

Procedures

CSE 410 - Computer Systems
October 8, 2001

Readings and References

- Reading
 - Section 4.2, Signed and Unsigned Numbers, P&H
 - another presentation of binary, hex, and decimal
 - ignore signed numbers for now, we will cover them next week
 - Sections 3.6, A5, A6, P&H
 - note error in figure 3.13 - \$a0-\$a3 are not preserved
- Other References
 - MIPSpro Assembly Language Programmer's Guide, document number 007-2418-001, Silicon Graphics, 1994

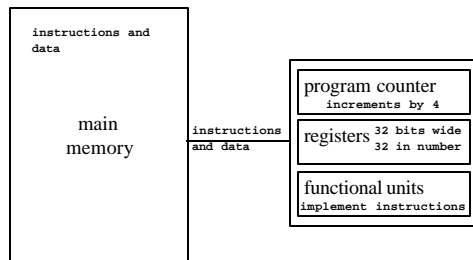
<http://techpubs.sgi.com:80/library/tpl/cgi-bin/browse.cgi?coll=0530&db=bks&ph=/SGI_Developer>

8-Oct-2001

CSE 410 - Procedures

2

Instructions and Data flow

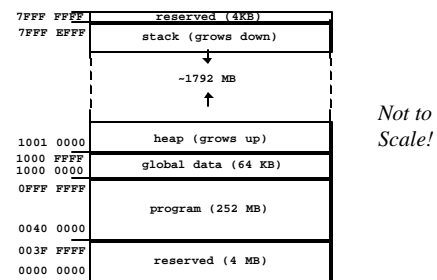


8-Oct-2001

CSE 410 - Procedures

3

Layout of program memory



8-Oct-2001

CSE 410 - Procedures

4

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

8-Oct-2001

CSE 410 - Procedures

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

8-Oct-2001

CSE 410 - Procedures

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

8-Oct-2001

CSE 410 - Procedures

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, ...

8-Oct-2001

CSE 410 - Procedures

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

8-Oct-2001

CSE 410 - Procedures

9

The Stack

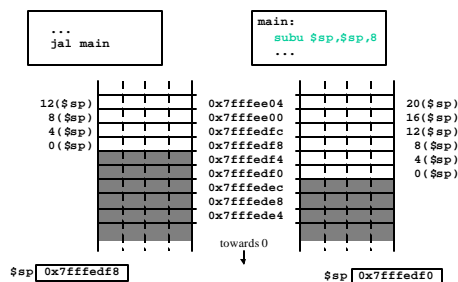
- Stack pointer (\$sp) points to the “top” value on the stack (ie, the lowest address in use)
- There are no “push” or “pop” instructions
 - we adjust the stack pointer directly
- stack grows downward towards zero
 - `subu $sp, $sp, xx` : make room for more data
 - `addu $sp, $sp, xx` : release space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

8-Oct-2001

CSE 410 - Procedures

10

Dynamic storage on the stack

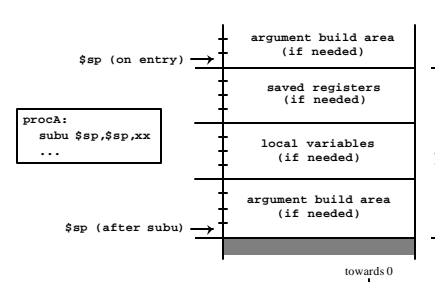


8-Oct-2001

CSE 410 - Procedures

11

Layout of stack frame



8-Oct-2001

CSE 410 - Procedures

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

8-Oct-2001 CSE 410 - Procedures 13

- 8-Oct-2001 CSE 410 - Procedures 13

Agreement

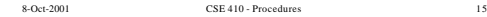
- A procedure and all of the programs that call it must agree on the calling convention
- This is one reason why changing the calling convention for system libraries is a big deal
- We will use
 - caller reserves stack space for all arguments
 - 16 bytes (4 words) left empty to mirror `$a0-$a3`

8-Oct-2001 CSE 410 - Procedures 14

- 8-Oct-2001 CSE 410 - Procedures 14

2. Transfer to procedure

The diagram illustrates the transfer to a procedure. It shows two code snippets: `jal main` and `main: subu $sp,$sp,8`. Below these, two memory stacks are shown. The left stack, starting at `12($sp)`, contains values `8($sp)`, `0($sp)`, and a series of `0x7ffedf8` values. The right stack, starting at `20($sp)`, contains values `16($sp)`, `12($sp)`, `8($sp)`, `4($sp)`, `0($sp)`, and a series of `0x7ffede4` values. An arrow labeled "towards 0" points from the `0x7ffede4` values towards the `0x7ffedf8` values, indicating the direction of stack growth or return path.



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

8-Oct-2001 CSE 410 - Procedures 16

- 8-Oct-2001 CSE 410 - Procedures 16

3. Acquire storage resources

Diagram illustrating the stack frame layout and the process of acquiring storage resources.

The stack grows downwards (towards 0).

The stack frame is divided into sections:

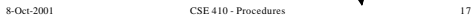
- argument build area (if needed)
- saved registers (if needed)
- local variables (if needed)
- argument build area (if needed)

The stack pointer (\$sp) is shown at the top (on entry) and after the `subu` instruction, it has moved down to the bottom of the stack frame.

The stack frame is labeled "stackframe" on the right.

The stack grows "towards 0".

```
procA:
subu $sp,$sp,40
sw   $ra,32($sp)
sw   $a0,28($sp)
```



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

8-Oct-2001 CSE 410 - Procedures 18

- 8-Oct-2001 CSE 410 - Procedures 18

Register numbers and names

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

8-Oct-2001

CSE 410 - Procedures

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 ...)

8-Oct-2001

CSE 410 - Procedures

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

8-Oct-2001

CSE 410 - Procedures

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

8-Oct-2001

CSE 410 - Procedures

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

8-Oct-2001

CSE 410 - Procedures

23

5. Make result available to caller

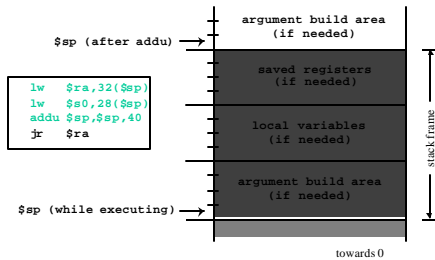
- Registers **\$v0** and **\$v1** are available for this
- Most procedures put a 32-bit value in **\$v0**
- Returning the address of a variable?
 - be very careful!
 - your portion of the stack is invalid as soon as you return
 - the object must be allocated in ancestor's part of stack or globally allocated

8-Oct-2001

CSE 410 - Procedures

24

6. Return storage resources



8-Oct-2001

CSE 410 - Procedures

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

8-Oct-2001

CSE 410 - Procedures

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
- Registers
 - 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

8-Oct-2001

CSE 410 - Procedures

27