New **MIPS** instructions

Instruction	Syntax	Example
sll/srl	sll dest, src0, immediate	sll $s0, $s1, 5
slt/sltu	slt dest, src0, src1	slt $s0, $s1, $s2
slti/sltiu	slti dest, src0, immediate	slti $s0, $s1, 10
lb/lbu	lb dest, offset(base addr)	lb $t0, 4($s0)
sb	sb src, offset(base addr)	sb $t0, 4($s0)
j/jal	jal jumpAddr	jal funcName
jr	jr dest	jr $ra

MIPS registers

Register Number	Register Name	Register Use
$0	$zero	The "zero-constant"
$1	$at	*Used by the assembler*
$2-$3	$v0-$v1	Return values
$4-$7	$a0-$a3	Function arguments
$8-$15	$t0-$t7	Temporary registers
$16-$23	$s0-$s7	Saved registers
$24-$25	$t8-$t9	Temporary registers
$26-$27	$k0-$k1	*Used by the kernel*
$28	$gp	Global pointer
$29	$sp	Stack pointer
$30	$fp	Frame pointer
$31	$ra	Return address

MIPS functions

If you plan on calling other functions or using saved registers, you'll need to use the following function template:

Prologue:

```
FunctionFoo:
    addiu $sp, $sp, -FrameSize #reserve space on the stack
    sw $ra, 0($sp) #store needed registers
    sw $s0, 4($sp)
    …save the rest of the registers …
    sw $sx, FrameSize-4($sp)
```

Body:

```
    …Do some stuff …
```

Epilogue:

```
    lw $sx, FrameSize-4($sp) #restore registers
    …load the rest of the registers…
    lw $s0, 4($sp)
    lw $ra, 0($sp)
    addiu $sp, $sp, FrameSize #release stack spaces
    jr $ra #return to normal execution
```

Exercises:

What are the instructions to branch on each of the following conditions?

`$s0 < $s1`, `$s0 <= $s1`, `$s0 > 1`, `$s0 >= 1`

```
slt $t0 $s0 $s1      slt $t0 $s1 $s0      addi $t1 $0 $1      slti $t0 $s0 1
bne $t0 $0 Lbl       beq $t0 $0 Lbl       slt $t0 $t1 $s0     beq $t0 $0 Lbl
                                          bne $t0 $0 Lbl
```

What are the 3 meanings unsigned can have in MIPS?

lbu – Sign extend the loaded byte into the register
addu/addiu/subu – Do not warn on overflow.
sltu/sltiu – Perform unsigned comparison

Translate the following MIPS function into C or vice versa:

C	MIPS

`int Foo(int *arr,int ind) {`
`  int res = 0;`
`  while(ind>=1) {`
`  res += arr[ind];`
`  ind--;`
`  }`
`}`
	`Foo:  add $v0, $zero, $zero`
	`Loop:  slti $t0, $a1, 1`
	`  bne $t0, $zero, End`
	`  sll $t1, $a1, 2`
	`  add $t2, $a0, $t1`
	`  lw $t3, 0($t2)`
	`  add $v0, $v0, $t3`
	`  addi $a1, $a1, -1`
	`  j Loop`
	`End:  jr $ra`

`/* What does this program do? */`
`computes floor of log base 2 of a. The b`
`argument in Recur is technically unnecessary.`
 	
`int Mystery(unsigned int a){`
`  int b = 0;`
`  return Recur(a,b);`
`}`
 	
`int Recur(unsigned int a, int b){`
`  if(a==0) {`
`  return 0;`
`  } else {`
`  b+=1;`
`  a>>=1;`
`  return Recur(a,b)+1;`
`  }`
`}`
	`Mystery:  addi $a1, $0, $0`
	`  addiu $sp, $sp, -4`
	`  sw $ra, 0($sp)`
	`  jal Recur`
	`  lw $ra, 0($sp)`
	`  addiu $sp, $sp 4`
	`  jr $ra`
	
	`Recur:  bne $a0, $0, Body`
	`  add $v0, $0, $0`
	`  jr $ra`
	`Body:  addi $a1, $a1, 1`
	`  srl $a0, $a0, 1`
	`  addiu $sp, $sp, -4`
	`  sw $ra, 0($sp)`
	`  jal Recur`
	`  addi $v0, $v0, 1`
	`  lw $ra, 0($sp)`
	`  addiu $sp, $sp 4`
	`  jr $ra`

```
void swap(int * a, in * b){
  int temp= *a; //use the stack
  *a = *b;
  *b = temp;
}
```

```
swap: addiu $sp, $sp, -4
lw $t0 0($a0)
sw $t0 0($sp)
lw $t0 0($a1)
sw $t0 0($a0)
lw $t0 0($sp)
sw $t0 0($a1)
addiu $sp, $sp, 4
jr $ra
```

```
void insertionSort(int * arr, int size){
  int i, j;
  for(i=1; i<size; i++){
    j=i;
    while(j>0 && arr[j]<arr[j- 1]){
      swap(arr + j, arr + (j- 1));
      j- -;
    }
  }
}
```

```
iSort: addiu $sp, $sp, -12
sw $ra, 0($sp)
sw $s0, 4($sp) #using $s0 for i
sw $s1, 8($sp) #using $s1 for j
addiu $s0 $0 1  #i=1
For:
slt $t0 $s0 $a1
beq $t0 $0 Done    # i<size
addu $s1 $0 $s0    # j=i
While:
sll $t0 $s1 2
addiu $t0 $t0 $s0 #$t0 = (arr+j)
lw $t1 0($t0)      #$t1 = arr[j]
addiu $t2 $t0 -4   #$t2 = (arr+j-1)
lw $t3 0($t2)      #$t3 = arr[j-1]
slt $t4 $0 $s1
beq $t4 $0 For     #j<=0
slt $t4 $t1 $t3
beq $t4 $0 For #OR arr[j]>=arr[j-1]
addu $a0 $0 $t0
addu $a1 $0 $t2
jal swap
#note - this wipes out the t and a
#registers! Fortunately, we don't
#need them anymore
addiu $s1 $s1 -1
j For
Done:
lw $s1, 8($sp) #restore s1,s0
lw $s0, 4($sp) #to previous values
lw $ra, 0($sp)
addiu $sp, $sp, 12
jr $ra
```