A brief description of P-stack machine from CS-UK used in cs441
Additional files:

- api.c contains emulator for P-stack machine
- api.d contains disassembler
- apm.h contains internal representations for opcodes

```
/* memory cells are assumed to be of type WORD */
typedef union word {
int Integer;
float Real;
} WORD;

#define STORAGE (50*1024)
```

There is some redundancy in these instructions. Bar and Jmp have the same semantics, Arrow and Jmp_If_True have the same semantics, Do and Jr_if_False have the same semantics. Bar and Jmp are the same, as are I_Variable and R_Variable. Choose the one which you are most comfortable with.

Also, in emitting instructions, the opcode should be represented in all capital letters, as in apm.h. Note, it is a good idea to change the parameters of the emitter routines to type WORD.

operation	opcode	operands	types	description
I_Add	1	0		As though, x := pop(); y := pop(); push(x+y);
And	2	0		As though, x := pop(); y := pop(); push(x and y); x and y = 1 if both are non-zero and 0 elsewise
Arrow	3	1	Integer	conditional goto to absolute address jmp to location given by operand if (tos) non-zero
I_Assign	4	1	Integer	the stack is of form $Dest_1\ Dest_2\ \ldots Dest_k$ $Val_1\ Val_2\ Val_3 \ldots Val_k$ where k is the integer and the values are moved into the destinations.
Bar	5	1	Integer	unconditional goto to absolute address
Call	6	2	Int Int	the first integer is the level, the second is the address to jump to builds a stack frame and jumps.
I_Constant	7	1	Integer	Push the argument
I_Divide	8	0		x := pop(); y := pop(); push(y / x)
EndProc	9	0		restore pc and b from frame
EndProg	10	0		exit program
I_Equal	11	0		pop top two elements push 1 if equal (as integers) or zero if not
Fi	12	0		generates run-time error and exits
I_Greater	13	0		see I_Equal
Index	14	2	Int Int	top of stack is index Next element is address of array first argument is upper bound second argument is line #. elements of stack are popped, and if index is between 1 and bound, a reference to the element is pushed. Otherwise a run-time error results
I_Less	15	0		See I_Equal
I_Minus	16	0		Negate top of stack.
I_Modulo	17	0		See I_Divide
I_Multiply	18	0		See I_Add

Table 1: Commands from PL

Not	19	0		pop(), push logical negation of it.
Or	20	0		see I_And
Proc	21	2	Int Int	Allocate local memory, set pc, the first integer is the amount of memory needed, the second the new pc.
Prog	22	2	Int Int	Allocate global memory, initialize stack pointer, and outermost frame. Parameters are as in Proc
I_Read	23	1	Integer	Read the given # of words (tos) is address into which first item is put (tos-1) is addr. into which 2nd item is put (tos-k+1) is addr. into which kth item is put. the k addresses are popped after they are used.
I_Subtract	24	0		As I_Divide.
I_Value	25	0		The top of stack holds an address. It is popped, and the integer held in the address is pushed, i.e., tos is dereferenced.
I_Variable	26	2	Int Int	the parameters are level and offset, we find the addr. of the cell indicated and push this addr, i.e. the cell is referenced.
I_Write	27	1	Integer	The parameter is # of items to write. The stack should hold this many values tos holds the LAST thing to be written.

Table 2: More Commands from PL

operation	opcode	operands	types	description
I_To_R	28	0		x := pop(); convert to real; push result.
R_Add	31	0		As I_Add, but stack should have real values, and result is real.
R_Assign	34	1	Integer	As I_Assign, but reals.
R_Constant	37	1	Real	pushes real value on stack
R_Divide	38	0		see R_Add and I_Add
R_Equal	41	0		see R_Add and I_Equal
R_Greater	43	0		...
R_Less	45	0		...
R_Minus	46	0		...
R_Multiply	48	0		...
R_Read	53	1	Integer	...
R_Subtract	54	0		...
R_Value	55	0		...
R_Variable	56	2	Int Int	... This is not necessary, a real variable is referenced on our machine exactly like a integer variable, but I thought that you might be more comfortable with this
R_Write	57	1	Int	See R_Add and I_Write
R_To_I	58	0		inverse of I_To_R
Swap	59	0		x:= pop(); y:= pop(); push(x); push(y);
Do	60	1	Integer	same as Jr_If_False, provided for readers of Per Brinch Hansen's book.
Jmp_if_True	61	1	Integer	if (tos) is non-zero goto given in parameter, else perform next instruction.
Jmp_if_False	62	1	Integer	if (tos) is zero goto given in parameter, else perform next instruction.
Jr_if_True	63	1	Integer	if (tos) is non-zero goto given by pc + parameter, else perform next instruction.
Jr_if_False	64	1	Integer	if (tos) is zero goto given by pc + parameter, else perform next instruction.
Jmp	65	1	Integer	Jump to address given by parameter.
Jr	66	1	Integer	Relative jump, offset given by parameter.

Table 3: New Commands