This mini-reference contains just what you’ll need for assignment 1; see the references on the assignment sheet for much more detail. This reference does not include enough information to do assignment 1’.

1. General-purpose registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>General purpose register</td>
</tr>
<tr>
<td>%ebx</td>
<td>General purpose register</td>
</tr>
<tr>
<td>%ecx</td>
<td>General purpose register</td>
</tr>
<tr>
<td>%edx</td>
<td>General purpose register</td>
</tr>
<tr>
<td>%edi</td>
<td>General purpose register</td>
</tr>
<tr>
<td>%esi</td>
<td>General purpose register</td>
</tr>
</tbody>
</table>

2. Opcodes

- `movl src, dest`: Move a longword integer (four bytes) from `src` to `dest`. Either operand may be in indexed, register, or memory mode, or the source may be immediate mode, but at most one may be memory mode.
  - `movl $4, %eax`: moves the number 4 to register %eax
  - `movl %eax, box`: moves the contents of register %eax to the box at the memory address symbolically specified by “box”
  - `movl $box, %ecx`: moves the address (not the contents) of “box” to register %ecx

- `int $number`: Interrupt: the only interrupt we ever need is $128 ($0x80 in hexadecimal), which means “system call to operating system.” On entry, the number of the system call must be in %eax. A system call generally takes between zero and three arguments. The first argument goes in %ebx, the second in %ecx, and the third in %edx.
  - `int $128`: invokes the Linux system call specified by %eax

- `addl src, dest`: Add (subtracts) the src to (from) the destination, changing only the destination. Addressing modes are as with `movl`. If the result is negative, the sign flag will be set; if it is zero, the zero flag will be set.
  - `subl %eax, %ebx`: subtracts the contents of register %eax from the contents of register %ebx, storing the result in %ebx

- `incl dest`: Increment (add one to) the destination.

- `jmp addr`: Jump to the given address (no dollar sign).
  - `jmp loop`: jumps to the part of the program after the assembler directive `loop`:

- `je addr`: If the zero flag has been set, for instance in a subtraction where the result is zero, jump. Otherwise, don’t jump. The opcode `jz` means the same thing. Among the other possibilities are `jne` (jump if not equal), `jl` (jump if less than, i.e., if negative), and `jge` (jump if greater than or equal).

- `testl op1, op2`: Do a bitwise AND between the two operands. Neither is changed; the only effect is to set condition flags.
  - `testl $1, %eax`: If the 1-bit of %eax is set (i.e., if the number is odd), jump to label `odd`
  - `jne odd`

- `cmpl op1, op2`: Like `subl` but does not change `op2`: sets condition flags.

3. Linux system calls

For the list of all Linux system calls, see the file `/usr/include/asm/unistd.h`. Section 2 of the Unix manual documents system calls. For example, for information on `write`, use the command `man 2 write`. Note that the manual pages assume the C language.

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exit</td>
<td>1 (n)</td>
<td>Exit with code (n), which will be returned to the shell. By convention, 0 means “normal exit,” and anything else means “abnormal.”</td>
</tr>
<tr>
<td>write</td>
<td>4 (fd, buf, size)</td>
<td>Write to an open file (unbuffered). The file descriptor <code>fd</code> should be 1 for standard output. The address <code>buf</code> tells where the bytes are, and <code>size</code> tells how many of them there are.</td>
</tr>
</tbody>
</table>
4. Pseudo-ops and assembler directives

Pseudo-opcodes don't generate machine code, but they tell the assembler to do something.

- **label**: Define the symbol `label` to be the address of what's about to happen, either data or code.

- `.section .data` Begin the section of fixed data.

- `.section .text` Begin the section of read-only, executable instructions (i.e., the program).

- `.globl symbol` Make `symbol` global. The only global symbol you'll need has the magic name `_start`: the linker needs to know where to start execution.

- `.ascii "string"` Put a bunch of bytes in the data section. The bytes are encoded in ASCII (see `man 7 ascii`).

- `.asciz "string"` Same as `.ascii` but add one additional byte, a zero, to mark the end of the string.

- `# comments` Anything on a line after a pound sign is interpreted as a comment (i.e., ignored).

- `symbol = constant` Makes `symbol` a synonym for `constant`

5. Addressing modes

- **name** example

- **immediate** `$42` the number 42

- **direct (memory)** `address` the contents of the box at `address`

- **register** `%eax` the contents of register `%eax`

- **indirect** `( %eax )` treat the contents of register `%eax` as an address; now access the contents of that address

- **base-pointer** `4( %eax )` again treat the contents of register `%eax` as an address; now add the number 4 to that address, and access the contents of the resulting address

See the references for indexed addressing mode. For this assignment, not even base-pointer addressing will be necessary, but indirect addressing will be.