Worksheet – 10 Solution

(From Lecture given on 02/13/2019)

- **Problem** : Use a 3-instruction window peephole matcher to generate optimized code for the following input IR fragment, by generating MIPS instructions in the subset shown below. You can perform whatever optimizations you think are reasonable as part of peephole matching, but state your assumptions.

- **Input IR:**

j = i+1

k = j+1

A[j] = k

l = A[k]

A[j] = l

Example MIPS instructions (subset)

1) Load Word instruction (read word at address ($t2+8) from memory and store in register $t1. **lw $t1,8($t2)**

2) Store Word instruction (write word from register $t1into memory address ($t2+4).
sw $t1,4($t2)

3) Add instruction (add $t1 and $t2, and store the result in $t3). **add $t3, $t1, $t2**

4) Load Immediate instruction (load constant, 99, into register $t4. **li $t4, 99**

5) Add immediate instruction (add $t1 and 1 and store the result in $t2) **addi $t2, $t1, 1**

Questions from students

- “I felt confused. Not sure what was the task for the worksheet.”
  - Sorry, we’ll try and make worksheets clearer in the future. Trade-off between make worksheet problems simple vs. interesting!
- “How do we know that it’s safe to remove/consolidate a **def** if we don’t look outside the fixed-size window?”
  - You don’t know in general. Often, a reaching definitions analysis is performed before peephole matching.

Peephole Matching (Recap)

Basic idea

- Compiler can discover local improvements locally
  - Look at a small set of adjacent operations
  - Move a "peephole" over code & search for improvement
- Classic example was store followed by load
  - Expressed using textbook's IR

Original code	Improved code
storeAI r_1 $\Rightarrow$ r_0,8	storeAI r_1 $\Rightarrow$ r_0,8
loadAI r_0,8 $\Rightarrow$ r_{15}	i2i r_1 $\Rightarrow$ r_{15}

The optimal MIPS code possible from the given IR code.

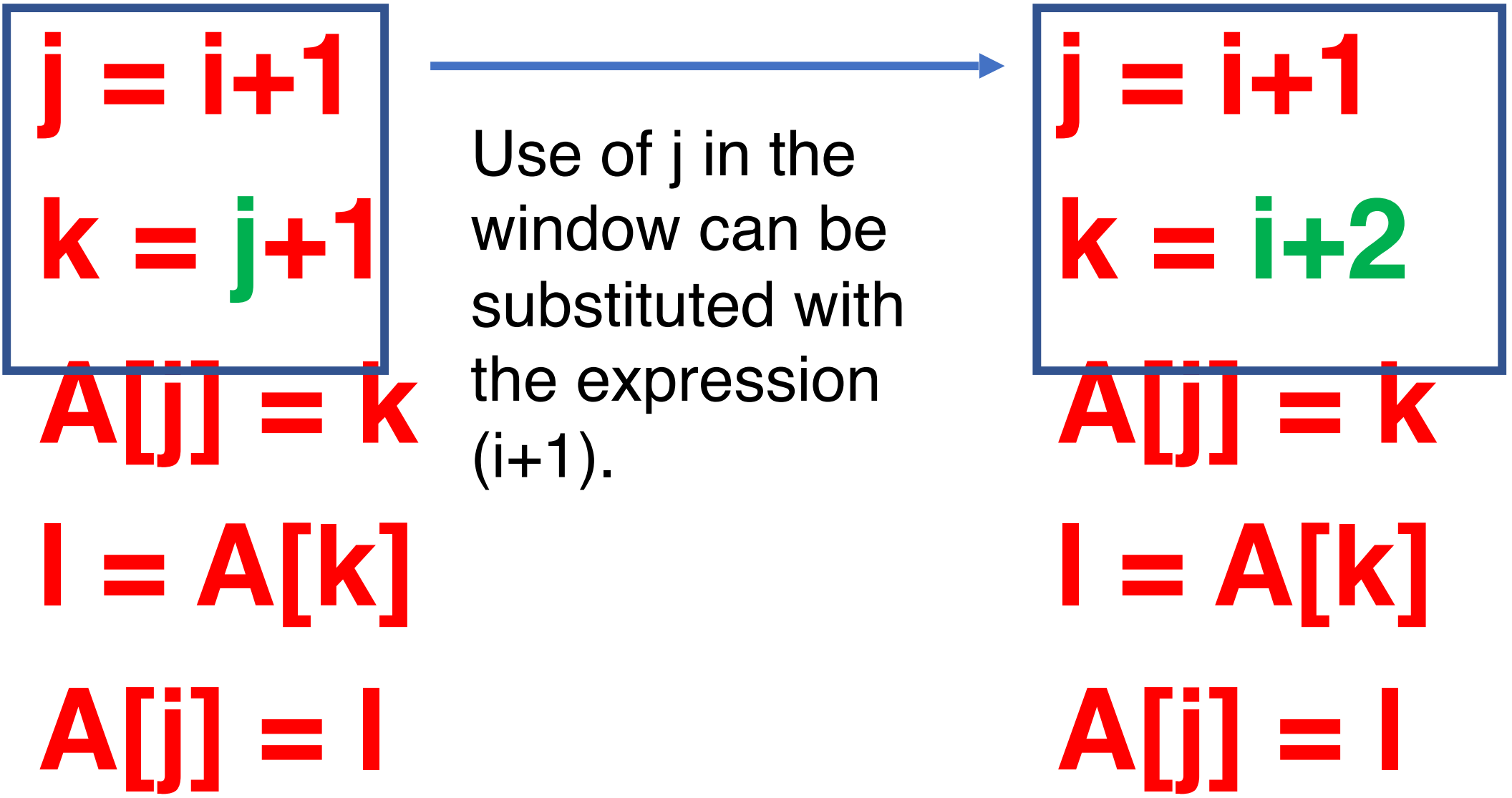
Input IR:
j = i+1
k = j+1
A[j] = k
l = A[k]
A[j] = l

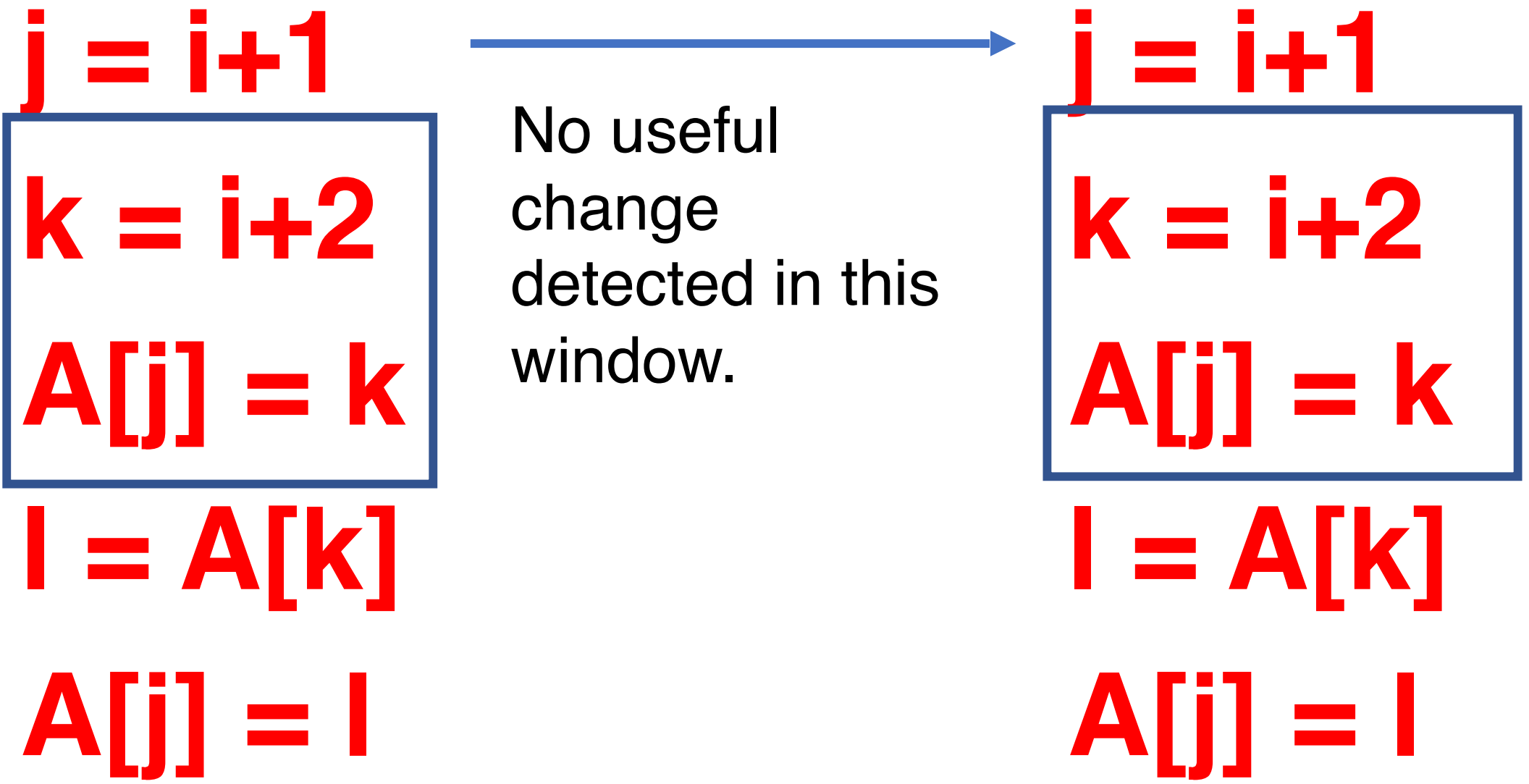
- Assumptions:
  $a0 = contains base address of array A
  $t0 = contains variable i

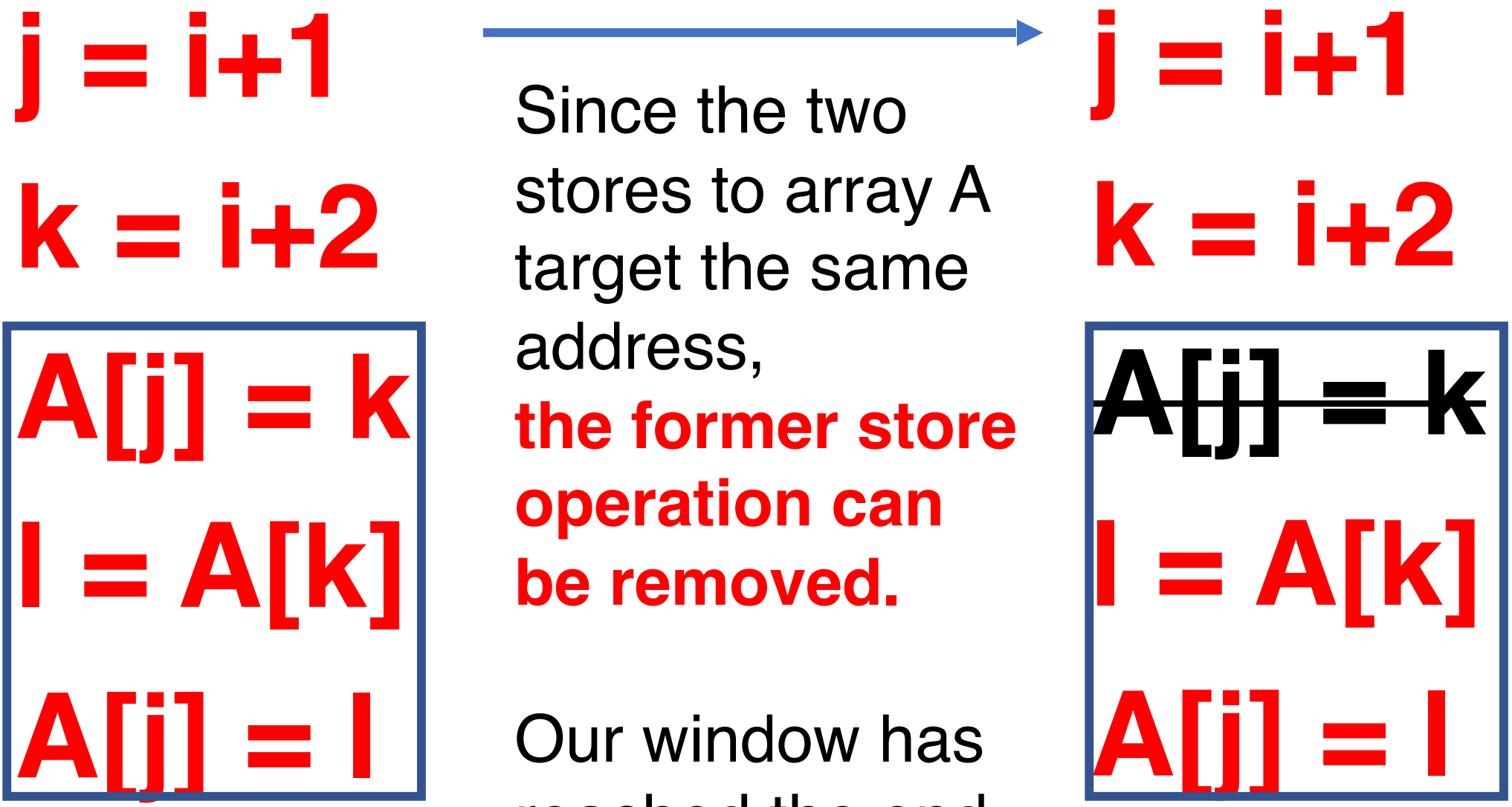
- ADDI $t1, $t0, $a0 // $t1 = address of A[i]
  LW $t2, 2($t1) // $t2 = A[i+2]
  SW $t2, 1($t1) // A[i+1] = $t2

Peephole Matching

- Using a 3-instruction window.
- Restrictions to the size of the observable region.


j = i+1
k = j+1
A[j] = k
l = A[k]
A[j] = l
Use of j in the window can be substituted with the expression (i+1).
j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l


j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l
No useful change detected in this window.
j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l


j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l
Since the two stores to array A target the same address,
the former store operation can be removed.
Our window has reached the end of the code.
End of peephole
j = i+1
k = i+2
A[j] = k
l = A[k]
A[j] = l

j = i+1

k = i+2

(REMOVED)

l = A[k]

A[j] = l

TRANSLATION TO MIPS ASSEMBLY (Example)

Assumptions:
$1 = variable i
$2 = variable j
$3 = variable k
$4 = variable l
$5 = address

1. Addi $2, $1, 1

2. Sll $2, $2, 2

3. Add $t0, $2, $5

4. lw $4, 0($t0)

5. Addi, $3, $1, 2

6. Sll $t0, $3, 2

7. Add $t0, $t0, $5

8. sw $4, 0($t0)