


Assignment Project Exam Help

Add WeChat powcoder

# L6\_1 Assembly Functions

EECS 370 – Introduction to Computer Organization – Fall 2020



# Assignment Project Exam Help

## Learning Objectives

Add WeChat powcoder

- Understand how program data, particularly at the granularity of a function, maps to memory
- Identify data passed between functions and the mapping of that data to memory

<https://powcoder.com>

Add WeChat powcoder



# Assignment Project Exam Help

## Schedule Reminders

Add WeChat powcoder

- Homework 2 is due 9/29
  - A homework assignment is usually released soon after the previous one is due
- Project 1s and 1m due 9/24

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Call and Return

Address:

```
void foo()
{
    int x = 5;
    bar();
    x = x + 1;
    return;
}
```

1000

```
int x = 5;
```

1004

```
bar();
```

1008

```
x = x + 1;
```

1012

```
return;
```

```
void bar()
```

```
{
```

3000

```
int y = 10;
```

3004

```
return;
```

```
}
```

Order of  
execution:

```
int x = 5;
```

```
bar();
```

```
int y = 10;
```

```
return;
```

```
x = x + 1;
```

```
return;
```

Notes:

```
// branch to 3000
```

```
return; // branch to 1008
```

```
return; // branch to ???
```

Remember:

There can be many call sites for a function in a program, i.e., more than just foo() will call bar()

```
void baz()
```

```
{
```

```
bar();
```

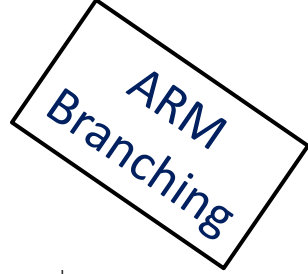
```
return;
```

```
}
```

# Assignment Project Exam Help

## Unconditional Branching Instructions

Add WeChat powcoder



Unconditional branch	branch	B	2500	go to PC + 10000	Branch to target address; PC-relative
	branch to register	BR	X30	go to X30	For switch, procedure return
	branch with link	BL	2500	X30 = PC + 4; PC + 10000	For procedure call PC-relative

Assignment Project Exam Help

- There are three types of unconditional branches in the EGV8 ISA.
  - The first (**B**) is the PC relative branch with the 26-bit offset from the last slide.
  - The second (**BR**) jumps to the address contained in a register (X30 above)
  - The third (**BL**) is like our PC relative branch but it does something else.
    - It sets X30 (always) to be the current PC+4 before it branches.
    - Why?
    - Function calls – return to next instruction

# Assignment Project Exam Help

## Branch with Link (BL)

Add WeChat powcoder



- Branch with Link is the branch instruction used to call functions
  - Functions need to know where they were called from so they can return.
    - In particular they will need to return to right after the function call
    - Can use "BR X30"
- Say that we execute the instruction `BL #500` when at PC 1000.
  - What address will be branched to?
  - What value is stored in X30?
  - How is that value in X30 useful?

<https://powcoder.com>

Add WeChat powcoder

# Converting function calls to assembly code



C code: `factorial(5);`

1. Need to pass arguments to the called function (`factorial()`)
2. Need to save return address of the caller
3. Need to save register values
4. Need to jump to `factorial` (callee)

<https://powcoder.com>

Add WeChat powcoder

Execute instructions for `factorial()`

Jump to return address

5. Need to get return value (for non-void return type)
6. Need to restore register values

# Task 1: Passing Arguments

- Where should you put all the arguments?
  - Registers?
    - Fast access but few in number and wrong size for some objects
  - Memory?
    - Good general solution but slow
- ARMv8 solution—and the usual answer
  - Registers and memory
    - Put the first few arguments in registers (if they fit) (X0 – X7)
    - Put the rest in memory on the call stack—important concept
- Comment: Make sure you understand the general idea behind a stack data structure—ubiquitous in computing
  - As basic concept it is a list in that you can only access at one end by pushing a data item into the top of the stack and popping an item off of the stack—real stacks are a little more complex



# Call stack



- ARM conventions (and most other processors) allocate a region of memory for the “call” stack
  - This memory is used to manage all the storage requirements to simulate function call semantics
    - Parameters (that were not passed through registers)
    - Local variables
    - Temporary storage (when you run out of registers and need somewhere to save a value)
    - Return address
    - etc.
- Sections of memory on the call stack [a.k.a. **stack frames**] are allocated when you make a function call, and de-allocated when you return from a function—the stack frame is a fixed template of memory locations

# An Older ARM (Linux) Memory Map

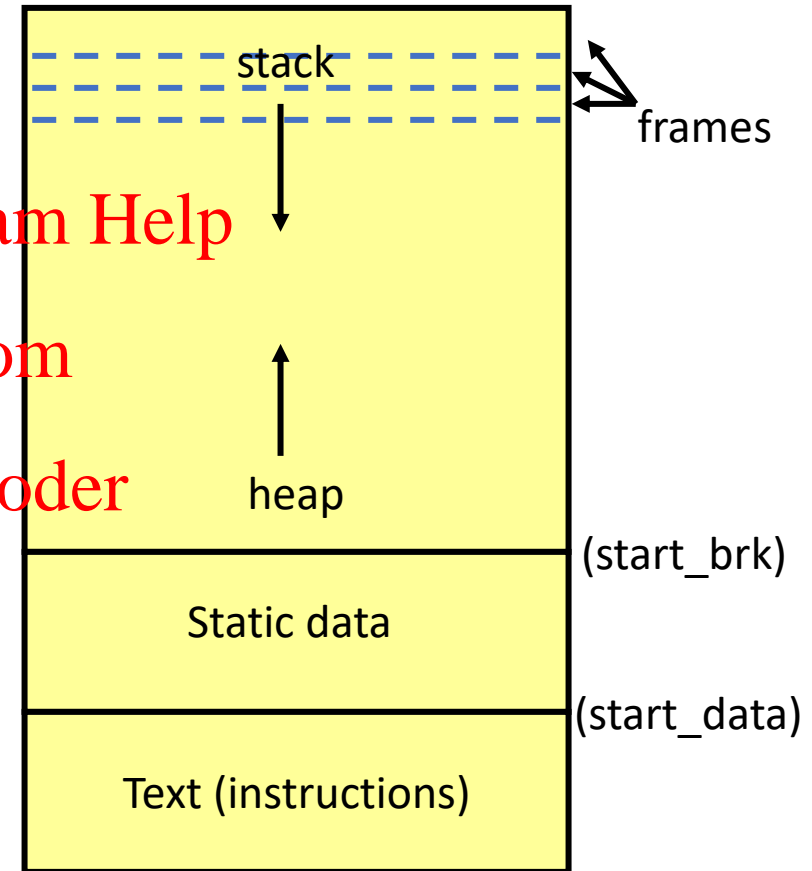


**Stack:** starts at `0x0000 007F FFFF FFFC` and grows down to lower addresses. Bottom of the stack resides in the SP register

**Heap:** starts above static data and grows up to higher addresses. Allocation done explicitly with `malloc()`. Deallocation with `free()`. Runtime error if no free memory before running into SP address. NB not same as data structure heap—just uninitialized mem.

**Static:** starts above text. Holds all global variables and those locals explicitly declared as “static”.

**Text:** starts at `0x0000 0000 0004 0000`. Holds all instructions in the program (except for dynamically linked library routines, DLLs)



# Assignment Project Exam Help

## Assigning variables to memory spaces

Add WeChat powcoder

Function Calls

```
int w;
```

w goes in static, as it is a global

```
void foo(int x)
```

x goes on the stack, as it is a parameter

```
{
```

```
    static int y[4];
```

Assignment Project Exam Help

y goes in static, 1 copy of this!!

```
    char *p;
```

<https://powcoder.com>

p goes on the stack

```
    p = malloc(10);
```

allocate 10 bytes on heap, p set to the address

Add WeChat powcoder

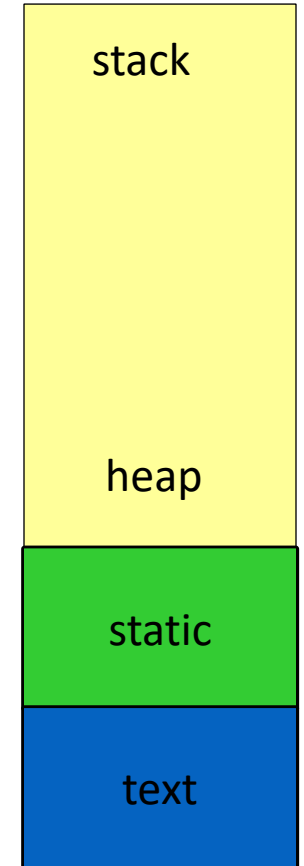
```
    // more instructions
```

```
    printf("%s\n", p);
```

string goes in static with a pointer to string on stack, p goes on stack

```
    return;
```

```
}
```



# The stack grows as functions are called

Add WeChat powcoder

Function  
Calls

```
void foo()  
{  
    int x, y[2];  
    bar(x);  
}
```

```
void bar(int x)  
{  
    int a[3];  
    printf();  
}
```

inside foo

foo's stack frame

bar returns

bar's stack frame

printf returns

printf's stack frame

bar calls printf

# The stack grows as functions are called

Add WeChat powcoder

Function  
Calls

```
void foo()
```

```
{  
    int x, y[2];  
    bar(x);  
}
```

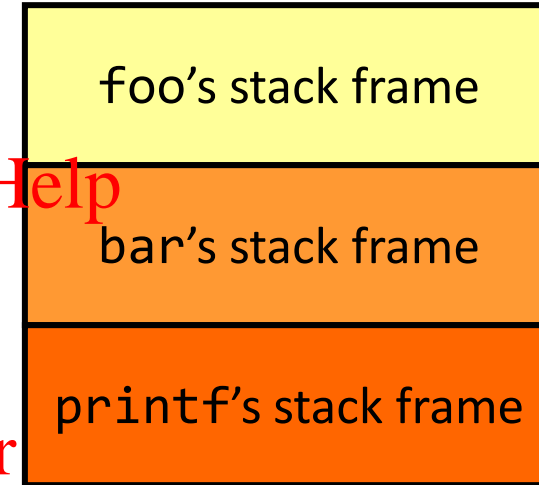
```
void bar(int x)
```

```
{  
    int a[3];  
    printf();  
}
```

inside foo

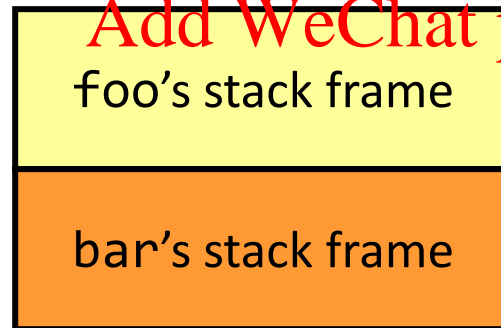


bar calls printf



Assignment Project Exam Help

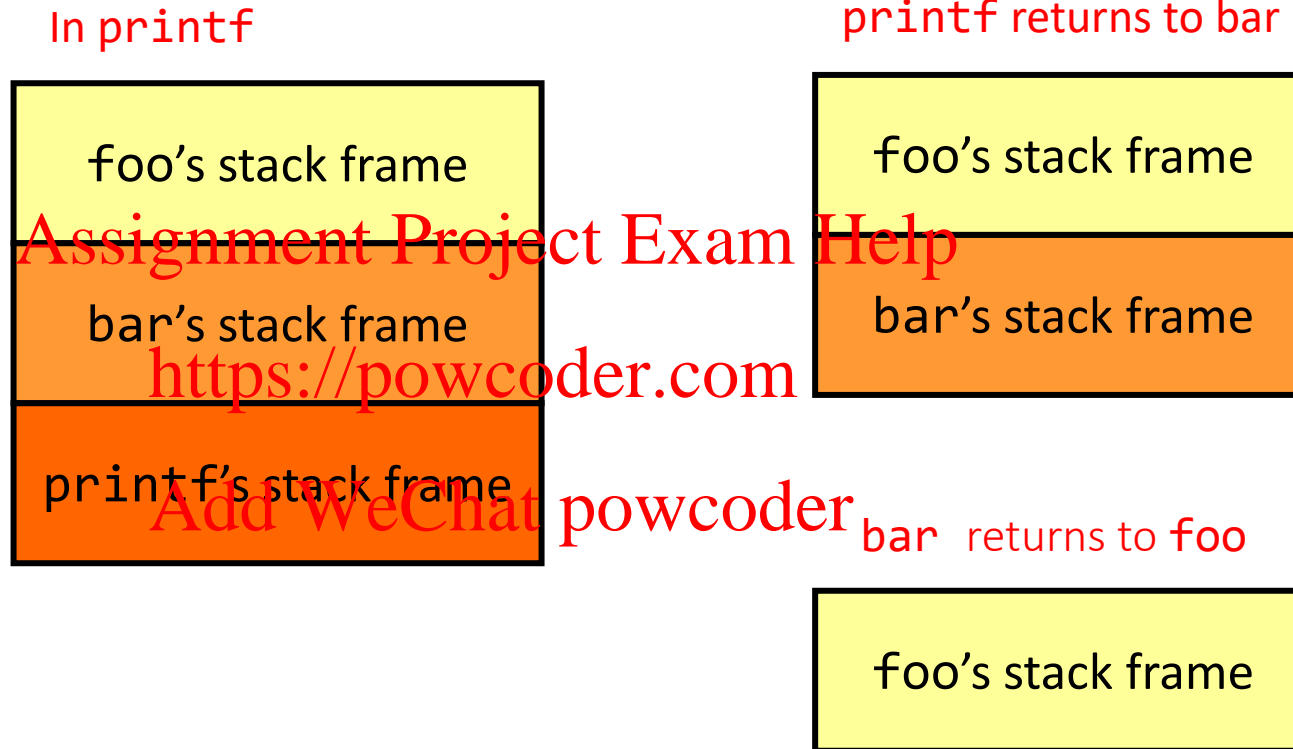
<https://powcoder.com>  
foo calls bar



Add WeChat powcoder

# The stack shrinks as functions return

```
void foo()  
{  
    int x, y[2];  
    bar(x);  
}  
  
void bar(int x)  
{  
    int a[3];  
    printf();  
}
```



# Stack frame contents (0)

Add WeChat powcoder

foo's stack frame

```
void foo()
{
    int x, y[2];
    bar(x);
}

void bar(int x)
{
    int a[3];
    printf();
}
```

return address to main
x
y[0]
y[1]
spilled registers in foo

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Stack frame contents (1)

Function Calls

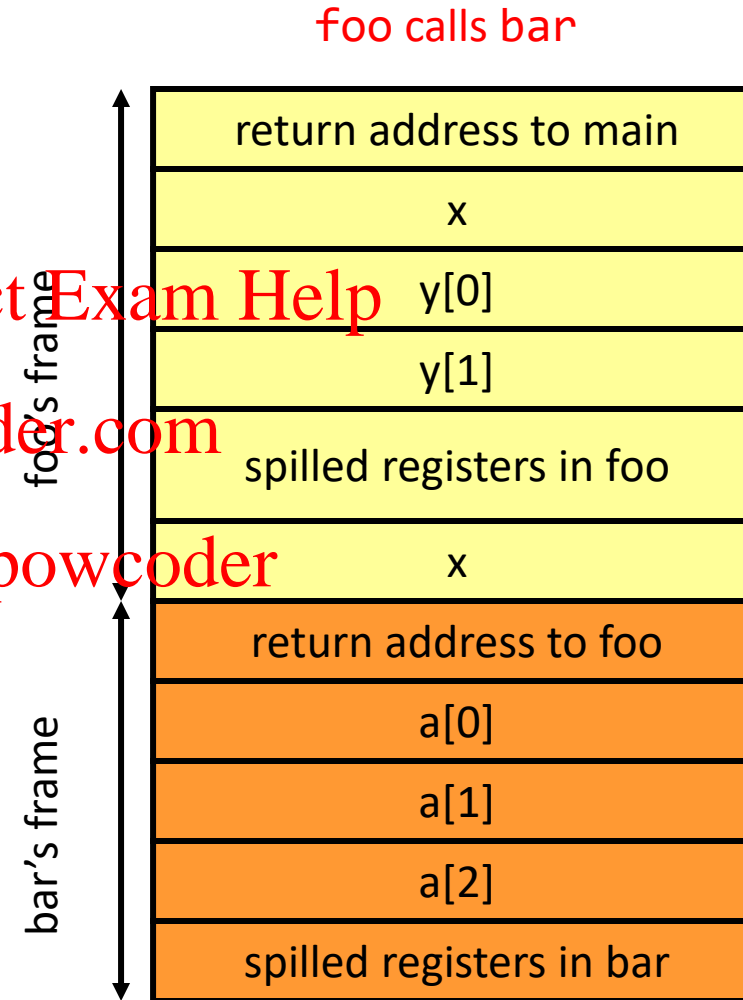
```
void foo()
{
    int x, y[2];
    bar(x);
}

void bar(int x)
{
    int a[3];
    printf();
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

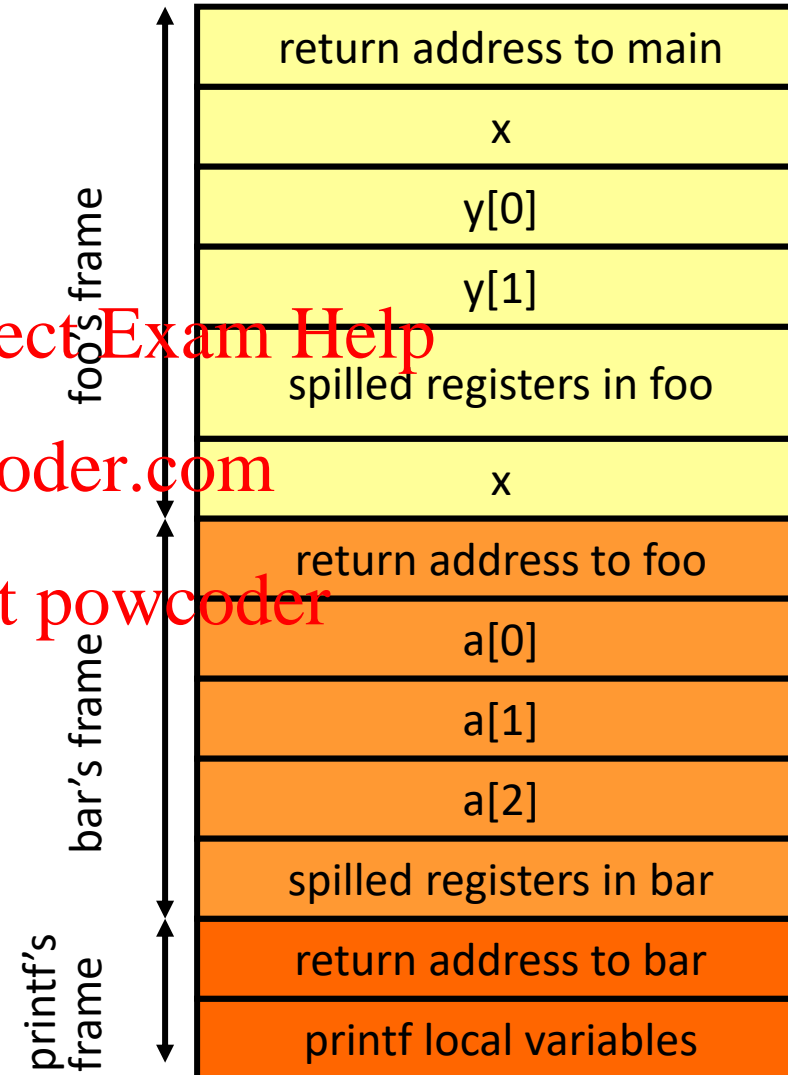




# Stack frame contents (2)

```
void foo()
{
    int x, y[2];
    bar(x);
}

void bar(int x)
{
    int a[3];
    printf();
}
```



Function Calls

# Recursive function example

Function Calls

```
void main()
{
    foo(2);
}
```

main calls foo

```
void foo(int a)
{
    int x, y[2];
    if (a > 0)
    {
        foo(a-1);
    }
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

foo calls foo

return address to ...
2
return address to main
x, y[0], y[1]
spills in foo
1
return addr to foo
x, y[0], y[1]
spills in foo
0
return addr to foo
x, y[0], y[1]
spills in foo

# Logistics

## Assignment Project Exam Help

Add WeChat powcoder


- There are 3 videos for lecture 6
    - L6\_1 – Assembly\_Functions
    - L6\_2 – Registers\_Caller/Callee
    - L6\_3 – Caller/Callee-Examples
  - There is one worksheet for lecture 6
    - 1. Caller/Callee-saved registers – wait!
- <https://powcoder.com>
- Add WeChat powcoder

Assignment Project Exam Help

Add WeChat powcoder

# L6\_2 Registers Caller/Callee

EECS 370 – Introduction to Computer Organization – Fall 2020



# Assignment Project Exam Help

## Learning Objectives

Add WeChat powcoder

- Understand how program data, particularly at the granularity of a function, maps to registers
- Identify data passed between functions and the mapping of that data to registers

<https://powcoder.com>

Add WeChat powcoder

# What about values in registers?

Function  
Calls

- When function “foo” calls function “bar”, function “bar” is, like all assembly code, going to store some values in registers.
- But function “foo” might have some values stored in registers that it wants to use after the call.
  - How can “foo” be sure “bar” won’t overwrite those values?
  - One answer: “foo” could save those values to memory (on the stack) before it calls “bar”.
    - Now “bar” can freely use registers
  - And “foo” will have to copy the values back from memory once “bar” returns.
- In this case the “caller” (foo) is saving the registers to the stack.

```
foo: addi x0, x0, #1  
     bl  bar  
     add x1, x0, x2
```

```
bar: movz x0, #3000
```

# Assignment Project Exam Help

## Register Spilling Example

Add WeChat powcoder

The process of putting less frequently used variables (or those needed later) into memory is called *spilling* registers

Caller-  
Callee

```
void foo(){
    int a,b,c,d;
    a = 5;
    b = 6;
    c = a+1;
    d=c-1;

    bar();

    d = a + d;
    return;
}
```

- The function foo is going to have values a, b, c, and d kept in registers.
  - For sake of argument, let's say that "a" is stored in X1, "b" in X2, etc.
- When foo calls bar, bar might end up writing to some of the same registers that foo is using.
  - In this case, if bar were to change the value of X1 (which holds a) or X4 (which holds d) then foo wouldn't behave correctly.
- **What we need is some way to ensure that when we call a function, it will not "trash" values we need after the call**
  - Definition: a value that is defined before a function call and needed after a function call is said to be "**live**" across the function call.

# Assignment Project Exam Help

## One solution: Caller Save

Add WeChat powcoder

Caller-  
Callee

```
void foo(){
    int a,b,c,d;
    a = 5;
    b = 6;
    c = a+1;
    d=c-1;
    save X1 to stack
    save X4 to stack
    bar();
    restore X1 from stack
    restore X4 from stack
    d = a + d;
    return;
}
```

- Anytime one function calls another function, the caller should first save to the stack any registers whose values it might need later.
  - After returning, those values will be copied back from the stack into their original register.
  - Now the “callee” function will not overwrite values its “caller” still needs.
  - We call this option “**caller save**”
- Again, let’s assume that a is stored in X1, b in X2, etc.
  - If we are using this “caller-save” option
    - What registers do we need to save to the stack before calling bar?
    - What registers do we need to restore from the stack?

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



# Assignment Project Exam Help

## Another solution: Callee Save

Add WeChat powcoder

```
void foo(){
    int a,b,c,d;
```

```
    a = 5;
    b = 6;
    c = a+1;
    d=c-1;
```

```
    bar();
```

```
    d = a + d;
```

```
    return;
```

```
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

- We could instead have each function save every register it is going to use before it does anything else.
  - Again, let's assume a is in X1, b in X2, etc.
- In this case, we'd save X1, X2, X3, and X4 before we did anything else and then restore them at the end of the function.
- Thus whatever function called "foo" would be sure its registers wouldn't get trashed by foo as foo would save and restore all registers.
- All functions would do the same thing.
  - so foo's values in X1 and X4 are safe from bar trashing them.

# Assignment Project Exam Help

## Another solution: Callee Save

Add WeChat powcoder

```
void foo(){
    int a,b,c,d;
    save X1, X2, X3, X4
    to stack

    a = 5;
    b = 6;
    c = a+1;
    d=c-1;

    bar();

    d = a + d;
    restore X1, X2, X3, X4
    from stack

    return;
}
```

- We could instead have each function save every register it is going to use before it does anything else.
  - Again, let's assume a is in X1, b in X2, etc.

## Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

- In this case, we'd save X1, X2, X3, and X4 before we did anything else and then restore them at the end of the function.
- Thus whatever function called "foo" would be sure its registers wouldn't get trashed by foo as foo would save and restore all registers.
- All functions would do the same thing.
  - so foo's values in X1 and X4 are safe from bar trashing them.

# “caller-save” vs. “callee-save”

Assignment Project Exam Help

Add WeChat powcoder

- So we have two basic options:
  - Each function can save registers **before** you make the function call and restore the registers when you return (**caller-save**).
    - What if the function you are calling doesn't use that register? No harm done, but wasted work!!!
  - You can save all registers you are going to use at the very start of each function (**callee-save**).
    - What if the caller function doesn't need that value? No harm done, but wasted work!!!
- Most common scheme is to have some of the registers be the responsibility of the caller, and others be the responsibility of the callee.

<https://powcoder.com>

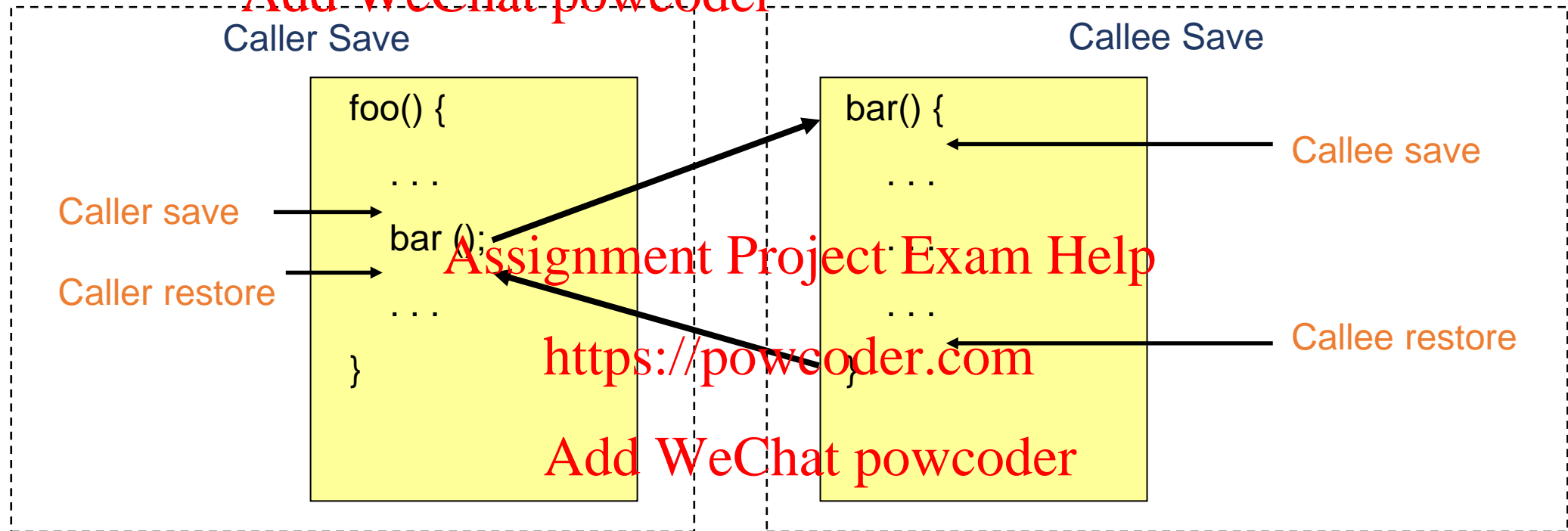
Add WeChat powcoder

# Assignment Project Exam Help

## Caller-Callee Save/Restore

Add WeChat powcoder

Caller-  
Callee



**Caller save registers:** Callee may change, so caller responsible for saving immediately before call and restoring immediately after call

**Callee save registers:** Must be the same value as when called. May do this by either not changing the value in a register **or** by inserting saves at the start of the function and restores at the end

# Saving/Restoring Optimizations



- Caller-saved
  - Only needs saving if it is “live” across a function call
  - Live = contains a useful value: Assign value before function call, use that value after the function call
  - In a leaf function (a function that calls no other function), caller saves can be used without saving/restoring
- Callee-saved
  - Only needs saving at beginning of function and restoring at end of function
  - Only save/restore it if function overwrites the register
- Each has its advantages. Neither is always better.

# Calling Convention

Caller-  
Callee

- This is a **convention**: calling convention
  - There is no difference in H/W between caller and callee save registers

Assignment Project Exam Help

- Passing parameters in registers is also a convention
- Allows assembly code written by different people to work together
  - Need conventions about who saves regs and where args are passed.
- These conventions collectively make up the ABI or “application binary interface”
- Why are these conventions important?
  - What happens if a programmer/compiler violates them?

# Assignment Project Exam Help

## Caller/Callee Selection

Add WeChat powcoder



- Select assignment of variables to registers such that the sum of caller/callee costs is minimized
  - Execute fewest save/restores
- Each function greedily picks its own assignment ignoring the assignments in other functions
  - Calling convention assures all necessary registers will be saved
- 2 types of problems
  1. Given a single function → Assume it is called 1 time
  2. Set of functions or program → Compute number of times each function is called if it is obvious (i.e., loops with known trip counts or you are told)

# Assignment Project Exam Help

## Assumptions

Add WeChat powcoder



- A function can be invoked by many different call sites in different functions.

## Assignment Project Exam Help

- Assume no inter-procedural analysis (hard problem)
  - A function has no knowledge about which registers are used in either its caller or callee
  - Assume main() is not invoked by another function
- Implication
  - Any register allocation optimization is done using function local information



# Logistics

## Assignment Project Exam Help

Add WeChat powcoder

- There are 3 videos for lecture 6
  - L6\_1 – Assembly\_Functions
  - L6\_2 – Registers\_Caller/Callee
  - L6\_3 – Caller/Callee-Examples
- There is one worksheet for lecture 6
  - 1. Caller/Callee-saved registers – wait!

<https://powcoder.com>


Add WeChat powcoder

Assignment Project Exam Help

Add WeChat powcoder

# L6\_3 Caller/Callee-Examples

EECS 370 – Introduction to Computer Organization – Fall 2020



# Assignment Project Exam Help

## Learning Objectives

Add WeChat powcoder

- Identify trade-offs between caller-save, callee-save, and mixed caller/callee-save register spilling
- Understanding of how to apply caller/callee/mixed register spilling to arbitrary functions and programs

<https://powcoder.com>

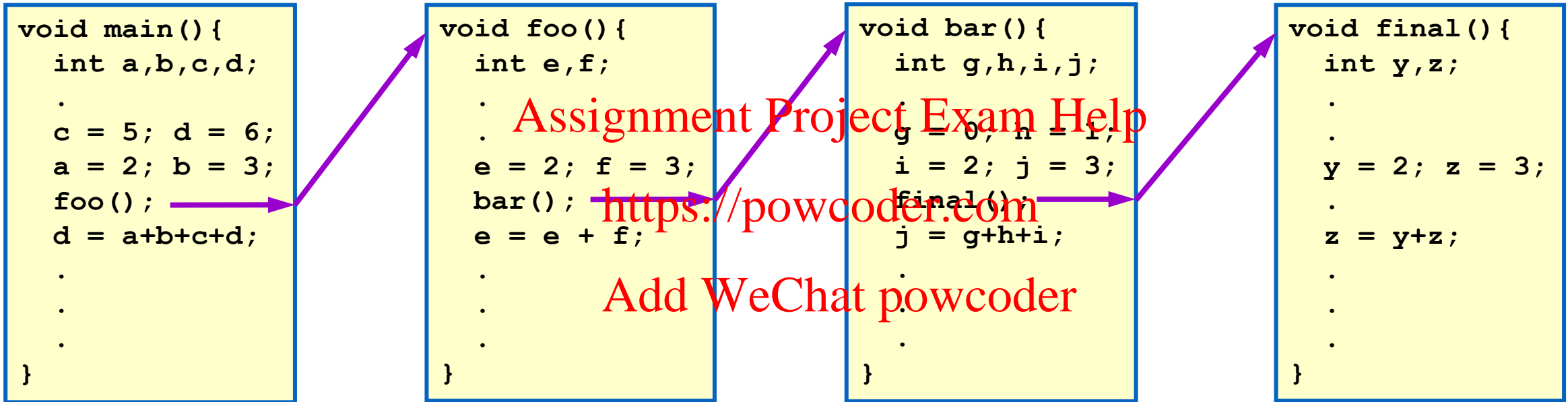
Add WeChat powcoder

# Assignment Project Exam Help

## Caller-saved vs. callee saved – Multiple function case (0)

Add WeChat powcoder

Caller-  
Callee



Note: assume main does not have to save any callee registers

# Caller-saved vs. callee saved – Multiple function case (1)

Add WeChat powcoder



- Questions:

1. In assembly code, how many registers need to be stored/loaded in total if we use a **caller-save** convention?

Assignment Project Exam Help

2. In assembly code, how many registers need to be stored/loaded in total if we use a **callee-save** convention?

<https://powcoder.com>

Add WeChat powcoder

3. In assembly code, how many registers need to be stored/loaded in total if we use a mixed **caller/callee**-save convention with 3 callee and 3 caller registers?

# Assignment Project Exam Help

## Question 1: Caller-Save

Add WeChat powcoder

Caller-  
Callee

```
void main() {  
    int a,b,c,d;  
    .  
    c = 5; d = 6;  
    a = 2; b = 3;  
    [4 STURW]  
    foo();  
    [4 LDURSW]  
    d = a+b+c+d;  
    .  
    .  
    .  
}
```

```
void foo() {  
    int e,f;  
    .  
    .  
    e = 2; f = 3;  
    [2 STURW]  
    bar();  
    [2 LDURSW]  
    e = e + f;  
    .  
    .  
    .  
}
```

```
void bar() {  
    int g,h,i,j;  
    .  
    .  
    g = 0; h = 1;  
    i = 2; j = 3;  
    [3 STURW]  
    final();  
    [3 LDURSW]  
    j = g+h+i;  
    .  
    .  
    .  
}
```

```
void final() {  
    int y,z;  
    .  
    .  
    y = 2; z = 3;  
    .  
    z = y+z;  
    .  
    .  
    .  
}
```

Total: 9 STURW / 9 LDURSW

# Assignment Project Exam Help

## Question 2: Callee-Save

Add WeChat powcoder

Caller-  
Callee

```
void main() {  
    int a,b,c,d;  
    .  
    c = 5; d = 6;  
    a = 2; b = 3;  
    foo();  
    d = a+b+c+d;  
    .  
    .  
}
```

```
void foo() {  
    [2 STURW]  
    int e,f;  
    .  
    .  
    e = 2; f = 3;  
    bar();  
    e = e + f;  
    .  
    .  
    .  
    [2 LDURSW]  
}
```

```
void bar() {  
    [4 STURW]  
    int g,h,i,j;  
    .  
    g = 0; h = 1;  
    i = 2; j = 3;  
    .  
    final();  
    j = g+h+i;  
    .  
    [4 LDURSW]  
}
```

```
void final() {  
    [2 STURW]  
    int y,z;  
    .  
    .  
    y = 2; z = 3;  
    .  
    z = y+z;  
    .  
    .  
    [2 LDURSW]  
}
```

Total: 8 STURw / 8 LDURSW

Note: assume main does not have to save any callee registers

# Assignment Project Exam Help

## Caller-Save and Callee-Save Registers(0)

Add WeChat powcoder

- Again, what we really tend to do is have some of the registers be the responsibility of the caller and others the responsibility of the callee.
    - So if you have six registers, then X0-X2 are caller-save registers and X3-X5 are callee-save registers.
  - How does that help?
- <https://powcoder.com>
- Add WeChat powcoder



# Assignment Project Exam Help

## Question 3: Mixed 3 Each Caller/Callee-Save

Add WeChat powcoder

X0-X2 are caller-save, X3-X5 are callee-save

```
void main() {
    int a,b,c,d;
    .
    c = 5; d = 6;
    a = 2; b = 3;

    foo();

    d = a+b+c+d;
}
```

```
void foo() {
    int e,f;
    .
    .
    e = 2; f = 3;
    bar();
    e = e + f;
    .
    .
    .
}
```

```
void bar() {
    int g,h,i,j;
    .
    g = 0; h = 1;
    i = 2; j = 3;

    final();

    j = g+h+i;
    .
}
```

```
void final() {
    int y,z;
    .
    .
    y = 2; z = 3;
    .
    z = y+z;
    .
    .
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Assignment Project Exam Help

X0-X2 are caller-save, X3-X5 are callee-save

## Question 3: Mixed 3 Each Caller/Callee-Save

Add WeChat powcoder

For main, we'd like to put all the variables into callee save registers. But we only have 3 callee save registers (X3-X5), so one variable needs to end up in a caller-save register.

```
void main() {
    int a,b,c,d;
    .
    c = 5; d = 6;
    a = 2; b = 3;
    [1 STURW]
    foo();
    [1 LDURSW]

    d = a+b+c+d;
}
```

1 caller reg.  
3 callee reg.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Question 3: Mixed 3 Each Caller/Callee-Save

X0-X2 are caller-save, X3-X5 are callee-save

```
void foo() {
    [2 STURW]
    int e, f;
    .
    .
    e = 2; f = 3;
    bar();
    e = e + f;
    .
    .
    .
    [2 LDURSW]
}
```

2 callee reg.

For foo it doesn't really matter what registers we use.  
Either way we will have to save and restore 2 values

# Assignment Project Exam Help

## Question 3: Mixed 3 Each Caller/Callee-Save

Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
void bar() {
    [3 STURW]
    int g, h, i, j;
    .
    g = 0; h = 1;
    i = 2; j = 3;

    final();
    j = g+h+i;
    .
    [3 LDURSW]
}
```

1 caller reg.

3 callee reg.

For bar, "j" should be allocated to caller-save register. The others don't matter.

# Question 3: Mixed 3 Each Caller/Callee-Save

X0-X2 are caller-save, X3-X5 are callee-save

Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
void final() {  
  
    int y,z;  
    .  
    .  
    y = 2; z = 3;  
    .  
    z = y+z;  
    .  
    .  
}
```

3 caller reg.

# Assignment Project Exam Help

## Question 3: Mixed 3 Each Caller/Callee-Save

Add WeChat powcoder

For main, we'd like to put all the variables into callee save registers. But we only have 3 callee save registers (X3-X5), so one variable needs to end up in a caller-save register.

```
void main() {
    int a,b,c,d;
    .
    c = 5; d = 6;
    a = 2; b = 3;
    [1 STURW]
    foo();
    [1 LDURSW]

    d = a+b+c+d;
}
```

1 caller reg.  
3 callee reg.

For foo it doesn't really matter what registers we use. Either way we will have to save and restore 2 values

```
void foo() {
    [2 STURW]
    int e,f;
    .
    .
    e = 2; f = 3;
    bar();
    e = e + f;
    .
    .
    .
    [2 LDURSW]
}
```

2 callee reg.

```
void bar() {
    [3 STURW]
    int g,h,i,j;
    .
    g = 0; h = 1;
    i = 2; j = 3;

    final();
    j = g+h+i;
    .
    .
    [3 LDURSW]
}
```

1 caller reg.  
3 callee reg.

For bar, "j" should be allocated to caller-save register. The others don't matter.

```
void final() {
    int y,z;
    .
    .
    y = 2; z = 3;
    .
    z = y+z;
    .
    .
}
```

3 caller reg.

Total: 6 STURW / 6 LDURSW

# Assignment Project Exam Help

It can get quite a bit more complex than this

Add WeChat powcoder



- Multiple function calls in a given function will make things more complex

- As will loops

Assignment Project Exam Help

- The video review for caller/callee save found on the class website is quite useful

- Discussion videos also go over some examples

<https://powcoder.com>

Add WeChat powcoder



EECS 370: INTRODUCTION TO COMPUTER ORGANIZATION  
FALL 2020

ANNOUNCEMENTS

COURSE OVERVIEW

LECTURES

LECTURE RECORDINGS

DISCUSSION RECORDINGS

DISCUSSIONS

STAFF AND HOURS

RESOURCES

VIDEO REVIEW

HOMEWORKS

## Video Review

- EECS 370 Bonus Review #1 - Binary Representation/Operations
- EECS 370 Bonus Review #2 - Floating Point Review
- EECS 370 Review #1 - Binary, Hexadecimal, and Two's Complement
- EECS 370 Review #2 - Struct Alignment
- EECS 370 Review #3 - Endianness and ARM Loads/Stores
- EECS 370 Review #4 - Branching and C to ARM Example
- EECS 370 Review #5 - Caller/Callee Save Registers
- EECS 370 Review #6 - Symbol and Relocation Tables
- EECS 370 Review #7 - Benchmarking Datapaths
- EECS 370 Reviews #8 - Data Hazard Resolution
- EECS 370 Review #9 - Benchmarking with Hazards
- EECS 370 Review #10 - Three C's of Cache Misses
- EECS 370 Review #11 - Reverse Engineering the Cache
- EECS 370 Review #12 - Virtual Memory Overview
- EECS 370 Review #13 - Virtually vs. Physically Addressed Caches

# Does Recursion Change Caller/Callee?

- No! Treat `foo()` just like any normal function and assume you are calling an unknown function.

```
void main() {
    int a,b,c,d;
    .
    c = 5; d = 6;
    a = 2; b = 3;
    foo();
    d = a+b+c+d;
    .
    .
}
```

```
void foo() {
    int a,b,c;
    c = 4;
    .
    a = 2; b = 3;
    foo(c-1,b+1);
    a = a + b;
    .
    .
    b = 4;
    foo(b,9);
    b = a - b;
    .
}
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



# LEGV8 ABI—Application Binary Interface



- The ABI is an agreement about how to use the various registers. These can be broken into three groups
  - Some registers are reserved for special use. Register usage definitions
    - (X31) zero register, (X30) link register, (X29) frame pointer, (x28) stack pointer, (X16-18) reserved for other uses
- Callee save: X19-X27
- Caller save: X0-X15
  - In addition, we pass arguments using registers X0-X7 (and memory if there are more arguments)
  - Return value goes into X0

# Logistics

## Assignment Project Exam Help

Add WeChat powcoder

- There are 3 videos for lecture 6
  - L6\_1 – Assembly\_Functions
  - L6\_2 – Registers\_Caller/Callee
  - L6\_3 – Caller/Callee-Examples
- There is one worksheet for lecture 6
  - 1. Caller/Callee-saved registers – complete now!

<https://powcoder.com>

Add WeChat powcoder