

Working with memory

Lecture B11



Lecture notes section B11

2002-01-24

CSE1303 Part B lecture notes

1

Last time

- Simple MIPS programs
 - arithmetic
 - string handling
- Input/Output
 - using system calls
- Global variables
 - stored in data segment
- Assembler directives

2002-01-24

CSE1303 Part B lecture notes

2

In this lecture

- Memory diagrams
- System stack
- Local variables
- Addressing modes

2002-01-24

CSE1303 Part B lecture notes

3

Memory diagrams

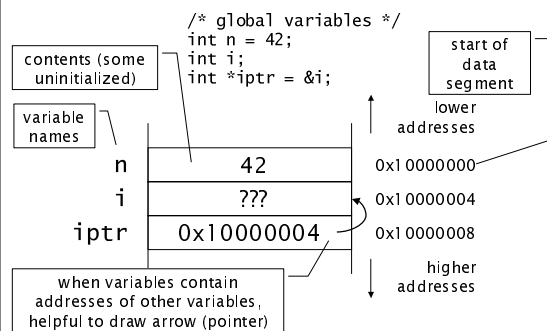
- Show memory allocated to variables
 - addresses
 - contents
 - variable names
- Not so helpful for global variables
 - every variable has a label to identify it
- Essential for local variables
 - since they are referred to without names in MIPS assembly language

2002-01-24

CSE1303 Part B lecture notes

4

Memory diagrams



2002-01-24

CSE1303 Part B lecture notes

5

Local variables

- Properties of local variables
 - accessible only within function
 - may have more than one variable with same name (in different functions)
 - may have more than one version of the same function's variables existing (recursion)
- Properties of data segment
 - accessible from all of program
 - all labels must be different
 - each location can hold only one discrete value
- Data segment is not suited to storing local variables

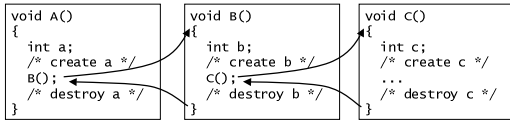
2002-01-24

CSE1303 Part B lecture notes

6

Local variables

- Properties of local variables
 - must be allocated at function entry
 - must be destroyed at function exit
 - other functions may be called in between, with same rules



2002-01-24

CSE1303 Part B lecture notes

7

Local variables

- Properties of local variables
 - allocation/destruction obeys LIFO (last in, first out) principle
 - like stack data structure
 - stack is ideal data structure for storing local variables
 - allocate a variable by pushing it on the stack
 - destroy a variable by popping it off the stack
 - stack also helpful for storing other function information, for same reason
 - saving registers
 - storing return address
 - passing function arguments

2002-01-24

CSE1303 Part B lecture notes

8

System stack

- Function call/return mechanism is very widespread among computers
- Most computers provide a stack in memory for programs to use
 - called system stack or runtime stack or stack
 - in MIPS, stack resides in its own segment of memory
 - stack segment, to address 0x7FFFFFFF
 - stack is initialized by operating system
 - user programs push/pop stack as needed
 - instruction set provides operations for doing this

2002-01-24

CSE1303 Part B lecture notes

9

System stack

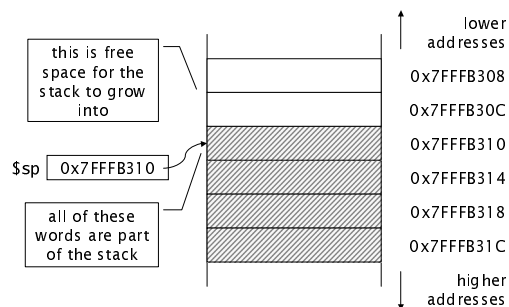
- Register \$sp (stack pointer) indicates top of stack
 - contains address of word of memory at top of stack (with lowest address)

2002-01-24

CSE1303 Part B lecture notes

10

System stack

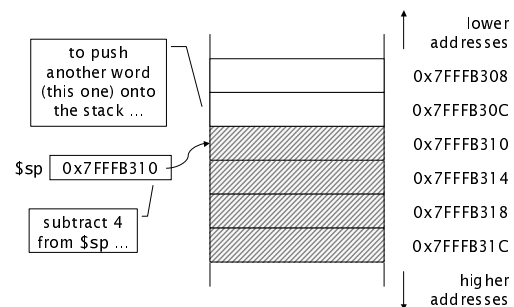


2002-01-24

CSE1303 Part B lecture notes

11

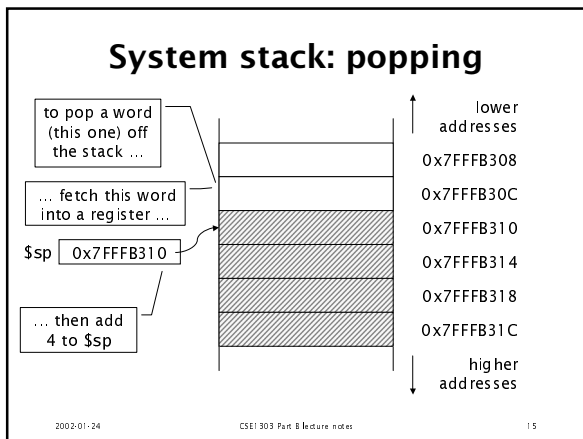
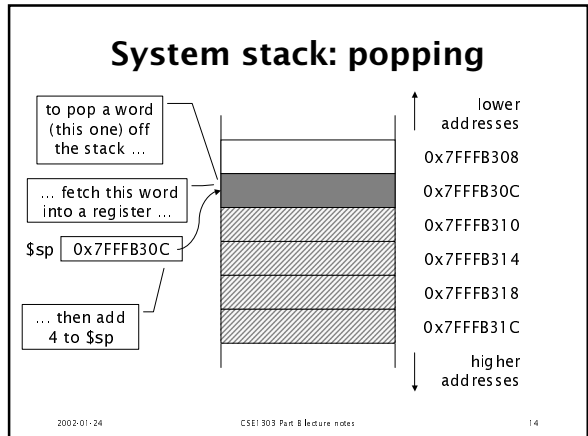
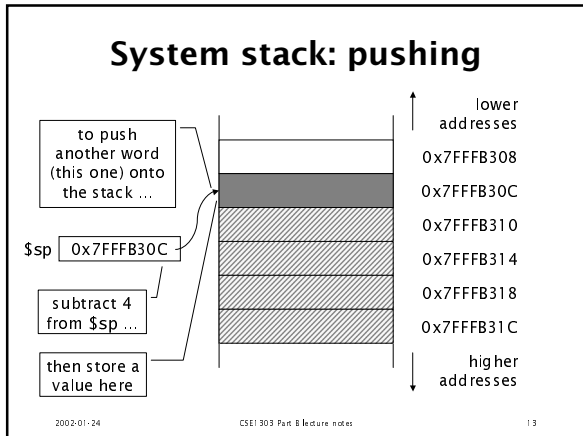
System stack: pushing



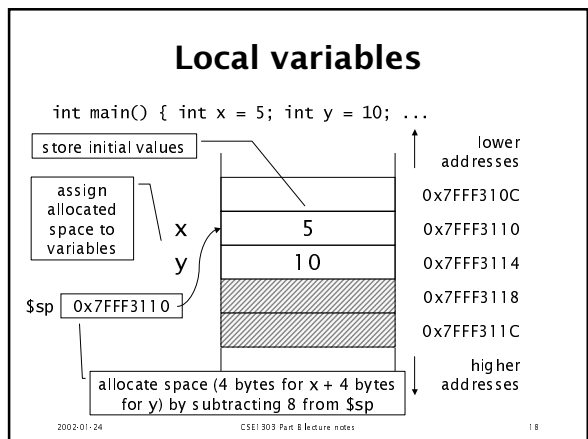
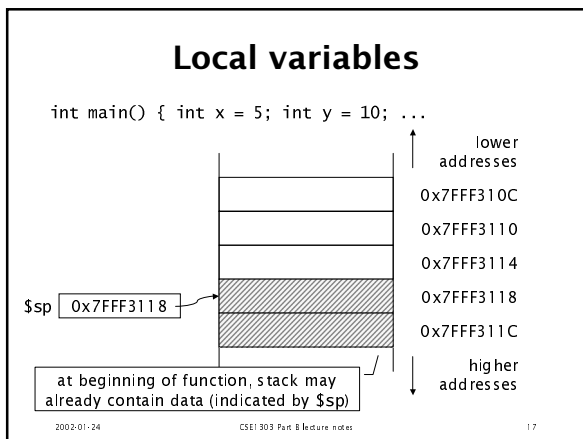
2002-01-24

CSE1303 Part B lecture notes

12



- ### Local variables
- At beginning of function
 - allocate variables by pushing necessary space onto stack (subtract n from $\$sp$)
 - initialize space by storing values in newly allocated space
 - During function
 - use variables using load/store instructions
 - At end of function
 - destroy variables by popping allocated space from stack (add n to $\$sp$)
 - not necessary on exit from main since program is ending
- 2002-01-24 CSE1303 Part B lecture notes 16



Local variables

store initial values

x 5 0x7FFF3110

y 10 0x7FFF3114

can't refer to location via label because labels are compile-time, not run-time

can't refer to location via address because stack may not be same depth every time

2002-01-24 CSE1 303 Part B lecture notes 19

Local variables

store y = 10 at address \$sp + 4 (0x7FFF3114)

store x = 5 at address \$sp + 0 (0x7FFF3110)

x 5 0x7FFF3110

y 10 0x7FFF3114

\$sp 0x7FFF3110

can use stack pointer, since variables are located relative to top of stack

2002-01-24 CSE1 303 Part B lecture notes 20

Addressing modes

- Address calculation like $\$sp + 4$ needs to be computed at run time
 - because $\$sp$'s value is not known at compile time
- register + constant is a very common address computation
 - MIPS provides method of doing this in one instruction

2002-01-24 CSE1 303 Part B lecture notes 21

Addressing modes

sw \$src, const(\$reg)

this syntax means "use the address computed by adding the address to the current contents of \$reg" (i.e., $\$reg + const$)

const may be any label or signed number or expression known at compile time, including 0

\$reg may be any general-purpose register, including \$0

2002-01-24 CSE1 303 Part B lecture notes 22

Addressing modes

| | |
|--------------------|-------------------------------------|
| sw \$t0, 4(\$sp) | address is (\$sp + 4) |
| sw \$t0, -4(\$fp) | address is (\$fp - 4) |
| la \$a0, 0(\$sp) | } address is (\$sp + 0) |
| la \$a0, (\$sp) | |
| sb \$t0, arr(\$t0) | address is (\$t0 + address of arr) |
| lbu \$a0, var(\$0) | } address is (\$0 + address of var) |
| lbu \$a0, var | |

2002-01-24 CSE1 303 Part B lecture notes 23

Frame pointer

- Can access local variables relative to stack pointer ($\$sp$), but ...
- Can be problematic when passing arguments to functions
 - stack pointer moves to accommodate arguments
 - relative locations of local variables change

2002-01-24 CSE1 303 Part B lecture notes 24

Frame pointer

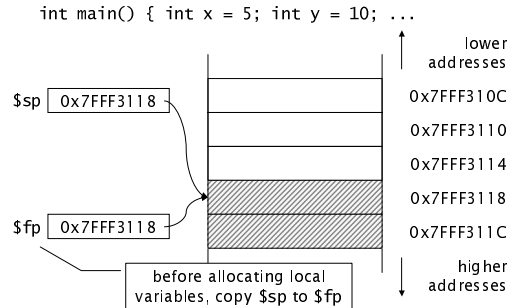
- Better to access local variables relative to saved copy of stack pointer
 - copy made before subtracting from \$sp to allocate local variables
- Saved copy stored in register \$fp (frame pointer)
 - local variables accessed relative to \$fp

2002-01-24

CSE1 303 Part B lecture notes

25

Local variables

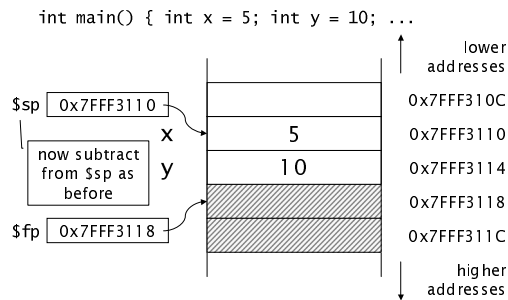


2002-01-24

CSE1 303 Part B lecture notes

26

Local variables

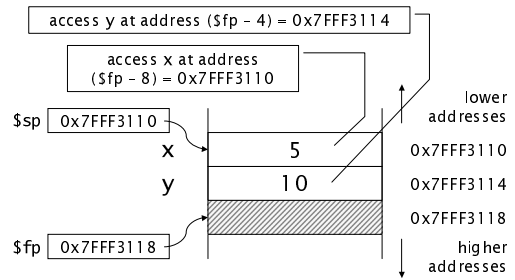


2002-01-24

CSE1 303 Part B lecture notes

27

Local variables



2002-01-24

CSE1 303 Part B lecture notes

28

```
#include <stdio.h>
#include <stdlib.h>

/* A global variable. */
int g = 123;

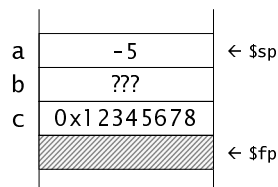
int main()
{
    /* Three local variables. */
    int a = -5;
    int b;
    int c = 0x12345678;

    /* Do some arithmetic. */
    b = g + a;

    /* Do some more arithmetic. */
    printf("%d", c - a);

    exit(0);
}
```

g is a global variable and is stored in data segment, not on stack

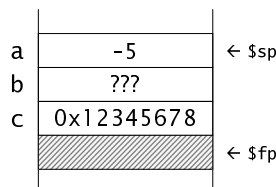


2002-01-24

CSE1 303 Part B lecture notes

29

a is at -12(\$fp)
b is at -8(\$fp)
c is at -4(\$fp)



```
.data
# g is global, allocate
# in data segment
g: .word 123

.text
main:
# Copy $sp into $fp.
move $fp, $sp

# Allocate 12 bytes of
# local variables.
subu $sp, $sp, 12

# Initialize local
# variables.
li $t0, -5 # a
sw $t0, -12($fp)

li $t0, 0x12345678 # c
sw $t0, -4($fp)

# ... rest of program
# follows next slide ...
```

2002-01-24

CSE1 303 Part B lecture notes

30

```

#include <stdio.h>
#include <stdlib.h>

/* A global variable. */
int g = 123;

int main()
{
    /* Three local variables. */
    int a = -5;
    int b;
    int c = 0x12345678;

    /* Do some arithmetic. */
    b = g + a;

    /* Do some more arithmetic. */
    printf("%d", c - a);

    exit(0);
}

```

```

.data
# g is global, allocate
# in data segment
g: .word 123

.text
main: # copy $sp into $fp.
      move $fp, $sp

      # Allocate 12 bytes of
      # local variables.
      subu $sp, $sp, 12

      # Initialize local
      # variables.
      li $t0, -5          # a
      sw $t0, -12($fp)

      li $t0, 0x12345678 # c
      sw $t0, -4($fp)

      # ... rest of program
      # follows next slide ...

```

2002-01-24 CSE1303 Part B lecture notes 31

```

#include <stdio.h>
#include <stdlib.h>

/* A global variable. */
int g = 123;

int main()
{
    /* Three local variables. */
    int a = -5;
    int b;
    int c = 0x12345678;

    /* Do some arithmetic. */
    b = g + a;

    /* Do some more arithmetic. */
    printf("%d", c - a);

    exit(0);
}

```

```

# ... here is the rest
# of the MIPS code ...

# Calculate g + a.
lw $t0, g
lw $t1, -12($fp) # a
add $t0, $t0, $t1 # a
# Store in b.
sw $t0, -8($fp) # b

li $v0, 1 # Print int.
# Calculate c - a.
lw $t0, -4($fp) # c
lw $t1, -12($fp) # a
sub $a0, $t0, $t1
syscall # Do print.

# Now exit.
li $v0, 10 # Exit.
syscall

# If function returned now
# it would need to destroy
# local variables with:
# addu $sp, $sp, 12

```

2002-01-24 CSE1303 Part B lecture notes 32

```

#include <stdio.h>
#include <stdlib.h>

/* A global variable. */
int g = 123;

int main()
{
    /* Three local variables. */
    int a = -5;
    int b;
    int c = 0x12345678;

    /* Do some arithmetic. */
    b = g + a;

    /* Do some more arithmetic. */
    printf("%d", c - a);

    exit(0);
}

```

```

# ... here is the rest
# of the MIPS code ...

# calculate g + a.
lw $t0, g
lw $t1, -12($fp) # a
add $t0, $t0, $t1 # a
# Store in b.
sw $t0, -8($fp) # b

li $v0, 1 # Print int.
# calculate c - a.
lw $t0, -4($fp) # c
lw $t1, -12($fp) # a
sub $a0, $t0, $t1
syscall # Do print.

# Now exit.
li $v0, 10 # Exit.
syscall

# If function returned now
# it would need to destroy
# local variables with:
# addu $sp, $sp, 12

```

2002-01-24 CSE1303 Part B lecture notes 33

Local variables

- Names of local variables do not appear in assembly language code
 - important to comment code
 - important to draw stack memory diagram to know correct addresses
- All local variables accessed with negative offset from frame pointer
 - $-4(\$fp)$, $-12(\$fp)$, ...
 - offset will be positive for function parameters (later)

2002-01-24 CSE1303 Part B lecture notes 34

Covered in this lecture

- Memory diagrams
- System stack
 - pushing and popping
 - $\$sp$ and $\$fp$
- Local variables
 - stored on stack
 - accessed with negative offset from $\$fp$
- Addressing modes
 - register + constant

2002-01-24 CSE1303 Part B lecture notes 35

Going further

- More complex addressing modes
 - not in MIPS
 - provided by other architectures
 - extra levels of indirection, register scaling, indexing
- Generic stacks in assembly language
 - making your own stack with an array and a register

2002-01-24 CSE1303 Part B lecture notes 36

Next time

- Decisions
 - looping (for, while)
 - selecting (if)



Reading:
Lecture notes section B1.2

2002-01-24

CSE1303 Part B lecture notes

37

Copyright

Copyright © 2001 Deborah Pickett.
No part of this presentation may be
duplicated without permission from
the author.

2002-01-24

CSE1303 Part B lecture notes

38