# Introduction:

This is a project for CSE332: Computer Organization and Architecture. our task to design to design a 16 bit single-cycle CPU that has separate Data and Instruction Memory.

# For Design :

1. How many operands?

* In our design ISA specify 0-3 operands for per instruction.

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

* In our design we use register type operands , memory based operands and also mixed operands .

3)How many operations? why?

* IN our design we have 17 operation . They are ADD ,SUB, AND, OR, NOR, DOUT, SLT, SLL, SRL, ADDI, J ,BEQ , LW ,SW , BNE , SLTI , DIN.

add: three register operands ,It adds two registers and stores the result in destination register.

* Operation: $t0 = $s 1+ $s2
* Syntax: add $t0, $s1, $s2

sub: three register operands ,It subtracts two registers and stores the result in destination register.

* Operation: $t0 = $s1 - $s2
* Syntax: sub $d, $s, $t

addi: used to add constant ,It adds a value from register with an integer value and stores the result in destination register.

* Operation: $t0 = $s1 + offset
* Syntax: addi $t0, $s1, offset

lw: word from memory to register .It loads required value from the memory and write it back into the register.

* Operation: $to= MEM[$s1+ offset]
* Syntax: lw $to, offset($s1)

sw: word from register to memory .It stores specific value from register to memory.

* Operation: MEM[$s1 + offset] = $s2
* Syntax: sw $s1, offset($s2)

and: three register operands , bit by bit AND .It AND’s two register values and stores the result in destination register.

* Operation: $t0 = $s1 && $s2
* Syntax: and $t0, $s1, $s2

or: three register operands , bit by bit OR . It OR’s two register values and stores the result in destination register.

* Operation: $t0=$s1 || $s2
* Syntax: or $t0, $s1, $s2

nor: three register operands , bit by bit NOR . It NOR’s two register values and stores the result in destination register. Sometimes we use nor to get NOT of register value.

* Operation: $t0=$s1 nor $s2
* Syntax: nor $t0, $s1, $s2

Sll : Shift left by constant , It shifts bits to the left and fill the empty bits with zeros. The shift amount is depended on the offset value.

* Operation: $t0= $s1 << offset
* Syntax: sll $t0, $s1, offset

srl: Shift right by constant, It shifts bits to the right and fill the empty bits with zeros. The shift amount is depended on the offset value.

* Operation: $t0= $s1>> offset
* Syntax: srl $t0, $s1, offset

beq: equal test pc related branch ,It checks whether the values of two register s are same or not. If it’s same it performs the operation located in the address at offset value.

* Operation: if ($s1==$s2) jump to offset

else goto next line

* Syntax: beq $s1, $s2, offset

bne: Not equal test pc related branch It checks whether the values of two register s are same or not. If it’s not same it performs the operation located in the address at offset value.

* Operation: if ($s1!=$s2) jump to offset

else goto next line

* Syntax: bne $s1, $s2, offset

Slt: Compare less than for beq bne , If $s is less than $t, $d is set to one. It gets zero otherwise.

* Operation: if $s1< $s2 = 1

else $t0 = 0

* Syntax: slt $t0, $s1, $s2

J: Jumps to the target address.

* Operation : J L1
* Syntax : Go to L1

D

## Dout :

For output.

## Din :

For input .

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).

=> types of operations :

1) Arithmetic : add , sub , addi , ( 3 category )

2) Logical : and , or , nor , dout ,slt , sll , srl , slti ,din ( 9 category )

3) Conditional : beq , bne ( 2 category )

4) Unconditional : j ( 1 category )

5) Data transfer : lw, sw ( 2 category )

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Type | Format | Details | Op Code | Function |
| ADD | Arithmetic | Register | ADD $S0,$S1,$S2  $S0 = $S1 + $S2 | 0 | 0 |
| SUB | Arithmetic | Register | SUB $S0,$S1,$S2  $S0 = $S1 - $S2 | 0 | 1 |
| AND | Logical | Register | AND $S0,$S1,$S2  $S0 = $S1 & $S2 | 0 | 2 |
| OR | Logical | Register | OR $S0,$S1,$S2  $S0 = $S1 | $S2 | 0 | 3 |
| NOR | Logical | Register | AND $S0,$S1, $S2  $S0 = ~ ($S1 | $S2) | 0 | 4 |
| Dout | Logical | Immediate | Dout $t0  Display $t0 |  |  |
| SLT | Logical | Register | SLT $S0, $S1,$S2  if ($S1 < $S2) $S0 = 1  Else $S0 = 0 | 0 | 5 |
| SLL | Logical | Register | SLL $S0,$S1,10  $S0 = $S1 << 10 | 0 | 6 |
| SRL | Logical | Register | SRL $S0,$S1,10  $S0 = $S1 >> 10 | 0 | 7 |
| ADDi | Arithmetic | Immediate | ADDi $S0,$S1, 20  $S0 = $S1 + 20 | 1 | XXXX |
| J | Unconditional | Target/M | J L1  Go to L1 | 2 | XXXX |
| BEQ | Conditional | Conditional | BEQ $S0,$S1,25 (not clear in paper) | 3 | XXXX |
| LW | Data Transfer | Immediate | LW $S0,20($S1)  $S0 = mem[$S1 + 20] | 4 | XXXX |
| SW | Data Transfer | Immediate | SW $S0,20($S1)  mem[$S1 + 20]= $S0 | 5 | XXXX |
| BNE | Conditional | Immediate | BEN $S0, $S1,25  if ($S1 != $S2)  Else  (not clear in paper) | 6 | XXXX |
| SLTi | Logical | Immediate | SLT $S0,$S1,20  if ($S1 < 20) $S0 = 1  Else $S0 = 0 | 7 | XXXX |
| Din |  | Immediate | Din $t0 5  $t0 = 5 | 8 | XXXX |

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

* In our design we use three format register format , immediate formats and target formats

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opcode** | **Rs** | **rt** | **rd** | **function** |

4bit 3bit 3bit 3bit 3bit

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | **rs** | **rt** | **Immediate** |

4bit 3 bit 3bit 6bit

|  |  |
| --- | --- |
| **Opcode** | **Target** |

4bit 12 bit

Instruction fields for Register formate :

Opcode : operation code .(4bit)

rs: first source register number . ( 3bit )

rt: second source register number . ( 3bit )

rd: destination register number . ( 3bit )

function : function code (extends opcode) . ( 3bit )

# instruction fields for immediate formate :

Opcode : operation code .(4bit)

rt: destination register number .(3bit)

rs: register number with address . ( 3 Bit)

Constant/ Address(immediate) : offset added to base address in rs (6 Bit).

# Instruction fields for target formate :

Opcode : operation code .(4bit)

Target : 12 bit .

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

* In our design ,We have only registers are $s0-$s3, $t0-$t3

|  |  |  |
| --- | --- | --- |
| Register Number | Conventional name | Value of register (3bit) |
| $1 | $s0 | 000 |
| $2 | $s1 | 001 |
| $3 | $s2 | 010 |
| $4 | $s3 | 011 |
| $ 5 | $t0 | 100 |
| $6 | $t1 | 101 |
| $7 | $t2 | 110 |
| $8 | $t3 | 111 |

7) Addressing Modes :

In our design we use :

* Register addressing modes
* Immediate addressing modes
* Base addressing modes
* Pc- relative addressing modes
* Indirect addressing modes
* Direct addressing modes

# Important Description :

In our ISA design , it has ability to execute the Simple arithmetic & logic operations , Programs that require checking conditions , Loop type of programs and other general purpose programs .

As like

C code : A = B + C

* Add $t0 , $s1 , $s2

C code : A = B|| C

* Or $t0 , $s1 , $s2

C Code : if (i==j) f = g+h;

Else f = g-h;

* bne $s2, $s3, Else

add $t0, $s1, $s0

j Exit

Else: sub $t0, $s1, $s0

C code : for(i= 0 ,i<j ,i++)

A = b+c

* Sub $t0 , $s0,$s0

Loop : Beq $t0 , $s0 exit

Add $t1 , $s1 ,$s2

Addi $t0 ,$t0,1

J loop

Exit ……..

But there are many difference between existing Isa and Mips , existing Isa is 16 bit , but Mips has 32 bit ,Mips has 6 bit Opcode and function but our existing Isa has 4 bit opcode and 3 bit function , Mips has 5 bit rs rd rt ,but this Isa has only 3 bit rs rd rt , that means we have less register than mips ,we have also less function ,opcode ,operation than Mips.