In-class Notes (Lecture 1)

CS 354

REMZI ARPACI-DUSSEAU

google remzi

Machine Org + Programming

How stuff works

Today

Start:

How computer works

[right now]

@end:

course logistics

Breaks

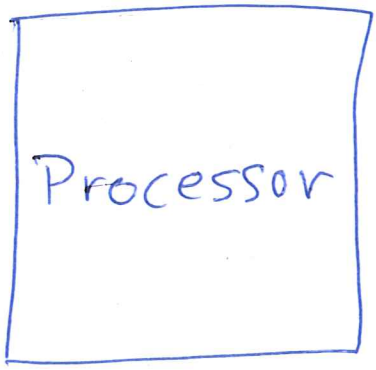
Three Parts:

1) C programming language

2) Lower-level programming (Assembly)

3) Beyond  Processing: input/output, mem systems,

Books:



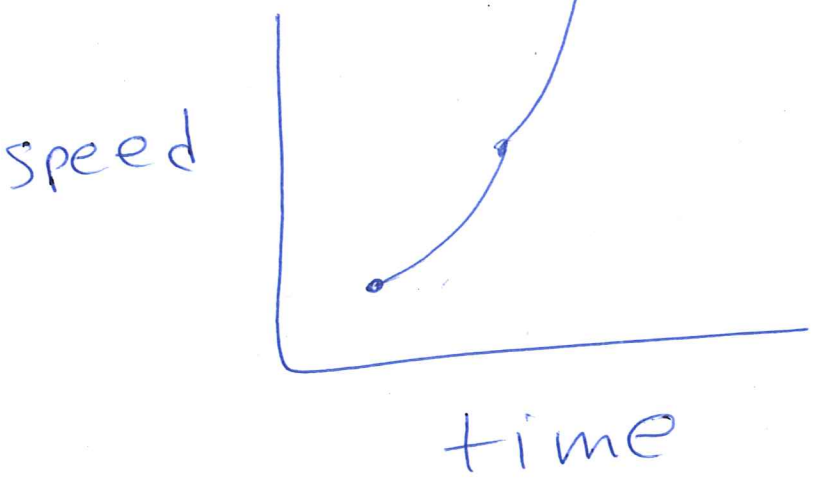
executes instructions:

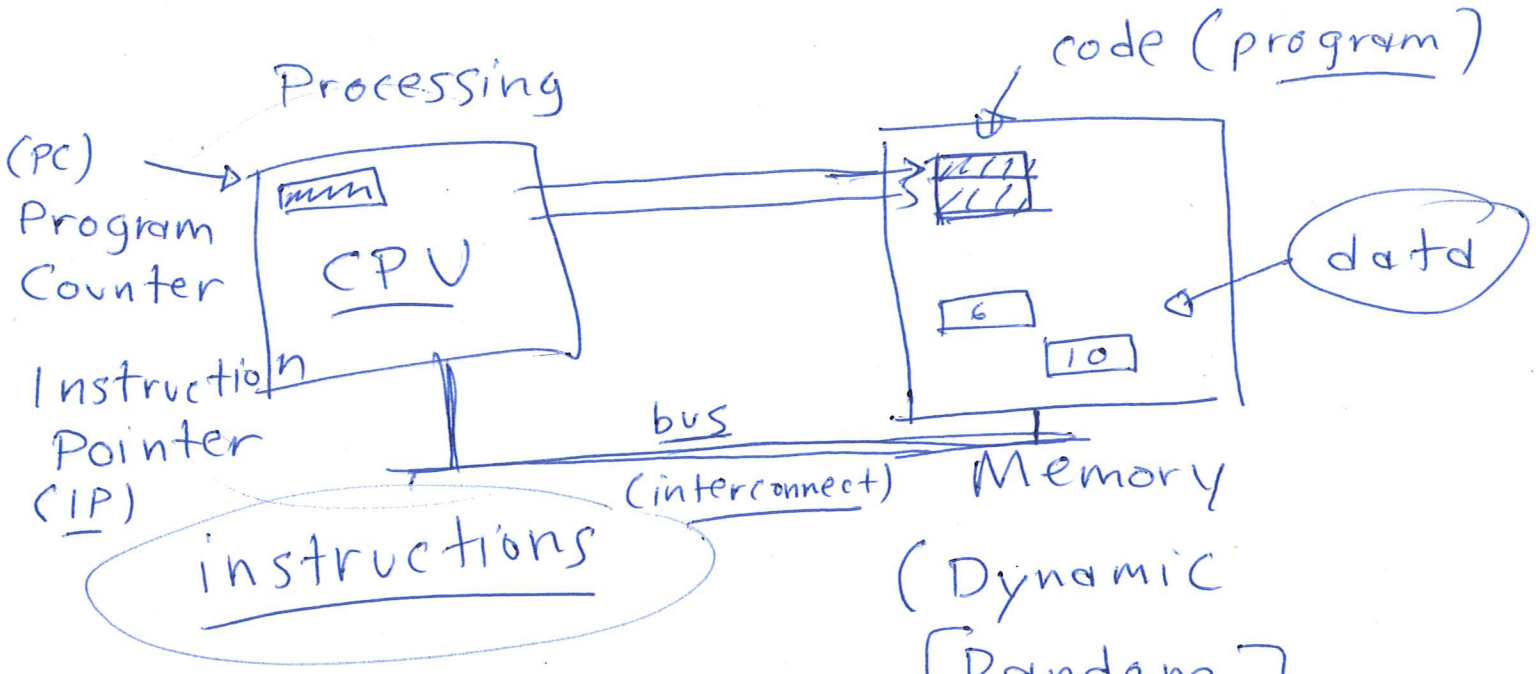
add (example)

Central Processing Unit (CPU)

speed:

billions/sec





(Dynamic Random Access memory: DRAM)

Processing:

Program is running

Loop:

Processor:

⇒ Fetch instruction from memory

⇒ Decode (figure out what the inst. does)

⇒ Execute the instruction

⇒ Get next instruction

memory:
10s or 100s of nano-secs

Processor:
exec. inst in ~1 nano sec

⇒ seconds

⇒ milli 1/1000

⇒ micro 1/1,000,000

⇒ nano 1/1e9

C programming language

Put C code / text on screen, want to understand each character

Bits / Binary / Hex

Inside computers:

Everything is represented by 0's and 1's

Why? real world of circuitry (analog)

⇒ digital

⇒ high / low ← reliably distinguish

voltage high (5.1V)

low (2V)

Represent as 1's and 0's

Numbers:

=> 5 =>

ENIAC

Bit => Binary Digit

=> decimal #s:

2 bits

00
01
10
11

=> 4 possibilities

2^2

B_{10}	B_9	...	B_1
1	0	0	... 0
0	1	0	... 0
0	...		0 1

10 bits => 2^{10} possibilities

Number in memory of computer:

3-bit binary

	X_2	X_1	X_0	$X_i = 0$ or 1
				decimal
	0	0	0	0
	0	0	1	1
	0	1	0	2
	0	1	1	3
	1	0	0	⇒ 4
	1	0	1	⇒ 5
	1	1	0	6
	1	1	1	7
	2^2	2^1	2^0	
	(4)	(2)	(1)	

$$\begin{array}{r} {\scriptstyle 1\,1} \\ 011 \\ +\,001 \\ \hline 100 \end{array} \quad \begin{array}{r} 3 \\ 1 \\ \hline 4 \end{array}$$

N bits ⇒

2^N possibilities

binary: a bit verbose for people

32-bit numbers (64-bit)

3 9

0011 1001 0111 1001

0000 1011 1111 0110

F

Hexadecimal:

each character: 16 values

0 ... 15

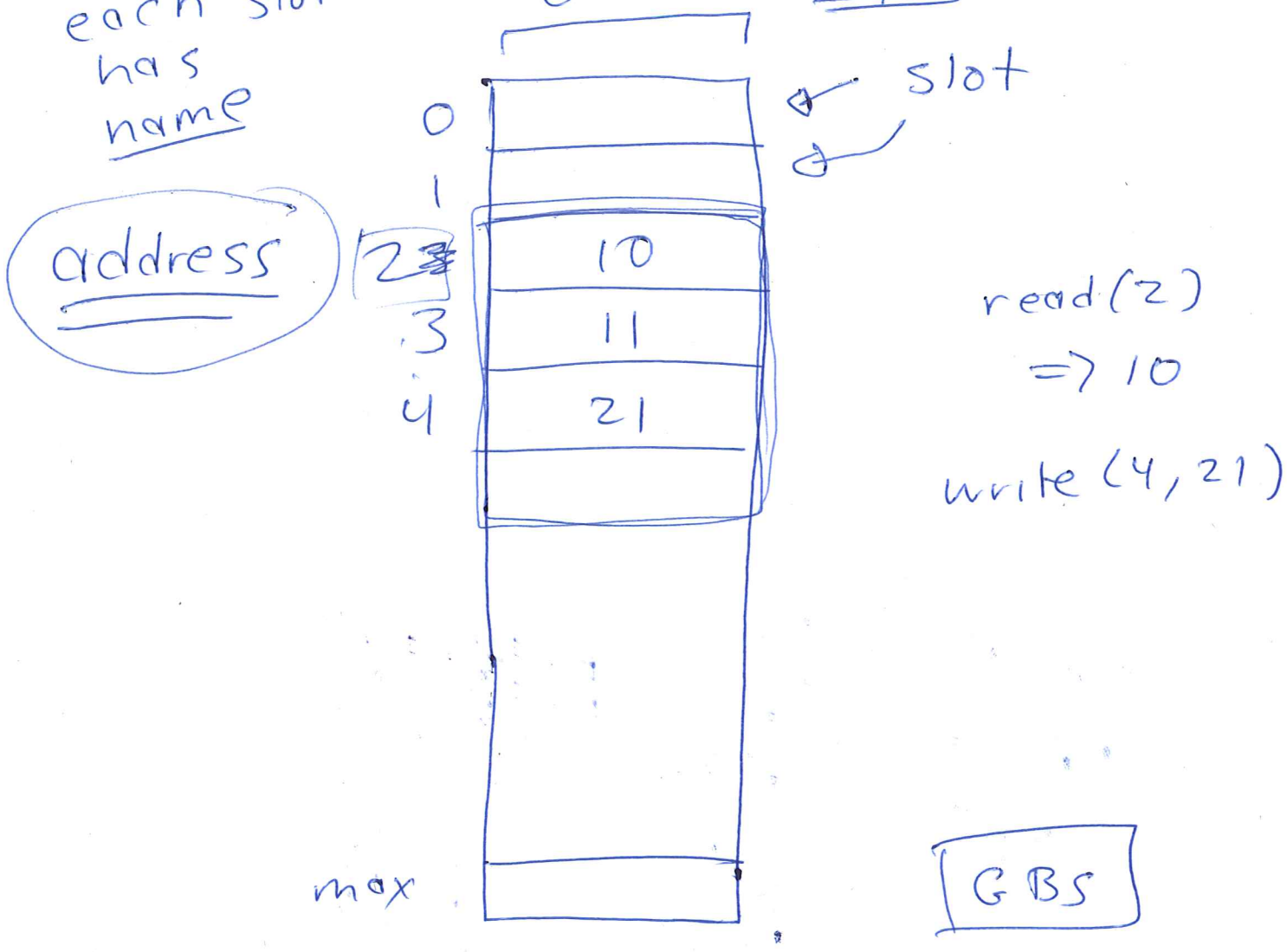
0 1 2 ... 9 A B C D E F

10 11 12 13 14 15

0x AAAA0000

zero and x

Memory: Abstraction / Model of Memory



Operations:

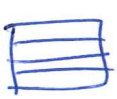
read(address) => returns value/data @ address
sometimes called *load*

write(address, data) => updates memory @ address to hold *value*

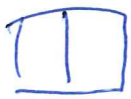
4 GB of memory

address:

how many bits do you need in address to refer to all 4 GB?

4 addresses 

2 bits 2^2



00
01
10
11

~thousand | KB : 1024 bytes

2^{10} 10 bits

~million | MB : ~~100~~ $1024 \cdot 1024$

$2^{10} \cdot 2^{10} = 2^{20}$

20 bits

~billion | GB : $1024 \cdot 1024 \cdot 1024$

30 bits + 2 bits

$\Rightarrow$ 32 bits $\Rightarrow$ 4GB memory

C : programming language

Numbers

5

23

-6

0x10

whole numbers

negative, positive integers

Operators (over integers)

=> divide

add

multiply

subtract

mod / remainder

unary operators

-3 +6

binary operators

2 entities

3 + 5

operators : binary

+ - * / %

expression :

3 + 5 ⇒ 8

(6 + 6) / 2 ⇒ ~~8~~ 6 (first)

order : precedence

higher: multiply, divide, remainder

lower: addition, subtraction

left → right

8 * 3 / 2 ⇒ 12

TIP:
when in doubt,
parenthesize

Law : C int is not a mathematical pure integer

C int : 4 byte (32 bits)

positive and negative and 0 (zero)

3 bits (not 32)

$X_2 X_1 X_0$

$2^3 \Rightarrow$ 8 numbers

$-4 \Rightarrow 3$

$-3 \Rightarrow 4$

positive

sign
0 => positive
1 => negative

			positive
0	0	1	1
0	1	0	2
0	1	1	3
0	0	0	0
1	0	1	-1
1	1	0	-2
1	1	1	-3

1	0 0
+1	0 0 1
-2	1 1 0
	1 1 1

-0

wrong

2's complement encoding

3 bits (for example)

	decimal
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

```
  1
+ -2
-----
 -1
```

```
 001
 110
-----
   1
```

```
  2
+ -1
-----
   1
```

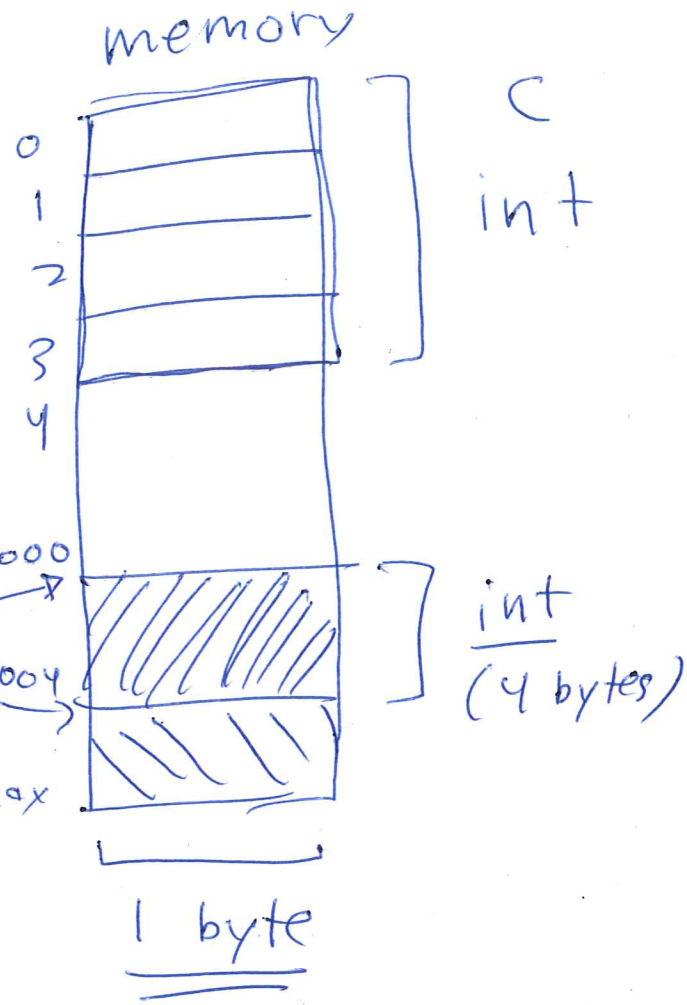
```
  010
+ 111
-----
 1001
```

1

Assignment: new operator] later
= (equals)

Variable: named container

readability => names for humans



C: define integer

int x; int y;

Legal names:

- lowercase letters
- uppercase letters
- underscore (_)
- digits (numeric)

} any mix: can't start w/ a digit

~~int x-y;~~

```
int x;
int y;
```

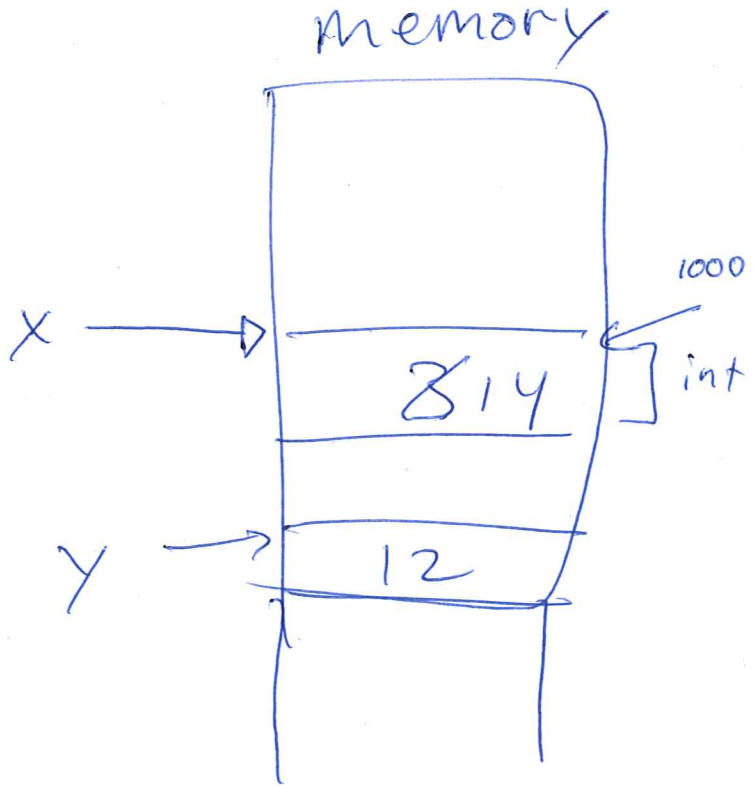
* / %

\+ −

new operator = (equals) = (low precedence)

```
x = 3;
y = 10 + (5/2);
x = (y + 2);
```

precedence



What is a good name?

```
int x;
int xx;
int y;
int yyy;
```

```
x = y / z;
```

```
mean = sum / count;
```

READABILITY

tension: concise, but readable

~~theSumOfAllTheNumbers~~, s

Comparisons : integers

> < >= <=

!= ==

result comparison:

True	False
1	0
(int)	(int)