

ECE 3140/CS 3420 Computer Organization Spring 2009

Procedures
Assemblers

ECE3140/CS3420



Cornell University

There are 10 types of people in the
world.

Those who understand binary ...
... and those who don't.

ECE3140/CS3420



Cornell University

Announcements

- Homework 2 to be posted today
- Project 1
 - Due TODAY, Feb 10 at 10:00pm
 - Don't wait until the last minute to post to CMS!
 - Problems in convert.s have been fixed
 - Your programs MUST follow the calling convention as described in class

ECE3140/CS3420



Cornell University

3

Hennessy and Patterson

- Read Chapter 1
 - 1.1-1.9
- Read Chapter 2
 - 2.1 through 2.14,
 - B.1-B.6, B.10
 - MIPS Calling Convention Document (website)
 - Notes on Programming in C (website)
- Read Chapter 3
 - 3.1 through 3.2
- Read Appendix C (for Tuesday)
 - C.1-C.6

ECE3140/CS3420



Cornell University

4

Procedures, Recap

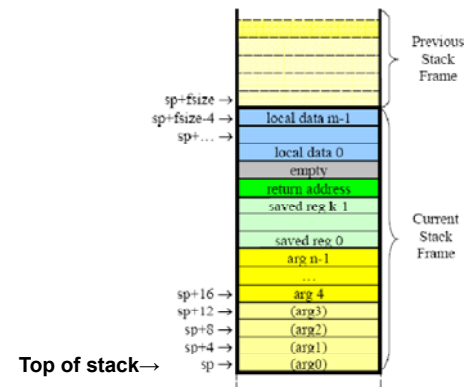
- A *stack frame* holds information about each procedure call
 - Each call pushes a new stack frame onto the stack
 - Each return pops the stack frame off the stack
- This information *may* include local data, return address, saved variables, and arguments
- Registers used by procedure calls/returns are $\$a0, \$a1, \$a2, \$a3, \$ra$, and $\$sp$
- Stack frames must always be double word aligned

ECE3140/CS3420



Cornell University

Our Stack Frames



ECE3140/CS3420



Cornell University

Local Data Section

- You *must* include a Local Data Section in the stack frame for a procedure if:
 - The procedure requires more storage than can fit in available registers
- The Local Data Section *must* always be
 - Double word aligned
 - Have a size that is a multiple of 8

ECE3140/CS3420



Cornell University

Return Address Slot

- You *must* include a Return Address Slot in the stack frame for a procedure if:
 - The procedure calls another procedure (i.e., is a non-leaf procedure)
- The return address *must* be copied from $\$31$ ($\$ra$) to the stack in the prologue
- The return address *must* be copied from the stack to $\$31$ ($\$ra$) in the epilogue
- A pad *may* be required above the return address slot to keep the Local Data doubleword aligned.

ECE3140/CS3420



Cornell University

Saved Registers Section

- You *must* include a Saved Registers Section in the stack frame for a procedure if:
 - The procedure body uses (i.e., changes the value of) registers \$s0 to \$s7
- The values of saved registers *must* be copied to the stack in the prologue
- The values of saved registers *must* be copied from the stack in the epilogue

ECE3140/CS3420



Cornell University

Arguments Section

- You *must* include an Arguments Section in the stack frame for a procedure if:
 - The procedure calls any procedure with 1 or more arguments
- The arguments section *must* be large enough for *all* parameters of *any* called procedure
- Arguments 1-4 are passed in registers \$a0-\$a3; Arguments 5 and up are copied into the argument section
- The procedure *must not* put anything in the first four argument slots; the called procedures *may* copy the values of their first four parameters into the first four argument slots.

ECE3140/CS3420



Cornell University

Exercise

- Construct a stack frame for the following:

```
int ece314(int a,b,c; char *s, int *p) {
    int x[5];
    int aa, bb, cc;
    ...
    aa = f(a,b,c);
    ...
    bb = g(a,b,c,s,p);
    ...
    cc = h(s,p);
    ...
    return aa + bb + cc;
}
```

Assume that we need 5 saved registers (\$s0-\$s4)

ECE3140/CS3420



Cornell University

Solution

- Stack Frame Design
 - Local Storage: YES - 6 Words for x[5]
 - Pad: YES - Space for 1 word
 - Return Address: YES - Non-Leaf Routine
 - Saved Registers: YES - Space for 5 words
 - Argument Block: YES - Space for 5 words
- Stack Size = 18 words (72 bytes)

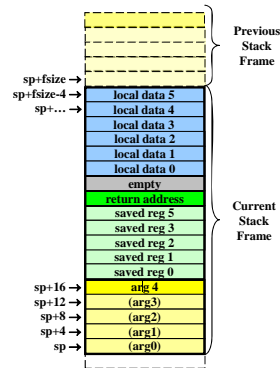
ECE3140/CS3420



Cornell University

12

Solution



ECE3140/CS3420



Cornell University

13

Those pesky argument blocks...

```
int f( int x, in y) {
    int z;
    ...
    z = g(a,b,c,d,e,f,g,h);
    ...
}
```

- Let's assume that a,b,c, ...,h are in registers \$t0, \$t1, ..., \$t7
 - Some arguments (a,b,c,d) are passed in \$a0, ... \$a3
 - Some arguments (e,f,g,h) are passed on stack!
 - Offsets \$sp+16, \$sp+20, \$sp+24, \$sp+28

ECE3140/CS3420

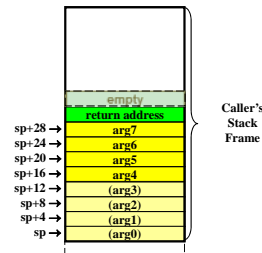


Cornell University

14

f()'s Stack Frame

```
f:
...
# start of body
...
# set up call to g()
sw $t7,28($sp)
sw $t6,24($sp)
sw $t5,20($sp)
sw $t4,16($sp)
move $a3,$t3
move $a2,$t2
move $a1,$t1
move $a0,$t0
jal g
...
# end of body
...
```



ECE3140/CS3420



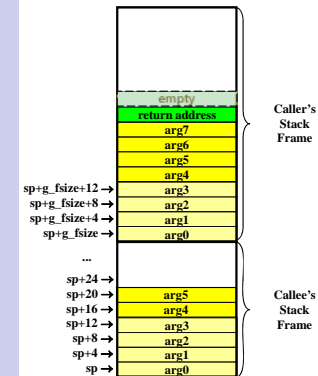
Cornell University

15

What would g() do?

```
g:
# start of prologue
addiu $sp,$sp,g_fsize
...
# save g()'s arguments
sw $a0,g_fsize+0($sp)
sw $a1,g_fsize+4($sp)
sw $a2,g_fsize+8($sp)
sw $a3,g_fsize+12($sp)

# start of body
...
# set up call to other procs
...
sw reg,20($sp)
sw reg,16($sp)
move $a3,reg
move $a2,reg
move $a1,reg
move $a0,reg
jal h
...
```



ECE3140/CS3420



Cornell University

16

Argument Blocks

- Up to 4 arguments can be passed in registers (\$a0, \$a1, \$a2, \$a3)
- Extra arguments (fifth, sixth, etc.) must be passed on the stack
 - Placed in *caller's stack frame*
- Procedures may be nested
 - If a procedure wants to save the value of its arguments, it must, in general, put them on the stack
 - Also placed in *caller's stack frame*
 - Using the caller's stack frame allows convenience of having all arguments in a contiguous block of memory

ECE3140/CS3420



Cornell University

17

But Why You Ask?

- Why don't we put arguments in the callee stack frame?
 - The answer is "Yes You Can!"
 - But this would be require a different calling convention! (*BTW, Inconsistent with our software*)
- Why "caller" arguments instead of "callee"?
 - Subroutines with variable number of parameters (e.g., printf)
 - Caller always knows the number of arguments for every subroutine it calls
 - Caller cannot always know the number of arguments
 - Must be able to calculate stack frame size at assembly/compile time

ECE3140/CS3420



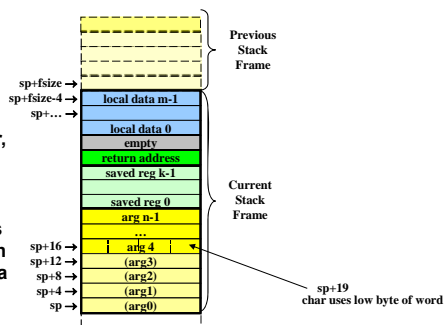
Cornell University

18

Byte-size and Halfword Arguments

- An argument always requires a full register, even if it is a byte or halfword

- An argument always requires a full word on the stack, even if it is a byte or halfword



ECE3140/CS3420



Cornell University

19

Our Register Set

Number	Name	Purpose
\$0	\$0	Always 0
\$1	\$at	The <i>Assembler Temporary</i> used by the assembler in expanding pseudo-ops.
\$2-\$3	\$v0-\$v1	These registers contain the <i>Returned Value</i> of a subroutine; if the value is 1 word only \$v0 is significant.
\$4-\$7	\$a0-\$a3	The <i>Argument registers</i> , these registers contain the first 4 argument values for a subroutine call.
\$8-\$15, \$24, \$25	\$t0-\$t9	The <i>Temporary Registers</i> .
\$16-\$23	\$s0-\$s7	The <i>Saved Registers</i> .
\$26-\$27	\$k0-\$k1	The <i>Kernel Reserved registers</i> . DO NOT USE.
\$28	\$gp	The <i>Globals Pointer</i> used for addressing static global variables. For now, ignore this.
\$29	\$sp	The <i>Stack Pointer</i> .
\$30	\$fp (or \$s8)	The <i>Frame Pointer</i> , if needed (this was discussed briefly in lecture). Programs that do not use an explicit frame pointer (e.g., everything assigned in ECE314) can use register \$30 as another saved register. Not recommended however.
\$31	\$ra	The <i>Return Address</i> in a subroutine call.

ECE3140/CS3420



Cornell University

20

Saved Registers

- Registers whose value must be preserved across a procedure call.
 - Caller expects that these registers will have the same value after the procedure call that they had before the procedure call
 - Callee must preserve the values of these registers with respect to the caller, by
 - Not changing the value of these registers, OR
 - Copying their values to the stack at the start of the procedure and copying their values from the stack at the end of the procedure

ECE3140/CS3420



Cornell University

21

Exercise

```
int ece314(int a,b,c; char *s, int *p) {
    int x[3];
    int aa, bb, cc;
    aa = f(a,b,c);
    bb = g(a,b,c,*p+1,s,p);
    cc = h(s,p);
    return aa + bb + cc;
}
```

- Suppose that this function requires 4 saved registers and 5 words of local storage. What is the size of the stack frame?
 - a) 64 bytes
 - b) 68 bytes
 - c) 72 bytes
 - d) 76 bytes
 - e) None of the above

ECE3140/CS3420



Cornell University

22

Exercise

```
g:
    ...
    li    $t0, 11
    li    $s0, 17
    jal   f

    # Next instruction to execute after the procedure call
    add   $t1,$s0,$t0
```

- What can you say about the value of \$t0 and \$s0?
 - a) They are guaranteed to be 11 and 17
 - b) The value of \$t0 is 11 and \$s0 might have changed
 - c) The value of \$s0 is 17 and \$t0 might have changed
 - d) They both might have changed
 - e) Don't know enough to say

ECE3140/CS3420



Cornell University

23

Exercise

```
g:
    ...
    # procedure body
    beq   $a0,$0,$skip
    addiu $s3,$a0,17
    sw    $s4,32($a1)
    or    $s2,$s2,$s4
    sub   $t3,$s2,$a2
loop:   ori    $a0,$t3,0
        jal   h
        addiu $s0,$v0,1
        ori   $a0,$s0,0
        jal   k
        addiu $s0,$v0,1
        beq   $s0,$a3,$loop
        xor   $s0,$s0,$s3
        ori   $a0,$s0,0
        ori   $a1,$s1,0
        jal   g
        addiu $s0,$v0,1
skip:   ...
```

- Which registers must be “saved” by g?
- a) None
 - b) \$s0, \$s2, \$s3
 - c) \$s0, \$s2, \$s3, \$s4, \$t0, \$t3
 - d) \$s0, \$s1, ..., \$s7
 - e) Not enough information to answer

ECE3140/CS3420



Cornell University

24

Endianness Review

- A 32-bit HEX value $W_1W_0X_1X_0Y_1Y_0Z_1Z_0$ is to be stored at address A
- Memory is byte addressable so this 32-bit value is stored at byte addresses
 - $A, A+1, A+2$ and $A+3$
- A Big-Endian Memory stores
 - W_1W_0 at A, X_1X_0 at $A+1, Y_1Y_0$ at $A+2, Z_1Z_0$ at $A+3$
- A Little-Endian Memory stores
 - W_1W_0 at $A+3, X_1X_0$ at $A+2, Y_1Y_0$ at $A+1, Z_1Z_0$ at A

ECE3140/CS3420



Cornell University

25

Endianness Review

- A 16-bit HEX value $Y_1Y_0Z_1Z_0$ is to be stored at address A
- Memory is byte addressable so this 32-bit value is stored at byte addresses
 - A and $A+1$
- A Big-Endian Memory stores
 - Y_1Y_0 at A, Z_1Z_0 at $A+1$
- A Little-Endian Memory stores
 - Y_1Y_0 at $A+1, Z_1Z_0$ at A

ECE3140/CS3420



Cornell University

26

Endianness Review

- An 8-bit HEX value Z_1Z_0 is to be stored at address A
- Memory is byte addressable so this 32-bit value is stored at byte address
 - A
- A Big-Endian Memory stores
 - Z_1Z_0 at A
- A Little-Endian Memory stores
 - Z_1Z_0 at A

ECE3140/CS3420



Cornell University

27

Exercise

```
li    $s0,0x00001000
li    $t0,0xBABEF00D

sw    $t0,0($s0)
lbu   $t1,0($s0)
lbu   $t2,1($s0)
lbu   $t3,2($s0)
lbu   $t4,3($s0)
```

- After execution of this sequence on a Big-Endian (Little-Endian) machine, \$t1, \$t2, \$t3, \$t4 will contain?
 - a) BA F0 BE 0D
 - b) 0D F0 BE BA
 - c) BA BE F0 0D
 - d) F0 0D BA BE
 - e) None of the above

ECE3140/CS3420



Cornell University

28

Exercise

```
li    $s0,0x00001000
li    $t0,0xDEADBEEF

sw    $t0,0($s0)
lhu   $t1,0($s0)
lhu   $t2,2($s0)
```

- After execution of this sequence on a Big-Endian (Little-Endian) machine, \$t1, \$t2 will contain?
 - a) DEAD BEEF
 - b) BEEF DEAD
 - c) DEBE ADEF
 - d) EFBE ADDE
 - e) None of the above

