Architecture

Architecture

The computer is a two-address machine, meaning that each instruction includes two direct memory addresses from which to fetch the operan
eliminates the need for a register or accumulator to provide one of the operands. A triple-port memory is needed to support this: one port for t
instruction and two ports for the data.

All instructions execute in one single-phase clock cycle, so there is no required sequencing logic. Aside from the 4-bit condition code, there a
control signals. There is no instruction decoder, instead the instruction is wide enough to fit all 12 control signals directly, similar to a horizo
microcoded design. Many instructions are possible based on the many possible combinations of the control signals.

The datapath is 8-bits, but instructions are 32-bits. Instructions and data are both stored together in memory. There are 256 memory locations
holds a 32-bit instruction. The lowest 8-bits of each location (corresponding to the B address field of the instruction) are used for data and ca
or written to by the running program.

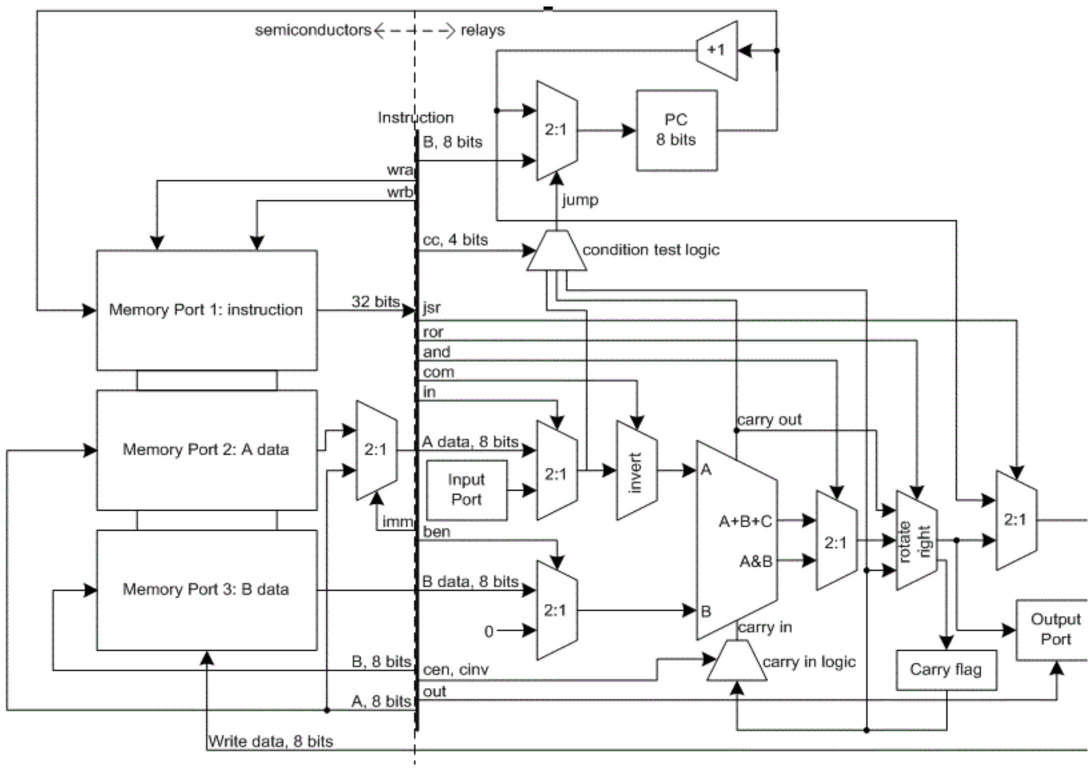
This is a "semi" Von Neumann architecture. Data and code are stored together in memory like a standard Von Neumann machine, but only pa
instruction can be accessed or modified by the program. Self-modified code is used to implement indexing and subroutine linkage by having
write to this part of an instruction. This is akin to early computers such as UNIVAC-I and Princeton IAS.

Originally it was intended to implement a clone of the 12-bit Digital PDP-8 minicomputer, but the result would have been a slower, more exp
computer (needing about 280 relays, with no I/O). One aspect of this original PDP-8 idea remains in the design: like the PDP-8, the ALU sup
and AND. Exclusive-OR and OR have to be implemented with a sequence of instructions.

It can be argued that the triple-port memory is "cheating" (meaning that early computers would not have used such a memory), but we are alr
by using semiconductor memory, so we might as well make the most of it.

Block Diagram

The memory, the immediate data MUX and the single-phase clock are implemented in a microcontroller (in silicon). The CPU, including AL
flag, input port, output port and condition logic is implemented in 83 DPDT relays.



The immediate data MUX should also be implemented in relays, but it was added later as an afterthought. Immediate mode is not strictly nee
can always store data in an addressed memory location. On the other hand, immediate mode saves memory and is convenient.

Another subtle bit of cheating is the ability to write back either to the A address or the B address. There should be a MUX to select between t
addresses, but it's implemented in the microcontroller as part of the memory.

PC (Program Counter)

There is an 8-bit Program Counter which points to the instruction in memory to execute. The program counter is normally incremented after
instruction, but it can be replaced with the B-field of the instruction for jumps.

The next instruction address (PC + 1) can be written to memory to record a subroutine return point.

The program counter has its own incrementer which requires 7 relays. This is cheaper and faster than the alternative of having the ALU incre
For the ALU to do it would require a 4 relay MUX to feed the PC through the ALU, plus at least 5 more relays to MUX the ALU control sig

otherwise deal with the non-single cycle / per instruction operation. An ALU operation could not have been done in the same cycle that the P incremented, so performance would have been halved.

Carry Flag

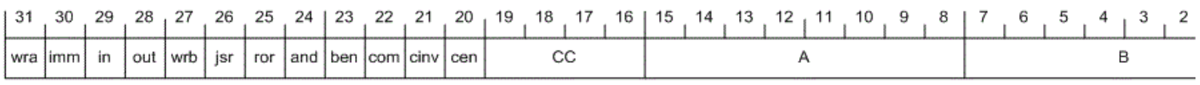
The carry-out bit of the ALU (possibly modified by the rotator) is saved into the carry flag after every executed instruction. In other words, al modify the carry flag.

Memory

The memory should be thought-of as having three ports: one for the instruction and two for data. When an instruction is read, the data from th given by the A and B fields of the instructions are also read. The instruction's results can optionally be written back to either the address give or the B field.

Instruction Format

All instructions have this format:



Key:

- **B**: B memory address field
- **A**: A memory address field or immediate data
- **CC**: Condition code
- **cen**: Carry (into ALU) enable (otherwise it's forced to zero).
- **cinv**: Carry (into ALU) invert.
- **com**: 1's Complement A operand
- **ben**: B operand enable (otherwise it's forced to zero)
- **ror**: Rotate ALU result right
- **jsr**: pc+1 is used as result instead of ALU result
- **wrb**: write result to B
- **out**: write result to output port
- **in**: replace lower 4 bits of A operand with input port
- **imm**: Use A field as literal data instead of reading data from A address
- **wra**: write result to A instead of to B

Instruction sequence

All instructions execute in a single cycle, and always follow this logical sequence:

1. First, the 32-bit instruction located at the address specified by the program counter is fetched from memory.
2. Next, the two 8-bit arguments, A and B, are fetched from memory from the addresses given by the A and B fields of the instruction.
3. If both 'wra' and 'wrb' are set, the clock is stopped and execution is halted until the user presses the 'step' or 'go' buttons.
4. The instruction is executed. The operation which is performed is determined by the control bits of the instruction. The data flow is as fo
    1. The operands for the ALU, A and B, are pre-processed:
        1. If the 'imm' bit is set, the A field of the instruction is used directly as the A operand, otherwise the data in the memory addı field is used.
        2. If the 'in' bit is set, the lower 4-bits of the A operand are replaced with data from the input port.
        3. If the 'com' (complement) bit is set, the A operand is inverted (the input port data will be inverted if this bit is set).
        4. If the 'ben' (B-enable) bit is set, the B argument is allowed to pass in to the ALU, otherwise it is set to zero.
        5. If the 'cen' (Carry-enable) bit is set, the carry flag is allowed to pass in to the ALU, otherwise it is set to zero.
        6. If the 'cinv' (Carry-invert) bit is set, the carry to the ALU is inverted. This happens after the carry enable gate, so if 'cen' is is one, then the ALU will get a carry input of one.
    2. The ALU performs two operations in parallel: Add: A + B + Carry and bitwise-AND: A & B.
    3. If the 'and' bit is set, the AND result is selected, otherwise the ADD result is selected.
    4. If the 'ror' bit is set, the result is rotated to the right by 1 bit. The carry flag will be placed into bit 7 and bit 0 of the result will rep output from the ALU.
    5. The carry flag is updated (either with the carry output from the ALU or bit 0 of the result if 'ror' was set).
    6. If the 'out' bit is set, the result is written to the output port.
    7. If 'jsr' bit is set, PC + 1 is used for the result instead of the ALU output.
    8. If the 'wrb' (write to b) bit is set, the result is written to memory, to the address specified by the B field of the instruction. If the 'v bit is set, the result is written to memory, to the address specified by the A field of the instruction.
5. The Condition Code field specifies various conditions to check, and if the result of this check is true, a jump takes place: the PC is repla contents of the B field of the instruction instead of with PC + 1