Chapter - 2

Instructions: Language of the Computers

- MIPS Architecture (32 bit)
  - 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 অতিরিক্তগুলো data main memory তে store করা যাবে।

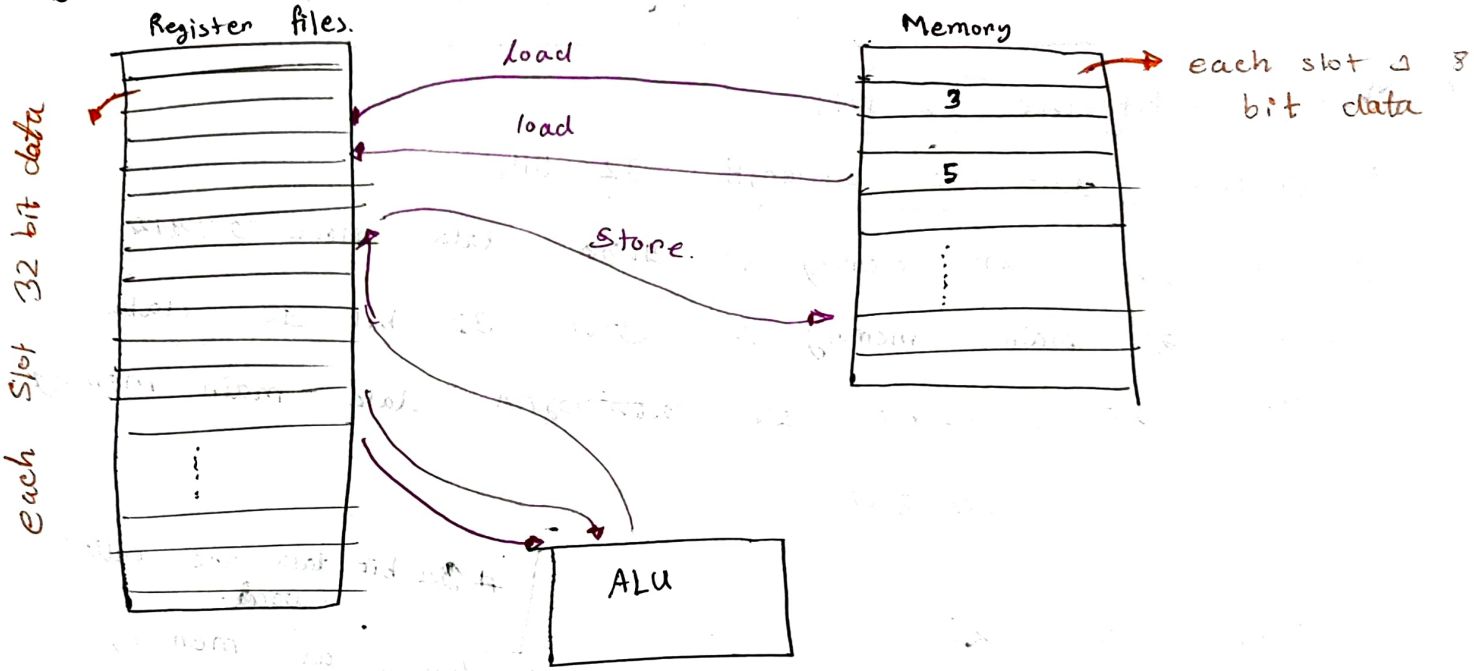
> # 32 bit data are called word.

- 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 পারবে, কিন্তু register file এ limited data রাখতে পারবে, কারণ reg. file এ data রাখার জন্য মাত্র 32 টা slot আছে।

CPU register file তখনই use করবে যখন বেশিবার data তার frequently use করা লাগবে।

যেকোনো arithmetic operation এ,



for example . 3 + 5 করতে,

memory থেকে 3 আর 5 reg file এ নিয়ে আসব,

then 3টার থেকে এ data ALU এ যায়; যেখানে

বিভিন্ন arithmetic logical operation হয়; then

ALU calculated result টা আবার reg file

এ return করে দিব, then reg file থেকে result

টা memory তে store করব;

b + c এর result : a তে store হবে

5 → 00000101 (memory তে)

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

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

Memory থেকে data load করি আর store করি,

Load = retrieve করা, memory থেকে নিচ্ছি.
↳ read operation

store = write operation

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

memory → register → ALU

memory address এর মাধ্যমে

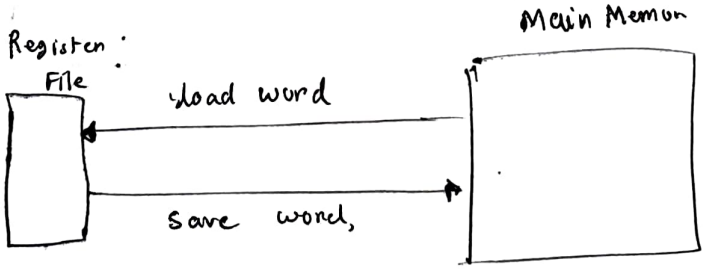
32 টা 32 combination

00000
00001
00010
00100
00011
...
11110
11111

Memory Operands:

- যখন data main memory তে থাকে then তখন তাকে register file এ পাঠানো হয়। Then reg file থেকে data ALU তে পাঠিয়ে Arithmetic operation হয়।
- reg. file একটা medium যেখানে frequently যেসব data use হয় সেগুলোকে main memory থেকে reg-file এ পাঠানো। Then তখন তাকে Instruction এর মধ্যে দিয়ে Arithmetic operation হওয়া দরকার সেগুলো perform হয়।
- Main memory is used for composite data (Arrays, Structures, Dynamic data, stack)
- Memory is byte address ✱
  ⇒ each address identifies an 8 bits
- Words are alling ed in memory. ✱
  ⇒ Adress must be a multiple of 4. ✓

8 bit = 1 byte.



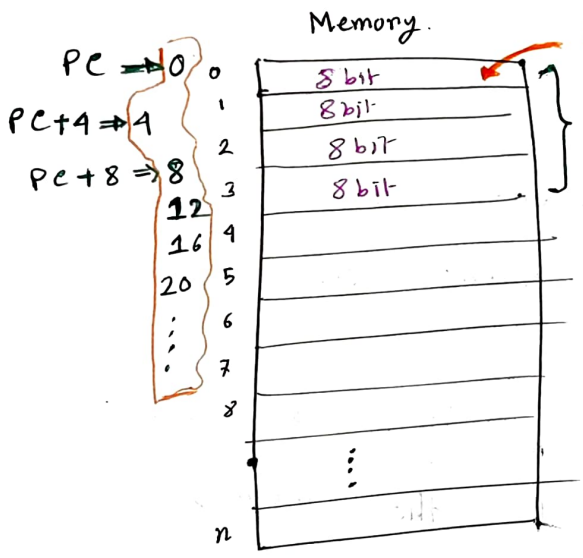
• MIPS is Big Endian.

⇒ MSB byte at least address of a word.

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

7 8 9 2 5

MSB ↓ LSB ↓



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.

★ MIPS architecture এ 32 bits follow করে। অর্থাৎ 32 bit এর একেকটা data represent করতে 4টা slot লাগে।

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

1st data occupy করছে memory এর (0-3)

2nd data occupy করছে memory এর (4-7)

Memory এর প্রত্যেকটা slot এর address n bit দ্বারা 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 (বা location এর) বানানো যাবে। প্রত্যেকটা location এ still 8 bit data hold করবে।

32 bit arch pe তে 4 করে increment করব।
64 bit arch pe 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 এ location বানানো যাবে।

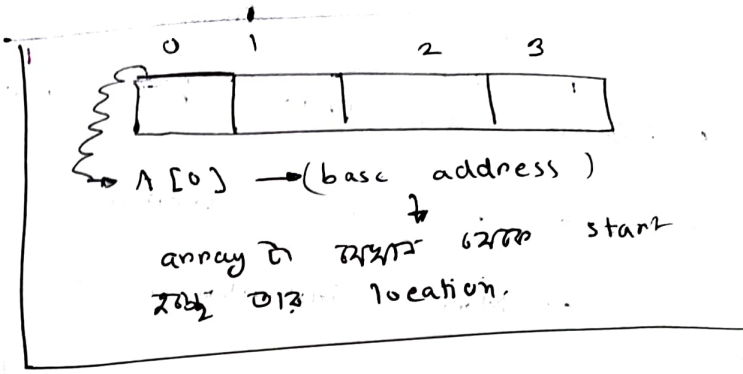
Memory address বের করা:

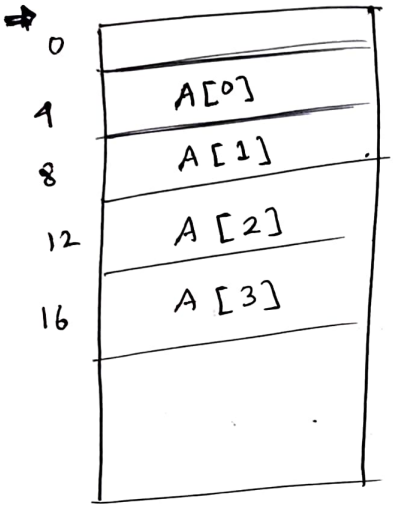
$g = h + A[3]$

Given that,

initial address = 4

Find the address of A[3].





কয়টা level cross করতে হবে,

= (3 × 4) = 12

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

= 4 + 12

= 16

base address of A in $S3

A[12] = h + A[8];

$2

equal এর left side এ array এর location থাকবে।

আর এই result টা main memory তে রাখতে হবে,

তাই main memory এর A[12] এ রাখবে।

```
lw  $t0, 32($s3)
add $t1, $s2, $t0
sw  $t1, 48($s3)
```

(des, source — lw)

(source, des — sw)

কয়টা main mem এর কোথায় location বা address আছে। destination হিসেবে সেটা থাকে।

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.
  - ⇒ Registers optimazation is important

MIPS Arch can support upto 32 address line

So an address could be 0x AB CD EF 1 2 3, in hexadecimal

8টা hex digit;

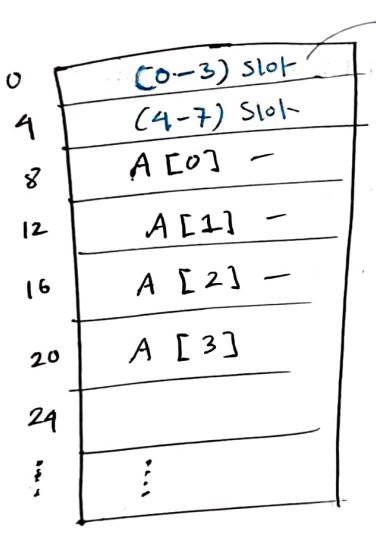
1 hex = 4 bit

∴ 8 hex digit = 32 bits

- memory তে কোন location এর address কত 32 bit- store করে রাখে।

- register file ত 32 টা register slot আছে, এবং প্রতিটা slot ত 1 word = 32 bit = 4 byte data থাকে।

- memory ত প্রত্যেক slot ত 8 bit থাকে, আমরা 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 4 টা slot সংরক্ষণ



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

⇒ From the figure,

base address of the array = 8 নং slot.

A[0] এর address থেকে 3 টা data skip করতে হবে, তার প্রত্যেক data এর জন্য 4 slot লাগে।

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

of A[3]

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

= 8 + 12

= 20.

Formula:

MIPS এ 32 bit এর জন্য

address = (4 x index number) + base address

64 bit architecture এর জন্য,

address = (8 x 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 এ সরাসরি immediate দেয়া

এতে integer value থাকে।

Register Operands :

Immediate Operation :

$f = \$s3 + 4$ ..... (i)

↳ addi $s3, $s3, 4.

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

$f = \$s3 - 4$

↳ addi $s3, $s3, -4

(ii) এই দুই example এ কোনো dependancy ছিলো না তাই register overwrite করতে পেরেছি।

dependancy থাকলে register overwrite করা যায় না।

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

$f \rightarrow \$s1$
$g \rightarrow \$s2$
$h \rightarrow \$s3$
$b \rightarrow \$s4$
$a \rightarrow \$s5$

↳ 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 লাগবে না; just কিছুক্ষণের জন্য কাজে লাগবে।

Saved register:

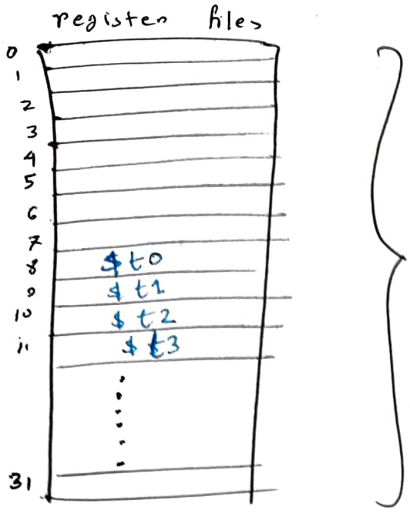
is used for storing variables or values যা পরে কাজে লাগতে পারে বা একাধিক কাজে লাগবে।

Saved register এ এমন value রাখবে যা over write করবে না,

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

$t = $9 ⇒ register এ এ এ এ এ

এই 8, 9 এর binary তে convert হবে।



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 তে 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 করার জন্য memory থেকে reg তে data নিয়ে then reg থেকে ALU তে গিয়ে calculate হয়, then result ALU থেকে register এ store হয়, then register থেকে finally memory তে store হয়,

Memory Operand [Example 1] : (Slide - 14) :

C- code :

g = h + A[8] → এইটা memory file এ আছে, so এটার address থেকে data টা fetch করার নিয়ম আমাদের।

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)

load operator

[$t0 register এ memory address থেকে data টা fetch করার নিয়ম আমাদের]

add $s1, $s2, $t0

---

load আর 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]

↳ যেখানে data store করবে।

এটার data memory থেকে 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, আমাদের memory এর size কত হতে পারে?

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

= 34 359 738 368 bit

= 4294 967 296 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 হচ্ছে $ Zero register.

⇒ Used for more operation.

Example:

$s1 = 7

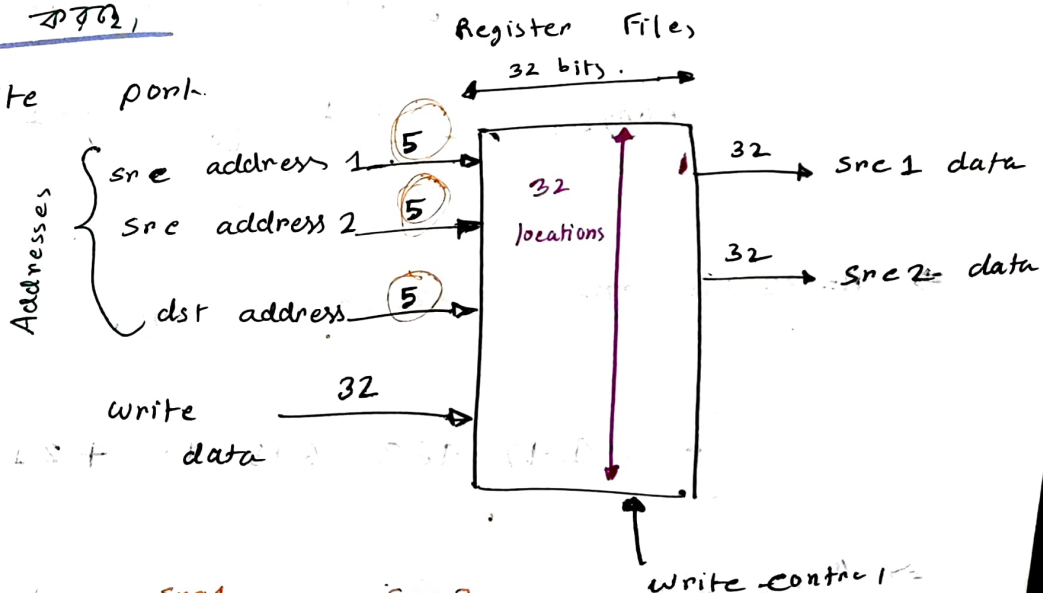
$t2 = $s1

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 পারি।
    - তাহলে তারপর register data collect করবে।
  - ⇒ One write port.



add $s1, $t0, $t2.

src1 src2

↳ destination register

dst adress তে destination থাকলে then data write করবে।

Unsigned binary integer:

⇒ আর্থ positive binary integer same করা,

⇒ n-bit system বা n bit number নিয়ে কাজ করলে,

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

8 bit এর জন্য একটা 8 bit register এ কত থেকে কত value store করাতে 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 967295

2's complement Signed Integer:

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

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

⇒ $n=32$ bit হলে তাহলে range:

$-2\,147\,483\,648$ to $+2147\,483\,642$

⇒ "Signed" binary এর first bit বা 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 (8bit) → 0000 0000 0000 0010
- −2 : 1111 1110 (8bit) → 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].

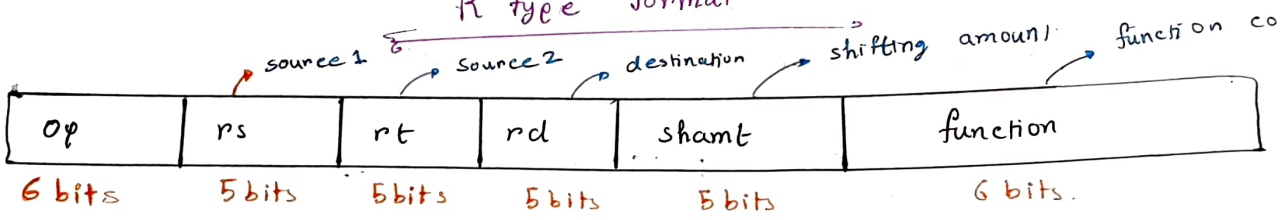
⇒ I type:

- lw, sw, addi, beq (branch equal), bne (branch not equal)
- slti, sltui → set less than I

⇒ J type:

- j (Jump)

R type format



R type এর জন্য Opcode সবসময় "000000" হয়।

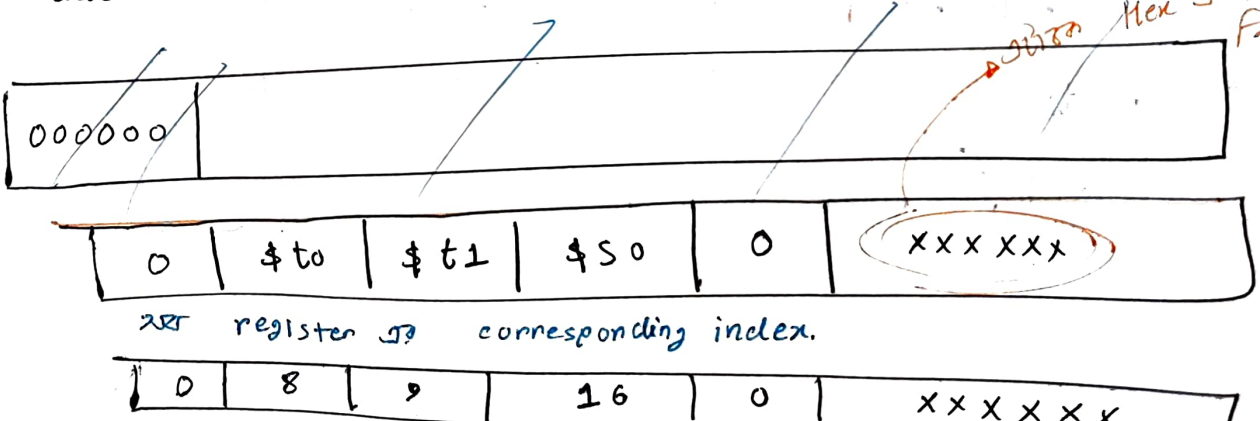
যখন কোনো shifting হচ্ছে না তখন shamt 0 হবে (5 bit তাতে)

তাহলে কত bit shifting হবে- সেটার coressponding binary (যে 5 bit তাতে)

function এর জন্য corresponding value থাকে function টা সেটার corresponding binary বিট, তাহলে 6 টা don't care. (XXXXXX)

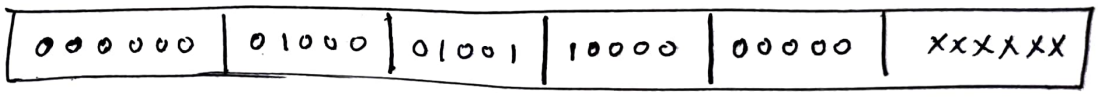
R type এ যেসব register তার register এর operation হয়।

add $s0, $t0, $t1.

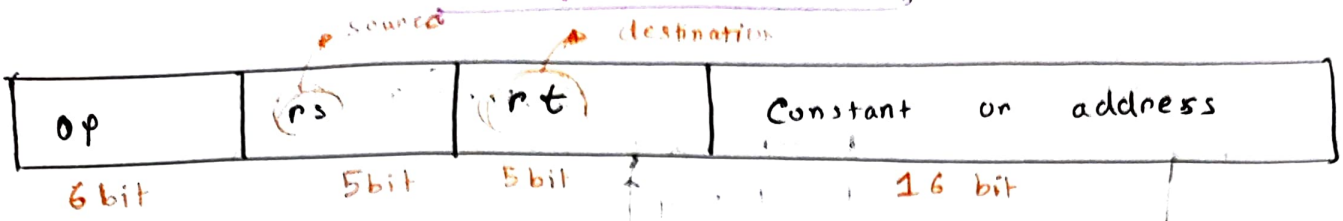


এটাকে Hex এ কিভাবে কিভাবে??

এরা register এর corresponding index.



I-Type format

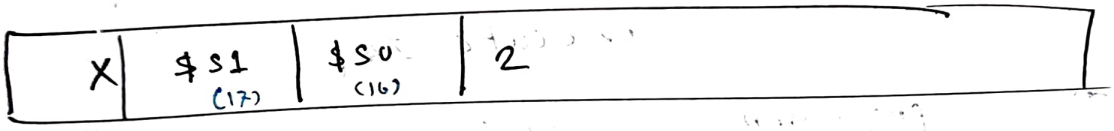


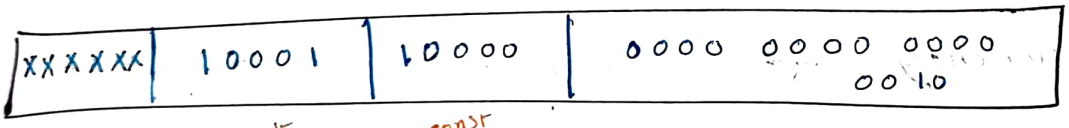
⇒ যদি opcode লেখা থাকে তাহলে কখনো লিখবে অন্যথায় "XXXXXX" লিখবে।

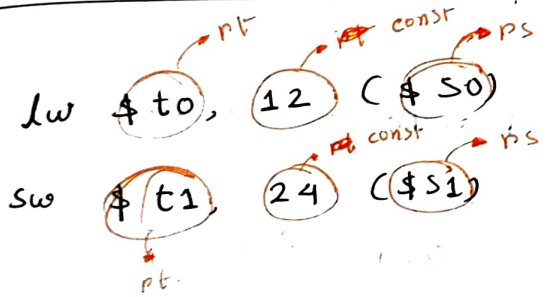
⇒ base address থাকলে rs এ লিখবে, offset part টুকু constant / address part এ লিখবে।

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

addi $s0, $s1, 2

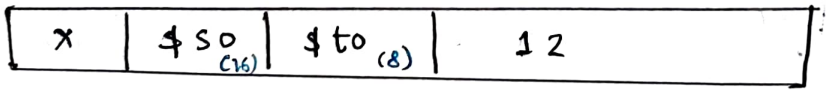


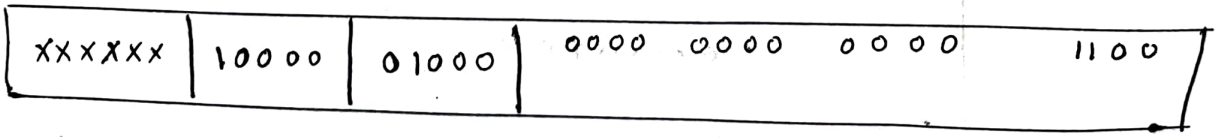




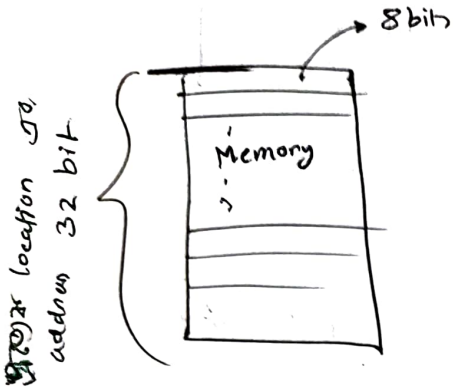
add $t1, $s1, 20 ($s3)
↳ এটাও I type format

• lw $t0, 12 ($s0)

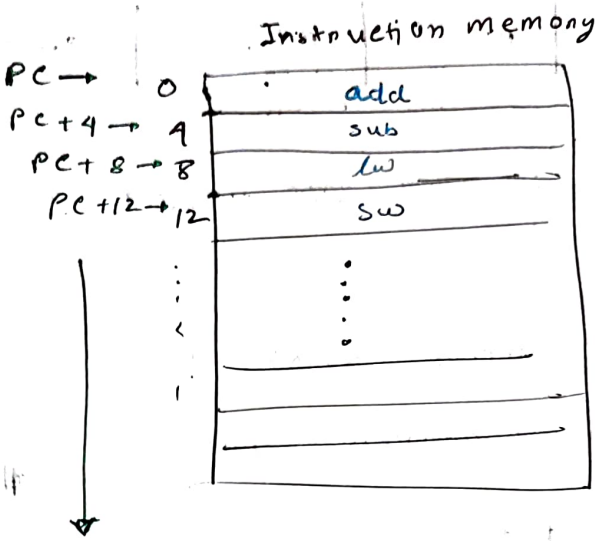




Conditional Operations and Branch Address.



∴ 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 করবে,
  - holds the address of the current instruction that is being executed
- Untill and unless কোনো condition satisfy হচ্ছে।

PC + 4 করব increase হয়; as memory এর 4টি slot-

মিলে 32 bit হয়.

PC (Program Counter) একটা 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 আর $b এর value equal কিনা

check করবে, equal হলে L1 function

এ jump করবে.

bne $8, $9, L2

↳ $8 আর $9 এর value যদি equal না-

হয় তবে L2 function এ jump

করবে.

Calculator ও neg দিলে 2's complement করে এবং 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, and then use sign extension to get the 16 bit representation of "-4"

এবং এই -4 এর 16 bit representation টা I type format এর constant বা address field এ বসিয়ে দিব।

branch instruction

- Conditional branch
  - যখন কোনো condition এর উপর base করে Jump করে কোনো location এ যাই।
- Unconditional branch
  - যখন কোনো condition ছাড়াই j (Jump) করতে বলা হবে।

(P.T.O)

C- code:

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

a is stored in $S1
b is stored in $S2

এখানে equal কিনা check করতে বলেছে ;
So আমরা MIPS এ চলমার সময় এখানে not equal কিনা check করব,

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

এবার, 1st instruction থেকে 8 টা slot skip করে আমরা else function এ jump করতে পারব।

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

register এর সাথে কোনো integer 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 :

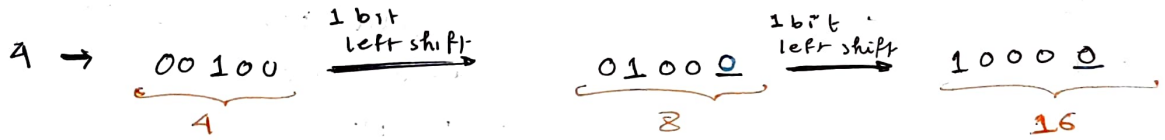
```
PC+0  : bne  $s3, $s4, Else
PC+4  : add  $s0, $s1, $s2
PC+8  : j    exit
PC+12 : Else:
PC+16 : sub  $s0, $s1, $s2
PC+20 : Exit:
```

Computer else block এর ভেতর একটা random integer generate করবে কিনা Suppose, 2 generate করবে কিনা, তার জন্য দুটি currently চলবে instruction এ জায়গা তার পরের 2 line (2×4 = 8 slot) skip করবে যাও,

2টা branching condition আছে,

beq
bne. } i - type instruction

5 bit register



কয় line skip 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 তত হবে.

General Representation:

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

32 bits architecture এর জন্য $x \times (4)$ করলাম multiplied by 2^2 বা 4 হবে, কারণ একটা data memory এর 4 টা slot থাকে আর তাহলে 2^2 দ্বারা multiply হচ্ছে এবং 32 bit architecture এর জন্য করলাম ~~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:
```

offset

branch address calculate করছি।

Current inst. থেকে 2 inst পরে L1 অবস্থিত

Current inst থেকে 8 slot পরে L1 আছে

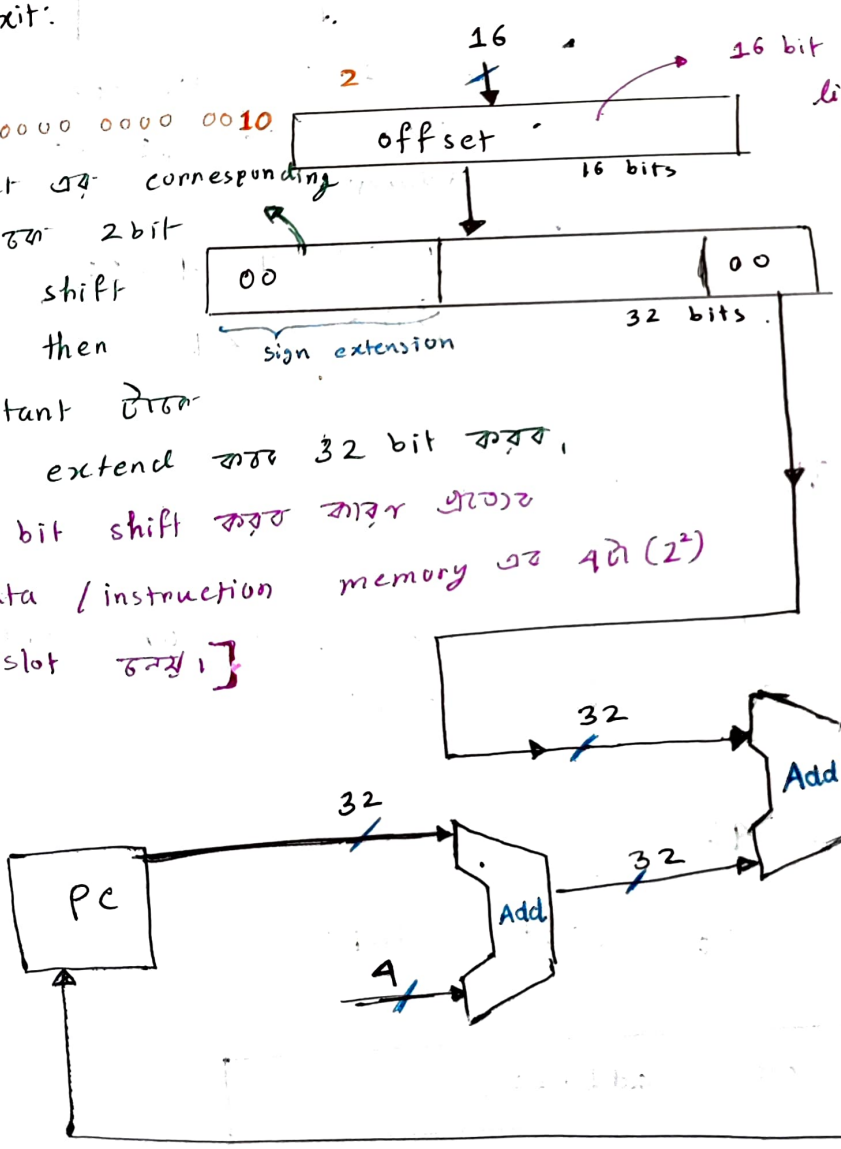
0000 0000 0000 0010

offset এর corresponding binary হল 2 bit left shift করব then resultant টাকে sign extend করে 32 bit করব।

[2 bit shift করার কারণ প্রত্যেক data / instruction memory এর 4টা (2^2) slot চলে।]

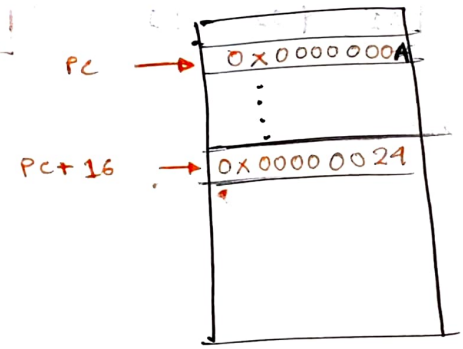
16 bit representation of কয়টা line skip করছি।

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



0x 0000 000A

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



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

$\therefore (0000\,0000\,A + 4 + 16)_{16}$

$= (0000.0024)_{16}$

J type Instruction

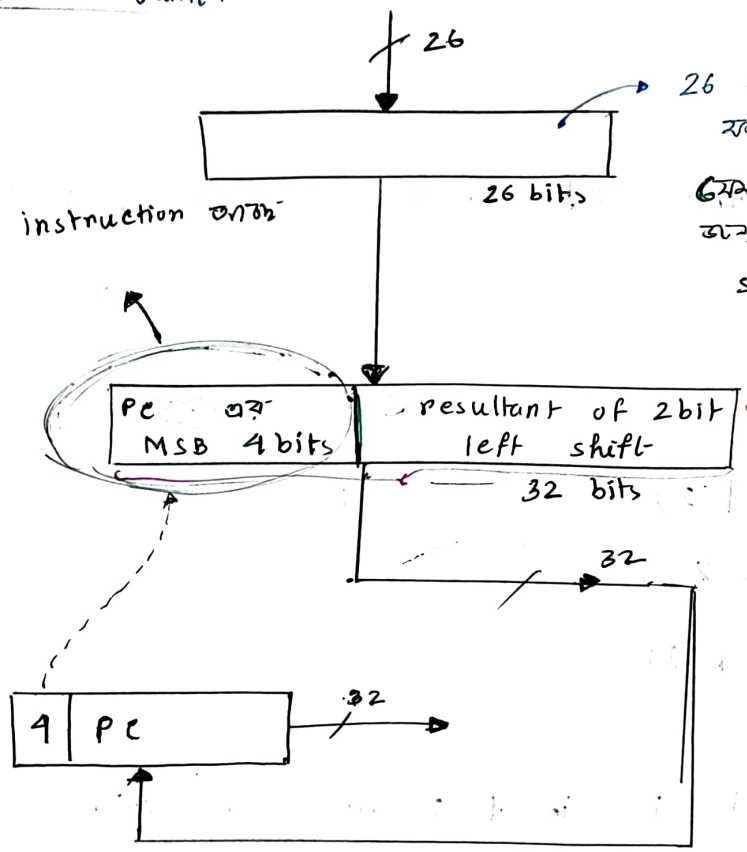
j label → 10 এর মান next 10 টি line skip করে level function এ যাও।

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

2 bit left shift

Opcode অংশ থাকলে opcode চলবে অন্যথায়, চুয়াটে don't care "XXXXXX" দিবে।

Branch Destination for Jump:



26 bit representation of যত line skip করতে হবে, যেমন: এখানে label এর জন্য 10 assume করছি; so 10 এর corresponding 26 bit এর binary করব,

26 bit number টাকে 2 bit left shift করব, তাহলে, 28 bit এর একটা number পাব, এখন আমি 4 digit এর জন্য PC এর first 4 MSB bit কে, সিব 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

MIPS Code Practice:

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

  Where base address of B and A are $s1 and $2 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 arr sm

→ 2^3 ← sll

⇒ MIPS code:

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

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

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

L2:
~~addi $t3, $t1~~
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 × (2^6) = 64z

6 bit left shift.

More Conditional Operation:

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

slt → set less than

set less than i

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

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

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

Format 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 save হবে।
অন্যথায়, $s3 < $s4 হলে
t1 এ 1 save হবে।

set করা মানে 1 assign করা।

reset করা মানে 0 assign করা।

তুলনার value এর মানের check করতে slti ব্যবহার করব as int এর মানের consider করছি।

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 করব।

MIPS:

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

Else:
   addi  $s0, $s0, 2

Exit:
```

$t0/16 = $t0/2^4 → 4 bit right shift.

signed comparison এর জন্য slt, slti

unsigned comparison এর জন্য sltu, sltui

Why not blt, bge, 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 address 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 ⇒ 26 bits rep then 2 bit left shift then PC ⇒ MSB 4 bit at the begining.

R type:

→ sll $t0 $s1 2 (shamt)

$t0 → rd, $s1 → rt

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

000000	00000	$17	$8	00010	xxxxx

(P.T.O)

☑ a and b are stoned in $s0 and $s1

C - code:

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

main code এ > ; s0 compare করার সময় less compare করব,

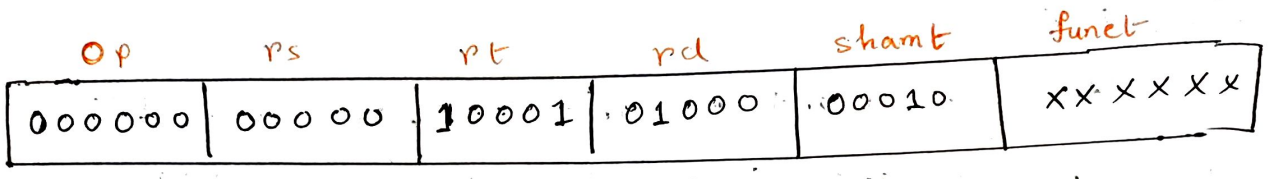
⇒ MIPS:

```
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



sll এবং srl operations এর ক্ষেত্রে rs সবসময়-

0 হবে,

C-code:

g = h + A[8]