null
**Pointer summary**

- In the “C” world and in the “machine” world:
  - a pointer is just the address of an object in memory
  - size of pointer is fixed regardless of size of object
  - to get to the next object increment by the objects size in bytes
  - to get the the i^th object add i*sizeof(object)
- More details:
  - int R[5] → R is int* constant address of 20 bytes
  - R[j] → *(R+i)
  - int *p = &R[3] → p = (R+3) (p points 12 bytes after R)

**Big Constants**

The MIPS architecture only allows immediate constants to be 16 bits

So how do we get bigger constants?

lui sets the upper 16 bits from the 16 bit immediate field
ori will “or” into the lower 16 bits from the immediate field

How to break your BIG number into the required two 16 bit chunks?

hi = BIG / 64k (e.g. 4,000,000 / 64k = 61)
lo = BIG % 64k (e.g. 4,000,000 % 64k = 2304)
lui $s0, hi
ori $s0, $s0, lo

**Pointer Size vs. Addressable Space**

- Pointers ARE addresses
- Number of unique addresses for N bits is 2^N
- With addresses that are 32 bits long you can address 4G bytes
- With addresses that are 13 bits long you can address 8k bytes
  - that’s 2k words

**Endians?**

Consider the following code

```
foo: .word 0  # foo is a 32 bit int
li $t0, 1  # t0 = 1
la $t1, foo  # t1 = &foo
sb $t0, 1($t1)  # stores a byte at foo+1
```

What is the value of the WORD at foo? In other words what is the value of register t2 after:

```
lw $t2, 0($t1)  # what is in t2?
```

Little Endian → t2 = 0x000000100 → 256

Big Endian → t2 = 0x00010000 → 64k

Consider the following code

```
foo: .word 0  # foo is a 32 bit int
li $t0, 1  # t0 = 1
la $t1, foo  # t1 = &foo
sb $t0, 2($t1)  # stores a byte at foo+2
```

What is the value of the WORD at foo? In other words what is the value of register t2 after:

```
lw $t2, 0($t1)  # what is in t2?
```

Little Endian → t2 = 0x00000001 → 1

Big Endian → t2 = 0x01000000 → 16M
Endians?

Consider the following code

```c
foo: word 0    # foo is a 32 bit int
    li $t0, 1    # t0 = 1
    la $t1, foo  # t1 = foo
    sb $t0, 3($t1) # stores a byte at foo+3
```

What is the value of the WORD at foo? In other words what is the value of register t2 after:

```c
lw $t2, 0($t1)    # what is in t2?
```

Little Endian → $t2 = = 0x01000000 = = 16M
Big Endian → $t2 = = 0x00000100 = = 1

On BOTH machines:

```c
lb $t3, 0($t1)    # t3 = = 'G'
```

### ASCII Chart

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>SP</td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td></td>
</tr>
<tr>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>DEL</td>
</tr>
</tbody>
</table>

### Hex

- Numbers in hex are commonly preceded by 0x:
  - 0x1 = 1, 0x10 = 16, etc.
- Hex is cool because each digit corresponds to 4 bits

```c
0x0 = 0000b, 0x1 = 0001b, 0x2 = 0010b, 0x3 = 0011b
0x4 = 0100b, 0x5 = 0101b, 0x6 = 0110b, 0x7 = 0111b
0x8 = 1000b, 0x9 = 1001b, 0xA = 1010b, 0xB = 1011b
0xC = 1100b, 0xD = 1101b, 0xE = 1110b, 0xF = 1111b
```

So hex to binary is EASY:

```c
0xA = 0101010b
0x0 = 0000000b
0x12 = 00010010b
```

Binary to hex is easy too!

```c
1000011110100000b = 0x5FA0
```

Choosing Registers

- Arguments in $a0-3
- Results in $v0-1
- In a "leaf" function
  - use $a0-7 for everything and you won't have to save and restore anything
  - if you need more then save $a0-7, use them, and then restore at end
- In a "non-leaf" function
  - use $t0-7 for temps that don't need to be saved across calls
  - use $a0-7 for variables that you need across calls
- Always save $a0-7, $ra, $sp if you modify them.
- Use memory pointed to by $sp to save and restore registers, allocate arrays, etc.

### pseudo instructions

- They aren't REAL instructions
- You can think of them as
  - shorthand
  - macros
  - inline functions
  - syntactic sugar
- They are supposed to make your life easier and your programs easier to read
  - move $1,$0 → add $1,$0, $zero
  - la $t0,foo → lui $t0, UPPER16(foo) on $t0, LOWER16(foo)
  - il $t0,23 → add $t0,$zero, 23
  - il $t0,0x230A8 → lui $t0,0x23 on $t0, 0x230A8