Overview

- Recursive Functions
- Computing the Stack Size for function calls

Recursive Functions

- A recursive function is both a caller and a callee of itself.
- Need to check both its source caller (that is not itself) and itself for register conflicts.
- Can be hard to compute the maximum stack space needed for recursive function calls.
  - Need to know how many times the function is nested (the depth of the calls).

An Example of Recursive Function Calls

```c
int sum(int n);
int main(void)
{
    int n=100;
    sum(n);
    return 0;
}
void sum(int n)
{
    if (n<=0) return 0;
    else return (n+ sum(n-1));
}
```
Call Trees

• A call tree is a weighted directed tree $G = (V, E, W)$ where
  - $V = \{v_1, v_2, \ldots, v_n\}$ is a set of nodes each of which denotes an execution of a function;
  - $E = \{v_i \rightarrow v_j: v_i \text{ calls } v_j\}$ is a set of directed edges each of which denotes the caller-callee relationship, and
  - $W = \{w_i (i=1, 2, \ldots, n): w_i \text{ is the frame size of } v_i\}$ is a set of stack frame sizes.

• The maximum size of stack space needed for the function calls can be derived from the call tree.

An Example of Call Trees

```c
int main(void)
{
    \ldots
    func1();
    \ldots
}

void func2()
{
    \ldots
    func4();
    func2();
    \ldots
}

void func1()
{
    \ldots
    func3();
    \ldots
}
```

An Example of Call Trees (Cont.)

```
  func1() 20  func2() 60
    \uparrow
  func3() 80  func4() 10
    \uparrow
  func5() 30
```

The number in red beside a function is its frame size in bytes.

Computing the Maximum Stack Size for Function Calls

Step 1: Draw the call tree.

Step 2: Find the longest weighted path in the call tree.

The total weight of the longest weighted path is the maximum stack size needed for the function calls.
**An Example**

The longest path is \texttt{main} \rightarrow \texttt{func1} \rightarrow \texttt{func3} with the total weight of 110. So the maximum stack space needed for this program is 110 bytes.

---

**Fibonacci Rabbits**

- Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits. Suppose that our rabbits never die and that the female always produces one new pair (one male, one female) every month from the second month on.

- How many pairs will there be in one year?

Fibonacci’s Puzzle

Italian, mathematician Leonardo of Pisa (also known as Fibonacci) 1202.

---

**Fibonacci Rabbits (Cont.)**

- The number of pairs of rabbits in the field at the start of each month is 1, 1, 2, 3, 5, 8, 13, 21, 34, ... .
- In general, the number of pairs of rabbits in the field at the start of month \( n \), denoted by \( F(n) \), is recursively defined as follows.

\[
F(n) = F(n-1) + F(n-2)
\]

Where \( F(0) = F(1) = 1 \).

\( F(n) \) \((n=1, 2, 3, ...)\) are called Fibonacci numbers.

---

**C Solution of Fibonacci Numbers**

```c
int month=4;
int main(void)
{
    fib(month);
}
int fib(int n)
{
    if(n == 0) return 1;
    if(n == 1) return 1;
    return (fib(n - 1) + fib(n - 2));
}
```
AVR Assembler Solution

<table>
<thead>
<tr>
<th>Return address</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>r16</td>
<td>X-2</td>
</tr>
<tr>
<td>r17</td>
<td>X-3</td>
</tr>
<tr>
<td>r28</td>
<td>X-4</td>
</tr>
<tr>
<td>r29</td>
<td>X-5</td>
</tr>
<tr>
<td>n</td>
<td>X-6</td>
</tr>
</tbody>
</table>

Frame structure for fib()

r16, r17, r28 and r29 are conflict registers.

An integer is 2 bytes long in WINAVR

Assembly Code for main()

```
.cseg
month: .dw 4
main:
    ; Prologue
    ldi r28, low(RAMEND)
    ldi r29, high(RAMEND)
    out SPH, r29
    out SPL, r28
    ; Initialise the stack pointer SP to point to
    ; the highest SRAM address
    ; End of prologue
    ldi r30, low(month<<1)
    ldi r31, high(month<<1)
    lpm r24, z+
    lpm r25, z
    rcall fib
    ; Call fib(4)
    ; Epilogue: no return

loopforever:
    rjmp loopforever
```

Assembly Code for fib()

```
fib: push r16               ; Prologue
    push r17               ; Save r16 and r17 on the stack
    push r28               ; Save Y on the stack
    push r29               ; Save Y on the stack
    in r28, SPL
    in r29, SPH
    sbiw r29:r28, 2       ; Let Y point to the bottom of the stack frame
    out SPH, r29          ; Update SP so that it points to
    out SPL, r28          ; the new stack top
    std Y+1, r24          ; Pass the actual parameter to the formal parameter
    std Y+2, r25
    cpi r24, 0            ; Compare n with 0
    clr r0
    cpc r25, r0
    brne L3               ; If n!=0, go to L3
    ldi r24, 1            ; n==1
    ldi r25, 0            ; Return 1
    rjmp L2               ; Jump to the epilogue

L3:  cpi r24, 1            ; Compare n with 1
    clr r0
    cpc r25, r0
    brne L4               ; If n!=1 go to L4
    ldi r24, 1            ; n==1
    ldi r25, 0            ; Return 1
    rjmp L2               ; Jump to the epilogue

L4:  ldd r24, Y+1          ; n>2
    ldd r25, Y+2          ; Load the actual parameter n
    sbiw r24, 1          ; Pass n-1 to the callee
    rcall fib            ; call fib(n-1)
    mov r16, r24         ; Store the return value in r17:r16
    mov r17, r25
    ldd r24, Y+1          ; Load the actual parameter n
    ldd r25, Y+2
    sbiw r24, 2          ; Pass n-2 to the callee
    rcall fib            ; call fib(n-2)
    add r24, r16         ; r25:r25=fib(n-1)+fib(n-2)
    adc r25, r17
```
**Assembly Code for fib() (Cont.)**

L2:

```
adw r29:r28, 2       ; Epilogue
out SPH, r29         ; Deallocate the stack frame for fib()
out SPL, r28         ; Restore SP
pop r29              ; Restore Y
pop r28              ; Restore r17 and r16
pop r17
pop r16
ret
```

**Computing the Maximum Stack Size (Cont.)**

Step 1: Find the longest weighted path.

The longest weighted path is `main() → fib(4) → fib(3) → fib(2) → fib(1)` with the total weight of 32. So a stack space of 32 bytes is needed for this program.