

**Fragmentation**

- External
  - Usually occurs when the block size is not fixed
  - Gaps in between blocks that are too small to use
- Internal
  - Usually occurs when the block size is fixed
  - Gaps inside the blocks that are allocated but not used

**Memory Allocation Schemes**

- Free List
  - Use best-fit, first-fit or next-fit to allocate space from list
- Slab
  - Breaks up memory into slabs of similar sized blocks
- Allocator
  - A bitmap can be efficiently used to record which blocks are in use
- Buddy
  - Like slab allocator, but dynamically adjusts to reduce internal fragmentation
- Allocator
  - Large blocks can be split in two and adjacent empty buddy blocks can combine into a bigger block

**Garbage Collection Techniques**

- Reference Counting
  - Keep track of the number of pointers to a memory location
  - Free when there are no more pointers pointing to it
  - Circular data structures wreak havoc on this
- Mark & Sweep
  - From *root set* (any accessible memory) do a depth first search and mark any object you encounter
  - Any garbage won't be marked; so all unmarked nodes can be freed
- Copying
  - Divide memory into two spaces, but only use one at a time
  - When garbage collecting, copy all objects to the other space and compact them in the process
  - Accomplished by using *forwarding pointers*

**MIPS Registers (First Look)**

- There are 32 registers (numbered \$0-\$31) and each can hold 32 bits
- Temporary Registers - \$t0-\$t9 - Used to hold "temporary" values
- Saved Registers - \$s0-\$s7 - Used to hold "saved" values
- Zero Register - \$0 or \$zero - Always 0, even if written to

**MIPS Instructions (A few to get started)**

Instruction	Syntax	Example	Effect
add	add dest src0 src1	add \$s0 \$s1 \$s2	$\$s0 = \$s1 + \$s2$
sub	sub dest src0 src1	sub \$s0 \$s1 \$s2	$\$s0 = \$s1 - \$s2$
addi	addi dest src0 immediate	addi \$s0 \$s0 12	$\$s0 = \$s1 + 12$
lw	lw dest offset(base address)	lw \$t0 4(\$s0)	$\$t0 = \text{Mem}[\$s0 + 4]$
sw	sw src offset(base address)	sw \$t0 4(\$s0)	$\text{Mem}[\$s0 + 4] = \$t0$
bne	bne src0 src1 branchAddress	bne \$t0 \$t1 notEq	if( $\$t0 \neq \$t1$ ) goto notEq
beq	beq src0 src1 branchAddress	beq \$t0 \$t1 equal	if( $\$t0 == \$t1$ ) goto Eq
j	j jumpAddress	j jumpTarget	goto jumpTarget

## MIPS Practice

- Fill in the gaps in the table and try to guess what it is doing

C	MIPS
<pre>a = 17; b = 71; product = 0; while(a != 0) {     product += b;     a--; }  // Mappings a→\$s0 b→\$s1 product→\$s2</pre>	<pre>addi    \$s0, \$0, 17 addi    \$s1, \$0, 71 add     \$s2, \$0, \$0 loop:   beq \$s0, \$0, done         add \$s2, \$s2, \$s1         addi \$s0, \$s0, -1         j  loop done:</pre>
<pre>// dest and source are int[] for(int i=0; i!=n; i++) {     dest[i] = source[i]; }  // Mappings i-&gt;t0 n-&gt;s0 &amp;source-&gt;s1 &amp;dest-&gt;s2</pre>	<pre>add \$t0, \$0, \$0 loop: beq \$t0, \$s0, done       lw  \$t1, 0(\$s1)       sw  \$t1, 0(\$s2)       addi \$t0, \$t0, 1       addi \$s1, \$s1, 4       addi \$s2, \$s2, 4       j  loop done:</pre>
<pre>List last = NULL; while(l != NULL) {     last = l;     l = l-&gt;next; }</pre>	<pre>add     \$t0, \$0, \$0 loop:   beq \$s0, \$0, done         add \$t0, \$s0, \$0         lw  \$s0, 4(\$s0)         j   loop done:</pre>