MIPS32 Assembly Language Basics
(load instructions)

- **`lb Rt, address`**
  - Loads byte at **`address`** into register **`Rt`**
    - Byte is *sign-extended*
  - Use **`lbu`** for unsigned load
    - Byte is *zero-extended*
- **`lh`**, **`lhu`**, **`lw`** → for halfword (signed/unsigned) & word
- Pseudoinstruction:
  - **`la Rdest, address`**
    - Loads computed address (<u>not</u> contents at address) into register **`Rdest`**

MIPS32 Assembly Language Basics
(store instructions)

- **`sb Rt, address`**
  - Stores low byte from register **`Rt`** at **`address`**
- **`sh`**, **`sw`** → for halfword & word

MIPS32 Assembly Language Basics
(data movement instructions)

- **`mfhi Rd`**
  - Moves value from **`Hi`** register to **`Rd`**
- **`mflo Rd`**
  - Moves value from **`Lo`** register to **`Rd`**
- **`mthi Rs`**
  - Moves value from **`Rs`** to **`Hi`** register
- **`mtlo Rs`**
  - Moves value from **`Rs`** to **`Lo`** register
- Pseudoinstruction:
  - **`move Rdest, Rsrc`**
    - Copies contents of register **`Rsrc`** to register **`Rdest`**
    - Can be implemented as

```
addu Rdest, $0, Rsrc
```

MIPS32 Assembly Language Basics
(constant-manipulating instructions)

- **`lui Rt, imm`**
  - Loads (16-bit) **`imm`** into upper halfword (16 bits) of **`Rt`**
    - Lower-halfword (16 bits) of **`Rt`** *set to 0's*
- Pseudoinstruction:
  - **`li Rdest, imm`**
    - Loads value of **`imm`** (sign-extended) into register **`Rdest`**
    - If **`imm`** is positive and less than 32,768, can be implemented as

```
ori Rdest, $0, imm
```

ori zero-extends imm

    - If **`imm`** is negative or greater than 32,768, can be implemented as

```
lui $at, upper 16-bit of imm
ori Rdest, $at, lower 16-bits of imm
```

MIPS32 Assembly Language Basics
(arithmetic instructions - addition)

- **`add Rd, Rs, Rt`**
  - Equivalent in C++: **`Rd = Rs + Rt`**
  - Numbers in **`Rs`** and **`Rt`** treated as signed integers
  - Generates exception when overflow occurs
    - Use **`addu`** instead if overflow exception not needed
- **`addi Rt,Rs,imm`**
  - Equivalent in C++: **`Rt = Rs + imm`**
  - 16-bit **`imm`** is sign-extended
  - Generates exception when overflow occurs
    - Use **`addiu`** instead if overflow exception not needed

MIPS32 Assembly Language Basics
(arithmetic instructions - subtraction)

- **`sub Rd, Rs, Rt`**
  - Equivalent in C++: **`Rd = Rs - Rt`**
  - Numbers treated as signed integers
  - Generates exception when overflow occurs
    - Use **`subu`** instead if overflow exception not needed
- No immediate version (like that for addition)
  - Not needed → can use **`addi`** with *negative* **`imm`**

MIPS32 Assembly Language Basics
(arithmetic instructions - multiplication)

- **`mult Rs, Rt`**
  - Puts 64-bit result of **`Rs`**×**`Rt`** in **`Hi`** register (high-order word) and **`Lo`** register (low-order word)
  - Numbers treated as signed integers
    - Use **`multu`** to treat numbers as unsigned integers
  - Use **`mfhi`** and **`mflo`** to retrieve results from **`Hi`** and **`Lo`**
- **`mul Rd, Rs, Rt`**
  - Puts low-order 32 bits of **`Rs`**×**`Rt`** into **`Rd`**
    - Accomplishes **`mult Rs, Rt`**
      `mflo Rd`

Textbook says **`mul`** is a pseudoinstruction.

- Pseudoinstructions:
  - **`mulo Rdest, Rsrc1, Src2`**
    - Puts low-order 32 bits of **`Rsrc1`**×**`Src2`** into **`Rdest`**
    - **`Src2`** may be *register* or *immediate*
    - Generates exception when overflow occurs
    - Numbers treated as signed integers
      - Use **`mulou`** to treat numbers as unsigned integers

MIPS32 Assembly Language Basics
(arithmetic instructions - division)

- **`div Rs, Rt`**
  - Puts 64-bit result of **`Rs/Rt`** in **`Hi`** (remainder) & **`Lo`** (quotient)
  - Numbers treated as signed integers
    - Generates exception when overflow occurs
  - Use **`divu`** to treat numbers as unsigned integers
    - Doesn't generate exception when overflow occurs
  - Use **`mfhi`** and **`mflo`** to retrieve results from **`Hi`** and **`Lo`**

Textbook says both **`div`** & **`divu`** don't generate any exceptions.

- Pseudoinstructions:
  - **`div Rdest, Rsrc1, Src2`**
    - Puts quotient of **`Rsrc1/Src2`** into **`Rdest`**
    - Numbers treated as signed integers
      - Generates exception when overflow occurs
    - Use **`divu`** to treat numbers as unsigned integers
      - Doesn't generates exception when overflow occurs

MIPS32 Assembly Language Basics
(arithmetic instructions - shift)

- `sll Rd, Rt, shamt`
  `sllv Rd, Rt, Rs`
  `srl Rd, Rt, shamt`
  `srlv Rd, Rt, Rs`
  `sra Rd, Rt, shamt`
  `srav Rd, Rt, Rs`
  - Shifts **`Rt`** left/right by distance **`shamt`**/**`Rs`** & puts result in **`Rd`**
  - **`Rs`** field in instruction ignored for **`sll`**, **`srl`** & **`sra`**
  - Shifting left always brings in 0's (at right end)
  - Shift right ***logical*** also brings in 0's (at left end)
  - Shift right ***arithmetic*** *sign-extends* higher-order bits

MIPS32 Assembly Language Basics
(arithmetic instructions - rotation)

- Pseudoinstructions:
  - `rol Rdest, Rsrc1, Rsrc2`
    `ror Rdest, Rsrc1, Rsrc2`
    - Rotates **`Rsrc1`** left/right by distance **`Rsrc2`** & puts result in **`Rdest`**

MIPS32 Assembly Language Basics
(arithmetic instructions - miscellaneous)

- Pseudoinstructions:
  - **`abs Rdest, Rsrc`**
    - Puts absolute value of **`Rsrc`** into **`Rdest`**
  - **`neg Rdest, Rsrc`**
    - Puts negative value of **`Rsrc`** into **`Rdest`**
    - Number in **`Rsrc`** treated as signed integer
    - Generates exception when overflow occurs ⇐ When'll overflow occur?
  - **`negu Rdest, Rsrc`**
    - Puts negative value of **`Rsrc`** into **`Rdest`**
    - Number in **`Rsrc`** treated as unsigned integer
    - Doesn't generate exception
  - **`rem Rdest, Rsrc1, Rsrc2`**
    - Puts remainder of **`Rsrc1/Rsrc2`** into **`Rdest`**
    - Numbers treated as signed integers
      - Use **`remu`** to treat numbers as unsigned integers

MIPS32 Assembly Language Basics
(logical instructions)

- **`and Rd, Rs, Rt`**
  - Puts bitwise AND of **`Rs`** and **`Rt`** in **`Rd`**
- **`andi Rt, Rs, imm`**
  - Puts bitwise AND of **`Rs`** and *zero-extended* **`imm`** in **`Rt`**
- **`or Rd, Rs, Rt`**
  - Puts bitwise OR of **`Rs`** and **`Rt`** in **`Rd`**
- **`ori Rt, Rs, imm`**
  - Puts bitwise NOR of **`Rs`** and *zero-extended* **`imm`** in **`Rt`**
- **`nor Rd, Rs, Rt`**
  - Puts bitwise OR of **`Rs`** and **`Rt`** in **`Rd`**
- **`xor Rd, Rs, Rt`**
  - Puts bitwise XOR of **`Rs`** and **`Rt`** in **`Rd`**
- **`xori Rt, Rs, imm`**
  - Puts bitwise XOR of **`Rs`** and *zero-extended* **`imm`** in **`Rt`**
- Pseudoinstruction:
  - **`not Rdest, Rsrc`**
    - Puts bitwise NOT of **`Rsrc`** into **`Rdest`**

constants rare for NOR
so no NORI is provided

`nor Rdest, Rsrc, $0`

MIPS32 Assembly Language Basics
(comparison instructions)

- **`slt Rd, Rs, Rt`**
  `sltu Rd, Rs, Rt`
  - Sets **`Rd`** to 1 if **`Rs`** **<** **`Rt`**, otherwise sets **`Rd`** to 0
- **`slti Rt, Rs, imm`**
  `sltiu Rt, Rs, imm`
  - Sets **`Rt`** to 1 if **`Rs`** **<** *zero-extended* **`imm`**, otherwise sets **`Rt`** to 0

MIPS32 Assembly Language Basics
(comparison instructions)

- Pseudoinstructions:
  - **`seq Rdest, Rsrc1, Rsrc2`**
    - Sets **`Rdest`** to 1 if **`Rsrc1`** **==** **`Rsrc2`**, otherwise sets **`Rdest`** to 0
  - **`sge Rdest, Rsrc1, Rsrc2`**
    `sgeu Rdest, Rsrc1, Rsrc2`
    - Sets **`Rdest`** to 1 if **`Rsrc1`** **>=** **`Rsrc2`**, otherwise sets **`Rdest`** to 0
  - **`sgt Rdest, Rsrc1, Rsrc2`**
    `sgtu Rdest, Rsrc1, Rsrc2`
    - Sets **`Rdest`** to 1 if **`Rsrc1`** **>** **`Rsrc2`**, otherwise sets **`Rdest`** to 0
  - **`sle Rdest, Rsrc1, Rsrc2`**
    `sleu Rdest, Rsrc1, Rsrc2`
    - Sets **`Rdest`** to 1 if **`Rsrc1`** **<=** **`Rsrc2`**, otherwise sets **`Rdest`** to 0
  - **`sne Rdest, Rsrc1, Rsrc2`**
    - Sets **`Rdest`** to 1 if **`Rsrc1`** **!=** **`Rsrc2`**, otherwise sets **`Rdest`** to 0

MIPS32 Assembly Language Basics
(branch instructions)

- **`beq Rs, Rt, label`**
  - Branches to **`label`** if **`Rs == Rt`**
- **`bne Rs, Rt, label`**
  - Branches to **`label`** if **`Rs != Rt`**
- **`bgez Rs, label`**
  - Branches to **`label`** if **`Rs >= 0`**
- **`bgtz Rs, label`**
  - Branches to **`label`** if **`Rs > 0`**
- **`blez Rs, label`**
  - Branches to **`label`** if **`Rs <= 0`**
- **`bltz Rs, label`**
  - Branches to **`label`** if **`Rs < 0`**

MIPS32 Assembly Language Basics
(branch instructions)

- Pseudoinstructions:
  - **`b label`**
    - Branches to **`label`**
  - **`beqz Rsrc, label`**
    - Branches to **`label`** if **`Rsrc1 ==`** 0
  - **`bnez Rsrc, label`**
    - Branches to **`label`** if **`Rsrc1 !=`** 0
  - **`bge Rsrc1, Rsrc2, label`**
    `bgeu Rsrc1, Rsrc2, label`
    - Branches to **`label`** if **`Rsrc1 >= Rsrc2`**
  - **`bgt Rsrc1, Rsrc2, label`**
    `bgtu Rsrc1, Rsrc2, label`
    - Branches to **`label`** if **`Rsrc1 > Rsrc2`**
  - **`ble Rsrc1, Rsrc2, label`**
    `bleu Rsrc1, Rsrc2, label`
    - Branches to **`label`** if **`Rsrc1 <= Rsrc2`**
  - **`blt Rsrc1, Rsrc2, label`**
    `bltu Rsrc1, Rsrc2, label`
    - Branches to **`label`** if **`Rsrc1 < Rsrc2`**

MIPS32 Assembly Language Basics
(jump instructions)

- **`j target`**
  - Jumps to **`target`**
- **`jal target`**
  - Jumps to **`target`**
  - Saves address of next instruction in **`$ra`**
- **`jr Rs`**
  - Jumps to instruction whose address is in **`Rs`**

MIPS32 Assembly Language Basics
(nop pseudoinstruction)

- **`nop`**
  - For introducing "bubble" when dealing with pipeline hazards
  - Implemented as **`sll $0, $0, 0`**

(more instructions/pseudoinstructions)

there are still other instructions/pseudoinstructions

will introduce them as & when & if ever they are needed

MIPS32 AL – Tips/Tricks/Traps/Observations
(some details/quirks of MIPS)

- How <u>*immediate-field*</u> values are extended:
  - ***Arithmetic*** *instructions* (**`addi`**, *etc*.) always *sign extend*
    - Even for *unsigned* versions!
  - ***Logical*** instructions (**`andi`**, **`ori`**, **`xori`**) always *zero extend*
  - ***Set*** *instructions* do *sign extend* or *zero extend*:
    - **`slti`** *sign extend* the immediate
    - **`sltiu`** *zero extend* the immediate
  - ***Load/store*** *instructions* (**`lb`**, **`lbu`**, **`lw`**, *etc*.) always *sign extend*
- How *values* in load instructions are extended:
  - **`lb`**, **`lh`** → sign extend
  - **`lbu`**, **`lhu`** → zero extend
- All bits of dst/src registers are involved in all instructions:
  - All 32 bits of destination register changed in **`lui`**, **`lb`**, **`lh`**, …
  - All 32 bits of source registers read in **`add`**, **`sub`**, **`and`**, **`or`**, …
- **`jal`** puts *PC+4* (not PC) into **`$ra`**
  - NB: PC+4 points to ***2nd*** instruction after function-call instruction

MIPS32 AL – Tips/Tricks/Traps/Observations
(MIPS' use of **`unsigned`** is "overloaded")

- *Don't sign extend*:
  - Example: **`lbu`** (vs **`lb`**)
- *Don't trap overflow*:
  - Example: **`addu`**, **`addiu`**, **`subu`**, **`multu`**, **`divu`** (vs **`add`**, **`...`**)
- *Do unsigned compare*:
  - Example: **`sltu`**, **`sltiu`** (vs **`slt`**, **`slti`**)