SOM2IF15 – Compilation

MIPS Assembly

Frédéric Loulergue

2022

The MIPS Processor

- RISC processor
- simple and representative of modern processors
- simulator (SPIM)

Presentation

- Memory and Registers
- Instruction Set

Outline

Memory

Memory Management and View

- All modern processors have a memory management unit (MMU) that allows to manipulate virtual addresses (cf. operating system class)
- For the user, memory is a huge array and its indices are memory addresses

Sizes and Alignment

- Generally, the smallest addressable unit is the **byte** (8 bits)
- The size of integers is usually bigger on modern processors (usually 64 bits, sometimes 32 bits nowdays)
- Memory words are aligned: the addresses of such words are multiple of 4 on a 32 bits architecture (8 on a 64 bits architecture)
- Non-aligned accesses (non multiple of 4 on a 32 bits architecture) are not allowed or very inefficient
- MIPS: `lw` (loading a memory word) requires an aligned address

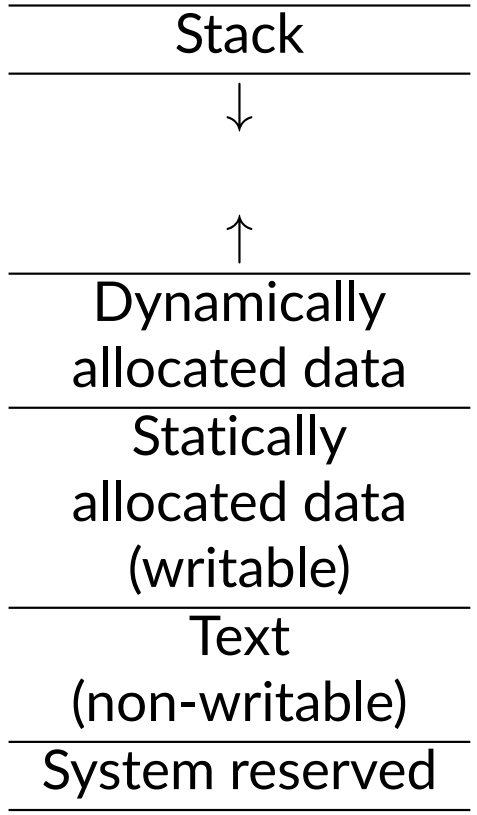
Memory

Memory Organization

- A program memory is divided in various zones
- The convention depends on the processor **and** the operating system

Memory Zones

- The **stack**: used by functions/routines of the program (for e.g. to store intermediate results)
- Unused memory (end of the stack to the next zone)
- The **heap**: contains data dynamically allocated by the program. This zone grows explicitly (upwards), in high level languages, by instructions such as `new` or `malloc`.
- Data **statically allocated** by the program (for e.g. global variables in C)
- The **text** of the program: the list of instructions



Registers

MIPS Processor (in SPIM)

32 general purpose registers, but:

- register 0 (`$zero`) always contains `0`, even after a writing
- register 1 reserved for assembler
- register 26 and 27 reserved for OS kernel
- register 28 (`$gp`) points to the global area (statically allocated memory)
- register 29 (`$sp`) points to the stack
- register 30 (`$fp`) frame pointer
- register 31 (`$ra`) implicitly used by some instructions to save the return address before a jump

Registers

Register Names in Assembly Code and Conventional Usage

- Register names start with `$`
- Names already discussed: `$zero`, `$gp`, `$sp`, `$fp`, `$ra`
- Registers 2–3: `$v0`, `$v1`
  (function result, system call)
- Registers 4–7: `$a0`, ..., `$a3`
  (routine arguments)
- Registers 8–15, 24–25: `$t0`, ..., `$t9`
  (temporary not preserved across call)
- Registers 16–23: `$s0`, ..., `$s7`
  (temporary preserved across call)

Outline

MIPS Instruction Set

Addressing Modes

Type	Description	Syntax
Immediate	an integer constant	constant
Direct	content of a register	register
Indirect	content at an address contained in a register	(register)
Indirect indexed	same as indirect + offset	offset(register)
Label	address of label	label
Indexed Label	address of label + offset	label +/- offset
Indirect Indexed Label		label +/- offset(reg.)

Example 1

```
        .data
msg:
        .asciiz "Hello World!\n"

        .text
main:
        li $v0, 4       # immediate (print_string is system call 4)
        la $a0, msg     # label     (the address of the string)
        syscall         # system call
        li $v0, 10      # immediate (exit is system call 10)
        syscall         # system call
```

Example 2

```
    .data
global:
    .space 4
    .asciiz "Hello World!\n"

    .text
main:
    la $gp, global # label
    li $v0, 4      # immediate (print_string is system call 4)
    la $a0, 4($gp) # indirect indexed (the address of the string)
    syscall        # system call
    li $v0, 10     # immediate (exit is system call 10)
    syscall        # system call
```

Instruction Set

Notations

n	constant
ℓ	address
r	register
a	absolute (n, ℓ)
o	operand (r, a)

Overview

- Instruction with a format similar to:

  `add` r_1, r_2, o writes in r_1 the value of $r_2 + o$

- Instructions to deal with memory:

  `lw` r_1, $n(r_2)$ writes in r_1 the word contained at address $r_2 + n$

  `sw` r_1, $n(r_2)$ writes r_1 in the word contained at address $r_2 + n$

- Control flow instructions

  `bne` r, a, ℓ jumps at address ℓ if r and a contain different values

  `jal` o writes $pc + 1$ in `$ra` and jumps at o

Instruction Set: Summary

Syntax	Effect
`move` r_1, r_2	$r_1 = r_2$
`add` r_1, r_2, o	$r1 = r_2 + o$
`sub` r_1, r_2, o	$r1 = r_2 - o$
`mul` r_1, r_2, o	$r1 = r_2 \times o$
`div` r_1, r_2, o	$r1 = r_2 \div o$
`li` r_1, n	$r1 = n$
`la` r_1, a	$r1 = a$
`and` r_1, r_2, o	$r1 = r_2$ `land` o
`or` r_1, r_2, o	$r1 = r_2$ `lor` o
`xor` r_1, r_2, o	$r1 = r_2$ `lxor` o
`syscall`	system call

Syntax	Effect
`lw` $r_1, o(r_2)$	$r_1 = mem[r_2 + o]$
`sw` $r_1, o(r_2)$	$mem[r_2 + o] = r_1$
`slt` r_1, r_2, o	$r1 = r_2 < o$
`sle` r_1, r_2, o	$r1 = r_2 <= o$
`seq` r_1, r_2, o	$r1 = r_2 == o$
`sne` r_1, r_2, o	$r1 = r_2 ! = o$
`j` o	$pc = o$
`jal` o	$ra = pc + 1$;
	$pc = o$
`beq` r_1, r_2, a	$pc = a$ if $r_1 == r_2$
`bne` r_1, r_2, a	$pc = a$ if r_1 != r_2

- registers are considered as variables in a high-level language
- $=$ denotes assignment
- for an `op`, `lop` denotes the corresponding bitwise boolean operation
  Example: 3 `land` 42 is 2 (11_b `land` 101010_b is 10_b)
- `mem` denotes the memory seen as an array

Instruction Set: System Calls

`syscall`

- Identifier of the system call (number) in `$v0`
- No argument or argument in `$a0`
- Some useful system calls (page A-44 for full list)

Service	Id	Arguments/Result
print_int	1	`$a0`: input integer
print_string	4	`$a0`: address of null-terminating string
read_int	5	read integer in `$v0`
sbrk	9	`$a0`: size to allocate `$v0`: address of the allocated memory
exit	10	

SPIM

Assembly / Machine Language

- SPIM takes assembly code directly as input
- Translation to machine language is done silently
- Linking is unnecessary

Directives

- `.text`: the instructions follow
- `.data`: statically allocated data follows
- `.space` *n*: allocates *n* bytes
- `.asciiz` *s*: string with null-termination
- `.byte` *n*: allocates a byte with initial value
- ...(see pages A-47–A49)

Labels

- Labels are symbolic names of addresses (data, program)
- a SPIM program requires a `main:` label

Exercise

Sign

Write a program in a file `sign.s` that prompts the user to input an integer number and then prints if the number is positive, zero, or negative.

Sieve of Eratosthene

```
SIZE = 1000
IS_PRIME = 1
        .data
newline:.asciiz "\n"
primes: .space SIZE

        .text
main:   li $t8, SIZE
        li $t9, IS_PRIME
        # Initialization
        sb $zero, primes
        sb $zero, primes + 1
        li $t0, 2
init_loop:
        sb $t9, primes($t0)
        add $t0, $t0, 1
        blt $t0, $t8, init_loop
        # The Sieve
        li $t0, 2
        lb $t2, primes($t0)
        beqz $t2, next_candidate
sieve_loop:
        move $t1, $t0
loop_remove_multiples:
        add $t1, $t1, $t0
        bge $t1, $t8, next_candidate
        sb $zero, primes($t1)
        j loop_remove_multiples
next_candidate:
        add $t0, $t0, 1
        blt $t0, $t8, sieve_loop
        # Print the results
        li $t0, -1
print_loop:
        add $t0, $t0, 1
        bge $t0, $t8, exit
        lb $t1, primes($t0)
        beqz $t1, print_loop
        li $v0, 1
        move $a0, $t0
        syscall
        li $v0, 4
        la $a0, newline
        syscall
        j print_loop
        # Exit
exit:   li $v0, 10
        syscall
```

Routines

Simple Routines

- That don't call other routines (including itself)
- With at most 4 arguments (in `$a0` to `$a3`)
- Return values in `$v0` and `$v1`
- To call: `jal label`
- Return to caller with `j $ra`

Example

```
#-----------------------------------
# Routine: println
# Argument: integer in $a0
# Return value: none
# Description:
#   Print the integer and a new line.
#-----------------------------------
println:
        li $v0, 1
        syscall
        li $v0, 4
        la $a0, newline
        syscall
        j $ra
```

Routines

Argument and Temporary Registers

- It is the responsibility of the **caller** to save `$t` and `$a` registers if needed (the caller is not guaranteed the called routine won't change their values)
- It is the responsibility of the **callee** to to save `$s` registers if needed (the caller is guaranteed the values of these registered won't change after the call).

Examples

```
#-------------------------------------
# Routine: fill_byte_array
# Arguments:
# - $a0: start address
# - $a1: end address (excluded)
# - $a2: initialization value
# - $a3: increment
# Description:
#   Fill the array with the given value
#-------------------------------------
fill_byte_array:
        move $t0, $a0
fill_loop:
        bge $t0, $a1, fill_end
        sb $a2, ($t0)
        add $t0, $t0, $a3
        j fill_loop
fill_end:
        j $ra
```

Examples

```
#-------------------------------------
# Routine: print_sieve
# Arguments:
# - $a0: address of array
# - $a1: size of array
# Description:
#  Print the index for non-zero cells
#-------------------------------------
print_sieve:
        sub $sp, $sp, 16
        sw $s0, 12($sp)
        sw $s1, 8($sp)
        sw $s2, 4($sp)
        sw $ra, 0($sp)
        move $s1, $a0
        move $s2, $a1
        li $s0, -1
print_loop:
        add $s0, $s0, 1
        add $t2, $s1, $s0
        bge $s0, $s2, end_print_loop
        lb $t1, ($t2)
        beqz $t1, print_loop
        move $a0, $s0
        jal println
        j print_loop
end_print_loop:
        lw $s0, 12($sp)
        lw $s1, 8($sp)
        lw $s2, 4($sp)
        lw $ra, 0($sp)
        add $sp, $sp, 16
        j $ra
```

Sieve of Eratosthene (Structured)

```
SIZE = 1000
IS_PRIME = 1
        .data
newline:.asciiz "\n"
primes: .space SIZE
        .text
main:
# Initialization
        la $a0, primes
        la $a1, primes + SIZE
        li $a2, IS_PRIME
        li $a3, 1
        jal fill_byte_array
        sb $zero, primes
        sb $zero, primes + 1

# The Sieve
        li $s0, 2
        lb $t0, primes($s0)
        beqz $t0, next_candidate
sieve_loop:
        add $t0, $s0, $s0
        la $a0, primes($t0)
        la $a1, primes + SIZE
        li $a2, 0
        move $a3, $s0
        jal fill_byte_array
next_candidate:
        add $s0, $s0, 1
        li $t0, SIZE
        blt $s0, $t0, sieve_loop
# Print the results
        la $a0, primes
        li $a1, SIZE
        jal print_sieve
# Exit
exit:   li $v0, 10
        syscall
```

Sieve of Eratosthene (Dynamic)

```
IS_PRIME = 1
    .data
prompt:
    .asciiz "Enter a positive number: "
newline:
    .asciiz "\n"
    .text
main:
# Allocation
    jal read_positive
    move $s2, $v0 # $s2: size
    move $a0, $v0
    li $v0, 9
    syscall
    move $s1, $v0 # $s1: array
# Initialization
    move $a0, $s1
    add $a1, $s1, $s2
    li $a2, IS_PRIME
    li $a3, 1
    jal fill_byte_array
    sb $zero, ($s1)
    sb $zero, 1($s1)
    li $s0, 2    # $s0: prime candidate
    add $t0, $s1, $s0
    lb $t0, ($t0)
    beqz $t0, next_candidate
sieve_loop:
    add $t0, $s0, $s0
    add $t0, $s1, $t0
    add $t1, $s1, $s2
    move $a0, $t0
    move $a1, $t1
    li $a2, 0
    move $a3, $s0
    jal fill_byte_array
next_candidate:
    add $s0, $s0, 1
    blt $s0, $s2, sieve_loop
# Print the results
    move $a0, $s1
    move $a1, $s2
    jal print_sieve
# Exit
exit:
    li $v0, 10
    syscall
```

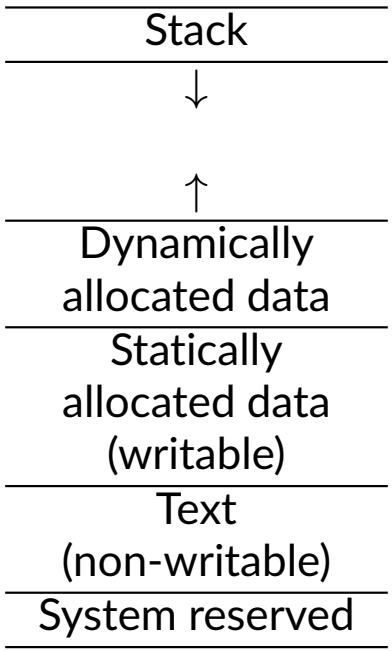
Factorial

Factorial

```
def fact(n):
  if n <= 0:
    return 1
  else:
    return n * fact(n - 1)
```

Recursion

- Depth of recursion is a priori unknown
- Register cannot be used to save callee saved registers
- We need to use the **stack**



Example

```
    .data
prompt:
    .asciiz "Enter a positive number: "
nl:
    .asciiz "\n"
    .text
main:
    li $v0, 4
    la $a0, prompt
    syscall
    li $v0, 5
    syscall
    bltz $v0, main
    move $a0, $v0
    jal fact
    move $s0, $v0
    li $v0, 1
    move $a0, $s0
    syscall
    li $v0, 4
    la $a0, nl
    syscall
    li $v0, 10
    syscall
```

```
#-----------------------
# Routine: fact
# Argument: an integer $a0
# Return: factorial in $v1
#-----------------------
fact:
    blez $a0, fact_1
    sub $sp, $sp, 8 # reserve two words
    sw $a0, 4($sp)  # push $a0
    sw $ra, 0($sp)  # push $ra
    sub $a0, $a0, 1
    jal fact
    lw $ra, 0($sp)  # pop $ra
    lw $a0, 4($sp)  # pop $a0
    add $sp, $sp, 8 # free unused
    mul $v0, $a0, $v0
    j $ra
fact_1:
    li $v0, 1
    j $ra
```