

Computer Architecture and Organization HW#4

Due by 10/17(Fri.) through LMS

[1] Write MIPS assembly code that does the following:

- a) In the .data section, three integer values are stored as num1, num2, and num3.
- b) Load these values into registers \$s0, \$s1, and \$s2.
- c) Find the median value (the middle value when the three numbers are ordered).
- d) Store the median into register \$s5.
- e) Print the median value to the console.

Your program should run at QtSpim correctly.

[2] Use AI (ChatGpt, Gemini, etc.) to solve the above problem. There may be errors. If there are any, correct them. Then compare with the program with yours and write down your thoughts in several sentences **in Korean**. Also, write down the number of dynamic instruction counts of two programs when the input data is as follows.

```
num1: .word 25
num2: .word 40
num3: .word 15
```

* Note:

- 1) For the following programs, dynamic instruction count is 20 not 8. (since loop body – from 'slt' instruction to 'j' instruction – is executed 3 times)

```
.text
main:      addi $s0, $zero, 0
           addi $t0, $zero, 0
           addi $t1, $zero, 3
loop:slt $t8, $t0, $t1
           beq $t8, $zero, exit
           addi $t0, $t0, 1
           add $s0, $s0, $t0
           j loop
exit:
.end
```

- 2) Compress (zip) the source program for prob1 into a zip file and the pdf file for prob2 together, and submit it through LMS.

[1]

(a)

.data

num1: .word 25

num2: .word 40

num3: .word 15

(b)

.text

-global main

main:

lw \$s0, num1

lw \$s1, num2

lw \$s2, num3

(c)

ble \$s0, \$s1, step2

move \$t0, \$s0

move \$s0, \$s1

move \$s1, \$t0

step2:

ble \$s1, \$s2, step3

move \$t0, \$s1

move \$s1, \$s2

move \$s2, \$t0

step3:

ble \$s0, \$s1, result

move \$t0, \$s0

move \$s0, \$s1

move \$s1, \$t0

(d)

result:

move \$s5, \$s1

(e)

li \$v0, 1

move \$a0, \$s5

syscall

li \$v0, 10

syscall

① s_0, s_1 비교

$s_0 > s_1 \rightarrow s_0 \rightarrow s_0$ 값과 s_1 값 바꿈

$s_1 \geq s_0 \rightarrow s_1 \rightarrow \text{②}$

② s_1, s_2 비교

$s_1 > s_2 \rightarrow s_1 \rightarrow s_1$ 값과 s_2 값 바꿈

$s_2 \geq s_1 \rightarrow s_2 \rightarrow \text{③}$

③ s_0, s_1 값 비교

$s_0 > s_1 \rightarrow s_0 \rightarrow s_0$ 값과 s_1 값 바꿈

$s_1 \geq s_0 \rightarrow s_1 \rightarrow \text{끝! } s_0 \leq s_1 \leq s_2$

s_1 이 중앙값을 가짐

```
.data
num1:  .word 25
num2:  .word 40
num3:  .word 15
```

```
.text
.globl main
main:
```

```
    lw $s0, num1      —————> |
    lw $s1, num2      —————> |
    lw $s2, num3      —————> |
```

```
    ble $s0, $s1, step2 —————> |
    move $t0, $s0
    move $s0, $s1
    move $s1, $t0
```

```
step2:
    ble $s1, $s2, step3 —————> |
    move $t0, $s1      —————> |
    move $s1, $s2      —————> |
    move $s2, $t0      —————> |
```

```
step3:
    ble $s0, $s1, result —————> |
    move $t0, $s0      —————> |
    move $s0, $s1      —————> |
    move $s1, $t0      —————> |
```

```
result:
    move $s5, $s1      —————> |
```

```
    li $v0, 1          —————> |
    move $a0, $s5       —————> |
    syscall             —————> |
```

```
    li $v0, 10         —————> |
    syscall             —————> |
```

총 18개

AI(Gemini)가 생성한 MIPS 중앙값 찾기 프로그램

```
.data
num1: .word 25
num2: .word 40
num3: .word 15

.text
.globl main

main:
    # 값 불러오기
    lw $s0, num1
    lw $s1, num2
    lw $s2, num3

    # 중앙값 계산 (Swap 방식)
    # 1. ($s0 > $s1) 이면 두 값을 교환
    bgt $s0, $s1, swap_s0_s1
after_swap1:

    # 2. ($s1 > $s2) 이면 두 값을 교환
    bgt $s1, $s2, swap_s1_s2
after_swap2:

    # 3. 마지막으로 ($s0 > $s1) 을 다시 확인하여 교환
    bgt $s0, $s1, swap_s0_s1_again
after_swap3:

    # 이 시점에서 $s1은 중앙값이 됨
    move $s5, $s1

    # 결과 출력.
    li $v0, 1
    move $a0, $s5.
    syscall

    # 프로그램 종료
    li $v0, 10
    syscall

# --- 값 교환(Swap)을 위한 서브 루틴 ---
swap_s0_s1:
    move $t0, $s0
    move $s0, $s1
    move $s1, $t0
    j after_swap1

swap_s1_s2:
    move $t0, $s1
    move $s1, $s2
    move $s2, $t0
    j after_swap2

swap_s0_s1_again:
    move $t0, $s0
    move $s0, $s1
    move $s1, $t0
    j after_swap3
```

총 20개

[2]

두 프로그램 비교 및 소감

제가 작성한 코드는 `while` 명령어를 사용해 조건이 맞으면 `gap` 요한 조적을 건너뛰는 방식으로 구현했습니다.
반면 AI (Gemini) 는 `goto` 를 사용해 조건이 맞으면 `gap` 요한을 하는 별도의 핸들러 (handler) 로 감쌌다가 돌아오는 구조를 가집니다.

이러한 구조적 차이에서 AI 코드는 핸들러에서 원래 코드로 돌아가기 위해 `jump` 명령어를 추가로 실행해야 하므로 2개 더 많은 명령어를 실행하였습니다.

이를 통해 사람이 직접 눈의 흐름을 추적하듯 하면 AI 보다 조금 더 효율적인 코드를 만들 수 있다는 점을 알게 되었습니다.
AI 가 여전히 문제 해결에서 합리적 능력을 보이지만, 만들어진 결과에 대해 비평적인 사고가 가능한 것은 결국 사람을 대 할 땐
깨닫게 되는 계기가 되었습니다.

내 코드 : 16개

AI 코드 : 20개