Arithematic and Logical Unit (ALU).

Arithematic Unit

A ⟶
B ⟶ ALU
Cin ⟶
⟶ $S_1 \, C_o$

S_1	S_o	Cin	F
0	0	0	$A+B$
0	0	1	$A+B+1$
0	1	0	$A+\bar{B}$
0	1	1	$A+\bar{B}+1$
1	0	0	A
1	0	1	$A+1$
1	1	0	$A-1$
1	1	1	A

(bit of binary of
1 and -1 are same)

$A+\bar{B} \rightarrow$ Subtraction with carry $A+\bar{B}+1$ Subtraction

Logical Unit

A
B
4x1
mux
$S_1 \, S_o$

Logical Unit.

S_1	S_o	F
0	0	AB
0	1	$A+B$
1	0	$A\oplus B$
1	1	$\bar{A}$

Arithmetic Logical Unit

A → ALU
B →
Cin →
P
S

→ arithmetic
→ logical operation
→ string operation

A → AU
B →
Cin →
$S_1 S_0$

A → LU
B →
$S_1 S_0$

Shift unit
$S_1 S_0$

$S_2 S_1$

MUX
$S_2 S_1$

S_3	S_2	S_1	S_0	Cin	F
0	0	0	0		A + B
0	0	0	1		A + B + 1
0	0	1	0		A + $\overline{B}$
0	0	1	1		A + $\overline{B}$ + 1
0	1	0	0		A
0	1	0	1		A + 1
0	1	1	0		A - 1
0	1	1	1		A
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0			

S_3	S_2	S_1	S_0	c_{in}	F
0	0	0	0	0	$A+B$
0	0	0	0	1	$A+B+1$
0	0	0	1	0	$A+\bar{B}$
0	0	0	1	1	$A+\bar{B}+1$
0	0	1	0	0	A
0	0	1	0	1	$A+1$
0	0	1	1	0	$A-1$
0	0	1	1	1	A
0	1	0	0	X	$A \wedge B$
0	1	0	1	X	$A \vee B$
0	1	1	0	X	$A \oplus B$
0	1	1	1	X	$\bar{A}$
1	0	0	0	X	$\cdot SW$
1	0	0	1	X	SW
1	0	1	0	X	SHA
1	0	1	1	X	BAR
1	1	X	X	X	0

Multiplication

16	8	4	2	1	
4	3	2	1	0	
		1	1	0	→ 6
		1	0	1	→ 5
		1	1	0	
	0	0	0	X	
1	1	0	X	X	
1	1	1	1	0	→ 30

1- Check the least significant bit of multiplier.
2- if 1, add multiplicant to product.
3- Shift multiply right
4- Shift multiplicant left

LECTURE 4.   28ᵗʰ Aug. 2019

Binary Multiplication :-

64 bit

Multiplicand

32 bits

Multiplier

1 sll   Shr
          1

Control unit

enable disable

Product 64 bit

Basic
multiplication
structure.

Shift

extended part →	0 0 0 0	1 0 1 1

```
            1 0  1 0
                    → Shift
    0 0 0 0 . 0 0  0 0
    0 0 0 1 0 1 1 0
    0 0 0 1 0 1 1 0
    0 0 0 0 0 0 0 0
    0 0 0 1 0 1 1 0
    0 1 0 1 1 0 0 0
    0 1 1 0 1 1 1 0
```

How we can optimize ?

• First Check multiplier bit
  one by one

• if 1 add multiplicands
  to products highier bits

  [alu / a]
      ↓ highier bit
  and then

• if 0 add multi 0 0 0 0
  to products highier bits

• shift the product to
  right .

- Advantage no need to increase/extend bits of multiplicand.
- multiplicand and multiplier will be of 32 bits
- Adder 32 bits use.

```
    1  0  1  1
    1  0  1  0
 0  0  0  0  0  0  0  0
 0  0  0  0
 0  0  0  0  0  0  0  0  →
    1  0  1  0
    1  0  1  0  0  0  0  0  →
 0  1  0  1  1  0  0  0
 0  1  0  1  1  0  0  0
 0  0  0  0
 0  1  0  1  1  0  0  0  →
 0  0  1  0  1  1  0  0
    1  0  1  1
 1  1  0  0  0  1  0  0  →
 0  1  1  0  1  1  1  0
```

How can we optimize it more?

- size of product increases everytime ⟵ shift
- size of multiplier decreases everytime → shift

- Remark multipliers 32 bit register.

Divider.

Quotient
Remainder.

Divisor + 1
iterations.

X subtraction can't be performed.
add zero to quotient
shift right

subtraction can be performed
add 1 to quotient.

Algo.

- Subtract divisor from dividend
- If remainder is +ve shift Q left and add 1.
- If " " -ve add divisor and shift Q left.
- Shift divisor right by one.
- Repeat for n+1
  → n is original size of divisor.

Problem → overflow → division for multiplication.

Optimizable ?

Rather shifting divisor to left, shift answer To left.

```
              0 1 0 0 1
      1 0 0 0 | 0 1 0 0   1 0 1 0
                1 0 0 0                  x
                1 0 0 1   0 1 0 0        ✓
                1 0 0 0
                0 0 0 1   0 1 0 0
                0 0 1 0 1 0 0 0
                1 0 0 0                  x
                0 1 0 1 0 0 0
                1 0 0 0                  x
                1 0 1 0 0 0 0
                1 0 0 0
                0 0 1 0 0 0 0           ✓
```

Divisor

Quotient

?

Controlan.

Dividend

Can be changed as multiplier.

(Decimal Number Representation in Computer Architecture)

2.75

2 + 0.75

10.11

0.75
× 2
———
1.5

0.5
× 2
———
1

IEEE 754 → Standard Notation

Sign	Exponent	Fraction
1 bit	8 bits	23 bits

Signed Notation

Biased Notation

eg. Signed Notation → -4 — 3
     Biased Notation → -2 — 3

8 bits   Signed Notation → -128 — 127
         biased Notation → -126 — 127

How to find binary Notation for a Bit number in biased Notation ?

Number + base = binary mapping
e.g.   2 + 3 = 5 ≈ 101

binary

	Biased notation		
0	000	0	8
1	001	-3	-2
2	010	-2	-1
3	011	-1	0
-4	100	0	1
-3	101	1	2
-2	110	2	3
-1	111	3	∞

Infinity

Converting Floating point to decimal.

IEEE 754

$$(-1)^s (1+ \text{fraction}) \times 2^{(\text{exponent} - \text{biased})}$$

biased.

signed → comes from sign

exponent → comes from conversion to biased notation (Number + base) → binary

fraction → comes as it is the value after point in normal notation

$$1.1 \times 2^{-2} \rightarrow 0.1 \text{ in fraction}$$

14.625

2^{10}

$8\,4\,2\,1$

14 → 1110

0.625 → 101

14.625 $\xrightarrow{\text{binary}}$ ~ → 1110 . 101

14.625 → 1.110101 × 2

1	0	1	0.625
			2
			1.25
			0.25
			× 2
			0.5
			× 2
			1

converting 1 in biased.

③ +127 = 130 ≈ 10000010

exponent

0	1 0 0 0 0 0 1 0	1 1 0 1 0 1 0 0 → 23

b.k.

64	32	16	8	4	2	1
1	1	1				

converting back from floating notation to decimal.

s	Exponent	fraction
1	10000111	10011000

$$(-1)^1 \ (1 + 0.10011000) \times 2^{135-127}$$

135 − 127   135 − 127

8   8

$$- (1.10011 \times 2^8)$$

$$(-1)(110011000)$$

$$(-1)(408)$$

Addition →

exponent

o First equalise the ~~base~~ of both number.

o قیل کم است کہ اگر بڑا ہے تو اس کی power کم ہے

Question

1. $(-1)^s + (1 + fraction) \times 2^{Exponent-biased}$ ⬇

Shouldn't it be base.

2. How we will represent 0.00. because there is no 1! in it. ②

Multiplication →

$$1.1 \ 10101 \times 2^6$$

$$1. \ 10011. \times 2^0$$

↑ shift poin to end in both

$$\boxed{1 \ 1 \ 10101. \times 2^0}$$ $$\boxed{\times 2^0}$$

$$1 \ 10011. \times 2^5$$

simple

6ᵗʰ multiplication 2^{-5}

What is double precision and single precision?

Lecture 6

o Assembly Review

AX	SI
BX	DI
CX	Flag
DX	BP
SS	SP
DS	PC/IP
CS	IR
ES	

ALU   FPU
Control Unit
Register & caches

MIPS — x86
- 4 addressing — immediate/memory — Immediate
- Register type — Register
- jump type — immediate
  - indirect memory
  - Base index

Instruction Set

> Complex Instruction Set architecture
> large no. of instruction

MIPS — set of 7 opcode

Reduced Instruction Set architecture
fewer instruction

R-type
add $R_1, R_2 R_3$

I-Type
load
$R_1, R_2 \, 40$
Sto " "

J-type -
unconditional jump

R-type

op code	RS	RT	Rd	funting/field

I-type

opcode	R_2	Rd	immediate value

Instruction Type
I-Type
R-Type
J-Type

I - Type

```
31          26 25                          0
┌────────┬──────────────────────────────┐
│ opcode │         Immediate            │
└────────┴──────────────────────────────┘
```

$\underline{Load}$: $R_2 \leftarrow R_1 + [\text{Immediate}]$

$R_1 \leftarrow R_2$

Store

$R_2 \leftarrow R_2 + [\text{Immediate}]$

$[R_2] \leftarrow R_1$

Store $R_1 \rightarrow [R_2 + 40]$

Load $R_1 \leftarrow [R_2 + 40]$

LECTURE 7

Instruction Types

- R-Type
- I - Type
- J-Type

IP → 32 bit → Program Memory → 32 bit instruction → (Division on the basis of Instruction) type

(31-26) opcode

(25-21)

(20-16)

contains Registers

Register file

R-type

Source ← Registers

Destination

Program Memory 32 bit

(31-26) opcode

function field (5-0)

32 R_S 32 R_T

ALU

32

Rd destination register.

(control unit is just to explain)
we will study it in detail later ——

CU

decides which operation to perform

- In R-Type instruction source and destinations are registers

(In memory)

Add R7, R3, R2   Instruction
     R_D R_S R_T

opcode	Rs	R_T	R_D	Shift	function field
000000 0011	000 10	00 11	00000	100000

for addition

Assume that for all register its value is double than its number.

                    R_D R_S R_T
        Sub   R9 , R12 , R15
              number
The value of R_D is 9
The value to be stored in R_D will be
$2(12) + 2(15) = 24 + 30 = 54$
$2(12) - 2(15) = 24 - 30 = -6$

opcode	Rs	RT	RD	shift	funct
000000	01100	01111	01001	00000	100010

} subtraction

```
      11000      (24)₁₀  in  32 bits
    - 1 1 1 0    (30)₁₀  in 32 bits
    _____
      1 1 0 0 0    ( I tried to perform )
      0 0 0 1 0    ( subtraction here... )
```

Note : A bit weird we are going to do I-type example
here now. 😊

I-Type instruction

```
        31        26 25      21 20    16 15                    0
      [ opcode |  Rs  |  RT  |   Immediate Value   ]
```

```
    sub   R 6 , R 3 , 45
          └─┬─┘ └─┬─┘ └─┬───┘
           RⱭT   RⱭS  Immediate
```

```
                                    Immediate Value
    [ X X X X X X |  0 0 0 1 1 | 0 0 1 1 0 |         ↓         ]
       opcode         Rs          RT
                                              45 in 32 bit
                                                 binary.
```

We want processing unit to perform (101101)
both R-Type or I-Type instruction ... in 32 bits.

We have used mux to incorporate multiple options.
i.e. to make system work properly for both

R-Type and I-Type. The selection bits will be opcode.

From instruct label values in decimal on diagram.

instruction →

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14	13 12	11 10	9 8 7 6 5 4 3 2 1 0
00 1011	0 0011	00110	0000 0000 0010 1101

For I-Type.

Sign Extension

0110 ⟶ positive no.

1011 ⟶ number −ve

1111 1011 ⟶

Adding 1's insted of 0's because adding 0's will make it positive.

0000 0110 ⟶ extend it to more bits

No issue while adding zero

Load   R6, R3, 40.

Change it for Store instruction.
Until: • No difference. Till address calculation
       • In load we were reading memory
         now we have to write.
       ◦ Add control signals to Data Memory
         to tell do (?) we need memory
         read or right.
       ◦ and give value of R7 to it while
         giving control signal to write.
       • Similar add register write
         signal to Register File
       why? To because store doesn't write
       any register.