Computer Architecture
(ENE1004)

Lec – 3: Instructions: Language of the Computer (Chapter 2) - 2

Review: MIPS Instructions

- MIPS instructions: assembly instructions (or its corresponding binary values)
- Arithmetic instructions
  - Three operands, each of which can be register or constant

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 instructions
  - Moving data between a register within CPU and a location of the memory
  - Two operands; one is a register and the other is the memory location

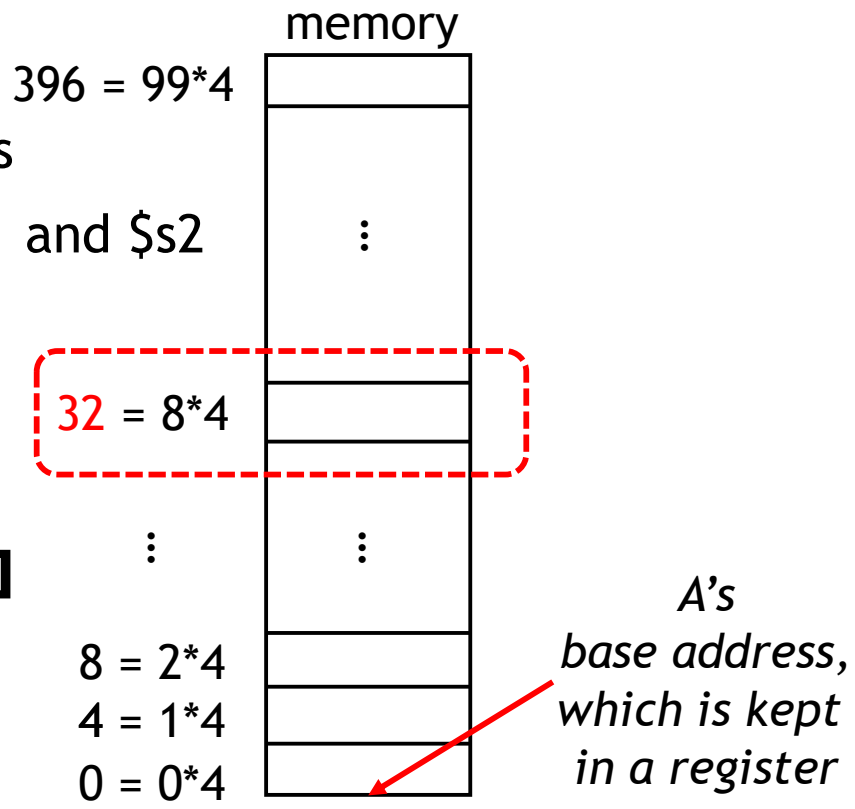
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory

Review: Load Word Instruction

- lw (load word)
  - lw register_to_be_loaded memory_address_to_be_accessed
  - memory_address_to_be_accessed: index(register that holds the base address of memory)
  - Here, index must be specified in bytes
- Example
  - Assumption 1: A is an (4-byte) integer array with 100 elements
  - Assumption 2: Compilers associated variables g and h with $s1 and $s2
  - Assumption 3: starting (base) address is in $s3
  - C code: **g = h + A[8];**

    is translated into
  - **lw $t0, 32($s3) # temporary reg $t0 gets A[8]**

    add $s1, $s2, $t0 # g = h + A[8]



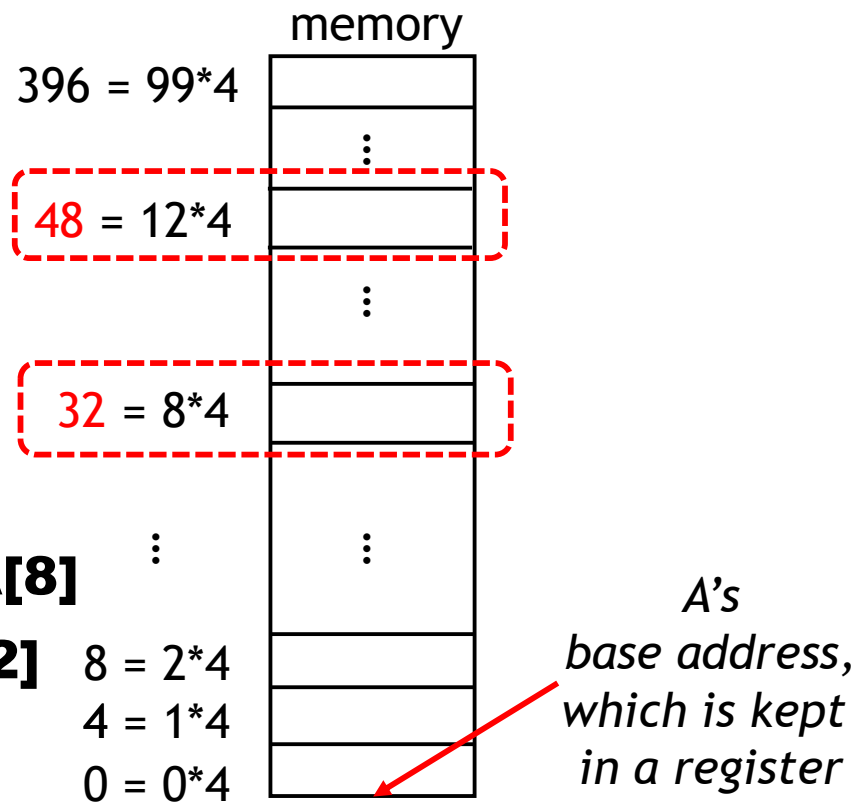
Review: Store Word Instruction

- sw (store word)
  - sw register_to_be_stored memory_address_to_be_accessed
  - memory_address_to_be_accessed: index(register that holds the base address of memory)
  - Here, index must be specified in bytes
- Example
  - Assumption 1: variable h is associated with register $s2
  - Assumption 2: the base address of the array A is in $s3
  - C code: A[12] **= h + A[8];**

    is translated into
  - **lw $t0, 32($s3) # temporary reg $t0 gets A[8]**

    add $t0, $s2, $t0 # temporary reg $t0 gets h + A[8]

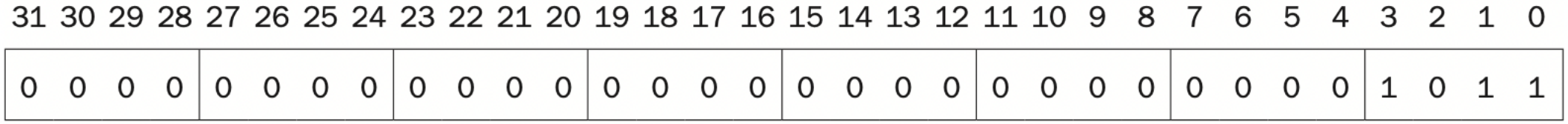
    sw $t0, 48($s3) # stores h + A[8] back into A[12]



Representing Numbers

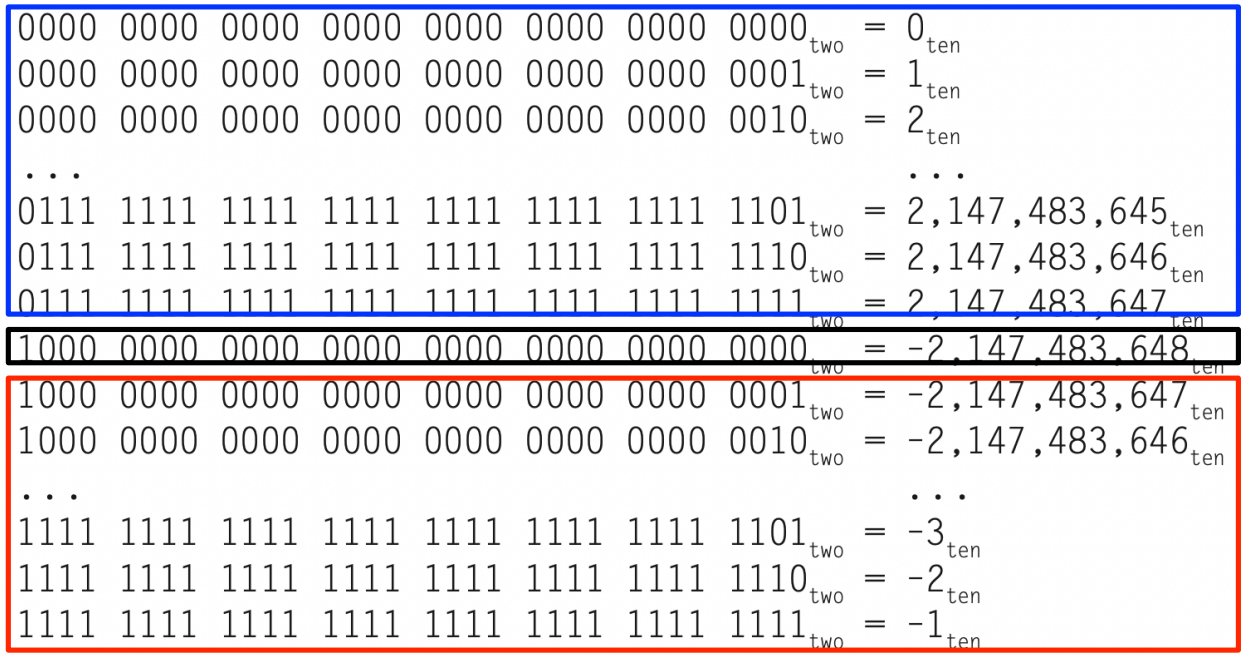
- Computer hardware (including MIPS processor) uses binary (base 2) numbers
  - We humans use decimal (base 10) numbers
  - $11 = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 1011_2$
- How does a MIPS processor understand/express a number?
  - Recall that MIPS is based on the unit of a word (1 word = 4 bytes = 32 bits)
  - MIPS numbers the 32 bits 0, 1, 2, 3, …. from right to left in a word

$11 = 1011_2$



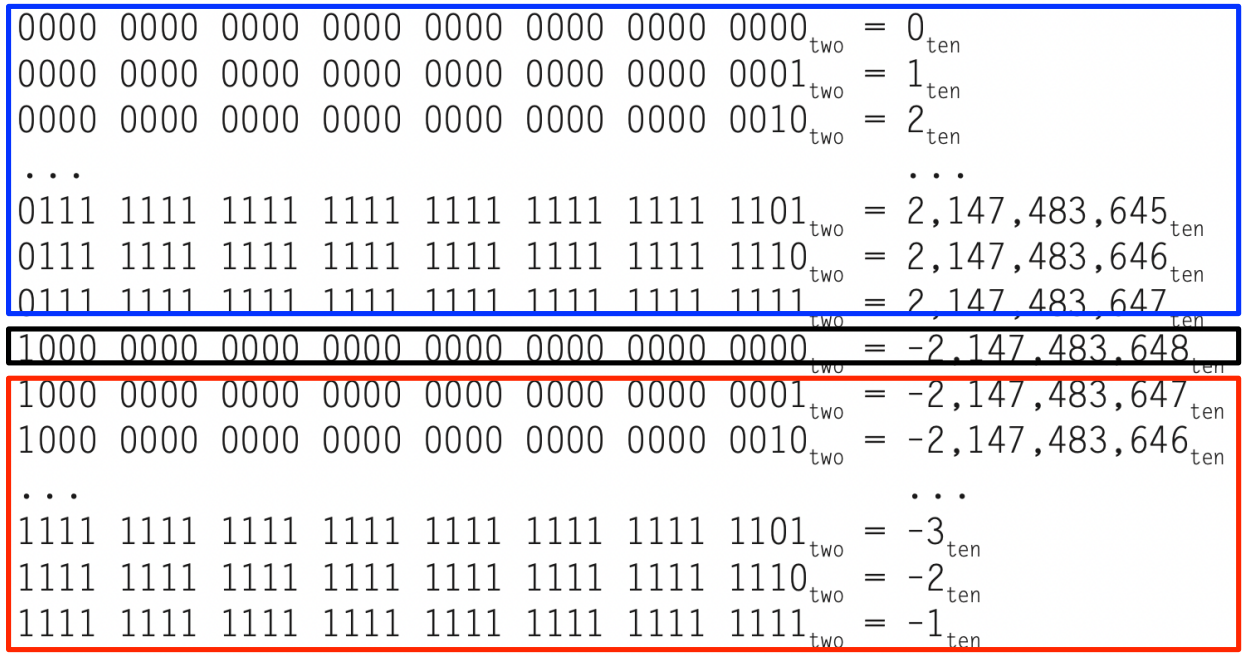
  - We refer to “bit 31” as most significant bit, and “bit 0” as least significant bit
- How many numbers a MIPS word can represent?
  - Using 32 bits, we can express 2^{32} = 4,294,967,296 different bit patterns (so different numbers)
  - We need to express both positive and negative numbers
  - How de we distinguish the positive from the negative?

Representing Numbers – Two's Complement



- There have been different ways of presenting negative numbers
- Today's processors pick two's complement representation
  - This makes design of hardware simple
  - Leading 0 mean a positive number
  - Leading 1 mean a negative number

Representing Numbers – Two's Complement



- The positive half of the numbers are from 0 to 2,147,483,647 (2^{31}-1)
- 1000…0000 represents the most negative number - 2,147,483,648
- It is followed by a declining set of negative numbers from -2,147,483,647 to -1
- Negation (e.g., conversion between 2 and -2)
  - 2 = 0000 0000 0000 0000 0000 0000 0000 0010
  - (1) inverting the bits: 1111 1111 1111 1111 1111 1111 1111 1101
  - (2) adding one: 1111 1111 1111 1111 1111 1111 1111 1110 = -2

Representing MIPS Instructions

- MIPS instructions (**add**, **sub**, **addi**, **lw**, **sw**) are also information
  - So, instructions can be represented as numbers
- How can a MIPS instruction be represented as a binary number in a word?
- Example: **add $t0, $s1, $s2**

0	17	18	8	0	32
000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - 32 bits is divided into segments, which is called fields
  - The first and last fields in combination tell that this is an addition (add: 0+32)
  - The second field gives the number of the register that is the first source of operands ($s1: 17)
  - The third field gives the the number of the register that is the other source operands ($s2: 18)
  - The fourth field contains the number of the register that is the destination ($t0: 8)
  - The fifth field is unused in this instruction (so, it is set to 0)

Representing MIPS Instructions: R-Type

- An R-type instruction is comprised of six different fields
  - op: basic operation of the instruction, called opcode
  - rs: the first register source operand
  - rt: the second register source operand
  - rd: the register destination operand
  - shamt: shift amount (this is used in shift instructions; otherwise, this field is filled with 0s)
  - funct: function, called function code, selects the specific variant of the operation in op
- Two representative R-type instructions we know

Instruction	Format	op	rs	rt	rd	shamt	funct
add	R	0	reg	reg	reg	0	32_{ten}
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}

Representing MIPS Instructions: Register Number

register	assembly name	Comment
r0	$zero	Always 0
r1	$at	Reserved for assembler
r2-r3	$v0-$v1	Stores results
r4-r7	$a0-$a3	Stores arguments
r8-r15	$t0-$t7	Temporaries, not saved
r16-r23	$s0-$s7	Contents saved for use later
r24-r25	$t8-$t9	More temporaries, not saved
r26-r27	$k0-$k1	Reserved by operating system
r28	$gp	Global pointer
r29	$sp	Stack pointer
r30	$fp	Frame pointer
r31	$ra	Return address

- A MIPS processor includes 32 registers
- Some are reserved for specific purposes
  - Register 1 ($at) for assembler
  - Registers 26-27 ($k0-$k1) for operating system
- Many are used for specific purposes
  - Registers 2-7 for function calls
  - Registers 28-31 for program flow
- Programs use the other registers
  - $t0 to $t7 are mapped to 8 to 15
  - $s0 to $s7 are mapped to 16 to 23
  - $t8 and $t9 are mapped to 24 and 25
- Example: **add $t0, $s1, $s2**
  - rs = 17, rt = 18, rd = 8

Representing MIPS Instructions: I-Type

- R-type does not fit to other instructions
  - Data transfer instructions (one register and a memory address)
  - Immediate instructions (two registers and a constant)
- An I-type instruction is comprised of four different fields
- Example: **lw** $t0, 32($s3)
  - *op* = instruction (**lw** = 35), *rs* = base address ($s3 = 19)
  - *rt* = destination register ($t0 = 8), *constant/address* = index (32)

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

Representing MIPS Instructions: Example

- Assumption: $t1 holds the base address of an integer array A, $s2 corresponds to h
- **A[300] = h + A[300];** is compiled to

```
lw  $t0,  1200($t1)    # temporary reg $t0 gets A[300]
add  $t0,  $s2,  $t0   # temporary reg $t0 gets h + A[300]
sw  $t0,  1200($t1)    # store h + A[300] back into A[300]
```

Op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Representing MIPS Instructions: Summary

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

- R-type and I-type
- Each type has different fields with fixed sizes
- Opcode is unique for each instruction
- Registers as source/destination operands are specified their numbers