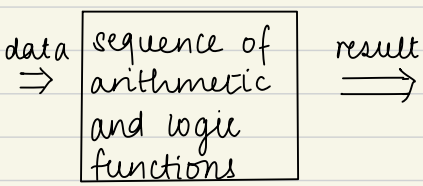
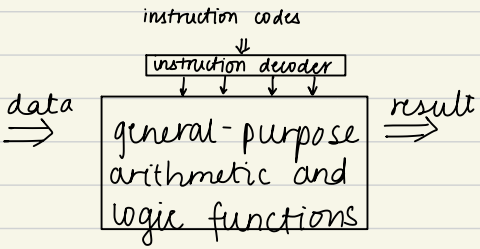
Role of memory in computing

1. approaches to computing

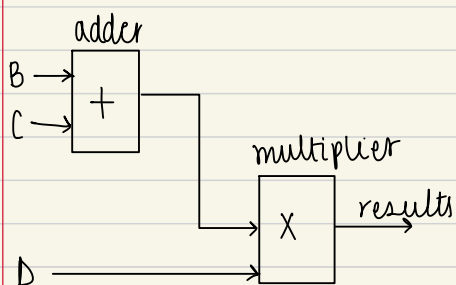
prog. in hardware



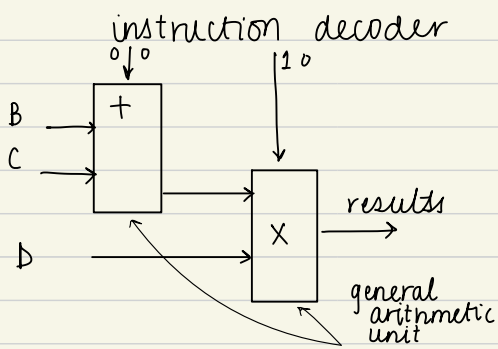
prog in software



implementing A = (B+C) x D



- circuit doesn't change, this is the only task it will perform
- fast computation, but not scalable



- slower and more complex but easily computable
- instructions and data need to be stored

2. Code, data and memory

- code is a sequence of instructions
- data are values these instructions operate on

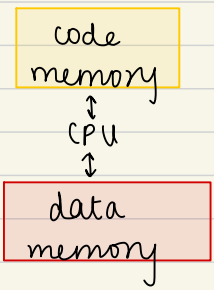
eg. $S = A + B$ code : add → store in memory → root
$x = \sqrt{S}$ data : A, B

- both code, and data is stored in memory
- memory is a sequential list of addressable storage elements for storing both, instructions and data

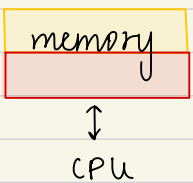
address	content	instruction
0x000	+	
0x001		
⋮		
⋮		
0x101	B	data
0x102	C	data
⋮		
⋮		
⋮		

3. Stored program concept

harvard architecture

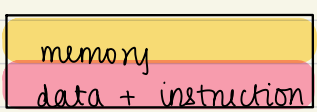


von neumann architecture



- most common in modern day computing

- input → CPU → output

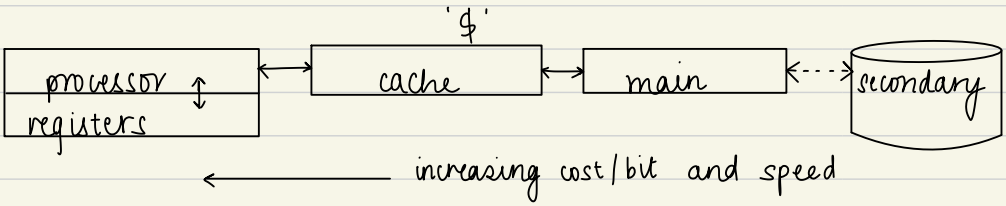


- contents of memory are addressable by location, without regard to data type
- execution occurs sequentially (unless explicitly modified)

the method of storing everything in one place works caus

4. memory hierarchy

- memories are organised in levels of increasing speed and cost/bit



a) registers : → a number of flipflops arranged to form a "word"
↓ multiple words
form a "file"

operates at CPU clock rate

→ fastest storage elements in the processor
→ can have very few since they are large and have huge energy consumption
→ 2 - 128 KB

b) cache memory : → very fast, hardware managed memory
→ built w/ static RAM
→ uses another structure to store memory (not flipflops) which is smaller than flipflops but higher energy
→ up to 512 KB

RAM : random access
D : dynamic
S : static

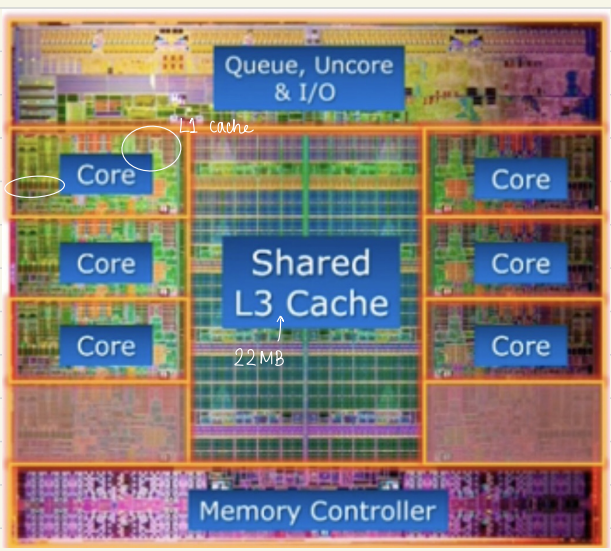
c) main memory : → usually DRAM (dynamic RAM, not static) or ROM
→ up to 16 GB
→ 30 - 70 ns access time

d) secondary memory : → not always random access
→ def. non-volatile
→ very cheap
→ magnetic or flash technology
→ 0.03 - 100 ms access time
→ up to 4TB

- volatile memory : when power goes off, the data vanishes.
registers, cache, main memory are all volatile

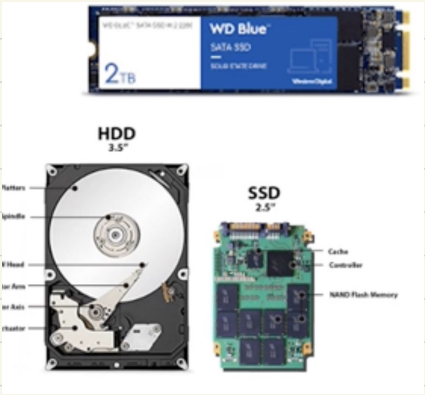
intel core registers + cache

registers 32KB



- main memory : DRAM

- secondary memory



5. Characteristics of main memory

- fix-sized (8-bit) storage location accessible at high speed and in any order
  - ↳ random access memory
    every address is equally accessible time-wise
- each byte-sized location has a unique address that is accessed by specifying its binary pattern of the address bus
- how do you decide how big the memory should be?
  it is a f^n of the number of lines of the address bus.
- ↑ lets say its 20 bits ∴ address will be $0 \rightarrow 2^{20}-1$ (2^{20} lines) and size of each location | depends on capacity of the databus

or memory capacity : $2^{20} = 1048576$
$= 1$ Mbyte

$2^{36} = 2^6$ Gbyte

1 KB = 2^{10}
1 MB = 2^{20}
1 GB = 2^{30}

Type	Bytes	Bits	Range	
signed char	1	8	-128 -> +127	
unsigned char	1	8	0 -> +255	
short int	2	16	-32,768 -> +32,767	(+/-32KB)
unsigned short int	2	16	0 -> +65,535	(64KB)
unsigned int	4	32	0 -> +4,294,967,295	(4GB)
int	4	32	-2,147,483,648 -> +2,147,483,647	(+/-2GB)
long int	4	32	-2,147,483,648 -> +2,147,483,647	(+/-2GB)
long long int	8	64	-(2^63) -> (2^63)-1	
float	4	32		
double	8	64		
long double	12	96		

6. number representation

ansi c reps nos in various ways

- int : signed or unsigned ← changes range (0-255 or -128-127)
- float : scientific calc + precision
  issue of trading off precision and range
  always signed
  1.2345×10^{12}
  precision range

- some CPUs are only int. based and make use of an additional floating point unit (FPU) to support fp. computations

7. Storing numbers in memory

add	memory (1 byte)
N	
N+1	
N+2	
N+3	

32 bit data.

MSB			LSB
0x1F	0x3D	0x00	0x49

↖ byte itself is not altered.

Big endian format : N: MSB, N+3: LSB
Mac, Sun, Motorola

Little endian format : N: LSB, N+3: MSB
Intel, Dec, Alpha

8. Character Representation

- char needs one byte
- data in binary is transformed into char through encoding standard
  - ↳ most common is 7-bit ASCII (now we use unicode ver 13.0)

MS / LS	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

- is contiguous
- byte · MSB + ascii
  - ↓ parity bit for error checking (1s)
- CR : character return (cursor to beg)
- LF : line feed (newline)
  - ↑

so to type in a new line, we need both

9. Boolean Representation

- _Bool in <stdbool.h> header file
- False = 0, True = non-zero
- memory storage for boolean variables is inefficient as most implementations just use the whole byte for the one bit

↑

to solve : 8051 microcontroller has 128 bit-addressable memory location → 16 bytes

↑

useful for ON/OFF controls.

10. Array Representation

- linear array of homogeneous datatype
- consecutive area in memory

↓

and so elements are accessed from the base address (BA) w/ appropriate off-set

↓

take into account the datatype

- usually big endian (?)
- 2-D arrays are flattend

int K[3][2] ≡ K00
= K0[2] K01
K1[2] K10
K2[2] K11
K20 K

4 bytes each

K[3][2]
K[a][b]

offset : sizeof(int) × ((2*a) + b)

10. Array Representation

- linear array of homogeneous datatype
- consecutive area in memory
  ↓
  and so elements are accessed from the base address (BA) w/ appropriate off-set
  ↓
  take into account, the datatype
- usually big endian (?)
- 2-D arrays are flattend

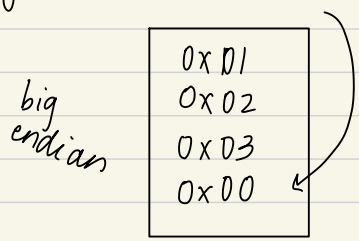
int K[3][2] ≡ K00
≡ K0[2] K01
K1[2] K10
K2[2] K11
K20
K21

4 bytes each

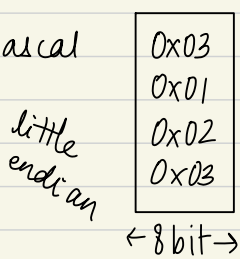
K[3][2]
K[a][b]
offset : sizeof(int) × ((2*a) + b)

11. String Representation

- char array + 0x00 ← null in C
  "123/0"



- Pascal



← stores length of string

max length of a pascal string: 255
2^8 + 0xFF as the first

12. Structure Representation

- each datatype in a declared structure variable occupies predefined consecutive locations based on datatype size

data padding : for data alignment issue ↑ address must be even

what problem??
if 32 bits → 4 byte so div by 4

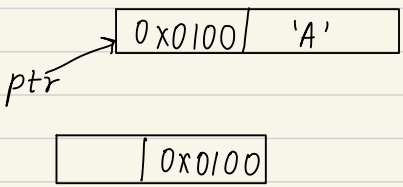
alignment has to do with the width of the databus. so 8 bit doesn't need to be aligned.

we still prefer the largest possible data bus because speed

13. Pointer Representation

- value stored is an address
- size is fixed regardless of the datatype (2 bytes)

```
c = 'A';
char ptr = &c
```



assuming addresses in CPU are specified at 16 bits

if memory is 8 bits
and address is 12 bits
ptr will use 16 bits

little endian

int address must be divisible by 4

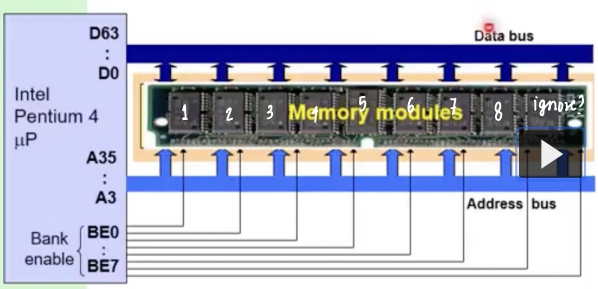
↑ because int is 4 bytes

14. Data Alignment

- most comp systems have restrictions on allowable address for accessing various data types
- multibyte data (int / double) must be aligned to addresses that are multiples of 2, 4, or 8
- programs written with microsoft (visual C++) or GNU (gcc) and compiled for a 64-bit intel processor uses

Data Type	Size (Byte)	Example of allowable start addresses due to alignment
char	1	0x..0000, 0x..0001, 0x..0002
short	2	0x..0000, 0x..0002, 0x..0004
int	4	0x..0000, 0x..0004, 0x..0008
float	4	0x..0000, 0x..0004, 0x..0008
double	8	0x..0000, 0x..0008, 0x..0010
pointer	8	0x..0000, 0x..0008, 0x..0010

Intel Pentium 4

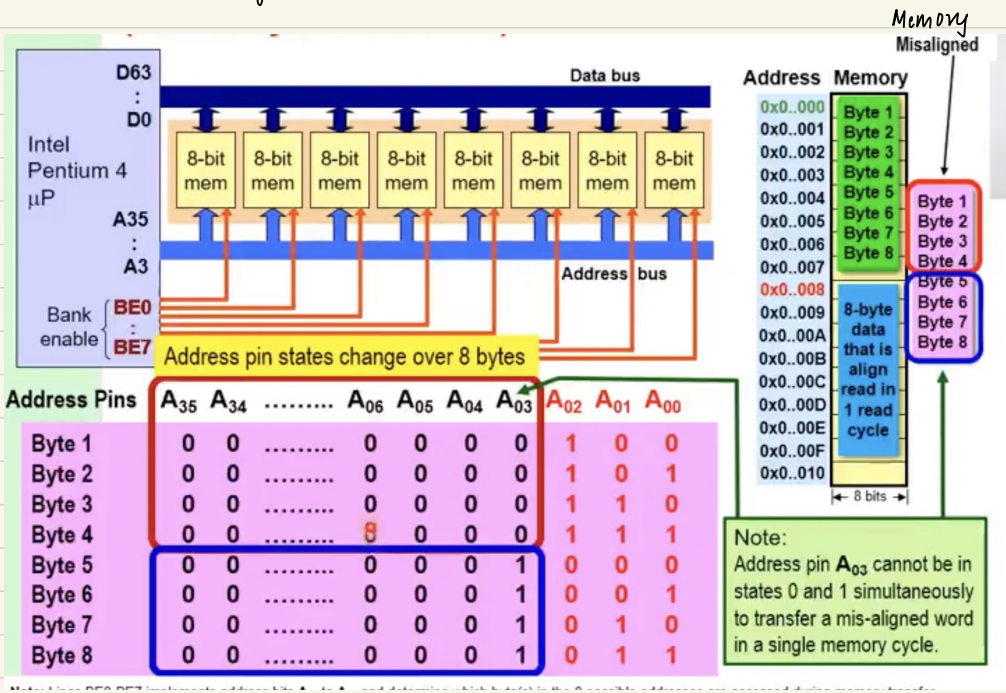


8 byte

8, 8-bit memory modules make up 64-bit data word size of processor

when stored right, 8 byte data can be transfered in one cycle

- Why is data alignment such an issue?

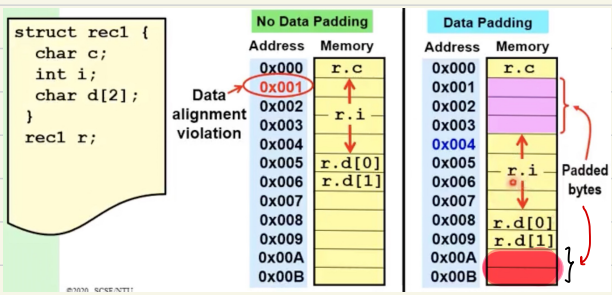


- Intel 8080 is an 8-bit processor and does not have any data alignment issues as it transfers all data byte-by-byte

- Motorola 68 transfers 2-bytes

- data padding : addition of useless bytes by compilers to ensure alignment of different data-types within a structure

- padded bytes are added at the end of last structure element and start of the next to maintain alignment in an array of structures



hence while constructing a structure, make sure the order obeys data alignment rules