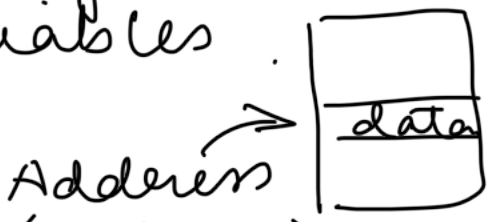
C - Week - 6 Notes

L1 Introduc^{n} to Pointers

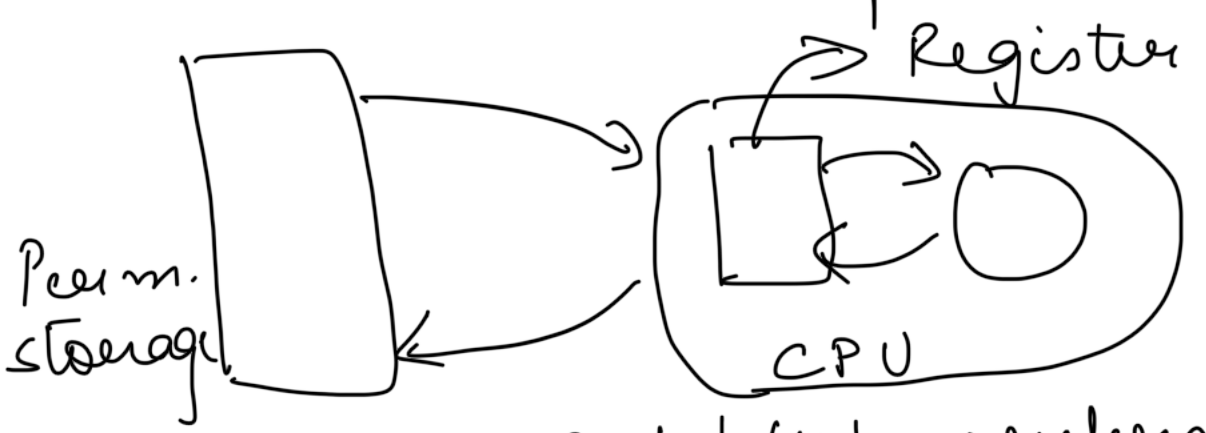
- Recall - memory (bag of words)
  ↳ write, read, address
- store some set of bits → width
- finite space of addresses → capacity
- Memory
  → RAM → in GBs on most computers, store arbitary data on instructions.
  → conventions → byte addr. address (not bit, not word)
  → binary (2^{10}, 2^{20}) vs decimal (10^3, 10^6)
- Pointer → convenient way to store an address, conceptually closer to hardware than variables




```
int x;
int *p;
p = &x // get the address of int.
       read & store it in the pointer
int y = *p // dereference the pointer
           get value from the mem.
*p = 10 //directly update the
          memory loc$^{n}$.
```

- Bits - actual unit of storage in memory

- Bytes - conventional unit for addressing
- Word - Practical unit of usage, usually related to CPU registers, def^n is hardware & OS dependent.



32b | 64b → refers to registers → impacts the size of words, pointers, etc.

- int (≥16, now 32bits), char(8), short(≥16), long (≥32), long long(≥64)
- #include <stdint.h> ⇒ to get exact width

L6.2 Endianness

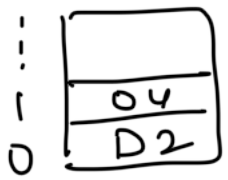
- only binary is stored inside computer memory, other formats are more readable.
- atleast 2 bytes are req. to store 1,234. Order of bytes in memory matters.
- Endianness → choice of storage impacts interpretation.

1. Big-Endian ⇒ L to R

↳ MSB is at the lowest address

[04 D2]
0 1 2 ...

2. Little-Endian ⇒ Top to Bottom
↳ MSB is at the highest address



Eg. 0x 12 34 56 78 (4 bytes)

Big -

0 → 0x12
1 → 0x34
2 → 0x56
3 → 0x78

Little →

3 → 12
2 → 34
1 → 56
0 → 78

- X86 → Big Endian, many older machines are big endian
- Network byte order - big endian

L6.3 Alignment

int x = 0x 12 34 56 78

Add[x] = 0

3	12
2	34
1	56
0	78

Add[x] = 2

5	12
4	34
3	56
2	78

- CPU bus → 32 or 64 bits wide

reading 4 bytes → 0 → 3, 4 → 7 } easy

2 → 5 but → 0 → 3, 4 → 7 } harder rearrangement

L6.4 Pointers & Variables

- int a = 10; → value
  - ↓ data type
  - ↓ name

&a → actual address

*(&a) → data value stored at address.

- int a;

  int *p;

  p = &a; → take the address of a & store it in p.

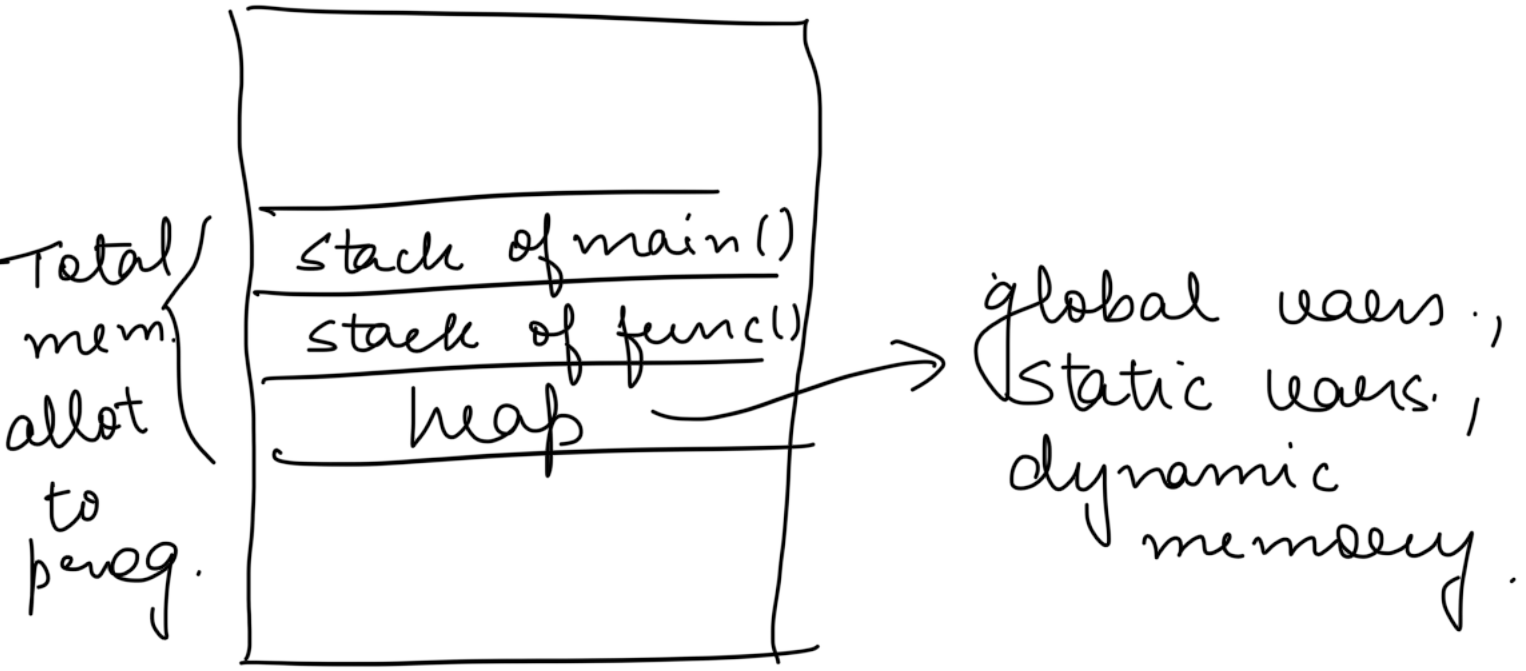
  printf("%lu", (unsigned int) p)

  ↓

typecast the pointer as an unsigned int, so that it can be printed.

- float x = 3.14;
  int y = (int) x
  x type casted to int.

float x = 3.14;
int *y = (int *)(&x);
no data conversion → 32 bit value at x is now interpreted as int.



L6.5 Compiler Organizations

- goal of compiler → correctness, speed, size
- eliminate unnecessary code (remove

unnece. R/W operations)
- might conflict with hardware reg.
- godbolt.org (compiler explorer)