Memory Management

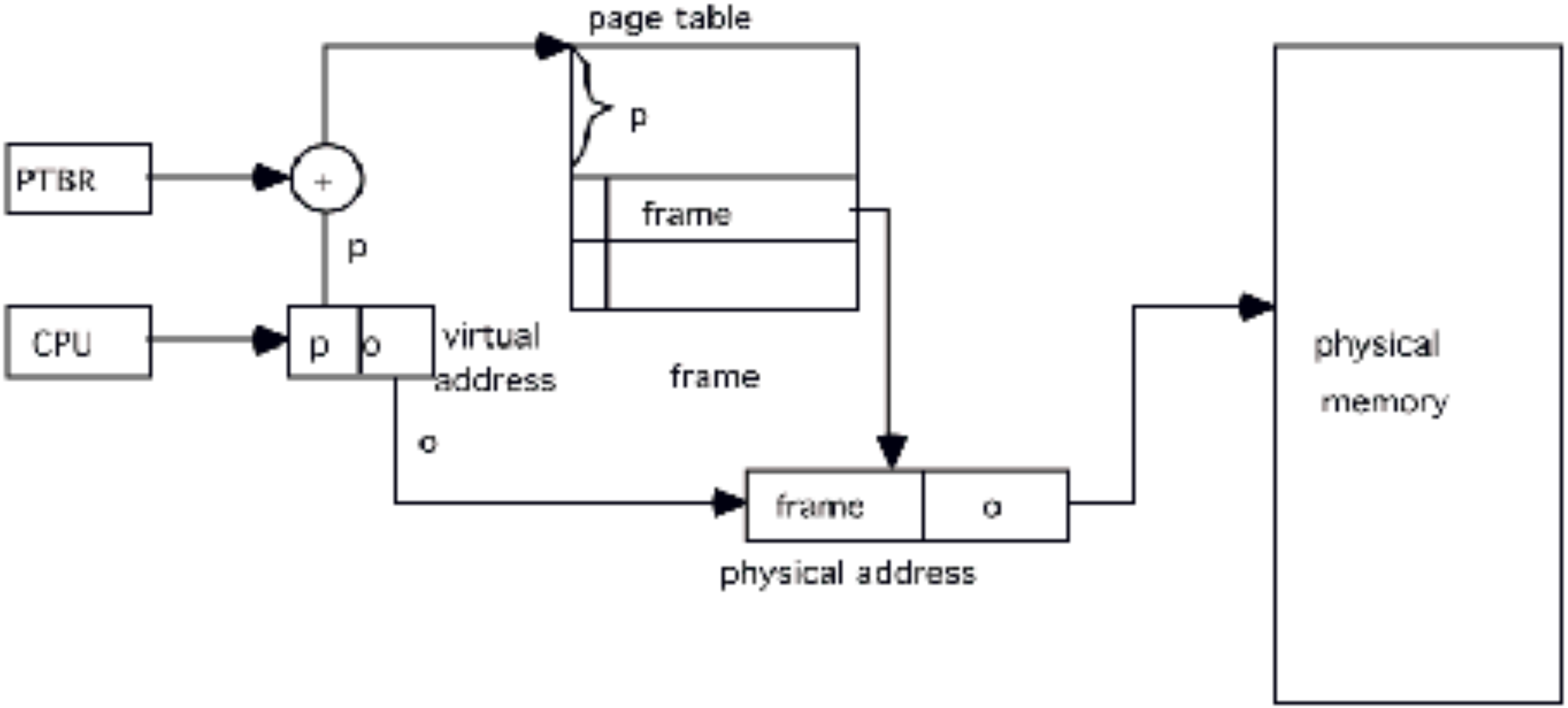
Paging

- Program memory need not be contiguous
  - Solves problem of compaction in MVT
- How it works
  - Virtual address split in two
  - High bits represent page number
  - Low bits represent page offset
  - Page table has the address of each page frame in physical memory

Address Translation

- Address of page table stored in Page Table Base Register (PTBR)
- Add this to page table number to get address of page table entry
- Get physical address of frame
- Add offset to that address to get physical memory address corresponding to virtual address

Address Translation



Page Frames

- Frame is physical memory into which a page is put
- Page is unit of virtual memory put into physical memory
- Both are same size, defined by hardware
    - Usually 1024, 2048, 4096, or 8192 words or bytes per page
- If page contains *p* words, virtual address *va* produces:
    - page number = *va* / *p*
    - page offset = *va* % *p*
- If *p* is power of 2, then
    - page number = high order bits of *va*
    - page offset = low order bits of *va*

Example

- Virtual address is 7
- Page size is 2 words (not realistic)
- So page number: 7 / 2 = 3
- And page offset: 7 % 2 = 1
- Frame is frame number 5
- Frame size = page size = 2, so base of frme is 5x2 = 10
- Offset is 1, so physical address of virtual word at 7 is 10 + 1 = 11

page number	frame number
0	4
1	3
2	1
3	5
4	2
5	6

Process Scheduling

- Process memory size given in pages
- Process has *n* pages, so it needs *n* frames
- External fragmentation: *none*
- Internal fragmentation: at most $p-1$, where *p* is page size
  - So expected internal fragmentation per process is $p/2$

Storing Page Tables

- Small number of pages: use register for the page table
  - Loading, modifying these are privileged
  - Address translation very efficient, as registers use high speed logic
- Large number of pages: store page table in memory
  - For this (usual) case, you need the PTBR

Context Switching

- Page tables means changing 1 register, the PTBR
  - Value of swapped out process goes into the process’ PCB
  - When swapped in, the PTBR is loaded with saved value

Memory Access Time

- Problem: now memory references are twice as slow!
  - First memory access is to the page table, to get physical address associated with virtual page
  - Next memory access is to the desired address
- Effective memory access time (EMAT)
  - Actual time needed to access memory

Optimizations

- Use *cache* (*associative memory*, *look-aside memory*)
- Registers store (key, value) pair
- Given key, cache hardware compares key to stored keys at once, returning corresponding values
- But this memory is expensive!

Caches and Paging

- Put some page table entries into cache
  - Usually too many to put them all there
- Here’s what happens:
  - Get page number from virtual address
  - Check cache for corresponding frame number
  - If there, use it
    - Checking is much faster than a main memory access
  - If not there, access memory to get frame number
    - And load it into the cache
  - Add page offset to frame address

Hit Ratio

- Percent of time page number is found in cache
- Used to measure efficiency of caching
- Example: 50ns to search cache, 750ns to access memory
  - In cache: access time is 50ns + 750ns = 800ns
  - Not in cache: access time is 50ns + 750ns + 750ns = 1550ns

Effective Memory Access Time

- Average time needed for a memory reference:

hit ratio × time needed to reference page when page number in cache
+
(1 – hit ratio) × time needed to reference page when page number not in cache

Examples

- Building on the earlier one:
- 80% hit ratio: EMAT is 0.8 × 800ns + (1–0.8) × 1550ns = 956ns
  - Slowdowb is (956 – 750) / 750 = 0.274 = 27.4%
- 90% hit ratio: EMAT is 0.9 × 800ns + (1–0.9) × 1550ns = 875ns
  - Slowdown is (875 – 750) / 750 = 0.167 = 16.7%

Sharing Pages

- Re-entrant code: code that is not altered
  - Also called pure code, non-self modifying code
- Just put appropriate entries in page table!
- Example: program instructions take up 250 pages, data at most 200 pages
  - With sharing: 250 + 200 + 200 = 650 frames used
  - Without sharing: 250 + 200 + 250 + 200 = 900 frames used
- Note: critical that shared pages not be altered!
  - This means the operating system must enforce this

Protection

- Protection bits associated with each page
  - Kept in the page table
- 1 bit to indicate if page is read/write or read only
- 1 bit to indicate whether value in page table is valid or invalid
- More bits for other forms of protection
- So during computation of physical address, operating system can verify the access is appropriate
  - If not (writing a read only page, accessing an invalid entry), trap to operating system

Trapping Illegal Addresses

- Uses the bits to allow or disallow access
- Example:
  - Page size 2048 words per page
  - Program uses addresses 0 . . 10040 (5 pages)
- Suppose it tries to access page 6
  - That’s memory address 12288, which is not in program’s address space
- Trap to operating system!

Example of Fragmentation

- Page size 2048 words per page
- Program uses addresses 0 . . 10068 (5 pages)
- 5 pages uses 10240 addresses
- So internal fragmentation is 200 words (space left over in page 5)
    - Cannot deny access to those words as you can’t block access to specific *words*, just pages
    - It’s all of a page or no part of a page

Alternate View of Memory

- User view: program sees memory as a contiguous memory space
  - The memory is divided into equally-sized blocks of instructions or data (pages)
- OS view: OS sees user's program scattered throughout physical memory
- How do we reconcile these?

Reconciliation

- Address translation mechanism maps virtual memory locations to physical locations under control of operating systems
- So physical and virtual addresses may be different
- Example: XNS-940 had virtual address space of 14 bits, but physical address space of 16 bits
  - Page number (3 bits) referenced page table entry to get 5 bit frame number
  - So 4 times as much physical memory as virtual memory
- Widely used when address spaces grow
  - Example: 16 bit address space grows to 32 bit address space

Reconciliation

- Widely used when address spaces grow
    - Example: 16 bit address space grows to 32 bit address space
    - Virtual addresses still 16 bits
    - Physical addresses become 32 bits
- Can’t use more memory than before

Tracking Used Frames

- Operating system keeps track of what frames are used and which are not, total number of frames, etc.
- Stored in a global frame table
  - Like a page table, but has one entry per *frame*, not per page
  - Entries indicate if frames are allocated and, if so, to which process

Segmentation

- View program as collection of variable-sized segments
  - 1 segment per function or data structure
  - Segments are of variable length
  - Words identified by offsets into segments
- Called *segmentation*
  - Virtual address space is collection of segments
  - Segments have name and length
  - Addresses specify name of segment, offset into that segment

Segment Names

- These are numbers
  - It’s the easiest thing to do
- Segments often generated by compiler — look for something like “.text *n*”, which says what follows is in segment number *n*
- Example: C program may have:
  - segment for global variables
  - segment for program stack
  - segments for instructions of each function
  - segments for local variables of each function

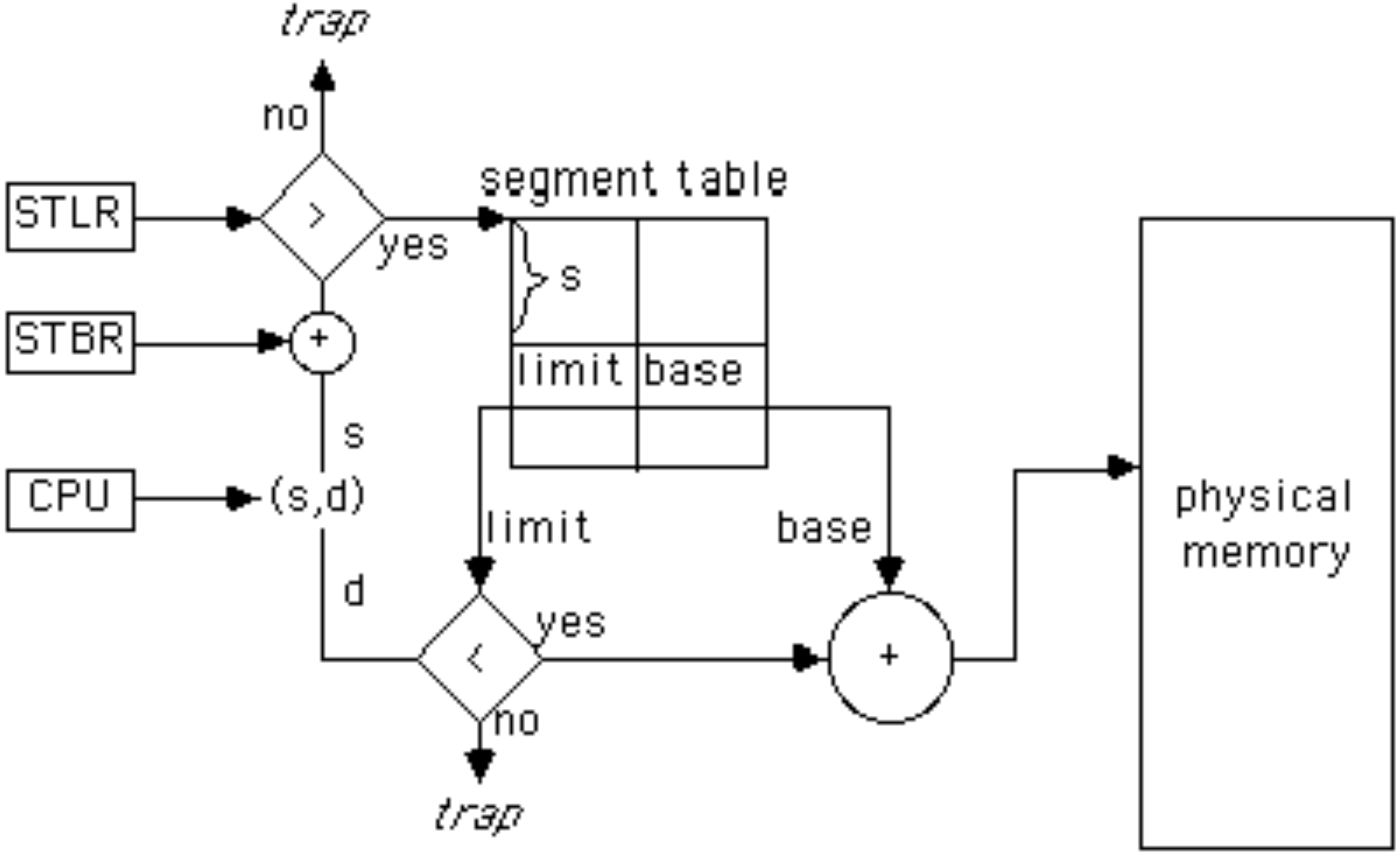
How Segmentation Is Done

- Associated with each segment table are 2 registers
  - Segment table base register (STBR) holds address of start of segment table
  - Segment table limit register (STLR) holds highest address of segment table
- Addresses are (*s*, *d*)
  - *s* is segment number
  - *d* is offset into segment

How Segmentation Is Done

- Add *s* to value stored in STBR
- If it exceeds value in STLR, trap; it’s an illegal address reference
- Use value to get segment table entry
  - This has (*base, limit*) physical addresses
- Compare limit to *d*
- If *d* exceeds *limit*, trap; it’s an illegal address reference
- Add *d* to *base*
- to get physical memory address

How Segmentation Is Done



Sharing Segments

- Keep just 1 copy of non-writeable segments in memory
- Problem: jumps in shared segments transfer to an address given as (segment number, offset)
    - Segment number is that of the shared code segment
    - Implies that if code is shared, the shared segment must have the same segment number in all processes sharing it
- Solutions
    - Only share read-only data segments without any pointers
        - So no addresses
    - In GE 645, addresses specified relative to a register containing current segment number

Fragmentation

- Pages are of fixed length, eliminating external fragmentation
- Segments are of variable length, so you do get external fragmentation
- Finding room for segments is dynamic storage allocation problem
  - Use first fit, best fit, . . . buddy algorithms
- Amount of external fragmentation depends on scheduling and segment size
- If no room in memory:
  - Wait until there is room;
  - Skip this process and put in the next one that fits; or
  - Compact memory

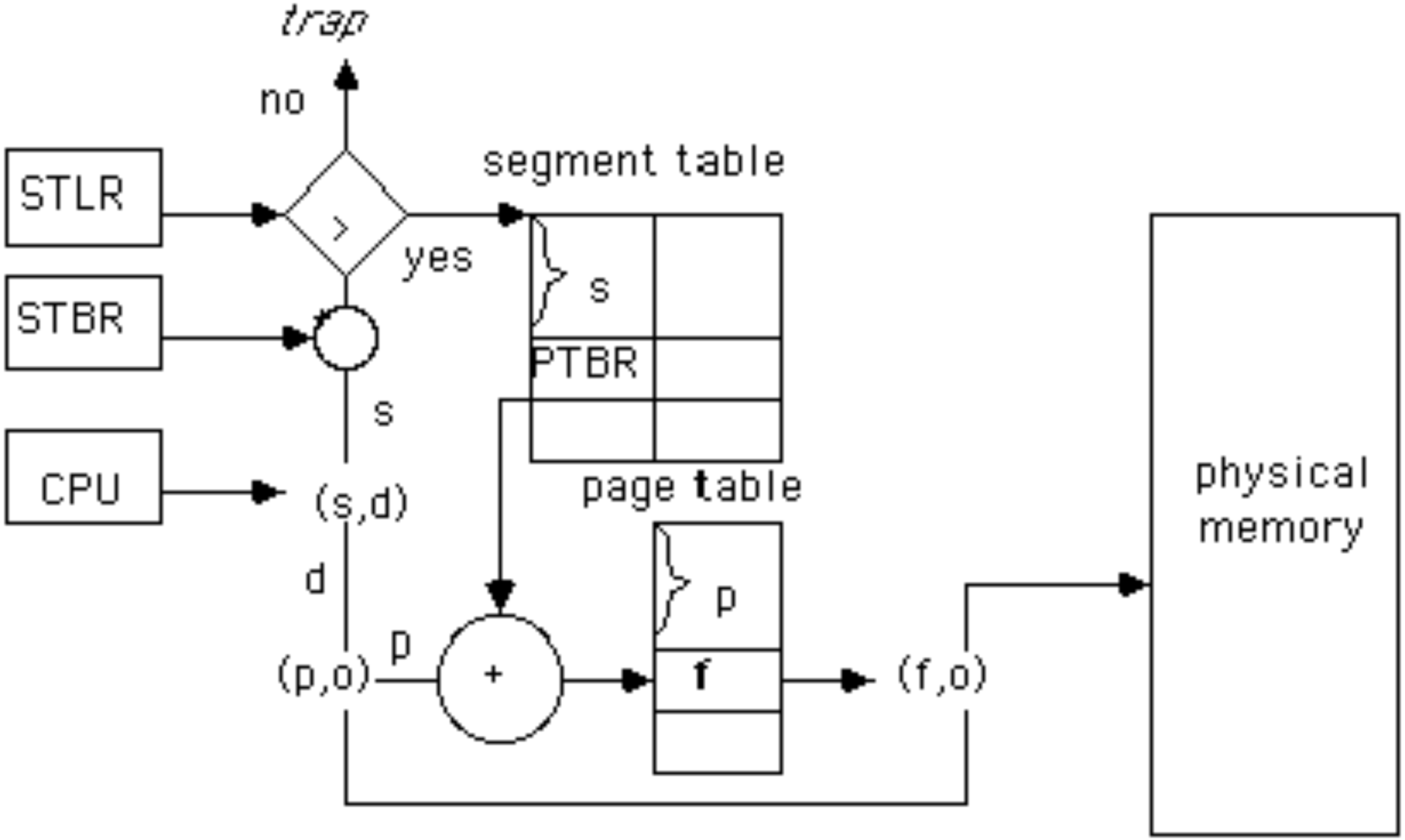
Combining These . . .

- Segmented paging: segment the page table
  - Each entry in segment table contains base, length of part of page table
- Paged segmentation: page the segment table
  - Segment table contains segment lengths, page table base (virtual) address

Segmented Paging

- Virtual address is (page number, page offset)
- In this address, page number is (segment number, segment offset)
- To get physical address from virtual address:
  1. Get segment number and add STBR
  2. Get segment table entry
  3. Compare segment offset with page table length; if offset greater, it’s an illegal reference
  4. Get page table base, add segment offset
  5. Get page table entry
  6. Use the frame number in it and page offset to get physical address

Segmented Paging



Segmented Paging

- Used when most of page table is empty
- This happens when address space is large and programs use just a small fraction of the memory space

Paged Segmentation

- Virtual address is (segment number, segment offset)
- In this address, segment offset is (page number, page offset)
- Entries in segment table are (page table base, page table length)
- To get physical address from virtual address:
    1. Get segment number and compare it to segment table length; if number greater, it's an illegal reference
    2. Add STBR to segment number
    3. Get segment table entry
    4. Add page number to page table base address
    5. Get page table entry
    6. Use the frame number in it and page offset to get physical address

Paged Segmentation

- Used when segment sizes are large and external fragmentation is a problem
- Also when fining free space takes a long time
- As with paging, last page of a segment may not be full
  - On average, half a page of internal fragmentation
- But no external fragmentation!