We have 2 registers in our cpu design: one register is used to save temporary value named “$t”. The other is used to save value that need to be used in the future named “$s”

We have exactly 16 instructions so that we reserve 4-bits for operation code.

R-type:

General instruction format:

|  |  |  |  |
| --- | --- | --- | --- |
| Op  4 | Extra Space  3 | Addr.  8 | Register  1 |

When the value of “Register” equals 0, we use register $t.

When the value of “Register” equals 1, we use register $s.

Add: add Addr. Reg add the value in an address of a memory to the value in selected register, and then save the calculated value in that register. Opcode: 00

And: and Addr. Reg do “and” operation between the value in an address of a memory and the value in register, and then save the calculated value in register. Opcode: 01

Or: or Addr. Reg do “or” operation between the value in an address of a memory and the value in register, and then save the calculated value in register.

Opcode: 02

Sub: sub Addr. Reg subtract the value in an address of a memory from the value in register, and then save the calculated value in register. Opcode: 03

Slt: slt Addr. Reg Compare the value in an address of a memory with the value in register, and if the value in register is smaller, set the value of register to 1, otherwise, set it to 0 Opcode: 04

J: j Addr. Unconditionally jump to the instruction whose address is in the register. Opcode: 05

Lr: lr Reg load the value of an address that stored in a register in to that register. Opcode: 07

Sw: sw Addr. Reg Store the 16 bit word of a register into at an address of memory. Opcode: 08

Jal: jal Addr. Reg Unconditionally jump to the instruction at the specified address. Save the address of the next instruction in the register. Opcode: 09

Beq: beq DO Compare the values in two registers, if they are equal, branch. Opcode: 0A

Bne: bne DO C Compare the values in two registers, if they are not equal, branch. Opcode: 0B

I-type:

|  |  |  |
| --- | --- | --- |
| Op  4 | Imm  11 | Register  1 |

Addi: addi Imm. Reg add the immediate value to the value in register, and then save the calculated value in register. Opcode: 0C

Andi: andi Imm. Reg do “and” operation between immediate value and the value in register, and then save the calculated value in register. Opcode: 0D

Ori: ori Imm. Reg do “or” operation between immediate value and the value in register, and then save the calculated value in register. Opcode: 0E

Lui: lui Imm. Reg Set the first first 8 bits of the value in a register to the last 8 bits of an immediate. Opcode: 0F

Li: li Imm. Reg. load an immediate to a register. Opcode: 06

Euclid’s algorithm

BMinus:

lui b(upper) $t

ori b(lower) $t

lr $t

sub a $t

sw b $t

j Loop

Exita:

lui a(upper) $s

ori a(lower) $s

lr $s

j return

Exitb:

lui b(upper) $s

ori b(lower) $s

lr $s

j return

Assembly

lui a(upper) $t

ori a(lower) $t

lr $t

li 0 $s

beq Exitb

Loop:

lui b(upper) $t

ori b(lower) $t

lr $t

beq Exita

slt a $t

bne AMinus

beq BMinus

AMinus:

lui a(upper) $t

ori a(lower) $t

lr $t

sub b $t

sw a $t

j Loop

int gcd(int a, int b){

if(a==0){

return b;

}

while(b!=0){

if(a>b){

a=a-b;

}else{

b=b-a;

}

}

return a;

}

C Code

Load/Store word&Exception

lui a(upper) $t

ori a(lower) $t

lr $t

addi 2 $t

sw a $t

li 0 $s

addi 5 $s

sw b $s

li 0 $s

slt b $t

bne Num

beq Leave

Num:

lui a(upper) $t

ori a(lower) $t

lr $t

addi 10 $t

sw a $t

Leave:

lui a(upper) $t

ori a(lower) $t

lr $t

addi 0x8fff $t

addi -10 $t

sw a $t

Assembly Code

int loadsave(int a){

a= a+2;

if(a<5){

a=a+10;

}

a = a+0x8fff;

a = a - 10

}

C Code

Load the value from an address

lui a(upper) $t

ori a(lower) $t

lr $t

We load a memory address to a register and then load the value to the register itself.

Iteration

Loop:

“content of loop”

beq address Exit

j Loop

Exit:

“exit code”

This is a loop that makes the program goes over and over until the value in the register equals to the value in the specified memory address. When they equal, the program will go to the “Exit” address and finish the loop.

Conditional Statement

beq(bne) address Target

j Else

Target:

“target content”

Else:

“else content”

This is an if statement. If the value of the register equals(not equal) to the data value in the specified address, then executes specified code in the “Target” address. Else, it will executes codes in the “Else” address.

Reading Data from Input Port: Undecided

Reading/Writing from Display Register: Undecided

Writing to Output Port: Undecided

**Euclid’s algorithm**

lui a(upper) $t

1111 000AAAAAAAA 0

ori a(lower) $t

1110 000AAAAAAAA 0

lr $t

0111 XXX XXXXXXXX 0

li 0 $s

0110 00000000000 1

beq Exitb

1010 XXX EXITBBBB X

Loop:

lui a(upper) $t

1111 000BBBBBBBB 0

ori a(lower) $t

1110 000BBBBBBBB 0

lr $t

0111 XXX XXXXXXXX 0

beq Exita

1010 XXX EXITAAAAA X

slt a $t

0100 XXX AAAAAAAA 0

bne AMinus

1011 XXX AMINUSSS X

beq BMinus

1010 XXX BMINUSSS X

AMinus:

lui a(upper) $t

1111 000BBBBBBBB 0

ori a(lower) $t

1110 000BBBBBBBB 0

lr $t

0111 XXX XXXXXXXX 0

sub b $t

0011 XXX BBBBBBBB 0

sw a $t

1000 XXX AAAAAAAA 0

j Loop

0101 XXX LOOPPPPP X

BMinus:

lui b(upper) $t

1111 000BBBBBBBB 0

ori b(lower) $t

1110 000BBBBBBBB 0

lr $t

0111 XXX XXXXXXXX 0

sub a $t

0011 XXX AAAAAAAA 0

sw b $t

1000 XXX BBBBBBBB 0

j Loop

0101 XXX LOOPPPPP X

Exita:

lui a(upper) $t

1111 000AAAAAAAA 0

ori a(lower) $t

1110 000AAAAAAAA 0

lr $t

0111 XXX XXXXXXXX 0

j return

0101 XXX RETURNNN X

Exitb:

lui b(upper) $t

1111 000BBBBBBBB 0

ori b(lower) $t

1110 000BBBBBBBB 0

lr $t

0111 XXX XXXXXXXX 0

j return

0101 XXX RETURNNN X

Load/Store words & Interrupt

lui a(upper) $t

1111 000AAAAAAAA 0

ori a(lower) $t

1110 000AAAAAAAA 0

lr $t

0111 XXX XXXXXXXX 0

addi 2 $t

1100 00000000010 0

sw a $t

1000 XXX AAAAAAAA 0

li 0 $s

0110 0000000000 1

addi 5 $s

1100 00000000101 1

sw b $s

1000 XXX BBBBBBBB 1

li $s

0110 00000000000 1

slt b $t

0100 XXX BBBBBBBB 0

bne Num

1011 XXX NUMMMMMM X

beq Leave

1010 XXX LEAVEEEE X

Num:

lui a(upper) $t

1111 000AAAAAAAA 0

ori a(lower) $t

1110 000AAAAAAAA 0

lr $t

0111 XXX XXXXXXXX 0

addi 10 $t

1100 00000001010 0

sw a $t

1000 XXX AAAAAAAA 0

Leave:

lui a(upper) $t

1111 000AAAAAAAA 0

ori a(lower) $t

1110 000AAAAAAAA 0

lr $t

0111 XXX XXXXXXXX 0

addi 0x8fff $t

1100 errorrrrr 0

addi -10 $t

1100 11110110 0

sw a $t

1000 XXX AAAAAAAA 0