בס"ד

Computer architecture - final paper:

מגישים:

אייל מקדושי – 208879718

רותם קשאני – 209073352

מרצה:

ד"ר מרטין לנד

תאריך:
11/08/2024

Chapter 2 - The program in C language:

```c
int main()
{
    int arr[] = { 23, 67, -12, 89, 3, 150, -45 };
    int number = sizeof(arr) / sizeof(arr [0]);
    int index, max;

     // Initialize max with the first element of the
     array
    max = arr [0];

     // Iterate through the array starting from the
     second element
    for (index = 1; index < number; index++){
         // If current element is greater than max,
         update max
         if (arr[i] > max){
            max = arr[i];
         }
     }

    return 0;
}
```

תוכנית זו מיועדת למציאת הערך הגבוה ביותר במערך מספרים שלמים (int).
בתחילת התוכנית, מוגדר מערך של מספרים שלמים וכן משתנה שמייצג את מספר האיברים במערך.
משתנה נוסף מאותחל לערכו של האיבר הראשון במערך ומשמש לאחסון הערך הגבוה ביותר
שנמצא. התוכנית סורקת את המערך החל מהאיבר השני ומשווה כל איבר לערך המקסימלי הנוכחי.
אם נמצא איבר שערכו גבוה יותר, הערך המקסימלי מתעדכן בהתאם. בסיום הסריקה, מודפס הערך
הגבוה ביותר שנמצא במערך.

Chapter 3 – The output of the program in assembly language:

```
1           .file   1 "208879718_209073352.c"
2       .rdata
3       .align  2
4   $LC0:
5       .word   23
6       .word   67
7       .word   -12
8       .word   89
9       .word   3
10      .word   150
11      .word   -45
12      .text
13      .align  2
14      .globl  main
15      .ent main
16  main:
17      .frame  $fp,72,$31          # vars= 48, regs=
18  2/0, args= 16, extra= 0
19      .mask   0xc0000000,-4
20      .fmask  0x00000000,0
21      subu $sp,$sp,72
22      sw  $31,68($sp)
23      sw  $fp,64($sp)
24      move $fp,$sp
25      jal  __main
26      lw  $2,$LC0
27      sw  $2,16($fp)
28      lw  $2,$LC0+4
29      sw  $2,20($fp)
30      lw  $2,$LC0+8
31      sw  $2,24($fp)
32      lw  $2,$LC0+12
33      sw  $2,28($fp)
34      lw  $2,$LC0+16
35      sw  $2,32($fp)
36      lw  $2,$LC0+20
37      sw  $2,36($fp)
38      lw  $2,$LC0+24
```

```
39        sw   $2,40($fp)
40        li   $2,7          # 0x7
41        sw   $2,48($fp)
42        lw   $2,16($fp)
43        sw   $2,56($fp)
44        li   $2,1          # 0x1
45        sw   $2,52($fp)
46    $L2:
47        lw   $2,52($fp)
48        lw   $3,48($fp)
49        slt  $2,$2,$3
50        bne  $2,$0,$L5
51        j    $L3
52    $L5:
53        lw   $2,52($fp)
54        sll  $3,$2,2
55        addu $2,$fp,16
56        addu $2,$3,$2
57        lw   $3,0($2)
58        lw   $2,56($fp)
59        slt  $2,$2,$3
60        beq  $2,$0,$L4
61        lw   $2,52($fp)
62        sll  $3,$2,2
63        addu $2,$fp,16
64        addu $2,$3,$2
65        lw   $2,0($2)
66        sw   $2,56($fp)
67    $L4:
68        lw   $2,52($fp)
69        addu $2,$2,1
70        sw   $2,52($fp)
71        j    $L2
72    $L3:
73        move $2,$0
74        move $sp,$fp
75        lw   $31,68($sp)
76        lw   $fp,64($sp)
77        addu $sp,$sp,72
78        j    $31
79        .end main
```

Chapter 4 – Description of the program output:

```
1         .file 1  "208879718_209073352.c"
2         .rdata
3         .align 2
4   $LC0:
5         .word 23
6         .word 67
7         .word -12
8         .word 89
9         .word 3
10        .word 150
11        .word -45
12        .text
13        .align 2
14        .globl main
15        .ent  main
16  main:
17        .frame $fp,72,$31          # vars= 48, regs= 2/0,
18  args= 16, extra= 0
19        .mask 0xc0000000,-4
20        .fmask 0x00000000,0
21        subu  $sp,$sp,72
22        sw    $31,68($sp)
23        sw    $fp,64($sp)
24        move  $fp,$sp
25        jal   __main
26        lw    $2,$LC0
27        sw    $2,16($fp)
28        lw    $2,$LC0+4
29        sw    $2,20($fp)
30        lw    $2,$LC0+8
31        sw    $2,24($fp)
32        lw    $2,$LC0+12
33        sw    $2,28($fp)
34        lw    $2,$LC0+16
35        sw    $2,32($fp)
36        lw    $2,$LC0+20
37        sw    $2,36($fp)
38        lw    $2,$LC0+24
39        sw    $2,40($fp)
40        li    $2,7                 # 0x7
```

שם קובץ המקור בשפת c והוראת בקרת מקום

אתחול המערך

נקודת כניסה לmain

הוראת קריאה

יצירת מסגרת נתונים לתוכנית על המחסנית

טעינת ערכים ממערך לאוגר ואחסון ברצף בזיכרון

```
41          sw      $2,48($fp)
42          lw      $2,16($fp)
43          sw      $2,56($fp)
44          li      $2,1                    # 0x1
45          sw      $2,52($fp)
46
47  $L2:
48          lw      $2,52($fp)
49          lw      $3,48($fp)
50          slt     $2,$2,$3
51          bne     $2,$0,$L5
52          j       $L3
53  $L5:
54          lw      $2,52($fp)
55          sll     $3,$2,2
56          addu    $2,$fp,16
57          addu    $2,$3,$2
58          lw      $3,0($2)
59          lw      $2,56($fp)
60          slt     $2,$2,$3
61          beq     $2,$0,$L4
62          lw      $2,52($fp)
63          sll     $3,$2,2
64          addu    $2,$fp,16
65          addu    $2,$3,$2
66          lw      $2,0($2)
67          sw      $2,56($fp)
68  $L4:
69          lw      $2,52($fp)
70          addu    $2,$2,1
71          sw      $2,52($fp)
72          j       $L2
73  $L3:
74          move    $2,$0
75          move    $sp,$fp
76          lw      $31,68($sp)
77          lw      $fp,64($sp)
78          addu    $sp,$sp,72
79          j       $31
80          .end    main
```

אתחול והעתקה למחסנית של המשתנים num ו- max

הצהרה על לולאת for ובדיקת תנאי הסיום

גוף הלולאה:
טעינת רכיב המערך הנוכחי והשוואתו לערך המקסימלי שמאוחסן ב-max ועדכון הערך המקסימלי במידת הצורך.

חזרה על הלולאה עד שהתנאי מתקיים.

הגדלת מונה הלולאה (index) וקפיצה חזרה לתחילת הלולאה.

ניקוי המחסנית וסיום ה-main.

Chapter 5 – Translating the commands to MIPS32:

```
1         .file 1 "208879718_209073352.c"
2         .rdata
3         .align    2
4   $LC0:
5         .word 23
6         .word 67
7         .word -12
8         .word 89
9         .word 3
10        .word 150
11        .word -45
12        .text
13        .align    2
14        .globl    main
15        .ent  main
16
17  main:
18        .frame    $fp,72,$31      # vars= 48,
19  regs= 2/0, args= 16, extra= 0
20        .mask 0xc0000000,-4
21        .fmask    0x00000000,0
22        subu  $sp,$sp,72     # Allocate stack space
23        sw    $31,68($sp)    # Save return address
24        sw    $fp,64($sp)    # Save frame pointer
25        add $fp,$sp,$0      # move  $fp,$sp
26        jal   __main         # Call initialization
27  function
28  # Load array elements into stack frame
29        lw    $2,$LC0  # Load first element of array
30        sw    $2,16($fp) # Store it in stack frame
31        lw    $2,$LC0+4   # Load second element of
32  array
33        sw    $2,20($fp)   # Store it in stack frame
34        lw    $2,$LC0+8    # Load third element of
35  array
36        sw    $2,24($fp)  # Store it in stack frame
```

```
37        lw    $2,$LC0+12    # Load fourth element of
38  array
39        sw    $2,28($fp)      # Store it in stack
40  frame
41        lw    $2,$LC0+16      # Load fifth element of
42  array
43        sw    $2,32($fp)      # Store it in stack
44        lw    $2,$LC0+20      # Load sixth element of
45  array
46
47        sw    $2,36($fp)      # Store it in stack
48        lw    $2,$LC0+24      # Load seventh element
49  of array
50
51        sw    $2,40($fp)      # Store it in stack
52  # Set number of elements in array
53        li    $2,7            # Load immediate value
54  7 (number of elements)
55        sw    $2,48($fp)      # Store it in stack
56  # Initialize max with the first element of the
57  array
58        lw    $2,16($fp)      # Load first element of
59  array
60        sw    $2,56($fp)      # Store it as initial
61  max value
62  # Initialize index to 1
63        li    $2,1        # Load immediate value 1
64        sw    $2,52($fp)      # Store it as initial
65  index
66  $L2:
67  # Check if index >= number
68        lw    $2,52($fp)      # Load current index
69        lw    $3,48($fp)      # Load number of
70  elements
71        slt   $2,$2,$3      # Set $2 to 1 if index <
72  number, else 0
73        bne   $2,$0,$L5   # If index < number,
74  continue loop
75        j     $L3             # If index >= number,
76  exit loop
77  $L5:
78  # Compare arr[index] with max
79        lw    $2,52($fp)      # Load current index
```

```
80          sll   $3,$2,2         # Multiply index by 4
81    (size of int)
82          addu  $2,$fp,16       # Calculate base
83    address of array
84          addu  $2,$3,$2        # Calculate address of
85    arr[index]
86          lw    $3,0($2)        # Load arr[index]
87          lw    $2,56($fp)      # Load current max
88          slt   $2,$2,$3        # Set $2 to 1 if max <
89    arr[index], else 0
90
91          beq   $2,$0,$L4       # If max >= arr[index],
92    skip update
93    # Update max if arr[index] > max
94          lw    $2,52($fp)      # Load current index
95          sll   $3,$2,2         # Multiply index by 4
96    (size of int)
97          addu  $2,$fp,16       # Calculate base
98    address of array
99          addu  $2,$3,$2        # Calculate address of
100   arr[index]
101         lw    $2,0($2)        # Load arr[index]
102         sw    $2,56($fp)      # Update max value
103   $L4:
104   # Increment index
105         lw    $2,52($fp)      # Load current index
106         addu  $2,$2,1         # Increment index
107         sw    $2,52($fp)      # Store updated index
108         j     $L2             # Repeat loop
109   $L3:
110         add $2,$0,$0         # move $2, $0
111         add $sp,$fp,$0       # move $sp, $fp
112         lw    $31,68($sp)     # Restore return
113   address
114         lw    $fp,64($sp)     # Restore frame pointer
115         addu  $sp,$sp,72      # Deallocate stack
116   space
117         j     $31             # Return from function
118         .end  main
```

Chapter 6 - Analysis of the program in assembly language:

Instruction Type Distribution

Instruction Type	Instruction	Count	Description
Load	lw	14	Load word from memory to register
Store	sw	13	Store word from register to memory
Arithmetic	li	2	Load immediate value to register
Arithmetic	addu	5	Add unsigned
Arithmetic	subu	1	Subtract unsigned
Arithmetic	sll	1	Shift left logical
Logical	slt	2	Set on less than
Logical	beq	1	Branch on equal
Logical	bne	2	Branch on not equal
Jump	jal	1	Jump and link
Jump	j	2	Jump

Instruction Analysis

Here's a breakdown of the instruction types in the code:

- **Load Instructions**: 14 (`lw`)
- **Store Instructions**: 13 (`sw`)
- **Arithmetic Instructions**: 9 (`li`, `addu`, `subu`, `sll`)
- **Logical Instructions**: 6 (`slt`, `beq`, `bne`)
- **Jump Instructions**: 3 (`jal`, `j`)

Identification of Stalls

Stalls occur when an instruction cannot proceed to the next stage of the pipeline because it is waiting for a previous instruction to complete. The main types of hazards that can cause stalls are:

- **Data Hazards**: Occur when instructions that exhibit data dependency modify data in different stages of the pipeline.
- **Control Hazards**: Occur due to branch instructions.

Potential Data Hazards:

- When `lw` is followed by an instruction that uses the loaded value.
- When `addu`, `subu`, or `slt` is followed by an instruction that uses the result.

Potential Control Hazards:

- Branch instructions like `beq`, `bne`, and `j`.

Number of Clock Cycles Required to Run the Program

To estimate the number of clock cycles, we will assume a simple model where each instruction takes one cycle, and we add extra cycles for stalls due to hazards.

1. **Initialization and Setup**:
    - 22 instructions (20 memory accesses, 1 call to `__main`, 1 return address save) = 22 cycles
    - Assume 3 cycles for potential stalls
2. **Main Loop Execution**:
    - Number of iterations = 6 (since the array length is 7, and index starts at 1)
    - Instructions per iteration = 13
    - Cycles per iteration = 13 + potential stalls

Assuming 1 stall per iteration due to data hazards:

- Total cycles for loop = 6 * (13 + 1) = 84 cycles

3. **Exit Sequence**:
    - 5 instructions = 5 cycles
4. **Total Estimated Cycles**:
    - Setup + Loop + Exit = 22 + 3 (initial stalls) + 84 + 5 = 114 cycles

Chapter 7 – Optimization:

```
1         .file 1 "208879718_209073352.c"
2         .rdata
3         .align      2
4   $LC0:
5         .word 23
6         .word 67
7         .word -12
8         .word 89
9         .word 3
10        .word 150
11        .word -45
12        .text
13        .align      2
14        .globl      main
15        .ent  main
16
17  main:
18        .frame      $fp,72,$31       # vars= 48,
19  regs= 2/0, args= 16, extra= 0
20        .mask 0xc0000000,-4
21        .fmask      0x00000000,0
22        subu  $sp,$sp,72      # Allocate stack space
23        sw    $31,68($sp)     # Save return address
24        sw    $fp,64($sp)     # Save frame pointer
25        add $fp,$sp,$0       # move  $fp,$sp
26        jal   __main          # Call initialization
27  function
28  # Load array elements into stack frame
29      lw $t0, $LC0 # Load first element of array
30      into $t0
31      lw $t1, $LC0 + 4 # Load second element of
32      array into $t1
33      lw $t2, $LC0 + 8 # Load third element of
34      array into $t2
35      lw $t3, $LC0 + 12 # Load fourth element of
36      array into $t3
37      lw $t4, $LC0 + 16 # Load fifth element of
38      array into $t4
```

```
39      lw $t5, $LC0 + 20 # Load sixth element of
40      array into $t5
41      lw $t6, $LC0 + 24 # Load seventh element of
42      array into $t6
43
44      # Set number of elements in array
45      li $t7, 7 # $t7 = 7 (number of elements)
46
47      # Initialize max with the first element of
48      the array
49      move $s0, $t0 # max = $t0 (first element)
50      # Initialize index to 1
51      li $s1, 1 # index = 1
52
53  $L2:
54      # Check if index >= number
55      bge $s1, $t7, $L3 # If index >= number, exit
56      loop
57
58      # Load current array element based on index
59      sll $t8, $s1, 2 # Multiply index by 4 (size
60      of int)
61      add $t9, $t8, $fp # Base address of array in
62      frame pointer
63      lw $t8, 16($t9) # Load arr[index]
64
65      # Compare arr[index]
66      with max ble $s0, $t8, $L4 # If max >=
67      arr[index], skip update
68
69      # Update max if arr[index] > max
70      move $s0, $t8 # max = arr[index]
71
72  $L4:
73      # Increment index
74      addi $s1, $s1, 1 # index++
75
76      # Repeat loop
77      j $L2
78
79  $L3:
80      # Exit program
81      move $v0, $0 # Set return value to 0
82      move $sp, $fp # Restore stack pointer
```

```
83        lw $31, 68($sp) # Restore return address
84        lw $fp, 64($sp) # Restore frame pointer
85        addu $sp, $sp, 72 # Deallocate stack space
86        j $31 # Return from function
87        .end main
88
```

Chapter 8 – Repeating Chapter 6 for the code after the optimization:

Identification of Stalls

Stalls occur when an instruction cannot proceed to the next stage of the pipeline because it is waiting for a previous instruction to complete. The main types of hazards that can cause stalls are:

- **Data Hazards**: Occur when instructions that exhibit data dependency modify data in different stages of the pipeline.
- **Control Hazards**: Occur due to branch instructions.

In the optimized code, we have:

Potential Data Hazards:

- The initial loads and moves should be carefully sequenced to avoid stalls.

Potential Control Hazards:

- Branch instructions like `bge`, `ble`, and `j`.

Number of Clock Cycles Required to Run the Program

To estimate the number of clock cycles, we assume a simplified model where each instruction takes one cycle, and we add extra cycles for stalls due to hazards.

1. **Initialization and Setup**:
    - 22 instructions (setup + load array elements) = 22 cycles
    - Assume 1 cycle for potential stalls during initialization
2. **Main Loop Execution**:
    - Number of iterations = 6 (since the array length is 7, and index starts at 1)
    - Instructions per iteration = 6 (without significant stalls due to optimization)
    - Total cycles for loop = 6 * 6 = 36 cycles
3. **Exit Sequence**:
    - 5 instructions = 5 cycles
4. **Total Estimated Cycles**:
    - Setup + Loop + Exit = 22 + 1 (initial stall) + 36 + 5 = 64 cycles

Instruction Type Distribution

Instruction Type	Instruction	Count	Description
Load	lw	8	Load word from memory to register
Store	sw	2	Store word from register to memory
Arithmetic	li	2	Load immediate value to register
Arithmetic	add	2	Add
Arithmetic	addi	1	Add immediate
Arithmetic	subu	1	Subtract unsigned
Arithmetic	sll	1	Shift left logical
Logical	move	5	Move value between registers
Logical	bge	1	Branch on greater than or equal
Logical	ble	1	Branch on less than or equal
Jump	jal	1	Jump and link
Jump	j	2	Jump

Chapter 9 – Bibliography:

מצגות הקורס ומאמרי הסיכום:
https://cs.hac.ac.il/staff/martin/Architecture/