MIPS Architecture and Programming

Chapter 18

What is MIPS?

- - RISC architecture
- - Microprocessor without Interlocked Pipeline Stages
- - Simple and fast
- - Used in academics and embedded systems

Learning MIPS Assembly

Registers

Symbolic Name	Number	Usage
zero	0	Constant 0.
at	1	Reserved for the assembler.
v0 - v1	2 - 3	Result Registers.
a0 - a3	4 - 7	Argument Registers $1 \cdots 4$.
t0 - t9	8 - 15, 24 - 25	Temporary Registers $0 \cdots 9$.
s0 - s7	16 - 23	Saved Registers $0 \cdots 7$.
k0 - k1	26 - 27	Kernel Registers $0 \cdots 1$.
gp	28	Global Data Pointer.
sp	29	Stack Pointer.
fp	30	Frame Pointer.
ra	31	Return Address.

MIPS Register Set

- 32 general-purpose registers
- $zero: always 0
- $t0-$t9: temporary
- $s0-$s7: saved
- $a0-$a3: arguments
- $v0, $v1: return values
- $sp: stack pointer
- $ra: return address

MIPS Instruction Format

- - All instructions are 32-bit
- - R-type: Arithmetic/logical
- - I-type: Memory/branch
- - J-type: Jump

- R: op rs rt rd shamt funct
- I: op rs rt immediate
- J: op address

Arithmetic & Logical Instructions (R-type)

- add $t0, $t1, $t2 # $t0 = $t1 + $t2
- sub $t0, $t1, $t2 # $t0 = $t1 - $t2
- and $t0, $t1, $t2 # $t0 = $t1 & $t2
- or $t0, $t1, $t2 # $t0 = $t1 | $t2
- slt $t0, $t1, $t2 # $t0 = ($t1 < $t2)

Immediate and Memory Instructions (I-type)

- addi $t0, $t1, 10 # $t0 = $t1 + 10
- andi $t0, $t1, 0xFF # $t0 = $t1 & 0xFF
- lw $t0, 0($t1) # Load word
- sw $t0, 0($t1) # Store word

Branch and Jump Instructions

- beq $t0, $t1, label # if equal
- bne $t0, $t1, label # if not equal
- j label # jump
- jal function # jump and link
- jr $ra # return

Example – Add Two Numbers

- addi $t1, $zero, 5
- addi $t2, $zero, 10
- add $t0, $t1, $t2

Example – Load and Store

- addi $t0, $zero, 20
- sw $t0, 0($sp)
- lw $t1, 0($sp)

Example – Loop (Sum 1 to 10)

- addi $t0, $zero, 1
- addi $t1, $zero, 10
- addi $t2, $zero, 0
- loop:
- add $t2, $t2, $t0
- addi $t0, $t0, 1
- ble $t0, $t1, loop

Example – Function Call

- main:
  - li $a0, 5
  - jal square
  - move $t0, $v0

- square:
  - mul $v0, $a0, $a0
  - jr $ra

Summary

- - Registers: $t, $s, $a, $v
- - Instruction types: R, I, J
- - Memory: lw, sw
- - Control: beq, bne, j, jal, jr
- - Function: $a0-$a3, $v0, $ra

Learning MIPS Assembly

Instructions

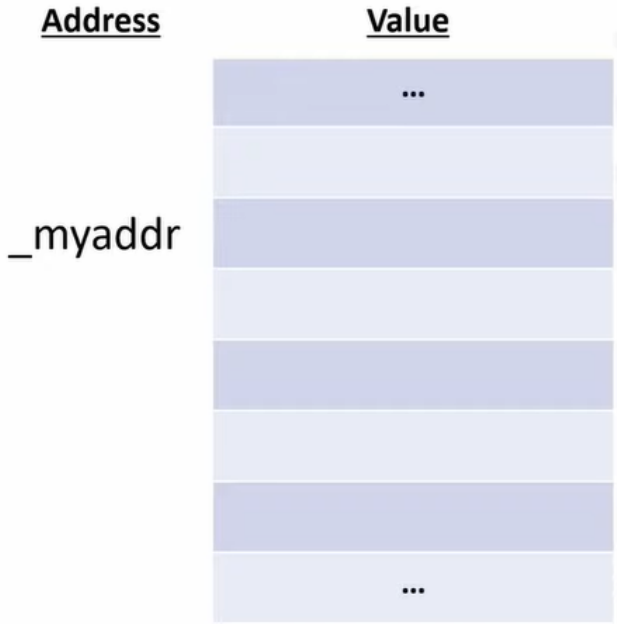
li $dst, #immediate – load integer into a register
la $dst, $symbol – load an address into a register
lw $dst, offset($src) – load a word from an address

sw $reg, offset($dst) – store a word at an address

Learning MIPS Assembly

Instructions

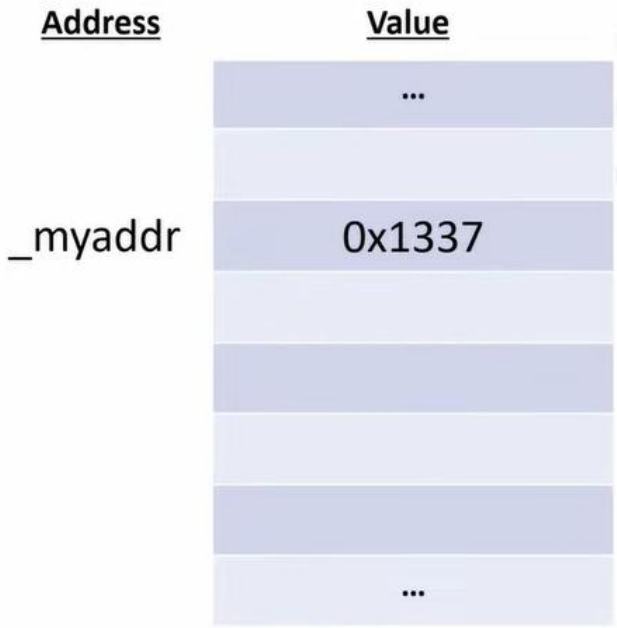
```
li $t1, 0x1337
la $t0, _myaddr
sw $t1, 0($t0)
```



Learning MIPS Assembly

Instructions

```
li $t1, 0x1337
la $t0, _myaddr
sw $t1, 0($t0)
```



Writing MIPS Code

add = addition
sub = subtraction
sll = shift left
srl = shift right
beq = branch if equal to
bne = branch if not equal to
j = jump
lw = load
sw = store
addi = add immediate
slt = set if less than

Writing MIPS Code

Convert the following to MIPS Code:

1. f = (g+h) - (i+j) where f - j is in $s0 - $s4

2. g = h + A[8] where g is in $s1, h is in $s2 and base address of A is in $s3

3. A[12] = h + A[8] where h is in $s2, base address of A is in $s3

4. while (save[i] == k) i+=1 where i is in $s3 base address of save is in $s6, k is in $s5

5. if (A[6] == C[6])
   f[8] = g[8] +C[5] +A[A[2]]
   else
   f[9] = g[5] + C[3]

   where base address of A is $s0,
   C is $s2, f is $s3 and g is $s1

Writing MIPS Code

f = (g+h) - (i+j) where f - j is in $s0 - $s4

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Writing MIPS Code

g = h + A[8] where g is in $s1, h is in $s2 and base address of A is in $s3

memory address = 4*offset +base address
memory address = 32 +$s3
= 32($s3)

lw $t0, 32($s3)
add $s1, $t0, $s2

Writing MIPS Code

A[12] = h + A[8] where h is in $s2, base address of A is in $s3

memory address = 32($s3)

```
lw $t0, 32($s3)
add $t1, $t0, $s2
sw $t1, 48($s3)
```