ECE154A — Discussion 05

George Higgins Hutchinson

October 29, 2021

Keep your eyes open for...

- PSet 3: due Friday, November 5
- Lab 3: due Friday, November 5

Recursion

```
int factorial(int n) {
  if (n <= 1)  return 1;
  return (n * factorial(n-1));
}
```

Note that this implementation is not tail-call optimized.

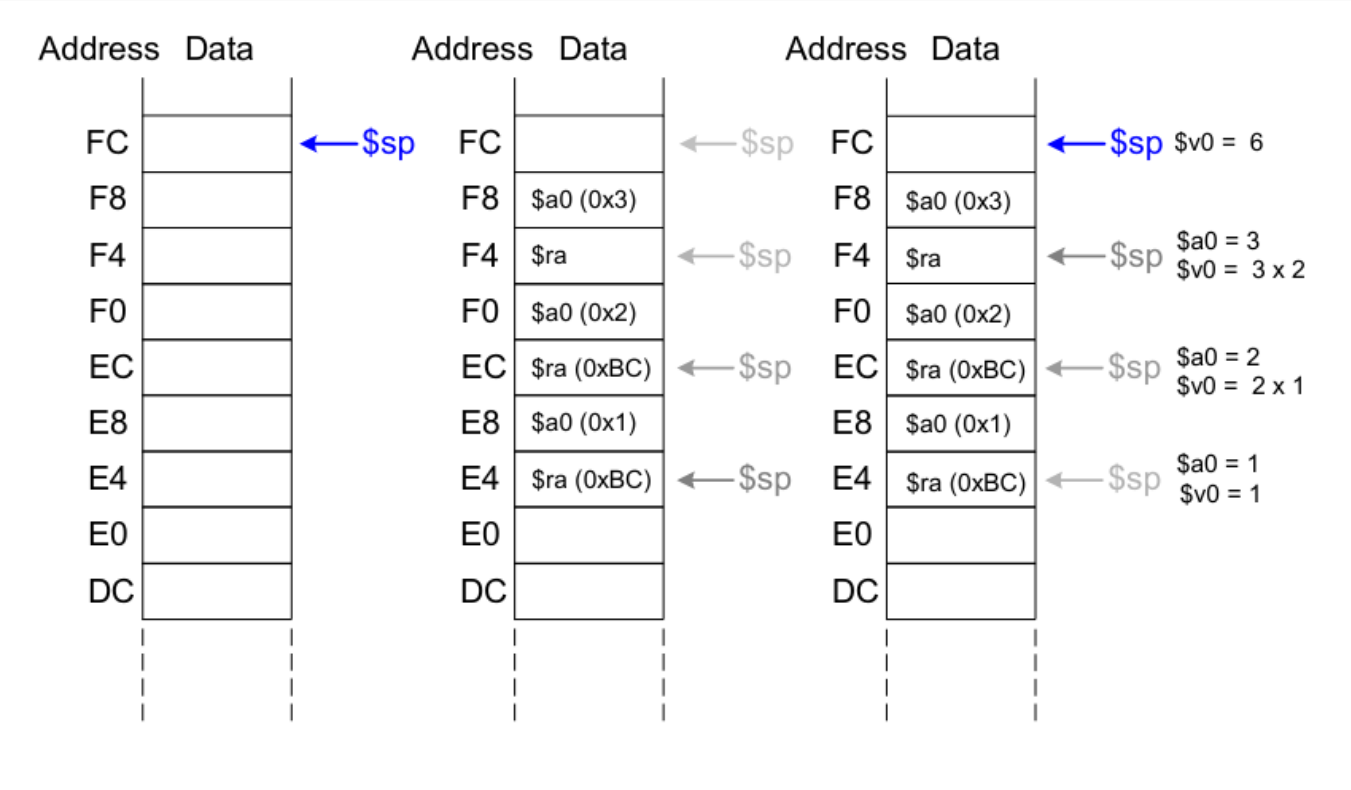
But what does it look like through the compiler?

Recursion in Assembly

```
0x90   factorial:   addi $sp, $sp, -8      # make room
0x94                sw $a0, 4($sp)         # store $a0
0x98                sw $ra, 0($sp)         # store $ra
0x9c                addi $t0, $0, 2
0xa0                slt $t0, $a0, $t0      # a ≤ 1 ?
0xa4                beq $t0, $0, else      # no: go to else
0xa8                addi $v0, $0, 1        # yes: return 1
0xac                addi $sp, $sp, 8       # restore $sp
0xb0                jr $ra                 # return
0xb4   else:        addi $a0, $a0, -1      # n = n - 1
0xb8                jal factorial          # recursive call
0xbc                lw $ra, 0($sp)         # restore $ra
0xd0                lw $a0, 4($sp)         # restore $a0
0xd4                addi $sp, $sp, 8       # restore $sp
0xd8                mul $v0, $a0, $v0      # n * factorial(n-1)
0xdc                jr $ra                 # return
```

Recursion in MIPS

What does this look like on the stack?



Linked Lists — from 2019:MT2:1

```
typedef struct llnode {
  unsigned int x;
  int *v;
  llnode *next;
}
int val;
llnode *p;
/* assume p initialized */
do {
  (p->v)[p->num] = val;
  p = p->next;
} while (p != null)
```

Translate the loop to MIPS.

Linked Lists solution

```
loop:   lw    $t0, 0($a1)     ; do {$t0 = p->num
        sll   $t0, $t0, 2     ; $t0 = (p->num)*4
        lw    $t1, 4($a1)     ; $t1 = p->array
        add   $t1, $t1, $t0   ; $t1 = p->array + (p->num)*4
        sw    $a0, 0($t1)     ; (p-> array)[p->num] = $a0
        lw    $a1, 8($a1)     ; p = p -> next
        bne   $0, $a1, loop   ; } while(p!= NULL)
```