Assembler Design Proposal

CS17B001::Abhishek Kaushik & CS17B002::Abhishek Kumar

➔ Objective :- To develop an assembler for a selected set of instructions of

RISC type (MIPS - 32 bit).

➔Abstract :-

An **Assembler** is a program that turns assembly language into machine code. More generally, it is a program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. The programmer can write a program using a sequence of these assembler instructions. This sequence of assembler instructions, known as the source code or source program, is then specified to the assembler program when that program is started. The assembler program takes each program statement in the source program and generates a corresponding bit stream or pattern (a series of 0's and 1's of a given length). The output of the assembler program is called the object code or object program relative to the input source program. The sequence of 0's and 1's that constitute the object program is sometimes called machine code. The object program can then be run (or executed) whenever desired. Some basic instructions have been selected and specified below.

R - FORMAT

opcode rs rt rd Shift

(shamt)

funct

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

I-FORMAT

opcode rs rt IMM

6 bits 5 bits 5 bits 16 bits

J-FORMAT

opcode Pseudo-Address

6 bits 26 bits

➔Instruction Set Used :-

❏ Basic Arithmetic and Logic :-

● add :- add $1,$2,$3 => $1=$2+$3

add Add R 0x00 0x20

● subtract:- sub $1,$2,$3 => $1=$2-$3

sub Subtract R 0x00 0x22

● power :- pow $1,$2,$3 => $1 = ( $2 ) power ( $3 )

power Power R 0x00 0x3

● add immediate:- addi $1,$2,100 => $1=$2+100

addi Add Immediate I 0x08 NA

● add unsigned:- addu $1,$2,$3 => $1=$2+$3

addu Add Unsigned R 0x00 0x21

● subtract unsigned:- subu $1,$2,$3 => $1=$2-$3

subu Unsigned Subtract R 0x00 0x23

● multiply:- mult $2,$3 => $hi,$low=$2\*$3

mult Multiply R 0x00 0x18

● divide:- div $2,$3 => $hi,$low=$2/$3

div Divide R 0x00 0x1A

● and:- and $1,$2,$3 => $1=$2&$3

and Bitwise AND R 0x00 0x24

● Or:- or $1,$2,$3 => $1=$2|$3

or Bitwise OR R 0x00 0x25

● and immediate:- andi $1,$2,100 => $1=$2&100

andi Bitwise AND Immediate I 0x0C NA

● or immediate:- or $1,$2,100 => $1=$2|100

ori Bitwise OR Immediate I 0x0D NA

● shift left logical:- sll $1,$2,10 => $1=$2<<10

sll Logical Shift Left R 0x00 0x00

● shift right logical:- srl $1,$2,10 => $1=$2>>10

srl Logical Shift Right (0-extended) R 0x00 0x02

❏ Data Transfer:-

● load word:- lw $1,100($2) => $1=Memory[$2+100]

lw Load Word I 0x23 NA

● store word:- sw $1,100($2) => Memory[$2+100]=$1

sw Store Word I 0x2B NA

● load address:- la $1,label => $1=Address of label **PSEUDO INSTRUCTION**

● load immediate:- li $1,100 => $1=100 **PSEUDO INSTRUCTION**

● move from hi:- mfhi $2 => $2=hi

mfhi Move from HI Register R 0x00 0x10

● move from lo:- mflo $2 => $2=lo

mflo Move from LO Register R 0x00 0x12

❏ Conditional branch:-

● branch on equal:-beq $1,$2,100 => if($1==$2) go to PC+4+100

beq Branch if Equal I 0x04 NA

● branch on not equal:-bne $1,$2,100 => if($1!=$2) go to PC+4+100

bne Branch if Not Equal I 0x05 NA

❏ Unconditional branch:-

● Jump:- j 1000 => go to address 1000

j Jump to Address J 0x02 NA

❏ System Calls:-

syscall Generates a software interrupt 0x00 0x12

● print\_int:- Print integer number (32 bit) + 1(in $v0) => $a0 = integer to be printed

● print\_string:- Print null-terminated character string + 4(in $v0) => $a0 = address of string in memory

● print\_char:- Print character + 11(in $v0) => $a0 = character to be printed

● read\_int:- Read integer number from user + 5(in $v0) => Integer returned in $v0

● read\_string:- Same functionality as C fgets() + 8(in $v0) => $a0 = memory address of string input buffer $a1 = length of string buffer (n)

● read\_char:- Read character from user + 12(in $v0) => Char returned in $v0

● exit:- Stop program from running 10(in $v0)

➔Instruction Set Design:

● Size of instruction used will be 32 bit to accomodate large range of numbers.

● Addressing modes used will be Immediate(addi $r4, 108), Register(addr $r4, $r5 ) , Displacement (addd $r4, 20($r1)) , Direct (addm $r1, (25) and Register indirect (addmi $r2, ($r4)) .

● Endian-ness: Little endian as little endian machines store the byte with the highest value digits at the highest address.

● Registers Size : 32 bits will be used.

● Data and address size: The largest data represented by ISA will be of 16 bit including sign. The memory is byte addressable and the address space is of size 32 bit.

➔Feasibility Of instruction set:

● To accommodate the basic C program like swapping of two numbers, addition of array of elements, recursion we can use stack pointer and frame pointer. Register $30 is reserved for frame pointer.

➔New Features in ISA:

● We have included power function to output the power of a given value in register.