Load and Store Instructions

Recall that MIPS is a Load-Store Architecture. Data has to be in registers before it can be massaged.

Load brings data from memory to a register

Store brings data back to memory from a register.

Each load/store must specify a memory address and the i-unit (byte, word, etc.)

In MIPS a memory address (recall byte addressable) is a 32-bit unsigned number.

But ... instructions are only 32-bit wide so...

Addressing in Load/Store instructions

Use of a *base register* (one of the 32 registers)

Use of a 16-bit *offset*, or displacement, (in the instruction) that represents a signed number.

The base register points somewhere in memory and the offset is added to that pointer value.

With a single base register, one can access any byte within $\pm$ 32KB of the address pointed to by the base register.

Examples of Load and Store Instructions

Load Word from memory

Opcode	Operands	Comments
LW	rt,offset(base)	#rt = memory[base + offset]

Store Word into memory

Opcode	Operands	Comments
SW	rt,offset(base)	#memory[base + offset] = rt

For bytes (or half-words) only the lower byte (half-word) of a register is addressable. For load, you need to specify whether it is zero-extended or sign extended.

Opcode	Operands	Comments
LB	rt,offset(base)	#rt = sign-extended(memory[base + offset])
LBU	rt,offset(base)	#rt = zero-extended(memory[base + offset])
SB	rt,offset(base)	#store least significant byte of rt

Load-Store Formats

A memory address is 32 bits so it cannot be directly encoded in one instruction.

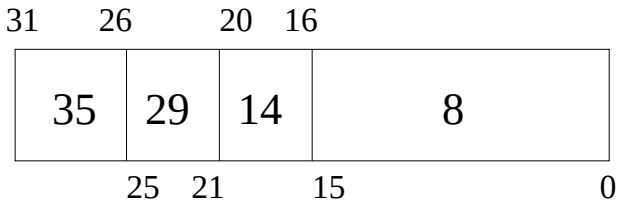
Recall use of a base register + offset (16-bits)

Thus, need for an OPCODE, a destination (for Load; source for Store) register, a base register, and an offset (i.e., same format as I-format)

Example:

Opcode	Operands	Comments
LW	$14,8($sp)	#r14 loaded from stack + 8

The LW opcode is 0x23 (decimal 35)



Loading constants in a register

If the constant is small, i.e., 16 bits or less, use LI.

Opcode	Operands	Comments
LI	$14,8	#r14 = 8

But LI does not exist (no opcode). LI is a *pseudoinstruction*. The assembler will generate ADDI (or the logical instruction ORI) as:

Opcode	Operands	Comments
ADDI	$14,$0,8	#r14 = 8

If the constant is large (does not fit in 16 bits , i.e., not in the range $\pm$ 32K), use LUI (load upper immediate).

Use of LUI

If we want to load a "large" constant, i.e., one that does not fit in 16 bits

- use LUI to load the leftmost 16 bits in the most significant half of the register (LUI will zero out the rightmost 16 bits)
- use ADDI (or ORI) for the lower 16 bits.

Opcode	Operands	Comments
LUI	rt,immed	#rt <31,16> = immed #rt <15,0> = 0

Example: Load the constant 0x1B236723 in Register $t0

```
LUI  $t0,0x1B23
ADDI $t0,$t0,0x6723
```

How to get the base address in the base register

Method 1. Let the assembler do it

```
          .data                 #define prog. data section
xyz:      .word 1               #some data here
           ....                 #possibly some other data
          .text                 #define the program code
           ...                  #lines of code
           lw     $5,xyz        #loads contents of xyz in r5
```

In fact the assembler generates an instruction of the form:

```
lw      $5, offset($gp)   #gp is register 28, the global pointer
```

Method 2. Do it yourself using the LA pseudoinstruction that Loads an address rather than the contents found at the address. LA is in fact LUI followed by ADDI.

```
la      $6,xyz      #r6 contains the address of xyz
lw      $5, 0($6)   #r5 contains the contents of xyz
```

Method 3. When the address is a constant that you know use LI (if less than $\pm$ 32K) or LUI and ADDI (or ORI) otherwise.