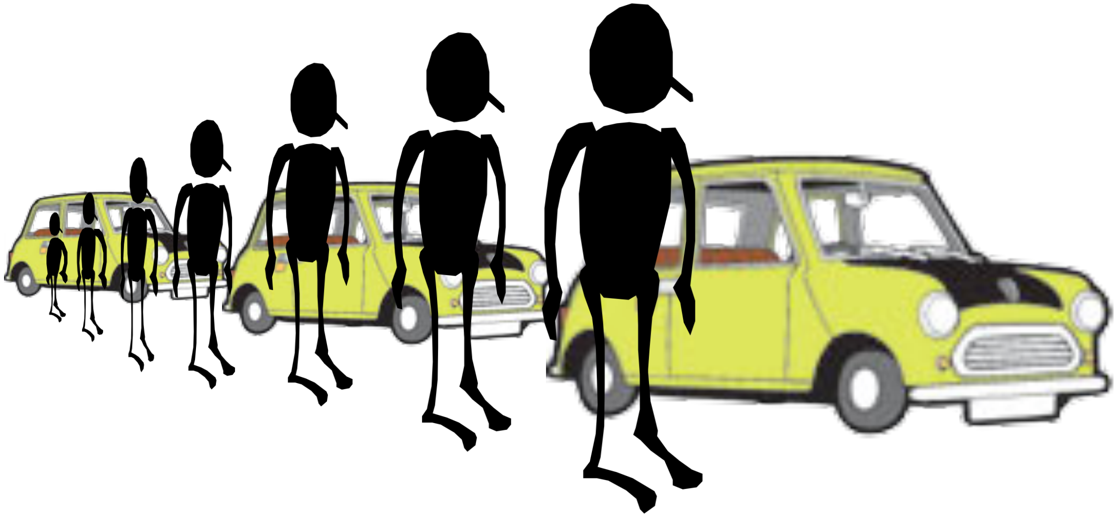
# A Last Bit of Assembly



Are  you really comfortable writing assembly code yet?

# Assembly Exercise

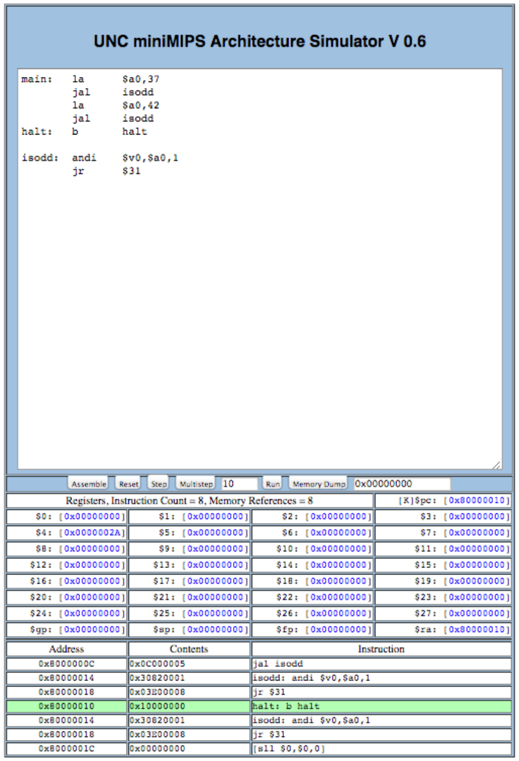
Let's write some assembly language programs

Program #1: A function "isodd(int X)" which returns 1 if it's argument "X" is odd and 0 otherwise

```
main:   la      $a0,37
        jal     isodd
        la      $a0,42
        jal     isodd
halt:   b       halt

isodd:  andi    $v0,$a0,1
        jr      $31
```

Does isOdd() obey our procedure -linkage conventions?



UNC miniMIPS Architecture Simulator V 0.6

```
main:   la      $a0,37
        jal     isodd
        la      $a0,42
        jal     isodd
halt:   b       halt

isodd:  andi    $v0,$a0,1
        jr      $31
```

Multistep: 10

Memory Dump: 0x00000000

| Registers, Instruction Count = 8, Memory References = 8 | | | [K]$pc: [0x80000010] |
|---|---|---|---|
| $0: [0x00000000] | $1: [0x00000000] | $2: [0x00000000] | $3: [0x00000000] |
| $4: [0x0000002A] | $5: [0x00000000] | $6: [0x00000000] | $7: [0x00000000] |
| $8: [0x00000000] | $9: [0x00000000] | $10: [0x00000000] | $11: [0x00000000] |
| $12: [0x00000000] | $13: [0x00000000] | $14: [0x00000000] | $15: [0x00000000] |
| $16: [0x00000000] | $17: [0x00000000] | $18: [0x00000000] | $19: [0x00000000] |
| $20: [0x00000000] | $21: [0x00000000] | $22: [0x00000000] | $23: [0x00000000] |
| $24: [0x00000000] | $25: [0x00000000] | $26: [0x00000000] | $27: [0x00000000] |
| $gp: [0x00000000] | $sp: [0x00000000] | $fp: [0x00000000] | $ra: [0x80000010] |

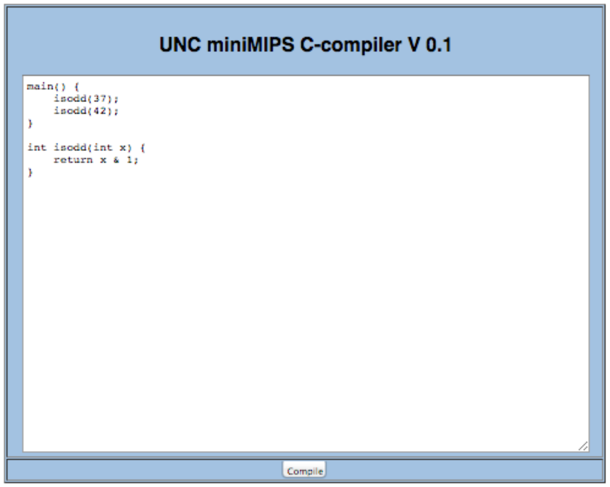
| Address | Contents | Instruction |
|---|---|---|
| 0x8000000C | 0x0C000005 | jal isodd |
| 0x80000014 | 0x30820001 | isodd: andi $v0,$a0,1 |
| 0x80000018 | 0x03E00008 | jr $31 |
| 0x80000010 | 0x10000000 | halt: b halt |
| 0x80000014 | 0x30820001 | isodd: andi $v0,$a0,1 |
| 0x80000018 | 0x03E00008 | jr $31 |
| 0x8000001C | 0x00000000 | [sll $0,$0,0] |

# Let's see C

Now let's write the same program in C and see what the compiler generates.

```
main() {
    isodd(37);
    isodd(42);
}

int isodd(int x) {
    return x & 1;
}
```

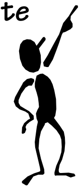


```
main() {
    isodd(37);
    isodd(42);
}

int isodd(int x) {
    return x & 1;
}
```

```
crt0:
lui $sp,0x7fff
ori $sp,$sp,0xfffc
jal main
halt:
b halt
.globl main
.text
.text
main:
addiu $sp,$sp,-32
sw $31,20($sp)
la $4,37
jal isodd
la $4,42
jal isodd
move $2,$0
L_1:
lw $25,16($sp)
lw $31,20($sp)
addiu $sp,$sp,32
jr $31
.globl isodd
.text
isodd:
andi $2,$4,1
L_2:
jr $31
```

This code is nearly identical to what we wrote



```
crt0: lui $sp,0x7fff
      ori $sp,$sp,0xfffc
      jal main
halt: b halt

.globl main
.text
.text
main: addiu $sp,$sp,-32
      sw $31,20($sp)
      la $4,37
      jal isodd
      la $4,42
      jal isodd
      move $2,$0
L_1:  lw $25,16($sp)
      lw $31,20($sp)
      addiu $sp,$sp,32
      jr $31

.globl isodd
.text
isodd: andi $2,$4,1
L_2:   jr $31
```

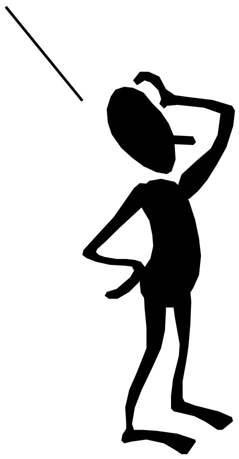
# Your Turn

Program #2: A function "ones(int X)" that returns a count of the number of ones in its argument "X"

Count the number of ones in a binary number

# And Now in C

```
int ones(int x) {
    int count = 0;
    while (x != 0) {
        count += x & 1;
        x = x >> 1;
    }
    return count;
}
```



It’s a little different this time. But, this works too.

```
.globl ones
.text
ones: addiu $sp,$sp,-16
      sw $30,0($sp)
      move $30,$0
      b L_4
L_3:  la $24,1
      and $15,$4,$24
      addu $30,$30,$15
      srav $4,$4,$24
L_4:  bne $4,$0,L_3
      move $2,$30
L_2:  lw $30,0($sp)
      addiu $sp,$sp,16
      jr $31
```

# I'm getting this!

Let's try a recursive example, and start with C this time.

```
int factorial(int x) {
    if (x > 1)
        return x*factorial(x-1);
    else
        return 1;
}

main () {
    factorial(7);
}
```

```
.globl factorial
.text
factorial: addiu $sp,$sp,-32
           sw $31,20($sp)
           sw $4,32($sp)
           lw $24,0+32($sp)
           la $15,1
           slt $1,$15,$24
           beq $1,$0,L_2
           lw $24,0+32($sp)
           sw $24,-4+32($sp)
           subiu $4,$24,1
           jal factorial
           move $24,$2
           lw $15,-4+32($sp)
           mul $2,$15,$24
           b L_1
L_2:       la $2,1
L_1:       lw $25,16($sp)
           lw $31,20($sp)
           addiu $sp,$sp,32
           jr $31
```

# Got this!

Now in assembly.

```
int factorial(int x) {
    if (x > 1)
        return x*factorial(x-1);
    else
        return 1;
}

main () {
    factorial(7);
}
```

```
.globl factorial
.text
factorial: addiu $sp,$sp,-32
           sw $ra,20($sp)
           sw $a0,32($sp)
           la $t0,1
           slt $t0,$t0,$a0
           beq $t0,$0,else
           subiu $a0,$a0,1
           jal factorial
           lw $a0,32($sp)
           mul $v0,$a0,$v0
           b return
else:      la $v0,1
return:    lw $ra,20($sp)
           addiu $sp,$sp,32
           jr $ra
```