### Machine Language Guide

#### Basic Program

The basic template of a machine language program is shown below.

```assembly
; Program name : XOR Implementation
; Programmer : Jayakanth Srinivasan
; Last Modified : Feb 18 2003

; code segment
load R1, 1 ; Load register R1 with 1
load R2, 0xff ; Load register R2 with 11111111
load R3, [first_number] ; move contents of location labeled first_number into register R3
xor R4, R3, R2 ; flip the 0's and 1's in the first number
store R4, [result] ; store the result in location labeled result
halt ; halt the program.

; data segment
first_number:  db 8
result: db 5
```

#### Instruction Set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 RXY</td>
<td>load R, XY</td>
<td>register[R] := XY</td>
</tr>
<tr>
<td>0 ORS</td>
<td>load R, [S]</td>
<td>register[R] := memory[register[S]]</td>
</tr>
<tr>
<td>0 ORS</td>
<td>store R, [S]</td>
<td>memory[register[S]] := register[R]</td>
</tr>
<tr>
<td>4 ORS</td>
<td>move S, R</td>
<td>register[S] := register[R]</td>
</tr>
<tr>
<td>7 RST</td>
<td>or R, S, T</td>
<td>register[R] := register[S] OR register[T]</td>
</tr>
</tbody>
</table>

- **bitwise AND**

- **bitwise OR**

- **bitwise XOR**

- **rotate right by X times**
  - register[R] := register[R] ROR X

- **jumps**
  - jmp EQ R=R0, XY, PC:=XY if R=R0
  - jmp XY, PC:=XY
  - jmpLE R<=R0, X, PC:=XY if R<=R0

- **halt**
  - halt program

The opcode is the first nibble (higher four bits of the first byte) and the three parts of the operand are the second, third and fourth nibble.

#### Assembler Syntax

##### Label

A label is a sequence of letters, decimal digits and special characters, but it may not start with a digit.

##### Instruction

An instruction starts with a mnemonic, followed by the operands. It has to be one of the 16 instructions listed in the previous section.

##### Comment

A comment starts after a semicolon ‘;’ and ends at the end of the line. Any character is allowed after the ‘;’.

##### Numbers

A number can be a decimal number, a binary number or a hexadecimal number.

- A decimal number is a sequence of decimal digits (‘0’ up to ‘9’). It may start with a ‘-’ to indicate the number is negative. It may end with a ‘d’ to emphasize that the number is decimal.
- A binary number is a sequence of binary digits (‘0’ and ‘1’) and ending with a ‘b’.
- A hexadecimal number is a sequence of hexadecimal digits (‘0’ up to ‘F’) and ending with a ‘h’.
A hexadecimal number can be written in 3 ways:
- C-style: The number starts with '0x', followed by a sequence of hexadecimal digits ('0' up to '9' and 'A' up to 'F').
- Pascal-style: The number starts with '$', followed by a sequence of hexadecimal digits ('0' up to '9' and 'A' up to 'F'), but it may not start with a letter. This sequence is followed by an 'h'. A number can always be made to start with a decimal digit by prefixing the number with a '0', so ABh is written as 0ABh.
- Assembler-style: The number is a sequence of hexadecimal digits ('0' up to '9' and 'A' up to 'F'), but it may not start with a letter. This sequence is followed by an 'h'.

Spaces are not allowed within a number.

**Remarks**

All identifiers (labels and mnemonics) and (hexadecimal) numbers are case-insensitive. This means that load, Load, LOAD and lOaD are all the same and so are 0xAB, 0Xab and 0XAB.

This editor uses syntax-highlighting:
- **keywords**: load, store, addi
- **numbers**: -123, 0x10, 1101011b
- **comments**: ;this is a comment
- **syntax errors**: 12A3, -0x10, 1+1

**Mnemonics and operand combinations**

* data byte


db dataitem_1, dataitem_2, ..., dataitem_n

- Puts data directly into the memory.
- A dataitem can be either a number or a string.
- An unlimited number of dataitems can be specified.

**Examples**:

db 1,4,9,16,25,36
db "Hello world",0

* origin

**org** adr
- The next code starts at address adr.
- Address adr must be a number.

- Different fragments of code are not allowed to overlap.

**Examples**:

org 60h
load R0,2 ;put this instruction at address $60

**immediate load**

load reg,number
load reg,label
- Assign the immediate value (number or address of label) to register reg.

**Examples**:

load R4,8
load R9,Label_of_something

**direct load**

load reg,[adr]
- Assign the memory contents at address adr to register reg.
- Address adr can be a number or a label.

**Examples**:

load R4,[8]
load R9,[Label_of_something]

**indirect load**

load reg,[reg2]
- Assign the memory contents of which register reg2 holds the address to register reg1.

**Example**:

load R4,[R8]

**direct store**

store reg,[adr]
- Put the value of register reg at memory location adr.
- Address adr can be a number or a label.

**Examples**:

store R4,[8]
store R9,[Label_of_something]

**indirect store**

store reg,[reg2]
- Put the value of register reg1 at memory location of which register reg2 holds the address.

**Example**:

store R4,[R8]

**move**

move reg1,reg2
- Assign the value of register reg2 to register reg1.

**Example**:

move R4,R8

**integer addition**

addi reg1,reg2,reg3
- Assign the integer, 2-complement sum of register reg2 and register reg3 to register reg1.
  Example:
  - `addi R7,R1,R2`

**Floating point addition**

- Assign the floating-point sum of register reg2 and register reg3 to register reg1.
  Example:
  - `addf R7,R1,R2`

**Bitwise**

- Assign the bitwise OR of register reg2 and register reg3 to register reg1.
  Example:
  - `or R7,R1,R2`

- Assign the bitwise AND of register reg2 and register reg3 to register reg1.
  Example:
  - `and R7,R1,R2`

- Assign the bitwise exclusive OR of register reg2 and register reg3 to register reg1.
  Example:
  - `xor R7,R1,R2`

**Rotate right**

- Rotate register reg to the right for num number of times.
  Example:
  - `ror RC,3`

**Unconditional jump**

- Jump to address adr.
  Example:
  - `jmp 42h`
  - `jmp Label_to_some_code`

**Jump when equal**

- Jump to address adr when register reg is equal to register R0.
  Example:
  - `jmpEQ RC=R0,adr`

**Jump when less or equal**

- Jump to address adr when register reg is less than or equal to register R0.
  Example:
  - `jmpLE RC<=R0,adr`

Notes:
This handout was put together with information from the help section of the Simple Simulator developed at [http://wwwes.cs.utwente.nl/software/simpsim/](http://wwwes.cs.utwente.nl/software/simpsim/)