16/20

ISA DESIGN

GROUP-8

Member 1: Mostafa Mushsharat 1931298042

Member 2: Tabassum Sadia Shahjahan 1931819042

Member 3: Mohammed Ahnaf Abid 1931897642

Member 4: Md. Samiur Rahman 1813598042

Member 5: Tajul Islam Sumon 1721135042

Group-8

ISA Design

What is an ISA?

An ISA is an instruction set architecture which is a model of a computer. It can also be called a computer architecture. When ISA is materialized in the form of a physical processor, it is called an implementation. An ISA defines the data and the registers to hold the data. It also defines how registers communicate with the main memory. It also defines how data is going to be taken as input as well as how data is going to be given out as output.

Brief Overview:

- We are constructing a 16-bit CPU.
- Every instruction is going to be 16-bit wide.
- Our design is based on RISC architecture because it utilizes a small set of instructions and uses lots of registers.
- Every logical and arithmetic operations involve the use of registers.
- Two main types of operations are going to be used: 1) Memory access (load and store between memory and registers) 2) ALU Operations (which only occur between registers).

Questions:

1) How many operands?

Answer: 3 operands

2) Types of operand? (Register based?? Memory-based? Mixed?)

Answer: Mixed. Operands deal with both read/write to register and read/write to memory.

3) How many operations? why?

Answer: 16 operations. This is because $2^4 = 16$. So, each op-code can be uniquely identified with 4-bit binary values. As such, all of these operations would be enough to meet the requirements of the benchmark programs. Provided the fact that the ISA can run basic arithmetic operations as well as logical operations.

4) Types of operations? (Arithmetic, logical, branch type?? How many from each category? Draw a table with list of instructions, instruction type, their opcode, functionality (if any).

Answer:

Operation Type	Number of Op-codes	Operations list
Arithmetic	3	ADD, SUB, ADDi
Logical	6	AND, ANDi, OR, NOR, SLL, SRL
Branch	3	BEQ, BLT, JUMP
Data Transfer	4	LW, SW, INPUT, OUTPUT

Opcode	Example	Meaning	Comments
ADD	ADD $rd, $rs, $rt	$rd=$rs+$rt	Adding values from second register and third register and storing it in the first register.
SUB	SUB $rd, $rs, $rt	$rd=$rs-$rt	Subtracting value of third register from second register and storing it in the first register.
ADDi	ADDi $rd, $rs, 30	$rd=$rs+constant	Adding a constant to second register and storing it in the first register.

AND	AND $rd, $rs, $rt	$rd=$rs & $rt	Bitwise AND function is performed on second and third register and result is stored in first register.
ANDi	ANDi $rd, $rs, 30	$rd=$rs & constant	Bitwise AND function is performed on second register and a constant and result is stored in first register.
OR	OR $rd, $rs, $rt	$rd=$rs | $rt	Bitwise OR function is performed on second and third register and result is stored in first register.
NOR	NOR $rd, $rs, $rt	$rd=~($rs | $rt)	Bitwise NOR function is performed on second and third register and result is stored in first register.
SLL	SLL $rd, $rs, 2	$rd=$rs<<2	Shift left by constant; used for multiplication.
SRL	SRL $rd, $rs, 2	$rd=$rs>>2	Shift right by constant; used for division.
LW	LW $rd, 20($rs)	$rd=Memory[$rs+20]	Load word from memory to register.
SW	SW $rd, 20($rs)	Memory[$rs+20]=$rd	Load word from register to memory.
INPUT	INPUT $rd, ~~$rs, 0~~	$rd= ~~$rs+0~~ $bufferin	Loading value of input buffer ($rs) to register.
OUTPUT	OUTPUT $rd, ~~$rs, 0~~	~~$rs+0~~=$rd $bufferout	Storing value from register to output buffer ($rd).

BEQ	BEQ $rd, $rs, 30	If($rd==$rs) go to LABEL+30x2	If registers are equal, branch relative to LABEL.
BLT	BLT $rd, $rs, 30	If($rd<$rs) go to LABEL +30x2	If first register is less than second register, branch relative to LABEL.
JUMP	JUMP 2500	go to LABEL	Jump to address LABEL.

Opcode	Binary Value
ADD	0000
SUB	0001
ADDi	0010
AND	0011
ANDi	0100
OR	0101
NOR	0110
SLL	0111
SRL	1000
LW	1001
SW	1010
INPUT	1011
OUTPUT	1100
BEQ	1101
BLT	1110
JUMP	1111

5) How many formats would you choose? Draw the formats along with field name and number of bits in each field.

Answer:

We are choosing 3 formats which are R-Type, I-Type and J-Type.

R-Type:

Op-code	rd	rs	rt
4	4	4	4

I-Type:

Op-code	rd	rs	Immediate
4	4	4	4

J-Type:

Op-code	Address
4	12

6) list of registers? Draw a register table. (with register name and values)

Answer:

Inside the ALU:

Register Number	Register Name	Description
0	$ZERO	It has a fixed value of 0.
1-8	$s0-$s7	Save variables
9-15	$t0-$t6	Temporary variables

~~Outside the ALU:~~

~~Register Number~~	~~Register Name~~	~~Description~~
~~0-15~~	~~$i0-$i15~~	~~Input Buffer Variables~~
~~0-15~~	~~$u0-$u15~~	~~Output Buffer Variables~~

7) Addressing Modes.

Answer:

1) Base Addressing Mode (Relative to LABEL, so LABEL is the base)
2) Immediate Addressing Mode
3) Register addressing mode

you must provide the diagrams of addressing mode

Guideline Questions:

1. Ability to execute the provided benchmark programs, and other general-purpose programs?

Answer: Since it can read and write to memory, as well as take inputs from user and provide outputs, it can perform basic arithmetic functions for the user such programs involving numeric calculations. It can perform addition, subtraction, division and multiplication.

A simple calculator can be constructed where the user-input values are stored and then an arithmetic function is performed on it and then a suitable output is displayed on the display such as a Seven-segment display.

Loops can also be implemented with the provided branch operations so any kind of programs involving recurring numbers can be done.

Since it can perform logical functions, conditional statements can be executed and a result can be displayed on the screen.

2. How much is it different from existing ISAs such as MIPS?

Answer:

Differences compared to MIPS:

1) If we want to run 32-bit instructions, then we need to break them down. In that case, software emulation is required as the 32-bit instructions need to be broken down for our 16-bit processor to process.
2) Less number of operations are available, specifically 16 operations, so there is less functionality compared to MIPS ISA.
3) If we want to perform operations outside our 16-bit ISA, we have to introduce more complexities into the compiler design.
4) Every instruction for our 16-bit ISA is of 16-bits in length but MIPS instructions are 32-bit in length so less data can be processed at any given time compared to the MIPS ISA.
5) Our ISA has only a total of 16 (4-bit) opcodes but the MIPS ISA allows for up to 64 (6-bit) opcodes.

3. How long does it take to run benchmark programs on your processor?

Answer:

Suppose we took a simple C++ program:

```
#include <iostream>
int main{
        int A=17, B=18, flag=1;
         if(A<B){
           cout<<flag;
          }
    return 0;
}
```

Lets assume $s5=flag

MIPS Assembly code:

```
Addi $s0, $zero, 17
Addi $s1, $zero, 18
 blt $s0, $s1, 4
 Addi $s5, $zero, 1
 print_int $s5
```

Our 16-bit ISA:

```
ADDi $s0, $ZERO, 15
ADDi $s1, $ZERO, 2
ADDi $s2, $ZERO, 15,
ADDi $s3, $ZERO, 3,
   BLT $s1, $s3, 6
   ADDi $s5, $ZERO, 1
   OUTPUT $u0, $s5, 0
```

Therefore, MIPS required 5 instruction counts whereas our ISA required 7 instruction counts since our ISA is only limited to a 16 bit architecture and each register can only hold a maximum denary value of 15.