Chapter 2

Instructions: Language of the Computer

Representing Instructions

- Each piece of an instruction can be considered as an individual number, placing these numbers side by side forms the instruction
- Registers are referred to in instructions, so there must be **a convention to map register names into numbers.**
- In MIPS assembly language
  - Register $t0 – $t7 map into register 8 – 15
  - Register $t8 – $t9 map into register 24 – 25
  - Register $s0 – $s7 map into register 16 – 23
  - So, $t0 means register 8, $t1 means register 9, and…

Representing Instructions

- **Translating a MIPS assembly instruction to a machine instruction**
- add $t0,$s1,$s2
- Decimal representation is

0	17	18	8	0	32

- Binary representation is

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- The first and the last fields in combination tell this instruction performs addition
- The second and third files are operands of the addition operation
- The fifth filed is unused

Representing Instructions

- **Instruction format:** A form of representation of an instruction composed of fields of binary numbers

- MIPS instruction takes exactly 32 bits (a word size)

- To distinguish it from assembly language, we call the numeric version of instructions **machine language** and a sequence of such instructions *machine code*.

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

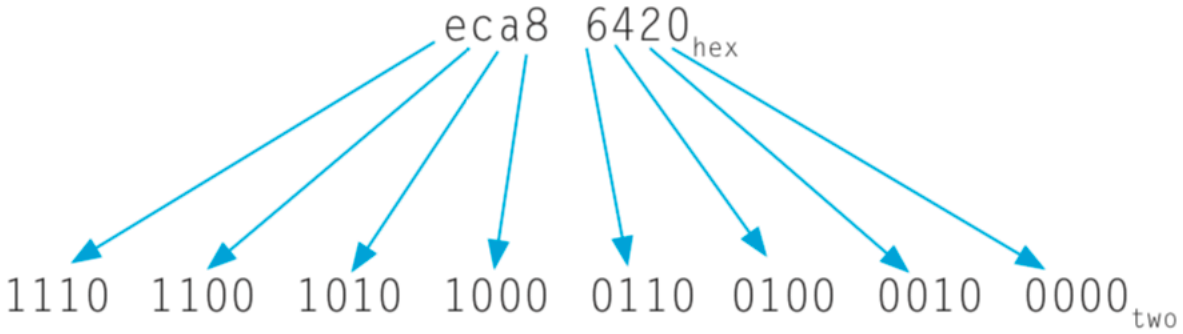
- Example: eca8 6420
  - 1110 1100 1010 1000 0110 0100 0010 0000

Hexadecimal

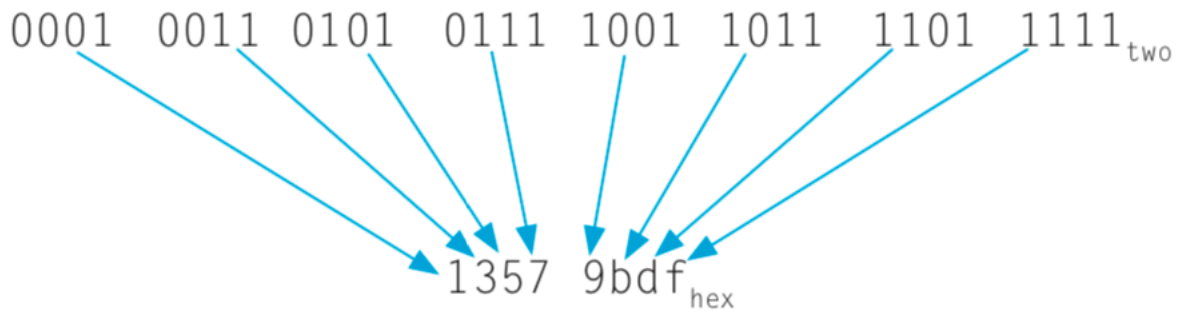
- Hexadecimal: Numbers in base 16.
- The following picture shows converts between hexadecimal and binary.

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

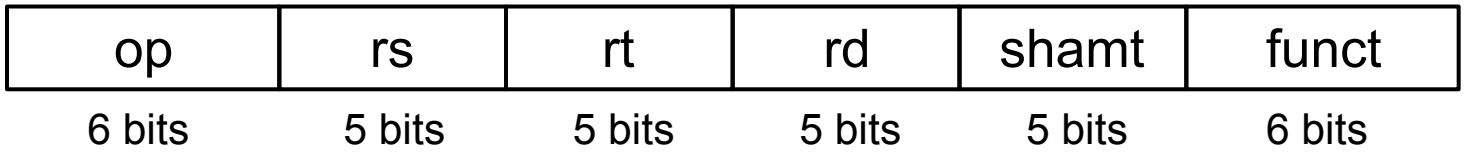
Hexadecimal



And then the other direction:

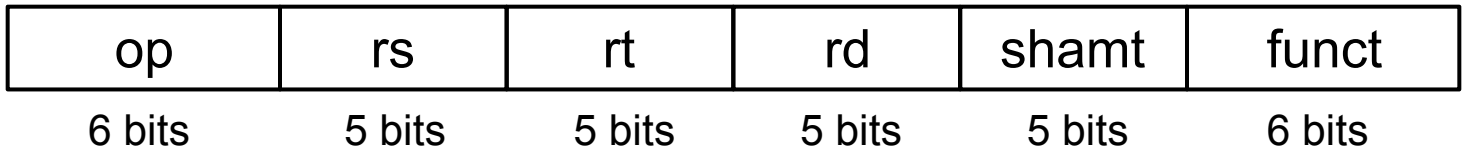


MIPS fileds



- Instruction fields
  - op: operation code (opcode)
  - rs: the first register source operand
  - rt: the second register source operand
  - rd: the register destination operand
  - shamt: shift amount (00000 for now)
  - funct: function code (selects the specific variant of the operation in the op field)

Example



add $t0, $s1, $s2

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

$00000010001100100100000000100000_2 = 02324020_{16}$

Instruction formats

- **A problem occurs when an instruction needs longer fields.** For example, load instruction must specify two registers and one constant. The constant within the load instruction would be limited to only 2^5 or 32.

- ***Design Principle 4: Good design demands good compromises.***
  - **Keep all instructions the same length**
  - **We need different kind of instruction formats for different kind of instructions**

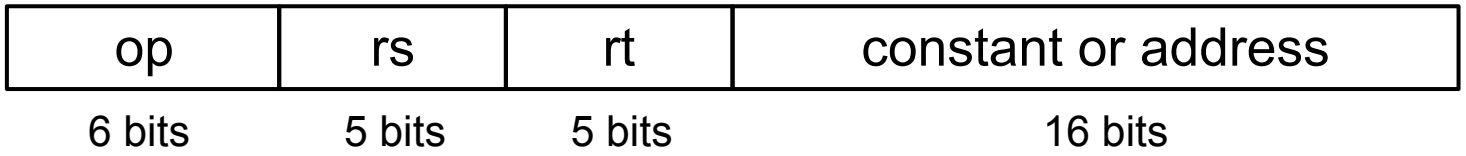
- The field of R-type or R-format

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

- The field of I-type or I-format

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

I- format instruction



- lw $t0,32($s3)
- **rs field is the base register**; 19 for $s3 is placed in rs field
- **rt field is the destination register** which receive the result of load instruction; 8 for $t0 is placed in rt field;
- The 16-bit address means a load word instruction can load any word within a region of +/- 2^15 or 32,768 bytes of the address in the base register rs.

MIPS instruction encoding

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

FIGURE 2.5 MIPS instruction encoding. In the table above, "reg" means a register number between 0 and 31, "address" means a 16-bit address, and "n.a." (not applicable) means this field does not appear in this format. Note that add and sub instructions have the same value in the op field; the hardware uses the funct field to decide the variant of the operation: add (32) or subtract (34).

Example- Translating MIPS Assembly Language into Machine Language

- $t1 has the base of the array A and $s2. correspond to h

```
A[300] = h + A[300];
```

is compiled into

```
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) # Stores h + A[300] back into A[300]
```

What is the MIPS machine language code for these three instructions?

- The three machine language instructions

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

- The binary equivalent to the decimal form

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

Summarize of the portions of MIPS machine language described in this section

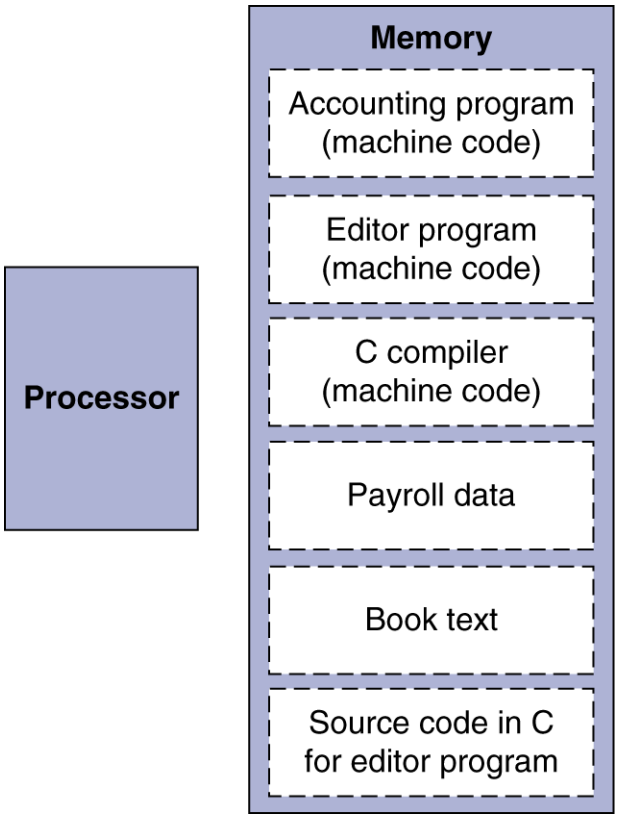
MIPS machine language

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

FIGURE 2.6 MIPS architecture revealed through Section 2.5. The two MIPS instruction formats so far are R and I. The first 16 bits are the same: both contain an *op* field, giving the base operation; an *rs* field, giving one of the sources; and the *rt* field, which specifies the other source operand, except for load word, where it specifies the destination register. R-format divides the last 16 bits into an *rd* field, specifying the destination register; the *shamt* field, which Section 2.6 explains; and the *funct* field, which specifies the specific operation of R-format instructions. I-format combines the last 16 bits into a single *address* field.

Stored Program Computers

The BIG Picture



- Instructions represented in binary, just like data
- Instructions and data stored in memory
- Programs can operate on programs
  - e.g., compilers, linkers, …
- Binary compatibility allows compiled programs to work on different computers
  - Standardized ISAs

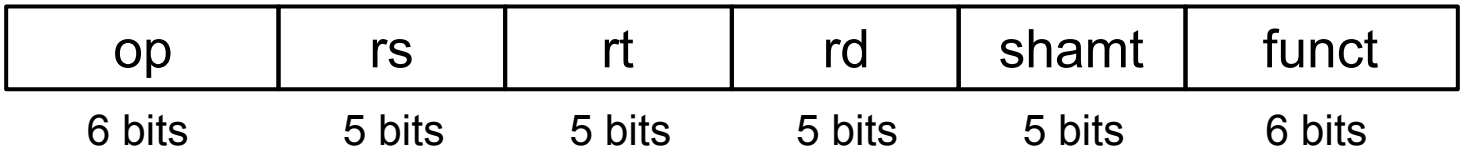
Logical Operations

- 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

Shift Operations



- shamt: how many positions to shift
- **Shift left logical (sll)**
  - Shift left and fill with 0 bits
  - sll by *i* bits multiplies by 2^i
- **Shift right logical (srl)**
  - Shift right and fill with 0 bits
  - srl by *i* bits divides by 2^i (unsigned only)
- For example, if $s0 contains

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001_{two} = 9_{ten}$$

and the instruction to shift left by 4 was executed, the new value would be:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_{two} = 144_{ten}$$

Shift Operations- example

- assuming that the original value is in register $s0 and the result should go in register $t2

```
sll  $t2,$s0,4  # reg $t2 = reg $s0 << 4 bits
```

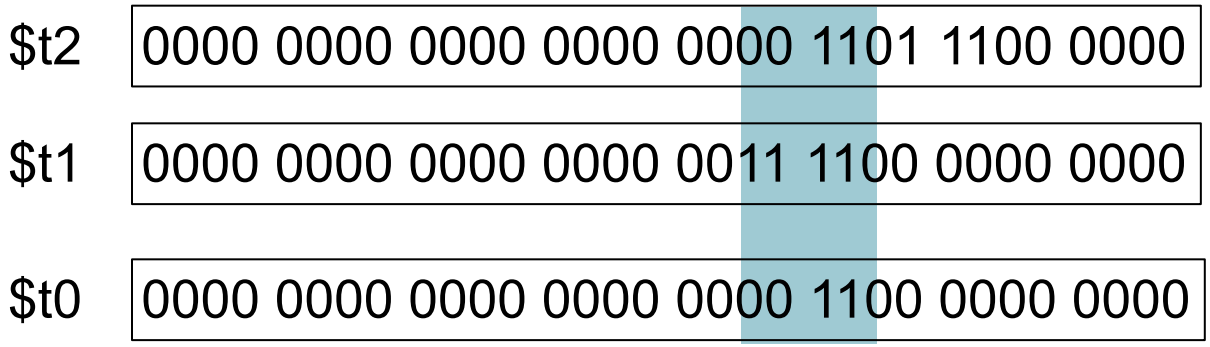
- The encoding of sll is 0 in both the op and funct fields, rd contains 10 (register $t2), rt contains 16 (register $s0), and shamt contains 4. The rs field is unused and thus is set to 0.

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

AND Operations

- Useful to mask bits in a word
  - Select some bits, clear others to 0

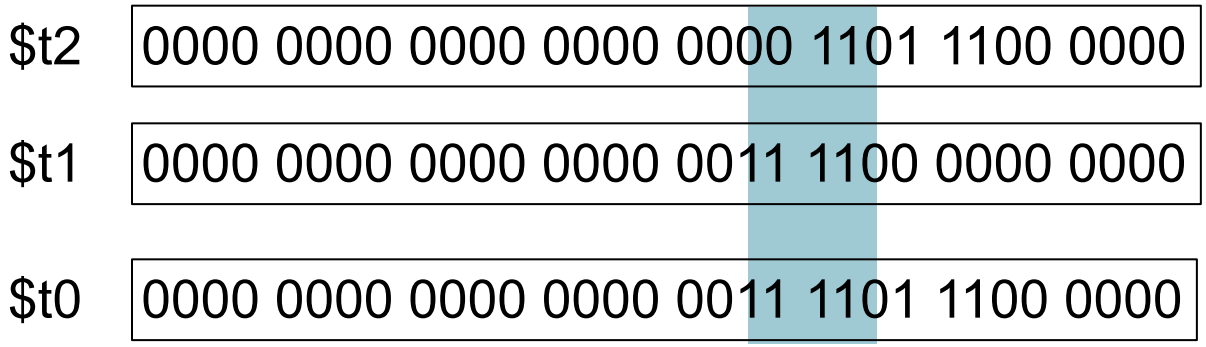
and $t0, $t1, $t2



OR Operations

- Useful to include bits in a word
  - Set some bits to 1, leave others unchanged

or $t0, $t1, $t2



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
```

Register 0: always read as zero

$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	1111 1111 1111 1111 11 00 0011 1111 1111

Reading assignment

- Read 2.5 and 2.6 of the text book