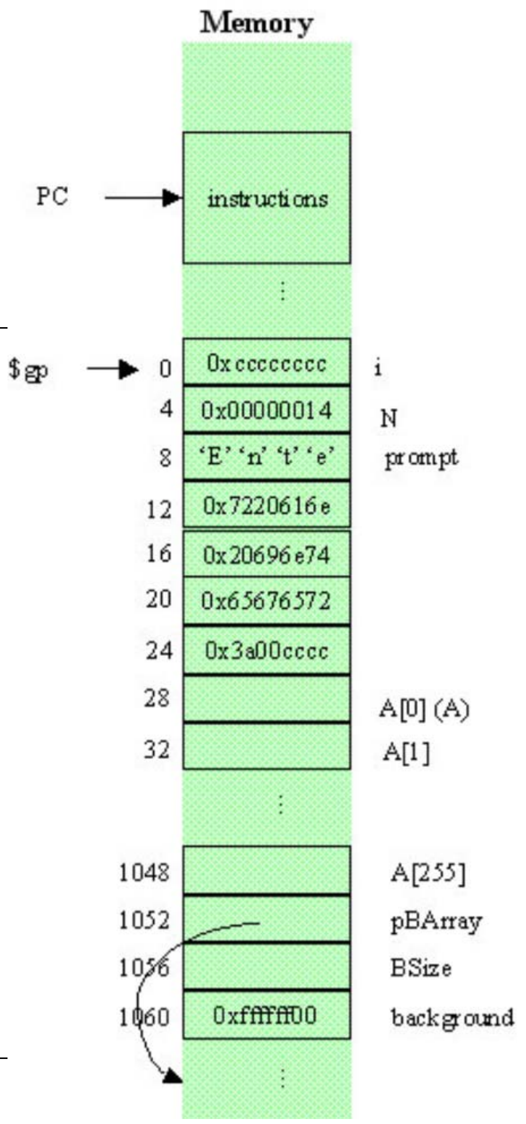
MIPS!

```
// none of these allocate any storage
#define MAX_SIZE  256
#define IF(a)     if (a) {
#define ENDIF     }
typedef struct {
    unsigned char red;      // 'unsigned char' is an unsigned, 8-bit int
    unsigned char green;
    unsigned char blue;
    unsigned char alpha;
} RGBa;

// these allocate storage
int     i;
int     N = 20;
char    prompt[] = "Enter an integer:";
int     A[MAX_SIZE];
int*    pBArray;
int     BSize;
RGBa    background = {0xff, 0xff, 0xff, 0x0};
```



C code:

```
i = N*N + 3*N
```

"Optimized":

```
lw      $t0, 4($gp)          # fetch N
add     $t1, $t0, $zero      # copy N to $t1
addi    $t1, $t1, 3          # N+3
mult    $t1, $t1, $t0        # N*(N+3)
sw      $t1, 0($gp)          # i = ...
```

```
A[i] = A[i/2] + 1;
A[i+1] = -1;
```

"Optimized":

```
# A[i] = A[i/2] + 1;
    lw      $t0, 0($gp)         # fetch i
    srl     $t1, $t0, 1         # i/2
    sll     $t1, $t1, 2         # turn i/2 into a byte offset (*4)
    add     $t1, $gp, $t1       # &A[i/2] - 28
    lw      $t1, 28($t1)        # fetch A[i/2]
    addi    $t1, $t1, 1         # A[i/2] + 1
    sll     $t2, $t0, 2         # turn i into a byte offset
    add     $t2, $t2, $gp       # &A[i] - 28
    sw      $t1, 28($t2)        # A[i] = ...
# A[i+1] = -1;
    addi    $t1, $zero, -1      # -1
    sw      $t1, 32($t2)        # A[i+1] = -1
```

```
if (i<N) {
    A[i] = 0;
}
```

MIPS assembler:

```
    lw      $t0, 0($gp)         # fetch i
    lw      $t1, 4($gp)         # fetch N
    slt     $t1, $t0, $t1       # set $t1 to 1 if $t0 < $t1, to 0 otherwise
    beq     $t1, $zero, skip    # branch if result of slt is 0 (i.e., !(i<N))
    sll     $t0, $t0, 2         # i as a byte offset
    add     $t0, $t0, $gp       # &A[i] - 28
    sw      $zero, 28($t0)      # A[i] = 0
skip:
```

C code:

```
background.blue = background.blue * 2;   // Note: overflow...
```

MIPS Assembler:

```
lw    $t0, 1060($gp)      # fetch background
andi  $t1, $t0, 0xff00    # isolate blue
sll   $t1, $t1, 2         # times 2
andi  $t1, $t1, 0xff00    # get rid of overflow
lui   $t2, 0xffff         # $t2 = 0xffff0000
ori   $t2, $t2, 0x00ff    # $t2 = 0xffff00ff
and   $t0, $t0, $t2       # get rid of old value of blue
or    $t0, $t0, $t1       # new value
sw    $t0, 1060($gp)      # background = ...
```

C code:

```c
// set N to the smallest odd no less than N
if ( N%2 == 0 ) N++;
```

MIPS Assembler:

```
lw    $t0, 4($gp)          # fetch N
ori   $t0, $t0, 1          # turn on low order bit
sw    $t0, 4($gp)          # store result in N
```

```
switch (i) {
    case 0:   A[0] = 0;
              break;
    case 1:
    case 2:   A[1] = 1;
              break;
    default:  A[0] = -1;
              break;
}
```

```
     lw    $t0, 0($gp)          # fetch i
     bltz  $t0, def             # i<0 -> default
     slti  $t1, $t0, 3          # i<3?
     beq   $t1, $zero, def      # no, -> default
     sll   $t0, $t0, 2          # turn i into a byte offset
     add   $t2, $t0, $gp
     lw    $t2, 1064($t2)       # fetch the branch table entry
     jr    $t2                  # go...
is0: sw    $zero, 28($gp)       # A[0] = 0
     j     done
is1:
is2: addi  $t0, $zero, 1        # = 1
     sw    $t0, 32($gp)         # A[1] = 1
     j     done
def: addi  $t0, $zero, -1       # = -1
     sw    $t0, 28($gp)         # A[0] = -1
     j     done
done:
```

C code:

```
for (i=0; i<N; i++) {
    A[i] = MAX_SIZE;
}
```

MIPS Assembler

```
    add     $t0, $gp, $zero         # &A[0] - 28
    lw      $t1, 4($gp)             # fetch N
    sll     $t1, $t1, 2             # N as byte offset
    add     $t1, $t1, $gp           # &A[N] - 28
    ori     $t2, $zero, 256         # MAX_SIZE
top:
    sltu    $t3, $t0, $t1           # have we reached the final address?
    beq     $t3, $zero, done        # yes, we're done
    sw      $t2, 28($t0)            # A[i] = 0
    addi    $t0, $t0, 4             # update $t0 to point to next element
    j       top                     # go to top of loop
done:
```