Umeå University

Computer Science Department, 2012/2013

Computer Organization and Architecture
Assignment 1

Authors :
Rémi Destigny (ens12rdy@cs.umu.se)
Paul Laturaze (ens12ple@cs.umu.se)
Isaline Laurent (ens12ilt@cs.umu.se)

4 décembre 2012

1 Introduction

MIPS instructions are represented as 32-bits numbers. Those numbers are split in fields, which indicate what the instruction is supposed to do. The common point among all instructions is the *opcode* field which is used to find out the instruction familly it belongs to. This field is always stored in the first 6 bits. All instructions comply with a format, which determines which fields are used, and how. There is several format, each corresponding to a part of this report.

The aim of this program is to analyze a file containing MIPS instructions in either hexadecimal or decimal representations. In output it must provide for each instruction the following information :

- The number analyzed, from the input file.
- The format of the instruction.
- The decomposed representation in decimal.
- The decomposed representation in hexadecimal.
- The decomposed representation in mnemonic format.

Table des matières

2 Building the solution

blabla

3 User manual

blablabla

4 Implementation

blablablabla

5 Handled MIPS instructions

5.1 Classical formats

R format

There is two possible decompositions for an instruction in R-format. The first one is the following :

opcode (6 bits)	rs (5 bits)	rt (5 bits)	rd (5 bits)	shamt (5 bits)	function (6 bits)

Table 1 – R-format first representation

The second one is described below :

opcode (6 bits)	rs (5 bits)	rt (5 bits)	0 (10 bits)	function (6 bits)

Table 2 – R-format second representation

The second representation corresponds to mnemonic representations which only use registers *rs* and *rt*. From mnemonic representations following recurrent display formats can be extracted :

function_name	rd
function_name	rd rs
function_name	rd rs rt
function_name	rd rt imm
function_name	rd rt rs
function_name	rs
function_name	rs rd

Table 3 – Reccurent display formats

To determine a mnemonic representation from hexadecimal or decimal value, two fields are important : the *opcode* field and the *function* field. For R-format instruction the *opcode* field can take two values : 0 or 28 (in decimal). Then *function* field is used to get the corresponding mnemonic representation.

I format

Instructions in I-format correspond to the following representation :

opcode (6 bits)	rs (5 bits)	rt (5 bits)	imm (16 bits)

Table 4 – I-format representation

From mnemonic representation, following recurrent formats can be extracted :

function_name	rs imm
function_name	rs label
function_name	rs rt imm
function_name	rs rt label
function_name	rt addr
function_name	rt imm
function_name	rt rs imm

Table 5 – Recurrent mnemonic format for instruction in I-format

To determine a mnemonic representation from its hexadecimal or decimal value, the most important field is the *opcode* field which can take the following values : 1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 32, 33, 34, 35, 36, 37, 38, 40, 41, 42, 43, 46, 48, 56. If the *opcode* is equal to 1 then the *rt* value determines which function has to be displayed. That value can be : 0, 1, 8, 9, 10, 11, 12, 14, 16, 17.

J format

Instruction in J-format have the following representation :

opcode (6 bits)	target (26 bits)

Table 6 – J-format representation

Mnemonic representation for that type of instruction is described below :

function_name	target

Table 7 – Mnemonic representation for J-format instructions

To determine the mnemonic representation from its hexadecimal or decimal value, the only important field is the *opcode* field which can take one of the following value : 2, 3.

5.2 Custom formats

Some instructions have a special format which does not match R,I or J. These instructions are the following : `bc1t`, `bc1f`, `mtc0`, `mtc1`, `mfc0`, `mfc1`, `eret`, `syscall`, `break` and `nop`.

Some of these operations involve the coprocessor, which are identified with C-format. For those which branch on the coprocessor, BC-format is used. Interruption as `syscall` and `break` are in IRQ-format. `eret` and `nop` have their own format, respectively E-format and NOP-format.

C format

The C-format has the following structure :

opcode (6 bits)	format_code (5 bits)	rt (5 bits)	rd or fs (5 bits)	0 (11 bits)

Table 8 – Description of C-format

The mnemonic representation depends on the value of *opcode* field and on the value of the *format_code* field, which respectively take value : 16 or 17 and 0 or 4.

BC format

The BC-format has the following structure :

opcode (17) (6 bits)	8 (5 bits)	cc (3 bits)	flag (2 bits)	offset (16 bits)

Table 9 – Description of BC-format

The mnemonic representation depends on the value of the *flag* field, which take value : 0 or 1.

IRQ format

The IRQ-format has the following structure :

opcode (0) (6 bits)	code (20 bits)	function_code (6 bits)

Table 10 – Description of IRQ-format

The mnemonic representation depends on the value of the *function_code* field, which take value : 12 or 13.

E format

The E-format has the following structure :

opcode (6 bits)	flag (1) (1 bits)	code (0) (19 bits)	function_code (24) (6 bits)

Table 11 – Description of C-format

The mnemonic representation for this format only match `eret` instruction.

NOP format

The `nop` instruction is treated as a special instruction. `nop` has the following structure :

0 (6 bits)	0 (5 bits)	0 (5 bits)	0 (5 bits)	0 (5 bits)	0 (6 bits)

Table 12 – nop representation

That representation is the same as `sll` with *rs*, *rd*, *rt* and *shamt* set to 0. So to differentiate these two instructions, an attempt to match `nop` representation is done before trying to match other representations.