CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Segmentation

With contiguous memory allocation, a program's view of memory was as one block from `[0, max]`

This is true regardless of whether we used fixed partitions, variable partitions, or variable partitions on top of fixed-size blocks

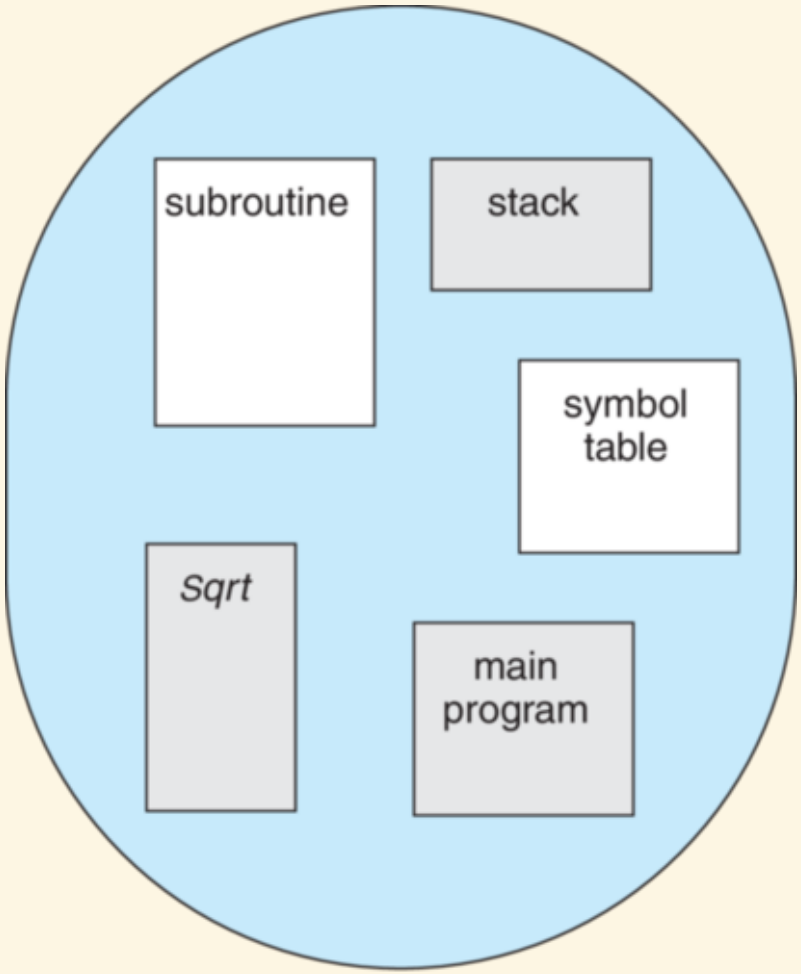
However, this is not how we intuitively think of memory when programming

Memory is usually thought of as groups, or **segments**, of related storage

Variables might be on "the heap" or "the stack"

Line of code might be "fourth line in function `Sqrt()`"

We do not care where in memory the stack, heap, or function `Sqrt()` is stored


subroutine
stack
symbol table
Sqrt
main program
logical address

Segmentation is a memory management scheme designed around this model

Logical address space of a process is split into segments

Segment is simply section of memory with given

- name (actually implemented as a number)
- length

Memory addresses are now accessed using (`segment number, offset into segment`) rather than absolute address

C compilers create object files according to this idea

```
$ readelf -S a.out
...
Section Headers:
  [Nr] Name              Type             Address           Offse
       Size              EntSize          Flags  Link  Info  Alig
  [ 0]                   NULL             0000000000000000  00000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         00000000000002a8  00000
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.bu[...] NOTE             00000000000002c4  00000
       0000000000000024  0000000000000000   A       0     0     4
  [ 3] .note.ABI-tag     NOTE             00000000000002e8  00000
       0000000000000020  0000000000000000   A       0     0     4
...
```

```
$ readelf -l a.out
...
Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.gnu.build-id .note.ABI-tag .gnu.hash ...
   03     .init .plt .text .fini
   04     .rodata .eh_frame_hdr .eh_frame
   05     .init_array .fini_array .dynamic .got .got.plt .data...
   06     .dynamic
   07     .note.gnu.build-id .note.ABI-tag
   08     .eh_frame_hdr
    ...
```

We now have a two-dimensional model of memory, but it is still an array of bytes underneath

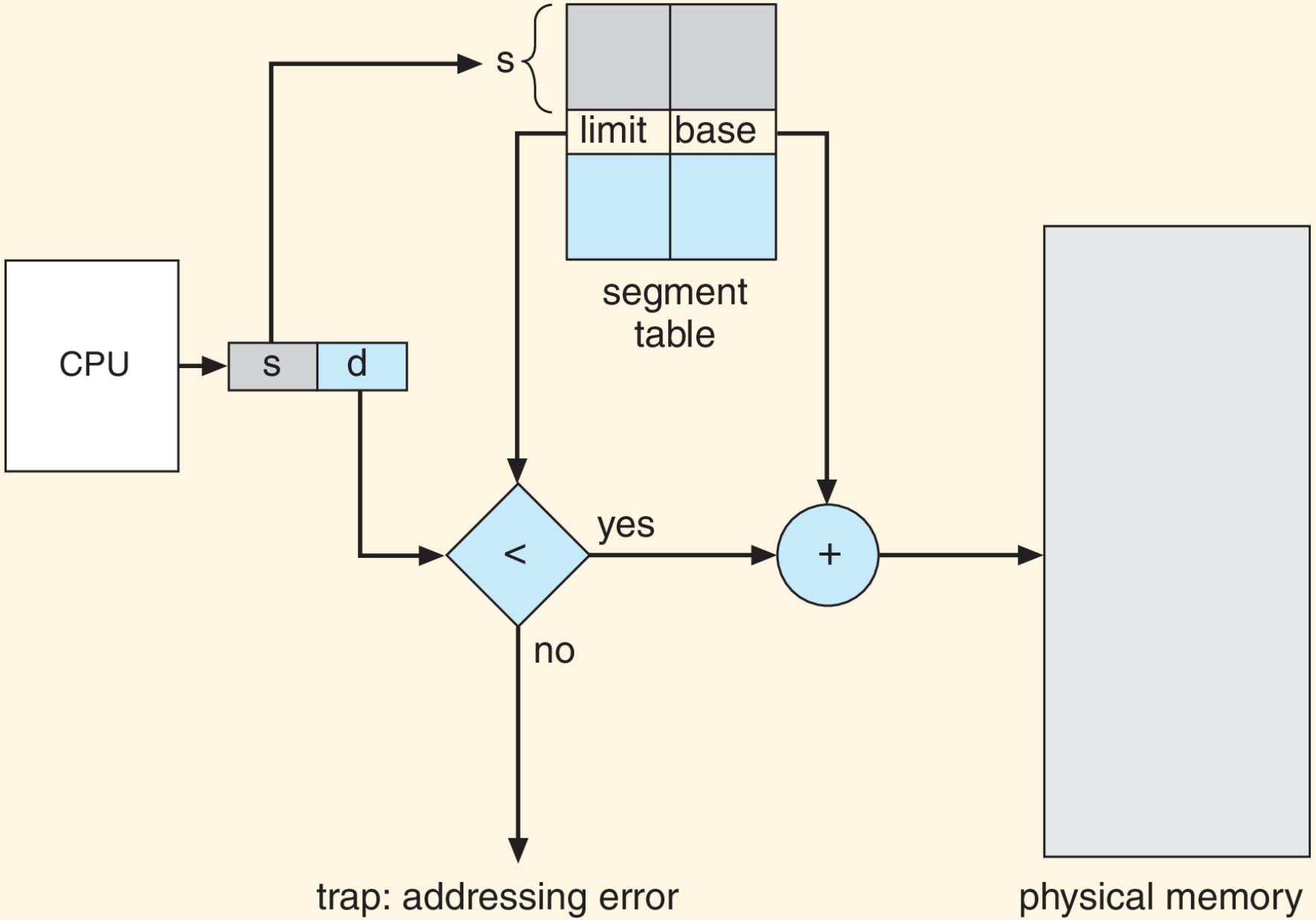
What is responible for translating this logical 2-d address into physical address?

MMU

Also need to store **segment table**

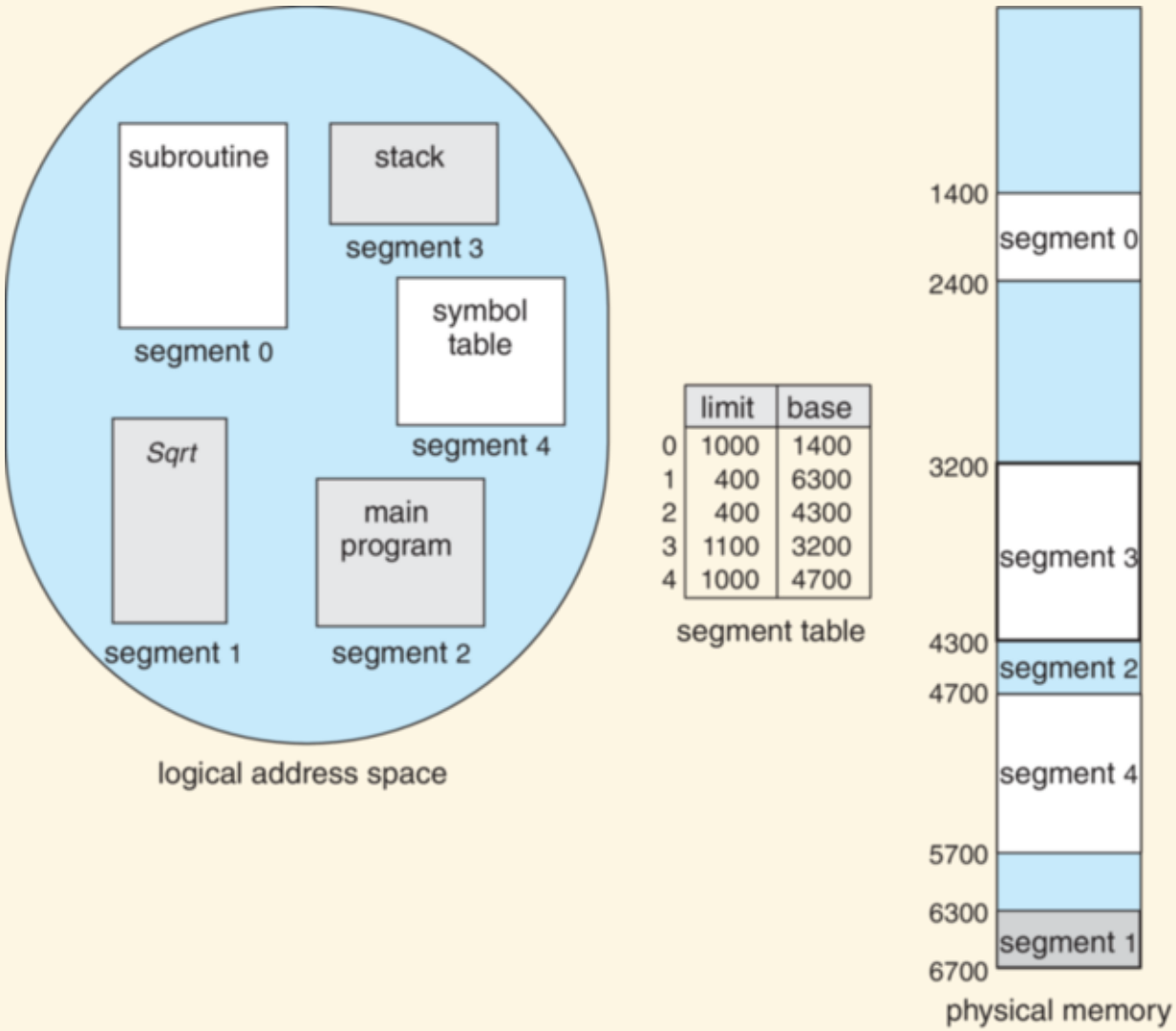
Consists of `[segment base, segment limit]` register pairs for each segment

As usual, **segment base** is first physical address in segment and **segment limit** is size of segment


s
limit
base
segment table
CPU
s
d
<
yes
no
+
trap: addressing error
physical memory

With this system, *memory for a process need no longer be contiguous*

Here's how the physical memory might be laid out for the example from earlier:


subroutine
stack
segment 3
symbol table
segment 0
segment 4
Sqrt
main program
segment 1
segment 2
logical address space
limit
base
0 1000 1400
1 400 6300
2 400 4300
3 1100 3200
4 1000 4700
segment table
1400
segment 0
2400
3200
segment 3
4300
segment 2
4700
segment 4
5700
6300
segment 1
6700
physical memory

Segmentation

- maps memory in a way that may be more intuitive to programmers
- does not require contiguous chunk of memory for a process