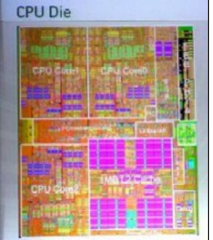
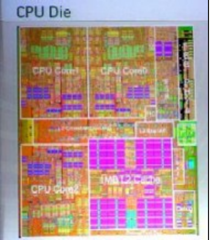
COMP 273
Introduction to Computer Systems



COMP 273

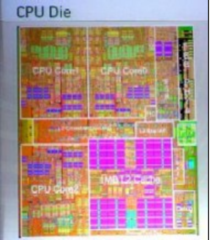
Overview of Assembler Programming

Prof. Joseph Vybihal



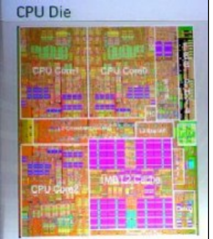
Announcements

- Project has been posted
- Ass#3 out by end of next week
- MIPS/SPIM/MARS Tutorials
  - TBD



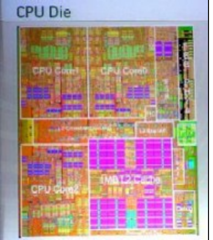
COMP 273
Introduction to Computer Systems

Project



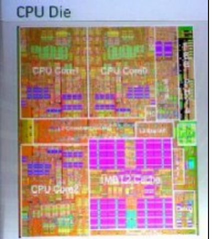
Development Order

- Now – Teams already picked
- Nov 10 – Select the circuits you will reuse from your assignments and divide the work up between team members, construct the high level architecture (somewhat specified in project)
- Nov 24 – Working basic CPU
- Dec 2 – Working CPU



Suggestions

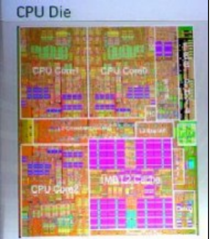
- Divide the CPU into modules and make each one work on its own as a black box.
- Merge working black boxes one at a time.
- Have an overall architectural plan for the CPU early, then
  - Built it in increments
  - Test each increment
- Only think about the bonus if you've got extra time.



Try This Out at Home

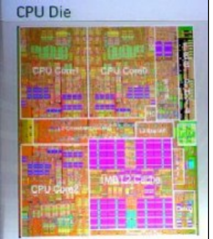
- Download one of the MIPS interpreters and get used to the interface. Load one of the sample programs. Study the code and then get it to run.
  - Load one of the sample programs. Study the code and then get it to run.
  - Write your own code to display the contents of an integer array

PCSPIM or MARS



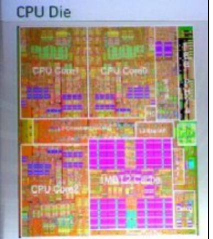
Part 1

Assembler Introduction



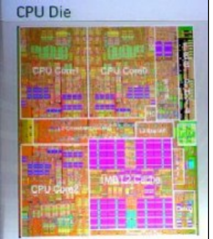
Things to know about Assembler Programming

- Just like programming in any language…
- Basic assembler functionality:
  - Math and Logic
  - Data movement
  - Pointers
  - Branching
  - Goto and subroutines
- Functionality not in assembler:
  - No functions (but can be simulated ~subroutines)
  - No objects (but can be simulated)
  - No arrays or data structures (but can be simulated)
  - No variables or constants (in the normal way)
  - Minimal distinction between code and data



Things to know about Assembler Programming

- Advanced assembler functionality:
  - Access to internal OS routines
  - Access to the control ROMS of all peripherals
- Take care of complexity explosion
  - Keep it simple
  - Modularize with subroutines
  - Use OS functions when useful
- Know how to use the registers and stack
  - Use them over RAM



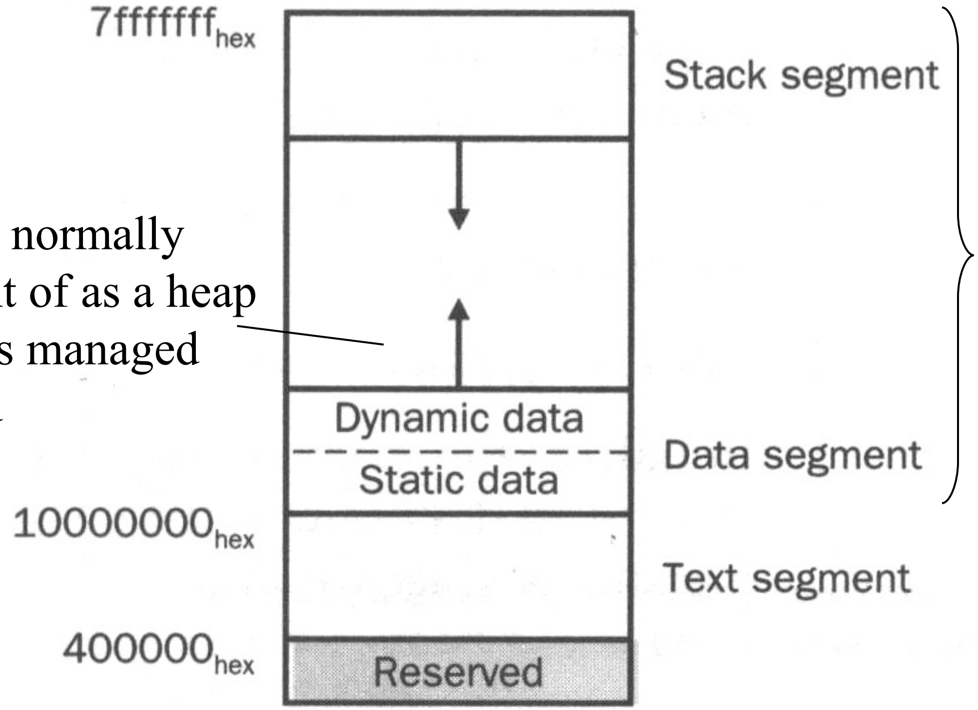
MIPS Virtual Memory

- 2^{30} memory words
  - Memory[0], memory[4], …, memory[4,294,967,292]
- Accessed only by data transfer instructions
- MIPS uses byte addresses
- Words are 4 bytes
- Memory holds:
  - Data structures
  - Arrays
  - Spilled (saved) registers
  - Assembler instructions
  - Variables and constants

Virtual Memory Usage

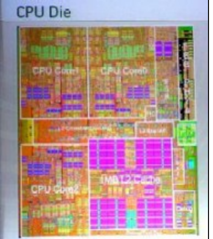
Bottom of stack

SP register for top of stack



This is normally thought of as a heap but it is managed by you

Stack and data segments are actually shared space and can *crash* (overlap)

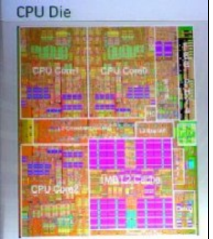


COMP 273
Introduction to Computer Systems

MIPS Registers

Register	Name	Used For
0	zero	Always returns 0
1	at	Reserved for use by assembler
2-3	v0, v1	Value returned by subroutine
4-7	a0-a3	First few parameters for subroutine
8-15	t0-t7	Temporary: can use without saving
24, 25	t8, t9	Temporary: can use without saving
16-23	s0-s7	If used, *must* save on stack (or other)
26, 27	k0, k1	Used by interrupt / trap handler
28	gp	A global pointer (extern/static vars…)
29	sp	Stack pointer
30	s8/fp	Frame pointer
31	ra	Subroutine return address

Also:
- Hi
- Lo
- Results of mult & div

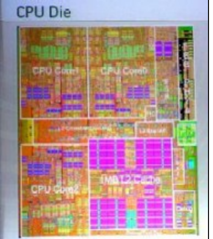


COMP 273
Introduction to Computer Systems

MIPS Registers

Register name	Number	Usage
$zero	0	constant 0
$at	1	reserved for assembler
$v0	2	expression evaluation and results of a function
$v1	3	expression evaluation and results of a function
$a0	4	argument 1
$a1	5	argument 2
$a2	6	argument 3
$a3	7	argument 4
$t0	8	temporary (not preserved across call)
$t1	9	temporary (not preserved across call)
$t2	10	temporary (not preserved across call)
$t3	11	temporary (not preserved across call)
$t4	12	temporary (not preserved across call)
$t5	13	temporary (not preserved across call)
$t6	14	temporary (not preserved across call)
$t7	15	temporary (not preserved across call)
$s0	16	saved temporary (preserved across call)
$s1	17	saved temporary (preserved across call)
$s2	18	saved temporary (preserved across call)
$s3	19	saved temporary (preserved across call)
$s4	20	saved temporary (preserved across call)
$s5	21	saved temporary (preserved across call)
$s6	22	saved temporary (preserved across call)
$s7	23	saved temporary (preserved across call)
$t8	24	temporary (not preserved across call)
$t9	25	temporary (not preserved across call)
$k0	26	reserved for OS kernel
$k1	27	reserved for OS kernel
$gp	28	pointer to global area
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address (used by function call)

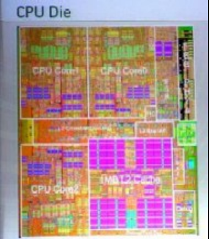
Return
Parameters
Temporary
Saved
Pointers



Addressing Modes

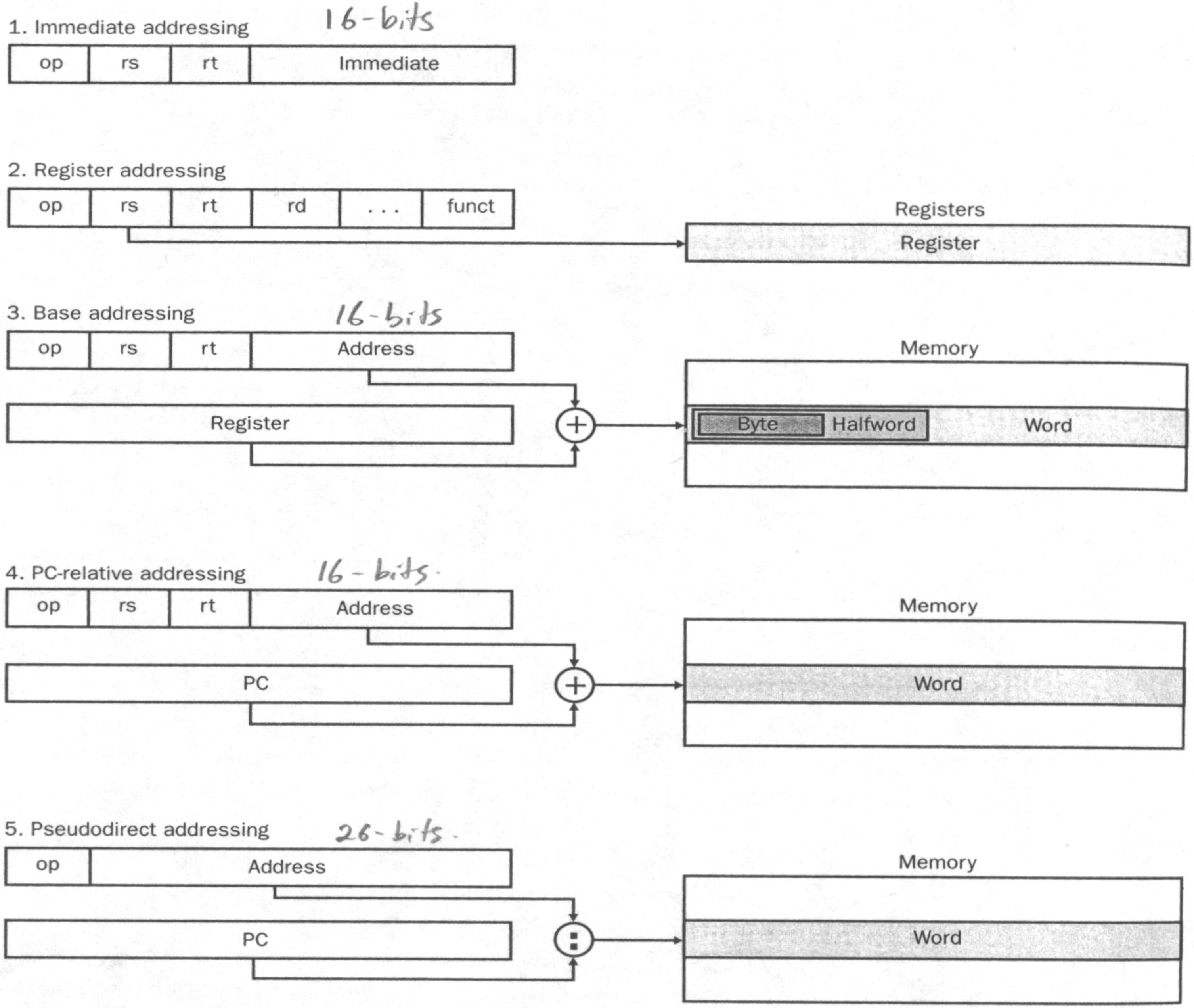
- Register Addressing
  - Operand is a register
  - E.g. add $s1, $s2, $s3
- Base or Displacement Addressing
  - Operand is a memory location
  - Register + offset ← a constant
  - E.g. lw $s1, 100($s2)
- Immediate Addressing
  - Operand is a constant (no addressing)
  - 16-bit constant
  - E.g. Addi $s1, $s2, 100
- PC-Relative Addressing
  - Memory location = PC + offset ← a constant
  - E.g. j 2500   or   j label
- Pseudo-Direct Addressing
  - Memory location = PC (top 6 bits) concat with 26-bit offset
  - Assumes 32-bit addressing
  - E.g. jal 2500   or   jal label

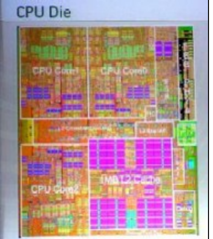
Discuss…



COMP 273

Introduction to Computer Systems





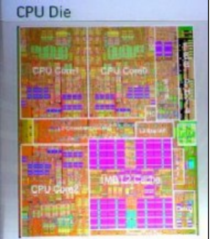
COMP 273
Introduction to Computer Systems

MIPS Instruction Formats

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

R – Register
I – Immediate
J - Jump

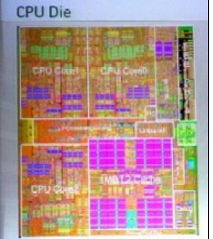
Immediate value
(like an instruction constant)



COMP 273

Introduction to Computer Systems

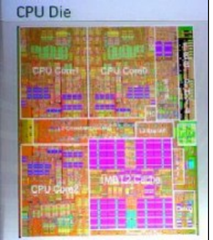
Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Three operands; overflow detected
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 – $s3`	Three operands; overflow detected
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2 + 100`	+ constant; overflow detected
	add unsigned	`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
	multiply	`mult $s2,$s3`	Hi, Lo = `$s2 × $s3`	64-bit signed product in Hi, Lo
	multiply unsigned	`multu $s2,$s3`	Hi, Lo = `$s2 × $s3`	64-bit unsigned product in Hi, Lo
	divide	`div $s2,$s3`	Lo = `$s2 / $s3`, Hi = `$s2` mod `$s3`	Lo = quotient, Hi = remainder
	divide unsigned	`divu $s2,$s3`	Lo = `$s2 / $s3`, Hi = `$s2` mod `$s3`	Unsigned quotient and remainder
	move from Hi	`mfhi $s1`	`$s1` = Hi	Used to get copy of Hi
	move from Lo	`mflo $s1`	`$s1` = Lo	Used to get copy of Lo
Logical	and	`and $s1,$s2,$s3`	`$s1 = $s2 & $s3`	Three reg. operands; logical AND
	or	`or $s1,$s2,$s3`	`$s1 = $s2 | $s3`	Three reg. operands; logical OR
	and immediate	`andi $s1,$s2,100`	`$s1 = $s2 & 100`	Logical AND reg, constant
	or immediate	`ori $s1,$s2,100`	`$s1 = $s2 | 100`	Logical OR reg, 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
Conditional 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
Unconditional 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



COMP 273
Introduction to Computer Systems

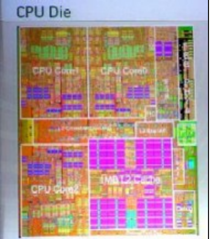
Partial MIPS Machine Language

Name	Format	Example						Comments
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
add	R	0	2	3	1	0	32	add $1,$2,$3
sub	R	0	2	3	1	0	34	sub $1,$2,$3
addi	I	8	2	1	100			addi $1,$2,100
addu	R	0	2	3	1	0	33	addu $1,$2,$3
subu	R	0	2	3	1	0	35	subu $1,$2,$3
addiu	I	9	2	1	100			addiu $1,$2,100
mfc0	R	16	0	1	14	0	0	mfc0 $1,$epc
mult	R	0	2	3	0	0	24	mult $2,$3
multu	R	0	2	3	0	0	25	multu $2,$3
div	R	0	2	3	0	0	26	div $2,$3
divu	R	0	2	3	0	0	27	divu $2,$3
mfhi	R	0	0	0	1	0	16	mfhi $1
mflo	R	0	0	0	1	0	18	mflo $1
and	R	0	2	3	1	0	36	and $1,$2,$3
or	R	0	2	3	1	0	37	or $1,$2,$3
andi	I	12	2	1	100			andi $1,$2,100
ori	I	13	2	1	100			ori $1,$2,100
sll	R	0	0	2	1	10	0	sll $1,$2,10
srl	R	0	0	2	1	10	2	srl $1,$2,10
lw	I	35	2	1	100			lw $1,100($2)
sw	I	43	2	1	100			sw $1,100($2)
lui	I	15	0	1	100			lui $1,100
beq	I	4	1	2	25			beq $1,$2,100
bne	I	5	1	2	25			bne $1,$2,100
slt	R	0	2	3	1	0	42	slt $1,$2,$3
slti	I	10	2	1	100			slti $1,$2,100
sltu	R	0	2	3	1	0	43	sltu $1,$2,$3
sltiu	I	11	2	1	100			sltiu $1,$2,100
j	J	2	2500					j 10000
jr	R	0	31	0	0	0	8	jr $31
jal	J	3	2500					jal 10000



Part 2

Coding Examples



Example 1

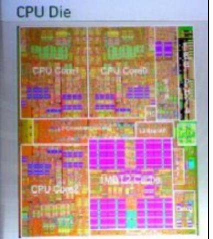
<u>C Language</u>:

```
a = b + c;
```

<u>MIPS</u>:

```
lw $t1, b
lw $t2, c
add $t0, $t1, $t2
sw $t0, a
```

<u>Note</u>:
Cannot use variables directly in assembler



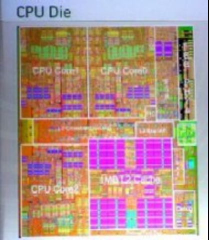
Example 2

C Language:

```
f = (g + h) – (i + j);
```

MIPS:

```
lw $s0, g
lw $s1, h
lw $s2, i
lw $s3, j
add $t0, $s0, $s1
add $t1, $s2, $s3
sub $s4, $t0, $t1
sw $s4, f
```



Example 3

<u>C Language</u>:

g = h + A[8];

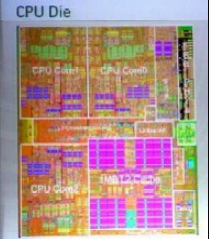
Assume the array A is of size 100 and contains 32-bit integer values.

<u>MIPS</u>:

Assume: $s1=g, $s2=h, $s3=base address of array

Note: *array* simply means *a contiguous block of memory* (nothing more)

```
lw $t0, 32($s3)
add $s1, $s2, $t0
sw $s1, g
```

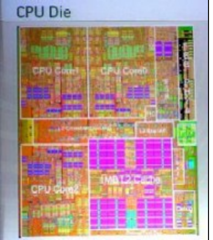


Example 4

C Language:
A[12] = h + A[8];

MIPS:
Assume similar setup from example 3.

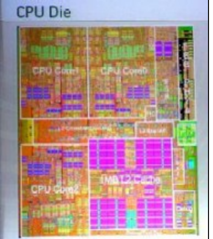
```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 48($s3)
```



IF-THEN-ELSE

- if (i == j) f = g + h; else f = g – h;

```
# Assume f,i,j,g,h are $s0,$s3,$s4,$s1,$s2

      bne $s3,$s4,Else   # go to Else if i != j
      add $s0,$s1,$s2    # f=g+h (skipped if i != j)
      j   Exit           # go to Exit
Else: sub $s0,$s1,$s2    # f=f-h (skipped if i == j)
Exit:
```



Example 5

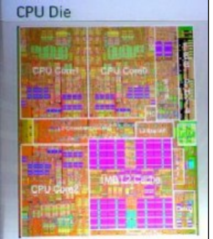
C Language:

```
if (i == A[j])
{
    f = g + h;
} else {
    f = f – i;
}
```

MIPS:

Assume: $s0=i, $s1=j, $s2=f, $s3=g, $s4=h
$s5=base address of array
containing 16-bit integer numbers

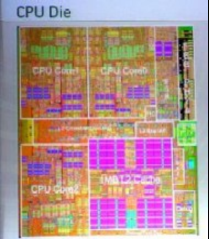
```
 add $t0, $s1, $s1 #calculate offset
 add $t0, $t0, $s5 # base + offset
 lh $t1, 0($t0)
 bne $s0, $t1, L1 # could do beq also
 add $s2, $s3, $s4
 j Exit
L1: sub $s2, $s2, $s0
Exit: …
```



WHILE LOOP

while(save[i] == k) i = i + j;

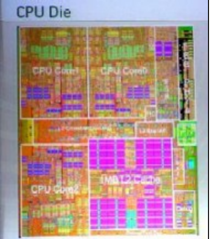
```
# Assume i,j,k are $s3,$s4,$s5
# Save address in $s6

Loop: add $t1,$s3,$s3      # Temp reg $t1 = 2 * i
      add $t1,$t1,$t1      # Temp reg $t1 = 4 * i
      add $t1,$t1,$s6      # $t1 = save[i]
      lw $t0,0($t1)        # $t0 = save[i]
      bne $t0,$s5, Exit    # go to Exit if save != k
      add $s3,$s3,$s4      # i=i+j
      j   Loop
Exit:
```



CASE/SWITCH Statement

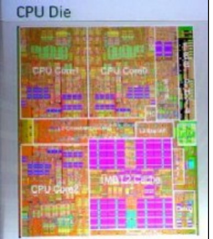
```
switch(k) {
    case 0: f = i+ j; break;
    case 1: f = g + h; break;
}
```

```
 # assume f,g,h,i,j,k are s0,s1,s2,s3,s4,s5

 switch: addi $t0,$zero, 0        # $t0 = 0
         beq  $t0,$s5,CASE0       # k == 0?
         addi $t0,$zero, 1        # $t0 = 1
         beq  $t0,$s5,CASE1       # k == 1?
         j    Exit
CASE0:   add $s0,$s3,$s4
         j    Exit
CASE1:   add $s0,$s1,$s2
Exit:
```



Set Less Than

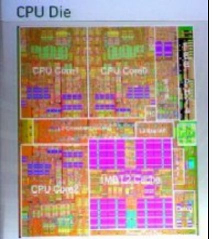
```
# Assume $s0 has var_a and $s1 has var_b
# is a < b?

 slt $t0,$s0,$s1      # $t0=1 if a<b otherwise $t0=0
 bne $t0,$zero, Less # go to Less if $t0 != 0 (if a<b)
```



Question

- Write a MIPS program snip that assumes someone’s age is in temporary register 1. If the person is an adult, the program will calculate how many days they have been alive (assume 365 for each year) and store that into v0. If the person is underage then the program stores 0 into v0.

Use loop to multiply.

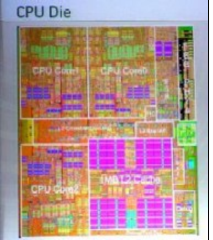


COMP 273

Introduction to Computer Systems

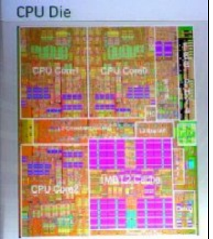
Question

- Write a code snip that calculates x to the power of y. X and Y are positive numbers.



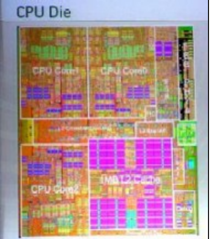
Part 3

MIPS Assembly Language Formatting



MIPS Distinctions

- Bit
  - The fundamental unit in a computer
- Byte
  - The fundamental grouping of units (it has an address)
- Word
  - The common data processing size
  - The size of a register (32 bits)
- Address
  - The built in size of memory addressing
  - The size of the address register
    (may be different from word) (486 24/32 bits)



Assembler Data Directives

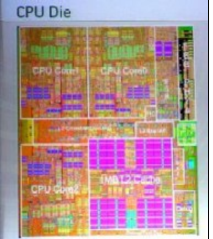
- Syntax:
  - LABEL: DIRECTIVE   DATA
- Where directives are:
  - .align   n
  - .ascii   str
  - .asciiz str
  - .space n
  - .byte   b1,b2,…,bn   … 8
  - .half   h1,h2,…,hn   … 16
  - .word   w1,w2,…,wn 32
  - .float   f1,f2,…,fn   32
  - .double d1,d2,…,dn   ….. 64

.data <addr>

.extern LABEL SIZE($gp)

.globl LABEL

.text <addr>



The File Structure

```
            .text
            .globl LABEL
LABEL:
        :
        :
        li $v0, 10
        syscall

        .data

Str:    .asciiz “hello”
```

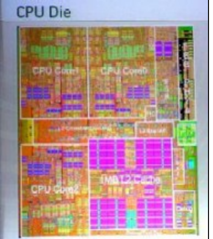
code

data

label | instr | comment

#	← comments
.command	← directive
Identifier:	← label
Identifier	← reference
INSTR ARGi	← assembler

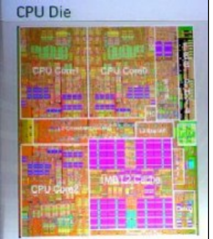
Exit



Commenting

(very important in assembler since so cryptic)

```
#
# Program name
# Date created
# Programmer Name
# Description of program
#
# List of major registers used by program
#

############## text segment ###################
        :
        :  # comments on almost every line
        :
        #
        # code sectional comments
        #

############## data segment ###################
        :
        :
############## end of file: filename #############
```

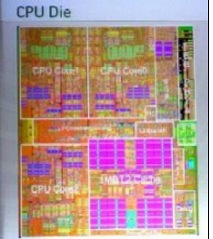


Commenting Example

```
##
## length.a - prints out the length of character
## string "str".
##
##      t0 - holds each byte from string in turn
##      t1 - contains count of characters
##      t2 - points to the string
##

##############################################
#                                            #
#              text segment                  #
#                                            #
##############################################

        .text
        .globl __start
__start:                    # execution starts here
        la $t2,str          # t2 points to the string
        li $t1,0            # t1 holds the count
nextCh: lb $t0,($t2)        # get a byte from string
        beqz $t0,strEnd     # zero means end of string
        add $t1,$t1,1       # increment count
        add $t2,1           # move pointer one character
        j nextCh            # go round the loop again

strEnd: la $a0,ans          # system call to print
        li $v0,4            # out a message
        syscall

        move $a0,$t1        # system call to print
        li $v0,1            # out the length worked out
        syscall

        la $a0,endl         # system call to print
        li $v0,4            # out a newline
        syscall

        li $v0,10
        syscall             # au revoir...
```

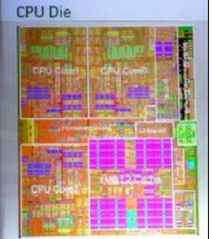
White spaces used to delineate sections but commenting would be good as well.

```
##############################################
#                                            #
#              data segment                  #
#                                            #
##############################################

        .data
str:    .asciiz "hello world"
ans:    .asciiz "Length is "
endl:   .asciiz "\n"

##
## end of file length.a
```



Part 4

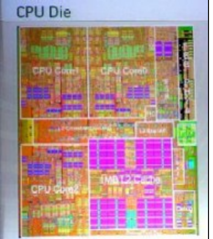
More Coding Examples



Example

```
clear1(int array[], int size)
{
  int i;
  for (i = 0; i < size; i = i + 1)
       array[i] = 0;
}

clear2(int *array, int size)
{
  int *p;
  for (p = &array[0]; p < &array[size]; p = p + 1)
       *p = 0;
}
```

COMP 273
Introduction to Computer Systems



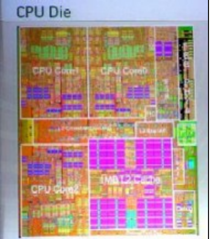
COMP 273
Introduction to Computer Systems

Array

```
        move $t0,$zero        # i = 0
loop1:  add  $t1,$t0,$t0      # $t1 = i * 2
        add  $t1,$t1,$t1      # $t1 = i * 4
        add  $t2,$a0,$t1      # $t2 = &array[i]
        sw   $zero, 0($t2)    # array[i] = 0
        addi $t0,$t0,1        # i = i + 1
        slt  $t3,$t0,$a1      # $t3 = (i < size)
        bne  $t3,$zero,loop1  # if () go to loop1
```

Pointer

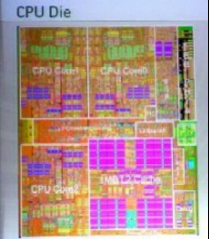
```
        move $t0,$a0          # p = & array[0]
        add  $t1,$a1,$a1      # $t1 = size * 2
        add  $t1,$t1,$t1      # $t1 = size * 4
        add  $t2,$a0,$t1      # $t2 = &array[size]
loop2:  sw   $zero,0($t0)     # Memory[p] = 0
        addi $t0,$t0,4        # p = p + 4
        slt  $t3,$t0,$t2      # $t3=(p<&array[size])
        bne  $t3,$zero,loop2  # if () go to loop2
```



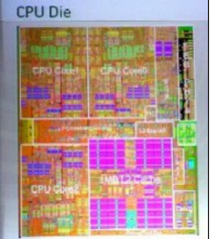
COMP 273
Introduction to Computer Systems

Another Example

```
sort (int v[], int n)
{
   int i, j;
   for (i = 0; i < n; i = i + 1) {
      for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j = j - 1) { swap(v,j);
      }
   }
}
```



Saving registers				
	sort:	addi	$sp,$sp, -20	# make room on stack for 5 registers
		sw	$ra, 16($sp)	# save $ra on stack
		sw	$s3,12($sp)	# save $s3 on stack
		sw	$s2, 8($sp)	# save $s2 on stack
		sw	$s1, 4($sp)	# save $s1 on stack
		sw	$s0, 0($sp)	# save $s0 on stack
Procedure body				
Move parameters		move	$s2, $a0	# copy parameter $a0 into $s2 (save $a0)
		move	$s3, $a1	# copy parameter $a1 into $s3 (save $a1)
Outer loop		move	$s0, $zero	# i = 0
	for1tst:	slt	$t0, $s0, $s3	# reg $t0 = 0 if $s0 ≥ $s3 (i ≥ n)
		beq	$t0, $zero, exit1	# go to exit1 if $s0 ≥ $s3 (i ≥ n)
Inner loop		addi	$s1, $s0, -1	# j = i - 1
	for2tst:	slti	$t0, $s1, 0	# reg $t0 = 1 if $s1 < 0 (j < 0)
		bne	$t0, $zero, exit2	# go to exit2 if $s1 < 0 (j < 0)
		add	$t1, $s1, $s1	# reg $t1 = j * 2
		add	$t1, $t1, $t1	# reg $t1 = j * 4
		add	$t2, $s2, $t1	# reg $t2 = v + (j * 4)
		lw	$t3, 0($t2)	# reg $t3 = v[j]
		lw	$t4, 4($t2)	# reg $t4 = v[j + 1]
		slt	$t0, $t4, $t3	# reg $t0 = 0 if $t4 ≥ $t3
		beq	$t0, $zero, exit2	# go to exit2 if $t4 ≥ $t3
Pass parameters and call		move	$a0, $s2	# 1st parameter of swap is v (old $a0)
		move	$a1, $s1	# 2nd parameter of swap is j
		jal	swap	# swap code shown in Figure 3.24
Inner loop		addi	$s1, $s1, -1	# j = j - 1
		j	for2tst	# jump to test of inner loop
Outer loop	exit2:	addi	$s0, $s0, 1	# i = i + 1
		j	for1tst	# jump to test of outer loop
Restoring registers				
	exit1:	lw	$s0, 0($sp)	# restore $s0 from stack
		lw	$s1, 4($sp)	# restore $s1 from stack
		lw	$s2, 8($sp)	# restore $s2 from stack
		lw	$s3,12($sp)	# restore $s3 from stack
		lw	$ra,16($sp)	# restore $ra from stack
		addi	$sp,$sp, 20	# restore stack pointer
Procedure return				
		jr	$ra	# return to calling routine



Questions

- Shift the contents of an array left n cells.