Example MIPS Snippets

Loading a word from memory

- lw $t0, testword1 # Load from a data symbol

[00400024] 3c011001 **lui $1, 4097** *; 14: lw $t0, test1 # The first number*
[00400028] 8c280000 **lw $8, 0($1)**

- Notice that the assembler uses two bare metal instructions, using temporary register $at ($1)

Loading relative to a register

- lw $t8, 0($t1) # $t1 contains source address

  # Destination is $t8

  # (immediate) offset is 0

- lw $t8, 0x240($t4)

- How do we get an address in $t4?

Data Area

- .data
- data_area: .space   4096 # Reserve 4096 bytes

```
la $t4, data_area  # place data address in $t4
```

- Now we can load relative to $t4,

```
lw $t0, 4($t4)
```

Computing Effective Address

- Take specified register on right
- Add the immediate offset value
- Immediate offset is signed
- Sign-extend to 32-bits
- Provides an offset of -32768 to +32767 bytes
- lw and sw MUST use effective addresses that are on *word* boundaries!!! (divisible by 4)

Storing Data to Memory

- Compute Effective Address in same way as lw (load) instruction
- Store content of data on left to memory address on right.
- Example:

sw $t0, 0x100($t4)

Testing Bit Values

- The ‘and’ instruction can be used to see if certain bits are on. If ‘anded’ with a single bit value it will test that particular bit.
- To test multiple bits, and with a “mask” that has those bits on; if the result is non-zero, then one of the bits is 1.
- If in low-order 16 bits, can use ‘andi’ to and with a 16-bit mask.
- Example:

```
andi $t5, $t0, 0x0020 # test bit 5
```

Testing the high 16 bits

- If in high bits, have to move the mask to the high 16 bits, then perform ‘and’ operation:
- Example:

```
lui $t5, 0x4000   # move mask to high 16
```

- This produces 0x40000000 in $t5. Now we can ‘and’ to get the result, and branch if not zero

```
and $t5, $t0, $t5 # result of and to $t5
beq $t5, $zero, notset  # branch if not set
```

Clearing Specific Bits

- Bits in a register can be cleared by “anding” with a mask, where a 0 value in the mask causes the resulting bit to be cleared.
- Example:

  andi $5, $5, 0xFFEF # Clear bit 4

  Note: this also clears bits 16-31 !!

Setting Specific Bits

- This is performed by “oring” with a mask
- Example:

```
or $t0, $t0, 0x0044 # Turn on bits 2 and 6
```

Note: to set or clear bits in the high portion of a register, you must load the mask into bits 16-31 of a register, using lui, then or with the register.

Using xor

- xor of anything with itself produces 0

    xor $t0, $t0, $t0 # clears register $t0

- xor of 1 with anything inverts the value
- Example: flip bit numbers 0 through 3 in $t0

    xori $t0, $t0, 0b00001111 # bits to flip

Calling a Function

- Save the $ra and $fp on entry

This uses the stack. To PUSH on the stack, first subtract 4 from the $sp. Then store the data.

Example:

```
addiu $sp, $sp, -4
sw $ra, 0($sp)     # store on the stack
```

Popping From the Stack

- First, obtain the data pointed to by $sp

```
lw      $ra, 0($sp)
```

- Then advance the stack pointer

```
addiu $sp, $sp, 4
```

Example simple_add_stack

```
addemup: addiu $sp, $sp, -4 # push, reserve 4 bytes
         sw     $ra, 0($sp)       # save $ra
         addiu  $sp, $sp, -4      # another push
         sw     $fp, 0($sp)       # save “frame pointer”
         or     $fp, $zero, $sp   # move copy sp to fp

         addu   $t0, $t0, $t1     # add the two integers together

         lw     $fp, 0($sp)       # restore frame pointer
         addiu  $sp, $sp, 4       # pop it from the stack

         lw     $ra, 0($sp)  # restore return address
         addiu  $sp, $sp, 4       # pop it

         jr     $ra               # and just return
```