Assembly language:

sto 0
1 –
rclo 0
7 –
* // type in particular value for x
rclo 0 // execute program to compute y
11 –
/
g rtn

- Memory used to store program
- Memory is addressed
- May compute memory addresses – unlike registers
- Registers may be selected – not indexed

struct registers {
    int r0, r1, r2, r3, r4, r5, r6, r7, r8, r9;
}

Machine language:
- Program stored using machine language – key codes
- Central processing unit (CPU) executes the codes
- Program counter (PC) holds address of next instruction to be executed

Assembly language vs. machine language:

<table>
<thead>
<tr>
<th>Address</th>
<th>M/C code</th>
<th>Keys</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>44 0</td>
<td>sto 0</td>
<td>store in r0</td>
</tr>
<tr>
<td>002</td>
<td>1 1</td>
<td>enter</td>
<td>enter 1</td>
</tr>
<tr>
<td>003</td>
<td>30 -</td>
<td></td>
<td>subtract</td>
</tr>
<tr>
<td>004</td>
<td>45 0</td>
<td>rcl 0</td>
<td>r0 to stack</td>
</tr>
<tr>
<td>006</td>
<td>7 7</td>
<td>enter</td>
<td>enter 7</td>
</tr>
<tr>
<td>007</td>
<td>30 -</td>
<td></td>
<td>subtract</td>
</tr>
<tr>
<td>008</td>
<td>20 *</td>
<td></td>
<td>multiply</td>
</tr>
<tr>
<td>009</td>
<td>45 0</td>
<td>rcl 0</td>
<td>r0 to stack</td>
</tr>
<tr>
<td>011</td>
<td>1 1</td>
<td>enter</td>
<td>enter 1</td>
</tr>
<tr>
<td>012</td>
<td>1 1</td>
<td>make it</td>
<td>make it 11</td>
</tr>
<tr>
<td>013</td>
<td>30 -</td>
<td></td>
<td>subtract</td>
</tr>
<tr>
<td>014</td>
<td>10 /</td>
<td></td>
<td>divide</td>
</tr>
<tr>
<td>015</td>
<td>43 32</td>
<td>g rtn</td>
<td>return to calc. mode</td>
</tr>
</tbody>
</table>

- Calculator mode – codes (m/c lang.) sent to ALU
• Program mode – codes (m/c lang.) sent to memory

**Macros**

• Macro processor **m4** – translates symbols into numeric constants
• Macros defined using **define** macro – two arguments

```plaintext
define(sto, 44 0)
define(rcl, 45 0)
define(div, 10)
```

• Macros may have up to 9 arguments
• Specify arguments by $n$
• If macro name followed immediately by ‘(‘, then arguments are present

```plaintext
define(cat, $1$2$3$4$5)
```

call it by:  cat(a, b, c, d, e)

then get:  ab cde
call it by: \text{cat}(a, b, c)

then get: \ a \ b \ c

define(sto, 44 \ 0) \ then \ sto \ always \ refers \ to \ r0

define(sto, '44 \ $1') \ then \ call \ it \ as \ sto(0)

\textbf{Conditionals \ & \ Branching}

- Testing and branching – requires targets
- Targets of branches are labels referring to memory locations
- Need to add memory address for instructions
- Use location counter – memory address of instruction being executed

define(f, 42)
define(g, 43)
define(loc, 0)
define(sto, 'loc: 44 \ $1 \ define('loc', eval(loc + 2))')
define(rcl, 'loc: 45 \ $1 \ define('loc', eval(loc + 2))')
define(div, ‘loc: 10 define(‘loc’, eval(loc + 1)))
define(mul, ‘loc: 20 define(‘loc’, eval(loc + 1)))

define(label, ‘define($1, loc’) )
define(ifeq, ‘loc g 20 define(‘loc’, eval(loc + 2)))
define(gto, ‘loc 22 $1 define(‘loc’, eval(loc + 2)))

ifeq -- test if current value =0
-- if 0, execute next instruction
-- else skip next instruction

gto -- argument is a label
-- assign value of argument to PC (target)

Program to execute equation in a loop:

label(c)    digit(7)
rttn        sub
label(a)    mul
digit(0)     rcl(x_r)
sto(x_r)    digit(1)
label(b)    digit(1)
label(c)    digit(1)
rcl(x_r)    sub
digit(1)     div
digit(1)     pse
sub               rcl(x_r)
ifeq              digit(1)
gto(c)            add
rcl(x_r)          sto(x_r)
digit(1)          gto(b)
sub               rcl(x_r)

Von Neumann Machine

- Contains addressable memory for instructions and data
- ALU executes instructions fetched from memory
- PC register holds address for next instruction to execute
- Defined an instruction cycle

```plaintext
pc = 0;
do {
instruction = memory[pc++];
decode (instruction);
fetch (operands);
execute;
```
Stack Machine

- Stack architecture does not have registers
- Use memory to place items onto stack
- Use load and store operations for moving data between memory and the stack
- Must specify memory address
- MAR – memory address register
- MDR – memory data register
- IR – instruction register holds fetched instruction
- ALU uses top two elements on the stack for all computations
Load Store Machines

- Initially memory limited to few hundred words
- Access time to all locations was the same
- As memory size increased time vs. cost issue arose
- New designs included variable access times
- Register file – high speed memory
- Use load and store instructions between registers and memory
- ALU would function on registers only
- Register file replaces the stack of the stack machine
- SPARC architecture is a load/store machine
add src1, src2, dest

**Assemblers**

- An assembler is a macro processor to translate symbolic programs into machine language programs
- Symbols may be used before they are defined – unlike using m4
- Two pass process
  - Once to determine all symbol definitions
  - Once to apply the definitions
• A symbol followed by a colon defines the symbol to have as its value the current value of the location counter
• The symbol is called a label

define(y_r, r0)
define(x_r, r1)
define(a2_r, r2)
define(a1_r, r3)
define(a0_r, r4)
define(temp_r, r5)

start: mov 0, %x_r
        mov a2, %a2_r
        mov a1, %a1_r
        mov a0, %a0_r
        sub %x_r, %a2_r, %y_r
        sub %x_r, %a1_r, %temp_r
        mul %y_r, %temp_r, %y_r
        sub %x_r, %a0_r, %temp_r
        div %y_r, %temp_r, %y_r