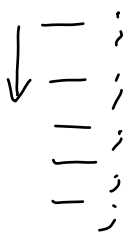
CS378: Machine Organization and Assembly Language

Lecture 2 – Winter 2009

What is an instruction? And a register?
What does register-to-register mean?
In what order is a program executed?



Why do we need memory?

Where are the instructions stored?

What is the C equivalent to: **sub $t0, $t1, $t2 ?**

$t0 = t1 - t2$

Announcements

- Website is up! Explore it (MIPS resources, Easter-eggs, etc)
- Homework 0 will be posted today – *not* graded, just for your benefit
  — on your own, explore SPIM.
- Homework 1 (for a grade, to be done individually) will be posted Friday
  — write a function in MIPS assembly
  — due a week later
- Lab 1 (to be done in partners) posted next week
  — Please find a lab partner soon
  — Or we will find one for you ☺

- Luis' office hours:
  M 1:30-2:30, or by appointment (in CSE 576)

- Textbook – sorry about the confusion. The bookstore only sells the newest edition. Take your pick, we will post readings for both editions of the book. Check for bugs on the 4th edition.

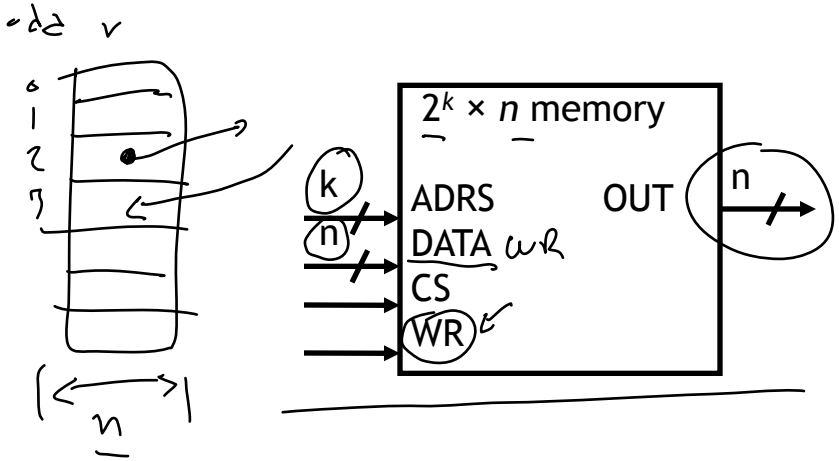
A more complete assembly example

- How would you write code in MIPS assembly to compute:
  - 1 + 2 + 3 * 4

```
addi $t0, $0, 1
addi $t0, $t0, 2
addi $t1, $0, 3
addi $t2, $0, 4
mul  $t2, $t1, $t2
add  $t0, $t0, $t2
```

Memory review

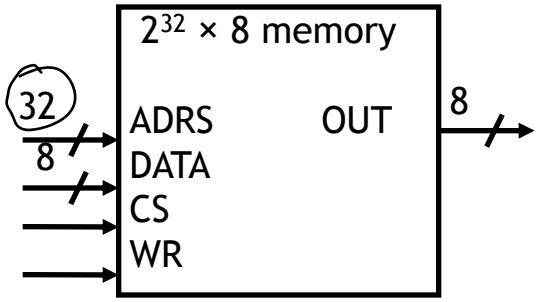
- Memory sizes are specified much like register files; here is a 2^k x n RAM.

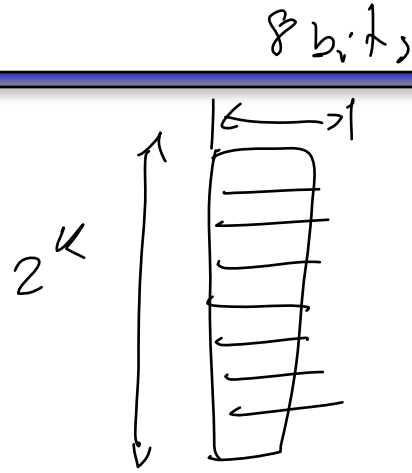


CS	WR	Operation
0	x	None
1	0	Read selected address
1	1	Write selected address

- A chip select input CS enables or "disables" the RAM.
- ADRS specifies the memory location to access.
- WR selects between reading from or writing to the memory.
  - To read from memory, WR should be set to 0. OUT will be the n-bit value stored at ADRS.
  - To write to memory, we set WR = 1. DATA is the n-bit value to store in memory.

MIPS memory



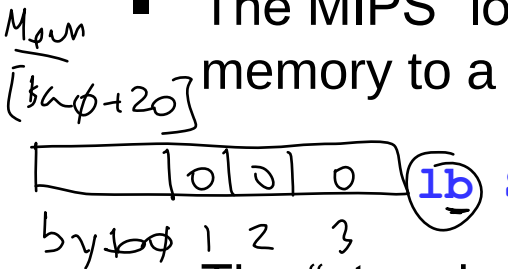


- MIPS memory is byte-addressable, which means that each memory address references an 8-bit quantity.
- The MIPS architecture can support up to 32 address lines.
  - This results in a 2^{32} x 8 RAM, which would be 4 GB of memory.
  - Not all actual MIPS machines will have this much!

address space

Loading and storing bytes

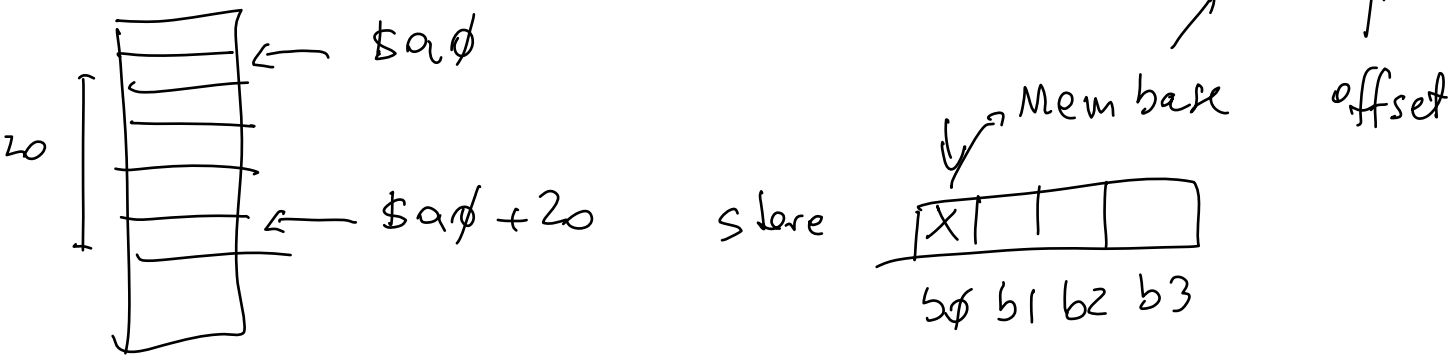
- The MIPS instruction set includes dedicated load and store instructions for accessing memory
- The main difference is that MIPS uses indexed addressing.
  - The address operand specifies a signed constant and a register.
  - These values are added to generate the effective address.
- The MIPS "load byte" instruction lb transfers one byte of data from main memory to a register.



```
lb $t0, 20($a0)     # $t0 = Memory[$a0 + 20]
```

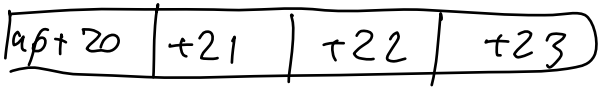
- The "store byte" instruction sb transfers the lowest byte of data from a register into main memory.

```
sb $t0, 20($a0)     # Memory[$a0 + 20] = $t0
```



Loading and storing words

- You can also load or store 32-bit quantities—a complete word instead of just a byte—with the lw and sw instructions.




```
lw $t0, 20($a0)          # $t0 = Memory[$a0 + 20]
sw $t0, 20($a0)          # Memory[$a0 + 20] = $t0
```

- Most programming languages support several 32-bit data types.
  - Integers
  - Single-precision floating-point numbers
  - Memory addresses, or pointers
- Unless otherwise stated, we'll assume words are the basic unit of data.

Computing with memory

- So, to compute with memory-based data, you must:
  1. Load the data from memory to the register file.
  2. Do the computation, leaving the result in a register.
  3. Store that value back to memory if needed.
- For example, let's say that you wanted to do the same addition, but the values were in memory. How can we do the following using MIPS assembly language? (A's address is in $a0, result's address is in $a1)

1 byte → 4 byte (word)

```
char A[4] = {1, 2, 3, 4};
int result;

result = A[0] + A[1] + A[2] + A[3];
```

```
lb   $t0, 0($a0)
lb   $t1, 1($a0)
lb   $t2, 2($a0)
lb   $t3, 3($a0)
add  $t0, $t0, $t1
add  $t0, $t0, $t2
add  $t0, $t0, $t3
sw   $t0, 0($a1)
```

$a3

```
int  A[4];
A[0] 0($a3)
A[1] 4($a3)
A[2] 8($a3)
A[3] 12($a3)
```