Math 230
Assembly Programming
(*AKA Computer Organization*)
Spring 2008

MIPS Intro II

Lect 10

Feb 15, 2008

Adapted from slides developed for:

Mary J. Irwin PSU CSE331

Dave Patterson's UCB CS152

MIPS Data Types

Bit: 0, 1

Bit String: sequence of bits of a particular length

4 bits is a nibble
8 bits is a byte
16 bits is a half-word
32 bits (4 bytes) is a word
64 bits is a double-word

Character:

ASCII 7 bit code

Decimal:

digits 0-9 encoded as 0000b thru 1001b
two decimal digits packed per 8 bit byte

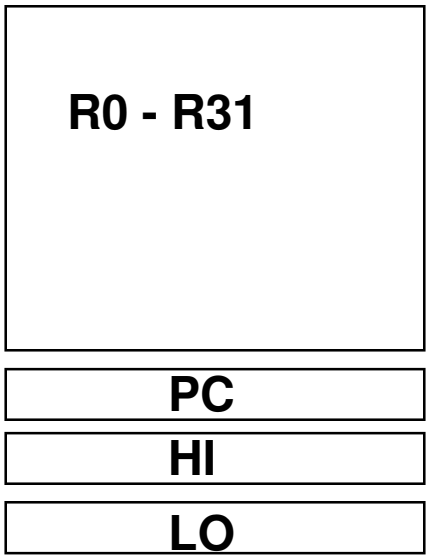
Integers: 2's complement

Floating Point

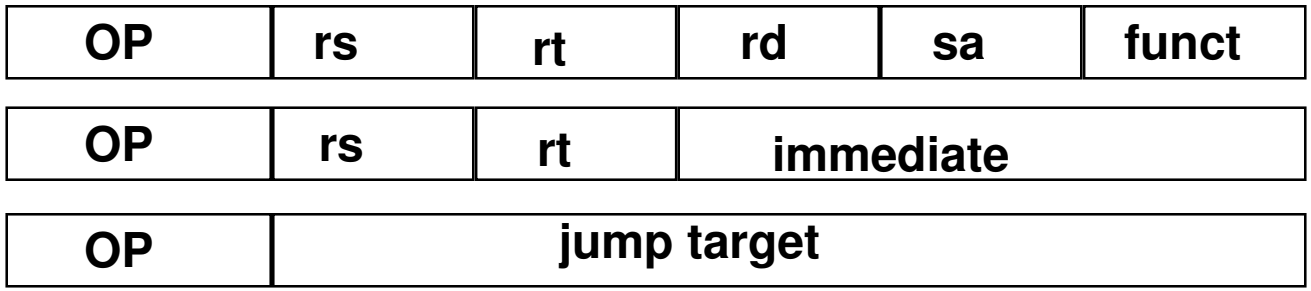
Review: MIPS R3000 Instruction Set Architecture

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

Registers



- **3 Instruction Formats: all 32 bits wide**

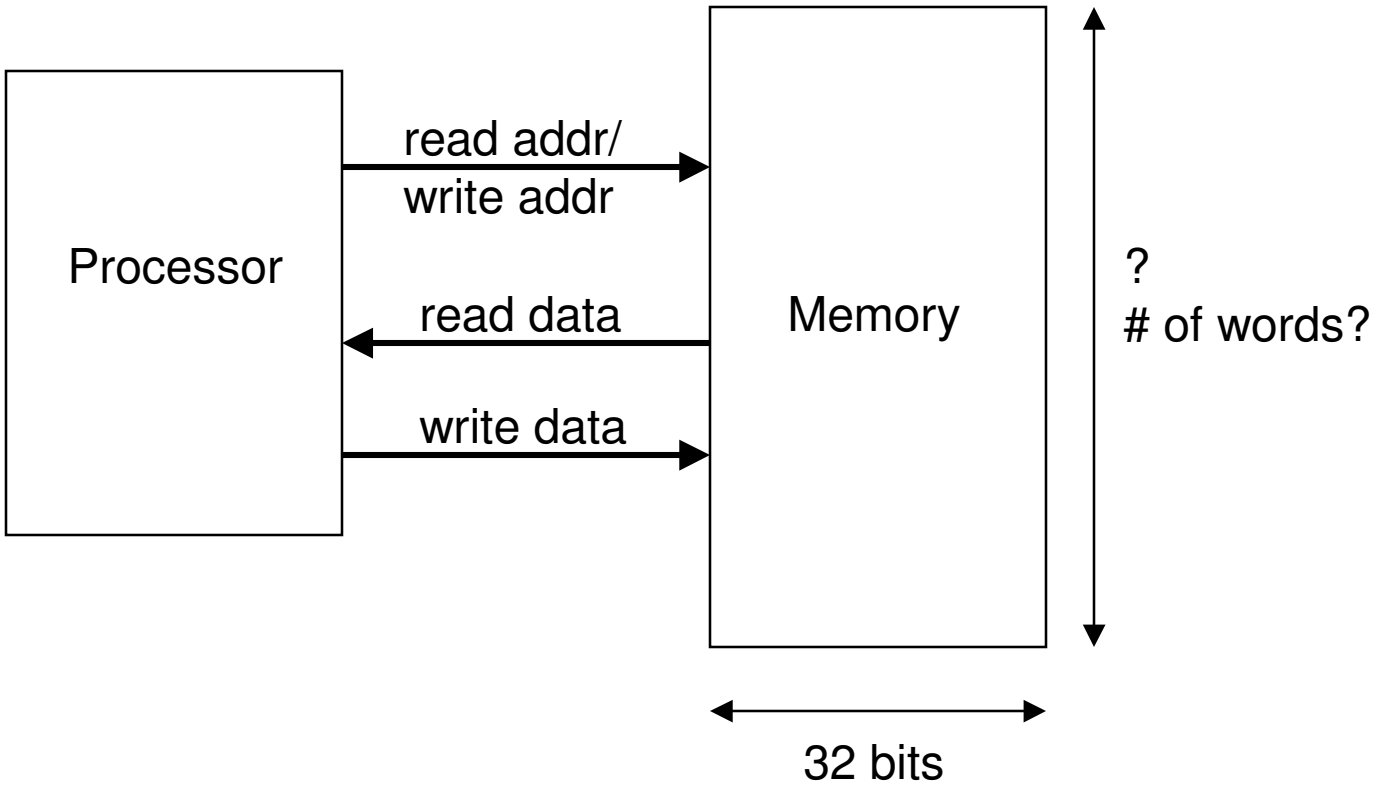


MIPS Instructions, so far

Category	Instr	Op Code	Example	Meaning
Arithmetic (R format)	**add**	0 and 32	**add** $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	0 and 34	**sub** $s1, $s2, $s3	$s1 = $s2 - $s3
Data transfer (I format)	**load word**	35	**lw** $s1, 100($s2)	$s1 = Memory($s2+100)
	store word	43	**sw** $s1, 100($s2)	Memory($s2+100) = $s1

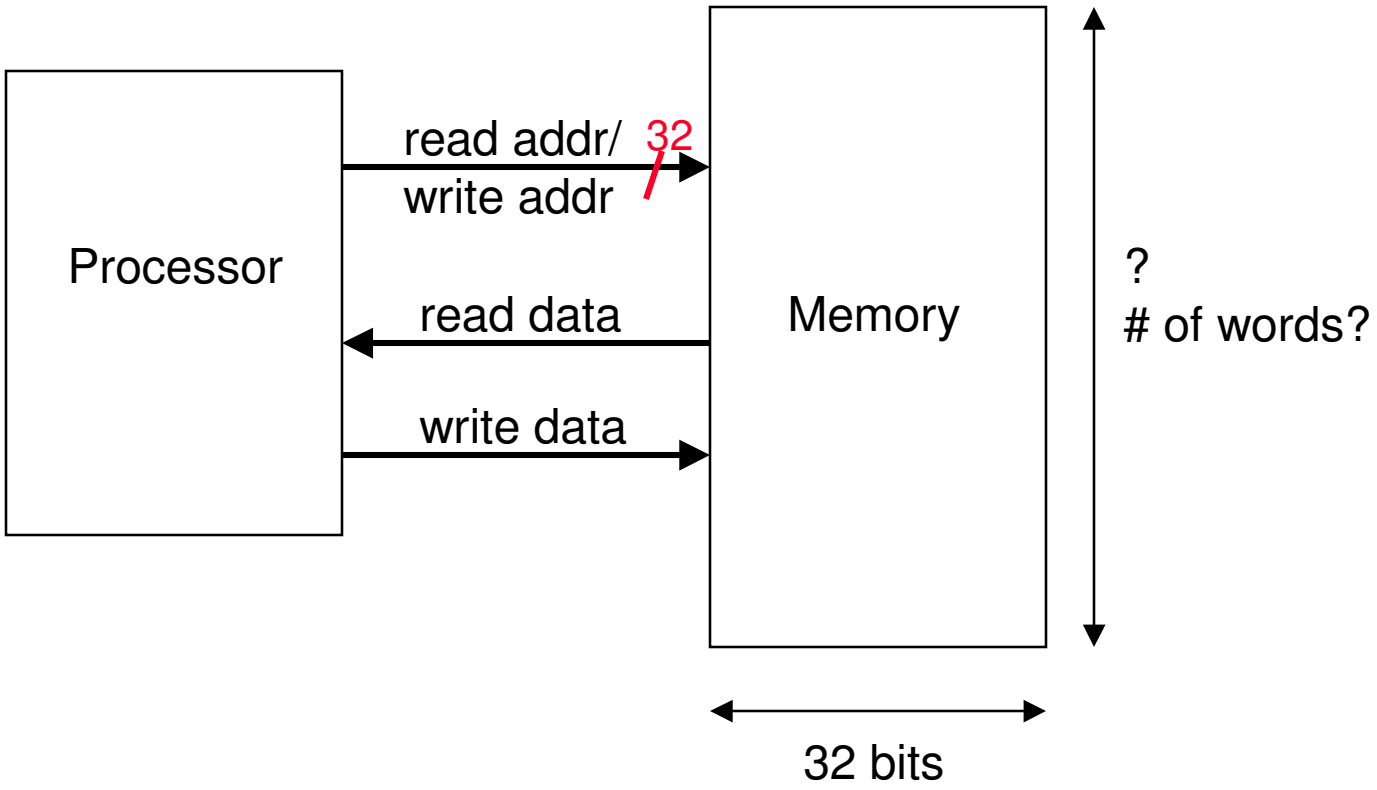
Processor – Memory Interconnections

- Memory is viewed as a large, single-dimension array, with an address
- A memory address is an index into the array



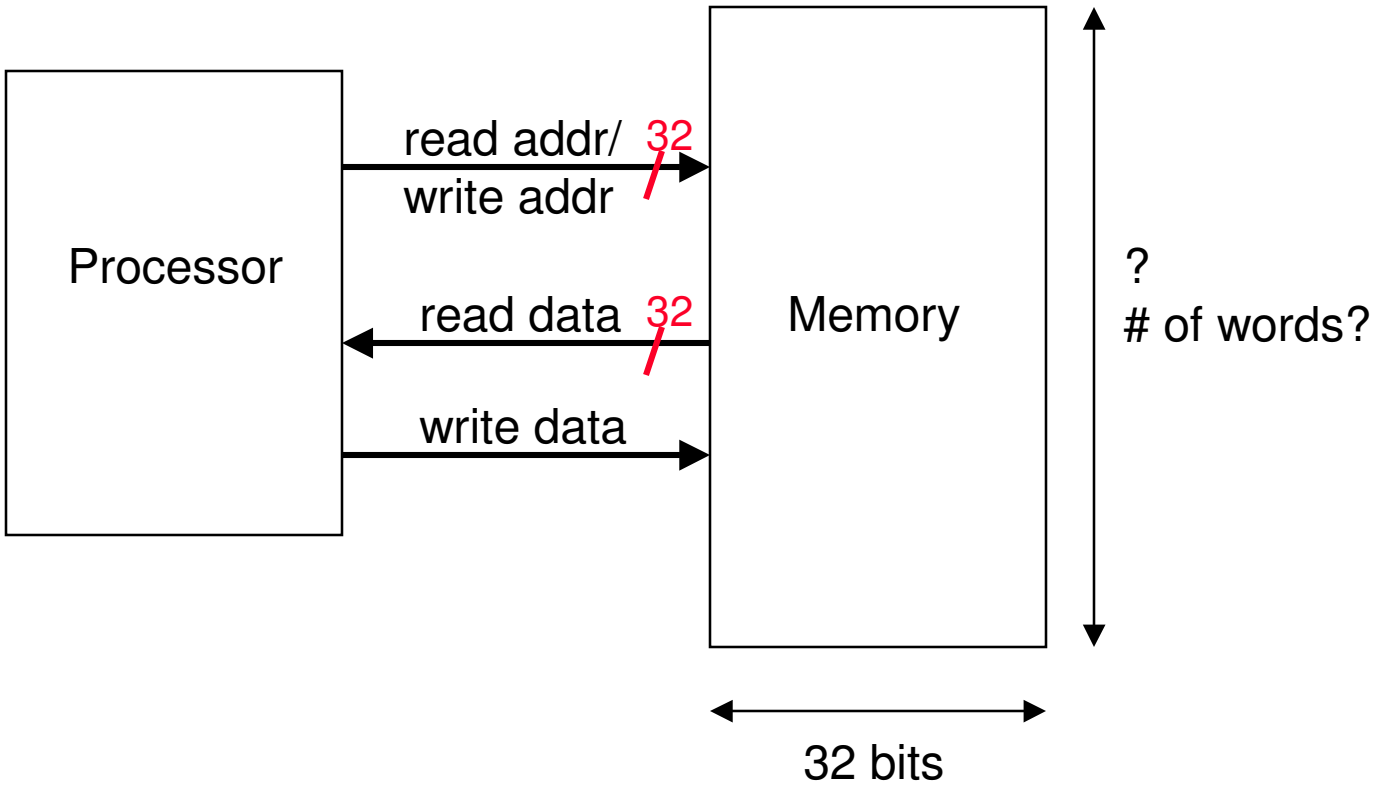
Processor – Memory Interconnections

- Memory is viewed as a large, single-dimension array, with an address
- A memory address is an index into the array



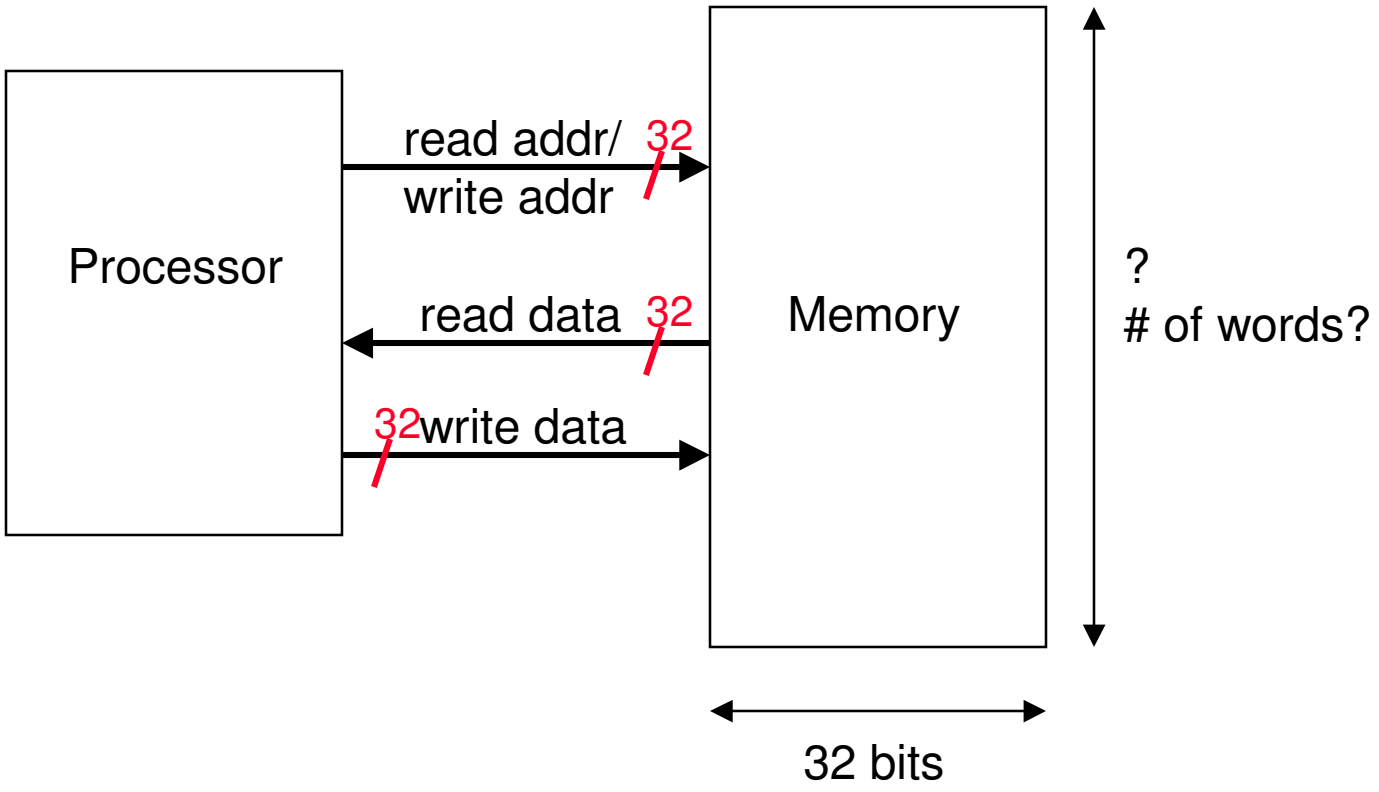
Processor – Memory Interconnections

- Memory is viewed as a large, single-dimension array, with an address
- A memory address is an index into the array



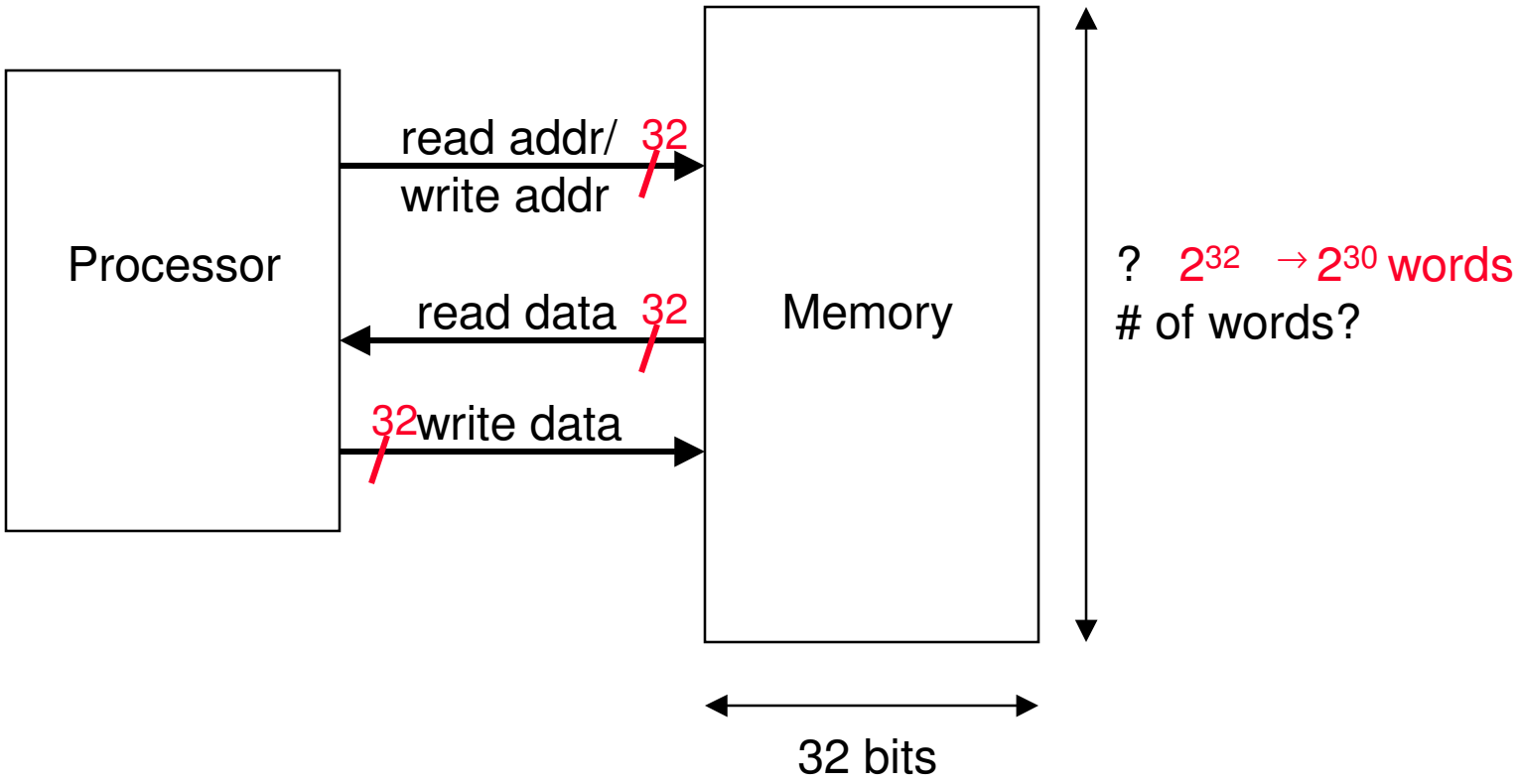
Processor – Memory Interconnections

- Memory is viewed as a large, single-dimension array, with an address
- A memory address is an index into the array



Processor – Memory Interconnections

- Memory is viewed as a large, single-dimension array, with an address
- A memory address is an index into the array



Review: Naming Conventions for Registers

0	$zero constant 0 (Hdware)
1	$at reserved for assembler
2	$v0 expression evaluation &
3	$v1 function results
4	$a0 arguments
5	$a1
6	$a2
7	$a3
8	$t0 temporary: caller saves
. . .	(callee can clobber)
15	$t7
16	$s0 callee saves
. . .	(caller can clobber)
23	$s7
24	$t8 temporary (cont'd)
25	$t9
26	$k0 reserved for OS kernel
27	$k1
28	$gp pointer to global area
29	$sp stack pointer
30	$fp frame pointer
31	$ra return address (Hdware)

Byte Addresses

- Since 8-bit bytes are so useful, most architectures address individual bytes in memory
- Therefore, the memory address of a word must be a multiple of 4 (alignment restriction)

- Big Endian: leftmost byte is word address

  IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA

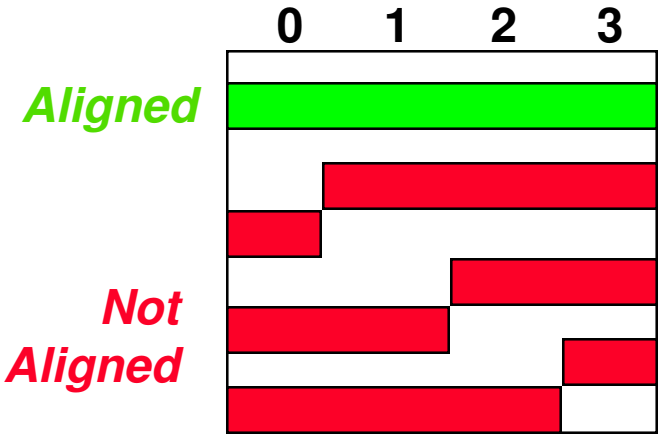
- Little Endian: rightmost byte is word address

  Intel 80x86, DEC Vax, DEC Alpha (Windows NT)

Addressing Objects: Endianess and Alignment

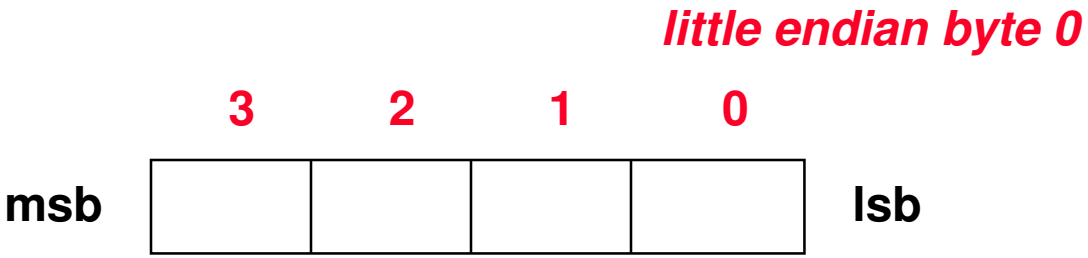
- Big Endian: leftmost byte is word address
- Little Endian: rightmost byte is word address

Alignment restriction: requires that objects fall on address that is multiple of their size.

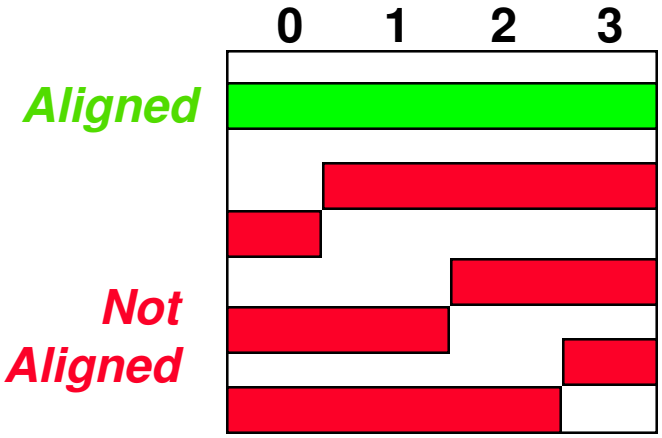


Addressing Objects: Endianess and Alignment

- Big Endian: leftmost byte is word address
- Little Endian: rightmost byte is word address

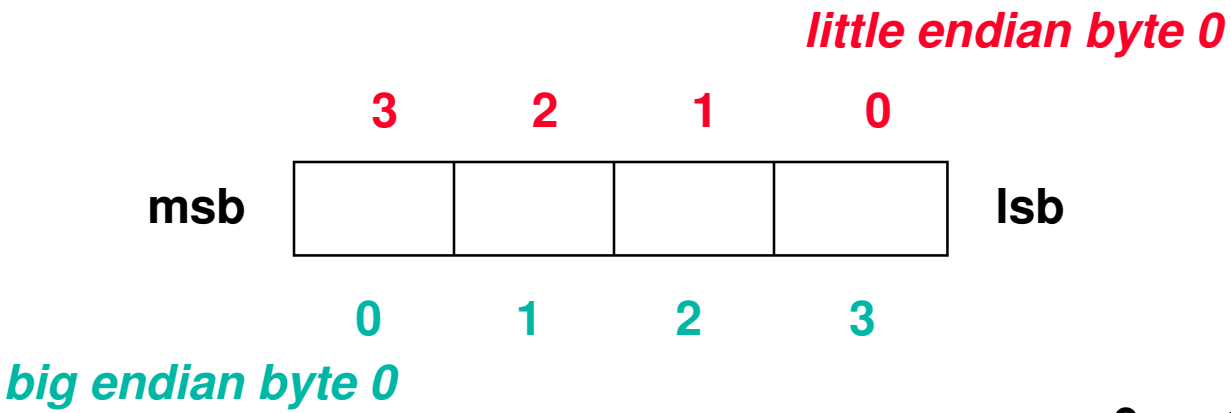


Alignment restriction: requires that objects fall on address that is multiple of their size.

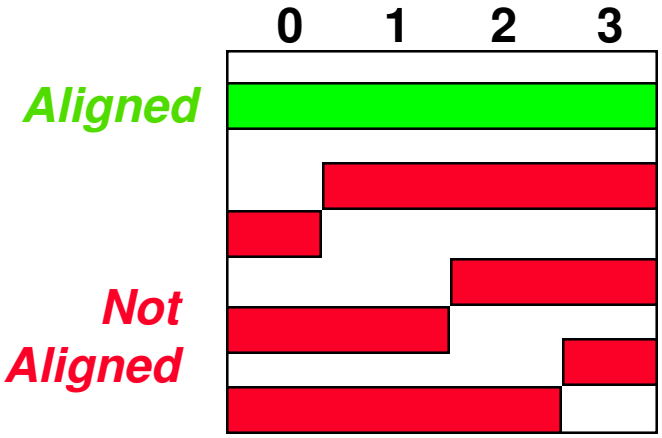


Addressing Objects: Endianess and Alignment

- Big Endian: leftmost byte is word address
- Little Endian: rightmost byte is word address

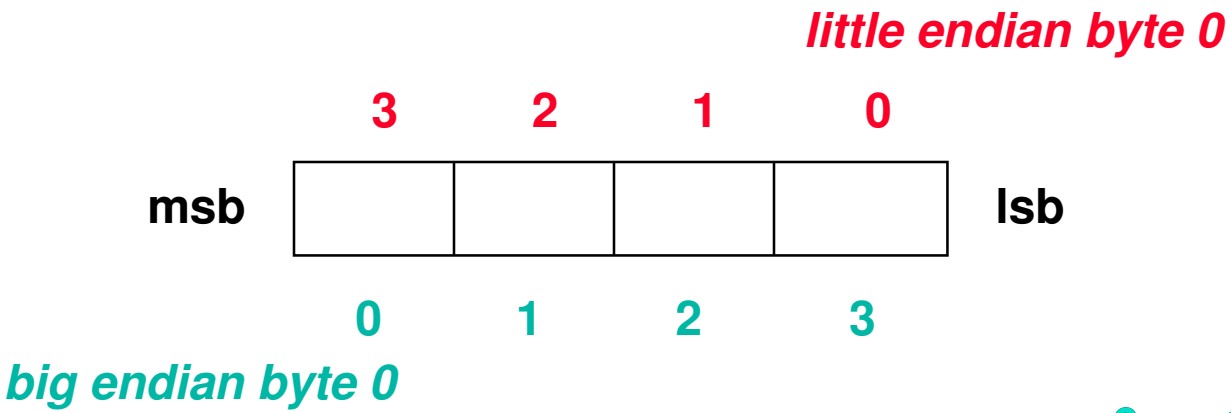


Alignment restriction: requires that objects fall on address that is multiple of their size.

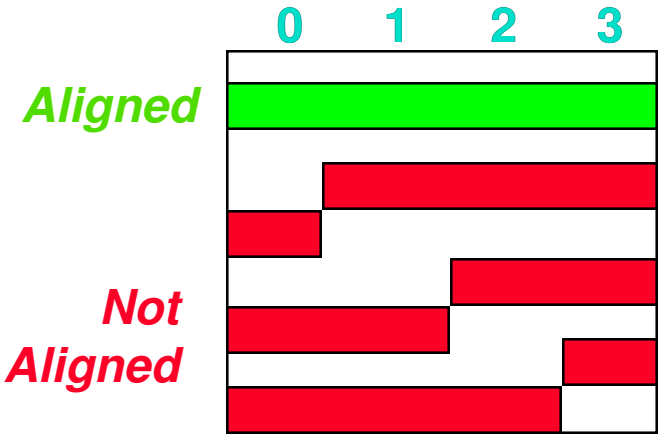


Addressing Objects: Endianess and Alignment

- Big Endian: leftmost byte is word address
- Little Endian: rightmost byte is word address

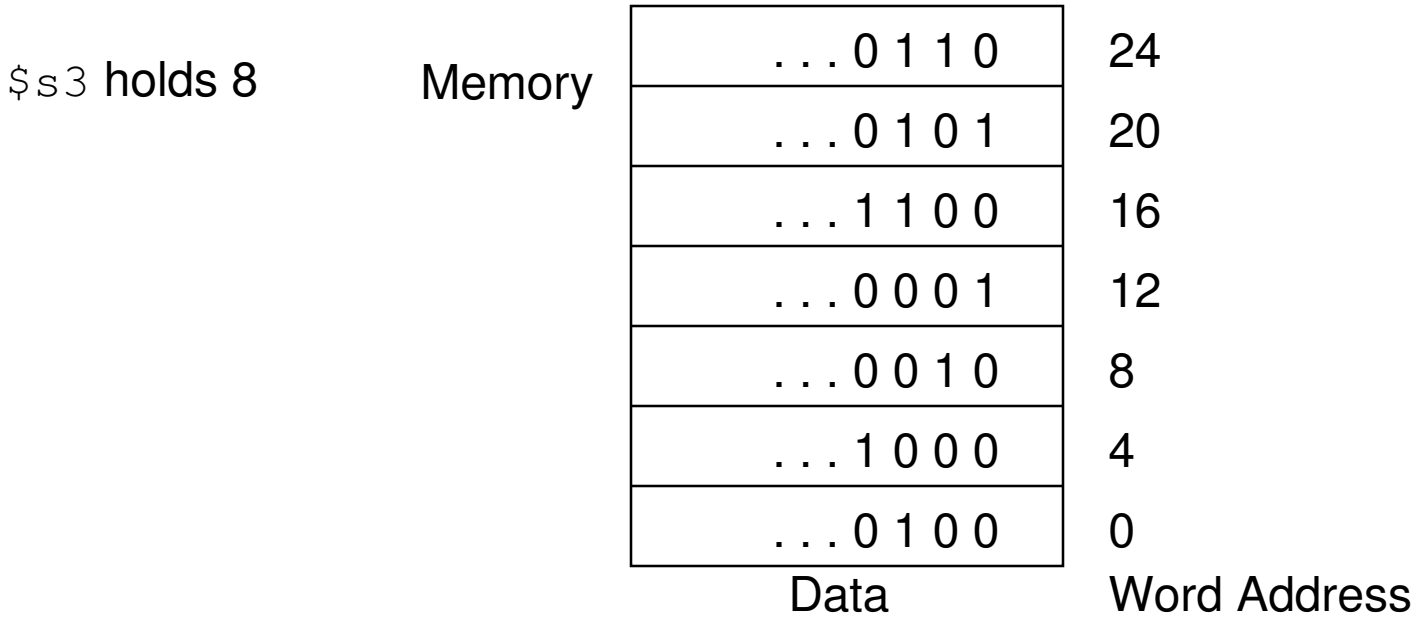


Alignment restriction: requires that objects fall on address that is multiple of their size.



MIPS Memory Addressing

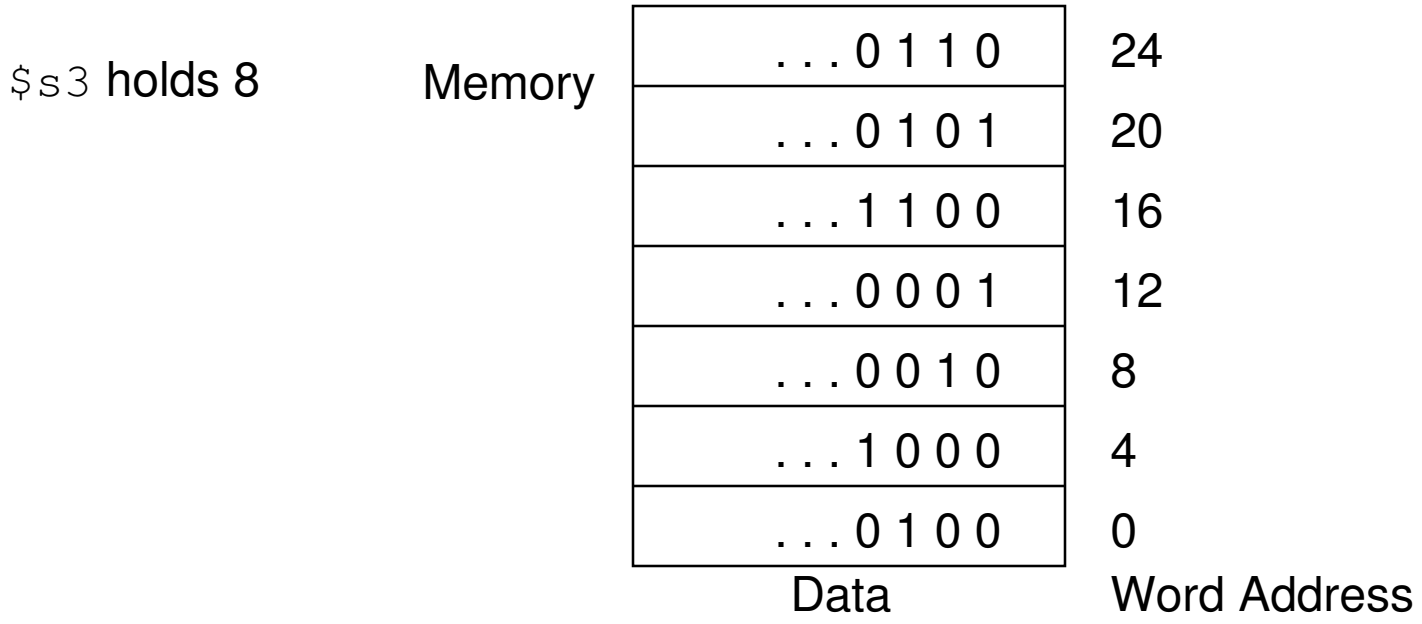
- The memory address is formed by summing the constant portion of the instruction and the contents of the second (base) register




```
lw    $t0, 4($s3)   #what? is loaded into $t0

sw    $t0, 8($s3)   #$t0 is stored where?
```

MIPS Memory Addressing

- The memory address is formed by summing the constant portion of the instruction and the contents of the second (base) register

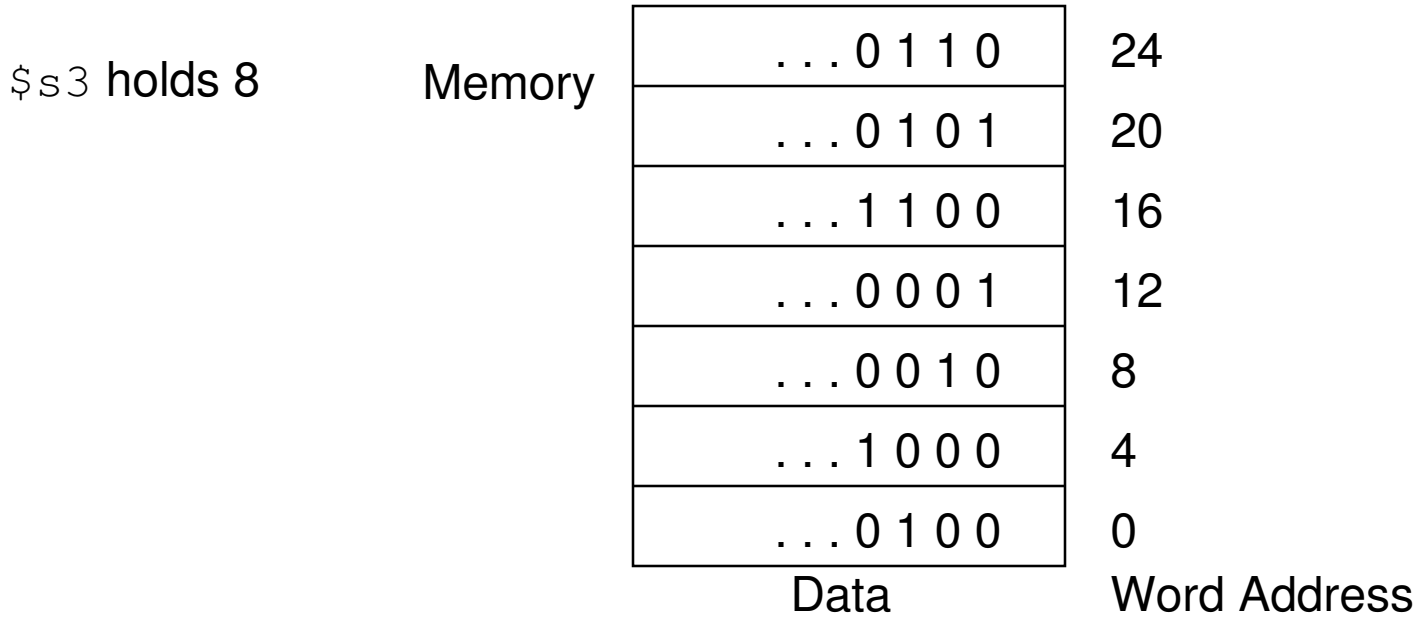


. . . 0001

```
lw    $t0, 4($s3)   #what? is loaded into $t0

sw    $t0, 8($s3)   #$t0 is stored where?
```

MIPS Memory Addressing

- The memory address is formed by summing the constant portion of the instruction and the contents of the second (base) register



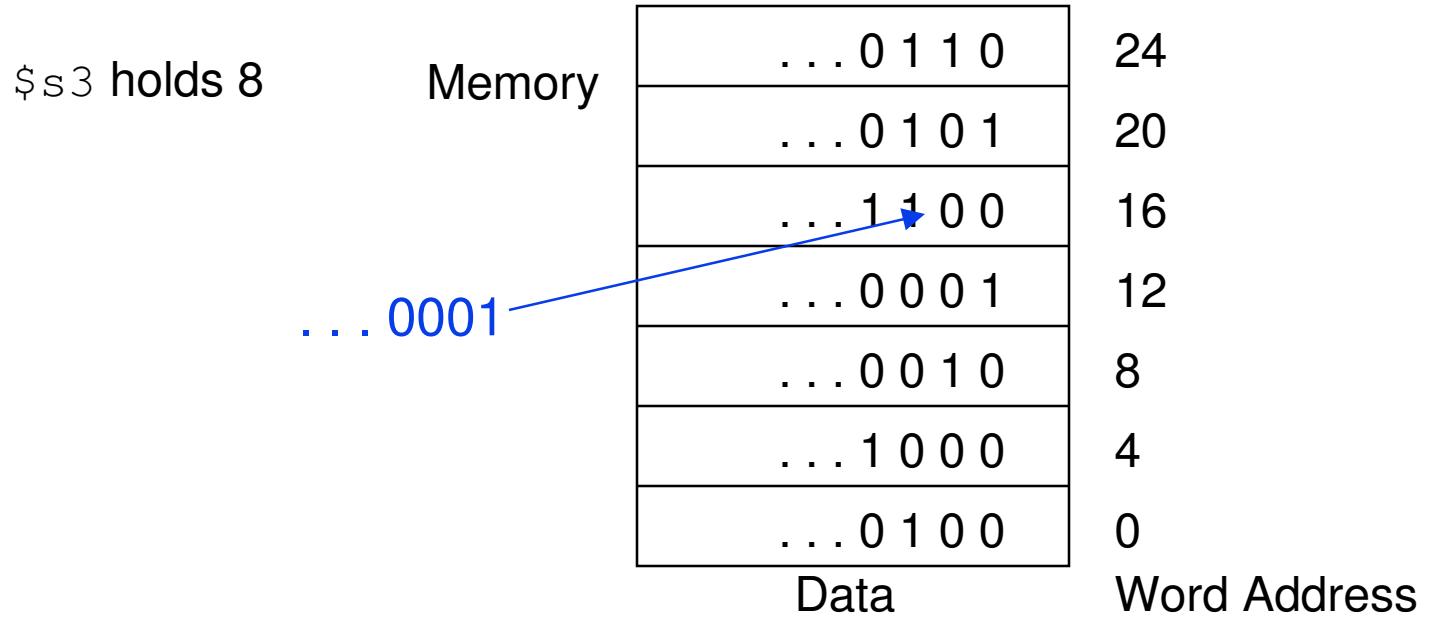
. . . 0001

```
lw    $t0, 4($s3)   #what? is loaded into $t0

sw    $t0, 8($s3)   #$t0 is stored where?
```

in location 16

MIPS Memory Addressing

- The memory address is formed by summing the constant portion of the instruction and the contents of the second (base) register



. . . 0001

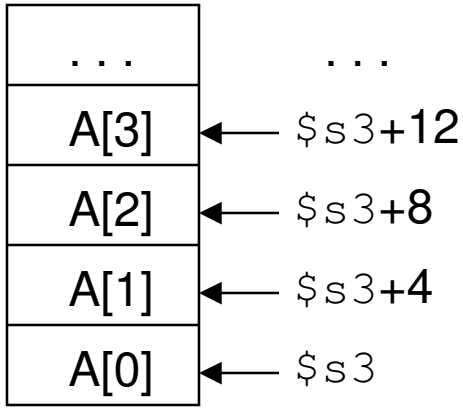
```
lw    $t0, 4($s3)   #what? is loaded into $t0

sw    $t0, 8($s3)   #$t0 is stored where?
```

in location 16

Compiling with Loads and Stores

- Assuming variable b is stored in `$s2` and that the base address of array A is in `$s3`, what is the MIPS assembly code for the C statement

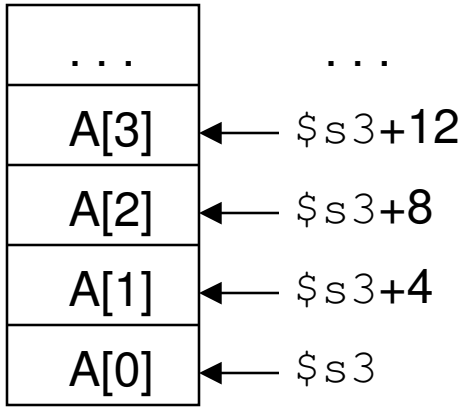
```
A[8] = A[2] - b
```



Compiling with Loads and Stores

- Assuming variable b is stored in `$s2` and that the base address of array A is in `$s3`, what is the MIPS assembly code for the C statement

```
A[8] = A[2] - b
```




```
lw   $t0, 8($s3)
sub  $t0, $t0, $s2
sw   $t0, 32($s3)
```

Compiling with a Variable Array Index

- Assuming A is an array of 50 elements whose base is in register `$s4`, and variables b, c, and i are in `$s1`, `$s2`, and `$s3`, respectively, what is the MIPS assembly code for the C statement

```
c = A[i] - b
```

```
add   $t1, $s3, $s3     #array index i is in $s3
add   $t1, $t1, $t1     #temp reg $t1 holds 4*i
```

Compiling with a Variable Array Index

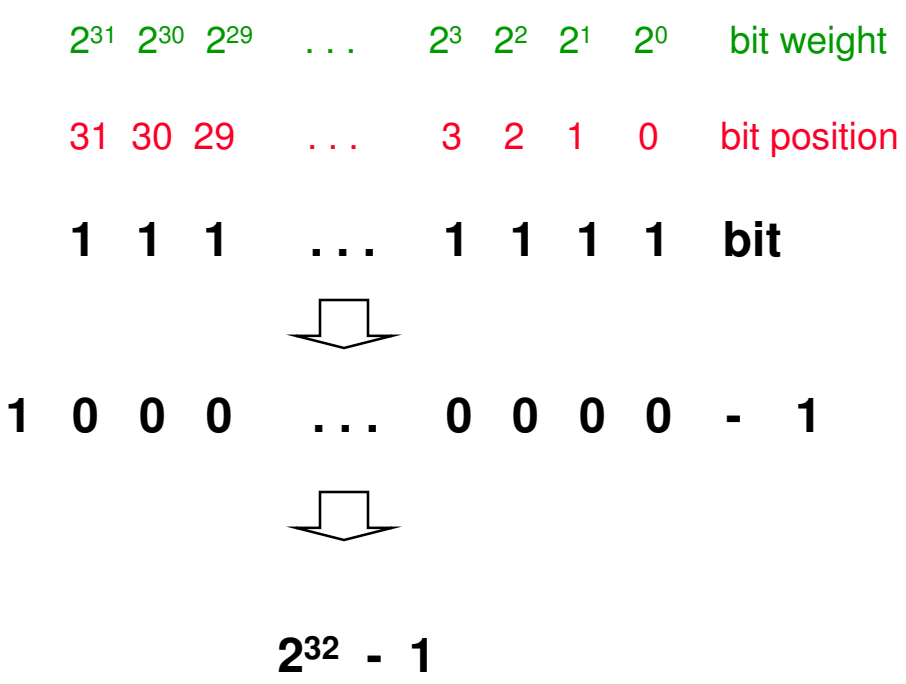
- Assuming A is an array of 50 elements whose base is in register `$s4`, and variables b, c, and i are in `$s1`, `$s2`, and `$s3`, respectively, what is the MIPS assembly code for the C statement

```
c = A[i] - b
```

```
add   $t1, $s3, $s3     #array index i is in $s3
add   $t1, $t1, $t1     #temp reg $t1 holds 4*i
add   $t1, $t1, $s4     #addr of A[i]
lw    $t0, 0($t1)
sub   $s2, $t0, $s1
```

Review: Unsigned Binary Representation

Hex	Binary	Decimal
0x00000000	0…0000	0
0x00000001	0…0001	1
0x00000002	0…0010	2
0x00000003	0…0011	3
0x00000004	0…0100	4
0x00000005	0…0101	5
0x00000006	0…0110	6
0x00000007	0…0111	7
0x00000008	0…1000	8
0x00000009	0…1001	9
	…	
0xFFFFFFFC	1…1100	$2^{32} - 4$
0xFFFFFFFD	1…1101	$2^{32} - 3$
0xFFFFFFFE	1…1110	$2^{32} - 2$
0xFFFFFFFF	1…1111	$2^{32} - 1$



Review: Signed Binary Representation

	2'sc binary	decimal
-2^3 =	1000	-8
$-(2^3 - 1)$ =	1001	-7
	1010	-6
	1011	-5
	1100	-4
	1101	-3
	1110	-2
	1111	-1
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
$2^3 - 1$ =	0111	7

Review: Signed Binary Representation

	2'sc binary	decimal
-2^3 =	1000	-8
$-(2^3 - 1)$ =	1001	-7
	1010	-6
	1011	-5
	1100	-4
	1101	-3
	1110	-2
	1111	-1
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
$2^3 - 1$ =	0111	7

1010

complement all the bits

Review: Signed Binary Representation

	2'sc binary	decimal
-2^3 =	1000	-8
$-(2^3 - 1)$ =	1001	-7
	1010	-6
	1011	-5
	1100	-4
	1101	-3
	1110	-2
	1111	-1
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
$2^3 - 1$ =	0111	7

1011

and add a 1

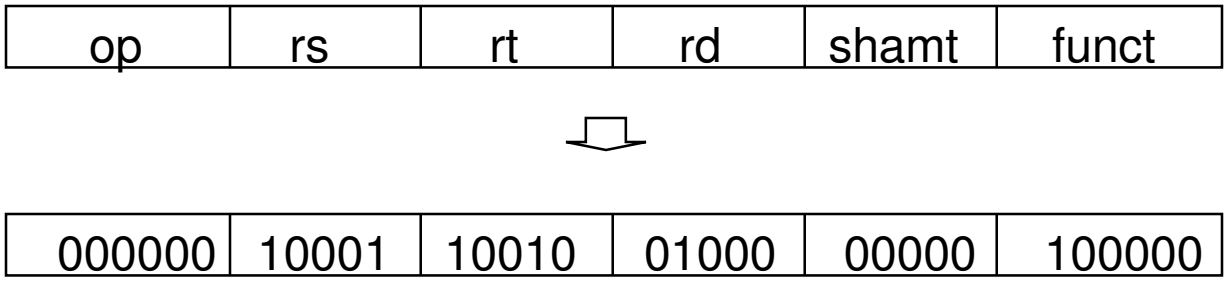
1010

complement all the bits

Machine Language - Arithmetic Instruction

- Instructions, like registers and words of data, are also 32 bits long
  - Example: `add $t0, $s1, $s2`
  - registers have numbers `$t0=$8, $s1=$17, $s2=$18`

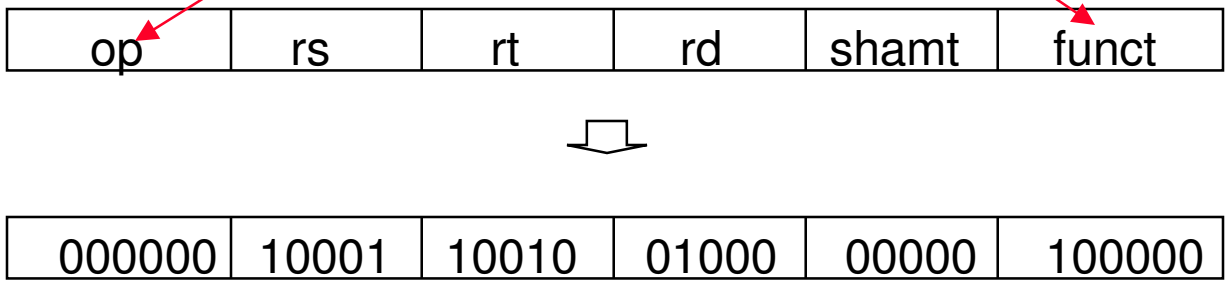
- Instruction Format:



Can you guess what the field names stand for?

Machine Language - Arithmetic Instruction

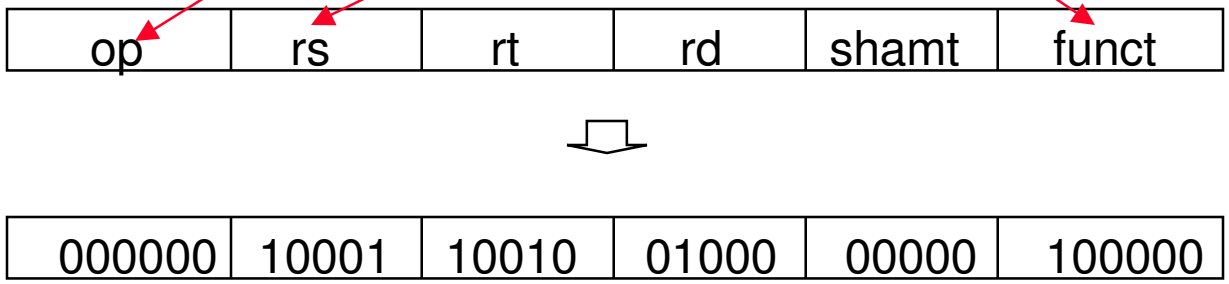
- Instructions, like registers and words of data, are also 32 bits long
  - Example: `add $t0, $s1, $s2`
  - registers have numbers `$t0=$8, $s1=$17, $s2=$18`
- Instruction Format:



Can you guess what the field names stand for?

Machine Language - Arithmetic Instruction

- Instructions, like registers and words of data, are also 32 bits long
  - Example: `add $t0, $s1, $s2`
  - registers have numbers `$t0=$8, $s1=$17, $s2=$18`
- Instruction Format:



Can you guess what the field names stand for?

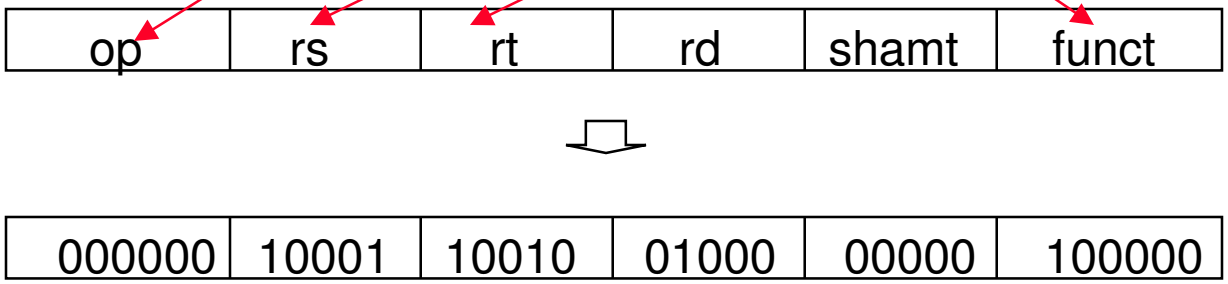
Machine Language - Arithmetic Instruction

- Instructions, like registers and words of data, are also 32 bits long
  - Example:
  - registers have numbers

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

```
$t0=$8, $s1=$17, $s2=$18
```

- Instruction Format:



Can you guess what the field names stand for?

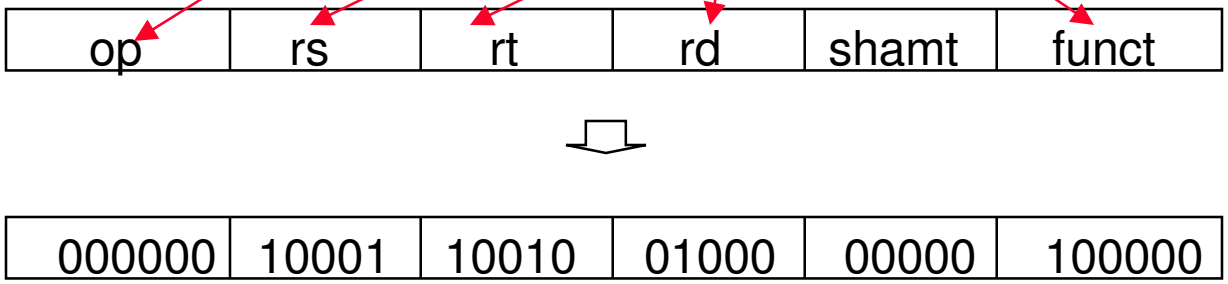
Machine Language - Arithmetic Instruction

- Instructions, like registers and words of data, are also 32 bits long
  - Example:
  - registers have numbers

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

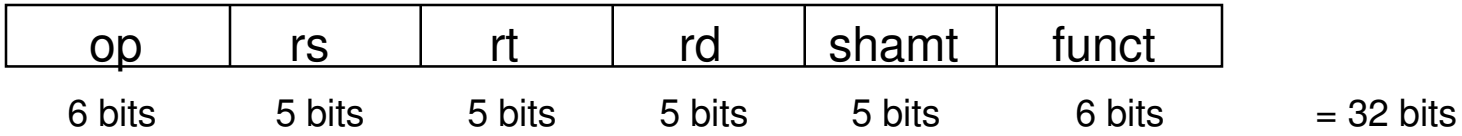
```
$t0=$8, $s1=$17, $s2=$18
```

- Instruction Format:



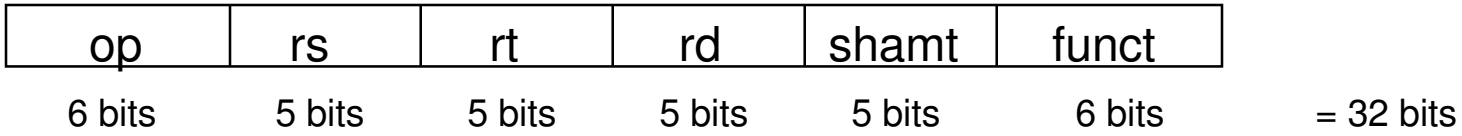
Can you guess what the field names stand for?

MIPS Instruction Fields



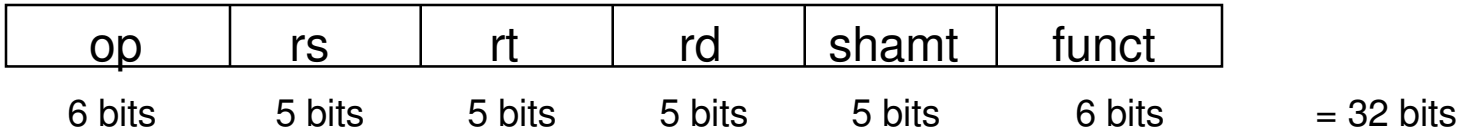
- *op*
- *rs*
- *rt*
- *rd*
- *shamt*
- *funct*

MIPS Instruction Fields



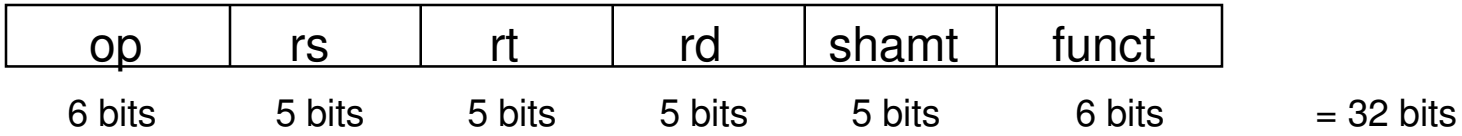
- *op* opcode indicating operation to be performed
- *rs*
- *rt*
- *rd*
- *shamt*
- *funct*

MIPS Instruction Fields



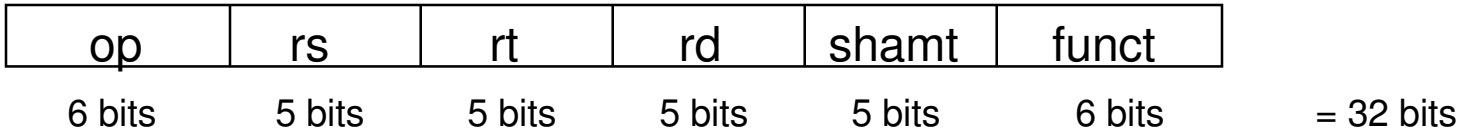
- *op* opcode indicating operation to be performed
- *rs* address of the first register source operand
- *rt*
- *rd*
- *shamt*
- *funct*

MIPS Instruction Fields



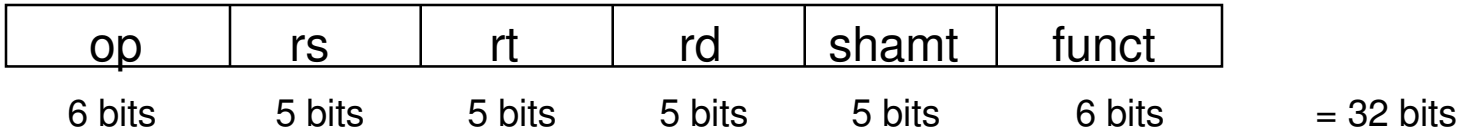
- *op* opcode indicating operation to be performed
- *rs* address of the first register source operand
- *rt* address of the second register source operand
- *rd*
- *shamt*
- *funct*

MIPS Instruction Fields



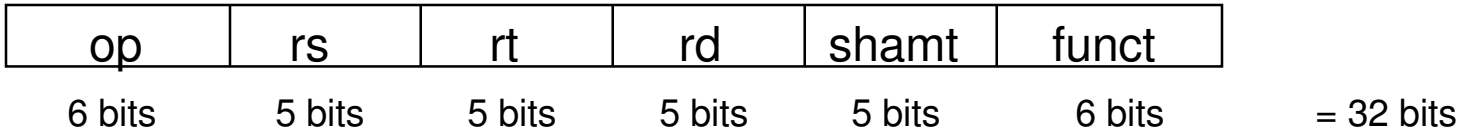
- *op* opcode indicating operation to be performed
- *rs* address of the first register source operand
- *rt* address of the second register source operand
- *rd* the register destination address
- *shamt*
- *funct*

MIPS Instruction Fields



- *op* opcode indicating operation to be performed
- *rs* address of the first register source operand
- *rt* address of the second register source operand
- *rd* the register destination address
- *shamt* shift amount (for shift instructions)
- *funct*

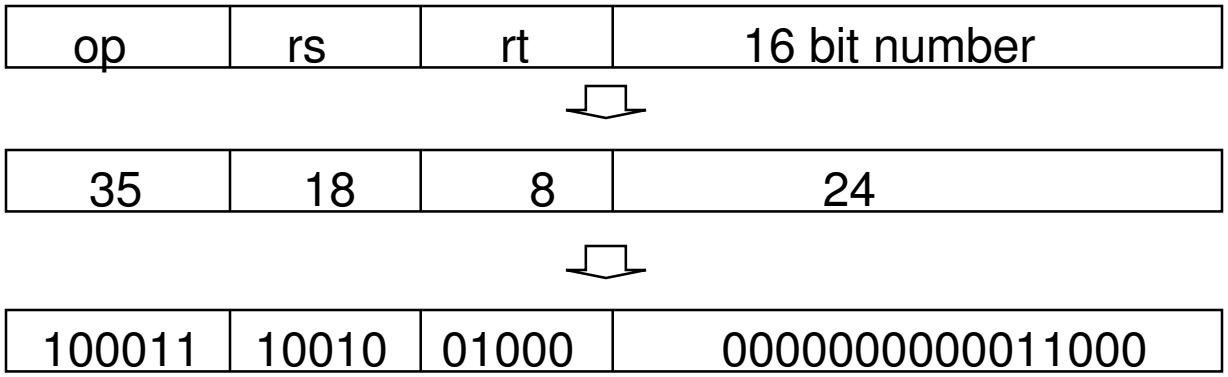
MIPS Instruction Fields



- *op* opcode indicating operation to be performed
- *rs* address of the first register source operand
- *rt* address of the second register source operand
- *rd* the register destination address
- *shamt* shift amount (for shift instructions)
- *funct* function code that selects the specific variant of the operation specified in the opcode field

Machine Language - Load Instruction

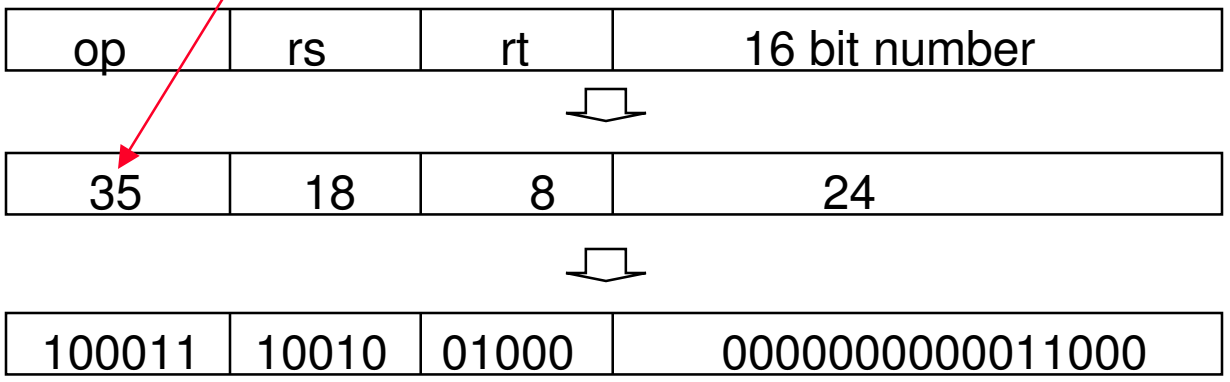
- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - New principle: Good design demands a compromise
- Introduce a new type of instruction format
  - I-type for data transfer instructions
  - previous format was R-type for register
- Example: `lw $t0, 24($s2)`



Where's the compromise?

Machine Language - Load Instruction

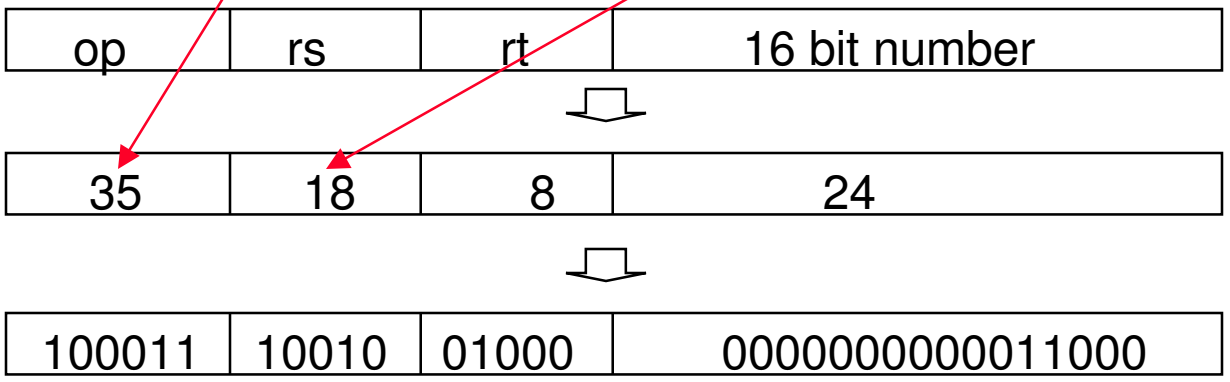
- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - New principle: Good design demands a compromise
- Introduce a new type of instruction format
  - I-type for data transfer instructions
  - previous format was R-type for register
- Example: `lw $t0, 24($s2)`



Where's the compromise?

Machine Language - Load Instruction

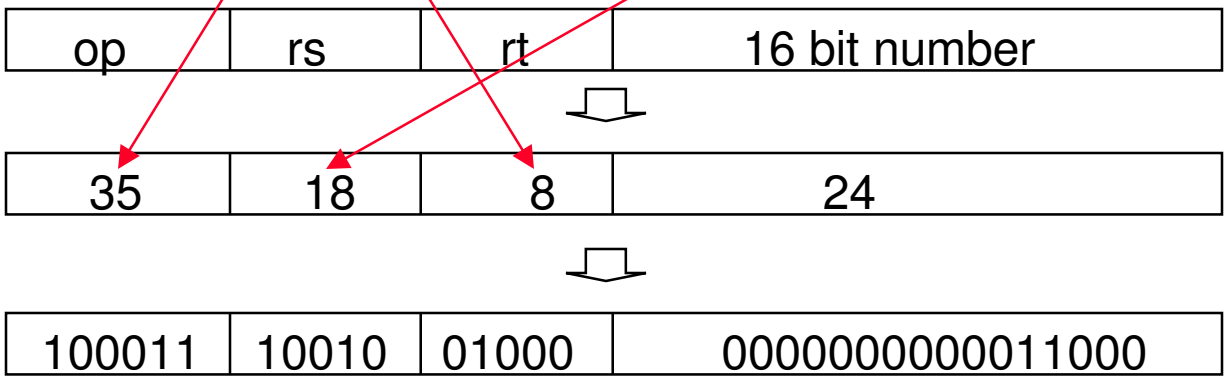
- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - New principle: Good design demands a compromise
- Introduce a new type of instruction format
  - I-type for data transfer instructions
  - previous format was R-type for register
- Example: `lw $t0, 24($s2)`



Where's the compromise?

Machine Language - Load Instruction

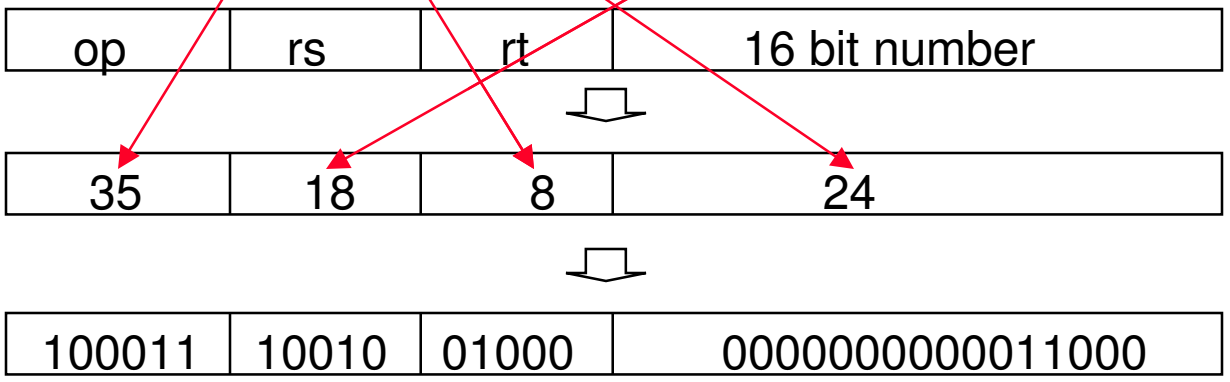
- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - New principle: Good design demands a compromise
- Introduce a new type of instruction format
  - I-type for data transfer instructions
  - previous format was R-type for register
- Example: `lw $t0, 24($s2)`



Where's the compromise?

Machine Language - Load Instruction

- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - New principle: Good design demands a compromise
- Introduce a new type of instruction format
  - I-type for data transfer instructions
  - previous format was R-type for register
- Example: `lw $t0, 24($s2)`

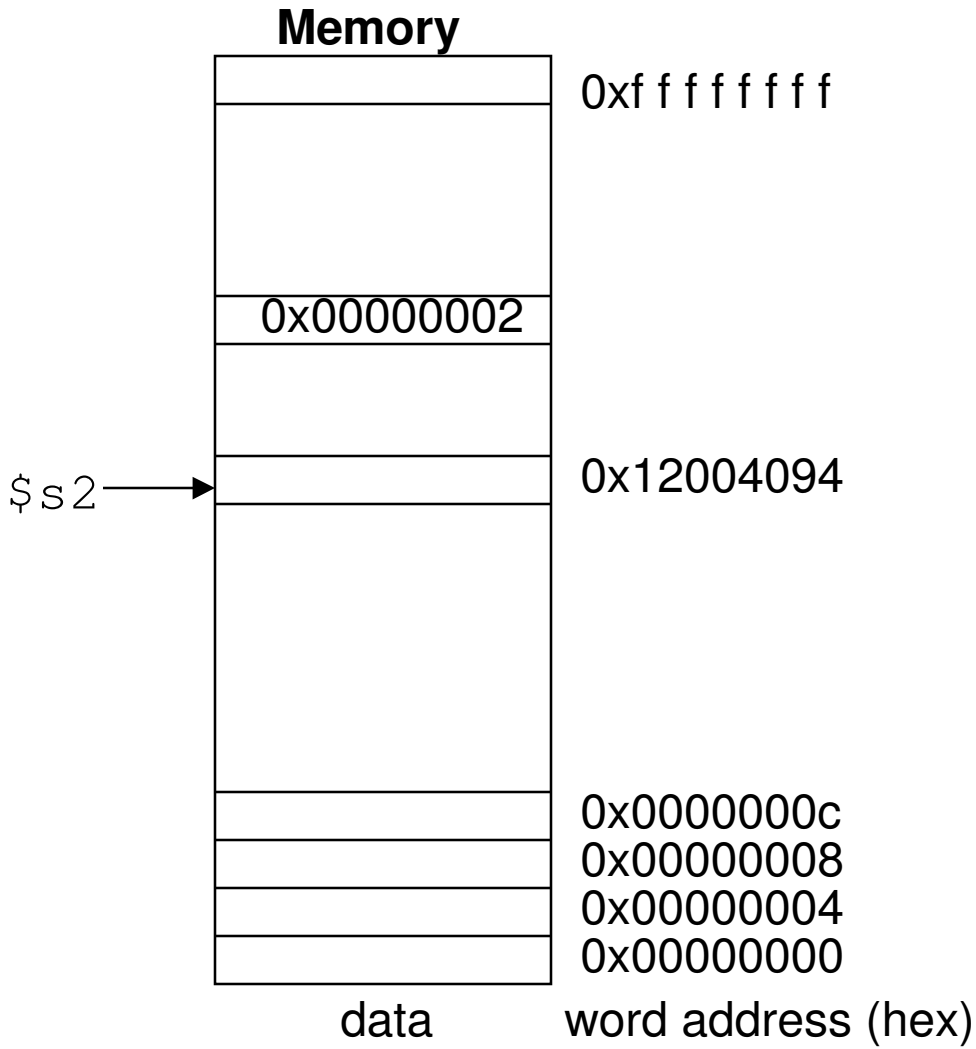


Where's the compromise?

Memory Address Location

- Example: `lw $t0, 24($s2)`

24_{10} + $s2 =



Note that the offset can be positive or negative

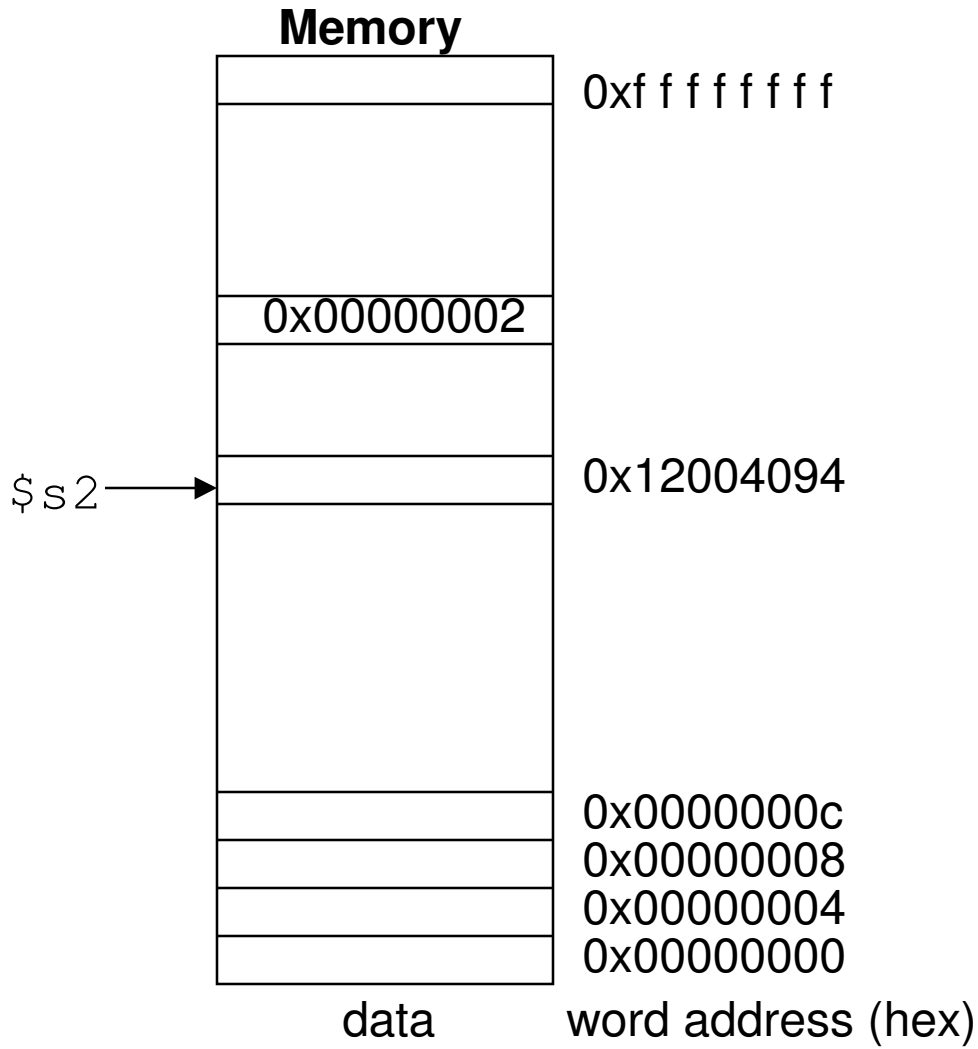
Memory Address Location

- Example: `lw $t0, 24($s2)`

24_{10} + $s2 =

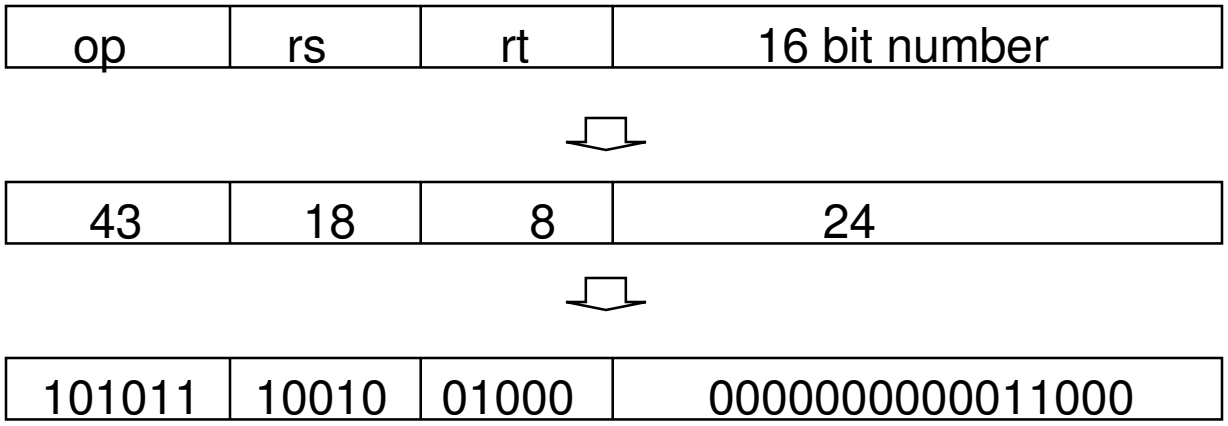
. . . 1001 0100
+ . . . 0001 1000
. . . 1010 1100 =
0x120040ac

Note that the offset can be positive or negative



Machine Language - Store Instruction

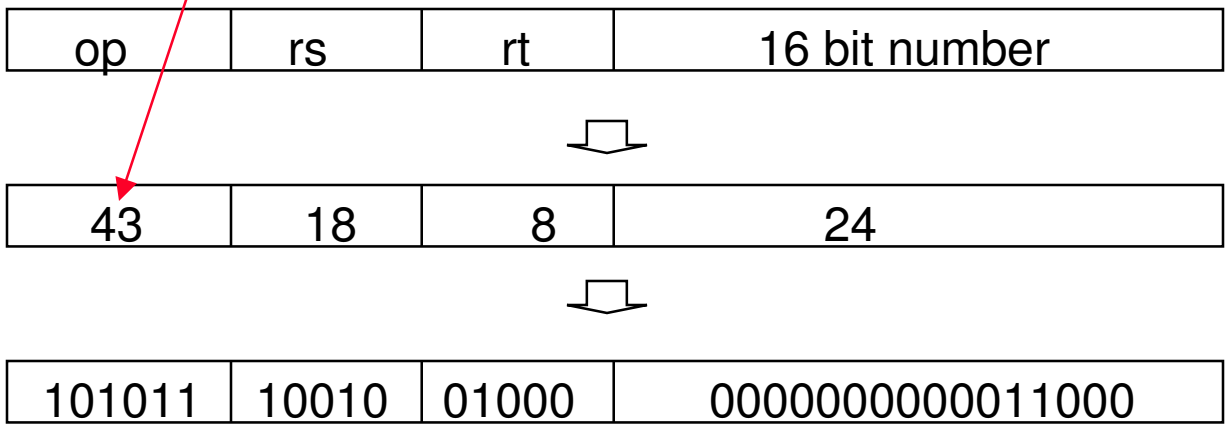
- Example: `sw $t0, 24($s2)`



- A 16-bit address means access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register `$s2`

Machine Language - Store Instruction

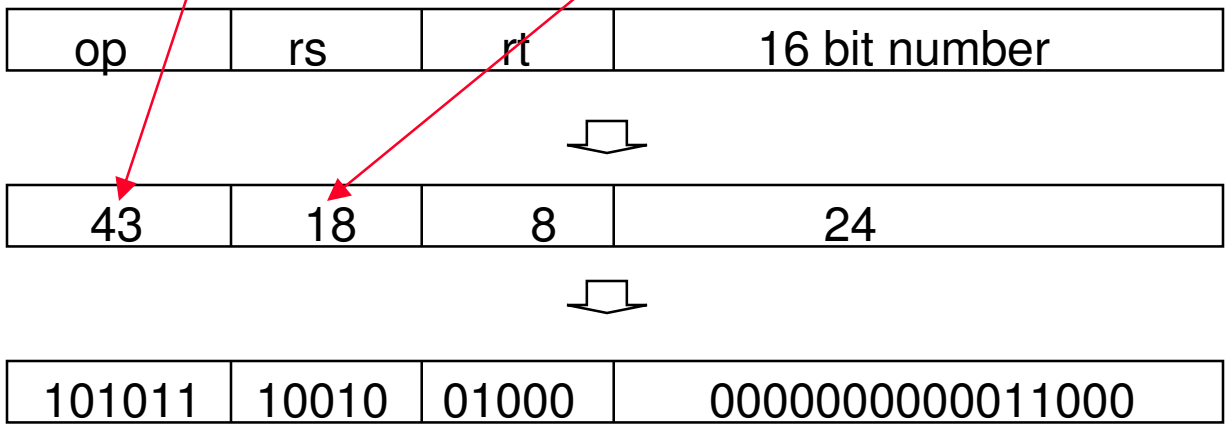
- Example: `sw $t0, 24($s2)`



- A 16-bit address means access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register `$s2`

Machine Language - Store Instruction

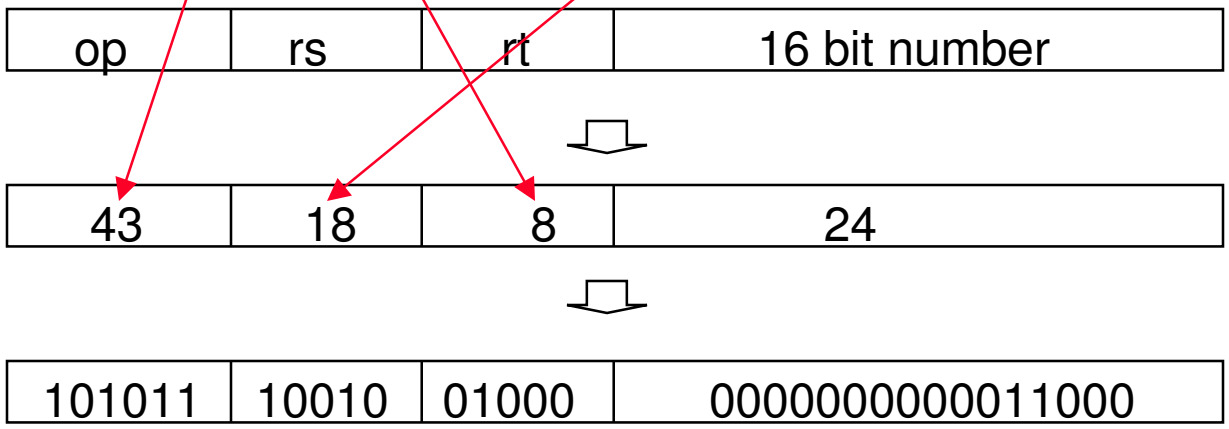
- Example: `sw $t0, 24($s2)`



- A 16-bit address means access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register `$s2`

Machine Language - Store Instruction

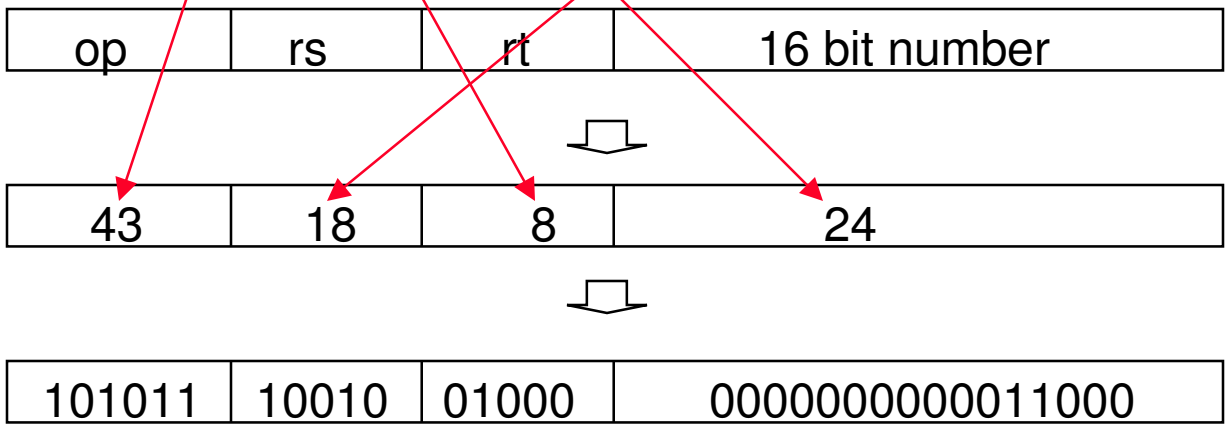
- Example: `sw $t0, 24($s2)`



- A 16-bit address means access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register `$s2`

Machine Language - Store Instruction

- Example: `sw $t0, 24($s2)`



- A 16-bit address means access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register `$s2`

Review: MIPS Organization

- Arithmetic instructions – to/from the register file
- Load/store word and byte instructions – from/to memory