COMP2611: Computer Organization

MIPS functions

Overview

- You will learn the following in this tutorial:
  - handling MIPS immediate numbers.
  - using MIPS functions including the recursive ones.

MIPS functions

MIPS immediate numbers

- exercises

MIPS simple functions

- exercises

Recursive functions

- exercises

Exercises

Exercises

Question 1: Write the shortest sequence of MIPS instructions for the following C++ code, assuming each variable is stored in a different register you named.

b = a + 0x37cf0010;

Exercises

Question 2: Write the shortest sequence of MIPS instructions for the following C++ code, assuming each variable is stored in a different register you named.

b = a + 0x37cff346;

How a 16-bit immediate is extended

- ❑ Arithmetic instructions (e.g. addi, addiu): always sign extend (deem zero-extend as sign-extend for unsigned number)
- ❑ Load/store instructions (e.g. lb, lbu): always sign extend
- ❑ Logical instructions (e.g. ori, andi): always zero extend
- ❑ Set instructions (e.g. slti, sltiu): sign extend
- ❑ shift instructions (e.g. srl): always sign extend

MIPS functions

MIPS immediate numbers

- exercises

MIPS simple functions

- exercises

Recursive functions

- exercises

Exercises

MIPS Simple Functions

- The Caller
  - Puts function arguments in $a0 - $a3 before invoking jal
  - Pushes argument registers ($a0 - $a3), temporary registers ($t0 - $t9) onto stack if needed after the call
  - jal ProcedureAddress
    - The jal saves the return address which is (PC + 4) in $ra
    - Then, jump to address specified by ProcedureAddress
  - Picks up the return values from $v0 - $v1
- The Callee
  - Pushes preserved registers ($s0 - $s8), argument registers ($a0 - $a1) onto stack if they are changed within callee
  - Performs the procedure
  - Pops the preserved registers if any from stack
  - Puts up to two return results in $v0 - $v1 if there is any
  - Invokes jr $ra to go back to the Caller

Exercises

Question 1: Translate the following C++ function into a MIPS function, using the registers $a0 and $a1 for its parameters and the register $v0 for its return value.

```
int equal(int p1, int p2) {
   if (p1 == p2)
         return 1;
   return 0;
}
```

Exercises

Question 2: Write the MIPS instructions that make the following call to the C++ function in the previous exercise, assuming variable b is stored in the register $s0.

```
int b = equal(3, 4);
```

Exercises

Question 3: Does the following MIPS function correspond to the C++ function, assuming the registers $a0 and $a1 store the parameters and the register $v0 stores the return value? If not, indicate the problem(s) and describe how to fix them.

```
int add(int p1, int p2) {
   cout << "Called add()";
   return p1 + p2;
}
```

```
.data
msg: .asciiz "Called add()"
.text
…
add:   li $v0, 4
        #to print a string
       la $a0, msg
       syscall
       add $v0, $a0, $a1
       jr $ra
```

MIPS functions

MIPS immediate numbers

- exercises

MIPS simple functions

- exercises

Recursive functions

- exercises

Exercises

MIPS recursive functions

- ❑ The Caller to a nested function call performs the same steps as to a simple function call. E.g. jal nestedProcedureAddress
- ❑ The nested callee (each callee becomes a caller for its next callee)

Within each callee

- Pushes preserved registers ($s0 - $s8), argument registers ($a0 - $a1) onto stack if changed within callee
- Pushes temporary registers ($t0 - $t9) onto stack if changed within callee and needed after the call
- Pushes $ra for its caller into stack
- Performs the recursive procedure by jal nestedProcedureAddress

After returning to each caller

- Pops the preserved registers, argument registers, and temporary registers from stack if there is any
- Pops its $ra
- Puts return results in $v0 - $v1
- Invokes jr $ra to go back to the caller

Exercises

Question 1: Translate the following C++ recursive function into a MIPS recursive function.

```
int multiply(int p1, int p2) {
   if (p2 == 0)
         return 0;
   return p1 + multiply(p1, p2 – 1);
}
```

MIPS functions

- MIPS immediate numbers
  - exercises
- MIPS simple functions
  - exercises
- Recursive functions
  - exercises
- Exercises

Exercises

Question 1: Write the shortest sequence of MIPS instructions for the following C++ code, assuming each variable is stored in a different register you named.

```
b = a + 60000;
```

Exercises

Question 2:

```
void saveElement(int a[], int x) {
        a[x] = x;
  }
```

Translate the above C++ function into a MIPS function. $s0 is the only extra register that can be used inside your function. The stack can also be used. Your function must work for the following MIPS sequence of calling to it.

```
la $a0, list1           #assuming an array list1 is already defined
addi $a1, $s0, 0     #$a1 stores x which is $s0
jal saveElememt
addi $a1, $s0, 1      #$a1 stores x which is $s0+1
jal saveElememt
```

Exercises

Question 3: Does the following MIPS recursive function correspond to the C++ recursive function? If not, indicate the problem(s) and describe how to fix them.

```
int sum(int x) {
   if (x == 0)
         return 0;
   return x + sum(x – 1);
}
```

```
 sum: bne $a0, $zero, recur
         li $v0, 0
         jr $ra
recur: addi $a0, $a0, -1
         jal sum
         add $v0, $v0, $a0
         jr $ra
```

Conclusion

- ❑ You have learnt:
  - ❑ handling MIPS immediate numbers.
  - ❑ using MIPS functions including the recursive ones.