Computer Systems and Architecture
MIPS: Introduction

Stephen Pauwels

Academic Year 2018-2019

Universiteit Antwerpen

Outline

- MIPS
- Registers and Memory
- Language Constructs
- Exercises

Assembly Language

- Very closely related to machine language
- easier to read
- MIPS
- 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 $0.
# Registers used:
# t0 : used to hold the result
# t1 : used to hold the constant 1

li     $t1, 1       # load value 1 into $t1
addi   $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 value 1 into $t1
        j add               # jump to label ‘add’
        addi $t0, $t1, 1    # $t0 = $t1 + 1
add:
        addi $t0, $t1, 2    # $t0 = $t1 + 2
# end of add.asm
```

Registers

- 32 normal registers
- 32 floating point registers (single precision)
- Register usage guidelines:

Symbolic Name	Number	Usage
zero	0	Constant 0
at	1	Reserved for 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

Syscalls

- Suspends the execution of your program and transfers control to the operating system
- Register $v0 contains operation code.

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

Syscall: example

```
# add.asm: A program that computes the sum of 1 and 2
# Printing the result
# Registers used:
# t0 : used to hold the result
# t1 : used to hold the constant 1
main:
        li      $t1, 1          # load 1 into $t1
        addi    $t0, $t1, 2     # $t0 = $t1 + 2

        move    $a0, $t0        # set result to $a0
        li      $v0, 1          # load code for print_int
        syscall
exit:
        li      $v0, 10         # load code for exit
        syscall
# end of add.asm
```

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` (load and store 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
hello_msg:    .asciiz      “Hello World!\n”

       .text
main:
       la $a0, hello_msg   # load the addr of hello_msg in $a0
       li $v0, 4           # load code for print_string
       syscall
exit:
       li $v0, 10          # load code for exit
       syscall
```

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             # load code for exit
       syscall
```

Conditional statements

```
# conditional.asm
# c = max(a, b)
        .data
var_a:  .word 8        # variable a
var_b:  .word 14       # 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      # branch if $t1 > $t2
        sw $t2, var_c                 # store b into c
        j endif                       # jump to endif
t1_greater:
        sw $t1, var_c                 # store a into c
endif:
        li $v0, 10                    # load code for exit
        syscall
```

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                       # jump to loop
endloop:
        sw $t3, var_c                # store result into c
```

Exercises

- Blackboard
- Course webpage
  - http://msdl.cs.mcgill.ca/people/hv/teaching/ComputerSystemsArchitecture/#CS6

Overview of arithmetic instructions

ADD	Addition (with overflow)
ADDI	Addition immediate (with overflow)
ADDIU	Addition immediate unsigned (no overflow)
ADDU	Addition unsigned (no overflow)

SUB, SUBI, SUBIU, SUBU	Identical to ADD but with subtraction
MUL, MULU	Identical to ADD but with multiplication
DIV, DIVU	Identical to ADD but with division

ADD.S	Addition (single precision)
ADD.D	Addition (double precision)
SUB.S, SUB.D	Subtraction (single or double precision)
MUL.S, MUL.D	Multiplication (single or double precision)
DIV.S, DIV.D	Division (single or double precision)

Overview of logical instructions

AND	Bitwise and
ANDI	Bitwise and immediate
OR	Bitwise or
ORI	Bitwise or immediate
XOR	Bitwise exclusive or
XORI	Bitwise exclusive or immediate

Overview of branching instructions

BEQ(Z)	Branch on equal than (zero)
BGE(Z)	Branch on greater than or equal to (zero)
BGT(Z)	Branch on greater than (zero)
BLE(Z)	Branch on less than or equal to (zero)
BLT(Z)	Branch on less than (zero)
BNE(Z)	Branch on not equal to (zero)
J	Jump
JAL	Jump and link
JR	Jump register

Overview of store/load instructions

LB	Load byte
LH	Load half-word
LW	Load word
LA	Load address

SB	Store byte
SH	Store half-word
SW	Store word