MIPS Instruction Set

Some charts provided by
Morgan Kauffman Pubs

The MIPS Instruction Set

- Used as the example throughout the book
- Stanford MIPS commercialized by MIPS Technologies (www.mips.com)
- Large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …
- Typical of many modern ISAs
  - See MIPS Reference Data tear-out card, and Appendixes B and E

Some charts provided by
Morgan Kauffman Pubs

Arithmetic Operations

- All arithmetic instructions are 3-address
  - Two sources and one destination

```
add a, b, c   # a ← b + c
```

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 3

Register Operands

- Arithmetic instructions use register operands and destination
- MIPS has a 32 × 32-bit register file
  - Numbered 0 to 31
  - 32-bit data called a “word”
- Assembler names
  - $t0, $t1, …, $t9
  - $s0, $s1, …, $s7
  - Compilers typically use $tx for temps, $sx for saved values – there’s no difference and we can use them for anything

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 4

Register Operand Example

- C code:

```
f = (g + h) - (i + j);
```

  - f, …, j in $s0, …, $s4

- Compiled MIPS code:

```
add $t0, $s1, $s2 # $t0 ← $s1+$s2
add $t1, $s3, $s4 # $t1 ← $s3+$s4
sub $s0, $t0, $t1 # $s0 ← $t0-$t1
```

Some charts provided by
Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 5

Memory Operands

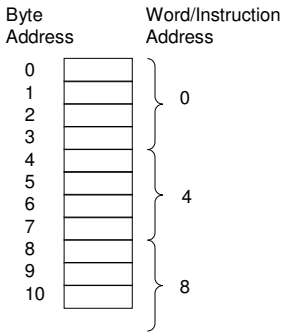
- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory

Some charts provided by
Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 6

Memory Addressing

- Memory is byte addressed
  - Each address identifies an 8-bit byte
- Words are aligned in memory
  - Address must be a multiple of 4
- MIPS is Big Endian
  - Most-significant byte at least address of a word
  - *c.f.* Little Endian: least-significant byte at least address



Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 7

Load and Store Instructions

- Use Base Addessing Mode (aka “offset” mode)
- lw rd, offset (rs) # load word from memory
  - where:
    - rd is the destination register
    - rs contains a base address
    - offset such that base address + offset = address you want
- sw rd, offset (rs) # store word to memory

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 8

Memory Operand Example 1

- C code:

```
g = h + A[8];
```

  - Assume the value of g is in $s1, h in $s2, the base address of A in $s3

- Compiled MIPS code:

```
lw  $t0, 32($s3)    # load word
add $s1, $s2, $t0
```

Some charts provided by Morgan Kauffman Pubs

Memory Operand Example 2

- C code:

```
A[12] = h + A[8];
```

  - h in $s2, base address of A in $s3

- Compiled MIPS code:

```
lw  $t0, 32($s3)    # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)    # store word
```

Some charts provided by Morgan Kauffman Pubs

Immediate Operands

- Constant data specified in an instruction
  `addi $s3, $s3, 4`
- No subtract immediate instruction
  - use a negative constant:
    `addi $s2, $s1, -1`
  - `Reg $zero gives the constant 0`

Some charts provided by
Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 11

Unsigned Binary Integers

- Given an n-bit number

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: 0 to $+2^n - 1$
- Example
  - $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$
    $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$
- Using 32 bits
  - 0 to +4,294,967,295

Some charts provided by
Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 12

2s-Complement Signed Integers

- Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: -2^{n-1} to $+2^{n-1} - 1$
- Example
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$
- Using 32 bits
  - –2,147,483,648 to +2,147,483,647

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 13

Sign Extension

- Representing a number using more bits
  - Preserve the numeric value
- In MIPS instruction set
  - `addi`: extend immediate value
  - `lb`, `lh`: extend loaded byte/halfword
  - `beq`, `bne`: extend the displacement
- Replicate the sign bit to the left
  - c.f. unsigned values: extend with 0s
- Examples: 8-bit to 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 14

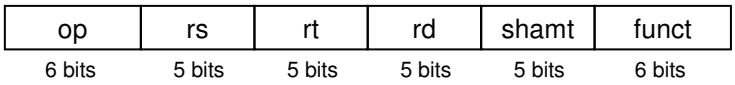
Machine code (encoding)

- MIPS instructions
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
- Register numbers
  - $t0 – $t7 are reg's 8 – 15
  - $t8 – $t9 are reg's 24 – 25
  - $s0 – $s7 are reg's 16 – 23

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 15

MIPS R-format Instructions



- Instruction fields
  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount (00000 for now)
  - funct: function code (extends opcode)

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 16

R-format Example

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

op	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$00000100011001001000000000100000_2 = 02324020_{16}$

Some charts provided by Morgan Kauffman Pubs

MIPS I-format Instructions

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Immediate arithmetic and load/store instructions
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$
  - Address: offset added to base address in rs

Some charts provided by Morgan Kauffman Pubs

Logical Operations

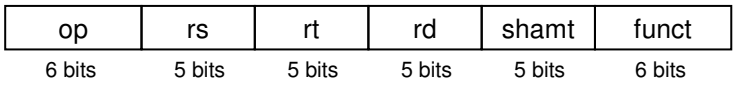
- Instructions for bitwise manipulation

Operation	C	Java	MIPS
Shift left	<<	<<	`sll`
Shift right	>>	>>>	`srl`
Bitwise AND	&	&	`and, andi`
Bitwise OR	|	|	`or, ori`
Bitwise NOT	~	~	`nor`

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 19

Shift Operations



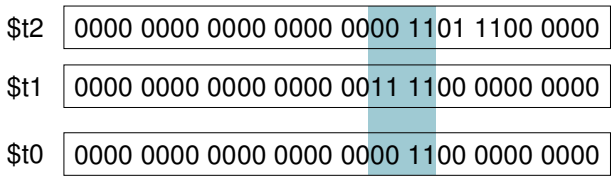
- shamt: number of positions to shift
- Shift left logical
  - Shift left and fill with 0 bits
  - `sll` by *i* bits multiplies by 2^i
- Shift right logical
  - Shift right and fill with 0 bits
  - `srl` by *i* bits divides by 2^i (unsigned only)

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 20

AND Operations

- `and $t0, $t1, $t2`

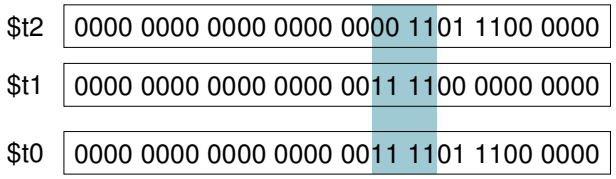


Some charts provided by
Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 21

OR Operations

- `or $t0, $t1, $t2`



Some charts provided by
Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 22

NOT Operations

- MIPS has NOR 3-operand instruction
  - a NOR b == NOT (a OR b)

```
nor $t0, $t1, $zero
```

$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	1111 1111 1111 1111 1100 0011 1111 1111

Some charts provided by
Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 23

Conditional Operations

- Branch to a labeled instruction if a condition is true, else, continue sequentially
  - `beq rs, rt, L1`
    - if (rs == rt) branch to instruction labeled L1
  - `bne rs, rt, L1`
    - if (rs != rt) branch to instruction labeled L1
- `j L1`
  - unconditional jump to instruction labeled L1

Some charts provided by
Morgan Kauffman Pubs

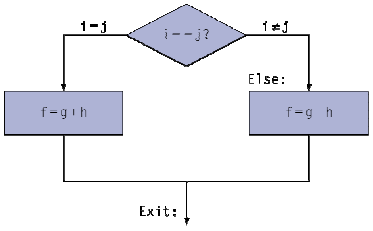
CS613 s12 -- MIPS Instruction Set — 24

Compiling If Statements

- C code:

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

  - f, g, … in $s0, $s1, …



- Compiled MIPS code:

```
      bne $s3, $s4, Else
      add $s0, $s1, $s2
      j   Exit
Else: sub $s0, $s1, $s2
Exit: …
```

Some charts provided by Morgan Kauffman Pubs

Compiling Loop Statements

- C code:

```
while (save[i] == k) i += 1;
```

  - i in $s3, k in $s5, address of save in $s6

- Compiled MIPS code:

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

Some charts provided by Morgan Kauffman Pubs

More Conditional Operations

- Set result to 1 if a condition is true
  - Otherwise, set to 0
- `slt rd, rs, rt`
  - if (rs < rt) rd = 1; else rd = 0;
- `slti rt, rs, constant`
  - if (rs < constant) rt = 1; else rt = 0;
- Use in combination with `beq`, `bne`

```
slt $t0, $s1, $s2  # if ($s1 < $s2)
bne $t0, $zero, L  #   branch to L
```

Some charts provided by Morgan Kauffman Pubs

Branch Instruction Design

- Why not `blt`, `bge`, etc?
- Hardware for <, ≥, … slower than =, ≠
  - Combining with branch involves more work per instruction, requiring a slower clock
  - All instructions penalized!
- `beq` and `bne` are the common case

Some charts provided by Morgan Kauffman Pubs

Signed vs. Unsigned

- Signed comparison: `slt`, `slti`
- Unsigned comparison: `sltu`, `sltui`
- Example
  - $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  - $s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - `slt  $t0, $s0, $s1  # signed`
    - −1 < +1 ⇒ $t0 = 1
  - `sltu $t0, $s0, $s1  # unsigned`
    - +4,294,967,295 > +1 ⇒ $t0 = 0

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 29

Byte/Halfword Operations

- Could use bitwise operations
- MIPS byte/halfword load/store

`lb rt, offset(rs)` `lh rt, offset(rs)`

- Sign extend to 32 bits in rt

`lbu rt, offset(rs)` `lhu rt, offset(rs)`

- Zero extend to 32 bits in rt

`sb rt, offset(rs)` `sh rt, offset(rs)`

- Store just rightmost byte/halfword

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 30

String Copy Example

- C code (naïve):
  - Null-terminated string

```
void strcpy (char x[], char y[])
{ int i;
  i = 0;
  while ((x[i]=y[i])!='\0')
    i += 1;
}
```

  - Addresses of x, y in $a0, $a1
  - i in $s0

Some charts provided by
Morgan Kauffman Pubs

String Copy Example

- MIPS code:

```
strcpy:
    addi $sp, $sp, -4       # adjust stack for 1 item
    sw   $s0, 0($sp)        # save $s0
    add  $s0, $zero, $zero  # i = 0
L1: add  $t1, $s0, $a1      # addr of y[i] in $t1
    lbu  $t2, 0($t1)        # $t2 = y[i]
    add  $t3, $s0, $a0      # addr of x[i] in $t3
    sb   $t2, 0($t3)        # x[i] = y[i]
    beq  $t2, $zero, L2     # exit loop if y[i] == 0
    addi $s0, $s0, 1        # i = i + 1
    j    L1                 # next iteration of loop
L2: lw   $s0, 0($sp)        # restore saved $s0
    addi $sp, $sp, 4        # pop 1 item from stack
    jr   $ra                # and return
```

Some charts provided by
Morgan Kauffman Pubs

32-bit Constants

- Most constants are small
  - 16-bit immediate is sufficient
- For the occasional 32-bit constant

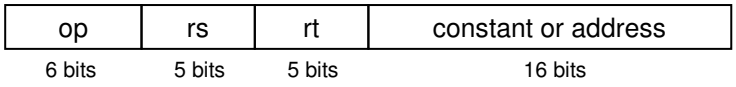
  `lui rt, constant`
  - Copies 16-bit constant to left 16 bits of rt
  - Clears right 16 bits of rt to 0

`lhi $s0, 61`	0000 0000 0111 1101	0000 0000 0000 0000
`ori $s0, $s0, 2304`	0000 0000 0111 1101	0000 1001 0000 0000

Some charts provided by Morgan Kauffman Pubs

Branch Addressing

- Branch instructions specify
  - Opcode, two registers, target address
- Most branch targets are near branch
  - Forward or backward

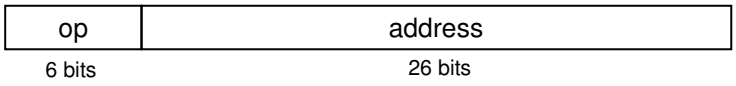


- PC-relative addressing
  - Target address = PC + offset × 4
  - PC already incremented by 4 by this time

Some charts provided by Morgan Kauffman Pubs

Jump Addressing

- Jump (`j` and `jal`) targets could be anywhere in text segment
  - Encode full address in instruction



- (Pseudo)Direct jump addressing
  - Target address = $PC_{31...28}$: (address × 4)

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 35

Target Addressing Example

- Loop code from earlier example
  - Assume Loop at location 80000

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

80000	0	0	19	9	4	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024						

Some charts provided by Morgan Kauffman Pubs

CS613 s12 -- MIPS Instruction Set — 36

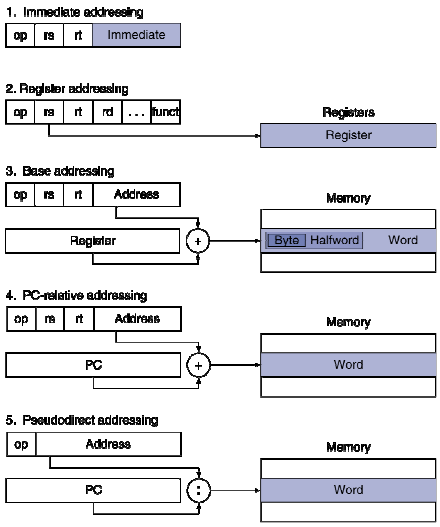
Branching Far Away

- If branch target is too far to encode with 16-bit offset, assembler rewrites the code
- Example

```
        beq $s0,$s1, L1
              ↓
        bne $s0,$s1, L2
        j L1
L2:     …
```

Some charts provided by Morgan Kauffman Pubs

Addressing Mode Summary



Some charts provided by Morgan Kauffman Pubs