Chapter - 2

Instructions: Language of the Computer

- MIPS Architecture 32 bits
  - ↳ চমৎকারো data এর bit length 32 bit
- Computer এর main memory তে প্রত্যেক data store অবস্থায় থাকে। এই main memory তে একটা 32 bit এর data store থাকবে, 32 bit এর অনেকগুলো data main memory তে store করা থাকে।

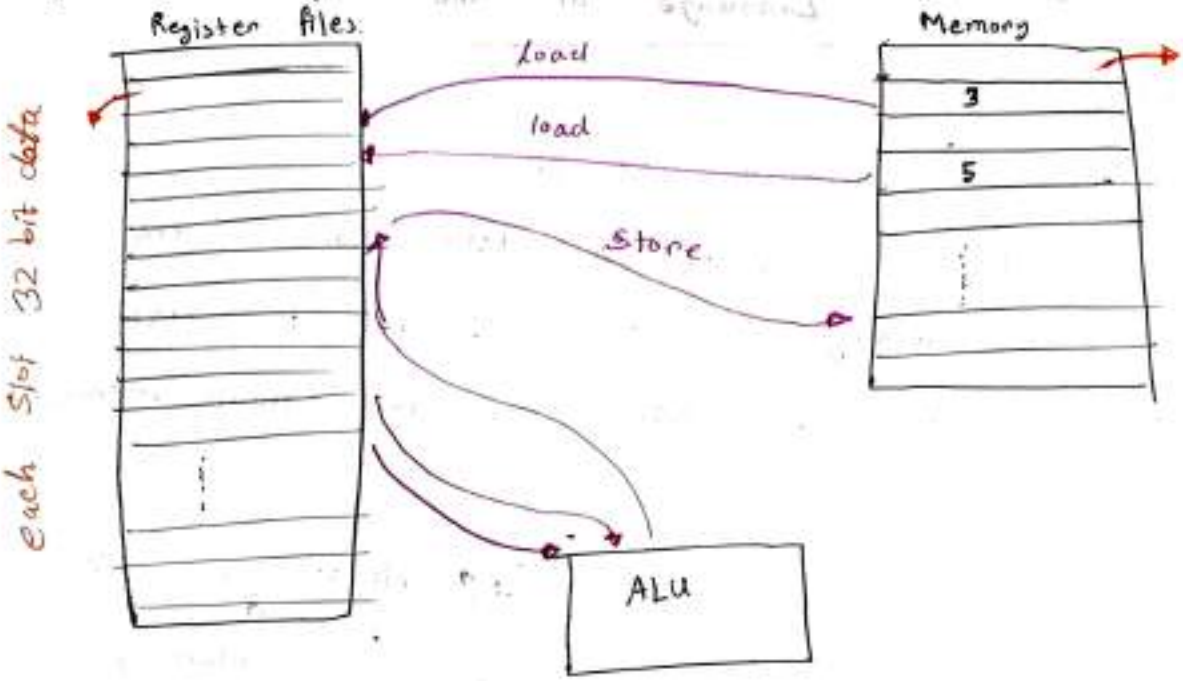
> # 32 bit data are called word.

- Memory Unit ২টা।
  - ⇒ Main Memory also generally known as memory
  - ⇒ register file
    - ↳ 32 টা register এর একটা memory।
    - ↳ each register 32 bit hold করে

main memory তে অনেক memory slot থাকে, so, অনেক data memory তে রাখতে পারি। কিন্তু register file এ limited data রাখতে পারি, কারণ reg. file এ data রাখার জন্য মাত্র 32 টা slot আছে।

CPU register file তখনই use করে যখন বেশিবার data তার frequently use করা লাগে।

কোথায় arithmetic operation ও,



for example . 3 + 5 করতে,

memory থেকে 3 এবং 5 reg file ও নিয়ে আসব,

then 3এবং 5এর ও data ALU ও যায়; সেখানে

বিভিন্ন arithmetic logical operation হয়, then

ALU calculated result টা আবার reg file

ও return করে দিব, then reg file থেকে result

টা memory তে store করব;

add (a), (b) (c)

destination, source 1, source 2

b+c এর result : a তে store হবে

5 → 00000101 (memory তে)

32 bit (reg file এ নিলে 32 bit এ convert হবে).

একেকটা memory slot এর address কত address represent করতে 32 bit লাগবে, কারণ $2^5 = 32$.

Memory থেকে data load করি আর store করি,

Load = retriere করা, memory থেকে নিচ্ছি.

↳ read operation

Store = write operation

ALU (Arithmetic Logic Unit), register থেকে data access করে.

memory → register → ALU

memory address এর সাহায্যে পাবে.

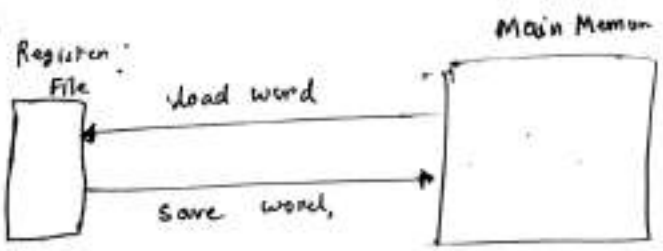
32 combination

00000
00001
00010
00100
00011
...
11110
11111

Memory Operands:

- সব data main memory তে থাকে then সেগুলো কোন register file এ পাঠানো হয়, Then reg file থেকে data ALU তে পাঠিয়ে Arithmetic operation হয়।
- reg. file একটা medium হিসেবে frequently কাজে data use হয় কিন্তু main memory থেকে reg-file এ পাঠাই। Then সেগুলো Instruction এর মাধ্যমে Arithmetic operation হওয়া পর্যন্ত কাজ করে।
- Main memory is used for composite data (Arrays, Structures, Dynamic data, stack)
- Memory is byte addressed ✱
  ⇒ each address identifies an 8 bit
- Words are alligned in memory. ✱
  ⇒ Adress must be a multiple of 4.

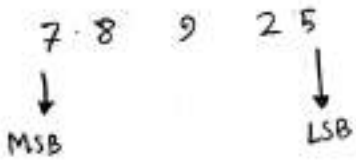
8 bit = 1 byte.

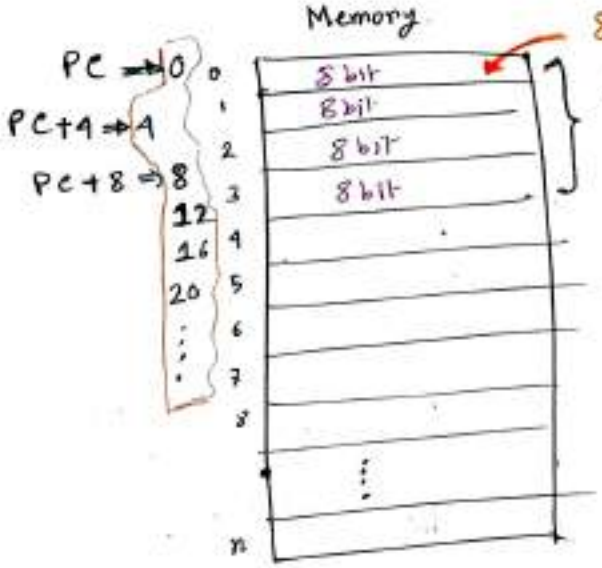


• MIPS is Big Endian.

⇒ MSB byte at least address of a word.

⇒ Little Endian : LSB at least address → LSB higher address ⊐ store.





8 bit of data = 1 byte data

1 slot = 8 bit data

4 Slot = (4×8) = 32 bit data

8 bit = 1 byte

32 bit = 4 byte

= 1 word.

★ as mips architecture ⊐ 32 bits follow করে, মানে 32 bit এর একটা data represent করতে 4টা memory slot লাগে।

★ তাই initial PC (Program counter) 0 তম slot ⊐ point করে, as 4টা slot মিলে একটা 32 bit data represent হয় তাই PC 4 করে Increment হতে থাকে।

1st data occupy করবে memory এর (0-3)

2nd data occupy করবে memory এর (4-7)

Memory এর প্রত্যেকটা slot এর address n bit দিয়ে represent করলে 2^n সংখ্যক address combination বানানো যাবে।

- Memory এর প্রত্যেকটা slot এর address 7 bit দিয়ে represent করলে 2^7 সংখ্যক address combination বানানো যাবে, প্রত্যেকটা location (বা location এর) still 8 bit data hold করবে।

32 bit arch pc তে 4 করে increment করব।
64 bit arch pc তে 8 করে increment করব।

If you address a memory slot by using 7 bits. Determine the size of the memory?

⇒ memory locations $= 2^7$.

$\therefore$ size $= (2^7 \times 8)$ bits

$\because$ MIPS architecture follow করছি; so, 2^{32} টা address combination বা location বানানো যাবে।

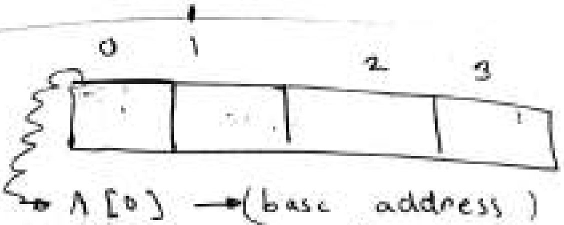
Memory address বের করা:

$g = h + A[3]$

Given that,

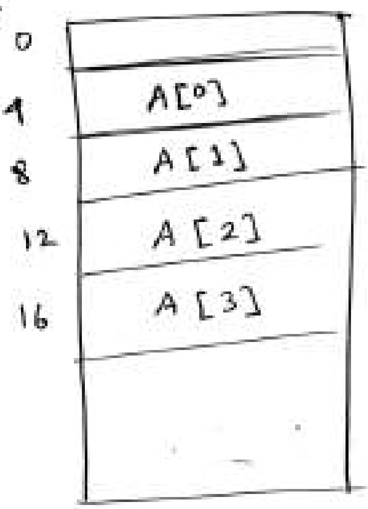
initial address = 4

Find the address of A[3].



↓

array টা যেখান থেকে start হয় তার location.



কয়টা ~~address~~ level cross করতে হবে,

$= (3 \times 4) = 12$

∴ A[3] এর address = initial address + 12

$= 4 + 12$

$= 16$

base address of A in $s3

A[12] = h + A[8]; (h → $s2)

equal এর left side এ array এর location থাকলে result টা main memory তে রাখতে হবে.

মানে এই result টা main memory এর A[12] এ রাখবে।

তার main memory এর

lw $t0, 32($s3) (des: $t0, source: 32($s3))

add $t1, $s2, $t0

sw $t1, 48($s3) (source: $t1, des: 48($s3))

কখনো main mem এর কোনো location বা address কখনো destination হিসেবে দিতে পারব না।

Register vs Memory

- Register are faster than memory.
- Operating on memory data requires load and store.
  - → More instruction to be executed
- Compiler uses registers for variable as much as possible.
  - → Only spill to memory for less frequently used variables.
  - → Register optimization is important

MIPS Arch can support upto 32 address line

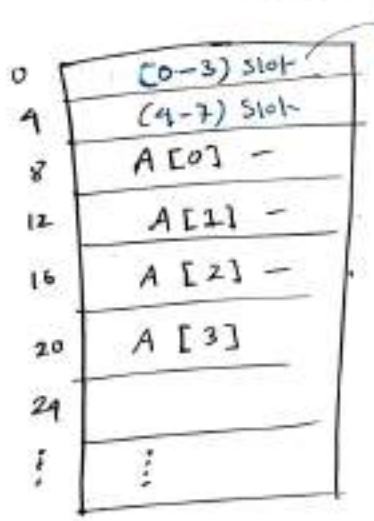
So an address could be 0xAB C0 E 123 in hexadecimal

8 हेक्स digit;
1 hex = 4 bit
∴ 8 hex digit = 32 bits

- সকল memory তে সকল location এর address হয় 32 bit নিয়ে ~~represent~~ store করা হয়।

• register file এ 32 টি register slot থাকে, এবং প্রতিটি slot এ 1 word = 32 bit = 4 byte data থাকে.

• memory ও প্রত্যেক slot এ 8 bit থাকে, তাহলে MIPS Architecture follow করছি, যেটা 32 bit. So, আমাদের প্রত্যেকটা data 32 bit এর memory এর. প্রত্যেক slot 8 bit Therefore, প্রত্যেকটা data এর জন্য memory এর 4টি slot সাগবে.



We have to retrive the data of A[3]. To do that at first we need to find the ~~destination~~ address of A[3].

⇒ From the figure

base address of the array = 8 তম slot.

A[0] এর address থেকে 3টি data skip করতে হবে, এবং প্রত্যেক data এর জন্য 4 slot লাগে,

∴ Number of slot to be skipped = (3×4)
= 12 slot।

of A[3]

∴ ~~Destination~~ address = Base Adress + number of slots to be skipped.

= 8 + 12

= 20.

Formula:

Mips ৩২ bit এর জন্য)

address = (4 × index number) + base address

64 bit architecture এর জন্য)

address = (8 × index number) + base address

Arithmetic and logical operation takes place in ALU.

memory → register (load operation)

register → memory (store operation)

"addi" এতে immediate operand থাকে। addi এ register এর মধ্যে immediate হিসেবে integer value থাকে।

Register Operands

Immediate Operation:

$f = \$s3 + 4$ ——— (i)

↳ addi $s3, $s3, 4.

একটা register এর সাথে সরাসরি কোন integer যোগ হলে তখন addi operation হয়।

$f = \$s3 - 4$

↳ addi $s3, $s3, -4

(ii) এই দুই example এ কোথাও dependancy ছিল না তাই register overwrite করতে পেরেছি।

dependancy থাকলে register overwrite করা যায় না,

$f = g + h$
$a = b + g$

$f \rightarrow \$s2$
$g \rightarrow \$s2$
$h \rightarrow \$s3$
$b \rightarrow \$s4$
$a \rightarrow \$s5$

↳ এর MIPS Code

```
add $t0, $s2, $s3;
add $t1, $s4, $s2;
```

$f = \$s3 + 4$

↳ addi $s3, $s3, 4

$f = \$s3 - 4$

↳ addi $s3, $s3, -4

Register Operands

Register file এ total 32 টি register আছে।

$t0, $t1, ..., $t9 ⇒ Temporary register ⇒ 8 থেকে 15 নং register ($t0 – $t7)

এবং 24 এ 25 নং register for $t8 and $t9

$s0, $s1, $s2, ... $s7 ⇒ Saved registers ⇒ 16 থেকে 23 নং registers.

Temporary register:

is used for storing temporary values or variables যা future এ কাজে আসবে না; just হিসাবের জন্য কাজে আসবে।

Saved register:

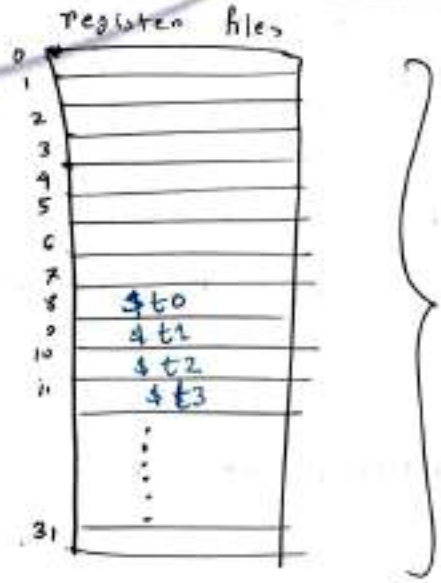
is used for storing variables or values যা পরে কাজে আসবে বা সবসময় কাজে আসবে।

Saved register এ একবার value রাখলে তা over write করে না।

$t0 = $8 ⇒ register file এ 8 নং register

$t = $9 ⇒ register এ 9 নং

এই 8, 9 কে binary তে convert করে।



32 টি register

C - code:

$a = b + c$

MIPS: (just concept বুঝার জন্য)

add $a, $b, $c.

destination Source

C - code:

$f = (g+h) - (i+j)$

g, h, i, j, f যথাক্রমে $s0, $s1, $s2, $s3, $s4 এ store আছে। Now write the MIPS code.

MIPS Code:

add $t0, $s0, $s1 [(g+h) এর মান $t0 consider করলাম]

add $t1, $s2, $s3 [(i+j) এর মান $t1 consider করলাম]

sub $s4, $t0, $t1

ALU operation করতে গেলে memory থেকে reg এ data নিয়ে আসতে then reg থেকে ALU এ গেলে calculate হয়, The result ALU থেকে register এ store হয়, then register থেকে finally memory তে store হয়।

Memory Operand [Example 1] : (Slide - 19) :

C- code:

g = h + (A[8]) → এইটা memory file এ থাকে, so এটার address বের করে data টা fetch করে নিয়ে আসতে হবে।

g in $s1 ; h in $s2 ; base address of A in $s3.

Now write the MIPS code for the given c-code,

⇒ A[8] এর memory address = (8 × 4) + $s3

= (32) + ($s3)

offset ↳ base address

∴ MIPS Code:

lw $t0, (32)($s3) [$t0 register এ memory address বের data টা fetch করে নিয়ে আসছি]

↳ load operator

add $s1, $s2, $t0

---

load এবং store operator দুইটা parameter নেয়, first parameter হলো register. 2nd parameter হলো memory address

load operator → represented as → lw

Store " → sw

load ও memory থেকে register এ আনে আর store ও register থেকে memory তে নেয়।

☑ C-code:

A [12] = h + A [5]

↳ এখানে data store করবে।

→ data read করে নিয়ে আসবে।

h in $s2 ; base address of A in $s3.

⇒ MIPS Code:

address of A[5] = (5×4) + $s3

= 20 + $s3.

address of A[12] = (12×4) + $s3

= 48 + $s3.

∴ MIPS code:

```
lw   $t0, 20 ($s3)
add  $t0, $t0, $s2
sw   $t0, 48 ($s3).
```

Memory এর প্রত্যেক slot এ 8 bit data রাখা যায়।
এবং যেহেতু, mips architecture use করছি বিধায় memory
এর প্রত্যেক slot এর address 32 bit, এখন,
অনুযায়ী, memory এর size কত হবে. দাও?

⇒ memory size, = 2^{32} (location address combination) × 8 (প্রত্যেক slot এ 8 bit data রাখে)

= 34359738368 bit

= 4294967296 byte.

= 4.29496 Gb

$ Zero Register

⇒ $ Zero register একটি unique register

⇒ $ Zero এর value সবসময় constant 0.

⇒ Can't be overwritten.

⇒ register file এর first register হলো
$ zero register.

⇒ Used for more operation. ~~example~~

Example:

$s1 = 7

$t2 = $s1

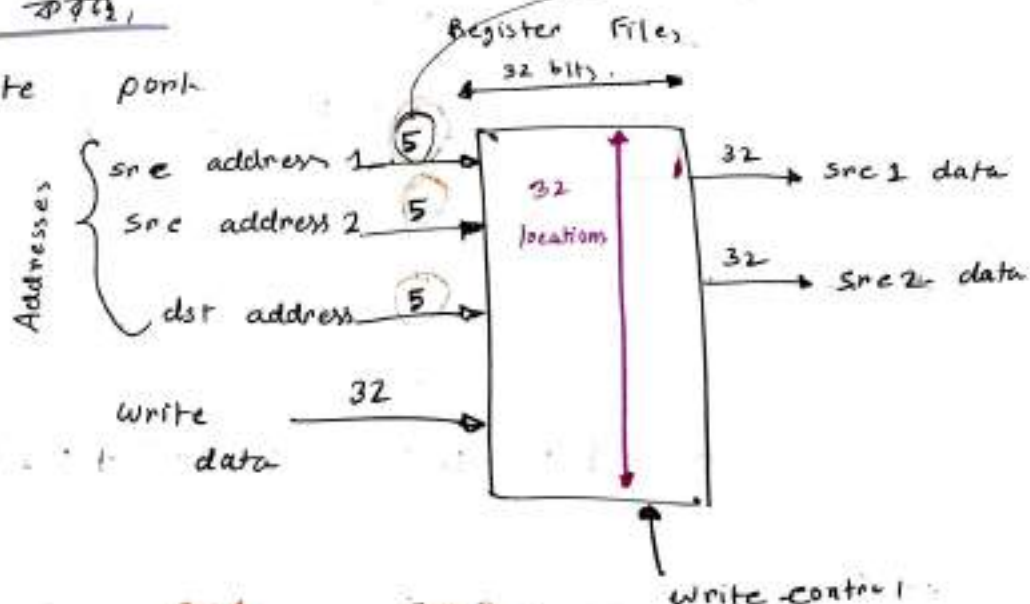
এটার mips code হবে.

add $t2, $s1, $zero

MIPS Register File:

- MIPS register file contains 32 bits registers $= 2^5$
  - ⇒ thirty two 32-bits registers
  - ⇒ Two read port.
    - ↳ data read করতে লাগি.
    - তাহলে যেকোন register data collect করতে,
  - ⇒ One write port

2^5



add ($s1), $t0, $t2.

- ↳ destination register
- dst address দিয়ে destination বোঝা যায়, then data write করতে.

src1 src2.

Unsigned binary Integer:

⇒ আর positive binary integer same কথা,

⇒ n-bit system এ n bit number নিয়ে কাজ করবে,

range: 0 to $+2^n-1$

8 bit এর মানে একটি 8 bit register এ আমরা কোনো একটা value store করতে পারি

যদি $n=8$ হয় তাহলে range 0 to 2^8-1 বা (0 to 255)

তার মানে 0 to 255 represent করতে 8 bit লাগবে,

Similarly, যদি $n=32$ bit হয় তাহলে, range 0 to 4294 967 295

2's complement Signed Integer:

⇒ n-bit system এ number নিয়ে কাজ করলে, 2's complement signed integer এর range:

range: -2^{n-1} to $2^{n-1}-1$

⇒ $n=32$ bit হলে উদাহরণ range:

$-2\,147\,483\,648$ to $+2\,147\,483\,647$

⇒ "Signed" binary এর first bit অথবা MSB যদি 1 হয় তাহলে negative number indicate করে আর MSB 0 হলে positive number indicate করে,

⇒ Most negative number: 100000 ---- 0000

Most positive number: 011111 ---- 1111

Sign Extension

⇒ representing a number using more bits.

⇒ example: (8 bit extended to 16 bit)

- +2: 0000 0010 (8 bit) → 0000 0000 0000 0010
- −2: 1111 1110 (8 bit) → 1111 1111 1111 1110

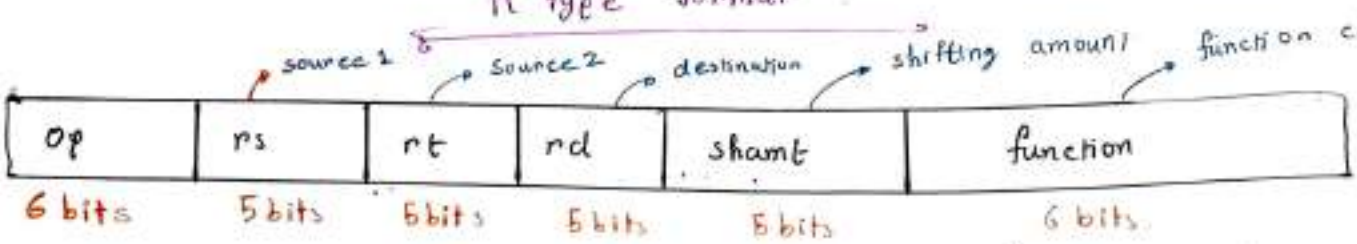
★ MIPs instructions (32 bits) can be divided into three classes,

⇒ R type
- add, sub, and, or, sll (shift left), srl (shift right), slt, sltu
- [Arithmetic operation]

⇒ I type:
- lw, sw, addi, beq (branch equal), bne (branch not equal)
- slti, sltui
  - Set less than i

⇒ J type:
- j (Jump), jal

R type format



R type এর ক্ষেত্রে Opcode সবসময় "000000" হয়।

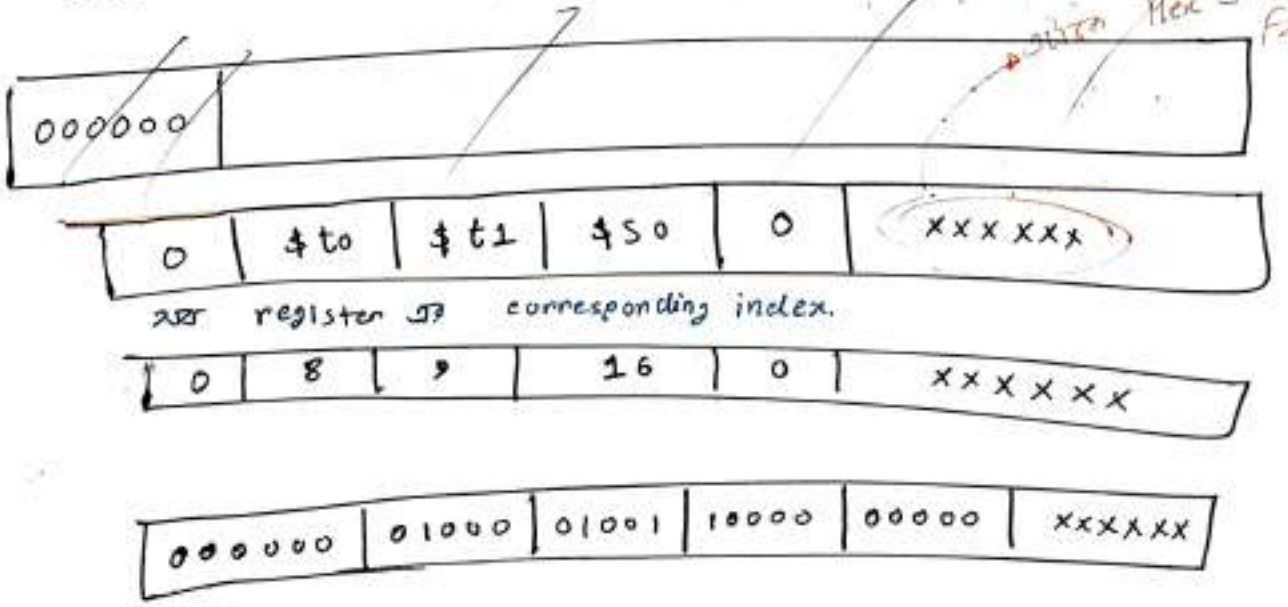
যখন কোনো shifting হচ্ছে না তখন shamt 0 হবে (৫ bit তাতেই)

তাহলে কত bit shifting হবে- তার corressponding binary (৫ bit তাতেই)

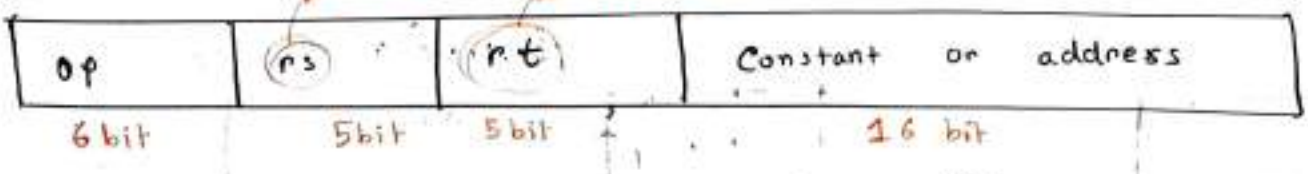
function এর ক্ষেত্রে corresponding value দেয়া থাকে function টার corresponding binary বিয়ে তাহলে 6টা don't care. (XXXXXX)

R type এ কোন register আগে register এর operation হয়।

add $s0, $t0, $t1.



I-Type format



⇒ যদি opcode চলমান থাকে তাহলে বাকী নিম্নরূপ অতিরিক্ত-

"XXXXXX" নিম্নরূপ.

⇒ base address বোঝালে rs এ নিম্নরূপ, offset part তথ্য

constant / address part এ নিম্নরূপ.

constant এর range ⇒ -2^{15} to $+2^{15}-1$

addi $s0, $s1, 2

X	$s1 (17)	$s0 (16)	2

XXXXXX	10001	10000	0000 0000 0000 0010

lw $t0, 12 ($s0)

sw $t1, 24 ($s1)

add $t1, $s1, 20 ($s3)

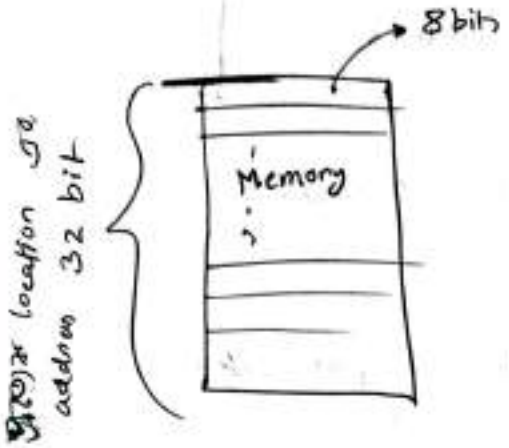
⇒ এটা Itype format

• lw $t0, 12 ($s0)

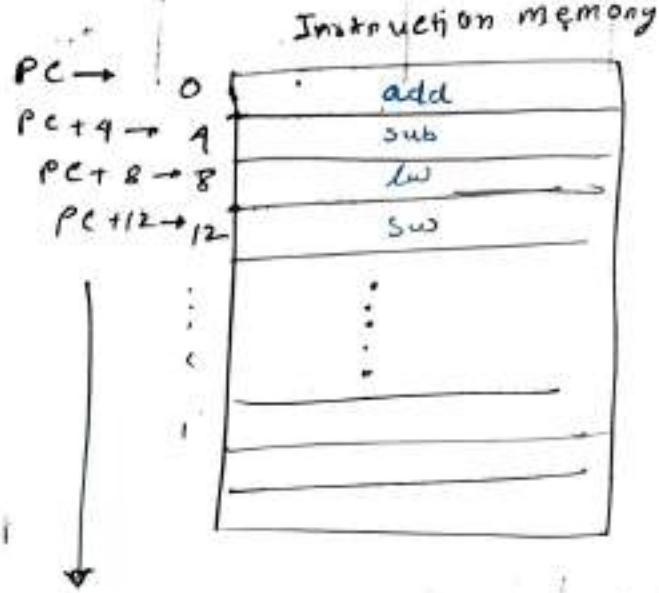
X	$s0 (16)	$t0 (8)	12

XXXXXX	10000	01000	0000 0000 0000 1100

Conditional Operations and Branch Address.



∴ Memory size = $2^{32} \times 8$



instruction গুলো sequentially execute হয়,

PC হলো programme counter, PC indicate করে currently কোন instruction execute হচ্ছে।

C-Code

↓

MIPS.

(i) add

(ii) lw

(iii) sw

instruction fetch → Decode → execute.

Program Counter (PC) first এ add instruction point করে,

holds the address of the current instruction that is being executed

Untill and unless কোনো condition সত্য হচ্ছে।

PC + 4 করে increase হয়; as memory তে 4টি slot-

বিশিষ্ট 32 bit হয়।

PC (Program Counter) একটি register.

PC first এ initial instruction এর address point করে।

এরপর তার next instruction এ point করার জন্য PC এর current value ALU এ পাঠাই। Then সেখানে PC এর current value এর সাথে 4 add হয়ে next instruction এর address point করে।

I-format:

beq $a, $b, L1.

↳ $a এবং $b এর value equal কিনা check করবে, equal হলে L1 function এ jump করবে।

bne $8, $9, L2

↳ $8 এবং $9 এর value যদি equal না হয় তবে L2 function এ jump করবে।

Calculator ও neg দিলে 2's complement করে দেয়।

~~দেয়~~ not দিলে 1's complement.

8 bit

+2 ⇒ 0000 0010 ;

1's complement of (-2) → 1111 1101

2's complement of (-2) ⇒ 1111 1110

∴ -2 ⇒ 1111 1110.

8 bit representation

+2 (8 bit extended to 16 bit)

↳ 0000 0000 0000 0010

-2 (8 bit extended to 16 bit)

↳ 1111 1111 1111 1110

addi $s0, $t1, (-4) → do 2's complement first, once then use sign extension to get the 16 bit representation of "-4"

এর: এই -4 এর 16 bit representation টা I type format এর constant বা address field এ রাখা হবে,

branch instruction

→ Conditional branch

- যখন কোনো condition এর উপর base করে Jump করে কোনো location এ যাই,

→ Unconditional branch

- যখন কোনো condition ছাড়াই ই (Jump) করতে বলা হবে,

(P.T.O).

C- code:

```
if (a == b){
    a = b+1
}
else {
    a = b + 2
}.
```

→ এখানে equal কিনা check করতে বলেছে ; so আমরা MIPs এ উল্টোভাবে ভাবব তাই not equal দিয়ে check করব,

a is stored in $s1
b is stored in $s2

MIPs Code:

```
0. bne $s1, $s2, Else      → conditional branch
4. addi $s1, $s2, 1
8. j Exit                  → unconditional branching.
12. Else:
16. addi $s1, $s2, 2
```

assume করছি else function এর জন্য computer random একটা number 2 generate করছে।

∴ (2×4) = 8 slot

আর, 1st instruction শেষ হওয়ার পর থেকেই 8 টা slot skip করে আমি else function এ jump করতে পারবো।

(c.r.s)

C - code :

```
if (a != b) {
    a = b + 1
}
else {
    a = b + 2
}
```

a is stored in $s1

b is stored in $s2

MIPS code :

```
0.  beq   $s1, $s2, Else
4.  addi  $s1, $s2, 1
8.  j     Exit
12. Else: addi $s1, $s2, 2
16. Exit:
```

Stored Program Computers :

- Instructions and data both are represented in binary.
- Instructions and data both are stored in memory.
- Programs can operate on programs.

  like : Compiler, linkers

- Binary Compatability allows compiled programs to work on different computers.
  - Standardized ISA.

Operation	MIPS
Shif left	sll
shif right	srl
Bitwise AND	and, andi
Bitwise OR	or, ori
Bitwise NOT	nor
Set if less than	slt, slti

registen এর সাথে কোনো integen value add করতে হলে "addi"

C- code:

```
if (i==j) {
   f= g+h
}
else {
   f= g-h
}
```

f, g, h, i, j are stored correspondly in

$s0, $s1, $s2, $s3, $s4

MIPS:

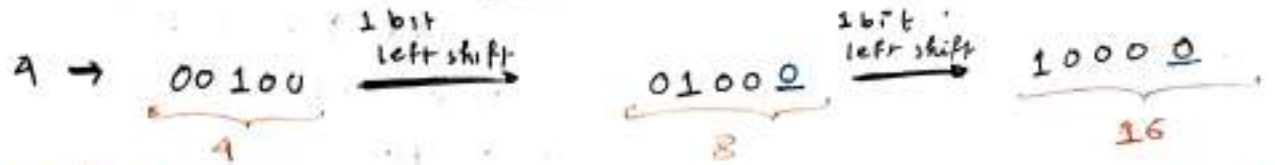
```
PC→0       bne  $s3, $s4, Else
PC+4  →4   add  $s0, $s1, $s2
PC+8  →8   j    exit
PC+12 →12  Else:
PC+16 →16  sub  $s0, $s1, $s2
PC+20 →20  Exit:
```

- Computer else block এর জন্য একটা random integer generate করে generate করে দিচ্ছে। Suppose, 2 generate করে দিলো, তার মানে ধরি currently যেই instruction এ আছি তার পরের 2 line (2×4 = 8 slot) skip করে যাও,

২টো branching condition আছে,

beq, bne. } i- type instruction

5 bit register



কয় line shift করব তা indicate করবে,

$4 \times 2 = 4 \times 2^{1} = 8$

$4 \times 4 = 4 \times 2^{2} = 16$

number of shift

2 এর power হবে হবে

General Representation:

- $x \times 2^{n}$ → number of left shift.
- $\frac{x}{2^{n}}$ → number of right shift.

32 bits architecture এর জন্য $x \times 4$ করলে multiplied by 2^2 বা 4 হবে, কারণ আমাদের data memory এর 4 টা slot চেঞ্জ করে থাকে, তার জন্য আমরা 2^2 দ্বারা multiply করেছি। 32 bit architecture এর জন্য করলে ~~left~~ 2 bit left shift হবে,

2 দ্বারা multiply করলে 1 bit left shift হবে,

```
beq $s1, $s2, L1
add ....
j Exit
L1:
add ...
Exit:
```

2 → offset

branch address calculate করছি।

Current inst. থেকে 2 inst পরে L1

Current inst থেকে 8 slot পরে L1 আছে

16 → offset (16 bits)

2

16 bit representation of করছো।

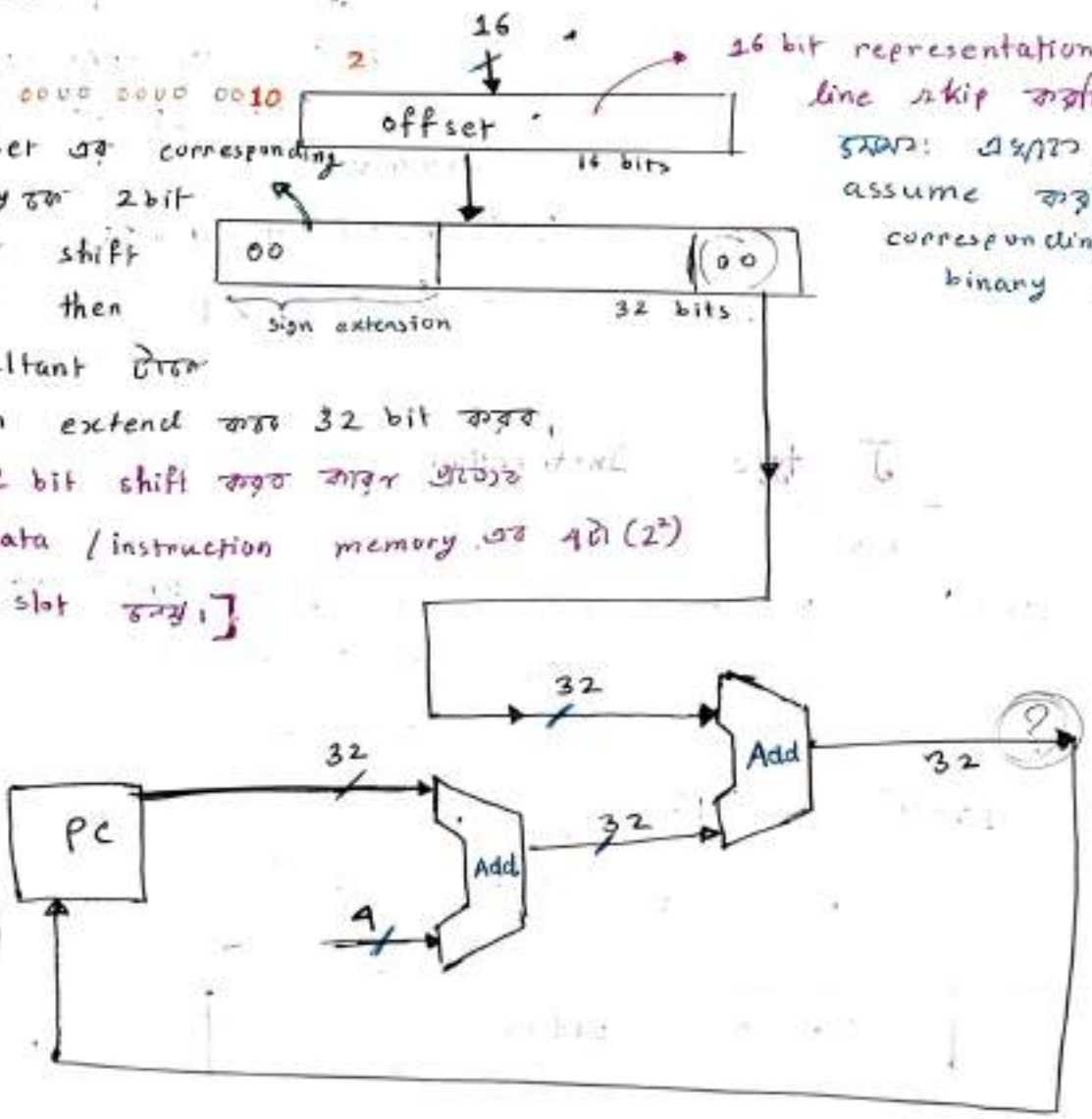
line skip করছি।

0000 0000 0000 0010

offset এর corresponding binary কে 2bit left shift করব then resultant কে sign extend করে 32 bit করব,

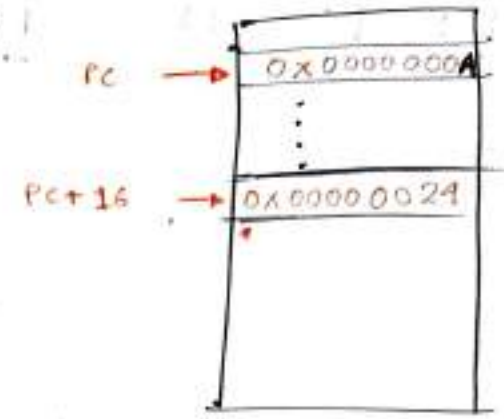
[2 bit shift করে কারণ প্রত্যেক data / instruction memory এর ঘর (2^2) slot করে।]

উদাহরণ: এখানে L1 কে 2 assume করছি so 2 এর corresponding 16 bit binary number



0x 0000 000A

```
beq  $s1, $s2, L1
add  . . .
j    sub . . .
     sll . . .
     srl . . .
L1:
add  . . .
Exit:
```



এখানে 16 টি slot skip করা হয়েছে, so L1 এর address

$\therefore (0000000A + 4 + 16)_{16}$

$= (00000024)_{16}$

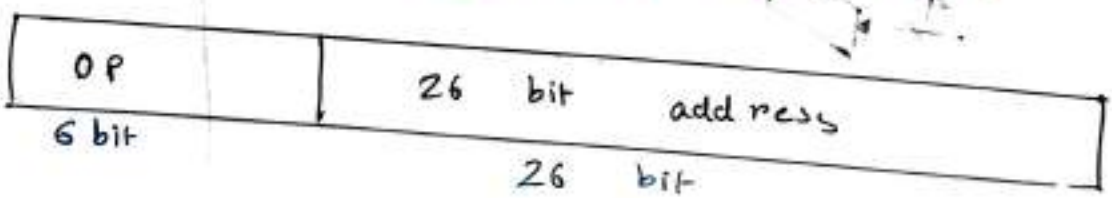
খ

J type Instruction

j label → 10 এটা দ্বারা next 10 টি line skip করা হয়েছে level function এ যাওয়া হয়।

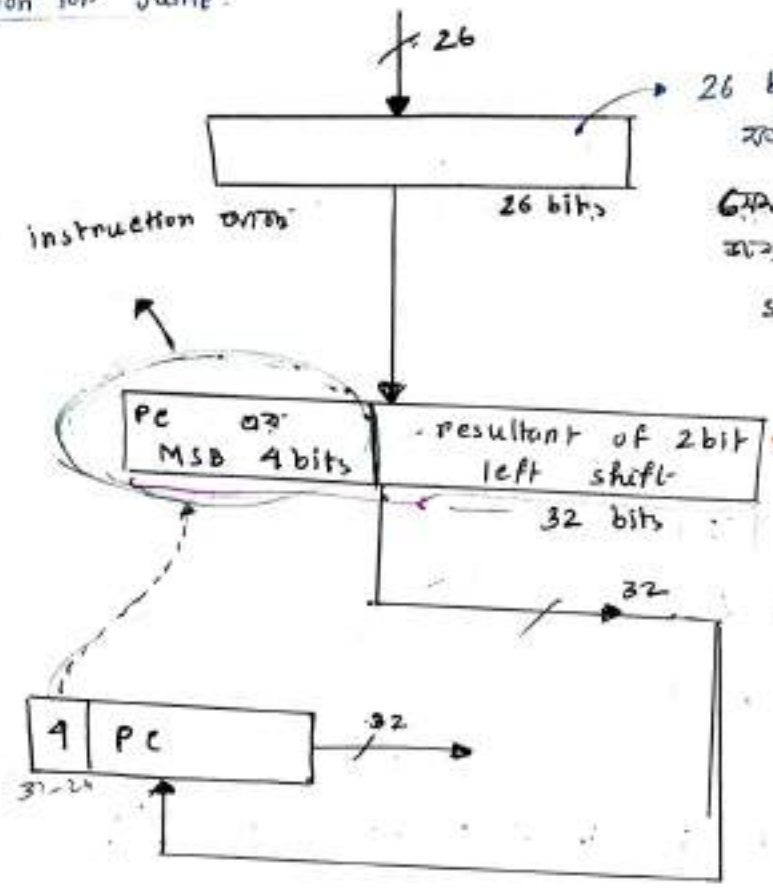
$10 \times 4 = 10 \times 2^2 = 40$

2 bit left shift



Opcode দেয়া থাকলে opcode গুলোর অংশ হয়, দুইটা don't care "XXXXXX" লিখব।

Branch Destination for Jump:



26 bit representation of যত line skip করতে হবে, উদাহরণ: এখানে label এর মান 10 assume করছি; so 10 এর corresponding 26 bit এর binary মান.

26 bit number টাকে 2 bit left shift করব। তাহলে, 28 bit এর একটা number পাবো, এখন বাকি 4 digit এর জন্য PC এর first 4 MSB bit কপি, নিব at the begining of that 28 bit number.

in J type instruction we didn't need any ALU. Because, J type instruction do not use any kind of operation on they do not use ALU for calculating address

↳ known as concatenation method.

j, jal

↳ J and link

MIPS Code Practice:

- B [10] = A[5] + 2

  Where base address of B and A are $s1 and $s2 respectively

→ **MIPS:**

```
lw  $t0, 20 ($s2)
addi $t0, $t0, 2
sw  $t0, 40 ($s1)
```

- Base address of A is in $s0. Write the MIPS code for the given set of c- code,

```
if (A[3] != A[6]){
    if (A[3] == 0){
        A[3] = A[3] +2;}
    else {
        A[6] = A[6] /16; }
}
else {
    A[6] = A[6] *8
}
```

x/24 and sra

2^3 sll

⇒ MIPS code:

```
lw   $t0 , 12 ($s0)     # A [3]
lw   $t1,  24 ($s0)     # A [6]

beq  $t0,  $t1, L1
bne  $t0,  $zero, L2

addi $t2 , $t0 , 2      # A[3] +2
sw,  $t2,  12 ($s0)
j  Exit

L2:
srl $t3,$t1, 4          # 4 bit right shift
sw  $t3, 24 ($s0)
j  Exit

L1:
  sll  $t4 , $t1, 3     # 3 bit left shift
  sw   $t4,  24 ($s0)

Exit:
```

x, y, z are stored in $s0, $s1, $s2

$x = (2y) + 65z + 10$

$2y$ → $t0, $65z$ → $t1, result → $t0

⇒ MIPS:

```
add  $t0, $s1, $s1   # 2y
sll  $t1, $s2, 6     # 6 bit left shift
add  $t1, $t1, $s2   # 65z
add  $t0, $t0, $t1   # 2y + 65z
addi $s0, $t0, -10
```

$64z + z = 65z$

$z \times 2^6 = 64z$

6 বার left shift.

More Conditional Operation :

- slt (R type)
- slti (I type)

slt → set less than

set less than ।

Greater than check করার জন্য 3 slt বা slti use হবে।

$s3 ≥ $s4 → greater than হলে রেজাল্ট একটা register এ 0 store হবে

$s3 < $s4 → less than হলে রেজাল্ট একটা register এ 1 store হবে।

Formal হচ্ছে।

slt rd, rs, rt

destination

↳ if (rs < rt) then rd = 1 else rd = 0

slti rt, rs, constant

destination

↳ if (rs < constant) then rt = 1 else rt = 0

- Use in combination with "beq" and "bne"

```
slt $t1, $s3, $s4
```

যদি $s3 ≥ $s4 হয়
তাহলে $t1 এ 0 store
হবে,
অন্যথায়, $s3 < $s4 হলে
t1 এ 1 save হবে.

set করা মানে 1 assign করা

reset করা মানে 0 assign করা

কোনো value এর মানের check করতে slti ব্যবহার করব as int এর মানে consider করছি.

a and b are stored in $s0 and $s1

C - code:

```
if (a > b) {
    a = a+1 ;
}
else {
    a = a+2 ;
}
```

main code এ > ; so compare করার সময় less compare করব।

→ MIPS:

```
slt $t1, $s0, $s1
bne $t1, $zero, Else
addi $s0, $s0, 1
j Exit
Else:
addi $s0, $s0, 2
Exit:
```

$s0 < $s1 হলে $t1 = 1

c- code:

```
if (a < b) {
    a = a+1 ;
}
else {
    a = a+2 ;
}
```

Main code এ < : so compare করার সময় greater than equal এর সাথে compare করছে।

MIPS:

```
slt   $t1, $s0, $s1
beq   $t1, $zero, Else
addi  $s0, $s0, 1
j     Exit

Else:
   addi  $s0, $s0, 2

Exit:
```

$t0/16 = $t0/2^1 → 1 bit right shift

signed comparison এর জন্য slt, slti

unsigned comparison এর জন্য sltu, sltiu

■ Why not blt, bgt, etc?

⇒
- Hardware for $<$, $\geq$, ... slower and complex than $=$, $\neq$.
  - Combining with branch involves more work per instruction, requiring a slower clock.
  - All instructions are penalized.

■ Write the MIPS code for the following C code,

$x = A[i] + 2$

where, x, &, base adress of A and i are in $s1, $s2 and $s3 respectively.

⇒

```
sll  $t0, $s3, 2   # multiplying (i*4) so 2 bit-
                     left shift.
add  $t0, $s2, $t0 # finding the mem. address
                     of 'A[i]'
lw   $t1, 0($t0)   # retriving the value of A[i] to
                     $t1.
addi $s1, $t1, 2.
```

A [B [i]] = x

Where base address of A and B are in $s1 and $s2 respectively and x and i are in $s3 and $s4

→

```
sll  $t0, $s4, 2
add  $t0, $t0, $s2
lw   $t1, 0($t0)
sll  $t0, $t1, 2
add  $t0, $t0, $s1
sw   $s3, 0($t0)
```

Formula:

- Memory Address for data in array = base address + (index x 4)
- Branch address = PC + 4 + (offset x 4)
- Jump address = PC (MSB 4 bits) + (offset x 4)

offset এর 26 bits নেব then 2 bit left shift then PC এর MSB 4 bit at the beginning

R type:

→ sll ($t0) ($s1) (2)

$t0 → rd, $s1 → rt, 2 → shamt

000000	0	$s1	$t0	2	xxxxxx
op	rs	rt	rd	shamt	funct

000000	00000	$17	$8	00010	xxxxx

(P.T.O)

Op	rs	rt	rd	shamt	funct
000000	00000	10001	01000	00010	xxxxxx

sll এর Spl operations এই এর rs

0 হবে,

C-code:

g = h * A[8]

Logical Operations.

Operation	MIPS
Shift left	sll
Shift right	srl
Bitwise AND	and, andi
Bitwise OR	or, ori
Bitwise NOT	nor

Bitwise AND → and, andi

Bitwise OR → or, ori → I type

Bitwise NOT → nor → R type

and $t0, $t1, $t2

rd rs rt

$t1 = 1110

$t2 = 0101

∴ $t0 = 0100

or $t0, $t1, $t2:

$t1 = 1110

$t2 = 0100

$t0 = 1110

operation between one register and an integer value

andi $t0, $t1, 2

$t1 = 1011

$t2 = 0010

∴ $t0 = 0010

ori $t0, $t1, 2

$t1 = 1011

$t2 = 0010

∴ $t0 = 1011

a nor b = Not (a or b) → তাহলে a or b এর মানটি or operation হবে, Then তাহার Complement টা হবে

সুতরাং Not করার উপায়.

nor $t1, $t2, $zero

$t2 = 1011

$zero = 0000

∴ Or operation এর ফল = 1011

∴ এখন তাহার not হবে,

∴ $t1 = 0100

nor $t1, $t2, $t3

$t2 = 0100

$t3 = 1000

Or operation এর ফল = 1100

Not এর ফল $t1 = 0011

Loops

২ ধরনের loop আছে while loop এবং for loop.

C-code: (While loop):

```
while (save[i] == k) {
    a = a+2;
    i += 1;
}
```

i, k, a are stored in $s3, $s5 and $s4 respectively and base address of save is in $s6.

MIPS Code:

```
Loop: SLL $t0, $s3, 2
      add $t0, $t0, $s6   # memory address of save[i]
      lw $t1, 0($t0)
      bne $t1, $s5, Exit
      addi $s4, $s4, 2.
      addi $s3, $s3, 1
      j loop   # loop iterate করাচ্ছে
Exit:
```

For loop:

C- code:

```
for (int i = 0;  save[i] > k ; i++) {
    a = a+2
}
```

i, k, a are stored in $s3, $s5 and $s4 respectively.

Base address of save is in $s6

⇒ MIPS Code:

```
add  $s3,  $zero,  $zero  ⟶ # initially assigns i = 0;
Loop:
    sll  $t0,  $s3, 2
    add  $t0,  $t0,  $s6
    lw   $t1,  0($t0)
    slt  $t2,  $t1,  $s5  ⟶ # যদি main code এ > then 1 এর সাথে compare করবে; less than হলে $t2 তে 1 set হবে।
    bne  $t2,  $zero, Exit
    addi $s4,  $s4,  2
    addi $s3,  $s3,  1
    j  loop
Exit.
```

Java Code:

```
for (int i=0 ; i< 15; i++){
    if (A [i+1] != 0) {
        sum = sum+1 ; }
    else {
        sum = sum -1;
    }
}
```

i is in $s3,
B.A. of A in $s0
Sum is $s1.

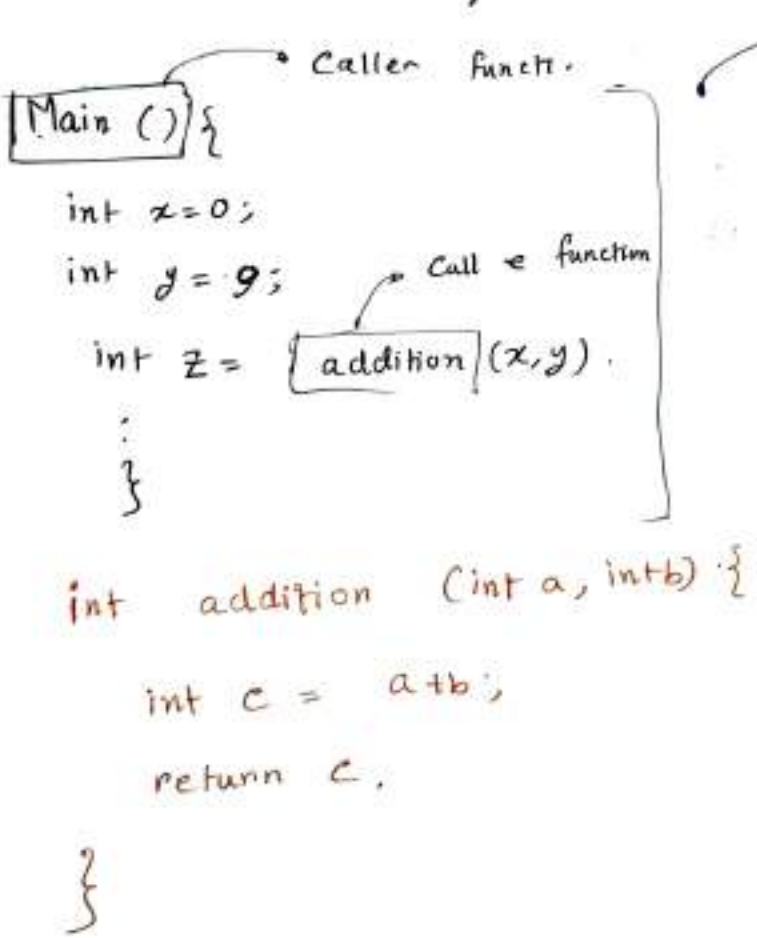
⇒ MIPS Code:

```
    add   $s3, $zero , $zero
Loop:
    slti   $t0, $s3, 15
    beq    $t0, $zero, Exit
    sll    $t1,
    addi   $t0 , $s3, 1 ——→ i+1
    sll    $t0, $t0, 2
    add    $t1, $s0, $t0
    lw     $t0, 0($t1)
    beq    $t0, $zero, Else
    addi   $s1, $s1, 1
    addi   $s3, $s3, 1
    j Loop
Else:
    addi   $s1, $s1, -1
    addi   $s3, $s3, 1
    j Loop
Exit:
```

Function

Function Calling Procedure.

- Place parameters in registers
- Transfer Control to procedure
- Acquire Storage for procedure.
- Perform procedures operation.
- Place result in register for caller.
- Return to place of call



Main function is the caller function, Where a function is called, and the parameters are provided for the called funct.

- Parameters can be passed in argument registers ($a0 ~ $a3)
- Return address of a function is stored in $ra.
- function থেকে value return করা হয় 2টি registers. $v0, $v1.

Registers:

- $zero.

- ($a0 – $a3) ⟶ (arguments)
  - ↳ reg's (4-7) no register
  - ⇒ function এর arguments pass করা হয়,

- ($v0, $v1) ⟶ (results value)
  - ⇒ reg's (2,3) no register
  - ⇒ function এর থেকে value return করব,

- ($t0 – $t9) ⟶ (Temporary registers).
  - ⇒ reg's (8, 9, 10, 11, 12, 13, 14, 15) registers and (24,25) reg.
    - 8 থেকে 15 এ register ($t0 – $t7)
    - 24 & 25 এ register ($t8, $t9)
  - ⇒ Temporarily value store করে.
  - ⇒ Can be over written by callee.

- ($s0 – $s7) ⟶ Saved (registers)
  - ⇒ Must be saved or restored by calle
  - ⇒ registers (16, 17, 18, 19, 20, 21, 22, 23) register

- $gp ⇒ Global Pointer for static Data
  - ⇒ 28's register
  - ⇒ static data hold করে এমন pointer

• $sp → Stack Pointer

⇒ 29 নং register

⇒ Stack এর top address কেও denote করে.

⇒ Top address টা point করে.

⇒ Stack এর top data এর memory address hold করে বা point করে.

• $fp → frame pointer

⇒ 30 নং register.

⇒ Stack related সব info hold করে.

• $ra ⇒ Return address

⇒ 31 নং register.

bubble sort algorithm এ যদি $a0, $a1 use হয়; so initially $a0, $a1 এর value গুলো কোথাও জায়গায় store করে রাখতে। then bcz $a0, $a1 কে বারবার use করা লাগে.

$a0, $a1 মানে argument এর জন্য অন্য argument use করা হবে না in case of bubble sort

jal

↳ jump and link

↳ function এর return address $ra register এ store থাকে,

↳ I will jump to that particular function and also that address in $ra

int z = leaf _ example (1, 2, 3, 4)

↳ এর জন্য jal leaf_example

jR → jump register

↳ jr $ra

↳ function এর last এ দিতে হবে, এতে যে function থেকে call হয়েছে সেখানে return হবে।

Procedure Call Instructions :

- Procedure call: jump and link

    jal procedureLabel

- Address of following instruction put in $ra
- jumps to target address

- Procedure return: jump register

  jr $ra

  - Copies $ra to program counter
  - Can also be used for computed jumps

function এর ভিতর যদি কোনো saved register use করি তাহলে problem arise হয়,

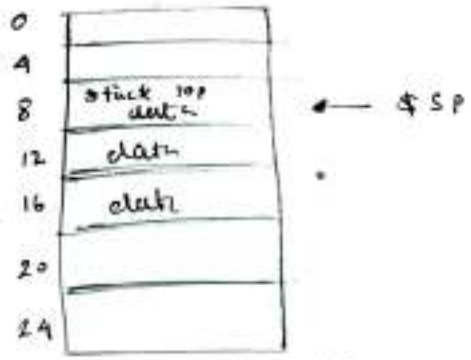
main method এ save register এ যে value ছিলো ফিবোনাচ্চি তে change হয়ে যায়।

যখন function এর ভিতর কোনো save register use করি তখন আমাদের করা হলো saved register এ previous যে data আছে তা আমরা আগে memory তে store করে নিই, then কোনো function এর ভিতর কাজ শেষ করে ঐ memory থেকে data কে ফিরায়ে আনবো নিয়ে নিবো।

যখন function এ কাজ করার পর আমরা saved value এর register এ অপরিবর্তিত রাখি। তার জন্য $sp লাগে।

Data memory তে Stack

চিত্রসহ Consider করি।



Called function এ $SP কে একবার উপরে

উঠতে হবে বিধায় $SP = $SP - 4 করতে হবে।

leaf function.

যেসব function এর ভিতর আরেকটা কোন

আর function কে call করা হয় না তাদেরকে

leaf function বলে।

Non leaf function.

যেসব function এর ভিতর আরেকটা আরেক-

function কে call করা হয়ে থাকলে তা non-

-leaf function.

Leaf procedure example.

C- codes.

```
Main() {
    int f=0;
    f= f+1;
     int z = leaf_example (1, 2, 3, 4);
      int y = f+z;
}

int leaf_example (int g, h, i, j) {
    int f;
    f= (g+h) - (i+j);
     return f;
}
```

- Arguments : g, h, i, j in $a0, $a1, $a2, $a3,
- f in $s0.
- Result in $v0
- z, y in $s1, $s2.

⇒ MIPS Code

```
add  $s0, $s0, $zero   # f=0
addi $s0, $s0, 1       # f= f+1
Jal  leaf_example
add  $s1, $v0, $zero
add  $s2, $s0, $s1
J Exits
leaf_example:
addi $sp, $sp, -4
sw   $s0, 0($sp)
add  $t0, $a0, $a1
add  $t1, $a2, $a3
sub  $s0, $t0, $t1
add  $v0, $s0, $zero
lw   $s0, 0($sp)
addi $sp, $sp, 4
jr   $ra
```

initial f এর value store করে রাখতে হবে।

(g+h)

(i+j)

f=(g+h)-(i+j)

initial f এর value already $s0 এ stored. এবং পরবর্তী line of code এ $s0 এর value নিয়ে update করে stack pointer update করার নতুন location point করছে।

f এর value return করার আগে $v0 এ move করে নিতে হবে।

$s0 এর previous value ফিরিয়ে আনার জন্য

C- Code:

```
int leaf_example (int g, h, i, j);
{ int f, x;
  f = (g + h) - (i + j);
  x = f + 2;
  return x;
}
```

- f in $s0, x in $s1
  - as এই দুইটা save register ও তাই so আমাদের stack এ save করতে হবে।
- Result in $v0

as দুইটা variable দুইটা save register ও stone. so function ও এই দুটা register এ আগে দুইটা location নামাব, এবং stack pointer ২ বার update হবে.

⇒ MIPS Code:

Leaf_example :

```
addi $SP, $SP, -4
sw $s0, 0($SP)        # stack pointer update করা হচ্ছে যাতে location point করে, and function এ ফিরলে $s0 এর value এ location এ store হবে
addi $sp, $sp, -4
sw $s1, 0($sp)        # (g+h) function এর ভিতরে x এর value যে যায় location এর $s1 এ save হবে
add $t0, $a0, $a1
add $t1, $a2, $a3     # (i+j)
sub $s0, $t0, $t1     # f = (g+h) - (i+j)
addi $s1, $s0, 1      # x = f+1
add $v0, $s1, $zero
lw $s1, 0($sp),
addi $sp, $sp, 4
lw $s0, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Branching far away:

যদি branch করার পর এমন একটা address হয় যা 16 bit দ্বারা represent করা যাচ্ছে না; আবার more than 16 bit লাগে সেক্ষেত্রে আমরা instead of branching; আমরা jump use করব যা 26 bit address allocate করতে পারে। This concept is known as branching far away.

- beq $s0, $s1, L1

Suppose L1 এর address 18 bit যা branching address যেটা i type format দ্বারা represent করা possible না, তাই আমরা- j type format use করব, তবে condition opposite করে দিব

এটা change করে দিয়ে

```
bne $s0, $s1, L2
j L1

L2:
  :

L1:
  ;
```

$s0 এবং $s1 equal না হলে L2 তে যাবে। যা 16 bit address, $s0, $s1 equal হলে bne condition satisfy করবে না তবে সেই L1 এ jump

করে, ~~সব j type~~ এখনও 11 18 bit ই assume করলাম but অনেকদূর jump করি না 26 bit address support করে so এখন কোনো problem হবে না.

একটা ASCII character বা character array 8 bit data নিয়ে কাজ করে, তাই ASCII বা char নিয়ে কাজ করার সময় lb, sb ব্যবহার করে

Deals with 8 bit:

- lb ⇒ load byte
- lbu ⇒ load unsigned byte
- sb ⇒ ~~Save~~ store byte

Deals with 16 bit:

- lh ⇒ load halfword
- lhu ⇒ load unsigned halfword
- sh ⇒ store halfword.

1 word = 32 bit

∴ 1 halfword = 16 bit

lb, lh

↳ sign extension হয়ে 32 bit হয়.

lbu, lhu

↳ 0 extend হয়ে 32 bit হয়

lb. rt, offset (rs) → Fetching 8 bit data

lh. rt, offset (rs) → Fetching 16 bit data

ASCII char are always unsigned

- 32 bit architecture takes = $(\frac{32}{8}) = 4$ slots in memory to store a data.
- 64 bit architecture takes = $(\frac{64}{8}) = 8$ slots in memory to store a data
- 256 bit architecture takes = $(\frac{256}{8}) = 32$ slots in memory to store a data

64 bit architecture এ A[6] এর content টা load করার জন্য register এ

Ex. lw $t0, (48)($s1)

48 because = (6×8) = 48

multiplied by 8 because we are considering 64 bit arch.

• Leaf function এ একটা function call হয়ে যেটা তার মধ্যে অন্য আরেকটা function কে call করে না।

• যদি একটা function এর মধ্যে আরেকটা function call হচ্ছে বা একটা function এর মধ্যে যখন function ই বারবার call হয় সেটা non leaf function

Bubble Sort এ Argument Pass এর জন্য মাত্র ২টা — argument register use করতে পারব $a0 এবং $a1, এটা fixed.

→ একটা memory থেকে data fetch করে আরেকটা mem. এ data রাখতে পারি না। মাঝখানে register এ value store করে রাখতে হবে, তো যেহেতু fixed দুইটা argument ই use করতে হবে তাই argument টা clear করে কোথাও store করে পরে আবার সেখান থেকে retrieve করব.

Bubble Sort এর Code টা FNK mam এর Playlist

থেকে দেখতে হবে।

Bubble Sort MIPS Code

```
add   $s2, $a0, $zero    # v
add   $s3, $a1, $zero    # n
add   $s0, $zero, $zero  # i = 0
Loop1:
  slt   $t0, $s0, $s3      # i < n
  beq   $t0, $zero, Loop1Exit
  addi  $s1, $s0, -1       # j = i - 1
Loop2:
    slti  $t0, $s1, 0        # j < 0, $t0 = 1.
    bne   $t0, $zero, Loop2Exit
    sll   $t1, $s1, 2        # $t1 = 4 * j
    add   $t2, $t1, $s2      # $t2 = 4*j + Base address of v
    lw    $t3, 0($t2)        # v[j]
    lw    $t4, 4($t2)        # v[j+1]
    slt   $t0, $t4, $t3      # v[j+1] > v[j], $t0 = 0
    beq   $t0, $zero, Loop2Exit
    add   $a0, $s2, $zero
    add   $a1, $s1, $zero
    jal   Swap
```

```
addi  $s1, $s1, -1
j Loop2
Loop2 Exit:
    addi $s0, $s0, 1
    j loop1
Swap :
    sll  $t1, $a1, 2  → k * 4
    ~~addd $t1, $a0, $a2~~
    add  $t1, $a0, $t1
    lw   $t0, 0(($t1)
    lw   $t2, 4(($t1)
    sw   $t2, 0(($t1)
    sw   $t2, 4(($t1)
    jr   $ra
```

Kindly FNK Mam ಅವರ recording ನಲ್ಲಿರುವ clearly ಹೇಳಿದ್ದಾರೆ ಅದನ್ನು ಹೇಳಿದ್ದಾರೆ ಅದನ್ನು ಹೇಳಿದ್ದಾರೆ

(P.T.O)

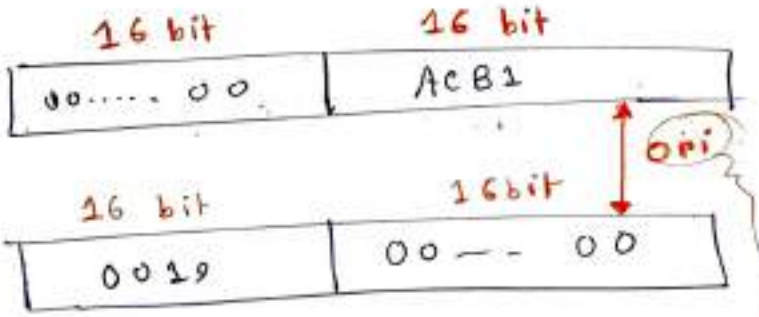
Suppose আমাকে একটা নিচের code টা দিল
করতে বা problem করে

```
lui   $s1,   0019ACB1
```

MSB = 0019
LSB = ACB1

প্রত্যেকটা 16 bit, এখন lui use করে এই 16 bit গুলোকে $s1 এ store করতে পারবো.

```
lui $s1, 0019
ori $s1, $s1, ACB1.
```



or operation

lui = load unsigned integer.

- lui হলো itype instruction যেখানে constant বা address এর জন্য allocated যে হলো 16 bit.
- কিন্তু 0019ACB1 হলো 32 bit ; which is larger than 16 bit.
- So এটা possible না, তাই আমরা 0019ACB1 কে MSB আর LSB তে ভাগ করে নিব.

Chapter 2 থেকে important topic for Mid

- Register file কি? Memory তে PC + 4 করে কত করে increment হয়?
- Basic C code to MIPS Code Transformation ★★★
- Different type of register, কয়টা register register file এর কতটার register?
- sp (stack Pointer) এ কিভাবে data রাখি.
- R, I, J type instruction format এর machine code, Question এ function, code দেয়া থাকবে সেই
- Target address, Branch Address, Memory Address বের করা ★★★
- lui, ori দিয়ে কিভাবে 32 bit data operation করতে পারি, ★
- Character বা character array বা Ascii character কিভাবে handle করবে? Question এ I দেয়া char array বলতে পারে so চেষ্টা করবা;
- Branching far away theory concept. ★★★
- left shift, right shift, slt, slti, loop.
- and, or, not