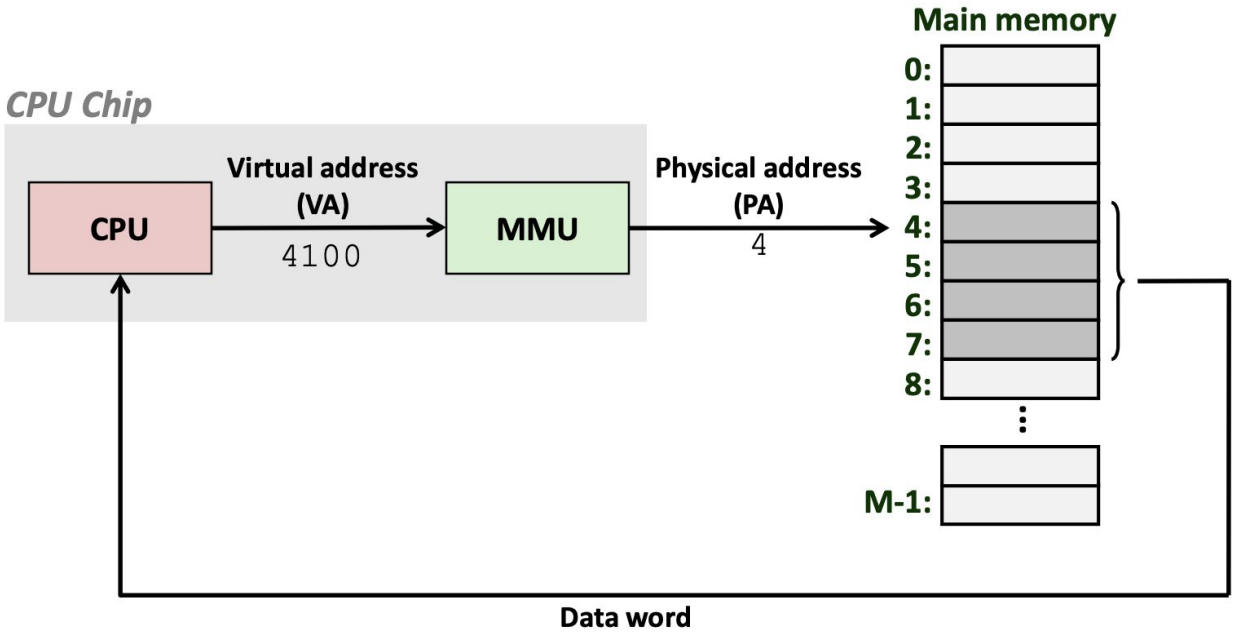
CS 33: Introduction to Computer Organization

TA: Aalisha Dalal
LA: Jonathan Myong
Office Hours: Friday, 9:30-11:30AM

Outline

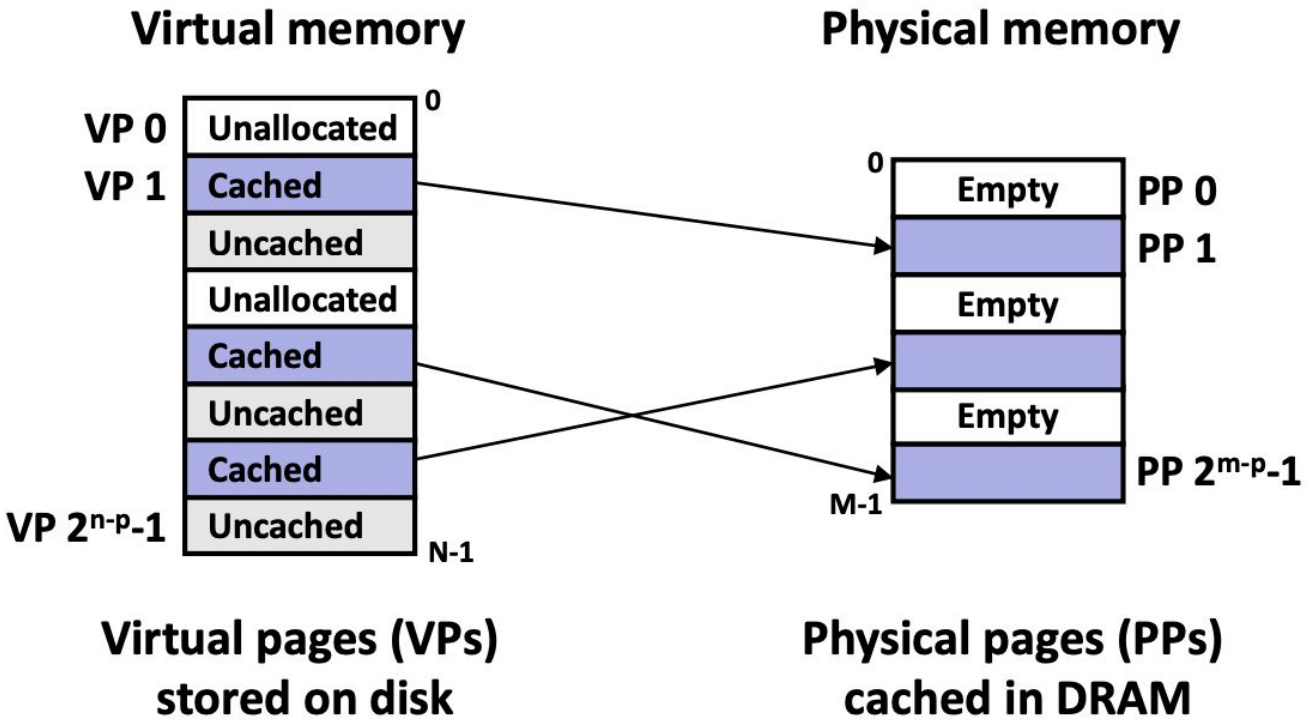
- **Virtual memory [Recap]**
- **MIPS**
- **Revision - Finals (Part I)**
- **Worksheet Problems**

Virtual Memory

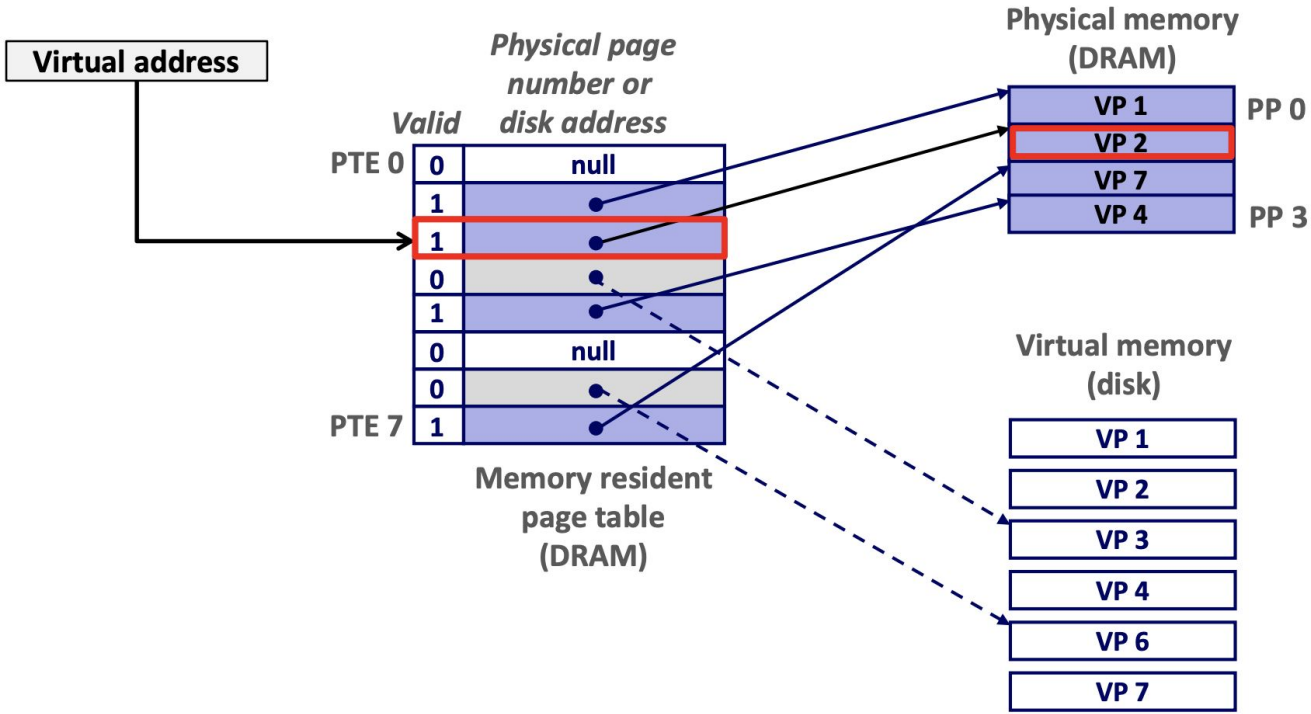


Virtual Memory Addressing

Virtual Memory - Caching



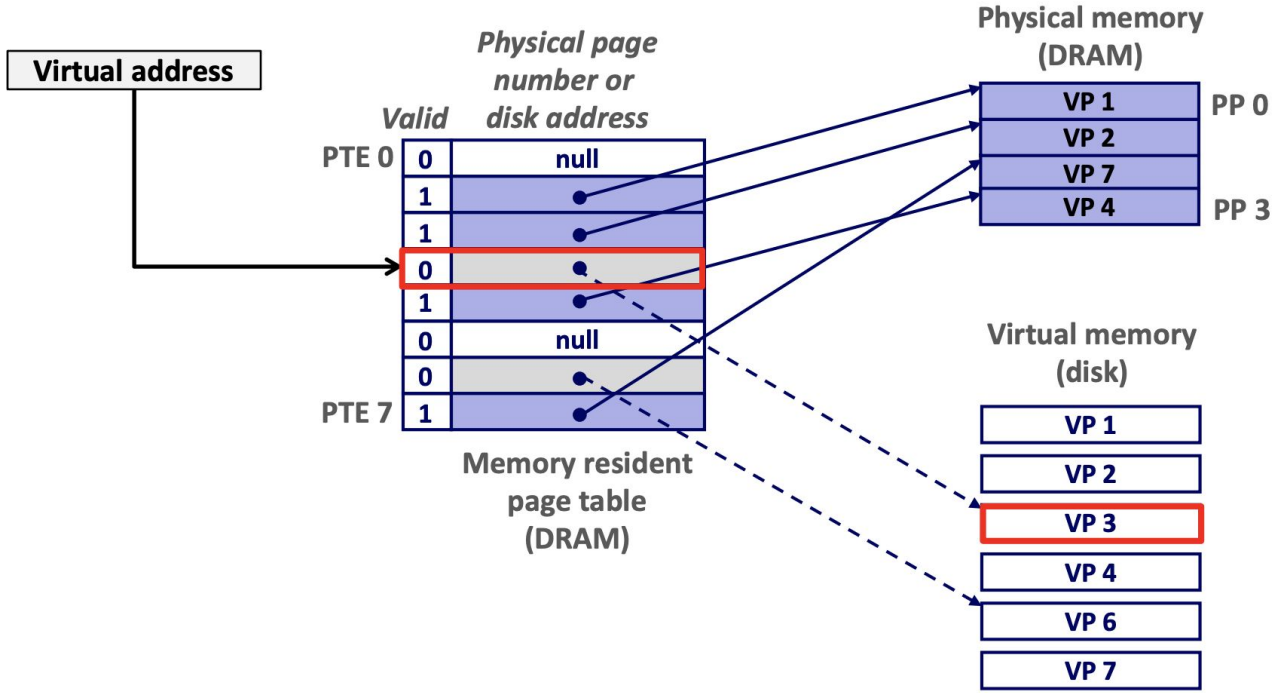
Virtual Memory - Page Hit



We found the address in the main memory!

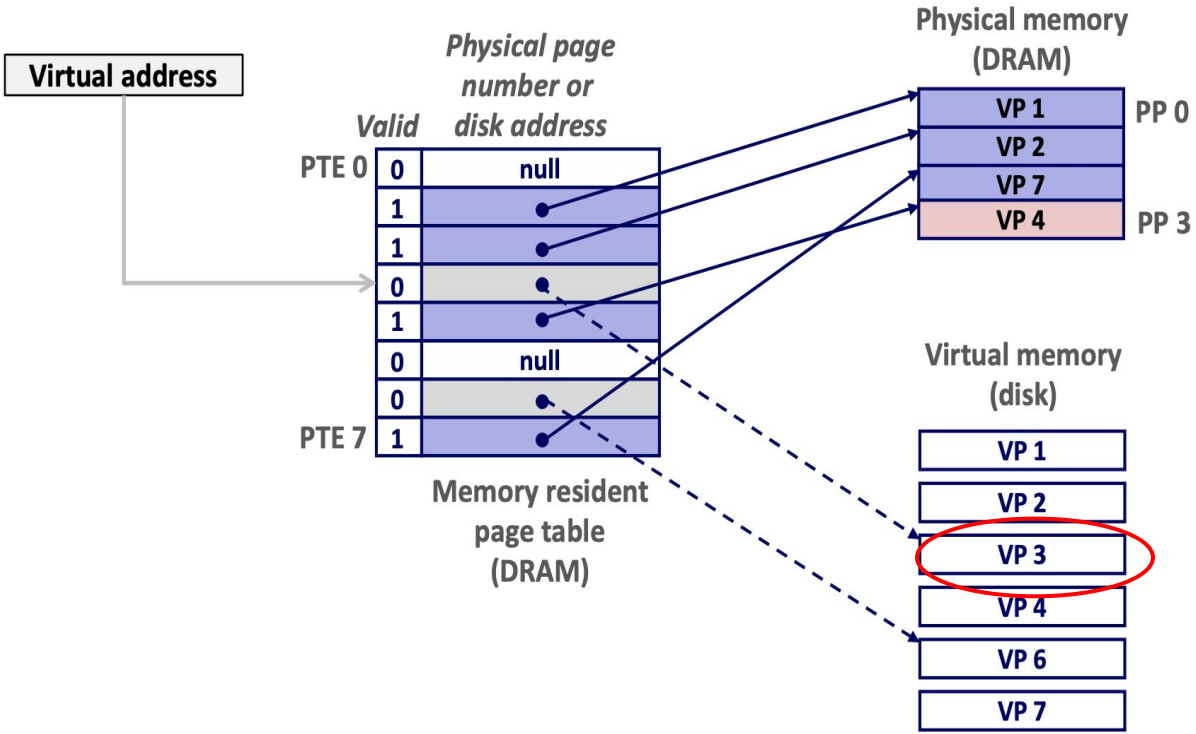
What if we don't find it?

Virtual Memory - Page Fault



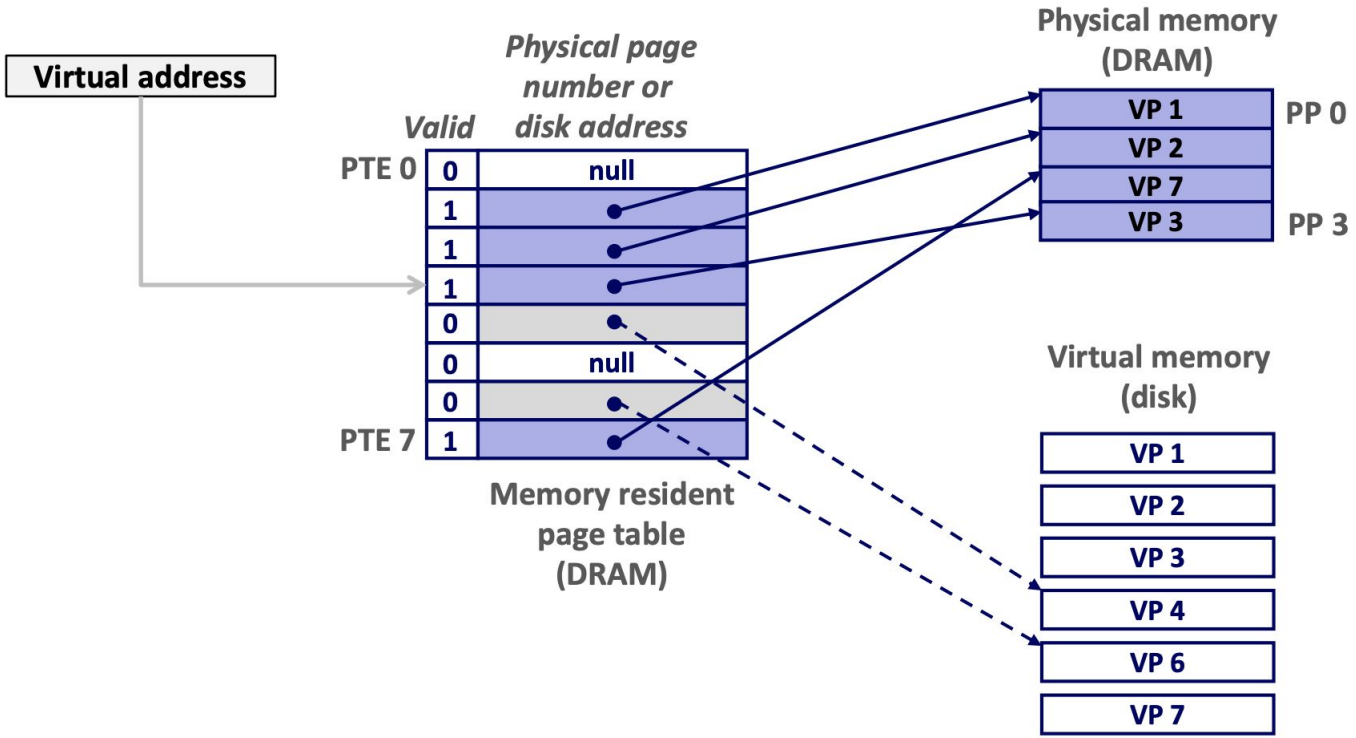
What should we do in this scenario?

Virtual Memory - Handling Page Fault

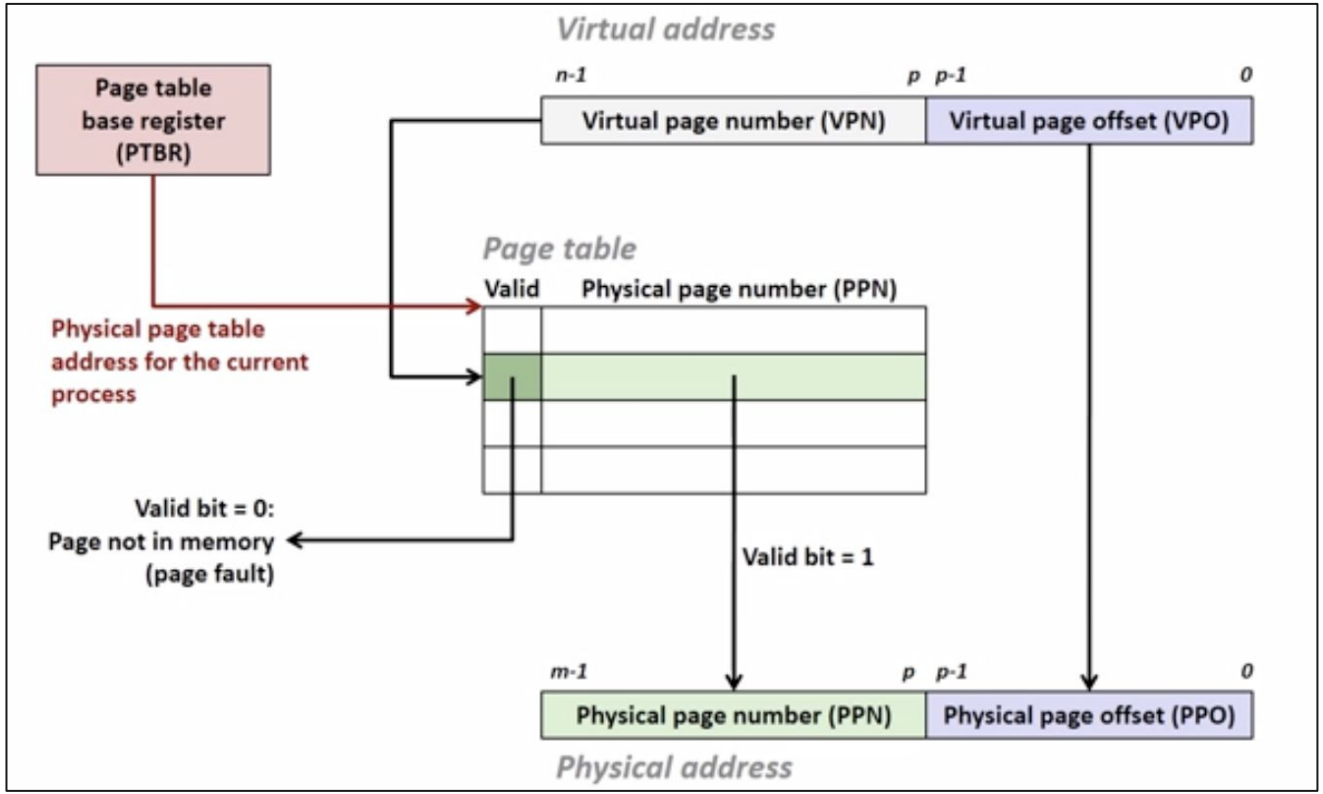


Virtual memory uses the idea of **Demand Paging**

Virtual Memory - Handling Page Fault



Address Translation - Page Table



Check your understanding!

A computer system has a 36-bit virtual address space with a page size of 8K, and 4 bytes per page table entry.

1. How many pages are in the virtual address space?
2. What is the maximum size of addressable physical memory in this system?

Check your understanding! - Solution

A computer system has a 36-bit virtual address space with a page size of 8K, and 4 bytes per page table entry.

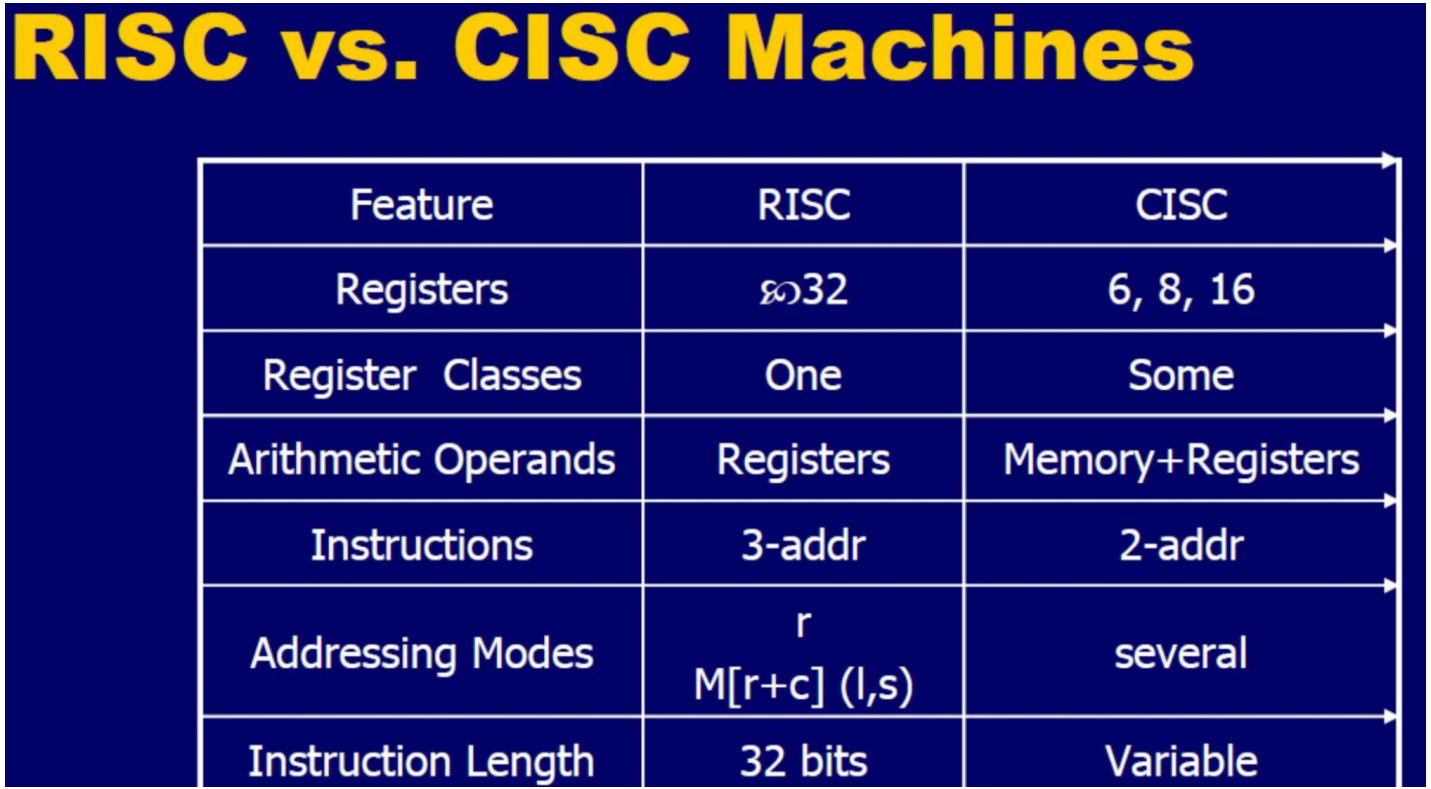
1. How many pages are in the virtual address space?

   => 2^36 bytes / (2^3*2^10 page-size) = 2^36/ 2^13 = 2^23 pages

2. What is the maximum size of addressable physical memory in this system?

   => 2^(4*8) = 2^32 pages. Page-size: 2^13 bytes. Size of the physical memory is (2^32*2^13) = 2^45 bytes

RISC vs. CISC


RISC vs. CISC Machines
Feature
RISC
CISC
Registers
ℵ32
6, 8, 16
Register Classes
One
Some
Arithmetic Operands
Registers
Memory+Registers
Instructions
3-addr
2-addr
Addressing Modes
r
M[r+c] (l,s)
several
Instruction Length
32 bits
Variable

Example - CISC

Complex instruction such as ‘mov’ in which operands can be both memory address/ registers

```
//Copy %eax register val to %ebx
mov %eax, %ebx

//Copy *(%esp+4) to %ebx
mov 4(%esp), %ebx

//Copy %ebx register val to *(%esp+4)
mov %ebx, 4(%esp)
```

MIPS - RISC Load/Store Architecture

```
a = b + c;
d = a + b;
```

```
Load b into register Rx
Load c into register Ry
Rz <- Rx + Ry
Store Rz into a
Rz <- Rz + Rx
Store Rz into d
```

Code Example

Hello World

```
        .text           # text segment
        .global __start
__start:                # execution starts here
        la $a0,str      # put string address into a0
        li $v0,4        #
        syscall         # print
        li v0, 10       #
        syscall         # au revoir...
        .data           # data segment
str:    .asciiz  "hello world\n"
```

Revision

Loop Unrolling

Why is it beneficial?

- Allows faster execution as lesser ‘jump’ and ‘branch’ conditions have to be evaluated

- Increase program efficiency while reducing loop overhead

Cons: Code readability reduces

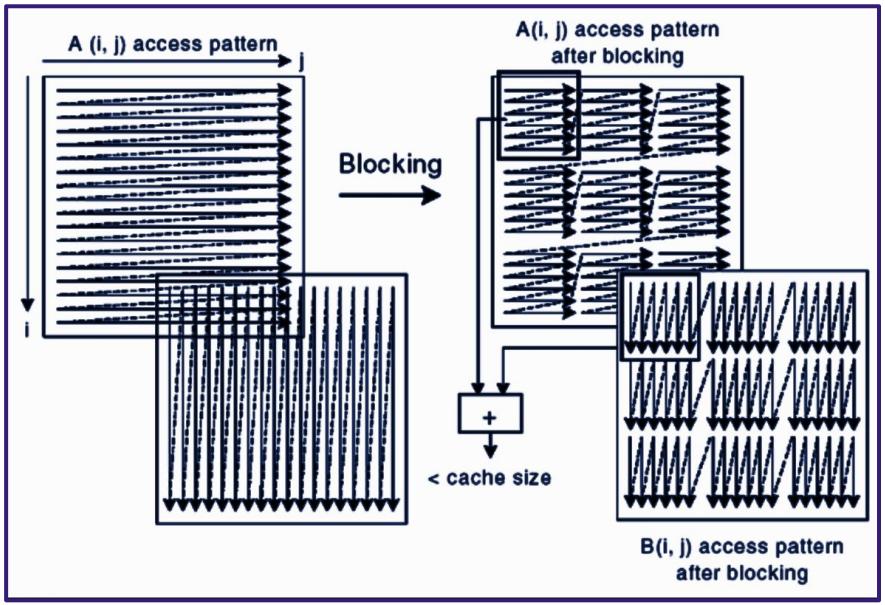
Loop Unrolling

Normal loop	After loop unrolling

`int x;`
`for (x = 0; x < 100; x++)`
`{`
`  delete(x);`
`}`
	`int x;`
	`for (x = 0; x < 100; x += 5 )`
	`{`
	`  delete(x);`
	`  delete(x + 1);`
	`  delete(x + 2);`
	`  delete(x + 3);`
	`  delete(x + 4);`
	`}`

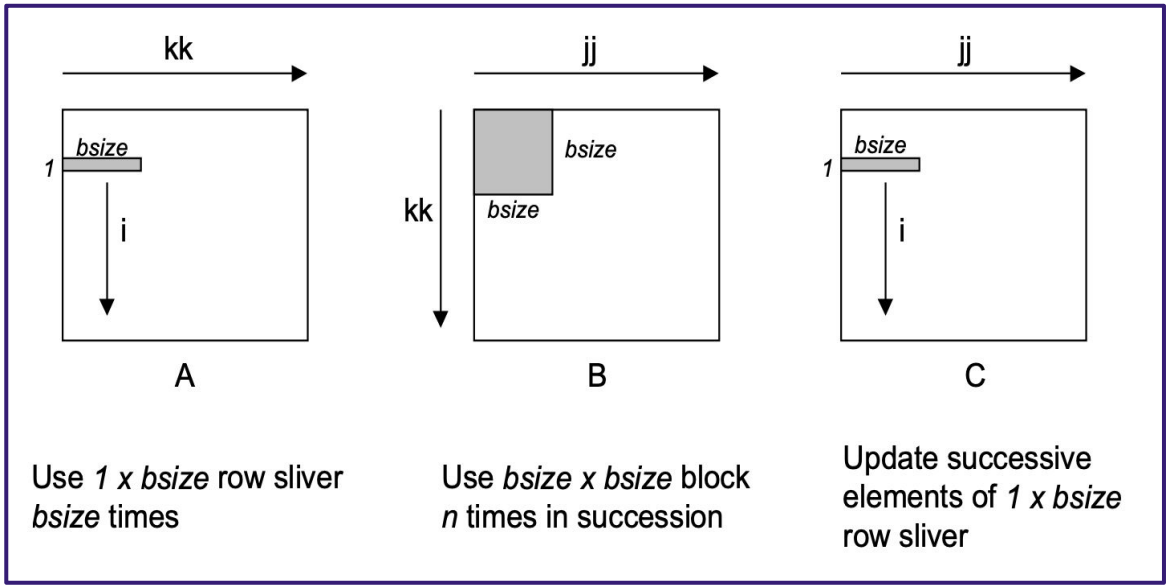
Loop Blocking/Tiling

"*This technique uses the idea of temporal locality in cache*"



Example for Loop blocking

Loop Blocking/Tiling



Code Snippet Example - Matrix Multiplication

```
1  void bijk(array A, array B, array C, int n, int bsize)
2  {
3      int i, j, k, kk, jj;
4      double sum;
5      int en = bsize * (n/bsize); /* Amount that fits evenly into blocks */
6
7      for (i = 0; i < n; i++)
8          for (j = 0; j < n; j++)
9              C[i][j] = 0.0;
10
11     for (kk = 0; kk < en; kk += bsize) {
12         for (jj = 0; jj < en; jj += bsize) {
13             for (i = 0; i < n; i++) {
14                 for (j = jj; j < jj + bsize; j++) {
15                     sum = C[i][j];
16                     for (k = kk; k < kk + bsize; k++) {
17                         sum += A[i][k]*B[k][j];
18                     }
19                     C[i][j] = sum;
20                 }
21             }
22         }
23     }
24 }
```

code/mem/matmult/bmm.c

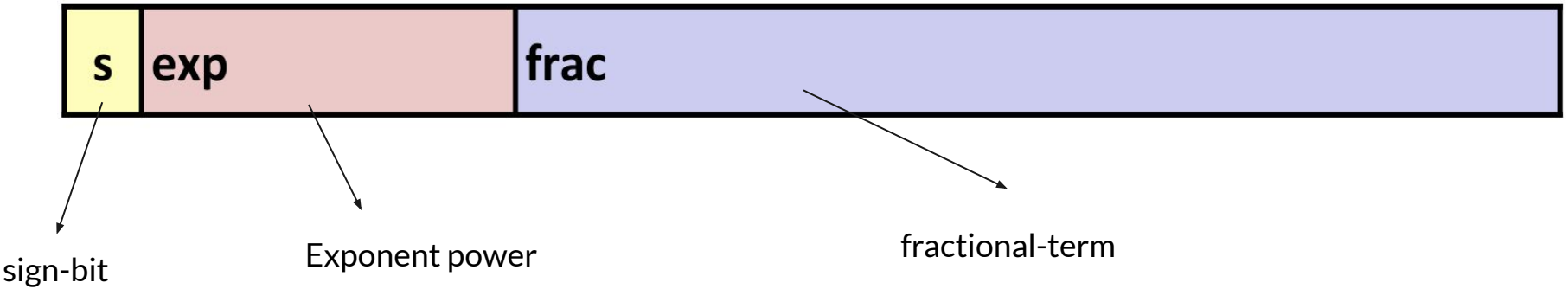
Floating Point Conversion

What is 1011.101_2?

Revision - Floating Point

What is 1011.101_2? ⟹ $11\ 5/8$

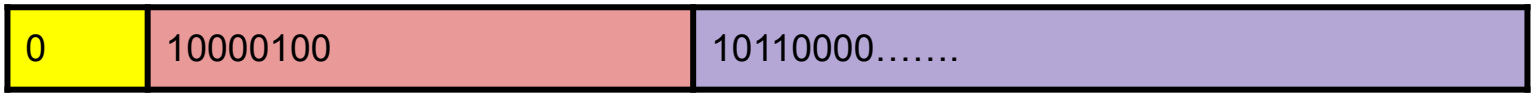
Floating point representation



How would we represent 27?

$$27_{10} = 11011_2$$

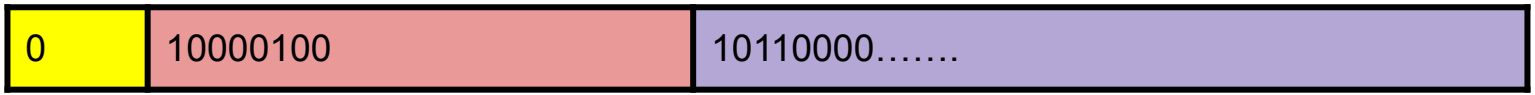
$$= 1.1011 * 2^4$$



How would we represent 27?

$$27_{10} = 11011_2$$

$$= 1.1011 * 2^4$$



Another example!

float: 0xC0A00000

Another example!

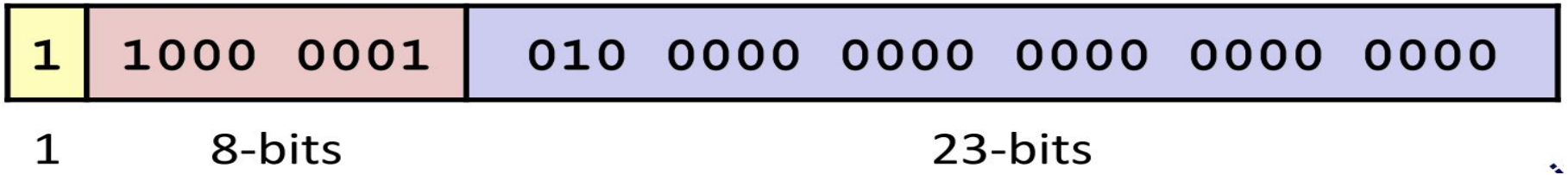
float: 0xC0A00000

binary: 1100 0000 1010 0000 0000 0000 0000 0000

Another example!

float: 0xC0A00000

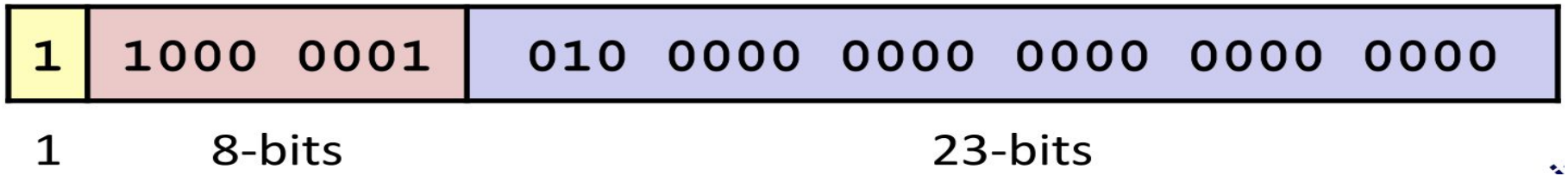
binary: 1100 0000 1010 0000 0000 0000 0000 0000



Another Example!

float: 0xC0A00000

binary: 1100 0000 1010 0000 0000 0000 0000 0000

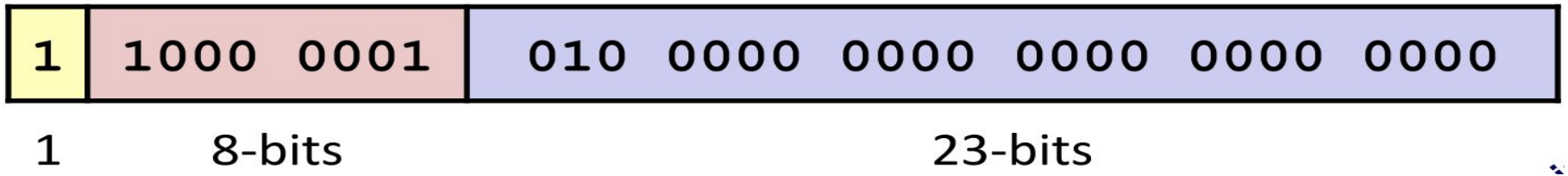


E = 129 -> Exp = 129 – 127 = 2 (decimal)

Another Example!

float: 0xC0A00000

binary: 1100 0000 1010 0000 0000 0000 0000 0000



M = 1.010 0000 0000 0000 0000 0000

E = 129 -> Exp = 129 – 127 = 2 (decimal)

Worksheet

https://tinyurl.com/cs33-2nd-to-last