

# Procedures

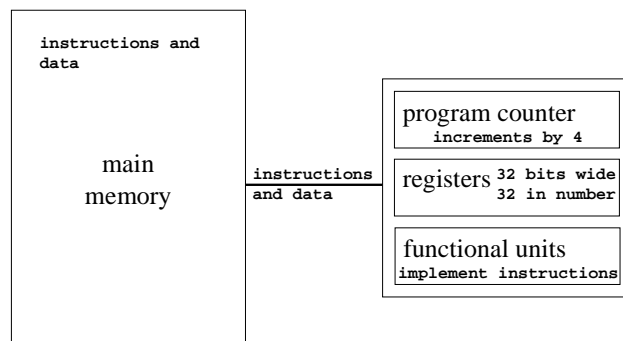
CSE 410, Spring 2007  
Computer Systems

<http://www.cs.washington.edu/410>

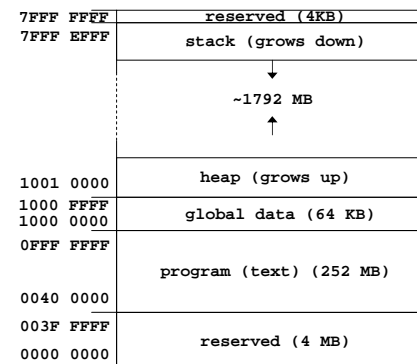
# Readings and References

- Reading
  - » Section 2.7, Supporting Procedures in Computer Hardware
  - » Section A.5, Memory Usage
  - » Section A.6, Procedure Call Convention
  - » Section 3.2, Signed and Unsigned Numbers, P&H
    - another presentation of binary, hex, and decimal
    - ignore signed numbers for now, we will cover them next week
- Other References
  - » MIPSpro Assembly Language Programmer's Guide, document number 007-2418-006, Silicon Graphics, 2003
    - copy linked from our web site on otherlinks page

# Instructions and Data flow



# Layout of program memory



*Not to Scale!*

## Why use procedures?

- So far, our program is just one long run of instructions
- We can do a lot this way, but the program rapidly gets too large to handle easily
- Procedures allow the programmer to organize the code into logical units

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

5

## What does a procedure do for us?

- A procedure provides a well defined and reusable interface to a particular capability
  - » entry, exit, parameters clearly identified
- Reduces the level of detail the programmer needs to know to accomplish a task
- The internals of a function can be ignored
  - » messy details can be hidden from innocent eyes
  - » internals can change without affecting caller

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

6

## How do you use a procedure?

1. set up parameters
2. transfer to procedure
3. acquire storage resources
4. do the desired function
5. make result available to caller
6. return storage resources
7. return to point of call

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

7

## Calling conventions

- The details of how you implement the steps for using a procedure are governed by the *calling conventions* being used
- There is much variation in conventions
  - » which causes much programmer pain
- Understand the calling conventions of the system you are writing for
  - » o32, n32, n64, P&H, cse410, ...

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

8

## 1. Set up parameters

- The registers are one obvious place to put parameters for a procedure to read
  - » very fast and easily referenced
- Many procedures have 4 or less arguments
  - » \$a0, \$a1, \$a2, \$a3 are used for arguments
- ... but some procedures have more
  - » we don't want to use up all the registers
  - » so we use memory to store the rest

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

9

## The Stack

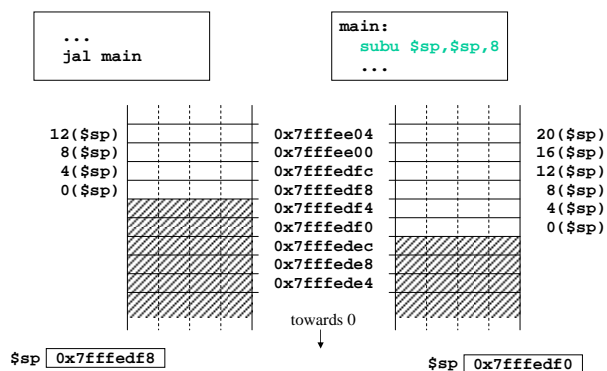
- Stack pointer (\$sp) points to the “top” value on the stack (ie, the lowest address in use)
- There are no “push” or “pop” instructions
  - » we adjust the stack pointer directly
- stack grows downward towards zero
  - » `subu $sp, $sp, xx` : make room for more data
  - » `addu $sp, $sp, xx` : release space on the stack
  - » note that both `subu` and `addu` become `addiu`

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

10

## Dynamic storage on the stack

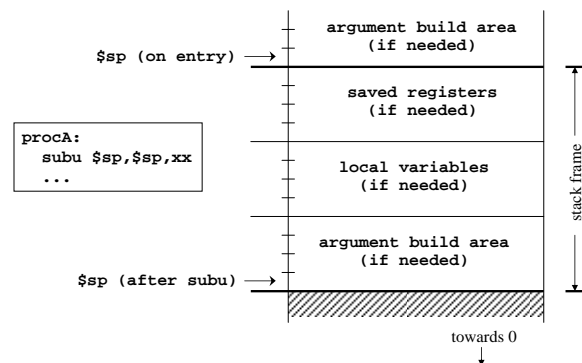


4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

11

## Layout of stack frame



4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

12

## Argument build area

- Some calling conventions require that caller reserve stack space for all arguments
  - » 16 bytes (4 words) left empty to mirror \$a0-\$a3
- Other calling conventions require that caller reserve stack space only for arguments that do not fit in \$a0 - \$a3
  - » so argument build area is only present if some arguments didn't fit in 4 registers

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

13

## Agreement

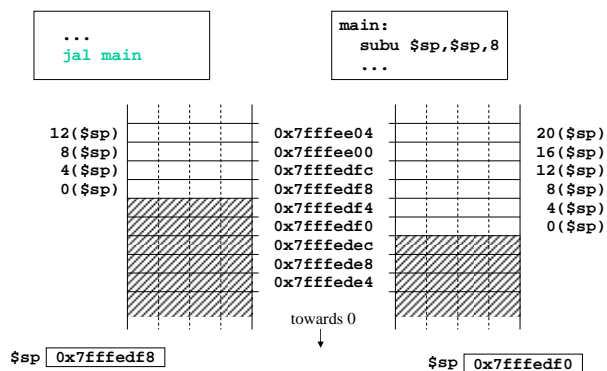
- A procedure and all of the programs that call it must agree on the calling convention
- This is one reason why changing the calling convention for system libraries is a big deal
- We will use
  - » caller reserves stack space for all arguments
  - » 16 bytes (4 words) left empty to mirror \$a0-\$a3

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

14

## 2. Transfer to procedure



4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

15

## Jump and link

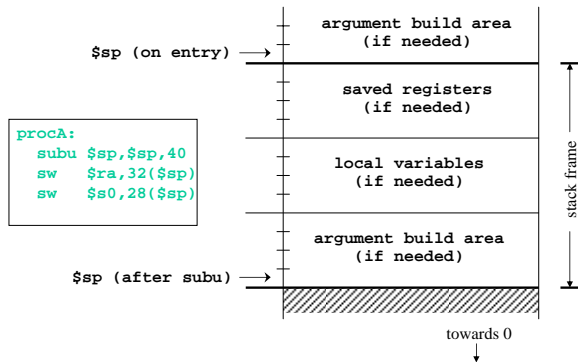
- Jump
  - » can take you anywhere within the currently active 256 MB segment
- Link
  - » store return address in \$ra
  - » note: this overwrites current value of \$ra

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

16

### 3. Acquire storage resources



4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

17

### 3a. Saved registers

- There is only one set of registers
  - » If called procedure unexpectedly overwrites them, caller will be surprised and distressed
- Another agreement
  - » called procedure can change \$a0-\$a3, \$v0-\$v1, \$t0-\$t9 without restoring original values
  - » called procedure must save and restore value of any other register it wants to use

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

18

### Register numbers and names

| <i>number</i> | <i>name</i>            | <i>usage</i>                                     |
|---------------|------------------------|--|
| 0             | <b>zero</b>            | always returns 0                                 |
| 1             | <b>at</b>              | reserved for use as assembler temporary          |
| 2-3           | <b>v0, v1</b>          | values returned by procedures                    |
| 4-7           | <b>a0-a3</b>           | first few procedure arguments                    |
| 8-15, 24, 25  | <b>t0-t9</b>           | temps - can use without saving                   |
| 16-23         | <b>s0-s7</b>           | temps - must save before using                   |
| 26, 27        | <b>k0, k1</b>          | reserved for kernel use - may change at any time |
| 28            | <b>gp</b>              | global pointer                                   |
| 29            | <b>sp</b>              | stack pointer                                    |
| 30            | <b>fp</b> or <b>s8</b> | frame pointer                                    |
| 31            | <b>ra</b>              | return address from procedure                    |

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

19

### 3b. Local variables

- If the called procedure needs to store values in memory while it is working, space must be reserved on the stack for them
- Debugging note
  - » compiler can often optimize so that all variables fit in registers and are never stored in memory
  - » so a memory dump may not contain all values
  - » use switches to turn off optimization (but ...)

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

20

### 3c. Argument build area

- Our convention is
  - » caller reserves stack space for all arguments
  - » 16 bytes (4 words) left empty to mirror `$a0-$a3`
- If your procedure does more than one call to other procedures, then ...
  - » the argument build area must be large enough for the largest set of arguments

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

21

### Using the stack pointer

- Adjust it once on entry, once on exit
  - » Initial adjustment should include all the space you will need in this procedure
- Remember that a word is 4 bytes
  - » so expect to see references like `8($SP)`, `20($SP)`
- Keep stack pointer double word aligned
  - » adjust by multiples of 8

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

22

### 4. Do the desired function

- You have saved the values of the registers that must be preserved across the call
- The arguments are in `$a0 - $a3` or on the stack
- The stack pointer points to the end of your stack frame
- Let 'er rip
  - » signal processing, image filter, encryption, ...

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

23

### 5. Make result available to caller

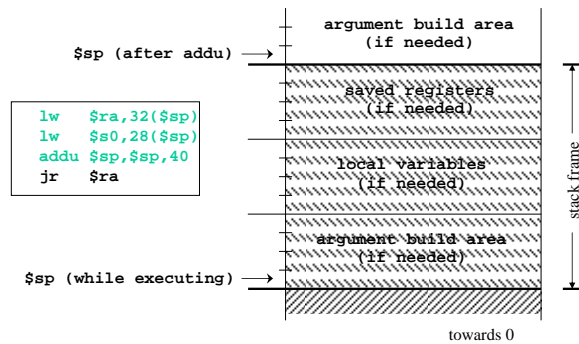
- Registers `$v0` and `$v1` are available for this
- Most procedures put a 32-bit value in `$v0`
- Returning the address of a variable?
  - » be very careful!
  - » your portion of the stack is invalid as soon as you return
  - » the object must be allocated in caller's part of stack (or somewhere further back) or globally allocated

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

24

## 6. Return storage resources



4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

25

## 7. Return to point of call

- Jump through register
- The address of the instruction following the jump and link was put in \$ra when we were called (the “link” in jump and link)
- We have carefully preserved \$ra while the procedure was executing
- So, “`jr $ra`” takes us right back to caller

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

26

## CSE 410 Calling Conventions

- Argument build area
  - » caller reserves stack space for all arguments
  - » 16 bytes (4 words) left empty to mirror \$a0-\$a3
- Called procedure adjusts stack pointer once on entry, once on exit, in units of 8 bytes
- Register usage in functions
  - » not required to save and restore \$t0-\$t9, \$a0-\$a3
  - » must save and restore \$s0-\$s8, \$ra if changed
  - » function results returned in \$v0, \$v1

4/3/2007

cse410-06-procedures-a © 2006-07 Perkins, DW Johnson and University of Washington

27