Tutorial 5.1

Subroutines

Lei Wang

lei.wang2@ucalgary.ca

Subroutines

- Open (inline)
  - Code is inserted inline wherever the subroutine is invoked
    - Usually using a macro preprocessor
  - Arguments are passed in/out using registers
  - Efficient, since overhead of branching and return is avoided
  - Suitable only for fairly short routines
- Closed
  - Machine code for the routine appears only once in RAM
  - When invoked, control “jumps” to the first instruction of the routine
  - When finished, control returns to the next instruction in the calling code
  - Arguments are placed in registers or on the stack
  - Slower than open routines, because of call/return overhead

Open (inline) subroutines

Eg: cube function

```
define(comment)

comment(cube(1 = input register, 2 = output register))
define(cube, `mul   $2, $1, $1
              mul   $2, $1, $2')

          .global main
main:     stp       x29, x30, [sp, -16]!
          ...
          mov       x19, 8
          cube(x19, x20)
          ...
```

m4 expands this to:

```
          .global main
main:     stp       x29, x30, [sp, -16]!
          ...
          mov       x19, 8
          mul       x20, x19, x19
          mul       x20, x19, x20
          ...
```

Closed Subroutines

- General form:

```
label: stp    x29, x30, [sp, alloc]!
       move   x29, sp
       ...// body
       ldp    x29, x30, [sp], -alloc
       ret
```

- *label*: name of the subroutine
- *alloc*: number of bytes (negated) to allocate for the subroutine's stack frame
  - SP must be quad-word aligned
  - minimum of 16 bytes

Subroutine Linkage

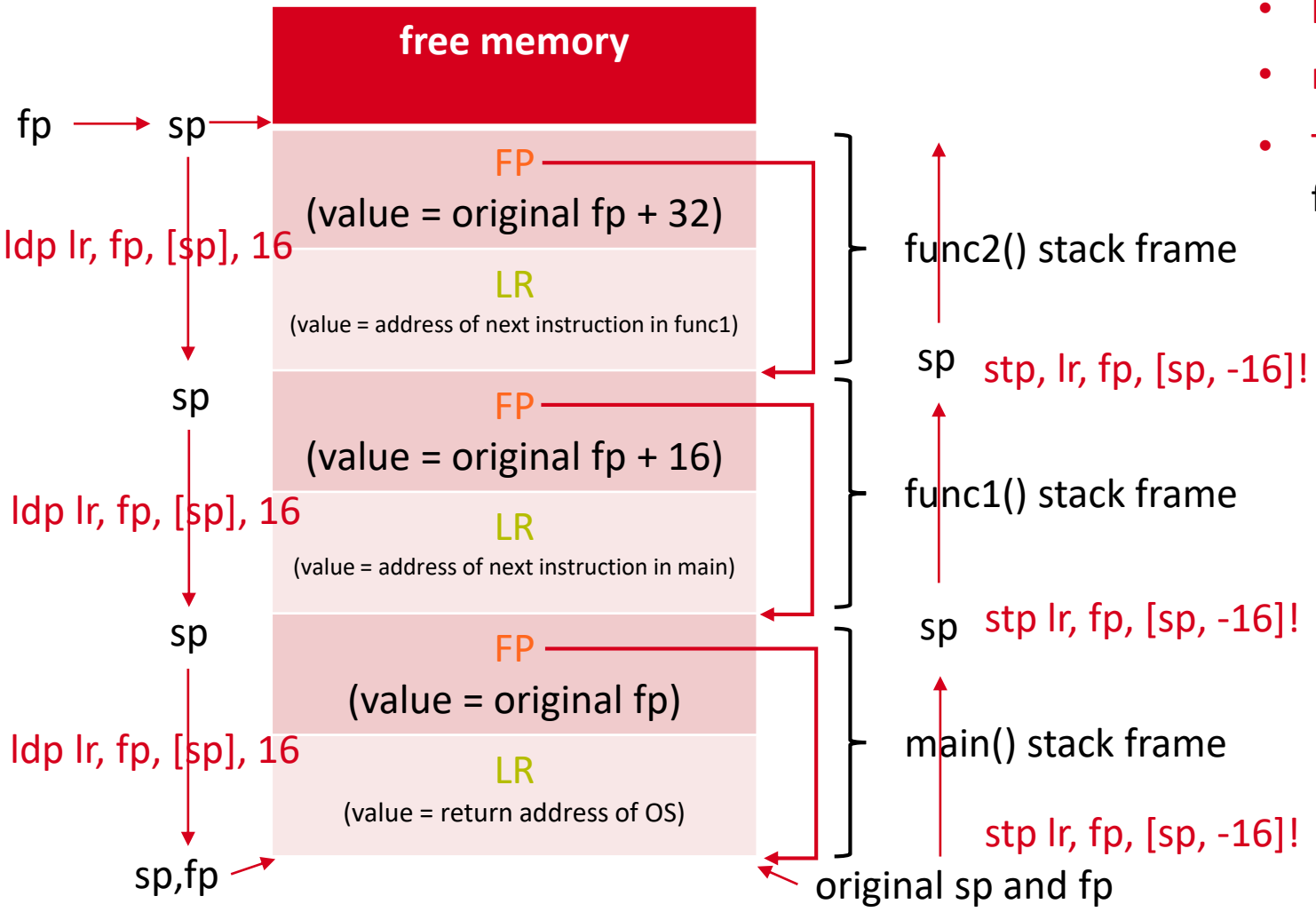
- branch and link instruction to invoke a subroutine
  - Form: *bl subroutine_label*
  - Stores the return address into link register: x30
- use *ret* instruction to return from a subroutine back to the calling code
- Eg: C

```c
int main()
{
        ...
        func1();
        ...
}
void func1()
{
        ...
        func2();
        ...
}
void func2()
{
        ...
}
```

assembly

```
main:   stp     x29, x30, [sp, -16]!
        mov     x29, sp
        ...
        bl      func1
        ...
        ldp     x29, x30, [sp], 16
        ret
func1:  stp     x29, x30, [sp, -16]!
        mov     x29, sp
        ...
        bl      func2
        ...
        ldp     x29, x30, [sp], 16
        ret
func2:  stp     x29, x30, [sp, -16]!
        mov     x29, sp
        ...
        ldp     x29, x30, [sp], 16
        ret
```

the stack while func2 is executing



- bl puts the address of the next instruction to the $lr
- ret jumps to $lr
- The FP and the stored FP values in the frame records form a linked list

UNIVERSITY OF CALGARY

Saving and Restoring Registers

- A called function must save/restore the state of the used registers if it uses any of the registers *x19 - x28*,
- the called function can also use *x9-x15* that are not saved/restored by the called function

```
        x19_size = 8
        alloc = -(16 + x29_size) & -16
        dealloc = -alloc
        x19_save = 16
func:   stp     x29, x30, [sp, alloc]!
        mov     x29, sp
        str     x19, [x29, x19_save]        //save x19
        ...
        mov     x19, 13                     //use x19
        ...
        ldr     x19, [x29, x19_save]        //restore 19
        ldp     x29, x30, [sp], dealloc
        ret
```

Arguments to Subroutines

- using registers x0 – x7(long ints), w0 – w7(ints, short ints, and chars)
- Eg

C

```
void sum(int a, int b)
{
        register int i;
        i = a + b;
        ...
}
int main()
{
        sum(3, 4)
        ...
}
```

assembly

```
define(i_r, w9)
sum:    stp             x29, x30, [sp, -16]!
        mov             x29, sp
        add             i_r, w0, w1
        ...
        ldp             x29, x30, [sp], 16
        ret
main:   stp             x29, x30, [sp, -16]!
        mov             x29, sp
        mov             w0, 3
        mov             w1, 4
        bl              sum
        ...
        ldp             x29, x30, [sp], 16
        ret
```

Coding Practice: Inline Subroutines

```
define(comment)

comment(minimum(1 = output register, 2 = input register, 3 = input register))
define(`minimum',
`mov $1, $3
cmp $2, $3
b.gt done
mov $1, $2
done:')


fmt1: .string "Between %d and %d, %d is smaller.\n"

   .balign 4                  // word align instructions
   .global main               // make main() global to call from OS
main:
   stp x29, x30, [sp, -16]!   // store frame record, allocate stack memory
   mov x29, sp                // update FP = SP

   mov x19, 3                 // make up a number
   mov x20, 5                 // make up another number

   adrp x0, fmt1              // 1st arg: print format
   add x0, x0, :lo12:fmt1     // load lower 12 bits of print format
   mov x1, x19                // 2nd arg: 1st number
   mov x2, x20                // 3rd arg: 2nd number

   // start #minimum#
   minimum(x3, x19, x20)      // 4th arg: x3 = min(x19, x20)
   // end #minimum#

   bl printf                      // Print result

   mov w0, 0
   ldp x29, x30, [sp], -16
   ret
```

```
//Definitions and comments are removed
//by M4, once they have been processed

fmt1: .string "Between %d and %d, %d is smaller.\n"

   .balign 4
   .global main
main:
   stp x29, x30, [sp, -16]!
   mov x29, sp

   mov x19, 3
   mov x20, 5

   adrp x0, fmt1
   add x0, x0, :lo12:fmt1
   mov x1, x19
   mov x2, x20

   // start #minimum#
   mov x3, x20
   cmp x19, x20
   b.gt done
   mov x3, x19
   done:
   // end #minimum#

   bl printf

   mov w0, 0
   ldp x29, x30, [sp], -16
   ret
```

Closed Subroutine

```
fmtinput: .string "The sum of %d, %d, %d, %d, %d, %d, and %d is: "
fmtsum: .string "%d\n"

.balign 4 // word align all instructions


//-SUM()---------------------------------------------------------
sum:
   stp x29, x30, [sp, -16]!
   mov x29, sp

   add w0, w0, w1 // input parameters are stored in w0-w7
   add w0, w0, w2 // but sum() only uses w0-w6
   add w0, w0, w3
   add w0, w0, w4
   add w0, w0, w5
   add w0, w0, w6

   return: ldp x29, x30, [sp], 16
   ret
//-MAIN()--------------------------------------------------------
-
```

```
//-MAIN()--------------------------------------------------------

.global main      // make main() global
                  // OS calls main(), so only main()
                  // needs to be global
main:
   stp x29, x30, [sp, -16]!
   mov x29, sp
   // Make up seven numbers as parameters
   mov w19, 10
   mov w20, 20
   mov w21, 30
   mov w22, 40
   mov w23, 50
   mov w24, 60
   mov w25, 70
   // Print parameters
   adrp x0, fmtinput // 1st arg to printf: string format
   add x0, x0, :lo12:fmtinput
   mov w1, w19       // 2nd arg to printf: first number
   mov w2, w20       // 3rd arg to printf: second number
   mov w3, w21       // 4th arg to printf: third number
   mov w4, w22       // 5th arg to printf: fourth number
   mov w5, w23       // 6th arg to printf: fifth number
   mov w6, w24       // 7th arg to printf: sixth number
   mov w7, w25       // 8th arg to printf: seventh number
   bl printf
   // Sum up the numbers, result is returned in w0.
   mov w0, w19       // 1st arg to sum: first number
   mov w1, w20       // 2nd arg to sum: second number
   mov w2, w21       // 3rd arg to sum: third number
   mov w3, w22       // 4th arg to sum: fourth number
   mov w4, w23       // 5th arg to sum: fifth number
   mov w5, w24       // 6th arg to sum: sixth number
   mov w6, w25       // 7th arg to sum: seventh number
   bl sum
   mov w1, w0       // Save the sum result before we
                    // overwrite x0 with print string format
   // Setup print format for result
   adrp x0, fmtsum
   add x0, x0, :lo12:fmtsum
   bl printf
done:
   mov w0, 0
   ldp x29, x30, [sp], 16
   ret
```

reference

- http://edwinckc.com/cpsc355/71-tutorial-5-oct-31-open-subroutines-simple-closed-subroutines
- lecture slides