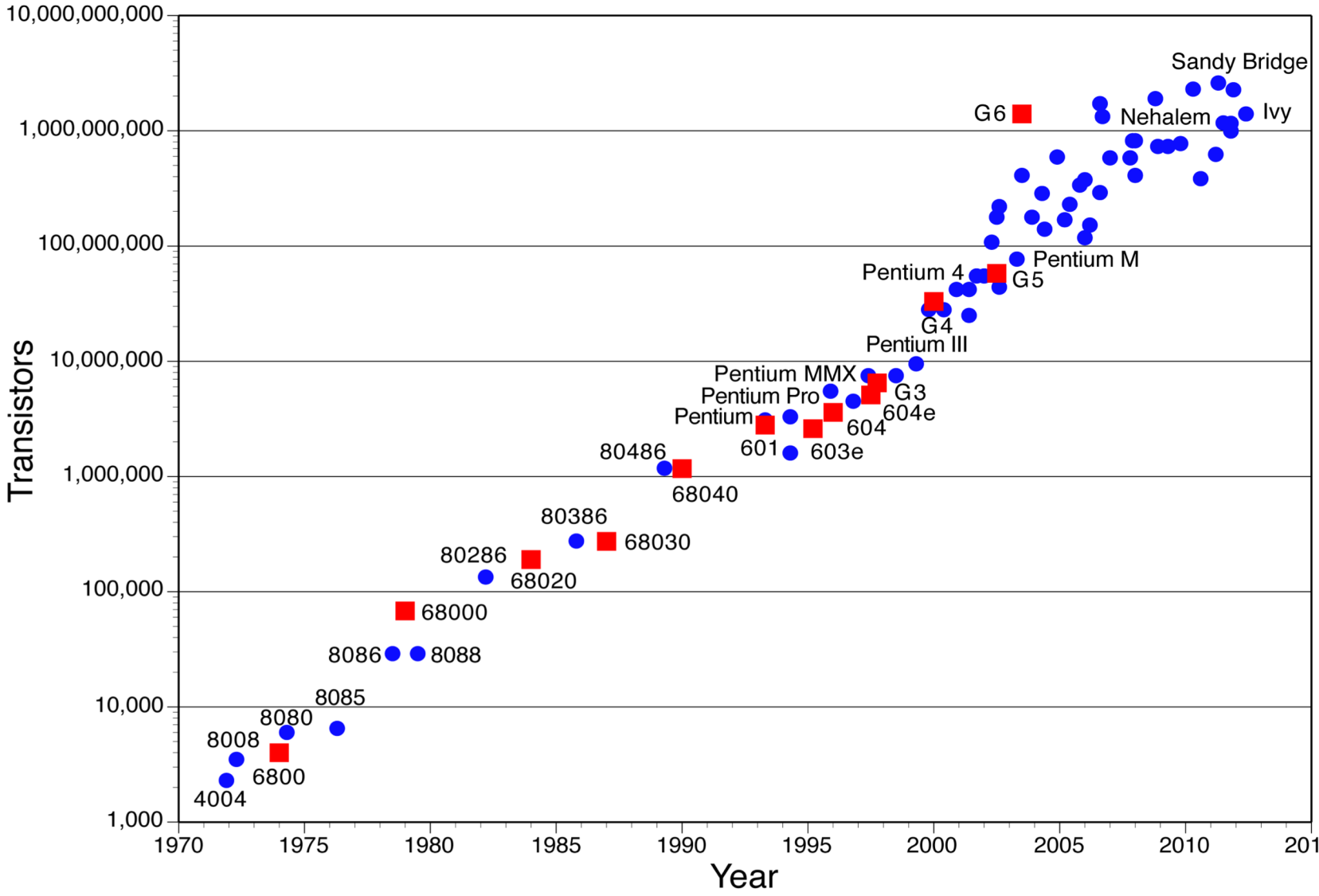
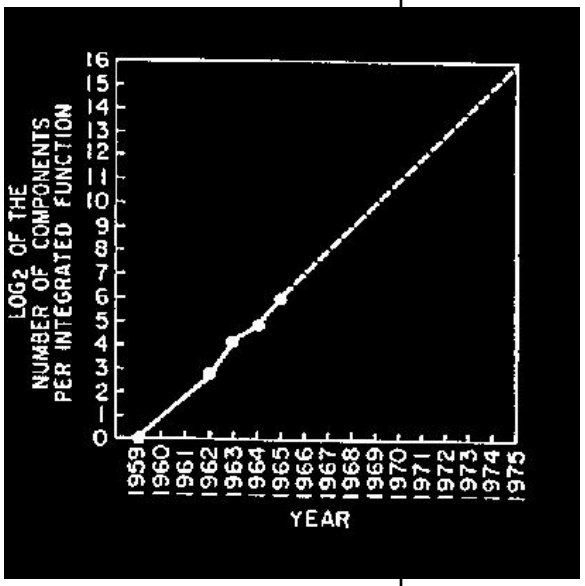
ISA Background

Processor performance Growth



The fact that there are no updates since 2015 is in itself diagnostic

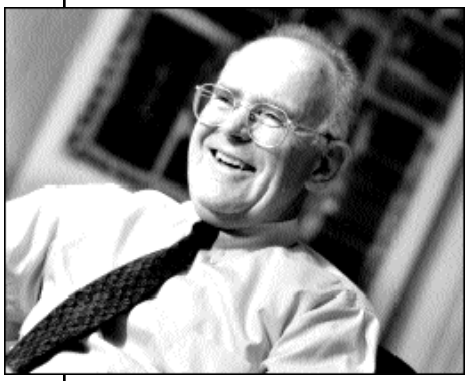
Improvements

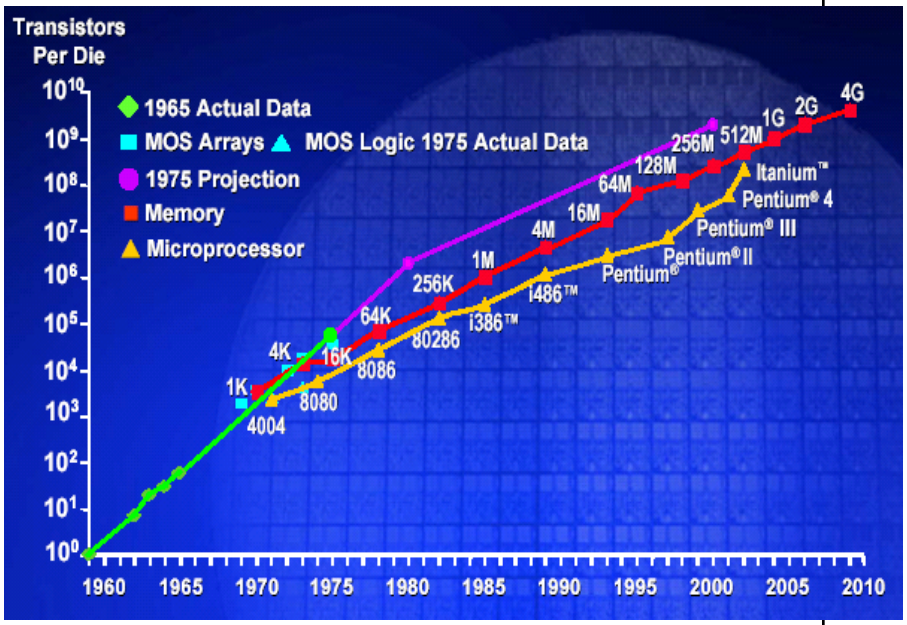


Data

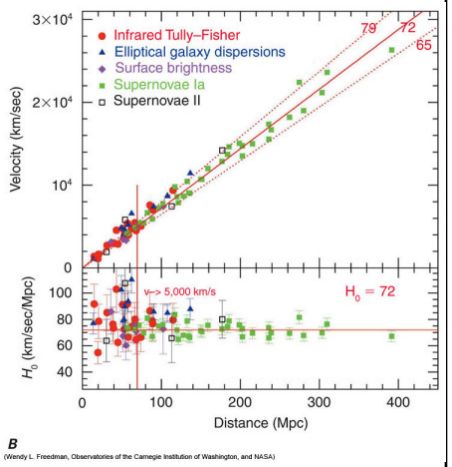
Moore's Law

Second most audacious extrapolation I know
Behind Hubble





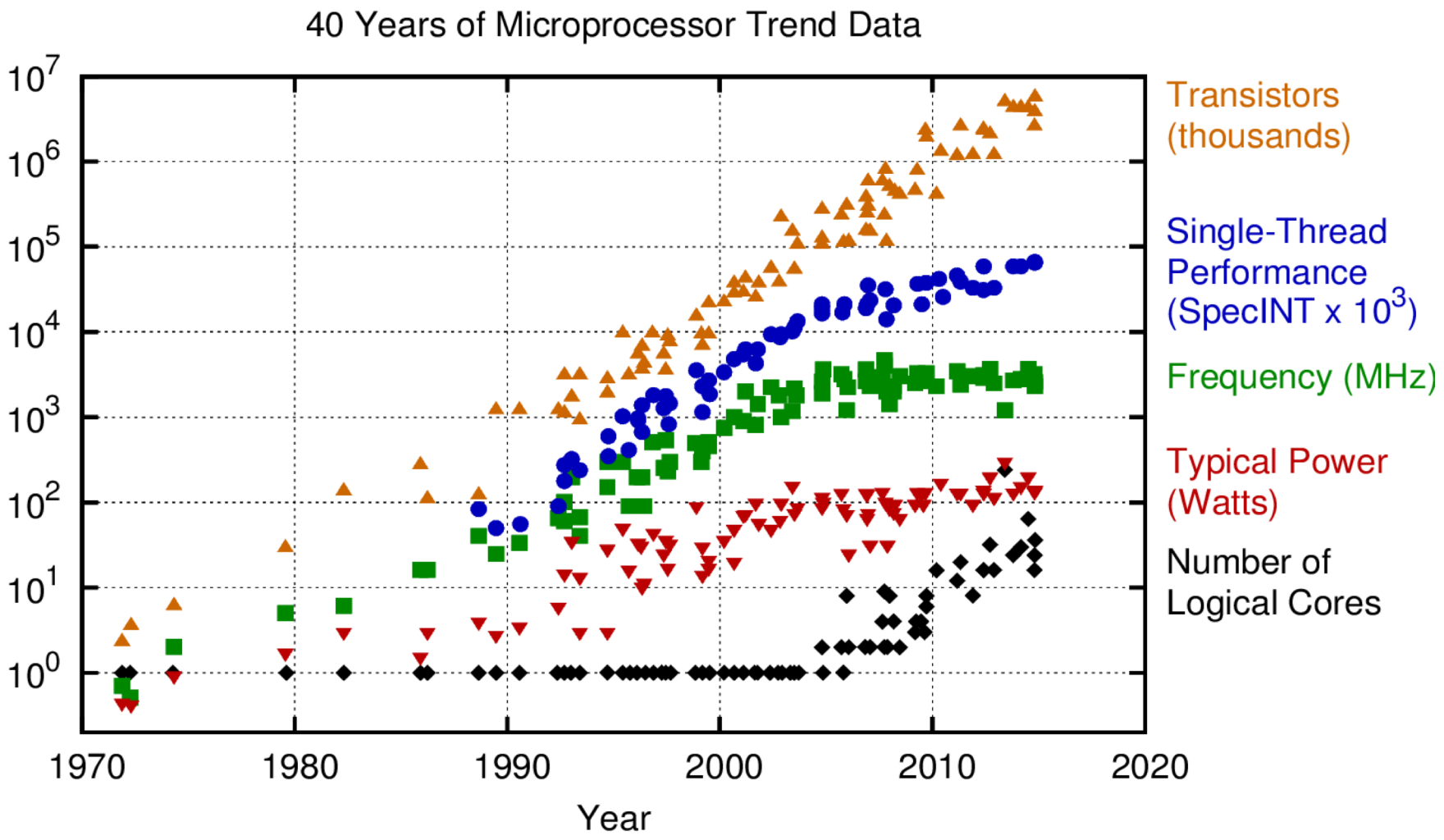
Number on transistors / cost-effective integrated circuit double every N months
$(12 \leq N \leq 24)$



Hubble's Law

ISA Bgrnd

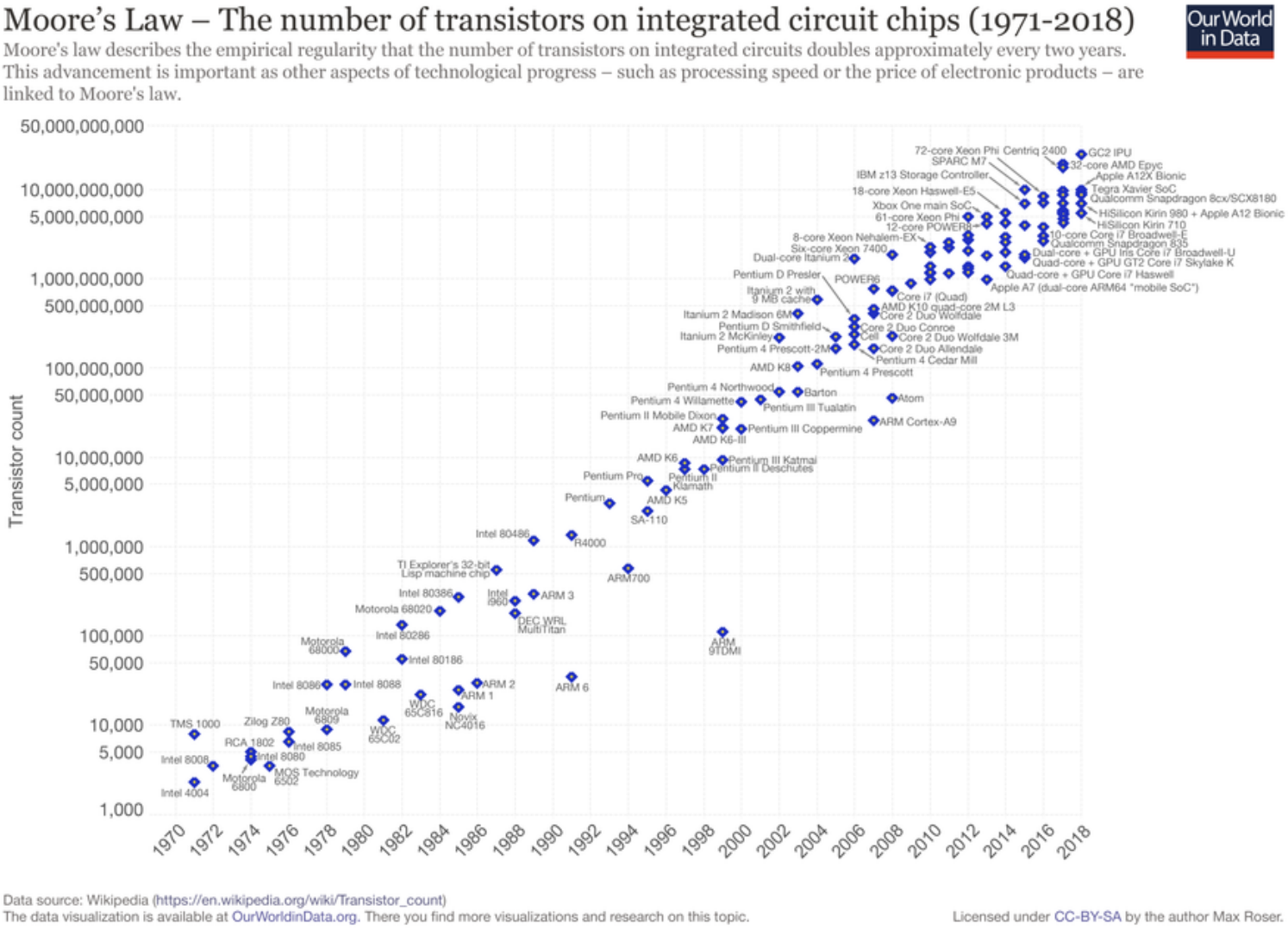
Moore's Law



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Increasing performance is driven use of extra gates to, for example add cores.
Clock speed flattens around 2000
power limits

Moore's Law



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Moore's law still applicable

Moore's Law

Clock rate. VAX 11/780: 5 MHz
Current rate: 3.5 GHz
Ratio about 700

Old numbers

Moore's Law about 100,000.

Performance about 10,000

From about the time clock speed flattened

Not all about clock rate – architecture drives the rest.

Multi-core complicates this analysis

VAX 11/780 about £150,000
Intel server about £1000
Price ratio about 150.

So performance per price about 1,000,000
Some effect of numbers sold.

Last thirty years clock rate has driven improvements (500:10).

What is the effect of Moore's Law.
Double transistor count.

Small units, shorter connection paths, faster switching. Higher clock speed.

More transistors – more complex architecture, more memory
Clock speed is not computer speed.

Word

Computer memory

Consists of a set of bits, units which can have a value of 0,1 – these bits are organised into words.

The word length is a decision for the system architect.

Must the word length be a factor of two.

PDP 8 had 8 bit words.
Intel chips have moved from 16 bit words to 32 bit words and now to 64 bit words.

DEC PDP 10 had 36 bit words
CDC 7600 had 60 bit words
Baby had 22 bit words

Modern systems standardised on powers of 2.

Can use 3 state units: -1, 0, 1. “trits”

Meaning

Interpretation

What does a set of 1's and 0's mean?

For the system architect to decide.

Need to represent instructions for the computer to execute.
Data for the computer to operate on.

The data words are straightforward and covered by standards.

The instructions are specific to a particular machine

opcode

Typical instruction Add $r1, $r2, $r3

$r1 <- $r2 + $r3

So we need some part of the word to tell us the instruction and some part to give us the addresses of the three registers.

Registers

Registers

Most of the memory in a computer runs much slower than the system clock.
To ensure maximum performance we need some memory which runs at the speed of the processor clock and will allow the processor to perform and operation every

The data words are straightforward and covered by standards.

The instructions are specific to a particular machine

So the instruction word of an instruction set is split into fields, where each field corresponds to a different part of the

Operations

Instruction set

How many instructions do we need for a computer

Say 90 – (we can see that is not too big). Needs 7 bits to encode the instruction.

That leaves 9 bits, so three per register. Or 8 addressable registers.

If we want more registers we might for two operand instructions.

Add $r1, $r2

$r1 <- $r1 + $r2

4 bits and so 16 registers are addressable.

There is a trade off between number of instructions, type of instructions and number of registers.

Moving data to the registers

```
LDR    $r1, A
```

Load register 1 with the contents of the memory location A.

As before we assume, that 7 bits encode the instruction and 4 bits encode the register.

That leaves 6 bits for an address – only 64 locations.

How can we address a reasonable amount of memory?

(note the argument does not change for a 32 bit word machine – in particular it is normally stated that 4 GB of memory are addressable and that requires 32 bits)

Two options:

Longer instructions
Different addressing modes.
Modified register architecture.

Long words

Variable instruction length.

```
LDR    $r1, A
```

Suppose when we see a load instruction we known that we need to look in the next word to get the address

We then have some instructions we require more than 1 word.

Machines in the 60's and 70's had such instructions and the intel chips still continue this tradition.

The problem comes when we want to introduce instruction pipelining or multi-issue cores.

But we cannot tell if a particular word corresponds to an instruction or an address until we have decoded all the instructions up to that point.

Fixed instruction length (and fixed instruction format) are useful in the production of high performance computers.

Speed is context dependant

Performance for a resource user may be different from a resources provider.

For a user it is likely to mean the time from job submission to job retrieval.

For the provider it is more likely to be defined in terms of the total amount of CPU delivered to the customers in an accounting period (day/week/month).

Possibly scaled by cost

Can be antagonistic

The user is only interested in the performance of one thing … ***their programme***.

The provider is only interested in the performance of
… ***their resources at their job mix.***

Neither can measure those things on all possible systems.

Both are likely to end up, using other measurements as proxies for their requirements.

Instruction speed looks like a good general purpose measurement.

Clock Speed

Instructions per Second

One way of quantifying the performance of a chip is via the clock speed. 2.4GHz, 2.7GHz, …….

Intel/AMD would argue for two chips which differ only in clock speed it does make sense

But how many ticks of the clock does it take to execute one instruction?

Because of this ambiguity people also measure speed in **instructions per second** (or more normally Millions of …)

Manufacturers quoted the processors **MIP** rating.

However:

- Not all instructions take the same length of time
- Some instructions (instruction sets) do more work

Has often been ridiculed as

Meaningless **I**ndicator of **P**rocessor **S**peed

MIPS is application specific

What interests us is

"How much work we can do per second"

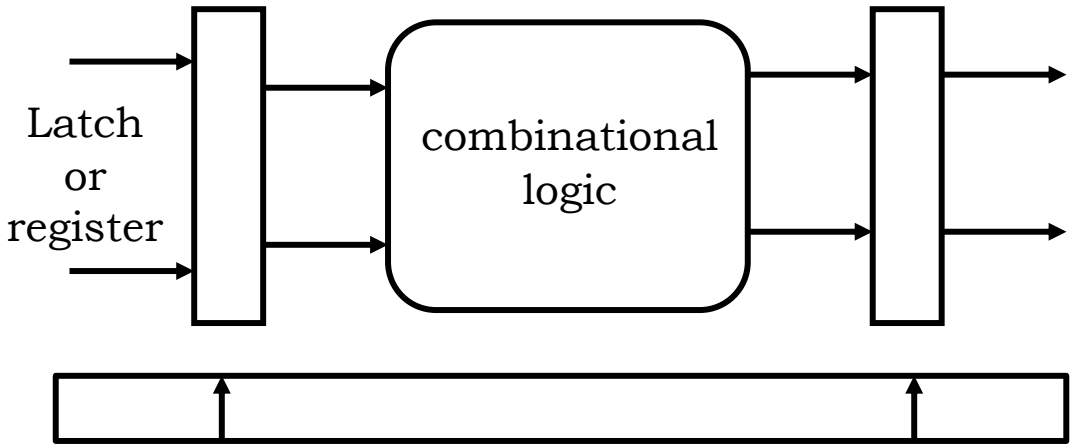
MIPS is not useful

Floating Point Operations per Second

This lead to the idea of measuring how many useful instructions can be completed per second.
The ones which gained favour were
Transactions … for database applications
FLOPS … for numerically intensive calculations
SPEC standardised set

Clock Cycle

Synchronisation between different parts of the system. Combinatorial logic does the work. Registers hold the intermediate results.



Speed is given by the speed of the slowest unit

The registers mean that the inputs to one unit remain stable, even when the outputs of the previous unit are switching.

Anatomy of an Instruction

What happens when a computer executes an instruction such

add $t0, $s0, $s1; Single instruction

add $t1, $s2, $s3;

Finish instruction
For the next one

The program counter PC set to the next instruction.

instruction fetched

instruction decoded

two registers added to give a third

By increment or jump

Even for integers the last stage clearly has a time structure because when adding, you need to know if the previous position had a carry.

Four general purpose registers
A program counter 15bit
1 bit carry register

The Nova called registers accumulators

Early

Early computers had a simple instruction set

NEG	Invert Source Accumulator (SAC) and place in destination accumulator (DAC)
MOV	Move SAC to DAC
INC	Increment SAC and store in DAC
ADC	Add ones compliment of SAC to DAC. Store in DAC
ADD	Add SAC to DAC. Result in DAC
SUB	Subtract SAC to DAC. Result in DAC
AND	Logical AND SAC & DAC. Result in DAC
JMP	Jump to an address
JSR	Jump to subroutine (first store PC in AC3)
ISZ	Increment and skip on zero
DSZ	Decrement and skip on zero
SKP	Skip unconditionally
SZC	Skip on zero carry
SNC	Skip on non zero carry.

Early

Also some I/O specific ops and some addressing modes

Direct – access the location pointed to

Indirect – access the location pointed to and use that as an address to access the instruction/data

Rather small number of instructions

Complex Instruction Set Computers (CISC)

People started to build machines which had more complex instructions which would do more things.

Limited memory so instructions which would do many things could save memory.

Complex instructions made it easy to implement high level instructions.

A feeling that by implementing "high level" instructions in hardware the computer would run faster.

In some places this worked.
Floating point processors

A number of reasons

In the resulting computers some instructions would run in just a few cycles while others might take many hundreds.

VAX complexities

The add subtract instructions mostly came in a 2 register and a 3 register version

INSQUE insert in queue

REMQUE remove from queue

MOVTC move translated characters

which moves characters defined in a translation table. The first six registers are filled with suitable parameters.

BPT break point – for debugging.

BBSSI branch on bit set and set interlocked shared access to data structures

Programming the Nova was easy.

Programming the VAX was impossible. (Well nearly!)

.

DUMP	start address, end address
XFC	Extended function call extend the instruction set (microcode)
LDPCTX	Load process context
SVPCTX	Save process context

If a machine allow multi threaded programming, then when one thread is removed from operation but before the replacements starts, “the context of the thread must be saved”. State of the registers, state of programme counter, state of the hardware stack., ……

The ***VAX*** did that with one assembler language instruction

Although the instruction might be microcode and have to be translated into simpler instructions to execute

Reduced Instruction Set Computers

In the 1970's it became clear that using the increased word length to increase the number of complexity of then instruction set was not as useful as it had seemed.

It was hard to create compilers which used the full set of instructions.

In a famous paper Patterson showed the VAX

INDEX Check array bounds on multidimensional arrays

Could be performed faster using simple VAX instructions.

The result was a move back to simpler instruction sets

"The case for the reduced instruction set computing." Pattersoin

By Patterson and Hennessey (among others)

80x86 – is not RISC (at least externally).

Historically goes back to CISC era – backward compatibility means must stay with the set.

But CISC is translated to RISC internally and the chip works as a RISC chip!

The lead time means that implementations lag research

Advantages

Denser encoding
smaller code size
- better memory utilization,
- saves off-chip bandwidth,
- better cache hit rate

simpler compiler
no need to optimize small instructions as much

Cache hits later

Disadvantages –

Larger chunks of work
compiler has less opportunity to optimize (limited in fine-grained optimizations it can do)
More complex hardware
translation from a high level to control signals and optimization needs to be done by hardware

The optimum point changed – not just fashion and knowledge but also the existing technology

Semantic Gap

Where does the ISA fit

Close to High level Language (HLL)
small semantic gap
complex instructions
simple compiler – at the time compilers not good enough

Close to hardware control signals
large semantic gap
simple instructions
complex compiler

RISC motivation

Registers and Memory

Computers have registers which the CPU can easily operate on.
They are the fastest memory. Access is a clock cycle or even less.
They are few of them so access is simple.

It is possible for the CPU to operate on the memory directly, but this has certain overheads.

There needs to be a path from all of memory to a number of different places in the CPU (especially if we wish to pipeline). Complexity & cost.
Memory accessed take more than a single clock cycle, so parts of the CPU must be able to work at different rate – again complexity and hence cost.

Computers may be classified by the way the CPU interacts with memory and registers.

Classification

Instruction set taxonomy

Stack
Accumulator
Register-register Load-Store
Register-memory

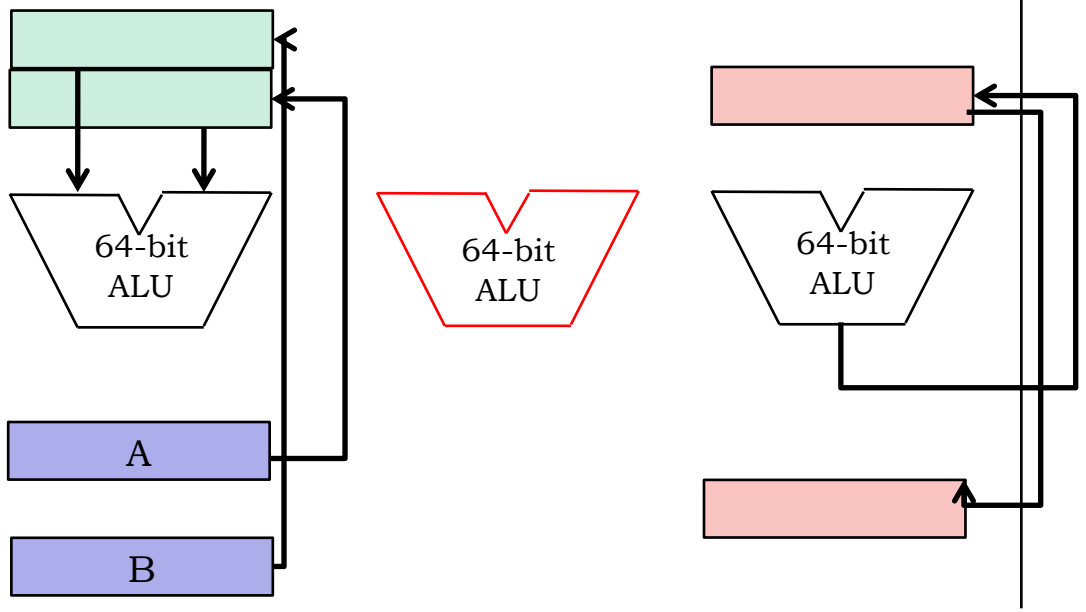
} No ops directly on memory

The VAX had registers and a stack.

Stack

All movements to and from memory are to the same implicit place. The top of the stack. Operations act on the top values in the stack and place the answer on the top of the stack. They destroy the operands.

Push A
Push B
Add
Pop C



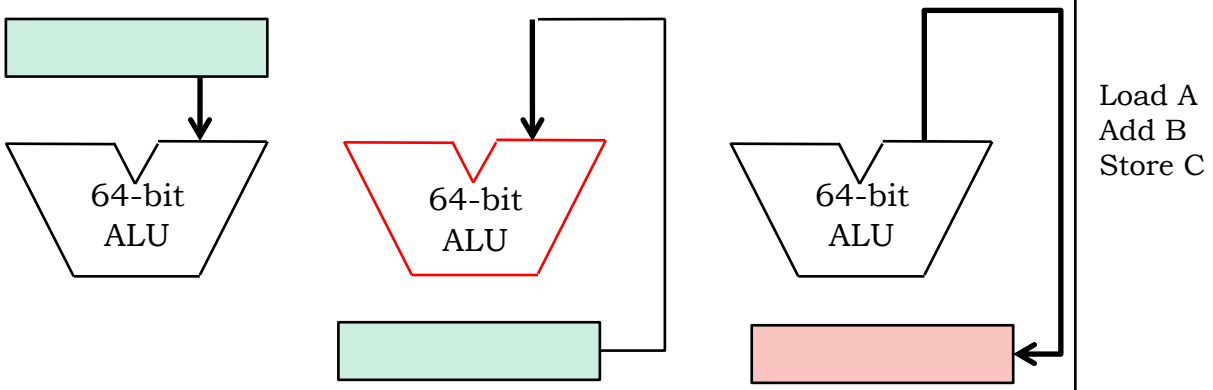
HP35
Postscript

Classification

Accumulator

There is only one place operations can be done and that is the accumulator. (Accumulates results).

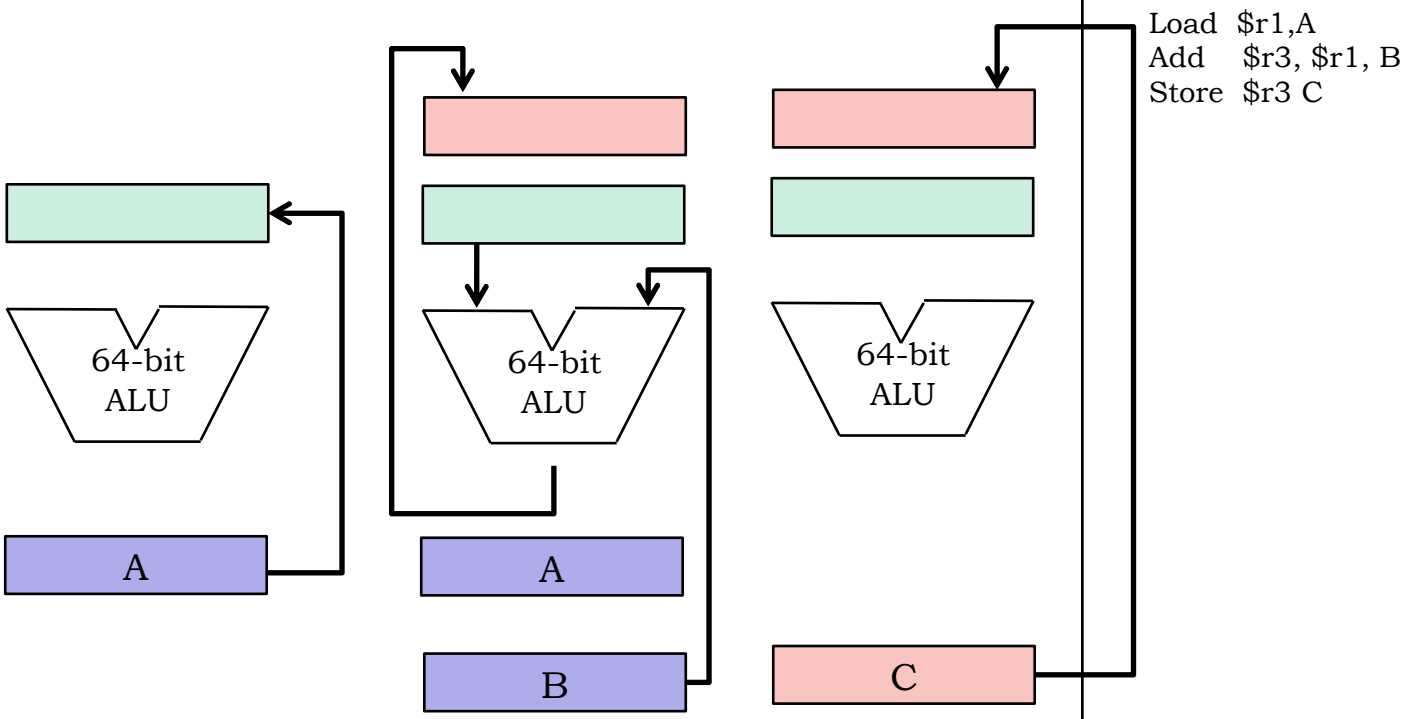
All operations implicitly refer to the accumulator



This is a single register machine of register-memory type.

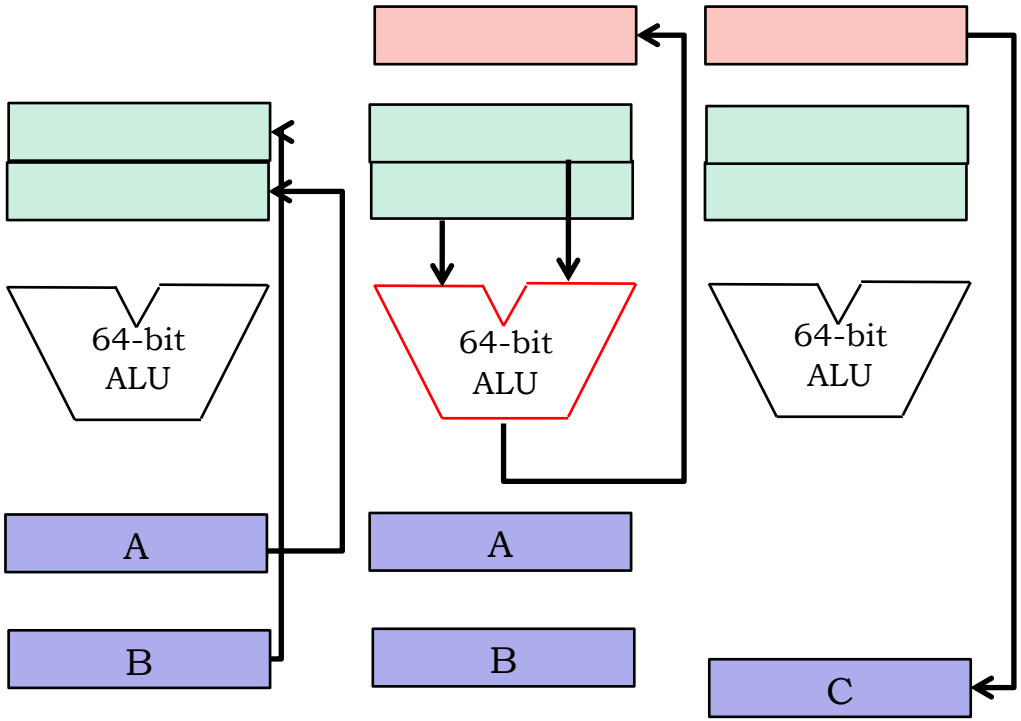
Register-Memory

We move to memory to register for manipulation, but the ALU can take data directly from memory.



Load-Store

Operations move between memory and register **OR** operate on the values in the registers.




```
Load $r1, A
Load $r2, B
Add  $r3, $r1, $r2
Store $r3,C
```

Properties

All three architectures have been used and they have different strengths.

Accumulator : Low component count for the *fast* electronics. No need to specify destination for Load or source for Store, more space for memory address or instruction space.

Register-Memory : Low instruction count. Used by machines with complex instruction sets, also tend to low instruction count. Complexity of two modes of operation. Useful if speed of memory access is a limiting factor.

Register-Register : Simplicity of implementing the system, both in hardware and software. Makes instruction pipelining more efficient.

History

The world's first stored programme computer

The Manchester Small Scale Experimental Machine
"Baby"

Speed was not an issue component count was.

It had an accumulator architecture.

32-bit word length
Binary arithmetic using 2's complement integers
A single address format order code
A random access main store of 32 words,
extendable up to 8192 words
A computing speed of around 1.2 milliseconds per instruction 0.8KHz

A millionth of a modern intel chip.

Transistor count
0

Instructions

If the accumulator is A and the contents of the memory at address S is represented by the address

LDN (Load negative)	A = - S
SUB(Subtract)	A = A - S
STO(Store)	S = A
CMP(Skip on negative A)	If A < 0, CI = CI + 1
JMP (Jump)	CI = S
JRP (Jump relative)	CI = CI + S
STP	Halt
	light the stop lamp

Memory: 32*32 bit array Williams-Kilburn tube
Two special register Williams-Kilburn tube
CI: address of current instruction
PI: current instruction
A: accumulator

CI now normally called the PC

Williams-Kilburn Tube

When Turing and the Americans started work there was only one sort of memory.
Mercury Acoustic Delay Line.

It was slower and more complex …
Williams and Kilburn invented the tube and the computer was designed to demonstrate it as a memory technology. It was simpler and as a result the Manchester machine was the first to run a programme.

The speed of the computer was limited by the memory access time …

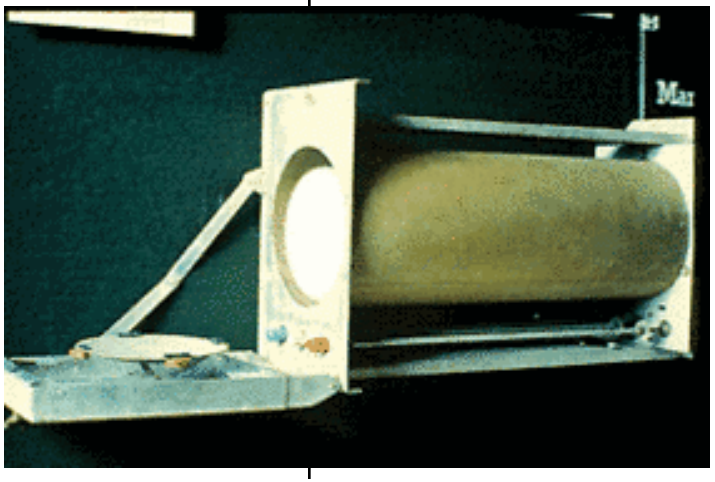
Cache memory; instruction pipelining;

Memory speed is still the bottleneck

Tube Memory

Principles

Cathode ray tube …
Read charge and refresh
Referred to in the paper
as dot-dash



Read charge and rewrite

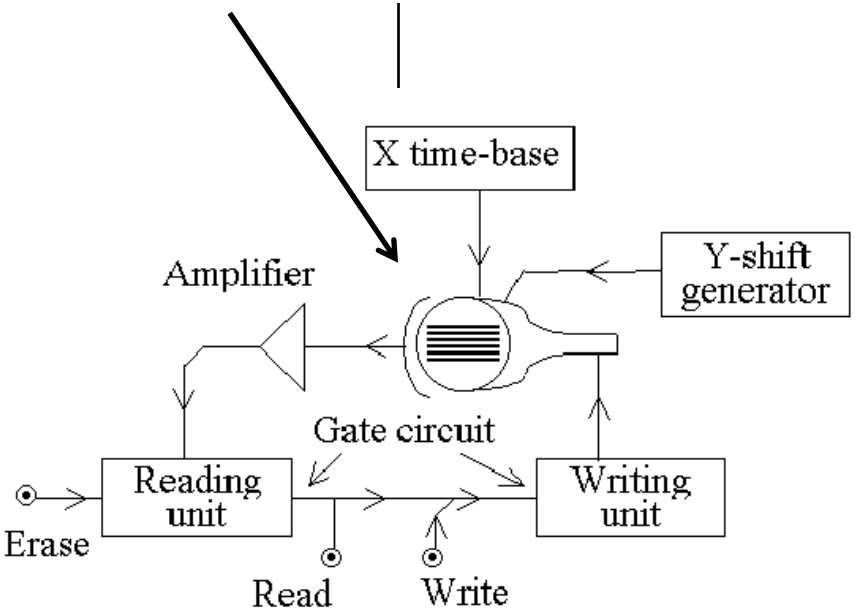


Fig. 3.-The main store, S.

Williams, Kilburn, Tootill (1951)

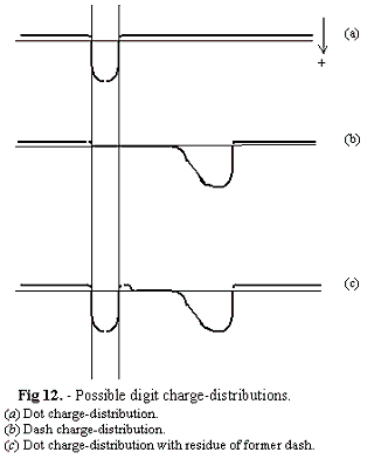


Fig 12. - Possible digit charge-distributions.
(*a*) Dot charge-distribution.
(*b*) Dash charge-distribution.
(*c*) Dot charge-distribution with residue of former dash.

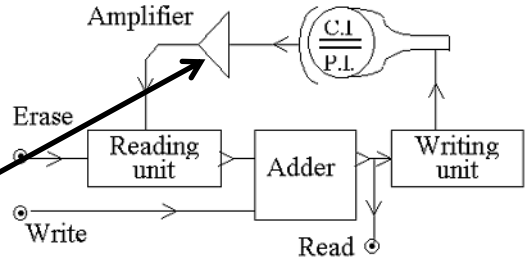


Fig. 6. The control, C.

PC and current instruction

Programmes

Addition

00 JMP 00	Start
01 LDN 07	-1012 into acc
02 SUB 08	-12 (contents of 8)
03 STO 09	store –(sum)
04 LDN 09	load -(-sum)
05 STO 09	store sum
06 STP	halt
07 1012	data word
08 12	data word

A programme to add two numbers.
The memory was a CRT tube.
Dots on a screen.
Bright = 1, dull = 0. Leave charge on the Screen. Read it and then refresh the Screen.

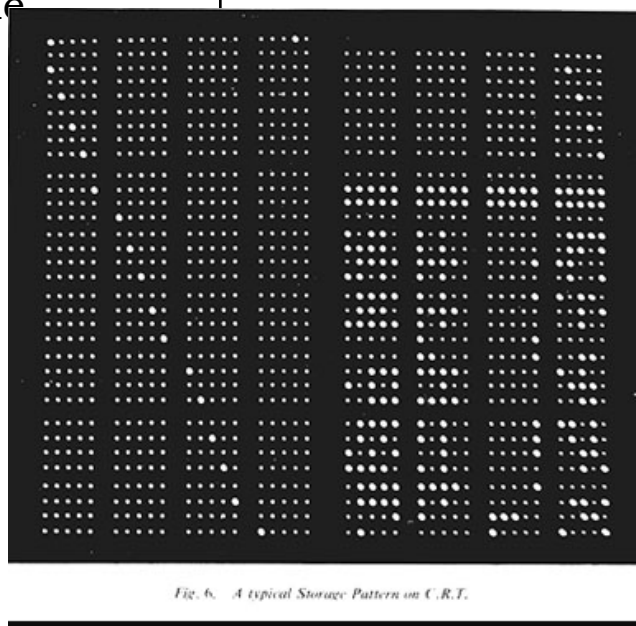
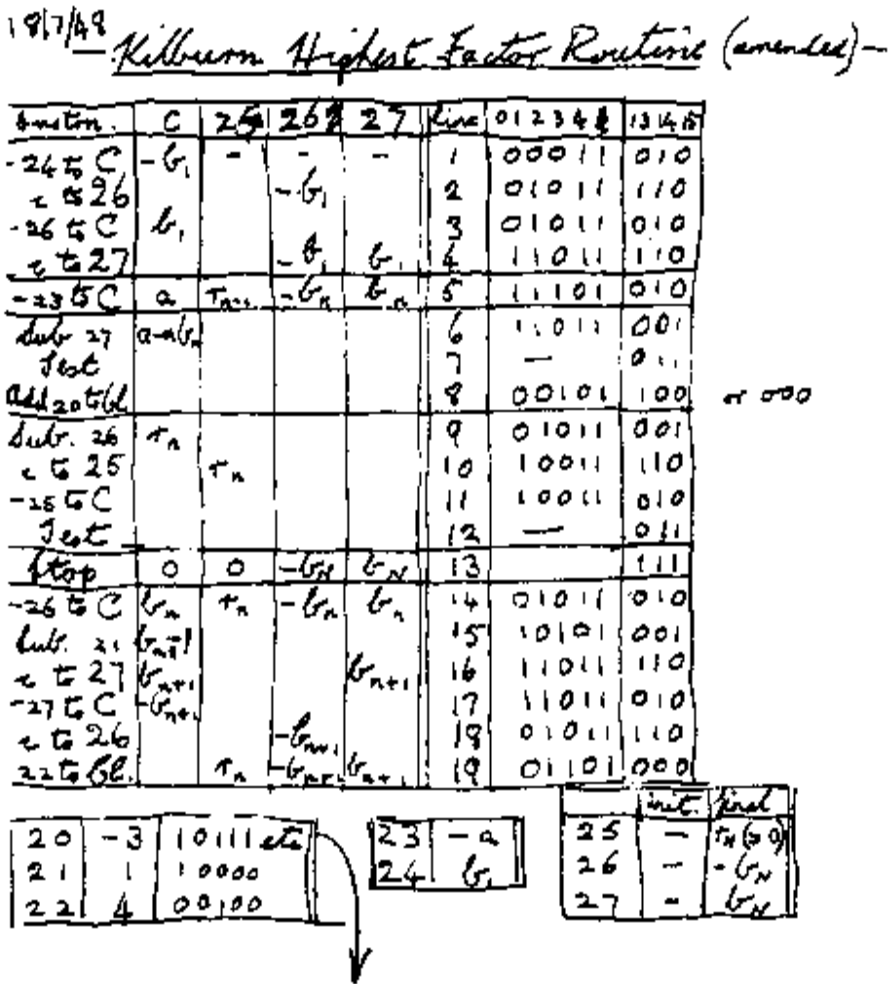


Fig. 6. A typical Storage Pattern on C.R.T.

RAM still uses a read/refresh cycle.

The world's first programme

Factor a small number 21st June 1948



19/7/48 — Kilburn Highest Factor Routine (amended) —

function	C	25	26	27	line	0 1 2 3 4 5	1 3 4 5
-24 to C	$-b_1$	-	-	-	1	0 0 0 1 1	0 1 0
c to 26			$-b_1$		2	0 1 0 1 1	1 1 0
-26 to C	b_1				3	0 1 0 1 1	0 1 0
c to 27			$-b_1$	b_1	4	1 1 0 1 1	1 1 0
-23 to C	a	T_{n-1}	$-b_n$	b_n	5	1 1 1 0 1	0 1 0
Subr 27	$a-b_n$				6	1 1 0 1 1	0 0 1
Test					7	—	0 1 1
Add 20 to C					8	0 0 1 0 1	1 0 0
Subr. 26	r_n				9	0 1 0 1 1	0 0 1
c to 25		r_n			10	1 0 0 1 1	1 1 0
-25 to C					11	1 0 0 1 1	0 1 0
Test					12	—	0 1 1
Stop	0	0	$-b_{n+1}$	b_{n+1}	13		1 1 1
-26 to C	b_n	r_n	$-b_n$	b_n	14	0 1 0 1 1	0 1 0
Subr 21	b_{n-1}				15	1 0 1 0 1	0 0 1
c to 27	b_{n+1}			b_{n+1}	16	1 1 0 1 1	1 1 0
-27 to C	$-b_{n+1}$				17	1 1 0 1 1	0 1 0
c to 26			$-b_{n+1}$		18	0 1 0 1 1	1 1 0
22 to 6E		r_n	$-b_{n+1}$	b_{n+1}	19	0 1 1 0 1	0 0 0

20	-3	10111 etc
21	1	10000
22	4	00100

23	$-a$
24	b_1

	init	final
25	—	r_n (0 9)
26	—	$-b_N$
27	—	b_N

a few days later they tried it on 2^{18}

It ran for 52 minutes – executing 2.1 million instructions and 3.9 million memory accesses

The world's first programme

Williams, Kilburn, Tootill (1951)

	Instructions
1	
2	
3	
4	

	Accumulator

	Control

	Numbers
1	
2	
3	
4	

RULES

Quiescent
Do nothing

Active

(1) When not otherwise occupied, read number in C, then
(2) Add 1 to it, and write sum in C, then
(3) Read and carry out instruction having number just written in C.

	FACILITIES	*Abbreviated form*
(1)	A can be read, and written into any line in N_r	a, N_r
(2)	N_r can be read and written into C.	N_r, C
(3)	N_r can be read, and written into A.	n_r, A
(4)	N_r can be read, and added to what is in A, the sum being written in A.	$a + n_r$, A
(5)	As 4, but subtracted.	$a - n_r$, A
(6)	The number in A can be tested for sign, zero counting as positive; 1 is added to C if number is negative, no action being taken if number is positive.	Test
(7)	The system can make itself quiescent.	Stop

N.B.-Reading does not erase the read quantity, but writing obliterates anything already present.

Fig. 1.-Schematic illustrating the behaviour of the computer.

	Instructions	A
1	n_1, A	x
2	s - n_2, A	x - y_0, x - $2y_0$, ... x-(r+1)y_0
3	Test	" " "
4	n_5, C	" " "
5	s + n_2, A	x - ry_0 = remainder
6	s, N_3	" "
7	n_4, A	0
8	s - n_3, A	-(x - ry_0)
9	Test	"
10	Stop	0
11	n_2, A	y_0
12	s - n_5, A	y_0 - 1
13	s, N_2	"
14	n_4, C	"

	Numbers
1	x
2	y_0
3	-
4	0
5	1

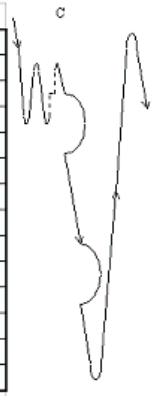


Fig 2. - Factorizing programme.

Looks like RISC!

Reference

A storage system for use with binary-digital computing machines

Proceedings of the IEE - Part III: Radio and Communication Engineering (Volume:96 , Issue: 40

A STORAGE SYSTEM FOR USE WITH BINARY-DIGITAL COMPUTING MACHINES

By Prof. F. C. WILLIAMS, O.B.E., D.Sc., D.Phil., Associate Member, and T. KILBURN, M.A.

(*The paper was first received 4th March, and in revised form 13th July, 1948. It was read before a Joint Meeting of the* MEASUREMENTS *and* RADIO SECTIONS *2nd November, 1948.*)

SUMMARY

The requirement for digital computing machines of large storage capacity has led to the development of a storage system in which the digits are represented by a charge pattern on the screen of a cathode-ray tube. Initial tests have been confined to commercial tubes. Short-term memory of the order of 0·2 sec is provided by the insulating properties of the screen material. Long-term memory is obtained by regenerating the charge pattern at a frequency greater than 5 c/s. The regeneration makes accurate stabilization of the position of the charge pattern on the c.r. tube unnecessary.

The properties required of a storage system, and its operation as part of a machine, are stated. If such a machine were operated in the series mode, an instruction would be set up and obeyed in 600 μsec.

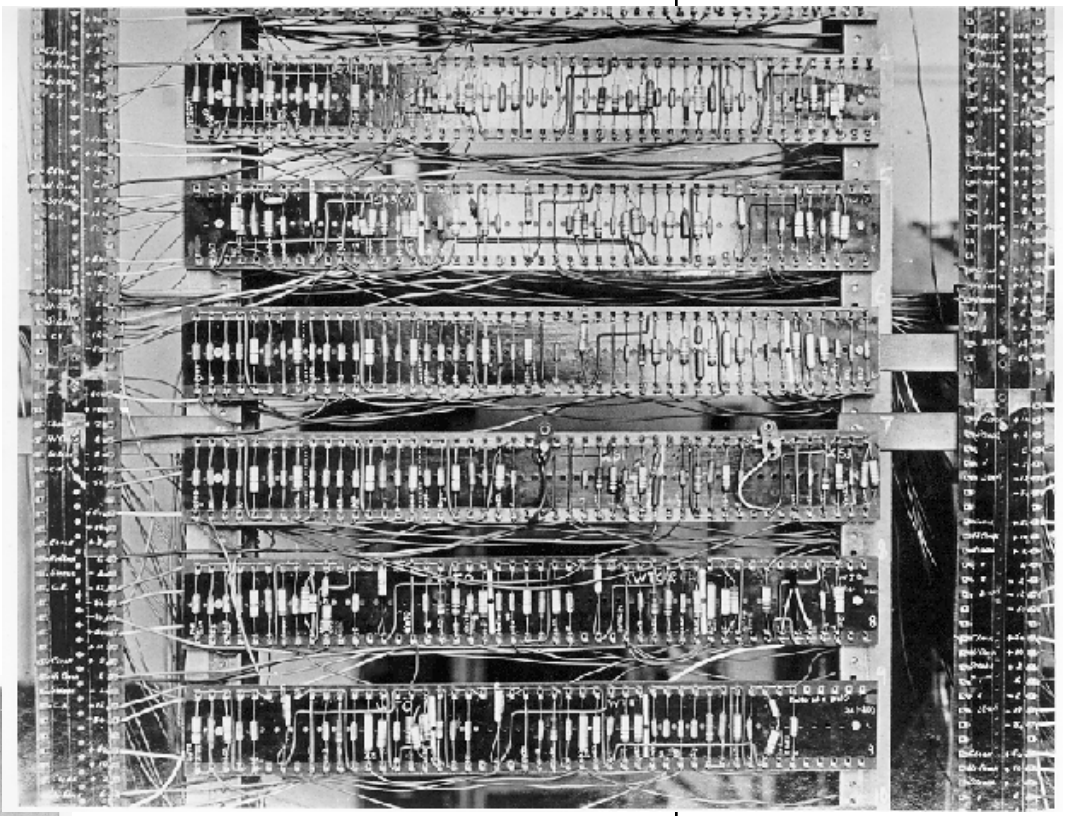
(1.2) Electronic Representation of the Binary System.

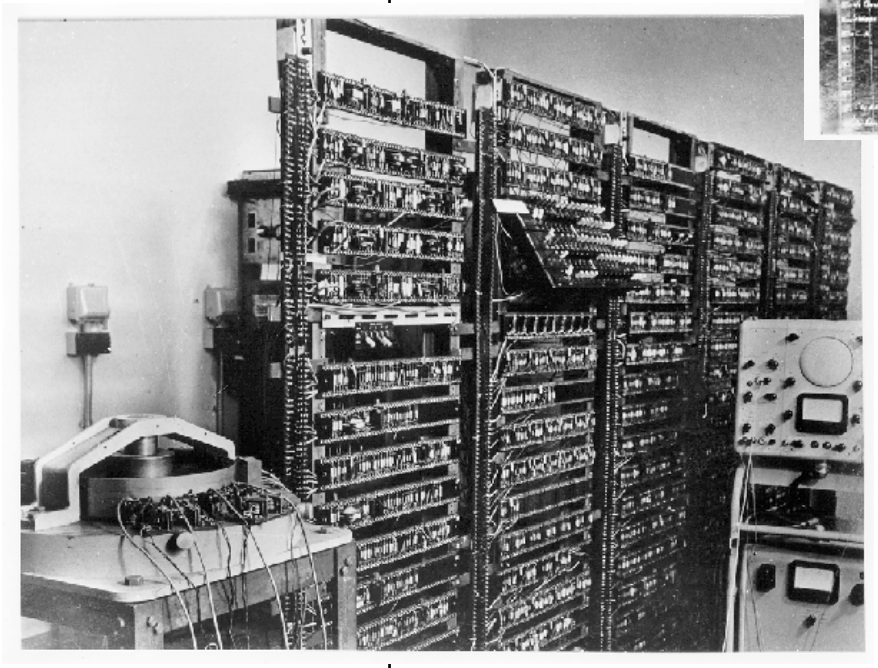
In the binary system only two values are possible for *a*, so that any two-state electronic device may be used to represent a binary digit. Examples of such devices are a variety of flip-flop circuits, the difference in level of two d.c. or a.c. voltages, the presence or absence of a video or r.f. pulse, and the device described in the paper, namely the presence or absence of a stored charge on the inner surface of a c.r.t. screen. As a digit is moved to different parts of the machine during computation, its representation will change, so that at one time or another during a computation it will have adopted several of these possible forms.

ISA Bgrnd

The world's transistorised computer

Manchester 1953





Transistor invented 1947

Registers

Properties

Early machine tended to use stack and accumulator.

Stacks are inflexible, they can only do one operation at a time in the order in which the operands have been loaded.

Registers can re-order operations and perform operations in parallel.

Best are **G**eneral **P**urpose **R**egisters (GPRs)

No optimum number although more is generally better.

Used to pass parameters, evaluate expressions and hold expressions which are repeatedly referred to. Loop variables.

2 operand architecture - add $r1, $r2
$r1=$r1+$r2

3 operand architecture - add $r0, $r1, $r2
$r0=$r1+$r2

How many ALU operands may be from memory.
0-3

The IBM 360, put Fortran implicit variables in registers and if you put them too deep two variables would end up in the same register !!!!

Some machines has reserved registers

The VAX had both they were memory-memory

Comparisons

Properties

Early machine tended to use stack and accumulator.
Modern machines use GPR.
Not simply improvement – different concerns.

Architecture	**Benefits**	**Drawbacks**
Register-register	Fixed length instructions. Almost constant number of clock ticks/instruction. Easy(!) to pipeline	High instruction count, means large memory requirements.
Register-Memory	Good instruction density and easy to encode	Variation in clock ticks/instruction. More requirement for registers than memory-memory
Memory-Memory	Compact. No need for temporary storage in registers. Low instruction/low register count	Variation in instruction size and clock ticks/instruction. Memory bottleneck; hard to pipeline

Machines in the 60's, 70's and early 80's were memory limited.
VAX 11/750 typically shipped with 1 Mbyte RAM and would support a small department … 10 users.
Pipelining was not a concern of manufacturers (except for supercomputers – and the main supercomputer manufacturer CDC/CRAY was using RISC.

See Hennessy App. B

CDC 6600

CDC 7600 – transistor based computer.

Address Modes

For multi-byte words how do we read them?

Is the lowest byte in memory the least significant byte or the most significant byte.

is 6892 stored as

6	8	9	2

or

2	9	8	6

Actually makes little difference except when exchanging data among computers.
Referred to as *Little Endian* or *Big Endian*
(Actually trivial compared with exchanging 32bit with 36bit or 60bit to anything).

Gulliver's Travels.
Jonathan Swift.
Lilliput v Blefescu
War over the proper end to break an egg.

Alignment

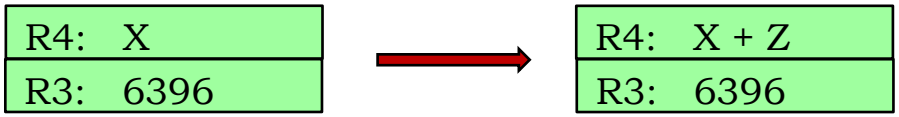
The other question is do words have to be aligned with word boundaries.
Does the first byte of an object s bytes long, have to be at an address such that address Mod(s) = 0.
It is also possible to have half word and double word alignment.
Some architectures allow both, but aligned access is faster

Address Modes for arithmetic

Immediate — Add R4, #3

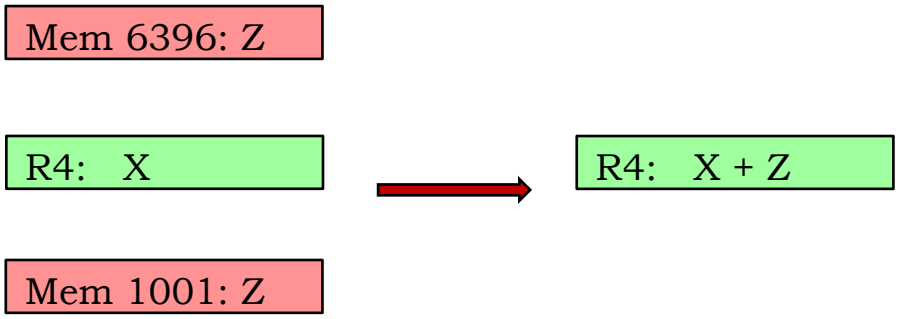
Register — Add R4, R3

Register indirect — Add R4, (R3)



Add the contents of the memory location pointed to by R3

Direct (absolute) — Add R4, (10001)

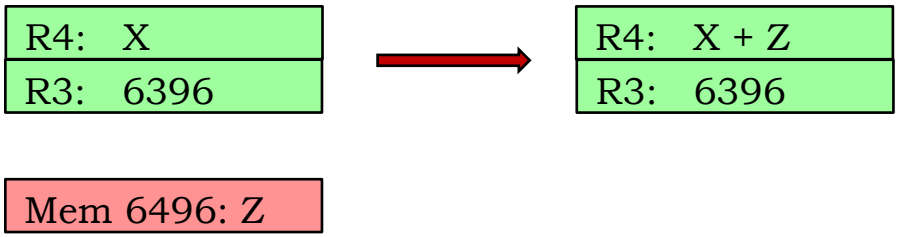


Whole word address: greater address range

Address Modes for arithmetic

Displacement

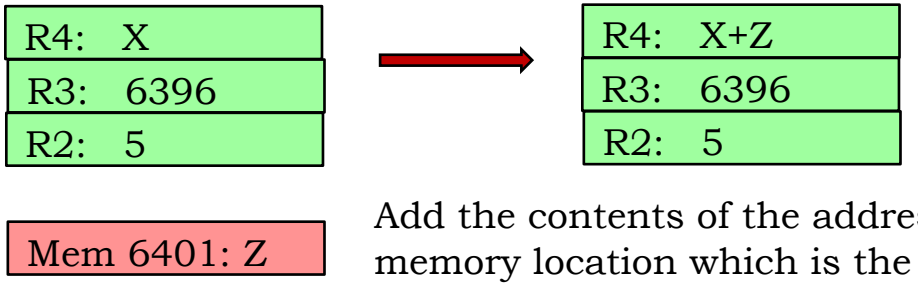
Add R4, 100(R3)



Add the contents of the memory location 100 after the address in R3

Indexed

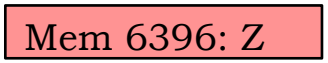
Add R4, (R3+R2)



Add the contents of the address from memory location which is the sum of the contents of R2 and R3

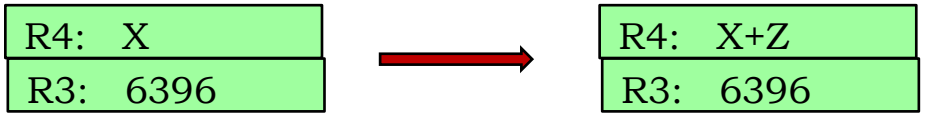
Address Modes for arithmetic

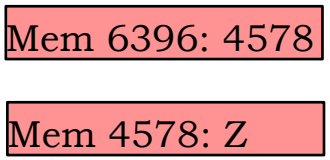
Auto increment — Add R4, (R3)+
Auto decrement — Add R4, (R3)-



As register indirect, but add a suitable off set to the register

Memory indirect — Add R4, @(R3)



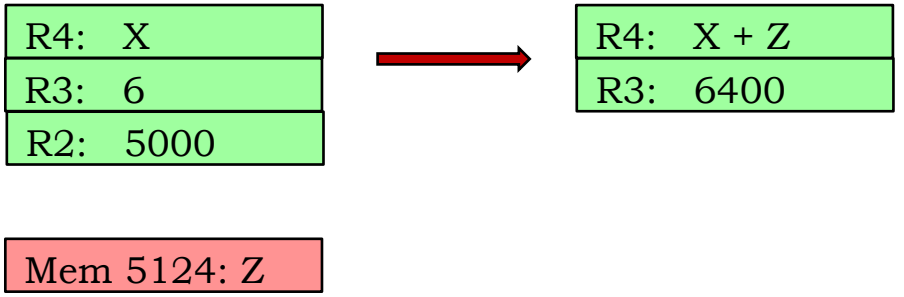


Use the address pointed to by R3 to provide an address whose contents are to be added.

The VAX had a memory indirect with an auto increment

Address Modes

Scaled



Add R4, (100)R2,[R3]

5124 = 100+5000+d*R3

R2 — The address is the base address
offset — plus and offset (100) plus an which is a word length times the value of R3.

Useful for stepping through an array.

Here again, measurements on real programmes show that the simplest modes are the ones most user.

Jumps

Transfer control
Transfer control conditionally

Jump to anywhere in memory (including virtual memory).

Addressable memory is then largest number for an *n bit* binary number

$$2^n - 1$$

32 bits gave the 4 Gbyte address space of earlier machine
65k for 16 bit machines.

But the instruction takes some of the space.
How do we get round this?

Jumps on Register content

So the jump contains the address of the register which contains the target address of the jump.

Get long words into the register by provides
A load upper and load lower to fill the register.

Used for subroutine/method returns

On call (current address+1) is placed in a register.

There is a single instruction placed at the end of the programme which causes a jump on the contents of that register.

ISA support for high level languages

.

Jump on current position

Can also be used for memory target for load or store

The other way is to provide an offset to the current position – which is m bits long

This means that the upper bits of the address are taken from the current position and the m bits can be used to provide the lower bits.

Negative jumps are a little more tricky

Because of spatial locality this is much more powerful than it might appear. **Most** instructions will be close to the existing instructions.

Data words also tend to be clustered, so even if the data is far away from the instruction.

Put a suitable address in a register and access data words relative to that address, again we can use fewer bits with very little impact on the actual performance of the programme

Locality is built into the ISA

also explains power of cache