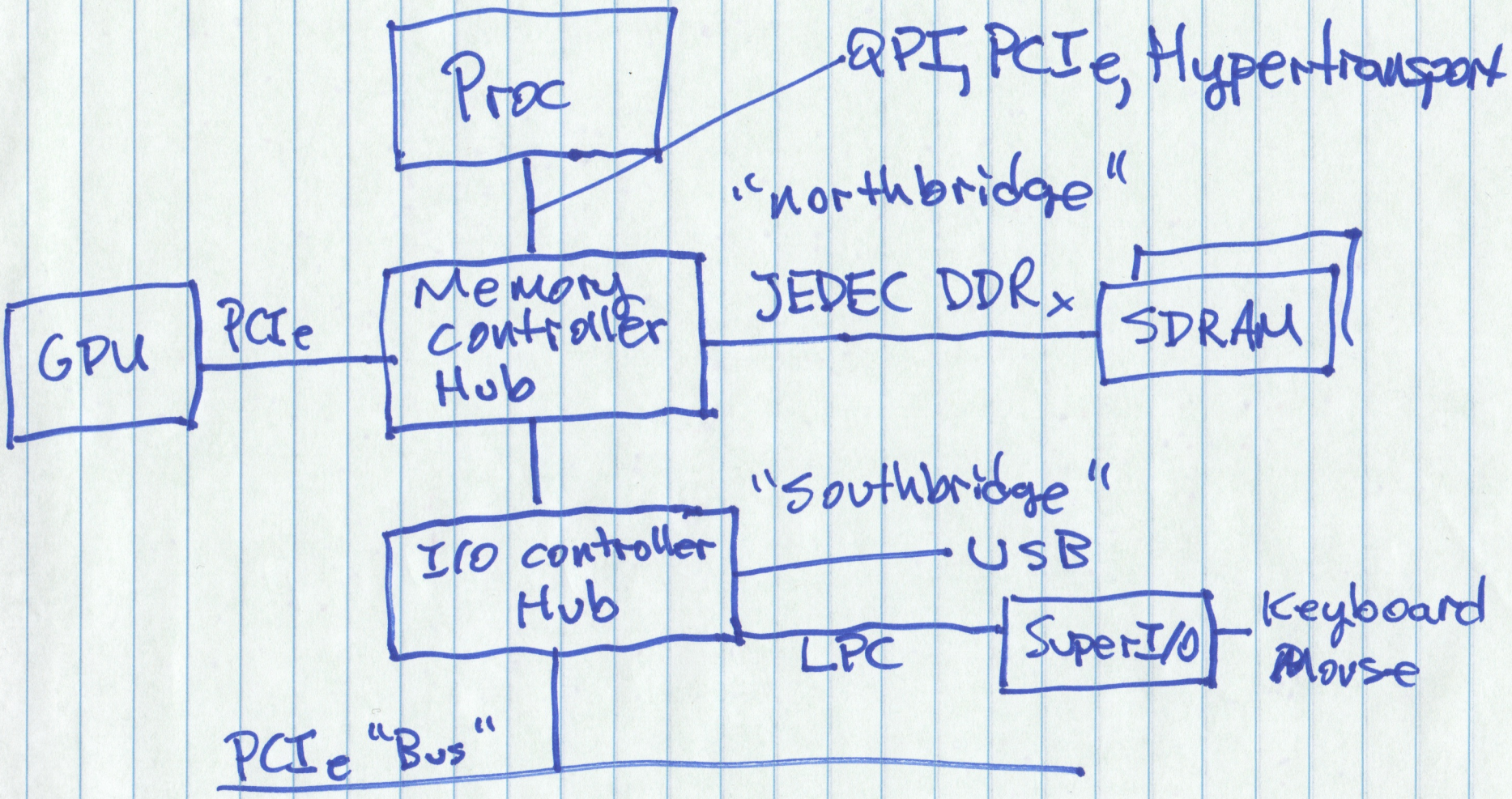
rebootingcomputing.ieee.org

4 things:

- Processors ("cores")
- Memory
- Peripherals
- I/O glue


Proc
QPI, PCIe, Hypertransport
"northbridge"
Memory controller Hub
GPU
PCIe
JEDEC DDR x
SDRAM
"southbridge"
I/O controller Hub
USB
LPC
SuperI/O
Keyboard
Mouse
PCIe "Bus"

Layers of abstraction

Problem

↓

Algorithm

↓

Video Game — Program

Compiler → Inst. Set. arch.

↓

Microarchitecture

↓

Circuits

↓

Devices

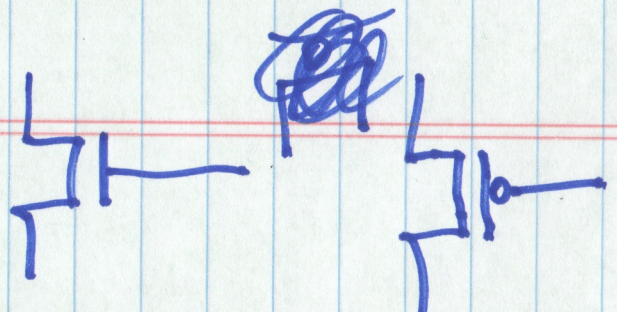
Software

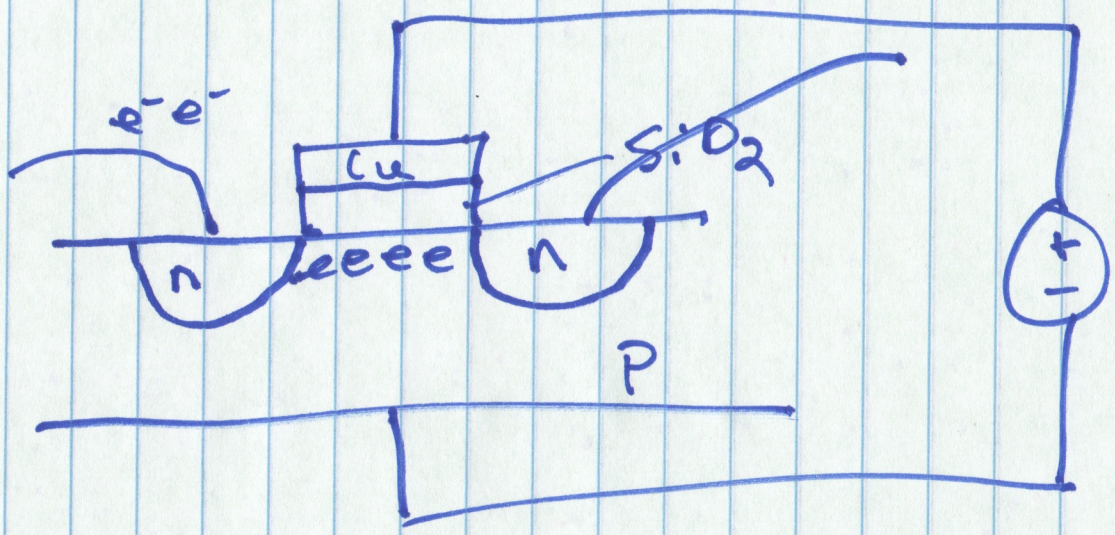
ISA

Mips LC3
x86 ARMv7, v8

Hardware

CMOS



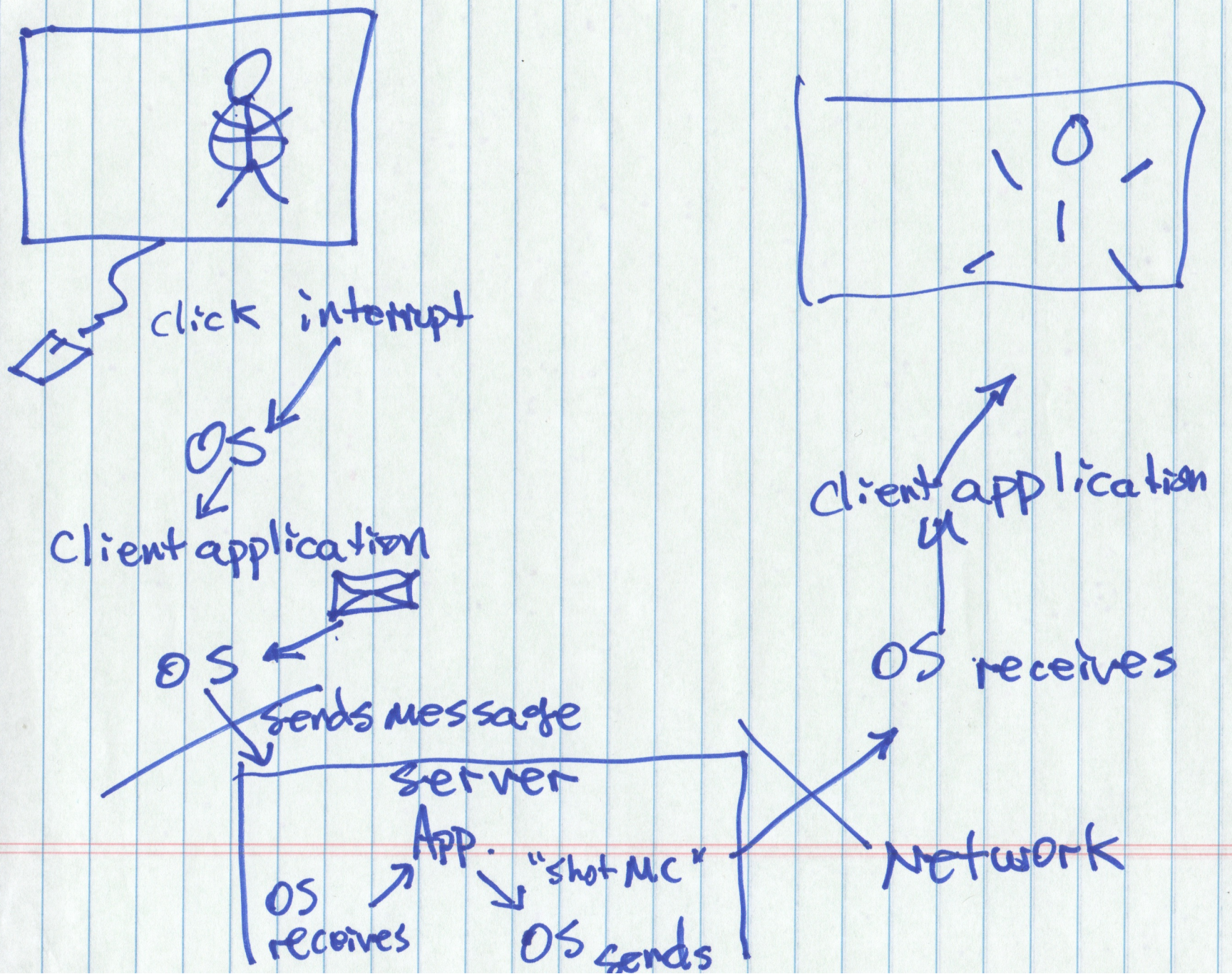

e⁻ e⁻
Cu
SiO2
n
eeeee
n
P
+
−

OS is _____ ?

Manages computer resources
Protection
Memory management
Interface to I/O

Multics

B → C


click interrupt
OS
Client application
OS
sends message
server
App.
"shot MC"
OS
receives
OS
sends
Network
OS receives
Client application

Processor ✓

|

Data path, control ✓

Interrupts

Performance

|

Pipelining ✓

|

Process scheduling in OS

|

Memory management

|

Caches ✓

Parallelism ✓

I/O disks

Networking

|

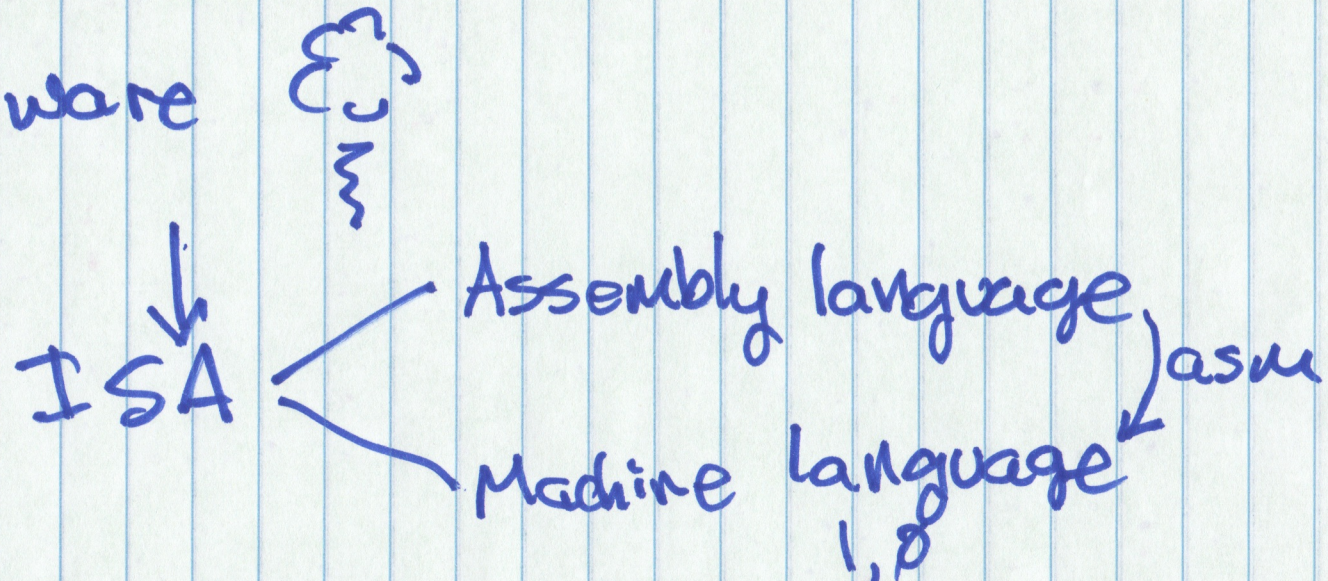
Disk scheduling

|

File systems

abebooks.com $14.65 + $4 S&H

ISAs Instruction Set Arch.



LC-3, x86, ARM, Mips (LC-2200), SPARC
PowerPC, Itanium, ARB, Z80, PDP-11, 68000
8051 AVR

ISA -

State of the running program

Instruction semantics & how they update the state

Instruction encoding -

syntax

1st Standardized ISA IBM-360 Gene Amdahl

Over time it gets more complicated

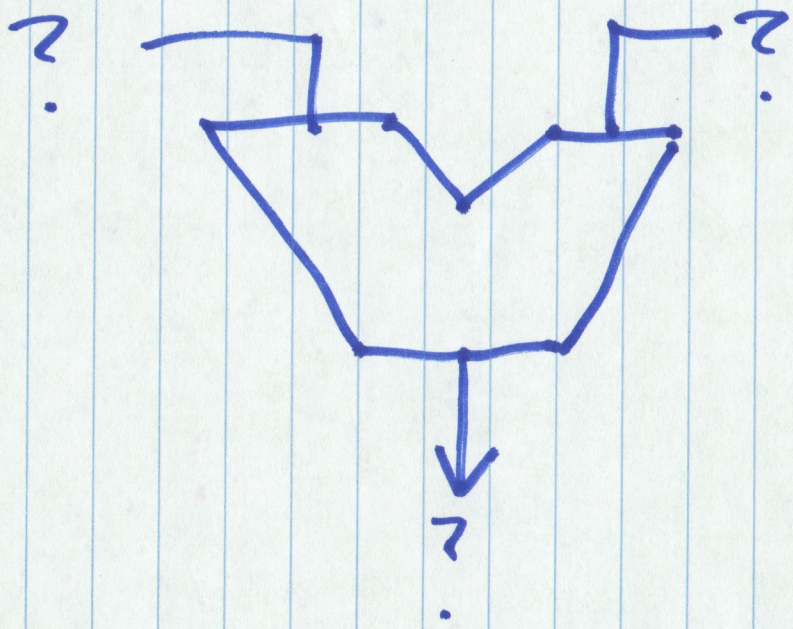
Care about...

- Fast
- Cost
  - Memory cost — Instructions are short, variable
  - Energy, Power → ≈ cooling cost
    - → Instruction semantics
- Ease of use
  - used to be: Humans
  - Today: can compile to it

Can't do everything in HLL

Compilers are not perfect

Hungry ALU Arithmetic Logic Unit

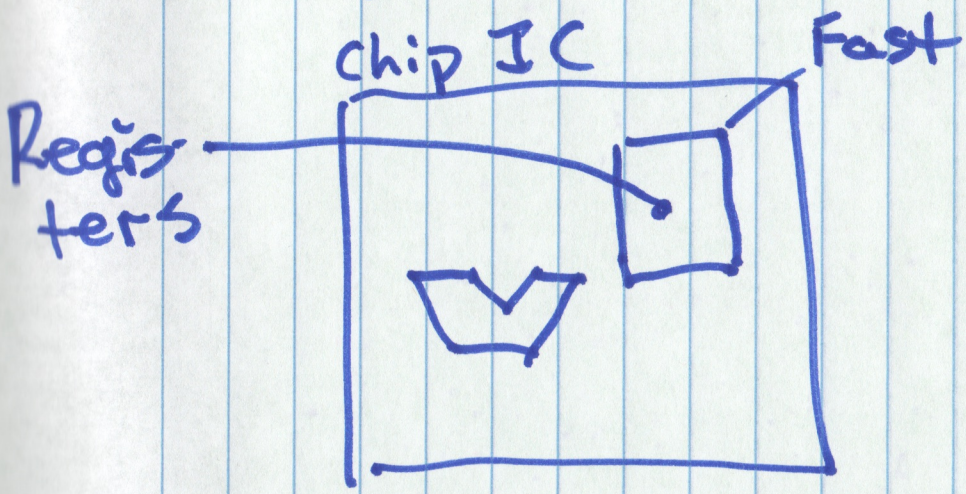


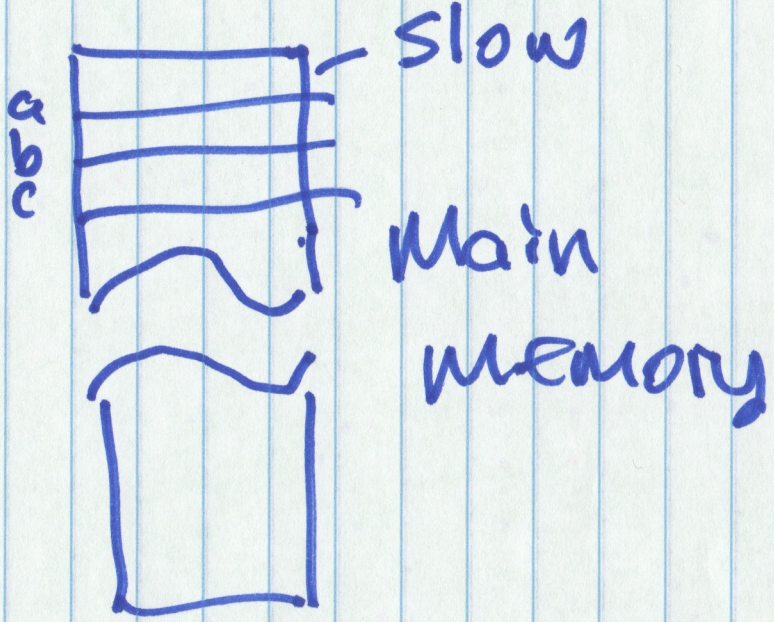
Does all calculations

```
a=b+c;
d=e-f;
g=h&i;
```

```
ADD a, b, c
SUB d, e, f
AND g, h, i
```

Operands are in memory





ADD a, b, C

SLOW

2 memory reads, 1 mem write

LC-3	8 Registers
LC-2200	16 Registers
x86	6 Registers

Ch 2

Using Registers

```
LD R1, B
LD R3, C
ADD R1, R2, R3
ST A, R1        ("ST R1, A")
```

① $X = B*B + 2*A*B + A*A$

Use more than once, putting in a reg. saves time!

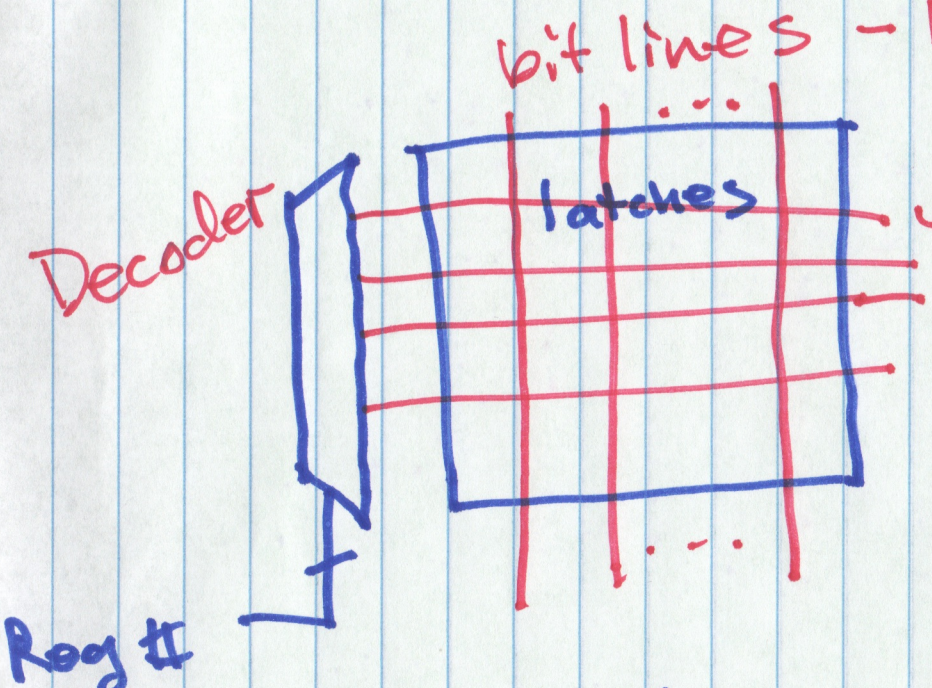
② Encoding

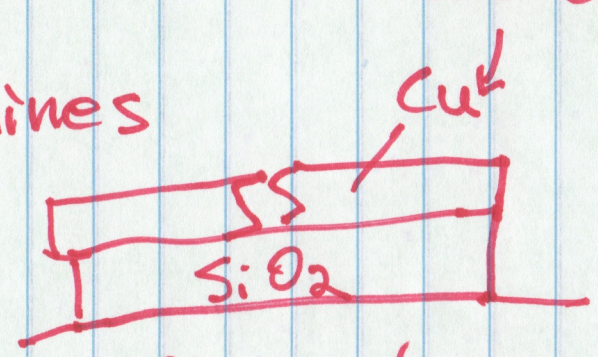
Addresses are today 64b

ADD A B C 192+ bits long!

Why not ∞ registers

1) Real estate on chip





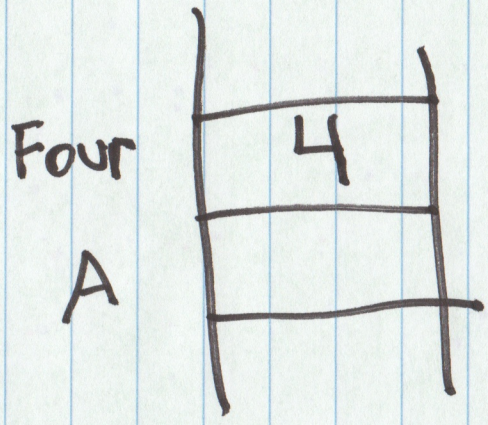
Si: "conductor"

Time ~ C

2) Speed ↑

Constants "Immediates"

a = a + 4;

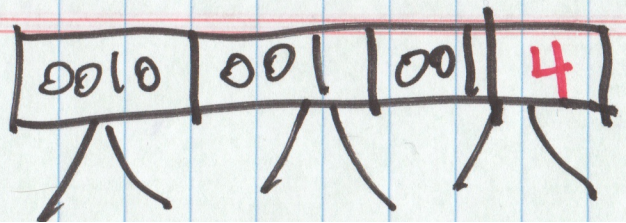



```
LD R1, A
LD R2, FOUR
ADD R1, R1, R2
ST A, R1
```

→

```
LD R1, A
ADDI R1, R1, 4
ST A, R1
```

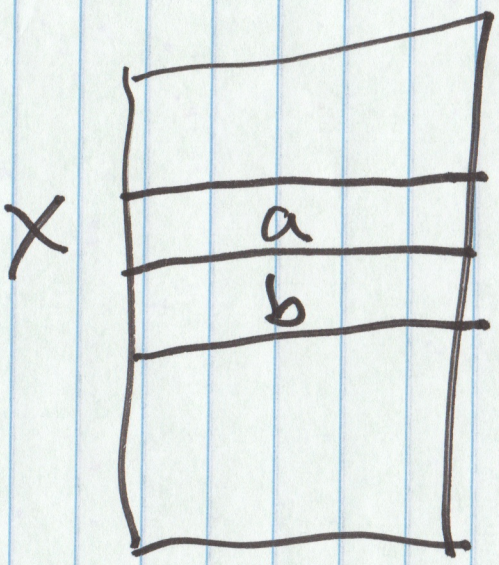
Inst. Reg.



```
struct {
    int a;
    int b;
} X;
```

X.a

X.b



R3

R1 ← Memory [X]

R2 ← Memory [X+4]

Base+Offset

```
LDR R1, R3, 0
LDR R2, R3, 4
ADD R1, R1, R2
STR R1, R3, 0
```

X.a = X.a + X.b

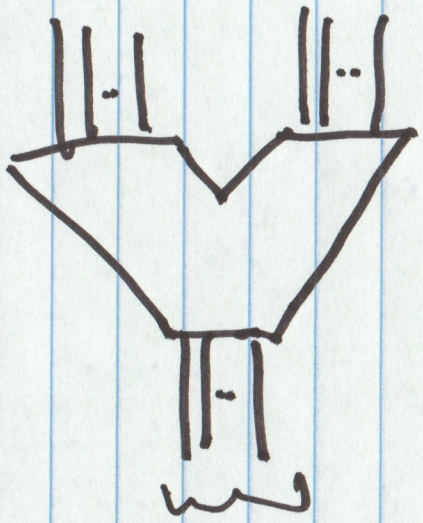
ADD R1, R1, R2

Base+Offset assembler syntax

```
LD R1, 0(R3)
LD R1, 4(R3)
```

↑ WHY?

Type	Size	ISA viewpoint
int	4 bytes	
char	1 byte	Byte
short	2 bytes	"word" wordsize
long	4 or 8 bytes	
float	4 bytes	
double	8 bytes	

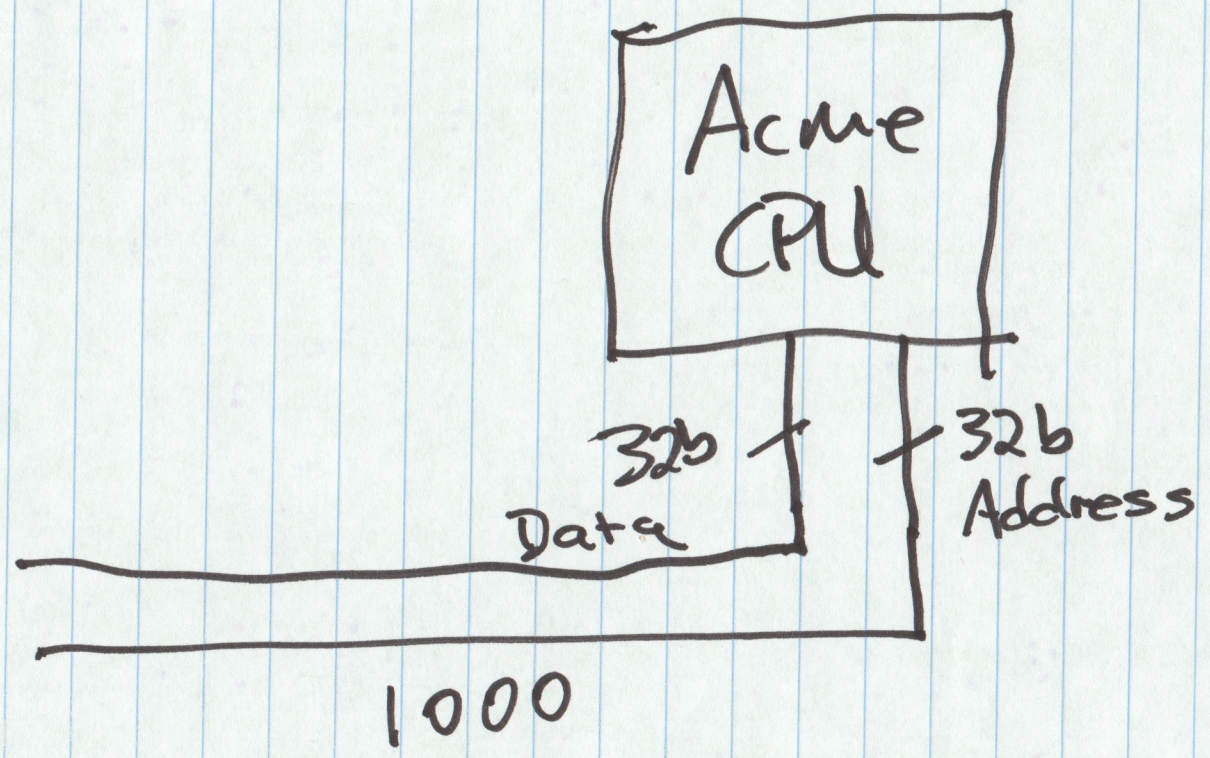


"Addressability"

1 byte

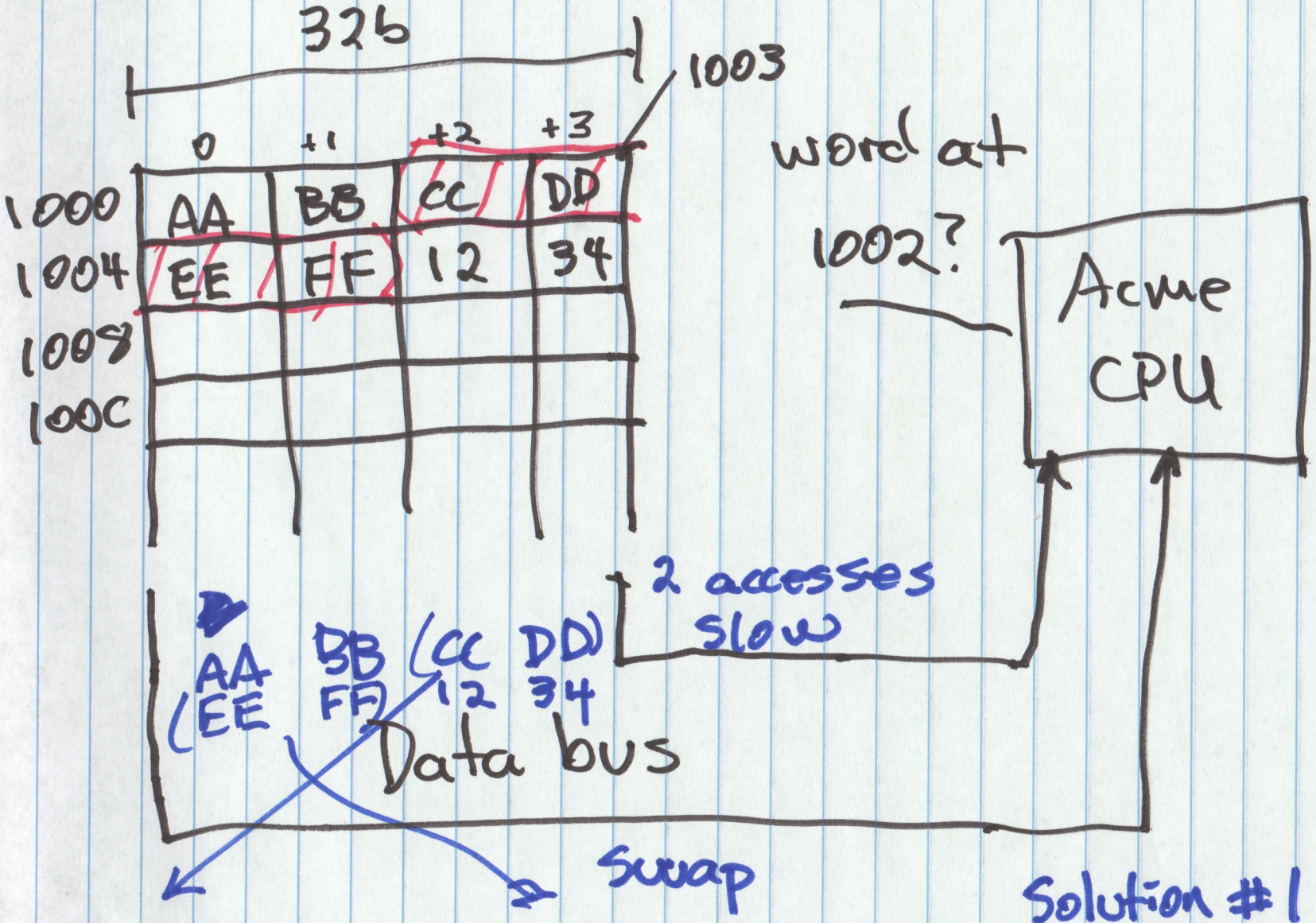
		Reg	Memory Address
LC-3	16b	16b	16b
8051	8b	8b	?
"Ours"	32b	32b	32b

	8b
1000	AA
1001	BB
1002	CC
1003	DD
1004	EE
1005	FF
1006	12
⋮	



① AA BB CC DD ? Big endian POWER xinu

② DD CC BB AA Little endian x86 "ours"



Unaligned Access

Solution #1
Add shift/swap
Realigning HW
Solution #2
Feature it

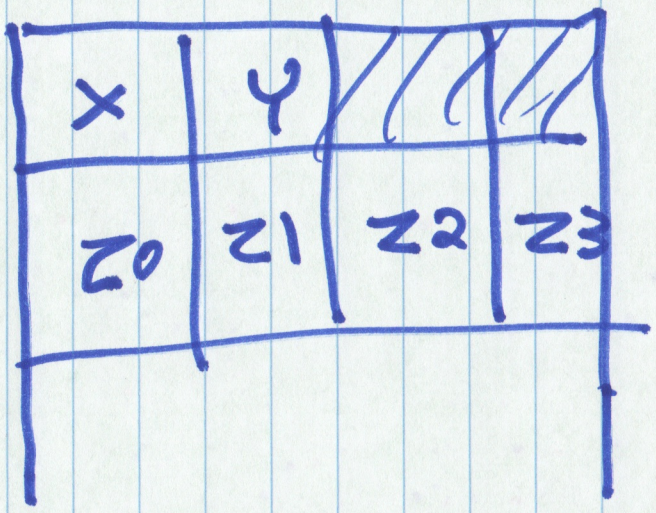
```
1000  char x;
1001  char y;
      int  z;
```

1B

4B

```
LD R1, x
LD R2, y
LD R3, z
```

Compiler protects you



```
int a[.], b[.]
for (i=0; i<N; i++)
    b[i] = a[i] + 4
```

a is in R1
b is in R2
i in R3

LB load byte
LW load word
LH load halfword

a + 4*i

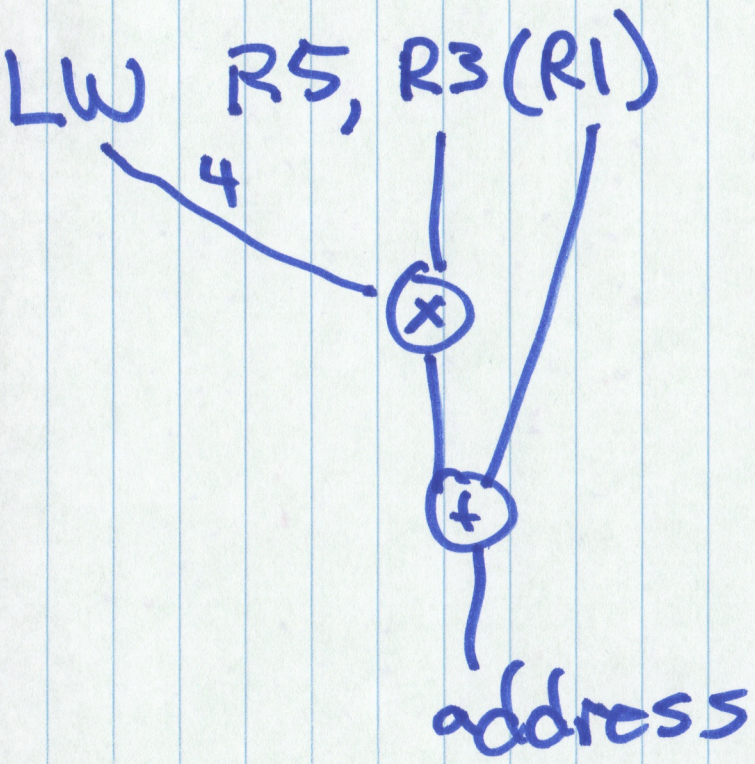
~~ADD R4, R1,~~

```
SHL R4, R3, 2
ADD R4, R1, R4
LW  R5, 0(R4)
```

} a[i]

.
.
.

New addressing mode: Base+index



Office hours cut short 2:15

Control flow

```
if (x == 0) {
    y = y + 3
}
y = y + 2;
```

```
BEQ R1, 0, label
ADDI R2, R2, 3
Label:
ADDI R2, R2, 2
```

```
if (x != 0) {
    y = y + 5
} else {
    y = y + 2
}
```

"jump"

```
BEQ R1, 0 Label
ADDI R2, R2, 5
J End
Label:
ADDI R2, R2, 2
End:
```

Unconditional vs. Conditional (Absolute addressing)

J <address> → PC ← <Address> (32b)

J R1

BEQ R1, 0, <address> : if(R1==0) { PC ← PC + <PCoffset>

<PCoffset> • 20b

Relative addring

Aside: unconditional branch

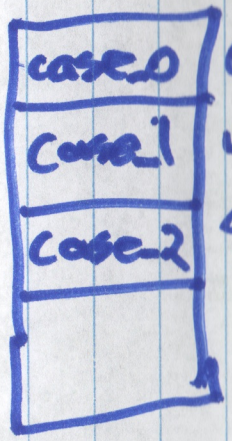
BEQ R1, R1, address

```
switch (k) {
    case 0:
        :
        break;
    case 1:
        :
        break;
    case 2:
        :
        break;
    default:
        :
        :
}
```

Convert to if-then-else

```
if (k==0) {
        :
} else if (k==1) {
        :
} else if (k==2) {
        :
} else {
        :
        :
}
```

Jump table



K is in R1

~~SHL R2, R1~~

~~LW R3, R1(R5)~~

~~J R3~~

Bounds check

```
LW R3, R1(R5)
J R3
```

Assembly

```
Case_0:
        .
        .
        .
        J Exit
Case_1:
        .
        .
        .
        J Exit
        .
        .
        .
Case_2:
        .
        .
        .
        J Exit
default:
        .
        .
        .
Exit:   .
        .
        .
```

```
for (i=0; i<5; i++){
    stuff
}
```

```
i=0;
while (i<5) {
    stuff
    i++;
}
```

→

```
    ANDI i, i, 0
Loop:
    BGE i, 5, Exit
    stuff
    ADDI i, i, 1
    J Loop
```

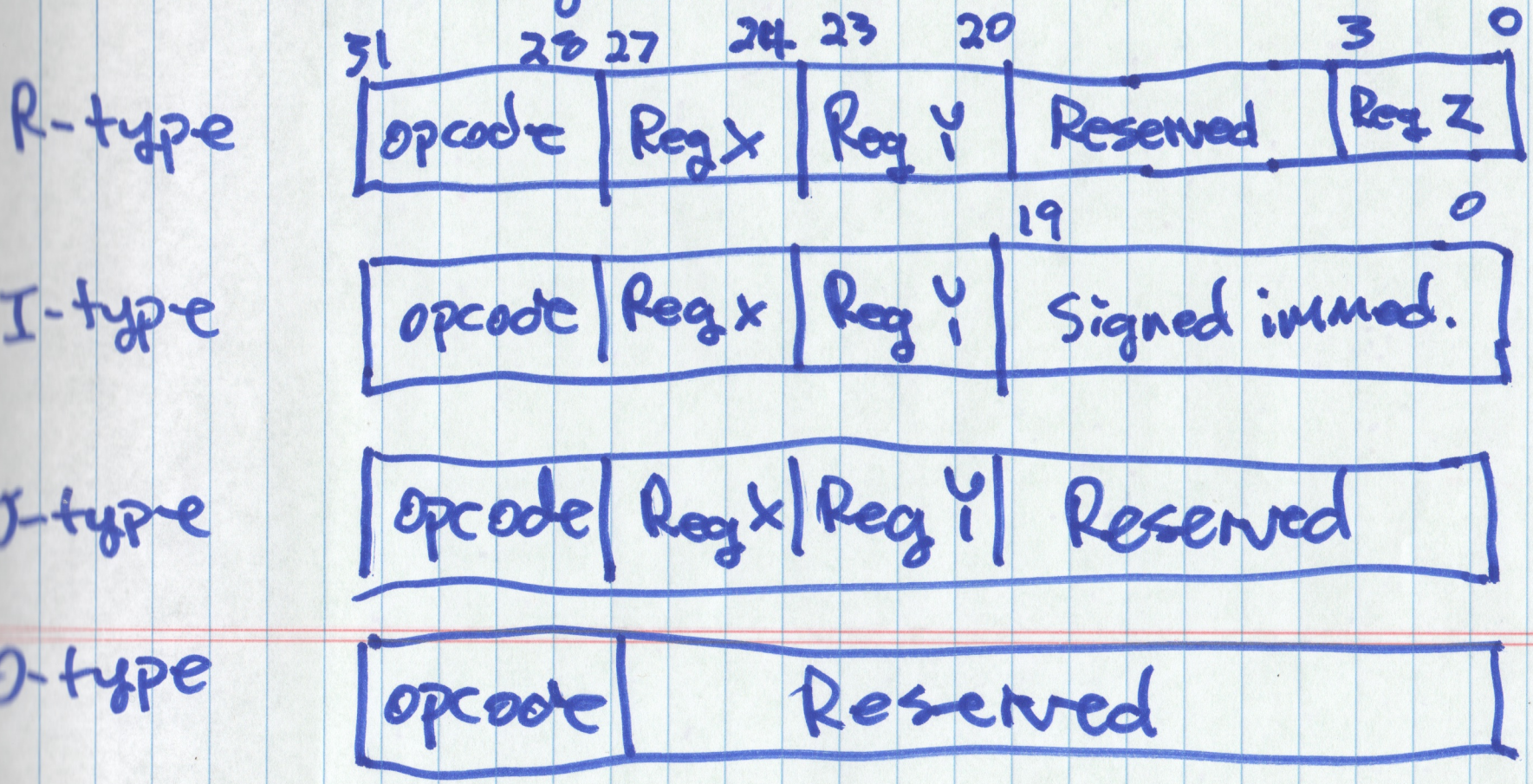
LC-2200

32-bit

16 Registers

Little endian

Fixed length instructions



add, nand	R-type
addi, lw, sw, beq	I-type
jalr	J type
Halt	O-type

$0	$zero	32 bits of constant 0
$1	$at	Assembler
$2	$v0	Return value
$3-$5	$a0-$a2	Arguments
$6-$8	$t0-$t2	Temporary
$9-$11	$s0-$s2	Saved general purpose regs
$12	$k0	OS
$13	$sp	Stack pointer
$14	$fp	Frame pointer
$15	$ra	Return address

Stacks

A push in LC-2200 is

```
add $sp, $sp, -1        push $ra
sw $ra, 0($sp)
```

A pop to reg $t0 is

```
lw $t0, ($sp)           pop $t0
add $sp, $sp, 1
```