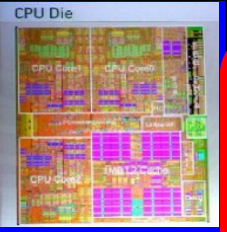


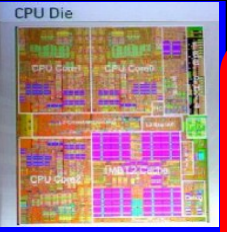
COMP 273
Introduction to Computer Systems

COMP 273

Memory Management

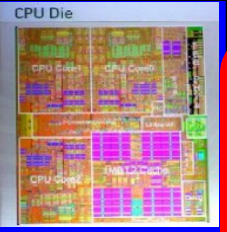
(Part 1)

Prof. Joseph Vybihal



Announcements

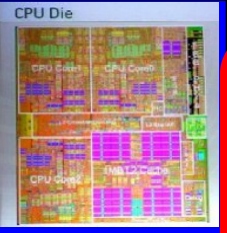
- Course evaluations



Try This Out

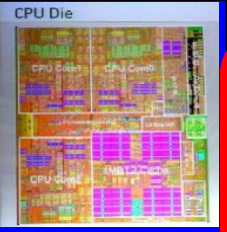
- Experiment with the stack and heap, simulated and OS generated versions.

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



Part 1

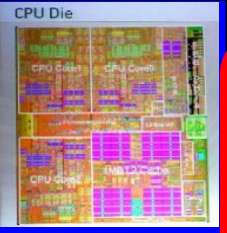
Memory Overview



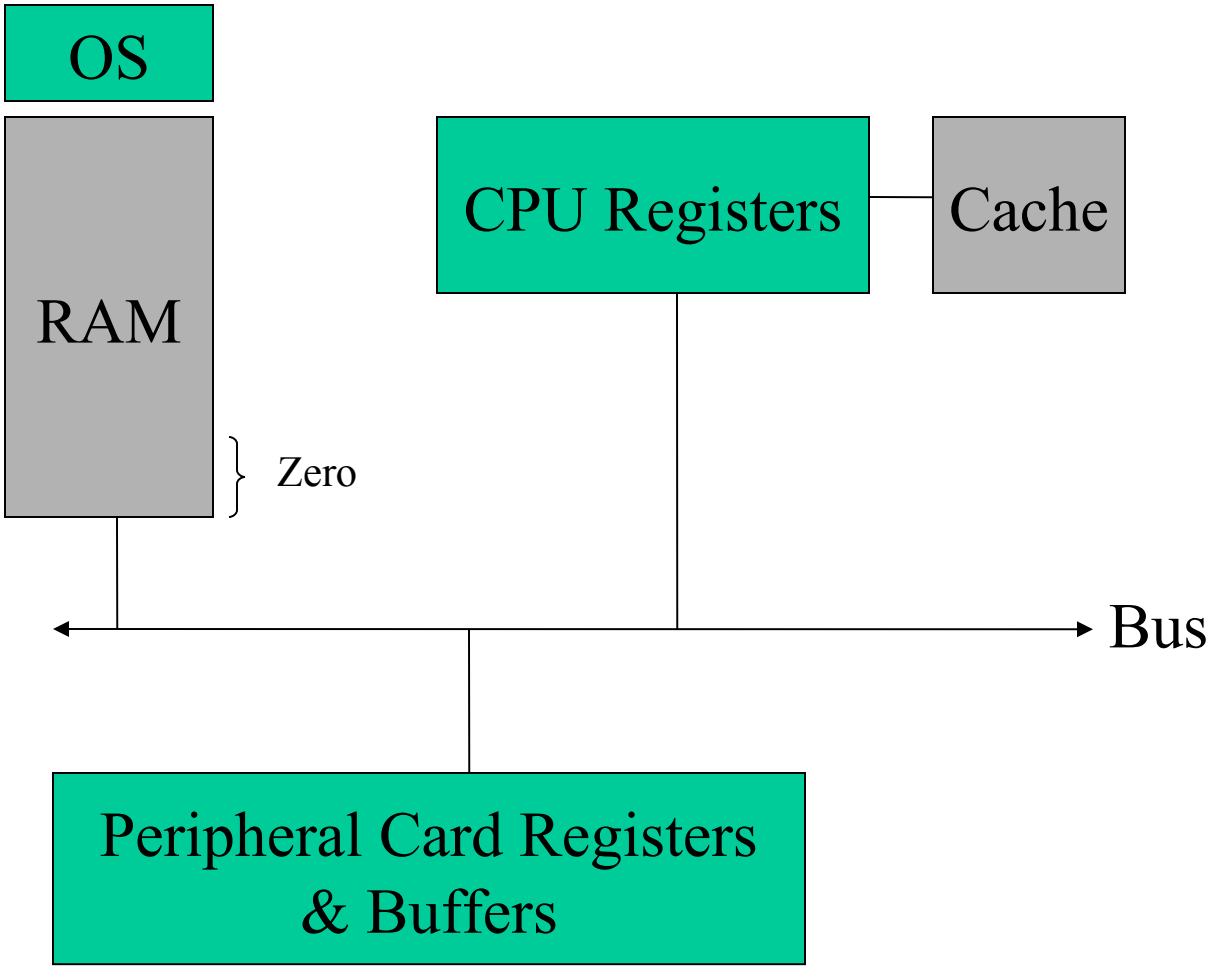
Types of Memory in MIPS

- Code Space
- Data Space
- Stack Space
- Heap Space
- Registers
- OS Space
- Peripheral Space (zero / card)
- Virtual vs. Real Memory
- Cache Memory (TLB, Code & Data)

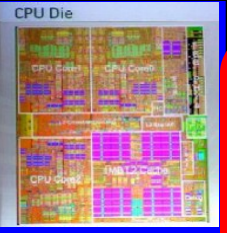
} Later



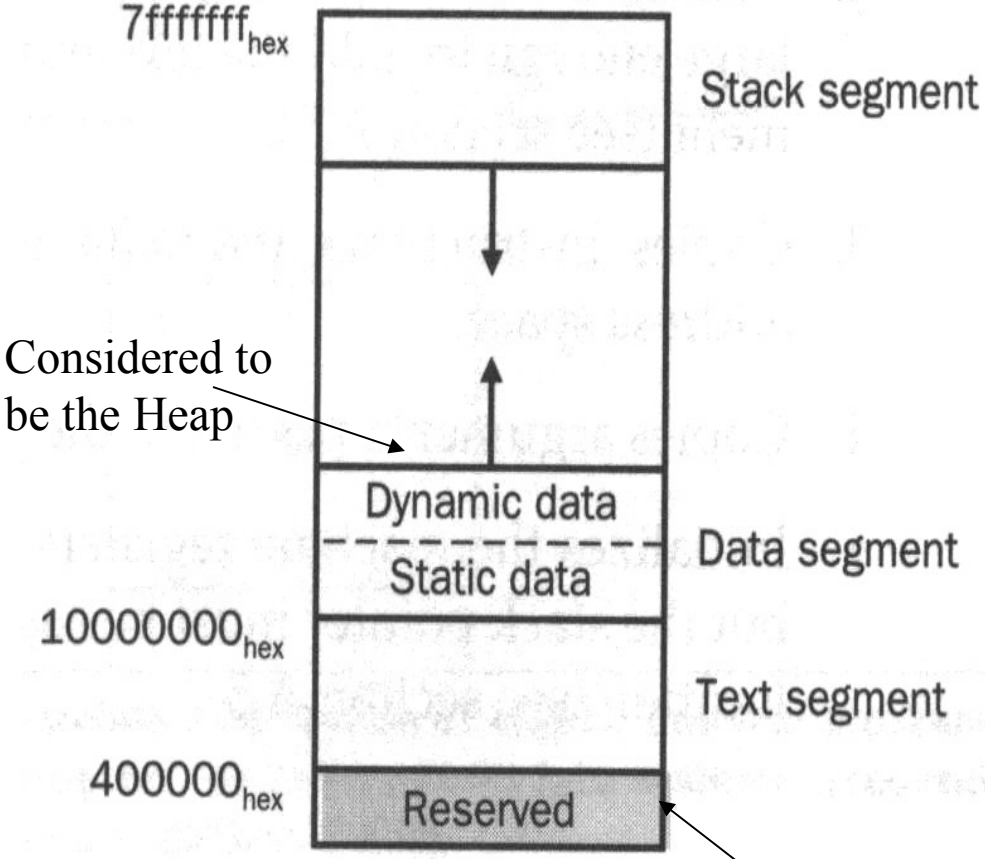
Simple Memory Layout



Discuss



Standard MIPS RAM Organization



Note:
- Stack starts at 7fffffff
- Moves opposite direction
- May crash into dynamic

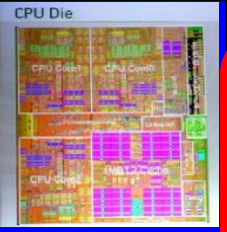
Note:
- Static and Dynamic data combined together
- Start at address 10000000

Code starts at 400000 to 9999999

Interface with hardware (zero page)

- Code = TTL Instr * Instr Size
- Data = Num of reserved bytes
- Fixed at compile time

Virtual Memory Addressing



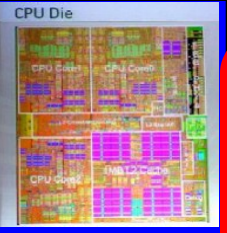
RAM Usage

Variables = Label + Size (no type)
Data Block = Label + Size * Quantity

X: .asciiz “Hi”
Y: .space 100

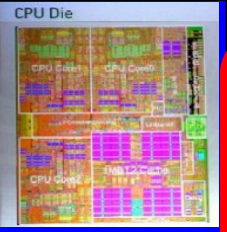
How do we create data blocks? → Arrays, structure, nodes

Discuss…



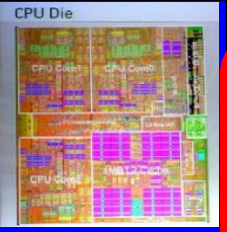
Part 2

OS Memory



OS Interaction

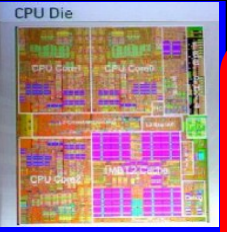
- Open
  - Jump table
    - TSR (terminate stay resident programs)
- Closed
  - System call table
    - Privileged call instruction switch



Access to OS Memory Controlled

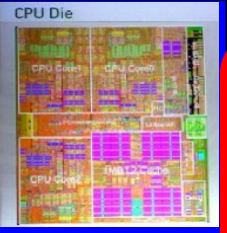
Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount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



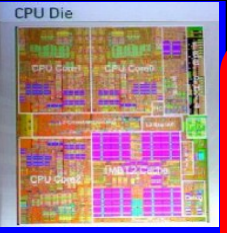
Example

```
        .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 = “
```



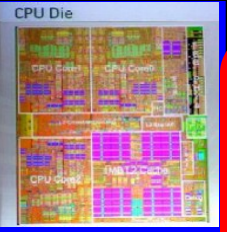
Question

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



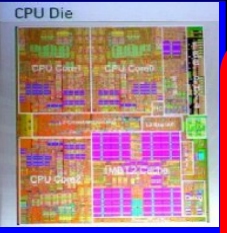
Part 3

The Heap



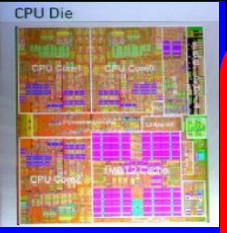
The Elements

- 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,….
```

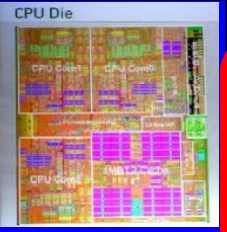


Example using Heap?

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

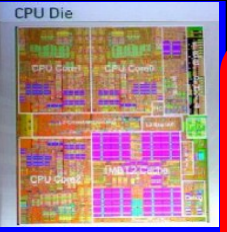
Note:

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



What about…

- Insertion
- Deletion (with or w/o fragmentation)



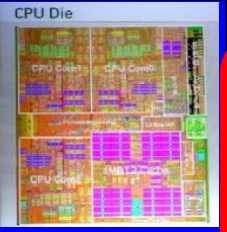
MIPS operands

Name	Example	Comments
32 registers	$s0–$s7, $t0–$t9, $gp, $fp, $zero, $sp, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.
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 assembly language

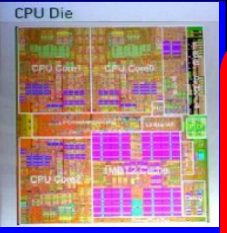
Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow detected
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	addu $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	subu $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow undetected
	add immediate unsigned	addiu $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	mfc0 $s1,$epc	$s1 = $epc	Used to copy Exception PC plus other special registers
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte unsigned	lbu $s1,100($s2)	$s1 = Memory[$s2 +100]	Byte from memory to register
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Condi-tional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; two's complement
	set less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; natural numbers
	set less than immediate unsigned	sltiu $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Uncondi-tional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

Help for question



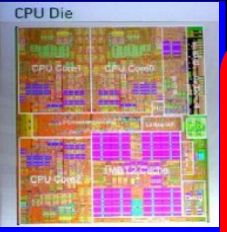
Heap with OS command

- Build a linked list…
  - Define space for pseudo-heap
  - Create our own malloc function
  - Talk about building the list
    - Some code
  - Talk about deleting a node in list
    - Some code

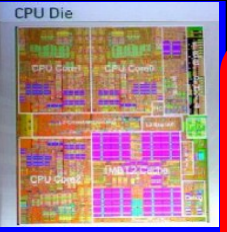


Example Code

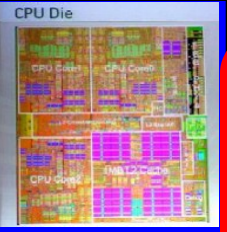
```
### Program composed of three loops:
###     init, which initialises variables and fills the list
###     loop, which eliminates people untill only one is left
###     elim, which removes a node from the list
### Variables used:
###    $s0 holds the address of the first node
###    $s1 (n) size of 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
```



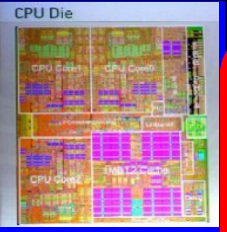

```
            .text
            .align 2
            .globl main

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
```



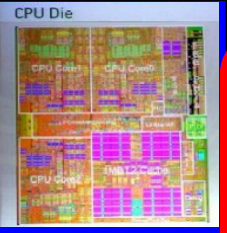

```
#prepare to enumerate eliminations
li $v0, 4    #print str3
la $a0, str3
syscall

#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)
```

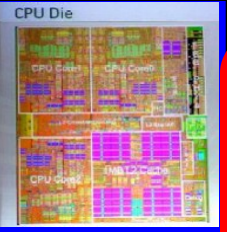



```
# start eliminating every m-th node untill only one is left
   # $t0 will point to the current node
   # $t1 will count down to the next elimination
   addi $t1, $s2, -1    # initializing counter
LOOP:
    # if length of list is 1, we have our answer
    addi $t2, $zero, 1
    beq $s1, $t2, ANSWER

    # 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 ######
```