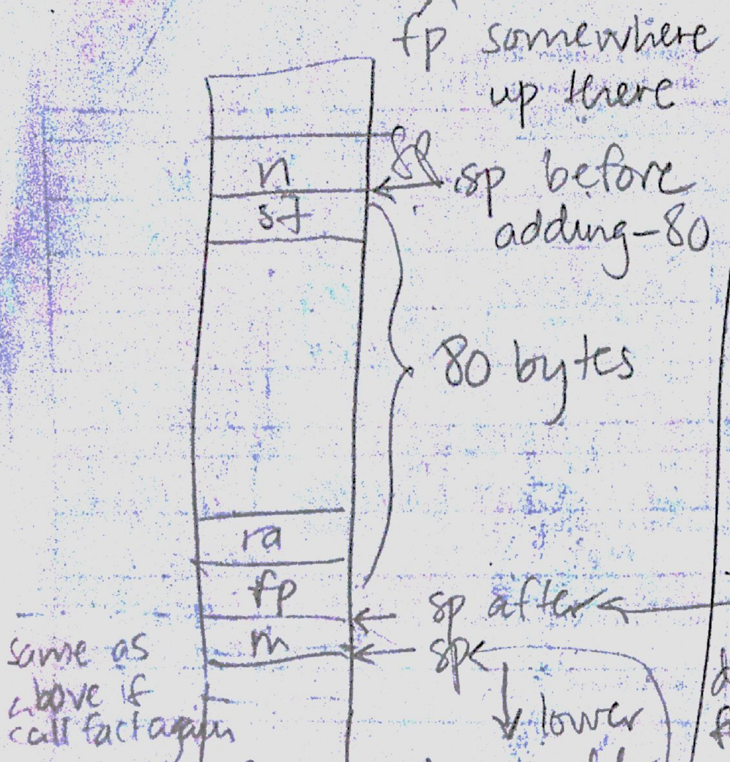

– Allocate
~~4 for n~~
20 ~~18~~ s-p registers (10 t, 8s, ra, fp) } 80 bytes

* Return instr just jumps back to add of last instruction before fn call

Machine code: (assume caller put n on stack)

```
fact: add $sp, $sp, -80
      sw $s7, 76($sp)
```

(~~stores what's at $ stack ptr + 76 into s7~~)

do once for each s → 8x

writes what's in $s7 into $sp + 76

s are callee saved
t —— caller ——

```
sw $ra, 4($sp)
sw $fp, 0($sp)
sub $sp, $sp, 4
```

x($ra) = x from content of $ra

should save in case you call another fn which changes $ra

code for fn entry

(move stack ptr forward (up) by 4)

reserve space for n-1 saving the caller's fp

```
add $fp, $sp, 84
```

(move frame ptr to 84 bytes from stack ptr (doesn't have to be after params)

add n

store caller's t regs

```
sw $t9, -36($fp)
sw $t1, -68($fp)
sw $t0, -72($fp)
```

x10

Actual fn running

```
lw $t0, 0($fp)     → loads n
li $t1, 1
sub $t3, $t0, $t2
```

// about to call fact again here so need to load its param here ie n-1

```
sub $sp, $sp, 4    → reserve space for n-1
sw $t3, 0($sp)     → put n-1 into reserved space
jal fact
```

(call fact again)
(fact was defined @ beginning)

```
mov $t4, $v0       → assuming called fact put return value here
lw $t5, 0($fp)     → load n
mul $t6, $t4, $t5
sw $t6, -84($fp)
lw $t7, -84($fp)
mov $v0, $t7
```

```
// done, return to old fp
lw $fp, -80($fp)
add $sp, $sp, 4
lw $ra 4($sp)
//reload s regs
lw $s0, 48($sp)
add $sp, $sp, 80
add $sp, $sp, 4
// go back to caller
jr $ra
```

Example machine code

```
int fact(int n) {
    int m;
    m = n * fact(n-1);
    return m;
}
```

→ symbol table

glob

fact (fn, $\overset{int}{\vee}$ 1, int)

fact → 8

n int } 8 bytes

m int }

- give params their own offset: n = 0 } different

— local vars ———— m = 0 } meaning

$\boxed{IR}$

```
LABEL fact | FN ENTRY fact (placeholder before inserting
                 actual implementation)
LW  t0, n
LW  t1, m
LI  t2, 1
SUB t3, t0, t2
PARAM t3
CALL t4, fact
(MUL t0, t4 → ok to load n again for now)
                       this is advanced
LW  t5, n
MUL t6, t4, t5
SW  t6, m | LW t7, m | RETURN t7
```

code → generation

```
struct list {
    void * first;
}
```

x = y++
x 1 = ++y

```
struct ir {
    int id;
    ir * next;
}
```

Reason element is next to itself when updating child, parent points to child so is updated too, then is updated a 2nd time

a++; → a = a+1;

TA office hours 4/23

```
int b[3];
int f(int c) {
    char a[5];
    a[3] = 2;
```

IR Assembly

```
li t2, 2
la t0, a  →  add $t0, 15, $fp
li t1, 3
```