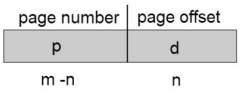
POWER OF 2



FOR MAIN MEMORY

Physical Address Space = Size of main memory (2^M)
Size of main memory = Total number of frames **x** Frame size
Frame size = Page size ($2^F = 2^P$)
Total number of frames = Size of main memory **/** Frame size
If number of frames in main memory = 2^M / 2^F = 2^{M-F} = 2^X, then number of bits in frame number = X bits
If Page size = 2^P, then number of bits in page = P bits
If size of main memory = 2^M, then number of bits in physical address = M bits

FOR PROCESS

Virtual Address Space = Size of process
Number of pages the process is divided = Process size / Page size
If process size = 2^X, then number of bits in virtual address space = X bits

FOR PAGE TABLE

Size of page table = Number of entries in page table **x** Page table entry size
Number of entries in pages table = Number of pages the process is divided
Page table entry size = Number of bits in frame number + Number of bits used for optional fields, if any.

在分页机制中，逻辑地址由页号（page number, p）和页内偏移（page offset, d）组成。图中展示了逻辑地址的结构，其中：

- 页号（page number, p）占用 $m - n$ 位。
- 页内偏移（page offset, d）占用 n 位。

页大小（Page Size）

页大小是由页内偏移（page offset, d）的位数决定的。具体来说，页大小等于 2^n 字节，其中 n 是页内偏移的位数。

详细解释

- **页内偏移（page offset, d）**：页内偏移决定了每一页的大小。假设页内偏移有 n 位，那么页内可以表示的地址范围就是 2^n。
- **页大小（page size）**：页大小就是这个范围，因此页大小为 2^n 字节。

主存（Main Memory）

1. **物理地址空间（Physical Address Space）**：
   - 物理地址空间是主存的大小，用 2^M 表示，其中 M 是物理地址的位数。
2. **主存大小（Size of main memory）**：
   - 主存大小等于页框数（Total number of frames）乘以页框大小（Frame size）。
3. **页框大小（Frame size）**：
   - 页框大小等于页大小（Page size），表示为 $2^F = 2^P$，其中 P 是页大小的位数。
4. **页框总数（Total number of frames）**：
   - 页框总数等于主存大小除以页框大小。
5. **位数计算**：
   - 如果主存中的页框数为 $2^M/2^F = 2^{M-F} = 2^X$，则页框编号所需的位数为 X 位。
   - 如果页大小为 2^P，则页内偏移所需的位数为 P 位。
   - 如果主存大小为 2^M，则物理地址的位数为 M 位。

进程（Process）

1. **虚拟地址空间（Virtual Address Space）**：
   - 虚拟地址空间等于进程的大小。
2. **页数（Number of pages）**：
   - 进程被分成的页数等于进程大小除以页大小。
3. **位数计算**：
   - 如果进程大小为 2^X，则虚拟地址空间所需的位数为 X 位。

页表（Page Table）

1. **页表大小（Size of page table）**：
   - 页表大小等于页表条目的数量乘以页表条目大小。
2. **页表条目数（Number of entries in page table）**：
   - 页表中的条目数等于进程被分成的页数。
3. **页表条目大小（Page table entry size）**：
   - 页表条目大小等于页框编号所需的位数加上用于可选字段（如访问位、修改位等）的位数。

例子解释

假设：

- 主存大小 2^{10} = 1024 单元（1 KB），那么 $M = 10$
- 页大小 2^2 = 4 单元，那么 $P = 2$
- 进程大小 2^5 = 32 单元，那么 $X = 5$

那么：

- 主存中的页框总数 $2^{10}/2^2 = 2^8$ = 256 个页框
- 页框编号所需的位数 $X = 8$
- 页内偏移所需的位数 $P = 2$
- 虚拟地址的位数 $X = 5$
- 进程的页数 $32/4 = 8$ 页

假设页表条目大小为页框编号的位数（8 位）加上可选字段（假设为 4 位），那么页表条目大小为 12 位。

In general, if the given address consists of 'n' bits, then using 'n' bits, 2^n locations are possible.

• Then, size of memory = 2^n x Size of one location.

If the memory is **byte-addressable**, then size of one location = 1 byte.

• Thus, size of memory = 2^n bytes.

If the memory is **word-addressable** where 1 word = m bytes, then size of one location = m bytes.

• Thus, size of memory = 2^n x m bytes.

内存大小的基本概念

1. **地址位数**:
   - 如果给定地址由 `n` 位组成，则使用 `n` 位可以表示 2^n 个位置（locations）。
   - 这些位置代表内存中的地址单元。
2. **内存大小**:
   - 内存的大小等于 2^n 乘以每个位置的大小。

字节寻址（Byte-addressable）

1. **字节寻址的定义**:
   - 在字节寻址中，每个地址单元表示一个字节。
   - 字节（Byte）是计算机内存的基本单位，通常等于 8 位。
2. **内存大小的计算**:
   - 如果内存是字节寻址的，那么每个位置的大小是 1 字节。
   - 因此，内存的大小等于 2^n 字节。

例子:

- 如果地址是 10 位，则可以表示 2^{10} = 1024 个位置。
- 因此，内存大小为 1024 字节（1 KB）。

字寻址（Word-addressable）

1. **字寻址的定义**：
   - 在字寻址中，每个地址单元表示一个字（Word）。
   - 一个字（Word）可以包含多个字节。例如，一个字可以是 2 字节、4 字节或 8 字节。
2. **内存大小的计算**：
   - 如果一个字（Word）的大小是 `m` 字节，那么每个位置的大小是 `m` 字节。
   - 因此，内存的大小等于 $2^n \times m$ 字节。

例子：

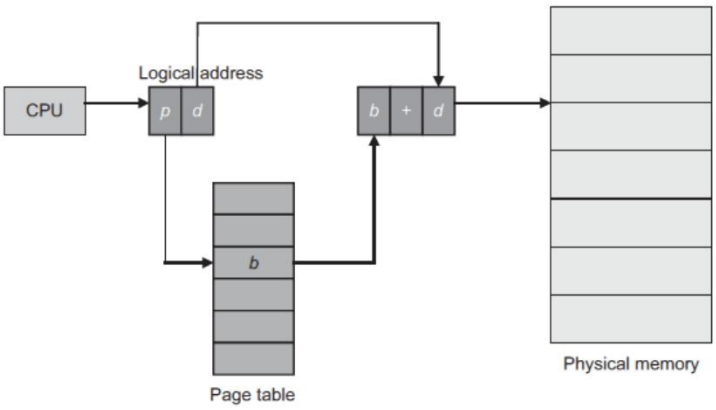
- 如果地址是 10 位，并且一个字是 4 字节，则可以表示 2^{10} = 1024 个位置。
- 因此，内存大小为 1024×4 = 4096 字节（4 KB）。

Address translation in paging

1. The processor generates a two-dimensional logical address, *(p, d)*.
2. The page number *p* is extracted from the logical address and is used as an index in the page table.
3. The base address *b*, corresponding to the page number, is retrieved.
4. *b* is added to the offset *d* to get the corresponding physical address.

The size of a page is determined by the computer's hardware (512 bytes to 1GB per page).

The size of the logical address is determined by the computer's architecture and the O.S.



这张图片解释了分页（paging）中的地址转换过程，详细描述了逻辑地址到物理地址的转换步骤。以下是详细解释：

地址转换过程

1. **生成逻辑地址**：
   - 处理器生成一个二维的逻辑地址，表示为 (p, d)，其中 p 是页号（page number），d 是页内偏移（offset）。
2. **提取页号**：
   - 从逻辑地址中提取页号 p，并将其用作页表（page table）中的索引。
3. **获取基地址**：
   - 从页表中检索与页号 p 对应的基地址 b。
4. **计算物理地址**：
   - 将基地址 b 加上偏移 d，得到对应的物理地址。

分页大小和逻辑地址大小

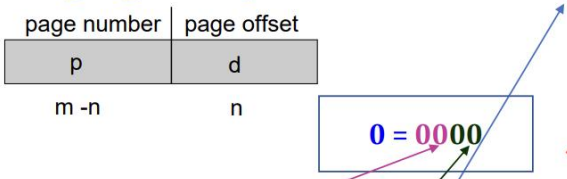
- **页大小（Page size）**：
  - 页的大小由计算机的硬件决定，通常在 512 字节到 1GB 每页不等。
- **逻辑地址大小（Size of the logical address）**：
  - 逻辑地址的大小由计算机的架构和操作系统决定。

示例过程

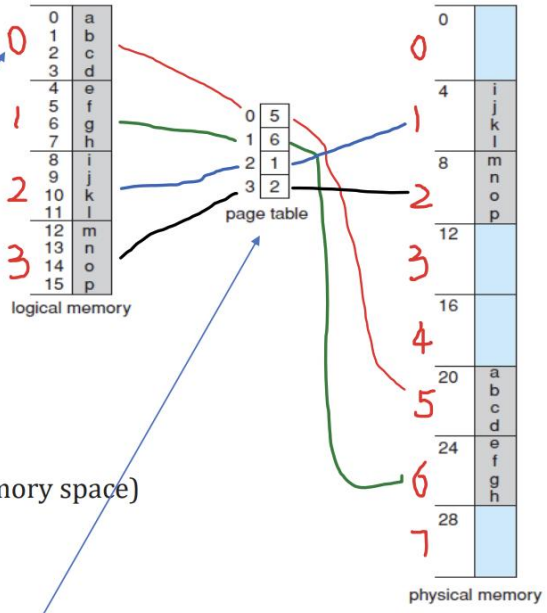
以下是具体示例过程的详细解释：

1. **生成逻辑地址**：
   - CPU 生成一个逻辑地址，比如说 (3, 15)，其中 3 是页号，15 是页内偏移。
2. **提取页号**：
   - 从逻辑地址 (3, 15) 中提取页号 3，作为页表的索引。
3. **获取基地址**：
   - 使用页号 3 从页表中查找基地址，比如说页表中存储的基地址是 200。
4. **计算物理地址**：
   - 将基地址 200 加上偏移 15，得到物理地址 215。

Paging Example 1



0 = 0000



Suppose the logical address is (2, 2)

(**page number** = 2 bits and **offset** = 2 bits)

- the logical address has m=4 bits

- 32-bytes of physical memory and

- 4-byte pages (8 pages can fit in physical memory space)

• **Example:**

- **Logical address 0** is in page = 0, offset=0

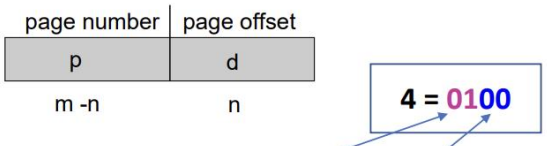
- According to the page table, **page 0 is in frame 5**

frame size = page size

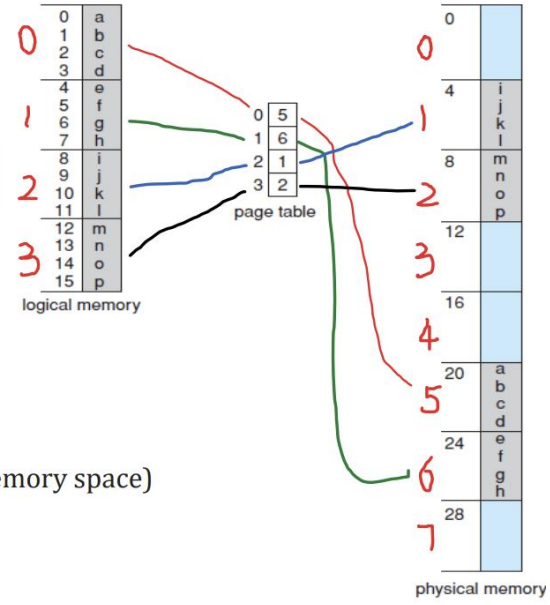
- Physical memory address is: **(Frame # x Frame size) + Offset**

- Physical memory address for **logical address 0** is : (5 x 4) + 0 = **20**

Paging Example 2



4 = 0100



Suppose the logical address is (2, 2)

(page number = 2-bit and offset = 2-bit)

- the logical address has m=4 bits

- **32-bytes of physical memory** and

- **4-byte pages** (8 pages can fit in physical memory space)

• **Example**:

- **Logical address 4** is in **page 1 at offset 0**

- According to the page table, page 1 is in frame 6

- Physical memory address is: (Frame # * Frame size) + Offset

- Physical memory address for **logical address 4** is : (6 x 4) + 0 = 24

Paging Example 3

There are **128 pages in** a **logical address space**, with a **page size of 1024** .

How many bits will be there in the logical address?

Solution

The logical address space contains 128 pages = 2^7 **pages**.

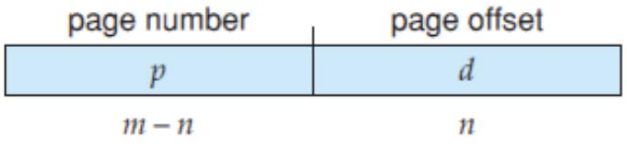
That means the **number of bits required for the page number**, **p = 7**

Page size, $2^d = 1024 = 2^{10}$

That means **the number of bits required for the offset, d = 10**

p + d = 17 bits

Therefore, **17 bits are required in the logical address**.



Paging Example 4

In a paging scheme, **16-bit** addresses are used with a page size of **512**.

If the logical address is **0000010001111101**, how many bits are used for the page number and offset?

Compute the **page number** and **offset** as well.

What will be the **physical address**, if the frame address corresponding to the computed page number is **15**.

Solution

logical address is **0000010001111101**

Logical address space = 2^{16} (16 bits)

Page size = 512 = 2^9

Number of bits required **for the page number** = 16 - 9 = **7**

Number of bits required **for offset, b = 9**

Page number is obtained by considering the **7 bits of the logical address**

i.e., Page number = $(0000010)_2$ = 2

Offset is obtained by considering **9 bits** of the logical address i.e., Offset = $(001111101)_2$ = 125

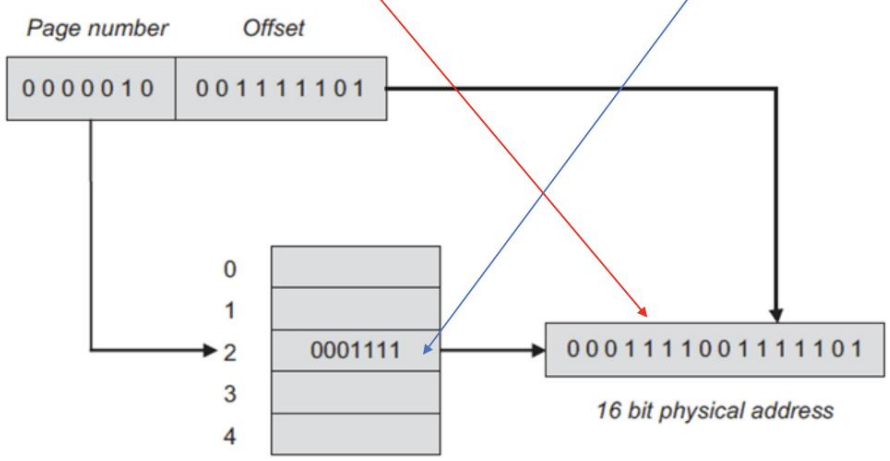
Frame address corresponding to the second page in bits = $(15)_{10}$ = 0001111

Appending the frame address to the offset, we get the

physical address = **0001111001111101**

Frame address corresponding to the second page in bits = $(15)_{10}$ = 0001111

Appending the frame address to the offset, we get the
physical address = 0001111001111101



CONTIGUOUS MEMORY ALLOCATION

Three processes **P1, P2**, and **P3** of size **21900**, **21950**, and **21990 bytes**, respectively, need space in the memory.

If equal-sized partitions of **22000 bytes** are allocated to P1, P2, and P3, will there be any fragmentation in this allocation?

Solution

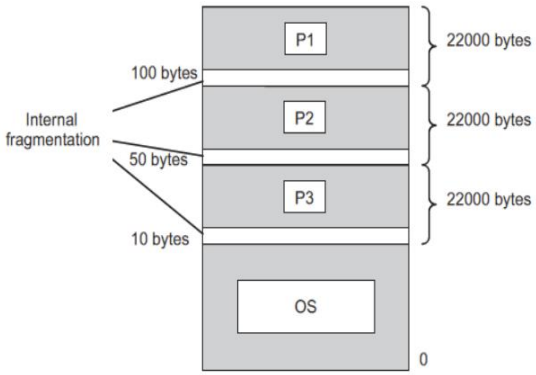
After allocating the partitions to the processes, the leftover space in each partition is estimated by the difference between partition size and process size.

The **leftover space** in the first partition = 22000 – 21900 = 100 bytes

The leftover space in the second partition = 22000 – 21950 = 50 bytes

The leftover space in the third partition = 22000 – 21990 = 10 bytes.

This leftover space in each partition is nothing but internal fragmentation, as shown in the following figure:



Dynamic Allocation Problem

Given the following information:

Job List:			Memory Block List:	
Job Number	**Memory Requested**		**Memory Block**	**Memory Block Size**
Job A	690K		Block 1	900K
Job B	275K		Block 2	910K
Job C	760K		Block 3	300K

a. Use the **first-fit algorithm** to indicate which memory blocks are allocated to each of the three arriving jobs.

b. Use the **best-fit algorithm** to indicate which memory blocks are allocated to each of the three arriving jobs.

a. Solution:

- **Job A is allocated to Block 1,**
- **Job B is allocated to Block 2,**
- **Job C is waiting**

b. Solution:

- **Job A is allocated to Block 1,**
- **Job B is allocated to Block 3,**
- **Job C is allocated to Block 2**

Hint!

Placement Algorithms:

First-fit: Allocate the first hole that's big enough

Best-fit: Allocate the smallest hole that's big enough

Worst-fit: Allocate the largest hole

FirstL:● 从内存的开始位置（即最低地址）开始扫描空闲内存块。

- 找到第一个能够满足请求大小的空闲块

Worst • **查找最大的空闲块：**

- • 扫描所有的空闲内存块，找到其中最大的那个。
- 如果有多个相同大小的最大块，通常选择第一个出现的。

- **分配内存：**

- • 将请求的内存分配给找到的最大空闲块。
- 如果分配后剩余的部分仍然足够大（大于或等于最小分配单元），则将其保留为新的空闲块；否则，这个空闲块就被完全使用掉了

PROBLEM-01

Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable.

2-byte memory addressable = memory that allows the hardware design to provide a single address and read or write 8 x 2 = 16 bits (2 byte) of data from that one address.

Solution

We have:

Number of locations possible with 22 bits = 2^{22} locations

It is given that the size of one location = 2 bytes

Thus, Size of memory

= 2^{22} x 2 bytes = 2^{23} bytes = 8 MB

PROBLEM-02

Calculate the number of bits required in the address for memory having size of 16 GB.
Assume the memory is 4-byte addressable.
4-byte memory addressable = memory that allows the hardware design to provide a single address and read or write 8 x 4 = 32 bits (4 byte) of data from that one address.

Solution

Let '*n*' number of bits are required.
Then, Size of memory = 2^n x 4 bytes.
Since, the given memory has size of 16 GB, so we have:
2^n x 4 bytes = 16 GB
2^n x 4 = 16 G
2^n x 2^2 = 2^{34} $\rightarrow$ 2^n = 2^{32}
So, n = 32 bits

PROBLEM-03

Consider a system with byte-addressable memory, 32-bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is _____.
a. 2
b. 4
c. 8
d. 16

Solution

Given
Number of bits in logical address = 32 bits
Page size = 4KB
Page table entry size = 4 bytes

Process Size

Number of bits in logical address = 32 bits
Process size = 2^{32} B = 4 GB

Number of Entries in Page Table

Number of pages the process is divided
= Process size / Page size = 4 GB / 4 KB = 2^{20} pages

Number of entries in page table = 2^{20} entries

Page Table Size

Page table size = Number of entries in page table x Page table entry size
= 2^{20} x 4 bytes = 4 MB

Option (B) is correct

PRACTICE PROBLEM BASED ON SEGMENTATION

Problem

Consider the following segment table:

Segment No.	Base	Length
0	1219	700
1	2300	14
2	90	100
3	1327	580
4	1952	96

Which of the following logical address will produce trap addressing error?

A. <0, 430>
B. <1, 11>
C. <2, 100>
D. <3, 425>
E. <4, 95>

Calculate the physical address if no trap is produced.

Solution

In a segmentation scheme, the generated logical address consists of two parts-

1. Segment Number
2. Segment Offset

We know-

- Segment Offset must always lie in the range [0, limit-1].
- If segment offset becomes greater than or equal to the limit of segment, then trap addressing error is produced.

Option-A: <0, 430>

Here,

- Segment Number = 0

- Segment Offset = 430

We have,

- In the segment table, limit of segment-0 is 700.
- Thus, segment offset must always lie in the range = [0, 700-1] = [0, 699]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1219 + 430 = 1649

Option-B: <1, 11>

Here,

- Segment Number = 1
- Segment Offset = 11

We have,

- In the segment table, limit of segment-1 is 14.
- Thus, segment offset must always lie in the range = [0, 14-1] = [0, 13]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 2300 + 11 = 2311

Option-C: <2, 100>

Here,

- Segment Number = 2
- Segment Offset = 100

We have,

- In the segment table, limit of segment-2 is 100.
- Thus, segment offset must always lie in the range = [0, 100-1] = [0, 99]

Now,

- Since generated segment offset does not lie in the above range, so request generated is invalid.

- Therefore, trap will be produced.

Option-D: <3, 425>

Here,

- Segment Number = 3
- Segment Offset = 425

We have,

- In the segment table, limit of segment-3 is 580.
- Thus, segment offset must always lie in the range = [0, 580-1] = [0, 579]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1327 + 425 = 1752

Option-E: <4, 95>

Here,

- Segment Number = 4
- Segment Offset = 95

We have,

- In the segment table, limit of segment-4 is 96.
- Thus, segment offset must always lie in the range = [0, 96-1] = [0, 95]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1952 + 95 = 2047

Option-(C) is correct

PRACTICE PROBLEMS BASED ON MULTILEVEL PAGING

Need

The need for multilevel paging arises when:

- The size of page table is greater than the frame size.
- As a result, the page table cannot be stored in a single frame in main memory.

Working

In multilevel paging,

- The page table having size greater than the frame size is divided into several parts.
- The size of each part is same as frame size except possibly the last part.
- The pages of page table are then stored in different frames of the main memory.
- To keep track of the frames storing the pages of the divided page table, another page table is maintained.
- As a result, the hierarchy of page tables get generated.
- Multilevel paging is done till the level is reached where the entire page table can be stored in a single frame.

Illustration of Multilevel Paging

Consider a system using paging scheme where:

- Logical Address Space = 4 GB
- Physical Address Space = 16 TB
- Page size = 4 KB

Now, let us find how many levels of page table will be required.

Number of Bits in Physical Address

Size of main memory = Physical Address Space = 16 TB = 2^{44} B
Number of bits in physical address = 44 bits

Number of Frames in Main Memory

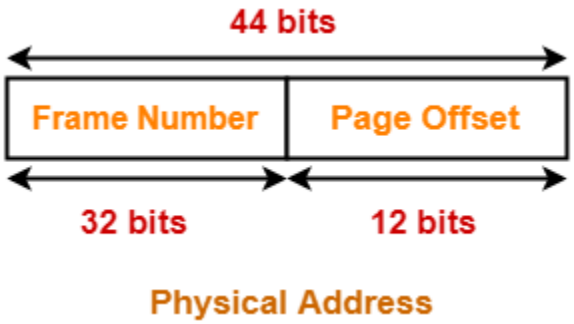
Number of frames in main memory
= Size of main memory / Frame size = 16 TB / 4 KB = 2^{32} frames

Number of bits in frame number = 32 bits

Number of Bits in Page Offset

We have,
Page size = 4 KB = 2^{12} B
Number of bits in page offset = 12 bits

Alternatively,
Number of bits in page offset = Number of bits in physical address – Number of bits in frame number
= 44 bits – 32 bits = 12 bits
So, Physical address is



Number of Pages of Process

Number of pages the process is divided

= Process size / Page size = 4 GB / 4 KB = 2^{20} pages

Inner Page Table Size

Inner page table keeps track of the frames storing the pages of process.

Inner Page table size = Number of entries in inner page table x Page table entry size

= Number of pages the process is divided x Number of bits in frame number

= 2^{20} x 32 bits = 2^{20} x 4 bytes = 4 MB

Now, we can observe:

- The size of inner page table is greater than the frame size (4 KB).
- Thus, inner page table can not be stored in a single frame.
- So, inner page table has to be divided into pages.

Number of Pages of Inner Page Table

Number of pages the inner page table is divided

= Inner page table size / Page size = 4 MB / 4 KB = 2^{10} pages

Now, these 2^{10} pages of inner page table are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Inner Page Table

Number of page table entries in one page of inner page table = Page size / Page table entry size =
= Page size / Number of bits in frame number = 4 KB / 32 bits
= 4 KB / 4 B = 2^{10}

Number of Bits Required to Search an Entry in One Page of Inner Page Table

One page of inner page table contains 2^{10} entries.
Thus, Number of bits required to search a particular entry in one page of inner page table = 10 bits

Outer Page Table Size

Outer page table is required to keep track of the frames storing the pages of inner page table.

Outer Page table size

= Number of entries in outer page table x Page table entry size

= Number of pages the inner page table is divided x Number of bits in frame number

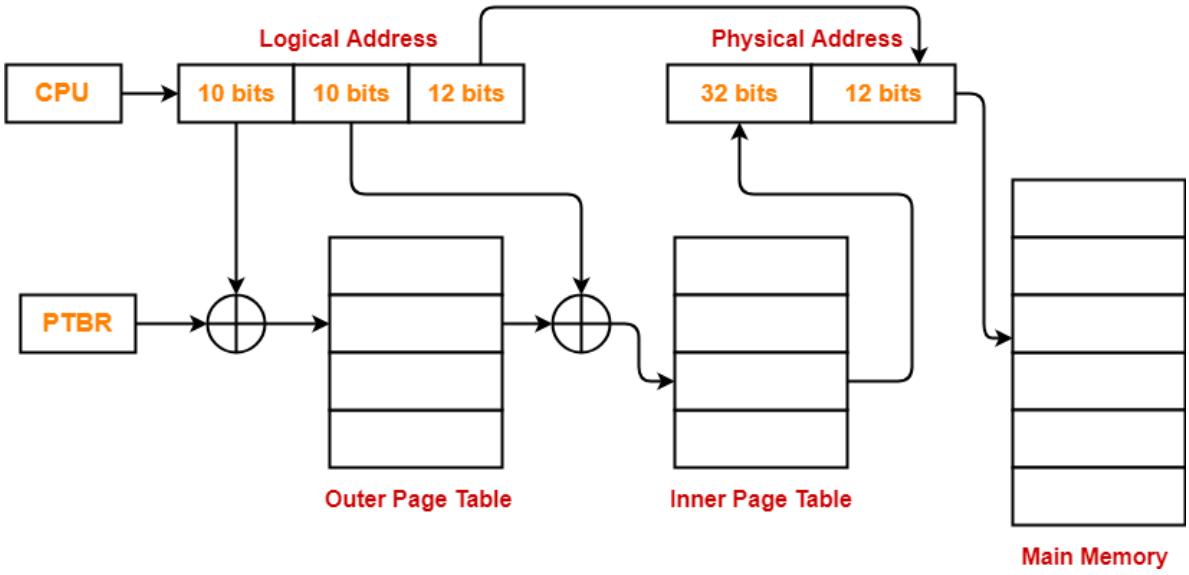
= 2^{10} x 32 bits = 2^{10} x 4 bytes = 4 KB

Now, we can observe:

- The size of outer page table is same as frame size (4 KB).
- Thus, outer page table can be stored in a single frame.
- So, for given system, we will have two levels of page table.
- Page Table Base Register (PTBR) will store the base address of the outer page table.

Number of Bits Required to Search an Entry in Outer Page Table

Outer page table contains 2^{10} entries.
Thus, Number of bits required to search a particular entry in outer page table = 10 bits
The paging system will look like as shown below:



Problem

Consider a system using multilevel paging scheme. The page size is 1 MB. The memory is byte addressable and virtual address is 64 bits long. The page table entry size is 4 bytes. Find:

1. How many levels of page table will be required?
2. Give the divided physical address and virtual address.

Solution

Given

- Virtual Address = 64 bits
- Page size = 1 MB
- Page table entry size = 4 bytes

Number of Bits in Frame Number

We have,
Page table entry size
= 4 bytes = 32 bits
Thus, Number of bits in frame number = 32 bits

Number of Frames in Main Memory

We have, Number of bits in frame number = 32 bits
Number of frames in main memory = 2^{32} frames

Size of Main Memory

Size of main memory
= Total number of frames x Frame size
= 2^{32} x 1 MB = 2^{52} B
Thus, Number of bits in physical address = 52 bits

Number of Bits in Page Offset

We have,
Page size = 1 MB = 2^{20} B
Thus, Number of bits in page offset = 20 bits

Alternatively,
Number of bits in page offset
= Number of bits in physical address – Number of bits in frame number
= 52 bits – 32 bits = 20 bits

Process Size

Number of bits in virtual address = 64 bits
Process size = 2^{64} bytes

Number of Pages of Process

Number of pages the process is divided
= Process size / Page size
= 2^{64} B / 1 MB = 2^{64} B / 2^{20} B = 2^{44} pages

Inner Page Table Size

Inner page table keeps track of the frames storing the pages of process.

Inner page table size

= Number of entries in inner page table x Page table entry size

= Number of pages the process is divided x Page table entry size

= 2^{44} x 4 bytes = 2^{46} bytes

Now, we can observe:

- The size of inner page table is greater than the frame size (1 MB).
- Thus, inner page table can not be stored in a single frame.
- So, inner page table has to be divided into pages.

Number of Pages of Inner Page Table

Number of pages the inner page table is divided

= Inner page table size / Page size

= 2^{46} B / 1 MB = 2^{46} B / 2^{20} B = 2^{26} pages

Now, these 2^{26} pages of inner page table are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Inner Page Table

Number of page table entries in one page of inner page table
= Page size / Page table entry size
= 1 MB / 4 B = 2^{20} B / 2^{2} B = 2^{18} entries

Number of Bits Required to Search an Entry in One Page of Inner Page Table

One page of inner page table contains 2^{18} entries.

Number of bits required to search a particular entry in one page of inner page table = 18 bits

Outer Page Table-1 Size

Outer page table-1 is required to keep track of the frames storing the pages of inner page table.

Outer page table-1 size

= Number of entries in outer page table-1 x Page table entry size

= Number of pages the inner page table is divided x Page table entry size

= 2^{26} x 4 bytes

= 2^{28} bytes

= 256 MB

Now, we can observe:

- The size of outer page table-1 is greater than the frame size (1 MB).
- Thus, outer page table-1 can not be stored in a single frame.
- So, outer page table-1 has to be divided into pages.

Number of Pages of Outer Page Table-1

Number of pages the outer page table-1 is divided
= Outer page table-1 size / Page size
= 256 MB / 1 MB
= 256 pages

Now, these 256 pages of outer page table-1 are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Outer Page Table-1

Number of page table entries in one page of outer page table-1
= Page size / Page table entry size
= 1 MB / 4 B
= 2^{20} B / 2^{2} B
= 2^{18} entries

Number of Bits Required to Search an Entry in One Page of Outer Page Table-1

One page of outer page table-1 contains 2^{18} entries.
Number of bits required to search a particular entry in one page of outer page table-1 = 18 bits

Outer Page Table-2 Size

Outer page table-2 is required to keep track of the frames storing the pages of outer page table-1.

Outer page table-2 size

= Number of entries in outer page table-2 x Page table entry size

= Number of pages the outer page table-1 is divided x Page table entry size

= 256 x 4 bytes

= 1 KB

Now, we can observe-

- The size of outer page table-2 is less than the frame size (1MB).
- Thus, outer page table-2 can be stored in a single frame.
- In fact, outer page table-2 will not completely occupy one frame and some space will remain vacant.
- So, for given system, we will have three levels of page table.
- Page Table Base Register (PTBR) will store the base address of the outer page table-2.

Number of Bits Required to Search an Entry in Outer Page Table-2

Outer page table-2 contains $256 = 2^8$ entries.
Number of bits required to search a particular entry in outer page table-2 = 8 bits

The paging system will look like as shown below: