1.

**ANS.**

#include<bits/stdc++.h>

using namespace std;

int size;

vector<pair<int, int> > free\_list[100000];

// Map used as hash map to store the starting

// address as key and size of allocated segment

// key as value

map<int, int> mp;

void initialize(int sz)

{

// Maximum number of powers of 2 possible

int n = ceil(log(sz) / log(2));

size = n + 1;

for(int i = 0; i <= n; i++)

free\_list[i].clear();

// Initially whole block of specified

// size is available

free\_list[n].push\_back(make\_pair(0, sz - 1));

}

void allocate(int sz)

{

// Calculate index in free list

// to search for block if available

int n = ceil(log(fabs(sz))/ log(2));

// Block available

if (free\_list[n].size() > 0)

{

pair<int, int> temp = free\_list[n][0];

// Remove block from free list

free\_list[n].erase(free\_list[n].begin());

cout << "Memory from " << temp.first

<< " to " << temp.second << " allocated"

<< "\n";

// map starting address with

// size to make deallocating easy

mp[temp.first] = temp.second -

temp.first + 1;

}

else

{

int i;

for(i = n + 1; i < size; i++)

{

// Find block size greater than request

if(free\_list[i].size() != 0)

break;

}

// If no such block is found

// i.e., no memory block available

if (i == size)

{

cout << "Sorry, failed to allocate memory \n";

}

// If found

else

{

pair<int, int> temp;

temp = free\_list[i][0];

// Remove first block to split it into halves

free\_list[i].erase(free\_list[i].begin());

i--;

for(; i >= n; i--)

{

// Divide block into twwo halves

pair<int, int> pair1, pair2;

pair1 = make\_pair(temp.first,

temp.first +

(temp.second -

temp.first) / 2);

pair2 = make\_pair(temp.first +

(temp.second -

temp.first + 1) / 2,

temp.second);

free\_list[i].push\_back(pair1);

// Push them in free list

free\_list[i].push\_back(pair2);

temp = free\_list[i][0];

// Remove first free block to

// further split

free\_list[i].erase(free\_list[i].begin());

}

cout << "Memory from " << temp.first

<< " to " << temp.second

<< " allocated" << "\n";

mp[temp.first] = temp.second -

temp.first + 1;

}

}

}

// Driver code

int main()

{

initialize(1024);

allocate(+20);

allocate(+35);

allocate(+90);

allocate(+40);

allocate(+240);

allocate(-40);

allocate(-20);

allocate(-90);

allocate(-35);

allocate(-240);

return 0;

}

2.

**ANS.**

Total bits of page table field = top-level bits + low-level bits

= 9 + 11

= 20 bits

Total bits of an offset = bits of and address – total bits of page table field

= 32 – 20

= 12 bits

Page size = 2bits of an offset

= 212

= 4096

= 4.096 KB

Page in an address space = 2Total bits of a page table field

= 220 pages

3.

**ANS.**

PC = 2020 =2020 / 512 = on 3rd page

SP = 10192 = 10192/512 = on 19th page. This means referenced data is on 18th page.

**1. Load word 8118 (a value of a procedure input parameter) into register 0**

As PC at 2020, an instruction is on 3rd page (2020/512 = 3)

Data referenced on the 15th page (8118/512 = 15)

**2. Push register 0 onto the stack**

As each instruction occupies 4 bytes, PC is at 2024 (2020 + 4).

This way, an instruction is on the 3rd page and SP is on the 18th page.

**3. Call a procedure at 4120 (the return address is stacked)**

PC is at 2028 (2024 + 4). An instruction is referenced on the 3rd page

SP is on the 18th page.

**4. Remove one word (the actual parameter) from the stack and**

After calling a procedure at 4120, PC is at 4120.

An instruction is on the 8th­ page.

SP is on the 18th page.

**5. Compare the actual parameter to constant 7**

PC is 4124 (4120 + 4).

An instruction is on the 8th page.

**6. If equal, jump to 5000**

PC is at 4128 (4124 + 4).

An instruction is on the 8th page.

An address to be jumped on the page 9th page.

4.

**ANS.**

Values of an inverted page table:

* It contains only one record for each frame of memory.
* The size of the inverted page table is proportional to that physical memory.

Drawbacks of an inverted page table:

* An inverted page table is sorted by a physical address. That’s why when page reference occurs; It might need to search the whole table since the match is met.
* Since only one virtual page recordfor each frame of memory, one physical page can’t contain more than one shared virtual address. Otherwise, it will result in a page fault. It involves sharing memory issues.

Value of TLB:

* It is fast – lookup hardware cache.
* If TLB hit occurs then frame number becomes available immediately. This way search becomes fast.

Drawbacks of TLB:

* If TLB miss occurs then one has to add page number and frame number to the TLB.
* If TLB is already full of records then replacement policies should run to remove old entry.
* TLB will remember only one-page entry at a time if one entry of memory is used by two pages

Inverted page table:

Entry no. | Page no. | PID

0 | 2 | Process 1   
1 | 3 | Process 2   
2 | 1 | Process 1   
3 | 0 | Process 2   
4 | 0 | Process 1

If match is found on ith entry of inverted page table then i will become a physical frame number.

|  |  |
| --- | --- |
| Virtual Page number | Physical page frame number |
| 0 | 4 |
| 1 | 2 |
| 2 | 0 |
| 3 | 1 |

5.



