

Question 4:

(5 marks)

You are required to develop a new MIPS-like instruction set architecture. The following are the processor specifications and instruction set requirements:

- i. The processor has 64 32-bit general purpose registers, 64 32-bit floating-point register, and the instruction register is 32-bit width.
- ii. The addressing modes are: Register, Immediate, Base-offset, PC-relative, and Pseudo-direct addressing modes; and the memory capacity is 4 GByte (Byte-addressable).
- iii. R-, I-, and J-type instruction formats are required, and the immediate operands should be 16-bit width.
- iv. A floating-point instruction format is needed to support both single and double precision instructions on three register operands, two sources and one destination.

Design the instruction formats to maximize the number of operations encoded by each format while satisfying the above requirements.

- a. Illustrate with neat plots the four instruction formats.
- b. Construct a table to indicate the number of instructions that can be encoded by each format.
- c. Indicate with equations how the memory address is calculated for PC-relative and Pseudo direct addressing modes in your architecture.

Good Luck

Dr. Mohammed Mersy

MIPS Reference Cheat Sheet

INSTSTRUCTION SET (SUBSET)

Name (format, op, funct)	Syntax	Operation
add (R,0,32)	add rd,rs,rt	reg(rd) := reg(rs) + reg(rt);
add immediate (I,8,na)	addi rt,rs,imm	reg(rt) := reg(rs) + signext(imm);
add immediate unsigned (I,9,na)	addiu rt,rs,imm	reg(rt) := reg(rs) + signext(imm);
add unsigned (R,0,33)	addu rd,rs,rt	reg(rd) := reg(rs) + reg(rt);
and (R,0,36)	and rd,rs,rt	reg(rd) := reg(rs) & reg(rt);
and immediate (I,12,na)	andi rt,rs,imm	reg(rt) := reg(rs) & zeroext(imm);
branch on equal (I,4,na)	beq rs,rt,label	if reg(rs) == reg(rt) then PC = BTA else NOP;
branch on not equal (I,5,na)	bne rs,rt,label	if reg(rs) != reg(rt) then PC = BTA else NOP;
jump and link register (R,0,9)	jalr rs	\$ra := PC + 4; PC := reg(rs);
jump register (R,0,8)	jr rs	PC := reg(rs);
jump (J,2,na)	j label	PC := JTA;
jump and link (J,3,na)	jal label	\$ra := PC + 4; PC := JTA;
load byte (I,32,na)	lb rt,imm(rs)	reg(rt) := signext(mem[reg(rs) + signext(imm)] _{7:0});
load byte unsigned (I,36,na)	lbu rt,imm(rs)	reg(rt) := zeroext(mem[reg(rs) + signext(imm)] _{7:0});
load upper immediate (I,14,na)	lui rt,imm	reg(rt) := concat(imm, 16 bits of 0);
load word (I,35,na)	lw rt,imm(rs)	reg(rt) := mem[reg(rs) + signext(imm)];
multiply, 32-bit result (R,28,2)	mul rd,rs,rt	reg(rd) := reg(rs) * reg(rt);
nor (R,0,39)	nor rd,rs,rt	reg(rd) := not(reg(rs) reg(rt));
or (R,0,37)	or rd,rs,rt	reg(rd) := reg(rs) reg(rt);
or immediate (I,13,na)	ori rt,rs,imm	reg(rt) := reg(rs) zeroext(imm);
set less than (R,0,42)	slt rd,rs,rt	reg(rd) := if reg(rs) < reg(rt) then 1 else 0;
set less than unsigned (R,0,43)	sltu rd,rs,rt	reg(rd) := if reg(rs) < reg(rt) then 1 else 0;
set less than immediate (I,10,na)	slti rt,rs,imm	reg(rt) := if reg(rs) < signext(imm) then 1 else 0;
set less than immediate unsigned (I,11,na)	sltiu rt,rs,imm	reg(rt) := if reg(rs) < signext(imm) then 1 else 0;
shift left logical (R,0,0)	sll rd,rt,shamt	reg(rd) := reg(rt) << shamt;
shift left logical variable (R,0,4)	sllv rd,rt,rs	reg(rd) := reg(rt) << reg(rs _{4:0});
shift right arithmetic (R,0,3)	sra rd,rt,shamt	reg(rd) := reg(rt) >>> shamt;
shift right logical (R,0,2)	srl rd,rt,shamt	reg(rd) := reg(rt) >> shamt;
shift right logical variable (R,0,6)	srlv rd,rt,rs	reg(rd) := reg(rt) >> reg(rs _{4:0});
store byte (I,40,na)	sb rt,imm(rs)	mem[reg(rs) + signext(imm)] _{7:0} := reg(rt) _{7:0} ;
store word (I,43,na)	sw rt,imm(rs)	mem[reg(rs) + signext(imm)] := reg(rt);
subtract (R,0,34)	sub rd,rs,rt	reg(rd) := reg(rs) - reg(rt);
subtract unsigned (R,0,35)	subu rd,rs,rt	reg(rd) := reg(rs) - reg(rt);
xor (R,0,38)	xor rd,rs,rt	reg(rd) := reg(rs) ^ reg(rt);
xor immediate (I,14,na)	xori rt,rs,imm	reg(rt) := reg(rs) ^ zeroext(imm);

Definitions

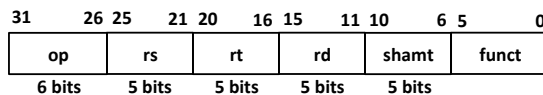
- Jump to target address: JTA = concat((PC + 4)_{31:28}, address(label), 00₂)
- Branch target address: BTA = PC + 4 + imm * 4

Clarifications

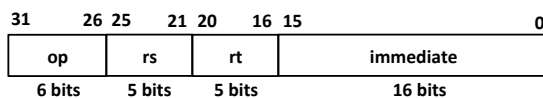
- All numbers are given in decimal form (base 10).
- Function signext(x) returns a 32-bit sign extended value of x in two's complement form.
- Function zeroext(x) returns a 32-bit value, where zero are added to the most significant side of x.
- Function concat(x, y, ..., z) concatenates the bits of expressions x, y, ..., z.
- Subscripts, for instance X_{8:2}, means that bits with index 8 to 2 are spliced out of the integer X.
- Function address(x) means the address of label x.
- NOP and na means "no operation" and "not applicable", respectively.
- shamt is an abbreviation for "shift amount", i.e. how much bit shifting that should be done.

INSTRUCTION FORMAT

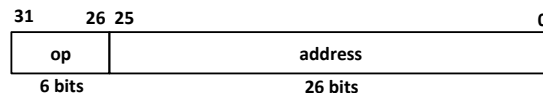
R-Type



I-Type



J-Type



REGISTERS

Name	Number	Description
\$zero	0	constant value 0
\$at	1	assembler temp
\$v0	2	function return
\$v1	3	function return argument
\$a0	4	argument
\$a1	5	argument
\$a2	6	argument
\$a3	7	argument
\$t0	8	temporary value
\$t1	9	temporary value
\$t2	10	temporary value
\$t3	11	temporary value
\$t4	12	temporary value
\$t5	13	temporary value
\$t6	14	temporary value
\$t7	15	temporary value
\$s0	16	saved temporary
\$s1	17	saved temporary
\$s2	18	saved temporary
\$s3	19	saved temporary
\$s4	20	saved temporary
\$s5	21	saved temporary
\$s6	22	saved temporary
\$s7	23	saved temporary
\$t8	24	temporary value
\$t9	25	temporary value
\$k0	26	reserved for OS
\$k1	27	reserved for OS
\$gp	28	global pointer
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address

MIPS Reference Cheat Sheet

By David Broman
KTH Royal Institute of Technology

If you find any errors or have any feedback on this document, please send me an email: dbro@kth.se