Computer Systems and Architecture

MIPS

Ruben Van den Bossche

University of Antwerp

Outline

MIPS

Registers and memory

Language constructs

Exercises

Outline

MIPS

Registers and memory

Language constructs

Exercises

Assembly language

- very closely related to *machine language*
- easier to read
- **MIPS**: RISC ISA
- **MARS**: MIPS Assembler and Runtime Simulator

Example: add

```
# add.asm-- A program that computes the sum of 1 and 2,
# leaving the result in register $t0.
# Registers used:
# t0 - used to hold the result.
# t1 - used to hold the constant 1.

li  $t1, 1       # load 1 into $t1.
add $t0, $t1, 2 # $t0 = $t1 + 2.

# end of add.asm
```

Labels

- symbolic name for an address in memory
- jump to a label with the *branch* instructions

```
# add.asm-- A program that computes the sum of 1 and 2,
# leaving the result in register $t0.
# Registers used:
# t0 - used to hold the result.
# t1 - used to hold the constant 1.

main:                  # start execution at main
    li   $t1, 1        # load 1 into $t1.
    add $t0, $t1, 2 # $t0 = $t1 + 2.

# end of add.asm
```

Syscalls

- suspends the execution of your program and transfers control to the operating system
- register `$v0` contains operation code[1]:

Service	Code	Arguments	Result
print_int	1	`$a0`	*none*
print_float	2	`$f12`	*none*
print_double	3	`$f12`	*none*
print_string	4	`$a0`	*none*
read_int	5	*none*	`$v0`
read_float	6	*none*	`$f0`
read_double	7	*none*	`$f0`
read_string	8	`$a0` (address), `$a1` (length)	*none*
sbrk	9	`$a0` (length)	`$v0`
exit	10	*none*	*none*

[1] MIPS Assembly Language Programming, CS50 Discussion and Project Book, Daniel J. Ellard

Syscalls: example

```
# add.asm-- A program that computes the sum of 1 and 2,
# leaving the result in register $t0.
# Registers used:
# t0 - used to hold the result.
# t1 - used to hold the constant 1.

main:                # start execution at main
    li   $t1, 1      # load 1 into $t1.
    add $t0, $t1, 2  # $t0 = $t1 + 2.

    move $a0, $t0        # move result to $a0
    li   $v0, 1      # load syscall print_int into $v0.
    syscall

exit:
    li   $v0, 10     # syscall code 10 is for exit
    syscall

# end of add.asm
```

Outline

MIPS

Registers and memory

Language constructs

Exercises

Registers

- 32 normal registers
- 32 floating point registers (single precision)
- register usage guidelines [2]:

Symbolic Name	Number	Usage
zero	0	Constant 0.
at	1	Reserved for the assembler.
v0-v1	2-3	Result registers.
a0-a3	4-7	Argument registers.
t0-t9	8-15, 24-25	Temporary registers.
s0-s7	16-23	Saved registers.
k0-k1	26-27	Kernel registers.
gp	28	Global data pointer
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address
f0-f31		Floating point registers

[2] MIPS Assembly Language Programming, CS50 Discussion and Project Book, Daniel J. Ellard

Memory

- Allocate space for variables and data in memory
- Assembler directives `.text` and `.data`
- Data directives:
  - `.ascii` string
  - `.asciiz` string (null-terminated)
  - `.byte` 8-bit integers
  - `.half` 16-bit integers
  - `.space` *n* bytes of space
  - `.word` 32-bit integers
- Load and store data to and from memory with `lw` and `sw` (words)

Memory: example (1)

```
# helloworld.asm-- A "Hello World" program.
# Registers used:
# $v0 - syscall parameter and return value.
# $a0 - syscall parameter-- the string to print.

# Data for the program:
    .data
hello_msg:  .asciiz "Hello World!\n"

    .text
main:
    la   $a0, hello_msg # load the addr of hello_msg into $a0.
    li   $v0, 4       # 4 is the print_string syscall.
    syscall           # do the syscall.

exit:
    li   $v0, 10      # syscall code 10 is for exit
    syscall

# end helloworld.asm
```

Memory: example (2)

```
# loadandstore.asm-- Demonstrate load and store instructions
# by implementing c = a + b

    .data
var_a:  .word 5     # variable a
var_b:  .word 8     # variable b
var_c:  .word 0     # variable c

    .text
main:
    lw      $t1, var_a      # load a in $t1
    lw      $t2, var_b      # load b in $t2
    add     $t0, $t1, $t2   # add a and b
    sw      $t0, var_c      # store sum into c

exit:
    li      $v0, 10     # syscall code 10 is for exit
    syscall

# end loadandstore.asm
```

Outline

MIPS

Registers and memory

Language constructs

Exercises

Conditional statements

```
# conditional.asm
# c = max( a, b )

    .data
var_a:  .word 8     # variable a
var_b:  .word 5     # variable b
var_c:  .word 0     # variable c

    .text
main:
    lw      $t1, var_a      # load a in $t1
    lw      $t2, var_b      # load b in $t2

    # conditional: if a > b
    bgt     $t1, $t2, t1_greater
    sw      $t2, var_c      # store b into c
    j       endif
t1_greater:
    sw      $t1, var_c      # store a into c
endif:
```

Loops

```
# loop.asm
# c =  a x b
    .data
var_a:  .word 8     # variable a
var_b:  .word 5     # variable b
var_c:  .word 0     # variable c

    .text
main:
    lw      $t1, var_a      # load a in $t1
    lw      $t2, var_b      # load b in $t2

    # loop: add a to result, do this b times
    li      $t0, 0          # loop register
    li      $t3, 0          # result register
loop:
    bge     $t0, $t2, endloop   # end loop if loop register >= b
    add     $t3, $t3, $t1       # add a to result
    addi        $t0, $t0, 1     # increase loop register
    j       loop
endloop:
    sw      $t3, var_c          # store result into c
```

Outline

MIPS

Registers and memory

Language constructs

Exercises

Exercises

- `http://msdl.cs.mcgill.ca/people/hv/teaching/ComputerSystemsArchitecture/#csw9`