Instruction Set Architecture (ISA - Ch 2)

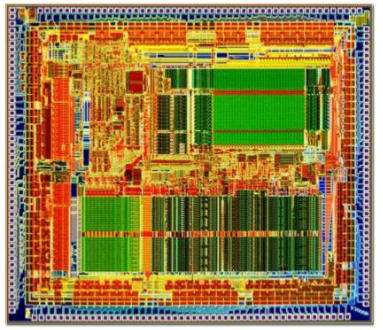
Dr. Rajib Ranjan Maiti (Mtech and PhD at IIT Kharagpur)

CSIS, BITS-Pilani, Hyderabad

MIPS:
Microprocessor without Interlocked Pipeline Stages

MIPS R3000 Processor

- 32-bit 2nd generation commercial processor (1988)
- Led by John Hennessy (Stanford, MIPS Founder)
- 32-64 KB Caches
- 1.2 μm process
- 111K Transistors
- Up to 12-40 MHz
- 66 mm^2 die
- 145 I/O Pins
- V_{DD} = 5 V
- 4 Watts
- SGI Workstations



http://gecko54000.free.fr/?documentations=1988_MIPS_R3000

MIPS Computer Systems Inc.

- MIPS Computer Systems Inc.
  - **Founded in 1984** by a group of researchers from **Stanford University**
  - Including **John L. Hennessy** and **Chris Rowen**.
  - Other principal founders: **Skip Stritter** (formerly a Motorola technologist) and **John Moussouris** (formerly of IBM)

- These researchers had worked on a project called **MIPS**
  - MIPS: ***Microprocessor without Interlocked Pipeline Stages***
  - This is one of the projects that pioneered the **RISC concept**

Operations of the Computer Hardware

- How to perform a = b + c + d +e?
  - add a, b, c # a ← b + c
  - add a, a, d # a ← a + d
  - add a, a, e # a ← a + e

MIPS Registers

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	procedure return values
$a0-$a3	4-7	procedure arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	procedure return address

Basic Components in MIPS

- 32 Registers
  - $s0–$s7 (8), $t0–$t9 (10), $zero, $a0–$a3 (4), $v0–$v1 (2), $gp, $fp, $sp, $ra, $at, $k0 - $k1 (2)
- In MIPS,
  - **data must be in registers** to perform arithmetic;
- **2^{30} memory words**
  - Memory[0], Memory[4], . . . , Memory[4294967292]
  - Accessed only by **data transfer instructions**;
  - MIPS uses byte addresses, so **sequential word addresses differ by 4**;
  - Memory holds **data structures**, **arrays**, and **spilled registers** (when a function has more variables than available registers).

MIPS assembly language: sample instructions

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1, $s2, $s3	$s1 ← $s2 + $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants

MIPS assembly language: sample instructions

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1, $s2, $s3	$s1 ← $s2 + $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits

MIPS assembly language: sample instructions

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1, $s2, $s3	$s1 ← $s2 + $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant

MIPS assembly language: sample instructions

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1, $s2, $s3	$s1 ← $s2 + $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
Cond. branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
Uncond. jump	jump	j 2500	go to 10000	Jump to target address

Compiling a Complex C Assignment into MIPS

- A somewhat complex statement contains five variables f, g, h, i, and j:
  - f = (g + h) – (i + j);
- What might a C compiler produce?

Compiling a Complex C Assignment into MIPS

- A somewhat complex statement contains five variables f, g, h, i, and j:
  - f = (g + h) – (i + j);
- What might a C compiler produce?
- Ans.
  - add t0, g, h # temporary variable t0 contains g + h
  - add t1, i, j # temporary variable t1 contains i + j
  - sub f, t0, t1 # f gets t0 – t1, which is (g + h) – (i + j)

Compiling a Complex C Assignment into MIPS

- A somewhat complex statement contains the fi ve variables f, g, h, i, and j:
  - f = (g + h) – (i + j);
- What might a C compiler produce?
- Ans.
  - add t0, g, h # temporary variable t0 contains g + h
  - add t1, i, j # temporary variable t1 contains i + j
  - sub f, t0, t1 # f gets t0 – t1, which is (g + h) – (i + j)
- For MIPS,
  - Assume that the variables f, g, h, i, and j are assigned to the registers $s0, $s1, $s2, $s3, and $s4, respectively.
- What is the compiled MIPS code?

Compiling a Complex C Assignment into MIPS

- A somewhat complex statement contains the fi ve variables f, g, h, i, and j:
  - f = (g + h) – (i + j);
- What might a C compiler produce?
- Ans.
  - add t0, g, h # temporary variable t0 contains g + h
  - add t1, i, j # temporary variable t1 contains i + j
  - sub f, t0, t1 # f gets t0 – t1, which is (g + h) – (i + j)
- For MIPS,
  - Assume that the variables f, g, h, i, and j are assigned to the registers $s0, $s1, $s2, $s3, and $s4, respectively.
- What is the compiled MIPS code?
- Ans.
  - add $t0, $s1, $s2 # register $t0 contains g + h
  - add $t1, $s3, $s4 # register $t1 contains i + j
  - sub $s0, $t0, $t1 # f gets $t0 – $t1, which is (g + h)–(i + j)

Memory access or pointers

- Compile this C assignment statement:
  - g = h + A[8];

Memory access or pointers

- Compile this C assignment statement:
  - g = h + A[8];
- In MIPS
  - Assume: the starting address, or *base address,* of the array is in **$s3**
  - A[8] is actually at (the base of the array A, i.e., $s3) + (index, i.e., 8)
  - So,
  - lw $t0, 8($s3) # Temporary register $t0 <- A[8]
  - add $s1, $s2, $t0 # assume h is in $s2; so, g = h + A[8]

Memory access or pointers

- Compile this C assignment statement:
  - g = h + A[8];
- In MIPS
  - lw $t0, 8($s3) # Temporary register $t0 <- A[8]
  - add $s1, $s2, $t0 # assume h is in $s2; so, g = h + A[8]
- How about this?
  - What is the MIPS assembly code for the C assignment statement below?
  - A[12] = h + A[8];

Binary to Decimal Conversion

- What is the decimal value of this 32-bit two's complement number?
  - $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100)_{two}$

Binary to Decimal Conversion

- What is the decimal value of this 32-bit two's complement number?
  - $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100)_{two}$
- Ans.
  - $(-4)_{10}$

Sign Extension

- Convert 16-bit binary versions of 2_{10} and -2_{10} to 32-bit binary numbers.

Sign Extension

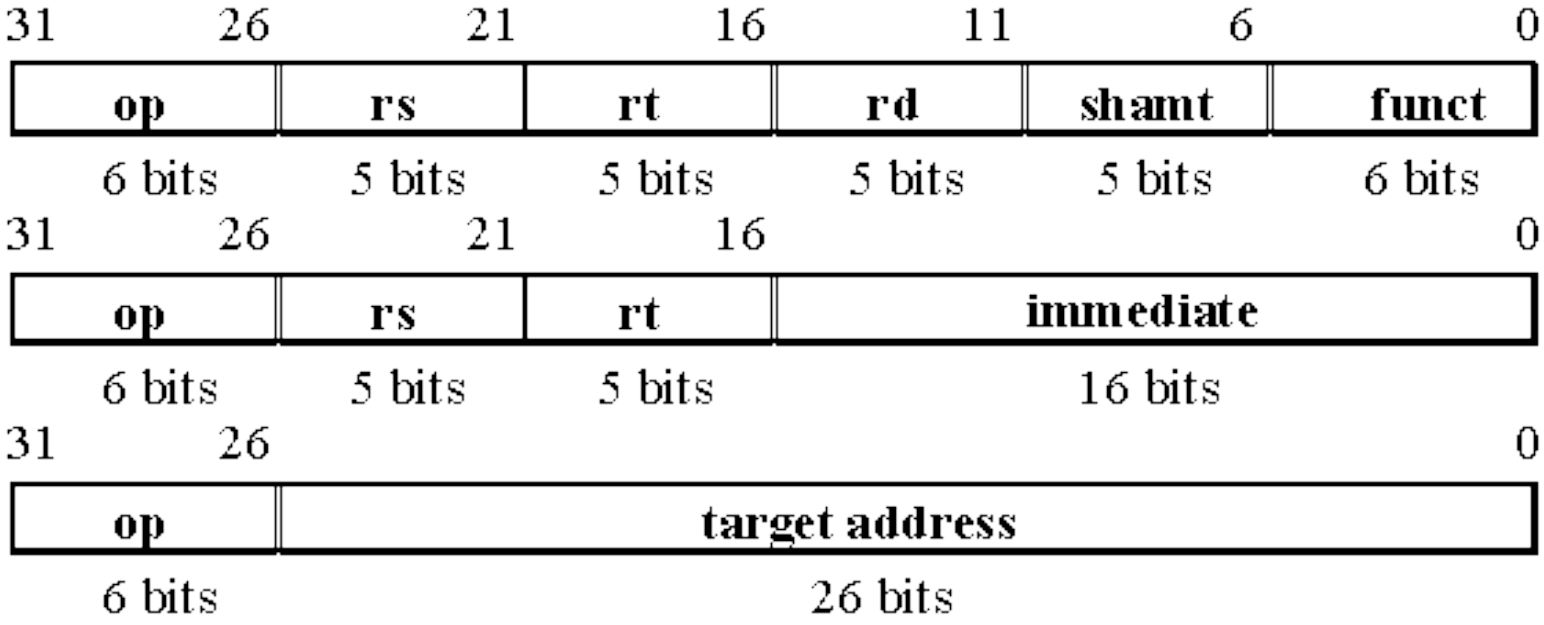
- Convert 16-bit binary versions of 2_{10} and -2_{10} to 32-bit binary numbers.
- Ans.
  - B0 B1 B2 B3 B4 B5 B6 B7
  - 0000 0000 0000 0000 0000 0000 0000 $0010_2 = 2_{10}$
  - 1111 1111 1111 1111 1111 1111 1111 $1110_2 = -2_{10}$

Representing Instructions in MIPS 32 bits

least significant bit (LSB): The rightmost bit in a MIPS word.
most significant bit (MSB): The left most bit in a MIPS word.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

Representation of 1011_2 in MIPS 32 bit word



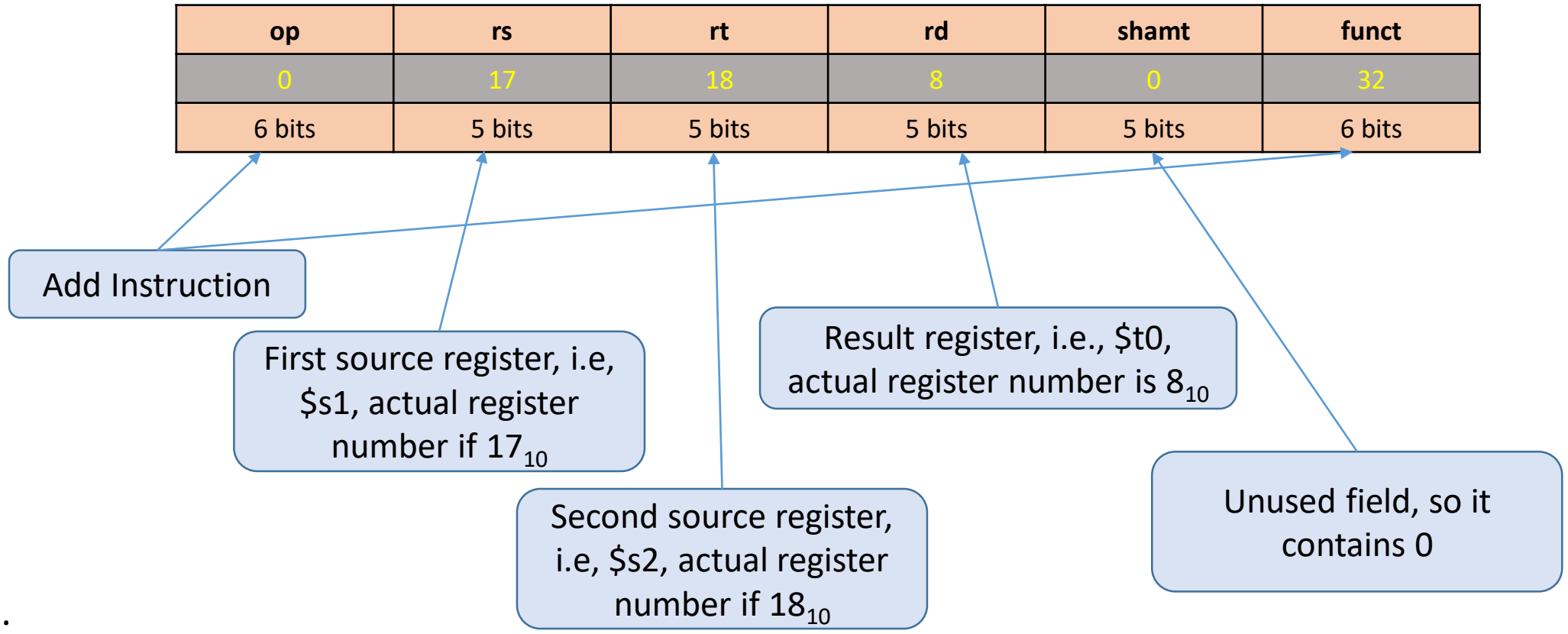
R-Type: arithmetic instructions

I-Type: Transfer, branch, immed

J-Type: jump

Representing Instructions in MIPS

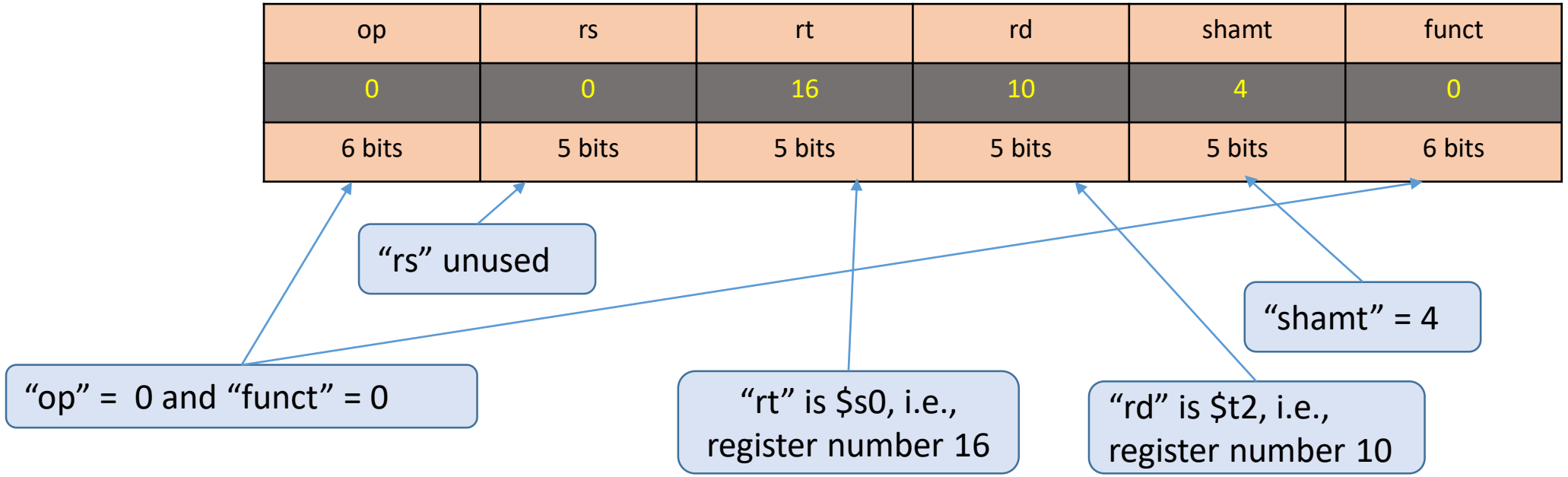
- Consider
  - **add $t0,$s1,$s2 #$t0 ← Ss1 + $s2**
- What is its binary 32 bit representation?



- Ans.
  - Convert all these decimal numbers to binary
  - this format is called *R-type* (for register) or *R-format*.

Logical Operations

- Let
  - $s0 = 0000 0000 0000 0000 0000 0000 0000 $1001_{two} = 9_{ten}$
- Shifting left by 4 will result
  - $s0 = 0000 0000 0000 0000 0000 0000 1001 $0000_{two} = 144_{ten}$
- MIPS instruction for this
  - **sll $t2, $s0, 4 # shift logical left, $t2 = $s0 << 4 bits**
- The instruction in 32 bit



Instructions for Making Decisions

- What is the compiled MIPS code for this C *if* statement?
  - if (i == j)
    - f = g + h;
  - else
    - f = g – h;

Instructions for Making Decisions

- What is the compiled MIPS code for this C *if* statement?
  - if (i == j)
    - f = g + h;
  - else
    - f = g – h;
- Ans.
  - 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 = g – h (skipped if i = j)
  - Exit:

Instructions for Making Decisions

- How about this C code?
  - **while (save[i] == k)**
  - **i += 1;**
- Assume that i and k are in $s3 and $s5 and the base of the array is in $s6

Instructions for Making Decisions

- How about this C code?
  - **while (save[i] == k)**
  - **i += 1;**
- Assume that i and k are in $s3 and $s5 and the base of the array is in $s6

```
Loop: sll $t1,$s3,2      # Temp reg $t1 = i * 4
      add $t1,$t1,$s6    # $t1 = address of save[i]
      lw $t0,0($t1)      # Temp reg $t0 = save[i]
      bne $t0,$s5, Exit  # go to Exit if save[i] ≠ k
      addi $s3,$s3,1     # i = i + 1
      j Loop             # go to Loop
Exit:
```

Supporting Procedures in Computer Hardware

- A procedure follow these six steps:
  - 1. Put parameters in a place where the procedure can access them.
  - 2. Transfer control to the procedure.
  - 3. Acquire the storage resources needed for the procedure.
  - 4. Perform the desired task.
  - 5. Put the result value in a place where the calling program can access it.
  - 6. Return control to the point of origin, since a procedure can be called from several points in a program.

- MIPS convention for procedure call :
  - $a0–$a3: four registers to pass arguments
  - $v0–$v1: two registers to return values
  - $ra: one register to hold return address

Supporting Procedures in Computer Hardware

- Lets Consider C procedure:
  - int leaf_example (int g, int h, int i, int j) {
    - int f;
    - f = (g + h) – (i + j);
    - return f;
  - }
- What is the compiled MIPS assembly code?

Supporting Procedures in Computer Hardware

- Lets Consider C procedure:
  - int leaf_example (int g, int h, int i, int j) {
    - int f;
    - f = (g + h) – (i + j);
    - return f;
  - }
- What is the compiled MIPS assembly code?

- The arguments : g, h, i, and j are in $a0, $a1, $a2, and $a3,
- Local variable : f is in $s0
- MIPS Code:

- leaf_example: addi $sp, $sp, –12 #adjust stack
  - sw $t1, 8($sp) # save $t1 for use afterwards
  - sw $t0, 4($sp) # save $t0 for use afterwards
  - sw $s0, 0($sp) # save $s0 for use afterwards
  - add $t0,$a0,$a1 # register $t0 contains g + h
  - add $t1,$a2,$a3 # register $t1 contains i + j
  - sub $s0,$t0,$t1 # f = $t0 – $t1, which is (g + h)–(i + j)
  - add $v0,$s0,$zero # returns f ($v0 = $s0 + 0)
  - lw $s0, 0($sp) # restore $s0 for caller
  - lw $t0, 4($sp) # restore $t0 for caller
  - lw $t1, 8($sp) # restore $t1 for caller
  - addi $sp,$sp,12 # adjust stack to delete 3 items
  - jr $ra # jump back to calling routine

Communicating with People

- **ASCII versus Binary Numbers**
- How much does storage increase if the number 1 billion is represented in ASCII versus a 32-bit integer?
  - Ans. One billion is 1,000,000,000, so it would take 10 ASCII digits, each 8 bits long.
- How MIPS deals with byte?
  - **lb $t0, 0($sp)** # Read byte from memory
  - **sb $t0, 0($gp)** # Write byte to memory

- Consider following C codes
  - void strcpy (char x[], char y[]) {
    - int i;
    - i = 0;
    - while ((x[i] = y[i]) != '\0') /* copy & test byte */
      - i += 1;
  - }
- What is the MIPS assembly code?

MIPS Addressing for I-Type instructions

- Although constants can fit in a 16-bit field,
  - sometimes they are bigger.

- Consider
  - **lui $t0, 255 # load upper immediate - $t0 is register 8:**
- 32 bit format of the instruction

op	rs	rd	Immid.
001111	00000	01000	0000 0000 1111 1111
6 bits	5 bits (unused)	5 bits	16 bits

- Content of $t0 after execution

0000 0000 1111 1111	0000 0000 0000 0000
Upper 16 bits	Lower 16 bits

MIPS Addressing I-Type Instructions

- bne $s0,$s1,Exit # go to Exit if $s0 ≠ $s1

Op	rs	rt	Immid.
5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

MIPS Addressing J-Type Instructions

- **Addressing in Branches and Jumps**
  - j 10000 # go to location 10000

Op	Immid.
2	10000
6 bits	26 bits


Thank You