1 Ch 1 stuff

- $[\text{Speedup}] = \frac{[\text{Latency 1}]}{[\text{Latency 2}]} = \frac{[\text{Throughput 1}]}{[\text{Throughput 2}]}$
- $[\text{Performance Improvement}] = [\text{Speedup}] - 1$
- $[\text{Exec. Time}] = [\text{Instr. Count}] \times [\text{CPI}] \times [\text{Clock Speed}]$
- $[\text{Performance}] = [\text{Execution Time}]^{-1}$
- $[\text{Dynamic Power}] \propto [\text{Activity}] \times [\text{Capacitance}] \times [\text{Voltage}]^2 \times [\text{Frequency}]$

2 ASM

- Calle saves \$s# registers, caller saves everything else
- Stack grows down

```
.data
prompt:          .asciiz "Ente...rs: \n"
input:           .space 81
inputSize:       .word 80
exitMessage_p1:  .asciiz "Word count: "
.text
```

3 Binary Stuff

Float

"But we're not dealing with real numbers, this is floating point baby!"

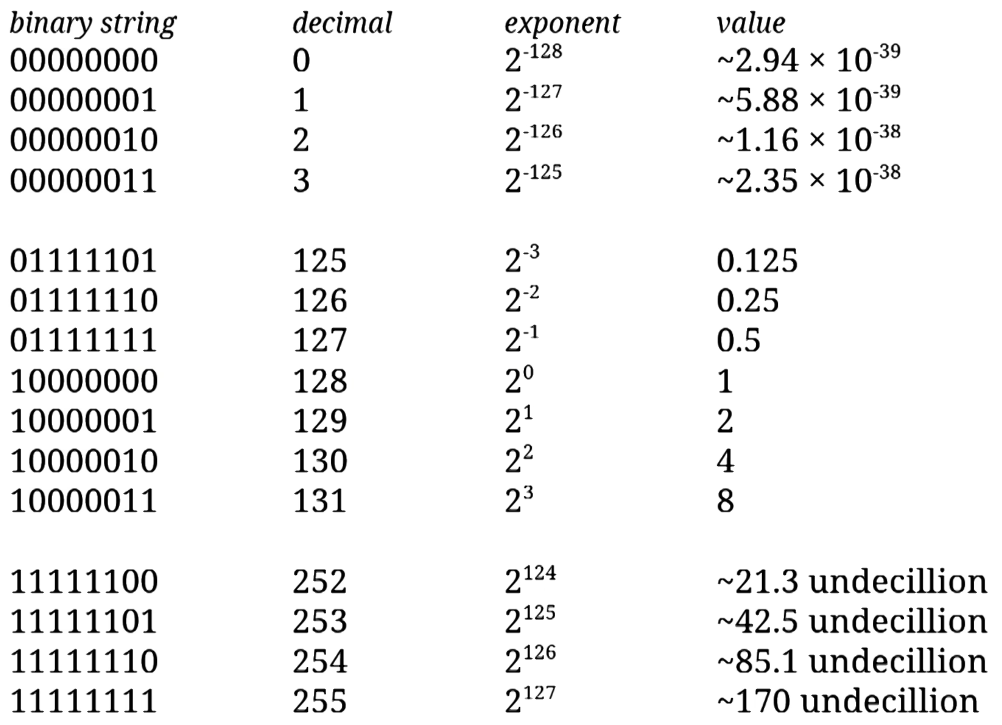
- Dec from Float: $(-1)^{[\text{Sign bit}]} \times (1 + [\text{Mantissa}]) \times 2^{([\text{Exponent Bits}] \text{ - } [\text{Bias}])}$
- Float from Dec: Convert to bin sci notation, 'sub

$$thebias$$

$$toget$$

  in' to floating exponent land
- Dec to F32:
- Zero: "00000000" exponent, all zero mantissa, sign as usual
- Inf: "11111111" exponent, all zero mantissa, sign as usual
- NANs: "11111111" exponent, sign and mantissa "left to implementer's discretion"
- Subnorms: "00000000" exponent, then replace the leading 1 with a zero and continue as usual. Getting increasingly smaller but less precise

Exponent chart from jan masali:



binary string	*decimal*	*exponent*	*value*
00000000	0	2^{-128}	$\sim 2.94 \times 10^{-39}$
00000001	1	2^{-127}	$\sim 5.88 \times 10^{-39}$
00000010	2	2^{-126}	$\sim 1.16 \times 10^{-38}$
00000011	3	2^{-125}	$\sim 2.35 \times 10^{-38}$
01111101	125	2^{-3}	0.125
01111110	126	2^{-2}	0.25
01111111	127	2^{-1}	0.5
10000000	128	2^{0}	1
10000001	129	2^{1}	2
10000010	130	2^{2}	4
10000011	131	2^{3}	8
11111100	252	2^{124}	~21.3 undecillion
11111101	253	2^{125}	~42.5 undecillion
11111110	254	2^{126}	~85.1 undecillion
11111111	255	2^{127}	~170 undecillion

Demorgans and Boolean algebra

- Xors are 1 if there's an even number of 1s, 0 if an even number. – Thus 'Parity'
- $\neg(A \vee B) = \neg(A) \wedge \neg(B)$
- $\neg(A \wedge B) = \neg(A) \vee \neg(B)$
- You can flip the operation and flip weather they're internally or externally negated.

4 FSM

Not that hard, just like, make sure all states and lines can be justified and are labeled.

5 Pipelining (oh god)

Usual steps of a 5 stage pipeline

- IF Instruction fetch 1 cycle, get next instruction from InstrMem or cache
- DR Data read from register
- AL ALU, does the computation
- DM Data memory:
- RW Read Write

Bypassing

- Point of production:

  add, sub, etc: end of ALU

  lw: end of DM – the mem it's reading into register

- Point of consumption:

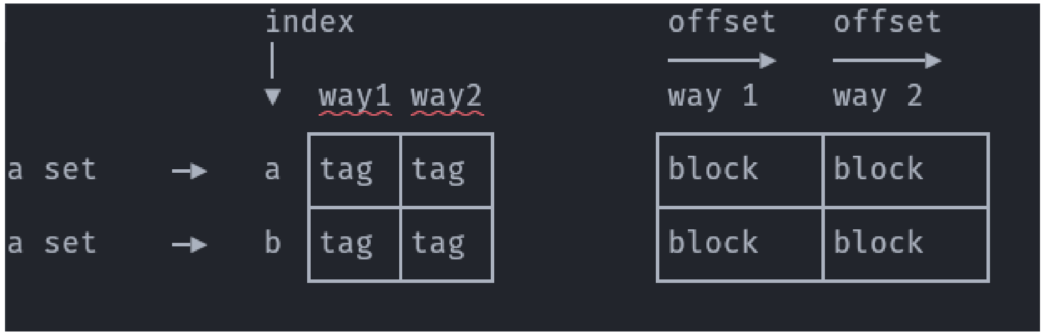
  add, sub, lw: start of ALU

  sw/lw $1, 8($2):

  start of ALU for $2

  start of DM for $1

6 Cache



- Offset and Index need the bits they need, tag gets the rest

  Offset is for within the block

  Index: which set we're referencing

  - Tag array size = sets * ways * tagSize
  - [Offset] = address%[block size in bytes]
  - [Index] = (address/[block size in bytes])% $\log_2$(Sets Count)
  - [Tag] = address/([block size in bytes] $*$ $\log_2$([Sets Count]))
  - Tag bits: remaining bits
  - Offset bits: depending on blocksize, determines what byte within the block is being referenced $\log_2(blocksize(B))Indexbits : determinedbyamountofsetslog_2(setcount)$

- Usually we write-allocate, briging a miss into cache. Read misses always bring block into cache

  Usually we evict the least-recently used block

- Bit string: tag index offset

7 Spectre/Meltdown

- "Spectre refers to a whole family of potential weaknesses of which meltdown is just one" - Computerphile video description
- These are the result of a combination of speculative execution and out of order execution
- Meltdown is a specific kernel memory exploit
- Need a: "lw $t1, 0(secret); lw $t2, $t1" series of two instructions to leave footprints in cache

8 Virtual Memory

- Memory virtualised by the kernel, the program thinks it has a single uninterrupted area to work with
- Programs have a pagetable, which is a list of how virtual pages are mapped to physical pages in memory
- Page table translations are mostly done via the Translation Lookaside Buffer, which falls back to the pagetable

9 Multiprocessor Design

- Each core has a cache, the LLC is shared
- TODO: Protocol descripions and examples of that chart

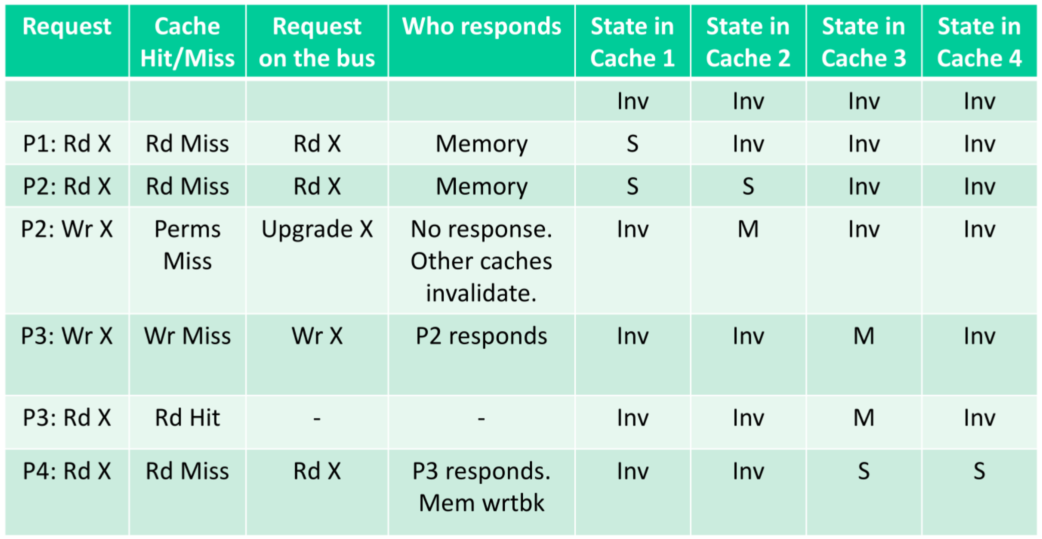
Cache Coherence Protocls

- Directory-based: A single location (directory) keeps track of the sharing status of a block of memory
- Snooping: Every cache block is accompanied by the sharing status of that block - all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary
- Write invalidate: a processor gains exclusive access of a block before writing by invalidating all other copies
- Write-update: when a process or writes, it updates other shared copies of that block

Locks

- Atomic exchange: simulatneous load and store operation, locks memory at the same moment it reads. – locks it while it transfers into register

MP exmaple



Request	Cache Hit/Miss	Request on the bus	Who responds	State in Cache 1	State in Cache 2	State in Cache 3	State in Cache 4
				Inv	Inv	Inv	Inv
P1: Rd X	Rd Miss	Rd X	Memory	S	Inv	Inv	Inv
P2: Rd X	Rd Miss	Rd X	Memory	S	S	Inv	Inv
P2: Wr X	Perms Miss	Upgrade X	No response. Other caches invalidate.	Inv	M	Inv	Inv
P3: Wr X	Wr Miss	Wr X	P2 responds	Inv	Inv	M	Inv
P3: Rd X	Rd Hit	-	-	Inv	Inv	M	Inv
P4: Rd X	Rd Miss	Rd X	P3 responds. Mem wrtbk	Inv	Inv	S	S

10 Basic structure of a GPU:

- many Single Instruction Multiple Thread cores, each with several warps and a warp scheduler. – large cache
- Very quick to abandon the current project if it stalls and start work on the next – minimal downtime, easy to context switch
- Each SIMT core has priavate L1 cache, large L2 shared by all cores. Each L2 bank services a subset of addresses
- The

11 Disk stuff:

- 1-12 platters (glass disks), 5-30k tracks (rings), 100-500 sectors, 512B/sector (circle around a track)
- Arm moving to correct track ⇒ seek time, 5-12ms, maybe less
- Rotational latency: time to rotate sector under head, usually 2ms
- Transfer time: time taken to transfer a block of bits out of disk, usually 3-65 MB/s
- Disk controller: maintains disk cache, sets up transfers.
- Mean time to Failure/Restore

  Reliability MTTF

  Availability: fraction of time service matches specs, MTTF / (MTTF + MTTR)

12 Raid types overview:

- Raid 0: No redundancy, stripes disks across drives to improve parallelism.
- Raid 1: Mirrors all disks, can sometimes increase read speed, highly redundant
- Raid 2: bit level striping, requires lockstep drives. Lots of parity drives
- Raid 3: byte level striping with dedicated parity disk. Highest sequential read speeds. Usually requires lockstep drives.
- Raid 4 and 5: block level striping, 4 has a dedicated parity disk, 5 distributes parity sections among drives.
- Raid 6: Same as raid 5, but has two parity blocks per stripe, again distributed among drives. Can work even after two drive failures. Technically raid 6 can have an arbitrary number of parity blocks added for increased redundancy.

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

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

BASIC INSTRUCTION FORMATS

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

ARITHMETIC CORE INSTRUCTION SET

②

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

FLOATING-POINT INSTRUCTION FORMATS

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]<=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	No

OPCODES, BASE CONVERSION, ASCII SYMBOLS

③

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Deci-mal	Hexa-deci-mal	ASCII Char-acter	Deci-mal	Hexa-deci-mal	ASCII Char-acter
(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

④

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

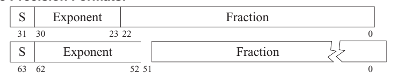
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE 754 Symbols

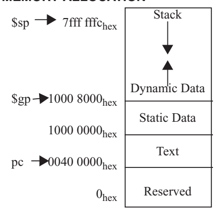
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

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

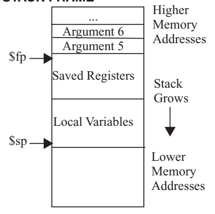
IEEE Single Precision and Double Precision Formats:



MEMORY ALLOCATION



STACK FRAME

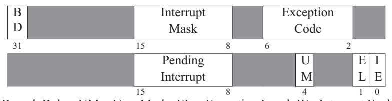


DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

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)

SI Size	Prefix	Symbol	IEC Size	Prefix	Symbol
10^{3}	Kilo-	K	2^{10}	Kibi-	Ki
10^{6}	Mega-	M	2^{20}	Mebi-	Mi
10^{9}	Giga-	G	2^{30}	Gibi-	Gi
10^{12}	Tera-	T	2^{40}	Tebi-	Ti
10^{15}	Peta-	P	2^{50}	Pebi-	Pi
10^{18}	Exa-	E	2^{60}	Exbi-	Ei
10^{21}	Zetta-	Z	2^{70}	Zebi-	Zi
10^{24}	Yotta-	Y	2^{80}	Yobi-	Yi

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