Karenbak-1 computer manual

Index

Chapter 1 Motivation and overview

The computer described in this document was originally inspired by one the very earliest 'first generation' machines specifically, the Electronic Delay Storage Automatic Calculator (EDSAC). This machine was built around 1949 and relied on a delay medium for its main memory. The initial plan was to build a fully transistorised computer however, as time passed the plan evolved into a computer based on CMOS logic. As a consequence, the final result more closely resembles the 'Kenbak-1' computer of the very early 1970s. Accordingly, the new machine was christened the 'Karenbak-1'.

The Kenbak-1 computer

Like the Kenbak-1, the Karenbak-1 is comprised almost entirely of small scale integration (SSI) and medium scale integration (MSI) integrated circuits. The single exception is the memory IC for which a shift register is used. The shift registers employed in the Kenbak-1 are hard to obtain now, and so this role has been fulfilled by a bucket brigade device (BBD) in the new machine. My choice of logic family (HCMOS) is entirely down to what I had in my parts store.

The Karenbak-1 is minimalist, employing only 25 ICs, the aim being economy of hardware rather than speed. It has a serial arithmetic and logic unit (ALU). Bits are processed least significant bit (LSB) first, since this facilitates the natural flow of the carry/borrow signal during addition and subtraction. A left shift (useful for multiplication) is also easily implemented within this scheme.

The Karenbak-1 is an 8 bit, single address machine with just three registers accessible to the user: a program counter (PC); an accumulator (ACC); and a memory address register (MAR). A fourth register, the instruction register (IR), is not accessible to the user. The Karenbak-1 processor executes approximately 100 instructions per second.

One operand for dyadic operators is always the ACC. Results of operations are always returned to the ACC. Immediate addressing mode is not supported; it is expected that constants required by a program will be entered into dedicated memory locations during program entry. Indirection is also not directly supported and must be accomplished using self-modifying code, as was the practise in the very early days of computing.

The Karenbak-1 supports program jumps, both conditional and unconditional. This makes the Karenbak-1 'Turing complete'. There are only two status flags: HALT and FLAG, the latter being tested by conditional jump instructions. The Karenbak-1 has around twenty instructions in its instruction set. With minor modifications the design can address up to 2048 bits of memory. With one or two additional ICs, the word length could be increased to 16 bits.

It is important to point out that the minimalism of the Karenbak-1 is not shared by the Kenbak-1. In this respect the comparison is a poor one. The latter was aimed at the educational market and as such, was entirely representative of other machines of its day (for instance, the Kenbak-1 has multiple addressing modes and an index register). By contrast, the Karenbak-1 architecture is some twenty years behind the Kenbak-1.

Chapter 2 Manual controls

The Karenbak-1 front panel has the following controls and indicators:

RESET button	Pressing this button puts the computer into known state. All registers are cleared and the computer is put into a HALT state. Reset does not affect memory contents.
MODE switch	This switch selects between four main activities.
COMMAND button	Pressing this button orders the option selected on the MODE switch.
Data input switches	Data entry for programming or data input is set on these switches.
Address LEDs	These LEDs show the current address for program entry or examination. They display the contents of the MAR.
Data LEDs	These LEDs show the current data for program entry or examination. They display the contents of the ACC.

The MODE switch is the main control. It selects between four basic activities:

PROGRAM	In this mode, pressing the COMMAND button will cause the data switches to be displayed on the data LEDs, and also stored at the location indicated on the address LEDs. The address will automatically increment following the write to memory.
EXAMINE	In this mode, pressing the COMMAND button will cause the contents of the memory at the address indicated on the address LEDs to be displayed on the data LEDs. The address will automatically increment following the read from memory.
INPUT	In this mode, pressing the COMMAND button will cause the data switches to be loaded into the ACC. This is typically done when a program HALTs for input of data. The program can be resumed using the EXECUTE mode.
EXECUTE	In this mode, pressing the COMMAND button will cause the computer to begin execution. A prior reset should be performed to run a program from its start. Where a program has halted for data input, it can be resumed using this mode, whereupon the program will continue from the instruction following the HALT.

It is permissible to switch between 'PROGRAM' and 'EXAMINE' while stepping through memory however, there is no means to decrement the address.

Chapter 3 Instruction set

While there are notionally single word instructions and two word instructions, all instructions are in truth single word. Where an instruction requires an operand address, this is obtained using a prior 'load MAR' instruction.

Single word instructions			
Opcode	**Encoding**	**FLAG**	**Description**
HALT	0 0 0 0 0 0 0 0 00	-	Stop computer
NOP	1 1 0 0 0 0 1 1 C3	-	No operation
SETF	1 0 1 0 0 0 0 0 A0	1	Set FLAG
CLRF	1 0 1 1 0 0 0 0 B0	0	Clear FLAG
SENS	1 0 0 0 0 0 0 1 81	Z	Sense switches
NOT	1 0 0 0 0 1 1 0 86	Z	Invert ACC
LSH	1 0 0 0 0 1 1 1 87	LSB / MSB	Left shift ACC
LDM	0 1 n n n n n n 40-7F	-	Load MAR

Two word instructions				
Opcode	**Encoding**		**FLAG**	**Description**
	First word	**Second word**		
LD	LDM 0 1 n n n n n n (40+n)	1 0 0 0 0 0 0 0 80	-	Load
LDN		1 0 0 0 1 0 0 0 88	-	Load NOT
ST		1 0 0 1 0 0 0 0 90	-	Store
ADD		1 0 0 0 0 0 1 0 82	C	Add
SUB		1 0 0 0 1 0 1 0 8A	B	Subtract
AND		1 0 0 0 0 0 1 1 83	Z	AND
ANDN		1 0 0 0 1 0 1 1 8B	Z	AND NOT
OR		1 0 0 0 0 1 0 0 84	Z	OR
ORN		1 0 0 0 1 1 0 0 8C	Z	OR NOT
XOR		1 0 0 0 0 1 0 1 85	Z	XOR
XORN		1 0 0 0 1 1 0 1 8D	Z	XOR NOT
JMP		1 1 0 0 0 0 0 0 C0	-	Jump always
JMPS		1 1 0 0 0 0 0 1 C1	-	Jump if FLAG set
JMPC		1 1 0 0 0 0 1 0 C2	-	Jump if FLAG clear

FLAG keys:

-	Not affected
Z	Set if any result bit is set
C	Carry
B	Borrow (inverted)

It is not mandatory to precede all instructions with a 'load MAR' instruction. The MAR post-increments, thereby allowing data accesses that step through memory. This feature is useful for filling memory since repeated store instructions will write the same value to multiple locations.

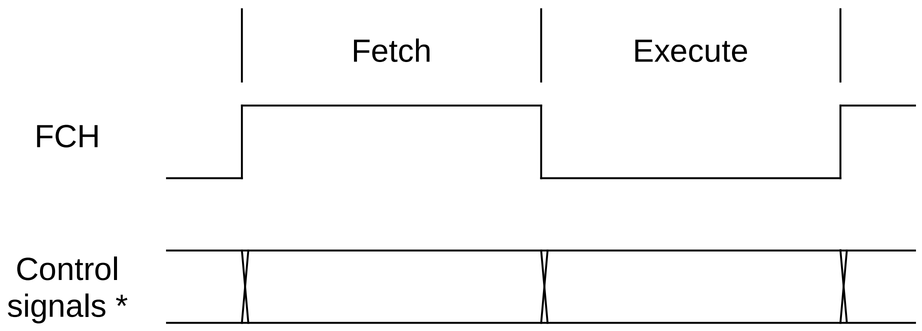
Strictly speaking, the left shift instruction (LSH) is a rotate left instruction, since the ACC LSB acquires the initial state of FLAG, while the final state of FLAG is the initial ACC most significant bit (MSB). The no operation instruction (NOP) is actually a 'jump never' instruction.

Finally, the 'sense switches' instruction (SENS) senses the instantaneous settings of the data input switches and does not wait for user input. Interactive user input should use a HALT instruction to prompt the user, who should then use the INPUT mode to enter data. A useful trick when using HALTs in this manner is to precede the HALT instruction with a LDM (load MAR) instruction. This will allow a number to be displayed on the address LEDs while the user is being prompted, which can serve as an identifier for the wanted information. In this respect one can view the HALT instruction as a two word instruction e.g. 'HALT 3'.

Chapter 4 Processor timing

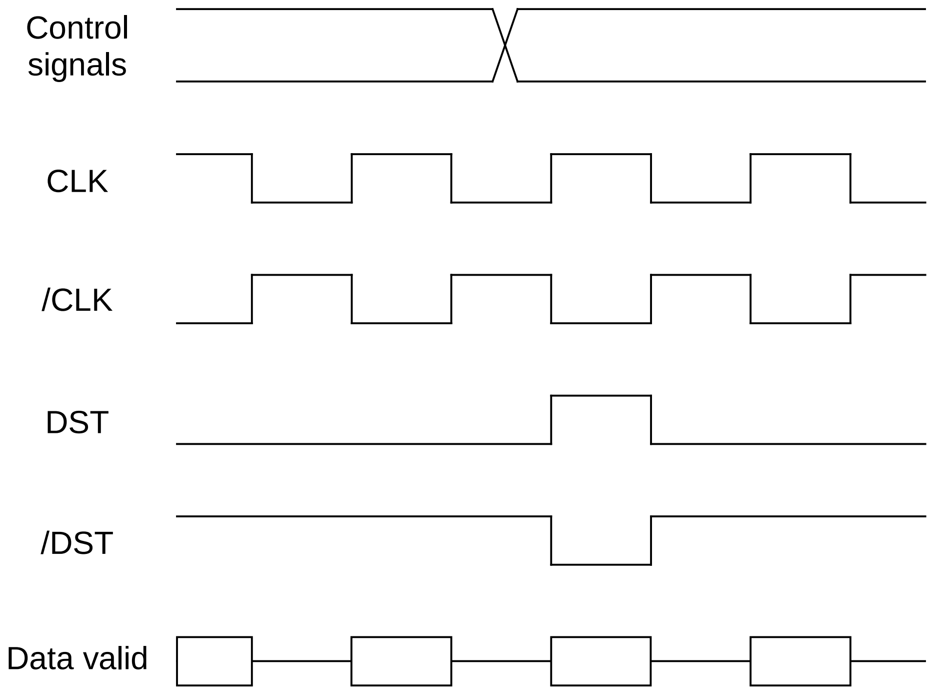
General timing

The processor control signals change state every 512 bit clocks. The main control signal is FCH, which alternates between the high state (instruction fetch cycle) and the low state (operand fetch and execute cycle).



* Control signals include ACE, /ALE, JMP, ME, /MLD, PCE and /STO

Timing relative to the bit clock is as follows.



Chapter 5 Memory addressing

The Karenbak-1 memory consists of a 1024 stage BBD - an MN3207. Although this IC is described as having 1024 stages, the actual number of analogue samples held at any one time is only half that figure. Hence the IC implements a 512 bit memory. The 512 bits are organised as 64 off 8 bit words. This is a small memory, but enough to demonstrate simple programs.

The Karenbak-1 uses an obscure method of memory addressing. The PC is of course a counter however, its counting ability is used in a novel way to select an instruction from memory. During a processor fetch cycle, the PC is clocked by the bit clock. But since the PC has only 6 bits, the 512 bit clocks contained within a processor cycle result in the PC overflowing repeatedly - eight times to be precise. Despite the overflows, 512 counts will still return the PC to its initial state. The PC is not therefore corrupted by 'spinning' it in this fashion.

The useful property of the PC overflowing is that each overflow of the PC can be used to identify 8 unique bits within the memory. Each initial value of the PC will identify a different group of 8 bits, thereby providing an addressing function. The MSB of the program counter is in fact used to identify wanted bits, positive transitions signalling to the instruction register that a wanted bit is currently emerging from the serial memory. The MAR too is a counter, and identifies wanted operand bits in the same way as the PC. The result is a strange mapping of ordinal address to logical address:

Ordinal bit address	Logical bit number	Logical address
1		31
2		30
..		..
31		1
32	0	0
33		63
..		..
63		33
64		32
65		31
66		30
..		..
95		1
96	1	0
97		63
..		..
127		33
128		32
..	..	..
385		31
386		30
..		..
415		1
416	6	0
417		63
..		..
447		33
448		32
449	7	31
450		30

Ordinal bit address	Logical bit number	Logical address
..		..
479		1
480		0
481		63
..		..
511		33
512		32

Following a reset the PC contains zero however, the Karenbak-1 uses PC ***pre-increment***, and so execution will actually begin from location 1 following a reset.

Jump instructions require special attention. During the execution of a jump instruction, the MAR is copied to the PC however, an offset is introduced by this process. The required setting of the MAR for a jump to location N is given by:

$$(N + 32_{10}) \bmod 64_{10}$$

In practise, this just requires that the most significant bit of the jump address is inverted. For example, to make a jump to location 10_{10} ($0A_{16}$) the MAR should be pre-loaded with 42_{10} ($2A_{16}$). To jump to location 50_{10} (32_{16}) the MAR should be loaded with 18_{10} (12_{16}).

Location zero

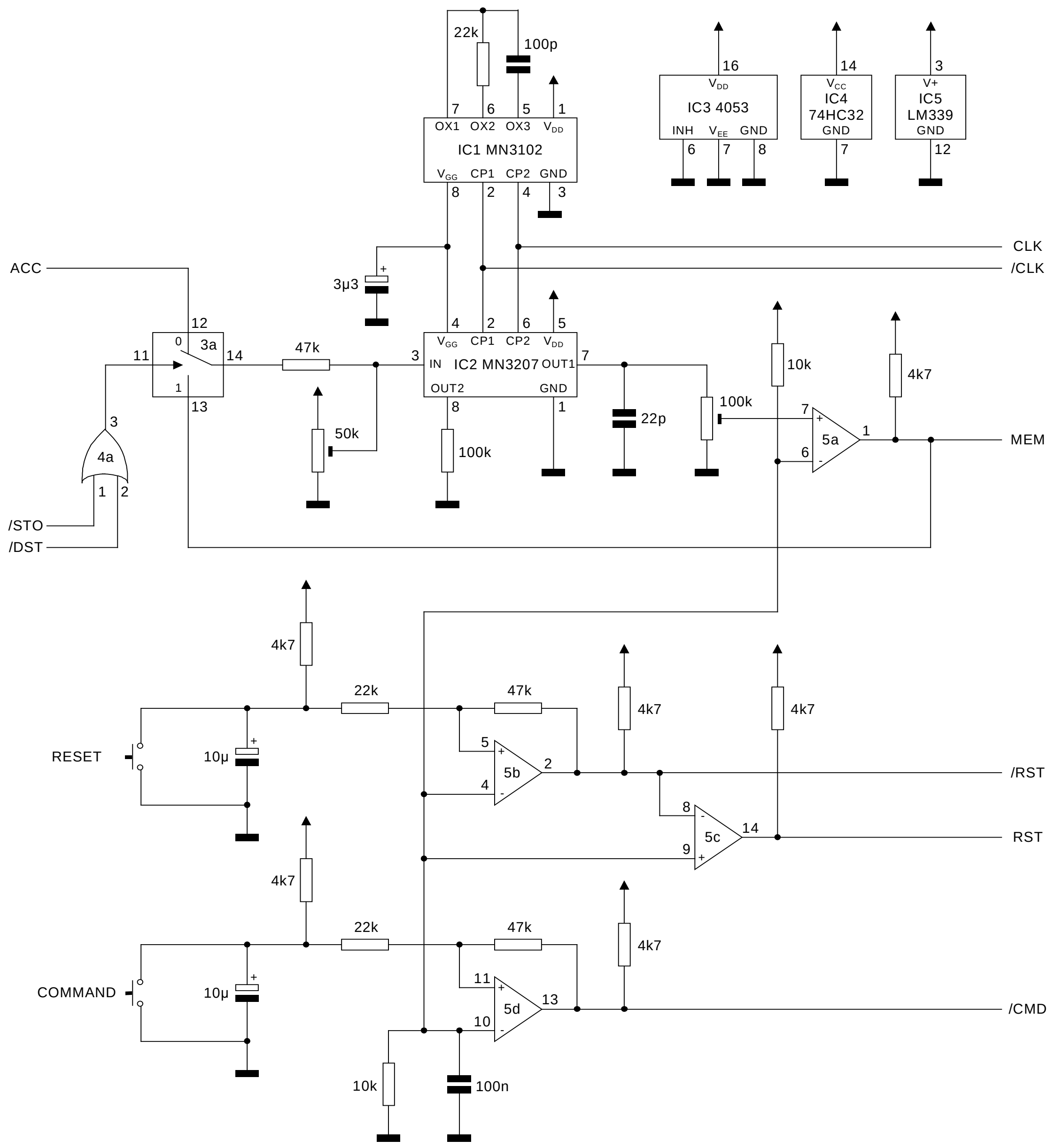
As mentioned earlier, execution begins from location 1 following a RESET. It is recommended that location zero be programmed with the constant zero, as this permits the following macro instructions to be used:

Address	Coding	Label	Opcode / Directive	Operand	Comment
00		ZERO	DATA	00	Constant: 0
		CLR	MACRO		Macro: Clear accumulator
00	40 80		LD	ZERO	
			END		
		INC	MACRO		Macro: Increment accumulator
00	A0		SETF		
01	40 82		ADD	ZERO	
			END		
		DEC	MACRO		Macro: Decrement accumulator
00	B0		CLRF		
01	40 8A		SUB	ZERO	
			END		
		TSTZ	MACRO		Macro: Test for zero accumulator
00	B0		CLRF		
01	40 84		OR	ZERO	
			END		

Chapter 6 Hardware circuits and descriptions

Decoupling capacitors are not shown in the circuits that follow. A 100n decoupling capacitor should be present close to the power pins of each IC. A single 100μ decoupling capacitor is recommended for the entire computer. All LEDs are of the low current (2mA) variety. The computer uses a single 5V power rail.

Memory



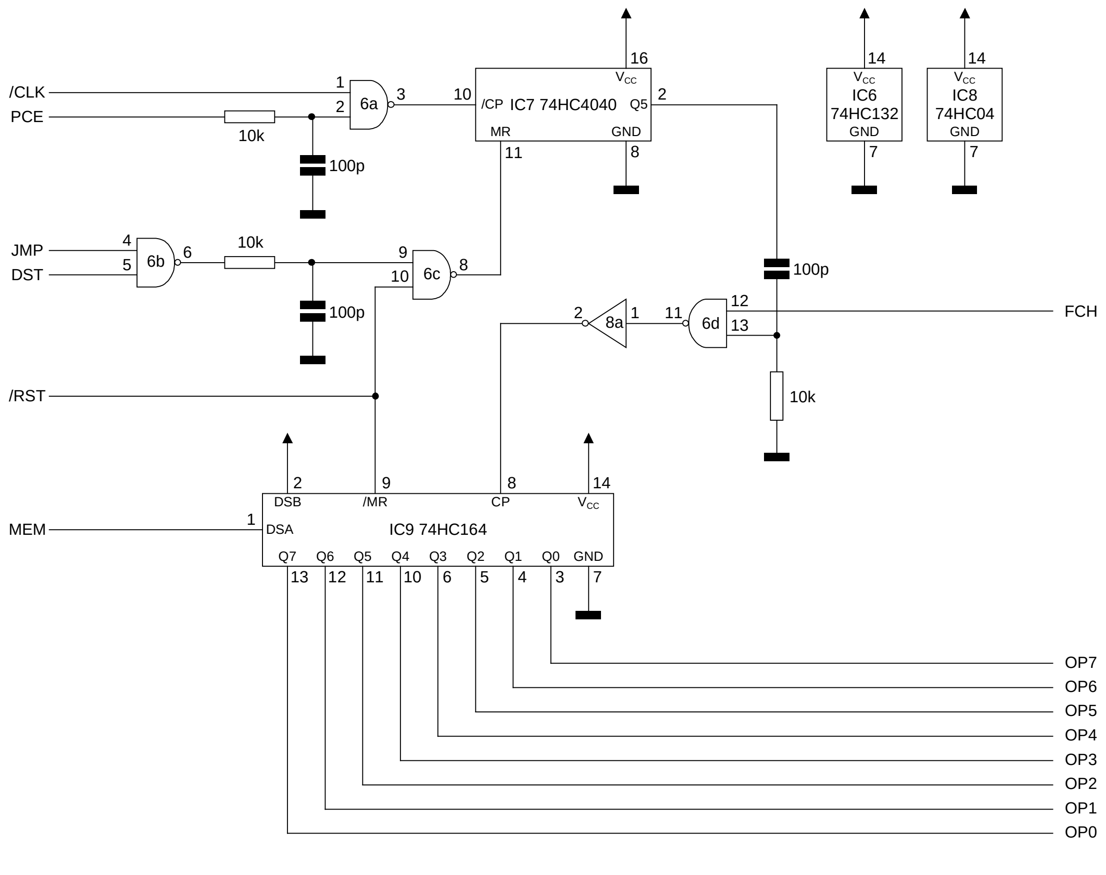
IC1 furnishes anti-phase clock pulses of high drive capability. These are used extensively throughout the design, providing a 100kHz bit clock for the system. 512

bit clocks (i.e. around 5 milliseconds) defines the re-circulation time of the memory. It also defines one processor cycle.

IC2 drives its OUT1 pin when its CP2 input (CLK) is in the high state. Coincidentally, IC2 samples its IN pin at the same time. A re-circulating digital memory is therefore created by simply feeding OUT1 back into IN. Input conditioning consists of an attenuator and a level shifter to bring the logic signal within the linear range of IC2's input. Output conditioning uses a section of a quad comparator package to slice IC2's output to convert it back to a logic level. The remaining comparators in the quad package are used for de-bouncing the RESET and COMMAND buttons.

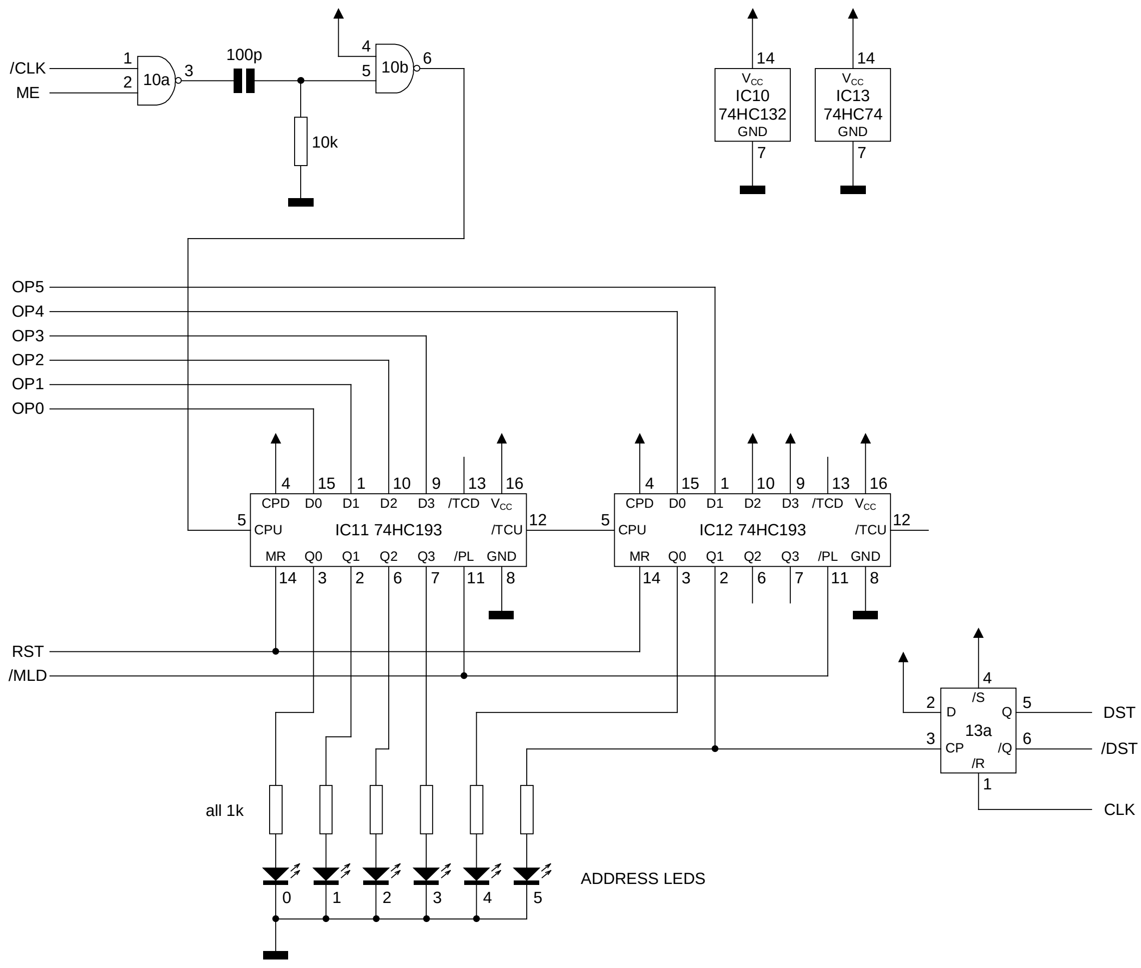
An analogue switch (IC3a) controls the input to IC2. Under most circumstances, memory data is simply re-circulated. However, for a store instruction, the switch is thrown and IC2's input is obtained from the ACC.

Program counter and instruction register



As mentioned earlier in this document, spinning the PC does not corrupt it however, there is still a need to increment the PC in order to step through instructions. This is achieved by another obscure method: the positive edge of the PCE signal injects an extra count in addition to the 512 accrued during the instruction fetch cycle. This provides a PC pre-increment.

Memory address register



Like the PC, the memory address register (MAR) is a counter. During an operand fetch/execute processor cycle, the MAR is spun, and positive transitions of the MSB of the MAR are used to signal to the ACC and FLAG that a wanted operand bit is currently emerging from the serial memory. The MAR has a post-increment function. It is implemented in a very similar way to the program counter pre-increment. Post increments of the MAR are used during manual program entry and examination.

The MAR signals the presence of wanted data on the output of the memory by pulsing DST and /DST. These signals would typically be used to clock data into the ACC, or to reroute the memory input source to allow data to be stored.

The MAR is key to the implementation of jump instructions. During the execute cycle of a jump instruction, the DST signal is gated onto the reset input of the PC (IC7 pin 11). As a result, 8 reset pulses are delivered to the PC although only one is strictly necessary. These timed resets force the PC into synchronism with the MAR and ensure that the final state of the PC matches that of the MAR, subject to an offset.

Accumulator, switches and FLAG

The ACC is a shift register (IC15). When performing an arithmetic or logic operation, the last stage (Q7) provides one bit of one operand. The corresponding bit of the other operand comes from the memory. The result of the operation is shifted back into the ACC via IC15's DSA input. Input from the switches is provided by IC16, a multiplexer. During program entry or data input, the data on the switches appear bit at a time on the IP signal.

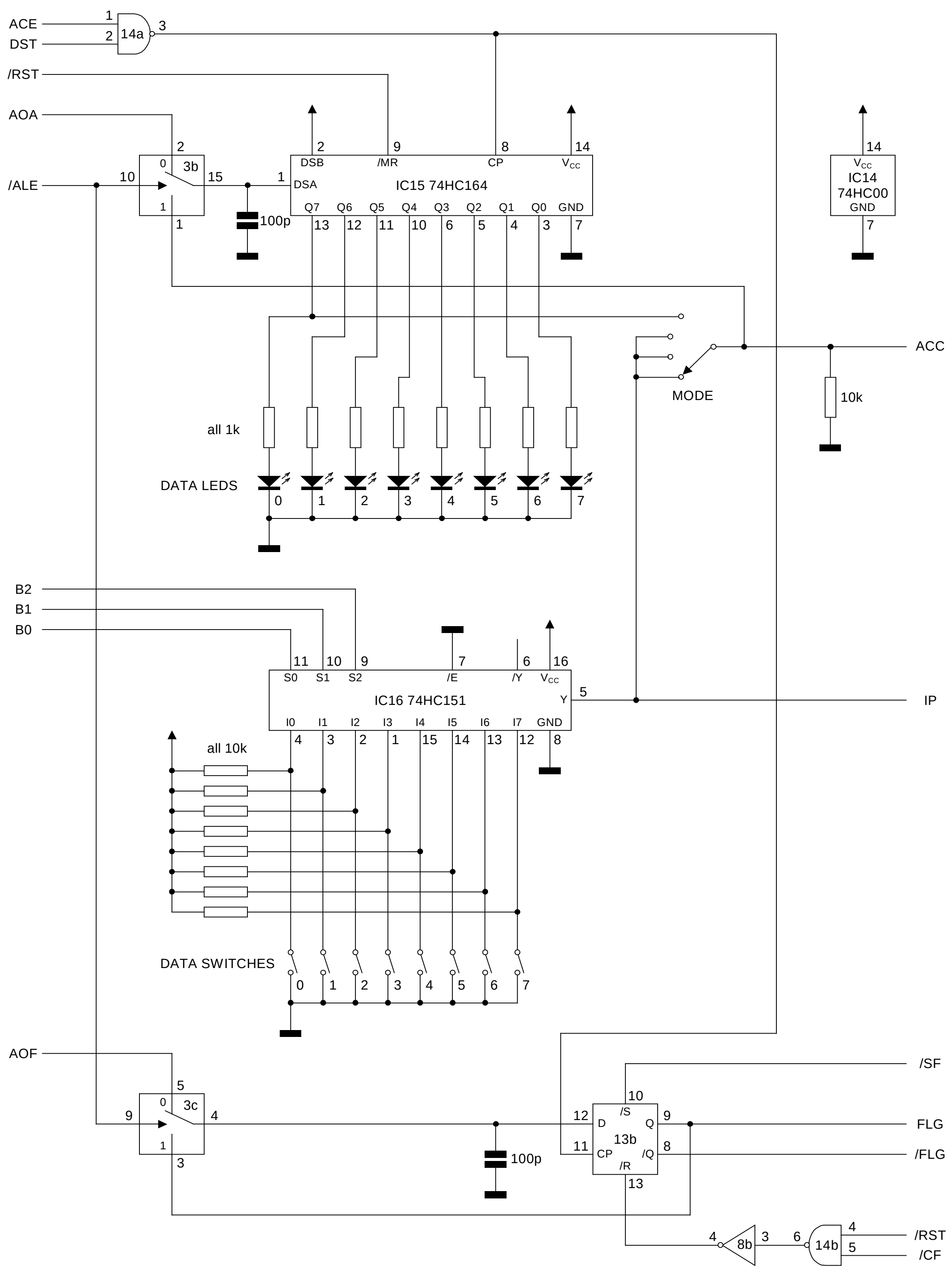
An analogue switch (IC3b) controls the input to IC15. During execution of an arithmetic or logical operation, the switch is thrown to select the output of the ALU. During a store instruction, ACC data is re-circulated so as to preserve the ACC contents. A similar analogue switch (IC3c) preserves the state of the FLAG (held by flip-flop IC13b) during a store instruction.

Instruction decode and sequencing

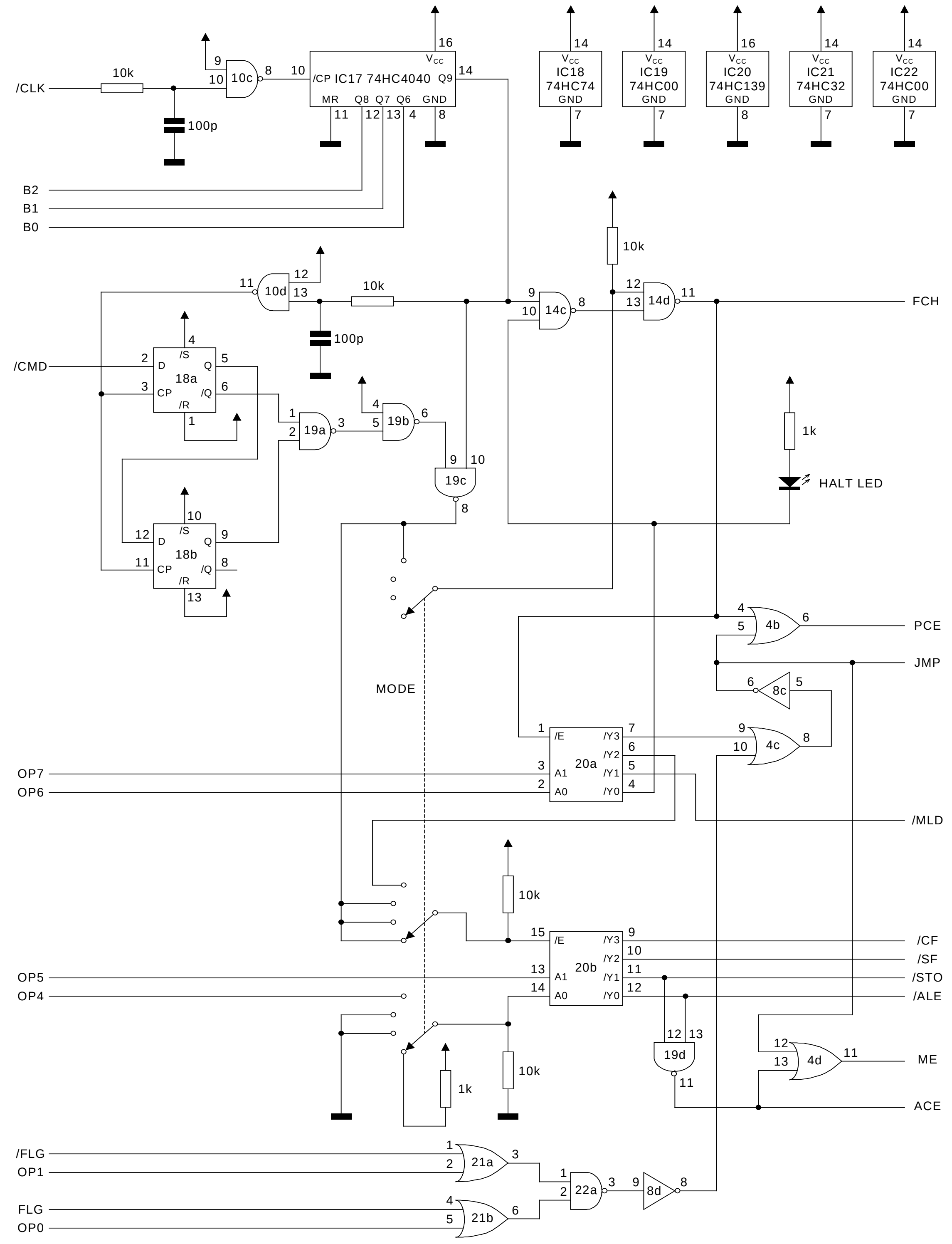
IC17 maintains track of data in the circulating memory. It is also responsible for the main sequencing of the computer: Q9 of this counter alternates between 512 bit clocks high, and 512 bit clocks low. When the computer is running, the high state of this signal corresponds to an instruction fetch cycle, while the low state corresponds to the operand fetch and execute cycle. The stopped state is implemented as a protracted HALT instruction (the all-zeros state of the IR). While the /Y0 output of IC20a is low IC17 Q9 cannot drive FCH, and so instruction fetches cease. This state can be exited by setting the MODE switch to the last setting (EXECUTE) and pressing the COMMAND button.

IC18 and associated gates generate a negative going pulse of exactly one processor cycle duration when the COMMAND button is pressed. This pulse is used to implement the various mode switch functions. Depending on the setting of the MODE switch, the COMMAND button can order a write of switch data to memory, a read of data from memory into the ACC, a transfer of switch data to the ACC (for interactive data input) or a program start.

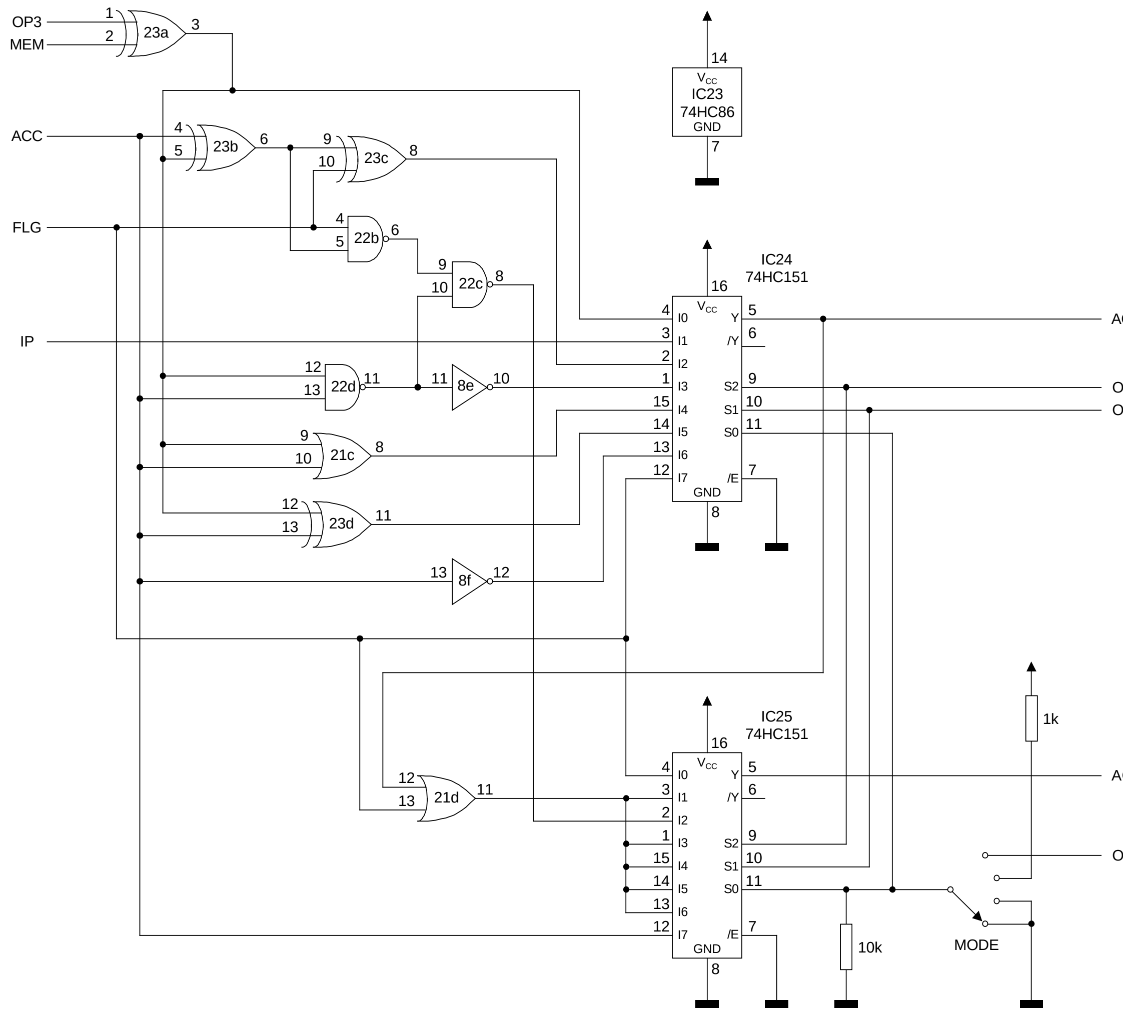
Finally, IC21a,b and IC22a implement the conditional logic for the jump instruction.



ACE
DST
14a
/RST
AOA
/ALE
3b
100p
IC15 74HC164
DSB
/MR
CP
VCC
DSA
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
GND
IC14
74HC00
GND
MODE
ACC
10k
all 1k
DATA LEDS
B2
B1
B0
IC16 74HC151
S0
S1
S2
/E
/Y
Y
IP
I0
I1
I2
I3
I4
I5
I6
I7
all 10k
DATA SWITCHES
AOF
3c
100p
13b
/S
D
Q
CP
/Q
/R
/SF
FLG
/FLG
8b
14b
/CF


/CLK
10k
100p
10c
IC17 74HC4040
/CP
Q9
MR
Q8
Q7
Q6
GND
IC18 74HC74
IC19 74HC00
IC20 74HC139
IC21 74HC32
IC22 74HC00
B2
B1
B0
10d
14c
14d
FCH
/CMD
18a
18b
19a
19b
19c
1k
HALT LED
MODE
4b
PCE
JMP
8c
4c
20a
OP7
OP6
/MLD
20b
/CF
/SF
/STO
/ALE
OP5
OP4
19d
4d
ME
ACE
/FLG
OP1
FLG
OP0
21a
21b
22a
8d

Arithmetic and logic unit



The ALU is fairly conventional, save for processing only one bit at a time. Multiplexers (ICs 24 and 25) select the operation to be performed for both the ACC and the FLAG. While not strictly an arithmetic or logical operation, the load instruction is implemented in the ALU (input I0 of the two multiplexers). Input from the switches is possible on multiplexer input I1. Switch input can be user-interactive (using INPUT mode while the computer is HALTed) or they can be sensed asynchronously using the SENS instruction.

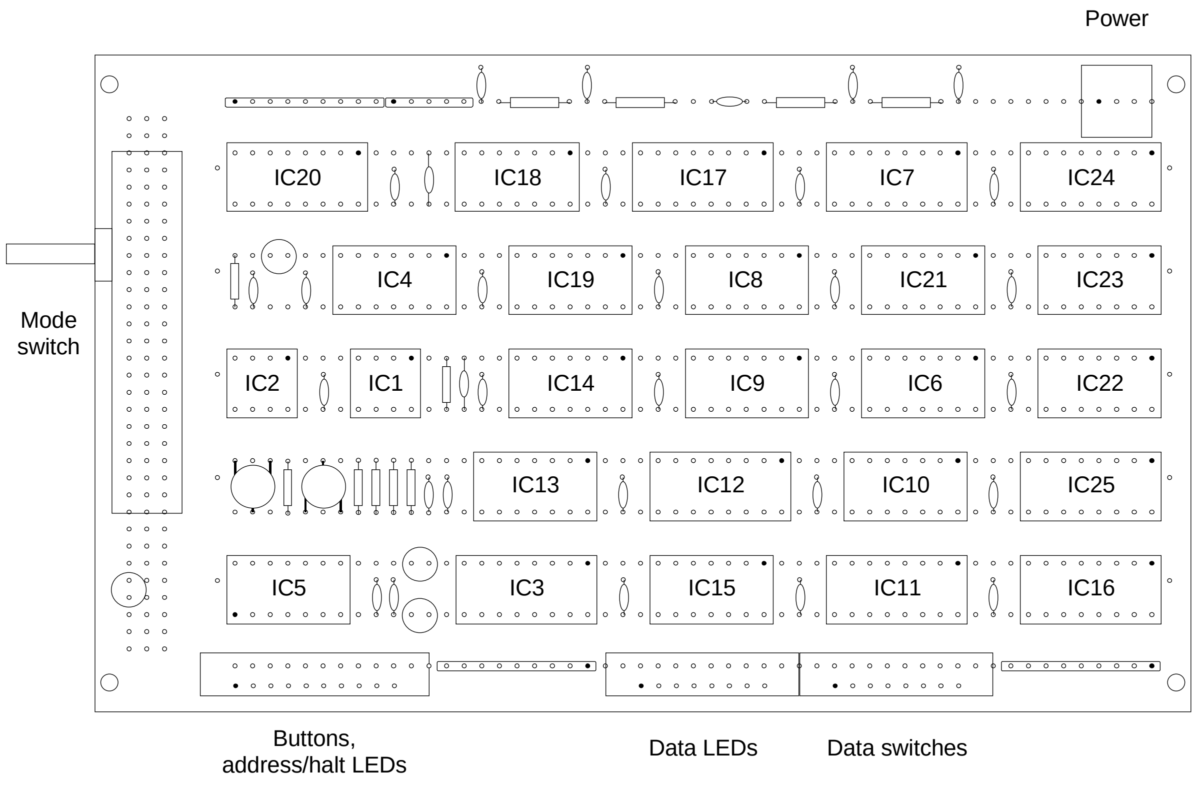
A full adder provides for both addition and subtraction, the FLAG serving to propagate carry/borrow status. For add instructions, the FLAG should be cleared prior to execution. For subtract instructions, the FLAG should be set prior to execution. On completion of an addition or subtraction instruction, a word carry/borrow is deposited in FLAG, thereby permitting multi-word additions and subtractions. Note that one bit of the operation code (OP3) selects between normal or inverted operand data from memory. This greatly increases the flexibility of the ALU.

The FLAG performs a special function during execution of logical operations (AND, ANDN, OR, ORN, XOR, XORN or NOT). When one of these instructions is executing, the FLAG will be logically ORed with all bits of the result. Consequently, the FLAG should be cleared prior to execution, so as to enable a zero result to be tested for after execution. Finally, the multiplexer I7 inputs implement the left shift instruction, whereby the ACC and FLAG simply exchange states.

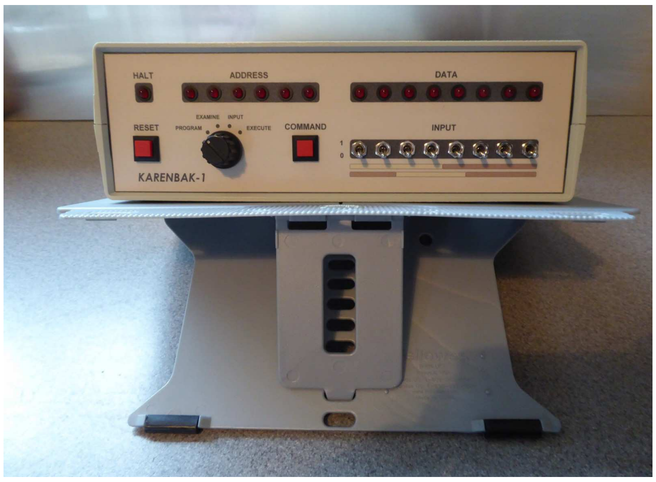
List of ICs

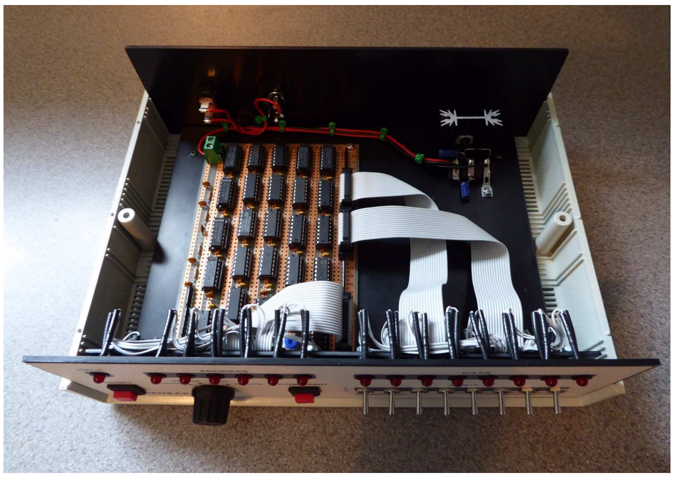
IC number(s)	Pin count	Type
IC1	8	MN3102
IC2	8	MN3207
IC3	16	CD4053
IC4, 21	14	74HC32
IC5	14	LM339
IC6, 10	14	74HC132
IC7, 17	16	74HC4040
IC8	14	74HC04
IC9, 15	14	74HC164
IC11, 12	16	74HC193
IC13, 18	14	74HC74
IC14, 19, 22	14	74HC00
IC16, 24, 25	16	74HC151
IC20	16	74HC139
IC23	14	74HC86

Card layout


Power
IC20
IC18
IC17
IC7
IC24
IC4
IC19
IC8
IC21
IC23
IC2
IC1
IC14
IC9
IC6
IC22
IC13
IC12
IC10
IC25
IC5
IC3
IC15
IC11
IC16
Mode switch
Buttons, address/halt LEDs
Data LEDs
Data switches

Chapter 7 Photos





Chapter 8 Finer points of the design

The capacitor on the output of IC2 serves to extend the valid time of the conditioned memory output. This gives other devices within the computer time to capture memory data as CLK returns to the low state and IC2's output returns to a quiescent level.

The rising edge of PCE injects a count into IC7 (the PC). This provides the PC pre-increment. It is possible that this count may cause the IR to shift in an erroneous bit. This in itself is not a problem because 8 further shifts would then occur, and the erroneous bit would ultimately be shifted out of the IR. The hazard is transient false decodes on the outputs of IC20, which may occur if IC20's address inputs (which derive from the IR) change too close to the rising edge of FCH. While it is thought that this is unlikely, the PCE signal is delayed slightly by an RC network on the input to IC6a. This will prevent transient false decodes from ever occurring.

During execution of a jump instruction, reset pulses are delivered to IC7 (the PC) which originate from the MAR's DST signal. The problem is that the trailing edges of these reset pulses coincide with valid count transitions on IC7's clock input. This makes it uncertain whether the count transition will be recognised and acted upon. To ensure that these count transitions are NOT acted upon, the reset pulses are delayed slightly by an RC network on the input to IC6c.

During an instruction fetch, the IR shifts in memory data on positive transitions of IC7's (the PC) Q5 output. These shifts must be gated by the FCH control signal however, it is not sufficient to use a simple gate because, depending on the initial state of IC7 Q5, an additional shift may occur on the rising edge of FCH. This might lead to transient false decodes (see paragraph before last). To prevent this, a monostable is implemented by a CR network on the input to IC6d.

The MAR (ICs11,12) is incremented on the leading edge of the clock. If the MSB of the MAR makes a positive transition as a result, then IC13a will be clocked, however, IC13a is only just coming out of reset at this time. In order to move the MAR increment, and any resulting IC13a clock, away from this condition (CLK low), a monostable is implemented by a CR network on the input to IC10b. The MAR is incremented at the end of the pulse on IC10b's output.

The MAR is also incremented on the falling edge of ME. This provides the MAR post-increment. This increment does not result in a spurious DST pulse on the output of IC13a because this flip-flop is held in the reset state (by CLK) when ME is falling.

Control signals must change only between bit clocks (see timing diagrams in Chapter 4). This is ensured by an RC delay network on the input to IC10c.

ICs 13b and 15 are often called upon to sample their own output. It is important to ensure that their inputs (D for IC13b, DSA for IC15) have sufficient data hold time. The capacitors on these inputs extend this hold time.

Chapter 9 Software

No assembler has yet been written for the Karenbak-1 and the following example program is hand coded. Presented is a simple program which multiplies two unsigned eight bit numbers:

Address	Coding	Label	Opcode / Directive	Operand	Comment
00	00	ZERO	DATA	00	Constant: 0
01	40 80	START	LD	ZERO	Prepare to multiply:
03	75 90		ST	HBE	Clear HBE
05	90		ST		Clear RESH
06	90		ST		Clear RESL
07	79 80		LD	ONE	Initialise bit mask
09	78 90		ST	MASK	
0B	74 80	LOOP	LD	MULB	Multiply loop: Bit set?
0D	B0		CLRF		
0E	78 83		AND	MASK	
10	7F C2		JMPC	@SHIFT	
12	73 80		LD	MULA	Yes - add to result
14	B0		CLRF		
15	77 82		ADD	RESL	
17	77 90		ST	RESL	
19	75 80		LD	HBE	
1B	76 82		ADD	RESH	
1D	76 90		ST	RESH	
1F	73 80	SHIFT	LD	MULA	Shift everything along
21	B0		CLRF		
22	87		LSH		
23	73 90		ST	MULA	
25	75 80		LD	HBE	
27	87		LSH		
28	75 90		ST	HBE	
2A	78 80		LD	MASK	
2C	B0		CLRF		
2D	87		LSH		
2E	78 90		ST	MASK	
30	6B C2		JMPC	@LOOP	All done?
32	00		HALT		Yes - stop
33	A7	MULA	DATA	A7	First number to multiply (167)
34	5D	MULB	DATA	5D	Second number to multiply (93)
35	00	HBE	DATA	00	High byte extension
36	00	RESH	DATA	00	Result (high byte)
37	00	RESL	DATA	00	Result (low byte)
38	00	MASK	DATA	00	Walking bit mask
39	01	ONE	DATA	01	Constant: 1

Note: The '@' symbol precedes jump operands to inform the 'assembler' that the formula supplied in Chapter 5 should be applied.

Chapter 10 Conclusion

Historically speaking, the fixed cycle time of the Karenbak-1 would have been considered wasteful, even though it leads to a much simplified design. Early computing machines that were based on circulating memory instead used 'coincidence detectors' to indicate when wanted data was emerging from the delay medium. Instruction fetch and execution were considered separate processes which weren't constrained to the memory circulation period. This freedom resulted in a potential doubling of a machine's speed.

Nonetheless, the Karenbak-1 is a successful proof of concept machine. It is expected that the design will provide useful insights into the inner workings and principles of computer hardware. The design's only major detraction is that it is severely lacking in memory, and this limits the scope of programs which can be demonstrated.

It is intended that a second machine will be constructed at a future date which will be based on the same principles as the Karenbak-1. This machine might use discrete transistors as opposed to IC logic. It might also bring out the PC, IR and FLAG to dedicated LEDs to create a classic 'blinkenlights' machine. Regardless, this new machine will most definitely have a larger memory.

The Karenbak-1 is currently limited to stepping through memory in order for the user to modify (PROGRAM) or interrogate (EXAMINE) the memory contents. This is just about tolerable where there are only 64 words of memory. With a larger memory however a means of setting the address directly (or at the very least, the means to step backwards through memory) will become a necessity.