COMP2611: Computer Organization

MIPS Procedure

MIPS procedures

32-bit Immediate Operands

- exercises

Simple MIPS procedures

- exercises

Exercises

A Recap on 32-bit Immediate operands

- Constants are frequently short and fit into 16-bit field
- But sometimes they are bigger than 16 bits, e.g. 32-bit constant

- `lui` ("load upper immediate")
  - e.g. `lui reg, constant`
  - set the upper 16 bits of register `reg` to the 16-bit value specified in `constant`
  - **Set the lower 16 bits of register `reg` to zeros**
- addi will sign-extend the 16-immediate operand especially for negative numbers
  - For unsigned immediate
- Advisable to use ori
  - For both signed and unsigned immediate

Question 1: Write down the shortest sequence of MIPS instructions for the following C++ codes, assuming each variable is stored in a different register (you name it).

```
b = a + 0x37cf0010;
```

Question 2: Write down the shortest sequence of MIPS instructions for the following C++ codes, assuming each variable is stored in a different register (you name it).

```
b = a + 0x37cf£346;
```

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 procedures

32-bit Immediate Operands

- exercises

Simple MIPS procedures

- exercises

Exercises

A Recap on MIPS Procedures

- The Caller
  - Puts function arguments in `$a0 - $a3` before invoking jal
  - Pushes arguments registers (`$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

A Recap on Stack: supporting procedures in MIPS

- Since **procedures** are like small programs themselves, they may **need to use the registers**, and they **may also call other procedures** (nested calls)
  - If we don't save some of the values stored in the registers, they will be wiped each time we call a new procedure
- In MIPS, we need to save the registers by ourselves
- The perfect place for this is called a **stack**
  - a memory accessible only from the top (Last In First Out, LIFO)
  - placing things on the stack is called **push**
  - removing them is called **pop**
  - **push** and **pop** are simply **storing** and **loading** words to and from a specific location in the memory pointed to by **the stack pointer $sp** which always points to top of the stack

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

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

Exercises

Question 4: The following C++ function takes as inputs the base address of an int array `A` and returns the minimum value in `A`. Using the registers `$a0` and `$a1` as arguments to the function, `$v0` as returned value, `$s0` as base address of `A` and `$s1` as the size of `A`, translate the C++ function into a MIPS function

```
int minArray(int A[], int arraySize) {
  int min = A[0];
  int i = 1;
  while(i < arraySize) {
     if(min < A[i])
        min = A[i];
     i++;
  }
  return min;
}
```

MIPS procedures

MIPS immediate numbers

- exercises

Simple MIPS procedures

- exercises

Exercises

Exercises

Question 1: Write down the shortest sequence of MIPS instructions for the following C++ codes, assuming each variable is stored in a different register (you name it).

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

Solution:

```
addi $s0, $s1, 60000 #$s0 and $s1 store b and a
                      respectively
```

Exercises

Question 2:

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

Translate the above C++ function into a MIPS function, assuming the registers `$a0` and `$a1` store the parameters. `$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 calls to it.

```
la $a0, list1      #assuming an array list1 is already defined
addi $a1, $s0, 0
jal saveElement
addi $a1, $s0, 1
jal saveElement
```

Solution to Question 2:

```
saveElement: addi $sp, $sp, -4
             sw $s0, 0($sp)
             sll $s0, $a1, 2
             add $s0, $s0, $a0
             sw $a1, 0($s0)
             lw $s0, 0($sp)
             addi $sp, $sp, 4
             jr $ra
```