was about what happens inside the processor when a piece of code is executed

* Execution cycle
* Instruction set architecture
* MIPS Architecture
* Types of MIPS Instruction

Today’s lecture more types of MIPS Instruction.

**Immediate Arithmetic instruction**

* An immediate instruction uses a constant number as one of the inputs instead of a register operand
* To use the Add instruction we have seen so far, we would have to load a constant from memory to use one example
  + So our code for a = b + c; will look like:

//add the contents of r2 and 20 and place the result in r1

// $r1= $r2 + 20

addi $r1, $r2,20

**Rmks**

- By including constants inside arithmetic instructions, operations are much faster and use less energy than if constants were loaded from memory

**Multiplication and Division**

* Add and Sub are performed on 32 bit numbers held in the general purpose registers (32bit each)
* The result is a 32bit number itself
* Multiplication and Division may generate results that are larger than 32bits
* The MIPS architecture provides 2 special 32bit registers that are the destination for the multiplication and division instructions. These registers are called
  + hi and lo
    - hi holds the higher 32bits of the results
    - lo holds the lower 32bits of the results
* Special instructions are also provided to move data from these registers into the special purpose ones ($r0 t0 $r31)
  + mfhi register & mflo register to move from hi, lo to another register Multiplication Example Program
  + in c : a = b \* c
  + In MIPS: let b be $r2, let c be $r3 and a be $r1 and $s1(since it may be up to 64 bits)

mult $r2, $r3

mfhi $r1

mflo $r2

/ / b \*c

/ / upper half of product into $r1

/ / lower half of product into $r2

**Syntax of division**

* Div register1, register2
* divides 32bit register1 by 32bit register2
* Puts remainder of division in hi, quotient in lo Division Example Program

/ / lo = $r2/$r3 hi = $r2 mod $r3

/ / $r1= hi

/ / $r2 = lo

* div $r2, $r3

mfhi $r1

mflo $r2

**Shift instructions**

* move all the bits in a word to the left or right, filling the emptied bits with 0s
* They belong to a class of operations called logical operations
* The actual name of the MIPS shift instructions are called
  + Shift left logical sll
  + Shift right logical srl

**Example**

* If register $r1 contained
* 0000 0000 0000 0000 0000 0000 0000 1001 = 9
* and the instruction to shift left by 4 was executed the new value would be
* 0000 0000 0000 0000 0000 0000 1001 0000 = 144
* Code
* Sll s$r2 , $r1, 4 //reg $r2 = reg $r1 << 4 bits

**Other MIPS logical instructions**

* **And**
* performs a logical bit-by-bit operation with 2 operands that calculates a 1 only if there is a 1 in both operands
* and $r1, $r2, $r3 // reg $r1 = reg $r2 & reg $r3
* **Or**
* A logical bit-by-bit operation with 2 operands that calculates a 1 if there is a 1 in either operand
* or $r1, $r2, $r3 // reg $r1 = reg $r2 | reg $r3
* Constants are useful in AND and OR logical operations as well as arithmetic operations, so MIPS also provides the instructions and immediate andi or immediate ori
* Other instructions include Not, Nor and Xor

**Branch and jump instructions**

**Branch Instructions**

* Branch instructions allow programmers to design decision making ability into a program
* For that reason they can be referred to as program control instructions since they support the ability for a program to determine when to change operation.
* In general, a branch performs a comparison. If the comparison is successful, the next instruction executed is at another point in the program
* If the desired comparison is not achieved the program simply executes the instruction following the branch
* The following list is not complete, but composes a relatively useful group of the branch instruction subset.

beq – branch on equal

beq $r1, $r2, label //if the content of r1= r2, branch to label; otherwise execute next instruction bgez- branch on greater than or equal zero

bgez $r1, label //if the content of r1 is >= 0 branch to label; otherwise execute next instruction

bgtz - branch on greater than zero

bgtz $r1, label //if the content of r1> 0, branch

nextinstruction

* Other instructions include :

; otherwise execute

blez - branch on less than or equal to zero

bltz - branch on less than zero

1. bne- branch on not equal
2. beqz- branch on equal zero
3. bge- branch on greater than or equal
4. bgt- branch on greater than
5. ble- branch on less than or equal
6. blt- branch on less than
7. bnez- branch on not equal zero

An Example: simple branch consider an if statement fragment

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

* The first expression compares for equality, so it would seem that we would want 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 the if*

bne $r1,$r2,Else

// go to Else if i is not equal to j

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

add $r3,$r4,$r45 // f = g + h (skipped if i is not equal to j)

* We now need to go to the end of the *if statement*

j Exit // go to Exit

* This example introduces another kind of branch, often called an *unconditional branch. 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***

Jump instructions

* This transfer is unconditional
  + There is no option on jump instructions; a jump is always to a labeled location
  + J loop // the next instruction executed is the one labelled loop

There are different variations of jump instruction, some may be used for subroutine or procedure calls

* There are different variations of jump instruction, some may be used for subroutine or procedure calls
* procedure is a stored subroutine that performs a specific task based on the parameters with which it is provided
* MIPS assembly language includes an instruction just for the procedures it jumps to an address and simultaneously saves the address of the following instruction in a special register $ra, The jump-and-link instruction **jal**

**Example**

jal loop // jump to loop and save the content of PC in $ra

* The link portion of the name means that an address or link is formed that points to the calling site to allow the procedure to return to the proper address.
* This “link,”stored in register $ra, is called the **return address.** The return address is needed because the same procedure could be called from several parts of the program
* MIPS use a *jump register instruction* (jr), meaning an unconditional jump to the address specified in a register:
* jr $ra // jump to the address specified in $ra
* Consider a simple for loop
* for (i>0; i<=x; i++){ y = y + i;
* }
* // More stuff
* After doing y = y + i; need to unconditionally jump back to the start of the loop
* - j &jal,jr does exactly that

**Set Instructions**

* The set instructions are used in decision-making functions( often with branch instructions) within a program
* These instructions instruction that compares two registers and sets a third register to 1 if the first is less than the second; otherwise, it is set to 0

– Example

A common use of if…else clauses is to set flags:

if(x < y) {

flag = 1;

}

else {

flag = 0;

}

Easy to do using:

* slt $r1, $r2, $r3 : “set-on-less-than”
* Sets r1=1 if r2<r3, else sets r1=0

// Load x into r1

// Load y into r2

// r3=1 if r1<r2, else r3=0

lw $r1, &x lw $r2, &y

slt $r3, $r1, $r2

**MIPS instruction coding**

MIPS instructions are classified into four groups according to their coding formats

* **R-type**: this group contains all instructions that do not require an immediate value, target offset, memory address displacement or memory address to specify an operand. This includes arithmetic and logic with all operands in registers, shift instructions and register direct jump instructions (jalr, jr)
* **I-type** this group includes instructions with an immediate operand, branch instructions, load and store. In MIPS architecture all memory accesses are handled by the main processor , so coprocessor load and store instructions are included in this group
* **J-type** this group consist of the two direct jump instructions ( j and jal). These instructions require a memory address to specify their operand
* **Coprocessor instructions** MIPS processors all have two standard coprocessors CP0 and CP1. CP0 processes various kinds of program exceptions while CP1 is a floating point processor