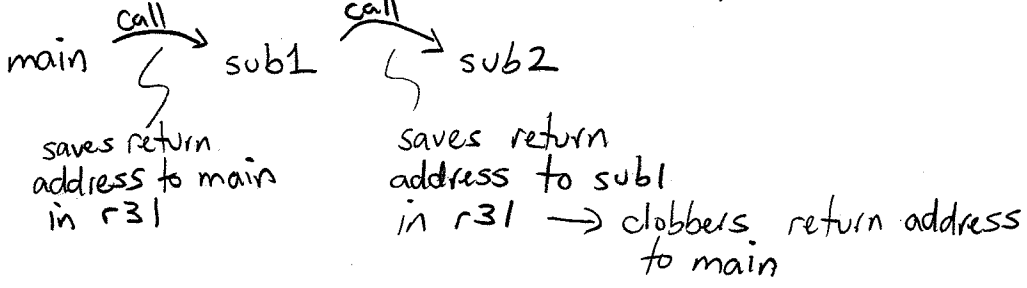
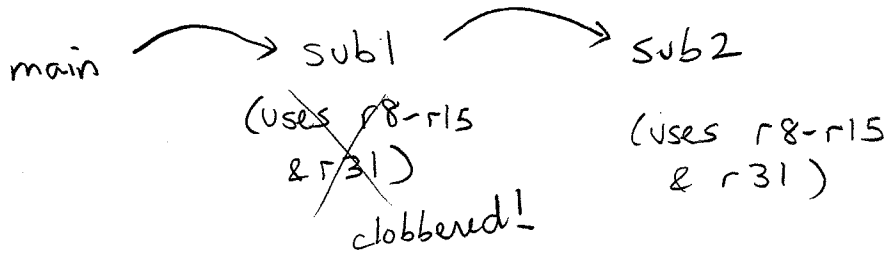
Stacks and Subroutines

Problem: nested subroutine calls clobber r31



other registers (used in subroutine) get clobbered too!



Solution: do two things a) save important registers <u>on stack</u> before subroutine call

b) restore registers <u>from stack</u> after subroutine returns

Stack: <u>ideally</u>, an <u>unbounded</u> region of memory where we store values in last-in-first-out (<u>LIFO</u>) order

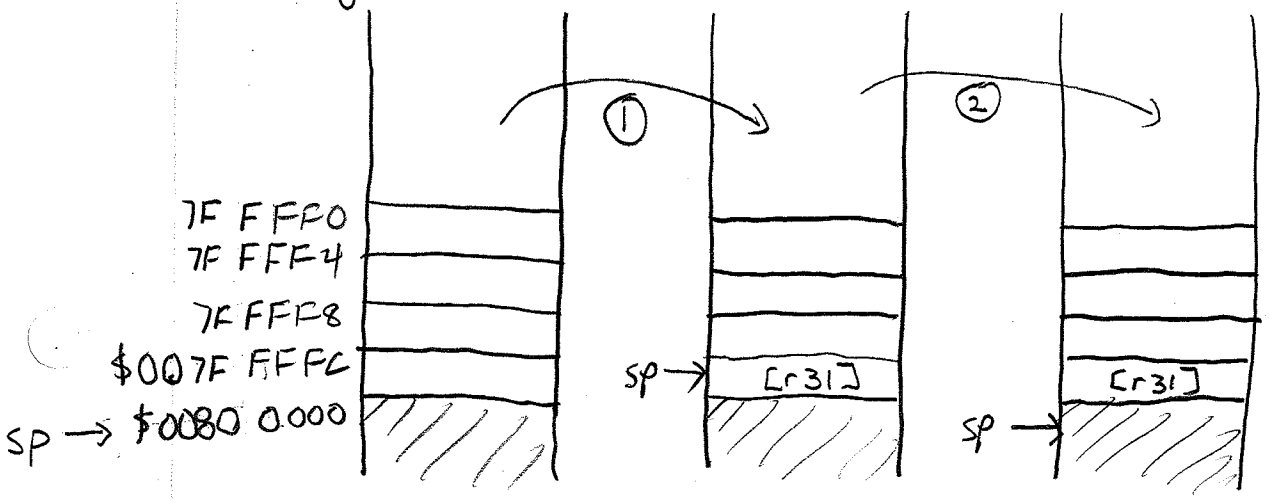
- <u>ideally</u> → practically, there will be a memory limit
- unbounded → no preset size limit
- LIFO → <u>reversing</u> eg: long driveway full of cars, last one in is first one out!

convention: r27 is "stack pointer" or sp used to keep track of <u>top of stack</u>

Example

```
.equ DRAM_END, 0x0080 0000

_start:  movia  sp, SRAM_END      <- init stack
         call   sub1
         call   sub3
stop:    br     stop

sub1:    addi   sp,sp, -4         <- save r31 (1)
         stw    r31, 0(sp)
         call   sub2
         ldw    r31, 0(sp)        <- restore r31 (2)
         addi   sp,sp, 4
         ret

sub2:    ret

sub3:    ret
```

Memory:



convention

r2, r3	return value
r4 - r7	incoming params
r8 - r15 (subr.)	caller-saved ∴ safe to use inside subroutine
r16 - r23 (main)	callee-saved ∴ subroutine must save + restore if it changes them
r1, r24 - r31	don't use (mostly specialized like r27 = sp, r31 = ra, ...)

caller employer
↓ ↓
callee employee

Saving multiple registers

```
sub1: addi  sp, sp, -4      } save ret. addr (sub1 will call sub2)
      stw   r31, 0(sp)
      ...                   } sub1 needs r4, r8, r9
```

Can also save r16-r23 if needed by sub1 (callee saved)

caller-saved:

```
      addi  sp, sp, -12     } save regs needed by sub1
      stw   r4, 8(sp)
      stw   r8, 4(sp)
      stw   r9, 0(sp)
```

sub2 clobbers (r4, r8, ... r15, r31)
result is in r2, r3

```
      mov   r4, someParamForSub2   } call sub2
      call  sub2
```

caller-saved:

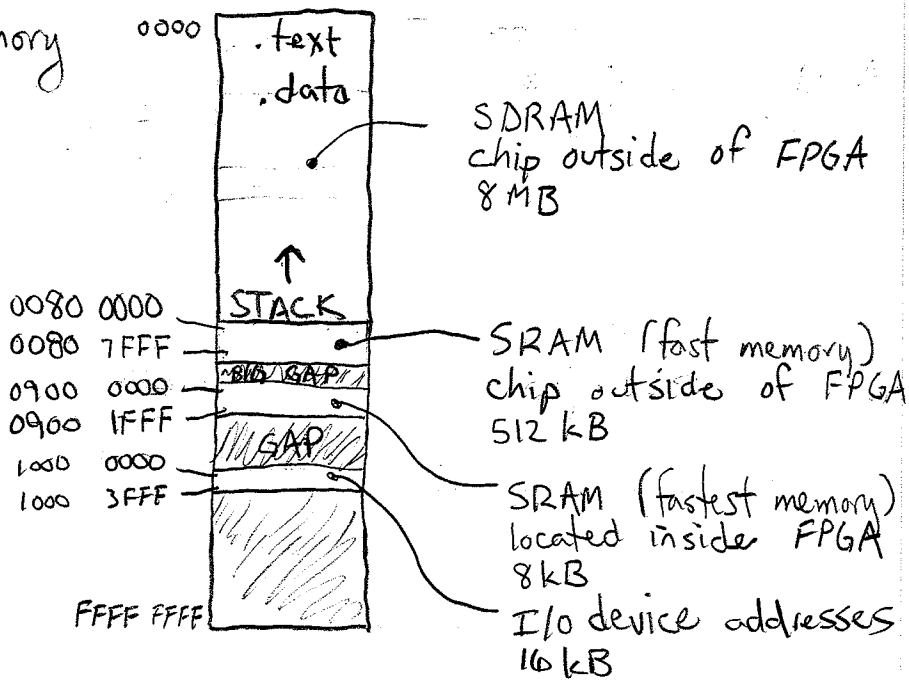
```
      ldw   r9, 0(sp)       } restore regs for sub1 in reverse order
      ldw   r8, 4(sp)
      ldw   r4, 8(sp)
      addi  sp, sp, 12
```

```
      ...                   } examine sub2 result in r2, r3
```

```
      ldw   r31, 0(sp)      } restore ret. addr & return from sub1
      addi  sp, sp, 4
      ret
```

(callee saved) also restore r16-r23 if used by sub1

Bigger view of memory



Stack rules

① sp always points to top of stack – most recent item put in stack

eg 1: `ldw r2, 0(sp)` — always checks top of stack

eg 2:
```
addi sp, sp, -4
stw  r31, 0(sp)
```
— places new item on top of stack

② all stack operations are done manually with ordinary instructions – no magic "stack instructions" or "hardware stack" to do things in hidden/automatic way for Nios II

③ initially, sp should point to just past the end of stack memory

④ as data { added to / removed from } stack, { sp gets smaller PUSH / sp gets bigger PULL }

⑤ MAINTAIN BALANCE always PULL what you PUSH in equal amounts & in reverse order