EE 109 Unit 16 – Stack Frames

Arguments and Return Values

- MIPS convention is to use certain registers for this task
  - _____ used to pass up to 4 arguments. If more arguments, use the stack
  - _____used for return value
    - Only 1 return value but it may be a double-word (64-bits) in which case $v1 will also be used

Number	Name	Purpose
$0	$zero	Constant 0
$1	$at	Assembler temporary (psuedo-instrucs.)
$2-$3	$v0-$v1	Return value ($v1 only used for dword)
$4-$7	$a0-$a3	Arguments (first 4 of a subroutine)
$8-$15, $24,$25	$t0-$t9	Temporary registers
$16-$23	$s0-$s7	Saved registers
$26-$27	$k0-$k1	Kernel reserved
$28	$gp	Global pointer (static global data var's.)
$29	$sp	Stack pointer
$30	$fp	Frame pointer
$31	$ra	Return address

Arguments and Return Values

```
void main() {
   int ans,arg1,arg2;
   ans = avg(arg1, arg2);
}


int avg(int a, int b) {
   int temp=1;  // local var's
   return a+b >> temp;
}
```

C Code

```
        ...
MAIN:   la   $s0, arg1    # Get addr.
        la   $s1, arg2    # of arg1/arg2
        lw   $a0, 0($s0)  # Get val. of
        lw   $a1, 0($s1)  # arg1/arg2
        jal  AVG          # call function
        la   $s2, ans
        sw   $v0, ($s2) # store ans.
        ...
AVG:    li   $t0, 1       # temp=1
        add  $v0,$a0,$a1  # do a+b
        srav $v0,$v0,$t0  # do shift
        jr   $ra
```

Equivalent Assembly

Assembly & HLL's

- When coding in assembly, a programmer can optimize usage of registers and store only what is needed to memory/stack
  - Can pass additional arguments in registers (beyond $a0-$a3)
  - Can allocate variables to registers (not use memory)
  - Can handle spilling registers to memory only when necessary
- When coding in an HLL & using a compiler, certain conventions are followed that may lead to heavier usage of the _____
  - We have to be careful not to _____ registers that have useful data

Compiler Handling of Subroutines

- High level languages (HLL) use the stack:
  - to _____ values including the return address
  - to pass additional _____ to a subroutine
  - for storage of _____ declared in the subroutine
- Compilers usually put data on the stack in a certain order, which we call a _____
- To access this data on the stack a pointer called the _____ ($30=$___) is often used in addition to the normal stack pointer ($sp)

Stack Frame Motivation

- Assume the following C code
- Now assume each function was written by a different programmer on their own (w/o talking to each other)
- What could go wrong?

```
int x=5, nums[10];
int main()
{ caller(x, nums);
  return 0;
}
int caller(int z, int* dat)
{  int a = dat[0]-1;
   return callee(5)+a+z;
}
int callee(int val)
{ return (val+3)/2; }
```


Stack Frame Motivation

- The caller needs to ensure the callee routine does not overwrite a needed register
  1. Caller may have his own _____ in $a0-$a3 and then need to call a subroutine and use $a0-$a3

```
int x=5, nums[10];
int main()
{ caller(x, nums); return 0; }

int caller(int z, int* dat)
{  int a = dat[0]-1;
   return callee(5)+a+z;  }

int callee(int val)
{ return (val+3)/2; }
```

```
        .text
MAIN:   la   $t0,x
        lw   $a0,0($t0)
        la   $a1,NUMS
        jal  CALLER
        ...
CALLER: lw   $s0, 0($a1)
        addi $s0,$s0,-1
        li   $a0, 5
        jal  CALLEE
        add  $v0,$v0,$s0
        add  $v0,$v0,$a0
        jr   $ra
CALLEE: addi $s0,$a0,3
        sra  $v0,$s0,1
        jr   $ra
```

New Value Loaded into $a0

But old value needed here

Stack Frame Motivation

- The caller needs to ensure the callee routine does not overwrite a needed register
  1. Caller may have his own arguments in $a0-$a3 and then need to call a subroutine and use $a0-$a3
  2. Return address ($ra) [We've already seen this problem]

```
int x=5, nums[10];
int main()
{ caller(x, nums); return 0; }

int caller(int z, int* dat)
{  int a = dat[0]-1;
   return callee(5)+a+z;  }

int callee(int val)
{ return (val+3)/2; }
```

```
        .text
MAIN:   la   $t0,x
        lw   $a0,0($t0)
        la   $a1,NUMS
        jal  CALLER
        ...
CALLER: lw   $s0, 0($a1)
        addi $s0,$s0,-1
        li   $a0, 5
        jal  CALLEE
        add  $v0,$v0,$s0
        add  $v0,$v0,$a0
        jr   $ra
CALLEE: addi $s0,$a0,3
        sra  $v0,$s0,1
        jr   $ra
```

2 2 2

Stack Frame Motivation

- The caller needs to ensure the callee routine does not overwrite a needed register
  1. Caller may have his own arguments in $a0-$a3 and then need to call a subroutine and use $a0-$a3
  2. Return address ($ra)
  3. Register values calculated _____ the call but used _____the call (e.g. $s0)

```
int x=5, nums[10];
int main()
{ caller(x, nums); return 0; }

int caller(int z, int* dat)
{  int a = dat[0]-1;
   return callee(5)+a+z;  }

int callee(int val)
{ return (val+3)/2; }
```

```
          .text
MAIN:     la    $t0,x
          lw    $a0,0($t0)
          la    $a1,NUMS
          jal   CALLER
          ...
CALLER:   lw    $s0, 0($a1)
          addi  $s0,$s0,-1      (3)
          li    $a0, 5
          jal   CALLEE
          add   $v0,$v0,$s0     (3)
          add   $v0,$v0,$a0
          jr    $ra
CALLEE:   addi  $s0,$a0,3       (3)
          sra   $v0,$s0,1
          jr    $ra
```

Callee unknowingly overwrites $s0 which will cause CALLER to malfunction

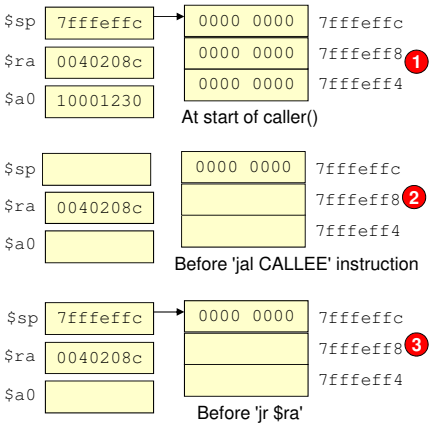
Solution

- If you're not sure whether some other subroutine is using a register (and needs it later)…
- _____ it to the stack before you overwrite it
  - Recall a push:
    addi $sp, $sp, -4
    sw *reg_to_save*, 0($sp)
- _____ it from the stack before you return
  - Recall a pop:
    lw *reg_to_restore*, 0($sp)
    addi $sp, $sp, 4

```
          .text
MAIN:     la    $t0,x
          lw    $a0,0($t0)
          la    $a1,NUMS
          jal   CALLER
          ...
CALLER:   addi  $sp,$sp,-4
          sw    $ra, 0($sp)     Save $ra

          lw    $s0, 0($a1)
          addi  $s0,$s0,-1

          addi  $sp,$sp,-4      Save $a0
          sw    $a0,0($sp)

          li    $a0, 5
          jal   CALLEE

          lw    $a0, 0($sp)     Restore $a0
          addi  $sp,$sp,4
          add   $v0,$v0,$s0
          add   $v0,$v0,$a0
          lw    $ra, 0($sp)     Restore $ra
          addi  $sp,$sp,4
          jr    $ra
```


Solution

- If you're not sure whether some other subroutine is using a register (and needs it later)…



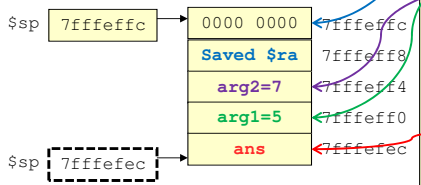

```
          .text
MAIN:     la    $t0,x
          lw    $a0,0($t0)
          la    $a1,NUMS
          jal   CALLER
          ...
CALLER:   addi  $sp,$sp,-4      (1)
          sw    $ra, 0($sp)

          lw    $s0, 0($a1)
          addi  $s0,$s0,-1

          addi  $sp,$sp,-4
          sw    $a0,0($sp)

          li    $a0, 5          (2)
          jal   CALLEE

          lw    $a0, 0($sp)
          addi  $sp,$sp,4
          add   $v0,$v0,$s0
          add   $v0,$v0,$a0
          lw    $ra, 0($sp)
          addi  $sp,$sp,4       (3)
          jr    $ra
```


Local Variables

- A functions **local variables** are also allocated on the stack

```
void main() {
   // Allocate 3 integers
   int ans, arg1=5, arg2=7;
   ans = avg(arg1, arg2);
} // vars. deallocated here
```

C Code



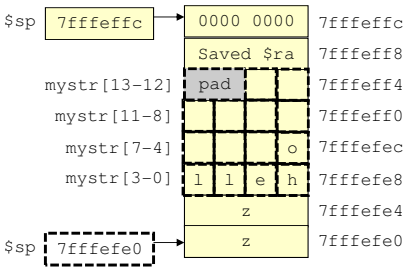

```
MAIN:   addi $sp, $sp, -4
        sw   $ra, 0($sp) # save $ra
        # Now allocate 3 integers
        addi $sp, $sp, _____
        li   $t0, 5
        sw   $t0, _____($sp)
        li   $t0, 7
        sw   $t0, _____($sp)

        lw   $a0, 4($sp) # Get val. of
        lw   $a1, 8($sp) # arg1/arg2
        jal  AVG         # call function
        sw   $v0, __($sp) #store ans.
        ...
        # deallocate local vars
        addi $sp,$sp,_____
        lw   $ra, 0($sp)
        addi $sp,$sp,4
        jr   $ra
```

Equivalent Assembly

Local Variables

- Locally declared arrays are also allocated on the stack
- **Be careful**: variables and arrays often must start on well-defined address boundaries

```
void main() {
   char mystr[14] = "hello...";
   double z;
}
```
C Code

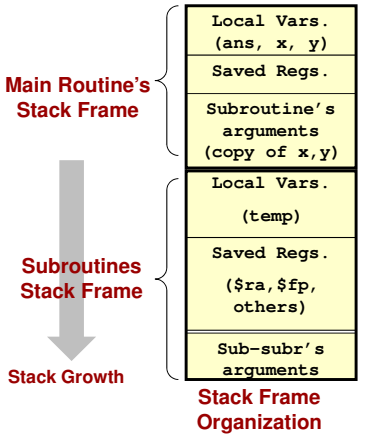



```
MAIN:   addi $sp, $sp, -4
        sw   $ra, 0($sp) # save $ra
        # Now allocate array
        addi $sp, $sp, _____ # not -14
        # May pad to get to 8-byte
        # boundary..

        # now alloc. z
        addi $sp, $sp, _____

        # deallocate local vars
        addi $sp,$sp,_____
        lw   $ra, 0($sp)
        addi $sp,$sp,4
        jr   $ra
```
Equivalent Assembly

Stack Frames

- Frame = ***Def:*** All data on stack belonging to a subroutine/function
  - Space for arguments (in addition to $a0-$a3)
  - Space for saved registers ($fp, $s0-$s7, $ra)
  - Space for local variables (those declared in a function)

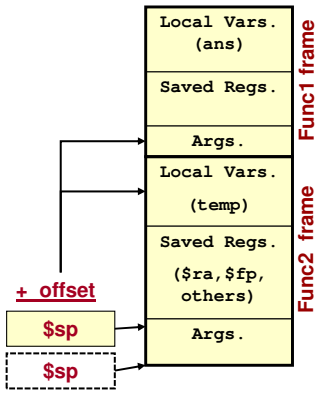
```
void main() {
  int ans, x, y;
  ...
  ans = avg(x, y);
}
int avg(int a, int b) {
  int temp=1;  // local var's
  ...
}
```



Accessing Values on the Stack

- Stack pointer ($sp) is usually used to access only the _____ value on the stack
- To access arguments and local variables, we need to access values _____ in the stack
  - We can simply use an _____from $sp [8($sp)]
- Unfortunately other push operations by the function may change the $sp requiring different displacements at different times for the same variable
- This can work, but can be confusing

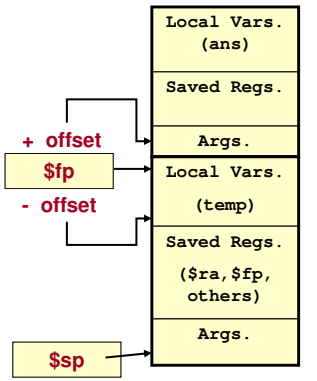
```
lw $t0, 16($sp) # access var. x
addi $sp,$sp,-4 # $sp changes
sw $t0, 20($sp) # access x with
                # diff. displacement
```



To access parameters we could try to use some displacement [i.e. d($sp)] but if $sp changes, must use new d value

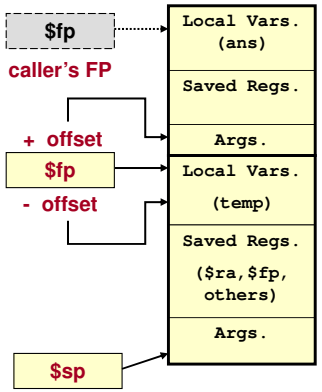
Frame Pointer

- Solution: Use another pointer that _____ _____ during execution of a subroutine
- We call this the Frame Pointer ($fp) and it usually points to the base of the current routines frame (i.e. the ***first*** word of the stack frame) [other implementations might have it points at the last word of the frame]
- $fp will not change during the course of subroutine execution
- Can use constant offsets from $fp to access parameters or local variables
  - Key 1: $fp doesn't change during subroutine execution
  - Key 2: Number of arguments, local variables, and saved registers is known at compile time so compiler can easily know what offsets to use



Frame Pointer and Subroutines

- Problem is that each executing subroutine needs _____ _____
- The called subroutine must _____ the caller's $fp and setup its own $fp
  - Usually performed immediately after allocating frame space and saving $ra
- The called subroutine must _____ the caller's $fp before it returns

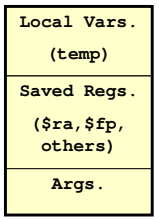


Part II

STACK FRAMES

Review

- The stack is used to store
  - Arguments passed by one subroutine to the next
    - Especially if there are more arguments than can fit in registers (i.e. MIPS uses $a0-$a3 for arguments…but what if there are 5 arguments)
  - Saved register values that a subroutine might need later
    - Best example is $ra
  - Local variables
- The stack will be accessed with $sp but also with $fp
  - Each offset from $fp is a different variable



Simple Example

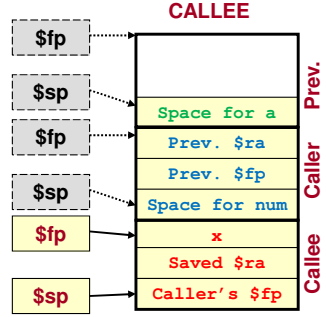
C Code

```
void caller(int a){
  callee(1);
}

int callee(int num){
  int x = 6;
  return x + num;

}
```

Assembly Code

```
          .text
CALLER:   addi  $sp,$sp,-12
          sw    $ra,8($sp)
          sw    $fp,4($sp)
          addi  $fp,$sp,8
          sw    $a0,4($fp)
          li    $a0, 1
          jal   CALLEE
          lw    $a0,4($fp)
          lw    $ra,8($sp)
          lw    $fp,4($sp)
          addi  $sp,$sp,12
          jr    $ra
CALLEE:   addi  $sp,$sp,-12
          sw    $ra,4($sp)
          sw    $fp,0($sp)
          addi  $fp,$sp,8
          li    $t0,6
          sw    $t0,0($fp)
          add   $v0,$t0,$a0
          lw    $fp,0($sp)
          lw    $ra,4($sp)
          addi  $sp,$sp,12
          jr    $ra
```

Stack during execution of CALLEE



Example 2

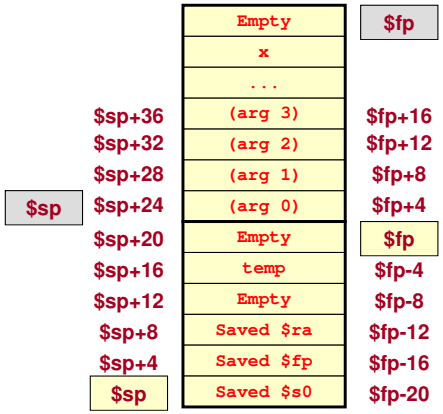
```
int ans;
void main() {
  int x = 3;
  ans = avg(1,5);
  x = x + 1;
}

int avg(int a, int b) {
  int temp = 1;
  return a + b >> temp;
}
```

```
        .text
MAIN:   ...
        li    $s0, 3
        li    $a0, 1
        li    $a1, 5
        jal   AVG
        sw    $v0,0($gp)
        addi  $s0,$s0,1
        sw    $s0,-4($fp)
        ...

AVG:    addi  $sp,$sp,-24
        sw    $ra,8($sp)
        sw    $fp,4($sp)
        addi  $fp,$sp,20
        sw    $s0,-20($fp)
        li    $s0,1
        sw    $s0,-4($fp)
        add   $v0,$a0,$a1
        srav  $v0,$v0,$s0
        lw    $s0,-20(fp)
        lw    $fp,4($sp)
        lw    $ra,8($sp)
        addi  $sp,$sp,24
        jr    $ra
```


Example 2

```
        .text
MAIN:   ...
        li    $s0, 3
        li    $a0, 1
        li    $a1, 5
        jal   AVG
        sw    $v0,0($gp)
        addi  $s0,$s0,1
        sw    $s0,-4($fp)
        ...

AVG:    addi  $sp,$sp,-24
        sw    $ra,8($sp)
        sw    $fp,4($sp)
        addi  $fp,$sp,20
        sw    $s0,-20($fp)
        li    $s0,1
        sw    $s0,-4($fp)
        add   $v0,$a0,$a1
        srav  $v0,$v0,$s0
        lw    $s0,-20(fp)
        lw    $fp,4($sp)
        lw    $ra,8($sp)
        addi  $sp,$sp,24
        jr    $ra
```

Convention: Local variable section must start and end on an 8-byte boundary



Stack during execution of AVG

Real Output from gcc

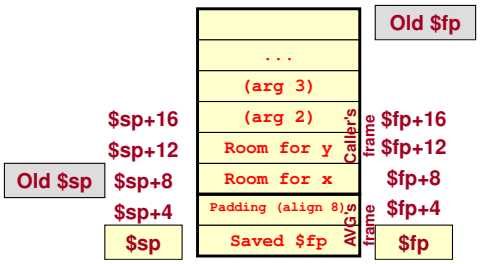
C Code

```
int avg(int x, int y)
{
  return (x+y)/2;
}
```

Assembly Code

Code generated by mips-gcc

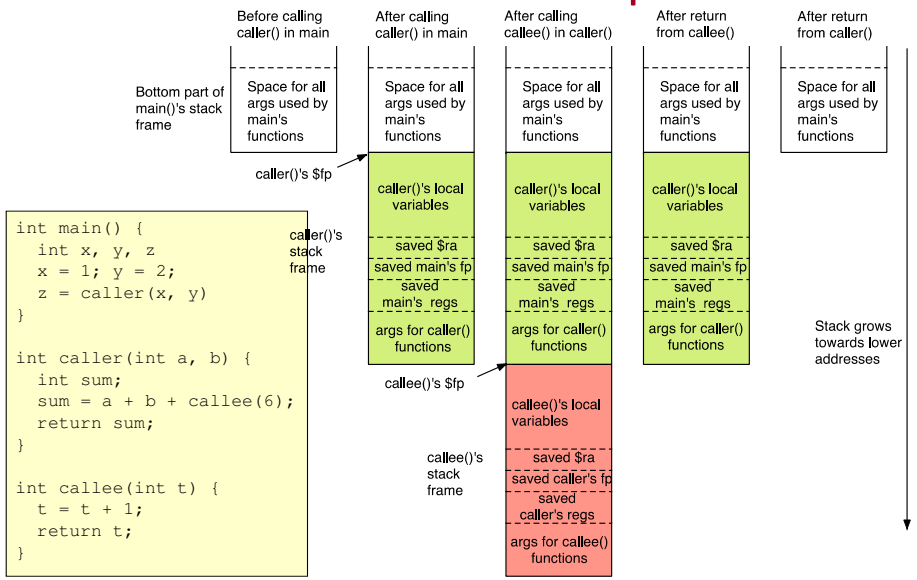
```
        .file   1 "avg.cpp"
        .text
        .align  2
        .globl  _Z3avgii
$LFB2:
        .ent    _Z3avgii
_Z3avgii:
        addiu   $sp,$sp,-8
$LCFI0:
        sw      $fp,0($sp)
$LCFI1:
        move    $fp,$sp
$LCFI2:
        sw      $4,8($fp)
        sw      $5,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        addu    $3,$3,$2
        sra     $2,$3,31
        srl     $2,$2,31
        addu    $2,$3,$2
        sra     $2,$2,1
        move    $sp,$fp
        lw      $fp,0($sp)
        addiu   $sp,$sp,8
        j       $31
        .end    _Z3avgii
$LFE2:
```



Stack during execution of AVG

This MIPS compiler points the $fp at the top of the frame and not the bottom (i.e. it will usually match $sp)

Stack Frame Example




```
int main() {
  int x, y, z
  x = 1; y = 2;
  z = caller(x, y)
}

int caller(int a, b) {
  int sum;
  sum = a + b + callee(6);
  return sum;
}

int callee(int t) {
  t = t + 1;
  return t;
}
```

Stack Summary

- Data associated with a subroutine is a _____ relationship (i.e. many instances may be running at the same time…recursion). A stack allows for any number of concurrent instances to all have their own storage.
- Stack grows towards _____ addresses
- Stack frames defines _____ of data related to a subroutine
- A subroutine should leave the stack & $sp in the same condition it found it
- _____ are dedicated registers to maintaining the system stack