Computer Architecture

Prof. Dr. Nizamettin AYDIN

naydin@yildiz.edu.tr
nizamettinaydin@gmail.com

http://www.yildiz.edu.tr/~naydin

MIPS Instruction Set-I

Outline

- MIPS Instruction Set
  - Overview
  - MIPS operands
    - Register operands
    - Memory operands
    - Immediate operands
  - MIPS instruction formats
  - MIPS operations
    - Aritmetic operations
    - Logical operations
  - Hexadecimal notation

Instructions: Overview

- Language of the machine
- More primitive than higher level languages,
  - e.g., no sophisticated control flow such as while or for loops
- Very restrictive
  - e.g., MIPS arithmetic instructions
- MIPS instruction set architecture
  - inspired most architectures developed since the 80's
    - used by NEC, Nintendo, Silicon Graphics, Sony
  - Design goals
    - maximize performance and minimize cost and reduce design time

MIPS operands

Name	Example	Comments
32 registers	$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic, register $zero always equals 0, and register $at is reserved by 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 word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

- Register operands
- Memory operands
- Immediate operands

MIPS operands

- Register Operands
  - Arithmetic instructions use register operands
    - MIPS has a 32 × 32-bit register file
  - Assembler names of registers
    - $t0, $t1, …, $t9 for temporary values
    - $s0, $s1, …, $s7 for saved variables
- Example
  - C code: `A = B + C`
  - MIPS code: `add $s0, $s1, $s2`

Copyright 2000 N. AYDIN. All rights reserved.

MIPS operands

- Memory Operands
  - Processor can only keep small amount of data in registers
  - Main memory used for composite data
    - Arrays, structures, dynamic data
  - MIPS has two basic data transfer instructions for accessing memory
    - Load values from memory into registers
    - **Store** result from register to memory
  - Memory is byte addressed
    - Each address identifies an 8-bit byte
  - In MIPS, arithmetic operations work only on registers
    - Compiler issues load/store instructions to get the operands to place them in registers

7

MIPS operands

- Memory Operand example 1:
  - C code:

    g = h + A[8];
    - g in $s1, h in $s2, address of A in $s3
  - Compiled MIPS code:
    - Index 8 requires offset of 32
      - 4 bytes per word

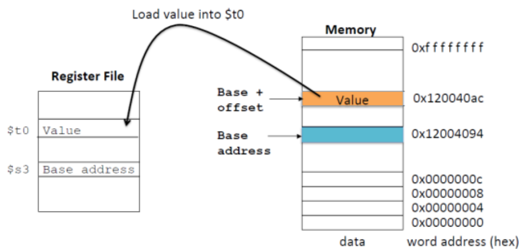
```
lw   $t0, 32($s3)   # load word
add $s1, $s2, $t0
```

offset — base register

8

MIPS operands

- Load Instruction illustrated



9

MIPS operands

- Memory Operand example 2:
  - C code:

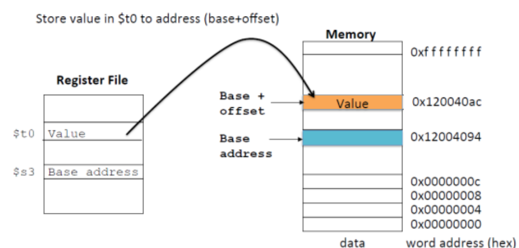
    A[12] = h + A[8];
    - h in $s2, base address of A in $s3
  - Compiled MIPS code:
    - Index 8 requires offset of 32

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

10

MIPS operands

- Store Instruction illustrated:



11

MIPS operands

- Register vs Memory
  - Registers are faster to access than memory
    - Access to registers takes 1 cycle, whereas to memory takes 100s of cycles
  - Operating on memory data requires loads and stores
    - More instructions to be executed
  - Compiler must use registers for variables as much as possible
    - Only spill to memory for less frequently used variables
    - Register optimization is important!
  - Registers are faster than main memory
    - But register files with more locations are slower
      - (e.g. a 64 Word file could be as 50% slower than a 32 word file)

12

Copyright 2000 N. AYDIN. All rights reserved.

MIPS operands

- Immediate Operands
  - Constant data specified in an instruction
    addi $s3, $s3, 4
  - No subtract immediate instruction
    - Just use a negative constant
    addi $s2, $s1, -1
- Constant Zero
  - MIPS register 0 ($zero) is the constant 0
    - Cannot be overwritten
  - Useful for common operations
    - e.g., move between registers
    add $t2, $s1, $zero

13

Question

- In the MIPS code below

```
lw    $v1,  0($a0)
addi  $v0,  $v0,  1
sw    $v1,  0($a1)
addi  $a0,  $a0,  1
```

1. How many times is instruction memory accessed?
2. How many times is data memory accessed?
   - (Count only accesses to memory, not registers.)
3. How many times is register file accessed?
4. Which ones are read, which ones are write to the register file?

14

Question- Solution

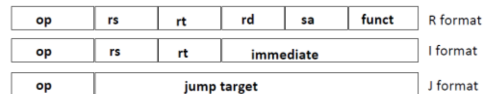
1. 4 times instruction memory
   - Because every instruction needs to be fetched (read) from memory
2. 2 times data memory
   - One for lw (read), one for sw (write)
3. 8 times register file is accessed
4.

lw $v1, 0($a0)	a0 is read, v1 is written into
addi $v0, $v0, 1	v0 is read, v0 is written into
sw $v1, 0($a1)	a1 is read and v1 is read
addi $a0, $a0, 1	a0 is read, a0 is written into

15

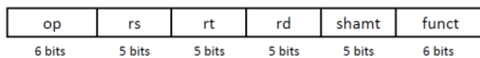
Representing Instructions

- MIPS-32 instructions
  - Encoded as 32-bit instruction words
  - Very Regular and Very Simple!
- 3 Instruction Formats
  - all 32 bits wide



16

MIPS R-format Instructions

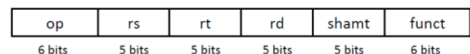


6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits

- 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)
    - How many different types of R-format instructions can there be?

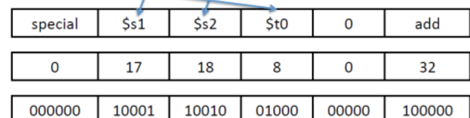
17

R-format example 1

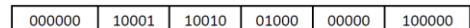


6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits

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



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



$00000010001100100100000000100000_2$

18

Copyright 2000 N. AYDIN. All rights reserved.

R-format example 2

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

- shamt: how many 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)

MIPS Register Convention

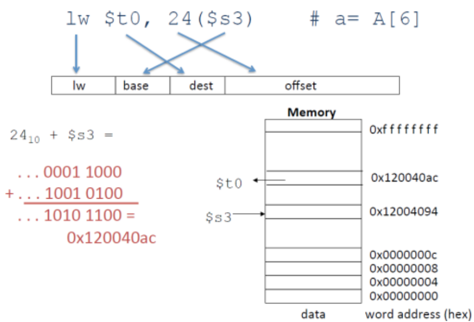
Name	Register Number	Usage	Preserve on call?
$zero	0	constant 0 (hardware)	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	yes
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values	yes
$t8 - $t9	24-25	temporaries	No
$k0 -$k1	26-27	reserved for OS	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return addr (hardware)	yes

- Temporaries and saved value registers are programmable.
- The rest of the registers have a special meaning.

MIPS I-format Instructions

- 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
  - Different formats complicate decoding
  - Keep formats as similar as possible

Load/Store Instruction (I format)



Question

- What is the corresponding C statement for the following MIPS assembly instructions?

```
sll   $t0,  $s0,  2
add   $t0,  $s6,  $t0
sll   $t1,  $s1,  2
add   $t1,  $s7,  $t1
lw    $s0,  0($t0)
addi  $t2,  $t0,  4
lw    $t0,  0($t2)
add   $t0,  $t0,  $s0
sw    $t0,  0($t1)
```

- Assume f, g, h, i and j are assigned to $s0, $s1, $s2, $s3, and $s4
- Base addresses of arrays A and B are in registers $s6 and $s7

Question- Solution

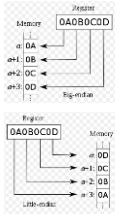
```
sll  $t0,  $s0,  2     //$t0 = f *4
add  $t0,  $s6,  $t0   //$t0 = &A[f]
sll  $t1,  $s1,  2     //$t1 = g * 4
add  $t1,  $s7,  $t1   //$t1 = &B[g]
lw   $s0,  0($t0)      //f = A[f]
addi $t2,  $t0,  4     //$t2 = &A[f + 1] //f is still of the original value
lw   $t0,  0($t2)      //$t0 = A[f + 1]
add  $t0,  $t0,  $s0   //$t0 = A[f + 1] + A[f]
sw   $t0,  0($t1)      //B[g] = A[f + 1] + A[f]
```

- Overall
  - B[g] = A[f+1] + A[f]
  - f = A[f]
  - g, i, j are not changed.
  - The two statements cannot swap order!

Copyright 2000 N. AYDIN. All rights reserved.

Byte Addresses

- Since 8-bit bytes are so useful, most architectures address individual bytes in memory
  - Most significant byte of a word is leftmost
  - Least significant byte of a word is rightmost



- Big Endian
  - leftmost byte is word address
    - IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- Little Endian
  - rightmost byte is word address
    - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)

25

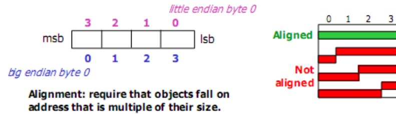
Endianness

- Big-endian representation is the most common convention in data networking;
  - fields in the protocols of the internet protocol are transmitted in big-endian order.
- Little-endian storage is popular for microprocessors in part due to significant historical influence on microprocessor designs by Intel
- Little-endian dominates but do not assume all use little-endian.

26

Instruction Alignment

- Alignment restriction
  - the memory address of a word must be on natural word boundaries
    - a multiple of 4 in MIPS-32



27

Immediate Instruction (I format)

- Small constants are used often in typical code
- Possible approaches?
  - put typical constants in memory and load them
  - create hard-wired registers (like $zero) for constants like 0
  - have special instructions that contain constants

```
addi   $sp,   $sp,   4     #$sp = $sp + 4
slti   $t0,   $s2,   15    #$t0 = 1 if $s2<15
```

6 bits	5 bits	5 bits	16 bits
opcode	rs	rt	constant

- The constant is kept inside the instruction itself!
  - Immediate format limits values to the range $+2^{15}-1$ to -2^{15}

28

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
- $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100)_2$

  $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$

  $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644$

  $= (-4)_{10}$

29

Signed Negation

- Complement and add 1
  - Complement means $1 \rightarrow 0$, $0 \rightarrow 1$

$$x + \bar{x} = 1111\ldots111_2 = -1$$

$$\bar{x} + 1 = -x$$

- Question: negate +2

  $+2 = (0000\ 0000\ \ldots\ 0010)_2$

  $-2 = (1111\ 1111\ \ldots\ 1101)_2 + 1$

  $= (1111\ 1111\ \ldots\ 1110)_2$

30

Copyright 2000 N. AYDIN. All rights reserved.

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
  - unsigned values: extend with 0s
  - Signed values: extend the sign bit
- Question: Extend +2 and -2 from 8-bit to 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110

MIPS instruction formats summary

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/logic instruction format
I-format	op	rs	rt	address/immediate			Data transfer, branch, immediate format
J-format	op	target address					Jump instruction format

Instruction encoding example

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	000000	reg	reg	reg	00000	100000	NA
sub	R	000000	reg	reg	reg	00000	100010	NA
addi (add immediate)	I	001000	reg	reg	NA	NA	NA	constant
lw (load word)	I	100011	reg	reg	NA	NA	NA	address
sw (store word)	I	101011	reg	reg	NA	NA	NA	address

MIPS operations

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$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
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	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
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd 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
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	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
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; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
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

MIPS Arithmetic Operations

- MIPS assembly language notation
  - add a, b, c
    - add the two variables b and c and put their sum in a
  - each MIPS arithmetic instruction
    - performs only one operation
    - must always have exactly three variables
- Example, to place the sum of four variables b, c, d, and e into variable a.
  - add a, b, c # The sum of b and c is placed in a
  - add a, a, d # The sum of b, c, and d is now in a
  - add a, a, e # The sum of b, c, d, and e is now in a

MIPS Arithmetic Operations

- Example:
  - Compiling C Assignment Statements into MIPS
  - C code : a = b + c
  - MIPS code : add $s0, $s1, $s2
- compiler's job is to associate variables with registers

Copyright 2000 N. AYDIN. All rights reserved.

MIPS Arithmetic Operations

- Example:
  - Compiling C Assignment Statements into MIPS
    - C code : a = b + c
    - MIPS code : add $s0, $s1, $s2
  - compiler's job is to associate variables with registers
- Design Principle 1:
  - simplicity favors regularity.
    - Regular instructions make for simple hardware!
    - Simpler hardware reduces design time and manufacturing cost.

37

MIPS Arithmetic Operations

- Consider the following C code:
  - f = (g + h) - (i + j);
  - 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?
- Compiled MIPS code:

```
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
```

38

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

- Useful for extracting and inserting groups of bits in a word

39

Logical Operations

- AND Operations
  - Useful to mask bits in a word
    - Select some bits, clear others to 0
  - and $t0, $t1, $t2

$t2	0000 0000 0000 0000 0000 1101 1100 0000
$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	0000 0000 0000 0000 0000 1100 0000 0000

40

Logical Operations

- OR Operations
  - Useful to include bits in a word
    - Set some bits to 1, leave others unchanged
  - or $t0, $t1, $t2

$t2	0000 0000 0000 0000 0000 1101 1100 0000
$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	0000 0000 0000 0000 0011 1101 1100 0000

41

Logical Operations

- NOT Operations
  - Useful to invert bits in a word
    - Change 0 to 1, and 1 to 0
  - 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

42

Copyright 2000 N. AYDIN. All rights reserved.

Hexadecimal

- Base 16
  - Compact representation of bit strings
  - 4 bits per hex digit

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

- Example: eca86420

  1110 1100 1010 1000 0110 0100 0010 0000

43

44

Copyright 2000 N. AYDIN. All rights reserved.