Hardware Instructions - II

S.Venkatesan
Network Security and Cryptography Lab
Department of Information Technology
Indian Institute of Information Technology, Allahabad
venkat@iiita.ac.in

Acknowledgement: The contents and figures are copied from various sources. Thanks to all authors and sources made those contents public and usable for educational purpose

Number System [Positional Numbers]

- Binary (0 and 1)
- Octal (0 to 7)
- Decimal (0 to 9)
- Hexa-decimal (0 to F that is upto 15)

Binary to Decimal

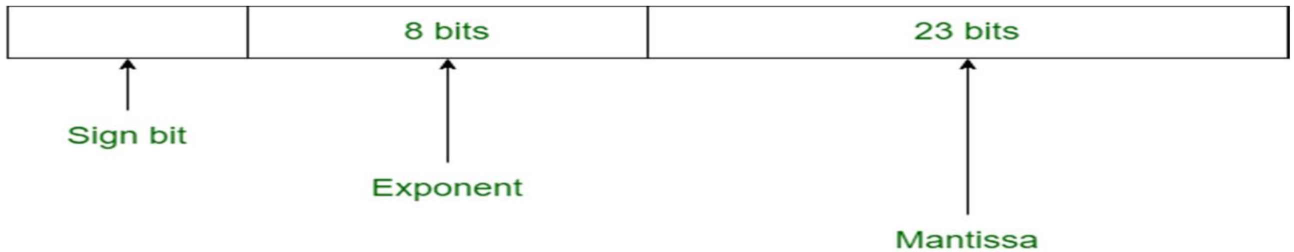
Decimal to Binary

Fractions

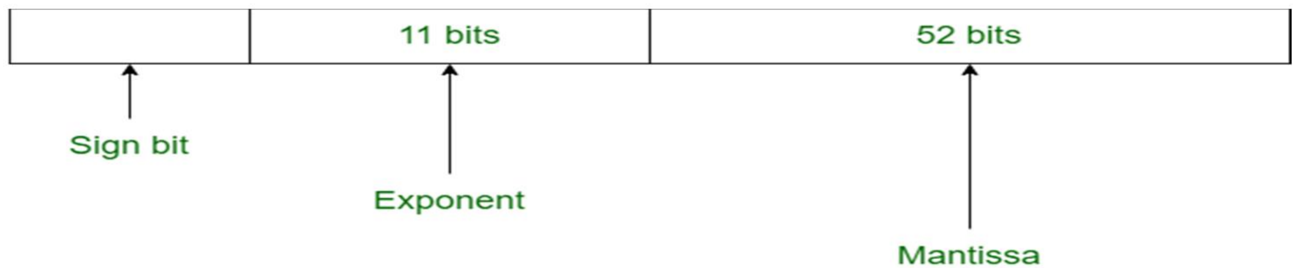
- Binary to Decimal – [.101 => $1X2^{-1} + 0X2^{-2} + 1X2^{-3}$]
- Decimal to Binary – [.23 => 0.23 X 2 = 0.46; 0.46 X 2 = 0.92; 0.92 X 2 = 1.84; 0.84 X 2 = 1.68; 0.68 X 2 = 1.36; 0.36 X 2 = 0.72;] - **Multiply it repeatedly by 2; Keep track of each integer part of the results; Stop when we get a fractional part that is equal to zero.**

Floating Point Numbers [IEEE 754]

Single Precision



Double Precision



Precision	**Base**	**Sign**	**Exponent**	**Significant**
Single precision	2	1	8	23 + 1 (implicit)
Double precision	2	1	11	52 + 1 (implicit)

The value of a normalized number is

$$(-1)^s \times 1.m \times 2^{e-127}$$

Example

Consider the following 32-bit pattern

1 1011 0110 011 0000 0000 0000 0000 0000

The value is

$$
\begin{aligned}
& (-1)^1 \times 2^{10110110-01111111} \times 1.011 \\
= & -1.375 \times 2^{55} \\
= & -49539595901075456.0 \\
= & -4.9539595901075456 \times 10^{16}
\end{aligned}
$$

Thanks to Gautam Biswas, IIT Kharagpur

Example

Consider the decimal number: $+105.625$. The equivalent binary representation is

$$\begin{aligned} & +1101001.101 \\ = & +1.101001101 \times 2^{6} \\ = & +1.101001101 \times 2^{133-127} \\ = & +1.101001101 \times 2^{10000101-01111111} \end{aligned}$$

In IEEE 754 format:

0 1000 0101 101 0011 0100 0000 0000 0000

Thanks to Gautam Biswas, IIT Kharagpur

Example

Consider the decimal number: +2.7. The equivalent binary representation is

$$\begin{aligned} & +10.10\ 1100\ 1100\ 1100\cdots \\ = & +1.010\ 1100\ 1100\cdots \times 2^{1} \\ = & +1.010\ 1100\ 1100\cdots \times 2^{128-127} \\ = & +1.010\ 1100\cdots \times 2^{10000000-01111111} \end{aligned}$$

In IEEE 754 format (approximate):

0 1000 0000 010 1100 1100 1100 1100 1101

Thanks to Gautam Biswas, IIT Kharagpur

Complexities and Solution

- Sign – Where to add (at left or right)
- Two’s complement

Signed and Unsigned numbers

- The Most significant bit tells the sign
- Use the Two's complement to find the value.

What is the decimal value of this 32-bit two's complement number?

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

Substituting the number's bit values into the formula above:

$$(1 \times -2^{31}) + (1 \times 2^{30}) + (1 \times 2^{29}) + \ldots + (1 \times 2^{1}) + (0 \times 2^{1}) + (0 \times 2^{0})$$
$$= -2^{31} + 2^{30} + 2^{29} + \ldots + 2^{2} + 0 + 0$$
$$= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten}$$
$$= -4_{ten}$$

- What about shortcut?
- Sign Extension (from 16 to 32) Shortcut [Add prefix zeros apply two's complement]

Instructions to Computer

Each piece of an instruction can be considered as an individual number.

ADD r5, r1, r2

MIPS representation in the following

add $t0,$s1,$s2

The decimal representation is

0	17	18	8	0	32

Each of these segments of an instruction is called a field.

- The first and last fields (containing 0 and 32 in this case) - addition.
- The second field gives the number of the register that is the first source operand of the addition operation(17 = $s1),
- The third field gives the other source operand for the addition (18 = $s2). The fourth field contains the number of the register that is to receive the sum (8 = $t0).
- The fifth field is unused in this instruction, so it is set to 0.

Instruction Format – Machine Language

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Fields

MIPS Fields

MIPS fields are given names to make them easier to discuss:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

ARM Fields

Cond – 4 bits (Condition Branch)
F – 2 bits (Allows ARM to different format)
I -1 bit (Immediate, if 0 then from register otherwise immediate)
Opcode – 4 bits
S – 1 bit (Condition)
Rn – 4 bits (first register source operand)
Rd – 4 bits (register destination operand)
Operand 2 – 12 bits

Example

ADD r3, r4, #4

14	0	1	4	0	4	3	4

Load and Store

Cond	F	Opcode	Rn	Rd	Offset
4 bits	2 bits	6 bits	4 bits	4 bits	12 bits

Cond	F	Opcode	Rn	Rd	Offset
4 bits	2 bits	6 bits	4 bits	4 bits	12 bits
14	1	24	3	5	32

Logical Operations

- Bit by Bit
  - AND
  - OR
  - NOT
  - Shift Left
  - Shift Right

Examples

ADD r5, r1, r2, LSL #2; r5 = r1 + (r2 <<2)
MOV r6, r5, LSR #4; r6 = r5 >> 4
MOV r6, r5, LSR #3; r6 = r5 >> 3

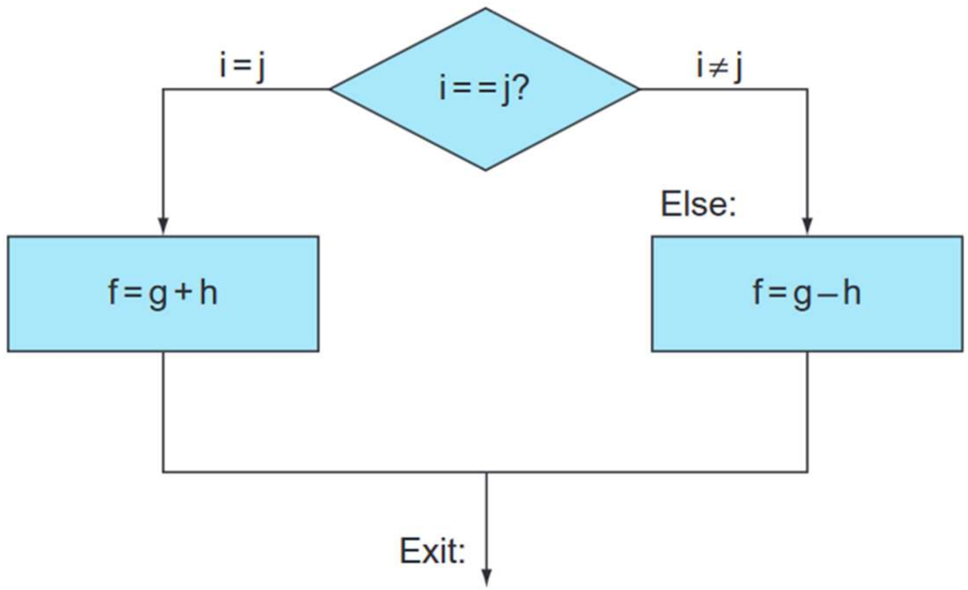
Rs – register shift length
Rm – second operand register

Cond	F	I	Opcode	S	Rn	Rd	Shift imm		Shift		Rm
							Rs	0	Shift		Rm
14	0	0	4	0	2	5	2 (immediate)		0	0	5
14	0	0	13	0	0	6	4 (immediate)		0	0	5
14	0	0	13	0	0	6	3 (register)	0	1	1	5

Insider Operand 2 [For shift operation]

11	8	7	6	5	4	3		0
Shift_imm			shift		0	Rm		
Rs		0	shift		1	Rm		

Making Decisions



CMP R1, R2
BEQ L1

CMP R1, R2
BNE L1

If(i==j) f = g+ h; else f = g – h

CMP r3, r4
BNE Else ; go to Else if i<>j

ADD r0, r1, r2
B Exit
ELSE: SUB r0, r1, r2
Exit

Loops

```
While (save[i] == k)
        i += 1;

Loop : ADD r12, r6, r3, LSL #2; r12 == address of save[i]

LDR r0, [r12,#0] ; r0 = save[i]

CMP r0, r5
BNE Exit ; go to Exit if save[i] <>k

ADD r3, r3, #1 ; i = i+1

B Loop: go to Loop
Exit
```

Basic Block – Just definition

Signed and Unsigned

Suppose register $s0 has the binary number

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two}$$

and that register $s1 has the binary number

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two}$$

Read $s as r in the above
CMP r0, r1
Condition to be taken are
BLO L1; unsigned branch – no because r0 greater than r1 lower unsigned instruction
BLT L2; signed branch – yes in case of signed

Bound check
BHS is the instructions

CMP r1, r2
BHS IndexoutofBounds ; if r1 > r2 or r1 < 0, goto error

Switch Statement

Method

- By performing *if-then-else*.
- Jump that is goto (Use Program counter)

Thank You