IT5002 Computer Systems and Applications

Assignment 1

ANSWER BOOK

Name: Wang Lurong	**Student ID: A0189107Y**
Name: Li Jiaru	**Student ID: A0332008U**

Question 1: (5 MARKS)

lui $1, 4096 [msg] # Load upper immediate, loads 4096, which in hexadecimal is 0x1000, into the upper 16 bits of register $1.
ori $4, $1, 256 [msg] # Or immediate, that performs or comparison between register $1 and 256.

Since or operation accepts returns 1 during bit wise comparison if any of the bit is 1, the upper 16 bits of register $1 0x1000 0000 and the lower 16 bits of 0x0000 0100 are stored in register $4, storing the message's address.

Question 2: (3 MARKS)

The label 'main:' is the designated entry point of any MIPS program. When execution starts, PC is set to the address of main. Because the order of executing the instructions is determined by the program counter, main gets executed first.

Question 3: (3 MARKS)

jr stands for jump register. $ra is updated to PC+4 when program counter points to 'jal [some function call address]'. Jr $ra sets PC to the value stored in $ra (original PC+4 prior to function call).

Question 4: (2 MARKS)

The output on the console is "This is function 1This is function 2".

We do not see "This is main".

Question 5: (5 MARKS)

In the 1st step, fun1 is called by main, $ra was initially updated to 0x00400054 (PC+4) with regards to main. Next, within fun1, another ‘jal 0x0040003c [fun2]’ is executed. This cause $ra to be updated to 0x00400038, overwriting the return address in the 1st step (0x00400054). When the program exits fun2 and performs a function call return with jr $31, it successfully returns to 0x00400038, which contains a fun1 instruction. However, within fun1, even when jr $31 is executed, the program cannot exit fun1 since its initial return address has been over-written. The failed function call return to main causes jal.asm to not work as expected.

Question 6: (4 MARKS)

The callee does not know which registers the caller is using and vice versa. The same registers could be used by both caller and callee. During a function call, the callee may potentially overwrite the registers used by the caller, causing registers to be overwritten, as shown in the jal.asm function’s $ra. Storing the ‘old registers’ in stack allows subsequent retrieval of the information.

Question 7: (3 MARKS)

$sp does not update automatically like program counter. Because $sp is copied to $fp, 0($fp) and 0($sp) both refer to the start of the stack, containing the old $fp. Saving $ra at 0($fp) and 0($sp) would overwrite the old $fp, losing the information in the old $fp. To save $ra we must use 4($fp).

Question 8: (2 MARKS)

subi $sp, $fp, 0

Question 9: (3 MARKS)

```
fun1:   li $v0, 4
        la $a0, str1
        syscall
        sw $fp, 0($sp)      #access to stack
        addi $fp, $sp, 0    #update $fp register to current $sp
        sw $ra, 4($fp)      #store old $ra (return address for main caller calling function 1) to 4($fp)
        addi $sp, $sp, 8    #increment of $sp by 2 words to prevent overwritting
        jal fun2            #original function call for function 2
        lw $ra, 4($fp)      #load the old $ra stored in 4($fp) back to $ra for return address to main
        addi $sp, $fp, 0    #copy $fp to $sp
        lw $fp, 0($fp)      #Restore $fp
        jr $ra
```

The yellow highlighted lines are the modified code. Previously, the old $ra, the return directory from function 1 back to caller (main) gets overwritten after fun1 calls fun2. The use of stack for storage of the old $ra allows the old $ra address to be retrieved in function 1, allowing successful return to the main caller (0x400070).

Code for Data Segment (1 MARK)

```
.data 0x10000110
X: .word 20 110 17 5 3 12 18 8 99 25
```

Code for printint (2 MARKS)

```
.text
printint:
  li $v0, 1
  syscall
  li $v0, 11
  li $a0, 32
  syscall
  jr $ra
```

Code for printarray (5 MARKS)

```
printarray:
  sw $fp, 0($sp)
  addi $fp, $sp, 0
  sw $ra, 4($fp)
  addi $sp, $sp, 8
  la $t0, X
  li $t2, 10
  li $t1, 0
loop:
  beq $t1, $t2, done
```

```
   sll $t3, $t1, 2
   addu $t4, $t0, $t3
   lw $a0, 0($t4)
   jal printint
   addi $t1, $t1, 1
   j loop
done:
   lw $ra, 4($fp)
   addi $sp, $fp, 0
   lw $fp, 0($fp)
   jr $ra
```

Code for main (2 MARKS)

```
main:
   jal printarray
   li $v0, 10
   syscall
```

TOTAL: _______________________ / 40