Paging

- Both fixed partitioning (equal & unequal size), and Dynamic partitioning are inefficient.

Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***n*** pages, need to find *n* free frames and load program
- Set up a page table to translate logical to physical addresses
- Internal fragmentation

Paging

- Partition memory into small equal-size chunks and divide each process into the same size chunks
- The chunks of a process are called pages and chunks of memory are called frames
- Operating system maintains a page table for each process
  - contains the frame location for each page in the process
  - memory address consist of a page number and offset within the page

- Only the fraction of last page is wasted.
- With paging logical to physical address translation is still done by processor hardware.
- With the help of logical address (page no. & offset) the processor uses page table to produce the physical address (frame no. & offset)

Paging

0	A.0	0	A.0	0	A.0
1	A.1	1	A.1	1	A.1
2	A.2	2	A.2	2	A.2
3	A.3	3	A.3	3	A.3
4	B.0	4		4	D.0
5	B.1	5		5	D.1
6	B.2	6		6	D.2
7	C.0	7	C.0	7	C.0
8	C.1	8	C.1	8	C.1
9	C.2	9	C.2	9	C.2
10	C.3	10	C.3	10	C.3
11		11		11	D.3
12		12		12	D.4
13		13		13	
14		14		14	

Page Tables for Example

0	0
1	1
2	2
3	3

Process A

0	---
1	---
2	---

Process B

0	7
1	8
2	9
3	10

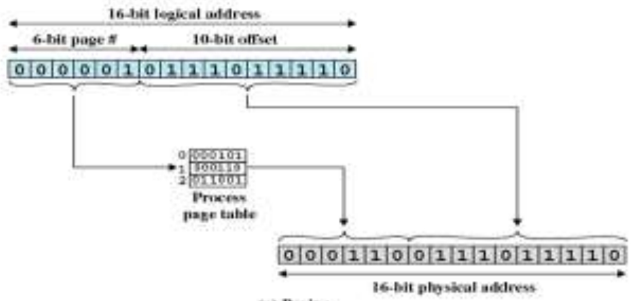
Process C

0	4
1	5
2	6
3	11
4	12

Process D

13
14

Free Frame List



(a) Paging

Background

- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

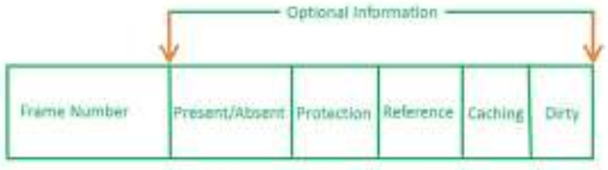
Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory

Translation Look Aside Buffer

- In normal sense a page table entry (see Figure below),
- the question is where to place the page table, such that overall access time (or reference time) will be less.
- Initially, some people thought of using registers to store page table, as they are high-speed memory so access time will be less.
- but the problem is register size is small (in practical, it can accommodate maximum of 0.5k to 1k page table entries) and process size may be big hence the required page table will also be big, so registers may not hold all the PTE's of Page table. So this is not a practical approach.



PAGE TABLE ENTRY

- To overcome this size issue, the entire page table was kept in main memory. but the problem here is two main memory references are required.
- To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used.

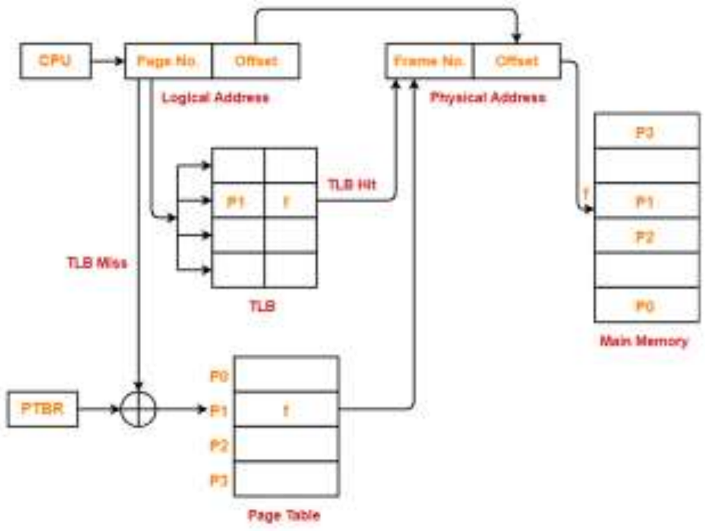
Translation Look Aside Buffer

- In a paging scheme using TLB,
- The logical address generated by the CPU is translated into the physical address using following steps-
- Step-1
  - TLB is checked to see if it contains an entry for the referenced page number.
  - The referenced page number is compared with the TLB entries all at once.
- Now, two cases are possible-
- **Case-01: If there is a TLB hit-**
  - If TLB contains an entry for the referenced page number, a TLB hit occurs.
  - In this case, TLB entry is used to get the corresponding frame number for the referenced page number.
- **Case-02: If there is a TLB miss-**
  - If TLB does not contain an entry for the referenced page number, a TLB miss occurs.
  - In this case, page table is used to get the corresponding frame number for the referenced page number.
  - Then, TLB is updated with the page number and frame number for future references.

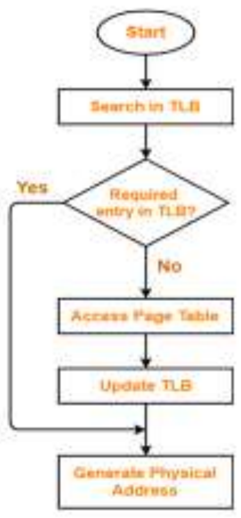
Translation Look Aside Buffer

- The logical address generated by the CPU is translated into the physical address using following steps-
- Step-2
  - After the frame number is obtained, it is combined with the page offset to generate the physical address.
  - Then, physical address is used to read the required word from the main memory.

Translation Look Aside Buffer



Translating Logical Address into Physical Address



Flowchart

Other important points about TLB

- Unlike page table, there exists only one TLB in the system.
- So, whenever context switching occurs, the entire content of TLB is flushed and deleted.
- TLB is then again updated with the currently running process.
- When a new process gets scheduled-
  - Initially, TLB is empty. So, TLB misses are frequent.
  - With every access from the page table, TLB is updated.
  - After some time, TLB hits increases and TLB misses reduces.
- The time taken to update TLB after getting the frame number from the page table is negligible.
- Also, TLB is updated in parallel while fetching the word from the main memory.
- The advantages of using TLB are-
  - TLB reduces the effective access time.
  - Only one memory access is required when TLB hit occurs.