

MIPS32 Assembly Language Getting Started (assembly language statements)

- May be grouped into 3 categories
 - ◆ Instructions
 - ☞ Tell processor what to do
 - ☞ Cause assembler to generate machine language instructions
 - ◆ Pseudoinstructions (macroinstructions, synthetic instructions)
 - ☞ Not directly supported by processor
 - ☞ Assembler generates 1 or more “true” instructions to implement them
 - ☞ Intended to reduce coding tedium, but may be disservice pedagogically
 - ◆ Assembler directives
 - ☞ Non-executable: no machine language instructions result from them
 - ☞ Provide information to assembler on various aspects of assembly process
- Entered *one per line* in source file

1

MIPS32 Assembly Language Getting Started (syntactic format for statements)

- As follows, with [] meaning optional or may not apply:

[label:] mnemonic [operands] [#comment]

- ◆ Label (must end with colon)
 - ☞ In *data* segment: creates variable name
 - ☞ In *text* segment: tags instruction for reference elsewhere
- ◆ Mnemonic
 - ☞ Tells which statement is intended (*e.g.*, **add**, **sub**, **.word**)
- ◆ Operands
 - ☞ Provide items to be operated on or indicate where they are located
 - ☞ Most instructions have 3 operands → some have less

- Examples: (each line is a *separate* example)

```
iArray: .word 1, 3, 5
loop:   addiu $t0, $t0, 1
        mflo $t1
        li $v0, 5
        la $t2, iArray
```

← C/C++: `int iArray[] = {1, 3, 5};`

2

MIPS32 Assembly Language Getting Started (comments)

- Each begins with # and extends until end of line
- Play important role, although ignored by assembler
 - ◆ More important in AL than in HLL
 - ☞ Since AL code is inherently less readable
 - ◆ Should be amply used to provide:
 - ☞ Functional description
 - ☞ Algorithmic description (pseudocode)
 - ☞ Register usage table
 - ☞ Inline documentation
 - ☞ Documentation for how parameters are passed and results returned
 - ☞ ...
 - ◆ §5.4 of textbook (pp.47-52): good reference for what's expected
- Examples:

```
##### Next line has an inline comment #####  
loop:    addiu $t0, $t0, 1      #increment $t0
```

3

MIPS32 Assembly Language Getting Started (segment declaration directives)

- **.data**
 - ◆ Marks start of a “data related section”
 - ☞ Specified in this section → data storage needs
 - ☞ (*Quick Quiz*: Where does section end?)
- **.text**
 - ◆ Marks start of a “code related section”
 - ☞ Placed in this section → code implementing algorithm
 - ☞ (*Quick Quiz*: Where does section end?)

4

MIPS32 Assembly Language Getting Started (integer & floating-point data directives)

- **.word** *w1, w2, ..., wn*
 - ◆ Allocates for *n* 32-bit items & initialize them to listed values
- **.word** *w:n*
 - ◆ Allocates for *n* 32-bit items & initialize all of them to *w*
- ✍ MIPS32 uses one *word* (32 bits) to store an integer
- ✍ Of course, a *word* can be used to store any 32-bit items
- **.float** *f1, f2, ..., fn*
 - ◆ Allocates for *n* single-precision floating-point numbers & initialize them to listed values
- **.double** *d1, d2, ..., dn*
 - ◆ Allocates for *n* double-precision floating-point numbers & initialize them to listed values

5

MIPS32 Assembly Language Getting Started (other data directives)

- **.byte** *b1, b2, ..., bn*
 - ◆ Allocates for *n* 8-bit items & initialize them to listed values
- **.half** *h1, h2, ..., hn*
 - ◆ Allocates for *n* 16-bit items & initialize them to listed values
- **.space** *n*
 - ◆ Allocates for *n* uninitialized bytes
- **.ascii** "*string*"
 - ◆ Allocates & initializes memory for/with characters in *string*
- **.asciiz** "*string*"
 - ◆ Same as **.ascii** but also adds *null-termination*
- ✍ Use C convention for special characters (**\n**, **\t**, **\"**, etc.)

6

MIPS32 Assembly Language Getting Started (data directive examples)

```
.data
var1: .byte 'A', 'E', 127, -1, '\n'
var2: .half -10, 0xffff
var3: .word 0x12345678
var4: .word 0:10
var5: .float 12.3, -0.1
var6: .double 1.5e-10
str1: .ascii "This string isn't null-terminated\n"
str2: .asciiz "This string is null-terminated\n"
array: .space 100
```

Assembler will report error if an initializing value is out of range.

7

MIPS32 Assembly Language Getting Started (miscellaneous directives)

- **.globl** *symbol*
 - ◆ Makes *symbol* global
 - ☞ Global symbols can be referenced from other files
 - ◆ We'll typically declare **main** global
 - ☞ In case it's needed to make some components (like "trap file") work
 - **.align** *n*
 - ◆ Aligns next data item on 2^n byte boundary
- ✍ **.half**, **.word**, **.float** & **.double** automatically aligns
- ✍ **.align 0** turns off such automatic alignment (within segment)

8

MIPS32 Assembly Language Getting Started (kind look @ instructions/pseudoinstructions)

- Arithmetic/logical
- Constant manipulating
- Comparison
- Branch
- Jump
- Load
- Store
- Data movement
- Floating-point
- Exception/interrupt
- Trap

as used by
James R Larus
in "Appendix A"
posted under
Other Resources

9

MIPS32 Assembly Language Getting Started (**syscall**: an exception/interrupt instr'n)

- Provided by MARS (and SPIM) for getting system-like services
 - ◆ Input/output various data types, exit program, request dynamic memory, *etc*
- How to use **syscall** to get services:
 - ◆ Load service code into register **\$v0**
 - ☞ Each service identified by a unique service code (see next slide)
 - ◆ Load arguments, if any, into registers **\$a0**, **\$a1**, **\$a2** & **\$a3**
 - ☞ For floating-point argument → load into **\$f12** instead
 - ◆ Issue **syscall** instruction
 - ◆ Retrieve return value, if any, from **\$v0**
 - ☞ For floating-point result → retrieve from **\$f12** instead

10

MIPS32 Assembly Language Getting Started (some `syscall` services - all we'll need for now)

Service	Code (in \$v0)	Arguments / Result
print_int	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_int	5	\$v0 = integer read
read_float	6	\$f0 = float read
read_double	7	\$f0 = double read
read_string	8	\$a0 = buffer address \$a1 = buffer size <i>reads up to "buffer size - 1" characters & null terminates</i>
sbrk	9	\$a0 = bytes to dynamically allocate \$v0 = address of allocated space
exit	10	
print_char	11	\$a0 = character to print
read_char	12	\$v0 = character read

11

MIPS32 Assembly Language Getting Started (intro-by-example instructions pre-explained)

- **addiu** *Rt, Rs, Imm*
 - ◆ Put sum of *Rs* and *sign-extended Imm* into *Rt*
- **addu** *Rd, Rs, Rt*
 - ◆ Put sum of *Rs* and *Rt* into *Rd*
- **beqz** *Rsrc, Label*
 - ◆ Conditionally branch to instruction at *Label* if *Rsrc* equals 0
- **bgt** *Rsrc1, Rsrc2, Label*
 - ◆ Conditionally branch to instruction at *Label* if *Rsrc1* is greater than *Rsrc2*
- **blt** *Rsrc1, Rsrc2, Label*
 - ◆ Conditionally branch to instruction at *Label* if *Rsrc1* is less than *Rsrc2*
- **j** *target*
 - ◆ Unconditionally jump to instruction at *target*
- **la** *Rdest, address*
 - ◆ Load computed *address* into *Rdest*
- **lb** *Rt, address*
 - ◆ Load byte at *address* into *Rt*, sign-extended
- **li** *Rdest, value*
 - ◆ Load *value* into *Rdest*
- **move** *Rdest, Rsrc*
 - ◆ Copy contents of *Rsrc* to *Rdest*
- **sb** *Rt, address*
 - ◆ Store low byte from *Rt* to memory at *address*

12

MIPS32 Assembly Language Getting Started (template for main program)

```
#####
# Title:                                     Author:
# Class: CS2318-???       Due:               Submitted:
#####
# Functional description:
# ...
#####
# Register usage:
# ...
#####
# Pseudocode description:
# ...
##### data segment #####
.data
...
##### code segment #####
.text
.globl main
main:
...
li $v0, 10          # exit
syscall
```

13

MIPS32 Assembly Language Getting Started (template for procedure/function)

```
#####
# Function name:
#####
# Functional description:
# ...
#####
# Input/output description:
# ...
#####
# Example calling sequence:
# ...
#####
# Register usage:
# ...
#####
# Pseudocode description:
# ...
#####

<proc name>:
...
```

← how parameters are passed
& how results are returned

14

MIPS32 Assembly Language Getting Started (reading & printing strings)

```
...
##### data segment #####
.data
prompt: .asciiz "Enter your name: "
hello: .asciiz "Hello "
newline: .asciiz "\n"
name: .space 101      # array of 101 bytes
##### code segment #####
.text
.globl main
main:
    la $a0, prompt    # $a0 = address of prompt string
    li $v0, 4          # print prompt string
    syscall

    la $a0, name      # $a0 = name buffer address
    li $a1, 101        # $a1 = name buffer size
    li $v0, 8          # read name string
    syscall
```

15

MIPS32 Assembly Language Getting Started (reading & printing strings)

```
    la $a0, newline    # $a0 = address of newline string
    li $v0, 4          # print newline string
    syscall

    la $a0, hello      # $a0 = address of hello string
    li $v0, 4          # print hello string
    syscall

    la $a0, name        # $a0 = address of name string
    li $v0, 4          # print name string
    syscall

    la $a0, newline    # $a0 = address of newline string
    li $v0, 4          # print newline string
    syscall

    li $v0, 10         #exit
    syscall
```

16

MIPS32 Assembly Language Getting Started (reading and printing an integer)

```
...
##### code segment #####
        .text
        .globl main
main:
        li $v0, 5          # read integer
        syscall            # $v0 = value read

        move $a0, $v0      # $a0 = value to print
        li $v0, 1          # print integer
        syscall

        li $v0, 10         # exit
        syscall
```

17

MIPS32 Assembly Language Getting Started (sum 3 integers)

```
...
##### data segment #####
        .data
prompt: .asciiz "Enter 3 integers: \n"
sum_msg: .asciiz "The sum of the 3 integers is "
##### code segment #####
        .text
        .globl main
main:
        la $a0, prompt      # print prompt string
        li $v0, 4
        syscall

        li $v0, 5           # read 1st integer into $t0
        syscall
        move $t0, $v0

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

18

MIPS32 Assembly Language Getting Started (sum 3 integers)

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

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

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

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

li $v0, 10          # exit
syscall
```

19

MIPS32 Assembly Language Getting Started (case conversion)

```
...
##### data segment #####
.data
name_prompt: .asciiz "Please type your name: "
out_msg: .asciiz "Your name in all-uppercase: "
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
    la $a0, out_msg      # print output message
    li $v0, 4
    syscall
```

20

MIPS32 Assembly Language Getting Started (case conversion)

```
loop:      la $t0, in_name
           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         # print converted string
           li $v0, 4
           syscall
           li $v0, 10              # exit
           syscall
```

21

MIPS32 Assembly Language Getting Started (end-of-getting-started caveat)

- Instruction naming "quirk" in MIPS32 Instruction Set:
 - ◆ *unsigned* as used in add and subtract instructions is a *misnomer*
 - ◆ Difference between signed and unsigned versions of those instructions is not in sign extension (or lack thereof) of operands
 - ◆ Difference is in whether trap is executed on overflow
 - ☞ Normal version → trap executed on overflow
 - ☞ Unsigned version → overflow ignored
 - ◆ Bottom line: immediate operand to add/subtract instructions is *always sign-extended*

22