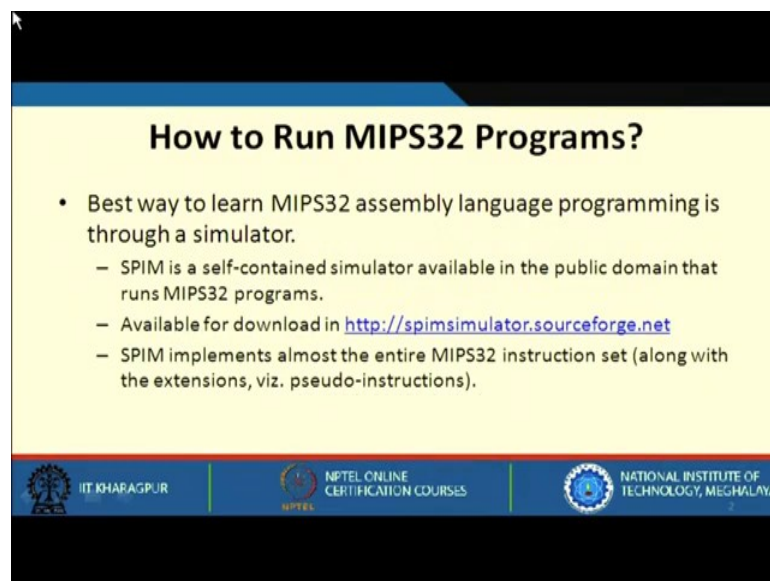


**Computer Architecture and Organization**  
**Prof. Kamalika Datta**  
**Department of Computer Science and Engineering**  
**National Institute of Technology, Meghalaya**

**Lecture – 11**  
**SPIM – A MIPS32 Simulator**


Welcome to lecture 11. In this lecture I will be talking about SPIM; a MIPS32 simulator; how you can write programs in SPIM.

(Refer Slide Time: 00:39)



**How to Run MIPS32 Programs?**


- Best way to learn MIPS32 assembly language programming is through a simulator.
  - SPIM is a self-contained simulator available in the public domain that runs MIPS32 programs.
  - Available for download in <http://spimsimulator.sourceforge.net>
  - SPIM implements almost the entire MIPS32 instruction set (along with the extensions, viz. pseudo-instructions).



As you all know the best way to learn any assembly language is through a simulator and we should start coding. So, how to start that; we need a simulator which we will be using (SPIM) that is a self contained simulator and it is available in public domain and you can download that from this particular web page. SPIM implements almost the entire MIPS32 instruction set along with the extension with pseudo instructions. What do you mean by pseudo instructions? I already discussed about pseudo instructions like in MIPS.

(Refer Slide Time: 02:09)


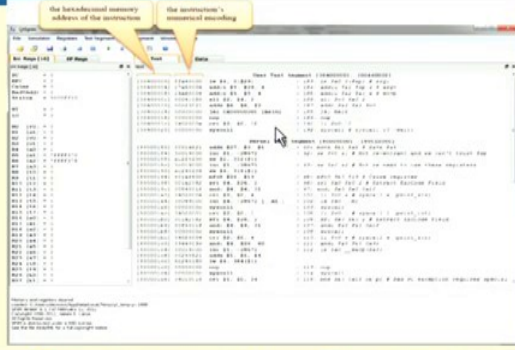
- SPIM has both terminal-based and window-based interfaces.
  - Terminal versions are available on Linux, Windows, and Mac OS X.
  - Window-based interface is provided by the QTSPIM program, which is also available on Linux, Windows and Mac OS X.
- *SPIM is copyrighted by James Larus and distributed under a BSD license.*
- What can SPIM do?
  - It can read and execute assembly language programs for MIPS32.
  - Provides a simple debugger.
  - Provides minimal set of OS services via system calls.



SPIM has both terminal based and window based interfaces which are available; it is up to you that which you will be using. SPIM is copyrighted by James Larus and distributed under a BSD license. So, we must know what SPIM can do --- it can read and execute assembly language programs for MIPS32 and provide a simple debugger and also provide minimal set of OS services via system calls. This is a screen shot of a QTSPIM.

(Refer Slide Time: 03:09)

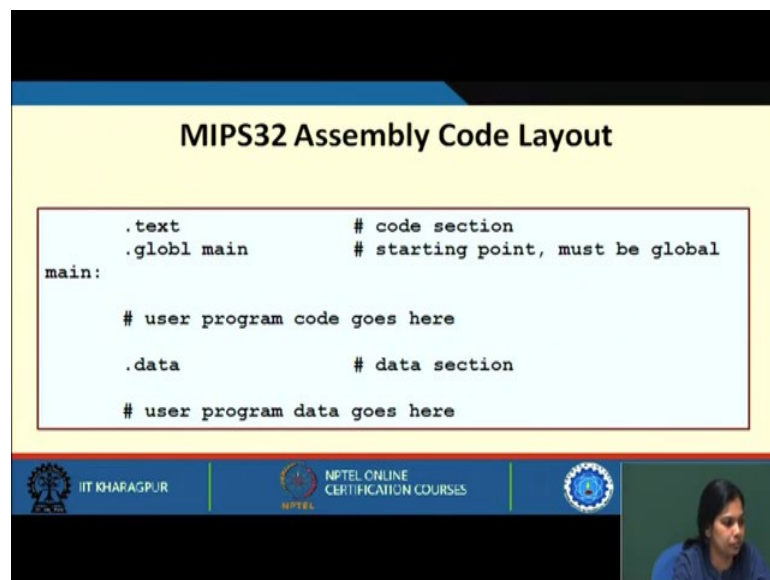
**Screenshot of QTSPIM**



So, if you install QtSPIM in your machine, you will be able to see a view like this. Here these are the registers that are having different values, and these are the hexadecimal addresses, and this is the instruction after encoding.

So, these are the list of instructions and this is the encoded form of the instructions. We have already learnt how we can encode a particular instruction knowing how many bits of registers are present, etc. So, this is basically the main instructions that are getting executed, this is the encoded form of the instructions, and this is the memory addresses and you can see the memory addresses will be incremented by 4, as we have 32-bit instructions. It is byte addressable, and so it will be added plus 4.

(Refer Slide Time: 04:25)



The slide displays the MIPS32 Assembly Code Layout. It features a yellow background with a blue header and footer. The main content is a light blue box containing assembly code. The code defines a text section for user program code and a data section for user program data. The text section starts with a global label 'main' and includes a comment indicating that the user program code goes here. The data section includes a comment indicating that the user program data goes here. The footer contains logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NPTEL, along with a small video inset of a person.

```
.text          # code section
.globl main    # starting point, must be global
main:

# user program code goes here

.data         # data section

# user program data goes here
```

Now, let us see the MIPS32 assembly code layout; it has got a text section, it has got a data section. In this section we have something called **globl main**. This actually shows the starting point that must be global, and now this is .global main, and main is the label from where your program execution starts. If you do not give this in the program it will be wrong. So, you have to mention that your main is global, and this starts from a particular level (that is main) and then you can write the user program code here.

(Refer Slide Time: 05:32)

**Assembler Directives**

- a) **.text**
  - Specifies the user text segment, which contains the instructions.
- b) **.data**
  - Specifies the data segment, where all the data items are defined.
- c) **.globl sym**
  - Specifies that the symbol "sym" is global, and can be referred from other files.
- d) **.word w1, w2, ..., wn**
  - Stores the specified 32-bit numbers in successive memory words.
- e) **.half h1, h2, ..., hn**
  - Stores the specified 16-bit numbers in successive memory half-words.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And the data portion is written in this part. These are some assembler directives --- .text specifies the user text segment which contains the instructions that are required to execute. We will see some examples in course of time. Then, .data specifies the data segment where we put all the data that will be used in our program. Then .globl sym specifies the starting point, but this symbol is global and can be referred from other files as well. So, from many other files you will be able to access this.

Next is .word --- it stores the specified 32-bit numbers in successive memory words.

(Refer Slide Time: 06:51)

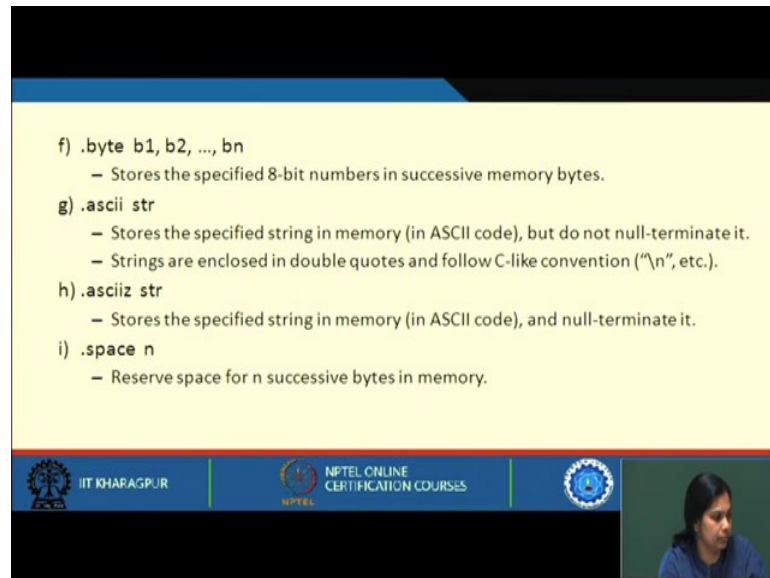
© CET I.I.T. KGP

```
0x 00000000 +4 30
0x 00000004 +4 31
0x 00000008 32
.
.
```

A hand is visible holding a pen, pointing to the code.

Let us say hexadecimal address starts from 0 0 0 0 0 0; your next address will be 0 0 0 0 0 4, and so on. So, it is increasing as plus 4 plus 4 and so on.

(Refer Slide Time: 08:00)



f) `.byte b1, b2, ..., bn`  
– Stores the specified 8-bit numbers in successive memory bytes.

g) `.ascii str`  
– Stores the specified string in memory (in ASCII code), but do not null-terminate it.  
– Strings are enclosed in double quotes and follow C-like convention (“\n”, etc.).

h) `.asciiz str`  
– Stores the specified string in memory (in ASCII code), and null-terminate it.

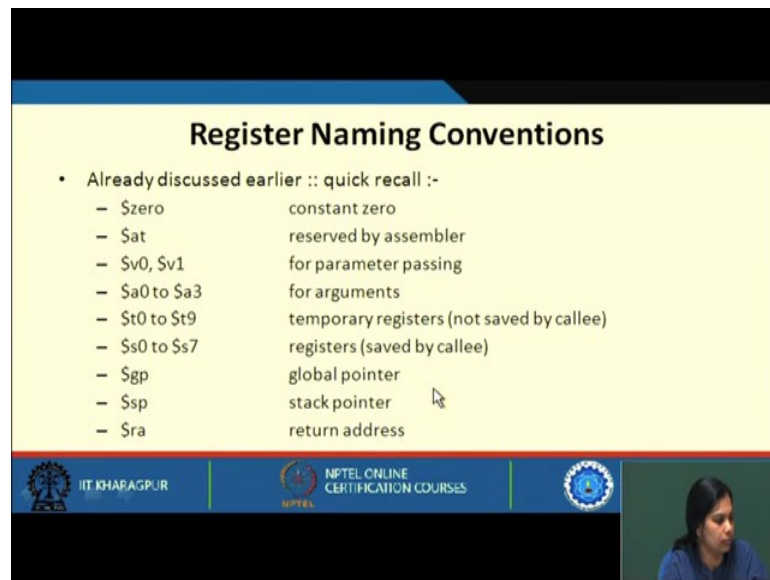
i) `.space n`  
– Reserve space for n successive bytes in memory.

The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video inset of a woman.

`.byte` stores a 8-bit specified number in successive memory bytes. We can also specify characters. So, `.ascii` and `.asciiz` are the two directives that are used to store the specified string in memory, but do not null terminate it. So, it is not null terminated --- a string is null terminated by “\0”. `.asciiz str` specifies a string in memory and it is also null terminated.

`.space n` reserves a space for n successive bytes of memory.

(Refer Slide Time: 09:09)



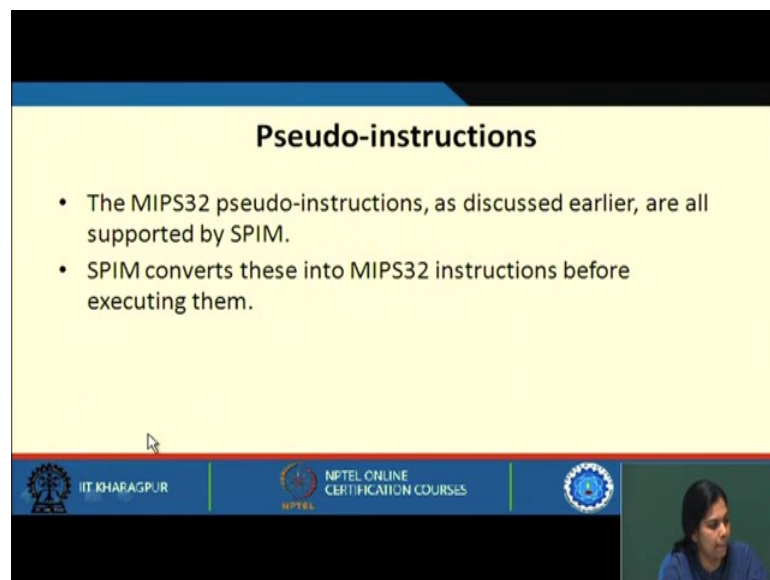
**Register Naming Conventions**

- Already discussed earlier :: quick recall :-
  - \$zero constant zero
  - \$at reserved by assembler
  - \$v0, \$v1 for parameter passing
  - \$a0 to \$a3 for arguments
  - \$t0 to \$t9 temporary registers (not saved by callee)
  - \$s0 to \$s7 registers (saved by callee)
  - \$gp global pointer
  - \$sp stack pointer
  - \$ra return address

The slide footer includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a woman.

For register naming convention we already discussed about all these things, and this is how it is named: \$zero, \$at, \$v0, \$v1, \$a0 to \$a3, \$t0 to \$t9, \$s0 to \$s7, \$gp, \$sp and \$ra.

(Refer Slide Time: 09:26)



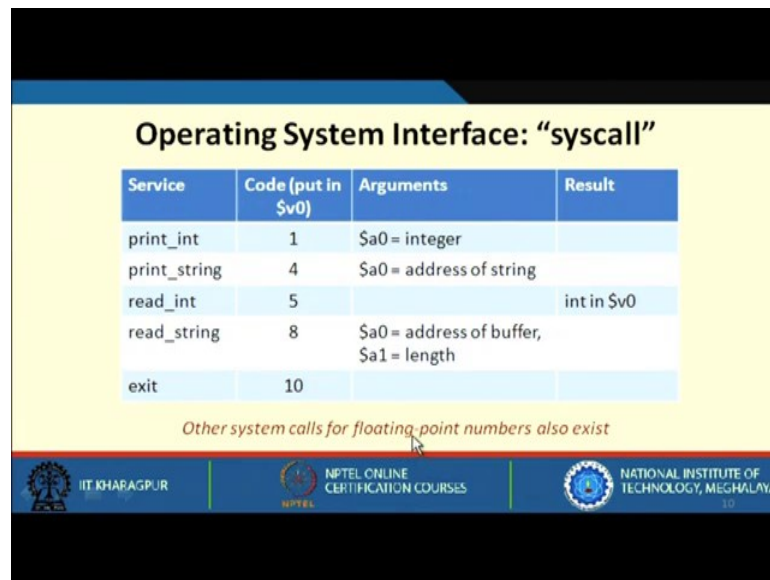
**Pseudo-instructions**

- The MIPS32 pseudo-instructions, as discussed earlier, are all supported by SPIM.
- SPIM converts these into MIPS32 instructions before executing them.

The slide footer includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a woman.

As you know MIPS pseudo instructions as discussed earlier are all supported by SPIM. SPIM converts these into MIPS32 instructions before executing them. So, we can write a program using pseudo instruction, but this pseudo instruction in turn gets converted into MIPS32 instruction before it gets executed.

(Refer Slide Time: 10:00)



**Operating System Interface: "syscall"**

Service	Code (put in \$v0)	Arguments	Result
print_int	1	\$a0 = integer	
print_string	4	\$a0 = address of string	
read_int	5		int in \$v0
read_string	8	\$a0 = address of buffer, \$a1 = length	
exit	10		

*Other system calls for floating-point numbers also exist*

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

For operating system interface syscall is used, like when you want to get a value from keyboard, display in the monitor, etc. In such cases you need to use syscall. So, if you want to print an integer you have to put the argument in \$a0 and you have to put the code \$v0 = 1 to print an integer. Similarly for printing a string you need to put the code 4 in \$v0 and the address of the string in \$a0. And then you perform a syscall then it will print that mean on the screen this value will get displayed, either integer or a string. Similarly we reading an integer it will read from the keyboard and store it in \$v0.

Similarly, for read string the code is 8, and in \$a0 the address of the buffer where you want to read it that will be given and how much length string you want to read that is given in \$a1. Also the code for exit is 10 and after putting the required value in these registers you have to do syscall. Other system calls for floating point numbers also exists which is not included here. Let us take a sample program.



(Refer Slide Time: 12:46)


**Example Program 1**

```
.text
.globl main

main: la    $t0, value
      lw    $t1, 0($t0)
      lw    $t2, 4($t0)
      add   $t3, $t1, $t2
      sw    $t3, 8($t0)

.data
value: .word 50, 30, 0
```

Add two numbers in memory and store the result in the next location.

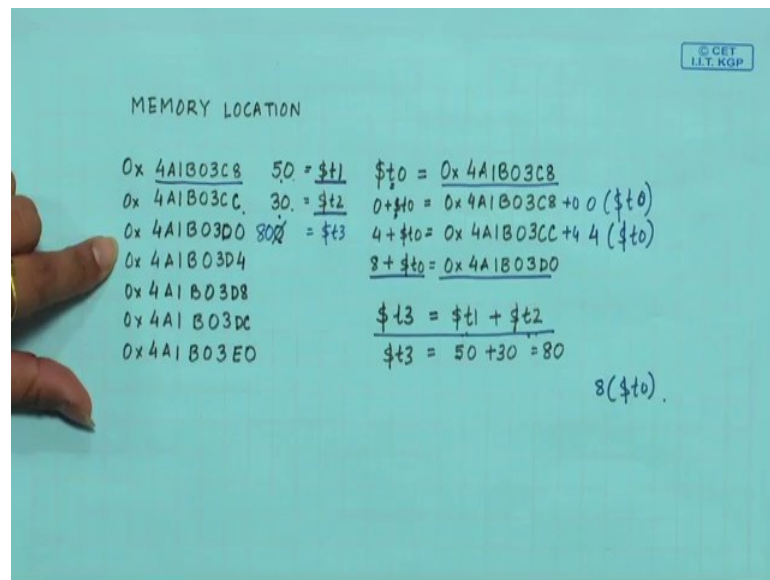


Let us see how this program is executed. So, in the .text we write .globl main and main is the label from where the execution of program starts. So, what this program is actually doing let us also look into the value; value is also a label which consists of some word and each word is of 32 bit and what it stores in the first location; it stores 50 and in the next location it stores 30, and the result will be stored in next location which is initialized with 0.

Now, just see this --- la meaning load address from value and store it in \$t0. \$t0 is a temporary register where we are loading this value. Once we load the address of this value in \$t0 we need to load the word 50 and 30. So, let us take an example to show this.



(Refer Slide Time: 14:27)



Now, you see these are my memory location 4A1B03C8 --- in this memory location we have stored 50, in the next memory location we have stored 30 depending on the value, and in this location we have stored 0. The various steps are shown.

(Refer Slide Time: 18:42)

### Example Program 2

```
.text
.globl main

main: add $t1, $zero, 0x2A
      add $t2, $zero, 0x0D
      add $s3, $t1, $t2
```

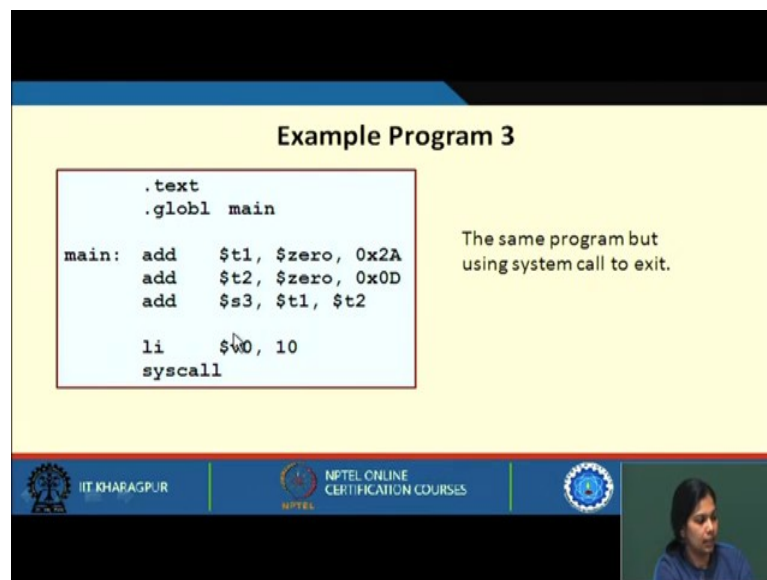
Add two constant numbers specified as immediate data, and store the result in a register.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, this is a program that adds two constant numbers specified as immediate data and store the result in a register. So, here we are adding some immediate values 0x2A and 0x0D. What we are doing we are adding this immediate value with \$zero and then we are

storing it in \$t1; why because we do not have a move instruction. So, instead we are doing directly this here and then we are again loading this particular data into \$t2 and then we are adding these two and storing it in register \$s3. So, these 3 steps will add two immediate value that is 0x2A and 0x0D into \$s3.

(Refer Slide Time: 19:46)



The slide is titled "Example Program 3" and features a light yellow background. On the left, there is a code block with assembly instructions. On the right, there is a text description. At the bottom, there is a blue footer bar with logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NPTEL, along with a small video inset of a woman.

```
.text
.globl main

main: add    $t1, $zero, 0x2A
      add    $t2, $zero, 0x0D
      add    $s3, $t1, $t2

      li    $v0, 10
      syscall
```

The same program but using system call to exit.

Coming to the next program that is the same program, but using a system call. In this system call if you recall if we load 10 in \$v0 and then we do a syscall then it will exit. So, this particular program is same as the previous one that is adding two immediate values, but at the end we have to load an immediate number into \$v0. So, in \$v0 value 10 will be loaded and once we do system call then it will get exit. So, it will exit from here.

(Refer Slide Time: 20:31)


### Example Program 4


Read two numbers from the keyboard and print the sum.


```
.data
str1: .asciiz "Enter first number: "
str2: .asciiz "Enter second number: "
str3: .asciiz "The sum is = "

.text
.globl main
main: li $v0, 4           # print string
      la $a0, str1
      syscall

      li $v0, 5           # read integer
      syscall
      move $t0, $v0
```

  
IIT KHARAGPUR

  
NPTEL ONLINE  
CERTIFICATION COURSES

  
NATIONAL INSTITUTE OF  
TECHNOLOGY, MEGHALAYA

Now, let us see a program. Here we read 2 numbers from the keyboard and print the sum. So, first of all we have to read 2 numbers. So, initially in the data section we initialize the three strings, now see all these numbers are entered from the keyboard. So, in the data section this str1 specifies “Enter the first number:”, str2 specifies “Enter the second number:”, and str3 specifies “The sum is =”.

So, first we have to display this “Enter the first number:” and then once it is displayed we need to read a value from the keyboard, let us see how we can do this. The process is repeated for the other number also.

(Refer Slide Time: 24:29)

```
li    $v0, 4
la    $a0, str2
syscall

li    $v0, 5
syscall
move  $t1, $v0

add   $t1, $t0, $t1
      # $t1= $t0 +
      $t1

li    $v0, 4
la    $a0, str3
syscall
```

```
li    $v0, 1
move  $a0, $t1
syscall

li    $v0, 10
syscall
```

The slide features a yellow background with two white boxes containing assembly code. The left box contains code for reading two integers, performing an addition, and displaying the result. The right box contains code for reading one integer and displaying it. The footer includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA, along with a small video inset of a woman.

So, this is a program that reads 2 integers from the keyboard perform addition and store back the result in another and displays the result in the keyboard itself.

(Refer Slide Time: 26:35)

**Example Program 5**

Calculate sum of 10 32-bit numbers stored in consecutive memory locations.

```
.data
num: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
.text
.globl main

main:
la    $t0, num
li    $t2, 0           # holds the sum
li    $t3, 0           # counter for loop
loop: lw  $t1, 0($t0)
      add $t2, $t2, $t1
      addi $t3, $t3, 1
      addi $t0, $t0, 4 # point to next
      bne $t3, 10, loop

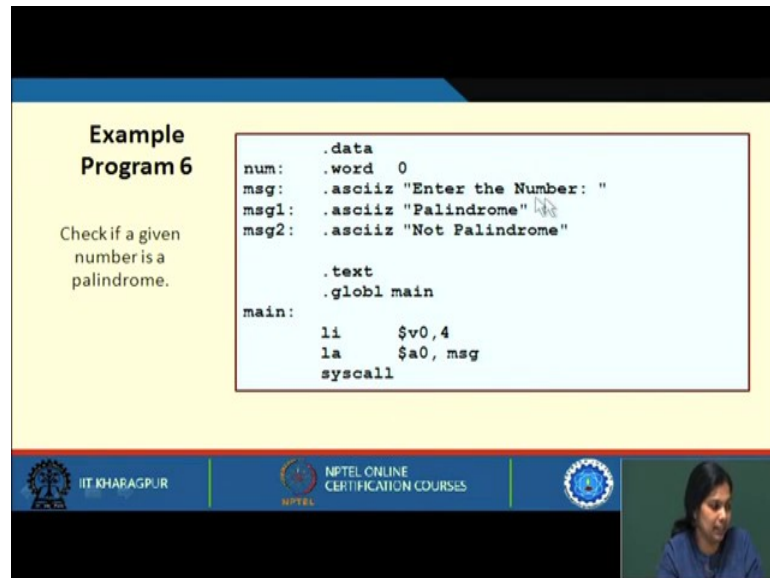
li    $v0, 10
syscall
```

The slide features a yellow background with a white box containing assembly code. The code defines an array of 10 numbers, initializes a sum register and a counter, and then enters a loop to calculate the sum. The footer includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA, along with a small video inset of a woman.

Now, let us see how we can do some other programs like adding 10 numbers. So, till now we are talking about simple program loading a value adding those values storing it back, but loops and other things are very common in programming. So, we often encounter loops everywhere whenever we write a program how loops will be executed using this SPIM how we can write loops using SPIM we will be seeing in now. So, we

will see how we can calculate sum of 10 numbers and the 10 numbers are stored in consecutive memory locations, and these 10 numbers are will be added and stores back in another memory location. So, we need to have one counter that will start from here if it starts from 0 it will go till 9; 0 1 2 3 4 5 6 7 8 9. So, less than 10 if you start from 1 it will be less than equal to 10.

(Refer Slide Time: 32:25)



**Example Program 6**

Check if a given number is a palindrome.

```
.data
num: .word 0
msg: .asciiz "Enter the Number: "
msg1: .asciiz "Palindrome"
msg2: .asciiz "Not Palindrome"

.text
.globl main
main:
li    $v0, 4
la    $a0, msg
syscall
```

The slide features a yellow background with a blue header and footer. The footer contains logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a woman in the bottom right corner.

The next example is another program which checks if a given number is palindrome or not. So, what is a palindrome? A palindrome is a number if you reverse the number it will be the same.

(Refer Slide Time: 32:41)

**PALINDROME**

$$212 = 2 \times 10^2 + 1 \times 10^1 + 2 \times 10^0 = 2 + 10 + 200 = 212$$

**CONSIDER**  $123$

$$123 \% 10 = 3$$
$$123 / 10 = 12$$
$$12 \% 10 = 2$$
$$12 / 10 = 1$$
$$1 \% 10 = 1$$
$$1 / 10 = 0$$

**COUNTER**  
Initially Counter = 0

$$0 \times 10 + 3 = 3$$
$$3 \times 10 + 2 = 32$$
$$32 \times 10 + 1 = 321$$

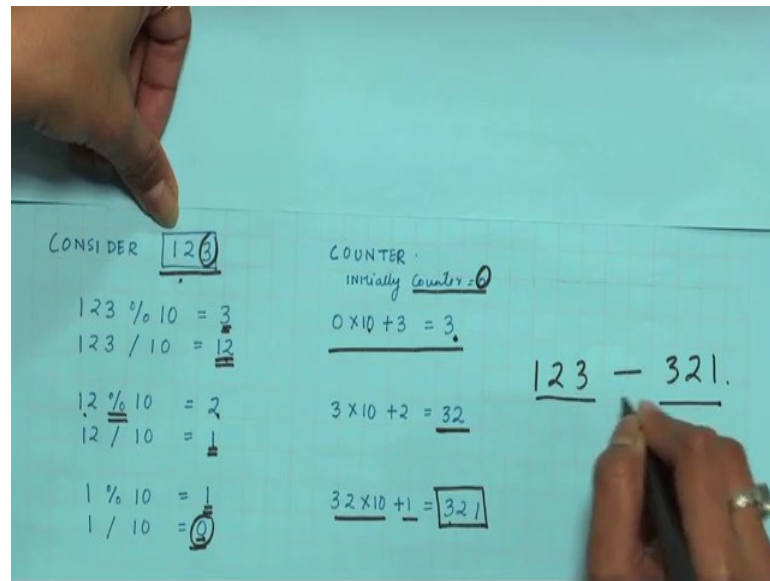
**212 = 212**

**123 = 321**

So, if you have a number 2 1 2 if you reverse the number it will be first 2 will come, then 1 will come, then 2 will come. So, both the numbers are same. Now let us have a number 1 2 3 is this palindrome number? You reverse the number 3 2 and 1 this is not a palindrome number because this is not same as this.

Let us take an example 1 2 3; basically what we need to do we need to extract the last digit. So, we need to multiply it with 10 and add it with the extracted digit and we store it whatever we get again we multiply with 10 and add it with the last digit that is the remainder.

(Refer Slide Time: 34:18)



So firstly, let me explain the concept if we consider a number 1 2 3, we take the remainder of the number. So, the remainder will be 3. So, we have extracted the last digit. So,  $123 \bmod 10$  we get 3 which is the remainder. So, initially you have to take a counter that is 0. So, we multiply 0 with 10 and we add with the remainder. So, we get 3 here. So, this is the 0th one. So, initially the counter was 0 it has multiplied with 10 and added with 3 now my counter is 3 and what I do again we need to get the quotient. So, 123 divided by 10 we get 12 as the quotient.

Now, we take this 12 again we take the remainder we get 2; once you get the remainder as 2 the previous counter value was 3 we multiply 3 with 10 and we add it with 2 we get 32 now my counter has changed to 32; now how many times you will be doing it till you get a 0 quotient. So, again for the next time the quotient is 1. So,  $1 \bmod 10$  it becomes one initial counter was 32 then we have to do 32 into 10 and I added one I add with one and we get 321, now 1 divided by 10 which is 0. So, we will not continue. So, we got the value which is the reverse of this 1 2 3. So, initially we have taken 1 2 3, we repeated certain steps we kept a counter in that counter we were storing the reverse value and finally, we got the value 3 2 1. So, 1 2 3 the reverse is 3 2 1, but now this is not equal to this. So, it is not a palindrome.



Let us see how we can code this using QTSPIM. So, here similarly we have to enter a number and this should be two messages; if it is a palindrome it should display palindrome, if it is not palindrome it will display not palindrome.

(Refer Slide Time: 37:18)

```
li    $v0, 5
syscall
move  $t0, $v0
move  $t3, $t0
li    $t2, 0

loop:
mul   $t2, $t2, 10
rem   $t1, $t0, 10
div   $t0, $t0, 10
add   $t2, $t2, $t1
bne   $t0, $zero, loop
bne   $t3, $t2, np

np:
li    $v0, 4
la    $a0, msg1
syscall

li    $v0, 10
syscall

li    $v0, 4
la    $a0, msg2
syscall

li    $v0, 10
syscall
```

Now first of all this message “Enter the number:” should come. So, we load 4 in \$v0 and the message in \$a0 and do a syscall then we enter the number. So, we load 5 in \$v0 we do a syscall the value the number is entered is stored in \$v0 which is moved to \$t0 we also move the value \$t0 to \$t3. So, \$t3 and \$t0 both contain my number for which I have to check whether it is a palindrome or not; now I am loading an immediate value \$t2, where in \$t2 I am initializing it with 0. So, the same process that I have explained I am doing it in a loop. So, what I am doing initially the counter value is 0. So, 0 is multiplied with 10 and it is stored in \$t2; then my number is in \$t0 the number for which I have to calculate the palindrome is in \$t0, I divide it with 10 and I get the remainder in \$t1. So, my remainder is now in \$t1, I divide it \$t0 by 10, I get the quotient in \$t0. So, \$t1 contains the remainder and \$t0 contains the quotient.

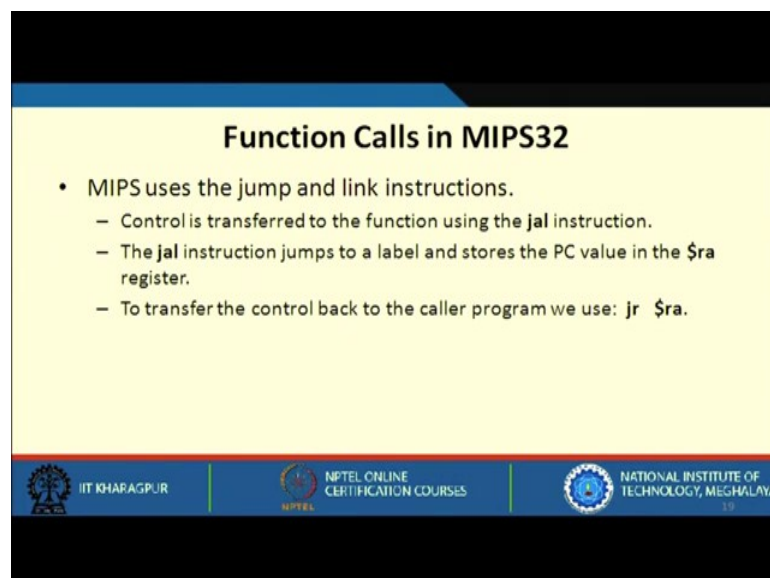
Now, I am adding the counter \$t1 with the counter value that is \$t2 and I am storing back in \$t2, next what I am checking whether \$t0 is equal to 0 or not; that means, my quotient has become 0 or not; if my quotient has become 0 then I will not loop it. So, branch if not equal if \$t0 is not equal to 0; again you go to the loop. Now you are taking the updated value of \$t2 which is depending on the last digit; that last digit is multiplied

by 10 and stored in \$t2 again similarly for that value \$t1 again it for that value \$t0 we will be dividing it with 10 we get the remainder we divide with 10 we get the quotient and we again add it.

So, we keep on doing this until we get this \$t0 is 0; that means, until the quotient become 0. Once the quotient becomes 0, we check the next thing what we are checking see in the first step we have also stored the number in \$t3 why we have stored the number and \$t3 because we have to finally check because the number is palindrome or not; we first reverse that number and now I am checking whether the number is palindrome or not.

So, \$t3 is checked with \$t2 that is the counter that I have kept if it is not equal then you go to a label np that is not palindrome and where you load this message that this particular number is not palindrome, but if it is not equal then only we are going in let us see if it is equal then this statement will not get executed and we will directly come to this statement where we will display the message the number is palindrome. So, just see what we have done and then we exit it. So, we have put one exit here one exit here why because if it is not palindrome it will come here and then, the exist code it will encounter here, but if it is palindrome then it will go here and then it will also exist with this particular code.

(Refer Slide Time: 41:49)



**Function Calls in MIPS32**

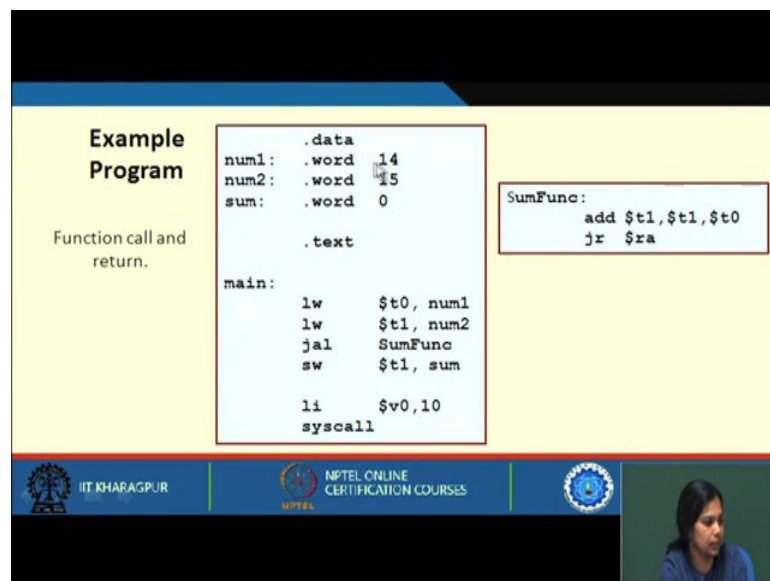
- MIPS uses the jump and link instructions.
  - Control is transferred to the function using the `jal` instruction.
  - The `jal` instruction jumps to a label and stores the PC value in the `$ra` register.
  - To transfer the control back to the caller program we use: `jr $ra`.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, this code actually shows whether a number is palindrome or not. So, we have written an assembly language code which shows a number is palindrome or not. So, these are

function calls in MIPS it uses jump and link instruction; we call it jal instruction and what it does it jumps to a label and it stores the PC value in the return address register \$31. And once the control after the subroutine is executed it has to return back to that particular place. So, returning back to that place it has to do jr to return automatically to that particular address; it will load the previous value of pc which was stored and it will go on executing it.

(Refer Slide Time: 42:34)



The slide displays MIPS assembly code for an example program. It is divided into three sections: data, text, and a function definition. The data section defines variables num1 (14), num2 (15), and sum (0). The text section shows the main function loading num1 into \$t0 and num2 into \$t1, calling SumFunc with jal, and storing the result in sum. The SumFunc function adds \$t0 and \$t1 into \$t1 and returns to the caller using jr \$ra. The slide also includes logos for IIT Kharagpur and NPTEL Online Certification Courses, and a small video inset of a presenter.

```
.data
num1: .word 14
num2: .word 15
sum: .word 0

.text
main:
    lw    $t0, num1
    lw    $t1, num2
    jal   SumFunc
    sw    $t1, sum

    li    $v0, 10
    syscall

SumFunc:
    add   $t1, $t1, $t0
    jr    $ra
```

So, this is a simple example of function call. So, 2 numbers are store in these 2 locations. So, we load these 2 numbers here load word from num1 and num2 in these 2 registers \$t0 and \$t1 and then we are doing a jal, where we are going to SumFunc. In SumFunc we are adding \$t1 and \$t0 and storing in \$t1 and what we are writing here after this execution of this we are writing jump to return address that is \$ra. So, it will load that address and it will come here because the pc value initially when you are accessed this pc value, then the pc value was incremented to the next one which was pointing to “sw \$t1,sum” --- but now you have executed a jal instruction where it has it has moved here it will execute this particular instruction and then it will jump to return address and it will get the return address from register 31 which is \$ra and it will execute this statement now starting from this statement.

Now, the pc value will be loaded with this again after the execution of jr \$ra and then these 2 instructions will get executed. So, this is how function call happens in MIPS. So,

now, we have come to end of module 2. In module 2 we have seen generally what is the kind of instruction format, addressing modes that are there, and then specifically we have gone into the details of MIPS32 instruction set and we have discussed about a simulator that is SPIM where we can write programs in assembly language.

Thank you.