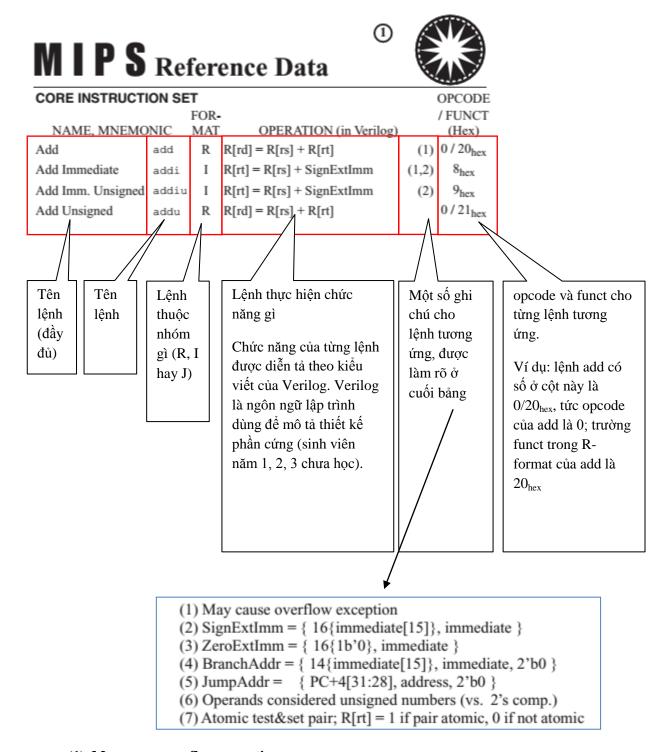
### Một số lệnh assembly MIPS cơ bản

MIDC				1			ARITHMETIC CO	ME INS	INU	CHON SET			(2	/	OPCO FMT
MIPS	Ref	er	ence Data		1				FOR-		OBEE	mio:		/	FUN
					- 4		NAME, MNEMO		MAT		OPERA			(4)	(He
CORE INSTRUCTI						OPCODE	Branch On FP True Branch On FP False			if(FPcond)P( if(!FPcond)P					11/8
NAME ANIEMO		FOR- MAT		.:1aa)		/ FUNCT	Divide	div		Lo=R[rs]/R[1					0//
NAME, MNEMO Add	add			mog)	(1)	(Hex) 0 / 20 <sub>hex</sub>	Divide Unsigned	divu		Lo=R[rs]/R[1	_				0//-
Add Immediate		I	R[rd] = R[rs] + R[rt]				FP Add Single	add.s		F[fd] = F[fs]				1	11/10
	addi		R[rt] = R[rs] + SignExtImm		(1,2)		FP Add	add.d	FR	{F[fd],F[fd+					11/11
Add Imm. Unsigned		I	R[rt] = R[rs] + SignExtImm	1	(2)		Double FP Compare Single			EDoord = /El		F[ft],F[f			11/10
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]			0 / 21 <sub>hex</sub>	FP Compare Single			FPcond = (F)					
And	and	R	R[rd] = R[rs] & R[rt]			0 / 24 <sub>hex</sub>	Double	c.x.d*	FR			[ft+1]})?			11/11
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImn	n	(3)	$c_{\text{hex}}$	* (x is eq, 1t,					2, 3c, or 3	Be)		/
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr		(4)	4 <sub>hex</sub>	FP Divide Single FP Divide	div.s	FR FR	F[fd] = F[fs] $\{F[fd], F[fd+$	1]} = {	F[fs],F[fs	s+1]} /		11/10 11/11
Branch On Not Equa	lbne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr		(4)	$5_{ m hex}$	Double FP Multiply Single	mul.s	FR	F[fd] = F[fs]	* F[ft]		t+1]}		11/10
Jump	j	J	PC=JumpAddr		(5)	2 <sub>hex</sub>	FP Multiply Double	mul.d	FR	{F[fd],F[fd+	1]} = {]	r[fs],F[fs	s+1]} * }+11\		11/11
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAdd	lr	(5)	$3_{\text{hex}}$	FP Subtract Single	sub.s	FR	F[fd]=F[fs] -		F[ft],F[f	115	1	11/10
Jump Register	jr	R	PC=R[rs]	Λ.		0 / 08 <sub>hex</sub>	FP Subtract	sub.d		{F[fd],F[fd+		F[fs],F[fs	s+1]} -		11/11
Load Byte Unsigned	1bu	Ι	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	}	(2)	24 <sub>hex</sub>	Double Load FP Single	lwcl	I	F[rt]=M[R[rs	s]+Sign	F[ft],F[f ExtImm]	t+1]}   (		31/
Load Halfword	1hu	I	R[rt]={16'b0,M[R[rs]	-	. ,	25.	Load FP	ldcl	Ι	F[rt]=M[R[rs				(2)	35/
Unsigned	Ind	1	+SignExtImm](15:0)	))}	(2)	25 <sub>hex</sub>	Double Move From Hi	mfhi	р	F[rt+1]=M[R R[rd] = Hi	qrsJ+Si	gnExtIm	m+4]		0 //
Load Linked	11	I	R[rt] = M[R[rs] + SignExtIm	nm]	(2,7)	$30_{ m hex}$	Move From Lo	mflo		R[rd] = Hi R[rd] = Lo					0 // 0 //
Load Upper Imm.	lui	I	$R[rt] = \{imm, 16'b0\}$			$f_{hex}$	Move From Contro		R	R[rd] = CR[r	rs]				10 /0
Load Word	lw	I	R[rt] = M[R[rs] + SignExtIm	nm]	(2)	*****	Multiply	mult	R	$\{Hi,Lo\} = R$					0//-
Nor	nor	R	$R[rd] = \sim (R[rs] \mid R[rt])$			0 / 27 <sub>hex</sub>	Multiply Unsigned		R	$\{Hi,Lo\} = R$			(	(6)	0//-
Or	or	R	R[rd] = R[rs]   R[rt]			0 / 25 <sub>hex</sub>	Shift Right Arith.	sra	R	R[rd] = R[rt] M[R[rs]+Sig			t1 (	(2)	0/
						o r = e nex	Store FP Single	SWC							
Or Immediate	ori	Ι	R[rt] = R[rs]   ZeroExtImm		(3)		Store FP Single Store FP	swcl	I					(2)	
Or Immediate Set Less Than	ori slt	I R			(3)	$d_{hex}$	_	swcl	I	M[R[rs]+Sig M[R[rs]+Sig	nExtIm	m] = F[r	t]; (	(2)	3d/
	slt		R[rd] = (R[rs] < R[rt]) ? 1 :	0		d <sub>hex</sub> 0 / 2a <sub>hex</sub>	Store FP Double	sdcl	Ι	M[R[rs]+Sig M[R[rs]+Sig	nExtIm nExtIm	m] = F[r	t]; (	(2)	
Set Less Than Set Less Than Imm.	slt slti	R I	R[rd] = (R[rs] < R[rt]) ? 1 : $R[rt] = (R[rs] < SignExtImn$	0 m)? 1 :		d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub>	Store FP Double FLOATING-POIN	sdc1	I RUC1	M[R[rs]+Sig M[R[rs]+Sig	nExtIm nExtIm	m] = F[r m+4] = 1	t]; ( F[rt+1]	(2)	3d/
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 :	0 m)? 1 : m)	0(2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub>	Store FP Double	sdc1	I RUC1	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig FION FORM	nExtIm nExtIm	m] = F[r m+4] = 1	t]; ( F[rt+1]	(2)	3d/
Set Less Than Imm. Set Less Than Imm.	slt slti sltiu	R I	R[rd] = (R[rs] < R[rt]) ? 1 : R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn	0 m)? 1 : m)	0(2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub>	Store FP Double  FLOATING-POIN  FR opcode	sdc1	I RUC1	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig FION FORM	nExtIm nExtIm ATS	m] = F[r m+4] = 1	t]; ( F[rt+1]	6 5	
Set Less Than Imm. Set Less Than Imm. Unsigned	slt slti sltiu	R I I	$\begin{aligned} R[rd] &= (R[rs] < R[rt]) ? 1 : \\ R[rt] &= (R[rs] < SignExtImn \\ R[rt] &= (R[rs] < SignExtImn \\ ? 1 : 0 \end{aligned}$	0 m)? 1 : m)	0(2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub>	Store FP Double  FLOATING-POIN  FR opcode  31	sdc1	I RUCT	M[R[rs]+Sig M[R[rs]+Sig TION FORM ft 1 20 16	nExtIm nExtIm ATS	m] = F[r m+4] = 1	t]; ( F[rt+1]	6 5	3d/
Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig	slt slti sltiu	R I I	$\begin{split} R[rd] &= (R[rs] < R[rt]) ? 1 : \\ R[rt] &= (R[rs] < SignExtImn \\ R[rt] &= (R[rs] < SignExtImn \\ ? 1 : 0 \\ R[rd] &= (R[rs] < R[rt]) ? 1 : \end{split}$	0 m)? 1 : m)	0(2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 2b <sub>hex</sub>	Store FP Double  FLOATING-POIN  FR opcode  31  FI opcode  31	sdc1  T INSTI 26 25 26 25	I PRUCT int	M[R[rs]+Sig M[R[rs]+Sig TION FORM ft 1 20 16	nExtIm nExtIm ATS fs	m] = F[r m+4] = 1	t]; ( F[rt+1]	6 5	3d/
Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical	slt slti sltiu .sltu slt	R I I R	$\begin{split} R[rd] &= (R[rs] < R[rt]) ? 1 : \\ R[rt] &= (R[rs] < SignExtImn \\ R[rt] &= (R[rs] < SignExtImn \\ ? 1 : 0 \\ R[rd] &= (R[rs] < R[rt]) ? 1 : \\ R[rd] &= R[rt] << shamt \end{split}$	0 m)? 1: m) 0	0(2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 2b <sub>hex</sub>	Store FP Double  FLOATING-POIN  FR opcode  31  FI opcode	sdc1  T INST  1	I PRUCT int	M[R[rs]+Sig M[R[rs]+Sig TION FORM ft 1 20 16	nExtIm nExtIm ATS fs 6 15	m] = F[r m+4] = 1 s   11 10 ir	fd fd mmedia	(2) 6 5 te	3d/
Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte	slti sltiu sltiu sltu sll srl sb	R I I R R I	$\begin{split} R[rd] &= (R[rs] < R[rt]) ? 1 :: \\ R[rt] &= (R[rs] < SignExtImn \\ R[rt] &= (R[rs] < SignExtImn ? 1 : 0 \\ R[rd] &= (R[rs] < R[rt]) ? 1 :: \\ R[rd] &= R[rt] << shamt \\ R[rd] &= R[rt] >>> shamt \\ M[R[rs] + SignExtImm] (7:0) \\ R[rt] (7:0) \\ M[R[rs] + SignExtImm] &= R[rt] << shamt \\ R[rs] + R[rt] >>> shamt \\ R[rt] &= R[rt] >> shamt \\ R[rt] &= R[rt] >>> shamt \\ R[rt] &= R[rt] >> shamt \\ R[rt] &= R[rt] >>> shamt \\ R[rt] &= R[rt] >> shamt \\ R[rt] &= R[rt] >>> shamt \\ R[rt] &= R[rt] >> shamt \\ R[rt] &= R[$	0 m)? 1: m) 0 ) = ) [rt];	0 (2) (2,6) (6)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 02 <sub>hex</sub> 28 <sub>hex</sub>	Store FP Double  FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU NA Branch Less TI Branch Greater	sdc1  T INSTE  26 25  CTION  ME  nan  Than	I RUCT	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig M[R]]	nExtIm nExtIm ATS fs 6 15 IC if(R[ if(R[	or   s	fd fd PERATI   PC =   PC =   PC =	(2) 6 5 te	fun
Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsigned Set Less Than Unsigned Shift Left Logical Shift Right Logical	slt slti sltiu sltu sltu sll	R I I R R	$\begin{split} R[rd] &= (R[rs] < R[rt]) ? 1 :: \\ R[rt] &= (R[rs] < SignExtImn \\ R[rt] &= (R[rs] < SignExtImn \\ ? 1 : 0 \\ R[rd] &= (R[rs] < R[rt]) ? 1 :: \\ R[rd] &= R[rt] << shamt \\ R[rd] &= R[rt] >>> shamt \\ M[R[rs] + SignExtImm] (7:0) \\ R[rt] (7:0) \\ M[R[rs] + SignExtImm] &= R[rt] = (atomic) ? 1 \\ \end{split}$	0 m)? 1: m) 0 ) = ) [rt]; 1:0	0(2)(2,6)(6)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 02 <sub>hex</sub> 28 <sub>hex</sub>	FLOATING-POIN  FR opcode  31  FI opcode  31  PSEUDOINSTRU  NA  Branch Less TI  Branch Greatet  Branch Less TI	sdc1  T INSTI  26 25  CTION  ME  nan  Than  nan or E	I RUCT	M[R[rs]+Sig M[R[rs]+Sig FION FORM  ft  1 20 16  MNEMONI blt bgt ble	nExtIm nExtIm ATS fs 6 15 IC if(R[ if(R[ if(R[	or o	fd  perati  perati  perati  perati  perati  perati	(2) 6 5 tte (ON Lab Lab = La	fun el el el
Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte	slti sltiu sltiu sltu sll srl sb	R I I R R I	R[rd] = (R[rs] < R[rt]) ? 1 : 1 R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0 R[rd] = (R[rs] < R[rt]) ? 1 : 1 R[rd] = R[rt] << shamt R[rd] = R[rt] >>> shamt M[R[rs] + SignExtImm](7:0) R[rt](7:0) M[R[rs] + SignExtImm] = R[rt] = (atomic) ? 1 M[R[rs] + SignExtImm](15:0	0 m)? 1: m) 0 = ) [rt]; 1:0	0 (2) (2,6) (6) (2) (2,7)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 02 <sub>hex</sub> 28 <sub>hex</sub>	Store FP Double  FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU NA Branch Less TI Branch Greater	sdc1  T INSTI  26 25  CTION  ME  nan  Than  nan or E  Than or	I RUCT	M[R[rs]+Sig M[R[rs]+Sig FION FORM  ft  1 20 16  MNEMONI blt bgt ble	nExtIm nExtIm ATS fs 6 15 IC if(R[if(R[if(R[if(R[if(R[if(R[if(R[if(R[	OF   OF 	fd  PERATI  PO = [] PC	(2) 6 5 tte (ON Lab Lab = La	fun el el el
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig Shift Left Logical Shift Right Logical Store Byte Store Conditional	slt sltiu sltiu sltu sltu sll srl sc sh	R I I R R I I	R[rd] = (R[rs] < R[rt]) ? 1 :: R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0 R[rd] = (R[rs] < R[rt]) ? 1 :: R[rd] = R[rt] << shamt R[rd] = R[rt] >>> shamt M[R[rs] + SignExtImm](7:0) R[rt](7:0) M[R[rs] + SignExtImm] = R[rt] = (atomic) ? 1 M[R[rs] + SignExtImm](15:0 R[rt](15:0)	0 m)? 1: m) 0 )= ) [rt]; 1:0 0) = 5:0)	0 (2) (2,6) (6) (2) (2,7) (2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 00 <sub>hex</sub> 28 <sub>hex</sub> 28 <sub>hex</sub> 29 <sub>hex</sub>	FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU NA Branch Less TI Branch Greater Branch Greater	sdc1  T INSTI  26 25  CTION  ME  nan  Than  nan or E  Than or	I RUCT	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig FION FORM  ft 1 20	nExtIm nExtIm ATS fs 6 15 IC if(R[if(R[if(R[rd]	or o	fd  PERATI  [] PC =	(2) 6 5 tte (ON Lab Lab = La	fun el el el
Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word	slt sltiu sltiu sltu sltu sll srl sc sc sh	R I I R R I I I I	R[rd] = (R[rs] < R[rt]) ? 1 :: R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0 R[rd] = (R[rs] < R[rt]) ? 1 :: R[rd] = R[rt] << shamt R[rd] = R[rt] >>> shamt M[R[rs] + SignExtImm] (7:0) R[rt] (7:0) M[R[rs] + SignExtImm] = R R[rt] = (atomic) ? 1 M[R[rs] + SignExtImm] (15:0 R[rt] (1: M[R[rs] + SignExtImm] = R	0 m)? 1: m) 0 )= ) [rt]; 1:0 0) = 5:0)	0 (2) (2,6) (6) (2) (2,7) (2) (2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 02b <sub>hex</sub> 0 / 00b <sub>hex</sub> 0 / 02b <sub>hex</sub> 28b <sub>hex</sub> 29b <sub>hex</sub>	FLOATING-POIN  FR opcode  31  FI opcode  31  PSEUDOINSTRU  NA  Branch Less TI  Branch Greater  Branch Less TI  Branch Greater  Load Immediat	sdc1  T INSTI  26 25  1 26 25  CTION  ME  man  Than man or E Than or	I I I I I I I I I I I I I I I I I I I	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig FION FORM  ft 1 20 16  MNEMONI blt bgt ble ble ble li move	nExtIm nExtIm ATS fs 6 15 IC if(R[if(R[if(R[rd] R[rd]	OF or or or or or or or or or or	fd  fd  PERATI  PPERATI  PPERATI  PPERATI  PPERATI  PPERATI  PPC =  pt]  pC =  pt]  pC =  pt]	(2) 6 5 tte (ON Lab Lab = La	fur el el el
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract	slti sltiu .sltu sltu sll srl sc sc sh sw sub	R I I R R I I I I R	R[rd] = (R[rs] < R[rt]) ? 1 :: R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0 R[rd] = (R[rs] < R[rt]) ? 1 :: R[rd] = R[rt] << shamt R[rd] = R[rt] >>> shamt M[R[rs] + SignExtImm](7:0) R[rt](7:0) M[R[rs] + SignExtImm] = R[rt] = (atomic) ? 1 M[R[rs] + SignExtImm] = R[rt] = (atomic) ? 1 M[R[rs] + SignExtImm] = R[rt](1: R[rd] = R[rs] - R[rt]	0 m)? 1: m) 0 )= ) [rt]; 1:0 0) = 5:0)	0 (2) (2,6) (6) (2) (2,7) (2) (2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 02 <sub>hex</sub> 28 <sub>hex</sub> 28 <sub>hex</sub> 29 <sub>hex</sub> 2b <sub>hex</sub> 0 / 22 <sub>hex</sub>	FLOATING-POIN  FR opcode  31  FI opcode  31  PSEUDOINSTRU  NA  Branch Less TI  Branch Greater  Load Immedian  Move  REGISTER NAMI	sdc1  T INST!  26 25  CTION  ME  nan or E  Than or the	I int 2 2 SET	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig FION FORM. If 1 20 It If 1 20 It MNEMONI blt bgt ble al bge 1 i move	nExtIm nExtIm ATS fs 6 15 IC if(R[if(R[if(R[rd] R[rd]	operation   Figure   Figure	fd  PERATI  []) PC =  []	(2) 6 6 5 tte (ON Labe Labe = La	fun el el bel bel
Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word	slt sltiu sltiu sltu sll srl sb sc sh sw sub	R I I R R I I I I R	R[rd] = (R[rs] < R[rt]) ? 1 :: R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 :: R[rd] = R[rt] << shamt R[rd] = R[rt] >>> shamt M[R[rs] + SignExtImm](7:0) M[R[rs] + SignExtImm] = R[rt] = (atomic) ? 1  M[R[rs] + SignExtImm](15:0 R[rt] = (atomic) ? 1 M[R[rs] + SignExtImm] = R[rt] =	0 m)? 1: m) 0 )= ) [rt]; 1:0 0) = 5:0)	0 (2) (2,6) (6) (2) (2,7) (2) (2)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 02b <sub>hex</sub> 0 / 00b <sub>hex</sub> 0 / 02b <sub>hex</sub> 28b <sub>hex</sub> 29b <sub>hex</sub>	FLOATING-POIN  FR opcode  31  FI opcode  31  PSEUDOINSTRU  NA  Branch Less TI  Branch Greatet Branch Greatet Load Immediat Move  REGISTER NAMI	sdc1  T INST!  26 25  CTION  ME  nan  Than  nan or E  Than or be  E, NUM  UMBER	I mt 2 mt 2 SET	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig TION FORMA ft 1 20 16 MNEMONI blt bgt ble al bge 1 i move USE, CALL	nExtIm nExtIm ATS fs 6 15 IC if(R[ if(R[ if(R[ rd(R[rd R[rd	operation   Figure   Figure	fd fd mmedia:  PERATI   PC =	(ON Laborate	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract	slt slti sltiu sltu sltu sll srl sc sh sw sub subu (l) May	R I I R R I I I I C C C C C C C C C C C	R[rd] = (R[rs] < R[rt]) ? 1 :: R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0 R[rd] = (R[rs] < R[rt]) ? 1 :: R[rd] = R[rt] << shamt R[rd] = R[rt] >>> shamt M[R[rs] + SignExtImm](7:0) R[rt](7:0) M[R[rs] + SignExtImm] = R[rt] = (atomic) ? 1 M[R[rs] + SignExtImm] = R[rt] = (atomic) ? 1 M[R[rs] + SignExtImm] = R[rt](1: R[rd] = R[rs] - R[rt]	0 m)? 1: m) 0 = ) [rt]; 1:0 0) = 5:0) [rt]	0 (2) (2,6) (6) (2) (2,7) (2) (2) (1)	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 00b <sub>hex</sub> 0 / 00b <sub>hex</sub> 0 / 00b <sub>hex</sub> 28b <sub>hex</sub> 28b <sub>hex</sub> 29b <sub>hex</sub> 2b <sub>hex</sub> 0 / 22b <sub>hex</sub> 0 / 23b <sub>hex</sub>	FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU NA  Branch Less TI Branch Greatet Branch Greatet Load Immedian Move  REGISTER NAMI	T INSTI 26 25 CTION ME an or E Than or E Than or E	I mint 2 2 mint 2 SET SET	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig MNEMONI blt blt blt ble	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	operation   Figure   Figure	fd f	(2) 6 5 tte (ON Lab Lab = La = La	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract	slti sltiu sltiu sltu sll srl sb sc sh sw sub (1) May (2) Sigr (3) Zerr (3) Zerr	R I I R R R I I I I Extline Ex	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] >>> shamt  M[R[rs] + SignExtImm] (7:0)  M[R[rs] + SignExtImm] = R[rt] (7:0)  M[R[rs] + SignExtImm] = R[rt] = (atomic) ? 1  M[R[rs] + SignExtImm] = R[rt] (1:0)  M[R[rs] + R[rt] = R[rt]  R[rd] = R[rs] - R[rt]  se overflow exception  mm = { 16{ immediate[15]}, mm = { 16{ 16 10 }, immediate	0	0 (2) (2,6) (6) (2) (2,7) (2) (2) (1) diate	d <sub>hex</sub> 0/2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0/2b <sub>hex</sub> 0/00 <sub>hex</sub> 0/02 <sub>hex</sub> 28 <sub>hex</sub> 38 <sub>hex</sub> 29 <sub>hex</sub> 20/22 <sub>hex</sub> 0/22 <sub>hex</sub>	FLOATING-POIN  FR opcode  31  FI opcode  31  PSEUDOINSTRU  NA  Branch Greater  Branch Greater  Load Immediat  Move  REGISTER NAMI  NAME NI  Szero  Sat	T INSTI 26 25 26 25 CTION ME man or E Than or E E, NUM UMBER 0	I mt 2 2 mt 2 SET  SET  The Asset	M[R[rs]+Sig M[rs]+Sig M[rs]+	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImn  ATS  fs 6 15  fi 6 15  if (R[ if(R[ if(R[ if(R[ R[rd R[rd R] rd	OF   OF 	fd  PERATI    PERATI   PERATI   PC =   PC =	(ON Laborate	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract	slt sltiu sltiu sltiu sltiu sltiu slli srl sb sc sh sw sub subu (1) May (2) Sigr (3) Zerc (4) Brai	R I I R R R I I I I ExtlinchA	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] >>> shamt  M[R[rs] + SignExtImm](7:0)  M[R[rs] + SignExtImm](7:0)  M[R[rs] + SignExtImm](15:0  R[rt] = (atomic) ? 1  M[R[rs] + SignExtImm] = R[rt] = (R[rt]) = R[rt] =	0 m)? 1: m) 0	0 (2) (2,6) (6) (2) (2,7) (2) (1) diate	d <sub>hex</sub> 0/2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0/2b <sub>hex</sub> 0/00 <sub>hex</sub> 0/02 <sub>hex</sub> 28 <sub>hex</sub> 38 <sub>hex</sub> 29 <sub>hex</sub> 20/22 <sub>hex</sub> 0/22 <sub>hex</sub>	FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU NA  Branch Less TI Branch Greatet Branch Greatet Load Immedian Move  REGISTER NAMI	T INSTI 26 25 CTION ME an or E Than or E Than or E	I I I I I I I I I I I I I I I I I I I	M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig M[R[rs]+Sig MNEMONI blt blt blt ble	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	OF   S   S   S   S   S   S   S   S   S	fd  PERATI    PERATI   PERATI   PC =   PC =	(2) 6 5 tte (ON Lab Lab = La = La	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract	slt slti sltiu sltiu sltu sll srl srl srl srl srl srl srl srl srl	R I I R R R I I I I ExtlinethAdopAdo	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] << shamt  M[R[rs] + SignExtImm] (7:0)  M[R[rs] + SignExtImm] = R[rt] (7:0)  M[R[rs] + SignExtImm] = R[rt] (1:0)  M[R[rs] + SignExtImm] = R[rt] (1:0)  M[R[rs] + SignExtImm] = R[rt] (1:0)  M[R[rs] + SignExtImm] = R[rt] = R[rt] = R[rt] (1:0)  M[R[rs] + SignExtImm] = R[rt] = R[rt] = R[rt] = R[rt] (1:0)  M[R[rs] + SignExtImm] = R[rt] = R[	0 m)? 1: m) 0	0 (2) (2,6) (6) (2) (2,7) (2) (1) diate diate, ) }	d <sub>hex</sub> 0/2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0/2b <sub>hex</sub> 0/00 <sub>hex</sub> 0/00 <sub>hex</sub> 0/02 <sub>hex</sub> 28 <sub>hex</sub> 28 <sub>hex</sub> 29 <sub>hex</sub> 2b <sub>hex</sub> 0/22 <sub>hex</sub> 0/23 <sub>hex</sub> }	FLOATING-POIN  FR opcode  31  FI opcode  31  PSEUDOINSTRU  NA  Branch Creater  Branch Greater  Load Immediat  Move  REGISTER NAMI  NAME NI  Szero  Sat	T INSTI 26 25 26 25 CTION ME man or E Than or E E, NUM UMBER 0	I I I I I I I I I I I I I I I I I I I	M[R[rs]+Sig M[rs]+Sig	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	OF   S   S   S   S   S   S   S   S   S	fd   fd   fd   fd   fd   fd   fd   fd	(ON Laborate	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract	slt slti sltiu sltiu sltu sll srl sc sh sw sub subu (1) May (2) Sigr (3) Zer (4) Brai (5) Jum (6) Ope	R I I R R R I I I I ExtinchAderand	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] >>> shamt  M[R[rs] + SignExtImm](7:0)  M[R[rs] + SignExtImm](7:0)  M[R[rs] + SignExtImm](15:0  R[rt] = (atomic) ? 1  M[R[rs] + SignExtImm] = R[rt] = (R[rt]) = R[rt] =	0 m)? 1: m) 0   =	0 (2) (2,6) (6) (2) (2,7) (2) (1) diate diate,	dhex 0/2ahex ahex bhex 0/2bhex 0/00hex 0/00hex 28hex 28hex 29hex 2bhex 0/22hex 0/22hex 0/23hex }	FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU NA Branch Less TI Branch Greater Branch Greater Load Immediat Move  REGISTER NAMI  NAME NI  \$zero \$at \$v0-\$v1	T INSTI 26 25 1 26 25 CTION ME nan or E Than or E Than or E Than or E Than or E	I I RUCT int 2 SET SET SET The Associated Argument and Ar	M[R[rs]+Sig M[rs]+Sig M[rs]+	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	OF   S   S   S   S   S   S   S   S   S	fd f	GON Laborate	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract	slt sltiu sltiu sltiu sltiu sltiu sltiu slli srl srl srb sc sh sw sub subu (1) May (2) Sigr (3) Zerc (4) Brai (5) Jum (6) Ope (7) Atoi	R I I R R R I I I I I A Cau Extl DExtl pAdd rand nic t	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] << shamt  M[R[rs] + SignExtImm] (7:0)  M[R[rs] + SignExtImm] = R[rt] (7:0)  M[R[rs] + SignExtImm] = R[rt] (1:0)  M[R[rs] + R[rt] = R[rt] (1:0)  R[rd] = R[rs] - R[rt]  Se overflow exception  mm = { 16{immediate[15]}, mm = { 16{immediate[15]}, rm = { 17{immediate[15]}, rm = { 17{i	0 m)? 1: m) 0   =	0 (2) (2,6) (6) (2) (2,7) (2) (1) diate diate,	dhex 0/2ahex ahex bhex 0/2bhex 0/00hex 0/00hex 28hex 28hex 29hex 2bhex 0/22hex 0/22hex 0/23hex }	Store FP Double  FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU  NA  Branch Less TI  Branch Greatet Branch Immediat Move  REGISTER NAMI  NAME NI  \$zero \$at  \$v0-\$v1  \$a0-\$a3 \$t0-\$t7 \$s0-\$s7	T INSTI 26 25 26 25 CTION ME nan or E Than or	I I I I I I I I I I I I I I I I I I I	M[R[rs]+Sig M[rs]+Sig M[rs]+Si	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	OF   S   S   S   S   S   S   S   S   S	fd f	(ON Laborate	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract Subtract Unsigned	slt slti sltiu sltiu sltu sltu sll srl srl srl srl srl srl srl srl srl	R I I R R R I I I I I R R R R R R R R R	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn  R[rt] = (R[rs] < SignExtImn  ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] << shamt  R[rd] = R[rt] >>> shamt  M[R[rs] + SignExtImm] (7:0)  M[R[rs] + SignExtImm] = R[rt] (atomic) ? 1  M[R[rs] + SignExtImm] = R[rt] (fr)  M[rt] = R[rs] - R[rt]  See overflow exception  mm = { 16 {immediate[15]}, mm = { 16 {immediate[15]}, rm = { 16 {immediate[	0 m)? 1: m) 0  [[rt]; 1:0 0)0)= 5:5:0) [[rt] [rt]  immed ss, 2*bd ers (vs. atomic	0 (2) (2,6) (6) (2) (2,7) (2) (1) diate diate,	d <sub>hex</sub> 0 / 2a <sub>hex</sub> a <sub>hex</sub> b <sub>hex</sub> 0 / 2b <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 00 <sub>hex</sub> 0 / 02 <sub>hex</sub> 28 <sub>hex</sub> 28 <sub>hex</sub> 29 <sub>hex</sub> 2b <sub>hex</sub> 0 / 22 <sub>hex</sub> 0 / 23 <sub>hex</sub> } 2'b0 } comp.) not atomic	FLOATING-POIN  FR opcode  31  FI opcode  31  PSEUDOINSTRU  NA  Branch Less TI  Branch Greater  Load Immedian  Move  REGISTER NAMI  NAME NI  \$zero \$at  \$v0-\$v1  \$a0-\$a3  \$t0-\$t7  \$s0-\$s7  \$t8-\$t9	T INSTI 26 25 1 26 25 CTION ME man or E Than or E Than or E Than or E 1 2-3 4-7 8-15 16-23 24-25	The Assa Value Arg Tem Save Tem	M[R[rs]+Sig M[rs]+Sig M[rs]+Si	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	OF   S   S   S   S   S   S   S   S   S	fd  peratification of the peratification of	(ON Laborate	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract Subtract Unsigned  BASIC INSTRUCT	slt sltiu sltiu sltiu sltiu sltiu sltiu slli srl srl srl srl sr sw sub subu (1) May (2) Sigr (3) Zerc (4) Brai (5) Jum (6) Ope (7) Atoi ION FOI	R I I R R R I I I I R R R r cau hExtl hchAdrand mic t RMA	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn  R[rt] = (R[rs] < SignExtImn  ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] << shamt  R[rd] = R[rt] >>> shamt  M[R[rs] + SignExtImm] (7:0)  M[R[rs] + SignExtImm] = R[rt] (atomic) ? 1  M[R[rs] + SignExtImm] = R[rt] (15:0)  M[R[rs] + SignExtImm] = R[rt] (15:0)  M[R[rs] + SignExtImm] = R[rt] (15:0)  M[R[rs] + SignExtImm] = R[rt] = R[rt] = R[rt] (15:0)  M[R[rs] + SignExtImm] = R[rt] = R	0 m)? 1: m) 0   =	0 (2) (2,6) (6) (2) (2,7) (2) (2) (1) diate diate, 0) } . 2's cc, 0 if	dhex 0/2ahex ahex bhex 0/2bhex 0/00hex 0/00hex 28hex 28hex 29hex 2bhex 0/22hex 0/22hex 0/23hex }	FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU  NA  Branch Less TI  Branch Greater Load Immedian Move  REGISTER NAMI  NAME NI  \$zero \$at \$v0-\$v1 \$a0-\$a3 \$t0-\$t7 \$s0-\$s7 \$t8-\$t9 \$k0-\$k1	T INSTI 26 25 1 26 25 CTION ME nam or E Than or E	The Assa Valuand Arg Tem Savv	M[R[rs]+Sig M[rs]+Sig M[	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	OF   S   S   S   S   S   S   S   S   S	ti; (t); (f); (f); (f); (f); (f); (f); (f); (f	(ON Laborate	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract Subtract Unsigned  BASIC INSTRUCT R  opcode 31	slt sltiu sltiu sltiu sltiu sltiu sltiu slli srl	R I I R R R I I I I R R R r cau hExtl bExtl pAddrand mic t RMA	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] << shamt  M[R[rs] + SignExtImm] (7:0)  M[R[rs] + SignExtImm] = R[rt] (7:0)  M[R[rs] + SignExtImm] = R[rt] (1:0)  M[R[rs] + R[rt] = R[rt]  M[R[rs] + R[rt] = R[rt]  se overflow exception  mm = { 16{immediate[15]},  mm = { 16{immediate[15]},  dr = { PC+4[31:28], addres  s considered unsigned numbe  est&set pair; R[rt] = 1 if pair:  ATS  Tt rd rd s  20 16 15 11 10	0 m)? 1: m) 0   =	0 (2) (2,6) (6) (2) (2,7) (2) (2) (1) diate 3. 2's c, 0 if	dhex 0 / 2ahex ahex bhex 0 / 00hex 0 / 00hex 0 / 00hex 28hex 28hex 29hex 2bhex 0 / 22hex 0 / 23hex } 2'b0 } comp.) not atomic	FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU NA  Branch Less TI Branch Greater Load Immediat Move  REGISTER NAMI  NAME NI  \$zero \$at \$v0-\$v1 \$a0-\$a3 \$t0-\$t7 \$s0-\$s7 \$t8-\$t9 \$k0-\$k1 \$gp	T INSTI 26 25 1 26 25 CTION ME nam or E Than or E	The Assavand Argman Sava Tem Ressavand Ressava	M[R[rs]+Sig M[rs]+Sig	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	OF   S   S   S   S   S   S   S   S   S	ti; (ft); (frt+1]  fd  mmedia  PERATI  I) PC = (ft)  II PC	(ON Lab. Lab. Lab. Lab. Lab. Lab. Lab. Lab.	fun el el bel bel
Set Less Than Set Less Than Imm. Set Less Than Imm. Unsigned Set Less Than Unsig. Shift Left Logical Shift Right Logical Store Byte Store Conditional Store Halfword Store Word Subtract Subtract Unsigned  BASIC INSTRUCT R opcode 31 opcode	slt sltiu sltiu sltiu sltiu sltiu sltiu slli srl srl srl srl sr sw sub subu (1) May (2) Sigr (3) Zerc (4) Brai (5) Jum (6) Ope (7) Atoi ION FOI	R I I R R R I I I I I R R R R R R R R R	R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rt] = (R[rs] < SignExtImn R[rt] = (R[rs] < SignExtImn ? 1 : 0  R[rd] = (R[rs] < R[rt]) ? 1 ::  R[rd] = R[rt] << shamt  R[rd] = R[rt] << shamt  M[R[rs] + SignExtImm] (7:0)  M[R[rs] + SignExtImm] = R[rt] (7:0)  M[R[rs] + SignExtImm] = R[rt] (1:0)  M[R[rs] + R[rt] = R[rt]  M[R[rs] + R[rt] = R[rt]  se overflow exception  mm = { 16{immediate[15]},  mm = { 16{immediate[15]},  dr = { PC+4[31:28], addres  s considered unsigned numbe  est&set pair; R[rt] = 1 if pair:  ATS  Tt rd rd s  20 16 15 11 10	0 m)? 1: m) 0  [[rt]; 1:0 0)0)= 5:5:0) [[rt] [rt]  immed ss, 2*bd ers (vs. atomic	0 (2) (2,6) (6) (2) (2,7) (2) (2) (1) diate 3. 2's c, 0 if	dhex 0 / 2ahex ahex bhex 0 / 00hex 0 / 00hex 0 / 00hex 28hex 28hex 29hex 2bhex 0 / 22hex 0 / 23hex } 2'b0 } comp.) not atomic	FLOATING-POIN  FR opcode 31  FI opcode 31  PSEUDOINSTRU  NA  Branch Less TI  Branch Greater Load Immedian Move  REGISTER NAMI  NAME NI  \$zero \$at \$v0-\$v1 \$a0-\$a3 \$t0-\$t7 \$s0-\$s7 \$t8-\$t9 \$k0-\$k1	T INSTI 26 25 1 26 25 CTION ME nam or E Than or E	I PRUCTION TO THE PROPERTY OF	M[R[rs]+Sig M[rs]+Sig M[	nExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtImnExtIm	OF   S   S   S   S   S   S   S   S   S	ti; (ft; (Frt+1])  fd  mmedia  PERATI  I) PC = (ft)  II PC	(ON Laborate	fun el el bel bel

Bảng 1. Tóm tắt các lệnh MIPS cơ bản (tham khảo [1])

### Các lệnh assembly MIPS trong tài liệu này sẽ được diễn tả theo từng hàng trong bảng 1



### (1) May cause overflow exception

Những lệnh có phần ghi chú (1) sẽ một thông báo lỗi, hay còn gọi là gây ra một ngoại lệ (exception) khi phép toán bị tràn (overflow)

### (2) SignExtImm = {16 {immediate[15]}, immediate}

Những lệnh có phần ghi chú (2) luôn chứa một số tức thời 16 bits (có dấu dạng bù 2), và số này được mở rộng thành số 32 bits theo kiểu mở rộng có dấu.

Viết theo cấu trúc của Verilog

16{immediate[15]}: là một chuỗi 16 bits; 16 bit này được tao ra giống y như bit thứ 15 của immediate

{16{immediate[15]}, immediate}: là chuỗi 32 bits, 16 bit thuộc nữa cao được tao ra giống như bit thứ 15 của immediate, và 16 bit thuộc nữa thấp chính là số tức thời Ví du:

- ⇒ Có thể hiểu đơn giản, nếu số tức thời là dương thì 16 bits của nữa cao thêm vào sẽ là 0, còn nếu số tức thời là âm, thì 16bits của nữa cao thêm vào sẽ là 1

### (3) $ZeroExtImm = \{16\{1b'0\}, immediate\}$

Những lệnh có phần ghi chú (3) luôn chứa một số tức thời 16 bits (có dấu dạng bù 2), và số này được mở rộng thành số 32 bits theo kiểu mở rộng Zero, tức không cần biết đây là âm hay dương, 16 bits của nữa cao thêm vào đều là 0.

Viết theo cấu trúc của Verilog

16{1b'0}: là một chuỗi 16 bits mà tất cả các bit đều là 0

{16{1b'0}, immediate}: là chuỗi 32 bits, 16 bit thuộc nữa cao là 0 và 16 bit thuộc nữa thấp chính là số tức thời

Ví du:

### (4) BranchAddr = {14{immediate[15]}, immediate, 2'b0}

sẽ được giải thích trong phần lệnh beq và bne

### (5) $JumpAddr = \{ PC + 4[31:28], address, 2'b0 \}$

sẽ được giải thích trong phần lệnh j và jal

### (6) Operations considered unsigned numbers (vs. 2'comp.)

Những lệnh có phần ghi chú (6) luôn làm việc trên số không dấu (unsigned)

### (7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

Trong bảng 1, chỉ có 2 lệnh *ll* và sc là có ghi chú (7)

⇒ 2 lệnh này liên quan đến một số lý thuyết không nằm trong phần giảng dạy lý thuyết, vì vậy bỏ qua 2 lệnh này

### Một số ghi chú:

### Ký hiệu số:

Ký hiệu 0x đầu được dùng để chỉ hệ 16

Ví dụ: 
$$0xffff = ffff_{hex} = ffff_{(16)}$$

• Số ghi bình thường sẽ được hiểu là đang trong hệ 10

### Thanh ghi:

- Bộ xử lý chứa 32 thanh ghi để hoạt động, mỗi thanh ghi 32 bits.
- Mỗi thanh ghi sẽ có tên gợi nhớ và số thứ tự tương ứng của nó. Bảng 2 mô tả số thứ tự và tên gợi nhớ của từng thanh ghi
- Như vậy, khi làm việc với thanh ghi có 2 vấn đề cần quan tâm: giá tri và địa chỉ
  - ✓ Giá trị là giá trị đang được chứa trong thanh ghi
  - ✓ Địa chỉ là chỉ số của thanh ghi trong tập 32 thanh ghi.

Ví dụ: Nếu nói thanh ghi \$t3 có giá trị là 5, hoặc thanh ghi \$t3 bằng 5, tức giá trị đang chứa trong \$t3 là 5 và chỉ số/địa chỉ của \$t3 là 11

### REGISTER NAME, NUMBER, USE, CALL CONVENTION PRESERVED ACROSS NUMBER NAME USE A CALL? 0 The Constant Value 0 N.A. \$zero Sat Assembler Temporary No Values for Function Results \$v0-\$v1 2-3 No and Expression Evaluation \$a0-\$a3 4-7 Arguments No 8-15 Temporaries \$t0-\$t7 No \$s0-\$s7 16-23 Saved Temporaries Yes \$t8-\$t9 24-25 Temporaries No \$k0-\$k1 Reserved for OS Kernel 26-27 No \$gp 28 Global Pointer Yes 29 Stack Pointer \$sp Yes 30 Frame Pointer \$fp Yes 31 Return Address Yes ghi (trích từ bảng 1) Bảng 2. Mô tả các the Chỉ số tương Thanh ghi nào "Yes" là Tên gọi Mục đích sử nhớ của các ứng của các dụng của từng những thanh ghi cần được lưu trữ lai khi thực hiện việc gọi thanh ghi thanh ghi thanh ghi một hàm con

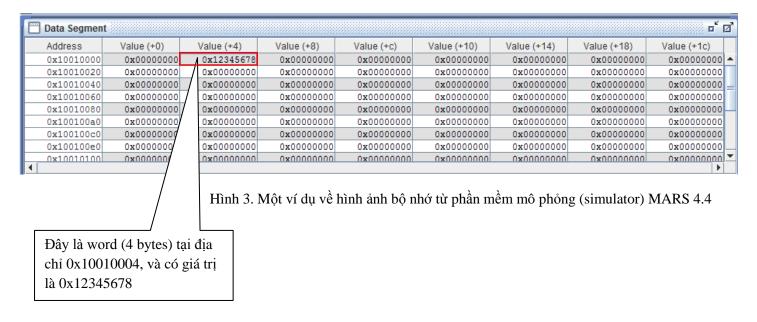
Lưu hành nội bộ

### Bộ nhớ:

Tương tự như thanh ghi, khi làm việc với bộ nhớ có 2 vấn đề cần quan tâm: giá trị và địa chỉ

- ✓ Giá trị là giá trị đang được chứa trong một từ nhớ (word), hoặc trong byte
- Địa chỉ địa chỉ được gán cho word hoặc byte đó.

Ví dụ:



# MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

### A. Xét các lệnh số học

NAME, MNEMONIC   MAT   OPERATION (in Verilog)   (a)   Add   add   add   R   R[rd] = R[rs] + R[rt]   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)   (1)	CORE INSTRUCTION	ON SET	)R-			OP(
Add Immediate addi I R[rt] = R[rs] + SignExtImm (1,2)  Add Imm. Unsigned addiu I R[rt] = R[rs] + SignExtImm (2)  Add Unsigned addiu R R[rd] = R[rs] + R[rt] 0.  And and R R[rd] = R[rs] & R[rt] 0.  And Immediate andi I R[rt] = R[rs] & ZeroExtImm (3)  Branch On Equal beq I PC=PC+4+BranchAddr (4)  Branch On Not Equal beq I R[rs]=R[rt] (5)  Jump And Link jal J R[31]=PC+8;PC=JumpAddr (5)  Jump And Link jal J R[31]=PC+8;PC=JumpAddr (5)  Jump And Link jal J R[31]=PC+8;PC=JumpAddr (5)  Jump Register jr R PC=R[rs] (6)  Jump And Link jal J R[rt]=R[rs] (7)  Load Byte Unsigned lbu I R[rt]=46*b0,M[R[rs] + SignExtImm](7:0)} (2)  Load Byte Unsigned lbu I R[rt]=46*b0,M[R[rs] + SignExtImm] (2)  Load Linked l1 R[rt]=M[R[rs]+SignExtImm] (2)  Load Unsigned lu I R[rt]=M[R[rs]+SignExtImm] (2)  Nor nor R R[rd]=R[rs] R[rt] (2)  Or or R R[rd]=R[rs] R[rt] (2)  Or or R R[rd]=R[rs] R[rt] (3)  Set Less Than slt R R[rd]=R[rs] R[rt] (6)  Or Immediate ori I R[rt]=R[rs] ZeroExtImm (3)  Set Less Than Imm. Slti I R[rt]=R[rs] ZeroExtImm (3)  Set Less Than Imm. Slti I R[rt]=R[rs] R[rt] (1) (2)  Set Less Than Imm. Slti I R[rd]=R[rs]<-R[rt])?1:0 (2)  Set Less Than Imm. Slti I R[rd]=R[rs]<-R[rt])?1:0 (2)  Set Less Than Imm. Slti I R[rd]=R[rs]<-R[rt] (7:0) (2,6)  Set Less Than Imm. Slti I R[rd]=R[rs]<-R[rt] (7:0) (2,6)  Set Less Than Imm. Slti I R[rd]=R[rs]<-R[rt] (7:0) (2,6)  Set Less Than Imm. Slti I R[rd]=R[rs]<-R[rt] (7:0) (2,6)  Set Less Than Imm. Slti I R[rd]=R[rs]-R[rt] (6)  Or Immediate ori I R[rt]=R[rs]-R[rt] (7:0) (6)  Set Less Than Imm. Slti I R[rd]=R[rs]-R[rt] (7:0) (6)  Set Less Than Imm. Slti I R[rd]=R[rs]-R[rt] (7:0) (2)  Set Less Than Imm. Slti	NAME, MNEMO					(F
Add Imm. Unsigned addiu I R[rt] = R[rs] + SignExtImm (2) Add Unsigned addiu R R[rd] = R[rs] + R[rt] (0) And and R R[rd] = R[rs] & R[rt] (1) And Immediate andi I R[rt] = R[rs] & ZeroExtImm (3) Branch On Equal beq I PCPC+4+BranchAddr (4) Branch On Not Equal beq I PCPC+4+BranchAddr (4) Branch On Not Equal beq I PCPC+4+BranchAddr (4) Jump J J PC=JumpAddr (5) Jump And Link jal J R[31] = PC+8; PC=JumpAddr (5) Jump Register jr R PC=R[rs] (0) Load Byte Unsigned 1bu I R[rt] = (24*b0,M[R[rs] + SignExtImm](7:0)) (2) Load Linked 1bu I R[rt] = H[rs] + SignExtImm](7:0) (2) Load Linked 1l R[rt] = H[R[rs] + SignExtImm] (2,7) Load Upper Imm. 1ui I R[rt] = H[R[rs] + SignExtImm] (2,7) Load Word 1w I R[rt] = M[R[rs] + SignExtImm] (2,7) Load Word 1w I R[rt] = M[R[rs] + R[rt] (0) Or or R R[rd] = (R[rs]   R[rt] (0) Or Immediate ori I R[rt] = R[rs]   ZeroExtImm (3) Set Less Than Imm. Stti I R[rt] = R[rs]   ZeroExtImm (3) Set Less Than Imm. Stti I R[rt] = R[rs]   SignExtImm) (1:0 (2) Set Less Than Unsig. sttu R R[rd] = (R[rs] < R[rt]) ? 1:0 (2,6) Set Less Than Unsig. sttu R R[rd] = R[rt] < SignExtImm] (7:0) = R[rt] (7:0) Shift Left Logical stl R R[rd] = R[rt] < SignExtImm] (5:0) = R[rt] (7:0)  Store Conditional sc I M[R[rs] + SignExtImm] (7:0) = R[rt] (7:0) (2,7) M[R[rs] + SignExtImm] = R[rt] (2) Store Conditional sc I M[R[rs] + SignExtImm] = R[rt] (2) Store Word sw I M[R[rs] + SignExtImm] = R[rt] (2) Subtract sub R R[rd] = R[rs] - R[rt] (1) 0 Subtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Subtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] (1) 0 Gubtract Unsigned subu R R[rd] = R[rs] - R[rt] = 1 if pair atomic, 0 if n	Add	add F	₹	R[rd] = R[rs] + R[rt]	(1)	0 /
Add Unsigned addu R R[rd] = R[rs] + R[rt] 0.4  And and R R[rd] = R[rs] & R[rt] 0.5  And Immediate and I R[rt] = R[rs] & R[rt] 0.5  Branch On Equal beq I iffR[rs] = R[rt] 1 pC = PC + 4 + Branch Addr (4)  Branch On Not Equal beq I iffR[rs] = R[rt] 1 pC = PC + 4 + Branch Addr (4)  Branch On Not Equal beq I iffR[rs] = R[rt] 1 pC = PC + 4 + Branch Addr (4)  Branch On Not Equal beq I iffR[rs] = R[rt] 1 pC = PC + 4 + Branch Addr (4)  Branch On Not Equal beq I iffR[rs] = R[rt] 1 pC = PC + 4 + Branch Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump Addr link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  And Jump And Link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump Addr link ja1 J R[31] = PC + 8; PC = Jump Addr (5)  Jump Addr link ja1 J R[31] =	Add Immediate	addi l	I	R[rt] = R[rs] + SignExtImm	(1,2)	8
And	Add Imm. Unsigned	addiu ]	I	R[rt] = R[rs] + SignExtImm	(2)	9
And Immediate andi I R[rt] = R[rs] & ZeroExtImm (3)  Branch On Equal beq I if(R[rs]==R[rt]) PC=PC+4+BranchAddr (4)  Branch On Not Equal bne I if(R[rs]==R[rt]) PC=PC+4+BranchAddr (4)  Jump j J PC=JumpAddr (5)  Jump And Link jal J R[31]=PC+8;PC=JumpAddr (5)  Jump Register jr R PC=R[rs] (0,  Load Byte Unsigned lbu I R[rt]={24'b0,M[R[rs] + SignExtImm](7:0)} (2)  Load Halfword Unsigned lbu I R[rt]={16'b0,M[R[rs] + SignExtImm](15:0)} (2)  Load Linked ll I R[rt] = M[R[rs]+SignExtImm] (2,7)  Load Upper Imm. lui I R[rt] = M[R[rs]+SignExtImm] (2,7)  Load Word lw I R[rt] = M[R[rs]+SignExtImm] (2)  Or nor R R[rd] = C(R[rs]   R[rt]) (0,  Or or R R[rd] = R[rs]   R[rt] (0,  Or Immediate ori I R[rt] = R[rs]   ZeroExtImm (3)  Set Less Than Imm. slt I R[rt] = (R[rs] < R[rt]); 1:0 (2)  Set Less Than Imm. slt I R[rt] = (R[rs] < R[rt]); 1:0 (2)  Set Less Than Unsigned slt R R[rd] = (R[rs] < R[rt]); 1:0 (6)  Set Less Than Unsigned slt R R[rd] = (R[rs] < R[rt]); 1:0 (2,6)  Set Less Than Unsigned slt R R[rd] = (R[rs] < R[rt]); 1:0 (6)  Set Less Than Unsigned slt R R[rd] = R[rt] < Shamt  Shift Right Logical sl R R[rd] = R[rt] >>> shamt  M[R[rs]+SignExtImm](7:0) = R[rt](7:0) (2)  Store Byte sb I M[R[rs]+SignExtImm] = R[rt];  Store Halfword sh I M[R[rs]+SignExtImm] = R[rt];  Store Word sw I M[R[rs]+SignExtImm] = R[rt];  Store Word sw I M[R[rs]+SignExtImm] = R[rt] (2)  Subtract sub R R[rd] = R[rs] - R[rt] (1)  (1) May cause overflow exception  (2) SignExtImm = {16{10'modaitate} } (1) May cause overflow exception  (2) SignExtImm = {16{10'modaitate} } (1) May cause overflow exception  (2) SignExtImm = {16{10'modaitate} } (1) May cause overflow exception  (2) SignExtImm = {16{10'modaitate} } (1) May cause overflow exception  (2) SignExtImm = {16{10'modaitate} } (1) May cause overflow exception  (2) SignExtImm = {16{10'modaitate} } (1) May cause overflow exception  (3) ZeroExtImm = {16{10'modaitate} } (1) May cause overflow exception  (4) BranchAddr = {14{10'modaitate} } (1) May cause overflow exception  (5) JumpAddr = {PC+431:28}, address, 2	Add Unsigned	addu F	₹	R[rd] = R[rs] + R[rt]		0 /
Branch On Equal   beq   I   if(R[rs]==R[rt])   PC=PC+4+BranchAddr   (4)	And	and F	3			0 /
PC=PC+4+BranchAddr	And Immediate	andi l	I		(3)	(
PC=PC+4+BranchAddr	Branch On Equal	beq l	Ι	PC=PC+4+BranchAddr	(4)	4
Jump And Link	Branch On Not Equal			PC=PC+4+BranchAddr	(4)	:
Load Byte Unsigned   Dun   First   F	Jump	j J	J	•	(5)	- 2
Load Byte Unsigned   1bu   I   R[rt]={24'b0,M[R[rs]   +SignExtImm](7:0)}   (2)	Jump And Link	jal J	J	R[31]=PC+8;PC=JumpAddr	(5)	3
Load Halfword Unsigned   Load Halfword Unsigned   Load Linked   Load Linked   Load Linked   Load Linked   Load Upper Imm.   Load Upper I	Jump Register	jr F	2			0 /
Unsigned	,	lbu l	I	+SignExtImm](7:0)}	(2)	2
Load Upper Imm.	Load Halfword Unsigned	lhu ]	I		(2)	2
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	Load Linked	11 ]	I	R[rt] = M[R[rs] + SignExtImm]	(2,7)	3
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	Load Upper Imm.	lui l	I	$R[rt] = \{imm, 16'b0\}$		
Or or R R[rd] = R[rs]   R[rt]	Load Word	lw ]	I	R[rt] = M[R[rs] + SignExtImm]	(2)	2
Or Immediate         ori         I         R[rt] = R[rs]   ZeroExtImm         (3)           Set Less Than         slt         R         R[rd] = (R[rs] < R[rt]) ? 1 : 0         0           Set Less Than Imm.         sltiu         I         R[rt] = (R[rs] < SignExtImm)? 1 : 0 (2)           Set Less Than Imm.         sltiu         I         R[rt] = (R[rs] < SignExtImm)? 1 : 0 (2,6)           Set Less Than Unsig.         sltu         R         R[rd] = (R[rs] < R[rt]) ? 1 : 0 (6)         0           Shift Left Logical         sll         R         R[rd] = R[rt] << shamt         0           Shift Right Logical         srl         R         R[rd] = R[rt] << shamt         0           Store Byte         sb         I         M[R[rs]+SignExtImm](7:0) = R[rt](7:0)         2           Store Byte         sb         I         M[R[rs]+SignExtImm](7:0) = R[rt](7:0)         2           Store Byte         sb         I         M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic)? 1: 0         (2,7)           Store Byte         sb         I         M[R[rs]+SignExtImm] = R[rt]; R[rt] = R[rt]         (2,7)           Store Halfword         sh         I         M[R[rs]+SignExtImm] = R[rt]; R[rt] = R[rt]         (2,7)           Store Word         sw         I         M[R[rs]+SignExt	Nor	nor F	3	$R[rd] = \sim (R[rs] \mid R[rt])$		0 /
Set Less Than   Sit   R   R[rd] = (R[rs] < R[rt]) ? 1 : 0   0	Or	or F	2	$R[rd] = R[rs] \mid R[rt]$		0 /
Set Less Than Imm.   siti   I   R[rt] = (R[rs] < SignExtImm)? 1:0 (2)	Or Immediate	ori l	I	R[rt] = R[rs]   ZeroExtImm	(3)	
Set Less Than Imm. Unsigned	Set Less Than	slt F	2	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 /
Unsigned SITIU I ? 1:0 (2,6)  Set Less Than Unsig. sltu R R[rd] = (R[rs] < R[rt]) ? 1:0 (6) 0.  Shift Left Logical S11 R R[rd] = R[rt] << shamt 0.  Shift Right Logical S11 R R[rd] = R[rt] >>> shamt 0.  Shift Right Logical S11 R R[rd] = R[rt] >>> shamt 0.  Store Byte Sb I M[R[rs] + SignExtImm] (7:0) = R[rt] (7:0) (2)  Store Conditional Sc I M[R[rs] + SignExtImm] = R[rt]; R[rt] = (atomic) ? 1:0 (2,7)  Store Halfword Sh I M[R[rs] + SignExtImm] = R[rt]; R[rt] = (atomic) ? 1:0 (2,7)  Store Word Sw I M[R[rs] + SignExtImm] = R[rt] (2)  Subtract Sub R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] (1) 0.  Subtract Unsigned Subu R R[rd] = R[rs] - R[rt] - R	Set Less Than Imm.	slti l	I	R[rt] = (R[rs] < SignExtImm)? 1	: 0(2)	
Shift Left Logical   s11		sltiu l	I		(2,6)	1
Shift Right Logical   sr1   R   R[rd] = R[rt] >>> shamt   0	Set Less Than Unsig.	sltu F	3	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	
Store Byte   Store Byte   Store Byte   Store Conditional   Store	Shift Left Logical	sll F	2	$R[rd] = R[rt] \le shamt$		0 /
Store Byte   SD   1   R[rt](7:0)   (2)	Shift Right Logical	srl F	2	R[rd] = R[rt] >>> shamt		0 /
Store Halfword   Shape   The state   Shape   The state   Store Halfword   Shape   The state   The state   Shape   The state   T	Store Byte	sb l	Ι		(2)	2
Martin   M	Store Conditional	sc 1	I			3
Store Word   Sw   I   M[R[rs]+SignExtImm] = R[rt]   (2)				M[R[rs]+SignExtImm](15:0) =		2
Subtract Unsigned						2
Comparison   Subu   R   R[rd] = R[rs] - R[rt]   Outline	Subtract	sub F	3	R[rd] = R[rs] - R[rt]	(1)	0 /
(1) May cause overflow exception (2) SignExtImm = { 16{immediate[15]}, immediate } (3) ZeroExtImm = { 16{Ib'0}, immediate } (4) BranchAddr = { 14{immediate[15]}, immediate, 2'b (5) JumpAddr = { PC+4[31:28], address, 2'b0 } (6) Operands considered unsigned numbers (vs. 2's con (7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not BASIC INSTRUCTION FORMATS  R	Subtract Unsigned	subu F	2	R[rd] = R[rs] - R[rt]		0 /
R         opcode         rs         rt         rd         shamt         fu           31         26 25         21 20         16 15         11 10         6 5           I         opcode         rs         rt         immediate	BASIC INSTRUCTI	(2) SignE: (3) ZeroE: (4) Brancl (5) JumpA (6) Operar (7) Atomic	xtIi hAdd Add nds	mm = { $16\{\text{immediate}[15]\}$ , imm mm = { $16\{1b^{\circ}0\}$ , immediate } ddr = { $14\{\text{immediate}[15]\}$ , immediate } dr = { $PC+4[31:28]$ , address, $2^{\circ}$ the seconsidered unsigned numbers (v. est&set pair; $R[rt] = 1$ if pair atom	ediate,	2'b
31 26 25 21 20 16 15 11 10 6 5 I opcode rs rt immediate		_			t	fu
-			21			
	I opcode	rs		rt immed	liate	

ARITHMETIC CORE INSTRU	JCTION SET (2) OPCO	DE
	/ FMT	/FT
FOR	,	
NAME, MNEMONIC MAT	(	
Branch On FP True bolt FI	if(FPcond)PC=PC+4+BranchAddr (4) 11/8/	
Branch On FP False belf FI	if(!FPcond)PC=PC+4+BranchAddr(4) 11/8/	
Divide div R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] 0//-	
Divide Unsigned divu R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] (6) 0//-	
FP Add Single add.s FR		/0
FP Add Double add.d FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}$	/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	$FPcond = (\{F[fs], F[fs+1]\} op  \{F[ft], F[ft+1]\}) ? 1 : 0$	//y
	==, <, 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	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}$	/3
FP Multiply Single mul.s FR	F[fd] = F[fs] * F[ft] 11/10/	/2
FP Multiply Double mul.d FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}$	/2
FP Subtract Single sub.s FR	F[fd]=F[fs] - F[ft] 11/10/	/1
FP Subtract Double sub.d FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}$	/1
Load FP Single 1wc1 I	F[rt]=M[R[rs]+SignExtImm] (2) 31//-	/
Load FP Double	F[rt]=M[R[rs]+SignExtImm]; (2) 35//-	/
Move From Hi mfhi R	R[rd] = Hi 0 //	<b>-</b> /10
Move From Lo mflo R	R[rd] = Lo 0 //	
Move From Control mfc0 R	R[rd] = CR[rs]    10 /0/	/0
Multiply mult R	$\{Hi, Lo\} = R[rs] * R[rt]$ 0//-	<b>-</b> /18
Multiply Unsigned multu R	$\{Hi, Lo\} = R[rs] * R[rt]$ (6) 0//-	
Shift Right Arith. sra R	R[rd] = R[rt] >> shamt 0//	
Store FP Single swc1 I	M[R[rs]+SignExtImm] = F[rt] (2) 39//-	/
Store FP Double sdc1 I	M[R[rs]+SignExtImm] = F[rt]; (2) $3d// M[R[rs]+SignExtImm+4] = F[rt+1]$	/

### FLOATING-POINT INSTRUCTION FORMATS

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

### 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	$if(R[rs] \le R[rt]) PC = Label$
Branch Greater Than or Equal	bge	$if(R[rs] \ge R[rt]) PC = Label$
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 <b>-</b> \$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0 <b>-</b> \$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

Computer Organization and Design, 4th ed.

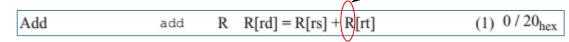
### Các lệnh số học:

- add, addi, addiu, addu
- sub, subu

R viết tắt của Register

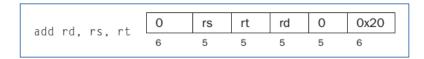
Ví dụ: R[rs] hiểu là giá trị của thanh ghi rs

### 1. Lệnh add



⇒ Lệnh này thuộc dạng R-format, có opcode là 0 và trường funct giá trị là 20<sub>hex</sub>

### Syntax (cú pháp): (tham khảo Appendix B của sách tham khảo [1])



 $\acute{\mathbf{Y}}$  nghĩa:  $\mathbf{R}[\mathbf{rd}] = \mathbf{R}[\mathbf{rs}] + \mathbf{R}[\mathbf{rt}]$ 

Thực hiện cộng giá trị thanh ghi rs với giá trị thanh ghi rt, tổng đưa vào thanh ghi rd

### Ví dụ:

add \$t0, \$t1, \$t2

Giả sử giá trị đang chứa trong thanh ghi \$t1 là 3, giá trị đang chứa trong thanh ghi \$t2 là 4

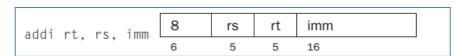
**Kết quả:** Sau khi lệnh add trên thực hiện, giá trị trong thanh ghi t0 là t0 (t0 là t0 (t0 là t0 là là là t0 là là là là là là là là là là

### 2. Lệnh addi

Add Immediate addi I 
$$R[rt] = R[rs] + SignExtImm$$
 (1,2)  $8_{hex}$ 

⇒ Lệnh này thuộc dạng I-format, có opcode 8<sub>hex</sub>

### Syntax (cú pháp):



 $\acute{Y}$  nghĩa: R[rt] = R[rs] + SignExtImm

Thực hiện công giá tri thanh ghi rs với số tức thời, kết quả đưa vào thanh ghi rt.

Lưu ý: Phạm vi cho số tức thời trong lệnh này là 16 bits. Số tức thời trước khi cộng với thanh ghi rs phải được **mở rông có dấu thành** (SignExtImm) thành số 32 bits.

### Ví dụ:

- a) addi \$t0, \$t1, 3
- b) addi \$t0, \$t1, -3
- c) addi \$t0, \$t1, 32768

Giả sử giá trị đang chứa trong thanh ghi \$t1 cho cả 3 câu đều là 4

### Kết quả:

a) Sau khi addi thực hiện xong, giá trị của \$t0 là 7

Quy trình lệnh thực hiện:

Giá trị trong t1 + SingExtImm của  $3_{(10)} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{(2)}$ 

b) Sau khi addi thực hiện xong, giá trị của \$t0 là 1

Quy trình lệnh thực hiện:

```
số tức thời là -3_{(10)} = 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 11111 \ 11111 \ 1111 \ 1111 \ 11111 \ 1111 \ 1111 \ 1111 \ 1111
```

Giá trị trong t1 + SingExtImm của  $3_{(10)} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$ 

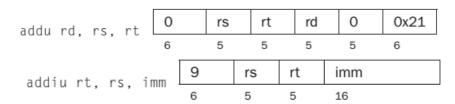
c) Lênh bi báo lỗi, do 32768 ra khỏi giới han của số 16 bits có dấu

### 3. Lệnh addiu và addu

- Addiu có cú pháp và thực hiện chức năng giống addi
- Addu có cú pháp và thực hiện chức năng giống add

Tuy nhiên, *addiu* và *addu* không xét kết quả có bị overflow hay không, trong khi đó *addi* và add sẽ báo khi overflow xuất hiên

### Syntax (cú pháp):



### Ví dụ:

a) addi \$t0, \$t1, 0x1



Giả sử thanh ghi t1 = 0x7fffffff

### Kết quả:

0x1 + 0x7fffffff = 0x80000000

Cộng một số dương với một số dương, kết quả ra một số âm => overflow

Khi lệnh addi trên thực hiện, một thông báo overflow sẽ xuất hiện

**b**) addiu \$t0, \$t1, 1

Giả sử thanh ghi t1 = 0x7fffffff

**Kết quả:** \$t0 = 0x80000000

Khi lệnh addi trên thực hiện, thanh ghi \$t0 vẫn nhận giá trị 0x8000000 và không có bất kì thông báo overflow nào xuất hiên.

### 4. Lệnh sub và subu

Lệnh sub có cú pháp tương tư như lệnh add, nhưng

- add thực hiện phép toán **cộng** 2 thanh ghi, kết quả lưu vào thanh ghi thứ 3
- trong khi đó, sub thực hiện phép toán trừ 2 thanh ghi, kết quả lưu vào thanh ghi thứ 3

Lệnh subu có cú pháp và chức năng giống như sub, nhưng

- subu không xét đến kết quả có bị overflow hay không
- sub có xét đến kết quả có bi overflow hay không; nếu bi overflow, sẽ có thông báo

### Syntax (cú pháp):

Lưu ý: không có lệnh subi (tức trừ một thanh ghi với một số tức thời) vì đã có lệnh addi và số tức thời trong addi có thể âm hoặc dương, nên subi không cần thiết.

### Tổng kết:

- ❖ add, addi, addiu, addu
- sub, subu

Nhìn lại côt ghi chú của 6 lệnh trên trong bảng 1:

 Chỉ có lệnh addi và addiu có ghi chú (2) → tức 2 lệnh này làm việc với số tức thời, và số tức thời 16 bits này được mở rông có dấu thành thành số 32 bits (có 'i' → làm việc với số tức thời)

Các lệnh không có "u" theo sau: add, addi, sub có thêm ghi chú (1); Các lệnh có "u" theo sau như: addiu, addu và subu không có ghi chú (1) → tức các lệnh không có "u" sẽ báo khi có overflow, còn các lệnh có "u" sẽ không báo khi có overflow

### **❖** Nhóm lệnh so sánh

slt / sltu

slti / sltiu

-----

### 5. Lệnh slt/sltu

Set Less Than slt R 
$$R[rd] = (R[rs] < R[rt]) ? 1 : 0$$
 0 /  $2a_{hex}$   
Set Less Than Unsig. sltu R  $R[rd] = (R[rs] < R[rt]) ? 1 : 0$  (6) 0 /  $2b_{hex}$ 

⇒ Hai lệnh này thuộc nhóm lệnh R-format, có opcode là 0 và funct trong slt là 2ahex, trong sltu là 2bhex

### **Syntax:**

### Ý nghĩa:

**slt:** 
$$R[rd] = (R[rs] < R[rt])? 1:0$$

Kiểm tra xem giá trị trong thanh ghi rs có nhỏ hơn thanh ghi rt hay không, nếu nhỏ hơn thì thanh ghi rd nhận giá trị là 1; ngược lại thanh ghi rd sẽ nhận giá trị 0

**sltu:** Ý nghĩa thực hiện giống như slt. Nhưng việc kiểm tra giá trị thanh ghi rs có nhỏ hơn thanh ghi rt hay không trong lệnh slt thực hiện trên số có dấu, còn trong sltu thực hiện trên số không dấu

### Ví du:

a. slt \$t0, \$t1, \$t2

Giả sử t1 = 0xfffffff1, t2 = 0x00000073

**Kết quả:** \$t0 = 1

Lệnh slt so sánh theo kiểu so sánh 2 số có dấu dạng bù 2

 $t2 = 0x00000073 = 01110011_{(2)} = 115_{(10)}$ 

 $V_{ay}$  \$t1 < \$t2 → giá trị trong thanh ghi \$t0 = 1

b. sltu \$t0, \$t1, \$t2

Giả sử \$t1 = 0xfffffff1, \$t2 = 0x00000073

Kết quả: \$t0 = 0

Lệnh slt so sánh theo kiểu so sánh 2 số không dấu

 $t1 = 0xfffffff1 = 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 0001_{(2)} = 4294967281_{(10)}$ 

 $t2 = 0x00000073 = 01110011_{(2)} = 115_{(10)}$ 

Vậy \$t1 > \$t2 → giá trị trong thanh ghi \$t0 = 0

### 6. Lệnh slti/sltiu

Set Less Than Imm. slti 
$$I = (R[rs] < SignExtImm)$$
? 1:0(2)  $a_{hex}$ 

Set Less Than Imm. Unsigned 
$$R[rt] = (R[rs] < SignExtImm)$$
  $? 1:0$   $b_{hex}$ 

⇒ Hai lệnh này thuộc nhóm lệnh I-format. Opcode của *slti* là a<sub>hex</sub>, opcode của *sltiu* là b<sub>hex</sub>

### **Syntax:**

### Ý nghĩa:

slti/sltiu: R[rd] = (R[rs] < SignExtImm)? 1:0

 $\acute{Y}$  nghĩa 2 lệnh này giống nhau là so sánh giá trị một thanh ghi với một số tức thời, nếu giá trị trong thanh ghi rs nhỏ hơn số tức thời thì thanh ghi rd nhận giá trị là 1; ngược lại thanh ghi rd sẽ nhận giá trị 0

Số tức thời cho phép trong lệnh này là số 16 bits. Trước khi so sánh với thanh ghi rs, số tức thời được mở rộng có dấu (SignExtImm) thành số 32 bits

slti khác sltiu là slti so sánh 2 giá trị theo kiểu có dấu dạng bù 2, trong khi đó sltiu so sánh theo kiểu số không dấu

### Ví dụ:

c. slti \$t0, \$t1, 0x73 Giả sử \$t1 = 0xfffffff1

### Kết quả: \$t0 = 1

Lệnh slt so sánh theo kiểu so sánh 2 số có dấu dạng bù 2

Số tức thời = 
$$0x73 = 01110011_{(2)}$$

 $SignExtImm(0x73) = 0000\ 0000\ 0000\ 0000\ 0000\ 0111\ 0011_{(2)} = 115_{(10)}$ 

Vậy \$t1 < \$t2 → giá trị trong thanh ghi \$t0 = 1

d. sltiu \$t0, \$t1, 0x83

Giå sử \$t1 = 0xfffffff1

### Kết quả: \$t0 = 0

Lệnh slt so sánh theo kiểu so sánh 2 số không dấu

$$t2 = 0x83 = 10000011_{(2)}$$

Vậy \$t1 > \$t2 → giá trị trong thanh ghi \$t0 = 0

FOR-

MAT

## 2. Fold bottom side (columns 3 and 4) together MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card

R

opcode

### B. Các lệnh logic

Į

### MIPS Reference Data

1

	IXC	CI	ence Data	V		
CORE INSTRUCTI	ON SE	Т			OPCODE	
		FOR-			/ FUNCT	
NAME, MNEMO		MAT	or (	(1)	(Hex)	
Add	add	R	R[rd] = R[rs] + R[rt]		0 / 20 <sub>hex</sub>	
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8 <sub>hex</sub>	
Add Imm. Unsigned		I	R[rt] = R[rs] + SignExtImm	(2)	9 <sub>hex</sub>	
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21 <sub>hex</sub>	
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24 <sub>hex</sub>	
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c <sub>hex</sub>	
Branch On Equal	beq	Ι	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4 <sub>hex</sub>	
Branch On Not Equa	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5 <sub>hex</sub>	
Jump	j	J	PC=JumpAddr	(5)	2 <sub>hex</sub>	
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	$3_{\text{hex}}$	
Jump Register	jr	R	PC=R[rs]		0 / 08 <sub>hex</sub>	
Load Byte Unsigned	1bu	I	$R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}$	(2)	24 <sub>hex</sub>	
Load Halfword Unsigned	1hu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25 <sub>hex</sub>	
Load Linked	11	I	R[rt] = M[R[rs] + SignExtImm]	(2,7)	30 <sub>hex</sub>	
Load Upper Imm.	lui	I	$R[rt] = \{imm, 16'b0\}$		$f_{hex}$	
Load Word	lw	I	R[rt] = M[R[rs] + SignExtImm]	(2)	23 <sub>hex</sub>	
Nor	nor	R	$R[rd] = \sim (R[rs] \mid R[rt])$		0 / 27 <sub>hex</sub>	
Or	or	R	$R[rd] = R[rs] \mid R[rt]$		0 / 25 <sub>hex</sub>	
Or Immediate	ori	I	R[rt] = R[rs]   ZeroExtImm	(3)	d <sub>hex</sub>	
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / 2a <sub>hex</sub>	
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1	: 0(2)	a <sub>hex</sub>	
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1:0	(2,6)	$b_{hex}$	
Set Less Than Unsig.	sltu	R	$R[rd] = (R[rs] \le R[rt]) ? 1 : 0$	(6)	0 / 2b <sub>hex</sub>	
Shift Left Logical	sll	R	$R[rd] = R[rt] \ll shamt$		0 / 00 <sub>hex</sub>	
Shift Right Logical	srl	R	R[rd] = R[rt] >>> shamt		0 / 02 <sub>hex</sub>	
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28 <sub>hex</sub>	
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; $R[rt] = (atomic)? 1:0$	(2,7)	$38_{\text{hex}}$	
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29 <sub>hex</sub>	
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	2b <sub>hex</sub>	
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22 <sub>hex</sub>	
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23 <sub>hex</sub>	
	(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						

rt

rd

address

ARITHMETIC CORE INSTRUCTION SET
---------------------------------

NAME, MNEMONIC

Branch On FP True bolt

Branch On FP False belf

Double

OPCODE / FMT /FT / FUNCT OPERATION (Hex) 11/8/1/--FI if(FPcond)PC=PC+4+BranchAddr (4) 11/8/0/--FI if(!FPcond)PC=PC+4+BranchAddr(4)

Dianen on it i and	2022		II( r cond)r c r c · + · Branch radir(+)	11,0,0,
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	add d	ED	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} +$	11/11//0
Double	auu.u	I IX	{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	ard*	FR	$FPcond = (\{F[fs], F[fs+1]\} op \{F[ft], F[ft+1]\}) ? 1 : 0$	11/11//v
Double	C.A.G	I IX	{F[ft],F[ft+1]})?1:0	11/11//y

\* (x is eq. 1t, or 1e) (op is ==, <, or <=) (y is 32, 3c, or 3e) FP Divide Single 11/10/--/3 div.s FR F[fd] = F[fs] / F[ft] $\label{eq:divd} _{\text{div.d }FR} \ \{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} \; / \;$ FP Divide 11/11/--/3 Double

FP Multiply Single mul.s FR F[fd] = F[fs] \* F[ft] 11/10/--/2  ${F[fd],F[fd+1]} = {F[fs],F[fs+1]} *$ FP Multiply mul.d FR 11/11/--/2 Double FP Subtract Single sub.s FR F[fd]=F[fs] - F[ft] 11/10/--/1  ${F[fd],F[fd+1]} = {F[fs],F[fs+1]}$ FP Subtract sub.d FR 11/11/--/1

{F[ft],F[ft+1]} (2) 31/--/--Load FP Single F[rt]=M[R[rs]+SignExtImm] Load FP F[rt]=M[R[rs]+SignExtImm]; (2)35/--/--/-ldcl Double F[rt+1]=M[R[rs]+SignExtImm+4] R[rd] = Hi R[rd] = Lo mfhi R 0 /--/--/10 Move From Hi 0 /--/--/12 Move From Lo mflo R

Move From Control mfc0 R R[rd] = CR[rs]10 /0/--/0  $\{Hi,Lo\} = R[rs] * R[rt]$  $\{Hi,Lo\} = R[rs] * R[rt]$ Multiply 0/--/--/18 mult (6) 0/--/--/19 Multiply Unsigned multu Shift Right Arith. R R[rd] = R[rt] >> shamt0/--/--/3 sra

Store FP Single swcl M[R[rs]+SignExtImm] = F[rt](2) 39/--/--Store FP M[R[rs]+SignExtImm] = F[rt];(2) 3d/--/--/-sdcl M[R[rs]+SignExtImm+4] = F[rt+1]Double

FR	opcode		fmt	ft		fs		fd	funct	
	31	26 25	21	20	16	15	11	10 6	5	0
FI	opcode		fmt	ft				immediat	е	
	21 '	26.25	21	20	16	16				0

### PSEUDOINSTRUCTION SET

FLOATING-POINT INSTRUCTION FORMATS

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	$if(R[rs] \le R[rt]) PC = Label$
Branch Greater Than or Equal	bge	$if(R[rs] \ge R[rt]) PC = Label$
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.
4	0		
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results	No
		and Expression Evaluation	
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0 <b>-</b> \$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.

shamt

immediate

funct

### Nhóm lệnh logic:

and, andi

nor

or, ori

sll, srl

\_\_\_\_\_

### 7. Lệnh and

And

and 
$$R R[rd] = R[rs] & R[rt]$$

0 / 24<sub>hex</sub>

⇒ Lệnh này thuộc dạng R-format, có opcode là 0 và trường funct là 24<sub>hex</sub>

### Syntax (cú pháp):

### $\acute{Y}$ nghĩa: R[rd] = R[rs] & R[rt]

Thực hiện and từng bit giá trị của thanh ghi rs và rt với nhau, kết quả lưu vào thanh ghi rd

### Ví dụ:

and \$t0, \$t1, \$t2

Giả sử giá trị đang chứa trong thanh ghi \$t1 là 0x12345678, giá trị đang chứa trong thanh ghi \$t2 là 0x0000000f thì

Kết quả: sau lệnh add trên, giá trị trong thanh ghi \$t0 là 0x00000008.

### 8. Lệnh andi

And Immediate

andi 
$$I R[rt] = R[rs] \& ZeroExtImm$$

(3) c<sub>hex</sub>

⇒ Lệnh này thuộc dạng I-format, có opcode là 0xc

### Syntax (cú pháp):

Ý nghĩa: R[rt] = R[rs] & ZeroExtImm

Lệnh này thực hiện *and* từng bit giá trị thanh ghi *rs* và một số tức thời. Số tức thời đang là số 16 bits, mở rộng thành số 32 bits theo kiểu ZeroExtImm, tức 16 bits nữa cao còn thiếu sẽ điền 0 vào. Sau đó thực hiện *and* từng bit giá trị của thanh ghi *rs* và số tức thời đã được mở rộng thành 32 bits với nhau, kết quả lưu vào thanh ghi *rd* 

### Ví dụ:

a) andi \$t0, \$t1, 0xfffff

Giả sử giá trị đang chứa trong thanh ghi \$t1 là 0x12345678.

Kết quả: sau lệnh trên, giá trị thanh ghi t0 = 0x00005678

Quy trình lệnh thực hiện:

Số tức thời:  $0xffff = 1111 1111 1111 1111_{(2)}$ 

 $ZeroExtImm(0xffff) = 0000\ 0000\ 0000\ 0000\ 1111\ 1111\ 1111\ 1111\ _{(2)}$ 

t0 = t1 & ZeroExtImm(0xffff) = 0x00005678

b) andi \$t0, \$t1, -3

Vấn đề đặt ra là imm ở đây có thể là số âm không?

- Một số simulator chấp nhận imm có thể là âm, ví dụ số -3 trên sẽ chuyển sang thành bù 2 của số 16 bits, sau đó mở rộng theo kiểu ZeroExtImm
- Một số simulator không chấp nhận imm có thể là âm, ví dụ số -3 trên đưa vào sẽ báo lỗi
- ⇒ Trong phạm vi môn học, chọn trường hợp thứ 2, không chấp nhập imm là âm

### 9. Các lệnh or, ori, nor

or và nor cách viết tương tự như and, nhưng thay vì thực hiện phép toán and, 2 lệnh này sẽ thực hiện phép toán or hoặc nor cho từng bit trong 2 thanh ghi, kết quả lưu vào thanh ghi thứ 3

ori tương tự như andi, thực hiện or một thanh ghi và một số tức thời 16 bits được mở rộng ZeroExtImm thành 32 bits

### 10.Lệnh sll/srl

\rm sll

Shift Left Logical sll R  $R[rd] = R[rt] \ll shamt$   $0 / 00_{hex}$ 

⇒ lệnh dịch trái số học, thuộc nhóm lệnh R, có opcode là 0 và funct 00<sub>hex</sub>

Shift Right Logical srl R R[rd] = R[rt] >>> shamt 0 / 02<sub>hex</sub>

 $\Rightarrow$  lệnh dịch phải số học, thuộc nhóm lệnh R, opcode là 0 và funct là  $02_{hex}$ 

### Syntax (cú pháp):

### Ý nghĩa:

sll: 
$$R[rd] = R[rt] \ll shamt$$

Thanh ghi rt dịch trái shamt bit và kết quả lưu vào thanh ghi rd ( '<< ' là ký hiệu của phép toán dịch trái)

$$srl: R[rd] = R[rt] >>> shamt$$

Thanh ghi rt dịch phải shamt bit và kết quả lưu vào thanh ghi rd ( '>>> ' là ký hiệu của phép toán dịch phải)

### Ví dụ:

a. sll \$t0, \$t1, 5

Giả sử thanh ghi \$11 đang chứa giá trị 0x12345678

Kết quả: sau lệnh trên, thanh ghi \$t0 = 0x468ACF00

Quy trình lệnh thực hiện: lệnh trên dịch trái 5 bit thanh ghi \$t1

 $t1 = 0x12345678 = 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000_{(2)}$ 

Dịch trái 5 bit  $t1 = 0100 0110 1000 1010 1100 1111 0000 0000_{(2)} = 0x468ACF00$ 

Vây kết quả thanh ghi t0 = 0x468ACF00

b. srl \$t0, \$t1, 5

Giả sử thanh ghi \$t1 đang chứa giá trị 0x12345678

Kết quả: sau lệnh trên, thanh ghi t0 = 0x91A2B3

Quy trình lênh thực hiện: lênh trên dịch phải 5 bit thanh ghi \$t1

 $t1 = 0x12345678 = 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000_{(2)}$ 

Dịch phải 5 bit  $t1 = 0000 0000 1001 0001 1010 0010 1011 0011_{(2)} = 0x91A2B3$ 

Vậy kết quả thanh ghi t0 = 0x91A2B3

### Tổng kết:

Các lệnh trong nhóm:

and, andi

nor

or, ori

sll, srl

Trong cột ghi chú ở bảng 1, chú ý chỉ có 2 lệnh *andi* và *ori* có ghi chú (3) – ghi chú 'zeroExtImm', tức các lệnh làm việc với số tức thời trong nhóm này khi mở rộng từ số tức thời 16 bits sang số 32 bits thì dùng zeroExtImm, không phải SignExtImm như nhóm lệnh số học.

### C. Nhóm lệnh Nhánh/Nhảy (Branch/Jump)

	<b>D</b> 0					1		
				ence D	ata			
CORE	NSTRUCT	ION S	FOR-					OPCOE / FUNC
NAM	E, MNEMO	NIC	MAT		RATION (in	Verilog)	,	(Hex)
Add		add	R	R[rd] = R[rs]	] + R[rt]	-	(1)	0 / 20 <sub>h</sub>
Add Imr	nediate	addi	I	R[rt] = R[rs]	+ SignExtI	mm	(1,2)	$8_{hex}$
Add Imr	n. Unsigned	addi	u I	R[rt] = R[rs]	+ SignExtI	mm	(2)	9 <sub>hex</sub>
Add Uns	signed	addu	R	R[rd] = R[rs]	] + R[rt]			0/21 <sub>h</sub>
And		and	R	R[rd] = R[rs]	] & R[rt]			0 / 24 <sub>h</sub>
And Imr	nediate	andi	I	R[rt] = R[rs]	& ZeroExtl	mm	(3)	c <sub>hex</sub>
Branch (	On Equal	beq	I		BranchAdd	г	(4)	4 <sub>hex</sub>
	On Not Equa		I		BranchAdd	r	(4)	5 <sub>hex</sub>
Jump		j	J	PC=JumpAd			(5)	2 <sub>hex</sub>
Jump An		jal	J	R[31]=PC+8	s;PC=JumpA	Addr A	(5)	3 <sub>hex</sub>
Jump Re	gister	jr	R	PC=R[rs]	MIDI1			0 / 08 <sub>b</sub>
Load By Load Ha	te Unsigned lfword		I	R[rt]={24'b( +Sign R[rt]={16'b(	nExtImm](7	:0)}	(2)	24 <sub>hex</sub>
	igned	1hu 11	I I		nExtImm](1		(2) (2,7)	25 <sub>he</sub> , 30 <sub>he</sub> ,
	per Imm.	lui	I	$R[rt] = \{imn$			(2,7)	fhex
Load Wo	-	1w	I	R[rt] = M[R]		tImml	(2)	23 <sub>he</sub>
Nor	,,,,	nor	R	$R[rd] = \sim (R$				0 / 27 <sub>h</sub>
Or		or	R	R[rd] = R[rs]				0 / 25 <sub>h</sub>
Or Imme	ediate	ori	I	R[rt] = R[rs]		nm	(3)	d <sub>hex</sub>
Set Less		slt	R	R[rd] = (R[r]	•			0 / 2a <sub>h</sub>
	Than Imm.	slti	I	R[rt] = (R[rs]				a <sub>hex</sub>
	Than Imm.	slti	u I	R[rt] = (R[rs]		mm)	(2,6)	b <sub>hex</sub>
Set Less	Than Unsig	.sltu	R	R[rd] = (R[rd])	$s$ ] $\leq$ R[rt]) ?	1:0		$0 / 2b_{h}$
Shift Let	t Logical	sll	R	R[rd] = R[rt]	<< shamt			0/00
Shift Rig	th Logical	srl	R	R[rd] = R[rt]	] >>> shamt			0 / 02 <sub>h</sub>
Store By	te	sb	I	M[R[rs]+Sig	nExtImm](7 R[rt](7		(2)	28 <sub>he</sub>
Store Co	nditional	sc	I		[] = (atomic)	? 1:0	(2,7)	38 <sub>he</sub>
Store Ha	lfword	sh	I	M[R[rs]+Sig		(15:0) = ](15:0)	(2)	29 <sub>he</sub>
Store Wo	ord	sw	I	M[R[rs]+Sig	_		(2)	2b <sub>he</sub>
Subtract		sub	R	R[rd] = R[rs]			(1)	0 / 22 <sub>h</sub>
Subtract	Unsigned	subu	R	R[rd] = R[rs]				0 / 23 <sub>h</sub>
		(2) S	ignExtI	se overflow e mm = { 16{ii mm = { 16{1	nmediate[15			
		(4) B	ranchA	ddr = { 14{in dr = { PC+4	nmediate[15	]}, immed		2'b0 }
		(7) A	tomic to	s considered u est&set pair;	_			
	NSTRUCT	ION F	ORMA	TS				
R	opcode		rs	rt	rd	shamt		funct
		26 25			15 11		6.5	
I	opcode	- 1	rs	l rt	l			
I	opcode 31	26 25	rs 21	20 16	15	immedia	ite	

ARITHMETIC CO	RE INS	TRU	ICTION SET (2)	OPCODE
			•	/ FMT /FT
		FOR-		/ FUNCT
NAME, MNEMO		MAT		(Hex)
Branch On FP True		FI	if(FPcond)PC=PC+4+BranchAddr (4)	
Branch On FP False	bclf	FI	if(!FPcond)PC=PC+4+BranchAddr(4	
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)	,
FP Add Single	add.s	FR	F[fd] = F[fs] + F[ft]	11/10//0
FP Add	add.d	FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} +$	11/11//0
Double FP Compare Single	c.r.s*	FR	$\{F[ft],F[ft+1]\}$ FPcond = $(F[fs] op F[ft])$ ? 1:0	11/10//v
FP Compare	c.x.d*	FR	$FPcond = (\{F[fs], F[fs+1]\} op$	11/11//y
Double	C.A.d	ГK	{F[ft],F[ft+1]})?1:0	11/11//y
			==, <, 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			{F[fd],F[fd+1]} = {F[fs],F[fs+1]} *	
Double	mul.d	FR	{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	sub.d	FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} -$	11/11//1
Double			{F[ft],F[ft+1]}	21/ / /
Load FP Single	lwcl	Ι	F[rt]=M[R[rs]+SignExtImm] (2)	
Load FP Double	ldcl	I	F[rt]=M[R[rs]+SignExtImm]; (2 F[rt+1]=M[R[rs]+SignExtImm+4]	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	swcl	I	M[R[rs]+SignExtImm] = F[rt] (2)	39//
Store FP			M[R[rs]+SignExtImm] = F[rt]; (2)	,
Double	sdcl	I	M[R[rs]+SignExtImm+4] = F[rt+1]	3d//

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt		ft	fs	fd	funct
	31 2	6 25	21 2	20 16	15 11	10 6	5 0
FI	opcode	fmt		ft		immediate	
	31 2	6 25	21 2	20 16	15		0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs] < R[rt]) PC = Label
Branch Greater Than		if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	$if(R[rs] \le R[rt]) PC = Label$
Branch Greater Than or Equal	bge	$if(R[rs] \ge R[rt]) PC = Label$
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 <b>-</b> \$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.

### Các lệnh xem xét:

beq, bne

j, jal, jr

Ngoài ra còn các lệnh thuộc bảng "PsedoInstruction Set"

blt/blte

bgt/bgte

Chú ý:

Các lệnh assembly có thể chia vào 2 nhóm: nhóm lệnh thật và nhóm lệnh giả

- Nhóm lệnh thật: là các lệnh mà thực chất processor sẽ chạy đúng lệnh đó
- Nhóm lệnh giả: là các lệnh mà khi thực thi thật sự thì lệnh này được chuyển thành một hoặc một số lệnh thuộc nhóm lệnh thật (nhóm lệnh này được đặt ra để thuận tiện cho người lập trình)

Các lệnh thuộc nhóm lệnh "PsedoInstruction Set" là những lệnh giả.

\_\_\_\_\_

### 11. Lệnh beq

Branch On Equal beq I 
$$if(R[rs]==R[rt])$$
  
PC=PC+4+BranchAddr (4)  $4_{hex}$ 

 $\Rightarrow$  lệnh này thuộc nhóm lệnh I-format, có opcode  $4_{hex}$ 

### Syntax (cú pháp):

Lênh beq có 2 cách viết cho vi trí 'label', 'label' có thể là một nhãn được viết bằng chữ, hoặc có thể là số

Ví dụ 2: 'label' viết bằng số:
Chạy đoạn chương trình sau: beq \$t1, \$t2, <b>2</b>
add \$s0, \$t3, \$t4
addi \$s1, \$t5, 3 or \$t1, \$t2, \$t3
sub \$t3, \$t4, \$t5
→ Trong ví dụ này, <b>số 2 thay cho label_A</b> Lệnh beq kiểm tra giá trị của \$t1 và \$t2, nếu:

- được thực hiện là lệnh "or \$t1, \$t2, \$t3". Sau khi "or" thực hiện xong thì luồng lệnh theo sau đó sẽ được thực hiện (ví dụ lệnh *sub* tiếp theo sau sẽ được thực hiện)
- 2 thanh ghi này không bằng nhau, thì lệnh tiếp theo được thực hiện là lệnh "add \$s0, \$t3, \$t4". Sau khi "add" thực hiện xong thì luồng lệnh theo sau đó sẽ được thực hiện (ví dụ chuỗi các lệnh addi, or, sub tiếp theo sau sẽ được thực hiện)
- 2 thanh ghi này bằng nhau, thì lệnh tiếp theo được thực hiện **là lệnh cách beq 2 lệnh**, tức là lệnh"or \$t1, \$t2, \$t3". Sau khi "or" thực hiện xong thì luồng lệnh theo sau đó sẽ được thực hiện (ví dụ lệnh *sub* tiếp theo sau sẽ được thực hiện)
- 2 thanh ghi này không bằng nhau, thì lệnh tiếp theo được thực hiện là lệnh "add \$s0, \$t3, \$t4". Sau khi "add" thực hiện xong thì luồng lệnh theo sau đó sẽ được thực hiện (ví dụ chuỗi các lệnh addi, or, sub tiếp theo sau sẽ được thực hiện)
- ⇒ Khi lập trình, ta có thể sử dụng một trong 2 cách như 2 ví dụ trên. Nhưng thực tế lệnh mà bộ xử lý hiểu là lệnh như ví dụ 2. Khi ta lập trình theo như ví dụ 1 thì lệnh cũng sẽ được chuyển về như ví dụ 2 để gởi cho bộ xử lý.

Như vậy beq chuẩn theo dạng:

### beq rs, rt, label/imm 4 rs rt Offset

Số tức thời label/imm này chính là số lệnh mà lệnh beq hiện tại cách lệnh sẽ nhảy tới bao nhiều, được lưu vào 16 bits của offset

### Ý nghĩa:

if(R[rs] = = R[rt]) PC = PC + 4 + BranchAddr

 $\Rightarrow$  Nếu giá trị thanh ghi rs bằng rt thì chương trình nhảy tới lệnh mà cách lệnh beq đang xét là imm lệnh, tức địa chỉ con trỏ/thanh ghi PC sẽ chuyển thành PC + 4 + imm\*4 (đối với trường hợp mỗi lệnh lưu trong một word 4 bytes) = PC + 4 + BranchAddr

BranchAddr = imm \* 4 (đối với trường hợp mỗi lênh lưu trong một word 4 bytes)

### 12. Lệnh bne:

Cách viết tương tự như beq, nhưng ý nghĩa trái ngược:

- beq: kiểm tra nếu 2 thanh ghi **bằng nhau** thì nhảy đến lệnh mong muốn
- bne: kiểm tra nếu 2 thanh ghi **không bằng nhau** thì nhảy đến lệnh mong muốn

### 13.Lệnh bge/bgt/ble/blt

Ngoài ra, còn một số lệnh so sánh và nhảy khác (trong bảng psedoInstruction Set)

bge \$t1, \$t2, label → Nhảy tới label thực hiện lệnh nếu thanh ghi \$t1 >= \$t2

bgt \$t1, \$t2, label → Nhảy tới label thực hiện lệnh nếu thanh ghi \$t1 > \$t2

ble \$t1, \$t2, label → Nhảy tới label thực hiện lệnh nếu thanh ghi \$t1 <= \$t2

blt \$t1, \$t2, label → Nhảy tới label thực hiện lệnh nếu thanh ghi \$t1 < \$t2

### 14. Lệnh j – lệnh nhảy không điều kiện

Jump j J PC=JumpAddr (5) 2<sub>hex</sub>

 $\Rightarrow$  Lệnh thuộc nhóm lệnh J-format, có opcode  $2_{hex}$ 

### Syntax (cú pháp):

### Ví dụ:

Chạy đoạn lệnh sau trên MARS 4.4:

```
beq $t1, $t2, label

j label

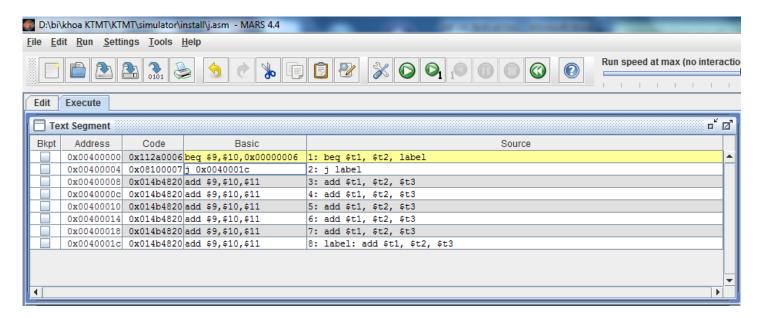
add $t1, $t2, $t3

label: add $t1, $t2, $t3
```

Kết quả: sau khi tới lệnh "j label", chương trình sẽ nhảy đến lệnh add cuối cùng sẽ để thực tiếp

### Quá trình thực hiện lệnh:

Khi biên dịch đoạn lệnh trên trong MARS 4.4, ta được hình như sau:



Cột Source là cột chứa các lệnh từ chương trình mà ta viết, cột này có thể chứa một số lệnh giả (pseudo-code); Cột basic là cột chứa các lệnh mà thực sự processor sẽ chuẩn bị chạy.

Lệnh "j label" khi thật sự chạy sẽ chuyển thành "j 0x0040001c", số này được tạo ra như thế nào?

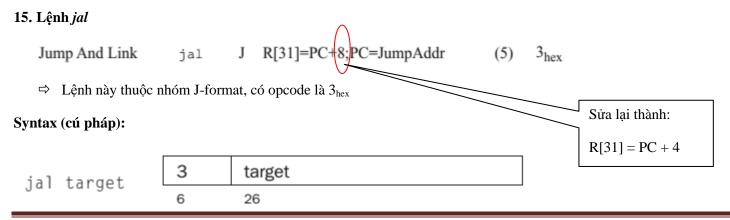
➡ Lệnh "j label" cách lệnh chứa nhãn 'label" 5 lệnh, vì vậy lệnh này cách lệnh cần nhảy tới 5\*4 = 20 byte
 Địa chỉ của lênh cần nhảy tới = PC + 4 + số byte cách lệnh sẽ nhảy tới

PC của lệnh "j label" = 0x00400004 số byte cách lênh sẽ nhảy tới = 20

Vậy Địa chỉ của lênh cần nhảy tới =  $0x00400004 + 4_{10} + 20_{10}$ = 0x0040001C

### Chú ý: Lệnh này chỉ cho nhảy trong phạm vi 256 $MB = 2^{28}$ byte

Lưu ý lệnh này khác với beq/bne là target trong syntax phải là nhãn của lệnh cần nhảy tới, không thể gán địa chỉ trực tiếp của lệnh cần nhảy tới, tức không thể gán một số vào đây được. Địa chỉ của lệnh cần nhảy tới sẽ do compiler tính toán và gởi cho processor trước khi thực hiện.



### Ý nghĩa:

$$R[31] = PC + 4$$
;  $PC = JumpAddr$ 

⇒ Lệnh này thực hiện việc nhảy giống y như lệnh *j*; nhưng địa chỉ của lệnh ngay sau lệnh *jal* được lưu vào thanh ghi 31 (thanh ghi *ra*) trước khi nhảy

Lệnh này nhằm phục vụ cho việc gọi một hàm con. Theo quy tắc, sau khi hàm con được gọi và thực hiện xong sẽ quay trở về chương trình chính. Do đó việc lưu lại địa chỉ của lệnh ngay sau *jal* vào *ra* nhằm lưu lại địa chỉ quay về này

### Ví dụ:

### Chạy đoạn lệnh sau trên MARS 4.4:

```
jal label
add $t1, $t2, $t3
label: add $t1, $t2, $t3
```

### Khi biên dịch:

Bkpt	Address	Code	Basic	Source	
	0x00400000	0x0c100006	jal 0x00400018	1: jal label	-
	0x00400004	0x014b4820	add \$9,\$10,\$11	2: add \$t1, \$t2, \$t3	
	0x00400008	0x014b4820	add \$9,\$10,\$11	3: add \$t1, \$t2, \$t3	
	0x0040000c	0x014b4820	add \$9,\$10,\$11	4: add \$t1, \$t2, \$t3	┚
	0x00400010	0x014b4820	add \$9,\$10,\$11	5: add \$t1, \$t2, \$t3	
	0x00400014	0x014b4820	add \$9,\$10,\$11	6: add \$t1, \$t2, \$t3	
	0x00400018	0x014b4820	add \$9,\$10,\$11	7: label: add \$t1, \$t2, \$t3	П

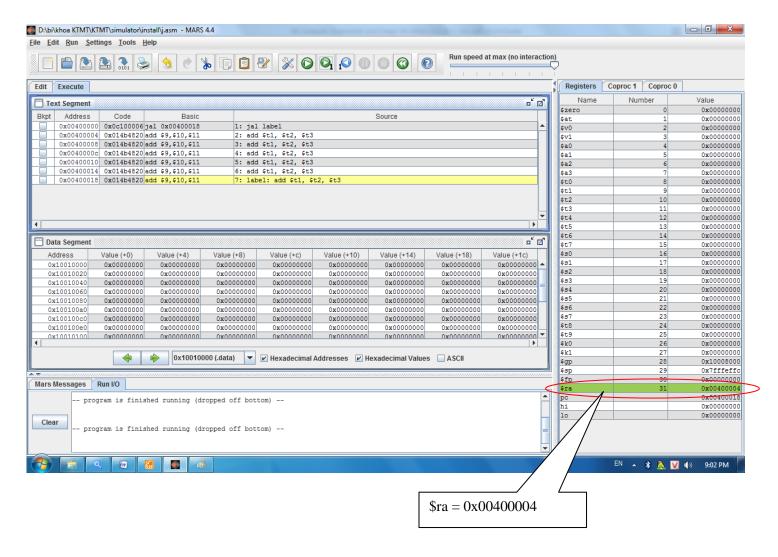
### Ta thấy:

"jal label" được chuyển thành "jal 0x0040018" (giá trị 'target' trong format lệnh lúc này sẽ bằng 0x0040018) trước khi gởi cho processor

Địa chỉ của lệnh này hiện tại là 0x00400000

### Khi chương trình chạy:

- PC = 0x00400000
- Đầu tiên, lệnh theo sau jal được lưu lại vào thanh ghi  $ra \rightarrow ra = PC + 4 = 0x00400004$
- Sau đó lệnh sẽ nhảy đến lệnh thứ 7, tức PC đang bằng = 0x00400000 sẽ chuyển thành PC= target/JumpAddr = 0x0040018



### 16. Lệnh *jr*

Jump Register jr R PC=R[rs]  $0/08_{hex}$ 

➡ Lệnh thuộc nhóm lệnh R (nhưng khá đặc biệt - chỉ quan tâm vùng thanh ghi rs), có opcode 0 và funct 08<sub>hex</sub>
Syntax (cú pháp):



 $\acute{\mathbf{Y}}$  nghĩa:  $\mathbf{PC} = \mathbf{R}[\mathbf{rs}]$ 

⇒ Nhảy tới lệnh có địa chỉ đang chứa trong thanh ghi rs

### Ví dụ:

### Chạy đoạn chương trình sau trong MARS

```
jr $s1
add $t1, $t2, $t3
addi $t1, $t2, 3
or $t1, $t2, $t3
xor $t1, $t2, $t3
addi $t1, $t2, 1
sub $t1, $t2, $t3
```

### Khi chương trình được biên dịch:

Bkpt	Address	Code	Basic	Source	
	0x00400000	0x02200008	jr \$17	1: jr \$s1	•
	0x00400004	0x014b4820	add \$9,\$10,\$11	2: add \$t1, \$t2, \$t3	Г
	0x00400008	0x21490003	addi \$9,\$10,0x00000003	3: addi \$t1, \$t2, 3	
	0x0040000c	0x014b4825	or \$9,\$10,\$11	4: or \$t1, \$t2, \$t3	
	0x00400010	0x014b4826	xor \$9,\$10,\$11	5: xor \$t1, \$t2, \$t3	
	0x00400014	0x21490001	addi \$9,\$10,0x00000001	6: addi \$t1, \$t2, 1	
	0x00400018	0x014b4822	sub \$9,\$10,\$11	7: sub \$t1, \$t2, \$t3	

Giả sử lúc này giá trị trong thanh ghi \$s1 = 0x0040000c, lệnh or sẽ được thực hiện ngay sau jr khi chương trình chạy.

Giả sử lúc này giá trị trong thanh ghi \$s1 = 0x00400018, lệnh sub sẽ được thực hiện ngay sau jr khi chương trình chạy.

### Tổng kết

### Nhóm lệnh:

beg, bne

j, jal, jr

Xét nhóm 4 lệnh: beq, bne, j, jal có cấu trúc như ví dụ sau:

beq/bne \$t1, \$t2, số\_16\_bits/label

j/jal label

### Đứng trên phương diện người lập trình, ta chỉ cần quan tâm:

Lệnh mà *beq/bne* nhảy tới có thể được đưa vào bằng cách gán "*label*" hoặc dùng "*số\_16\_bits*", "*số\_16\_bits*" trong *beq/bne* là số lệnh cách lệnh sẽ nhảy tới bao nhiều.

Lệnh mà j/jal nhảy tới chỉ có thể được đưa vào bằng cách gán "label"

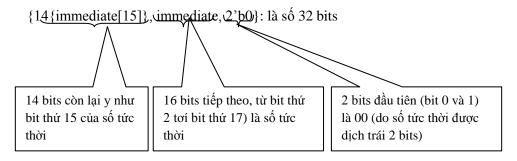
Đứng trên phương diện thiết kế processor, như thế nào processor tính toán ra địa chỉ của lệnh tiếp theo cần nhảy tới dựa vào các label hoặc các số\_16\_bits này?

Nhìn lại bảng 1 ta thấy beq/bne có ghi chú (4), còn nhóm j/jal có ghi chú (5), trong khi jr không có ghi chú gì cả:

- (4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 } (5) JumpAddr = { PC+4[31:28], address, 2'b0 }
  - ✓ Trong cột ý nghĩa của lệnh *beq/bne* ta thấy: PC = PC + 4 + BranchAddr
  - ✓ Trong cột ý nghĩa của lệnh j/jal ta thấy: PC = JumpAddr

Như đã trình bày trong phần trước, lệnh cần nhảy tới trong beq/bne có thể đưa vào là "*label*" hoặc số tức thời 16 bits - chỉ lệnh sẽ nhảy đến cách lệnh hiện tại bao nhiều lệnh. Nếu người lập trình đưa vào bằng "*label*", thì compiler sẽ tự tính toán ra số lệnh sẽ nhảy tới cách lệnh hiện tại bao nhiều. Tóm lại:

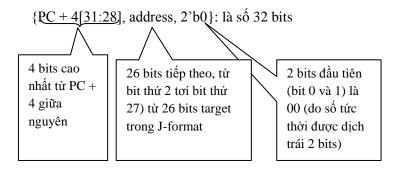
số tức thời trong lệnh *beq/bne* là số lệnh cách lệnh cần nhảy tới bao nhiều, nên BranchAddr được tính bằng cách lấy số tức thời 16 bits này nhân 4, tức dịch trái 2 bits rồi mở rộng theo kiểu có dấu thành số 32 bits, sau đó được cộng với PC + 4. Cách viết trong ghi chú (4) tương tự ý nghĩa này



Trong khi đó đối với lệnh *j/jal*, người lập trình đưa vào nhãn của địa chỉ cần nhảy tới, compiler sẽ tự tính toán ra lệnh cần nhảy tới cách lệnh hiện tại bao nhiều lệnh. Địa chỉ của lệnh cần nhảy tới (JumpAddr) sẽ được tính bằng cách lấy số lệnh này nhân 4 (dịch trái 2 bits) và cộng với PC + 4. Tuy nhiên, vì lệnh này chỉ cho phép nhảy trong phạm vi 256MB = 2<sup>28</sup> bytes, tức 4 bit cao nhất trong PC + 4 không đổi.

JumpAdd =  $PC + 4 + s\delta$  lệnh cách lệnh sẽ nhảy tới \* 4

Cách viết trong ghi chú (4) tương tự ý nghĩa này



### 2. Fold bottom side (columns 3 and 4) together MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card

### D. Nhóm lệnh memory-instruction

NAME, MNEMO Add Add Immediate Add Imm. Unsigned Add Unsigned And And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link Jump Register	NIC add addiu addiu and andi beq	FOR- MAT R I I R R I I	OPERATION (in Verilog)  R[rd] = R[rs] + R[rt]  R[rt] = R[rs] + SignExtImm  R[rt] = R[rs] + SignExtImm  R[rd] = R[rs] + R[rt]  R[rd] = R[rs] & R[rt]  R[rt] = R[rs] & ZeroExtImm  if(R[rs]==R[rt])  PC=PC+4+BranchAddr		OPCODI / FUNCI (Hex) 0 / 20 <sub>hes</sub> 8 <sub>hex</sub> 9 <sub>hex</sub> 0 / 21 <sub>hes</sub> 0 / 24 <sub>hes</sub>
NAME, MNEMO Add Add Immediate Add Imm. Unsigned Add Unsigned And And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link	NIC add addiu addiu addu and andi beq [bne j jal	FOR- MAT R I I R R I I	OPERATION (in Verilog)  R[rd] = R[rs] + R[rt]  R[rt] = R[rs] + SignExtImm  R[rt] = R[rs] + SignExtImm  R[rd] = R[rs] + R[rt]  R[rd] = R[rs] & R[rt]  R[rt] = R[rs] & ZeroExtImm  if(R[rs]==R[rt])  PC=PC+4+BranchAddr	(1) (1,2) (2)	/ FUNC (Hex) 0 / 20 <sub>hex</sub> 8 <sub>hex</sub> 9 <sub>hex</sub> 0 / 21 <sub>hex</sub> 0 / 24 <sub>hex</sub>
Add Add Immediate Add Imm. Unsigned Add Unsigned And And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link	addi addiu addu and andi beq bne j jal	MAT R I I R R I I I	OPERATION (in Verilog)  R[rd] = R[rs] + R[rt]  R[rt] = R[rs] + SignExtImm  R[rt] = R[rs] + SignExtImm  R[rd] = R[rs] + R[rt]  R[rd] = R[rs] & R[rt]  R[rt] = R[rs] & ZeroExtImm  if(R[rs]==R[rt])  PC=PC+4+BranchAddr	(1,2)	(Hex) 0/20 <sub>hex</sub> 8 <sub>hex</sub> 9 <sub>hex</sub> 0/21 <sub>hex</sub> 0/24 <sub>hex</sub>
Add Add Immediate Add Imm. Unsigned Add Unsigned And And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link	addi addiu addu and andi beq bne j jal	R I I R R I I	R[rd] = R[rs] + R[rt] R[rt] = R[rs] + SignExtImm R[rt] = R[rs] + SignExtImm R[rd] = R[rs] + R[rt] R[rd] = R[rs] & R[rt] R[rt] = R[rs] & ZeroExtImm if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(1,2)	0 / 20 <sub>hex</sub> 8 <sub>hex</sub> 9 <sub>hex</sub> 0 / 21 <sub>hex</sub> 0 / 24 <sub>hex</sub>
Add Immediate Add Imm. Unsigned Add Unsigned And And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link	addi addiu addu and andi beq bne j	I I R R I I	R[rt] = R[rs] + SignExtImm $R[rt] = R[rs] + SignExtImm$ $R[rd] = R[rs] + R[rt]$ $R[rd] = R[rs] & R[rt]$ $R[rt] = R[rs] & ZeroExtImm$ $if(R[rs] = R[rt])$ $PC = PC + 4 + BranchAddr$	(1,2)	8 <sub>hex</sub> 9 <sub>hex</sub> 0 / 21 <sub>hex</sub> 0 / 24 <sub>hex</sub>
Add Imm. Unsigned Add Unsigned And And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link	addiu addu and andi beq bne j	I R R I I	R[rt] = R[rs] + SignExtImm $R[rd] = R[rs] + R[rt]$ $R[rd] = R[rs] & R[rt]$ $R[rt] = R[rs] & ZeroExtImm$ $if(R[rs] = R[rt])$ $PC = PC + 4 + BranchAddr$	(2)	9 <sub>hex</sub> 0 / 21 <sub>he</sub> 0 / 24 <sub>he</sub>
Add Unsigned And And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link	addu and andi beq bne j	R R I I	R[rd] = R[rs] + R[rt] $R[rd] = R[rs] & R[rt]$ $R[rt] = R[rs] & ZeroExtImm$ $if(R[rs] = R[rt])$ $PC = PC + 4 + BranchAddr$		0 / 21 <sub>he</sub> 0 / 24 <sub>he</sub>
And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link	andi beq bne j jal	R I I	R[rd] = R[rs] & R[rt] R[rt] = R[rs] & ZeroExtImm if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(3)	0 / 24 <sub>he</sub>
And Immediate Branch On Equal Branch On Not Equal Jump Jump And Link	andi beq  bne  j	I I	R[rt] = R[rs] & ZeroExtImm if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(3)	
Branch On Equal Branch On Not Equal Jump Jump And Link	beq  bne  j	I I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(5)	"nex
Jump Jump And Link	j jal			(4)	4 <sub>hex</sub>
Jump And Link	jal		if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5 <sub>hex</sub>
-	-	J	PC=JumpAddr	(5)	2 <sub>hex</sub>
Jump Register	ir	J	R[31]=PC+8;PC=JumpAddr	(5)	$3_{\text{hex}}$
	J -	R	PC=R[rs]	- /	0 / 08 <sub>he</sub>
Load Byte Unsigned	1bu	Ι	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24 <sub>hex</sub>
Load Halfword Unsigned	1hu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25 <sub>hex</sub>
Load Linked	11	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)	
Nor	nor	R	$R[rd] = \sim (R[rs] \mid R[rt])$		0 / 27 <sub>he</sub>
Or	or	R	R[rd] = R[rs]   R[rt]		0 / 25 <sub>he</sub>
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		$0/2a_{he}$
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 <sub>he</sub>
Shift Left Logical	sll	R	$R[rd] = R[rt] \le shamt$		0 / 00 <sub>he</sub>
Shift Right Logical	srl	R	R[rd] = R[rt] >>> shamt		0 / 02 <sub>he</sub>
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28 <sub>hex</sub>
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7)	38 <sub>hex</sub>
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29 <sub>hex</sub>
Store Word	SW	I	M[R[rs]+SignExtImm] = R[rt]	(2)	2b <sub>hex</sub>
Subtract Unsigned	sub	R	$R[rd] = R[rs] \cdot R[rt]$	(1)	0 / 22 <sub>he</sub> 0 / 23 <sub>he</sub>
Subtract Unsigned	subu (1) Ma	R	R[rd] = R[rs] - R[rt] se overflow exception		0 / 23 <sub>he</sub>
	(2) Sig (3) Zer (4) Bra (5) Jur (6) Op	nExtI roExtI anchA npAde	mm = { 16{immediate[15]}, imme mm = { 16{1b'0}, immediate } ddr = { 14{immediate[15]}, imme dir = { PC+4[31:28], address, 2'b s considered unsigned numbers (vs est&set pair; R[rt] = 1 if pair atomi	diate, 0 } s. 2's c	2'b0 }
BASIC INSTRUCTI	ON FO	RMA	TS		
R opcode		s	rt rd shamt		funct
	26 25		20 16 15 11 10	6.5	
I opcode	26 25	S 21	rt immed		

ARITHMETIC CORE INS	TRU		OPCODE
			/ FMT /FT
	FOR		/ FUNCT
NAME, MNEMONIC	MAT		(Hex)
Branch On FP True bolt	FI	if(FPcond)PC=PC+4+BranchAddr (4)	11/8/1/
Branch On FP False belf	FI	if(!FPcond)PC=PC+4+BranchAddr(4)	
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)	
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/ <b></b> /y
	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 1wc1	I	F[rt]=M[R[rs]+SignExtImm] (2)	31//
Load FP Double	I	F[rt]=M[R[rs]+SignExtImm]; (2) F[rt+1]=M[R[rs]+SignExtImm+4]	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]; (2) M[R[rs]+SignExtImm+4] = F[rt+1]	3d//

### FLOATING-POINT INSTRUCTION FORMATS

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

### **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	$if(R[rs] \le R[rt]) PC = Label$
Branch Greater Than or Equal	bge	$if(R[rs] \ge R[rt]) PC = Label$
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

### REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVEDACROSS
14711411	HOMBLIC	OSE	A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results	No
		and Expression Evaluation	- 110
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0 <b>-</b> \$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.

address

opcode

### Các lệnh xét:

lbu, lhu, lui, lw

sb, sh, sw

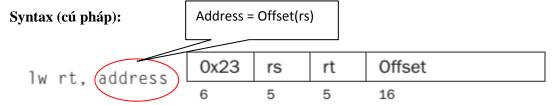
Chú ý: lệnh *ll* và sc bỏ qua

-----

### 17. Lệnh lw

Load Word 
$$lw$$
 I  $R[rt] = M[R[rs] + SignExtImm]$  (2)  $23_{hex}$ 

 $\Rightarrow$  Lệnh thuộc dạng I-format, có opcode =  $23_{hex}$ 



### $\acute{Y}$ nghĩa: R[rt] = M[R[rs] + SingExtImm]

⇒ Lấy giá trị trong thanh ghi rs cộng với số tức thời đang lưu trong offset (số tức thời này này được mở rộng có dấu thanh 32 bits) ta được địa chỉ của từ nhớ cần lấy dữ liệu. Dữ liệu của từ nhớ này sẽ được lấy để lưu vào thanh ghi rt

Lưu ý:

M[X]: là lấy giá trị của từ nhớ tại địa chỉ X

### Ví dụ:

giả sử 
$$t0 = 0x10010000$$

và từ nhớ tại địa chỉ 0x10010004 có giá trị 0x12345678

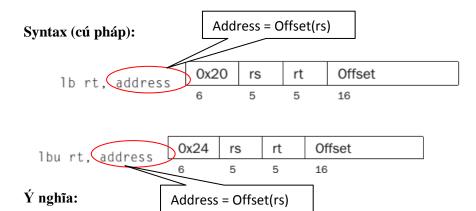
⇒ Lệnh lw thực hiện việc load một từ nhớ (word) tại địa chỉ \$t0 + 4 = 0x10010004 vào thanh ghi \$t1 Kết quả: \$t1 = 0x12345678

### 18. Lệnh lbu/lb

### lbu

Load Byte Unsigned 1bu I 
$$R[rt]=\{24'b0,M[R[rs] + SignExtImm](7:0)\}$$
 (2)

⇒ Lênh thuộc nhóm lệnh I-format, có opcode = 24<sub>hex</sub>



 $R[rt] = \{24'b0, M[R[rs] + SignExtImm](7:0)\}$ 

⇒ Không giống lw là load hết toàn bộ 1 từ nhớ vào thanh ghi rt, lệnh này chỉ load về 1 byte đầu tiên trong từ nhớ vào rt

Vì rt là thanh ghi 32 bits nên 24 bit còn lại của rt có 2 kiểu mở rộng dấu:

- lb: sign-ext
- lbu: zero-ext

### Ví dụ:

a. lbu \$t1, 8(\$t0)

giå sử \$t0 = 0x10010000

và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12345678

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

⇒ Lệnh *lbu* thực hiện việc load một byte vào thanh ghi \$t1, và byte được load là byte 0. Nhưng do thanh ghi \$t1 là
 32 bits, nên 24 bits còn lại là 0

Kết quả: \$t1 = 0x00000078

b. lb \$t1, 8(\$t0)

giả sử \$t0 = 0x10010000

và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x123456f8

Từ nhớ này chứa 4 bytes:

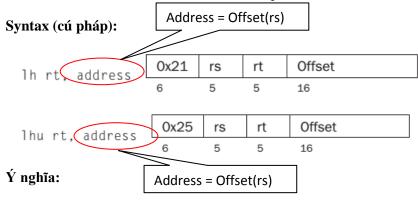
Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	f8

□ Lệnh lb thực hiện việc load một byte vào thanh ghi \$t1, và byte được load là byte 0. Nhưng do thanh ghi \$t1 là 32 bits, nên 24 bits còn lại trong lệnh này được mở rộng có dấu theo bit lớn nhất của byte được load về Kết quả: \$t1 = 0xfffffff8

### 19. Lệnh lhu/lh

Load Halfword Unsigned I 
$$R[rt]=\{16^{\circ}b0,M[R[rs] + SignExtImm](15:0)\}$$
 (2)

⇒ Lệnh thuộc nhóm lệnh I-format, có opcode 25<sub>hex</sub>



 $R[rt] = \{16'b0, M[R[rs] + SignExtImm](15:0)\}$ 

- ➡ Lệnh này chỉ load về 2 byte đầu tiên (nữa word thấp) trong từ nhớ vào rt
   Vì rt là thanh ghi 32 bits nên 16 bit còn lại của rt có 2 kiểu mở rộng dấu:
  - lh: sign-ext
  - lhu: zero-ext

### Ví dụ:

a. lhu \$t1, 8(\$t0)
 giả sử \$t0 = 0x10010000
 và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12345678

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

- □ Lệnh lhu thực hiện việc load 2 byte vào thanh ghi \$11, và byte được load là byte 0 và byte 1. Nhưng do thanh ghi \$11 là 32 bits, nên 16 bits còn lại là 0
- Kết quả: \$t1 = 0x00005678

b. lh \$t1, 8(\$t0)
 giả sử \$t0 = 0x10010000
 và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x123456f8

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	f8

- c. lh \$t1, 8(\$t0)

giả sử \$t0 = 0x10010000

và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12348cde

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	8c	de

### 20. Lệnh sw

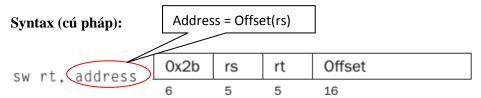
Store Word

SW

I M[R[rs]+SignExtImm] = R[rt]

(2) 2b<sub>hex</sub>

 $\Rightarrow$  Lệnh thuộc nhóm lệnh I-format, có opcode =  $2b_{hex}$ 



 $\acute{\mathbf{Y}}$  nghĩa: M[R[rs] + SignExtImm] = R[rt]

□ Lưu giá trị thanh ghi rt vào từ nhớ có địa chỉ được tính bằng giá trị thanh ghi rs cộng với offset (offset được mở rộng có dấu thành số 32 bits trước khi cộng)

### Ví du:

sw \$t1, 8(\$t0)

giả sử \$t0 = 0x10010000

t1 = 0x87654321

và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12345678

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

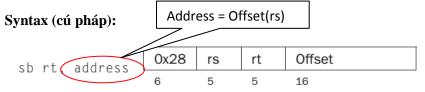
⇒ Lệnh sw thực hiện việc lưu giá trị của thanh ghi \$t1 vào từ nhớ có địa chỉ = \$t0 + 8 = 0x10010008
 Giá trị của từ nhớ tại địa chỉ 0x10010008 sau khi lệnh trên thực hiện là:

Byte 4	Byte 2	Byte 1	Byte 0
87	65	43	21

### 21.Lệnh sb

Store Byte sb I 
$$M[R[rs]+SignExtImm](7:0) = R[rt](7:0)$$
 (2)  $28_{hex}$ 

 $\Rightarrow$  lệnh thuộc nhóm I-format, có opcode =  $28_{hex}$ 



### $\acute{Y}$ nghĩa: M[R[rs] + SignExtImm](7:0) = R[rt](7:0)

### Ví dụ:

sb \$t1, 8(\$t0) 
$$\label{eq:sb} \mbox{giả sử $t0 = 0x10010000} \\ \mbox{$t1 = 0x87654321}$$
 và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12345678

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

⇒ Lệnh sb thực hiện việc lưu byte 0 của thanh ghi t1 (0x21) vào byte 0 của từ nhớ tại địa chỉ 0x10010008. Nên sau lệnh trên, hình ảnh từ nhớ:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	21

fhex

### 22. Lệnh sh

Store Halfword sh I 
$$M[R[rs]+SignExtImm](15:0) = R[rt](15:0)$$
 (2)  $29_{hex}$ 

 $\Rightarrow$  lệnh thuộc nhóm I-format, có opcode =  $29_{hex}$ 

### Syntax (cú pháp):

### $\acute{Y}$ nghĩa: M[R[rs] + SignExtImm](15:0) = R[rt](15:0)

⇒ Lấy 2 byte thấp nhất trong thanh ghi rt (nữa thấp) lưu vào 2 byte thấp nhất của từ nhớ có địa chỉ được tính bằng giá trị thanh ghi rs cộng với offset (offset được mở rộng có dấu thành số 32 bits trước khi cộng)

### Ví du:

giả sử 
$$$t0 = 0x10010000$$

$$t1 = 0x87654321$$

và từ nhớ tại địa chỉ 0x10010008 có giá trị 0x12345678

Từ nhớ này chứa 4 bytes:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	56	78

⇒ Lệnh sh thực hiện việc lưu byte 0 và byte 1 của thanh ghi t1 (0x4321) vào byte 0 và byte 1 của từ nhớ tại địa chỉ 0x10010008. Nên sau lênh trên, hình ảnh từ nhớ:

Byte 4	Byte 2	Byte 1	Byte 0
12	34	43	21

### 23. Lệnh lui

Load Upper Imm. lui 
$$I R[rt] = \{imm, 16'b0\}$$

⇒ Lệnh thuộc I-format, có opcode là f<sub>hex</sub>

### Syntax (cú pháp):

### $\acute{Y}$ nghĩa: $R[rt] = \{imm, 16'b0\}$

⇒ Gán số tức thời 16 bits vào nữa cao của thanh ghi rt, nữa thấp đưa 0 vào

### Ví dụ:

a. lui \$t1, 0x1234

Kết quả: \$t1 = 0x12340000

- b. lui \$t1, 0x12345
- ⇒ báo lỗi do số tức thời tràn quá số 16 bits

### ❖ Ngoài ra còn 2 lệnh thuộc nhóm PseudoInstruction set: *li* và *move*

Load Immediate 1i R[rd] = immediateMove move R[rd] = R[rs]

### 24. Lệnh li

Ý nghĩa: đưa một số tức thời (32 bits) vào một thanh ghi

### Ví dụ:

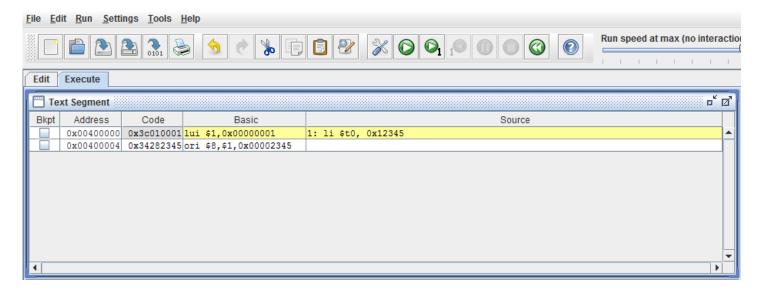
li \$t0, 0x12345

**Kết quả:** \$t0 = 0x00012345

### Luu ý:

Số đưa vào lui chỉ được phép là số 16 bits, còn số đưa và li có thể lên đến 32 bits

Lệnh *li* này thực chất là lệnh giả, lệnh này được chuyển thành 2 lệnh (lui và or) khi processor chạy thật sự:



### 25. Lệnh move

Ý nghĩa: sao chép/di chuyển giá trị từ thanh ghi này sang thanh ghi kia

### Ví dụ:

move \$t1, \$t2

Giả sử \$t1 = 0x12345678

t2 = 0x87654321

Khi lệnh trên thực thi, giá trị thanh ghi \$t2 được đưa vào thanh ghi \$t1

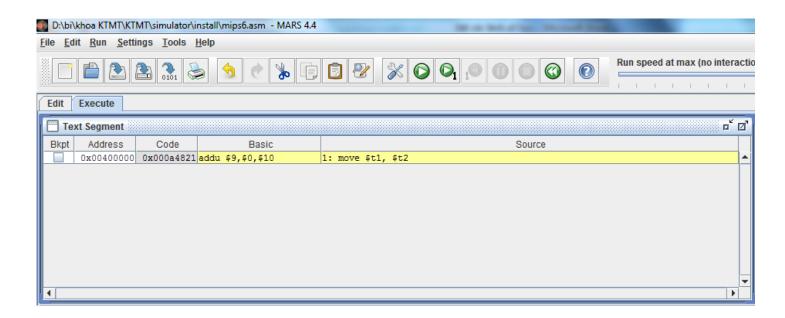
Kết quả sau khi chay:

t1 = 0x87654321

t2 = 0x87654321

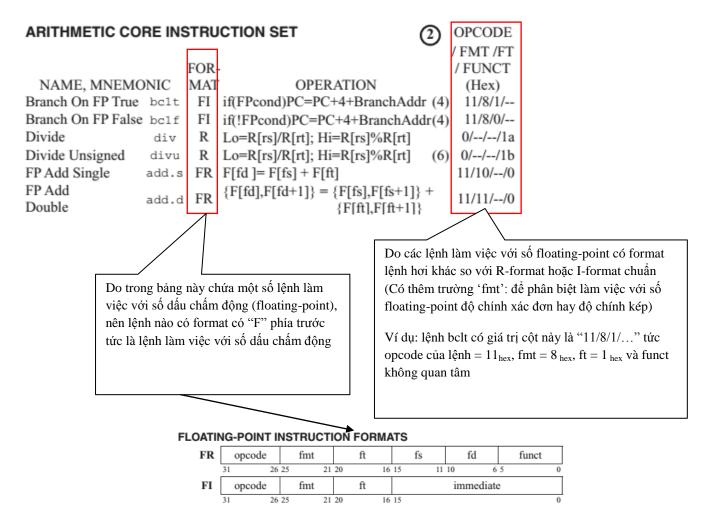
### Chú ý:

Lệnh li này thực chất là lệnh giả, lệnh này được chuyển thành lệnh 'addu' khi processor chạy thật sự (thực hiện việc cộng thanh ghi \$t2 với thanh ghi zero, kết quả nạp vào thanh ghi \$t1:



### CÁC LỆNH ASSEMBLY TRONG KHỐI "ARITHMETIC CORE INSTRUCTION SET"

Các lệnh assembly nằm ở bảng con bên phải của bảng số 1 chứa các lệnh số học phức tạp hơn so với các lệnh bên bảng con trái.



### A. Các lệnh nhân và chia

MIPS	Ref	fer	ence D	ata	1		X
CORE INSTRUCTI			circe D	utu		-4	OPCOD
	0.1.02	FOR-					/ FUNC
NAME, MNEMO	NIC	MAT	OPER	RATION (in	Verilog)		(Hex)
Add	add	R	R[rd] = R[rs]	] + R[rt]		(1)	0 / 20 <sub>he</sub>
Add Immediate	addi	I	R[rt] = R[rs]	+ SignExtI	mm	(1,2)	8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I	R[rt] = R[rs]	+ SignExtI	mm	(2)	9 <sub>hex</sub>
Add Unsigned	addu	R	R[rd] = R[rs]	] + R[rt]			0 / 21 <sub>he</sub>
And	and	R	R[rd] = R[rs]	] & R[rt]			0 / 24 <sub>he</sub>
And Immediate	andi	I	R[rt] = R[rs]	& ZeroExtl	mm	(3)	$c_{\text{hex}}$
Branch On Equal	beq	I		BranchAdd	г	(4)	4 <sub>hex</sub>
Branch On Not Equa	bne	I	if(R[rs]!=R[: PC=PC+4+	rt]) -BranchAdd:	r	(4)	5 <sub>hex</sub>
Jump	j	J	PC=JumpAc	ldr		(5)	2 <sub>hex</sub>
Jump And Link	jal	J	R[31]=PC+8	3;PC=JumpA	Addr	(5)	$3_{\text{hex}}$
Jump Register	jr	R	PC=R[rs]		٨		0 / 08 <sub>he</sub>
Load Byte Unsigned	1bu	I	R[rt]={24'b( +Sig	0,M[R[rs] nExtImm](7	:0)}	(2)	24 <sub>hex</sub>
Load Halfword Unsigned	1hu	I	R[rt]={16'b( +Sig	0,M[R[rs] nExtImm](1	5:0)}	(2)	25 <sub>hex</sub>
Load Linked	11	I	R[rt] = M[R	[rs]+SignEx	tImm]	(2,7)	$30_{hex}$
Load Upper Imm.	lui	I	R[rt] = {imn	n, 16'b0}			$f_{hex}$
Load Word	lw	I	R[rt] = M[R	[rs]+SignEx	tImm]	(2)	23 <sub>hex</sub>
Nor	nor	R	$R[rd] = \sim (R$	[rs]   R[rt])			0 / 27 <sub>hc</sub>
Or	or	R	R[rd] = R[rs]	]   R[rt]			0 / 25 <sub>he</sub>
Or Immediate	ori	I	R[rt] = R[rs]	ZeroExtIn	nm	(3)	$d_{hex}$
Set Less Than	slt	R	R[rd] = (R[r]	s] < R[rt]) ?	1:0		0 / 2a <sub>he</sub>
Set Less Than Imm.	slti	I	R[rt] = (R[rs]	s] < SignExtl	[mm)? 1	: 0(2)	a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs]	s] < SignExtl ? 1 : 0	lmm)	(2,6)	$b_{hex}$
Set Less Than Unsig.	sltu	R	R[rd] = (R[r]	$s$ ] $\leq$ $R[rt]) ?$	1:0	(6)	$0/2b_{hc}$
Shift Left Logical	sll	R	R[rd] = R[rt]	] << shamt			$0 / 00_{hc}$
Shift Right Logical	srl	R	R[rd] = R[rt]	] >>> shamt			$0 / 02_{hc}$
Store Byte	sb	I	M[R[rs]+Sig	gnExtImm](7 R[rt](7		(2)	28 <sub>hex</sub>
Store Conditional	sc	I	M[R[rs]+Sig R[r	gnExtImm] = t] = (atomic)		(2,7)	38 <sub>hex</sub>
Store Halfword	sh	I	M[R[rs]+Sig		15:0) = [(15:0)	(2)	29 <sub>hex</sub>
Store Word	sw	I	M[R[rs]+Sig	gnExtImm] =	R[rt]	(2)	2b <sub>hex</sub>
Subtract	sub	R	R[rd] = R[rs]	] - R[rt]		(1)	0 / 22 <sub>hc</sub>
Subtract Unsigned	subu	R	R[rd] = R[rs]	] - R[rt]			$0/23_{hc}$
	(2) Sig	nExtI	se overflow e mm = { 16{ii	mmediate[15		ediate	}
			mm = { 16{1 ddr = { 14{ir			diate,	2'b0 }
	(5) Jun	npAd	dr = { PC+4 s considered t	4[31:28], add	iress, 2'b	0 }	
DACIO INICEDIA	(7) Ato	mic t	est&set pair;				
BASIC INSTRUCTI	1				.1		C
R opcode	6 25	'S	20 16	rd 15 11	shamt 10	6.5	funct
I opcode	Т.	'S	rt		immed	_	
31 2	6 25	21	20 16	15			
J opcode	1			address			

ARITHMETIC CO	RE INS	TRU	CTION SET (2)	OPCODE
			0	FMT/FT
		FOR-		/ FUNCT
NAME, MNEMO		MAT	OPERATION	(Hex)
Branch On FP True	bclt	FΙ	if(FPcond)PC=PC+4+BranchAddr (4)	11/8/1/
Branch On FP False	bclf	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/ <b></b> /y
			==, <, 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		${F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}$	11/11//1
Load FP Single	lwcl	I	F[rt]=M[R[rs]+SignExtImm] (2)	31//
Load FP Double	ldcl	I	$\begin{split} F[rt] &= M[R[rs] + SignExtImm]; \\ F[rt+1] &= M[R[rs] + SignExtImm + 4] \end{split} \tag{2}$	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	swcl	I		39//
Store FP Double	sdcl	I	$\begin{aligned} &M[R[rs]+SignExtImm] = F[rt]; \\ &M[R[rs]+SignExtImm+4] = F[rt+1] \end{aligned} \tag{2}$	3d//

### FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft		immediate	3
	31 26	25 21	20 16	15		0

### PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	$if(R[rs] \le R[rt]) PC = Label$
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	$if(R[rs] \le R[rt]) PC = Label$
Branch Greater Than or Equal	bge	$if(R[rs] \ge R[rt]) PC = Label$
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 <b>-</b> \$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0 <b>-</b> \$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.

Các lệnh assembly cần khi thực hiện phép nhân và chia:

• Nhân: mult/multu

• Chia: div/divu

• Hai lệnh hỗ trợ: mfhi/mflo

### Lệnh mult/multu

Multiply mult R 
$$\{Hi,Lo\} = R[rs] * R[rt]$$
 0/--/--/18  
Multiply Unsigned multu R  $\{Hi,Lo\} = R[rs] * R[rt]$  (6) 0/--/--/19

⇒ Hai lênh này thuộc nhóm lênh R-format, có opcode là 0. Lênh *mult* có funct là 18, lênh *multu* có funct là 19

### **Syntax:**

### Ý nghĩa:

$${Hi, Lo} = R[rs] * R[rt]$$

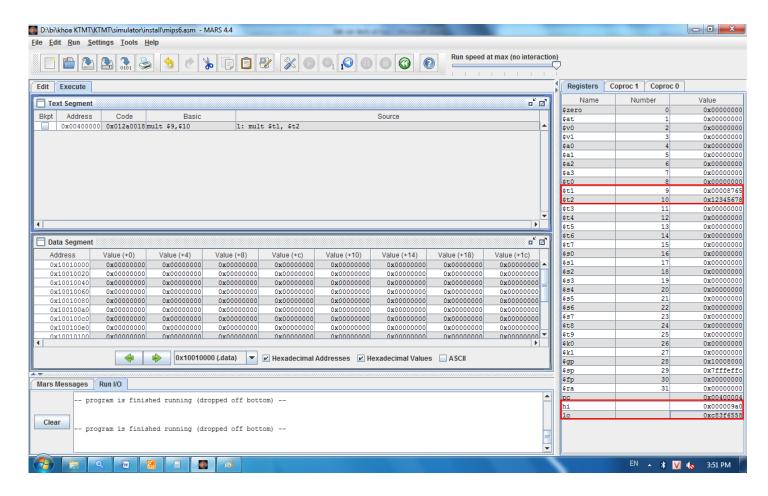
➡ Giá trị trong thanh ghi rs (số 32 bits) nhân với giá trị trong thanh ghi rt (số 32 bits), kết quả là số 64 bits. 32 bits thuộc nữa thấp của kết quả được lưu trong thanh ghi Lo, và 32 bits thuộc nữa cao của kết quả được lưu trong thanh ghi Hi

Chú ý: *Hi* và *Lo* là 2 thanh ghi phụ thêm cho processor khi thực hiện phép toán nhân chia. Vì các thanh ghi đều là 32 bits, mà kết quả phép toán nhân là 64 bits, nên phải dùng 2 thanh ghi tạm này ghép lại

• mult: nhân 2 số có dấu

• multu: nhân 2 số không dấu

### Ví dụ:



b. mult \$t1, \$t2

Giả sử t1 = 0x80008765

t2 = 0x12345678

Kết quả:

hi = 0xf6e5de64

lo = 0xc83f6558

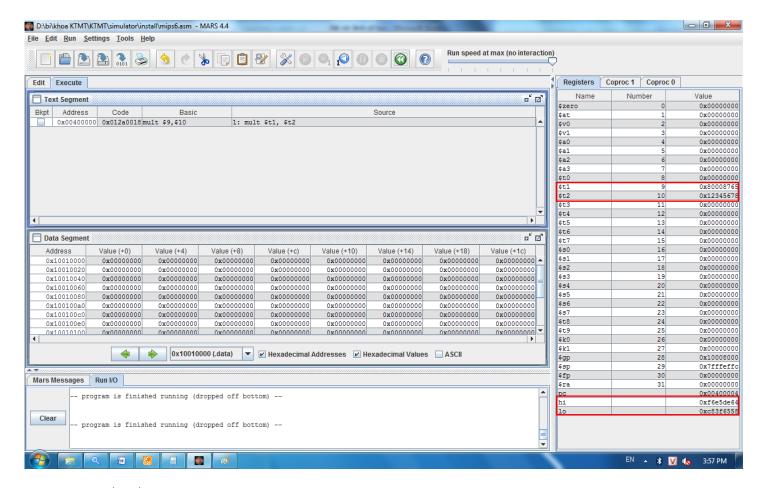
Tức t1\*t2 = -0x91A219B37C09AA8 (xét trên số có dấu dạng bù 2)

### Kiểm tra lai:

 $t1 = 0x80008765 = 1000\ 0000\ 0000\ 0000\ 1000\ 0111\ 0110\ 0101_{(2)}$ = -0x7FFF789B

t2 = 0x12345678

 $\Rightarrow $t1 * $t2 = -(0x7FFF789B x 0x12345678) = -0x91A219B37C09AA8$ = 0xF6E5DE64C83F6558 (bù 2)



c. multu \$t1, \$t2

Giả sử \$t1 = 0x80008765

t2 = 0x12345678

Kết quả:

hi = 0x091a34dc

lo = 0xc83f6558

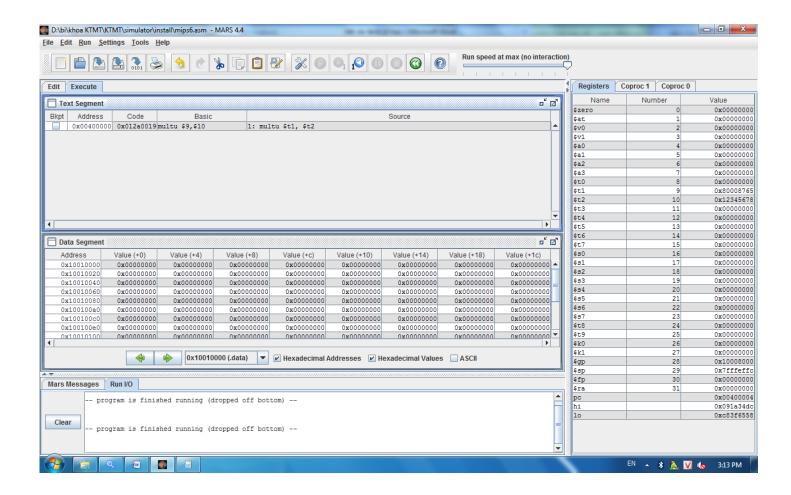
### Kiểm chứng kết quả:

multu thực hiện phép nhân 2 số dạng không dấu

 $t1 = 0x80008765 = 20000103545_{(10)}$ 

 $t2 = 0x12345678 = 2215053170_{(10)}$ 

 $t1 x t2 = 20000103545_{(10)} x 2215053170_{(10)} = 44321515631017662530_{(10)} = 0x91A34DCC83F6558$ 



### Chia: div/divu

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

⇒ Hai lệnh này thuộc nhóm lệnh R-format, có opcode là 0. Lệnh *div* có funct là 1a<sub>hex</sub>, lệnh *divu* có funct là 1b<sub>hex</sub>

### **Syntax:**

### Divide (with overflow)

div rdest, rsrc1, src2 pseudoinstruction

### Divide (without overflow)

divu rdest, rsrc1, src2 pseudoinstruction

### Ý nghĩa:

rdest, rsrcl và src2 là 3 thanh ghi.

Hai lệnh trên lấy giá trị trong thanh ghi rsrcl chia cho src2, thương số đặt vào thanh ghi rdest. Đồng thời thanh ghi Lo cũng chứa thương số giống than ghi rdest và thanh ghi Hi chứa phần dư.

- div: chia 2 số có xét tràn
- divu: chia 2 số không xét tràn

### Ví dụ:

a. div \$t1, \$t2, \$t3

giả sử 
$$$t2 = 0x6$$
,  $$t3 = 0x5$ 

**Kết quả**: \$t1 = 0x1

Lo = 0x1

Hi = 0x1

♣ Đưa thêm ví dụ kiểm chứng trường hợp tràn và không tràn (div/divu)

### CÁC LỆNH ASSEMBLY CHO SỐ DẦU CHẨM ĐỘNG (floating-point number)

MIPS  CORE INSTRUCTI			ence Data	1	1	OP
CORE INSTRUCTI	ON SE	FOR-				/FI
NAME, MNEMO	NIC	MAT		N (in Verilog)	)	(I
Add	add	R	R[rd] = R[rs] + R[	rt]	(1)	0 /
Add Immediate	addi	I	R[rt] = R[rs] + Sig	nExtImm	(1,2)	
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + Sig	nExtImm	(2)	
Add Unsigned	addu	R	R[rd] = R[rs] + R[	rt]		0
And	and	R	R[rd] = R[rs] & R	[rt]		0 /
And Immediate	andi	I	R[rt] = R[rs] & Ze	roExtImm	(3)	
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+Branc	hAddr	(4)	
Branch On Not Equa	bne	Ι	if(R[rs]!=R[rt]) PC=PC+4+Branc	hAddr	(4)	
Jump	j	J	PC=JumpAddr		(5)	
Jump And Link	jal	J	R[31]=PC+8;PC=	JumpAddr	(5)	
Jump Register	jr	R	PC=R[rs]			0
Load Byte Unsigned	1bu	Ι	R[rt]={24'b0,M[R +SignExtIr	nm](7:0)}	(2)	2
Load Halfword Unsigned	1hu	I	R[rt]={16'b0,M[R +SignExtIr		(2)	2
Load Linked	11	I	R[rt] = M[R[rs] + S	ignExtImm]	(2,7)	
Load Upper Imm.	lui	I	R[rt] = {imm, 16't	00}		
Load Word	lw	I	R[rt] = M[R[rs] + S	ignExtImm]	(2)	
Nor	nor	R	$R[rd] = \sim (R[rs] \mid I$	R[rt])		0
Or	or	R	R[rd] = R[rs]   R[r]	t]		0
Or Immediate	ori	I	R[rt] = R[rs]   Zero	ExtImm	(3)	
Set Less Than	slt	R	R[rd] = (R[rs] < R	[rt]) ? 1 : 0		0
Set Less Than Imm.	slti	Ι	R[rt] = (R[rs] < Si		: 0(2)	
Set Less Than Imm. Unsigned	sltiu	Ι	R[rt] = (R[rs] < Si ? 1:0	gnExtImm)	(2,6)	
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R	[rt])?1:0	(6)	0
Shift Left Logical	sll	R	R[rd] = R[rt] << sl	namt		0
Shift Right Logical	srl	R	R[rd] = R[rt] >>>	shamt		0
Store Byte	sb	I	M[R[rs]+SignExtI		(2)	
Store Conditional	sc	Ι	M[R[rs]+SignExtI			
Store Halfword	sh	I	R[rt] = (a $M[R[rs]+SignExtI$		(2,7)	
Stone Word	sw		MIDImaly Circuit of	R[rt](15:0)	(2)	
Store Word		I	M[R[rs]+SignExtI		(2)	
Subtract Unional	sub	R	$R[rd] = R[rs] \cdot R[r$		(1)	0
Subtract Unsigned	subu (1) Ma	R v can	R[rd] = R[rs] - R[r se overflow excepti			U
	(2) Sig (3) Zer (4) Bra (5) Jun (6) Ope	nExtl roExtl inchA npAderand	mm = { 16{immedi mm = { 16{1b'0}, ddr = { 14{immedi dr = { PC+4[31:2: s considered unsign est&set pair; R[rt] =	iate[15]}, imm immediate } ate[15]}, imm 8], address, 2'l ed numbers (v	ediate, b0 }	2'l
BASIC INSTRUCTI				_		
R opcode	r			d sham	ıt	fi
	26 25		20 16 15	11 10	6 5	
I opcode	r		rt	immed	diate	
J opcode	26 25	21	20 16 15 ado			_

ARITHMETIC CORE	INS	TRU	CTION SET (2)	OPCODE
			•	FMT/FT
		OR-		/ FUNCT
NAME, MNEMONI		MAT		(Hex)
Branch On FP True bo		FΙ	if(FPcond)PC=PC+4+BranchAddr (4)	11/8/1/
Branch On FP False be	clf	FΙ	if(!FPcond)PC=PC+4+BranchAddr(4)	
Divide d	liv	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0//-1a
Divide Unsigned di	ivu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] (6)	0///1b
FP Add Single ad	id.s	FR	F[fd] = F[fs] + F[ft]	11/10//0
FP Add	id.d	FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} +$	11/11//0
Double	ia.a	ЛЛ	{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/ <b></b> /y
FP Compare	x.d*	FR	$FPcond = ({F[fs], F[fs+1]}) op$	11/11//v
Double	x.a-	и	{F[ft],F[ft+1]})?1:0	11/11//y
			==, <, or <=) ( y is 32, 3c, or 3e)	
	v.s	FR	F[fd] = F[fs] / F[ft]	11/10//3
FP Divide	v.d	FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} /$	11/11//3
Double			{F[ft],F[ft+1]}	
FP Multiply Single mu	ıl.s	FR	F[fd] = F[fs] * F[ft]	11/10//2
FP Multiply	11.d	FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} *$	11/11//2
Double			{F[ft],F[ft+1]}	
	ıb.s	FR	F[fd]=F[fs] - F[ft]	11/10//1
FP Subtract	ıb.d	FR	${F[fd],F[fd+1]} = {F[fs],F[fs+1]} -$	11/11//1
Double	ю.ч		{F[ft],F[ft+1]}	,, ,-
	wcl	I	F[rt]=M[R[rs]+SignExtImm] (2)	31//
Load FP	dcl	I	F[rt]=M[R[rs]+SignExtImm]; (2)	35//
Double	401	1	F[rt+1]=M[R[rs]+SignExtImm+4]	33//
Move From Hi m:	fhi	R	R[rd] = Hi	0 ///10
Move From Lo m:	flo	R	R[rd] = Lo	0 ///12
Move From Control m:	fc0	R	R[rd] = CR[rs]	10 /0//0
Multiply m	ult	R	$\{Hi,Lo\} = R[rs] * R[rt]$	0///18
Multiply Unsigned mu	ıltu	R	$\{Hi,Lo\} = R[rs] * R[rt] $ (6)	0///19
Shift Right Arith. s	sra	R	R[rd] = R[rt] >> shamt	0//-3
Store FP Single st	wcl	I	M[R[rs]+SignExtImm] = F[rt] (2)	39//
Store FP			M[R[rs]+SignExtImm] = F[rt]; (2)	3d//
Double	dcl	I	M[R[rs]+SignExtImm+4] = F[rt+1]	30//

### FLOATING-POINT INSTRUCTION FORMATS

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

### 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	$if(R[rs] \le R[rt]) PC = Label$
Branch Greater Than or Equal	bge	$if(R[rs] \ge R[rt]) PC = Label$
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 <b>-</b> \$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0 <b>-</b> \$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

Computer Organization and Design, 4th ed.

### Nhóm lệnh số học:

### Cộng

add.s

add.d

### Trừ

sub.s

sub.d

### Nhân

mul.s

mul.d

### Chia

Double

div.s

div.d

Chú ý: Các lênh làm việc với số floating-point làm việc trên 32 thanh ghi f

FP Add Single add.s FR F[fd] = F[fs] + F[ft]11/10/--/0 FP Add  ${F[fd],F[fd+1]} = {F[fs],F[fs+1]} +$ add.d FR 11/11/--/0 Double {F[ft],F[ft+1]} FP Subtract Single sub.s FR F[fd]=F[fs] - F[ft] 11/10/--/1 sub.d  $FR \{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\}$ FP Subtract 11/11/--/1 Double {F[ft],F[ft+1]} 11/10/--/2 FP Multiply Single mul.s FR F[fd] = F[fs] \* F[ft]FP Multiply  ${F[fd],F[fd+1]} = {F[fs],F[fs+1]} *$ 11/11/--/2 Double {F[ft],F[ft+1]} FP Divide Single div.s FR F[fd] = F[fs] / F[ft]11/10/--/3 FP Divide  ${F[fd],F[fd+1]} = {F[fs],F[fs+1]} /$ div.d FR 11/11/--/3

 $\Rightarrow$  Các lệnh này thuộc nhóm lệnh FR, có opcode đều bằng  $11_{\rm hex}$ 

Lệnh add có funct bằng 0, lệnh sub có funct bằng 1 hex, lệnh mul có funct bằng 2 hex, lệnh div có funct bằng 3 hex

{F[ft],F[ft+1]}

Chú ý: các lệnh .s là làm việc với độ chính xác đơn, có trường fmt luôn bằng  $10_{\rm hex}$ ; các lệnh .d là làm việc với độ chính xác kep, có trường fmt luôn bằng  $11_{\rm hex}$ 

### add.s

**Syntax** 

### Ý nghĩa:

$$F[fd] = F[fs] + F[ft]$$

⇒ Giá trị trong thanh ghi fs cộng với giá trị trong thanh ghi ft, tổng lưu vào thanh ghi fd

### add.d

### **Syntax**

### Ý nghĩa:

$$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$$

- ⇒ Số tức thời lúc này do biểu diễn trong độ chính xác kép nên cần 64 bits, tức phải cần 2 thanh ghi f liên tục nhau.

  Lệnh add.d sẽ thực hiện việc cộng giá trị của số floating point độ chính xác kép đang lưu trong
- ⇒ Giá trị trong thanh ghi fs cộng với giá trị trong thanh ghi ft, tổng lưu vào thanh ghi fd