Module 4

MIPS Assembly Language;
Control Structures

Module Four

- MIPS Assembly Language; Control Structures - Part Two
- In this presentation, we are going to talk about :
- MIPS Assembly Language support for Subroutines

Overview

- Previously we talked about:
  - Process Flow
  - Control Structures
    - IF – THEN – ELSE
    - Loops
- Now: Subroutines

Subroutines

- Sub-programs
- Program code that is used multiple places in a program.
- Program code that is common and copied from a library.
- Main program jumps to the subroutine code, and subroutine jumps back to the main program.
- MIPS

  JAL - Jump and Link
- Accepts argument values
- Functions

  Subset of subroutines that return a result value.

Subroutines

- Main Program
  - Sets argument values
  - Jumps to the subroutine code
- Subroutine
  - Allocates memory space
  - Runs the procedure task
  - Sets the return values
  - Frees memory space
  - Jumps back to Main Program

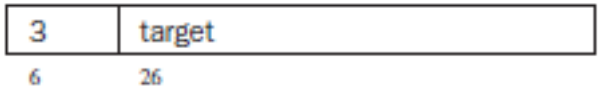
MIPS

- Call a subroutine
- Jump and Link instructions **JAL** routine-name
- Address of subroutine being called
- Return from subroutine
- Return Jump instructions **JR** $ra

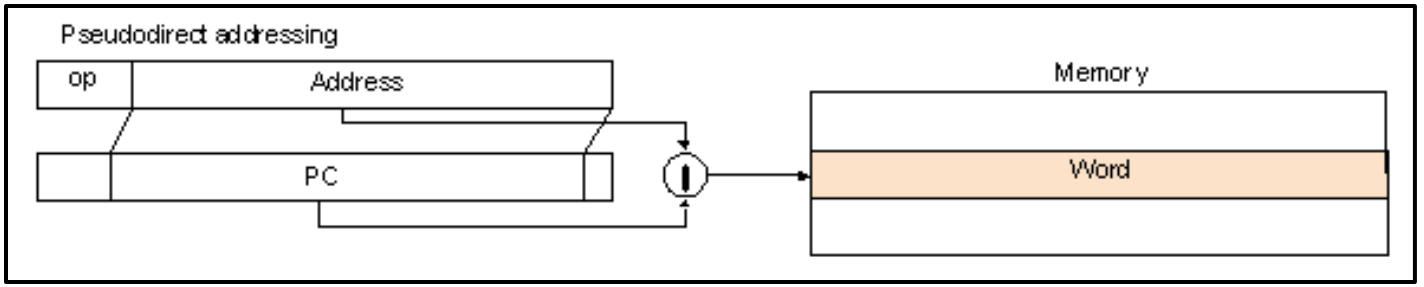
Jump and Link

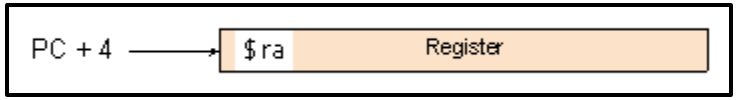
Jump and link

`jal target`



Unconditionally jump to the instruction at target. Save the address of the next instruction in register `$ra`.

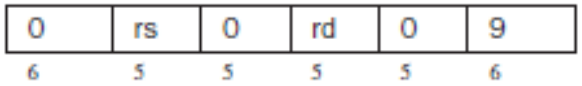




Jump Register

Jump and link register

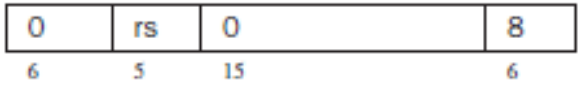
`jalr rs, rd`



Unconditionally jump to the instruction whose address is in register `rs`. Save the address of the next instruction in register `rd` (which defaults to 31).

Jump register

`jr rs`



Unconditionally jump to the instruction whose address is in register `rs`.

Arguments

- Argument values passed to the subroutine
- Several methods
- In registers

  As values - copy of value - very safe

  As addresses - address - can cause ‘side effects’
- In memory
- On the Stack

MIPS Registers

- General agreement of MIPS programmers
- $a0 - $a3 first four values input to the subroutine
  if more - then on Stack
- $v0 - $v1 returned result values
  if more - then on Stack
- $ra - the return address
- $t0 - $t9 registers may be used by subroutine
- $s0 - $s7 values must be saved if registers used
- $sp - Stack pointer

A MIPS example

- Can you figure out the code?

```
swap(int v[], int k);
{  int temp;

    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;

}
```

```
swap:
        muli $v0,  $a1, 4    ; k * 4
        add  $v0,  $a0, $v0  ; v + k

        lw   $t6, 0($v0)  ; v[k]
        lw   $t7, 4($v0)  ; v[k+1]
        sw   $t6, 4($v0)  ; v[k+1]
        sw   $t7, 0($v0)  ; v[k]

        jr   $ra
```

Another Method

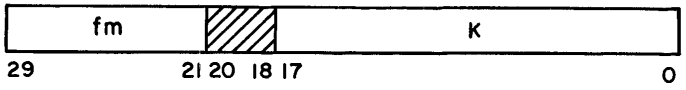
- The CDC 6500 was an early super computer. 1965
- It used direct addresses in the instructions.
- Some instructions were 30 bits.
- The word size is 60 bits.

- The subroutine call instruction is the Return Jump.

CDC 6500 Return Jump

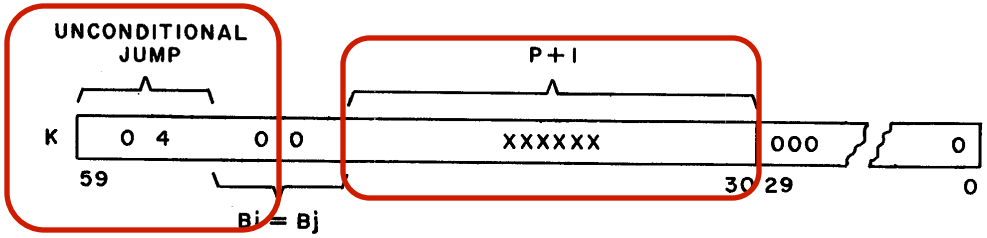
Branch

010 RJ K Return jump to K (30 Bits)



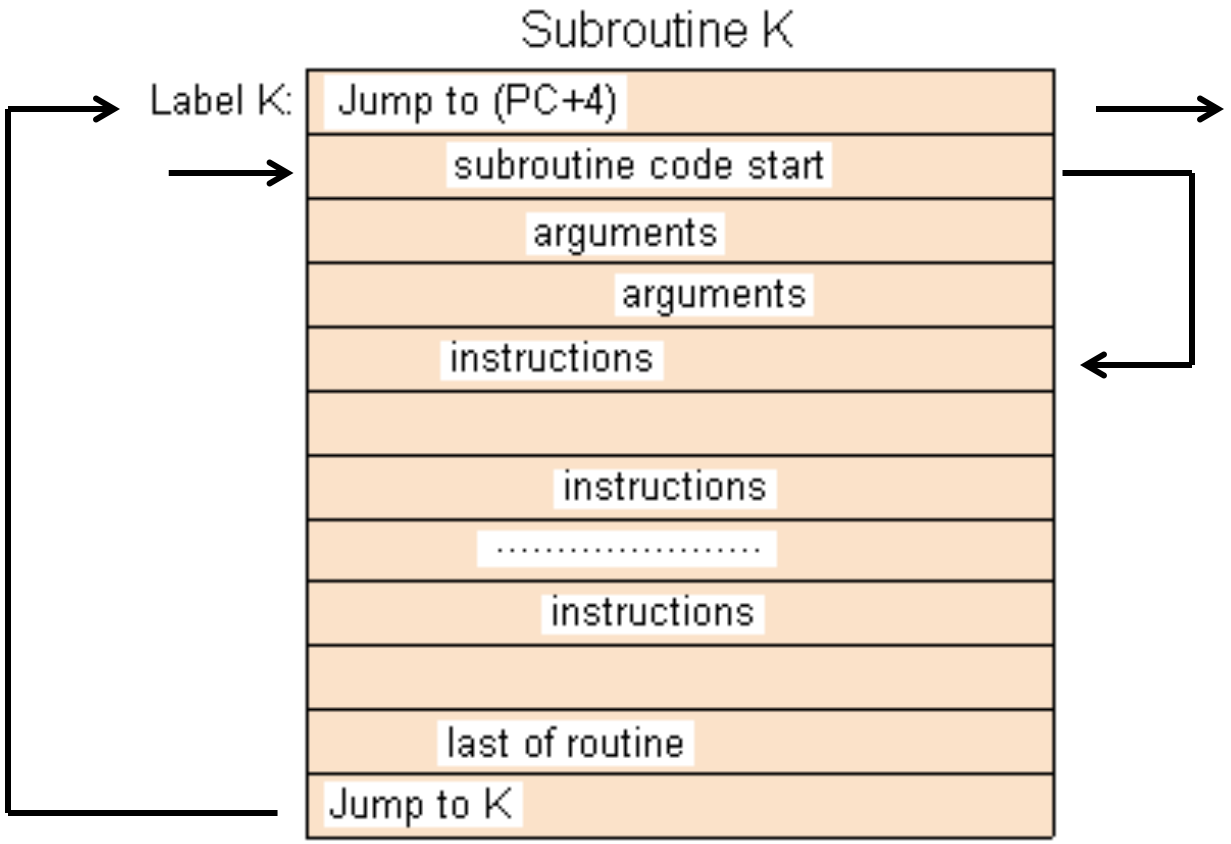
The instruction stores an 04 unconditional jump and the current address plus one [(P) + 1] in the upper half of address K, then branches to K + 1 for the next instruction. Note that this instruction is always out of the instruction stack, thus voiding the stack.

The octal word at K after the instruction appears as follows:



A jump to address K at the end of the branch routine returns the program to the original sequence.

Return Jump in action



Types of Subroutines

- Leaf - subroutine that processes and returns
- Nested - subroutines that call other subroutines before returning
  Need to save the return address value in $ra
- Recursive - subroutines that call themselves
  Can be direct or via additional subroutines
- Re-entrant - allocates stack and memory space each time called

Summary

- MIPS Assembly Language and control structures
  - Subroutines
- Next: Data structure support for Subroutines