Compilation from C to MIPS

Original C code :

```
c  =  (a+b) - (i+j)
```

Compiled MIPS code :

```
add t0, a, b   #  temporary register t0 = a+b
add t1, i, j   #  temporary register t1 = i+j
sub c, t0, t1  #  c = t0 - t1
```

Registers in MIPS

There are exactly 32 registers in the MIPS architecture.
Each register is able to store exactly one « word ».
Here, by « word » one means a sequence of 32 bits.
Each MIPS register thus contains exactly 32 bits.

By convention, each register has its own status or function :

Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

For more information, see Figure 2.14 in Patterson & Hennessy

Compilation from C to MIPS

Original C code :

```
c  =  (a+b) - (i+j)
```

Compiled MIPS code :

```
add $t0, $s1, $s2  #  temp register $t0 = $s1+$s2
add $t1, $s3, $s4  #  temp register $t1 = $s2+$s3
sub $s0, $t0, $t1  #  $s0 = $t0 - $t1
```

Here, we make the important assumption that

- the registers $s1 and $s2 contain the values of a and b
- the registers $s3 and $s4 contain the values of i and j

Another example

Original C code :

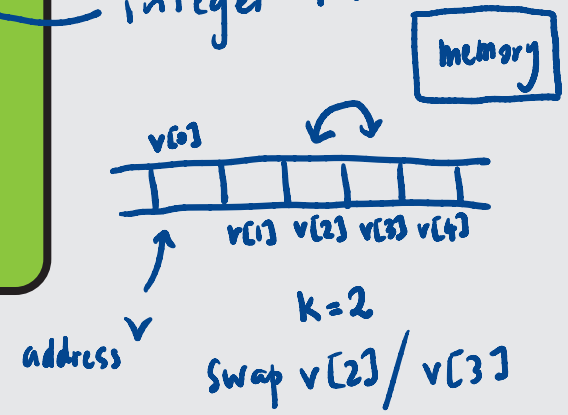
```
swap (int v[], int k)
{int temp;
   temp = v[k];
  v[k] = v[k+1];
 v[k+1] = temp;
}
```

array of integers v

integer k

$4 contains the address of v

$5 contains the value of k

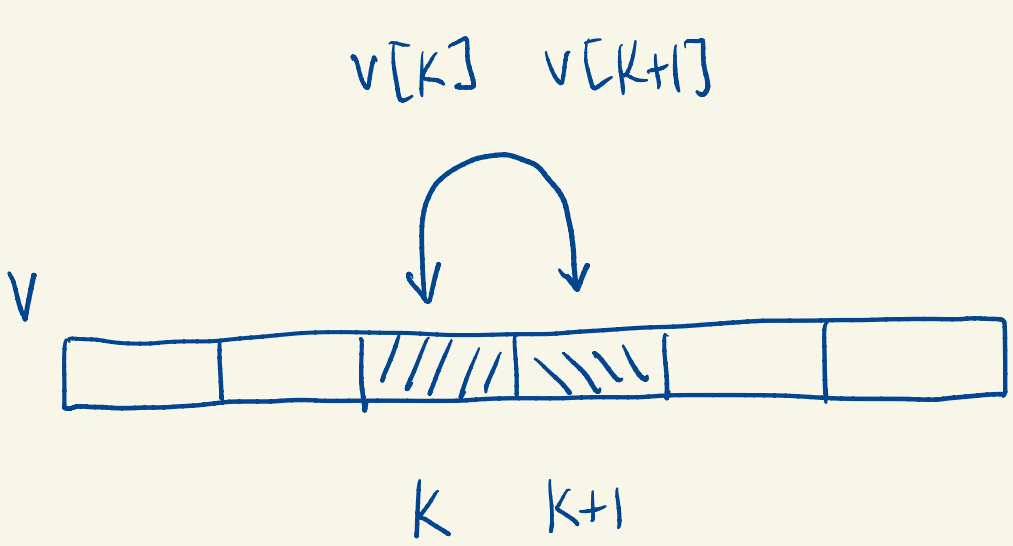


Compiled MIPS code : address of v[k] = "v + 4k"

```
multi $2, $5,4     # multiplies by 4 the value of register 5
                   # and writes the result in register 2
add   $2, $4,$2    # adds the value of register 4 to the value of register 2
                   # and writes the result in register 2
lw    $15, 0($2)   # loads the value of Mem[$2] in register 15
lw    $16, 4($2)   # loads the value of Mem[$2+4] in register 16
sw    $16, 0($2)   # stores the value of register 16 in Mem[$2]
sw    $15, 4($2)   # stores the value of register 15 in Mem[$2+4]
jr    $31          # jumps to the address loaded in register 31
```

Here, we suppose that :

- register 4 contains the address in memory of the array `v[]` of integers
- register 5 contains the value of the integer `k`.

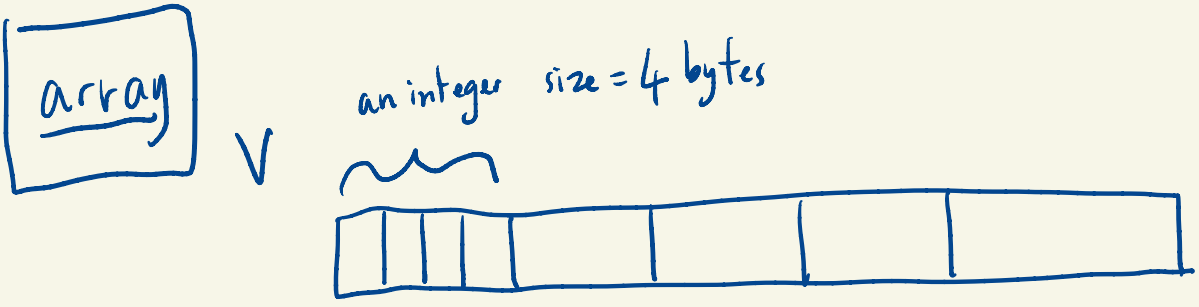



```
temp = v[k]
v[k] = v[k+1]
v[k+1] = temp
```

v[k]

array

an integer size = 4 bytes

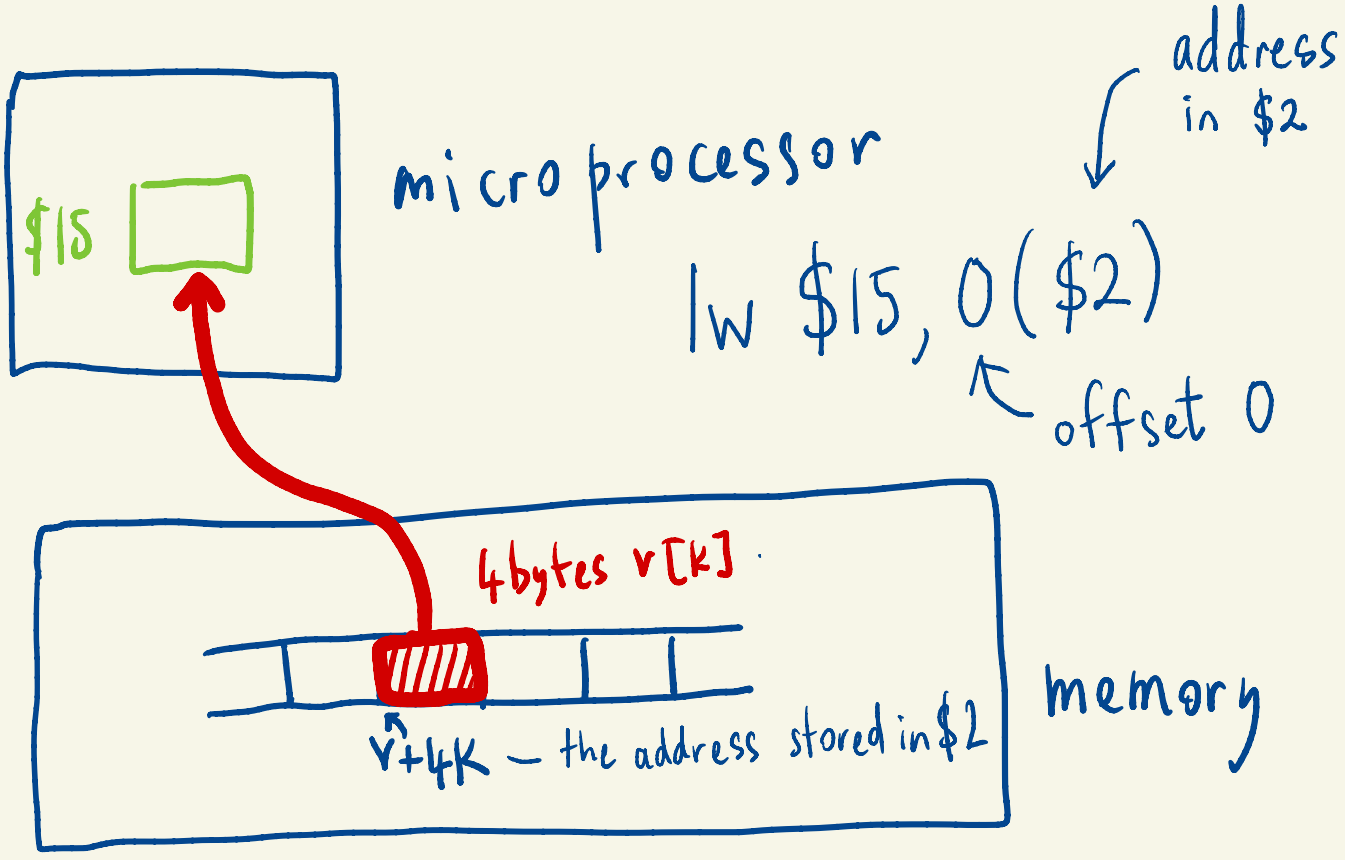


address of v = address of v[0]

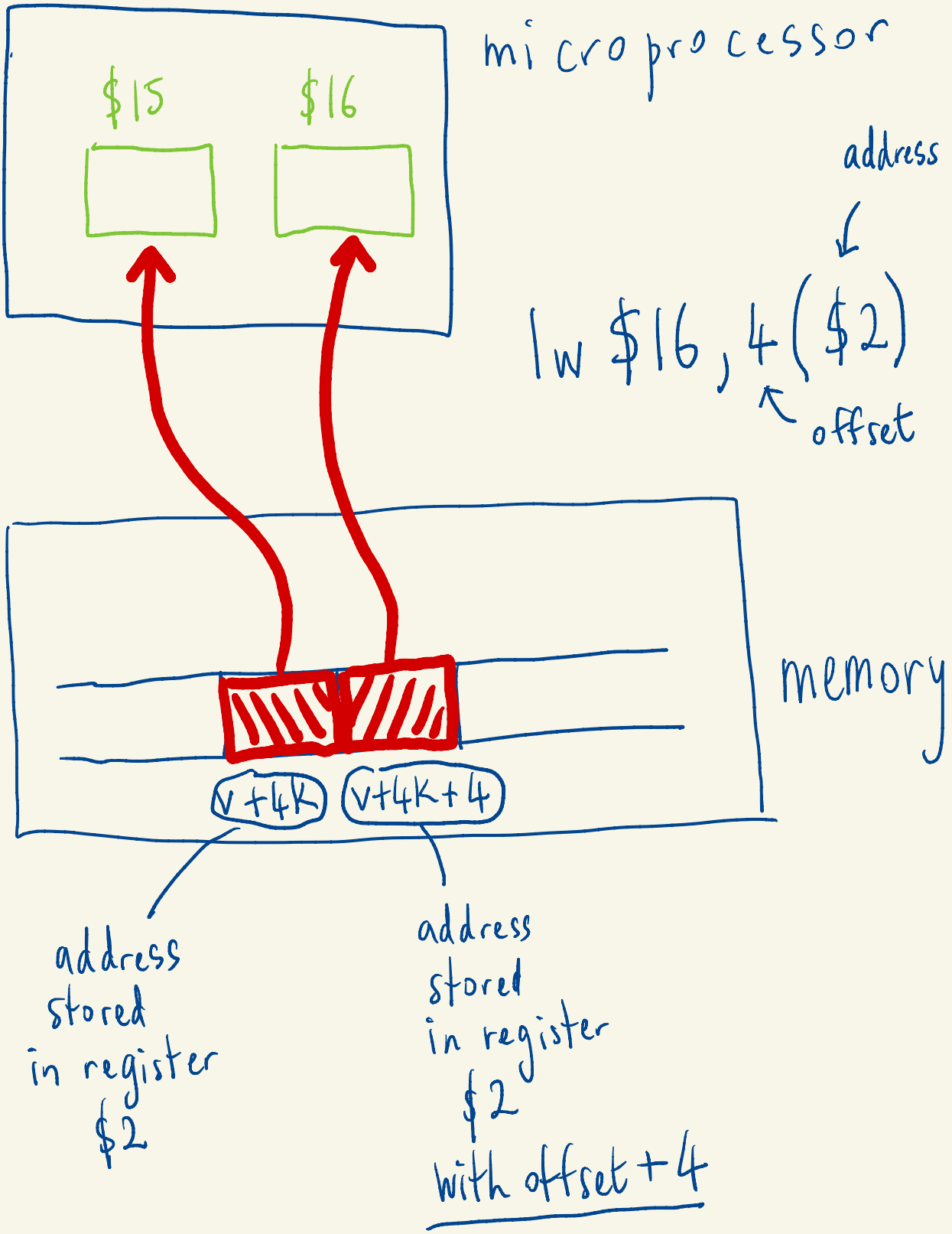
address of v[k] = address of v + 4 x k

① the first two instructions compute the value of v + 4×k and store it in register $2

② the two lw instructions load the word v[k] at address $2 in register $15



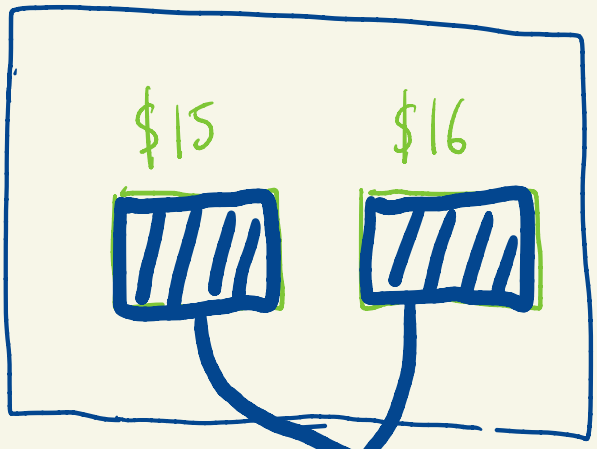
load word instructions



store word instructions

microprocessor

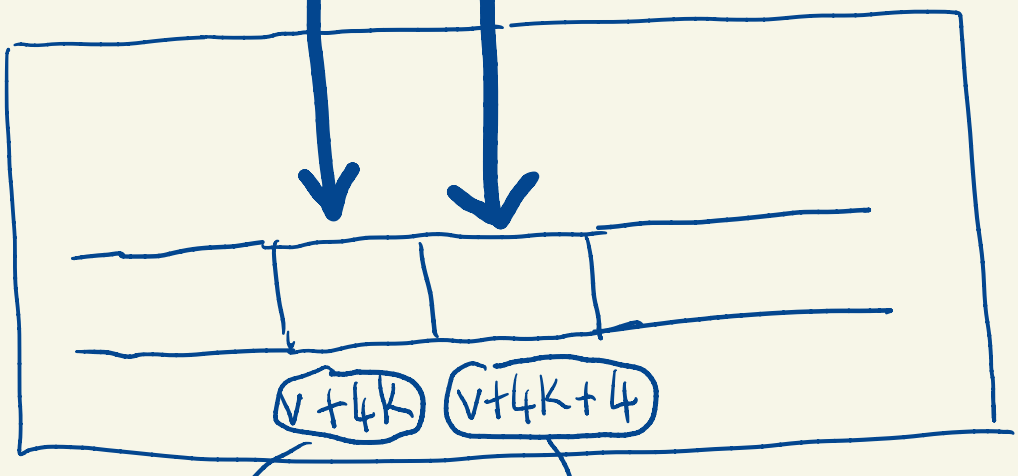
sw $16, 0($2)



address

sw $15, 4($2)

offset



address stored in register $2

address stored in register $2 with offset +4

memory

A troublesome fact

Almost every C compiler is bugged ...

To convince yourself, please have a look at :

Xuejun Yang , Yang Chen , Eric Eide , John Regehr
Finding and Understanding Bugs in C Compilers
2011 ACM SIGPLAN Conference on
Programming Language Design and Implementation (PLDI)

Compilers and operating systems you can trust

Certified Compilers

CompCert : a formally checked C compiler

Certified Operating Systems

seL4 project : a formally checked OS microkernel

For the most curious among you : Xavier Leroy
Proof assistants in computer science research
https://www.youtube.com/watch?v=PE4v6rVpX2g