μMachine/μCode

For the final part of the CS450 project, you are to complete a fully functional μPascal compiler. Since it would be impractical to have you generate assembly code for a real machine (with all the intricacies of the target machine), we have created a virtual machine that has been designed specifically for a μPascal compiler. The μMachine (and is associated assembly language μCode) greatly simplifies the task of code generation while still requiring you to handle many of the problems faced by other compiler writers.

At this point in time, you should have written a scanner and parser for μPascal, should be working on the symbol table and should be thinking about semantic processing and code generation. The following information about the μMachine and μCode is provided to assist you in your design and implementation of the remaining parts of the μPascal compiler project:

<u>μMachine Specification:</u>

The μMachine is a virtual machine (simulated by a program) with the following hardware characteristics:

- Separate instruction space (for assembly code) and RAM (for data storage/retrieval)
- 10 general purpose registers (D0 - D9)
- Special stack pointer register (SP)

The μMachine is a stack-based machine; all memory is allocated/deallocated on the data stack residing in RAM:

- The data stack supports types: Integer, Float/Fixed, Strings.
- All data types have the same size of 1.
- The data stack grows upwards (starts at 0, pushes increment the SP, pops decrement the SP)

<u>Supported Data Types</u>

<u>Integer</u>:
As defined in the μPascal tokens document. Size: 1.

<u>Float/Fixed</u>:
Numbers represented as floating point or fixed point are supported and have a size of 1. Specifically, it will accept all floats/fixed that scanf(“%f”) will from the C programming language.

Legal Examples: 1.23, -1.3, -8.4e10, 3.0e-4, -4.21412e+2
Illegal examples: 4e10, 4.e12

String:
String literals supported and are of size 1. They are defined on a single line directly followed by a new line. Supports the following escape sequences:

\n => New Line
\r => Carriage Return
\t => Horizontal Tab
\v => Vertical Tab
\\ => Backslash

No other escape sequences are supported and none are needed for standard characters(except backslash).

μCode Specifications:

μCode (assembly language) is based on *QUADRUPLES*. Each quadruple consists of an opcode and up to three operands.

Opcodes:

At present there are 70 valid opcodes (instructions) in the μCode assembly language...all are detailed on the uCode Quick Reference page at the end of this document.

Operands Address Modes:

MODE	**FORM**	**SAMPLE**	**DESCRIPTION**
IMMEDIATE	**#***d*	**#4**	Integer literal value
IMMEDIATE FLOAT	**#***f*	**#-1.2**	Float literal value
IMMEDIATE STRING	**#”***s***”**	**#”abc\n b”e”**	String literal value defined on a single line.
REGISTER	**D***n*	**D6**	Contents of register ***n***
INDEXED	*m***(D***n***)**	**5(D3)**	Address = **D*n*** + ***m***
INDIRECT	**@***m***(D***n***)**	**@7(D1)**	Address = Contents of (**D*n*** + ***m***)
STACK REGISTER	**SP**	**SP**	Stack pointer
STACK INDEXED	*m***(SP)**	**6(SP)**	Address = **SP** + ***m***
STACK INDIRECT	**@***m***(SP)**	**@2(SP)**	Address = Contents of (**SP** + ***m***)

Labels:

Labels are specified with either **L***n***:** (defining a label) or **L***n* (using a label).

µCode Quick Reference Page

INSTRUCTION	DESCRIPTION
HLT	Terminate program execution
RD dst	Read an integer value from the keyboard into dst
RDF dst	Read a float value from the keyboard into dst
RDS dst	Read a string value from the keyboard into dst. Quotations not needed.
WRT src	Write a value in src to the screen
WRTS	Performs: POP A WRT A
WRTLN src	Write a value in src with a new line appended to the screen.
WRTLNS src	Performs: POP A WRTLN A
MOV src dst	Performs: dst <- src
NEG src dst	Performs: dst <- -src (Integer)
ADD src1 src2 dst	Performs: dst <- src1 + src2 (Integer)
SUB src1 src2 dst	Performs: dst <- src1 - src2 (Integer)
MUL src1 src2 dst	Performs: dst <- src1 * src2 (Integer)
DIV src1 src2 dst	Performs: dst <- src1 / src2 (Integer)
MOD src1 src2 dst	Performs: dst <- src1 % src2 (Integer)
NEGF src dst	Performs: dst <- -src (Float or Fixed)
ADDF src1 src2 dst	Performs: dst <- src1 + src2 (Float or Fixed)
SUBF src1 src2 dst	Performs: dst <- src1 - src2 (Float or Fixed)
MULF src1 src2 dst	Performs: dst <- src1 * src2 (Float or Fixed)
DIVF src1 src2 dst	Performs: dst <- src1 / src2 (Float or Fixed)
PUSH src	Push src onto the data stack
POP dst	Pop the stack top into dst
NEGS	Performs: POP A PUSH –A (Integer)
ADDS	Performs: POP A POP B PUSH B + A (Integer)
SUBS	Performs: POP A POP B PUSH B – A (Integer)
MULS	Performs: POP A POP B PUSH B * A (Integer)
DIVS	Performs: POP A POP B PUSH B / A (Integer)
MODS	Performs: POP A POP B PUSH B % A (Integer)
NEGSF	Performs: POP A PUSH –A (Float or Fixed)
ADDSF	Performs: POP A POP B PUSH B + A (Float or Fixed)
SUBSF	Performs: POP A POP B PUSH B – A (Float or Fixed)
MULSF	Performs: POP A POP B PUSH B * A (Float or Fixed)
DIVSF	Performs: POP A POP B PUSH B / A (Float or Fixed)
CASTSI	Performs: POP A PUSH (float)A
CASTSF	Performs: POP A PUSH (int)A
Ln:	Drop a label at the current line
ANDS	Performs POP A POP B PUSH B and A
ORS	Performs POP A POP B PUSH B or A
NOTS	Performs POP A PUSH not A
CMPEQS	Performs POP A POP B PUSH B = A (Integer)

```
CMPGES                      Performs  POP A  POP B  PUSH B >=  A  (Integer)
CMPGTS                      Performs  POP A  POP B  PUSH B >   A  (Integer)
CMPLES                      Performs  POP A  POP B  PUSH B <=  A  (Integer)
CMPLTS                      Performs  POP A  POP B  PUSH B <   A  (Integer)
CMPNES                      Performs  POP A  POP B  PUSH B <>  A  (Integer)

CMPEQSF                     Performs  POP A  POP B  PUSH B =   A  (Float or Fixed)
CMPGESF                     Performs  POP A  POP B  PUSH B >=  A  (Float or Fixed)
CMPGTSF                     Performs  POP A  POP B  PUSH B >   A  (Float or Fixed)
CMPLESF                     Performs  POP A  POP B  PUSH B <=  A  (Float or Fixed)
CMPLTSF                     Performs  POP A  POP B  PUSH B <   A  (Float or Fixed)
CMPNESF                     Performs  POP A  POP B  PUSH B <>  A  (Float or Fixed)

BRTS   Ln                   Performs  POP A  BEQ A #1 Ln
BRFS   Ln                   Performs  POP A  BEQ A #0 Ln

BR     Ln                   Branch to label n
BEQ    src1  src2  Ln       Branch to label n if src1 =  src2     (Integer)
BGE    src1  src2  Ln       Branch to label n if src1 >= src2     (Integer)
BGT    src1  src2  Ln       Branch to label n if src1 >  src2     (Integer)
BLE    src1  src2  Ln       Branch to label n if src1 <= src2     (Integer)
BLT    src1  src2  Ln       Branch to label n if src1 <  src2     (Integer)
BNE    src1  src2  Ln       Branch to label n if src1 <> src2     (Integer)

BEQF   src1  src2  Ln       Branch to label n if src1 =  src2     (Float or Fixed)
BGEF   src1  src2  Ln       Branch to label n if src1 >= src2     (Float or Fixed)
BGTF   src1  src2  Ln       Branch to label n if src1 >  src2     (Float or Fixed)
BLEF   src1  src2  Ln       Branch to label n if src1 <= src2     (Float or Fixed)
BLTF   src1  src2  Ln       Branch to label n if src1 <  src2     (Float or Fixed)
BNEF   src1  src2  Ln       Branch to label n if src1 <> src2     (Float or Fixed)

CALL   Ln                   Performs:  PUSH PC  BR Ln
RET                         Performs:  POP  PC

PRTS                        Prints out stack addresses and values – Doesn’t affect state of machine.
PRTR                        Prints out registers – Doesn’t affect state of machine.
```