# CSC 411

**Computer Organization (Spring 2023)**
**Lecture 11: Procedures**

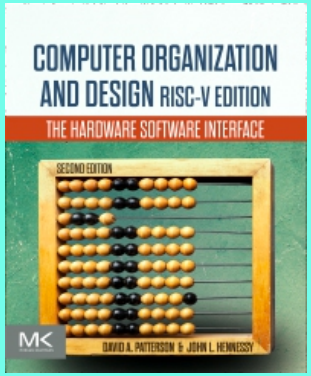
Prof. Marco Alvarez, University of Rhode Island

## Disclaimer

Some of the following slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)

The Hardware/Software Interface




## Procedure calling

- Think about the register file as a scratchpad
  - each procedure uses the scratchpad
  - when another procedure is called values may have to be save to **resume work** after returning from the callee

## Procedure calling steps

- Place parameters in **registers x10 to x17**
  - so the function can access them
- Transfer control to procedure
- Acquire storage for procedure
  - save registers that are needed
- Perform procedure's operations
- Place result in register for caller
  - restore any registers
- Return to place of call
  - address in x1

# Procedure call instructions

- Procedure call: **jump and link**

```
jal x1, <label>
```

  - address of following instruction put in x1
  - jumps to target address

- Procedure return: **jump and link register**

```
jalr x0, 0(x1)
```

  - like `jal` but jumps to 0 + address in x1
  - use `x0` as rd (cannot be changed)
  - can also be used for computed jumps (case/switch)

# Leaf procedure example

```c
int leaf_example(int g, int h, int i, int j) {
    int f;
    f = (g + h) - (i + j);
    return f;
}

// arguments g, ..., j in x10, ..., x13
// f in x20
// temporaries x5, x6
// need to save x5, x6, x20 on stack
```

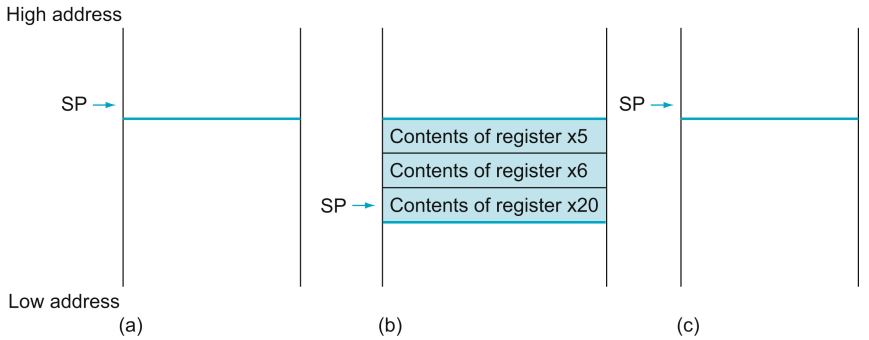
# Leaf procedure example


```c
int leaf_example(int g, int h, int i, int j)
{
    int f;
    f = (g + h) - (i + j);
    return f;
}

// arguments g, ..., j in x10, ..., x13
// f in x20
// temporaries x5, x6
// need to save x5, x6, x20 on stack
```


```
leaf_example:
    addi sp, sp, -12      # save register values on stack
    sw   x5, 8(sp)
    sw   x6, 4(sp)
    sw   x20, 0(sp)
    add  x5, x10, x11
    add  x6, x12, x13
    sub  x20, x5, x6
    addi x10, x20, 0      # copy result to return register
    lw   x20, 0(sp)       # restore register values from stack
    lw   x6, 8(sp)
    lw   x5, 16(sp)
    addi sp, sp, 12
    jalr x0, 0(x1)        # return to caller
```

# Local data on Stack



# Register usage

- x5 – x7, x28 – x31
  - temporary registers, **not preserved by the callee**
- x8 – x9, x18 – x27
  - saved registers, **callee saves and restores them**

| Preserved | Not preserved |
|---|---|
| Saved registers: `x8-x9, x18-x27` | Temporary registers: `x5-x7, x28-x31` |
| Stack pointer register: `x2(sp)` | Argument/result registers: `x10-x17` |
| Frame pointer: `x8(fp)` | |
| Return address: `x1(ra)` | |
| Stack above the stack pointer | Stack below the stack pointer |

# Register conventions

| Name | Register number | Usage | Preserved on call? |
|---|---|---|---|
| `x0` | 0 | The constant value 0 | n.a. |
| `x1 (ra)` | 1 | Return address (link register) | yes |
| `x2 (sp)` | 2 | Stack pointer | yes |
| `x3 (gp)` | 3 | Global pointer | yes |
| `x4 (tp)` | 4 | Thread pointer | yes |
| `x5-x7` | 5–7 | Temporaries | no |
| `x8-x9` | 8–9 | Saved | yes |
| `x10-x17` | 10–17 | Arguments/results | no |
| `x18-x27` | 18–27 | Saved | yes |
| `x28-x31` | 28–31 | Temporaries | no |

# Non-leaf Procedures

- Procedures that call other procedures
- For nested call, caller needs to save on the stack:
  - its return address
  - any arguments and temporaries needed after the call
- Restore from the stack after the call

# Non-leaf procedure example

```
int fact (int n) {
    if (n < 1) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}

// argument n in x10
// result in x10
```

# Non-leaf procedure example

```c
int fact (int n) {
    if (n < 1) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}

// argument n in x10
// result in x10
```

```
fact:
    addi sp, sp, -8     # save register values on stack
    sw   x1, 4(sp)      # save return address
    sw   x10, 0(sp)     # save n
    addi x5, x10, -1    # x5 = n-1
    bge  x5, x0, L1     # if n >= 1 go to L1
    addi x10, x0, 1     # set return value to 1
    addi sp, sp, 8      # pop stack (no need to restore values)
    jalr x0, 0(x1)      # return (base case)
L1:
    addi x10, x10, -1   # n = n-1
    jal  x1, fact       # make recursive call
    addi x6, x10, 0     # move result from recursive call to x6
    lw   x10, 0(sp)     # restore caller's n
    lw   x1, 4(sp)      # restore caller's return address
    addi sp, sp, 8      # pop stack
    mul  x10, x10, x6   # set return value
    jalr x0, 0(x1)      # return
```

# Memory layout

- Text
  - program code
- Static data
  - global variables, static variables, constants
  - x3 (global pointer) initialized to address allowing offsets into this segment
- Dynamic data
  - heap (e.g. malloc or new)
- Stack
  - automatic storage