Advanced MIPS coding

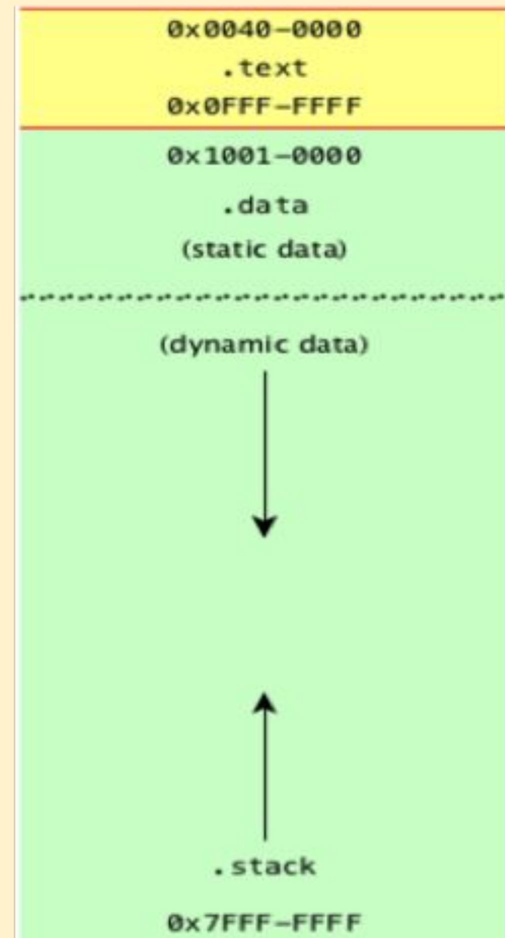
- dynamic memory
- file I/O
- macros
- multiple file programs
- exceptions and interrupts

dynamic memory (heap) allocation in MARS

In a real OS:

- programs request additional memory dynamically (at run time)
- the os finds a block of memory and allocates it

In MARS we are emulating this



sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory

```
1  # heap1.asm copy an array to the heap
2  .data
3  array:  .word   1, 2, 3, 4, 5
4  size:   .word   5
5  p:      .word   0                # pointer to copy of array in heap
6  .text
7  main:
8          li      $v0, 9           #allocate
9          li      $a0, 20          # 20 bytes
10         syscall
11         sw      $v0, p           # save pointer
12
13         # loop through static array and copy to dynamic array
14         la      $t1, array
15         lw      $t2, p
16         li      $t0, 0           # i = 0
17         lw      $t3, size        # size of array
18 loop:   bge     $t0, $t3, exit
19         lw      $t5, ($t1)       # get static array element
20         sw      $t5, ($t2)       # copy to dynamic array
21         addi    $t1, $t1, 4      # point to next word
22         addi    $t2, $t2, 4      # point to next word
23         addi    $t0, $t0, 1      # i++
24         j       loop
25
26 # note: we did not deallocate the heap memory: BAD BAD BAD
27
28 exit:   li      $v0, 10
29         syscall
30
```

```
1  # heap2.asm copy a string to the heap
2  .data
3  msg:    .asciiz "abcdefg"
4  p:      .word   0               # pointer to copy of string in heap
5  .text
6  main:
7          li      $v0, 9          #allocate
8          li      $a0, 8          # 8 bytes
9          syscall
10         sw      $v0, p          # save pointer
11
12         # loop through string and copy to heap
13         la      $t1, msg
14         lw      $t2, p
15 loop:
16         lbu     $t5, ($t1)      # get static char
17         sb      $t5, ($t2)      # copy to dynamic string
18         addi    $t1, $t1, 1     # point to next char
19         addi    $t2, $t2, 1     # point to next char
20         bne     $t5, $zero, loop
21
22 # note: we did not deallocate the heap memory: BAD BAD BAD
23
24         # print string from heap
25         li      $v0, 4
26         lw      $a0, p
27         syscall
28
29 exit:   li      $v0, 10
30         syscall
31
```

struct

- in C, a struct is a user-defined composite data type
- compilers may actually store the items in a different order to optimize memory

```
struct
{
    int age;
    char gender;
    char class;
 }
```

```
1  # heap3.asm struct example
2  #struct {
3  #     int age;
4  #     char gender;
5  #     char class;  }
6  #
7
8          .data
9   p:     .word   0         # pointer to struct
10  msga:  .asciiz "\nAge = "
11  msgg:  .asciiz "\nGender = "
12  msgc:  .asciiz "\nClass = "
13
14         .text
15  main:
16         # create a struct
17         li      $v0, 9
18         li      $a0, 8    # age=4 + gender=1 + class=1 + 2 extra wasted bytes
19         syscall
20         sw      $v0, p    # save pointer
21         move    $s1, $v0 # keep pointer in register
22
23         # put data in struct
24         li      $t0, 23
25         sw      $t0, ($s1)  # age
26         li      $t0, 'M'
27         sb      $t0, 4($s1)  # gender
28         li      $t0, 'J'
29         sb      $t0, 5($s1)  # class
30
31         # print struct data
32         jal     print
33
34  exit:  li      $v0, 10
35         syscall
36
```

```
37  ############# print struct ###############
38  print:
39          # print age
40          li      $v0, 4
41          la      $a0, msga
42          syscall
43          lw      $a0, ($s1)
44          li      $v0, 1
45          syscall
46          # print gender
47          li      $v0, 4
48          la      $a0, msgg
49          syscall
50          lb      $a0, 4($s1)  #notice we load a byte
51          li      $v0, 11
52          syscall
53          # print class
54          li      $v0, 4
55          la      $a0, msgc
56          syscall
57          lb      $a0, 5($s1)
58          li      $v0, 11
59          syscall
60          jr      $ra
61
```

program could be upgraded to change "p" to an array of pointers

MARS file I/O syscalls

open file	13	\$a0 = address of null-terminated string containing filename \$a1 = flags \$a2 = mode	\$v0 contains file descriptor (negative if error). *See note below table*
read from file	14	\$a0 = file descriptor \$a1 = address of input buffer \$a2 = maximum number of characters to read	\$v0 contains number of characters read (0 if end-of-file, negative if error). *See note below table*
write to file	15	\$a0 = file descriptor \$a1 = address of output buffer \$a2 = number of characters to write	\$v0 contains number of characters written (negative if error). *See note below table*
close file	16	\$a0 = file descriptor	

Example

```
# Sample MIPS program that writes to a new file.
#   by Kenneth Vollmar and Pete Sanderson

        .data
fout:   .asciiz "testout.txt"      # filename for output
buffer: .asciiz "The quick brown fox jumps over the lazy dog."
        .text
  ###############################################################
  # Open (for writing) a file that does not exist
  li   $v0, 13       # system call for open file
  la   $a0, fout     # output file name
  li   $a1, 1        # Open for writing (flags are 0: read, 1: write)
  li   $a2, 0        # mode is ignored
  syscall            # open a file (file descriptor returned in $v0)
  move $s6, $v0      # save the file descriptor
  ###############################################################
  # Write to file just opened
  li   $v0, 15       # system call for write to file
  move $a0, $s6      # file descriptor
  la   $a1, buffer   # address of buffer from which to write
  li   $a2, 44       # hardcoded buffer length
  syscall            # write to file
  ###############################################################
  # Close the file
  li   $v0, 16       # system call for close file
  move $a0, $s6      # file descriptor to close
  syscall            # close file
  ###############################################################
```

Macros in MARS

- macros enable you to specify a set of instructions that can be invoked with a single line of code
- macros are expanded by the assembler by substituting the macro body for each use in the program
- although it conceals implementation details like a function does, but it implemented in a completely different way

Example: macro for program termination

- define the macro:

```
.macro done
    li $v0, 10
    syscall
.end_macro
```

- invoke the macro:

```
done
```

Macro arguments

- arguments can be a register or immediate value

```
.macro print_int (%x)
li $v0, 1
add $a0, $zero, %x
syscall
.end_macro

print_int ($s0)
print_int (10)
```

.include

- the include assembler directive can be used to include a file of macros into the current file
- macros can only be invoked after they are defined
- put the .include above the .data section

```
.include "macros.asm"
```

```
1  # macrol.asm demonstrates macro usage
2  # program converts Fahrenheit temperatures to Celsius
3  .include        "macro_file.asm"
4          .data
5  const5: .float  5.0
6  const9: .float  9.0
7  const32:.float  32
8  fahr:   .float  72.0
9  celc:   .float  0
10         .text
11 main:
12         lwc1    $f12, fahr
13         lwc1    $f16, const5
14         lwc1    $f18, const9
15         div.s   $f16, $f16, $f18
16         lwc1    $f18, const32
17         sub.s   $f18, $f12, $f18
18         mul.s   $f0, $f16, $f18
19         swc1    $f0, celc
20
21         # display results
22         print_str("\nFahrenheit temperature of ")
23         lwc1    $f12, fahr
24         print_float($f12)
25         print_str(" is equivalent to Celsius temp ")
26         lwc1    $f12, celc
27         print_float($f12)
28         syscall
29
30         # exit program
31 exit:   li      $v0, 10
32         syscall
33
```

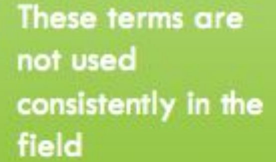
```
1  # file for macros
2
3  #### print_int ####     print_int(4)    print_int($t0)
4  .macro print_int (%x)
5      li  $v0, 1
6      add $a0, $zero, %x
7      syscall
8  .end_macro
9  #### print_gloat ####   print_loat(4.2) print_int($f0)
10 .macro print_float (%f)
11     li  $v0, 2
12     mov.s $f12, %f
13     syscall
14 .end_macro
15 #### print_str ####     print_str("string in quotes")
16 .macro print_str (%str)
17     .data
18 macro_str:      .asciiz %str
19     .text
20     li  $v0, 4
21     la  $a0, macro_str
22     syscall
23 .end_macro
```

multiple files

- you can put your code in multiple files and assemble them together
- files must be in the same directory
- make subs global with the ,global directive
- Settings -> Assemble all files in directory
- Settings -> Initialize program counter to global 'main'

```
.text
.global main
main:  ...
```

exceptions, interrupts, and traps



exceptions - an event that causes a change in the normal flow of execution

interrupts (external events) - often signals from sensors, I/O, or other hardware devices

traps (internal events) - software-defined exceptions like breakpoints

exception

when an exception occurs, the program is interrupted and a branch occurs to an exception handler, aka ISR (interrupt service routine)

the exception could be:

- a fatal error, in which case the program needs to halt
- a recoverable error that can be serviced so that the program can continue

MIPS exception handling

handled by coprocessor 0

system enters kernel (not user) mode

coprocessor 0 has 4 special registers:

- $14 EPC (exception PC) holds the address of the offending instruction
- $13 cause - contains a cause code
- $8 badvaddress register - address for bad address exception
- $12 status register contains additional information

cause codes

- 4 address exception load
- 5 address exception store
- 8 syscall exception
- 9 breakpoint exception
- 10 reserved instruction exception
- 12 arith overflow exception
- 13 trap exception
- 15 divide by zero exception
- 16 floating-point overflow
- 17 floating-point underflow

special instructions

```
mtc0 Rsrc, C0dest  # copy Rsc to C0dest
mfc0 Rdest, C0src  # copy C0src to Rdest
lwc0 C0dest, addr  # load word from addr
swc0 C0dest, addr  # store word at addressµ
```

Example

MARS MIPS does not cause an exception on divide by zero

Using the code from the MARS site we could create a trap

```
1           .data
2   n:      .word   5
3   d:      .word   0
4   n_float:.float  5.0
5
6           .text
7           # integer divide by zero
8           lw      $t1, n
9           lw      $t2, d
10          teq     $t2, $0         #trap
11          div     $t1, $t2        # no exception
12
13  exit:   li      $v0, 10
14          syscall
15
16  # Trap handler in the standard MIPS32 kernel text segment
17
18     .ktext 0x80000180
19     move $k0,$v0   # Save $v0 value
20     move $k1,$a0   # Save $a0 value
21     la   $a0, msg  # address of string to print
22     li   $v0, 4    # Print String service
23     syscall
24     move $v0,$k0   # Restore $v0
25     move $a0,$k1   # Restore $a0
26     mfc0 $k0,$14   # Coprocessor 0 register $14 has address of trapping instruction
27     addi $k0,$k0,4 # Add 4 to point to next instruction
28     mtc0 $k0,$14   # Store new address back into $14
29     eret           # Error return; set PC to value in $14
30     .kdata
31  msg:
32     .asciiz "\ndivide by zero"
33
```

Bubble Sort

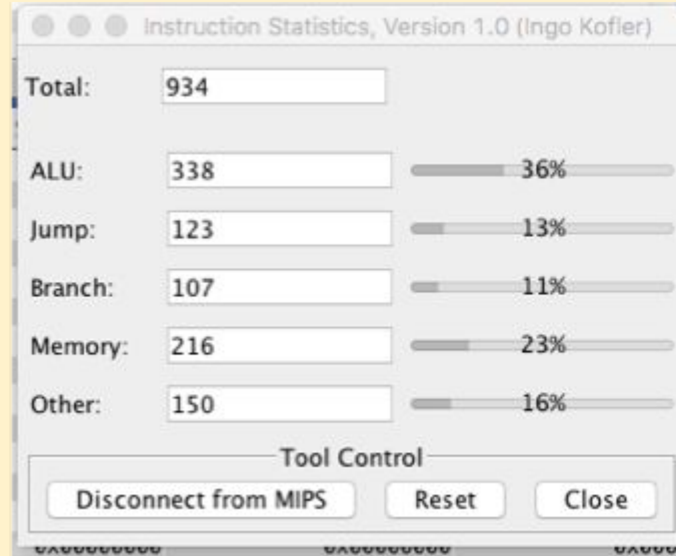
```
Array before sort: 19 2 95 26 83 17 -5 69 -16 10
Array after sort: -16 -5 2 10 17 19 26 69 83 95
```

```
1  # implementing bubble sort (Chapter 2)
2  # void sort (int v[], int n)
3  # {
4  #     int i, j;
5  #     for (i=0; i<n; i+= 1) {
6  #         for (j=i-1; j>=0 && v[j] > v[j+1]; j=1) {
7  #             swap(v, j);
8  #         }
9  #     }
10 #}
11
12         .data
13 array:  .word           19, 2, 95, 26, 83, 17, -5, 69, -16, 10
14 msg1:   .asciiz         "\nArray before sort: "
15 msg2:   .asciiz         "\nArray after sort: "
16
17         .text
18         la      $a0, msg1
19         li      $v0, 4
20         syscall                 # print before message
21         la      $a0, array
22         li      $a1, 10
23         jal     print
24         la      $a0, array      # array pointer, v
25         li      $a1, 10         # array size,  n
26         jal     sort
27         la      $a0, msg2
28         li      $v0, 4
29         syscall                 # print after message
30         la      $a0, array
31         li      $a1, 10
32         jal     print
33 main:
34
35         li      $v0, 10
36         syscall
```

```
37  ############################## PRINT ##################################
38  # print an integer array
39  # address in $a0
40  # length in $a1
41  print:  li      $t0, 0          # counter
42          add     $t1, $zero, $a0 # pointer to words
43  loop:   beq     $t0, $a1, done
44          li      $v0, 1          # print integer sevice call
45          lw      $a0, ($t1)      # load next integer
46          syscall                 # print
47          li      $v0, 11
48          li      $a0, 0x20
49          syscall                 # print a space
50          addi    $t1, $t1, 4     # point to next word
51          addi    $t0, $t0, 1     # add 1 to LCV
52          j       loop
53  done:   jr      $ra
54  ############################## SWAP ##################################
55  swap:   sll     $t1, $a1, 2             # $t1 = k * 4
56          add     $t1, $a0, $t1           # $t1 = v + (k * 4)
57          lw      $t0, ($t1)              # load the two values
58          lw      $t2, 4($t1)
59          sw      $t2, ($t1)              # store (swap) the two values
60          sw      $t0, 4($t1)
61          jr      $ra
```

```
62  ############################## SORT ##################################
63  sort:    addi   $sp, $sp, -20   # push 5 registers onto stack
64           sw     $ra, 16($sp)
65           sw     $s3, 12($sp)
66           sw     $s2, 8($sp)
67           sw     $s1, 4($sp)
68           sw     $s0, ($sp)
69           move   $s2, $a0        # save $a0
70           move   $s3, $a1        # save $a1
71           # outer loop
72           move   $s0, $zero      # i = 0
73  for1tst: slt    $t0, $s0, $s3   # check if i < n
74           beq    $t0, $zero, exit1
75           # inner loop
76           addi   $s1, $s0, -1    # j = i - 1
77  for2tst: slti   $t0, $s1, 0     # check if j < 0
78           bne    $t0, $zero, exit2
79           sll    $t1, $s1, 2     # j * 4
80           add    $t2, $s2, $t1   # v + j * 4
81           lw     $t3, ($t2)      # v[j]
82           lw     $t4, 4($t2)     # v[j+1]
83           slt    $t0, $t4, $t3   # need to swap?
84           beq    $t0, $zero, exit2
85           # swap
86           move   $a0, $s2
87           move   $a1, $s1
88           jal    swap
89           addi   $s1, $s1, -1    # j --
90           j      for2tst
91           # end of inner loop
92  exit2:   addi   $s0, $s0, 1     # i++
93           j      for1tst
94           # end of outer loop
95  exit1:   lw     $s0, ($sp)      # pop (restore) registers
96           lw     $s1, 4($sp)
97           lw     $s2, 8($sp)
98           lw     $s3, 12($sp)
99           lw     $ra, 16($sp)
100          addi   $sp, $sp, 20
101          jr     $ra
```

Tools -> Instruction Statistics



Tools -> Instruction Counter