lecture 10

MIPS assembly language 3

- arrays
- strings
- MIPS assembler directives and pseudoinstructions
- system calls (I/O)

February 10, 2016

Arrays in C

Example:

```
int   a[ 50 ];
  :
a[ 15 ]  =  a[ 7 ] ;
```

In C:

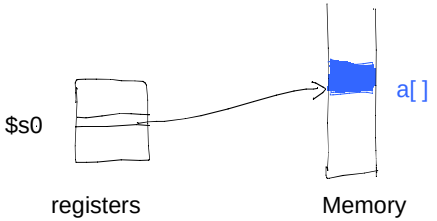
```
a[ 15 ]  =  a[ 7 ] ;
```

In MIPS ?

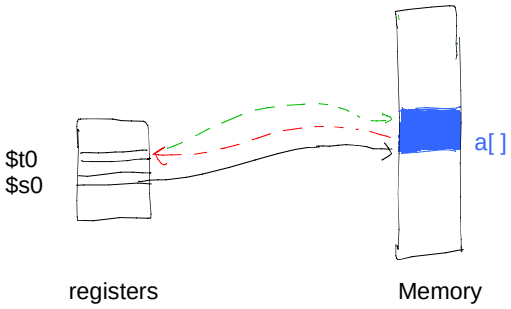
there are no "arrays" in MIPS

e.g. $s0 holds starting address of array a[ ] in Memory.

NOTE: You cannot transfer data directly between memory addresses




```
lw   $t0,   28( $s0 )        #  a [ 7 ]
sw   $t0,   60( $s0 )        #  a [ 15 ]
```



Another Example

```
m  =  a [ i ];        //   C instruction
```

$s1 $s0 $s2

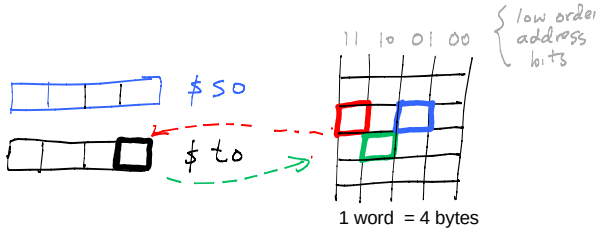
How to translate this into MIPS ?

```
sll     $t0,   $s2,   2          # offset = i * 4
add     $t0,   $s0,   $t0        # base + offset
lw      $s1,   0( $t0 )
```

How to manipulate single bytes in Memory ?

Recall "lw" and "sw". There is also a load byte "lb" and a store byte "sb" instruction.

```
lb   $t0,   2 ( $s0 )
sb   $t0,   -3 ( $s0 )
```

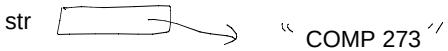


Strings in C (COMP 206)

- stored as consecutive bytes
  (essentially the same as an array of char)
- ASCII coded
- terminated with null char (0 in ASCII, we write '\0')

```
char    *str;          //  Declare a pointer to a string.
                       //  str is an address (a 32 bit number).
str   =   "COMP 273";
```

vague picture of what's going on....

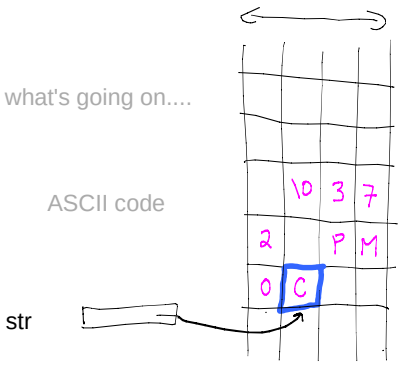



```
char  *str;       //  Declare a pointer to a string.
                  //  str is an address (a 32 bit number).
str =  "COMP 273";
```

1 word = 4 bytes

better picture of what's going on....

ASCII code



Count the number of chars in a string (C)

```
char   *str;          //  Declare a pointer to a string.
                      //  str is an address (a 32 bit number).

int      ct = 0;

str =   "COMP 273";


while  ( *(str + ct) != '\0' ){   //  coming soon in  COMP 206

     ct++;

}
```

Strings in MIPS

1 word = 4 bytes

a much better picture of what's going on....

C CODE

```
str =   "COMP 273";
ct  = 0;
while  ( *(str + ct) != '\0' ){
     ct++;
}
```

MIPS CODE ?

```
                # load the address where string begins
                # initialize ct to 0  (use a register)
loop:
                # compute address of Memory byte to examine next
                # load that byte into a register
                # if that byte is '\0',  branch to exit
                # increment ct
                # jump back to "loop"
exit:
```

C CODE

```
str =   "COMP 273";
while  ( *(str + ct) != '\0' ){
     ct++;
}
```

MIPS CODE

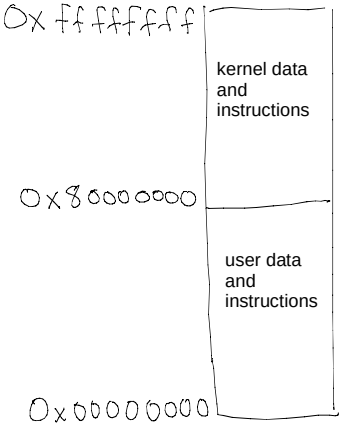
```
        la      $s0, str            # pseudoinstruction (load address)
                                    # I will explain this soon.
        add     $s1, $zero, $zero    # initialize ct,    $s1 = 0.
loop:
        add     $t0, $s0,   $s1      # address of byte to examine next
        lb      $t1, 0( $t0 )        # load that byte
        beq     $t1, $zero, exit     # branch if  *(s + ct) ==  '\0'
        addi    $s1, $s1, 1          # increment ct
        j       loop
exit:
```

Q: How to get data into and out of Memory ?

A: 1) "assembler directives"

2) "system calls"

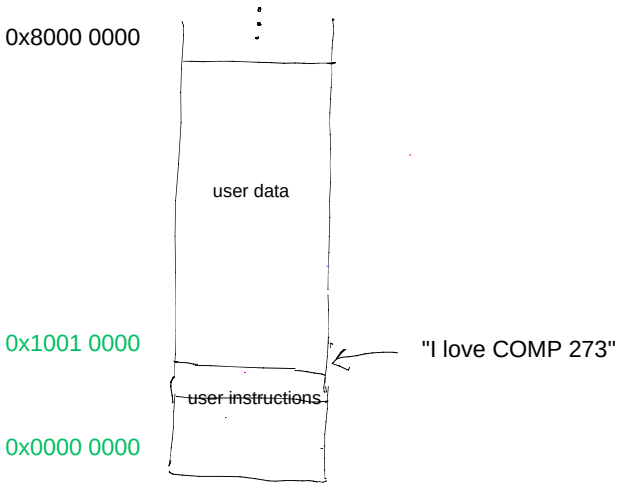
recall MIPS Memory



Assembler Directives (Example)

```
                .data

str     :       .asciiz    "I love COMP 273"

                .text
                .globl     main

main:
```

str is a label that aids in programming. Think of it as a label for an address (similar to the "Exit" labels that we saw in conditional branches earlier).



load address (la) pseudoinstruction

assembler

```
la      $s0,    str       # pseudoinstruction
lui     $s0,  4097        # true MIPS instruction
                          # load upper immediate
```

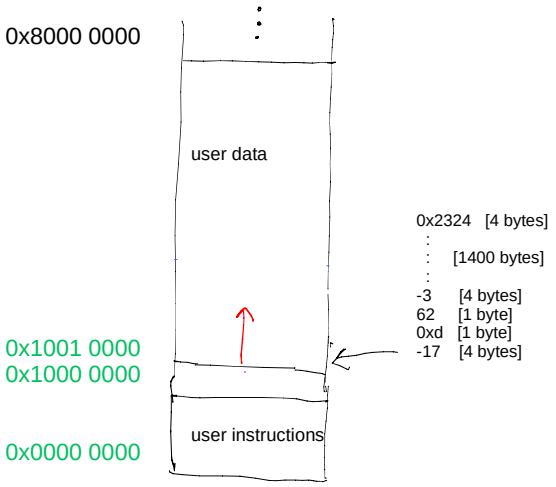
(4097)_10 = 2^12 + 2^0

= (0001000000000001)_2

= 0x1001

More Assembler Directives

```
y0    :    .word        -17

b0    :    .byte        0xd,   62,   -3    # signed
b1    :    .byte        250                 #  out of range

arr0  :    .space       1400

y1    :    .word        0x2c24
```



Example: swap

C code

```
tmp  = y0;
y0   = y1;
y1   = tmp;
```

MIPS code

This code assumes that the variables are already in registers.

```
move    $t0,  $s0
move    $s0,  $s1        #  "move" is a pseudoinstruction
move    $s1,  $t0        #
```

```
    .data
y0: .word  162        # value of y0
y1: .word  -17        # value of y1

    .text
    .globl main

main:

  #  Here the variables are NOT already in registers.

    la      $s0,  y0        #   load addresses of y0, y1
    la      $s1,  y1

    lw      $t0,  0( $s0 )    #   load contents into registers
    lw      $t1,  0( $s1 )


    sw      $t0,  0( $s1 )    #  store the swapped values to Memory
    sw      $t1,  0( $s0 )
```

```
    la      $s0,  y0        #   load addresses
    la      $s1,  y1
```

MARS

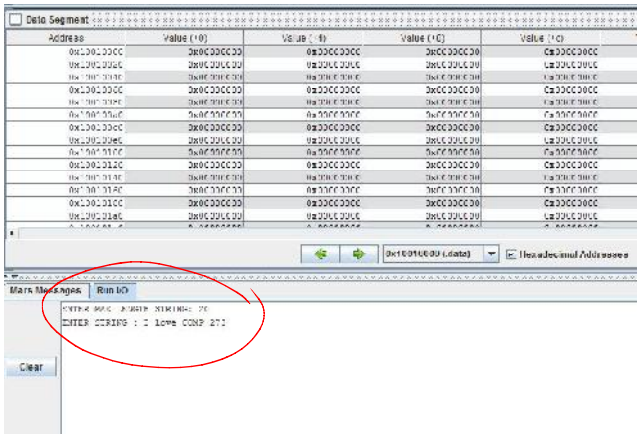
```
    lui     $s0,  0x1001    #   load addresses

    lui     $1,   0x1001
    ori     $s1,  $1,   4

  #   user not allowed to use  $1  register
```

Q: How to get data into and out of Memory ?

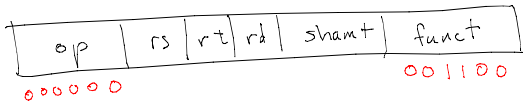
A: 1) "assembler directives"

2) "system calls"

System calls ("syscall" instruction) uses the console.



syscall



This instruction uses registers $2, $4, $5 which you can also write $v0 and $a0, $a1, respectively.

Example: print a string

```
la   $a0,  str
li   $v0,   4          #   li is a pseudoinstruction "load immediate"

#  ori   $v0,   $zero, 4     is   the real instruction

syscall
```

Example: read a string from console

```
li      $v0, 8                  #   code for reading string is 8

add     $a0, $zero, $s1         #   $s1 specifies address
                                #   where string will start

la    $t0, sizeBuffer     #  specifies a buffer size (see A2)
lw    $a1, 0($t0)         #  load that buffer size.
syscall
```

The OS/kernel stops the program and waits for a string to be typed into the console (hitting "enter" signals the end of the string, or max length is reached). The string is then written from the buffer into Memory starting at address specified by $s1. Only the string is written (not the whole buffer size). Then the program continues.

ASIDE: technical detail about reading a string from console

Every string must end with a "null terminator", namely 0x00 or '\0'.

If the user types in maxLenString - 1 characters, then the OS reads it and returns the program to running state. Any extra keystrokes are ignored.

e.g. suppose maximum length string (buffer size) is set to 4.

Typing "abc" (3 characters) will cause "abc\0" to be written into Memory.

Typing "a<enter>" will cause "a\n\0" to be written into Memory, where "\n" is C notation for 'line feed' or 'enter'.

Experiment with this yourself before plunging into Assignment 2.

Example syscall codes for $v0

	int	float	double	string
print	1	2	3	4
read	5	6	7	8
exit				

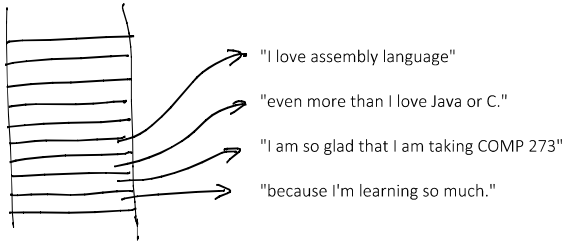
See documentation. Do not memorize this stuff...

Assignment 2 posted today

Task: manipulate an array of string references (addresses).

MIPS Memory

the strings below are also stored in Memory



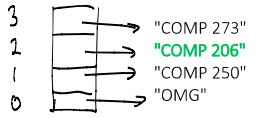
Assignment 2: two parts

1) read in a list of strings from the console (loop)
   - store the strings in Memory
   - store the addresses of the strings in an array in Memory (this array is a list)

2) manipulate the list of strings using "move to front"
   - user enters an index i, and the i-th string address is moved to the front

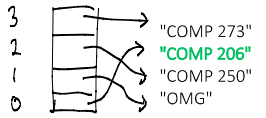
"Move to front"

BEFORE

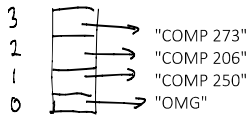


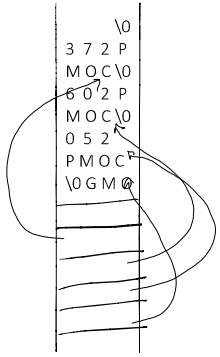
move to front: 2

AFTER



The addresses and strings are all in Memory.





ADDED Feb 21:
In the original slides, I had mistakenly put '\n' instead of both '\n\0' in the strings on the right. The strings in the figure now are missing the line feeds '\n' (see discussion in Q4).

[EDITED Feb 21] It is important to understand where your variables are in Memory. Note we use assembler directives to assign Memory for :

- maxLengthString (integer i.e. 1 word)
- stringReferenceArray (5 words)
- strings (100 bytes)
- prompts e.g. "enter maximum length of a string: "
  "enter a string:"
  "move to front index: "

The following slide shows how they are layed out, starting at address 0x10010000. Note in MARS the addresses increase to right and down (opposite from slides).

[ADDED Feb 21]