MIPS Architecture

Presented By

Dr. Banchhanidhi Dash

School of Computer Engineering
KIIT University

Why MIPS?

Architecture: Defined by the instruction set. Examples: MIPS, x86 family, 68k family, ARM, PowerPC, VAX, SPARC,etc.

MIPS - (Microprocessor without Interlocking Pipeline Stages)archiceture developed by MIPS Computer Systems, now MIPS Technologies, based in the United States.

The **MIPS** is an example of **RISC (Reduced Instruction Set Computing)**

- RISC uses simplified instruction set (as opposed to a complex set which is used in CISC)
- Provides higher performance architecture capable of executing those instructions using fewer microprocessor cycles per instruction

MIPS

- MIPS architecture:
  - First publicly known implementations of RISC architectures
  - Grew out of research at Stanford University
- MIPS computer system founded in 1984:
  - The R2000 is a 32-bit microprocessor chip set developed by MIPS Computer Systems that implemented the MIPS I instruction set architecture (ISA).
  - Introduced in January 1986, it was the first commercial implementation of the MIPS architecture and the first commercial RISC processor available to all companies.
  - The R2000 competed with Motorola 68000 and Intel Corporation 80386 microprocessors

Commercial Success of MIPS

- Popularly used as IP-cores (building-blocks) for embedded processor designs.
  - Both 32-bit and 64-bit basic cores are offered --- the design is licensed as MIPS32 and MIPS64.
  - MIPS cores have been commercially successful --- used in many consumer and industrial applications.
- MIPS cores can be found in:
  - Modern Cisco and Linksys routers, cable modems and ADSL modems, smartcards, laser printer engines, set-top boxes, robots, handheld computers, Sony PlayStation 2 and Sony PlayStation Portable.

- There are multiple versions of MIPS: including MIPS I, II, III, IV, and V
- the current version of MIPS is MIPS32/64
- MIPS is a load/store architecture (also known as a register-register architecture); except for the load/store instructions used to access memory, all instructions operate on the registers.

Operands in MIPS

❑ Registers

– MIPS has 32 architected 32-bit registers
– Effective use of registers is key to program performance

❑ Data Transfer

– 32 words in registers, millions of words in main memory
– Instruction accesses memory via memory address
– Address indexes memory, a large single-dimensional array

❑ Alignment

– Most architectures address individual bytes
– Addresses of sequential words differ by 4
– Words must always start at addresses that are multiples of 4

MSB

byte-3	byte-2	byte-1	byte-0
byte-7	byte-6	byte-5	byte-4

LSB

Register Number	Conventional Name	Usage
$0	$zero	Hard-wired to 0
$1	$at	Reserved for pseudo-instructions
$2 - $3	$v0, $v1	Return values from functions
$4 - $7	$a0 - $a3	Arguments to functions - not preserved by subprograms
$8 - $15	$t0 - $t7	Temporary data, not preserved by subprograms
$16 - $23	$s0 - $s7	Saved registers, preserved by subprograms
$24 - $25	$t8 - $t9	More temporary registers, not preserved by subprograms
$26 - $27	$k0 - $k1	Reserved for kernel. Do not use.
$28	$gp	Global Area Pointer (base of global data segment)
$29	$sp	Stack Pointer
$30	$fp	Frame Pointer
$31	$ra	Return Address
$f0 - $f3	-	Floating point return values
$f4 - $f10	-	Temporary registers, not preserved by subprograms
$f12 - $f14	-	First two arguments to subprograms, not preserved by subprograms
$f16 - $f18	-	More temporary registers, not preserved by subprograms
$f20 - $f31	-	Saved registers, preserved by subprograms

MIPS Instruction format

Instructions are divided into three types: R, I and J.
Every instruction starts with a 6-bit opcode. In addition to the opcode, R-type instructions specify three registers, a shift amount field, and a function field; I-type instructions specify two registers and a 16-bit immediate value; J-type instructions follow the opcode with a 26-bit jump target

Type	-31- format (bits) -0-					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

Instruction format

MIPS Addressing Modes

Register addressing

Operand is stored in a register. R-Type

Base or displacement addressing

Operand at the memory location specified by a register value plus a displacement given in the instruction. I-Type
Eg: lw, $t0, 25($s0)

Immediate addressing

Operand is a constant within the instruction itself. I-Type

PC-relative addressing

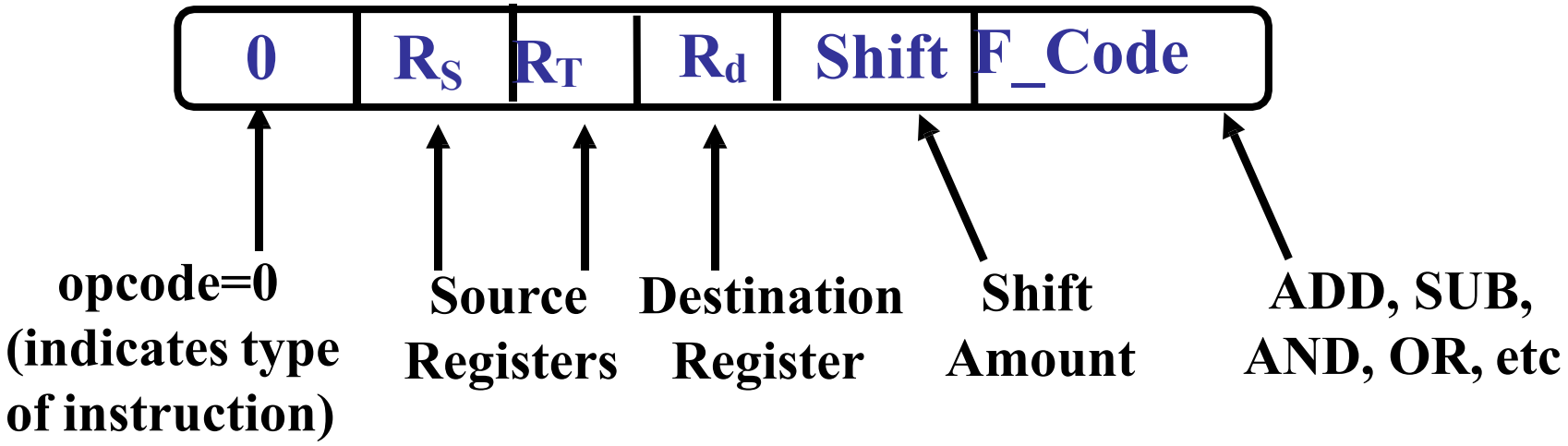
The address is the sum of the PC and a constant in the instruction. I-Type
Eg: beq $t2, $t3, 25 # if ($t2==$t3), goto PC+4+100

Pseudodirect addressing

The 26-bit constant is logically shifted left 2 positions to get 28 bits. Then the upper 4 bits of PC+4 is concatenated with this 28 bits to get the new PC address. J-type, e. g.,

R Instructions

R-Format (32-bits): *ALU (core) Instructions*



Format	Fields						Used by
	6 bits 31–26	5 bits 25–21	5 bits 20–16	5 bits 15–11	5 bits 10–6	6 bits 5–0	
R-format	opcode	rs	rt	rd	shamt	F_code (funct)	ALU instructions except immediate, Jump Register (JR)

Basic R-format Instructions
Have op 0. (all of them!)
Also have:
rs: 1st register operand (register source) (5 bits)
rt: 2nd register operand (5 bits)
rd: register destination (5 bits)
shamt: shift amount (0 when not applicable) (5 bits)
funct: function code (identifies the specific R-format instruction) (6 bits)

Name	Format	Layout						Example
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
		op	rs	rt	rd	shamt	funct	
add	R	0	2	3	1	0	32	add $1, $2, $3
addu	R	0	2	3	1	0	33	addu $1, $2, $3
sub	R	0	2	3	1	0	34	sub $1, $2, $3
subu	R	0	2	3	1	0	35	subu $1, $2, $3
and	R	0	2	3	1	0	36	and $1, $2, $3
or	R	0	2	3	1	0	37	or $1, $2, $3
nor	R	0	2	3	1	0	39	nor $1, $2, $3
slt	R	0	2	3	1	0	42	slt $1, $2, $3
sltu	R	0	2	3	1	0	43	sltu $1, $2, $3

NOTE: op is 0, so funct disambiguates

R type Instruction example

MIPS Assembly Language

add	rd, rs, rt	rd = rs + rt;
addu	rd, rs, rt	rd = rs + rt;
sub	rd, rs, rt	rd = rs - rt;
subu	rd, rs, rt	rd = rs - rt;
and	rd, rs, rt	rd = rs & rt;
or	rd, rs, rt	rd = rs | rt;
xor	rd, rs, rt	rd = rs ^ rt;
nor	rd, rs, rt	rd = ~(rs | rt);
sll	rd, rt, amount	rd = rt << amount;
srl	rd, rt, amount	rd = (unsigned) rt >> amount;
sra	rd, rt, amount	rd = (signed) rt >> amount;
slt	rd, rs, rt	rd = (rs < rt);
sltu	rd, rs, rt	rd = (rs < rt);
mult	rs, rt	hilo = rs * rt;
multu	rs, rt	hilo = rs * rt;
div	rs, rt	hi = rs % rt; lo = rs / rt;
divu	rs, rt	hi = rs % rt; lo = rs / rt;

opcode and function code value

Mnemonic	Meaning	Type	Opcode	Functioncode
add	Add	R	0x00	0x20
addi	Add Immediate	I	0x08	NA
addiu	Add Unsigned Immediate	I	0x09	NA
addu	Add Unsigned	R	0x00	0x21
and	Bitwise AND	R	0x00	0x24
andi	Bitwise AND Immediate	I	0x0C	NA
beq	Branch if Equal	I	0x04	NA
bgtz	Branch on Greater Than Zero	I	0x07	NA
blez	Branch if Less Than or Equal to Ze	I	0x06	NA
bne	Branch if Not Equal	I	0x05	NA
div	Divide	R	0x00	0x1A
divu	Unsigned Divide	R	0x00	0x1B
j	Jump to Address	J	0x02	NA
jal	Jump and Link	J	0x03	NA
jr	Jump to Address in Register	R	0x00	0x08
lb	Load Byte	I	0x20	NA
lbu	Load Byte Unsigned	I	0x24	NA
lhu	Load Halfword Unsigned	I	0x25	NA
lui	Load Upper Immediate	I	0x0F	NA
lw	Load Word	I	0x23	NA
mfc0	Move from Coprocessor 0	R	0x10	NA
mfhi	Move from HI Register	R	0x00	0x10
mflo	Move from LO Register	R	0x00	0x12
mthi	Move to HI Register	R	0x00	0x11
mtlo	Move to LO Register	R	0x00	0x13
mult	Multiply	R	0x00	0x18
multu	Unsigned Multiply	R	0x00	0x19
nor	Bitwise NOR (NOT-OR)	R	0x00	0x27
or	Bitwise OR	R	0x00	0x25
ori	Bitwise OR Immediate	I	0x0D	NA
sb	Store Byte	I	0x28	NA
sh	Store Halfword	I	0x29	NA
sll	Logical Shift Left	R	0x00	0x00
slt	Set to 1 if Less Than	R	0x00	0x2A
slti	Set to 1 if Less Than Immediate	I	0x0A	NA
sltiu	Set to 1 if Less Than Unsigned Imr	I	0x0B	NA
sltu	Set to 1 if Less Than Unsigned	R	0x00	0x2B
sra	Arithmetic Shift Right (sign-extenc	R	0x00	0x03
srl	Logical Shift Right (0-extended)	R	0x00	0x02
sub	Subtract	R	0x00	0x22
subu	Unsigned Subtract	R	0x00	0x23
sw	Store Word	I	0x2B	NA
xor	Bitwise XOR (Exclusive-OR)	R	0x00	0x26

Example

```
add $s0, $s1, $s2        (registers 16, 17, 18)
```

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
000000	10001	10010	10000	00000	100000

C: `a = b + c`

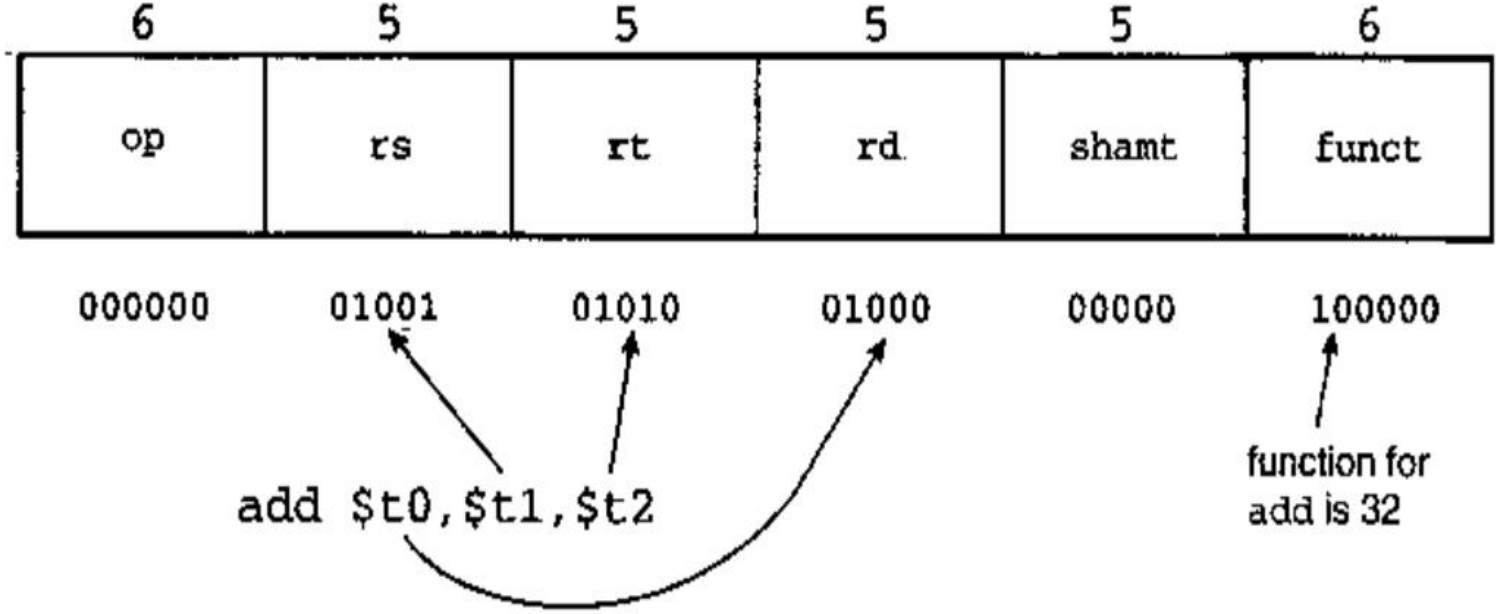
assembly code: `add $s0, $s1, $s2    # add rd, rs, rt`

machine code: `000000 10001 10010 10000 0000 100000`

`(op rs rt rd shamt funct)`

NOTE: Order of components in machine code is different from assembly code. Assembly code order is similar to C, destination first. Machine code has destination last.

R type Instruction encoding example



Note well that the registers appear in most assembly language instructions in the order rd,rs,rt, but are physically encoded in the instruction in the order rs,rt,rd.

R type Instruction encoding example

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

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

$$00000010001100100100000000100000_2 = 02324020_{16}$$

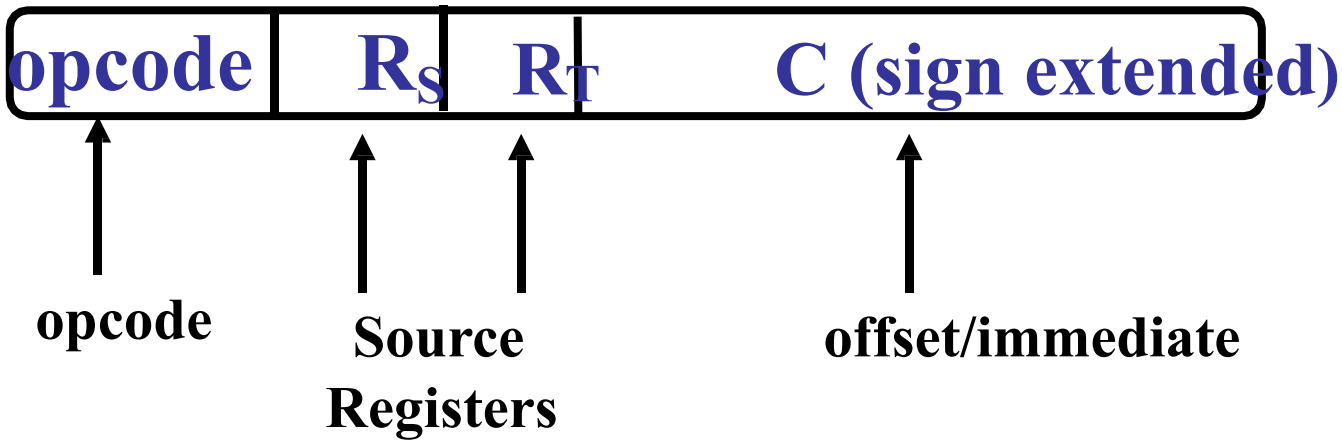
For example: add $4, $3, $2

000000 00011 00010 00100 00000 10 0000

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

I Instructions

I-Format (32-bits): Load/Store, Immediate ALU, Branch



Format	Fields						Used by
	6 bits 31–26	5 bits 25–21	5 bits 20–16	5 bits 15–11	5 bits 10–6	6 bits 5–0	
I-format	opcode	rs	rt	offset/immediate			Load, store, Immediate ALU, beq, bne

Basic I-format Instructions

Still have 2 registers and a constant value immediately present in the instruction.

rs: operand or base address (5 bits)

rt: operand or data register (5 bits)

immediate: value or offset (16 bits)

Example

```
addi $t2, $s3, 4        (registers 10 and 19)
```

op	rs	rt	immediate
8	19	10	4
001000	10011	01010	0000000000000100

Immediate instructions

`lw $t0, 32($s3)` (registers 8 and 19)

op	rs	rt	immediate
35	19	8	32
100011	10011	01000	0000000000100000

`lui $t0, 1028` (register 8)

op	rs	rt	immediate
15	0	8	1028
001111	00000	01000	0000010000000100

This format is used for three different types of MIPS instruction

- Immediate instructions
- Memory Reference instructions
- Conditional branch instructions

Instruction	opcode
addi rt, rs, imm	001000
addiu rt, rs, imm	001001
andi rt, rs, imm	001100

Immediate instructions

TABLE 9-3: Memory Access Instructions in the MIPS ISA

Instruction	Assembly Code	Operation	Comments
load word	lw $s1, k($s2)	$s1 = Memory[$s2 + k]	Read 32 bits from memory; memory address = register content + *k*; *k* is 16-bit offset
store word	sw $s1, k($s2)	Memory[$s2 + k] = $s1	Write 32 bits to memory; memory address = register content + *k*; *k* is 16-bit offset;
load halfword	lh $s1, k($s2)	$s1 = Memory[$s2 + k]	Read 16 bits from memory; sign-extend and load into register
store halfword	sw $s1, k($s2)	Memory[$s2 + k] = $s1	Write 16 bits to memory
load byte	lb $s1, k($s2)	$s1 = Memory[$s2 + k]	Read byte from memory; sign-extend and load to register
store byte	sb $s1, k($s2)	Memory[$s2 + k] = $s1	Write byte to memory
load byte unsigned	lbu $s1, k($s2)	$s1 = Memory[$s2 + k]	Read byte from memory; byte is 0-extended
load upper immediate	lui $s1, k	$s1 = k * 2^{16}	Loads constant *k* to upper 16 bits of register

Memory Reference instructions

beq rs, rt, label if (rs == rt) goto label

bne rs, rt, label if (rs != rt) goto label

Conditional branch instructions

Example:beq $t1,$t2,Label

bne $t1,$t2,Label

Instruction	opcode
beq rs, rt, label	000100
bgez rs, label	000001
bgtz rs, label	000111
blez rs, label	000110
bltz rs, label	000001

Conditional branch instructions

For the bgez, bgtz, blez, and bltz instructions, the rt field is used as an extension of the opcode field.

BEQ Instruction

The BEQ instruction branches the PC if the first source register's contents and the second source register's contents are equal.

It's syntax is:

BEQ $first source register's address, $second source register's address, branch value.

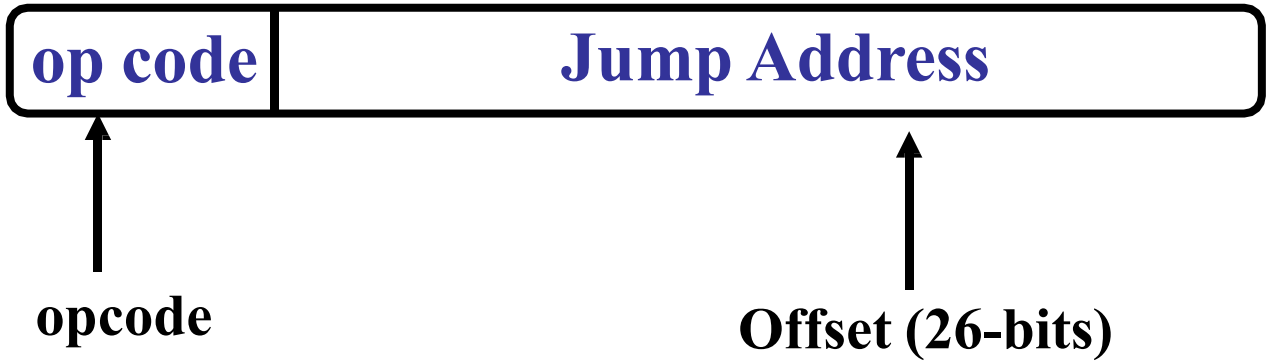
Example:BEQ $9, $11, 8.

The instruction's equivalent in binary is:

branch target address=PC+4+[[sign-extended(offset)]<<2]

JUMP Instructions

J-Format (32-bits)



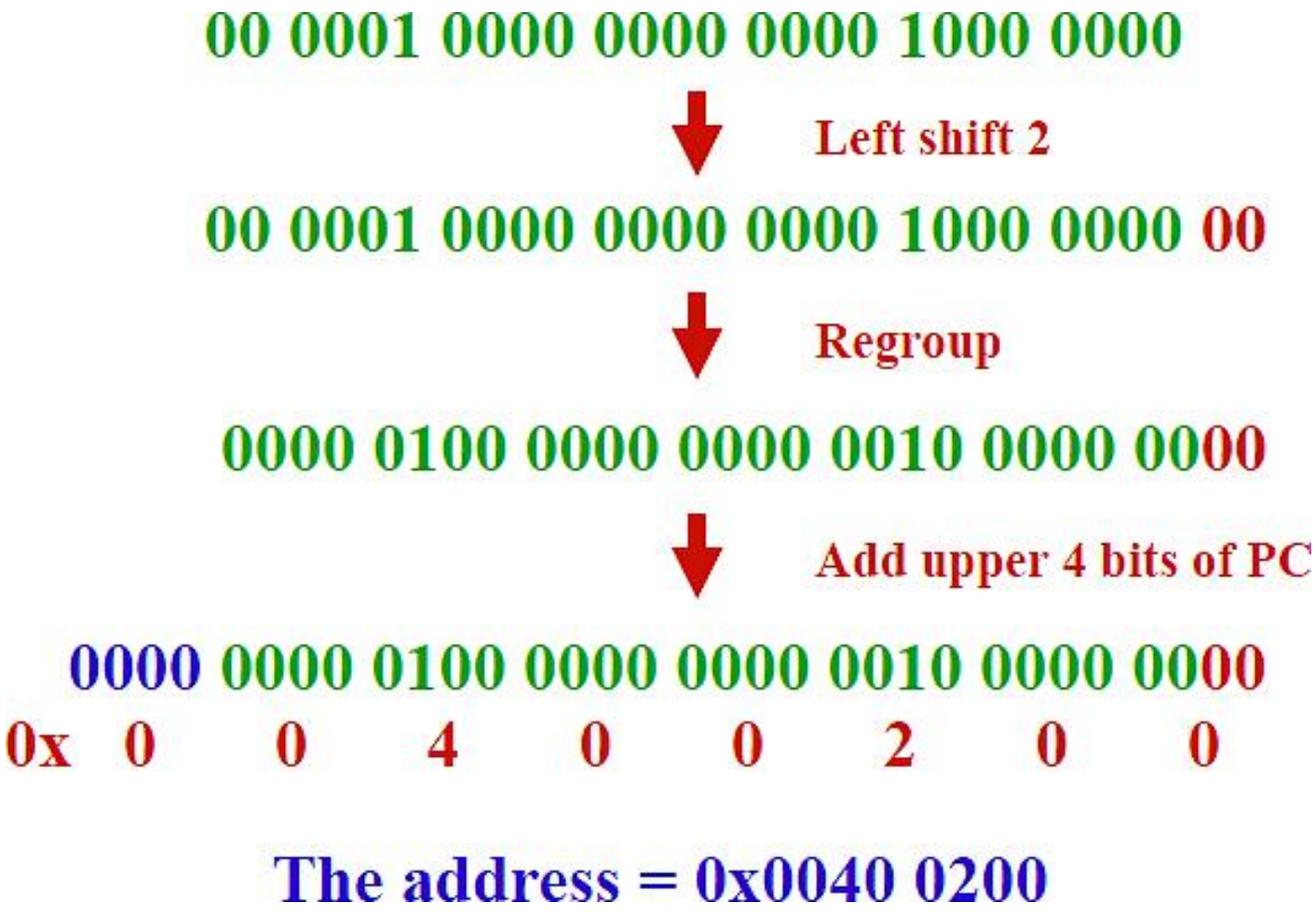
Format	Fields						Used by
	6 bits 31–26	5 bits 25–21	5 bits 20–16	5 bits 15–11	5 bits 10–6	6 bits 5–0	
J-format	opcode	target address					Jump (J), Jump and Link (JAL)

JUMP Instructions Example: J addr
J 2500

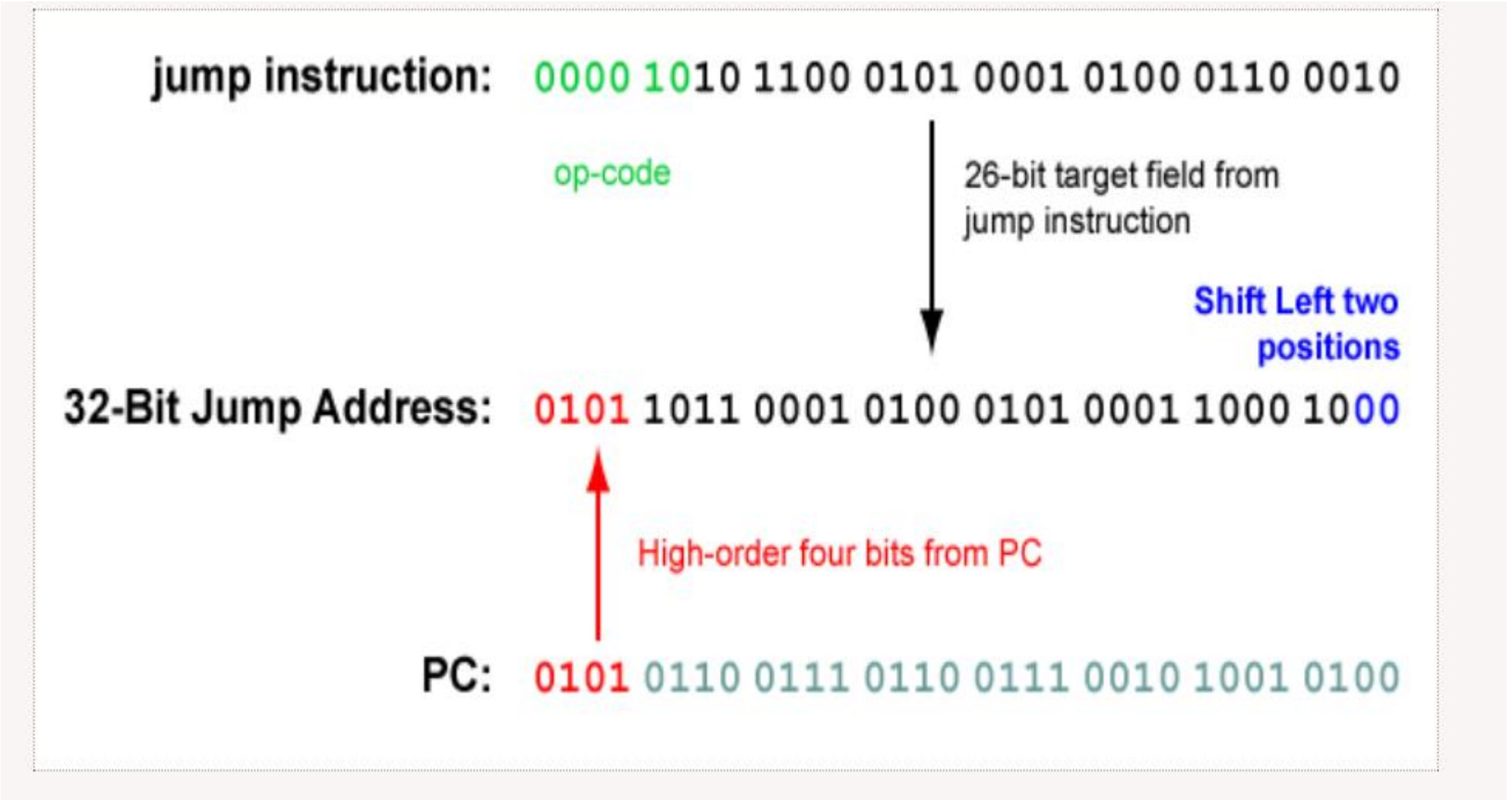
JUMP Instructions

Example

The 26-bit field in a jump instruction is 0x0100080. What is the actual address the jump instruction refers to if the top four bits of the Program Counter are 0000?



The address = 0x0040 0200


jump instruction:
0000 1010 1100 0101 0001 0100 0110 0010
op-code
26-bit target field from jump instruction
Shift Left two positions
32-Bit Jump Address:
0101 1011 0001 0100 0101 0001 1000 1000
High-order four bits from PC
PC:
0101 0110 0111 0110 0111 0010 1001 0100

J Instructions: Examples

TABLE 9-5: Unconditional Control Transfer Instructions in the MIPS ISA

Instruction	Assembly Code	Operation	Comments
jump	j addr	Go to addr * 4; i.e., PC = addr * 4	Target address = Imm offset * 4; addr is 26 bits
jump register	jr $reg	Go to $reg; i.e., PC = $reg	$reg contains 32-bit target address
jump and link	jal addr	return address = PC + 4; go to addr * 4	For procedure call, return address saved in the link register $31

Represent the machine code of the following MIPS instruction in 32 bit binary .Assume the opcode of LW,SW, ADDI ,SUB and BEQ is 0X23, **0X2B , 0X08 , 0X00 and 0X04**respectively and the function field of SUB is 0X22.

(I) LW R8, 16(R2)

(II) SW R3, 32(R1)

(III) ADDI R5,R6, 8

(IV) SUB R7, R8, R9

v) BEQ R10,R11,32