NORTH SOUTH UNIVERSITY

Computer Organization & Design

CSE 332

ISA DESIGN

Faculty Initialize: SAS3

Sec: 12

Group Members:

Name	ID
Md Badiuzzaman Pranto	151 22360 42
Md Nazrul Islam Safa	151 24930 42
Akif Arshad Chowdhury	151 17286 42
Rakibul Islam	151 10406 42

Introduction:

Instruction set architecture (ISA) is an abstract model of computer. Our task is to design and implement an assembler for 16-bit RISC type CPU.

Objective:

After converting a high level programming language into assembly language by a compiler, the assembler takes the assembly language as input and convert into a binary instruction (machine language) for hardware as 16 bit output.

Operands:

By following the design principle 1, we have used 3 type of operands in our design. They are mainly register based.

Format

R-Type

Op-Code 4 bit	rs 4 bit	rt 4 bit	rd 4 bit

I-Type

Op-Code 4 bit	rs 4 bit	rt 4 bit	Immediate 4 bit

J-Type

Op-Code 4 bit	Target Address 12 bit

Design principle 3 says, good design requires compromises. In our Structure we have chosen 3 formats. If we deal with constant (like a = a + 4), we can’t deal with R-Type format. So we need an immediate type format. For conditional statements, sometime we need to jump to a particular line or need to out of a loop. In that case, we need J-Type format.

Registers

Register Number	Name of the Registers	Usage	Value Assigned (4 bit)
0	$zero	Hard-wired to 0 (constant value 0)	0000
1	$t0	Temporary	0001
2	$t1	Temporary	0010
3	$t2	Temporary	0011
4	$t3	Temporary	0100
5	$s0	Save	0101
6	$s1	Save	0110
7	$s2	Save	0111
8	$s3	Save	1000
9	$s4	Save	1001
10	$s5	Save	1010
11	$gp	Global Pointer	1011
12	$sp	Stack Pointer	1100
13	$fp	Frame Pointer	1101
14	$ra	Return Address	1110
15	$a0	Arguments to functions	1111

Operations

R-Type Table

Instruction Type	Instruction	Op-Code
Arithmetic	add	0000
	sub	0001
Logical	and	0010
	or	0011
	nor	0100
	xor	1110

I-Type Table

Instruction Type	Instruction	Op-Code
Data Transfer	lw	0101
	sw	0110
Conditional Branch	beq	0111
	mul	1000
Arithmetic	addi	1001
Logical	andi	1010
	ori	1011
	sll	1100
	srl	1101

J-Type Table

Instruction Type	Instruction	Op-Code
Unconditional Jump	J	1111

Operations' Instructions

add:

It adds the content of the source register 1 to the contents of the source register 2 and saves it in the destination register.

Operation: $\$s_0 = \$s_0 + \$t_0$

Syntax: add $\$s_0$, $\$s_{0,}$ $\$t_0$

sub:

It subtracts the content of the source register 2 from the contents of the source register 1 and saves it in the destination register.

Operation: $\$s_0 = \$s_0 - \$t_0$

Syntax: sub $\$s_0$, $\$s_{0,}$ $\$t_0$

and:

It does a bit by bit logical and operation between two source registers contents.

Operation: $\$s_0 = \$s_0 \& \$t_0$

Syntax: and $\$s_0$, $\$s_{0,}$ $\$t_0$

or:

It does a bit by bit logical or operation between two source registers contents.

Operation: $\$s_0 = \$s_0 \,||\, \$t_0$

Syntax: or $\$s_0$, $\$s_0$, $\$t_0$

nor:

It does a bit by bit logical “nor” operation between two source registers contents.

Operation: $\$s_0 = \sim (\$s_0 \mid \$t_0)$

Syntax: nor $\$s_0$, $\$s_0$, $\$t_0$

slt:

It compares the contents of two source registers. If the content of 1st source register is less than the second source register, then it sets the value of destination register to 1, otherwise 0

Operation: if ($\$s_0 < \$t_0$)

$\$s_0 = 1$

Else $\$s_0 = 0$

Syntax: slt $\$s_0$, $\$s_0$, $\$t_0$

lw:

It loads the contents of the memory specified by the offset and saves it into a destination register.

Operation: $\$s_0$ = Memory [$\$s_1$ + offset]

Syntax: lw $\$s_0$, 2($\$s_1$)

sw:

It stores the contents of a source register to the memory address specified by the offset.

Operation: Memory [$\$s_1$ + offset] = $\$s_0$

Syntax: sw $\$s_0$, 2($\$s_1$)

beq:

It checks if the content of the provided register is equal to the contents of another register or not. If equal jumps a relative number of instructions else continue.

Operation: if ($\$s_0$==$\$t_0$) jump to L (Label)

Else go to next line

Syntax: beq $\$s_0$, $\$t_{0,}$ L

bne:

It checks if the content of the provided register is not equal to the contents of another register or not. If not equal, jumps a relative number of instructions else continue.

Operation: if ($\$s_0$!= $\$t_0$) jump to L (Label)

Else go to next line

Syntax: bne $\$s_0$, $\$t_{0,}$ L

addi:

It can add direct value (constant) with a content of a source register.

Operation: $\$s_0$ = $\$t_0$ + immediate value

Syntax: addi $\$s_0$, $\$t_0$, 4

andi:

It does logical bit by bit and operation with a constant value and a content of a source register.

Operation: $\$s_0$ = $\$t_0$ & immediate value

Syntax: andi $\$s_0$, $\$t_0$, 4

ori:

It does logical bit by bit or operation with a constant value and content of a source register.

Operation: $\$s_0$ = $\$t_0$ || immediate value

Syntax: ori $\$s_0$, $\$t_0$, 4

srl:

This operation shift the content of a register to right by a constant and stores shifted value to destination register.

Operation: $\$s_0 = \$t_0 >> 4$

Syntax: srl $\$s_0$, $\$t_0$, 4

sll:

This operation shift the content of a register to left by a constant and stores shifted value to destination register.

Operation: $\$s_0 = \$t_0 << 4$

Syntax: sll $\$s_0$, $\$t_0$, 4

J:

It jumps a relative number of instruction/s specified by given label number.

Operation: jump to Label

Syntax: J, L

Limitations

Our operations are limited because of limited instruction size for CPU. And user must provide valid instructions in the correct manner in order to decode and execute instructions and must strictly follow syntax rules. Each instruction is separated by a new line and user must follow spacing rules.