LECTURE 16: MIPS MEMORY ADDRESSING

Computer Systems and Networks

Dr. Pallipuram
(vpallipuramkrishnamani@pacific.edu)

Some Deadlines

Lab 10 November 11th

Lab 11 November 19th

Class Today

Arrays and memory variables in MIPS

Coding: Lab 10

USE THIS CODE AS A STUB. Also on Lab 10 Page (a link)

```
        # Declare main as a global function
        # Pound is for comments
        .globl main
        # All program code is placed after the
        # .text assembler directive
        .text
# The label 'main' represents the starting point
main:
        #fill out main here

# Exit program by syscall
         li $v0, 10 # select exit syscall
         syscall # Exit the program
 Assembler directive .data
         .data
# Reserves space in memory for word with initial value 0
# used to store Z in memory
Z:  .word 0
```

Today’s MIPS

Declaring memory values and loading/storing them

Handling arrays in MIPS

Declaring Memory Values in MIPS

All of the memory values are declared in the `.data` section of the code

example (int Z = 12):

`Z:  .word 12` #to declare a 32-bit word & set to 12

example (int array[64] or char array[256]):

`array:  .space 256`#to create a space of 256 bytes, Can be 64 integers or 256 chars

example (char msg[] = “Hello world!”):

`msg: .asciiz “Hello world!”` #to create a string message

Memory Fundamentals

MIPS **<u>cannot</u>** directly manipulate data in memory!

Data must be moved to a register first! (And results must be saved to a register when finished)

This is a common design in *RISC-style* machines: a *load-store* architecture

Memory Fundamentals

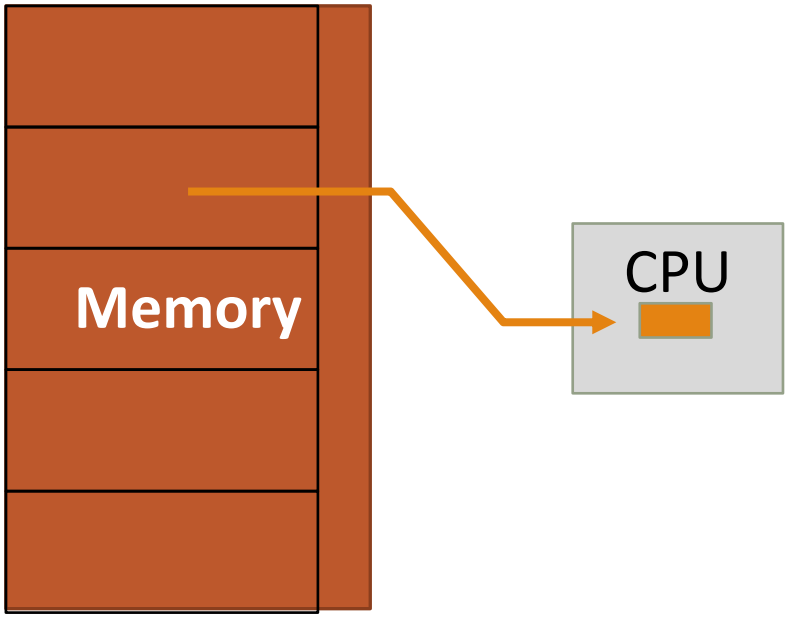
Yes, it's a **pain** to keep moving data between registers and memory.

But consider it your *motivation* to reduce the number of memory accesses. That will **improve program performance**!

Memory – Fundamental Operations

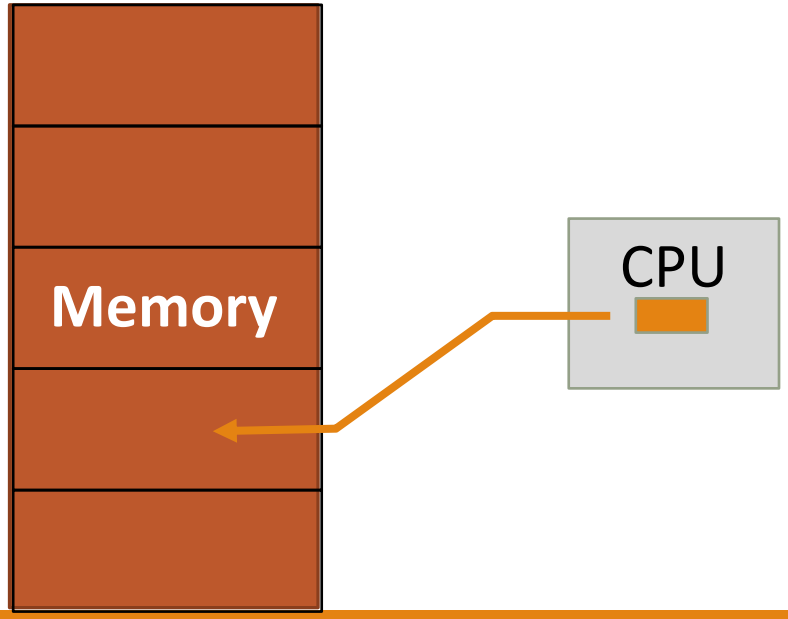
LOAD

Copy data from memory to register



STORE

Copy data from register to memory



Loading and Storing memory values

Load (Copy a value from memory variable to register):

```
lw <destination register>, memory_var
E.g.: lw $s0, A
```

Store (Copy a value from register to memory):

```
sw <source register>, memory_var
E.g.: sw $s0, C
```

Problem 1: A complete program

Declare memory variables, A and B, initialized to 20 and 45, respectively. Declare C and initialize it to 0. In main, set C to sum of A and B

```
.globl main
.text
main: #Main goes here


li $v0, 10 #v0 argument set to 10 for
#system call “exit”
syscall
.data #data goes under
```

Accessing Arrays

Array Recap

Name of the array is the address of the very first value.
E.g.:

```
int array[20];
printf("Address of the first element:%u",array);
```

Values are spaced by the size of the data. Integers are spaced by 4 bytes, doubles are spaced by 8 bytes, etc.

```
int array[20];
printf("Address of the first
element:%u",&array[0]); //say it prints 65530
printf("Address of the second
element:%u",&array[1]); //prints 65534
```

Accessing Arrays

Base offset addressing:

`A[5], array[i],` etc.

A:	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
address:	10	14	18	22	26	30	34	38	42	46

Base

offset=5

Pointer arithmetic:

pointer arithmetic done w.r.t data size

```
int array[10];
printf("\n array[5]:%u",*(array+5)); //adds 20 bytes to base address to access array[5]
```

C vs. MIPS

C has the following format:
`base[offset]`

C compiler multiplies the `offset`
with the size of the data to compute the correct offset in bytes

MIPS has the following format:
`offset(<register storing base addr.>)`

In MIPS, YOU multiply the offset with size of the data to compute the correct `offset` in bytes

MIPS – Base Offset Addressing

Load (Copy a value from memory to register):

```
lw <destination register>, <constant offset in
bytes>(<register that stores base address>)
E.g.:
lw $s0, 20($s1) #load $s0 with a value stored
#at an offset of 20 bytes from the base address in $s1
```

Store (Copy a value from register to memory):

```
sw <source register>, <constant offset in
bytes>(<register that stores base address>)
E.g.:
sw $s0, 20($s1) #store $s0 at an offset of 20 bytes
from base address in $s1
```

MIPS – Base Offset Addressing

Load **byte** (Copy a value from memory to register):

```
lb <destination register>, <constant offset in
bytes>(<register that stores base address>)
E.g.:
     lb $s0, 20($s1) #load an 8-bit value stored at
an offset of 20 bytes from base address in $s1
```

Store **byte** (Copy a value from register to memory):

```
sb <source register>, <constant offset in
bytes>(<register that stores base address>)
E.g.:
     sb $s0, 20($s1) #store 8-bit $s0 at an offset
of 20 bytes from base address in $s1
```

Problem 3 – Base Offset addressing

Write MIPS assembly for:

array[12] = h + array[8]

(Array of words. Assume h is in register)

Map:

$s2 = h

$s3 = base address of array

$t1 = temp

Problem 4 – Pointer Arithmetic

Write MIPS assembly for:

g = h + array[i]

(Array of words. Assume g, h, and i are in registers)

Map:

$s1 = g
$s2 = h
$s3 = base address of array
$s4 = i

How do I get the address of an array declared in `.data` section?

Load Address:

```
la <destination register to store the address>,
arrayname

E.g: la $s0, array #s0 stores the starting
address of the array
```

Problem 5 – Base-Offset and Pointer Arithmetic

```
//memory variable
int array[7];
int main()
{
     int i=0; //use register
     array[0]=5;
     for(i=1;i<7;i++)
       array[i] += array[i-1];
}
```

Read

MIPS example on I/O. See Lab 10 > MIPS Examples

MIPS_RandomGenerator.txt gives you solution for (`random_in_range()` and `get_random()`) in Lab 11. Carefully read and adapt it

find instructions for multiplication, division, and bit shifting: http://ecs-network.serv.pacific.edu/ecpe-170/tutorials/mips-instruction-set