COMP360 Assembler

Write an assembler that reads the source code of an assembly program (for an imaginary machine) and displays the machine language for that program.

The following assembler instructions must be supported.

Instruction	opcode	Length	Format	Example	
add	0	2	0	`add  R1, R2, R3`	R3 = R1 + R2
sub	1	2	0	`sub  R1, R2, R3`	R3 = R1 - R2
mult	2	2	0	`mult R1, R2, R3`	R3 = R1 * R2
div	3	2	0	`div  R1, R2, R3`	R3 = R1 / R2
load	4	4	1	`load  R1,addr[R7]`	R1 ← memory value
laddr	5	4	1	`laddr R1,addr[R7]`	R1 ← address
store	6	4	1	`store R1,addr[R7]`	R1 → memory
call	7	4	2	`call addr`	push return addr on stack and jump to addr
rtn	8	1	4	`rtn`	pop address from stack and jump to there
jump	9	4	2	`jump addr[R7]`	jump to address
jz	A	4	1	`jz   R1,addr[R7]`	jump to address only if R1 is zero
jneg	B	4	1	`jn   R1,addr[R7]`	jump to address only if R1 is negative
push	C	1	3	`push R1`	push R1 on stack
pop	D	1	3	`pop  R1`	pop stack into R1
lpsw	E	1	3	`lpsw R1`	load PSW into R1
spsw	F	1	3	`spsw R1`	store R1 into PSW
data		4	5	`data  123`	created a data word

Machine language instructions have the following formats:

0: arithmetic

opcode	Reg 1	Reg 2	Reg 3
4 bits	4 bits	4 bits	4 bits

1: Load, store, conditional jump

opcode	Reg	index	addr
4 bits	4 bits	4 bits	20 bits

2: call, jump

opcode	unused	index	addr
4 bits	4 bits	4 bits	20 bits

3: push, pop, lpsw, spsw

opcode	Reg
4 bits	4 bits

4: rtn

opcode	unused
4 bits	4 bits

5: data

integer data value
32 bits

A comment in the assembly program is indicated by an asterisk, "*", in the first column of the source code. Labels must start in the first column. Your assembler does not have to detect every possible error in the assembler program, but it should produce a reasonable error message if an undefined label is used.

For a maximum score of 95 points, your program can ignore all index registers. The presence of an [index register] may or may not be considered an error. The index field of all instructions must be set to zero if index registers are not implemented. For a maximum score of 100, all memory addressing instructions must support an optional index register.

Example output:

```
address  machine    source

                  * Example assembler program for COMP360
                  *
    0  58000034        laddr R8,cat
    4  41800030        load  R1, dog[R8]
    8  42800038        load  R2, goat[R8]
    c       123 more   add   R1, R2, R3
    e      2324        mult  R3, R2, R4
   10  b400000c        jn    R4, more
   14      1415        sub   R4, R1, R5
   16  a300001c        jz    R3, nodiv
   1a      3536        div   R5, R3, R6
   1c  66000025 nodiv store R6, addr
   20        c6        push  R6
   21  7000002d        call  mthd
   25        fa addr   spsw  R10
   26       a27        add   R10, R2, R7
   28        e7        lpsw  R7
   29  9000000c        jump  more
                  *  example method
   2d       128 mthd   add   R1, R2, R8
   2f        80        rtn
                  *  data
   30         1 dog    data 1
   34         0 cat    data  0
   38       801 goat   data  2049
```

Hints:
You may wish to create an object to hold the symbol table entries. This would contain:

- opcode, address or other value of the symbol
- length of instruction in bytes
- format of the machine instruction

A hash table works well to hold the symbol table. Symbol table entries can be keyed by the mnemonic or address label.

The assembler can be written to use one or two passes. After the first pass, a two pass assembler closes the input file and then reads the input again. The goal of the first pass is to create a symbol table containing the address of all symbols. The second pass creates the output. It uses the symbol table to get the value of all names used in the program.

General outline for pass 1.

- Initialize the current address to zero
- Read a line of assembler
- Split the line into tokens
- If label, put label and current address in symbol table
- Get mnemonic and find in symbol table
- Add instruction length to current address
- repeat until end of file

General outline for pass 2

- Initialize the current address to zero
- Read a line of assembler
- Split the line into tokens
- Get mnemonic information from symbol table
- Get additional fields and create the machine language
- Display the results

The following Java classes and methods may be useful in writing your assembler. Similar C++ methods exist.

java.util.HashMap
java.util.StringTokenizer
Integer.toUnsignedString(intNum, 16)