ICS 233
Computer Architecture & Assembly Language

MIPS PROCESSOR INSTRUCTION SET

ICS 233
Computer Architecture & Assembly Language

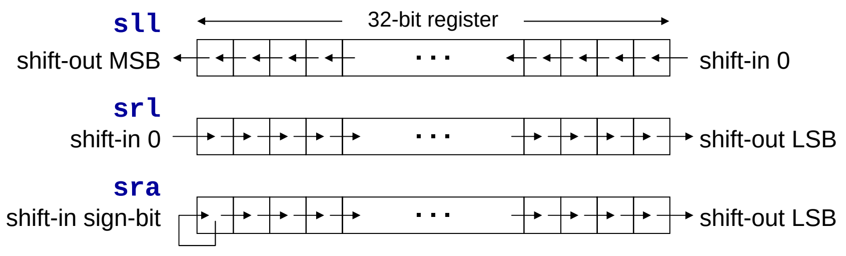
Lecture 6

Lecture Outline

- ❑ **MIPS Shift Instructions**
- ❑ **MIPS Multiply & Divide Instructions**
- ❑ **MIPS Data Transfer Instructions**

Shift Operations

- **Shifting is to move all the bits in a register left or right**
- **Shifts by a constant amount: `sll, srl, sra`**
  - **`sll/srl` mean shift left/right logical by a constant amount**
  - **The 5-bit shift amount field is used by these instructions**
  - **`sra` means shift right arithmetic by a constant amount**
  - **The sign-bit (rather than 0) is shifted from the left**



Shift Instructions

Instruction		Meaning	R-Type Format					
sll	$s1,$s2,10	$s1 = $s2 << 10	op = 0	rs = 0	rt = $s2	rd = $s1	sa = 10	f = 0
srl	$s1,$s2,10	$s1 = $s2>>>10	op = 0	rs = 0	rt = $s2	rd = $s1	sa = 10	f = 2
sra	$s1, $s2, 10	$s1 = $s2 >> 10	op = 0	rs = 0	rt = $s2	rd = $s1	sa = 10	f = 3
sllv	$s1,$s2,$s3	$s1 = $s2 << $s3	op = 0	rs = $s3	rt = $s2	rd = $s1	sa = 0	f = 4
srlv	$s1,$s2,$s3	$s1 = $s2>>>$s3	op = 0	rs = $s3	rt = $s2	rd = $s1	sa = 0	f = 6
srav	$s1,$s2,$s3	$s1 = $s2 >> $s3	op = 0	rs = $s3	rt = $s2	rd = $s1	sa = 0	f = 7

- Shifts by a variable amount: **`sllv`**, **`srlv`**, **`srav`**
  - Same as **`sll`**, **`srl`**, **`sra`**, but a register is used for shift amount
- Examples: assume that $s2 = 0xabcd1234, $s3 = 16

```
sll  $s1,$s2,8     $s1 = $s2<<8     $s1 = 0xcd123400
sra  $s1,$s2,4     $s1 = $s2>>4     $s1 = 0xfabcd123
srlv $s1,$s2,$s3   $s1 = $s2>>$s3   $s1 = 0x0000abcd
```

op=000000	rs=$s3=10011	rt=$s2=10010	rd=$s1=10001	sa=00000	f=000110

MIPS - Shift Instructions

❑ **sll (shift left logical)**

➢ **Instruction Mnemonic :**

sll rd, rs, const ;where rs, rd are registers,
; const is a 5-bit constant (shift count)

➢ **Meaning :**

rd ← rs << const ; logical shift left rs by const bits

➢ **Example :**

sll $s1, $s2, 10 ; $s1← $s2 << 10

➢ ____

❑ **srl (shift right logical)**

➢ **Instruction Mnemonic :**

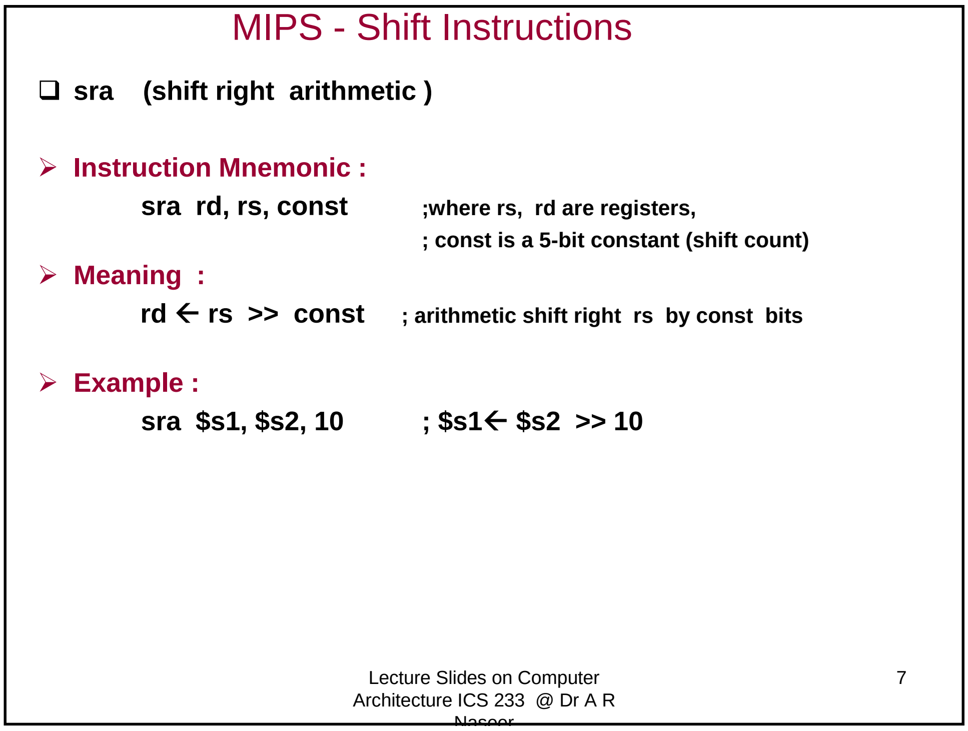
srl rd, rs, const ;where rs, rd are registers,
; const is a 5-bit constant (shift count)

➢ **Meaning :**

rd ← rs >> const ; logical shift right rs by const bits

➢ **Example :**

srl $s1, $s2, 10 ; $s1← $s2 >> 10



MIPS - Shift Instructions

- ❑ **sra (shift right arithmetic)**

- ➢ **Instruction Mnemonic :**

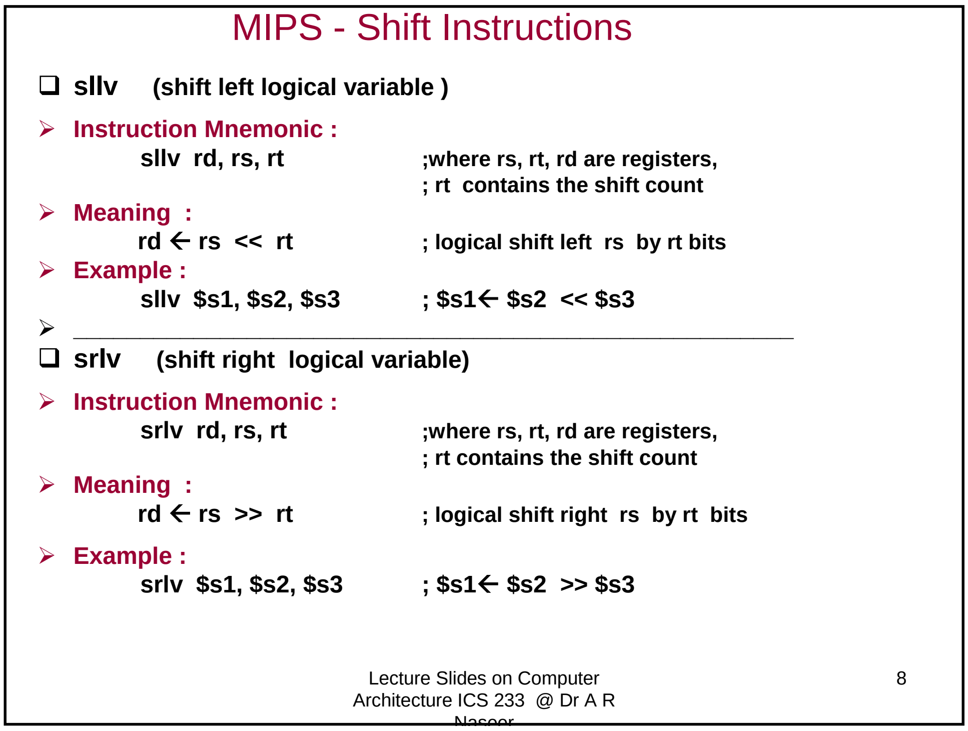
```
sra  rd, rs, const          ;where rs,  rd are registers,
                            ; const is a 5-bit constant (shift count)
```

- ➢ **Meaning :**

```
rd ← rs  >>  const          ; arithmetic shift right  rs  by const  bits
```

- ➢ **Example :**

```
sra  $s1, $s2, 10           ; $s1← $s2  >> 10
```



MIPS - Shift Instructions

- ❑ **sllv (shift left logical variable)**
- ➢ **Instruction Mnemonic :**

```
sllv  rd, rs, rt            ;where rs, rt, rd are registers,
                            ; rt  contains the shift count
```

- ➢ **Meaning :**

```
rd ← rs  <<  rt             ; logical shift left  rs  by rt bits
```

- ➢ **Example :**

```
sllv  $s1, $s2, $s3         ; $s1← $s2  << $s3
```

- ➢ ____

- ❑ **srlv (shift right logical variable)**
- ➢ **Instruction Mnemonic :**

```
srlv  rd, rs, rt            ;where rs, rt, rd are registers,
                            ; rt contains the shift count
```

- ➢ **Meaning :**

```
rd ← rs  >>  rt             ; logical shift right  rs  by rt  bits
```

- ➢ **Example :**

```
srlv  $s1, $s2, $s3         ; $s1← $s2  >> $s3
```

MIPS - Shift Instructions

❑ **srav (shift right arithmetic variable)**

➢ **Instruction Mnemonic :**

srav rd, rs, rt ;where rs, rt, rd are registers,
; rt contains the shift count

➢ **Meaning :**

rd ← rs >> rt ; arithmetic shift right rs by rt bits

➢ **Example :**

srav $s1, $s2, $s3 ; $s1← $s2 >> $s3

Binary Multiplication

- **Shift-left (`sll`) instruction can perform multiplication**
  - When the multiplier is a power of 2
- You can factor any binary number into powers of 2
  - Example: multiply **`$s1`** by **`36`**
    - Factor 36 into (4 + 32) and use distributive property of multiplication
  - **`$s2 = $s1*36 = $s1*(4 + 32) = $s1*4 + $s1*32`**

```
sll  $t0, $s1, 2      ; $t0 = $s1 * 4
sll  $t1, $s1, 5      ; $t1 = $s1 * 32
addu $s2, $t0, $t1    ; $s2 = $s1 * 36
```

Try This.. Binary Multiplication

Multiply $s1 by 26, using shift and add instructions

Hint: 26 = 2 + 8 + 16

```
sll  $t0, $s1, 1        ; $t0 = $s1 * 2
sll  $t1, $s1, 3        ; $t1 = $s1 * 8
addu $s2, $t0, $t1      ; $s2 = $s1 * 10
sll  $t0, $s1, 4        ; $t0 = $s1 * 16
addu $s2, $s2, $t0      ; $s2 = $s1 * 26
```

Multiply $s1 by 31, Hint: 31 = 32 – 1

```
sll  $s2, $s1, 5        ; $s2 = $s1 * 32
subu $s2, $s2, $s1      ; $s2 = $s1 * 31
```

MIPS - Multiply Instructions

❑ **mult (multiply registers signed)**

➢ **Instruction Mnemonic :**

mult rd, rs ;where rs, rd are registers

➢ **Meaning :**

Hi, Lo = rd * rs ; 64-bit signed product in Hi,Lo

➢ **Example :**

mult $s2, $s3 ; Hi,Lo ← $s2 * $s3

❑ **multu (multiply registers unsigned)**

➢ **Instruction Mnemonic :**

multu rd, rs ;where rs, rd are registers

➢ **Meaning :**

Hi, Lo = rd * rs ; 64-bit unsigned product in Hi,Lo

➢ **Example :**

multu $s2, $s3 ; Hi,Lo ← $s2 * $s3

MIPS - divide Instructions

- ❑ div (divide registers signed)
- ➢ **Instruction Mnemonic :**

  div rd, rs ;where rs, rd are registers
- ➢ **Meaning :**

  Lo = rd / rs ; signed quotient in Lo

  Hi = rd mod rs ; signed remainder in Hi
- ➢ **Example :**

  div $s2, $s3 ; Lo ← $s2 / $s3 ; Hi ← $s2 mod $s3

- ❑ divu (divide registers unsigned)
- ➢ **Instruction Mnemonic :**

  divu rd, rs ;where rs, rd are registers

  Meaning :

  Lo = rd / rs ; unsigned quotient in Lo

  Hi = rd mod rs ; unsigned remainder in Hi
- ➢ **Example :**

  divu $s2, $s3 ; Lo ← $s2 / $s3 ; Hi ← $s2 mod $s3

MIPS – Data Transfer Instructions

- ❑ mfhi (move from Hi)
- ➢ **Instruction Mnemonic :**

  mfhi rd ;where rd is a register
- ➢ **Meaning :**

  rd = Hi ; get a copy of Hi in rd
- ➢ **Example :**

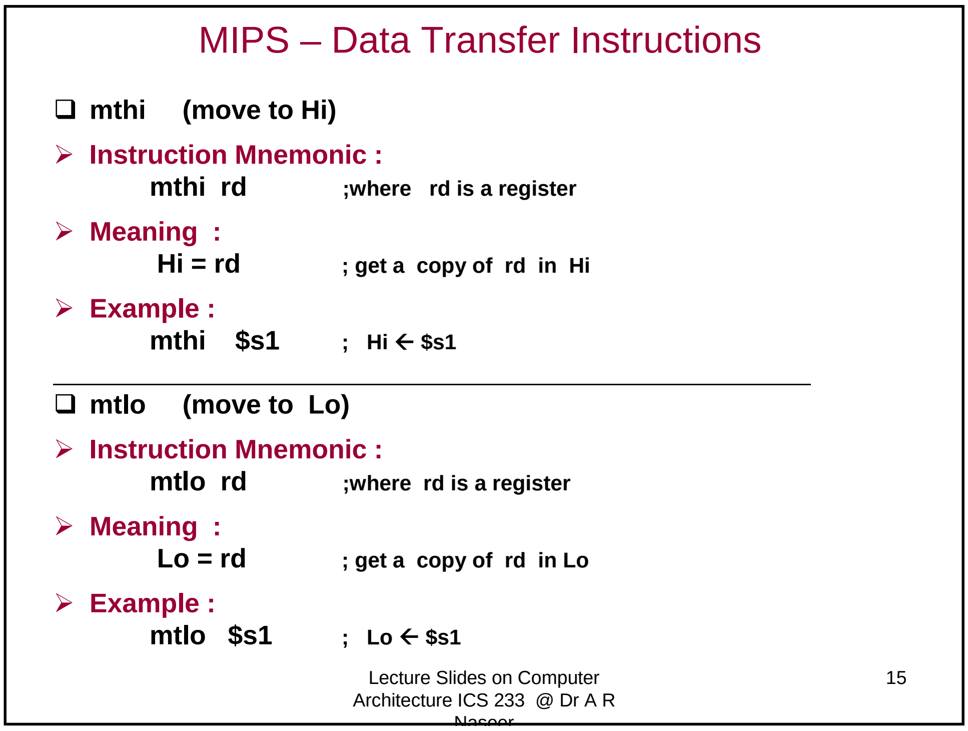
  mfhi $s1 ; $s1 ← Hi

- ❑ mflo (move from Lo)
- ➢ **Instruction Mnemonic :**

  mflo rd ;where rd is a register
- ➢ **Meaning :**

  rd = Lo ; get a copy of Lo in rd
- ➢ **Example :**

  mflo $s1 ; $s1 ← Lo



MIPS – Data Transfer Instructions

- ❑ **mthi (move to Hi)**
- ➢ **Instruction Mnemonic :**

  mthi rd ;where rd is a register
- ➢ **Meaning :**

  Hi = rd ; get a copy of rd in Hi
- ➢ **Example :**

  mthi $s1 ; Hi ← $s1

- ❑ **mtlo (move to Lo)**
- ➢ **Instruction Mnemonic :**

  mtlo rd ;where rd is a register
- ➢ **Meaning :**

  Lo = rd ; get a copy of rd in Lo
- ➢ **Example :**

  mtlo $s1 ; Lo ← $s1