CPS104 Computer Organization

Lecture 10

MIPS Assembly Language Programming
Functions, Subroutines, Methods

September 23 , 2009

Gershon Kedem

©GK Fall 2009

Administratrivia

- **Homework 2 is posted on-line Due: September 28**
- **Reading: Chapter 2.**
- **Reading: Appendix B**
  - **Available on:. . . `cps104/Handouts/spim_appendix.pdf`**

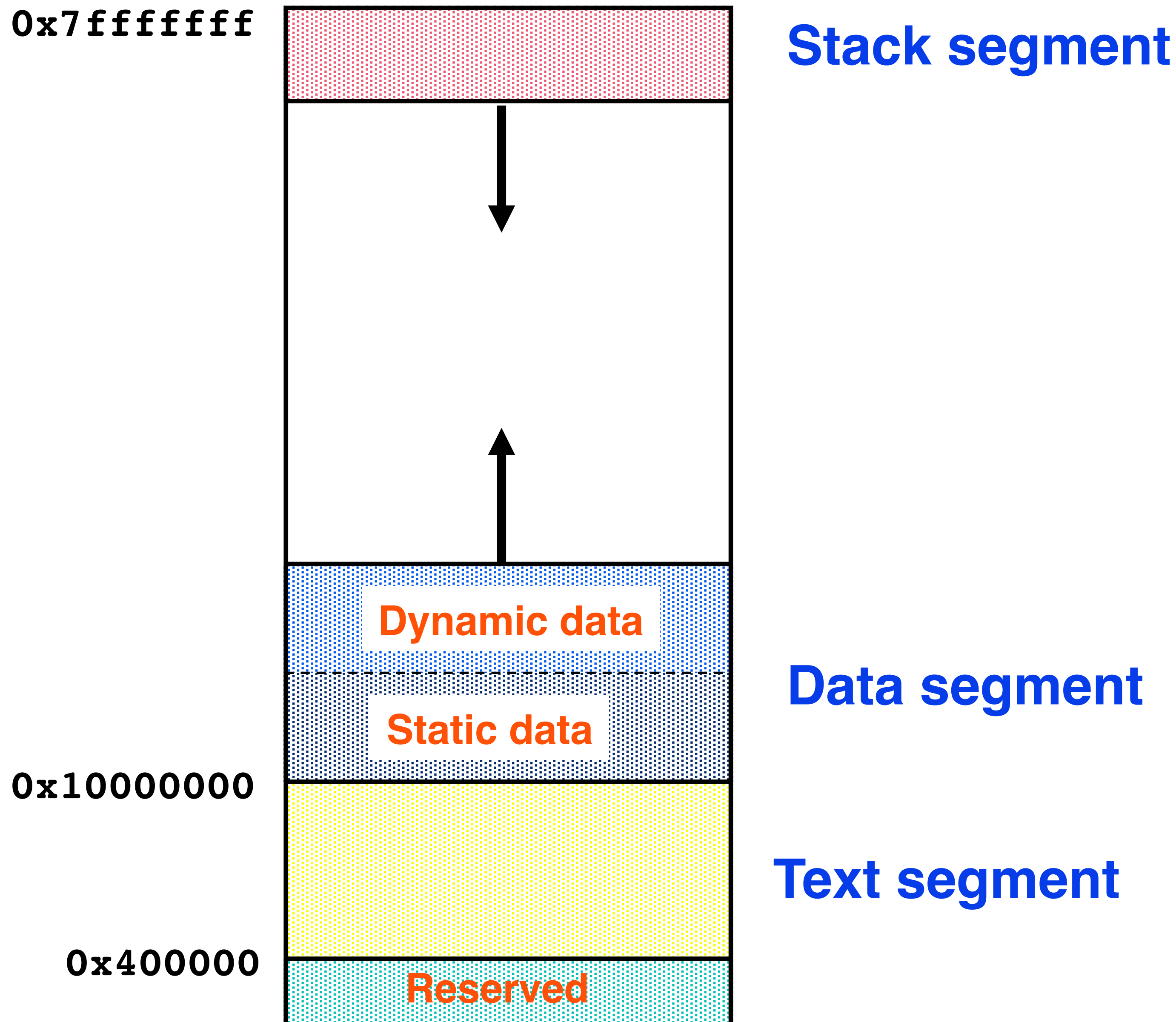
©GK Fall 2009

Review: Software conventions for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		
15	t7	
16	s0	callee saves
. . .		
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	frame pointer
31	ra	Return Address (HW)

©GK Fall 2009

Review: Memory Layout



©GK Fall 2009

Review: Example2

```
# Example for CPS 104
# Program to add together list of 9 numbers.

        .text                       # Code
        .align  2
        .globl  main
main:                               # MAIN procedure Entrance
        move    $s1, $0               #\
        la      $s0, list             # \ Initialization
        la      $s2, msg              # /
        la      $s3, list+36          #/
```


©GK Fall 2009

Example2 (cont.)

```
#                        Main code segment

again:                                  #   Begin main loop
        lw      $t6, 0(s0)              #\
        add     $s1, $s1, $t6           #/  Actual "work"
                                        #    SPIM I/O
        li      $v0, 4                  #\
        move    $a0, $s2                # >  Print a string
        syscall                         #/
        li      $v0, 1                  #\
        move    $a0, $s1                # >  Print a number
        syscall                         #/
        li      $v0, 4                  #\
        la      $a0, nln                # > Print a string (eol)
        syscall                         #/

        addiu   $s0, $s0, 4               #\ index update and
        bne     $s0, $s3, again           #/  end of loop
```


©GK Fall 2009

Example2 (cont.)

```
#                          Exit Code

         li      $v0, 10                      #\     load exit code ;
         syscall                              # >    exit
         .end     main                        #/     end of program
```

```
#                 Data Segment

   .data                                      # Start of data segment
list:   .word   35, 16, 42, 19, 55, 91, 24, 61, 53
msg:    .asciiz "The sum is "
nln:    .asciiz "\n"
```


©GK Fall 2009

Review: System call

- System call is used to communicate with the system and do simple I/O.
- Load system call code into Register `$v0`
- Load arguments (if any) into registers `$a0`, `$a1` or `$f12` (for floating point).
- do: `syscall`
- Results returned in registers `$v0` or `$f0`.

code	service	Arguments	Result	comments
1	print integer	$a0		
2	print float	$f12		
3	print double	$f12,$f13		
4	print string	$a0		(address)
5	read integer		integer in $v0	
6	read float		float in $f0	
7	read double		double in $f0	
8	read string	$a0=buffer, $a1=length		
9	sbrk	$a0=amount	address in $v0	
10	exit			

©GK Fall 2009

Details of the MIPS instruction set

- Register zero always has the value zero (even if you try to write it)
- Link instructions put the return address PC+4 into the link register
- All instructions change all 32 bits of the destination register (including `lui, lb, lh`) and all read all 32 bits of sources (`add, sub, and, or`, ...)
- Immediate arithmetic and logical instructions are extended as follows:
  - logical immediate are zero extended to 32 bits
  - arithmetic immediate are sign extended to 32 bits
- The data loaded by the instructions `lb` and `lh` are extended as follows:
  - `lbu, lhu` are zero extended
  - `lb, lh` are sign extended
- Overflow can occur in these arithmetic and logical instructions:
  - `add, sub, addi`
  - it cannot occur in: `addu, subu, addiu, and, or, xor, nor, shifts, mult, multu, div, divu`

©GK Fall 2009

Miscellaneous MIPS Instructions

- **break** — A breakpoint trap occurs, transfers control to exception handler
- **syscall** — A system trap occurs, transfers control to exception handler
- **coprocessor instrs.** — Support for floating point.
- **TLB instructions** — Support for virtual memory: discussed later
- **restore from exception** — Restores previous interrupt mask & kernel/user mode bits into status register
- **load word left/right** — Supports misaligned word loads
- **store word left/right** — Supports misaligned word stores

©GK Fall 2009

Hints to The Assembly Language Programmer

- Start by writing an equivalent high-level language (Java) program. Test it, and if needed, instrument it to print out intermediate results.
- Read the SPIM manual!
  http://kedem.cs.duke.edu/cps104/Handouts/spim.pdf
- Translate the Java code into assembler, line by line, construct by construct.
- Keep the table of common MIPS Instructions handy.
  http://kedem.cs.duke.edu/cps104/Handouts/mips_assembler.pdf
- Break down the assembler code into logical segments (I.e. I/O, array access, if-then-else) and test each one separately.
- Assign variables to registers and document it clearly!
- Document each assembler line.
- Start debugging each fragment and keep adding fragments to debugged code.

©GK Fall 2009

Hints to The Assembly Language Programmer Cont.

- Use “cut & paste” liberally to replicate functionality (I.e. input-output functions).
- Follow the register convention religiously.
- Use single-steps to debug your code on small examples.
- Print out intermediate results.
- Think about how to test your program before you write it.
- Test your program!!
- If you are not sure how to do something (I.e. Input a number) experiment by writing a separate small test program. Experiment!!
  - If you are not sure how to do something and you are stuck,

ASK for help

©GK Fall 2009

Assembly Examples: code fragments

- `If ( ... ) { ... };` **Example**

The C++ code

```
a=b=c=0;
if (i < 5) {
a = 1;
b = 2;
c = 3;
}
d = 5;
```

The Assembler code

```
      move $a0,0   # a=0
      move $a1,0   # b=0
      move $a2,0   # c=0
      bge  $s0,5,flbl # if i >=5
                      # goto flbl
      move $a0,1   # a=1
      move $a1,2   # b=2
      move $a2,3   # c=3

flbl: move $a3,5   # d=5
```


©GK Fall 2009

Assembly Examples: code fragments

- `If ( ... ){ ... } else { ...} ;` **Example**

The C++ code

```
a=b=c=0;
if (i < 5) {
a = 1;
b = 2;
c = 3;
}
else{
a = 4;
b = 5;
c = 6;
}
d = 5;
```

The Assembler code

```
       move $a0,0   # a=0
       move $a1,0   # b=0
       move $a2,0   # c=0
       bge  $s0,5,lbl1 # if i >=5
                       # goto lbl1
       move $a0,1   # a=1
       move $a1,2   # b=2
       move $a2,3   # c=3
       j    lbl2    # goto lbl2
lbl1:  move $a0,4   # a=4
       move $a1,5   # b=5
       move $a2,6   # c=6
lbl2:  move $a3,5   # d=5
```


©GK Fall 2009

Assembly Examples: code fragments

- `for(i=0; i<100; i++)sum=sum+A[i];` **Example-1:**

The C++ code

```
int a[100];
. . .
sum = 0;
for(i=0;i<100;i++)
  sum=sum+a[i];
```

The Assembler code

```
      la   $t0,a          # $t0 = &a[0]
      move $a0,0          # sum=0
      move $a1,0          # i=0
      move $a3,100        # $a3 = 100
loop: mult $a2,$a1,4      # $a2=i*4
      addu $t1,$t0,$a2    # $t1=&a[i]
      lw   $t2,0($t1)     # $t2=a[i]
      add  $a0,$a0,$t2    # sum=sum+a[i]
      addi $a1,$a1,1      # i++
      blt  $a1,$a3,loop   # if i<100
                          # goto loop
```


©GK Fall 2009

Assembly Examples: code fragments

- `for(i=0; i<100; i++)sum=sum+A[i];` Example-2:

The C++ code

```
int a[100];
int* p=&a[0];
sum = 0;
for(i=0;i<100;i++)
   {sum=sum+*p;
    p++;
   }
```

The Assembler code

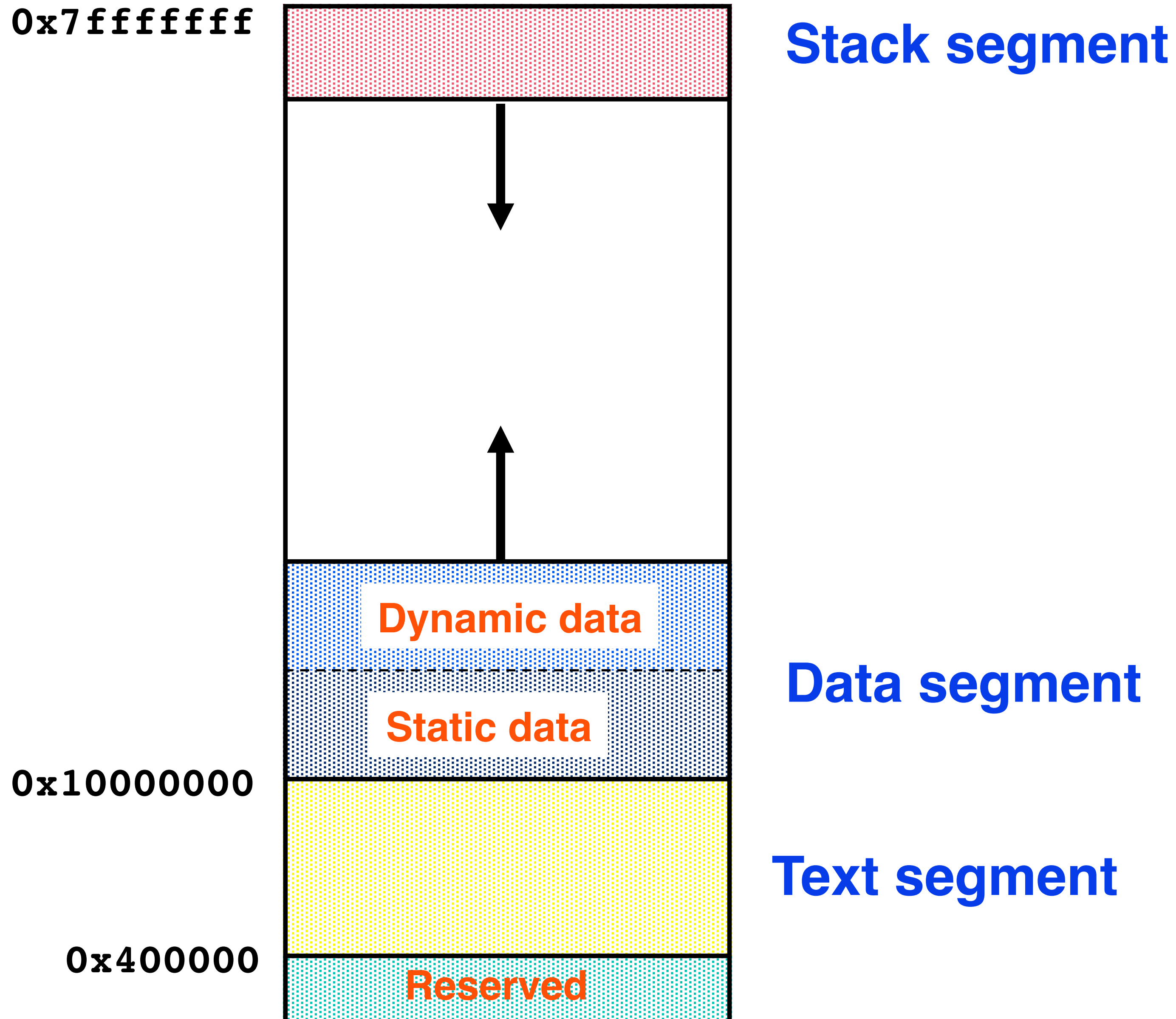
```
        la    $t0,a          # p = &a[0]
        move  $a0,0          # sum=0
        move  $a1,0          # i=0
        move  $a3,100        # $a3 = 100
loop:   lw    $t2,0($t0)     # $t2=a[i]
        add   $a0,$a0,$t2    # sum=sum+a[i]
        addi  $a1,$ai,1      # i++
        addui  $t0,4         # p++
        blt   $a1,$a3,loop   # if i<100
                             # goto loop
```


©GK Fall 2009

Functions Subroutines & Methods

©GK Fall 2009

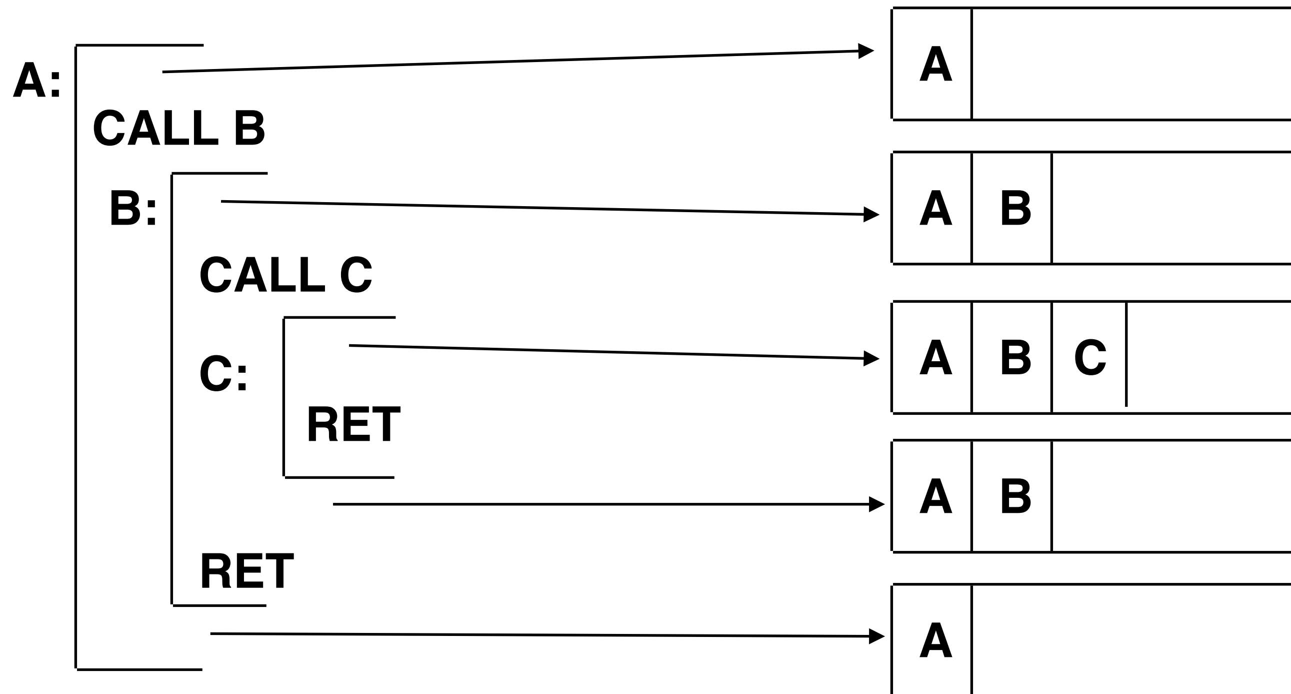
Memory Layout



©GK Fall 2009

Calls: Why Are Stacks So Great?

Stacking of Subroutine Calls & Returns and Environments:



Some machines provide a memory stack as part of the architecture (e.g., VAX)

Sometimes stacks are implemented via software convention (e.g., MIPS)

©GK Fall 2009

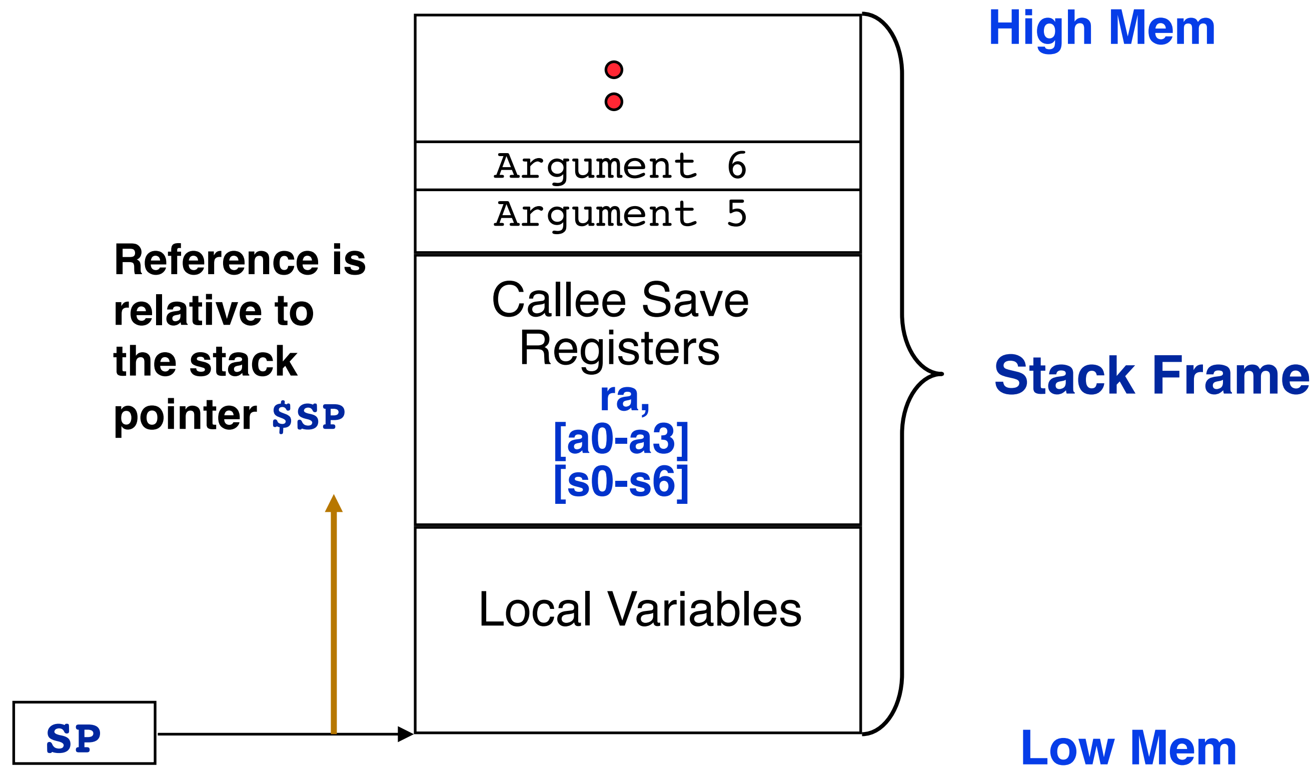
Procedure Call (Stack) Frame

- **Procedures use a frame in the stack to:**
  - **Hold values passed to procedures as arguments.**
  - **Save registers that a procedure may modify, but which the procedure's caller does not want changed. (ex: `$s0-$s7`)**
  - **Save the procedure return address (`$ra`),**
  - **provide space for local variables (variables with local scope)**
  - **Evaluate complex expressions.**

- **There is a special registers the `$sp` that are used as special data reference**
  - **The stack pointer $sp points to the top of the stack.**

©GK Fall 2009

Call-Return Linkage: Stack Frames



- Many variations on stacks possible (up/down, last pushed / next)
- Block structured languages contain link to lexically enclosing frame
- Compilers normally keep scalar variables in registers, not memory!

©GK Fall 2009

MIPS/GCC Procedure Calling Conventions

Calling Procedure:

- **Step-1: Save caller-saved registers**
  - **Save registers `$t0-$t9` if they contain live values at the call site.**
- **Step-2: Pass the arguments:**
  - **The first four arguments are passed in registers `$a0-$a3`**
  - **Remaining arguments are pushed into the stack**
    - **(in reversed order `arg5` is at the top of the stack).**
- **Step-3: Execute a `jal` instruction.**

©GK Fall 2009

MIPS/GCC Procedure Calling Conventions (cont.)

Called Routine

- Step-1: Establish stack frame.
  - Subtract the frame size from the stack pointer.

```
subiu $sp,  $sp,  <frame-size>
```

  - Typically, minimum frame size is 32 bytes (8 words).

- Step-2: Save callee saved registers in the frame.
  - Register `$ra` is saved if routine makes a call.
  - Registers `$a0-$a3` are saved if they are changed.
  - Registers `$s0-$s7` are saved if they are used.

©GK Fall 2009

MIPS/GCC Procedure Calling Conventions (cont.)

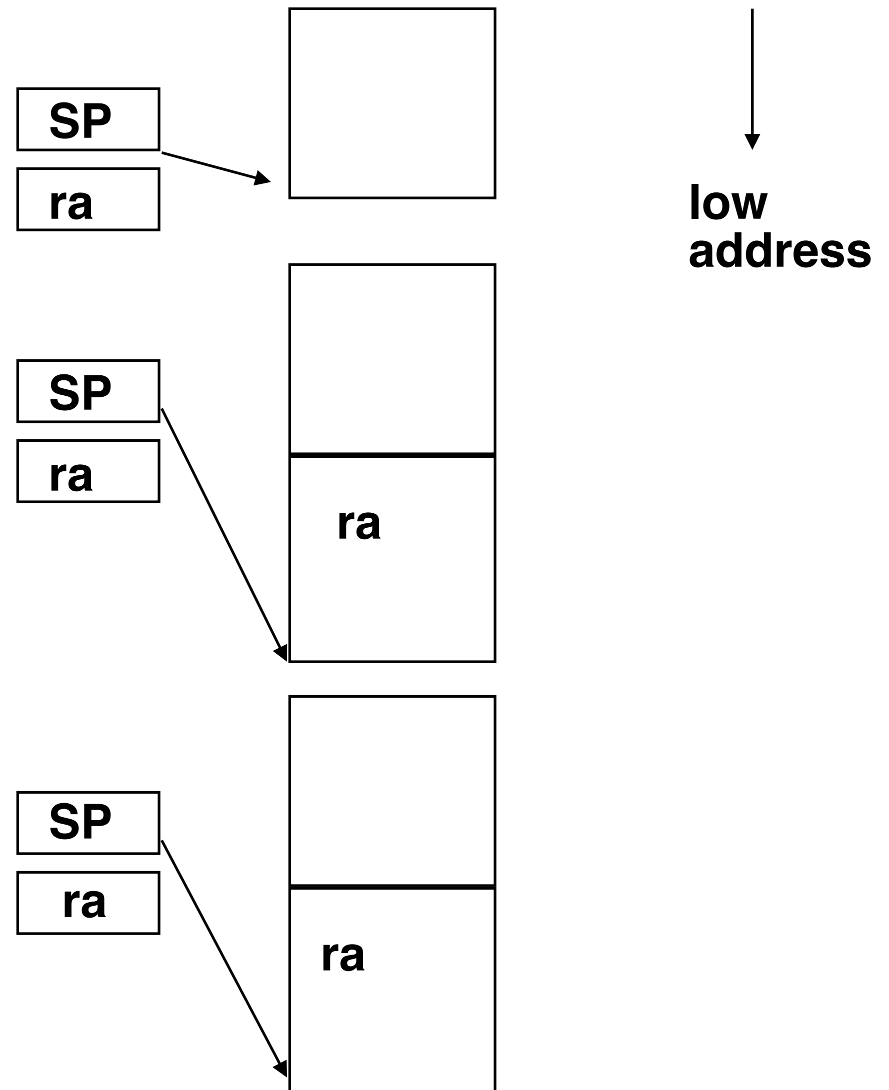
On return from a call

- Step-1: Put returned values in registers `$v0, [$v1].`
  (if values are returned)
- Step-2: Restore callee-saved registers.
  - Restore `$sp` and other saved registers.
    ```
    [$ra,$a0-$a3,$s0-$s7]
    ```
- Step-3: Pop the stack
  - Add the frame size to `$sp`.
    ```
    addiu $sp, $sp, <frame-size>
    ```
- Step-4: Return
  - Jump to the address in `$ra`.
    ```
    jr        $ra
    ```


©GK Fall 2009

MIPS / GCC Calling Conventions

```
fact:
  subiu $sp, $sp, 32
  sw    $ra, 20($sp)
  addiu $fp,$sp,28
. . .
  sw    $a0, 0($fp)
...
  lw    $ra, 20($sp)
        addiu   $sp,$sp,32
  jr    $ra
```



First four arguments are passed in registers!

©GK Fall 2009

Example2B

```
# Example for CPS 104
# Program to add together list of 9 numbers.

        .text                       # Code
        .align  2
        .globl  main
main:                               # MAIN procedure Entrance
        subu    $sp, 40             #\ Push the stack
        sw      $ra, 36($sp)        # \ Save return address
        sw      $s3, 32($sp)        #  \
        sw      $s2, 28($sp)        #   > Entry Housekeeping
        sw      $s1, 24($sp)        #  / save registers on stack
        sw      $s0, 20($sp)        # /
        move    $v0, $0             #/ initialize exit code to 0

        move    $s1, $0             #\
        la      $s0, list           # \ Initialization
        la      $s2, msg            # /
        la      $s3, list+36        #/
```


©GK Fall 2009

Example2B (cont.)

```
#                         Main code segment

again:                                      #    Begin main loop
        lw       $t6, 0(s0)                 #\
        add      $s1, $s1, $t6              #/   Actual "work"
                                            #     SPIM I/O
        li       $v0, 4                     #\
        move     $a0, $s2                   # >   Print a string
        syscall                             #/
        li       $v0, 1                     #\
        move     $a0, $s1                   # >   Print a number
        syscall                             #/
        li       $v0, 4                     #\
        la       $a0, nln                   # > Print a string (eol)
        syscall                             #/

        addiu    $s0, $s0, 4                  #\ index update and
        bne      $s0, $s3, again              #/   end of loop
```


©GK Fall 2009

Example2B (cont.)

```
#                           Exit Code

       move     $v0, $0          #\
        lw      $s0, 20($sp)     # \
        lw      $s1, 24($sp)     #  \
        lw      $s2, 28($sp)     #   \ Closing Housekeeping
        lw      $s3, 32($sp)     #   /   restore registers
        lw      $ra, 36($sp)     #  / load return address
        addu    $sp, 40          # / Pop the stack
        jr      $ra              #/    exit(0) ;
        .end    main             #  end of program

#               Data Segment

        .data                    # Start of data segment
list:   .word   35, 16, 42, 19, 55, 91, 24, 61, 53
msg:    .asciiz "The sum is "
nln:    .asciiz "\n"
```


©GK Fall 2009