5.21 Functions

Functions using registers only

In C, a ***function*** is a named group of statements that performs a specific operation. A ***function call*** involves passing argum function's parameters, executing the function's statements, and returning the function's return value. For a function with a f like 1 or 2, registers can be used to pass arguments to the function and return a value from the function. This material assu for the first argument, $t1 for the second argument, if needed, and $t2 for the return value, if needed.

A function definition can be converted to an assembly subroutine that assumes $t0 and $t1 hold the arguments, and write to $t2 before returning. A function call is converted to assembly following a simple pattern that assigns $t0 and $t1 with th jumps to the subroutine, and reads the return value from $t2.

PARTICIPATION ACTIVITY 5.21.1: Functions in assembly.

Start 2x speed

w	z
5	20
30	30

C

```
int Max(int x, int y) {
  if (x > y) {
     return x;
  }
  else {
     return y;
  }
}
...
z = Max(w, 20);
```

Assembly

```
Max:
   slt $t3, $t0, $t1
   bne $t3, $zero, MaxIsY
   add $t2, $zero, $t0
   j MaxEnd
   MaxIsY: add $t2, $zero, $t1
   MaxEnd: jr $ra

...

addi $t6, $zero, 5004  # w
lw $t0, 0($t6)
addi $t1, $zero, 20
jal Max
addi $t6, $zero, 5020  # z
```

Register file

$t0		*x*
$t1		*y*
$t2		*return value*

```
sw $t2, 0($t6)
```

MIPS argument and return value registers

MIPS, having more registers than MIPSzy, reserves registers $a0 to $a3 for a subroutine's arguments and $v0 and $v1 for the return value.

PARTICIPATION ACTIVITY 5.21.2: Functions using registers.

Implement the C by completing the assembly. Assume $t0 is used for the first parameter, $t1 for the second parameter, and $t2 for the return value.

```
int CalcFunc(int aVal, int bVal) {
   return aVal * 4 + bVal;
}
```

```
(a) _____:
   addi $t3, $zero, 4
   mul $t2, (b) ___, $t3
   add (c) ___, $t2, $t1
   CalcFuncEnd: (d) _____
```

1) (a)

Check Show answer

2) (b)

Check Show answer

3) (c)

Check Show answer

4) (d)

Check Show answer

Functions using the stack

The program stack can be used to pass arguments to and return values from a function. In converting the function definitic lw instruction is used to load a function argument, and a sw instruction is used to store to the return value. The location of and return value depends on the number of function parameters. Ex: For a function with 2 parameters and a return value, 0 address for the return value, 4($sp) is for the second parameter, and 8($sp) for the first parameter.

Figure 5.21.1: Function definition converted to assembly using program stack.

```
int Max(int x, int y) {
  if (x > y) {
     return x;
  }
  else {
     return y;
  }
}
```

```
Max:
   lw $t0, 8($sp)         # Load x from stack
   lw $t1, 4($sp)         # Load y from stack
   slt $t3, $t0, $t1
   bne $t3, $zero, MaxIsY
   sw $t0, 0($sp)         # Store return value to stack
   j MaxEnd
   MaxIsY: sw $t1, 0($sp) # Store return value to stack
   MaxEnd: jr $ra
```

Table 5.21.1: Example stack addresses for various functions.

Function	Stack frame
`void OutLen(int feet, int inches)`	`0($sp)`: inches `4($sp)`: feet
`int CompAvg(int a, int b, int c)`	`0($sp)`: return value `4($sp)`: c `8($sp)`: b `12($sp)`: a
`int CalcSum(int w, int x, int y, int z)`	`0($sp)`: return value `4($sp)`: z `8($sp)`: y `12($sp)`: x `16($sp)`: w

PARTICIPATION ACTIVITY 5.21.3: Function using the program stack.

Implement the C by completing the assembly.

```
int ConvFtToIn(int feet, int inches) {
      return feet * 12 + inches;
}
```

```
ConvFtToIn:
   lw $t0, (a) _____
   lw  (b) _____
   addi $t2, $zero, 12
   mul $t3, $t0, $t2
   add $t2, $t3, $t1
   (c) _____
   ConvFtToInEnd: jr $ra
```

1) (a)

Check Show answer

2) (b)

Check Show answer

3) (c)

Check Show answer

A function call using the program stack is converted to instructions that push each argument to the stack, reserving a stac return value, jumping to the subroutine, and popping the return value and arguments afterwards.

Figure 5.21.2: Function call in assembly using program stack.

Assume w is held in $t0 and 20 is held in $t1.

```
z = Max(w, 20);
```

```
addi $sp, $sp, -4
sw $t0, 0($sp)        # Push w to stack
addi $sp, $sp, -4
sw $t1, 0($sp)        # Push 20 to stack
addi $sp, $sp, -4     # Make space for return value
jal Max               # Jump to Max subroutine
lw $t2, 0($sp)        # Pop return value to $t2
addi $sp, $sp, 4
addi $sp, $sp, 4      # Pop argument from stack
addi $sp, $sp, 4      # Pop argument from stack
```

PARTICIPATION ACTIVITY 5.21.4: Function call using the program stack.

Detect the error in the assembly for each function call. Assume $t0 holds xVal and $t1 holds yVal. Assume zVal is located at memory address 5004.

1) C

```
PrintVals(xVal, yVal);
```

Assembly

```
addi $sp, $sp, -4
sw $t0, 0($sp)
addi $sp, $sp, -4
sw $t1, 0($sp)
addi $sp, $sp, -4
jal PrintVals
addi $sp, $sp, 4
addi $sp, $sp, 4
```

2) C

```
zVal = ConvKmToMiles(yVal);
```

Assembly

```
addi $sp, $sp, -4
  sw $t1, 0($sp)
addi $sp, $sp, -4
  sw $t0, 0($sp)
```

```
addi $sp, $sp, -4
jal ConvKmToMiles
lw $t4, 0($sp)
addi $sp, $sp, 4
addi $sp, $sp, 4
addi $t6, $zero, 5004
sw $t4, 0($t6)
```

3) C

```
zVal = CompMin(100, xVal);
```

Assembly

```
addi $t3, $zero, 100
addi $sp, $sp, -4
sw $t3, 0($sp)
addi $sp, $sp, -4
sw $t0, 0($sp)
addi $sp, $sp, -4
jal CompMin
lw $t4, 4($sp)
addi $sp, $sp, 4
addi $sp, $sp, 4
addi $sp, $sp, 4
addi $t6, $zero, 5004
sw $t4, 0($t6)
```

Functions with local variables

In C, A ***local variable*** declared in a function has a scope limited to the function, meaning the variable only exists when the fu executes. A local variable can be stored in the program stack. In assembly, each variable declaration is implemented by pus value to the stack. Before the function returns, the variable is popped from the stack.

PARTICIPATION ACTIVITY 5.21.5: Function with local variable.

Start 2x speed

C

```
int CalcVal(int x, int y) {
  int w = 0;

  // Function statements
  return w;
}
```

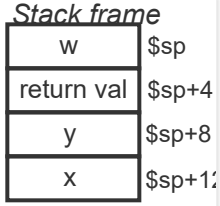
Assembly

```
CalcVal:
   addi $sp, $sp, -4
   sw $zero, 0($sp)    # Push w to stack

   # Instructions for function statements
CalcValEnd:
   lw $t3, 0($sp)      # Load w
   sw $t3, 4($sp)      # Store return value

   addi $sp, $sp, 4    # Pop w from stack
   jr $ra
```

Stack frame



PARTICIPATION ACTIVITY 5.21.6: Function with local variables.

Consider the following C function.

```
int CalcBonus(int totSales, int salesGoal) {
   int maxBonus = 100;
   int extraSales = 0;
   int bonus = 0;

   extraSales = totSales - salesGoal;
   if (extraSales > 0) {
       bonus = extraSales * 10;
   }

   if (bonus > maxBonus) {
      bonus = maxBonus;
   }

   return bonus;
}
```

1) If all local variables are allocated to the stack, how many elements will the stack frame contain?

Check Show answer

2) Complete the assembly for the maxBonus declaration?

```
addi $t0, $zero,

addi $sp, $sp, -4

sw $t0, 0($sp)
```

Check Show answer

3) Complete the assembly for the extraSales declaration?

```
addi $sp, $sp, -4

sw
```

Check Show answer

4) Complete the assembly for the statement, assuming $t1 holds totSales and $t2 holds salesGoal.

```
extraSales = totSales -
salesGoal;

sub $t4, $t1, $t2

sw $t4,  ($sp)
```

Check Show answer

CHALLENGE ACTIVITY 5.21.1: Functions in assembly.

Start

1

2

Convert the C to assembly. Assume first parameter is in $t0, second parameter is in $t1, and return value is in $t2.

```
int CalcFunc(int aVal, int bVal) {
    return (aVal - 4) + bVal;
}
```

```
CalcFunc: addi $t3 , $zero , 4
          add  $t2 , $t2   , $t2
          add  $t2 , $t2   , $t2
          add  $t2 , $t2   , $t2
```

Registers

$zero	0
$t0	2
$t1	3
$t2	0
$t3	0
$ra	160

1 2

Check Next

Provide feedback on this section