Limited Basic Machine Application Interface Directions

LBMAID

`Shade H. St Claire`

LBMAID is the virtual machine/bytecode to be used in my prospected project implementing a subset of BBC BASIC. It is a limitted form of a much larger register machine first developed based on the Intel 8086. It thus inherets some of the quirks associated with the intel line of machines while (hopefully) being compact enough to be easily implemented over the course of a half-week.

LBMAID consists of 9 which various instructions reference both indirectly or directly. Being a 32 bit bytecode designed to run on the test machine (a Toshiba satellite from 2009 running Windows 10 to... arguable success) the registers too are 32 bit. These registers are as so:

○	`0000 ACC`	:	The accumulator.
○	`0001 B`	:	General purpose/array register.
○	`0010 C`	:	Counter register.
○	`0011 XI`	:	X index register.
○	`0100 YI`	:	Y index register.
○	`0101 SP`	:	Stack pointer.
○	`0110 BP`	:	Stack base pointer.
○	`0111 FLG`	:	Flags register.
○	`1000 IP`	:	Instruction pointer.

The use of these registers will become clear throughout the continuation of this document. We can begin the specifications ahead with that of a brief description of MAID-ASM, the assembly language assumed for the rest of this document. Its instructions are of the following form:

```
instr = (instruction | directive) *(arguments *SP) comment
comment = ';' *WHATEVER NL | '/*' *WHATEVER '*/'
```

An example might be:

```
LDI ACC 48h
PUTCHAR
LDI ACC 49h
PUTCHAR
LABEL .loop
GETCHAR
PUTCHAR
CMP ACC 3h /* 3h is Ctr-C */
JNZ .loop
```

Which would print '`HI`' and echo user input until Ctr-C is pressed. LBMAID does make quite a few concessions for the sake of time, for example many expected syntaxes are not to be present in this first version of LBMAID, i.e. **`identifier:`** for labels is not surported, nor is: `LOD ACC [B + 5]` for pointer dereference. Instead we might have: `LABEL identifier` or `LOD ACC +B 5` .

First we will define the various directives:

`LABEL` *name* : Defines a label with the name *name* which is at compile time replaced with the current address in all immediate/memory calculations.
`ORG` *number* : Sets the current address to *number*.
`$` : Replaced by the current address at compile time.
`MEM` *amount* : Sets the amount of allocated memory to be at least *amount*.
`DB [`*bytes*`]` : Defines the bytes at the current address as though an instruction had evalutated to them.
`DW [`*words*`]` : Similar to `DB` except that values are in little endian double-byte format.
`DD [`*dwords*`]` : Similar to `DB` except that values are in little endian quat-byte format.

Before the instructions are defined it bears mentioning that the header consists of a quad-byte which determines the initial allocation memory (minimum) and is defined by the assembler automatically. This means all code is offset in binary by four bytes. It also bears mentioning that code is initially loaded at address 0x00000000 without regard for stack size/needs. Considering that the stack must grow downwards it is highly advised to move the code to an appropriate spot at runtime using a block copy command (`REP MOVS BYTE`).

Now to define the instructions. These instructions are of the form:

```
NAME $MAX_ARG_NUM (ARGUMENT_TYPES)
```

Where MAX_ARG_NUM is the maximum number of arguments which may be passed to the instruction where ARGUMENT_TYPES is a list of argument type signiatures. Each argument may be one of the following:

- ○ `R` : Any register.
- ○ `I` : Immediate.
- ○ `M` : Memory.
- ○ `SHORT` : 8 bit immediate.
- ○ `NEAR` : 16 bit immediate.
- ○ `FAR` : 32 bit immediate.
- ○ `BYTE` : The keyword `BYTE`.
- ○ `WORD` : The keyword `WORD`.
- ○ `DWORD` : The keyword `DWORD`.
- ○ `NULL` : An empty argument.
- ○ `A` : Accumulator.
- ○ `B` : B register.
- ○ `C` : C register.
- ○ `XI` : XI register.
- ○ `YI` : YI register.
- ○ `SP` : SP register.
- ○ `BP` : BP register.
- ○ `F` : FLG register.
- ○ `IP` : IP register.

Memory access is also of one of the following modes:

- ○ `0000 immediate` : Value at the address of the provided immediate.
- ○ `0001 B` : Value pointed to by B.
- ○ `0010 XI` : Value pointed to by XI.
- ○ `0011 YI` : Value pointed to by YI.
- ○ `0100 BP` : Value pointed to by BP.
- ○ `0101 + B immediate` : Value at B + the immidiate provided.
- ○ `0110 + XI immediate` : Value at XI + the immediate provided.
- ○ `0111 + BP immediate` : Value at BP + the immediate provided.
- ○ `1000 ⟨ B` : Value at B ⟨⟨ 1.
- ○ `1001 ⟨⟨ B` : Value at B ⟨⟨ 2.

Finally we may consider the instructions of LBMAID. They are as so:

00	ADD	(A, I)
01	ADV	(C, I)
02	ADD	(A, M)
03	ADD	(A, B)
04	AND	(A, I)
05	AND	(A, M)
06	AND	(A, B)
07	CALL	(SHORT)
08	CALL	(NEAR)
09	CALL	(FAR)
0A	CMP	(A, B)
0B	CMP	(A, I)
0C	CMP	(A, M)
0E	CMPS	(BYTE)
0F	CMPS	(WORD)
10	CMPS	(DWORD)
11	CURPOS	()
12	FADD	(R, R)
13	FSUB	(R, R)
14	GETCHAR	()
15	HLT	()
16	JC	(SHORT)
17	JNC	(SHORT)
18	JZ	(SHORT)
19	JNZ	(SHORT)
1A	JO	(SHORT)
1B	JS	(SHORT)
1C	JMP	(SHORT)
1D	JMP	(NEAR)
1E	JMP	(FAR)
1F	LODS	(BYTE)
20	LODS	(WORD)
21	LODS	(DWORD)
22	LOOP	(SHORT)
23	LOOPZ	(SHORT)
24	LOOPNZ	(SHORT)
25	LOD	(A, M)
26	LOD	(C, M)
27	LOD	(R, M)
28	LDI	(A, I)
29	LDI	(R, I)
2A	MOV	(A, R)
2B	MOV	(R, R)
2C	MOV	(SP, BP)
2D	MOV	(BP, SP)
2E	MOVS	(BYTE)
2F	MOVS	(WORD)
20	MOVS	(DWORD)
21	NOT	(A)
32	OR	(A, I)
33	OR	(A, R)
34	POP	(A)

35	POP	(M)
36	POP	(BP)
37	POPF	()
38	PUSH	(A)
39	PUSH	(I)
3A	PUSH	(M)
3B	PUSH	(BP)
4C	REP/REPZ	()
3D	REPNZ	()
3E	RET	(SHORT)
3F	SCAS	(BYTE)
40	SCAS	(WORD)
41	SCAS	(DWORD)
42	STO	(M, A)
43	STO	(M, I)
44	STOS	(BYTE)
45	STOS	(WORD)
46	STOS	(DWORD)
47	SUB	(A, I)
48	SUV	(C, I)
49	SUB	(A, M)
4A	SUB	(A, B)
4B	TEST	(A, B)
4C	TEST	(A, I)
4D	TEST	(A, M)
4E	ZERO	(R)
4F	XOR	(A, B)
50	PUTCHAR	()
51	PUSHF	()

An example program follows:

```
LDI XI message
/* Size optimized instructions to load string length into C and increment XI to
    the appropriate length */
LOD C % XI ; % preceeds mem arguments for clarity
LODS DWORD
LABEL .loop
    LODS BYTE
    PUTCHAR
LOOP .loop
HLT
LABEL message
DD 0Ch% ; % also proceeds lists
DB 'Hello World!'%
```