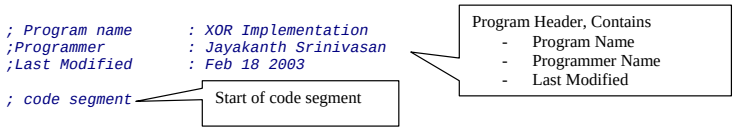
Machine Language Guide

Basic Program

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




```
      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
```

Start of data segment

```
first_number:     db 8
result:           db 5
```

Instruction Set

Opcode		Instruction		Operation
2	RXY	**load**	**R,XY**	register[R]:=XY
1	RXY	**load**	**R,[XY]**	register[R]:=memory[XY]
3	RXY	**store**	**R,[XY]**	memory[XY]:=register[R]
D	0RS	**load**	**R,[S]**	register[R]:=memory[register[S]]
E	0RS	**store**	**R,[S]**	memory[register[S]]:=register[R]
4	0RS	**move**	**S,R**	register[S]:=register[R]
5	RST	**addi**	**R,S,T**	register[R]:=register[S]+register[T] **integer add**
6	RST	**addf**	**R,S,T**	register[R]:=register[S]+register[T] **floating-point add**
7	RST	**or**	**R,S,T**	register[R]:=register[S] OR register[T] **bitwise OR**
8	RST	**and**	**R,S,T**	register[R]:=register[S] AND register[T] **bitwise AND**
9	RST	**xor**	**R,S,T**	register[R]:=register[S] XOR register[T] **bitwise eXclusive OR**
A	R0X	**ror**	**R,X**	register[R]:=register[R] ROR X **Rotate Right register R for X times**
B	RXY	**jmpEQ**	**R=R0,XY**	PC:=XY, if R=R0
	0XY	**jmp**	**XY**	PC:=XY
F	RXY	**jmpLE**	**R<=R0,X**	PC:=XY, if R<=R0
C	000	**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 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').
  - 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'. 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.
- 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, 11001011b`
- 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   reg1,[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  reg1,[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

```
addf   reg1,reg2,reg3
```

- Assign the floating-point sum of register reg2 and register reg3 to register reg1.

Example:

```
    addf   R7,R1,R2
```

bitwise or

```
or     reg1,reg2,reg3
```

- reg1 := reg2 OR reg3

Example:

```
    OR     R7,R1,R2
```

bitwise and

```
and    reg1,reg2,reg3
```

- reg1 := reg2 AND reg3

Example:

```
    AND    R7,R1,R2
```

bitwise exclusive or

```
xor    reg1,reg2,reg3
```

- reg1 := reg2 XOR regr3

Example:

```
    XOR    R7,R1,R2
```

rotate right

```
ror    reg,num
```

- Rotate register reg to the right for num number of times.

Example:

```
    ror    RC,3
```

jump when equal

```
jmpEQ  reg=R0,adr
```

- Jump to address adr when register reg is equal to register R0.
- Address adr can be a number or a label.

Examples:

```
    jmpEQ  R7=R0,42h
    jmpEQ  R2=R0,Label_to_some_code
```

jump when less or equal

```
jmpLE  reg<=R0,adr
```

- Jump to address adr when register reg is less than or equal to register R0.
- Address adr can be a number or a label.

Examples:

```
    jmpLE  R7<=R0,42h
    jmpLE  R2<=R0,Label_to_some_code
```

unconditional jump

```
jmp    adr
```

- Jump to address adr.
- Address adr can be a number or a label.

Examples:

```
    jmp    42h
    jmp    Label_to_some_code
```

stop program

```
halt
```

- Stop the execution of the program.

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/