ARCHITECTURE

1) TYPES OF MACHINES

In this section we shall consider 4, 3, 2, 1 & 0 address machines in term of instruction formats and execution times.

4 Address Machine

OPCODE	RESULT	OPERAND 1	OPERAND 2	NEXT INSTRUCTION

3 Address Machine (eg CDC Cyber series, word size 60 bits)

If we assume that memory is addressed segmentally, we can remove the NI address

OPCODE	RESULT	OPERAND 1	OPERAND 2

2 Address Machine (eg PDP 11 - VAx)

Assume that the result is placed in operand 1:

OPCODE	OPERAND 1	OPERAND 2

1 Address Machine (eg IBM 360, Unvac 1100)

Assume that operand 1 is an accumulator.

OPCODE	OPERAND

Zero Address Machine (eg ICL 2900)

Assume that all operands are on a stack. We require some instructions to unload the stack.

OPCODE

As an example of execution times, we will write:

$$A := B + (C * D) / E$$

in each format, and then estimate the time required by each. We assume that memory accesses (including instruction fetches) take 1 unit of time, and processing time is negligible.

3 Address

A := C * D	MULT A C D	4 accesses	12 units
A := A / E	DIV A A E	4 accesses	
A := A + B	ADD A A B	4 accesses	

2 Address

A := C	LOAD A C	accesses	15 units
A := A * D	MULT A D	accesses	
A := A / E	DIV A E	accesses	
A := A + B	ADD A B	4 accesses	

1 Address :

LOAD C	2 accesses	10 units
MULT D	2 accesses	
DIV E	2 accesses	
ADD B	2 accesses	
STORE A	2 accesses	

0 Address

LOAD C	2 accesses	13 units
LOAD D	2 "	
MULT	1 "	
LOAD E	2 "	
DIV	1 "	
LOAD B	2 "	
ADD	1 "	
STORE A	2 "	

Thus the 1 address machine (accumulator) is fastest

2) INSTRUCTIONS & WORD SIZE

Assume a one-address machine. Each instruction must have:

OPCODE

OPERAND

INDIRECT ADDRESSING BIT

MODIFICATION (INDEXING) BIT.

In pre-modification, the index register is added to the operand to give the indirect address. In post-modification, the index register is added to the contents of the operands address.

Thus in pre-modification, we have an array of pointers, whereas in post-modification we have a single pointer to the start of an array.

It is often useful to have more than one accumulator and index register. If we have many accumulators this constitutes a stack. We can indicate which accumulator by numbering from 0 (ie 8 accumulators 3 bits) and which index register by numbering from 1 (0 means no indexing)

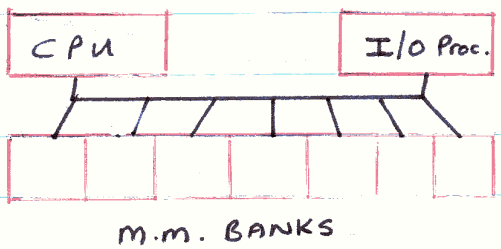
So, for example, a one-address machine with 16 accumulators, 15 index registers, 2^{20} address space and indirection requires:

Opcode	7 bits	36 bit instruction
Indirect	1 bit	
Address	20 bits	
Accum	4 bits	
Index	4 bits	

An alternative is to use variable length instructions. For example, for single byte immediate loads we only need 8 bits for the operand, not 20. We can pack instructions into multiples of 1 byte with different instructions taking different amounts of storage. The program counter must then be incremented by the instruction size each time.

Most machines have their registers mapping onto the first locations in memory. A program swap thus involves copying all the registers to memory, which is time consuming, particularly after an interrupt. A possible solution is to have banks of registers which can be swapped rapidly by switching.

All instructions cause MM accesses. If cycle stealing is used the I/O channel will have priority and will steal from the CPU, bringing it to a standstill. By allowing multiple paths into memory, this can be avoided, provided different banks of memory are being accessed. This also allows instruction fetches to be overlapped with execution, allowing considerable speed improvement.



INSTRUCTION

1) Fetch Execute
2) Fetch Execute
3) Fetch Execute

3) BUS STRUCTURES

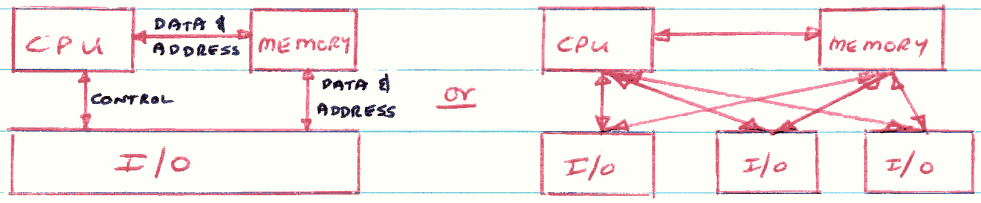
A bus is a connection between two components of the computer along which data passes. Busses can be serial or parallel (8 bit parallel = byte serial). The bus size is not necessarily the same as the word size. The number of fetches is a function of the degree of parallelism of the bus and the size of the fetch.

A normal bus contains address, synchronisation, sense and data lines. For a mainframe these will be in parallel, making a bus size of over 100 bits wide.

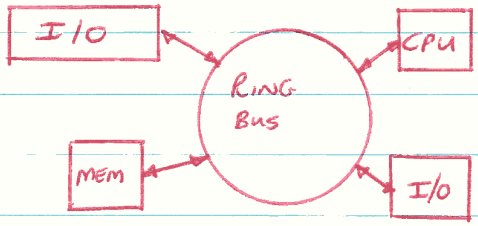
The system can have a control bus, whose function is:

- operation synchronisation
- CPU signalling to components
- relaying fault signals, interrupts and resets
- relaying inter-component timing signals

Some machines use seperate busses:



OR



Separate buses are faster as there is less I/O contention, but ring structures are more flexible as it is easy to interface new units.

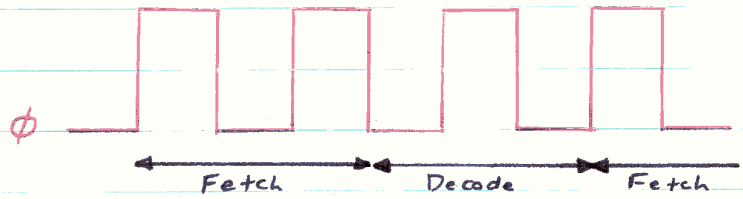
4) CPU TIMING.

The timing of instructions takes place automatically. The major instruction sections are:

- fetch instruction
- decode instruction
- fetch operand
- execute instruction
- store results (if any)

} can be repeated in any one instruction

Each stage is made up of minor machine cycles driven by clock (± 20 - 50 nanosec/cycle).



This is synchronous operation. Alternatively we can have asynchronous machines where each instruction signals the next when it is completed.

5) INPUT / OUTPUT & DATA COMMUNICATION

Each I/O device is connected to a controller via a port. The port is a connection slot and determines the number of devices a controller can support. The functions of the controller are:

- make the computing system and I/O devices compatible (ie same coding, block size, etc)
- convert signal levels, timing, etc.
- convert serial to parallel and vice-versa
- provide processor interrupts
- provide I/O status to CPU
- handle I/O errors.

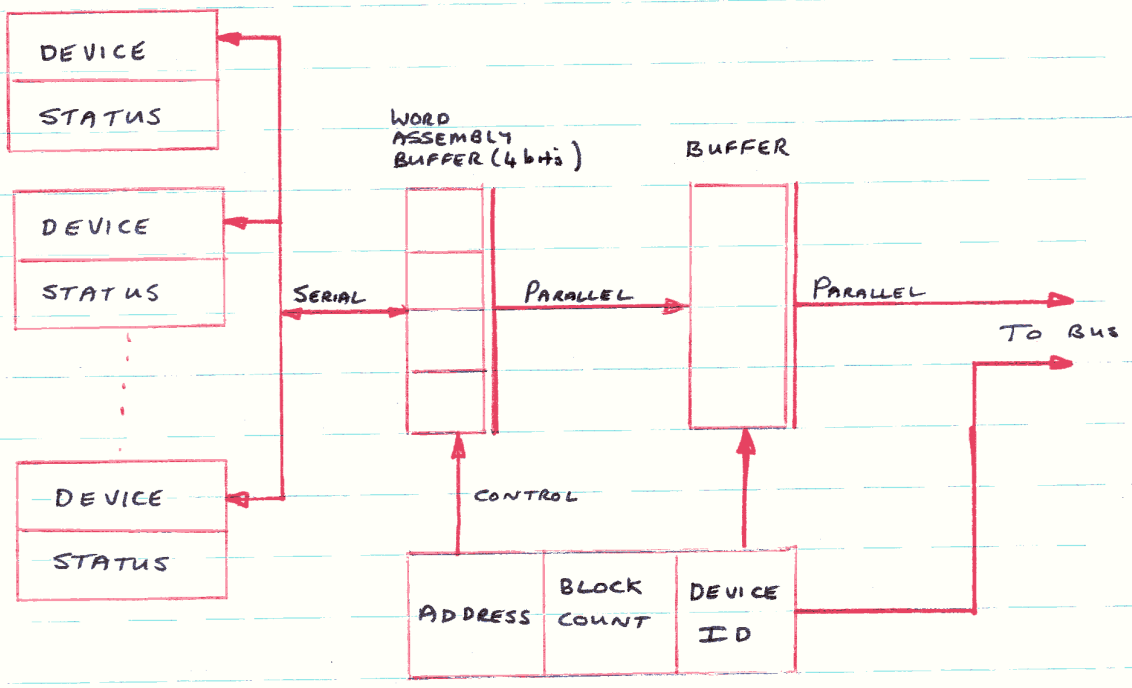
Typically, an I/O interface consists of input, output and status registers, each of which is assigned a unique address which is written to and read from by the O/S. On small machines a block of memory (2K or 4K) is usually reserved for I/O addressing.

The channel can be connected to the CPU or directly to memory (DMA). In the DMA case, the CPU sets up the transfer and the channel takes over, interrupting the CPU upon completion (low priority interrupt).

In the (DMA) method the I/O controller needs an address register and possibly a block count register. The controller puts the address on the bus followed by the state (read / write). The BC is decremented as the transfer takes place until 0 when it stops.

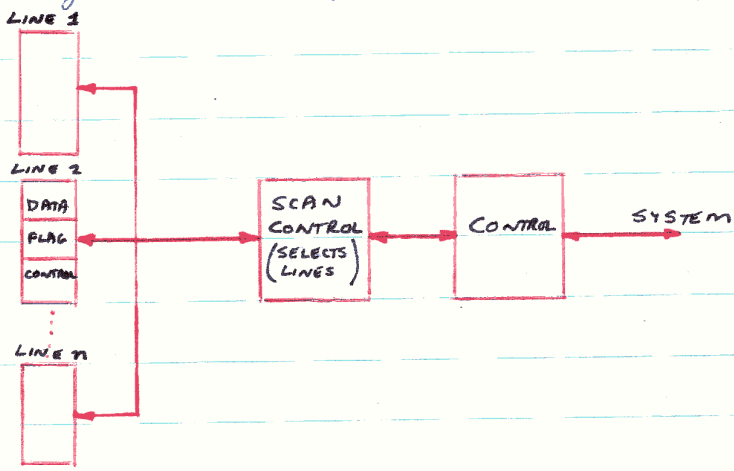
There are two types of controller - block multiplexers and byte multiplexers.

Block multiplexers are for large fast devices and transmit blocks at a time. They may perform multiple operations - concurrent seeks and mutex reads / writes.



One device is selected at a time, and a transfer of a block of data takes place. The device is then deselected and the next device selected.

A byte multiplexer is for slow devices. It may transmit a block or byte at a time, and is suitable for terminals, printers, etc.



The control unit acts as the coordinator with memory, while the scan control is responsible for organising the scan of all the communication lines. The process is:

- scan each flag in turn (this tells whether the device is awaiting service)
- if ready then perform a byte transfer (the data is passed from to main memory from to the control module)
- move onto next line

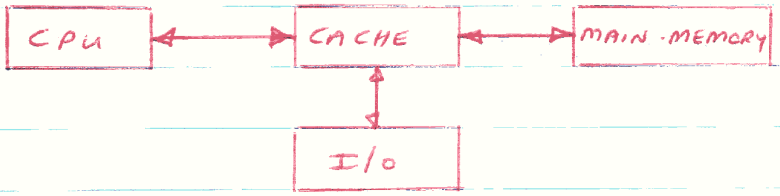
The address to which the byte is sent must be updated and the block count decremented. These are usually held in main memory (unlike block multiplexer) as each line needs a set.

The number of lines possible depends on the baud rate and the time taken by the scan control to access every line and return to that line before the data is lost.

Communication between devices can be of two types - synchronous and asynchronous. In synchronous data communication both devices are synchronised by some external signal. In asynchronous the data needs a prompt (pulse) from the sender to inform the receiver of the start or end of transmission.

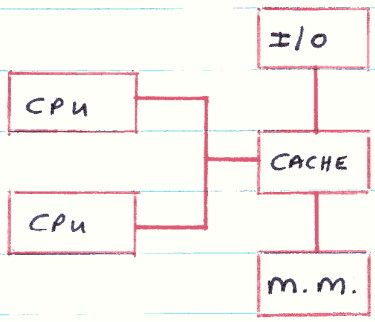
There are two major methods used by manufacturers to speed up machines. These are cache memory and pipelining.

Cache memory is very fast (100 ns or less) and is used for holding data currently being used either in processing or in I/O transfers. It is handled as virtual storage using demand paging.

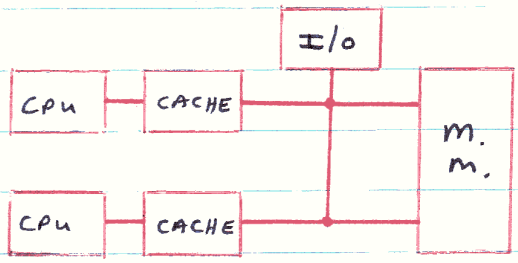


The size of the cache is small (about 8K) with about 1 path for every 128 words. Pages are very small (4-8 words). The transfers from main memory are word serial (ie 32 bits/pa

The Unwac uses 1 cache for all processors whereas the IBM uses one for each processor. The IBM is thus faster, but the Unwac works better with shared data.



UNIVAC



IBM

A pipeline consists of a number of processing stages called pipeline stages or segments, each capable of doing a part of its own work in a fixed cycle time.

A simple version of a pipeline overlaps instruction and operand fetch. A more advanced one will overlap

- instruction fetch
- operand fetch
- indirect addressing
- indexing
- execution.

INS. FETCH → INDEXING → OPERAND → INDIRECT → EXECUTE

This implies a fixed time per stage. Some computers have more advanced pipelines with many instructions. These have each instruction being decoded with operands accessed all ready for execution. Problems can sometimes arise with instructions that modify the operands of other instructions, etc.

6) UNIVAC 1100 CHARACTERISTICS

CPU

1 - 4 per system
128 control registers
4 level instruction overlap (pipeline)
50 ns cycle time (200 MHz)
200 ns basic instruction cycle time
Quantum timer & Jump history stack

MAIN STORAGE UNIT (MSU)

512K - 4096K 36-bit words (9 bit bytes) in 512K increments
650 ns cycle time for 4 words (read) and 1-word write
900 ns cycle time for 1-word partial write
bus is 43 bit including error correction

STORAGE INTERFACE UNIT (SIU - CACHE)

High speed buffer, 1 or 2 per system
8K - 32K bit words (4K increments)
200 ns cycle time

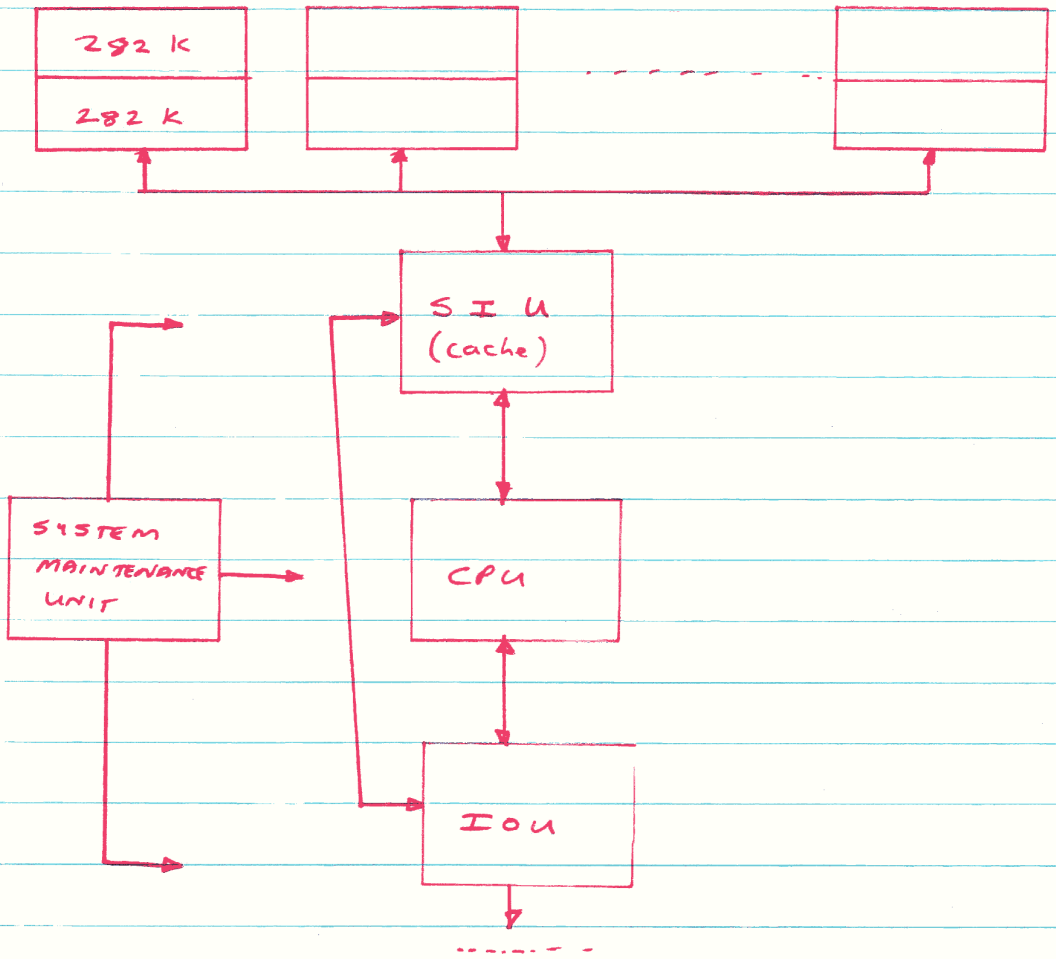
INPUT - OUTPUT UNIT (IOU)

1 - 4 per system
Byte multiplexer channel module (200 K bytes/sec)
Block multiplexer " " (1670 - 3000 K byte/sec)
Word channel module (500K words/sec)
Aggregate transfer rate 2 megawords/sec

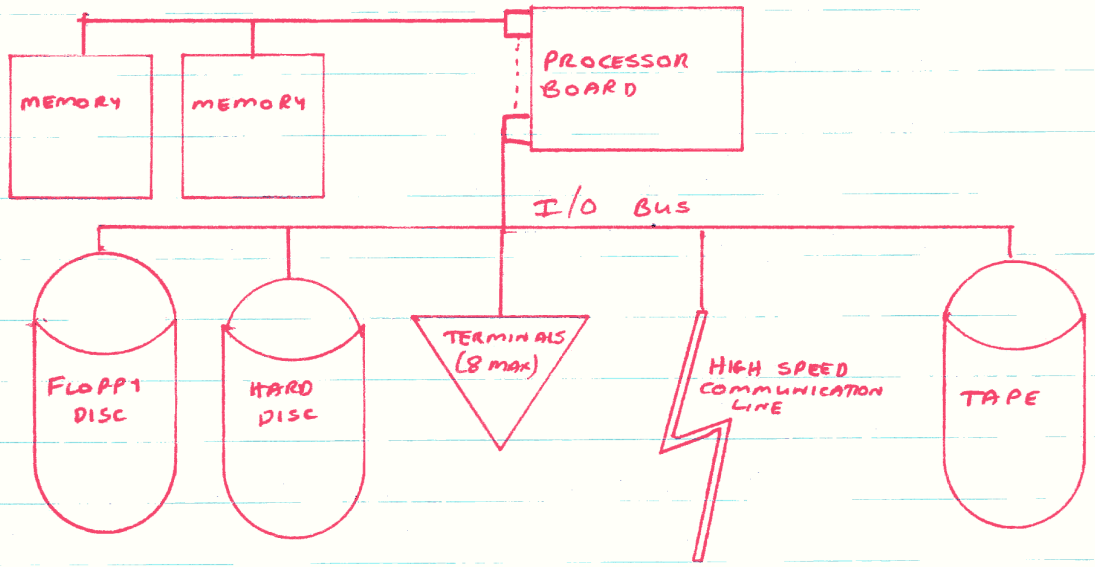
SYSTEM TRANSITION UNIT

Partition with 3 options
Initial read / write recovery capability
Operators panel

CONFIGURATION



7) NCR TOWER CHARACTERISTICS



PROCESSOR

Motorola 68000 @ 10 MHz
32 bit internal, 16 bit external architecture
8 arithmetic registers, 4K RAM scratchpad
24 bit address bus ≅ 16 MB (actually 32 bit : 24+8 for page table)
Rgsta - register add = 400 ns
32K ROM startup & diagnostics, error correction
2K additional for page table.

I/O BUS

IEEE 796 Multibus, 8 or 16 bit transfers @ 5 MB words/sec
Thus CPU requires 2 transfers per 32 bit instruction.

MEMORY

16 bit word with 6 error correcting bits.

DISC CONTROLLER

Floppy and Winchester share 1 DMA port

Floppy: 250K/sec, capacity 1MB.
Average access time 147 ms ~ 7/sec

Winchester: 32-40 MB @ 5 MB/sec
Average access time ~ 39 msec ~ 25/sec

Controller: Can support 2 of each, total 84 MB
Ave access (39) 20 ms/device
Transfer rate (5) 9.6 MB/sec

BYTE MULTIPLEXER

Controls up to 8 terminals, not DMA but interrupt driven

8) THE CONTROL UNIT

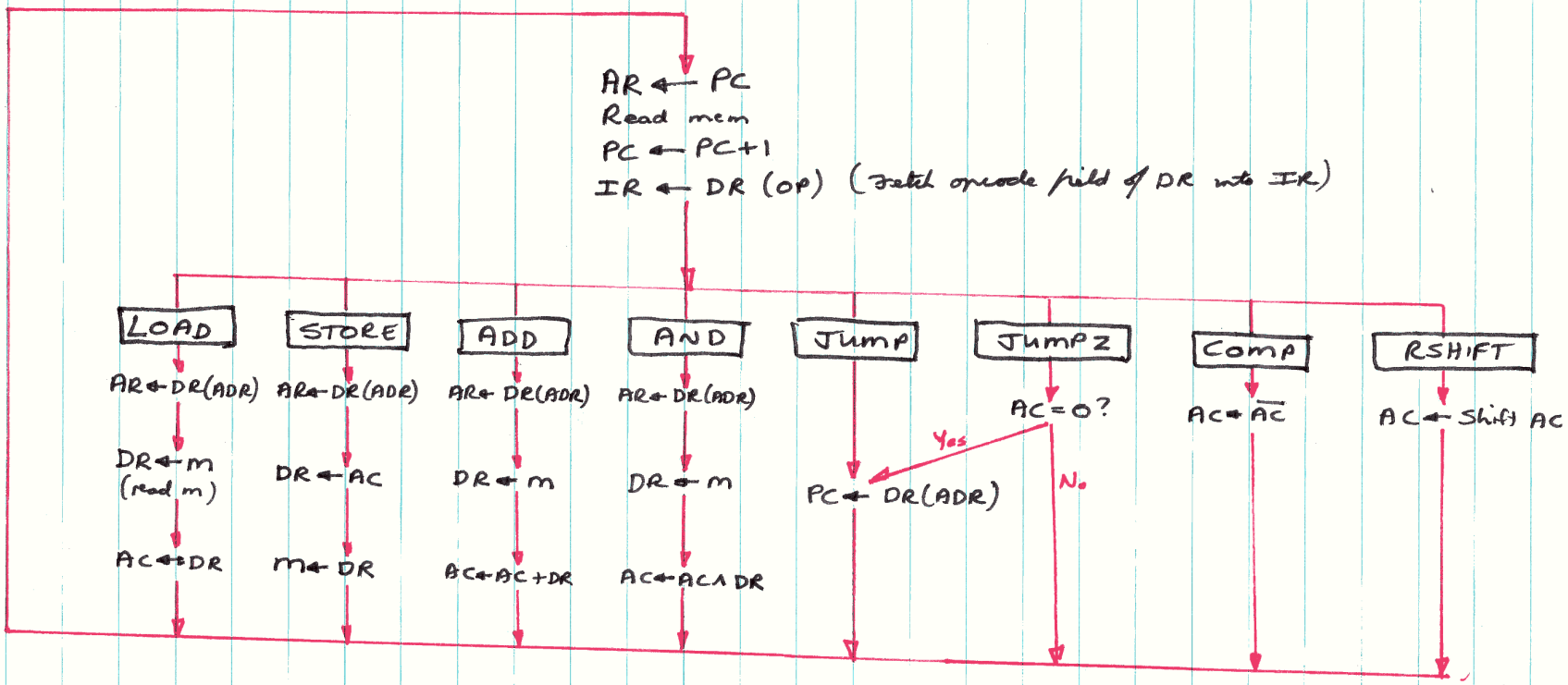
The basic sequence of operations is:

Repeat indefinitely:
- transfer program counter to memory address register
- fetch instruction from memory
- increment program counter and decode instruction
- transfer operand address to memory address register
- fetch operands from main memory
- perform operation

Consider the following instruction set:

LOAD x	$AC \leftarrow m(x)$
STORE x	$m(x) \leftarrow AC$
ADD x	$AC \leftarrow AC + m(x)$
AND x	$AC \leftarrow AC \wedge m(x)$
JUMP x	$PC \leftarrow x$
JUMPZ x	if $AC = 0$ then $PC \leftarrow x$
COMP	$AC \leftarrow \overline{AC}$
RSHIFT	$AC \leftarrow AC / 2$

The operation of the CPU is shown schematically overleaf. Each instruction, if it is to take a standard length of time, may need null instructions (eg jump and comp)



AC - accumulator
DR - data register
DR(ADR) - " " address field
DR(OP) - " " operand field
M - memory
AR - address register
IR - instruction register
PC - program counter

9) MICROPROGRAMMED CONTROL

This is a method of control design in which the control selection and sequencing is stored in a random access array called control memory. The instructions are called microinstructions and are grouped in microprograms.

Each machine instruction acts as a real-time interpreter for the instruction. The set of microprograms that interprets a particular instruction set or language L is called an emulator for L. This means the microprogram control is made of signals, and by changing the sequence of these signals we can get a new instruction.

The advantages of microprogramming are:

- "soft" computers
- variable instruction sets
- ability to emulate other machines
- ability to change speed
- cheap upgrading

The disadvantage is that soft instructions are about 5 times slower than hardwired instructions.

In its simplest form, a microinstruction has two parts:

- a set of control fields which indicate the control lines to be activated
- an address field which indicates the address in the control memory of the next microinstruction.

In general microinstructions can have a degree of parallelism so nowadays a microinstruction is divided into a set of control fields, each of which is associated with a set of

microoperations any one of which can be performed simultaneously with the micro-ops in any of the other control fields. A microinstruction typically is:

OPCODE	OPERAND 1	OPERAND 2	N.I. ADDRESS

The requirement of 1 bit per control line leads to large microinstruction words, often 90-180 bits. To prevent this, we can have vertical microinsts. These have:

- short formats
- limited degree of parallelism
- encoding of control information
- slower.

The long words are called horizontal micro-instructs, and have - long formats
- high degree of parallelism
- little or no encoding
- faster

If we have to produce a microcode for our small CPU we must first produce a microprogrammed emulator in symbolic form: Eg

FETCH AR ← PC
READ M
PC ← PC + 1
IR ← DR (OP)
GTO IR

LOAD AR ← DR (ADR)
READ M
AC ← DR
GOTO FETCH

STORE AR ← DR (ADR)
DR ← AC
WRITE M
GOTO FETCH

ADD	AR ← DR (ADR)	AND	AR ← DR (ADR)
	READ M		READ M
	AC ← AC + DR		AC ← AC ∧ DR
	GOTO FETCH		GOTO FETCH
JUMP	PC ← DR (ADR)	JUMPZ	IF AC ≠ 0 THEN GOTO FETCH
	GOTO FETCH		PC ← DR (ADR)
			GOTO FETCH
COMP	AC ← $\overline{AC}$	RSHIFT	RSHIFT (AC)
	GOTO FETCH		GOTO FETCH

The instruction can often be implemented using a 12 bit control field and a next instruction address. The microprogramming method is highly flexible, so new instructions can be added, etc.