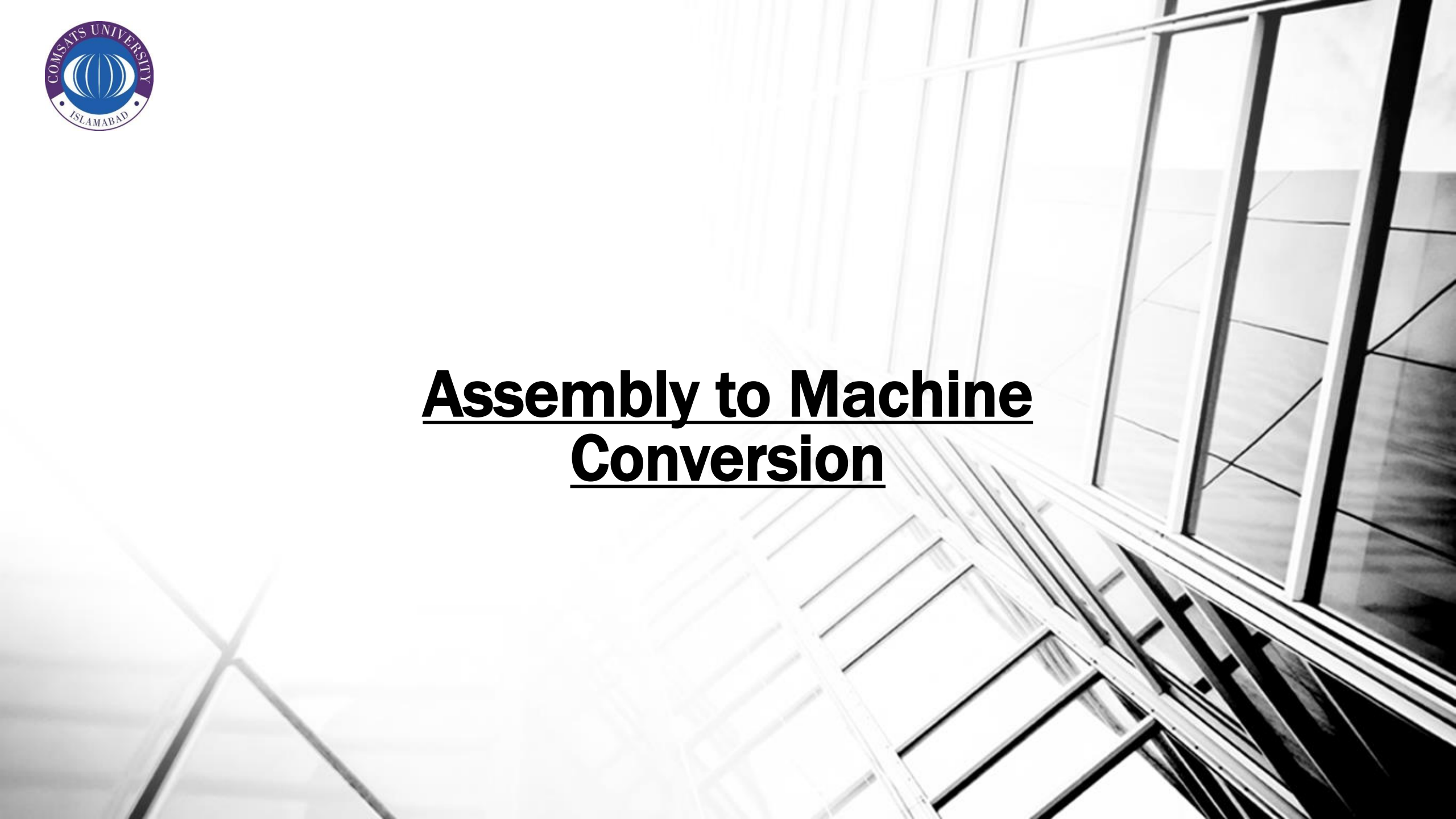
Computer Organization & Architecture



Assembly to Machine Conversion

EGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

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).

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.

MIPS instructions	Name	Format	Pseudo MIPS	Name	Format
add	add	R	move	move	R
subtract	sub	R	multiply	mult	R
add immediate	addi	I	multiply immediate	multi	I
load word	lw	I	load immediate	li	I
store word	sw	I	branch less than	blt	I
load half	lh	I	branch less than or equal	ble	I
load half unsigned	lhu	I			
store half	sh	I	branch greater than	bgt	I
load byte	lb	I	branch greater than or equal	bge	I
load byte unsigned	lbu	I			
store byte	sb	I			
load linked	ll	I			
store conditional	sc	I			
load upper immediate	lui	I			
and	and	R			
or	or	R			
nor	nor	R			
and immediate	andi	I			
or immediate	ori	I			
shift left logical	sll	R			
shift right logical	srl	R			
branch on equal	beq	I			
branch on not equal	bne	I			
set less than	slt	R			
set less than immediate	slti	I			
set less than immediate unsigned	sltiu	I			
jump	j	J			
jump register	jr	R			
jump and link	jal	J			

FIGURE 2.44 The MIPS instruction set covered so far, with the real MIPS instructions on the left and the pseudoinstructions on the right. Appendix A (Section A.10) describes the full MIPS architecture. Figure 2.1 shows more details of the MIPS architecture revealed in this chapter. The information given here is also found in Columns 1 and 2 of the MIPS Reference Data Card at the front of the book.

MIPS assembly language

Category	Instruction	Example		Meaning	Comments
Arithmetic	add	add	$s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub	$s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow detected
	add immediate	addi	$s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	addu	$s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	subu	$s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow undetected
	add immediate unsigned	addiu	$s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	mfc0	$s1,$epc	$s1 = $epc	Copy Exception PC + special regs
	multiply	mult	$s2,$s3	Hi, Lo = $s2 × $s3	64-bit signed product in Hi, Lo
	multiply unsigned	multu	$s2,$s3	Hi, Lo = $s2 × $s3	64-bit unsigned product in Hi, Lo
	divide	div	$s2,$s3	Lo = $s2 / $s3, Hi = $s2 mod $s3	Lo = quotient, Hi = remainder
	divide unsigned	divu	$s2,$s3	Lo = $s2 / $s3, Hi = $s2 mod $s3	Unsigned quotient and remainder
	move from Hi	mfhi	$s1	$s1 = Hi	Used to get copy of Hi
	move from Lo	mflo	$s1	$s1 = Lo	Used to get copy of Lo
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 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 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 conditional word	sc	$s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half atomic swap
	load upper immediate	lui	$s1,100	$s1 = 100 * 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	$s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND with constant
	OR immediate	ORi	$s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR 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
Condi-tional 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; two's complement
	set less than immediate	slti	$s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1=0	Compare < constant; two's complement
	set less than unsigned	sltu	$s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1=0	Compare less than; natural numbers
	set less than immediate unsigned	sltiu	$s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Uncondi-tional 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

FIGURE 3.12 MIPS core architecture. The memory and registers of the MIPS architecture are not included for space reasons, but this section added the Hi and Lo registers to support multiply and divide. MIPS machine language is listed in the MIPS Reference Data Card at the front of this book.

MIPS core instructions	Name	Format	MIPS arithmetic core	Name	Format
add	add	R	multiply	mult	R
add immediate	addi	I	multiply unsigned	multu	R
add unsigned	addu	R	divide	div	R
add immediate unsigned	addiu	I	divide unsigned	divu	R
subtract	sub	R	move from Hi	mfhi	R
subtract unsigned	subu	R	move from Lo	mflo	R
AND	AND	R	move from system control (EPC)	mfc0	R
AND immediate	ANDi	I	floating-point add single	add.s	R
OR	OR	R	floating-point add double	add.d	R
OR immediate	ORi	I	floating-point subtract single	sub.s	R
NOR	NOR	R	floating-point subtract double	sub.d	R
shift left logical	sll	R	floating-point multiply single	mul.s	R
shift right logical	srl	R	floating-point multiply double	mul.d	R
load upper immediate	lui	I	floating-point divide single	div.s	R
load word	lw	I	floating-point divide double	div.d	R
store word	sw	I	load word to floating-point single	lwc1	I
load halfword unsigned	lhu	I	store word to floating-point single	swc1	I
store halfword	sh	I	load word to floating-point double	ldc1	I
load byte unsigned	lbu	I	store word to floating-point double	sdc1	I
store byte	sb	I	branch on floating-point true	bc1t	I
load linked (*atomic update*)	ll	I	branch on floating-point false	bc1f	I
store cond. (*atomic update*)	sc	I	floating-point compare single	c.x.s	R
branch on equal	beq	I	(x = eq, neq, lt, le, gt, ge)		
branch on not equal	bne	I	floating-point compare double	c.x.d	R
jump	j	J	(x = eq, neq, lt, le, gt, ge)		
jump and link	jal	J			
jump register	jr	R			
set less than	slt	R			
set less than immediate	slti	I			
set less than unsigned	sltu	R			
set less than immediate unsigned	sltiu	I			

FIGURE 3.26 The MIPS instruction set. This book concentrates on the instructions in the left column. This information is also found in columns 1 and 2 of the MIPS Reference Data Card at the front of this book.

Remaining MIPS-32	Name	Format
exclusive or (rs ⊕ rt)	xor	R
exclusive or immediate	xori	I
shift right arithmetic	sra	R
shift left logical variable	sllv	R
shift right logical variable	srlv	R
shift right arithmetic variable	srav	R
move to Hi	mthi	R
move to Lo	mtlo	R
load halfword	lh	I
load byte	lb	I
load word left *(unaligned)*	lwl	I
load word right *(unaligned)*	lwr	I
store word left *(unaligned)*	swl	I
store word right *(unaligned)*	swr	I
load linked (*atomic update*)	ll	I
store cond. (*atomic update*)	sc	I
move if zero	movz	R
move if not zero	movn	R
multiply and add (S or *uns.*)	madd*s*	R
multiply and subtract (S or *uns.*)	msub*s*	I
branch on ≥ zero and link	bgezal	I
branch on < zero and link	bltzal	I
jump and link register	jalr	R
branch compare to zero	bxz	I
branch compare to zero likely	bxzl	I
(x = lt, le, gt, ge)		
branch compare reg likely	bxl	I
trap if compare reg	tx	R
trap if compare immediate	txi	I
(x = eq, neq, lt, le, gt, ge)		
return from exception	rfe	R
system call	syscall	I
break *(cause exception)*	break	I
move from FP to integer	mfc1	R
move to FP from integer	mtc1	R
FP move (*s* or *d*)	mov.*f*	R
FP move if zero (*s* or *d*)	movz.*f*	R
FP move if not zero (*s* or *d*)	movn.*f*	R
FP square root (*s* or *d*)	sqrt.*f*	R
FP absolute value (*s* or *d*)	abs.*f*	R
FP negate (*s* or *d*)	neg.*f*	R
FP convert (*w*, *s*, or *d*)	cvt.*f*.*f*	R
FP compare un (*s* or *d*)	c.xn.*f*	R

Pseudo MIPS	Name	Format
absolute value	abs	rd,rs
negate *(signed or uns.)*	neg*s*	rd,rs
rotate left	rol	rd,rs,rt
rotate right	ror	rd,rs,rt
multiply and don't check oflw *(signed or uns.)*	mul*s*	rd,rs,rt
multiply and check oflw *(signed or uns.)*	mulo*s*	rd,rs,rt
divide and check overflow	div	rd,rs,rt
divide and don't check overflow	divu	rd,rs,rt
remainder *(signed or uns.)*	rem*s*	rd,rs,rt
load immediate	li	rd,imm
load address	la	rd,addr
load double	ld	rd,addr
store double	sd	rd,addr
unaligned load word	ulw	rd,addr
unaligned store word	usw	rd,addr
unaligned load halfword *(signed or uns.)*	ulh*s*	rd,addr
unaligned store halfword	ush	rd,addr
branch	b	Label
branch on equal zero	beqz	rs,L
branch on compare *(signed or unsigned)*	bx*s*	rs,rt,L
(x = lt, le, gt, ge)		
set equal	seq	rd,rs,rt
set not equal	sne	rd,rs,rt
set on compare *(signed or unsigned)*	sx*s*	rd,rs,rt
(x = lt, le, gt, ge)		
load to floating point (*s* or *d*)	l.*f*	rd,addr
store from floating point (*s* or *d*)	s.*f*	rd,addr

FIGURE 3.27 Remaining MIPS-32 and Pseudo MIPS instruction sets. *f* means single (s) or double (d) precision floating-point instructions, and *s* means signed and unsigned (u) versions. MIPS-32 also has FP instructions for multiply and add/sub (madd.f/ msub.*f*), ceiling (ceil.*f*), truncate (trunc.*f*), round (round.*f*), and reciprocal (recip.*f*). The underscore represents the letter to include to represent that datatype.

MIPS Reference Data

①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate		
J	opcode	address				

ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
FR	opcode	fmt	ft	fs	fd	funct
FI	opcode	fmt	ft	immediate		

PSEUDOINSTRUCTION SET

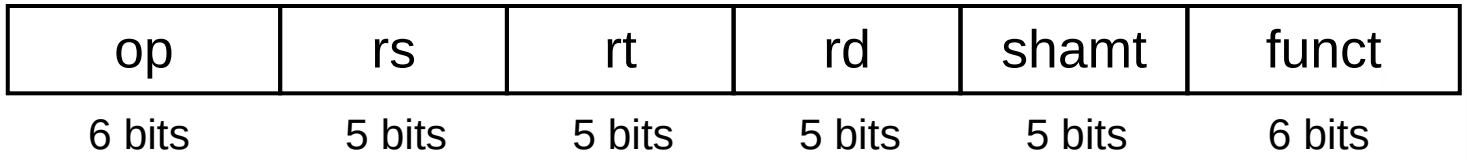
NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design,* 4th ed.

MIPS R-format Instructions



- 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)

R-format Example

op	rs	rt	rd	shamt	funct
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

$00000010001100100100000000100000_2 = 02324020_{16}$

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

R-format Example

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

```
and $t8, $s4, $s5
```

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

special	$s4	$s5	$t8	0	and
0	14_{16}	15_{16}	18_{16}	0	24_{16}
000000	10100	10101	11000	00000	100100

$0000\ 0010\ 1001\ 0101\ 1100\ 0000\ 0010\ 0100_2 = 0295C024_{16}$

R-format Example

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

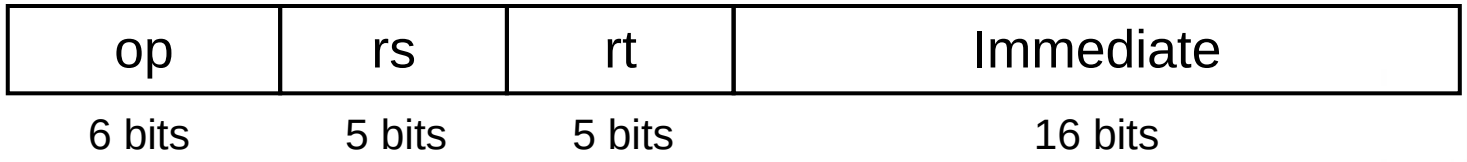
```
sll $t2, $s0, 4
```

special	0	$s0	$t2	4	0
0	0	16	10	4	0
000000	00000	10000	01000	00100	000000

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

$0000\ 0000\ 0001\ 0000\ 0100\ 0001\ 0000\ 0000_2 = 00104100_{16}$

MIPS I-format Instructions



- Instruction fields
  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - Immediate: Offset Address or Immediate value

I-format Example

op	rs	rt	Immediate
6 bits	5 bits	5 bits	16 bits

```
lw $t0, 20($s1)
```

special	$s1	$t0	20
23_{16}	11_{16}	8_{16}	14_{16}
100011	10001	01000	0000 0000 0001 0100

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

$1000\ 1101\ 0001\ 0001\ 0000\ 0000\ 0001\ 0100_2 = 8D110014_{16}$

I-format Example

op	rs	rt	Immediate
6 bits	5 bits	5 bits	16 bits

```
addi $s1, $s2, 20
```

special	$s2	$s1	20
8_{16}	12_{16}	11_{16}	14_{16}
001000	01000	10001	0000 0000 0001 0100

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

$0010\ 0001\ 0001\ 0001\ 0000\ 0000\ 0001\ 0100_2 = 21110014_{16}$

I-format Example

op	rs	rt	Immediate
6 bits	5 bits	5 bits	16 bits

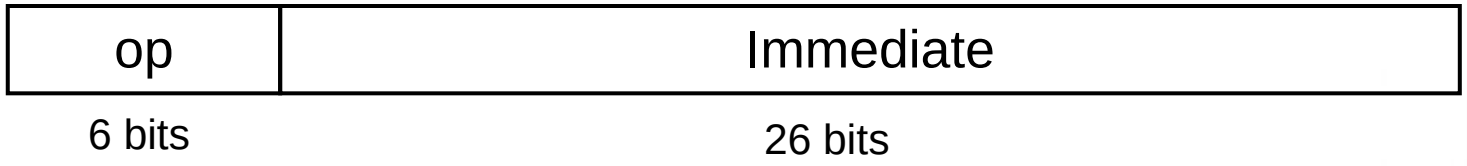
```
beq $t0,$s1, Exit_
```

special	$t0	$s1	Label Exit_
4_{16}	8_{16}	11_{16}	Let Address of Exit_ $=14_{16}$
000100	01000	10001	0000 0000 0001 0100

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

$0001\ 0001\ 0001\ 0001\ 0000\ 0000\ 0001\ 0100_2 = 11110014_{16}$

MIPS J-format Instructions



- Instruction fields
  - op: operation code (opcode)
  - Immediate: Offset Address

J-format Example

op	Immediate
6 bits	26 bits

```
j Fin
```

special	Address to jump
2_{16}	Address of Fin Label e-g: 8_{16}
000010	00 0000 0000 0000 0000 0000 1000

$$0000\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1000_{2} = 08000008_{16}$$

J-format Example

op	Immediate
6 bits	26 bits

```
jal Fin
```

special	Address to jump
3_{16}	Address of Fin Label e-g: 8_{16}
000011	00 0000 0000 0000 0000 0000 1000

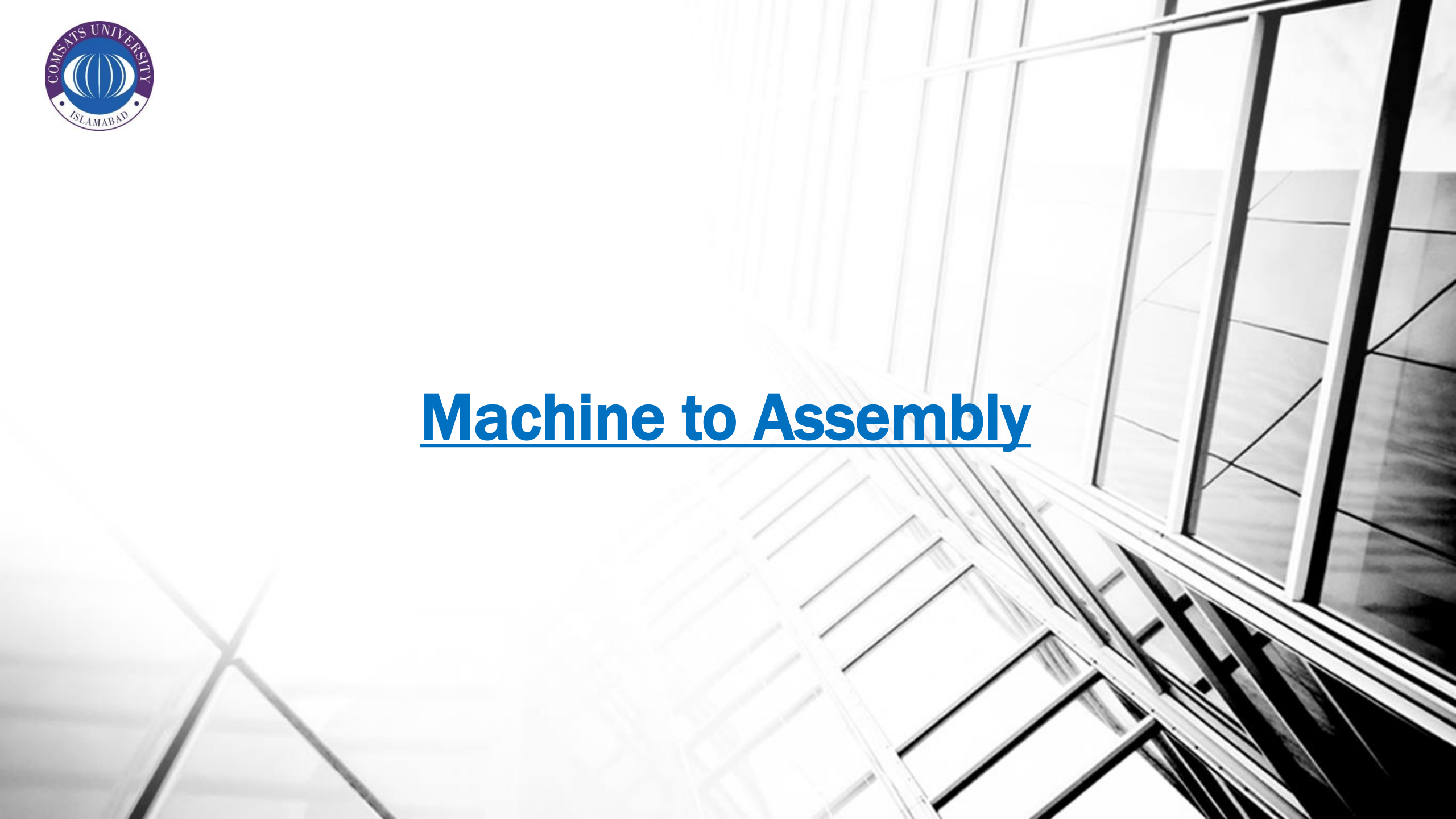
$0000\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000\ 1000_{2} = 0C000008_{16}$

Example (Adding all up)

Convert the following C-Code into MIPS assembly and later convert it into MIPS Machine instruction.

```
do
{
 g= g+ A[i];
 i = i + j;
}while(i != h);
```

g, h, i, j, A
$s1, $s2, $s3, $s4,$s5



Machine to Assembly

Examples

- 0x02324020 add $t0,$s1,$s2
- 0x8E680020 lw $t0, 32($s3)
- 0x02F3482A slt $t1,$s7,$s3