Paging

(*uniform size* "mini" segments)

Segment Table vs. Page Table

What if all segments have the **SAME size**?

Limit	Segment Addr
2000	6000
2000	0
2000	12000
2000	8000

Segment Table

Segment Addr
6000
0
12000
8000

Page Number
3
0
6
4

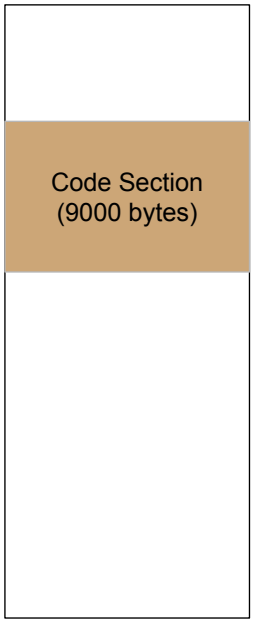
Page Table

STBR: Segment Table Base Register

PTBR: Page Table Base Register

Page address is ALWAYS multiple of page size

Example: Segmentation vs. Paging

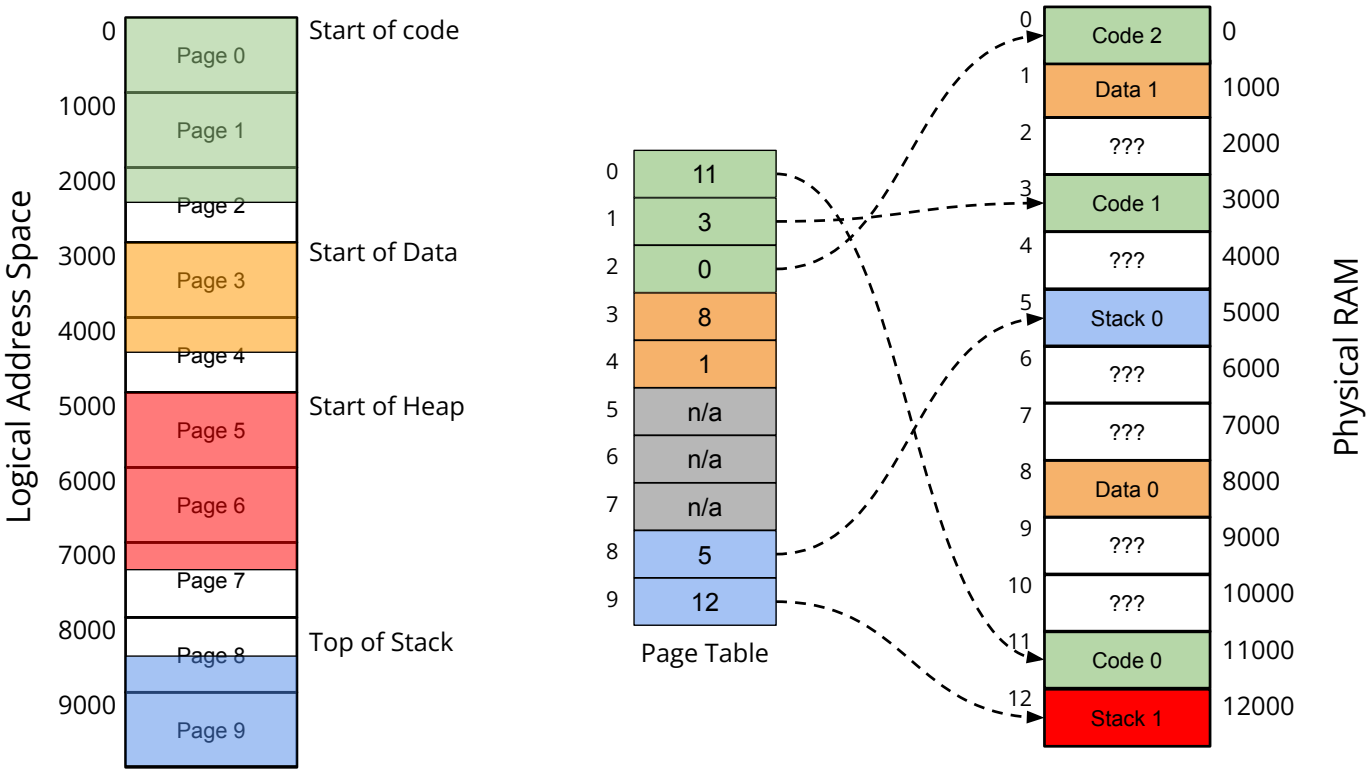


- **Segmentation**: Place the Code section if ONE segment of size 9000 bytes
- **Paging (2000 byte/page)**: place the Code section in FIVE separate pages
  - These five pages DON'T have to be next to each other
  - Wasted 1000 bytes in the last page of code section (INTERNAL fragmentation)

Paging Mechanism Simplifies MMU Hardware

(when page size is carefully chosen)

Logical Pages vs. Physical Frames



Address Arithmetic (div and mod)

Page Size = 2000 bytes

Linear Address	Page Number	Offset (within a page)	(Page #|Offset) Pair
617	0 (= 617 div 2000)	617 (= 617 % 2000)	(0, 617)
1500	0 (= 1500 div 2000)	1500 (= 1500 % 2000)	(0, 1500)
2571	1 (= 2571 div 2000)	571 (= 2571 % 2000)	(1, 571)
9754	4 (= 9754 div 2000)	1754 (= 9754 % 2000)	(4, 1754)

Page Size = 1000 bytes

Linear Address	Page Number	Offset (within a page)	(Page #|Offset) Pair
617	0 (= 617 div 1000)	617 (= 617 % 1000)	(0, 617)
1500	1 (= 1500 div 1000)	500 (= 1500 % 1000)	(1, 500)
2571	2 (= 2571 div 1000)	571 (= 2571 % 1000)	(2, 571)
9754	9 (= 9754 div 1000)	754 (= 9754 % 1000)	(9, 754)

When page size is 10^n, linear addresses have a "natural split".
The *last n digits* are the offset within a page

When page size is 10^n, memory addresses exhibit a "*natural split*".
The last **n** digits are the offset within a page

When page size is 2^n,
the last **n** bits are the offset within a page

Address Translation Examples: 7-digit addr

Logical addr 0456789 (base 10)

Page Size	Max Offset	Page Number	Offset
10^2	99	04567	**89**
10^3	999	0456	**789**
10^4	9999	045	**6789**

Logical addr 0345abc (base 16)

Page Size	Max Offset	Page Number	Offset
16^2	FF	0345a	**bc**
16^3	FFF	0345	**abc**
16^4	FFFF	034	**5abc**

Logical addr 0234567 (base 8)

Page Size	Max Offset	Page Number	Offset
8^2	77	02345	**67**
8^3	777	0234	**567**
8^4	7777	023	**4567**

General rule:
When page size is B^d the last d digits of address are the offset within a page/frame.

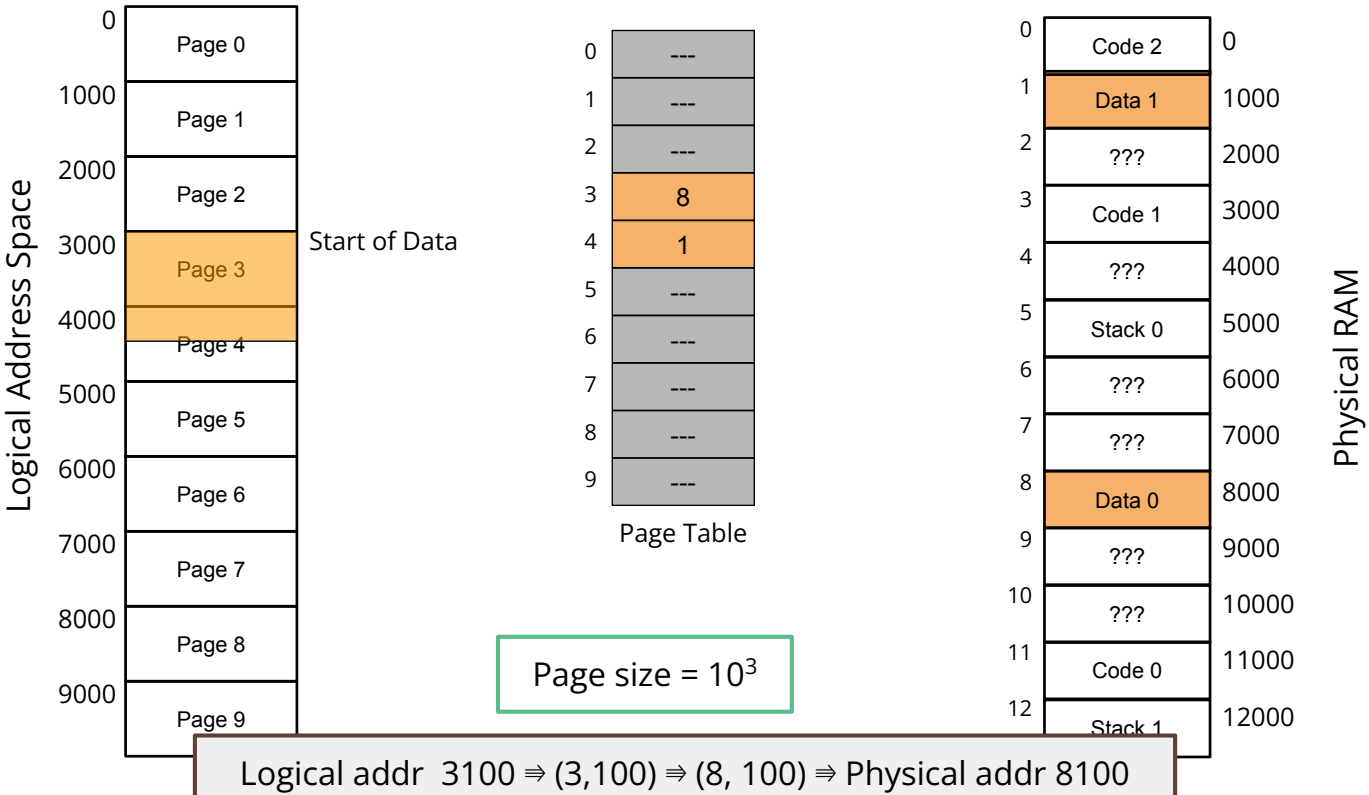
*Same principle applies when memory address is expressed in **binary** numbers.*

When page size is 2^d then the last d bits are the offset within a page/frame

Address "*natural split*" allows the compiler to simply use **linear** address

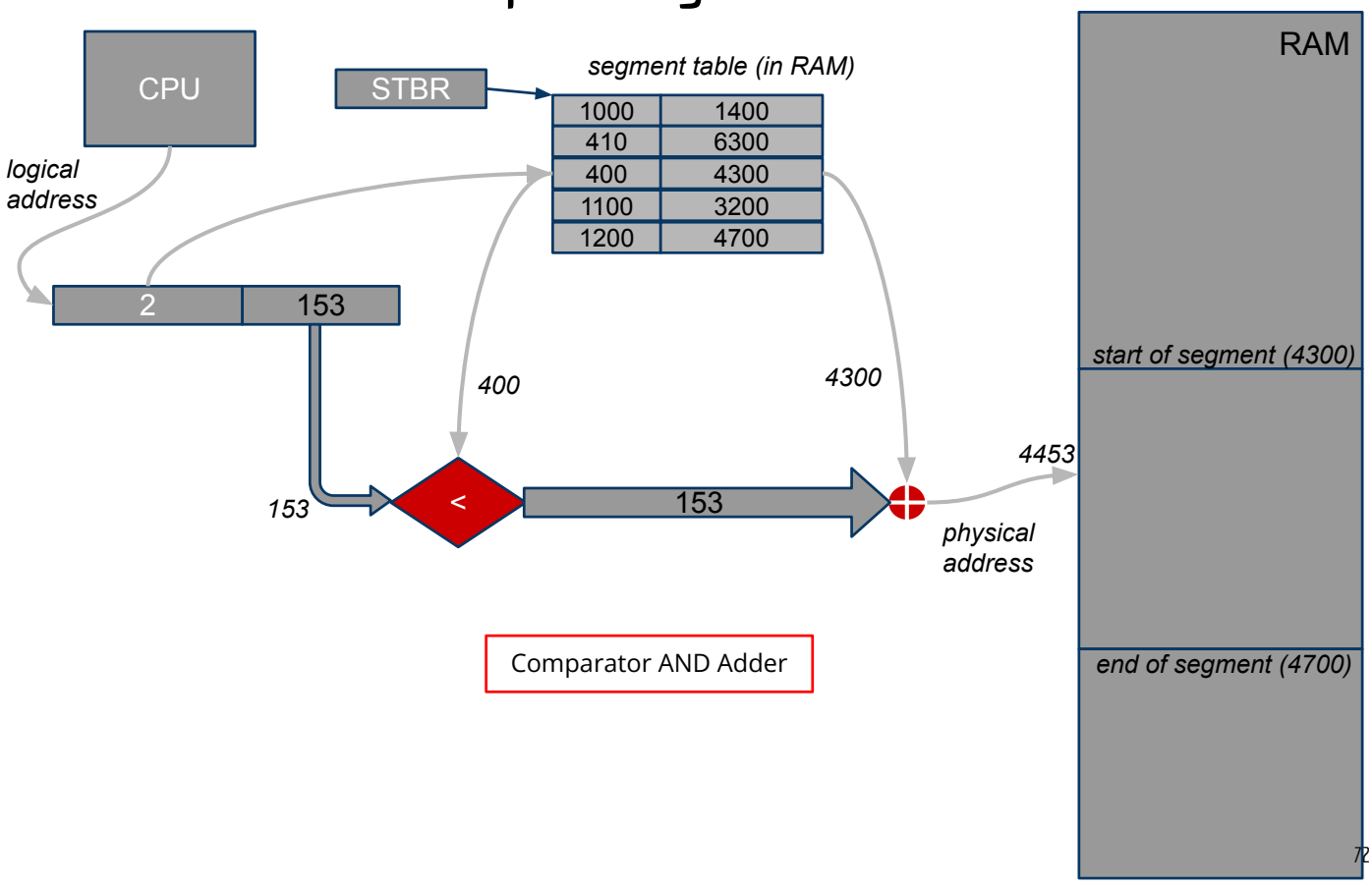
No need to use **(page, offset)** pair

Address Translation: Logical to Physical

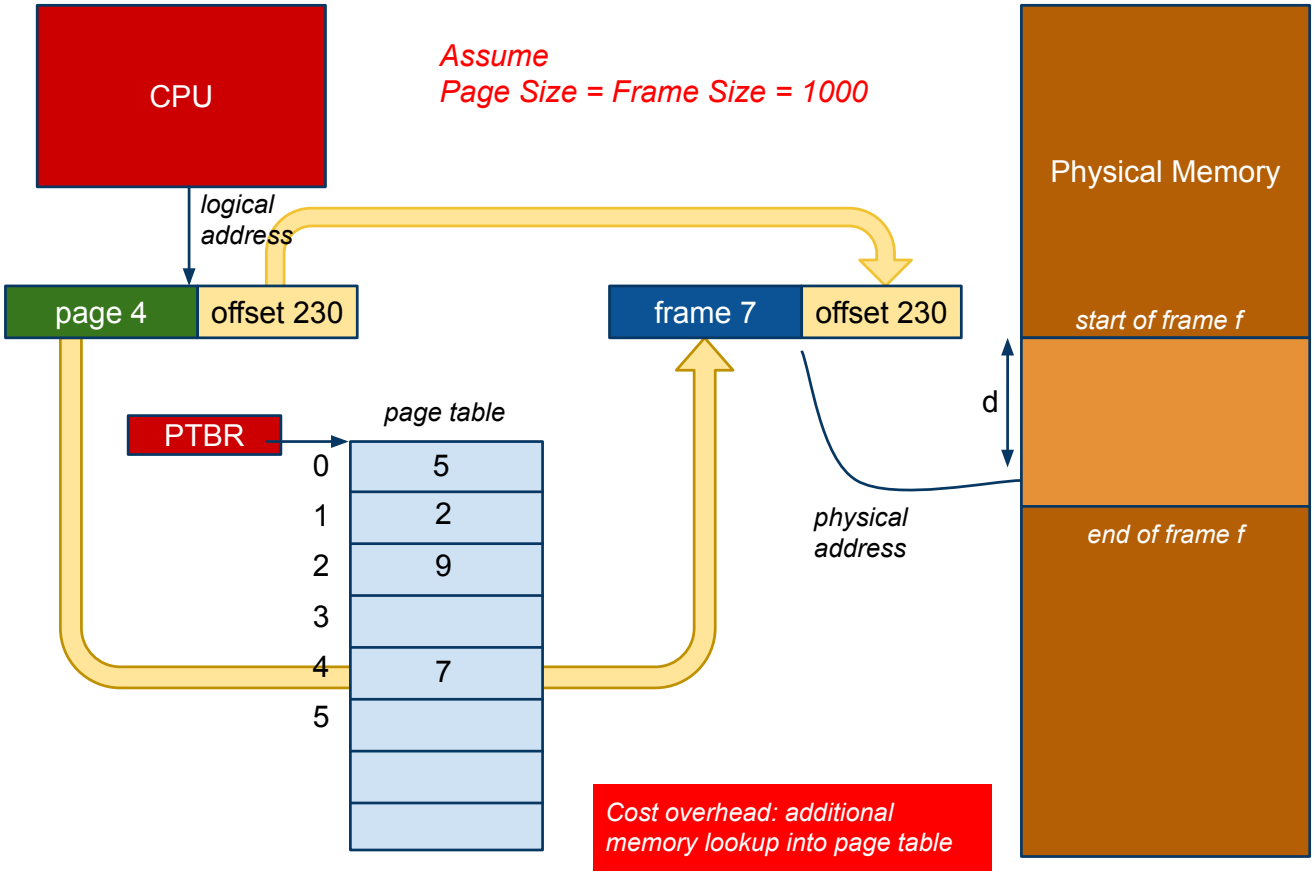


MMU Hardware for Paging

MMU Hardware for Segmentation



MMU for Paging (No Comparator + No Adder)



Hexadecimal Numbers

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Decimal	Binary	Hexadecimal
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Kilo, Mega, Giga, ...

Binary	Exact Value	Unit	Approximate Value	Approx. Decimal
2^{10}	1,024	Kilo	1,000	10^3
2^{20}	1024x1024 = 2^{10}x2^{10}	Mega	1,000,000	10^6
2^{30}	1024x1024x1024 = 2^{10}x2^{10}x2^{10}	Giga	1,000,000,000	10^9
2^{40}	...	Tera	...	10^{12}
2^{50}	...	Peta	...	10^{15}
2^{60}	...	Exa	...	10^{18}
2^{70}	...	Zetta	...	10^{21}
2^{80}	...	Yotta	...	10^{24}

2K is NOT two thousand

Interpreting Numbers

Decimal (base 10)

$400 = 4 \times 100 = 4 \times 10^2$

$7\_000 = 7 \times 1000 = 7 \times 10^3$

$\mathbf{37\_000 = 37 \times 1000 = 37 \times 10^3 = (30 + 7) \times 10^3}$

$9\_000\_000 = 9 \times 10^6$

Binary (base 2)

$0100 = 1 \times 2^2$

$\mathbf{1100 = 3 \times 2^2}$

$1000 = 1 \times 2^3$

Hex (base 16)

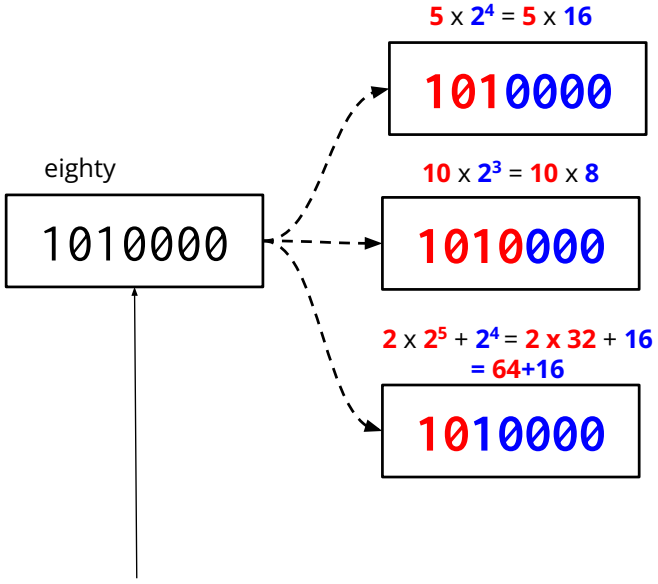
$100 = 1 \times 16^2$

$\mathbf{2100 = 33 \times 16^2}$

$C0\_0000 = 12 \times 16^5$

$7000 = 7 \times 16^3$

Memory Addresses (with "*Natural Split*")



(**5**, **0**) 8 pages
80 = 5 x 16 + 0 pg size:16 bytes

(**10**, **0**) 16 pages
80 = 10 x 8 + 0 pg size: 8 bytes

(**2**, **16**) 4 pages
80 = 2 x32 + 16 pg size:32 bytes

Address generated by compiler

Addresses interpreted by memory hardware in various formats

Hexadecimal/Binary Arithmetic Operations

- Kilo, Mega, Giga, Tera, Peta, Exa, Zeta, Iota
- Rules of multiplication: $2^{10} \times 2^{8} = 2^{18}$ $2^{34} = 2^{4} \times 2^{10} \times 2^{10} \times 2^{10}$

Hex	Binary	Value
0x10	0001 0000	$2^4 = 16$
0x100	0001 0000 0000	$2^8 = 256$
0x400	0100 0000 0000	$2^{10} = 1024 = 1K$
0x3000	0011 0000 0000 0000	$3x2^{12} = 3x2^2x2^{10} = 3x4K = 12K$
0x60000	0110 0000 0000 0000 0000	$3x2^{17} = 3x2^7x\ 2^{10} = 3x128K$
0x10 0000	0001 0000 0000 0000 0000 0000	$2^{20} = 1M$
0x500 0000	0101 0000 0000 0000 0000 0000 0000	$5 \times 2^{24} = 5 \times 16M = 80M$
0x4000 0000	0100 0000 0000 0000 0000 0000 0000 0000	$2^{30} = 1G$

Exercises

- Express the following values into hexadecimal
  - 4K
  - 5K
  - 48K
  - 9M

Hexadecimal Address Arithmetic

1 Hex digit => 4 binary digits

20-bit Hex address DC93A

Page: D Offset: C93A

- Using 4-bit page number, 16-bit offset

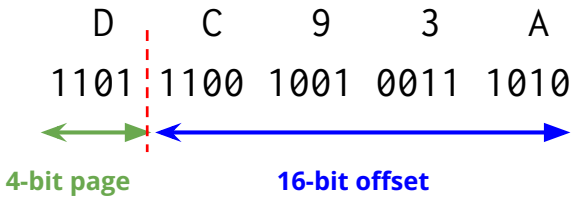
1. The address refers to which page at what offset?
   a. Page D Offset C93A
2. What is the maximum size (in bytes) of a page?
   a. 2^{16} bytes, **2^{15} bytes,**
3. How many pages available in the system?
   a. 2^4 pages, **2^5 pages**

Repeat the question using 5-bit page number, *hence* 15-bit offset!

Page size $B^d \Rightarrow$ the last d digits of address are the offset.

Address must be expressed in base B

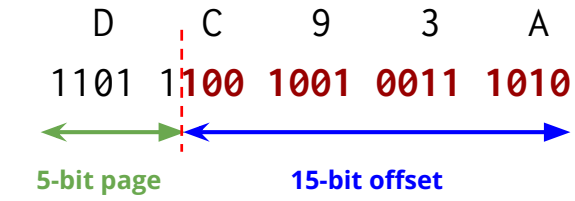
20-bit Hex address DC93A (4-bit page number, 16-bit offset)



- Page: 0xD, offset: 0xC93A
- Maximum offset is 0xFFFF ($2^{16} - 1$) = (2^6 x 2^{10}) − 1= (64 x 1024) − 1 = 64K − 1
- Total number of pages 2^4 = 16

- Total addressable memory is 1M = 2^{20} (which for this case = 2^4 x 2^{16})
  - The system provides 16 pages (2^4), each page is 64K bytes ($2^{16=}$ 2^{10}x 2^6)

20-bit Hex address DC93A (5-bit page number, 15-bit offset)



- Page: `1 1011 = 0x1B`, offset: **100 1001 0011 1010** = 0x493A
- Maximum offset is 0x7FFF ($2^{15} - 1$) = (2^5 x 2^{10}) − 1= (32 x 1024) − 1 = 32K − 1
- Total number of pages 2^5 = 32

- Total addressable memory is 1M = 2^{20} (which for this case = 2^5 x 2^{15})
  - The system provides 32 pages (2^5), each page is 32K bytes (2^{15})

Demand Paging

Paging

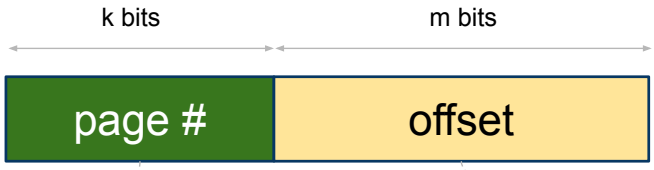
- Partition RAM into equal fixed-sized blocks (**frames**) and split processes into blocks of the same size (**pages**)
- Logical pages must be mapped to physical frames
  - Use per-process page tables (**one PT per process**)
  - The size of PT depends on the size of the process
- The base address of a page table is kept in Page Table Base Register (PTBR)
  - PTBR must be saved/restored during context-switch
- Pages (of a process) can be loaded to RAM ***on-demand***
  - We don't have to load ALL the pages at once!
  - Allow processes to have more pages than available RAM frames
  - **Individual pages** can be swapped in/out (*don't have to swap the entire process*)

How do we read a (printed) book?

One page at a time?
One word at a time?

Only the current page(s) must be visible, all other pages don't have to be accessible

Virtual/Logical Address



Array index into Page Table

Range: 0 to $2^k - 1$

Total number of pages 2^k

Displacement/distance of data/instruction from beginning of page.

Range: 0 to $2^m - 1$

Total number of bytes per page 2^m

Total available **logical** space = $2^k\ 2^m = 2^{k+m}$