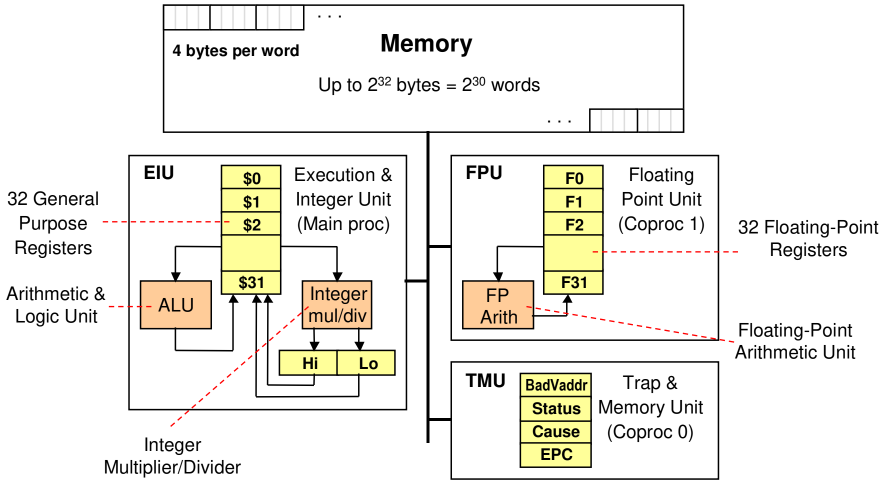
EEE 343 Computer Organization

Adeel Israr

Computer Organization Lab

- ❖ Lab 1 to Lab 4
  - ✧ MIPS Assembly Language Programming
- ❖ Lab 5 to Lab 12
  - ✧ Verilog implementation of MIPS Processor

Overview of the MIPS Architecture


Memory
4 bytes per word
Up to 2^{32} bytes = 2^{30} words
. . .
. . .
EIU
$0
$1
$2
$31
Execution & Integer Unit (Main proc)
32 General Purpose Registers
Arithmetic & Logic Unit
ALU
Integer mul/div
Hi
Lo
Integer Multiplier/Divider
FPU
F0
F1
F2
F31
Floating Point Unit (Coproc 1)
32 Floating-Point Registers
FP Arith
Floating-Point Arithmetic Unit
TMU
BadVaddr
Status
Cause
EPC
Trap & Memory Unit (Coproc 0)

MIPS General-Purpose Registers

- ❖ 32 General Purpose Registers (GPRs)
  - ✧ Assembler uses the dollar notation to name registers
    - $0 is register 0, $1 is register 1, …, and $31 is register 31
  - ✧ All registers are 32-bit wide in MIPS32
  - ✧ Register $0 is always zero
    - Any value written to $0 is discarded
- ❖ Software conventions
  - ✧ There are many registers (32)
  - ✧ Software defines names to all registers
    - To standardize their use in programs
  - ✧ Example: $8 - $15 are called $t0 - $t7
    - Used for temporary values

$0 = $zero	$16 = $s0
$1 = $at	$17 = $s1
$2 = $v0	$18 = $s2
$3 = $v1	$19 = $s3
$4 = $a0	$20 = $s4
$5 = $a1	$21 = $s5
$6 = $a2	$22 = $s6
$7 = $a3	$23 = $s7
$8 = $t0	$24 = $t8
$9 = $t1	$25 = $t9
$10 = $t2	$26 = $k0
$11 = $t3	$27 = $k1
$12 = $t4	$28 = $gp
$13 = $t5	$29 = $sp
$14 = $t6	$30 = $fp
$15 = $t7	$31 = $ra

MIPS Register Conventions

- ❖ Assembler can refer to registers by name or by number
  - ✧ It is easier for you to remember registers by name
  - ✧ Assembler converts register name to its corresponding number

Name	Register	Usage
$zero	$0	Always 0 (forced by hardware)
$at	$1	Reserved for assembler use
$v0 – $v1	$2 – $3	Result values of a function
$a0 – $a3	$4 – $7	Arguments of a function
$t0 – $t7	$8 – $15	Temporary Values
$s0 – $s7	$16 – $23	Saved registers (preserved across call)
$t8 – $t9	$24 – $25	More temporaries
$k0 – $k1	$26 – $27	Reserved for OS kernel
$gp	$28	Global pointer (points to global data)
$sp	$29	Stack pointer (points to top of stack)
$fp	$30	Frame pointer (points to stack frame)
$ra	$31	Return address (used by jal for function call)

MIPS Instruction Set

- ❖ R-Type
  - ✧ ADD $1, $2, $3
  - ✧ SUB $1, $2, $3
  - ✧ AND $1, $2, $3
  - ✧ OR $1, $2, $3
  - ✧ XOR $1, $2, $3
  - ✧ NOR $1, $2, $3
  - ✧ SLT $1, $2, $3
- ❖ Memory Instruction
  - ✧ LW $1, offset($2)
  - ✧ SW $1, offset($2)

MIPS Instructions (Cont)

- Jump Instructions
  - J Label
  - JR Label
  - JAL Label
- Immediate Instructions
  - ADDI $1,$2, Constant
- Pseudo Instructions
  - LI $1, Constant
  - Move $1,$2
  - MUL $1,$2,$3

System Calls

- Programs do input/output through system calls
- MIPS provides a special **`syscall`** instruction
  - To obtain services from the operating system
  - Many services are provided in the SPIM and MARS simulators
- Using the **`syscall`** system services
  - Load the service number in register $v0
  - Load argument values, if any, in registers $a0, $a1, etc.
  - Issue the **`syscall`** instruction
  - Retrieve return values, if any, from result registers

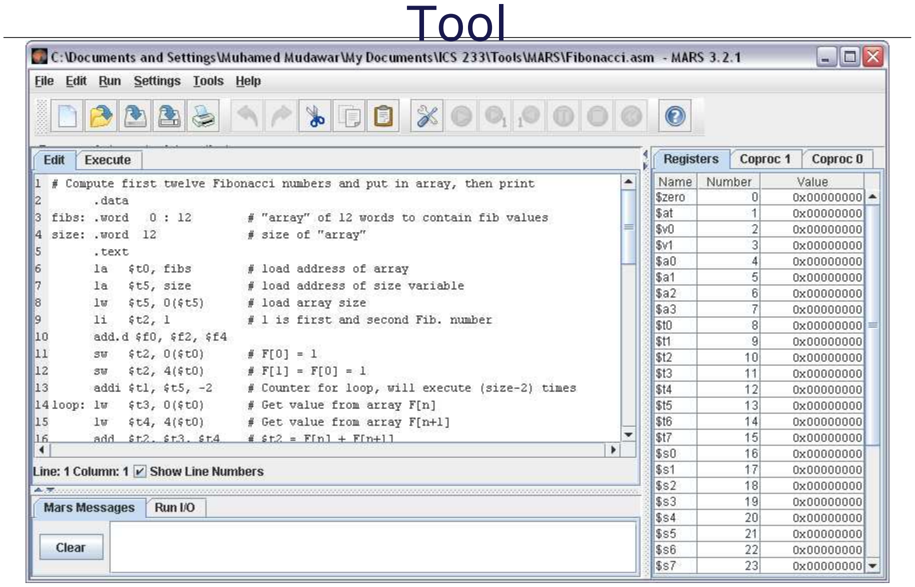
Syscall Services

Service	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print Float	2	$f12 = float value to print
Print Double	3	$f12 = double value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	Return integer value in $v0
Read Float	6	Return float value in $f0
Read Double	7	Return double value in $f0
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read
Allocate Heap memory	9	$a0 = number of bytes to allocate Return address of allocated memory in $v0
Exit Program	10	

Syscall Services – Cont'd

Print Char	11	$a0 = character to print
Read Char	12	Return character read in $v0
Open File	13	$a0 = address of null-terminated filename string $a1 = flags (0 = read-only, 1 = write-only) $a2 = mode (ignored) Return file descriptor in $v0 (negative if error)
Read from File	14	$a0 = File descriptor $a1 = address of input buffer $a2 = maximum number of characters to read Return number of characters read in $v0
Write to File	15	$a0 = File descriptor $a1 = address of buffer $a2 = number of characters to write Return number of characters written in $v0
Close File	16	$a0 = File descriptor

MARS Assembler and Simulator Tool



C:\Documents and Settings\Muhamed Mudawar\My Documents\ICS 233\Tools\MARS\Fibonacci.asm - MARS 3.2.1
File Edit Run Settings Tools Help
Edit Execute
Line: 1 Column: 1 ☑ Show Line Numbers
Mars Messages Run I/O
Clear
Registers Coproc 1 Coproc 0

```
# Compute first twelve Fibonacci numbers and put in array, then print
      .data
fibs: .word   0 : 12        # "array" of 12 words to contain fib values
size: .word  12             # size of "array"
      .text
      la   $t0, fibs        # load address of array
      la   $t5, size        # load address of size variable
      lw   $t5, 0($t5)      # load array size
      li   $t2, 1           # 1 is first and second Fib. number
      add.d $f0, $f2, $f4
      sw   $t2, 0($t0)      # F[0] = 1
      sw   $t2, 4($t0)      # F[1] = F[0] = 1
      addi $t1, $t5, -2     # Counter for loop, will execute (size-2) times
loop: lw   $t3, 0($t0)      # Get value from array F[n]
      lw   $t4, 4($t0)      # Get value from array F[n+1]
      add  $t2, $t3, $t4    # $t2 = F[n] + F[n+1]
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000

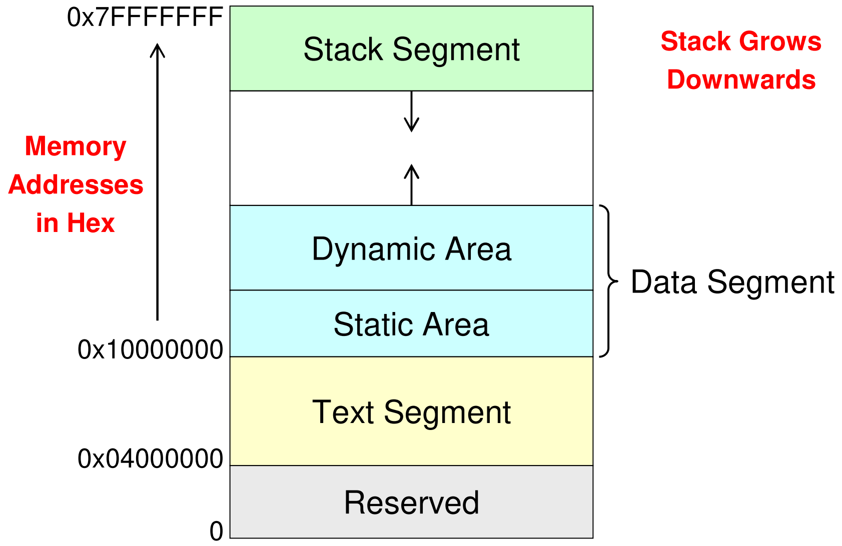
Program Template

```
# Title:                          Filename:
# Author:                         Date:
# Description:
# Input:
# Output:
################# Data segment #####################
.data
 . . .
################# Code segment #####################
.text
.globl main
main:                             # main program entry
 . . .
li $v0, 10                        # Exit program
syscall
```

.DATA, .TEXT, & .GLOBL Directives

- **.DATA** directive
  - Defines the data segment of a program containing data
  - The program's variables should be defined under this directive
  - Assembler will allocate and initialize the storage of variables
- **.TEXT** directive
  - Defines the code segment of a program containing instructions
- **.GLOBL** directive
  - Declares a symbol as global
  - Global symbols can be referenced from other files
  - We use this directive to declare *main* procedure of a program

Layout of a Program in Memory

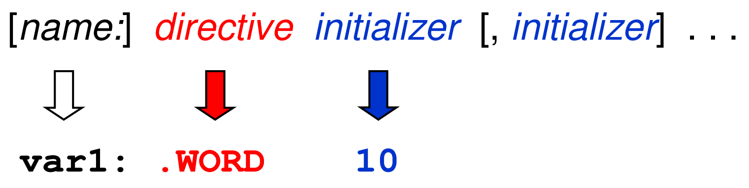


Next . . .

- ❖ Assembly Language Statements
- ❖ Assembly Language Program Template
- ❖ Defining Data
- ❖ Memory Alignment and Byte Ordering
- ❖ System Calls
- ❖ Procedures
- ❖ Parameter Passing and the Runtime Stack

Data Definition Statement

- ❖ Sets aside storage in memory for a variable
- ❖ May optionally assign a name (label) to the data
- ❖ Syntax:



[*name:*] *directive* *initializer* [, *initializer*] . . .

```
var1: .WORD   10
```

- ❖ All initializers become binary data in memory

Data Directives

- **.BYTE** Directive
  - Stores the list of values as 8-bit bytes
- **.HALF** Directive
  - Stores the list as 16-bit values aligned on half-word boundary
- **.WORD** Directive
  - Stores the list as 32-bit values aligned on a word boundary
- **.FLOAT** Directive
  - Stores the listed values as single-precision floating point
- **.DOUBLE** Directive
  - Stores the listed values as double-precision floating point

String Directives

- **.ASCII** Directive
  - Allocates a sequence of bytes for an ASCII string
- **.ASCIIZ** Directive
  - Same as **.ASCII** directive, but adds a NULL char at end of string
  - Strings are null-terminated, as in the C programming language
- **.SPACE** Directive
  - Allocates space of *n* uninitialized bytes in the data segment

Examples of Data Definitions

```
.DATA
var1:   .BYTE     'A', 'E', 127, -1, '\n'
var2:   .HALF     -10, 0xffff
var3:   .WORD     0x12345678:100
var4:   .FLOAT    12.3, -0.1
var5:   .DOUBLE   1.5e-10
str1:   .ASCII    "A String\n"
str2:   .ASCIIZ   "NULL Terminated String"
array:  .SPACE    100
```

Array of 100 words

100 bytes (not initialized)

Program 1: Sum of Three Integers

```
# Sum of three integers
#
# Objective: Computes the sum of three integers.
#     Input: Requests three numbers.
#    Output: Outputs the sum.
################### Data segment ###################
.data
prompt:   .asciiz     "Please enter three numbers: \n"
sum_msg:  .asciiz     "The sum is: "
################### Code segment ###################
.text
.globl main
main:
      la    $a0,prompt          # display prompt string
      li    $v0,4
      syscall
      li    $v0,5               # read 1st integer into $t0
      syscall
      move  $t0,$v0
```

Sum of Three Integers – Slide 2 of 2

```
      li    $v0,5                # read 2nd integer into $t1
      syscall
      move  $t1,$v0

      li    $v0,5                # read 3rd integer into $t2
      syscall
      move  $t2,$v0

      addu  $t0,$t0,$t1          # accumulate the sum
      addu  $t0,$t0,$t2

      la    $a0,sum_msg          # write sum message
      li    $v0,4
      syscall

      move  $a0,$t0              # output sum
      li    $v0,1
      syscall

      li    $v0,10               # exit
      syscall
```

Program 2: Case Conversion

```
# Objective: Convert lowercase letters to uppercase
#     Input: Requests a character string from the user.
#    Output: Prints the input string in uppercase.
################### Data segment #####################
.data
name_prompt:  .asciiz         "Please type your name: "
out_msg:      .asciiz         "Your name in capitals is: "
in_name:      .space 31       # space for input string
################### Code segment #####################
.text
.globl main
main:
      la    $a0,name_prompt   # print prompt string
      li    $v0,4
      syscall
      la    $a0,in_name       # read the input string
      li    $a1,31            # at most 30 chars + 1 null char
      li    $v0,8
      syscall
```

Case Conversion – Slide 2 of 2

```
      la    $a0,out_msg      # write output message
      li    $v0,4
      syscall
      la    $t0,in_name
loop:
      lb    $t1,($t0)
      beqz  $t1,exit_loop    # if NULL, we are done
      blt   $t1,'a',no_change
      bgt   $t1,'z',no_change
      addiu $t1,$t1,-32      # convert to uppercase: 'A'-'a'=-32
      sb    $t1,($t0)
no_change:
      addiu $t0,$t0,1        # increment pointer
      j     loop
exit_loop:
      la    $a0,in_name      # output converted string
      li    $v0,4
      syscall
      li    $v0,10           # exit
      syscall
```