Computer Architecture

CSEN 3104

Lecture 5

Dr. Debranjan Sarkar

MIPS Design Principles

- Keep all instructions a single size
- Always require three register operands in arithmetic instructions
- Has only 32 registers
- PC-relative addressing for conditional branches
- Immediate addressing for constant operands

MIPS: Registers and Memory

- 32 numbers of General Purpose Registers (32-bit each) (R0 to R31)
- One 32-bit Program Counter (PC)
- 32 bit addressing capability for memory (capacity 4GB max)
- Two views of memory:
  - 2^{32} bytes with addresses 0, 1, 2, …, 2^{32}-1
  - 2^{30} 4-byte words with addresses 0, 4, 8, …, 2^{32}-4
- Both views use byte addresses
- Word address must be multiple of 4

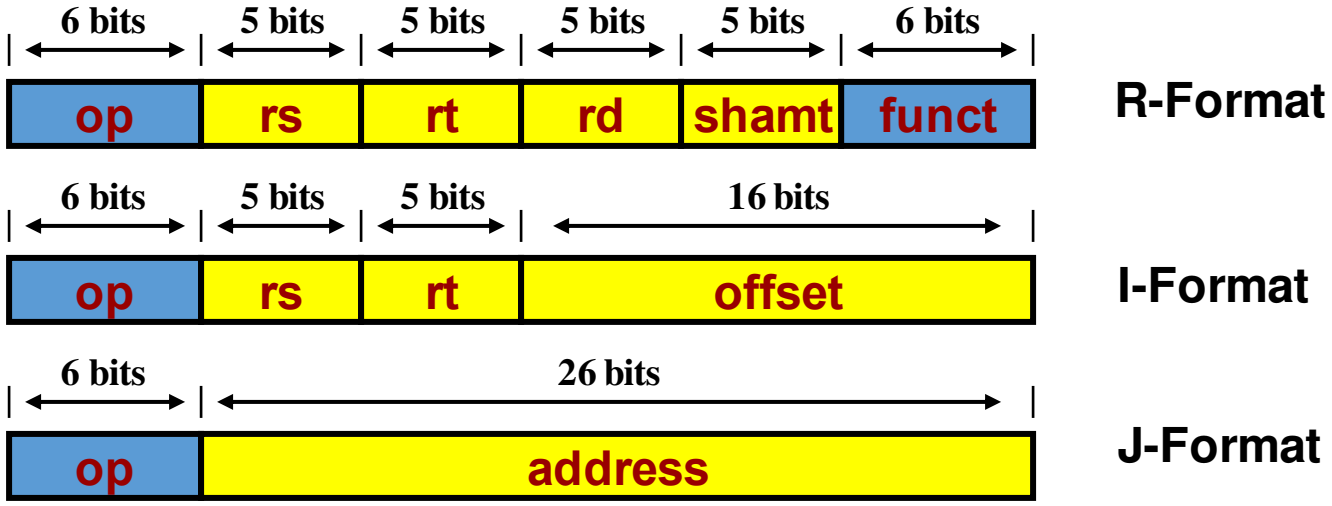
MIPS registers and usage

Each register may be referred to by number or name

Name	Register Number	Usage
$zero	0	The constant value 0
$at	1	Reserved for assembler
$v0 - $v1	2 – 3	Values for results and expression evaluation
$a0 - $a3	4 – 7	Arguments
$t0 - $t7	8 – 15	Temporary registers
$s0 - $s7	16 – 23	Saved registers
$t8 - $t9	24 – 25	More temporary registers
$k0 - $k1	26 – 27	Reserved for Operating System kernel
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

MIPS Instructions

- All instructions 1 word = 32 bits
- 3 different formats
- Different formats for different purposes



MIPS Arithmetic & Logical Instructions

- Manipulate data in registers
  - Always 3 operands: destination + 2 sources
  - Operand order is fixed
  - Operands are always general purpose registers
  - Instruction usage (assembly)

    add dest, src1, src2 dest=src1 + src2
  - Example

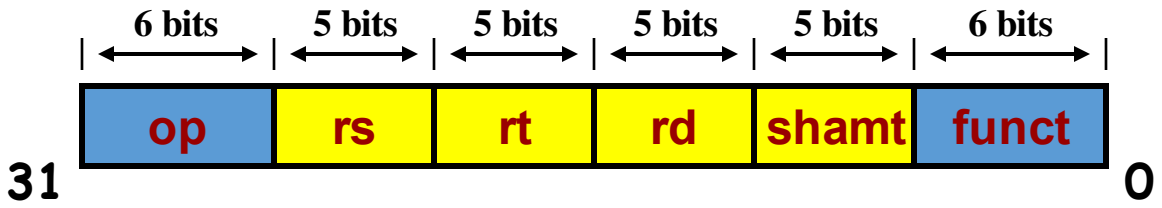
    add $s1, $s2, $s3 $s1 = $s2 + $s3

    or $s3, $s4, $s5 $s3 = $s4 OR $s5

    sub $s1, $s2, $s3 $s1 = $s2 - $s3

    and $s3, $s4, $s5 $s3 = $s4 AND $s5

Arithmetic & Logical Instructions (R-Format)



- Used for arithmetic, logical, shift instructions
  - op: Basic operation of the instruction (*opcode*) (Always 0 for R-Format)
  - rs: first register source operand
  - rt: second register source operand
  - rd: register destination operand
  - shamt: shift amount (0 when Not Applicable)
  - funct: function code (identifies the specific R-format instruction)

MIPS Data Transfer Instructions

- Transfer data between registers and memory
- Instruction format (assembly)

```
lw $dest, offset($addr)    load word
sw $src, offset($addr)     store word
```

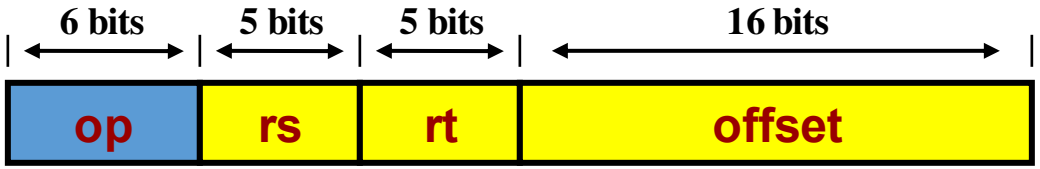
- Example
-

```
lw $s1, 100($s2)        $s1 = Memory[$s2 + 100]
sw $s1, 100($s2)        Memory[$s2 + 100] = $s1
```

- Uses:
  - Accessing a variable in main memory
  - Accessing an array element

MIPS Data Transfer Instructions (I-Format)

- Transfer data between registers and memory
- Have a constant value immediately present in the instruction
- Used for load, store instructions
  - op: Basic operation of the instruction (*opcode*)
  - rs: register containing base address
  - rt: register destination/source
  - offset: 16-bit signed address offset (-32,768 to +32,767)



MIPS Branch Instructions

- Alter program flow

```
beq $s1, $s2, 25     if ($s1==$s2) PC = PC + 4 + 4*25
```

- Unconditional jump

```
j LABEL                    # goto Label
```

- Conditional branches allow decision making

```
beq R1, R2, LABEL    if R1==R2 goto LABEL
bne R3, R4, LABEL    if R3!=R4 goto LABEL
```

- Example

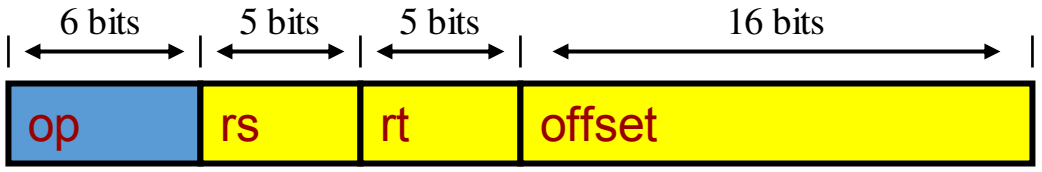
C Code

```
        if (i==j) goto L1;
        f = g + h;
L1:     f = f - i;
```

Assembly

```
        beq $s3, $s4, L1
        add $s0, $s1, $s2
L1:     sub $s0, $s0, $s3
```

MIPS Branch Instructions (I-Format)



- Branch instructions use I-Format
- Offset is added to PC when branch is effected

```
beq r0, r1, offset
```

has the effect:

Conversion to word offset

```
if (r0==r1) pc = pc + 4 + (offset << 2)
else pc = pc + 4;
```

Comparisons - What about <, <=, >, >=?

- `bne`, `beq` provide equality comparison
- `Slt dest, src1, src2` instruction sets `dest` if `src1 < src2`
-

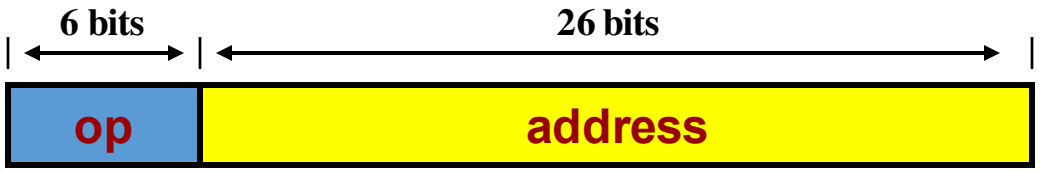
```
slt $t0,$s3,$s4        # if $s3<$s4 $t0=1;        here $t0 is the condition register
                       # else $t0=0;
```

- Combine `Slt` with `bne` or `beq` to branch if less than

```
slt $t0,$s3,$s4        # if (a<b)
bne $t0,$zero, Less    #    goto Less;
```

- Why not include a `blt` instruction in hardware?
  - Supporting in hardware would lower performance
  - Assembler provides this function if desired (by generating the two instructions)

Jump Instructions (J-Format)



- Jump Instruction uses J-Format (op=2)
- The 26 bits are achieved by dropping the high-order 4 bits of the address and the low-order 2 bits
- The low order 2 bits are always 00, since addresses are always divisible by 4
- What happens during execution?

PC = PC[31:28] : (IR[25:0] << 2)

Concatenate upper 4 bits of PC to form complete 32-bit address

Conversion to word offset

- jal (Jump and Link) Instruction has op=3

Constants / Immediate Instructions

- Small constants are used quite frequently (50% of operands)

  e.g.,
  ```
  A = A + 5;
  B = B + 1;
  C = C - 18;
  ```

- MIPS Immediate Instructions (I-Format):

  ```
  addi $29, $29, 4
  slti $8, $18, 10
  andi $29, $29, 6
  ori $29, $29, 4
  ```

  } **Arithmetic instructions sign-extend immediate data**

  } **Logical instructions don't sign extend immediate data**

- Allows upto 16-bit constants, because
  - 16 bits fits neatly in a 32-bit instruction
  - most constants are small (i.e. < 16 bits)
- How do you load just a constant into a register?
  - ori $5, $zero, 666

MIPS Logical Instructions

- and, andi - bitwise AND
- or, ori - bitwise OR
- Example
- and $s2,$s0,$s1 $s2 ← $s0 AND $s1
- Bitwise AND the content of register $s0 with that of $s1 and put the result in register $s2
- ori $s3,s2,252 $s3 ← $s2 OR 252_{10}
- Bitwise OR the content of register $s2 with 252_{10} and put the result in register $s3

Loading 32-Bit Immediate data in a register

- Normally, Immediate operations provide for 16-bit constants.
- 32-bit constant can be loaded in a register, using two instructions
- Suppose we want to load in register $t0 the value $0A50FB2F0_{16}$
- *load upper immediate - lui* (I-Format) instruction is used to set the upper 16 bits of a constant in a register
- After execution of the instruction
  - lui $t0, 1010010100001111
- The content of the register $t0 would be $0A50F0000_{16}$ (lower 16 bits filled with 0)
- Then *ori* instruction is used to fill in lower 16 bits
  - ori $t0, $t0, 1011001011110000
- After execution of this instruction
- The content of the register $t0 would be $0A50FB2F0_{16}$

MIPS Shift Instructions

- MIPS Logical Shift Instructions
  - Shift left: sll (shift-left logical) instruction
  - Right shift: srl (shift-right logical) instruction
- Example
- sll $s1,$s0,4 $s1 ← $s0 << 4 Shift left logical register $s0 by 4 bits and put the result in $s1
- srl $s2,$s1,8 $s2 ← $s1 >> 8 Shift right logical register $s1 by 8 bits and put the result in $s2

Thank You