TUTORIAL 5 INTRODUCTION TO MIPS ASSEMBLY

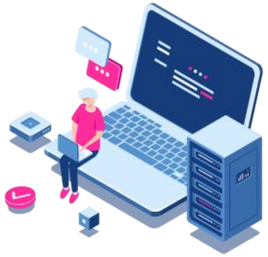
Overview

- **You will review/learn the following in this tutorial:**
  - Instruction Set Architecture (ISA)
  - MIPS data types
  - Pseudo instructions
- **You will practice to write short MIPS programs:**
  - To describe the behavior of the circuit along time

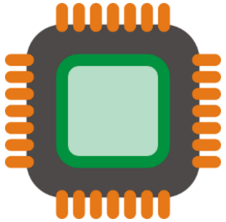
Instruction Set Architecture

- ISA defines the hardware/software interface

Software



Hardware



ISA

Instruction Set *I/O* *Memory Organization and Register Set*

ISA Types:

- *CISC (Complex Instruction Set Computer)*

e.g., x86

- *RISC (Reduced Instruction Set Computer)*

e.g., ARM, MIPS

Instruction Set Architecture (cont.)

- **instruction set architecture (ISA), is the part of the processor that is visible to the programmer or compiler writer**
  - the native data types,
  - instructions,
  - registers,
  - addressing modes,
  - memory architecture,
  - interrupt and exception handling,
  - and external I/O.

MIPS Registers

- **The MIPS central processing unit contains**
  - 32 general purpose 32-bit registers
  - numbered 0-31
- **MIPS has established a set of conventions as to how registers should be used**
  - not enforced by the hardware

MIPS General Purpose Registers

Register number	*Name*	*Used for*
0	**`zero`**	Always returns 0
1	**`at`**	(assembly temporary) Reserved for use by assembly
2–3	**`v0`**, **`v1`**	Value returned by subroutine
4–7	**`a0`–`a3`**	(arguments) First few parameters for a subroutine
8–15	**`t0`–`t7`**	(temporaries) Subroutines can use without saving
24, 25	**`t8`**, **`t9`**	
16–23	**`s0`–`s7`**	Subroutine register variables; a subroutine that writes one of these must save the old value and restore it before it exits, so the *calling* routine sees the values preserved
26, 27	**`k0`**, **`k1`**	Reserved for use by interrupt/trap handler; may change under your feet
28	**`gp`**	Global pointer; some runtime systems maintain this to give easy access to (some) `static` or `extern` variables
29	**`sp`**	Stack pointer
30	**`s8/fp`**	Ninth register variable; subroutines that need one can use this as a frame pointer
31	**`ra`**	Return address for subroutine

MIPS Data Types

.ascii	String (without null terminator)
.asciiz	String (with null terminator)
.byte	Byte (We can write the values either in base 10 or hex)
.half	2 Bytes (We can write the values either in base 10 or hex)
.word	4 Bytes (We can write the values either in base 10 or hex)
.space num	Reserves num bytes of space in memory.

Examples

Variable name	Data type	Initialized value	Remarks
var1:	.half	14	# A half-word storing the integer 14
array1:	.word	5 6 7 8	# same as int array1[4] = {5,6,7,8} in C++
array2:	.word	3:5	# the part before ":" in the initialized value is # the initial value of each element in the # array, and the part after ":" is the array size. # same as int array2[5] = {3,3,3,3,3} in C++
string1:	.byte	0x32 # '2' in ASCII code 0x4a # 'J' in ASCII code 0 # '\0' in ASCII code	# string type is actually an array of char (a byte) # same as char string1[3] = {'2','J','\0'} in C++
string2:	.asciiz	"2J"	# equivalent to string1
array3:	.space	10	An array of 10 bytes is allocated for array3 in memory.

Warm up exercise 1

- Write down the shortest sequence (any one) of MIPS instructions for the following C++ code, assuming variable a is stored in \$s0
  - a = a - 1;
- Solution:
- addi \$s0, \$s0, -1
- # note: no subi instruction in MIPS (it is pseudo-instruction)

Warm up exercise 2

- Write down the shortest sequence (any one) of MIPS instructions for the following C++ code, assuming variable a,b are stored in $s0 and $s1 respectively
  - b = a * 5;
- Solution:
- sll $s1, $s0, 2
- add $s1, $s1, $s0

MIPS Hello World

HelloWorld.asm | AddSub.asm | Logical.asm | Shift.asm | Pseudo.asm | Memory.asm

```
#helloworld
.data #data segment
message: .asciiz "\nHello World!\n"

.text #text segment
.globl _main
_main: #your code starts here
li $v0, 4 #load syscall code 4 to $v0 for print string
la $a0, message #load address of string to $a0
syscall #invoke syscall
li $v0, 10 #load syscall code 10 to $v0 for exit
syscall #invoke syscall to terminate the program
```

Mars Messages | Run I/O

Clear

```
Hello World!

-- program is finished running --
```

MIPS Arithmetic Operation

HelloWorld.asm | AddSub.asm | Logical.asm | Shift.asm | Pseudo.asm | Memory.asm

```
#simple arithmetic
.data # data segment
.text # text segment
.globl _main
_main: #your code starts here
sub $t0, $t0, $t0 #init t0=0
addi $t1, $zero, 1 #init $t1=1
addi $t2, $0, -2 #init $t2=-2
add $t3, $t0, $t1
add $t3, $t3, $t2 #t3=t0+t1+t2
```

$t0	8	0x0000ffff
$t1	9	0xaaaa0000
$t2	10	0xffff1111
$t3	11	0x0000eeee
$t4	12	0x55551111

MIPS Logical Operation

HelloWorld.asm | AddSub.asm | Logical.asm | Shift.asm | Pseudo.asm | Memory.asm

```
#logical operations
.data # data segment
.text # text segment
.globl _main
_main: #your code starts here
addi $s0, $0, 0xffff0000
addi $s1, $0, 0xaaaa1111
not $t0, $s0
and $t1, $s0, $s1
or $t2, $s0, $s1
nor $t3, $s0, $s1
xor $t4, $s0, $s1
```

$t0	8	0x0000ffff
$t1	9	0xaaaa0000
$t2	10	0xffff1111
$t3	11	0x0000eeee
$t4	12	0x55551111

MIPS Logical Shift

HelloWorld.asm | AddSub.asm | Logical.asm | Shift.asm* | Pseudo.asm | Memory.asm

```
_main: #your code starts here
addi $t0, $zero, 0xf0
sll $t1, $t0, 4 #logical shift
srl $t2, $t0, 4

addi $t0, $zero, 0xffff0000
srl $t1, $t0, 4
sra $t2, $t0, 4 #arithmetic shift

addi $t0, $0, 1 #t0=1
sll $t0, $t0, 3 #t0=8*t0, a trick for fast multiplication

addi $t0, $0, 0x12345678
addi $t1, $0, 0xf
sll $t1, $t1, 4
and $t1, $t0, $t1 #pick bit 4-7 of $t0
```

Pseudo-instruction

- The MIPS (real) instruction set is very small
- **Pseudo-instructions** are assembly language instructions that do not have a direct hardware implementation
  - Defined by assembler
  - When using pseudo-instructions in a assembly language program, the assembler translates them into equivalent real MIPS instructions
  - Provided as a convenience for the programmer

MIPS Pseudo-instructions

- abs (absolute value)
- blt (branch if less than)
- bgt (branch if greater than)
- ble (branch if less than or equal)
- bge (branch if greater than or equal)
- neg (computes the two's complement negative value)
- not (bitwise ’flip’, 1 is changed to 0 and 0 is changed to 1)
- li (load immediate)
- la (load address)
- move (move the contents of one register to another)
- sge (set greater than equal)
- sgt (set greater than)

Example

HelloWorld.asm | AddSub.asm | Logical.asm | Shift.asm | Pseudo.asm* | Memory.asm

```
#Pseudo instructions
.data # data segment
message: .asciiz "\nPseudo instructions!\n"
.text # text segment
.globl _main
_main: #your code starts here
subi $t0, $zero, 1 #t0=-1
move $t1, $t0 #t1=t0
li $v0, 4 #load syscall code 4 to $v0 for print strin
la $a0, message #load address of string to $a0
syscall #invoke syscall
li $v0, 1 #load syscall code 4 to $v0 for print
move $a0, $t1 #load address of string to $a0
syscall #invoke syscall
```

Data Transfer

Memory.asm

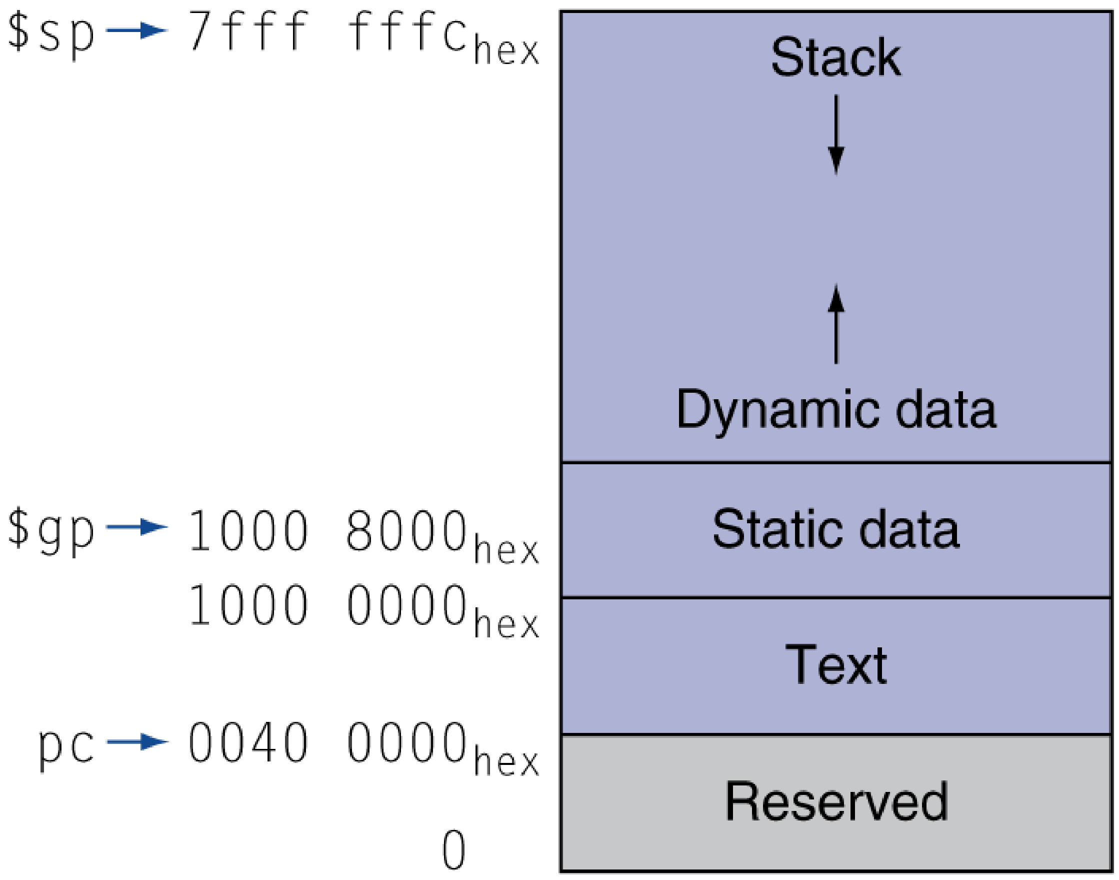
```
# data transfer
.data # data segment
fourbytes: .ascii "12AB"
fourwords: .word 1 -1 1024 -65536
.text # text segment
.globl _main
_main: #your code starts here
# let's play with byte array first
la $s0, fourbytes #base address of byte array
lb $t0, 0($s0)
lb $t1, 2($s0)
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010000	0x42413231	0x00000001	0xffffffff	0x00000400	0xffff0000

$s0	16	0x10010000
$s1	17	0x10010004

MIPS Memory Layout



Load Byte

```
# let's play with byte array first
la $s0, fourbytes #base address of byte array
lb $t0, 0($s0)
lb $t1, 2($s0)
lb $t2, 1($s0)
lb $t3, 3($s0)
```

$t0	8	0x00000031
$t1	9	0x00000041
$t2	10	0x00000032
$t3	11	0x00000042

Load Word

```
# check the word array
la $s1, fourwords
lw $t0, 0($s1)
lw $t1, 4($s1)
lw $t2, 8($s1)
lw $t3, 12($s1)
```

$t0	8	0x00000001
$t1	9	0xffffffff
$t2	10	0x00000400
$t3	11	0xffff0000

Exercise

- Write down the shortest sequence (any one) of MIPS instructions for the following C++ code, assuming the base address of the int array A is stored in the register s0. You can use $t0 for storing temporary values.
  - A[2] = A[7] + 11;
- Solution:
- lw $t0, 28($s0)
- addi $t0, $t0, 11
- sw $t0, 8($s0)

Extra exercise

- Write down the shortest sequence (any one) of MIPS instructions for the following C++ code, assuming the base address of the integer array A, variable b, variable c are stored in the register $s0, $s1 and $s2 respectively. You can use some registers for storing temporary values.
  - A[c++] = A[b] + 17;

- **Solution:**
- sll $t0, $s1, 2
- add $t0, $s0, $t0
- lw $t1, 0($t0)
- addi $t1, $t1, 17
- sll $t0, $s2, 2
- add $t0, $s0, $t0
- sw $t1, 0($t0)
- addi $s2, $s2, 1