Appendix A.
Programming LMC / TBC

A.1 Operation Codes

Operation codes in gray are for future implementation.

Op Code	Mnemonic	Meaning
0xx	HLT <addr>	**Halt.** Instruction execution stops. Operating the "reset" switch will restart the program from location zero. The mnemonic COB is accepted as a synonym for HLT. The address field is ignored.
1xx	ADD <addr>	**Add.** The contents of memory at address <addr> are added to the accumulator. An Overflow machine-check occurs if the result is greater than 2047 or less than -2048; the truncated result is stored in the accumulator.
2xx	SUB <addr>	**Subtract.** The contents of memory at address <addr> are subtracted from the accumulator. An *Overflow* machine-check occurs if the result is greater than 2047 or less than -2048; the truncated result is stored in the accumulator.
3xx	STO <addr>	**Store accumulator.** The contents of the accumulator are stored in memory at address <addr>; the previous contents of memory at <addr> are lost. The mnemonic STA is accepted as a synonym for STO.
4xx		Not used
5xx	LDA <addr>	**Load accumulator.** The contents of memory at address <addr> are loaded into the accumulator; the previous contents of the accumulator are lost.

6xx	BR <addr>	**Branch unconditionally.** The value of <addr> is stored in the program counter, causing the next instruction to be fetched from that location. The mnemonic BRA is accepted as a synonym for BR.
7xx	BRZ <addr>	**Branch if zero.** If the accumulator holds the value zero, the value of <addr> is stored in the program counter, causing the next instruction to be fetched from that location. If the accumulator is non-zero, the program counter is not modified and the next instruction will be fetched in sequence. The accumulator is not modified.
8xx	BRP <addr>	**Branch if positive.** If the sign bit of the accumulator is zero, the value of <addr> is stored in the program counter, causing the next instruction to be fetched from that location. If the sign bit is one, the program counter is not modified and the next instruction will be fetched in sequence. The accumulator is not modified. Note that a value of zero in the accumulator will cause the branch to be taken because the sign bit of the number zero is zero.
901	IN	**Input.** A number is requested from the input subsystem and stored in the accumulator. The previous contents of the accumulator are lost. The input subsystem will not deliver values greater than 2047 nor less than -2048. The mnemonic INP is accepted as a synonym for IN.
902	OUT	**Output.** The contents of the accumulator are copied to the output subsystem as a decimal value, followed by the newline character. The accumulator is not changed.

Axx	CALL <addr>	**Subprogram call.** The number at the highest memory address is decremented, the program counter is stored at the address pointed by the contents of the highest memory address, and the <addr> portion of the instruction is placed in the program counter. In other words, the highest memory address is used as a stack pointer for a stack that grows downward from high memory. Resetting the machine or loading a program stores the address of the highest memory location in that location. For a machine with 128 words of storage, the value 0x7F is stored at location 0x7F.
Bxx	RET <addr>	**Return from subprogram.** The memory location pointed by the contents of the highest memory address is loaded into the program counter and the value at the highest memory address is decremented. Only the rightmost seven bits of the value on the stack are used. The address field of the instruction is not used and is reserved.
Cxx	OUTN <addr>	**Output with No newline.** Identical to OUT except that sending the newline character is suppressed.
Dxx	OUTC <addr>	**Output character.** The rightmost eight bits of the accumulator are sent to the output subsystem, to be interpreted as an ISO-8859 character rather than a number. No newline character is sent.
Exx	DAT "<STR>"	Character data pseudo-instruction. The characters in <str>, which must be enclosed in double-quotes, are stored in the rightmost eight bits of consecutive words. Example: DAT "Hello, world!" Escaped characters are not allowed. A newline character may be stored by coding DAT 0x00A. If 0x00A is processed by an OUT instruction, it will be rendered as 10. If it is processed by OUTC, it will be rendered as newline.

Fxx	PIO	Programmed Input/Output. TLC uses the three highest possible memory addresses, 0xfd, 0xfe, and 0xff as registers for I/O. The programmer must store a device address in 0xfd; the input device is 0x000 and the output device is 0x001. For output, the data word to be written must be stored at address 0xff. For input, the value read will be placed in address 0xff. After the registers are set up, the program issues the PIO instruction. The program must then loop, testing address 0xfe until it becomes one, indicating the completion of the I/O operation. Any input or output operation attempted before 0xfe becomes one will cause a machine check.

A.2 Format of Assembly Language Statements

Assembly language statements for TBC are of the form:

[label] op code [address] [comments]

The label, if present, must begin at the left, with no preceding white space. The operation code must be preceded by white space regardless of whether a label is present. (This is different from LMC, which allows operation codes to begin at the left.)

Labels, operation codes, and symbolic addresses are case-insensitive. Labels and symbolic addresses are not length-limited. For valid operation codes, see A.1, Operation Codes.

Addresses may be symbolic addresses, which correspond to labels defined elsewhere in the source program, or numeric addresses. Numeric addresses may be expressed as decimal numbers or hexadecimal numbers. Hexadecimal numbers must be prefixed with "0x" –the digit zero followed by the letter x. Example: 0x5A or 0x5a. Addresses are limited to the range 0-127 or 0x0 to 0x7f.

Anything following the address, or for operation codes that do not take an address, anything following the operation code, is treated as a comment.

By default, the assembler generates object code beginning at location zero. The address where a sequence of instructions will be loaded can be changed using the org assembler directive, which takes a numeric address as an operand. Example:

```
org 0x50
add const1      The "add" instruction will be at address 0x50
brp done        BRP will be at 0x51, etc.
```

The first executable instruction must be at location zero, as it will be by default.

The DAT assembler directive reserves storage for a 12-bit numeric constant and optionally initializes its value. The constant may be a signed or unsigned decimal number or a hexadecimal number. To express a negative number in hexadecimal, use the two's complement of the unsigned value. Because TLC uses 12-bit words in two's complement format, constants are limited to the range -2048 to +2047. If no operand is given, DAT reserves space but does not initialize it. Example:

```
const1 dat 1
minus1 dat 0xfff
minus1 dat -1
datum dat       // one word, not initialized
```

Appendix B.
The TBC Control Unit

TBC's control unit is microprogrammed. The instruction decoder and control unit must generate 17 control signals, as shown in the table. In addition, we need two bits to control branching in the microprogram, one bit to control use of the op code as a branch target, and eight bits of address for jumps. Each microprogram word is 28 bits; for simplicity, we would use a ROM with 32-bit words because these are likely to be commercially available. The layout of the microprogram requires 256 words of ROM, although not all of them are used. The control signals that must be generated are these:

		Data Path Control Signals
ALU	4	Enable A (EnaA); Invert A (InvA); Enable B (EnaB); Increment (Inc)
Memory	3	Read, write, PD
A-bus	4	Select one of four registers to enable onto the A-bus
B-bus	1	Indicates that Acc is to be enabled onto B-bus.
C-bus	5	Selects any of five registers for write from the C-bus
		Microprogram Controls
Op Code	1	If 1, the operation code from the instruction register, shifted left 4, is used as branch target
Jump	2	00=no jump; 01=jump if positive (jp); 10= jump if zero (jz); 11=unconditional jump (ju)
Next Addr	8	If *jump* is non-zero, the address of the next microinstruction

Two of the possibilities for the *jump* control test the bits pf the P/Z Latch, which reflect the current contents of the accumulator.

Tanenbaum (1989) described a notation for representing microcode which he called the micro assembly language, or MAL. The two tables

below show the ADD and BRP instructions in a notation similar to Tanenbaum's MAL, and with the actual bits of the microprogram.

Operation (for ADD, op code 1	Loc	ALU				Mem			Bus control										Op Cd	Jmp	Next Addr
									A				B	C							
		EnaA	InvA	EnaB	Inc	Read	Write	PD	MDR	Acc	PC	IR	Acc	MAR	MDR	Acc	PC	IR			
PC → MAR; rd	00	•				•					•			•							
PC+1 → PC	01	•			•						•						•				
MDR → IR; op	02	•							•									•	•		
IR[addr] → MAR; rd	10	•										•		•							
(wait)	11																				
Acc + MDR → Acc;	12	•		•					•				•			•				ju	00

Operation (for BRP, op code 8)	Loc	ALU				Mem			Bus control										Op Cd	Jmp	Next Addr
									A				B	C							
		EnaA	InvA	EnaB	Inc	Read	Write	PD	MDR	Acc	PC	IR	Acc	MAR	MDR	Acc	PC	IR			
PC → MAR; rd	00	•				•					•			•							
PC+1 → PC	01	•			•						•						•				
MDR → IR; op	02	•							•									•	•		
jump positive 82	80																			jp	82
jmp 00	81																			ju	00
IR[addr] → PC;	82	•										•					•			ju	00

Figure B-1

Operation code zero (HLT) is handled with digital logic as a special case. A zero in the operation code field is detected using a four-input nor. The output is ANDed with the Op Code bit of the control word. A result of true stops the processor clock and so stops execution of the microprogram.

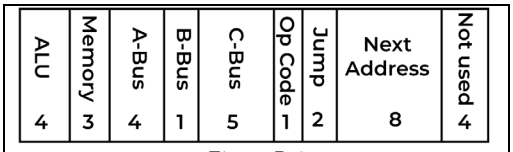

Figure B-2

Figure B-2 shows the layout of a control word in TLC's microprogram control store. The bits are shown in the order they were discussed above. Tanenbaum (1989) pointed out that they would probably be arranged in a way that minimized crossing of conductors when the CPU was laid out for a semiconductor die. That's an engineering detail that need not concern us while we are working at the level of logical design.