Lecture 5

CMPEN 331

R-format Example

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

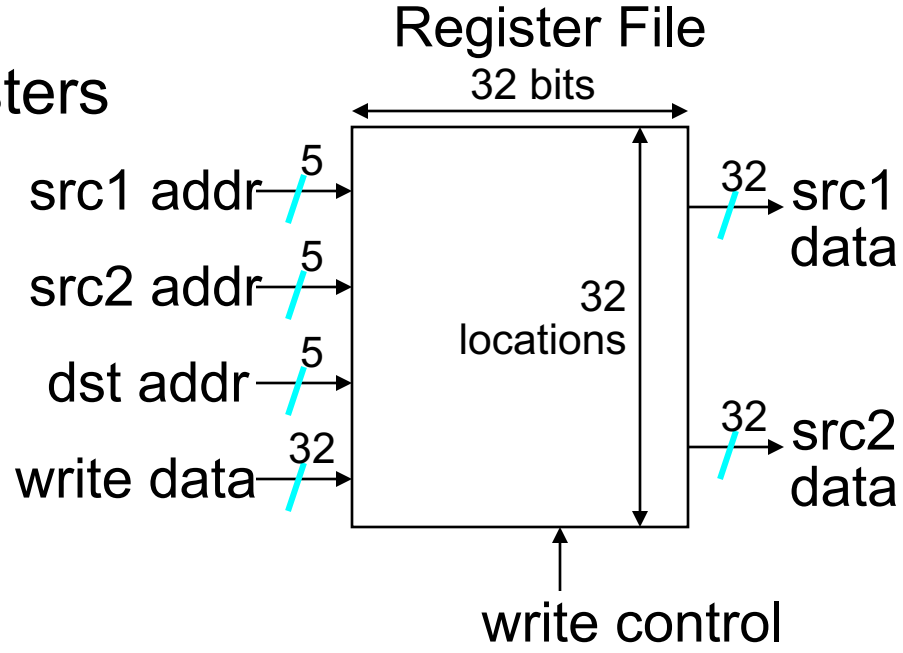
```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$$00000010001100100100000000100000_2 = 02324020_{16}$$

MIPS Register File

- Holds thirty-two 32-bit registers
  - Two read ports and
  - One write port



- Registers are
  - Faster than main memory
    - But register files with more locations are slower (e.g., a 64 word file could be as much as 50% slower than a 32 word file)
  - Can hold variables so that
    - Code density improves (since register are named with fewer bits than a memory location)

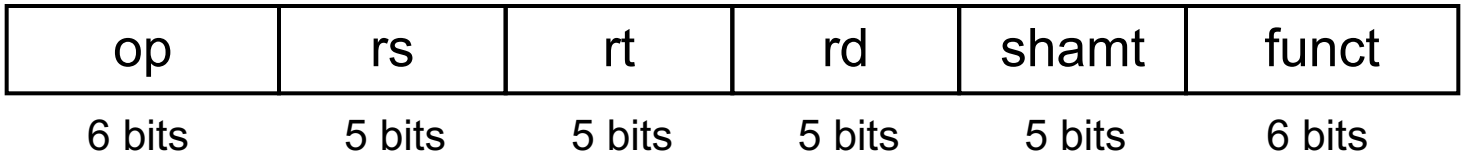
Logical Operations

- Instructions for bitwise manipulation

Operation	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR	|	|	or, ori
Bitwise NOT	~	~	nor

- Useful for extracting and inserting groups of bits in a word

Shift Operations



- shamt: how many positions to shift
- Shift left logical
  - Shift left and fill with 0 bits
  - `sll` by *i* bits multiplies by 2^i
- Shift right logical
  - Shift right and fill with 0 bits
  - `srl` by *i* bits divides by 2^i (unsigned only)

MIPS Shift Operations

- Shifts move all the bits in a word left or right

```
sll $t2, $s0, 8    #$t2 = $s0 << 8 bits
srl $t2, $s0, 8    #$t2 = $s0 >> 8 bits
```

- Instruction Format (R format)

```
sll $t2, $s0, 8
```

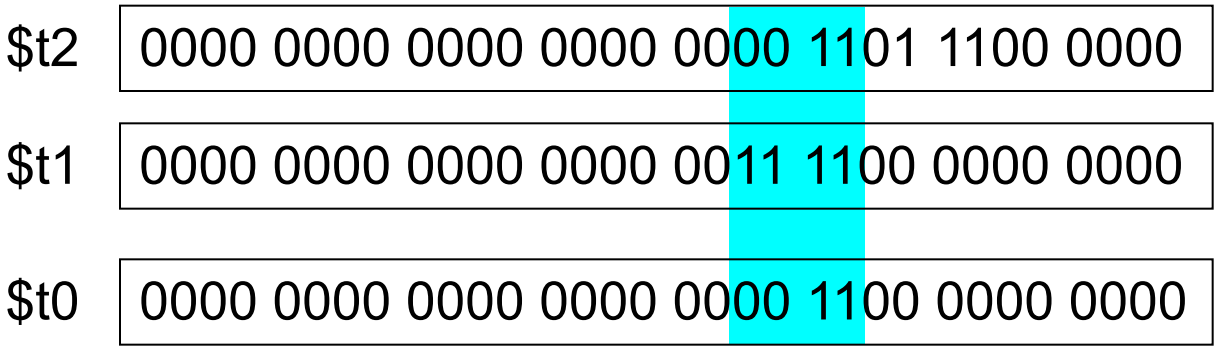
0		16	10	8	0x00

- Such shifts are called logical because they fill with zeros
  - Notice that a 5-bit shamt field is enough to shift a 32-bit value $2^5 - 1$ or 31 bit positions

AND Operations

- Useful to mask bits in a word
  - Select some bits, clear others to 0

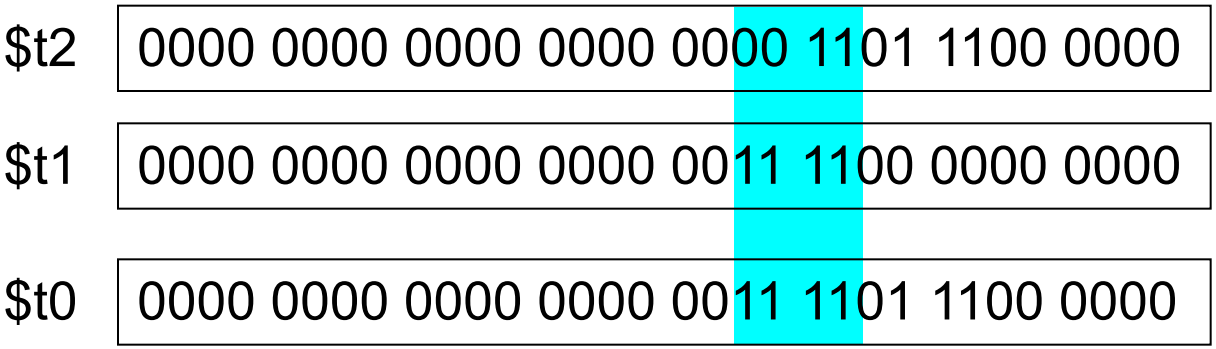
```
and $t0, $t1, $t2
```



OR Operations

- Useful to include bits in a word
  - Set some bits to 1, leave others unchanged

```
or $t0, $t1, $t2
```



NOT Operations

- Useful to invert bits in a word
  - Change 0 to 1, and 1 to 0
- MIPS has NOR 3-operand instruction
  - a NOR b == NOT (a OR b)

```
nor $t0, $t1, $zero
```

Register 0: always read as zero

$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	1111 1111 1111 1111 1100 0011 1111 1111

MIPS Memory Access Instructions

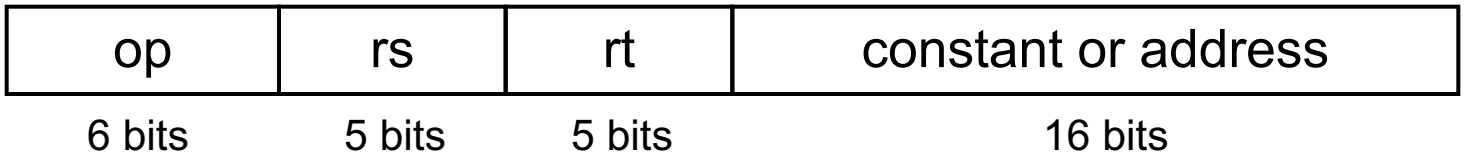
- MIPS has two basic data transfer instructions for accessing memory

```
lw   $t0, 4($s3)   #load word from memory
sw   $t0, 8($s3)   #store word to memory
```

- The data is loaded into (lw) or stored from (sw) a register in the register file – a 5 bit address

- The memory address – a 32 bit address – is formed by adding the contents of the base address register to the offset value
  - offset can be positive or negative.

MIPS I-format Instructions



- Immediate arithmetic and load/store instructions
  - rs: destination or source register number
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$
  - Address: offset added to base address in rs

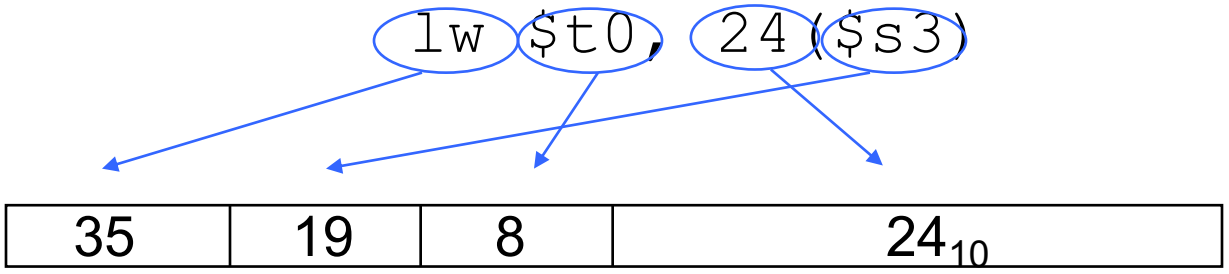
- *Design Principle 4:* Good design demands good compromises
  - Keep formats as similar as possible

Lecture 6

CMPEN 331

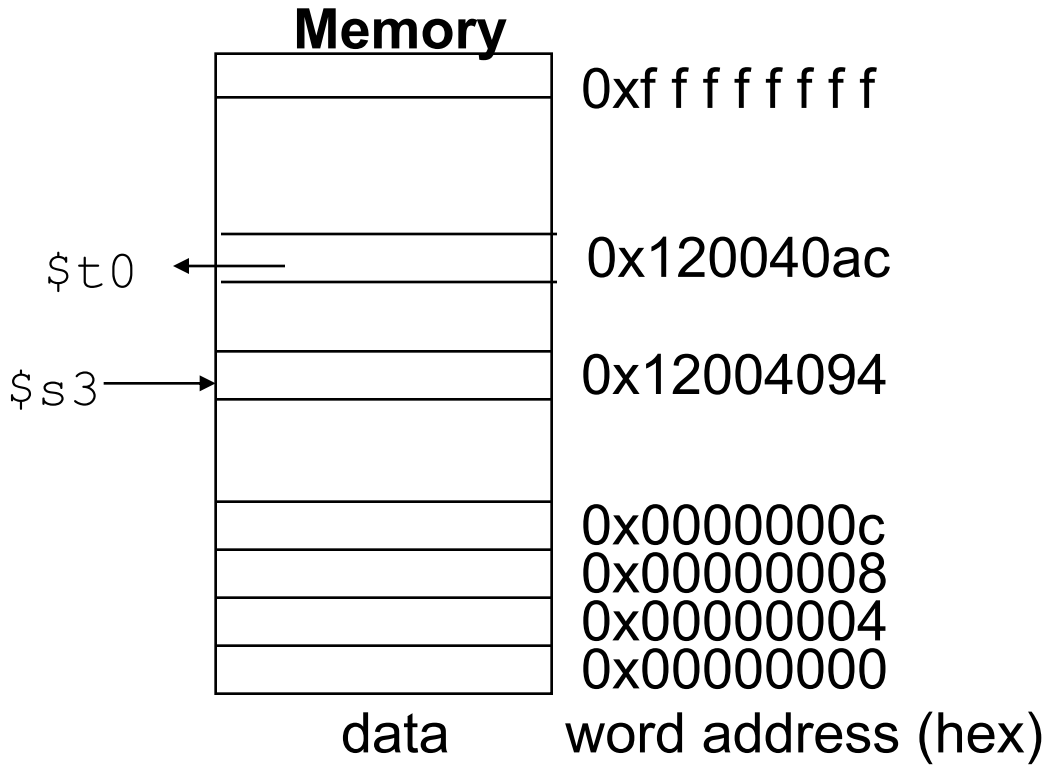
Machine Language - Load Instruction

- Load/Store Instruction Format (I format):



24_{10} + $s3 =

. . . 0001 1000
+ . . . 1001 0100
. . . 1010 1100 =
0x120040ac



MIPS Logical Operations

- There are a number of bit-wise logical operations in the MIPS ISA
- Instruction Format (R format)

```
and $t0, $t1, $t2 #$t0 = $t1 & $t2
or  $t0, $t1, $t2 #$t0 = $t1 | $t2
nor $t0, $t1, $t2 #$t0 = not($t1 | $t2)
```

- Instruction Format (I format)

```
andi $t0, $t1, 0xFF00  #$t0 = [$t1 & ff00]
ori  $t0, $t1, 0xFF00  #$t0 = [$t1 | ff00]
```

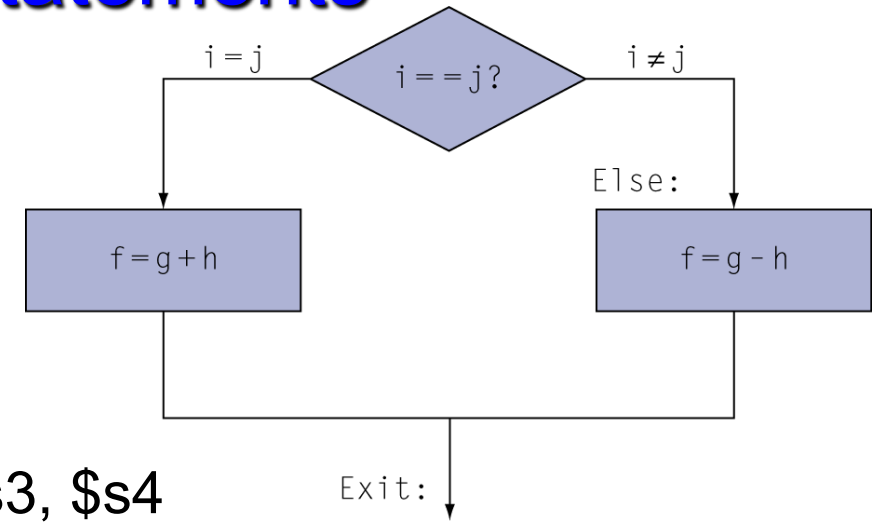
Conditional Operations

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially
- `beq rs, rt, L1`
  - if (rs == rt) branch to instruction labeled L1;
- `bne rs, rt, L1`
  - if (rs != rt) branch to instruction labeled L1;
- `j L1`
  - unconditional jump to instruction labeled L1

Compiling If Statements

- C code:

```
if (i==j)
f = g+h;
else f = g-h;
```



  - f, g, h, i, j … in $s0, $s1, $s2, $s3, $s4
  - Compiled MIPS code:

```
bne $s3, $s4, L1  #go to Else if i≠j
add $s0, $s1, $s2
j   Exit              #go to Exit
L1: sub $s0, $s1, $s2
Exit: …
```

Assembler calculates addresses

Compiling Loop Statements

- C code:

```
while (save[i] == k) i += 1;
```

  - i in $s3, k in $s5, address of save in $s6

- Compiled MIPS code:

The first step is to load save[i] into a temporary register, we need to have its address first. We have to multiply the index i by 4.

```
      Loop: sll $t1, $s3, 2       # Temp register $t1= i*4
#To get the address of save[i], we need to add $t1 and
   the
#base of save in $s6
      add  $t1, $t1, $s6    # $t1 address of save[i]
      lw   $t0, 0($t1)      # Temp reg $t0 = save[i]
      bne  $t0, $s5, Exit   # go to Exit if save[i] ≠ k
      addi $s3, $s3, 1      #  i = i +1
        j    Loop
   Exit:
```

Quiz

- Quiz 1