MIPS Assembly:
Practice Questions for Midterm 1
Saving to and Loading from Memory

CS 64: Computer Organization and Design Logic
Lecture #6

Ziad Matni
Dept. of Computer Science, UCSB

MIDTERM IS COMING!

- **Tuesday, 4/24** in this classroom
- **Starts at 11:00 AM **SHARP****
  - Please start arriving 5-10 minutes before class
- **I may ask you to change seats**
- Please bring your UCSB IDs with you

- **Closed book**: no calculators, no phones, no computers
- ***Only*** the MIPS Reference Card is allowed
- **You will write your answers on the exam sheet itself.**

What's on the Midterm?? 1/2

- Data Representation
  - Convert bin ←→ hex ←→ decimal ←→ bin
  - Signed and unsigned binaries
- Logic and Arithmetic
  - Binary addition, subtraction
    - Carry and Overflow
  - Bitwise AND, OR, NOT, XOR
  - General rules of AND, OR, XOR, using NOR as NOT
- All demos done in class
- Lab assignments 1 and 2

What's on the Midterm?? 2/2

Assembly

- Core components of a CPU
  - How instructions work
- Registers ($t, $s, $a, $v)
- Arithmetic in assembly (add, subtract, multiply, divide)
  - What's the difference between add, addi, addu, addui, etc…
- Conditionals and loops in assembly
- Conversion to and from Assembly and C/C++
- syscall and its various uses (printing output, taking input, ending program)
- **.data** and **.text** declarations
- Memory in MIPS
- Big Endian vs Little Endian

Review Questions

- I will put up review questions for Midterm #1 up on the class website
  - Later on today…

About the Midterm Exam

- Made up of Multiple Choice & Short Answers/Coding.

EXAMPLES:

Complete the following MIPS assembly code that is supposed to add the number in register $t0 to 15:

```
li $t0, 12
_____________

A. add $t2, $t1, $t2
B. addu $t2, $t1, $t2
C. addi $t2, $t0, F
D. addi $t2, $t0, 0xF
E. addui $t2, $t0, 0xF
```

What is the 2's complement of 0x5EC?

```
A. 1x5EC
B. 0x5EC
C. 0xA13
D. 0xA14
E. 0xA15
```

Sample Questions

Translate this C-style code into 4 lines of MIPS assembly code:

```
int t1=10, t2=3;
int t3=t1+2*t2
```

```
li $t1, 10
li $t2, 3
sll $t2, $t2, 1
add $t3, $t2, $t1
```

What is the result of these operations?

0xF2 & ~(0x55) **0xA2**

0x102A99D8 ^ 0xABA11CAB

```
  0001 0000 0010 1010 1001 1001 1101 1000
^ 1010 1011 1010 0001 0001 1100 1010 1011
= 1011 1011 1000 1011 1000 0101 0111 0011
= 0xBB8B8573
```

Sample Questions

Translate this MIPS assembly code into pseudo-code (C or C++ accepted):

```
li $s0, 2
li $s1, 6
li $t0, 2
add $s2, $s1, $s0
sll $s2, $s2, 3
mult $s2, $t0
mflo $s3
```

```
int s0=2, s1=6, t0=2;
int s2 = s1 + s0;
s2 *= 8;
int s3 = s2 * t0;
```

Sample Questions

Translate this C++ code into MIPS (just the .text part)

```
int t0 = 3;
if (t0 < 7)
   t1 = 7 + t0;
else
  t1 = t0 + t0;
```

```
.text
main:
    li $t0, 3
    li $t2, 7
    slt $t3, $t0, $t2
    beq $t3, $zero, else
    add $t1, t0, $t2;
    j end
else:
    add $t1, $t0, $t0
end:
    li $v0, 10
    syscall
```

Any Questions From Last Lecture?

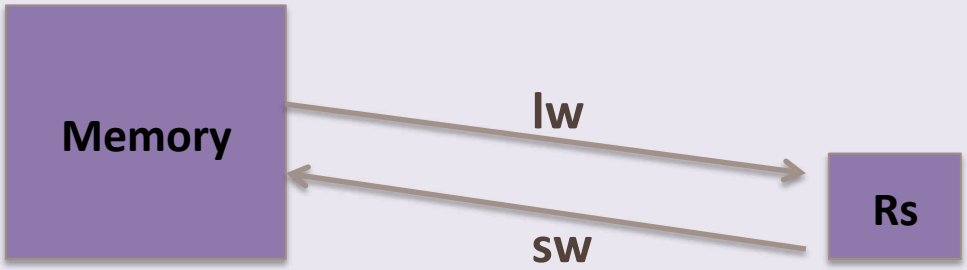
Larger Data Structures

- Recall: registers vs. memory
  - Where would data structures, arrays, etc. go?
  - Which is faster to access? Why?

- Some data structures have to be stored in memory
  - So we need instructions that "shuttle" data to/ from the CPU and computer memory (RAM)

Accessing Memory

- Two base instructions:
  - load-word (**lw**) from memory to registers
  - store-word (**sw**) from registers to memory

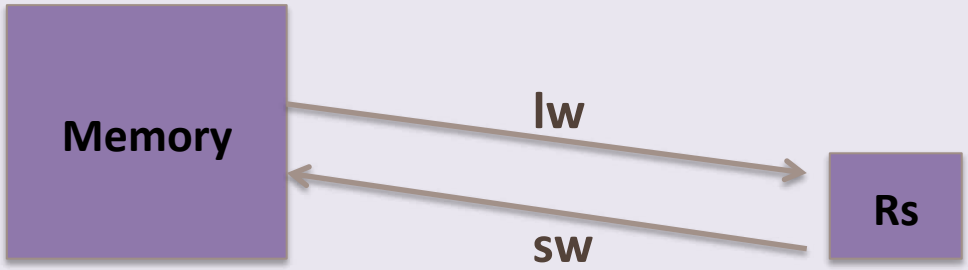


- MIPS lacks instructions that do more with memory than access it (e.g., retrieve something from memory and then add)
  - Operations are done step-by-step
  - Mark of RISC architecture

```
.data
num1: .word 42
num2: .word 7
num3: .space 1

.text
main:
    lw $t0, num1
    lw $t1, num2
    add $t2, $t0, $t1
    sw $t2, num3

    li $v0, 1
    lw $a0, num3
    syscall

    li $v0, 10
    syscall
```

Example 4

What does this do?



Addressing Memory

- If you're not using the **.data** declarations, then you need *starting addresses* of the data in memory with ***lw*** and ***sw*** instructions

Example: `lw $t0, 0x0000400A` (← not a real address)

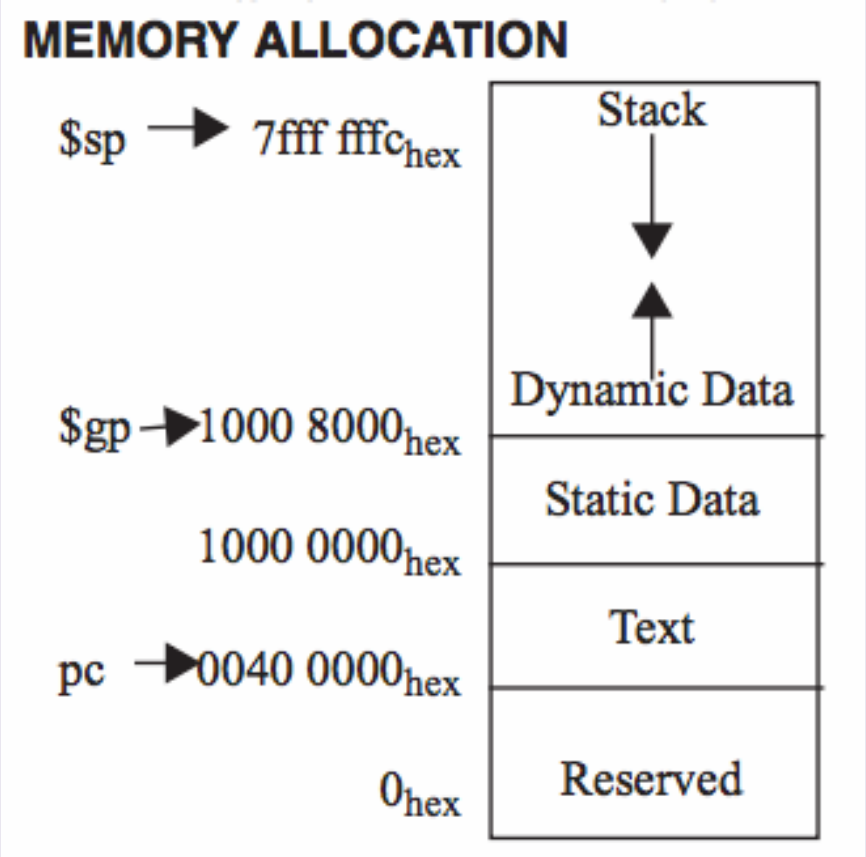
Example: `lw $t0, 0x0000400A($s0)` (← not a real address)

- 1 word = 32 bits (in MIPS)
  - So, in a 32-bit unit of memory, that's 4 bytes
  - Represented with 8 hexadecimals 8 x 4 bits = 32 bits... checks out...

- MIPS addresses sequential memory addresses, but not in "words"
  - Addresses are in Bytes instead
  - MIPS words *must* start at addresses that are multiples of 4
  - Called an ***alignment restriction***

Memory Allocation Map

How much memory does a programmer get to directly use in MIPS?



This is found on your ***MIPS Reference Card***

NOTE:

Not all memory addresses can be accessed by the programmer.

Although the address space is 32 bits, the top addresses from **0x80000000** to **0xFFFFFFFF** are not available to user programs. They are used mostly by the OS.

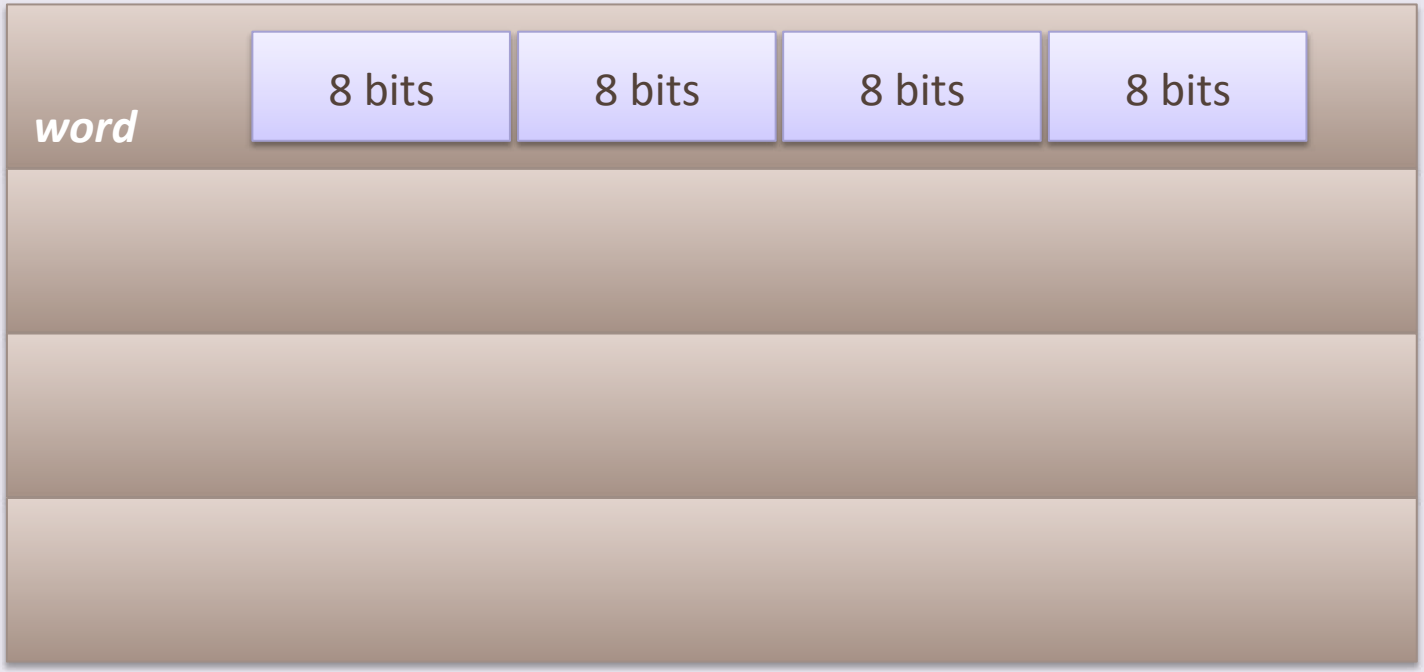
Mapping MIPS Memory

(say that 10 times fast!)

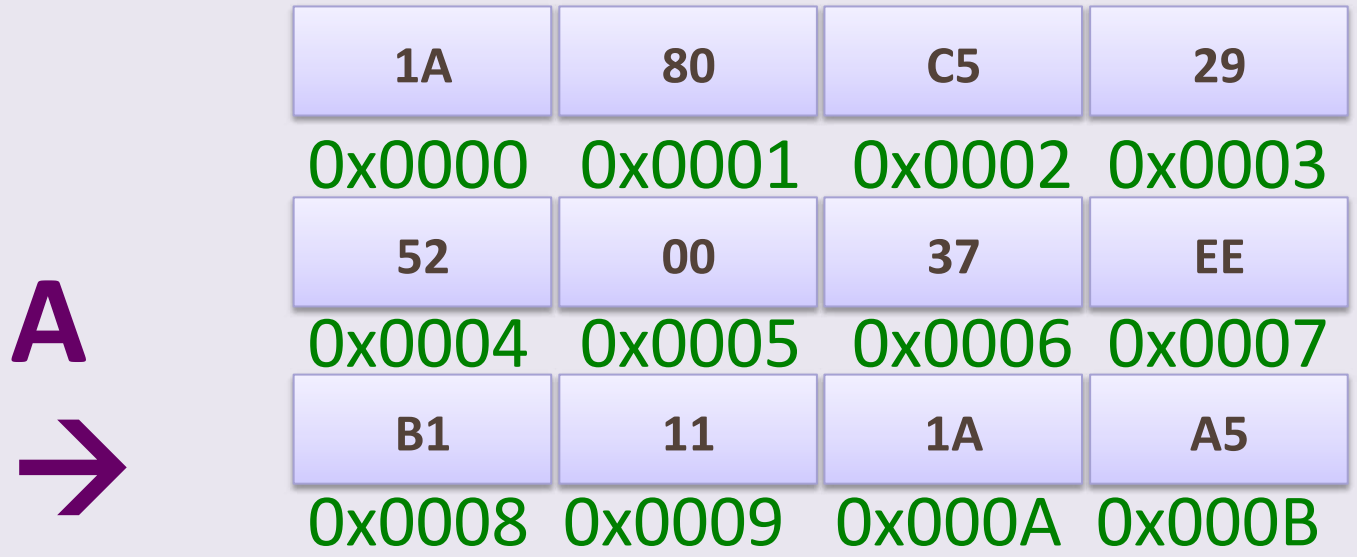
- Imagine computer memory like a big array of words
- Size of computer memory is:

  2^{32} = 4 Gbits, or 512 MBytes (MB)

  - We only get to use 2 Gbits, or 256 MB
  - That's (256 MB/ groups of 4 B) = 64 million words

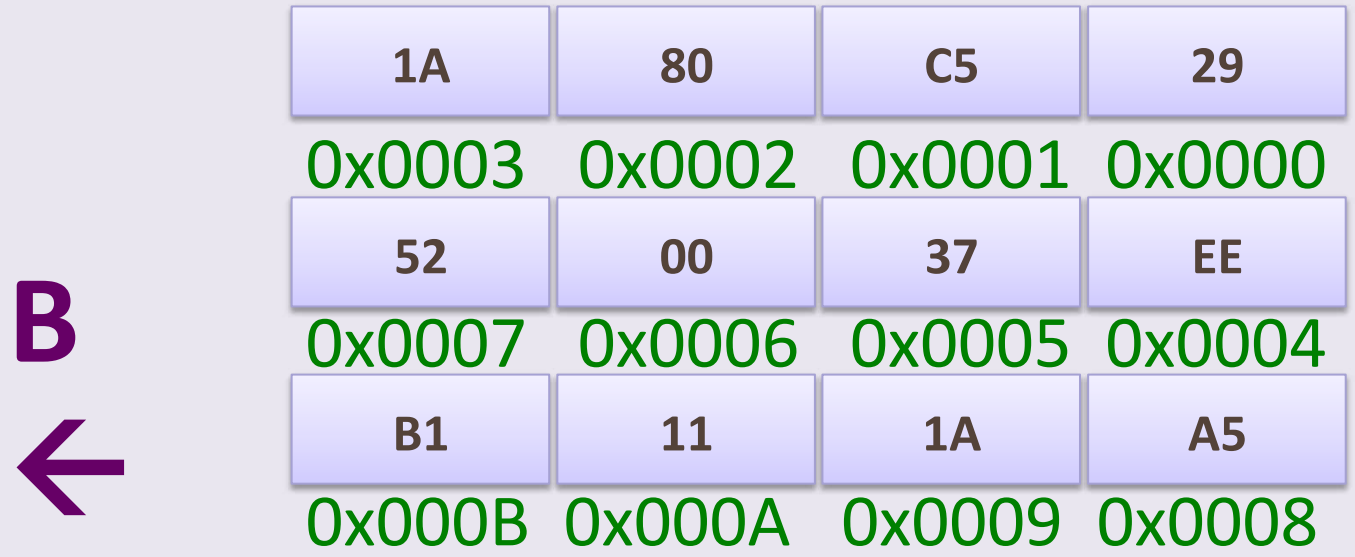


MIPS Computer Memory Addressing Conventions



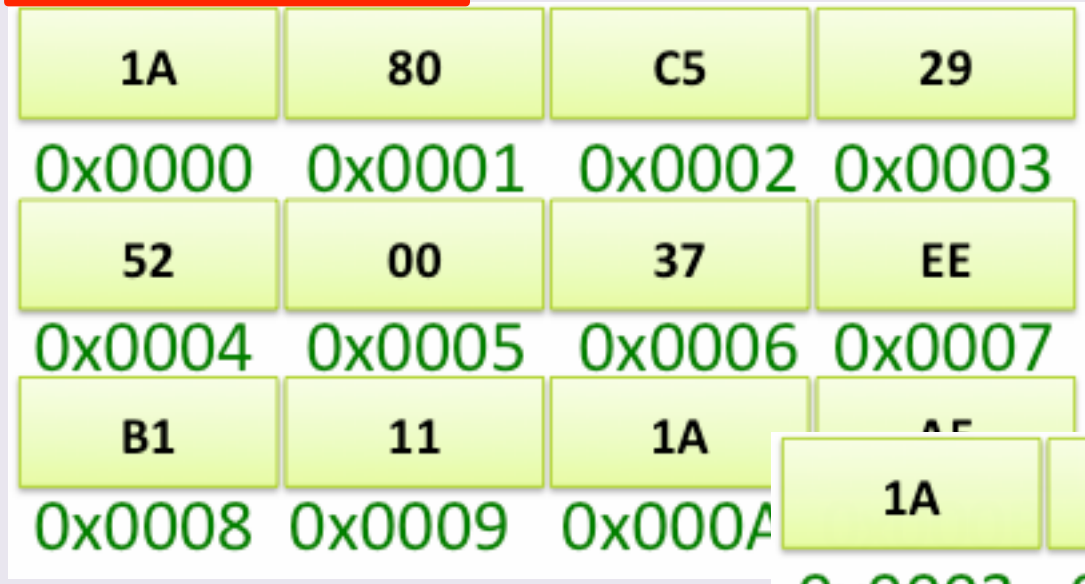
MIPS Computer Memory Addressing Conventions

or...



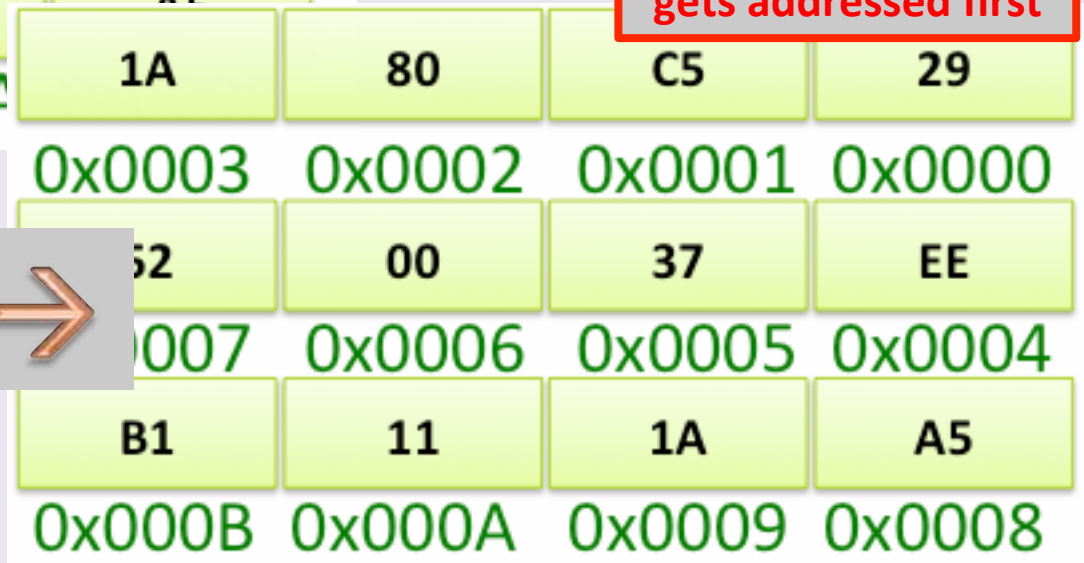
A Tale of 2 Conventions…

BIG END (MSByte) gets addressed first



← BIG ENDIAN

LITTLE END (LSByte) gets addressed first



LITTLE ENDIAN →

The Use of Big Endian vs. Little Endian

Origin: Jonathan Swift (author) in "Gulliver's Travels".

Some people preferred to eat their hard boiled eggs from the "little end" first (thus, little endians), while others prefer to eat from the "big end" (i.e. big endians).

- MIPS users typically go with Big Endian convention
  - MIPS allows you to program "endian-ness"
- Most Intel processors go with Little Endian…
- It's just a convention – it makes no difference to a CPU!

YOUR TO-DOs

- Finish assignment/Lab #2
  - Assignment due on FRIDAY
- Study for your midterm!! ☺
- Relax a little

</LECTURE>