Name ________________________________

Do Not Open The Test Until Told To Do So

MIPS Reference Data ①

CORE INSTRUCTION SET

NAME	MNE-MON-IC	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 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 Unsigned	`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 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.)

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	MNE-MON-IC	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
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

PSEUDO INSTRUCTION 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

Midterm Exam – CSE378 Autumn 2009 Snyder

This is closed book, closed notes, closed calculator and closed neighbor test.

1. [3 points] If x = 0011 1010 1010 1001 0101 0011 1011 1100 as a binary integer, what is –x as a binary integer?

2. [2 points] The x in Question1 is what hexadecimal number?

3. [2 points] When the x of Question 1 is interpreted as a single precision floating point number, what are the bit values for its
   sign ____, exponent ___________________, fraction ______________________

4. [2 points] What number is the zero for single precision floating point exponents?

5. [4 points] Express the 0 of an n-bit bias-represented number as an expression.

6. [5 points] Using the "green card", translate the following bit sequence interpreted as MIPS machine code into assembly language – be sure to include the correct register names, addresses, immediate values, etc.

0000 0011 1110 0000 0000 0000 0000 1000 ________________________

7. [6 points] Register $t0 contains an arbitrary bit sequence; write three assembly instructions to set the "even bits" to 0 and to keep the odd bits. (Bit 0 is even.)

8. [8 points] MIPS hardware does not directly implement the pseudo-instruction:
   abs $result, $source

for storing the absolute value of register $source into register $result. Show the kind of MIPS code the assembler might create for this instruction.

9. [5 points] If we try the experiment of executing

```
lb $t0, 0($t1)
lh $t0, 0($t1)
lw $t0, 0($t1)
```

and discover that only two of the three instructions is legal, what do we know about the value in $t1?

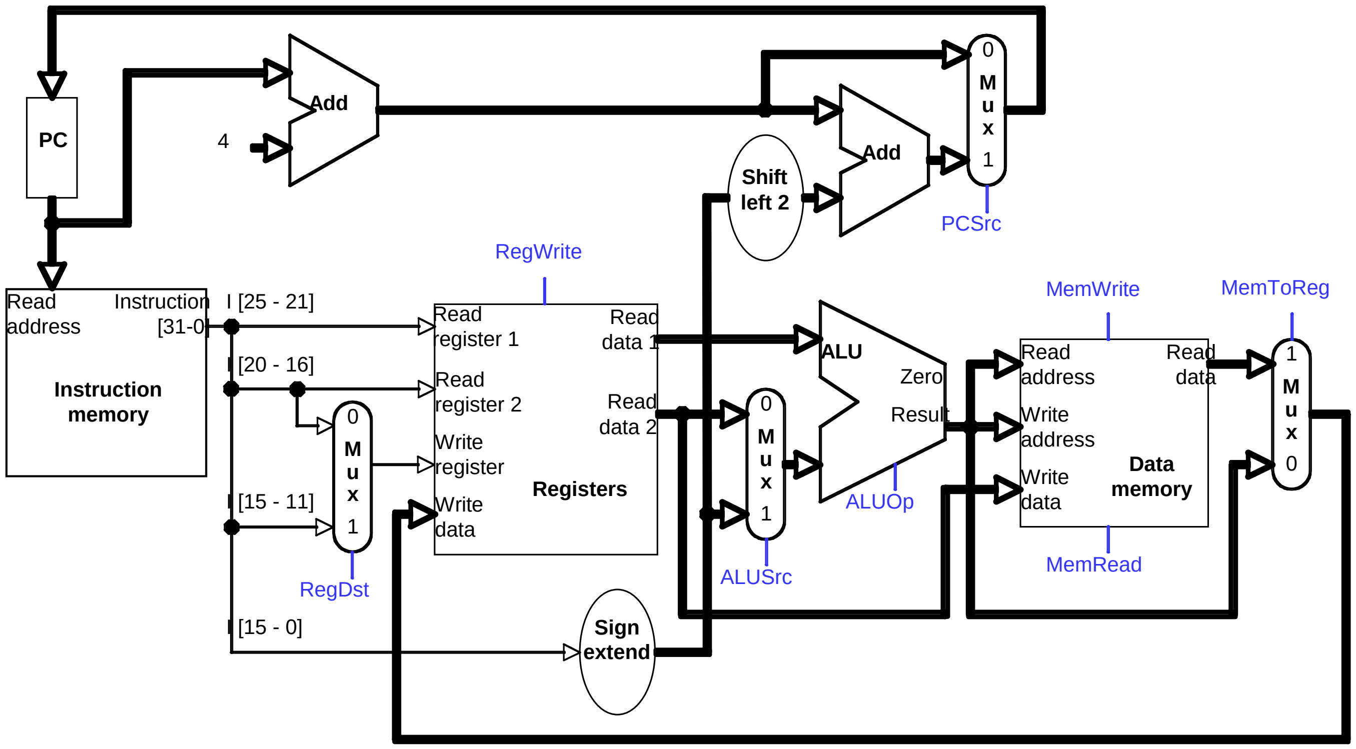
10. [25 Points] The following “move to front” C code moves A[n] of an integer array A to A[0], (n < m, the size of the array.)

```
t = A[n];                // Remove A[n]
for (i=n; 0<i; i--) {    // Start at hi, go to lo
    A[i] = A[i-1];       // Shift item “right”
}
A[0] = t;                // Refill A[0]
```

Write this operation in assembly language assuming the base address of A is in $a0 and the value of the array index n is in $a1. Comment profusely.
(Hint: Focus on getting the code correct, rather than finding the most clever solution.)

11. [8 points] Suppose the top element on the stack is the integer i in a C function. Show assembly code to push i-- onto the stack.

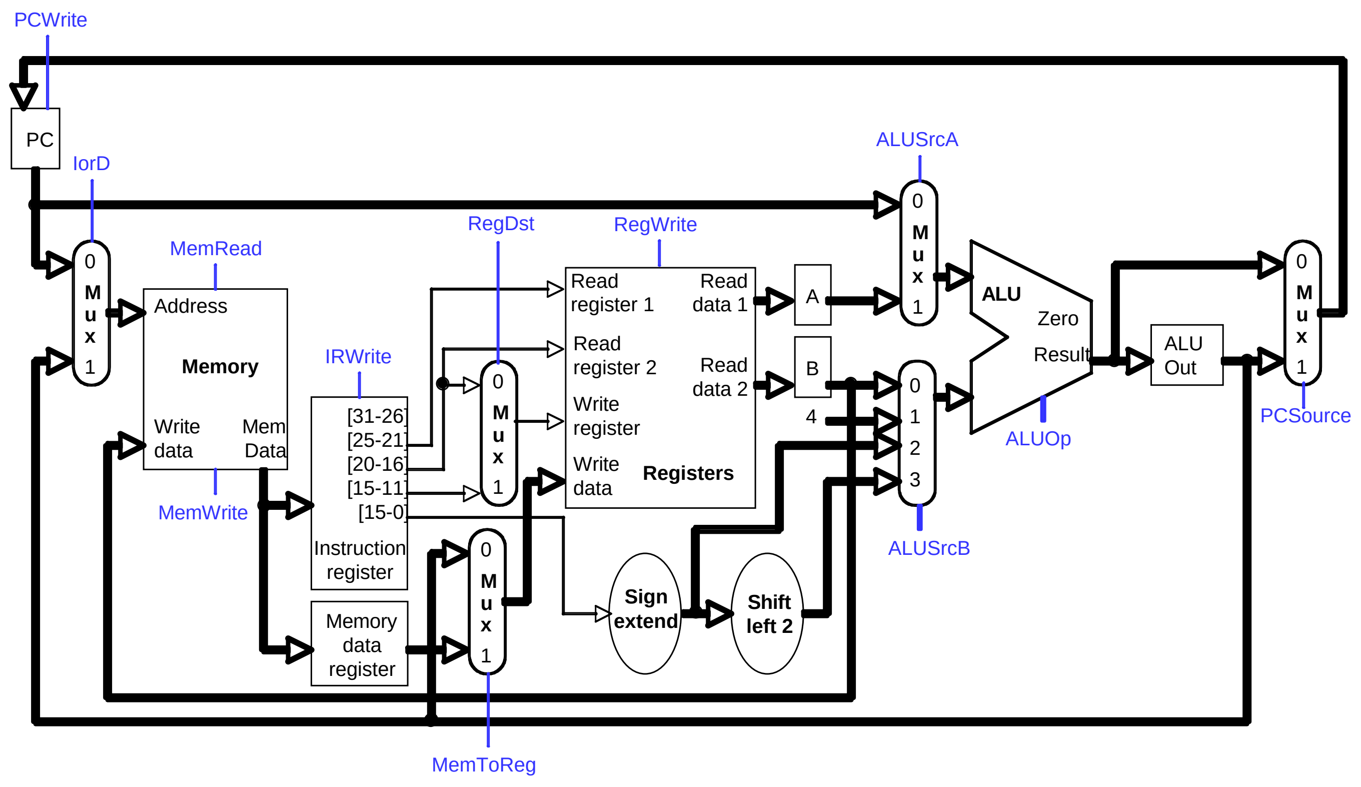
12. [5 points] List the opcodes (of the core instruction set, i.e. no pseudos) for all of the instructions that use the "Shift left 2" circuit of the following diagram.



13. [8 points] In the diagram above, highlight in color those portions of the circuit that are *active when computing store for the store word instruction*. (Note that other portions will be active in this single cycle implementation; mark *only those portions that contribute to the operation of the instruction*, including control.)

14. [5 points] Give the control lines and their settings required to implement the single cycle store instruction above.

15. [8 points] In the accompanying diagram mark in color those portions of the circuit active during the third cycle of a store instruction in this multicycle processor design. (Hint: Think of what you know from cycle 3.)



16. [4 points] If ALU operations take 2 ns, register reads/writes take 1ns, and memory operations each take (thanks to a miracle!) 10ns,

a. How long is the single cycle design's clock cycle?

b. How long is the multicycle design's clock cycle?

Show your work.

③

OPCODES, BASE CONVERSION, ASCII SYMBOLS

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)

④

IEEE 754 FLOATING POINT STANDARD

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

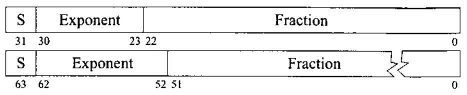
where Single Precision Bias = 127,
Double Precision Bias = 1023.

IEEE 754 Symbols

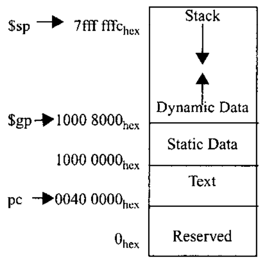
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

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

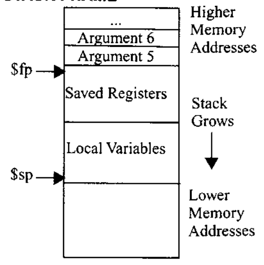
IEEE Single Precision and Double Precision Formats:



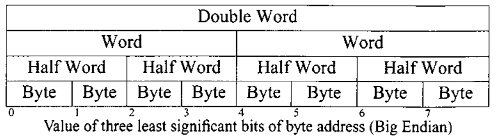
MEMORY ALLOCATION



STACK FRAME

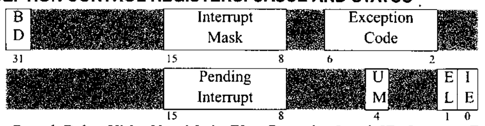


DATA ALIGNMENT



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	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX
$10^3, 2^{10}$	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 μ is used for micro.