More on Instruction Sets

(including some moron instruction sets)

The MIPS instructions of Chapter 3 are summarized and unified. Alternative instructions from other architectures are discussed. An additional example, append to a list, is presented.

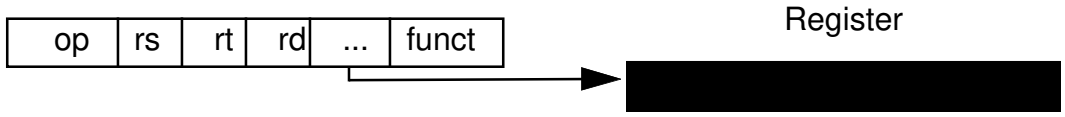
© 1997 LS All Rights Reserved

Addressing Modes

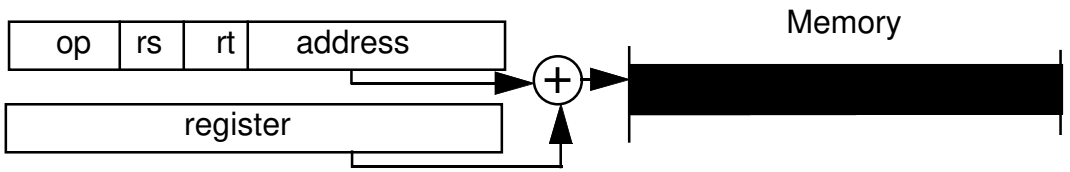
•The term "addressing modes" refers to the mechanisms computers use to refer to their operand values. MIPS has four addressing modes.

–Immediate addressing

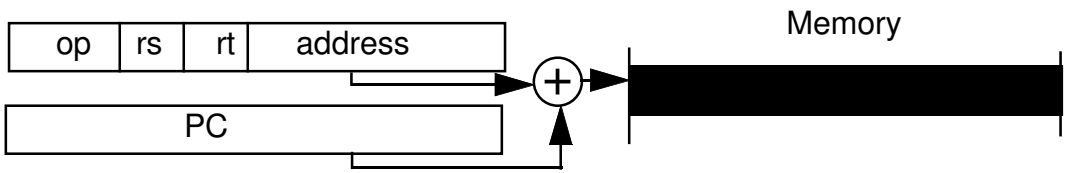
–Register addressing



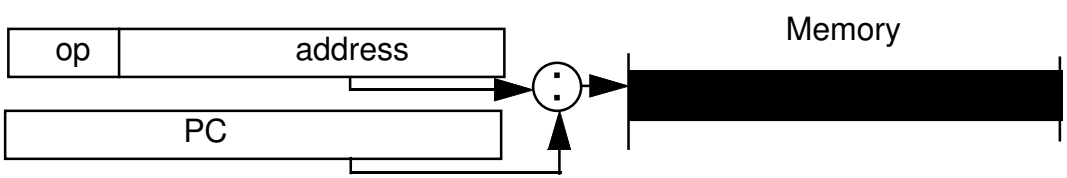
–Base/offset addressing



–PC-relative addressing



–Pseudodirect addressing



© 1997 LS All Rights Reserved

Non Address Modes

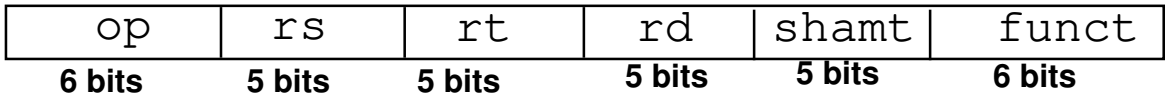
- Two addressing modes that the MIPS doesn't have:
  - Base/displacement/index -- the operand address is found by adding two named registers to a field of the instruction; base/offset with a second register.
  - Stack addressing -- operand registers are assumed to be the top two items on a stack, which are removed during the operation and replaced by the result.

© 1997 LS All Rights Reserved

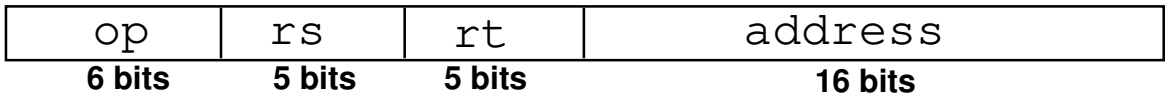
Instruction Formats

The format gives the structure of the bits of an instruction. The MIPS has three types:

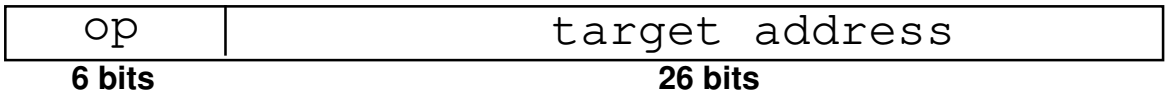
R-type



I-type



J-type



Instruction formats and addressing modes are related:

All Register addressing uses R-type instructions

Immediate addressing uses I-type

Base/offset addressing uses I-type

PC-relative addressing uses I-type and J-type instructions

© 1997 LS All Rights Reserved

Other Instructions

Autoincrement/autodecrement: **`lwad $8, A($16)  #BOGUS`**

```
lw     $8,A($16) # Fetch A[$16]
addi   $16,$16,-4 # Reduce $16 index for next loc
```

Memory resident operands: **`subm $17,$17,A($16) #BOGUS`**

```
lw  $8, A($16)    # Fetch A[$16]
sub $17, $17, $8  # $17 = $17 - A[$16]
```

Increment/compare and branch: **`icb $16,$17,Loop #BOGUS`**

```
addi   $16,$16,1 # Bump $16
slt $8, $16, $17  # Set unless loop limit reached
bne $8, $0, Loop  # Iterate if limit not reached
```

Save Registers: **`savreg 16,31  #BOGUS`**

```
sw  $16, 0($29)   # Save register  16
sw  $17, 4($29)   # Save register  17
sw  $18, 8($29)   # Save register  18
```

•••

© 1997 LS All Rights Reserved

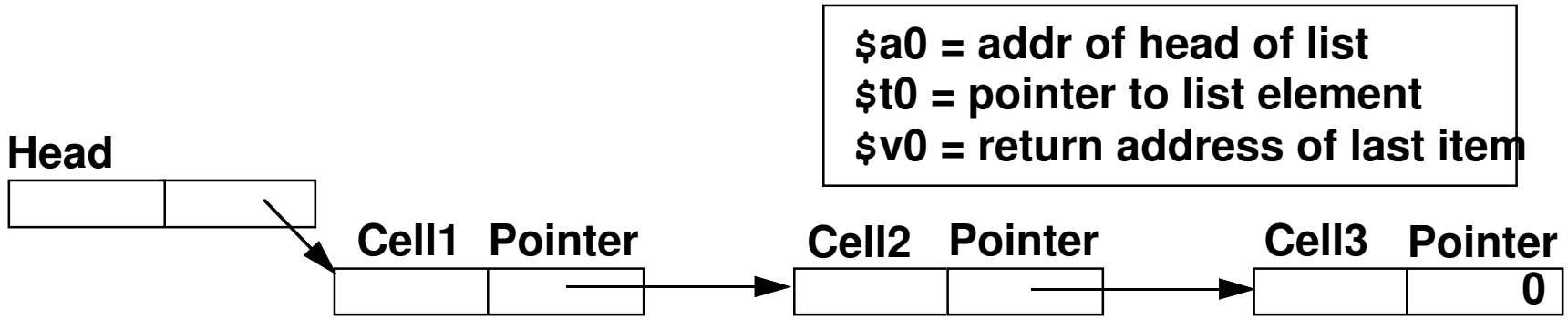
Limitations of the MIPS

- The instruction set for the MIPS has some shortcomings that require multiple instructions just to form an address and reference a value. For example:
  - –Address field is only 16 bits -- limits base range
  - –Load/Store can have only one index register -- prevents use of offset, base, and index.
- Could the 16 bit address field be increased?
- Could a second index register be added?

© 1995 LS All Rights Reserved

Example -- Appending

Postulate a forward linked list stored in memory:
Write a procedure that scans to find the last item in the list.
Use that procedure in another procedure to append to the list.

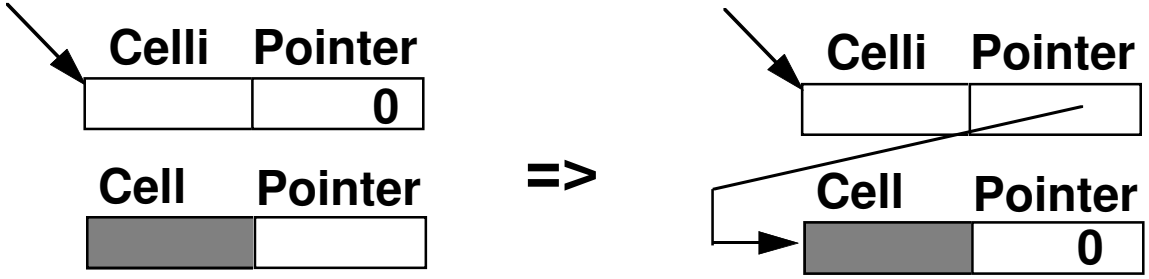



```
findend: add   $v0,$a0,$zero# Put head addr as return val
                            #   in case list is empty
gofwd:   lw    $t0,4($v0)    # Get addr of next item
         beq   $t0,$zero,done # If its nil, exit
         add   $v0,$t0,$zero # Move this as result register
                            #    as possible answer
         j     gofwd         # Keep looking
done:    jr    ra            # Return
```


© 1997 LS All Rights Reserved

Now Append

Given the address of the head of the list ($a0), the address of an item to be appended ($a1), and the address of the last item ($v0), append it.




```
append: addi  $sp,$sp, -4   # Make space to save return
        sw    $ra,0($sp)    # Save return address
        jal   findend       # Locate last item
        sw    $a1, 4($v0)   # Link new cell
        sw    $zero, 4($a1) # Set pointer of last to nil
        lw    $ra, 0($sp)   # Recover the return  pointer
        addi  $sp, $sp, 4   #  pop stack pointer
        jr    $ra           #  and return
```


© 1997 LS All Rights Reserved

Design Goals

•A principle of computing is that computations on complex data can be realized by a program using smaller computations operating on alternate data representations.
•Applying this principle repeatedly implies computers can have a few very simple "hardwired" operations for binary values.
•Why does the MIPS instruction set look like this?
•Design principles:

-Smaller is faster

-Simplicity favors regularity

-Good design demands compromise

-Make the common case fast

			Frequency	
Category	**E.G.**	**Prog. Lang. Correspondence**	**GCC**	**Spice**
Arithmetic	add	Expressions, assignments	48%	50%
Data Xfer	lw	Data structure maniipulation	33%	41%
Branches	bne	If statements, looping	17%	8%
Jump	j, jal	Procedures/returns	2%	1%

© 1997 LS All Rights Reserved