For CS 250p ~~winter~~ fall 19

Discussion - 1

Lecture 7 - Rajeev

CS 3810 - Computer Organization

MIPS Instruction Set

start defining ISA

processor

more efficient in terms of processor usage ←

↑ 11

RISC ← e.g. MIPS architecture

Reduced IS computer

CISC e.g. x86, ARM, IBM & Power

ISA ← every computer comes with an ISA.

(1)

What's the purpose of an IS ?

The IS lies at the H/W S/W interface in a computer system.

Let's understand this through a simplistic example :-

C ⟶ a = b + c ;

compiler

add a, b, c ← operands

assembler

human-friendly

hardware-friendly 32-bit M/C instr

these can be a 100 different types of instructions ↑ serves

actually around 20-25 basic instns & variations of them

32 bits 32 b

4 Bytes 4B

usually

> 1 a line of C will converted to multiple lines of ~~H instructions~~ assembly instructions.

Skip slide # 17 on discussion of IS design

slide #'s 18 – 21

> There may be different ways you may generate ~~compiler~~ assembly code, some may be better than others. (use the term temporary location)

~~slide #5, #6~~

[No discussion on floating point operations.

$f = (g + h) - (i + j)$;

Compiler is more likely to generate the code on the left. (although temporaries are used)

[Due the design of the PL & compiler .... but it all depends]

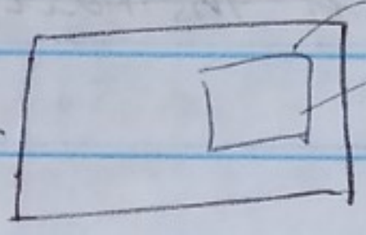
Lecture #8

Operand Locations

⇛

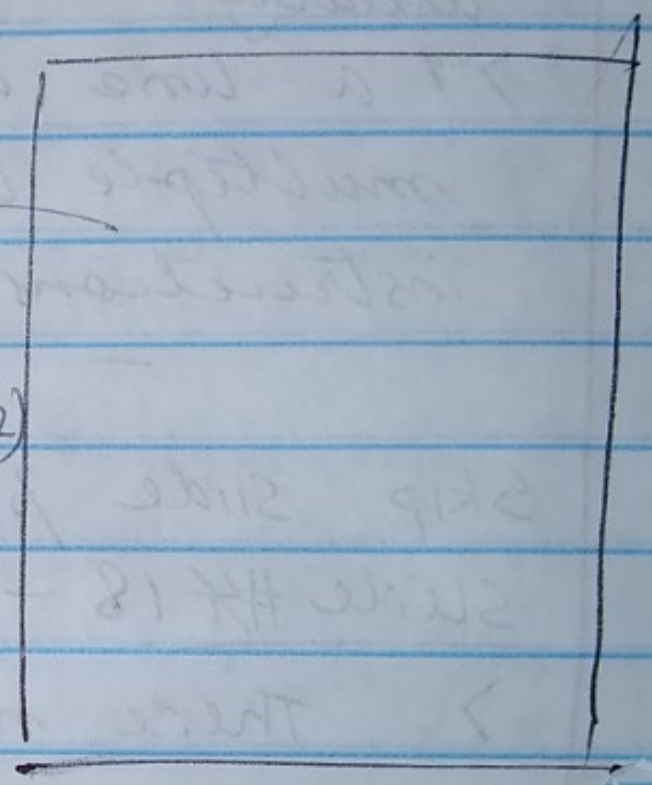
In x86 → 8 regs.

In MIPS → 32 regs.



Processor

Register file: containing a set of registers (#32) (each ~~taking 32 bits~~ of size 32 bits)



memory (8GB/16GB/32GB)

(1) In C all vars are located in the memory.

Explain as you draw.

(2) Accessing memory is expensive.

(3) To avoid repeatedly accessing memory, move values from memory to register

(4) MIPS, ~~in fact,~~ instructions, in fact, require operands to be register values only.

(5) So, registers are used as a scratchpad.

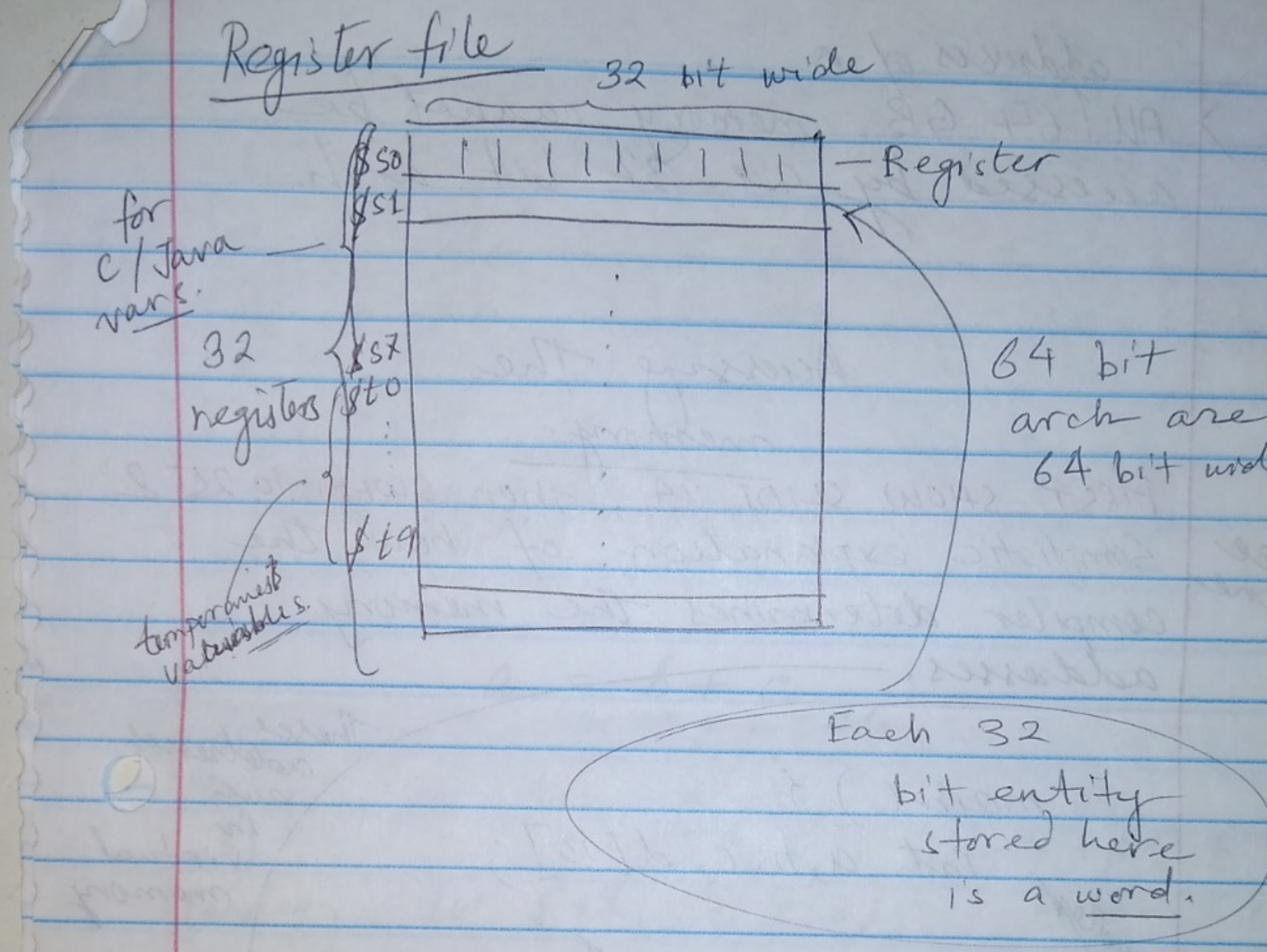
(6) Note register file is much smaller than the memory.

Tradeoffs

(7) less registers fo more expensive data transfer, more registers means more expensive resources on the processor (real estate)

add, & sub

Register file



Question

> How many addresses can a 32-bit word encode / represent?

$$2^{32} = 2^{10} \times 2^{10} \times 2^{10} \times 2^{2}$$
$$= 1k \times 1k \times 1k \times 4$$
$$\approx 4 \text{ billion values.}$$

Then your address space is 16 GB

> If you assume each address to refer to a 4-byte entity & you have 4 billion unique addresses.

addresses of a
> All 64 GB memory cannot be accessed by a 32 bit arch

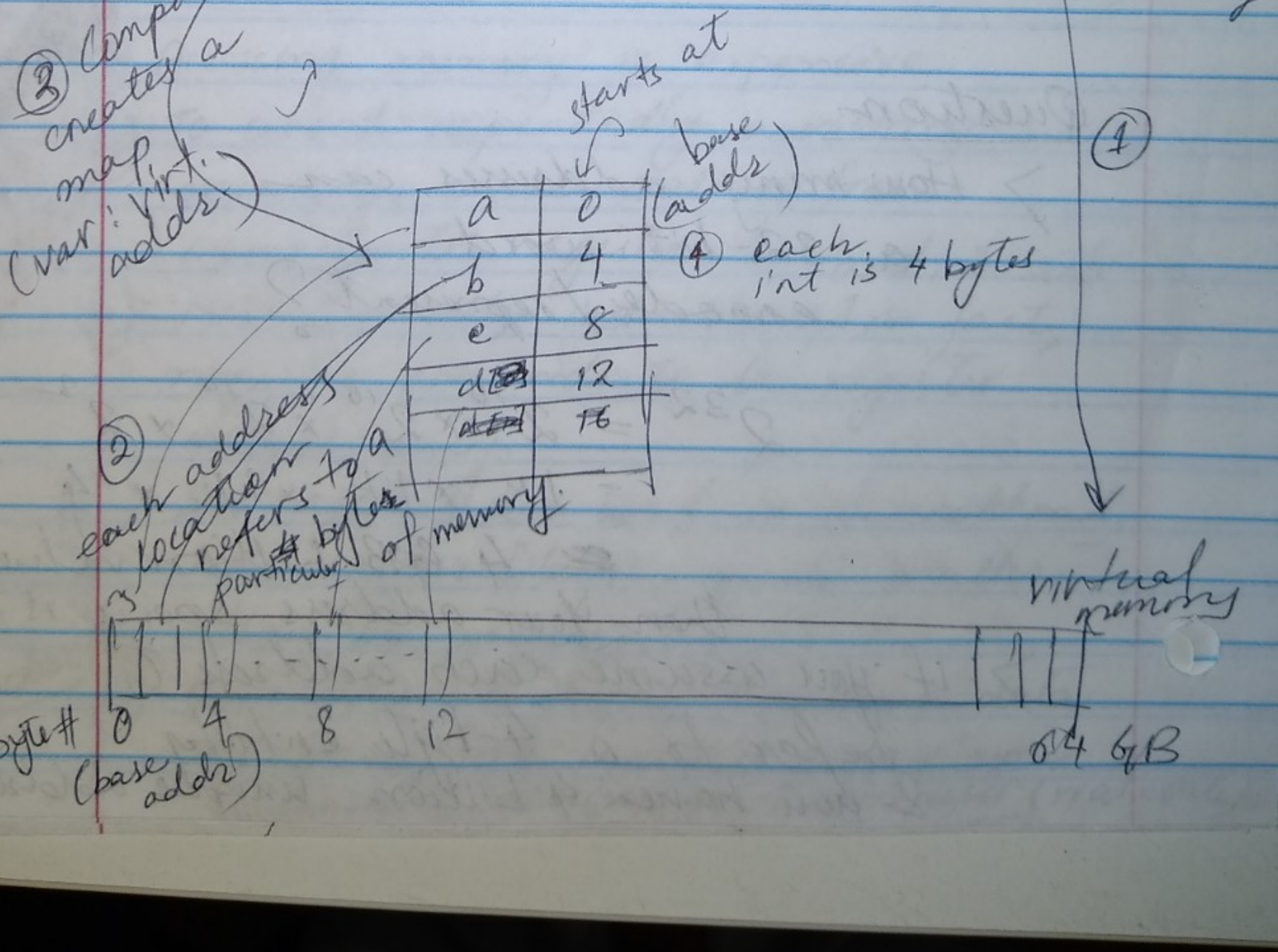
Accessing the memory.

FIRST SHOW SLIDE 24, then switch to 25, &

see none

Simplistic explanation of how the compiler determines the memory addresses.

```
main() {
    int a, b, c, d[10];
}
```



(5) compiler knows what's where in memory ~~registers~~ from the table

(6) compiler also know base address from register $t0 (lets say it's stored there)

Now, say we have,

$$a = b + c$$

the 1st available register

(loading b) lw $s0, 4($t0)

(loading c) lw $s1, 8($t0)

add $s2, $s0, $s1

(corresponds to address a)

(store in a in mem) sw $s2, 0($t0)

→ add 4 to address in t0, & move the value to $s0.

Immediate Operands

> Previously, we had, instructions like,

add $s0, $s1, $s2

~~addi $s0, $s1, 1000~~

Say u want to add
1000 to $s0

We use,
addi $c

there has to be
at least 1 operand
as a register

addi $s0, $zero, 1000