Schriftlicher Test
Rechnerorganisation
26.02.2019

Fakultät IV
Institut für Technische Informatik und Mikroelektronik
FG Architektur Eingebetteter Systeme

Nachname	:	.....
Vorname	:	.....
Matrikelnummer	:	.....
Studiengang	:	.....

Aufgabe	1	2	3	4	$\sum$
max. Punkte	18	12	12	18	60
erreichte Punkte					
Kürzel					

Wichtige Hinweise:

- Füllen Sie das Deckblatt aus und versehen Sie alle abgegebenen Seiten mit Namen und Matrikelnummer.
- Es sind nur dokumentenechte Stifte in schwarz und blau erlaubt (keine Bleistifte, radierbare Kulis, Buntstifte...).
- Es darf ein nicht-programmierbarer Taschenrechner verwendet werden.
- Alle elektronischen Geräte sind auszuschalten. Ein eingeschaltetes Handy gilt als Täuschungsversuch.
- Der Bearbeitungszeit beträgt **75 Minuten**.
- Der Test besteht aus 12 Seiten und darf nicht auseinandergenommen werden. Ausnahme: die MIPS-Green-Card darf abgetrennt werden.
- Bei jeder Aufgabe steht, wieviele Punkte erzielt werden können.
- Für die Lösungen sind die beigehefteten Aufgabenblätter zu verwenden; Zusatzblätter werden nach Bedarf ausgegeben.
- Alle Lösungswege müssen nachvollziehbar und alle Endergebnisse deutlich gekennzeichnet sein.
- Täuschungsversuche werden mit einem Nichtbestehen des Moduls geahndet.

1. Aufgabe: Pipelining (18 Punkte)

Gegeben ist folgender Assemblercode:

```
1 add   $sp, $sp, 4
2 slt   $s0, $sp, $a0
3 lb    $t1, 0($sp)
4 sub   $t3, $t1, $a1
5 lw    $t2, 4($a2)
6 add   $t4, $a0, $t2
```

Der Code soll auf der aus der Vorlesung bekannten 5-Stufen-Pipeline des MIPS-Prozessors (IF, ID, EX, MEM, WB) ausgeführt werden. Der Registerspeicher wird in der zweiten Hälfte der Decode-Phase gelesen und in der ersten Hälfte der Write-Back Phase geschrieben.

(a) In welchen Code-Zeilen können potentiell Hazards (Pipeline-Hemmnisse, Pipeline-Konflikte) auftreten? Um welche Art von Hazards handelt es sich?

(b) Durch das Einfügen von NOP-Instruktionen können Hazards verhindert werden. Kreuzen Sie die **minimal** benötige Anzahl von NOP-Instruktionen an, die benötigt werden, damit das Programm korrekt ausgeführt werden kann.

	Anzahl NOPs			
Instruktion	0	1	2	3
`add $sp, $sp, $4`				
	☐	☐	☐	☐
`slt $s0, $sp, $a0`				
	☐	☐	☐	☐
`lb $t1, 0($sp)`				
	☐	☐	☐	☐
`sub $t3, $t1, $a1`				
	☐	☐	☐	☐
`lw $t2, 4($a2)`				
	☐	☐	☐	☐
`add $t4, $a0, $t2`				

(c) Wie groß ist der CPI des Programms aus (b)? Gehen Sie davon aus, dass die Pipeline zu Beginn der Ausführung leer ist.

Zeile ↓	Takt →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1: add $sp, $sp, $4																
2: slt $s0, $sp, $a0																
3: lb $t1, 0($sp)																
4: sub $t3, $t1, $a1																
5: lw $t2, 4($a2)																
6: add $t4, $a0, $t2																
7:																

(d) Nehmen Sie jetzt an, dass der Prozessor über eine Forwarding-Unit, Hazard-Detection und einen Branch-Delay-Slot verfügt. Vervollständigen Sie das oben stehende Pipeline-Diagramm für den gezeigten Code. Fügen Sie die Kürzel IF, ID, EX, MEM, WB für jede Instruktion in die Tabelle ein und markieren Sie alle Wartezyklen (stall cycles), indem Sie an entsprechender Stelle ein X notieren. Markieren Sie alle Data-Forwards mit Pfeilen zwischen den beiden beteiligten Stufen.

(e) Wie groß ist der CPI Ihrer Lösungen aus (d)? Welcher Speedup kann demnach durch den Einsatz der Forwarding-Unit und der Hazard-Detection erzielt werden?

ERSATZTABELLE:

Zeile ↓	Takt →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1: add $sp, $sp, $4																
2: slt $s0, $sp, $a0																
3: lb $t1, 0($sp)																
4: sub $t3, $t1, $a1																
5: lw $t2, 4($a2)																
6: add $t4, $a0, $t2																
7:																

2. Aufgabe: Rechenleistung (12 Punkte)

Nehmen Sie an, ein Speicherzugriff kostet 150 ps, eine ALU-Operation 100 ps und ein Zugriff auf den Registersatz 50 ps. Andere Latenzen können vernachlässigt werden.

(a) Wie groß ist die minimale Taktzykluszeit eines Eintaktprozessors mit diesen Latenzen?

(b) Wie groß ist die minimale Taktzykluszeit eines Pipelined-Prozessors mit der klassischen 5-Stufen-Pipeline, ebenfalls unter Annahme der oben genannten Latenzen?

(c) Welcher maximale Speedup ist demnach möglich, wenn man den Pipelined-Prozessor einsetzt?

(d) Der Faktor aus der vorhergehenden Teilaufgabe kann nur erreicht werden, wenn der Pipelined-Prozessor einen CPI von 1 erreicht. Selbst wenn der Prozessor Forwarding unterstützt und Sprünge verzögert ausführen kann (durch einen *Branch Delay Slot*), ist dies nur selten möglich. Geben Sie hierfür zwei Gründe an.

(e) Sie möchten sowohl Eintakt- als auch Pipelined-Prozessor um den Faktor 1,15 beschleunigen. Angenommen, Sie können dafür genau eine der Komponenten (Speicherzugriff, ALU-Operation oder Registersatz) beschleunigen. Wie groß darf die Latenz der Komponente nach der Verbesserung noch sein, um den gewünschten Speedupfaktor zu erzielen? Nutzen Sie die folgende Tabelle für Ihre Ergebnisse und vergessen Sie nicht, Ihren Lösungsweg zu notieren!

	Eintakt			Pipelined
Komponente	Speicher	ALU	Register	_____
Neue Latenz	_____	_____	_____	_____

Lösungsweg:

3. Aufgabe: Speicherhierarchie (12 Punkte)

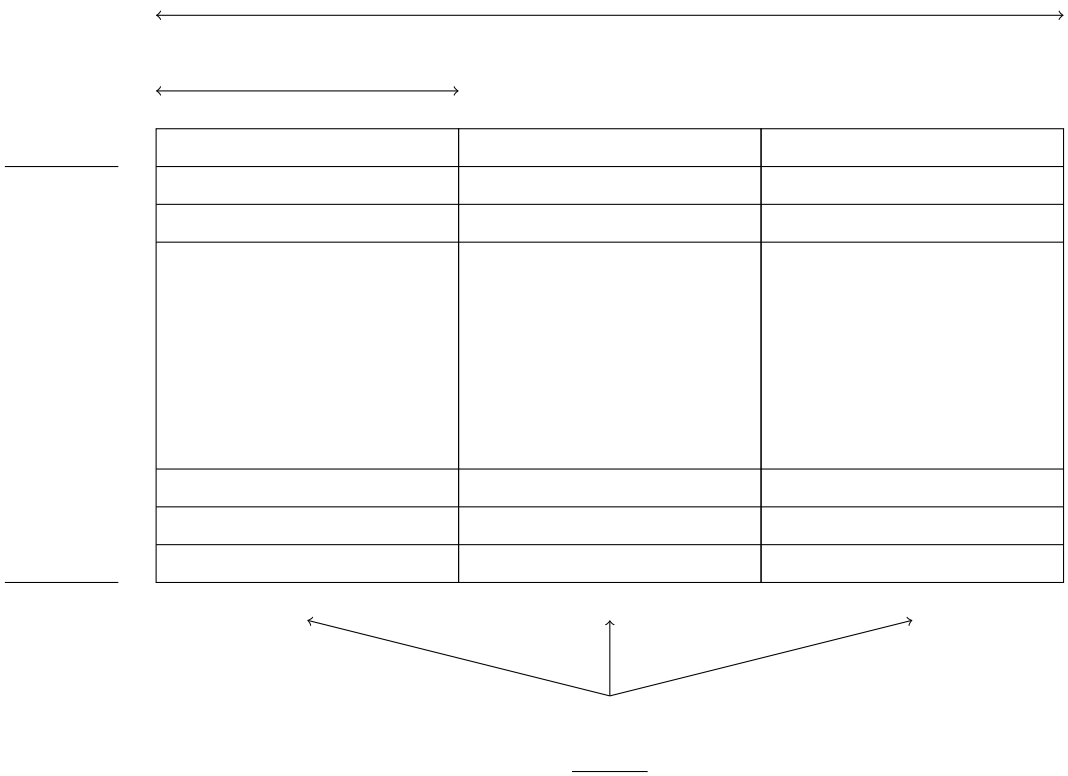
Ein Prozessor verfügt über einen 3-fach-satzassoziativen Cache. Für die Adressierung des Caches werden 36 bit für den Tag, 8 bit für den Index und 4 bit für den Blockoffset verwendet.

(a) Mit welcher Adressgröße arbeitet der Prozessor?

(b) Wie viele Sätze gibt es in dem Cache? Wie groß ist ein Block?

(c) Vervollständigen Sie die Skizze des Caches, indem Sie die folgenden Werte eintragen:

- minimaler und maximaler Index
- Blockgröße in Byte
- Satzgröße in Byte
- Anzahl der Ways



(d) Wie groß ist die Kapazität des Caches?

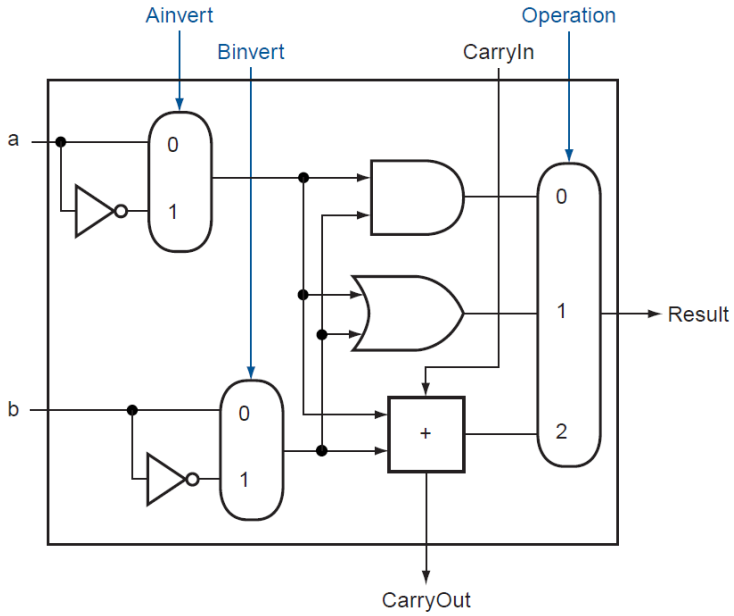
4. Aufgabe: Datenpfaderweiterung (18 Punkte)

Der aus der Vorlesung bekannte Eintaktprozessor (siehe gegenüberliegende Seite) soll um den R-Typ Befehl *xor* erweitert werden.

(a) Verfassen Sie ein MIPS-Assemblerprogramm, welches die `xor`-Operation mit den Logikfunktionen `and, not` und `or` implementiert.

(b) Erweitern Sie die aus der Übung bekannte 1-Bit-ALU so, dass diese die `xor`-Operation ausführen kann. Benutzen Sie die Vorgabe auf dieser Seite und beachten Sie, dass es auch möglich sein soll, die negierten Eingänge für die Berechnung zu verwenden!

(c) Wie müssen die Steuersignale der ALU gesetzt sein, damit `a xor b` ausgeführt wird? Achten Sie auf die korrekten Bitbreiten der Signale!



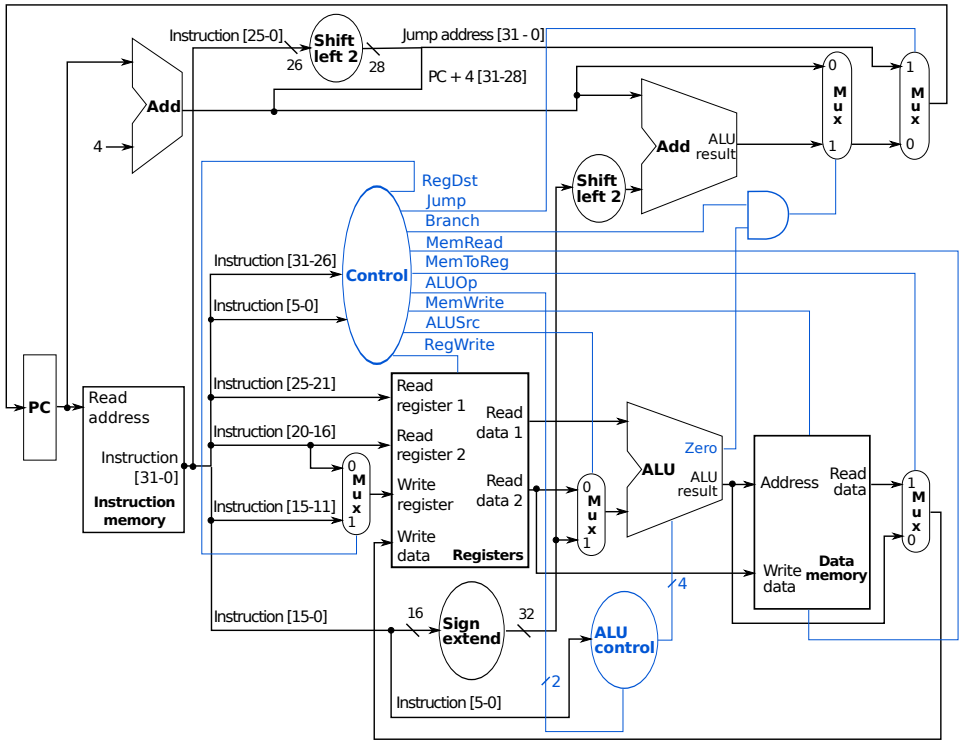
Steuersignal	Wert
A_{invert}	
B_{invert}	
$Operation$	

(d) Erweitern Sie die ALU-Steuerung, indem Sie die folgende Tabelle ergänzen.
(Hinweis: Die `xor`-Operation gehört nicht zum Core Instruction Set, ihr Function Code ist aber auf der Rückseite der Green Card aufgeführt.)

Opcode	Funct.	ALUOp		Funct. Field [5:0]						ALU Ctrl. Signals	Desired ALU Action
`lw/sw`	-	0	0	x	x	x	x	x	x	0010	`add`
`beq`	-	0	1	x	x	x	x	x	x	0110	`sub`
`R-Typ`	`add`	1	0	1	0	0	0	0	0	0010	`add`
`R-Typ`	`sub`	1	0	1	0	0	0	1	0	0110	`sub`
`R-Typ`	`and`	1	0	1	0	0	1	0	0	0000	`and`
`R-Typ`	`or`	1	0	1	0	0	1	0	1	0001	`or`
`R-Typ`	`xor`										

(e) Geben Sie die Steuersignale an, damit der Eintaktprozessor (siehe unten) den `xor`-Befehl ausführt. Verwenden Sie *don't care*, falls möglich.

Steuersignal	Wert	Steuersignal	Wert	Steuersignal	Wert
RegDst	_____	*MemRead*	_____	*ALUOp*	_____
Jump	_____	*MemWrite*	_____	*RegWrite*	_____
Branch	_____	*MemToReg*	_____	*ALUSrc*	_____



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

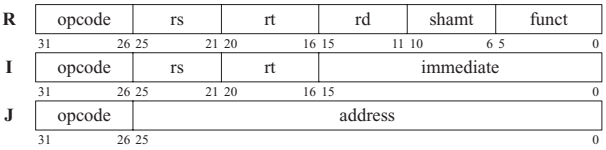
MIPS Reference Data ①

CORE INSTRUCTION SET

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

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

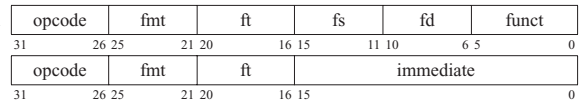
BASIC INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET ②

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

FLOATING-POINT INSTRUCTION FORMATS



PSEUDOINSTRUCTION SET

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

REGISTER NAME, NUMBER, USE, CALL CONVENTION

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

OPCODES, BASE CONVERSION, ASCII SYMBOLS ③

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Decimal	Hexadecimal	ASCII Character	Decimal	Hexadecimal	ASCII Character
(1)	sll	add.*f*	00 0000	0	0	NUL	64	40	@
		sub.*f*	00 0001	1	1	SOH	65	41	A
j	srl	mul.*f*	00 0010	2	2	STX	66	42	B
jal	sra	div.*f*	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.*f*	00 0100	4	4	EOT	68	44	D
bne		abs.*f*	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.*f*	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.*f*	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.*f*	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.*f*	00 1101	13	d	CR	77	4d	M
xori		ceil.w.*f*	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.*f*	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.*f*	01 0010	18	12	DC2	82	52	R
	mtlo	movn.*f*	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.*f*	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.*f*	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.*f*	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.*f*	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.*f*	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.*f*	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.*f*	11 0011	51	33	3	115	73	s
	teq	c.olt.*f*	11 0100	52	34	4	116	74	t
ldc1		c.ult.*f*	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.*f*	11 0110	54	36	6	118	76	v
		c.ule.*f*	11 0111	55	37	7	119	77	w
sc		c.sf.*f*	11 1000	56	38	8	120	78	x
swc1		c.ngle.*f*	11 1001	57	39	9	121	79	y
swc2		c.seq.*f*	11 1010	58	3a	:	122	7a	z
		c.ngl.*f*	11 1011	59	3b	;	123	7b	{
		c.lt.*f*	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.*f*	11 1101	61	3d	=	125	7d	}
sdc2		c.le.*f*	11 1110	62	3e	>	126	7e	~
		c.ngt.*f*	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0

(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

IEEE 754 FLOATING-POINT STANDARD

$$(-1)^S \quad (1 + \text{Fraction}) \quad 2^{(\text{Exponent} - \text{Bias})}$$

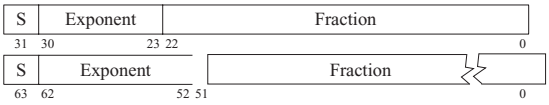
where Single Precision Bias = 127, Double Precision Bias = 1023.

IEEE 754 Symbols ④

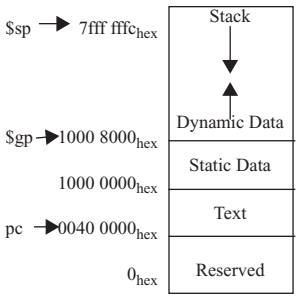
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

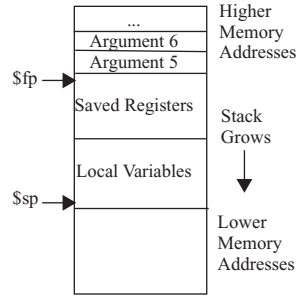
IEEE Single Precision and Double Precision Formats:



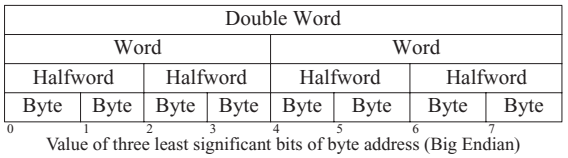
MEMORY ALLOCATION



STACK FRAME

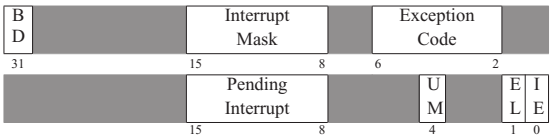


DATA ALIGNMENT



Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except is used for micro.

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