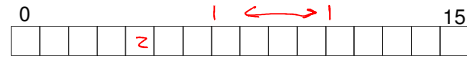


## Assignment 3 tips

## Pointer Recap

Memory addresses are just numbers

- Assume 4-bit addresses, i.e. address range 0 – 15

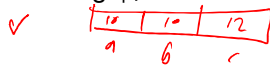


```
char *c;
int *i;
```

*c = 4      i = 8*  
*\*c = '2'    \*i = 42*  
*c = 8*  
*✓ = \*c*

## Converting Page/Frame numbers of addresses

- Bit shifting operators '<<', '>>'
- Bit masking '|', '&'

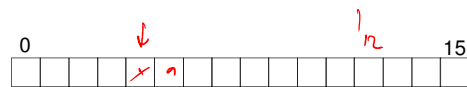


*a = ✓ >> 22*  
*b = ✓ << 10 >> 22*  
*c*

## Indexing off Pointers

Memory

- 4-bit addresses, i.e. address range 0 – 15



```
char *c;
```

*c = 4      c[1] = 9*  
*\*c = x      [0] = x*

## Bump pointer allocator

Memory

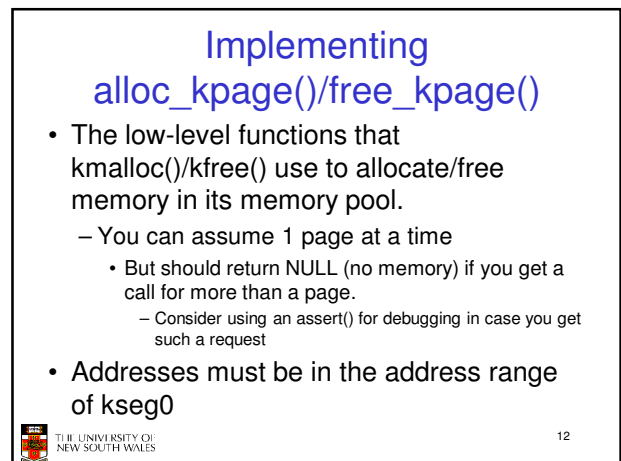
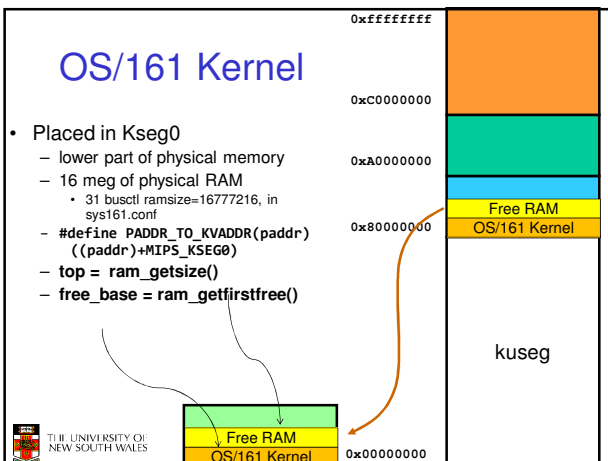
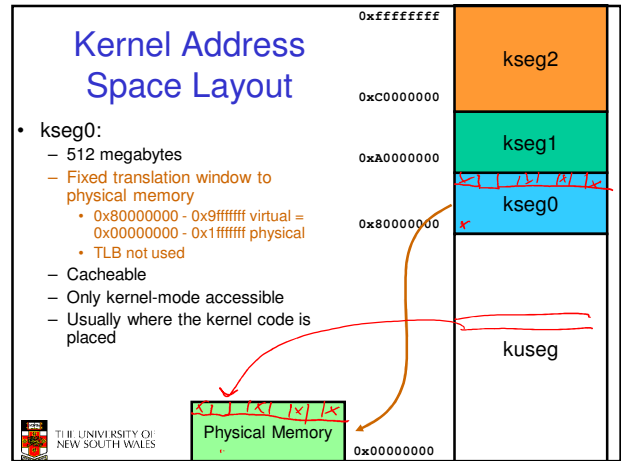
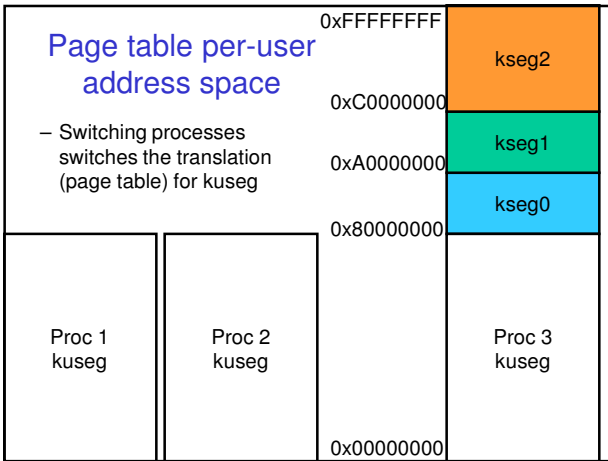
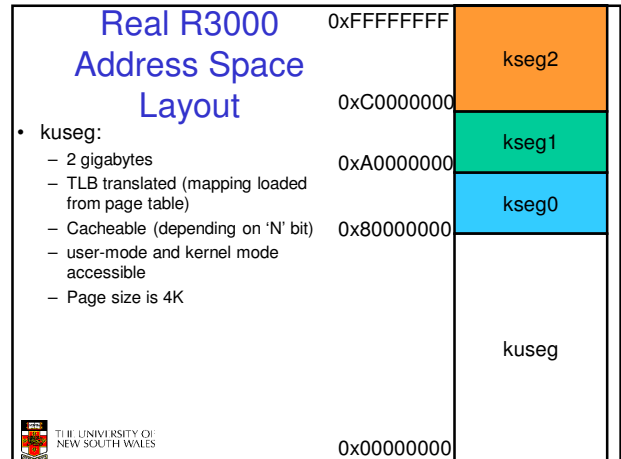
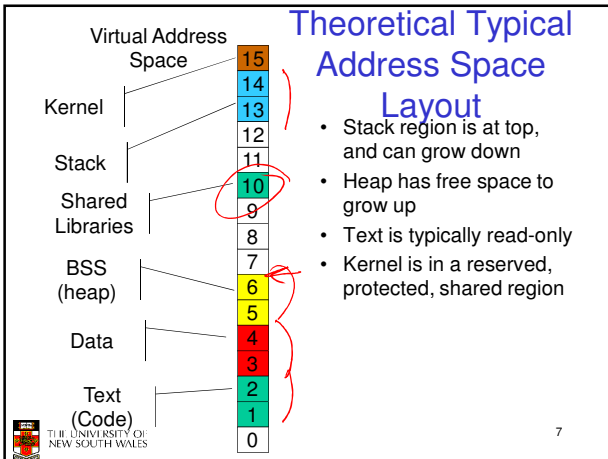
- 4-bit addresses, i.e. address range 0 – 15

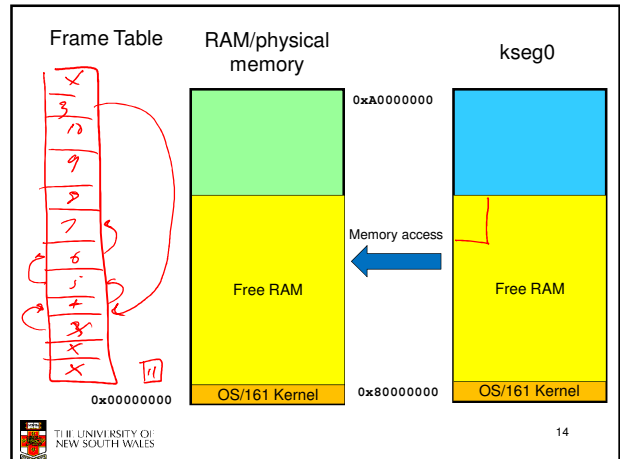
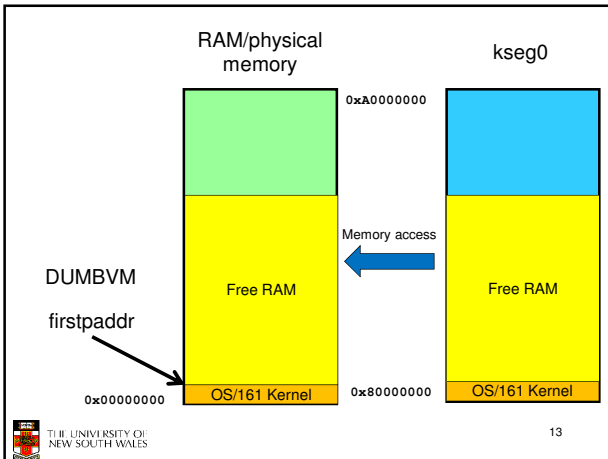


```
char *c;
unsigned int next_free;
```

## Assignment 3

- Two parts (taking a data-structure view)
  - Frame table
    - An allocator for physical memory
  - Two-level page table and 'region' support
    - Virtual memory for applications





## How to 'place' my frame table?

```
struct frame_table_entry *frame_table;

location = top_of_ram - (top_of_ram / PAGE_SIZE *
    sizeof(page_table_entry));

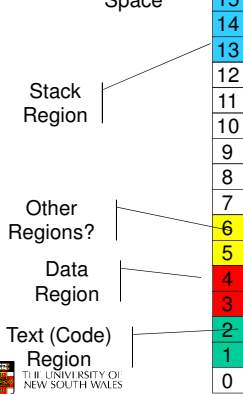
frame_table = (struct frame_table_entry *) location;
```

## How can my my allocator work before and after it is initialised?

```
struct frame_table_entry *frame_table = 0;

alloc_kpages()
{
    if (ft == 0) {
        /* use ram_stealmem */
    } else {
        /* use my allocator as frame table is now initialised */
    }
}
```

## Virtual Address Space



## KUseg layout

- Stack region is at top, and can grow down
- Other regions determined by ELF file
  - see load\_elf()
  - number can vary
  - permissions specified also
  - cs161-objdump -p testbin/huge

```
thresher% cs161-objdump -h ../bin/true

../bin/true: file format elf32-tradbigmips

Sections:
Idx Name          Size      VMA           LMA             File off  Algn
 0 .reginfo        00000018  00400094  00400094  00000094  2**2
   CONTENTS, ALLOC, LOAD, READONLY, DATA, LINK_ONCE_SAME_SIZE
 1 .text           000001d0  004000b0  004000b0  000000b0  2**4
   CONTENTS, ALLOC, LOAD, READONLY, CODE
 2 .data           00000000  10000000  10000000  00001000  2**4
   CONTENTS, ALLOC, LOAD, DATA
 3 .sbss           00000008  10000000  10000000  00001000  2**2
   ALLOC
 4 .bss            00000000  10000010  10000010  00001008  2**4
   ALLOC
 5 .comment        00000036  00000000  00000000  00001008  2**0
   CONTENTS, READONLY
 6 .pd_r           000004a0  00000000  00000000  00001040  2**2
   CONTENTS, READONLY
 7 .mdebug.abi32  00000000  00000000  00000000  000014e0  2**0
   CONTENTS, READONLY

thresher%
```

```

thresher% cs161-objdump-p ./bin/true
./bin/true: file format elf32-tradbigmips

Program Header:
0x70000000 off 0x00000094 vaddr 0x00400094 paddr 0x00400094 align 2**2
  filesz 0x00000018 memsz 0x00000018 flags r--
LOAD off 0x00000000 vaddr 0x00400000 paddr 0x00400000 align 2**12
  filesz 0x00000280 memsz 0x00000280 flags r-x
LOAD off 0x00001000 vaddr 0x10000000 paddr 0x10000000 align 2**12
  filesz 0x00000000 memsz 0x00000010 flags rw-
private flags = 1001:[abi=O32][mips1][not 32bitmode]

thresher%

```

## Process Layout

- Process layout in KUseg
  - regions specified by calls to
    - as\_define\_stack()
      - usually implemented as a linked list of region specifications
    - as\_define\_region()
      - make READONLY regions READWRITE for loading purposes
  - as\_prepare\_load()
    - enforce READONLY again
  - as\_complete\_load()
    - enforce READONLY again

## Process Layout

- Need to keep translation table for KUSEG

- as\_create()
  - allocate a data structure used to keep track of an address space
    - i.e. regions
    - PT Level 1
    - proc\_getas() used to get access to current address space struct
- as\_destroy()
  - deallocate book keeping and page tables.
    - deallocating frames used

- as\_copy()
  - allocates a new (destination) address space
  - adds all the same regions as source
  - roughly, for each mapped page in source
    - allocate a frame in dest
    - copy contents from source frame to dest frame
    - add PT entry for dest
- as\_activate()
  - flush TLB
  - (or set the hardware asid)
- as\_deactivate()
  - flush TLB
  - (or flush an asid)

## VM Fault Approximate Flow Chart

```

graph TD
    vm_fault[vm_fault] --> VM_FAULT_READONLY{VM_FAULT_READONLY}
    VM_FAULT_READONLY -- Yes --> EFAULT1[EFAULT]
    VM_FAULT_READONLY -- No --> lookup_PT[lookup PT]
    lookup_PT --> Valid_Translation{Valid Translation}
    Valid_Translation -- Yes --> Load_TLB1[Load TLB]
    Valid_Translation -- No --> Look_up_region[Look up region]
    Look_up_region --> Valid_Region{Valid Region}
    Valid_Region -- No --> EFAULT2[EFAULT]
    Valid_Region -- Yes --> Allocate_Frame[Allocate Frame, Zero-fill, Insert PTE]
    Allocate_Frame --> Load_TLB2[Load TLB]
    Allocate_Frame --> Note[Newly allocated user-level pages are expected to be zero-filled]

```

## kprintf()

- Do not use it in `vm_fault()`!

## trace161 can help with debugging

<http://cgi.cse.unsw.edu.au/~cs3231/06s1/os161/man/sys161/index.html>

- The following additional options control trace161's tracing and are ignored by `sys161`:
- `-f tracefile`
  - Set the file trace information is logged to. By default, `stderr` is used. Specifying `-f` sends output to `stdout` instead of `stderr`.
- `-t traceflags`
  - Tell System/161 what to trace. The following flags are available:
    - `d` Trace disk I/O
    - `e` Trace emults I/O
    - `j` Trace jumps and branches
    - `k` Trace instructions in kernel mode
    - `n` Trace network I/O
    - `r` Trace TLB/MMU activity
    - `u` Trace instructions in user mode
    - `x` Trace exceptions
- Caution: tracing instructions generates huge amounts of output that may overwhelm smaller host systems.

```
wagner% trace161 -tt kernel
sys161: System/161 release 1.12, compiled Jun 21 2005 10:34:06
sys161: Tracing enabled: tb
trace: tlbp: 81000/000 -> 00000 ----: [0]
trace: tlbp: 81001/000 -> 00000 ----: [1]
trace: tlbp: 81002/000 -> 00000 ----: [2]
trace: tlbp: 81003/000 -> 00000 ----: [3]
trace: tlbp: 81004/000 -> 00000 ----: [4]
trace: tlbp: 81005/000 -> 00000 ----: [5]
trace: tlbp: 81006/000 -> 00000 ----: [6]
trace: tlbp: 81007/000 -> 00000 ----: [7]
trace: tlbp: 81008/000 -> 00000 ----: [8]
trace: tlbp: 81009/000 -> 00000 ----: [9]
trace: tlbp: 8100a/000 -> 00000 ----: [10]
trace: tlbp: 8100b/000 -> 00000 ----: [11]
trace: tlbp: 8100c/000 -> 00000 ----: [12]
trace: tlbp: 8100d/000 -> 00000 ----: [13]
trace: tlbp: 8100e/000 -> 00000 ----: [14]
trace: tlbp: 8100f/000 -> 00000 ----: [15]
trace: tlbp: 81010/000 -> 00000 ----: [16]
trace: tlbp: 81011/000 -> 00000 ----: [17]
trace: tlbp: 81012/000 -> 00000 ----: [18]
trace: tlbp: 81013/000 -> 00000 ----: [19]
trace: tlbp: 81014/000 -> 00000 ----: [20]
```

```
trace: tlbp: 81015/000 -> 00000 ----: [21]
.....
trace: tlbp: 8103c/000 -> 00000 ----: [60]
trace: tlbp: 8103d/000 -> 00000 ----: [61]
trace: tlbp: 8103e/000 -> 00000 ----: [62]
trace: tlbp: 8103f/000 -> 00000 ----: [63]
trace: tlbp: 81040/000 -> NOT FOUND
trace: tlbwi: [0] 81000/000 -> 00000 ---- ==> 81040/000 -> 00000 ----
trace: tlbp: 81041/000 -> NOT FOUND
trace: tlbwi: [1] 81001/000 -> 00000 ---- ==> 81041/000 -> 00000 ----
trace: tlbp: 81042/000 -> NOT FOUND
trace: tlbwi: [2] 81002/000 -> 00000 ---- ==> 81042/000 -> 00000 ----
trace: tlbp: 81043/000 -> NOT FOUND
trace: tlbwi: [3] 81003/000 -> 00000 ---- ==> 81043/000 -> 00000 ----
trace: tlbp: 81044/000 -> NOT FOUND
trace: tlbwi: [4] 81004/000 -> 00000 ---- ==> 81044/000 -> 00000 ----
trace: tlbp: 81045/000 -> NOT FOUND
trace: tlbwi: [5] 81005/000 -> 00000 ---- ==> 81045/000 -> 00000 ----
trace: tlbp: 81046/000 -> NOT FOUND
trace: tlbwi: [6] 81006/000 -> 00000 ---- ==> 81046/000 -> 00000 ----
trace: tlbp: 81047/000 -> NOT FOUND
trace: tlbwi: [7] 81007/000 -> 00000 ---- ==> 81047/000 -> 00000 ----
trace: tlbp: 81048/000 -> NOT FOUND
```

```
trace: tlbp: 81015/000 -> 00000 ----: [21]
.....

trace: tlbp: 8107c/000 -> NOT FOUND
trace: tlbwi: [60] 8103c/000 -> 00000 ---- ==> 8107c/000 -> 00000 ----
trace: tlbp: 8107d/000 -> NOT FOUND
trace: tlbwi: [61] 8103d/000 -> 00000 ---- ==> 8107d/000 -> 00000 ----
trace: tlbp: 8107e/000 -> NOT FOUND
trace: tlbwi: [62] 8103e/000 -> 00000 ---- ==> 8107e/000 -> 00000 ----
trace: tlbp: 8107f/000 -> NOT FOUND
trace: tlbwi: [63] 8103f/000 -> 00000 ---- ==> 8107f/000 -> 00000 ----
```

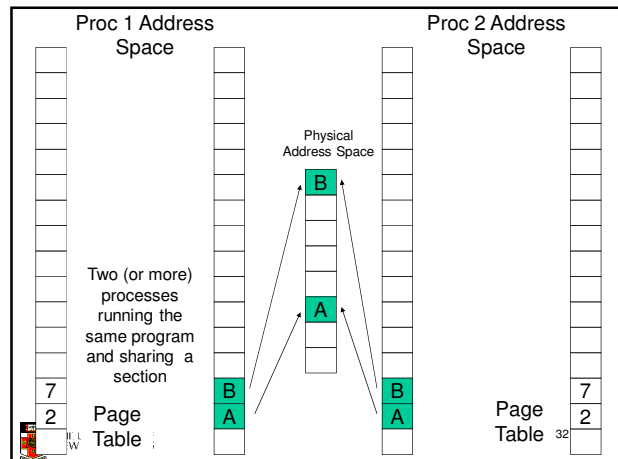
OS/161 base system version 1.10  
Copyright (c) 2000, 2001, 2002, 2003  
President and Fellows of Harvard College. All rights reserved.

Put-your-group-name-here's system version 0 (ASST1 #1)

Cpu is MIPS r2000/r3000  
344k physical memory available  
Device probe...  
lamebus0 (system main bus)  
emu0 at lamebus0

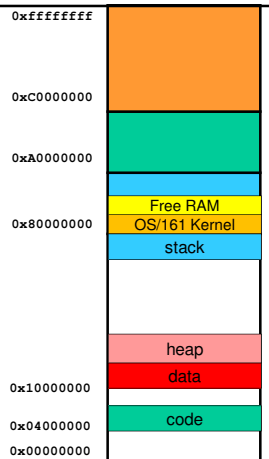
## Advance Assignment

- Shared pages and copy-on-write
- Sbrk()
- Demand loading and mmap
- Paging



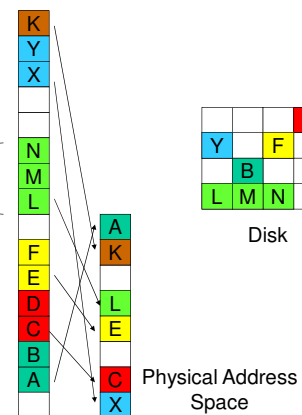
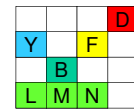
## sbrk

- The "break" is the end address of a process's heap region.
- The sbrk call adjusts the "break" by the amount.
- It returns the old "break". Thus, to determine the old current "break", call sbrk(0).
- The heap region is initially empty, so at process startup, the beginning of the heap region is the same as the end and may thus be retrieved using sbrk(0).



## Memory-mapped files and paging

Memory mapped file



## mmap semantics

```
void *mmap(size_t length, int prot, int fd, off_t offset);
int munmap(void *addr);
```