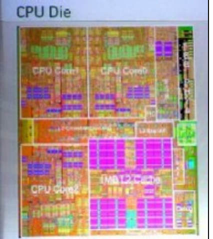
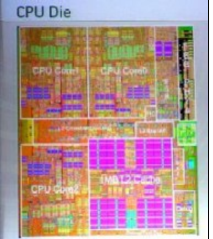


COMP 273
Introduction to Computer Systems

COMP 273

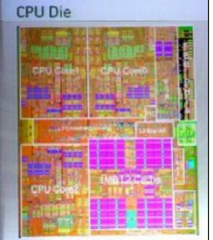
Floating Point, Buffers and the Heap

Prof. Joseph Vybihal



Announcements

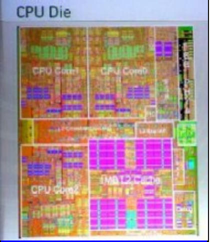
- Last assignment posted
- Course evaluations have started



At Home

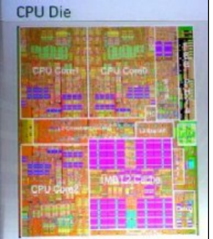
- Try the FP multiplication and addition example programs that come with the interpreter

- Textbook:
  - See MIPS Run; By Sweetman; Morgan Kaufmann Publishers, ISBN 1-55860-410-3 Chapters 6 and 7



Part 1

Floating Point Numbers



COMP 273
Introduction to Computer Systems

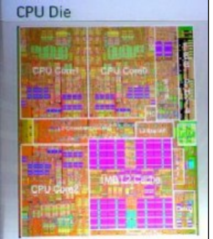
Floating Point Instructions

MIPS floating-point operands

Name	Example	Comments
32 floating-point registers	$f0, $f1, $f2, . . . , $f31	MIPS floating-point registers are used in pairs for double precision numbers.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

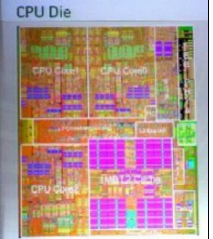
MIPS floating-point assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	FP add single	add.s $f2,$f4,$f6	$f2 = $f4 + $f6	FP add (single precision)
	FP subtract single	sub.s $f2,$f4,$f6	$f2 = $f4 – $f6	FP sub (single precision)
	FP multiply single	mul.s $f2,$f4,$f6	$f2 = $f4 × $f6	FP. multiply (single precision)
	FP divide single	div.s $f2,$f4,$f6	$f2 = $f4 / $f6	FP divide (single precision)
	FP add double	add.d $f2,$f4,$f6	$f2 = $f4 + $f6	FP add (double precision)
	FP subtract double	sub.d $f2,$f4,$f6	$f2 = $f4 – $f6	FP sub (double precision)
	FP multiply double	mul.d $f2,$f4,$f6	$f2 = $f4 × $f6	FP multiply (double precision)
	FP divide double	div.d $f2,$f4,$f6	$f2 = $f4 / $f6	FP divide (double precision)
Data transfer	load word copr. 1	lwc1 $f1,100($s2)	$f1 = Memory[$s2 + 100]	32-bit data to FP register
	store word copr. 1	swc1 $f1,100($s2)	Memory[$s2 + 100] = $f1	32-bit data to memory
Conditional branch	branch on FP true	bc1t 25	if (cond == 1) go to PC + 4 + 100	PC-relative branch if FP cond.
	branch on FP false	bc1f 25	if (cond == 0) go to PC + 4 + 100	PC-relative branch if not cond.
	FP compare single (eq,ne,lt,le,gt,ge)	c.lt.s $f2,$f4	if ($f2 < $f4) cond = 1; else cond = 0	FP compare less than single precision
	FP compare double (eq,ne,lt,le,gt,ge)	c.lt.d $f2,$f4	if ($f2 < $f4) cond = 1; else cond = 0	FP compare less than double precision



MIPS floating-point machine language

Name	Format	Example						Comments
add.s	R	17	16	6	4	2	0	add.s $f2,$f4,$f6
sub.s	R	17	16	6	4	2	1	sub.s $f2,$f4,$f6
mul.s	R	17	16	6	4	2	2	mul.s $f2,$f4,$f6
div.s	R	17	16	6	4	2	3	div.s $f2,$f4,$f6
add.d	R	17	17	6	4	2	0	add.d $f2,$f4,$f6
sub.d	R	17	17	6	4	2	1	sub.d $f2,$f4,$f6
mul.d	R	17	17	6	4	2	2	mul.d $f2,$f4,$f6
div.d	R	17	17	6	4	2	3	div.d $f2,$f4,$f6
lwc1	I	49	20	2	100			lwc1 $f2,100($s4)
swc1	I	57	20	2	100			swc1 $f2,100($s4)
bc1t	I	17	8	1	25			bc1t 25
bc1f	I	17	8	0	25			bc1f 25
c.lt.s	R	17	16	4	2	0	60	c.lt.s $f2,$f4
c.lt.d	R	17	17	4	2	0	60	c.lt.d $f2,$f4
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits



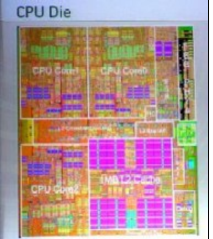
Example Program

Let's convert a temperature in Fahrenheit to Celsius:

```
float f2c (float fahr)
    {
        return ((5.0/9.0) * (fahr - 32.0));
    }
```

Assume that the floating-point argument `fahr` is passed in `$f12` and the result should go in `$f0`. (Unlike integer registers, floating-point register 0 can contain a number.) What is the MIPS assembly code?

Note: the $gp register is a global pointer to RAM. Normally, in C, it points to the first byte of a block of memory in the .data area that contains all the **extern** declared data. Providing rapid access. We can also use it as a general pointer to our own defined global memory space. Usage: offset($gp).



We assume that the compiler places the three floating-point constants in memory within easy reach of the global pointer `$gp`. The first two instructions load the constants 5.0 and 9.0 into floating-point registers:

```
f2c:
        lwc1  $f16,const5($gp)    # $f16 = 5.0 (5.0 in memory)
        lwc1  $f18,const9($gp)    # $f18 = 9.0 (9.0 in memory)
```

They are then divided to get the fraction 5.0/9.0:

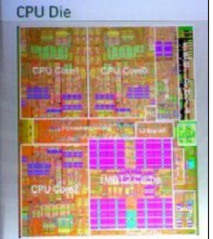
```
        div.s $f16, $f16, $f18   # $f16 = 5.0 / 9.0
```

(Many compilers would divide 5.0 by 9.0 at compile time and save the single constant 5.0/9.0 in memory, thereby avoiding the divide at runtime.) Next we load the constant 32.0 and then subtract it from `fahr` (`$f12`):

```
        lwc1   $f18, const32($gp)# $f18 = 32.0
        sub.s  $f18, $f12, $f18  # $f18 = fahr - 32.0
```

Finally, we multiply the two intermediate results, placing the product in `$f0` as the return result, and then return:

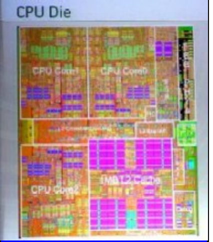
```
        mul.s  $f0,  $f16, $f18  # $f0 = (5/9)*(fahr - 32.0)
        jr     $ra               # return
```



Floating Point Instructions

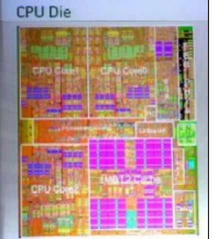
- FP absolute value double — abs.d fd, fs
- FP absolute value single — abs.s fd, fs
- FP addition double — add.d fd, fs, ft
- FP addition single — add.s fd, fs, ft
- Compare equal double — c.eq.d fs, ft
- Compare less than — c.le.d fs, ft
- Convert single to double — cvt.d.s fd, fs
- Convert int to double — cvt.d.w fd, fs
- FP divide double — div.d fd, fs, ft

See end of text book (appendix A)



Part 2

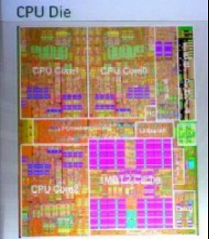
Programming with Data



COMP 273
Introduction to Computer Systems

Typed Instructions

C type	Data transfers	Operations
int	lw, sw, lui	add addi sub mult, div, and, andi, or, ori, slt, slti
unsigned int	lw, sw, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, sltu, sltiu
char	lb, sb, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, sltu, sltiu
bit field	lw, sw, lui	and, andi, or, ori, sll, srl
float	lwc1, swc1	add.s, sub.s, mult.s, div.s, c.eq.s, c.lt.s, c.le.s
double	lwc1, swc1	add.d, sub.d, mult.d, div.d, c.eq.d, c.lt.d, c.le.d

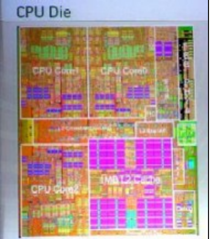


COMP 273
Introduction to Computer Systems

Load Instructions

- Load address — la rdest, address
- Load byte — lb rt, address
- Load unsigned byte — lbu rt, address
- Load halfword — lh rt, address
- Load unsigned halfowrd — lhu rt, address
- Load word coprocessor — lwc1 rt, address
  - Where 1 is the co-processor number

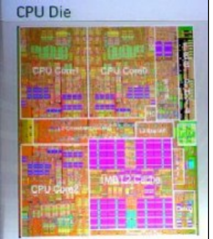
See end of text book (appendix A)



Store Instructions

- Store byte — sb rt, address
- Store halfword — sh rt, address
- Store word — sw rt, address
- Store word coprocessor — swc1 rt, address
  - Where 1 is the co-processor number
- Store doubleword — sd rsrc, address
  - Where rsrc is two consecutive registers
  - You identify the lower register number

See end of text book (appendix A)



Move Data

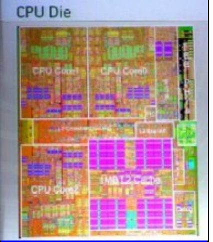
Where Z is the co-pro no.

• Move	move rdest, rsrc
• Move from hi	mfhi rd
• Move from lo	mflo rd
• Move to hi	mthi rs
• Move to lo	mtlo rs
• Move from coprocessor-z	mfcz rt, rd
• Move double from Co1	mfc1.d rdest, rc1
• Move to co-z	mtcz rd, rt

See end of text book (appendix A)

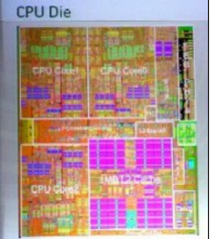
Processors & RAM

- Main Processor
  - MIPS
- Co-processors
  - Co-0 = Exceptions and Interrupts
  - Co-1 = Floating-point operations
- Addresses
  - 0xffff0000 to 0xffff000c = I/O ports
  - Syscall 1 to 10 for OS API



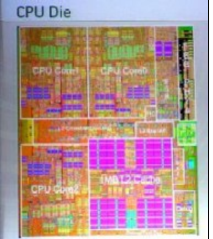
PART 3

Buffers



A Buffer

- A temporary intermediate location for data during a move operation from point A to B
  - Copying data to a subroutine
  - Loading a file from disk
  - Sending a packed to the network
- Implemented as an untyped array
  - It is untyped because the buffer can be used by any program for any reason
  - This buffer could be in RAM or there could be special hardware similar to RAM



Assembler Data Directives

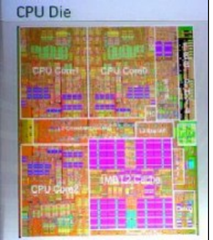
- Syntax:
  - LABEL: DIRECTIVE DATA
- Where:
  - .align n
  - .ascii str
  - .asciiz str
  - .space n
  - .byte b1,b2,…,bn … 8
  - .half h1,h2,…,hn … 16
  - .word w1,w2,…,wn …. 32
  - .float f1,f2,…,fn ….. 32
  - .double d1,d2,…,dn ….. 64

.data <addr>

.extern sym size ($gp)

.globl sym

.text <addr>



Buffer Example

- How would we define and use a buffer?

CPU Die

COMP 273

Introduction to Computer Systems

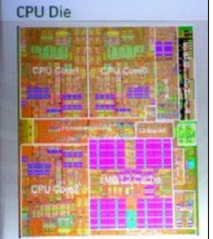
Part 4

OS Memory

Access to OS Memory Controlled

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		

- print_NUMBER displays value in argument
- print_string as C puts(char *), pointer in argument, assumes \0
- read_NUMBER reads digits until CR or character
- read_string as C gets(), read until CR or buffer length reached
- sbrk as C malloc(size in bytes), returns address
- exit, terminates your program



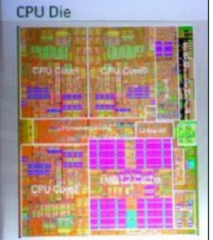
COMP 273
Introduction to Computer Systems

Example

```
        .text
        li        $v0, 4      # system call code for print_str
        la        $a0, str    # address of string to print
        syscall               # pass control to OS

        li        $v0, 1      # system call code for print_int
        li        $a0, 5      # integer 5 set to print
        syscall               # pass control to OS

        .data
Str:    .asciiz “the answer = “
```

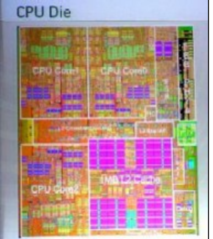
Question

- Ask the user for N. Then ask the user for N numbers. Sum these numbers and display the answer.



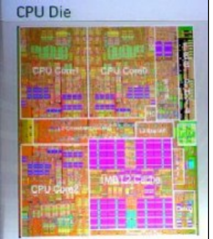
Part 5

The Heap



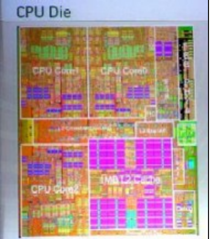
The Elements

- MIPS CPU support is limited since it assumes OS management of it.
- Support consists of:
  - $gp, used like $fp to point to beginning of heap frame
    - lw $t0, 800($gp)
  - The system call, sbrk (syscall code 9):
    - Asks the OS for n-bytes of data
    - Returns address of the first byte
- Programmer’s can simulate their own Heap by defining a fixed memory block in their .data area
  - .space 500
  - Good practise for understanding how things work



Simulated Heap Example

```
        .data
Block:  .space 400      # allocate 400 bytes (not initialized)


        .text
        .align 2        # make sure it starts at an even address
        .globl Main
Main:   la $s0, Block   # start of heap ($gp could be used)
        la $s2, Block   # end of heap pointer
        addi $s1, $zero, 8      # size of node (DATA+NEXT)

        # allocate a node (assume $t1 has data)
        sw $t1, 0($s2)          # store data
        sw $zero, 4($s2)        # store next = NULL
        add $s2, $s2, $s1       # move pointer based on offset

        # link a new node (assume $t1 has data)
        sw $t1, 0($s2)          # store data
        sw $zero,….
```



Example using Heap

```
li    $v0, 9       # system call code for malloc
li    $a0, 8       # ask for 8 bytes (two words)
syscall            # pass control to OS
```

Note:

- $gp will be updated by 8 bytes.
- $v0 contains the pointer to the beginning of the data block

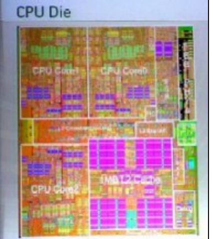
Heap with OS command

- Build a linked list…
  - Define space for pseudo-heap
  - Create our own malloc function
  - Building a struct
  - Building the list
  - Deleting a node in list

Example Code

```
### Program composed of three loops:
###     init, which initialises variables and fills the list
###     loop, which eliminates people untill only one is left
###     elim, which removes a node from the list
### Variables used:
###    $s0 holds the address of the first node
###    $s1 (n) size of the list, initial number of people/nodes
###    $s2 (m) offset of the next person to eliminate
###    $s3 (i) position of current element to be eliminated
###    $t0, $t1, $t3  temporary values

            .data
array:  .space 400    #allocate 400 bytes = 100 words of space
                                  #(50 for numbers, 50 for links)
str1:    .asciiz "\nJosephus problem with linked list\nEnter size of circle (n): "
str2:    .asciiz "Enter number to skip (m): "
str3:    .asciiz "Execution order: "
str4:    .asciiz "\nSurvivor: "
spc:    .asciiz " "    #space character
```

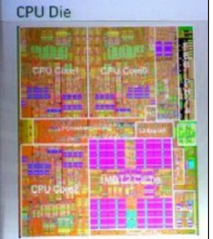
```
        .text
        .align 2
        .globl main

main:

    #print str1
    li $v0, 4
    la $a0, str1
    syscall

    #ask for integer n
    li $v0, 5
    syscall
    move $s1, $v0

    #print str2
    li $v0, 4
    la $a0, str2
    syscall

    #ask for integer m
    li $v0, 5
    syscall
    move $s2, $v0
```



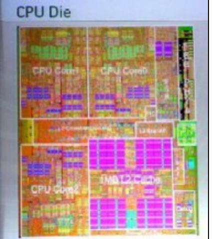

```
#prepare to enumerate eliminations
li $v0, 4    #print str3
la $a0, str3
syscall

#initialize variables
la $s0, array
addi $s3, $zero, 0
# $s1, $s2 already contain n, m respectively
move $t0, $zero
move $t1, $zero
move $t2, $zero

#initialize array with numbers 1 to n
move $t0, $s0           # $t0 now points to beginning of list
addi $t1, $zero, 1      # $t1 is the next number to be stored in array
addi $t2, $t0, 0       # $t2 points to position i-1 in list
addi $t3, $t2, 4       # $t3 points to position i in list
addi $t4, $t3, 4       # $t4 points to position i+1 in list
###note: the pair ($t2, $t3) form a node, $t2 holding the element, while $t3
###         holds the address of the next element/node
```

```
INIT:
    sw $t1, 0($t2)     #store next number from $t1
    sw $t4, 0($t3)     #store link to the next node
    # change current node
    addi $t2, $t2, 8
    addi $t3, $t3, 8
    addi $t4, $t4, 8
    # increment number
    addi $t1, $t1, 1
    # check if more nodes need to be filled
    bgt $t1, $s1, END_INIT
    # fill next node
    j INIT
END_INIT:
    # link last node to first one
    mul $t5, $s1, 8
    add $t5, $t0, $t5
    addi $t5, $t5, -4    # $t5 now points to the last link
    #move $t5, $t0
    sw $t0, 0($t5)
```

```
# start eliminating every m-th node untill only one is left
   # $t0 will point to the current node
   # $t1 will count down to the next elimination
   addi $t1, $s2, -1     # initializing counter
LOOP:
    # if length of list is 1, we have our answer
    addi $t2, $zero, 1
    beq $s1, $t2, ANSWER

    # if counter is 1, we eliminate the next node
    beq $t1, $t2, ELIM

    # else, we go to next node, decrement counter, and repeat loop
    lw $t0, 4($t0)
    addi $t1, -1
    j LOOP
```




```
ELIM:    # eliminate the node following $t0
    lw $t2, 4($t0)     # $t2 points to next node, the one to be removed
    lw $t3, 4($t2)

    # print node being eliminated
    li $v0, 1     #print_int
    lw $a0, 0($t2)
    syscall
    li $v0, 4     #print spc (this string is a single space)
    la $a0, spc
    syscall

    # relink list
    sw $t3, 4($t0)     # node $t0 now links to node $t3

    # decrement length of list
    addi $s1, $s1, -1

    # re-initialize counter
    addi $t1, $s2, -1

    # go to next node and repeat
    move $t0, $t3
    J LOOP
```




```
#position 0 (i.e. the first element) of array at $s0 now contains the only element le

ANSWER:
        lw $t2, 0($t0)
        #print the answer from $t2
        li $v0, 4       #print str4
        la $a0, str4
        syscall
        li $v0, 1       #print_int
        add $a0, $zero, $t2
        syscall

EXIT:
#exit main correctly
jr $ra
###### END PROGRAM ######
```