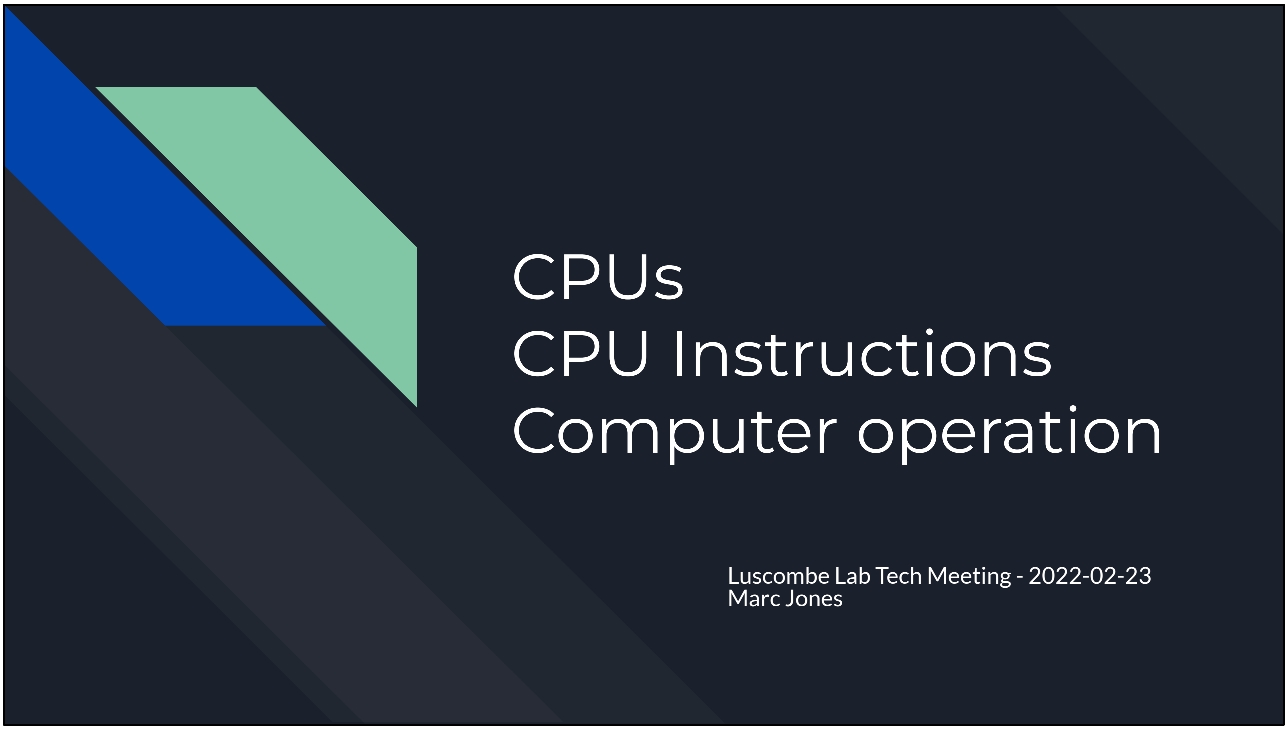
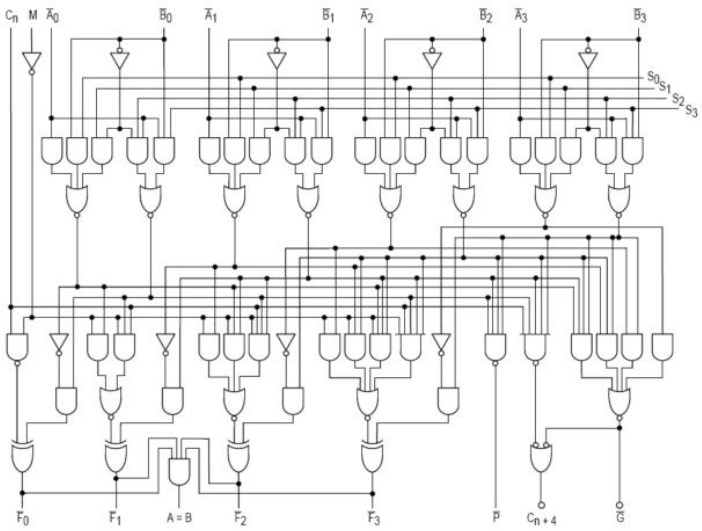


CPUs
CPU Instructions
Computer operation

Luscombe Lab Tech Meeting - 2022-02-23
Marc Jones

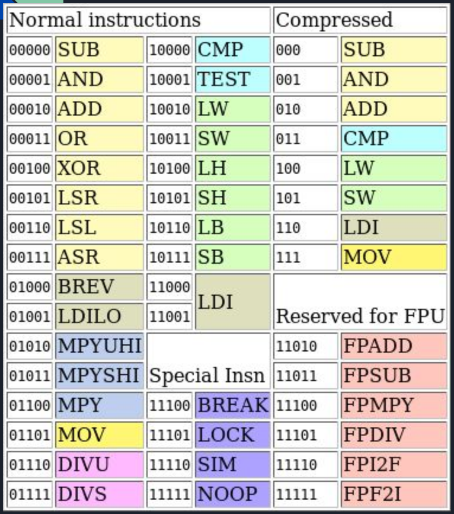
Where did we stop last time?



Arithmetic Logic Unit

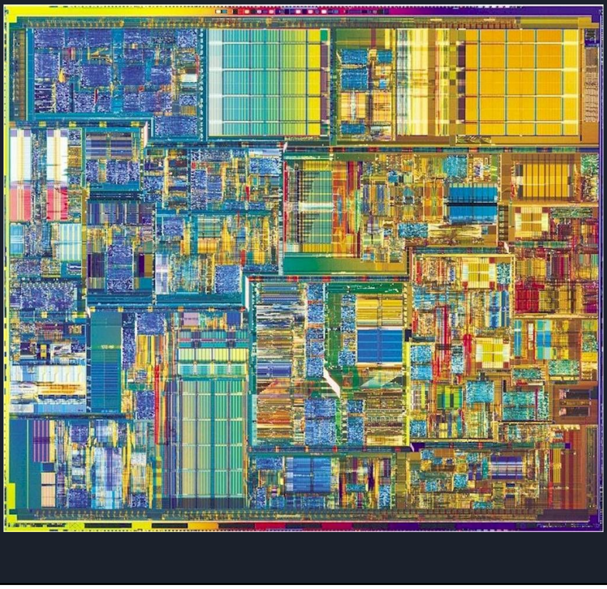
Chris explained how to string logic gates together into Arithmetic Logic Units

Where did we stop last time?

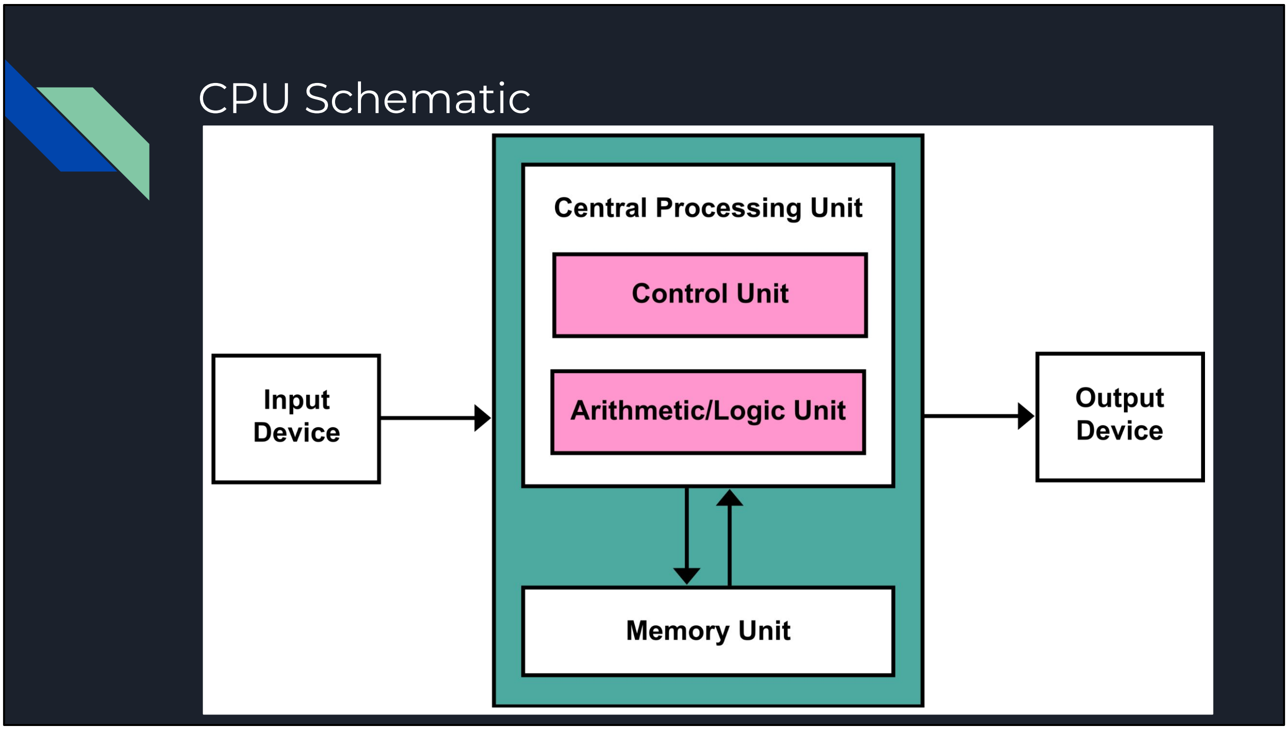


Normal instructions				Compressed	
00000	SUB	10000	CMP	000	SUB
00001	AND	10001	TEST	001	AND
00010	ADD	10010	LW	010	ADD
00011	OR	10011	SW	011	CMP
00100	XOR	10100	LH	100	LW
00101	LSR	10101	SH	101	SW
00110	LSL	10110	LB	110	LDI
00111	ASR	10111	SB	111	MOV
01000	BREV	11000	LDI	Reserved for FPU	
01001	LDILO	11001			
01010	MPYUHI	Special Insn		11010	FPADD
01011	MPYSHI			11011	FPSUB
01100	MPY	11100	BREAK	11100	FPMPY
01101	MOV	11101	LOCK	11101	FPDIV
01110	DIVU	11110	SIM	11110	FPI2F
01111	DIVS	11111	NOOP	11111	FPF2I

ZipCPU Instruction Set



And touched on CPU instruction sets and how CPUs are actually made



CPU Schematic

Central Processing Unit

Control Unit

Arithmetic/Logic Unit

Input Device

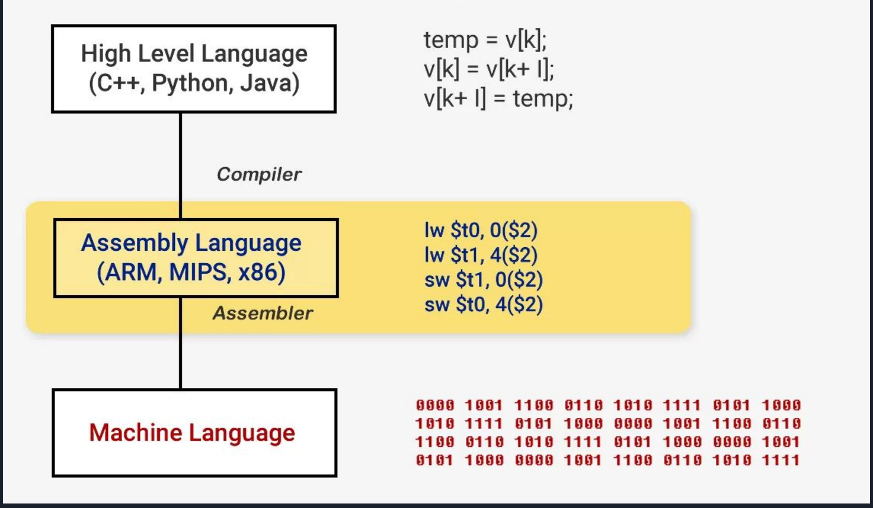
Output Device

Memory Unit

The simplest explanation is that a CPU follows a set of instructions to perform some operation on a set of inputs.

Von Neumann architecture

How is code translated to binary?



High Level Language
(C++, Python, Java)

```
temp = v[k];
v[k] = v[k+ l];
v[k+ l] = temp;
```

Compiler

Assembly Language
(ARM, MIPS, x86)

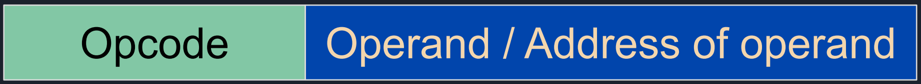
```
lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)
```

Assembler

Machine Language

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

General structure of a machine code instruction



Opcode	Operand / Address of operand

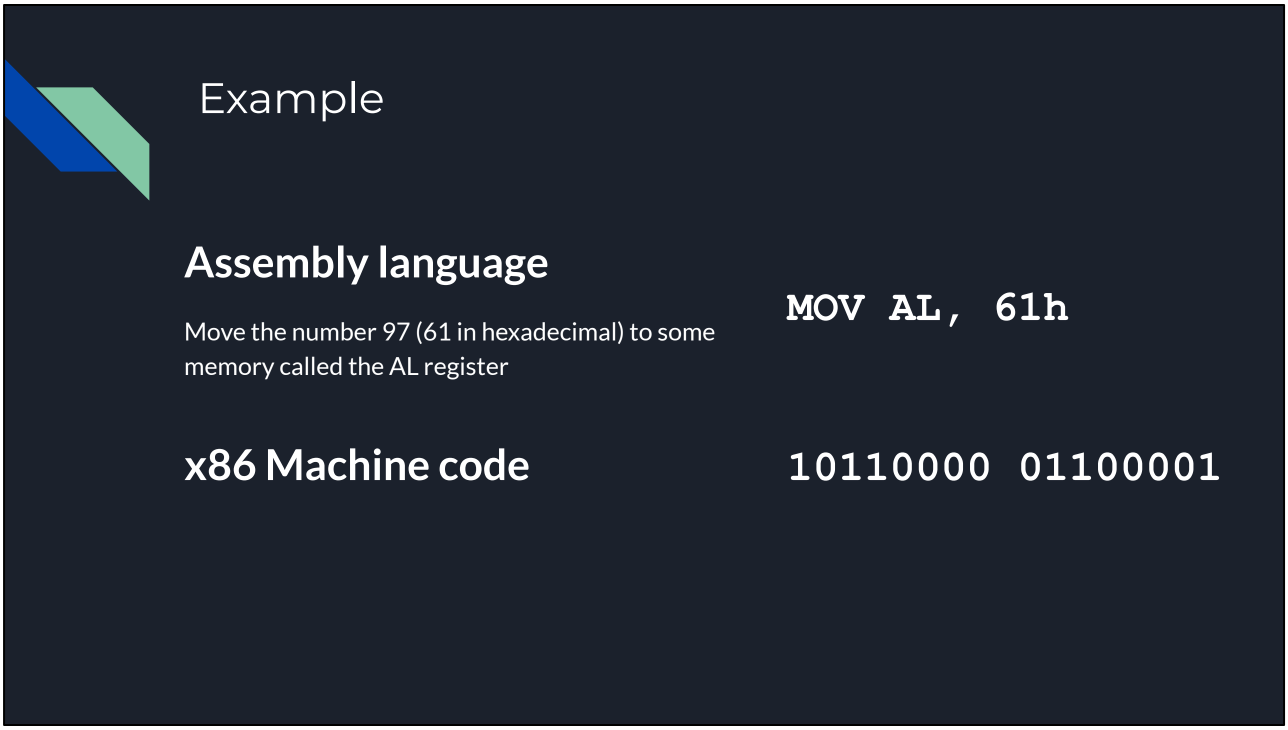
- The Operation code (Opcode) field specifies the operation to be performed
- The Address field contains the location of the operand (register or memory location)

Example

Assembly language

Move the number 97 (61 in hexadecimal) to some memory called the AL register

```
MOV AL, 61h
```



Example

Assembly language

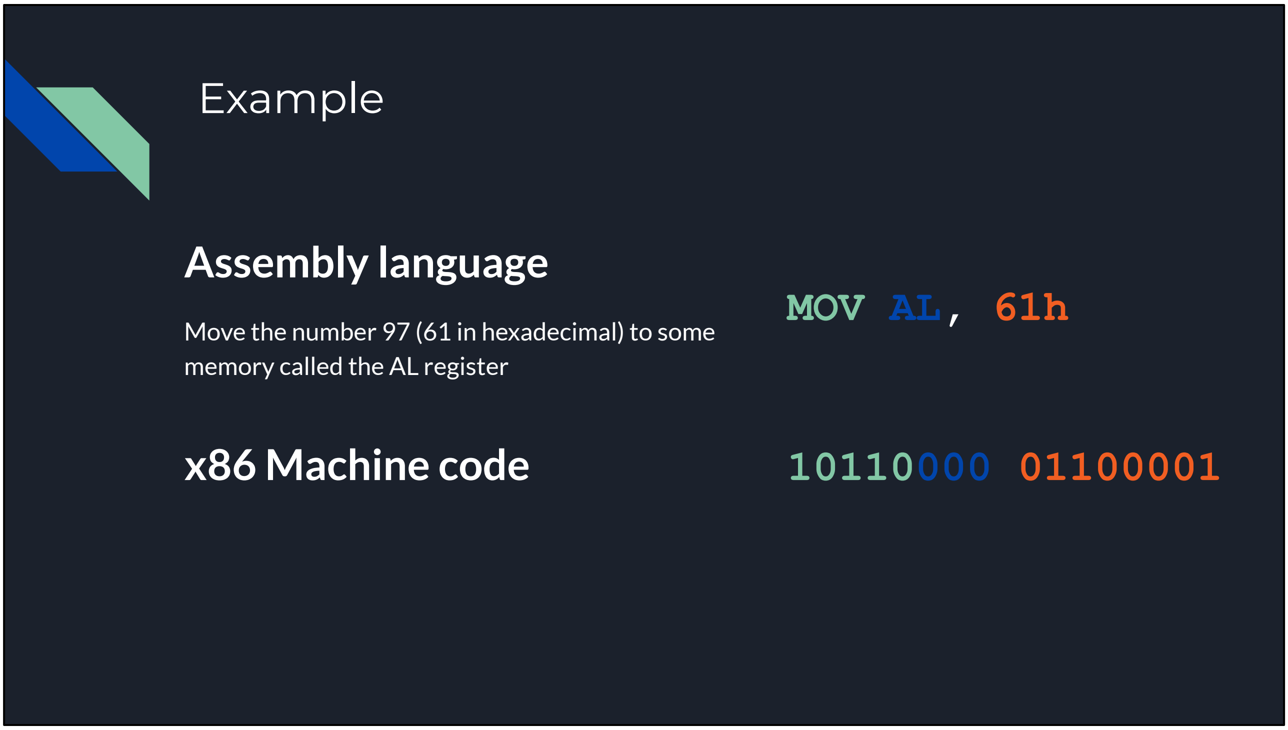
Move the number 97 (61 in hexadecimal) to some memory called the AL register

```
MOV AL, 61h
```

x86 Machine code

```
10110000 01100001
```

Mnemonic and opcode



Example

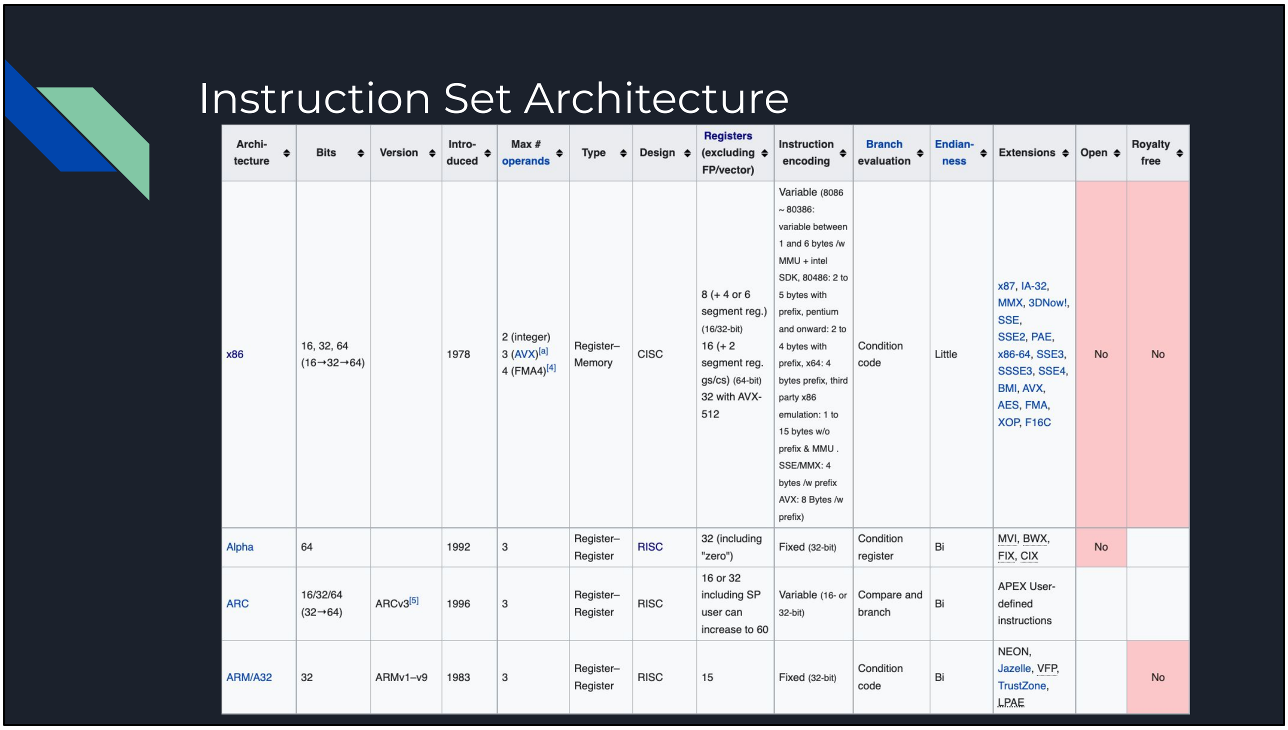
Assembly language

Move the number 97 (61 in hexadecimal) to some memory called the AL register

```
MOV AL, 61h
```

x86 Machine code

```
10110000 01100001
```



Instruction Set Architecture

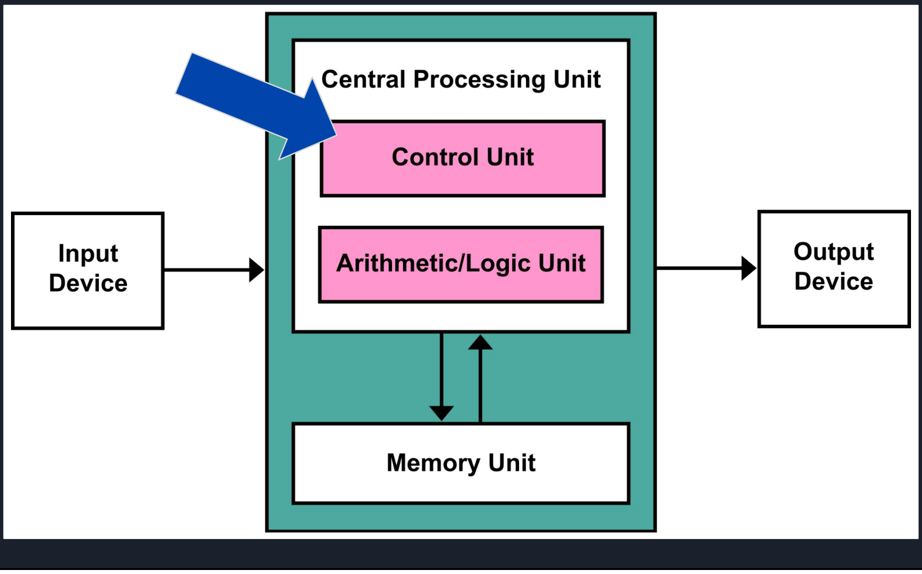
Architecture	Bits	Version	Introduced	Max # operands	Type	Design	Registers (excluding FP/vector)	Instruction encoding	Branch evaluation	Endianness	Extensions	Open	Royalty free
x86	16, 32, 64 (16→32→64)		1978	2 (integer) 3 (AVX)[a] 4 (FMA4)[4]	Register–Memory	CISC	8 (+ 4 or 6 segment reg.) (16/32-bit) 16 (+ 2 segment reg. gs/cs) (64-bit) 32 with AVX-512	Variable (8086 ~ 80386: variable between 1 and 6 bytes /w MMU + intel SDK, 80486: 2 to 5 bytes with prefix, pentium and onward: 2 to 4 bytes with prefix, x64: 4 bytes prefix, third party x86 emulation: 1 to 15 bytes w/o prefix & MMU . SSE/MMX: 4 bytes /w prefix AVX: 8 Bytes /w prefix)	Condition code	Little	x87, IA-32, MMX, 3DNow!, SSE, SSE2, PAE, x86-64, SSE3, SSSE3, SSE4, BMI, AVX, AES, FMA, XOP, F16C	No	No
Alpha	64		1992	3	Register–Register	RISC	32 (including "zero")	Fixed (32-bit)	Condition register	Bi	MVI, BWX, FIX, CIX	No	
ARC	16/32/64 (32→64)	ARCv3[5]	1996	3	Register–Register	RISC	16 or 32 including SP user can increase to 60	Variable (16- or 32-bit)	Compare and branch	Bi	APEX User-defined instructions		
ARM/A32	32	ARMv1–v9	1983	3	Register–Register	RISC	15	Fixed (32-bit)	Condition code	Bi	NEON, Jazelle, VFP, TrustZone, LPAE		No

ISAs vary in quality and completeness.

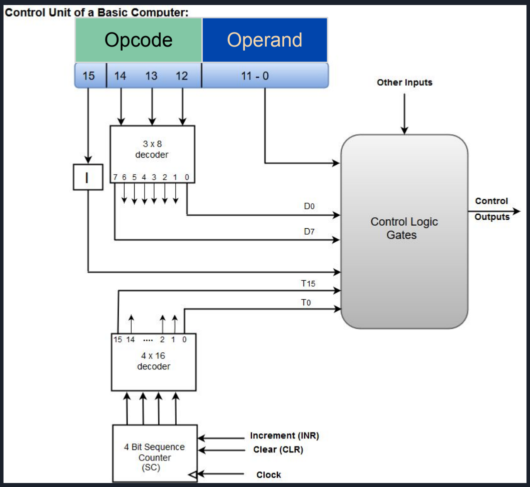
1) how easy the code is to understand
2) how much code is required to do a specific action
3) cost of the computer to interpret the instructions - more complexity means more hardware needed to decode and execute the instructions
4) speed of the computer (with more complex decoding hardware comes longer decode time)

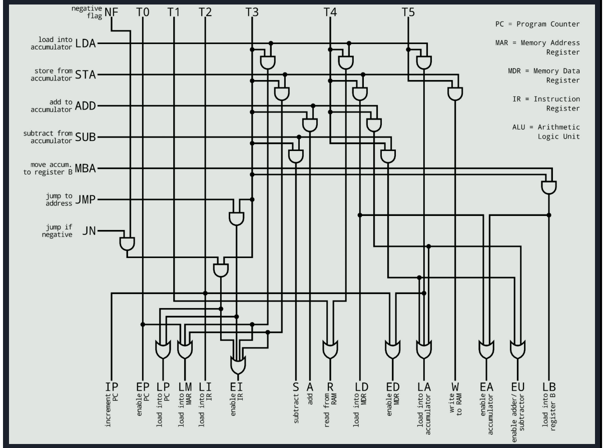
Reduced instruction set computer
Complex instruction set computer

How are the instructions interpreted?



How are the instructions interpreted?

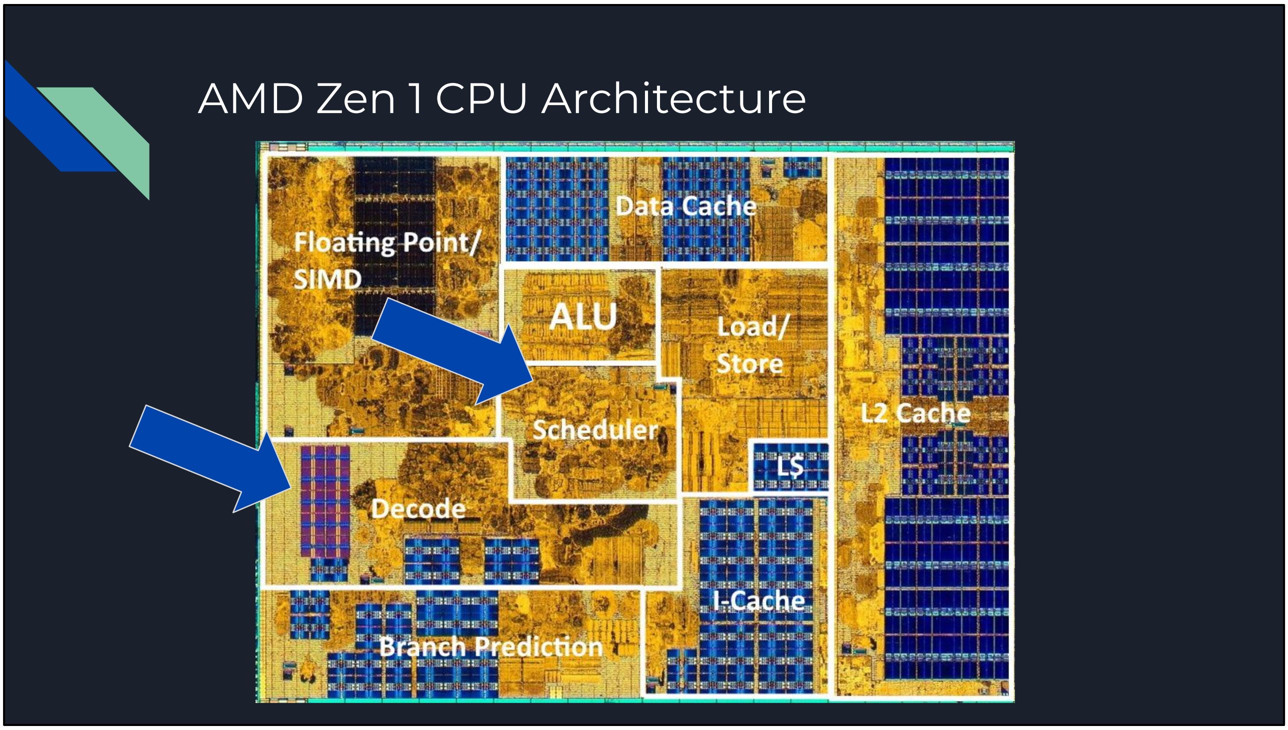




The control unit decodes what each instruction means, and can then controls how the other components operate

When the control unit receives an instruction, which is just a binary number, it will then signal what the ALU and memory is supposed to do

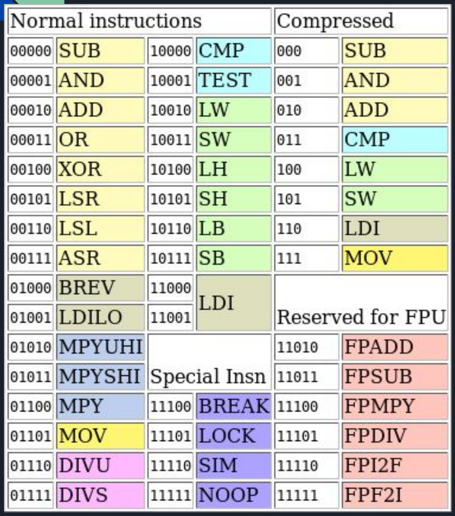
The control unit also contains a clock. This is a tiny oscillating crystal, which controls the rate at which calculations are performed by the CPU



AMD Zen 1 CPU Architecture

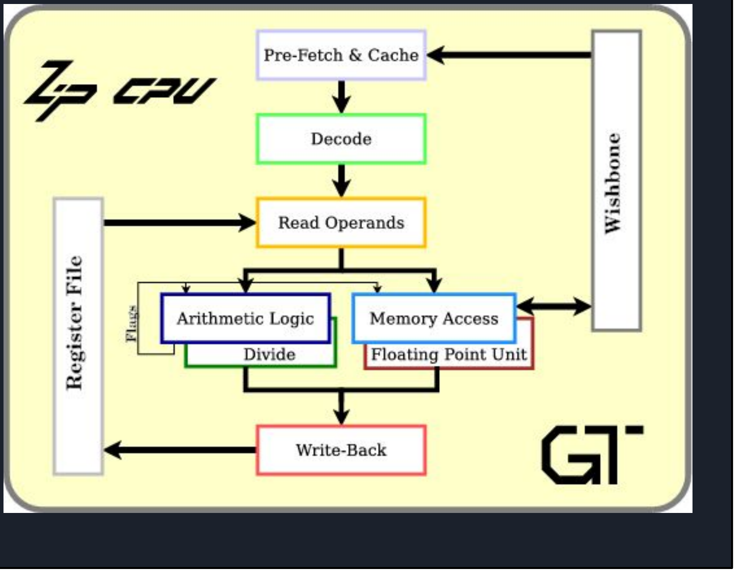
Control unit on an actual CPU

Fetch - Decode -Execute

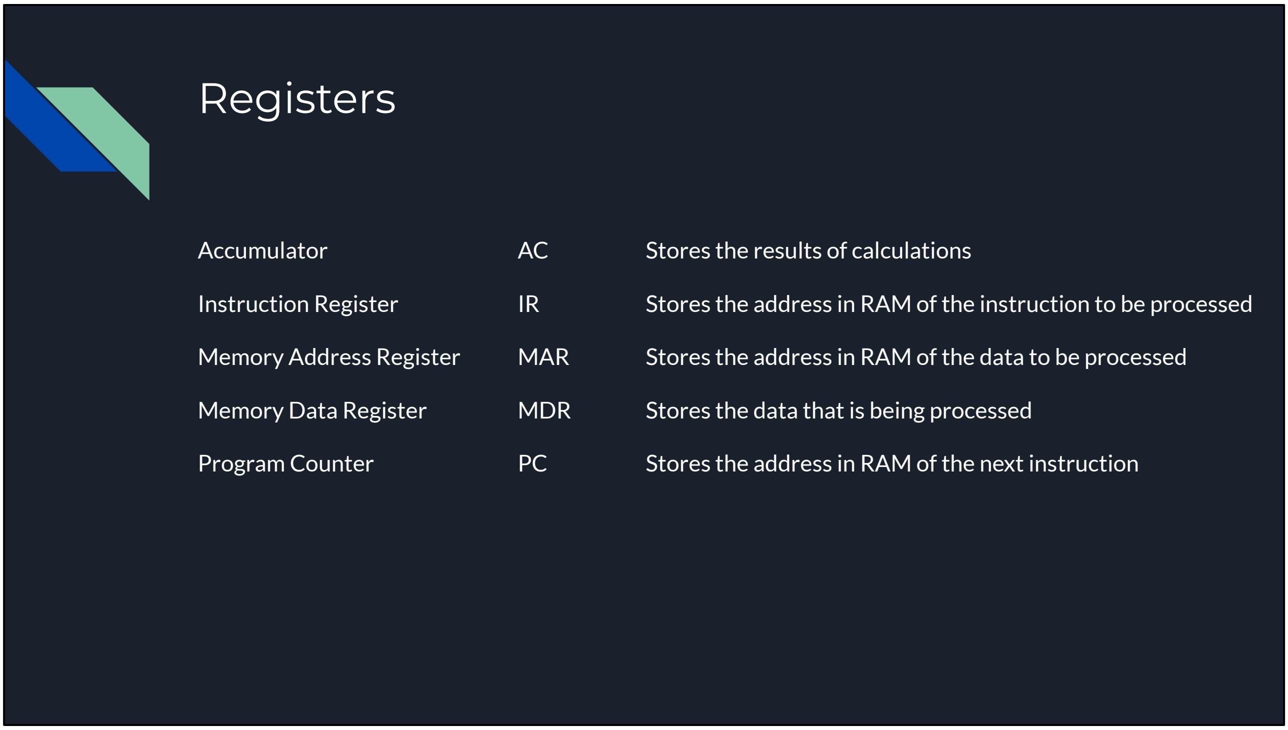


Normal instructions				Compressed	
00000	SUB	10000	CMP	000	SUB
00001	AND	10001	TEST	001	AND
00010	ADD	10010	LW	010	ADD
00011	OR	10011	SW	011	CMP
00100	XOR	10100	LH	100	LW
00101	LSR	10101	SH	101	SW
00110	LSL	10110	LB	110	LDI
00111	ASR	10111	SB	111	MOV
01000	BREV	11000	LDI	Reserved for FPU	
01001	LDILO	11001			
01010	MPYUHI	Special Insn		11010	FPADD
01011	MPYSHI			11011	FPSUB
01100	MPY	11100	BREAK	11100	FPMPY
01101	MOV	11101	LOCK	11101	FPDIV
01110	DIVU	11110	SIM	11110	FPI2F
01111	DIVS	11111	NOOP	11111	FPF2I

ZipCPU Instruction Set



That’s how a CPU interprets one instruction, but how does a CPU follow a set of instructions



Registers

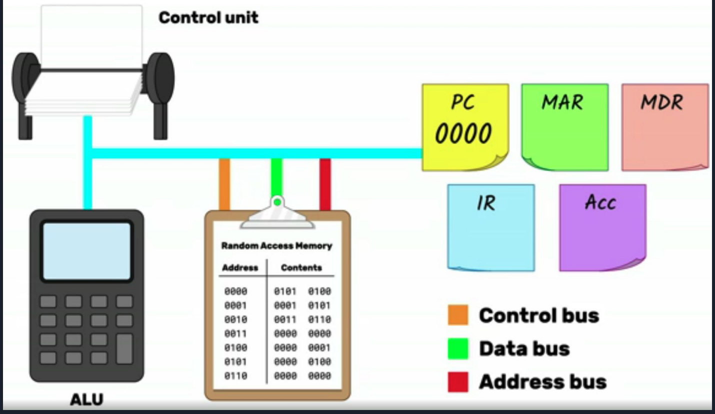
Accumulator	AC	Stores the results of calculations
Instruction Register	IR	Stores the address in RAM of the instruction to be processed
Memory Address Register	MAR	Stores the address in RAM of the data to be processed
Memory Data Register	MDR	Stores the data that is being processed
Program Counter	PC	Stores the address in RAM of the next instruction

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers.

A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

Fetch - Decode - Execute Video



Random Access Memory

Address	Contents	
0000	0101	0100
0001	0001	0101
0010	0011	0110
0011	0000	0000
0100	0000	0001
0101	0000	0100
0110	0000	0000

🟧 Control bus
🟩 Data bus
🟥 Address bus

https://www.futurelearn.com/info/courses/how-computers-work/0/steps/49284

Video

Little Man Computer

Opcode	Address
5	01

1xx - Add

2xx - Subtract

3xx - Store

5xx - Load

6xx - Branch / Go To xx

7xx - Branch if 0

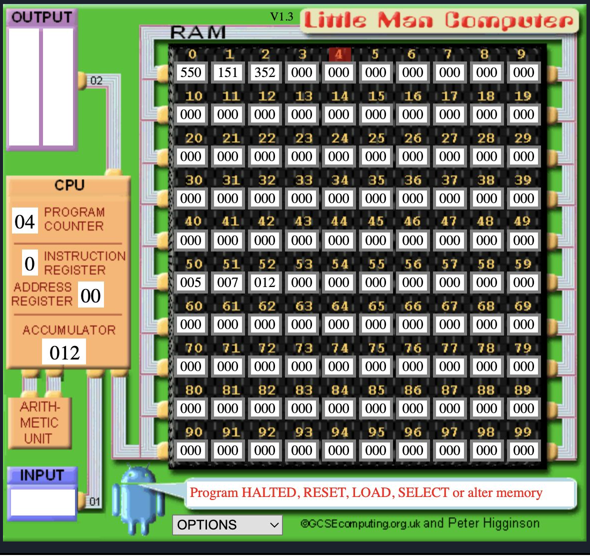
8xx - Branch if +

901 - Input

902 - Output

000 - Halt

https://www.peterhigginson.co.uk/LMC



PROGRAM COUNTER 04

INSTRUCTION REGISTER 0

ADDRESS REGISTER 00

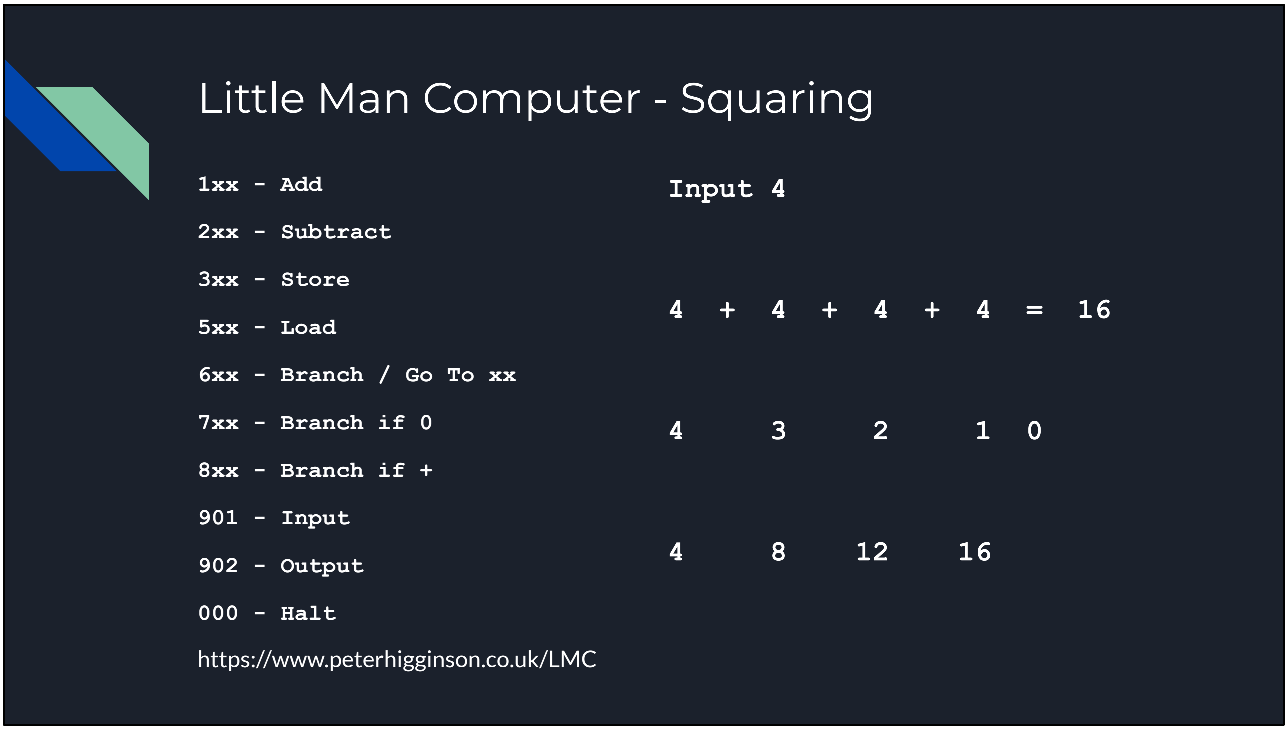
ACCUMULATOR 012

0	1	2	3	4	5	6	7	8	9
550	151	352	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
005	007	012	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

Program HALTED, RESET, LOAD, SELECT or alter memory

©GCSEcomputing.org.uk and Peter Higginson

Video



Little Man Computer - Squaring

1xx - Add

2xx - Subtract

3xx - Store

5xx - Load

6xx - Branch / Go To xx

7xx - Branch if 0

8xx - Branch if +

901 - Input

902 - Output

000 - Halt

https://www.peterhigginson.co.uk/LMC

Input 4

4	+	4	+	4	+	4	=	16
4		3		2		1	0	
4		8		12		16		

Little Man Computer - Squaring

1xx - Add

2xx - Subtract

3xx - Store

5xx - Load

6xx - Branch / Go To xx

7xx - Branch if 0

8xx - Branch if +

901 - Input

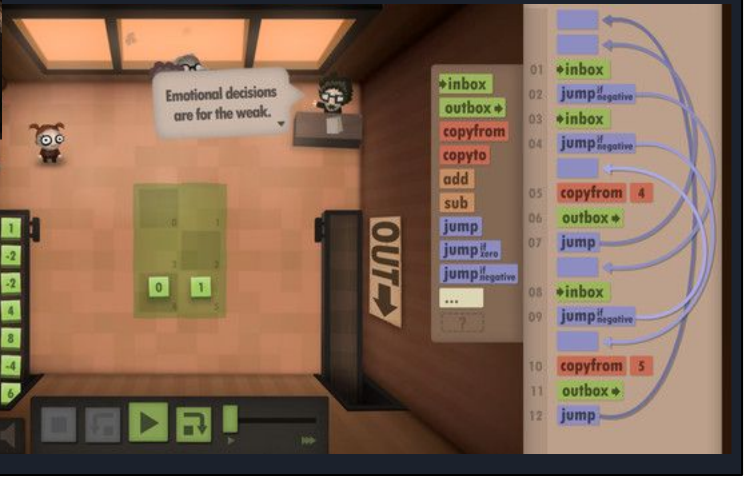
902 - Output

000 - Halt

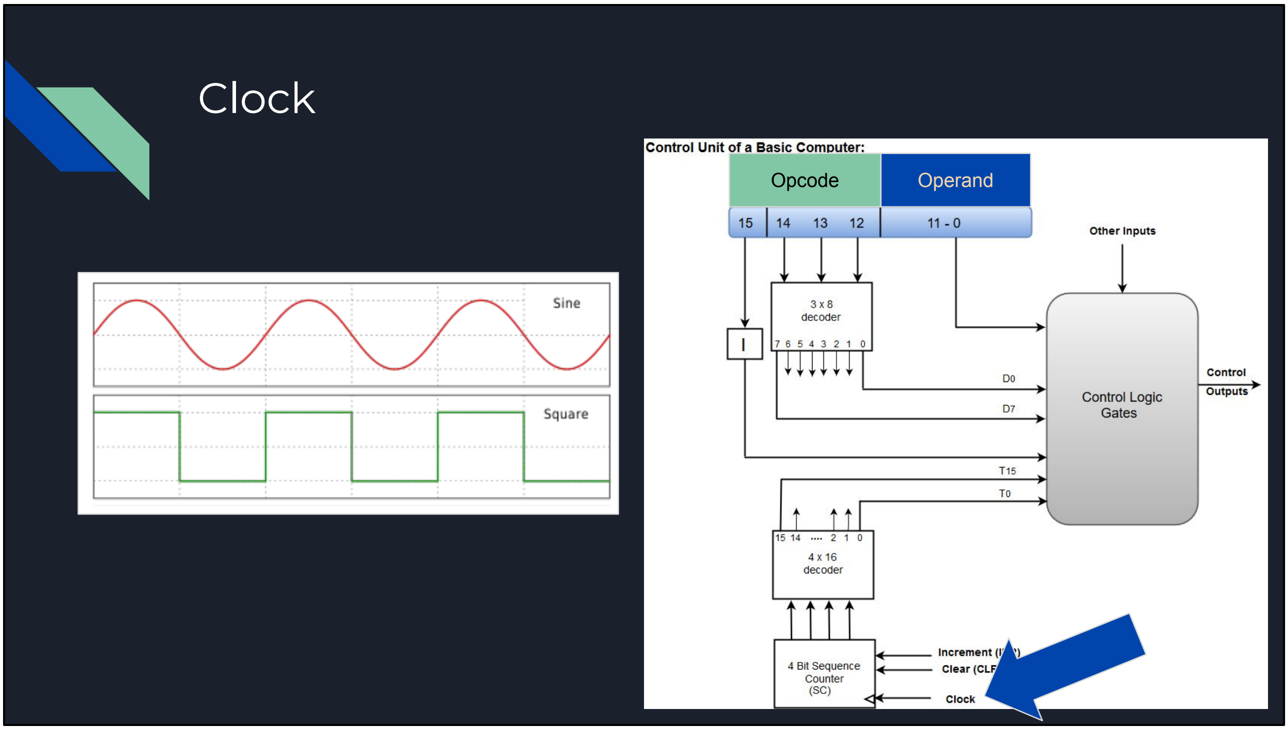
https://www.peterhigginson.co.uk/LMC

```
Input 4                       (Mem 51)

4  +  4  +  4  +  4  =  16

4     3     2     1  0  (Mem 52)

4     8    12    16      (Mem 53)
```

~~Game~~ Learning Resource Recommendation!



```
inbox
jump if negative
inbox
jump if negative
copyfrom 4
outbox
jump
inbox
jump if negative
copyfrom 5
outbox
jump
```



Clock

Need to keep the various parts of a CPU synchronised

This is done by a clock, which alternates between high and low at a particular frequency

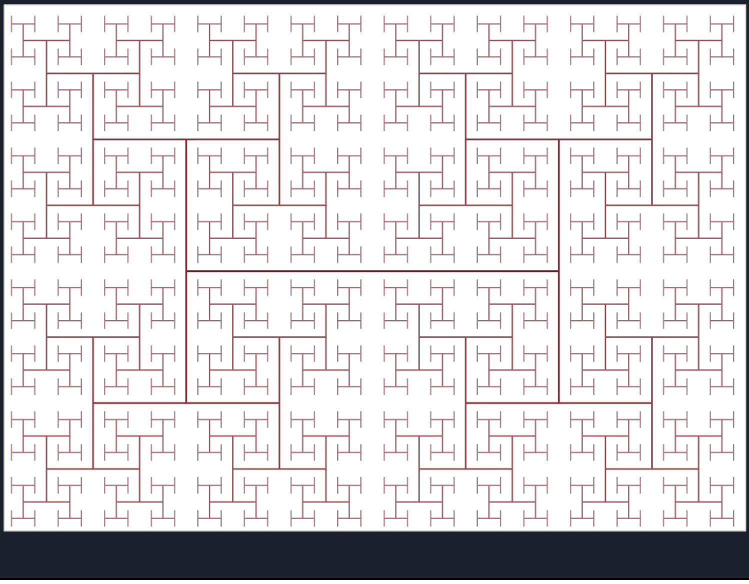
Done by an oscillator crystal

Increasing the frequency increases the power usage

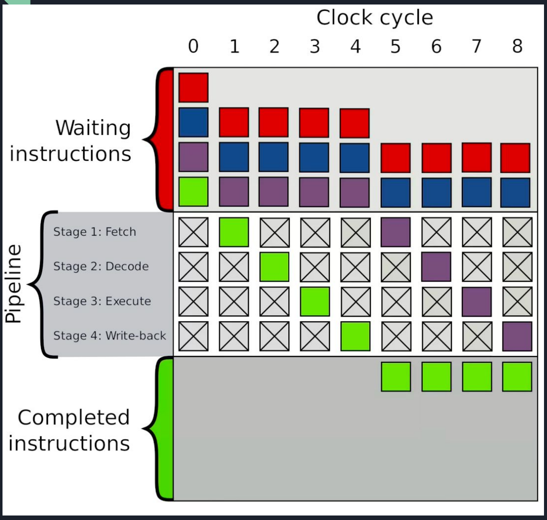
the power used to drive the clock signal can be over 30% of the total power used by the entire chip

To put this amount of power into perspective, a CPU generates more heat per unit area than a nuclear reactor

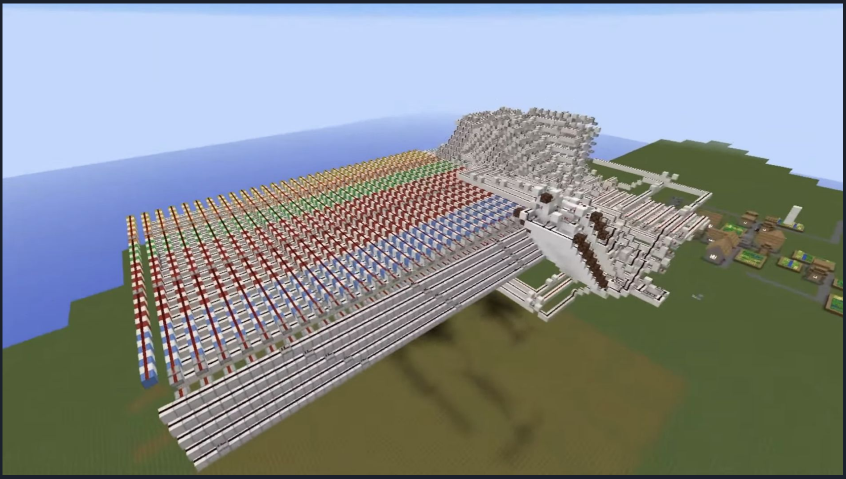
Distributing the clock signal



Clock


Clock cycle
0
1
2
3
4
5
6
7
8
Waiting instructions
Pipeline
Stage 1: Fetch
Stage 2: Decode
Stage 3: Execute
Stage 4: Write-back
Completed instructions

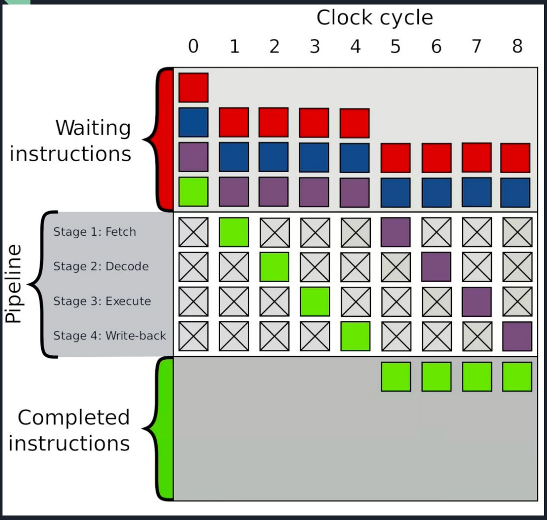
CPUs Built in Minecraft



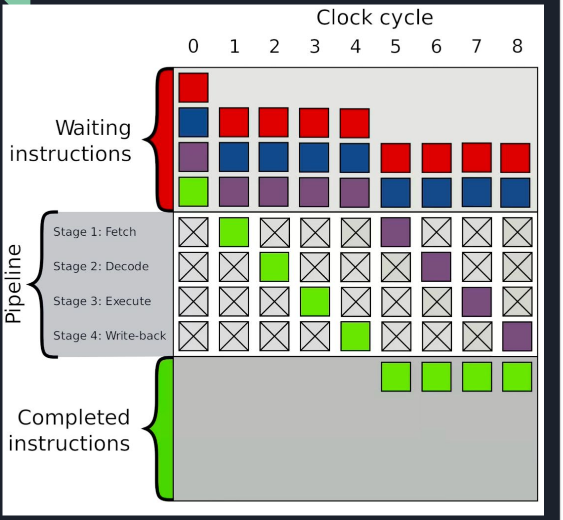
https://www.youtube.com/watch?v=ziv8SXfMbkk

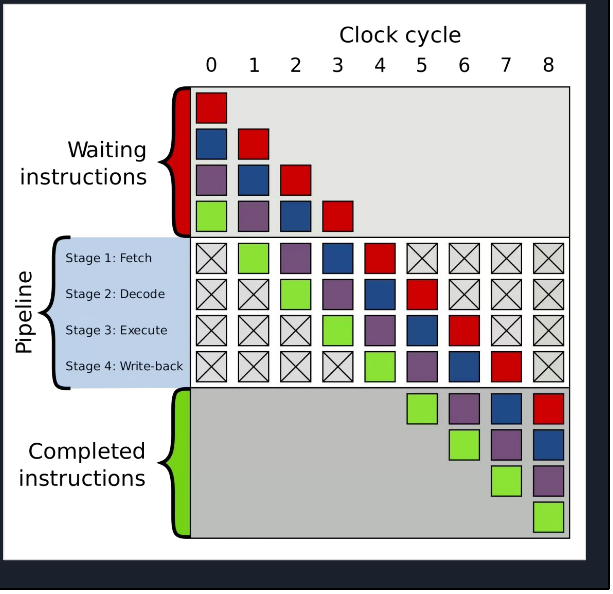
VIDEO

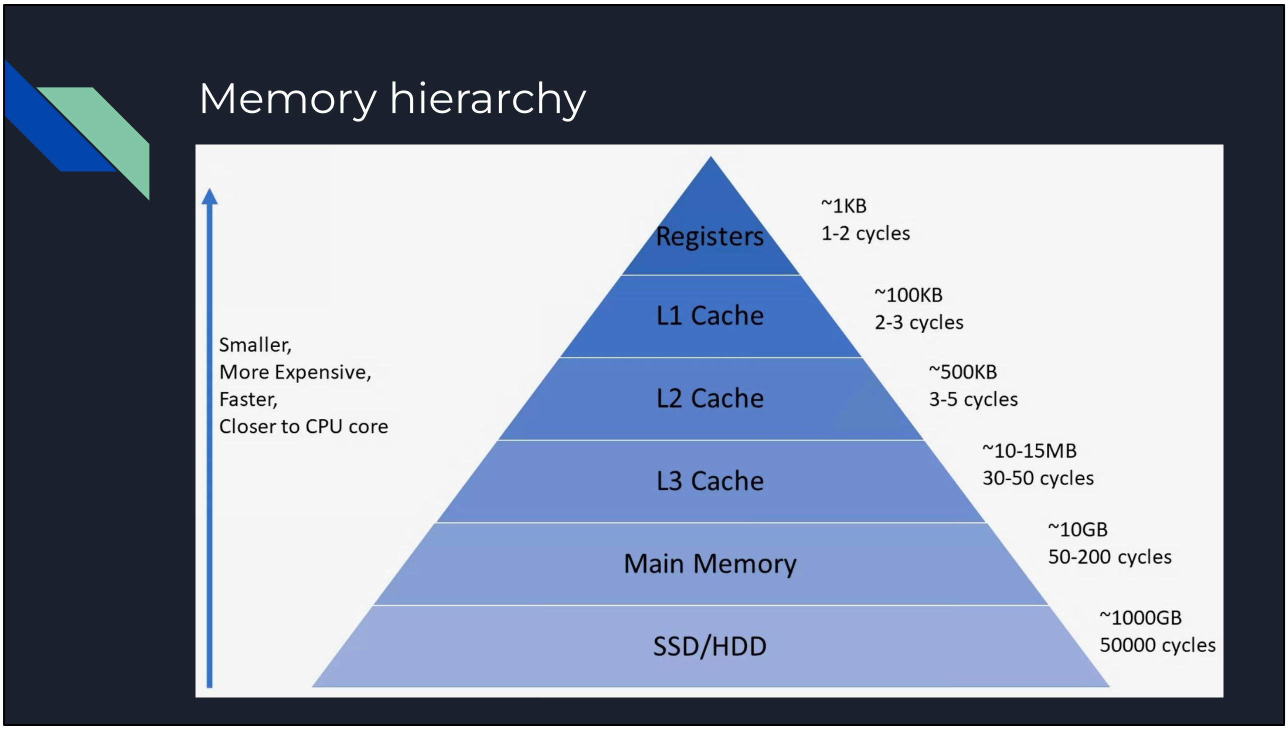
Clock


Clock cycle
0
1
2
3
4
5
6
7
8
Waiting instructions
Pipeline
Stage 1: Fetch
Stage 2: Decode
Stage 3: Execute
Stage 4: Write-back
Completed instructions

Pipelining


Clock cycle
0 1 2 3 4 5 6 7 8
Waiting instructions
Pipeline
Stage 1: Fetch
Stage 2: Decode
Stage 3: Execute
Stage 4: Write-back
Completed instructions


Clock cycle
0 1 2 3 4 5 6 7 8
Waiting instructions
Pipeline
Stage 1: Fetch
Stage 2: Decode
Stage 3: Execute
Stage 4: Write-back
Completed instructions



Memory hierarchy

Smaller,
More Expensive,
Faster,
Closer to CPU core

Registers
~1KB
1-2 cycles

L1 Cache
~100KB
2-3 cycles

L2 Cache
~500KB
3-5 cycles

L3 Cache
~10-15MB
30-50 cycles

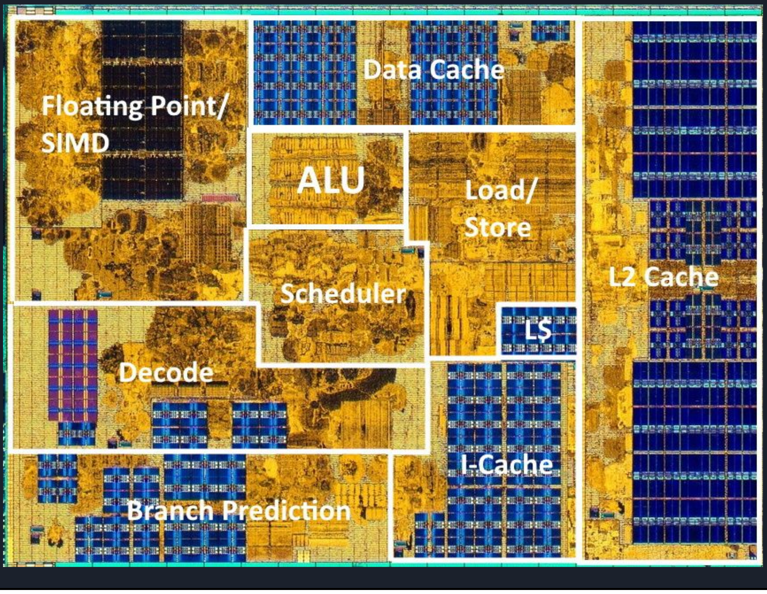
Main Memory
~10GB
50-200 cycles

SSD/HDD
~1000GB
50000 cycles

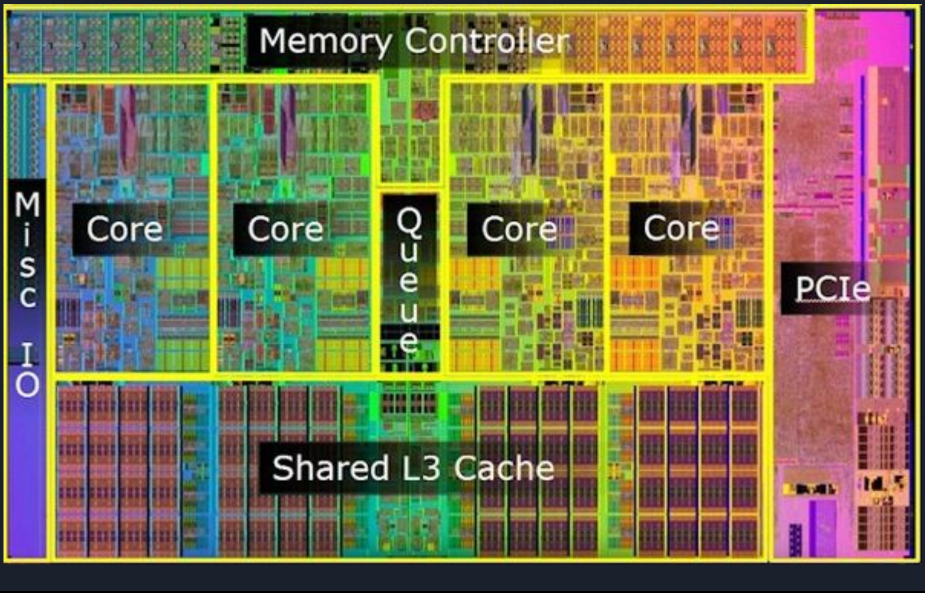
When trying to read from or write to a location in the main memory, the processor checks whether the data from that location is already in the cache. If so, the processor will read from or write to the cache instead of the much slower main memory.

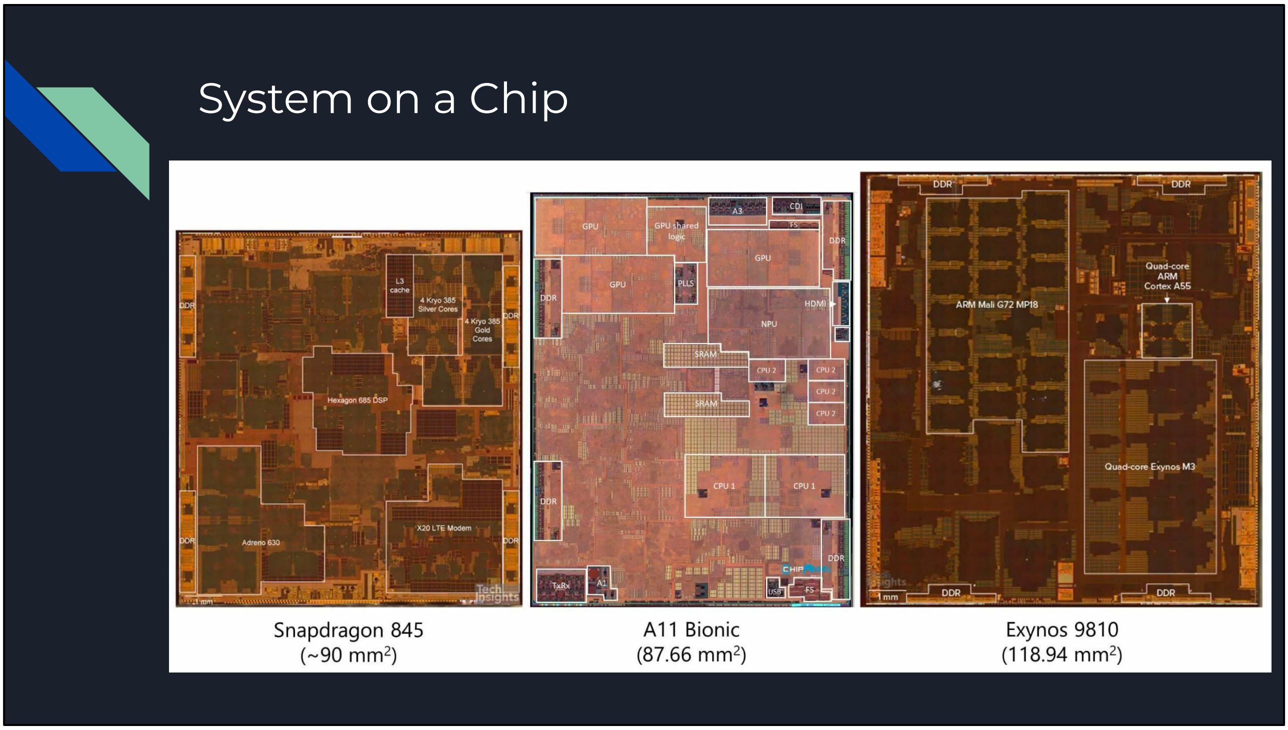
L1 cache is typically split into cache for data, and cache for instructions
L2 cache - per core
L3 cache - shared between different cores

AMD Zen 1 CPU Architecture



Intel i5 / i7 Architecture





System on a Chip

Snapdragon 845
(~90 mm^2)

A11 Bionic
(87.66 mm^2)

Exynos 9810
(118.94 mm^2)

Smartphone chips have accelerators for various functions:

1) Image processors
2) Neural processing units
3) Graphics processing units
4) Machine learning units
5) Crypto processors
6) Video encoding

Summary

- CPUs follow a set of instructions on inputs
- These instructions are provided as machine code
  - Opcode and operands
- Fetch - Decode - Execute / Instruction cycle
- Clock
- Memory hierarchy
  - Registers
  - L1 / L2 / L3 cache
  - RAM - Mike!
  - SSD / HDD - Mike!
- Optimisations
  - Pipelining
  - Multi-core
  - Accelerators