Hardware Overview:

RAM:

The main system RAM is a 64Kbit 6264 CMOS static RAM which is divided into 8,192 8-bit words. To simplify the hardware design memory access is limited to 512 bytes (9 bits). This is further subdivided into 256 bytes of program memory which can be set with the hardware DIP switches and 256 bytes of data memory which can be read and modified by a running program.

To set the program memory push the PROGRAM button located adjacent to the right DIP switches. The status LED adjacent the left DIP switches will illuminate RED. The ram address (orange LEDs) will be mapped to the left DIP switches and the currently programmed value will be shown on the data LEDs (green). While in program mode, when the LOAD button is pushed, the value on the right DIP switches will be copied into RAM.

To exit program mode, push the PROGRAM button. The status LED will illuminate GREEN. The LOAD button and dip switches do nothing while in run mode. In run mode, the address is set by the Memory Address Register (chip to the left of the memory LEDs). This register is not accessed directly and is instead set as required by the various operations.

The DATA MEM status LED will illuminate whenever a program is accessing data memory.

Clock:

The main system clock can be driven automatically or manually. The AUTO/MAN button toggles between the two modes and the adjacent LEDs will indicate green for Automatic, or red for Manual.

In automatic mode, the clock will pulse at a frequency of 1 Hz to 60 Hz, depending on the position of the potentiometer. In manual mode, the clock will pulse once every time the CLOCK ADV button is pushed.

If the computer is halted (red HALT LED lit), the clock will not advance in manual or automatic mode. A RESET will be required to release the system from halt.

Program Counter:

The program counter keeps track of the next command to run in the program. It increments automatically once its contained value is transferred to the memory address register. Jump operations are completed by loading a value into the program counter and overriding the current count.

Step Counter:

Every operation takes multiple clock cycles to complete (minimum 3). The step counter tracks the current progress through the operation and increments on a clock falling edge. The counter will reset to zero whenever the micro step reset (MSR) signal is active and the clock goes low.

Eg. The signals in a MOV A, B operation are:

0: Program Counter OUT, Memory Address Register IN
1: RAM OUT, Instruction Register IN, Program Counter ENABLE
2: B OUT, A IN, Micro Step Reset

Note that steps 0 and 1 are the same in all operations.

Reset:

The RESET signal sets the Program Counter and Step Counter Values to 0 and starts operation at the beginning of the loaded program.

Register and Memory contents are not erased.

Instruction Register:

The instruction register latches the currently executing operation to the control ROM, freeing the memory to conduct other tasks. It always latches the memory contents on Step 1.

Microcode ROM:

The Microcode ROMs are comprised of 3x 64kbit (8K) parallel EEPROMs. The 13 address lines of the ROMs are driven in parallel, with 8 bits coming from the instruction register, 3 bits coming from the step counter and the remaining two bits originating from the ALU flags register.

Microcode is written to the board by writing to a 32KB I^2C EEPROM using a raspberry pi and a 3.3V (pi) to 5V (board) level shifter. The EEPROM can be added as a block device in Linux using the 24C256 kernel driver and then written or read directly with `dd(1)`. A full read of the EEPROM takes around 3 seconds. A full write takes around 30 seconds. To limit re-writes of unchanged data a python flashing script first dumps the contents of the EEPROM and then compares with the new bin file and flashes only the changed bytes.

At board power on a dedicated ATMega16 microcontroller reads byte 32,768 of the I^2C EEPROM. If it finds a non-zero value, it checks the contents of each of the 3 parallel 8K EEPROMS in turn and updates any changed values. Read times on the parallel EEPROMS are in the order of 100ns. Write times are in the order of 10ms per byte. A full read takes around 3 seconds. A full write takes ~ 5 minutes. At the conclusion of the check, the microcontroller writes a zero to byte 32,768 and then shuts down.

Each microcode ROM drives 8 control signals for a total of 24. Any signal denoted with a bar on the schematic ie: $\acute{AO}$ is an active low signal.

AI	A register IN
AO	A register OUT
BI	B register IN
BO	B register OUT
CI	C register IN
CO	C register OUT
DI	D register IN
DO	D register OUT
ΣI	ALU IN
ΣO	ALU OUT
Σ0	ALU Command Bit 0
Σ1	ALU Command Bit 1
Σ2	ALU Command Bit 2
Σ3	ALU Command Bit 3
MP	Program Memory / Data Memory Select (H/L)
MI	Memory Address Register In
RI	RAM Data IN
RO	RAM Data OUT
PI	Program Counter IN
PO	Program Counter OUT
PE	Program Counter ENABLE (count on next clock pulse)
HLT	Halt
IRI	Instruction Register IN
MSR	Micro Step Reset
RESET	Reset the Program Counter & Step Counter to 0 Note: This is user generated by a push button.

Arithmetic Logic Unit (ALU):

The ALU is built around an Atmel ATMega16 AVR microcontroller as most discrete ALU logic chips have been obsolete for 10-20 years and old stock is expensive. The microcontroller has 32 I/O lines which are split as following:

- 8 bits Input (ALU Input Register)
- 8 bits Input (Connected to the B Register)
- 4 bits Input (ALU Command Select)
- 8 bits Output (ALU Result)
- 4 bits Output (Flags Register)

In the spirit of the project, the ALU is largely programmed to be compatible with a 74LS382 TTL ALU with a few additional operations that could easily be implemented in logic – ie no multiplications, no division.

The ALU sets four flags on the result of a calculation which are used for conditional jumps.

Flags:

Flag	**Description**
Carry	The result overflowed from 255 to 0 or underflowed from 0 to 255.
Overflow	The result overflowed from 127 to -128 or underflowed from -128 to 127
Negative	The result is negative (ie >128 unsigned)
Zero	The result is zero

Registers:

The computer has four registers. Note that Register D serves double duty as the stack pointer, if it is overwritten the current stack position will be lost unless copied elsewhere and recovered.

Register	**Address**	**Special Function**
A	000	Decimal Display
B	001	ALU B Input
C	010	None
D	011	Stack Pointer

Special Addresses:

Item	Address	Description
SP	100	Stack Pointer (Alias of Register D). **Note:** This would be used for Register E if there were enough control signals to add another register without adding a fourth ROM.
PC	101	Program Counter
SPi	110	Stack Pointer Increment (Used in PUSH/POP operations to INC/DEC the SP).
IMM	111	Immediate memory access, loads a value from the next byte of program memory.

Decimal Display:

Register A is equipped with a 4-digit, 7-segment display. The adjacent dip switch can be used to select the output mode.

Switches	Description
00	Display outputs in unsigned decimal
01	Display outputs in signed decimal. The last decimal place is illuminated.
10	Display outputs in hexadecimal. Display is prefixed with a H
11	Display outputs in binary

Instruction Set:

Note, all undefined operations will result in a clock halt.

Move Class Operations (00 XXX YYY)

Operation	Opcode	Description	Notes
MOV X, Y	`00 XXX YYY`	Copy the contents of register Y into register X	X & Y from the register table. Moving a register to itself will cause a halt except for the NOP instruction below.
MOV X, IMM	`00 XXX 111`	Copy the next byte of memory into register X	X from the register table.
NOP	`00 000 000`	No operation	Implemented as MOV A, A
HLT	`00 101 101`	Halt the clock (reset counters to release)	Implemented as MOV PC, PC
JMP	`00 101 111`	Jump to address stored in next byte of memory	Implemented as MOV PC, IMM
JC	`00 111 000`	Jump to address stored in next byte of memory when the CARRY flag is set	Implemented as MOV IMM, Flags The 3 least significant bits are decoded in hardware
JZ	`00 111 001`	Jump to address stored in next byte of memory when the ZERO flag is set	Implemented as MOV IMM, Flags The 3 least significant bits are decoded in hardware
JN	`00 111 010`	Jump to address stored in next byte of memory when the NEGATIVE flag is set	Implemented as MOV IMM, Flags The 3 least significant bits are decoded in hardware
JO	`00 111 100`	Jump to address stored in next byte of memory when the OVERFLOW flag is set	Implemented as MOV IMM, Flags The 3 least significant bits are decoded in hardware

Data Memory Operations:

Unless otherwise specified, all operations listed in this section act on the 256-byte data memory.

Load Class Operations (01 XXX YYY)

Operation	Opcode	Description	Notes
LOD X, [Y]	01 XXX YYY	Copies memory located at address held in register Y into register X.	
LOD X, IMM	01 XXX 111	Copies memory at address held in the next byte of program memory into register X.	
POP X	01 XXX 110	Decrements the stack pointer, copies memory at the new stack pointer address into register X.	

Store Class Operations (10 XXX YYY)

Operation	Opcode	Description	Notes
STO [X], Y	10 XXX YYY	Copies data in register Y into memory address held in register X	
STO IMM, Y	10 111 YYY	Copies data in register Y into memory at address held in the next byte of program memory.	
PUSH Y	10 110 YYY	Copies data in register Y into the address held in the stack pointer & increments the stack pointer.	

ALU:

ALU Operations (11 ΣΣΣΣ XX):
All ALU operations take a command and a single register which acts as the “A” input. The supplied register is updated with the result from the calculation ie: X = ALU(Σ, X). The ALU requires 4 bits to set the desired calculation, the remaining two bits are the least significant bits of the A, B, C or D register. As a result, the ALU is not able to operate on the program counter directly.

The slightly unusual ordering of the four command bits is because the ALU microcontroller is programmed to be bit-compatible with the (very) obsolete 74LS382 ALU chip which defines the order of the first 8 commands.

Single operand commands

Operation	Opcode	Description	Notes
LST X	`11 0000 XX`	Copies the last ALU result into register X	This is required to latch the ALU result. The four ALU command signals are only active for one microstep of an ALU operation and are 0000 at all other times.
ZER X	`11 1000 XX`	Copies 0 into register X	Used heavily for initialising registers
NOT X	`11 1001 XX`	Bitwise NOT	~X
INC X	`11 1010 XX`	Increment X	X + 1
DEC X	`11 1011 XX`	Decrement X	X - 1
RET X	`11 0111 XX`	Returns X	Lets the ALU be used a temporary register. **Note:** This isn’t that useful and may be changed to a test for equality operation in a future microcode update.

Some ALU operations require a second input which is supplied by the B register (hardwired to the ALU). The B Register can also be used as the A input. Ie ZER B, and XOR B are functionally equivalent operations.

Operation	Opcode	Description	Notes
SBX X	`11 0001 XX`	Subtract X from the B Register	B - X
SUB X	`11 0010 XX`	Subtract B from the X Register	X - B
SBC X	`11 1111 XX`	Subtract B from the X Register, with carry	X - B - 1
ADD X	`11 0011 XX`	Add B to the X Register	X + B
ADC X	`11 1110 XX`	Add B to the X Register, with carry	X + B + 1
XOR X	`11 0100 XX`	Bitwise XOR	X ^ B
OR X	`11 0101 XX`	Bitwise OR	X | B
AND X	`11 0110 XX`	Bitwise AND	X & B
LSH X	`11 1100 XX`	Left shift	X << B
RSH X	`11 1101 XX`	Right shift	X >> B