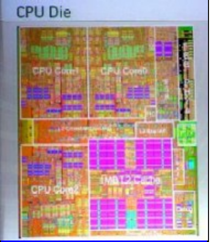


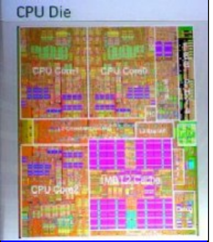
COMP 273

Introduction to Computer Systems

COMP 273

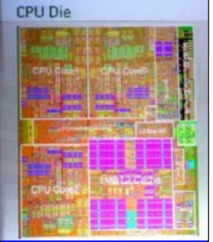
Programming with Data and in Floating Point

Prof. Joseph Vybihal



Announcements

- Last assignment posted



COMP 273
Introduction to Computer Systems

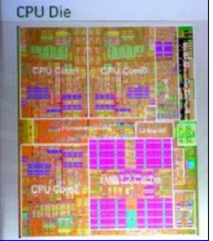
At Home

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

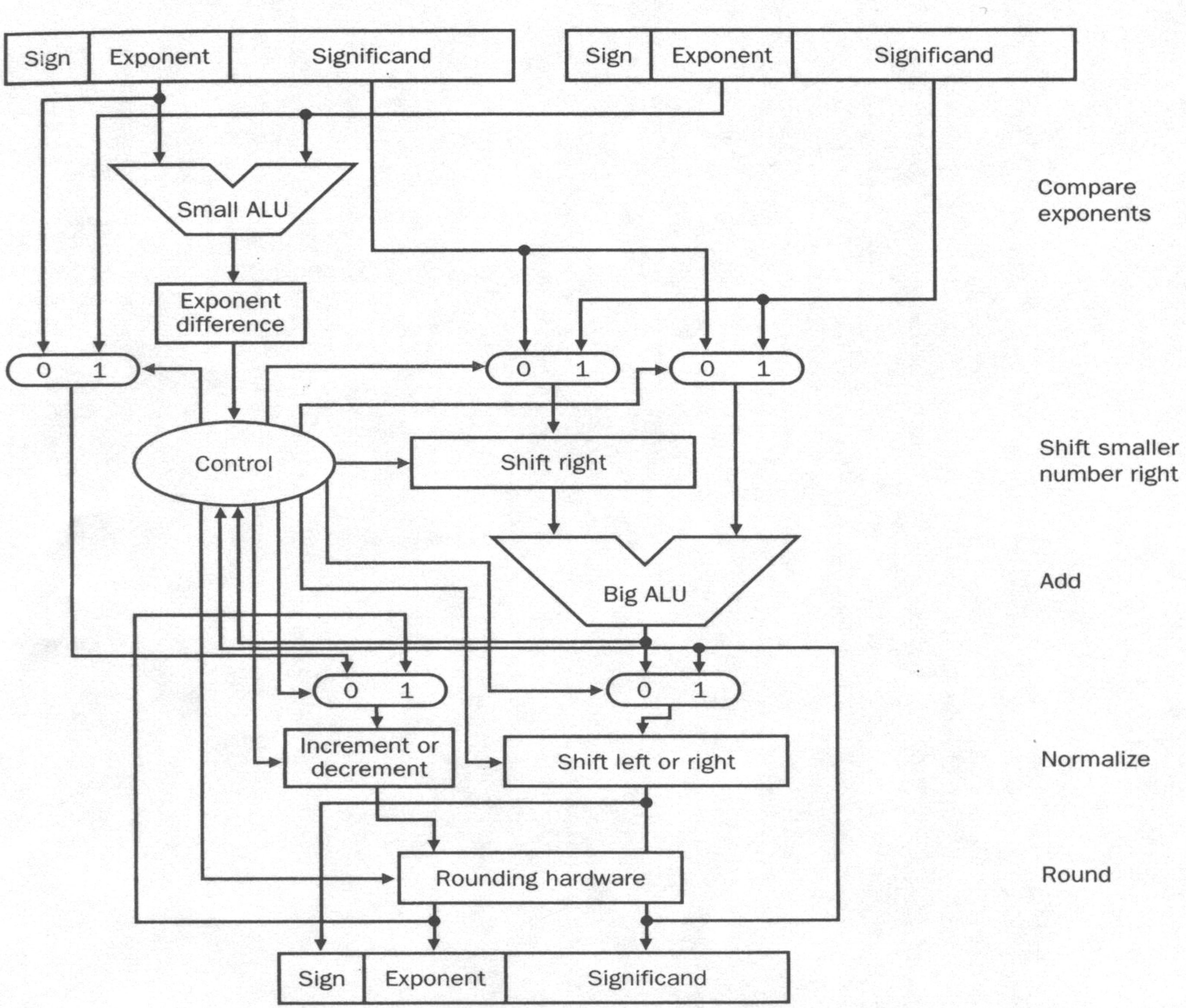
- Textbook:
  - See MIPS Run; By Sweetman; Morgan Kaufmann Publishers, ISBN 1-55860-410-3 Chapter 7

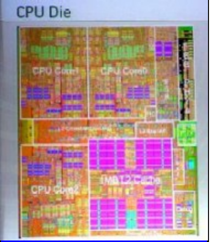
Part 1

Floating Point Numbers

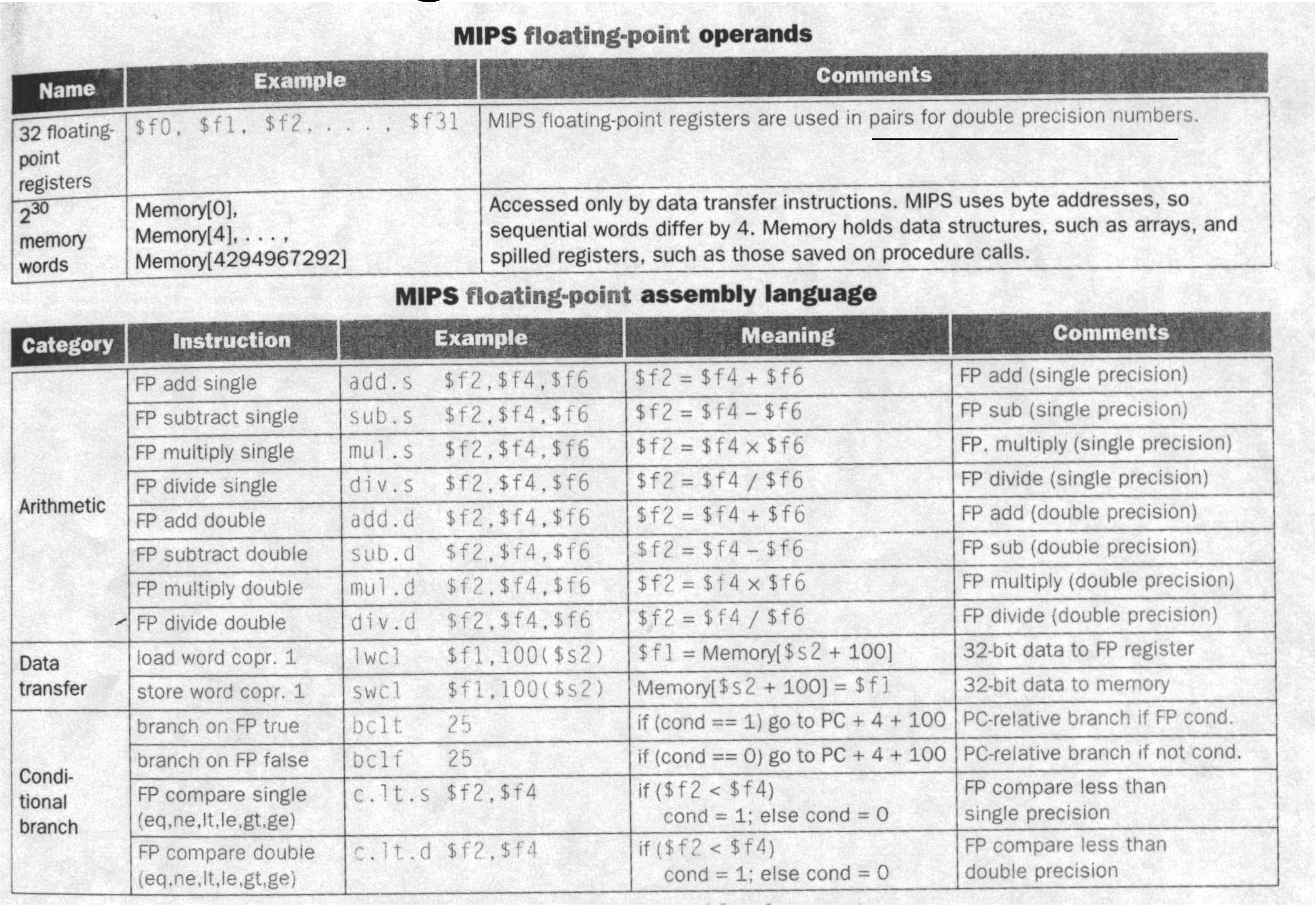


Floating Point Circuitry





Floating Point Instructions

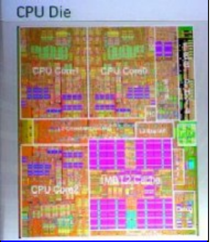


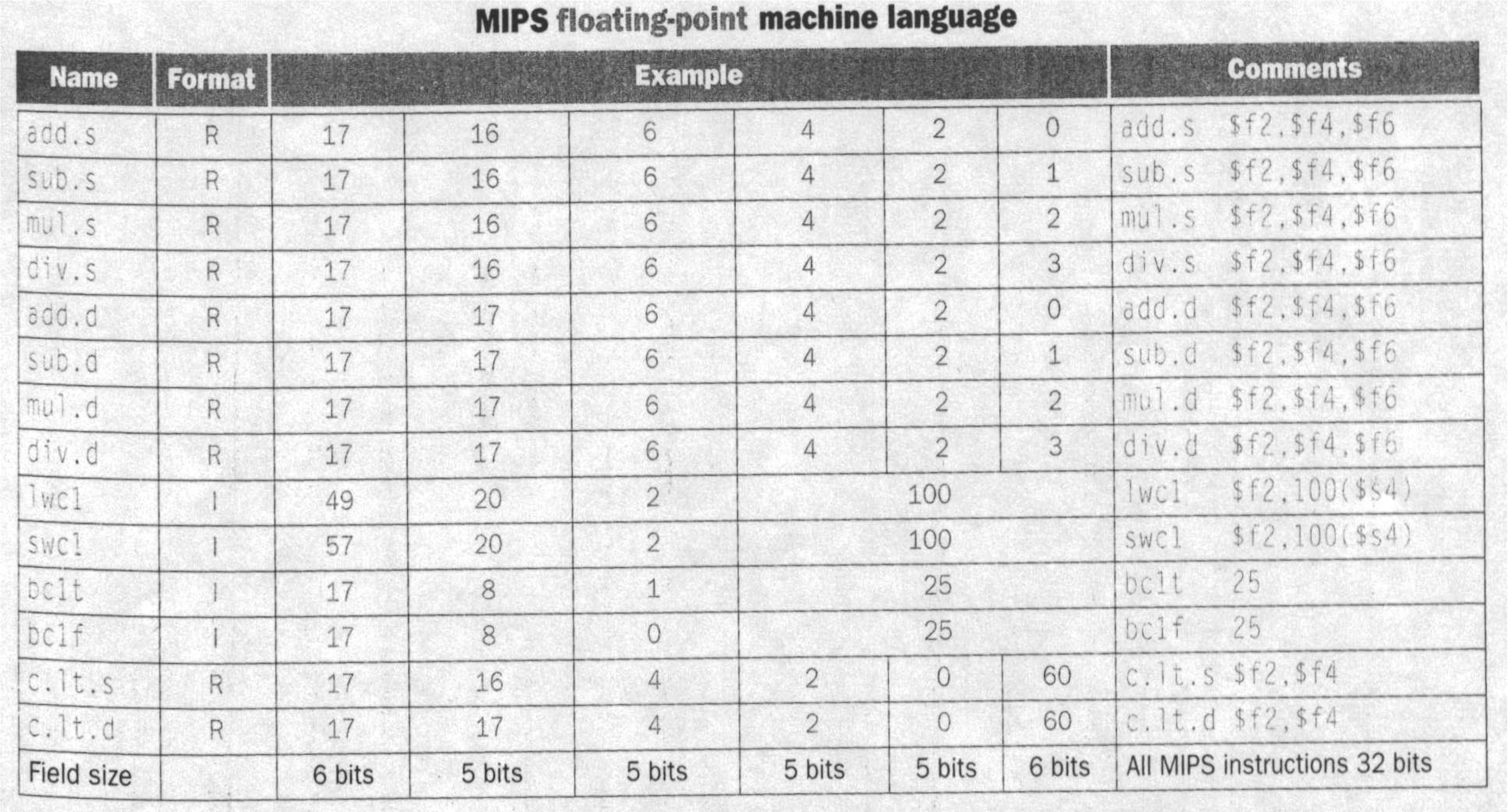
MIPS floating-point operands

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

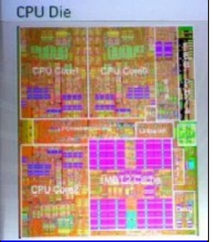
Category	Instruction	Example	Meaning	Comments
Arithmetic	FP add single	add.s $f2,$f4,$f6	$f2 = $f4 + $f6	FP add (single precision)
	FP subtract single	sub.s $f2,$f4,$f6	$f2 = $f4 – $f6	FP sub (single precision)
	FP multiply single	mul.s $f2,$f4,$f6	$f2 = $f4 × $f6	FP. multiply (single precision)
	FP 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



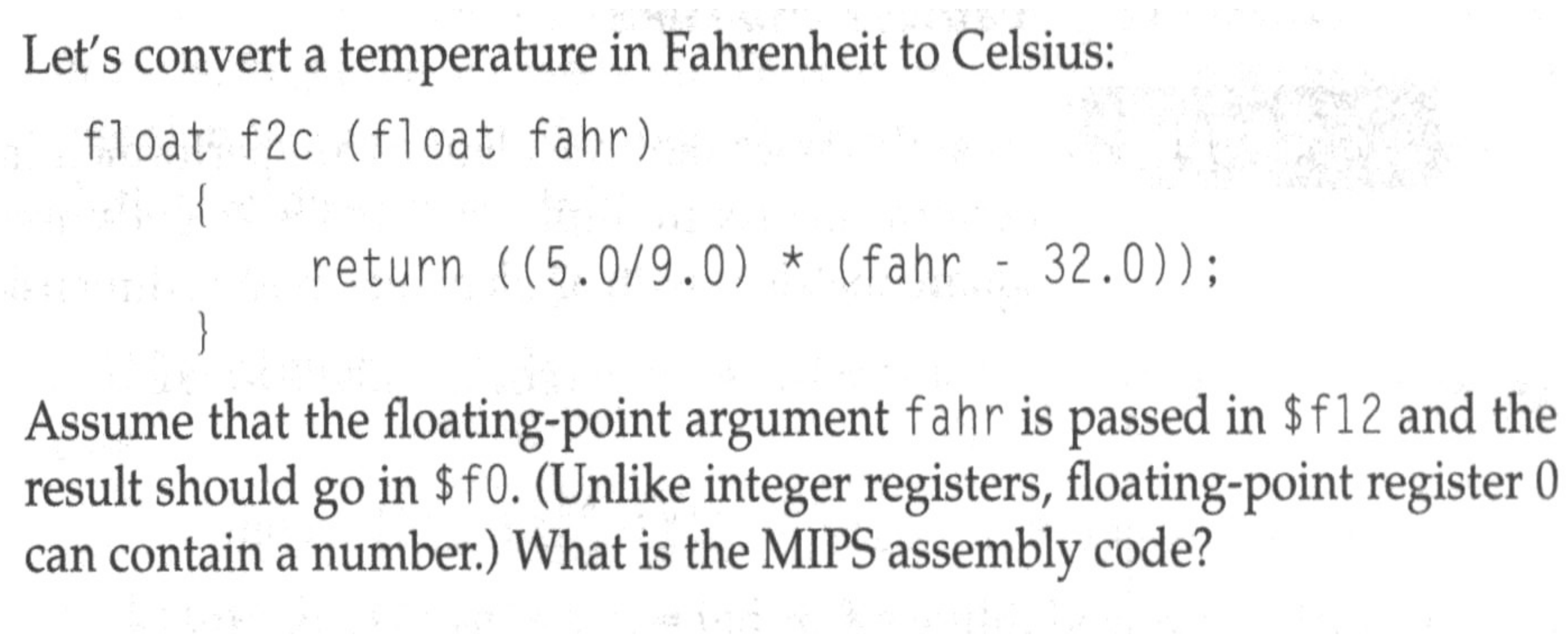


MIPS floating-point machine language

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

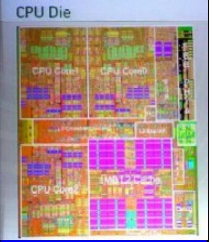


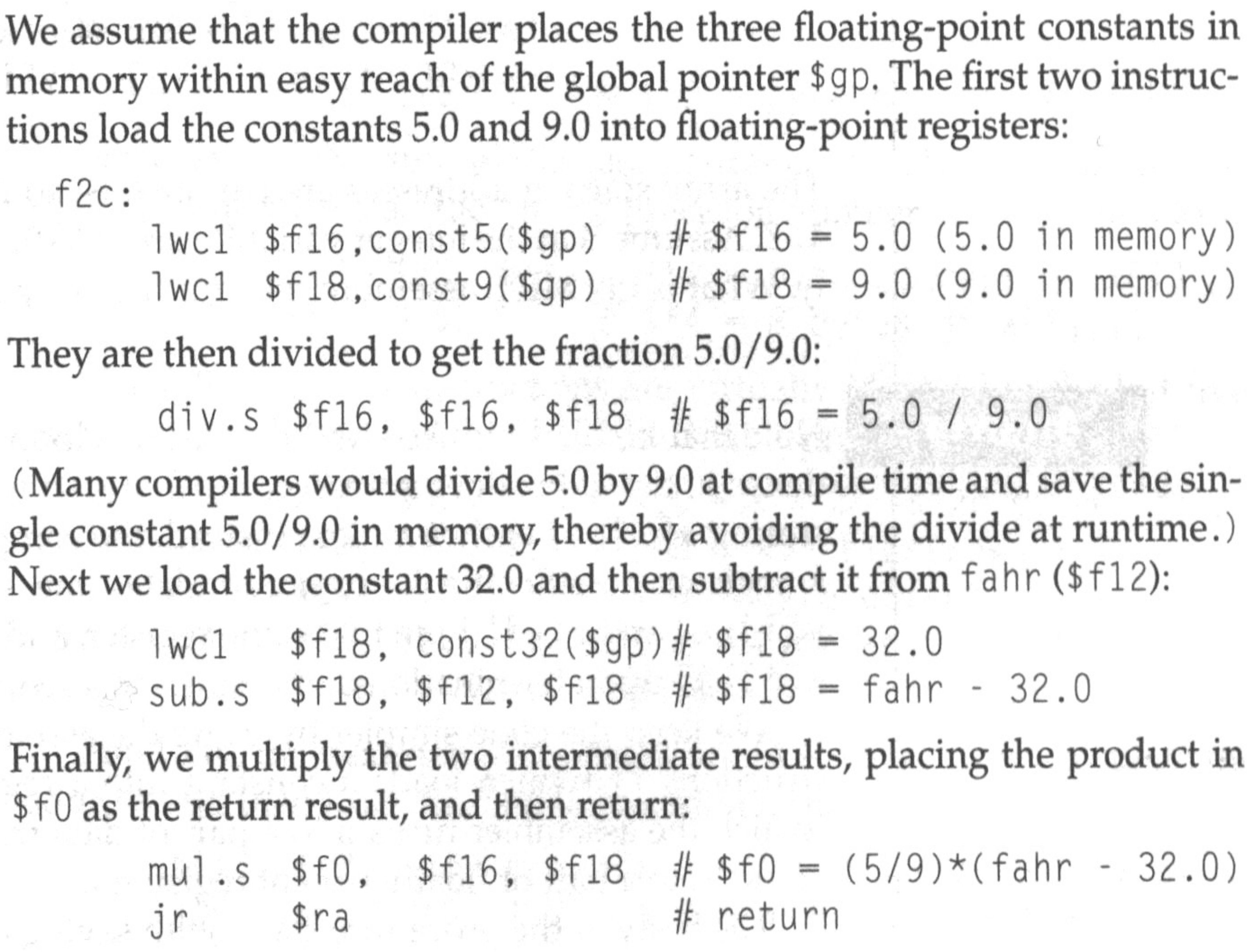
Let's convert a temperature in Fahrenheit to Celsius:

```
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
```

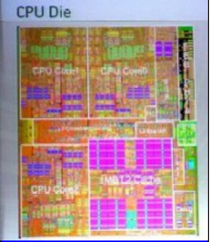
Finally, we multiply the two intermediate results, placing the product in $f0 as the return result, and then return:

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

Floating Point Instructions

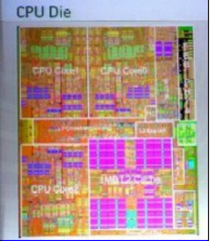
• FP absolute value double	abs.d fd, fs
• FP absolute value single	abs.s fd, fs
• FP addition double	add.d fd, fs, ft
• FP addition single	add.s fd, fs, ft
• Compare equal double	c.eq.d fs, ft
• Compare less than	c.le.d fs, ft
• Convert single to double	cvt.d.s fd, fs
• Convert int to double	cvt.d.w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

See end of text book (appendix A)

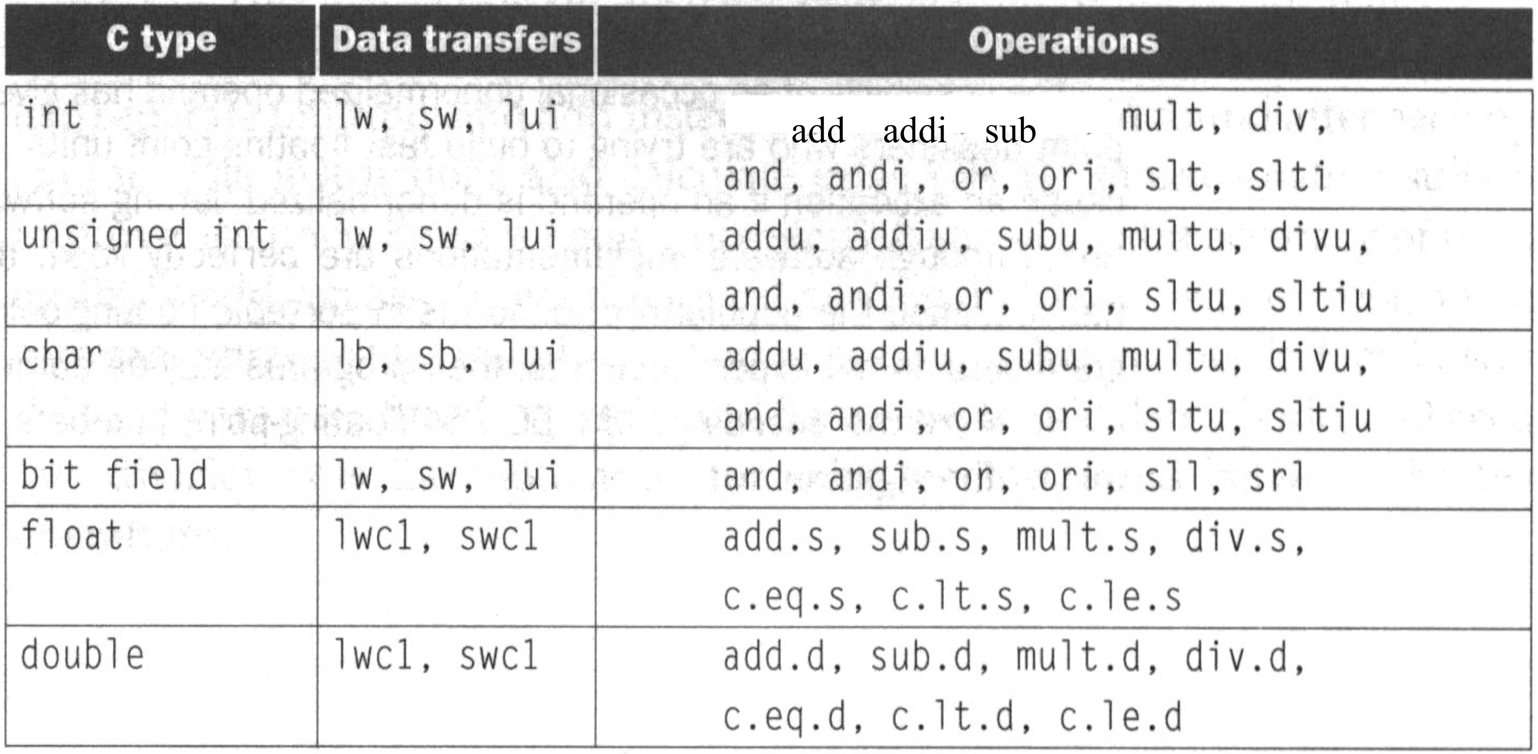


Part 2

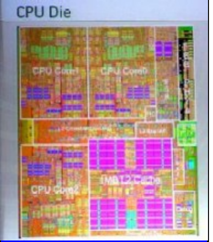
Programming with Data



Typed Instructions

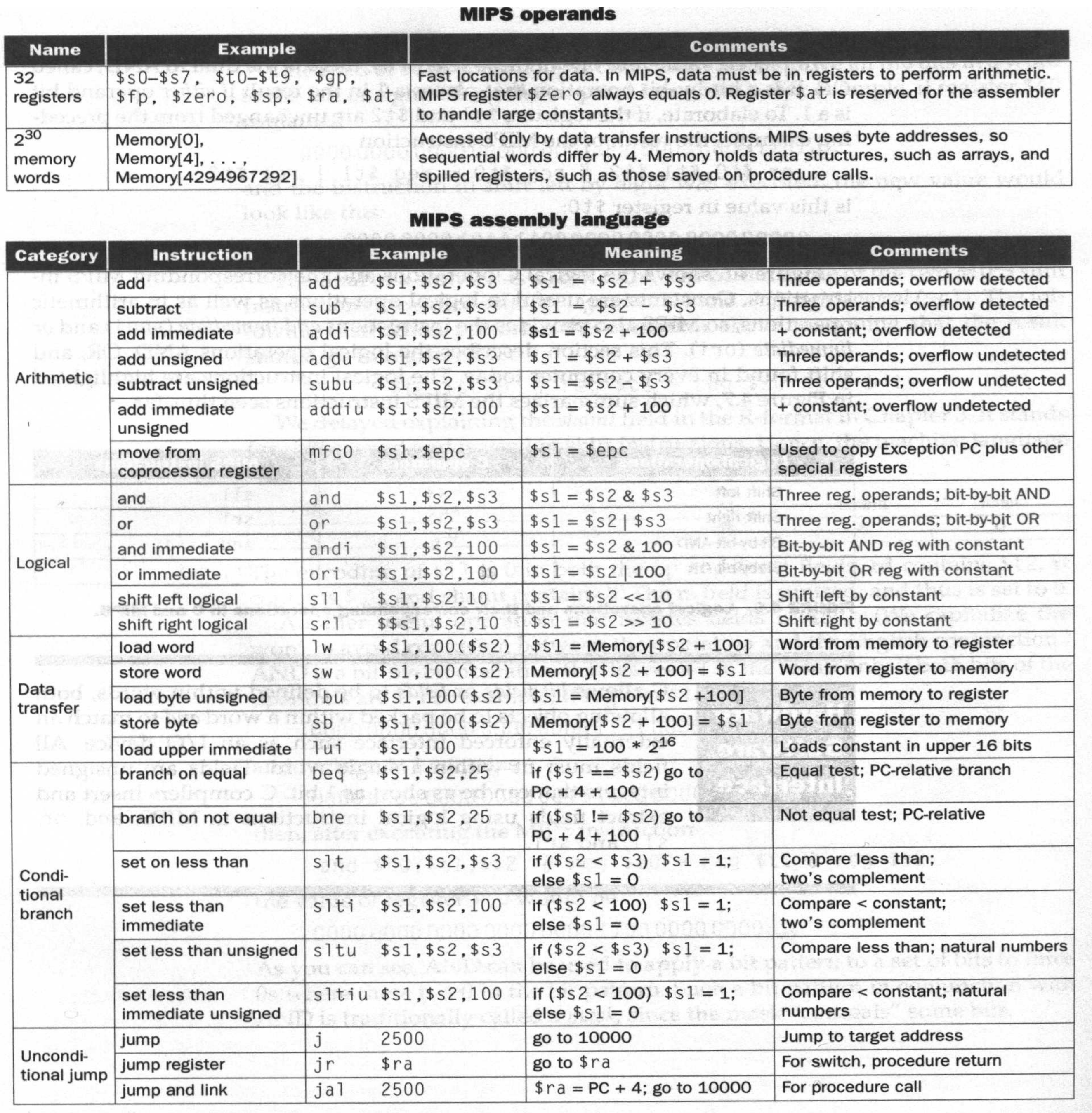


C type	Data transfers	Operations
int	lw, sw, lui	add addi sub mult, div, and, andi, or, ori, slt, slti
unsigned int	lw, sw, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, sltu, sltiu
char	lb, sb, lui	addu, addiu, subu, multu, divu, and, andi, or, ori, sltu, sltiu
bit field	lw, sw, lui	and, andi, 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

What we have seen

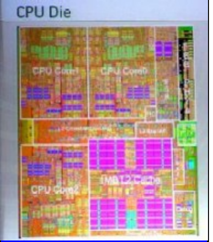


MIPS operands

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
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

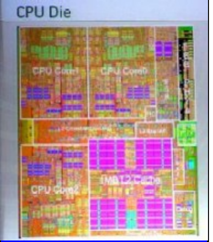


Load Instructions

• 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

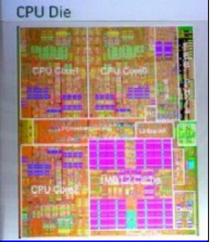
See end of text book (appendix A)



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)

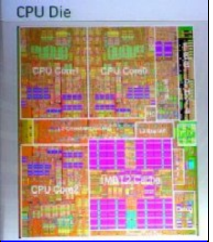


Move Data

Where Z is the co-pro no.

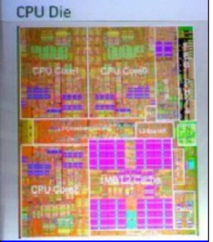
Move	move rdest, rsrc
Move from hi	mfhi rd
Move from lo	mflo rd
Move to hi	mthi rs
Move to lo	mtlo rs
Move from coprocessor-z	mfcz rt, rd
Move double from Co1	mfc1.d rdest, rc1
Move to co-z	mtcz rd, rt

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



Reminder Assembler Data Directives

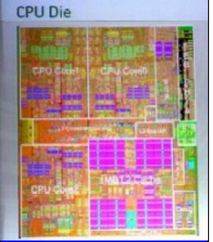
- Syntax:
  - LABEL: DIRECTIVE DATA
- Where:
  - .align n
  - .ascii str
  - .asciiz str
  - .space n
  - .byte b1,b2,…,bn … 8
  - .half h1,h2,…,hn … 16
  - .word w1,w2,…,wn 32
  - .float f1,f2,…,fn 32
  - .double d1,d2,…,dn ….. 64

.data <addr>

.extern sym size ($gp)

.globl sym

.text <addr>



Problems

- Binary search 32-bit integer array

- Traverse a linked list
  - One word integer
  - One word pointer