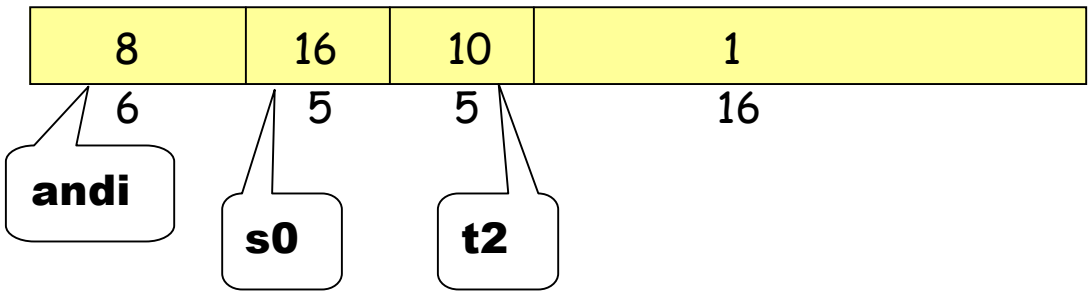
Using AND for bit manipulation

To check if a register $s0 contains an odd number, AND it with a mask that contains all 0's except a 1 in the LSB position, and check if the result is zero (we will discuss decision making later)

andi $t2, $s0, 1

This uses **I-type format** (why?):



Now we have to test if $t2 = 1 or 0

Making decisions

```
if (i == j)   then   f = g + h;   else   f = g – h
```

Use **bne** = branch-nor-equal, **beq** = branch-equal, and **j** = jump

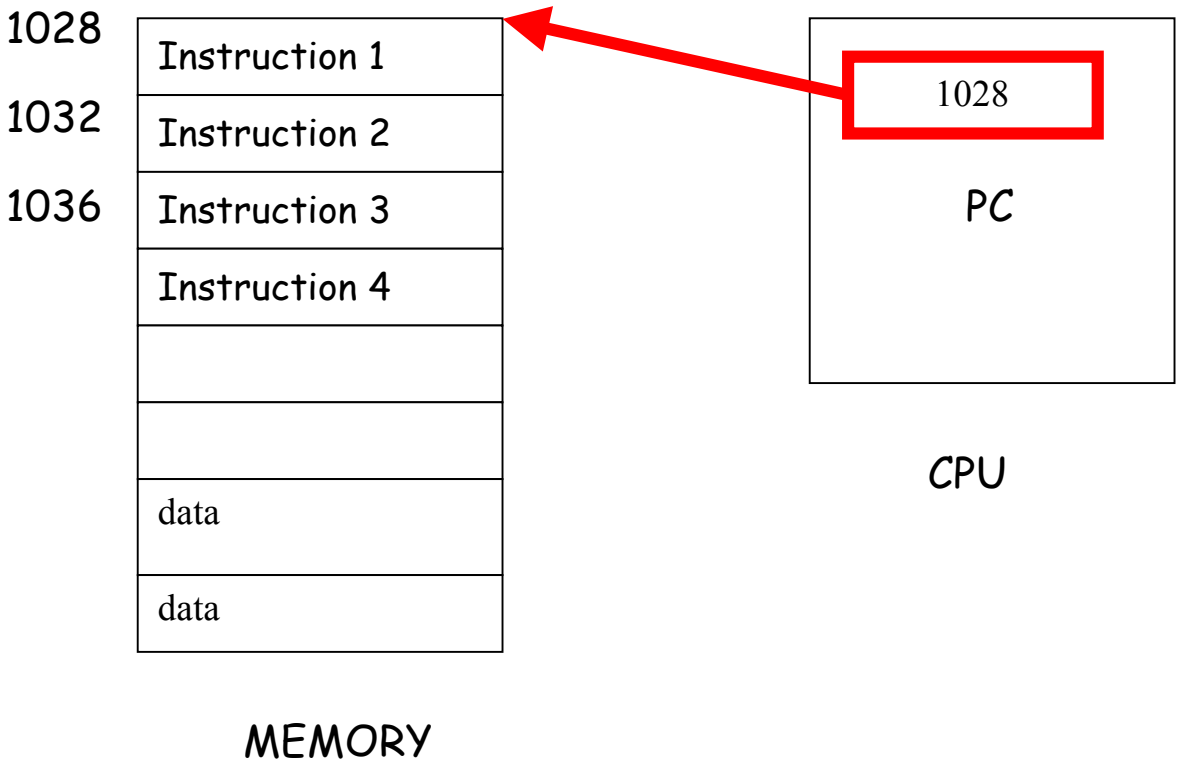
Assume that f, g, h, are mapped into $s0, $s1, $s2

i, j are mapped into $s3, $s4

```
        bne $s3, $s4, Else      # goto Else when i≠j
        add $s0, $s1, $s2       # f = g + h
        j   Exit                # goto Exit
Else:   sub $s0, $s1, $s2       # f = g – h
Exit:
```

The program counter and control flow

Every machine has a **program counter** (called PC) that points to the next instruction to be executed.



Ordinarily, PC is incremented by 4 after each instruction is executed. A branch instruction alters the flow of control by modifying the PC.

Compiling a while loop

while (A[i] == k) i = i + j;

Initially $s3, $s4, $s5 contains i, j, k respectively.

Let $s6 store the base of the array A. Each element of A is a 32-bit word.

```
Loop:   add $t1, $s3, $s3    # $t1 = 2*i
        add $t1, $t1, $t1    # $t1 = 4*i
        add $t1, $t1, $s6    # $t1 contains address of A[i]
        lw $t0, 0($t1)       # $t0 contains $A[i]
        add $s3, $s3, $s4    # i = i + j
        bne $t0, $s5, Exit   # goto Exit if A[i] ≠ k
        j  Loop              # goto Loop
Exit:   <next instruction>
```

Note the use of pointers.

Anatomy of a MIPS assembly language program running on the SPIM simulator

```
        .data
L1: .word 0x2345          # some arbitrary value
L2: .word 0x3366          # some arbitrary value
Res: .space 4
```

```
        .text
        .globl main
main:  lw $t0, L1($0)           #load the first value
        lw $t1, L2($0)          # load the second value
        and $t2, $t0, $t1       # compute bit-by-bit AND
        or $t3, $t0, $t1        # compute  bit-by-bit OR
        sw $t3, Res($0)         # store result in memory
```

```
      li $v0, 10            # code for program end
      syscall
```

Another example of input-output

```
            .data
str1:       .asciiz         "Enter the number:"
            .align 2        #move to a word boundary
res:        .space 4        # reserve space to store result

            .text
            .globl main
main:       li $v0, 4       # code to print string
            la $a0, str1
            syscall

            li $v0, 5       # code to read integer
            syscall

            move $t0, $v0   # move the value to $t0
            add $t1, $t0, $t0   # multiply by 2
            sw $t1, res($0)     # store result in memory

            li $v0, 1       # code to print integer
            move $a0, $t1   # move value to be printed to $a0
            syscall         # print to the screen

            li $v0, 10      # code for program end
            syscall
```

Practice

Add the elements of an array A[0..63]. Assume that the first element of the array is stored from address 200. Store the sum in address 800.

Read Appendix B of the textbook for a list of system calls used by the SPIM simulator.