Section 1: MIPS Fundamentals (30 marks)

Questions (1)-(2) are Multiple Choice Questions. Identify all correct answers.

(1) Which of the following instruction(s) is/are not a MIPS I-type instruction(s)? (3)

- (a) `sll $t1, $t0, 3`
- (b) `addi $t1, $t0, 3`
- (c) `jal label`
- (d) `bne $t1, $t0, label`
- (e) `slt $t1, $t0, $0`

(2) In the MIPS processor, which execution stage(s) below is/are necessary for all instructions? (3)

- (a) IF
- (b) ID
- (c) EX
- (d) MEM
- (e) WB

Questions (3)-(6) are Short Answer Questions. Provide your answers concisely, with clear steps of calculation if necessary.

(3) Given what you understand by the MIPS instruction format, do you think the current MIPS ISA (32-bit) can support a total of 200 different R-, I-, and J-type instructions? (1) Give a [yes/no/others] answer, and (2) Provide your brief explanation, with calculations if necessary. (4)

(4) What does the MIPS register $ra hold? Which scenario is it used in MIPS programming? How is it typically handled in that scenario? Name the related MIPS instructions. (6)

Section 1: MIPS Fundamentals (30 marks)

Questions (1)-(2) are Multiple Choice Questions. Identify all correct answers.

(1) Which of the following instruction(s) is/are not a MIPS I-type instruction(s)? (3)

(a) `sll $t1, $t0, 3`

(b) `addi $t1, $t0, 3`

(c) `jal label`

(d) `bne $t1, $t0, label`

(e) `slt $t1, $t0, $0`

(2) In the MIPS processor, which execution stage(s) below is/are necessary for all instructions? (3)

(a) IF

(b) ID

(c) EX

(d) MEM

(e) WB

Questions (3)-(6) are Short Answer Questions. Provide your answers concisely, with clear steps of calculation if necessary.

(3) Given what you understand by the MIPS instruction format, do you think the current MIPS ISA (32-bit) can support a total of 200 different R-, I-, and J-type instructions? (1) Give a [yes/no/others] answer, and (2) Provide your brief explanation, with calculations if necessary. (4)

(4) What does the MIPS register $ra hold? Which scenario is it used in MIPS programming? How is it typically handled in that scenario? Name the related MIPS instructions. (6)

(5) Given the following bit pattern as a single precision floating point representation, what decimal number does it represent? 1011 1111 1111 0000 0000 0000 0000 0000: (6)

(6) Suppose that you have a Computer A running a program named ALPHA with the following mix of instructions: (8)

Operation	**Percentage**	**CPI**
ALU Ops	35%	1
Loads	25%	2
Stores	15%	4
Branches	25%	3

(a) Compute the average CPI of Computer A.

(b) With a different compiler, the same program ALPHA is compiled for another Computer B, whose CPU is the same as Computer A. It produces equal number of 'load' and 'store' instructions, while the number of ALU ops and Branch instructions remain unchanged. The total number of instructions remain unchanged. The CPU clock frequency of Computer B is 20% higher than that of Computer A. Which computer runs ALPHA faster? And by how much percent?

Section 2: MIPS Programming (12 marks)

Convert the following high-level code into MIPS assembly. Use \$s0, \$t1 and \$t2 to store variables 'i', 'a', and 'b', respectively. Besides others, the instructions that you must use in your code are: {`slti, bne, ori`}. Note that pseudo instructions are not allowed to be used. (12)

```
int i = 14;
int a = 0x10010000, b = 5;

while (i % b < 1) {
    a = (int) (a / b);
    i--;
}
```

Section 3: CPU design (28 marks)

(1) Refer to the following MIPS assembly code and answer the questions below: (12)

```
  addi $s0, $0, 0
  addi $t0, $0, 3
for:
  beq  $s0, $t0, done
  lw   $s1, 4($s3)
  add  $s3, $s1, $s0
  sub  $t8, $t1, $t6
  or   $t6, $t3, $t1
  addi $s0, $s0, 1
  j    for
done:
```

(a) Find the number of executed instructions of the above code for each of the following types:
R-type =
I-type =
Loads =
Unconditional jumps =
Conditional jumps =

(b) Assume the code is running on a single-cycle CPU, calculate the total number of execution cycles.

(c) Assume the code is running on a 5-stage pipelined CPU, what is the total number of cycles to execute the above code? Assume that the CPU can do forwarding, stalling, and early branching.

(2) Design a single-cycled CPU for an instruction set that contains only the following three instructions: 'lw', 'add', and 'nand'. Assume that the instruction formats are the same as in the MIPS architecture. Draw the CPU design, by showing all the hardware functional blocks and all the data links. No control path needs to be shown. Specifically, draw the block diagram of a 1-bit ALU design that supports the abovementioned instructions, You must show only the minimal hardware required to implement the CPU and ALU. (16)

Section 4: Computer Networks (30 marks)

Questions (1)-(6) are Single Choice Questions. **(12)**

(1) In the OSI model, which layer deals with bit-level transmission?

- (a) Network Layer.
- (b) Transportation Layer.
- (c) Physical Layer.
- (d) Data Link Layer.

(2) The vase network of computers that connects millions of people all over the world is called:

- (a) Internet.
- (b) Hypertext.
- (c) LAN.
- (d) Web.

(3) Bluetooth is an example of:

- (a) Wide area network.
- (b) Virtual private network.
- (c) Local area network.
- (d) Personal area network.

(4) Which of the following protocol is used for remote terminal connection service?

- (a) RARP.
- (b) TELNET.
- (c) UDP.
- (d) FTP.

(5) A device which can be connected to a network without using cable is called:

- (a) Open-source device.

(b) Centralized device.
(c) Distributed device.
(d) Wireless device.

(6) Which device can read data and convert them to a form that a computer can use?

(a) Logic device.
(b) Control device.
(c) Input device.
(d) Storage device.

Questions (7)-(12) are Multiple Choice Questions. Identify all correct answers. Each incorrect answer causes mark deduction. (18)

(7) Which of the following OSI layers correspond(s) to TCP/IP's application layer?

(a) Transport Layer.
(b) Session Layer.
(c) Application Layer.
(d) Network Layer.
(e) Presentation Layer.

(8) Which of the following is/are correct?

(a) Web cache contains copies of recently requested objects.
(b) Web cache might reduce the response time.
(c) Web cache does not have its own disk space.
(d) Web cache can act like server.
(e) Web cache can act like client.

(9) Which of the following operations can be performed using FTP?

(a) Connect to a remote host.
(b) Automatic allocation.

(c) Define the transfer mode.
(d) Select directory.
(e) List the files available.

(10) Which of the following are the solutions to network security?

(a) Authentication.
(b) Authorization.
(c) Check.
(d) Encryption.
(e) Encapsulation

(11) Match the following HTTP status code to their respective definitions.

i) 100	ii) 200	iii) 404	iv) 504
a) OK	b) Continue	c) Gateway Timeout	d) Not Found

(a) i-a, ii-b, iii-d, iv-c.
(b) i-b, ii-a, iii-d, iv-c.
(c) i-b, ii-a, iii-c, iv-d.
(d) i-c, ii-b, iii-d, iv-a.
(e) i-c, ii-b, iii-a, iv-d.

(12) Match the following IEEE Numbers to their corresponding Names for IEEE 802 standards for LANs..

i) 802.16	ii) 802.15.1	iii) 802.11	iv) 802.3
a) Ethernet	b) WiFi	c) WiMa	d) Bluetooth

(a) i-a, ii-d, iii-b, iv-c.
(b) i-b, ii-a, iii-d, iv-c.
(c) i-c, ii-d, iii-b, iv-a.
(d) i-c, ii-a, iii-d, iv-b.
(e) i-d, ii-c, iii-a, iv-b.

MIPS/SPIM Reference Card

CORE INSTRUCTION SET (INCLUDING PSEUDO INSTRUCTIONS)

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add	`add`	R	R[rd]=R[rs]+R[rt]	(1)	0/20
Add Immediate	`addi`	I	R[rt]=R[rs]+SignExtImm	(1)(2)	8
Add Imm. Unsigned	`addiu`	I	R[rt]=R[rs]+SignExtImm	(2)	9
Add Unsigned	`addu`	R	R[rd]=R[rs]+R[rt]	(2)	0/21
Subtract	`sub`	R	R[rd]=R[rs]-R[rt]	(1)	0/22
Subtract Unsigned	`subu`	R	R[rd]=R[rs]-R[rt]		0/23
And	`and`	R	R[rd]=R[rs]&R[rt]		0/24
And Immediate	`andi`	I	R[rt]=R[rs]&ZeroExtImm	(3)	c
Nor	`nor`	R	R[rd]=∼(R[rs]|R[rt])		0/27
Or	`or`	R	R[rd]=R[rs]|R[rt]		0/25
Or Immediate	`ori`	I	R[rt]=R[rs]|ZeroExtImm	(3)	d
Xor	`xor`	R	R[rd]=R[rs]ˆR[rt]		0/26
Xor Immediate	`xori`	I	R[rt]=R[rs]ˆZeroExtImm		e
Shift Left Logical	`sll`	R	R[rd]=R[rs]≪shamt		0/00
Shift Right Logical	`srl`	R	R[rd]=R[rs]≫shamt		0/02
Shift Right Arithmetic	`sra`	R	R[rd]=R[rs]≫>shamt		0/03
Shift Left Logical Var.	`sllv`	R	R[rd]=R[rs]≪R[rt]		0/04
Shift Right Logical Var.	`srlv`	R	R[rd]=R[rs]≫R[rt]		0/06
Shift Right Arithmetic Var.	`srav`	R	R[rd]=R[rs]≫>R[rt]		0/07
Set Less Than	`slt`	R	R[rd]=(R[rs]<R[rt])?1:0		0/2a
Set Less Than Imm.	`slti`	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)	a
Set Less Than Imm. Unsign.	`sltiu`	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)(6)	b
Set Less Than Unsigned	`sltu`	R	R[rd]=(R[rs]<R[rt])?1:0	(6)	0/2b
Branch On Equal	`beq`	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4
Branch On Not Equal	`bne`	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5
Branch Less Than	`blt`	P	if(R[rs]<R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than	`bgt`	P	if(R[rs]>R[rt]) PC=PC+4+BranchAddr		
Branch Less Than Or Equal	`ble`	P	if(R[rs]<=R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than Or Equal	`bge`	P	if(R[rs]>=R[rt]) PC=PC+4+BranchAddr		
Jump	`j`	J	PC=JumpAddr	(5)	2
Jump And Link	`jal`	J	R[31]=PC+4; PC=JumpAddr	(5)	2
Jump Register	`jr`	R	PC=R[rs]		0/08
Jump And Link Register	`jalr`	R	R[31]=PC+4; PC=R[rs]		0/09
Move	`move`	P	R[rd]=R[rs]		
Load Byte	`lb`	I	R[rt]={24’b0, M[R[rs]+ZeroExtImm](7:0)}	(3)	20
Load Byte Unsigned	`lbu`	I	R[rt]={24’b0, M[R[rs]+SignExtImm](7:0)}	(2)	24
Load Halfword	`lh`	I	R[rt]={16’b0, M[R[rs]+ZeroExtImm](15:0)}	(3)	25
Load Halfword Unsigned	`lhu`	I	R[rt]={16’b0, M[R[rs]+SignExtImm](15:0)}	(2)	25
Load Upper Imm.	`lui`	I	R[rt]={imm,16’b0}		f
Load Word	`lw`	I	R[rt]=M[R[rs]+SignExtImm]	(2)	23
Load Immediate	`li`	P	R[rd]=immediate		
Load Address	`la`	P	R[rd]=immediate		
Store Byte	`sb`	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)	(2)	28
Store Halfword	`sh`	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)	(2)	29
Store Word	`sw`	I	M[R[rs]+SignExtImm]=R[rt]	(2)	2b

REGISTERS

NAME	NMBR	USE	STORE?
$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
$f0-$f31	0-31	Floating Point Registers	Yes

(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 }
(4) JumpAddr = {PC[31:28], address, 2’b0 }
(6) Operands considered unsigned numbers (vs. 2 s comp.)

BASIC INSTRUCTION FORMATS, FLOATING POINT INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate (15–0)		
J	opcode	immediate (25–0)				
FR	opcode	fmt	ft	fs	fd	funct
FI	opcode	fmt	rt	immediate (15–0)		

ARITHMETIC CORE INSTRUCTION SET

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FMT/FT/ FUNCT
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
Multiply	`mult`	R	{Hi,Lo}=R[rs]∗R[rt]		0/–/–/18
Multiply Unsigned	`multu`	R	{Hi,Lo}=R[rs]∗R[rt]	(6)	0/–/–/19
Branch On FP True	`bc1t`	FI	if(FPCond) PC=PC+4+BranchAddr	(4)	11/8/1/–
Branch On FP False	`bc1f`	FR	if(!FPCond) PC=PC+4+BranchAddr	(4)	11/8/0/–
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 Add Single	`add.s`	FR	F[fd]=F[fs]+F[ft]		11/10/–/0
FP Divide Single	`div.s`	FR	F[fd]=F[fs]/F[ft]		11/10/–/3
FP Multiply Single	`mul.s`	FR	F[fd]=F[fs]∗F[ft]		11/10/–/2
FP Subtract Single	`sub.s`	FR	F[fd]=F[fs]-F[ft]		11/10/–/1
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 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 Double	`mul.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}∗{F[ft],F[ft+1]}		11/11/–/2
FP Subtract Double	`sub.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}-{F[ft],F[ft+1]}		11/11/–/1
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]		16/0/–/0
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/–/–/–
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/–/–/–

ASSEMBLER DIRECTIVES

`.data` [*addr*]*	Subsequent items are stored in the data segment
`.kdata` [*addr*]*	Subsequent items are stored in the kernel data segment
`.ktext` [*addr*]*	Subsequent items are stored in the kernel text segment
`.text` [*addr*]*	Subsequent items are stored in the text
	* starting at [*addr*] if specified
`.ascii` *str*	Store string *str* in memory, but do not null-terminate it
`.asciiz` *str*	Store string *str* in memory and null-terminate it
`.byte` $b_1, \ldots, b_n$	Store the n values in successive bytes of memory
`.double` $d_1, \ldots, d_n$	Store the n floating-point double precision numbers in successive memory locations
`.float` $f_1, \ldots, f_1$	Store the n floating-point single precision numbers in successive memory locations
`.half` $h_1, \ldots, h_n$	Store the n 16-bit quantities in successive memory halfwords
`.word` $w_1, \ldots, w_n$	Store the n 32-bit quantities in successive memory words
`.space` n	Allocate n bytes of space in the current segment
`.extern` *symsize*	Declare that the datum stored at *sym* is *size* bytes large and is a global label
`.globl` *sym*	Declare that label *sym* is global and can be referenced from other files
`.align` n	Align the next datum on a 2^n byte boundary, until the next `.data` or `.kdata` directive
`.set at`	Tells SPIM to complain if subsequent instructions use `$at`
`.set noat`	prevents SPIM from complaining if subsequent instructions use `$at`

SYSCALLS

SERVICE	`$v0`	ARGS	RESULT
`print_int`	1	integer `$a0`	
`print_float`	2	float `$f12`	
`print_double`	3	double `$f12/$f13`	
`print_string`	4	string `$a0`	
`read_int`	5		integer (in `$v0`)
`read_float`	6		float (in `$f0`)
`read_double`	7		double (in `$f0`)
`read_string`	8	buf `$a0`, buflen `$a1`	
`sbrk`	9	amount `$a`	address (in `$v0`)
`exit`	10		

EXCEPTION CODES

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

[1] Patterson, David A; Hennessy, John J.: Computer Organization and Design, 3rd Edition. Morgan Kaufmann Publishers. San Francisco, 2005.

(5) Given the following bit pattern as a single precision floating point representation, what decimal number does it represent? 1011 1111 1111 0000 0000 0000 0000 0000: (6)

(6) Suppose that you have a Computer A running a program named ALPHA with the following mix of instructions: (8)

Operation	**Percentage**	**CPI**
ALU Ops	35%	1
Loads	25%	2
Stores	15%	4
Branches	25%	3

(a) Compute the average CPI of Computer A.

(b) With a different compiler, the same program ALPHA is compiled for another Computer B, whose CPU is the same as Computer A. It produces equal number of 'load' and 'store' instructions, while the number of ALU ops and Branch instructions remain unchanged. The total number of instructions remain unchanged. The CPU clock frequency of Computer B is 20% higher than that of Computer A. Which computer runs ALPHA faster? And by how much percent?

Section 2: MIPS Programming (12 marks)

Convert the following high-level code into MIPS assembly. Use \$s0, \$t1 and \$t2 to store variables 'i', 'a', and 'b', respectively. Besides others, the instructions that you must use in your code are: {`slti, bne, ori`}. Note that pseudo instructions are not allowed to be used. (12)

```
int i = 14;
int a = 0x10010000, b = 5;

while (i % b < 1) {
    a = (int) (a / b);
    i--;
}
```

Section 3: CPU design (28 marks)

(1) Refer to the following MIPS assembly code and answer the questions below: (12)

```
  addi $s0, $0, 0
  addi $t0, $0, 3
for:
  beq  $s0, $t0, done
  lw   $s1, 4($s3)
  add  $s3, $s1, $s0
  sub  $t8, $t1, $t6
  or   $t6, $t3, $t1
  addi $s0, $s0, 1
  j    for
done:
```

(a) Find the number of executed instructions of the above code for each of the following types:
R-type =
I-type =
Loads =
Unconditional jumps =
Conditional jumps =

(b) Assume the code is running on a single-cycle CPU, calculate the total number of execution cycles.

(c) Assume the code is running on a 5-stage pipelined CPU, what is the total number of cycles to execute the above code? Assume that the CPU can do forwarding, stalling, and early branching.

(2) Design a single-cycled CPU for an instruction set that contains only the following three instructions: 'lw', 'add', and 'nand'. Assume that the instruction formats are the same as in the MIPS architecture. Draw the CPU design, by showing all the hardware functional blocks and all the data links. No control path needs to be shown. Specifically, draw the block diagram of a 1-bit ALU design that supports the abovementioned instructions, You must show only the minimal hardware required to implement the CPU and ALU. (16)

Section 4: Computer Networks (30 marks)

Questions (1)-(6) are Single Choice Questions. **(12)**

(1) In the OSI model, which layer deals with bit-level transmission?

- (a) Network Layer.
- (b) Transportation Layer.
- (c) Physical Layer.
- (d) Data Link Layer.

(2) The vase network of computers that connects millions of people all over the world is called:

- (a) Internet.
- (b) Hypertext.
- (c) LAN.
- (d) Web.

(3) Bluetooth is an example of:

- (a) Wide area network.
- (b) Virtual private network.
- (c) Local area network.
- (d) Personal area network.

(4) Which of the following protocol is used for remote terminal connection service?

- (a) RARP.
- (b) TELNET.
- (c) UDP.
- (d) FTP.

(5) A device which can be connected to a network without using cable is called:

- (a) Open-source device.

(b) Centralized device.
(c) Distributed device.
(d) Wireless device.

(6) Which device can read data and convert them to a form that a computer can use?

(a) Logic device.
(b) Control device.
(c) Input device.
(d) Storage device.

Questions (7)-(12) are Multiple Choice Questions. Identify all correct answers. Each incorrect answer causes mark deduction. (18)

(7) Which of the following OSI layers correspond(s) to TCP/IP's application layer?

(a) Transport Layer.
(b) Session Layer.
(c) Application Layer.
(d) Network Layer.
(e) Presentation Layer.

(8) Which of the following is/are correct?

(a) Web cache contains copies of recently requested objects.
(b) Web cache might reduce the response time.
(c) Web cache does not have its own disk space.
(d) Web cache can act like server.
(e) Web cache can act like client.

(9) Which of the following operations can be performed using FTP?

(a) Connect to a remote host.
(b) Automatic allocation.

(c) Define the transfer mode.
(d) Select directory.
(e) List the files available.

(10) Which of the following are the solutions to network security?

(a) Authentication.
(b) Authorization.
(c) Check.
(d) Encryption.
(e) Encapsulation

(11) Match the following HTTP status code to their respective definitions.

i) 100	ii) 200	iii) 404	iv) 504
a) OK	b) Continue	c) Gateway Timeout	d) Not Found

(a) i-a, ii-b, iii-d, iv-c.
(b) i-b, ii-a, iii-d, iv-c.
(c) i-b, ii-a, iii-c, iv-d.
(d) i-c, ii-b, iii-d, iv-a.
(e) i-c, ii-b, iii-a, iv-d.

(12) Match the following IEEE Numbers to their corresponding Names for IEEE 802 standards for LANs..

i) 802.16	ii) 802.15.1	iii) 802.11	iv) 802.3
a) Ethernet	b) WiFi	c) WiMa	d) Bluetooth

(a) i-a, ii-d, iii-b, iv-c.
(b) i-b, ii-a, iii-d, iv-c.
(c) i-c, ii-d, iii-b, iv-a.
(d) i-c, ii-a, iii-d, iv-b.
(e) i-d, ii-c, iii-a, iv-b.

MIPS/SPIM Reference Card

CORE INSTRUCTION SET (INCLUDING PSEUDO INSTRUCTIONS)

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add	add	R	R[rd]=R[rs]+R[rt]	(1)	0/20
Add Immediate	addi	I	R[rt]=R[rs]+SignExtImm	(1)(2)	8
Add Imm. Unsigned	addiu	I	R[rt]=R[rs]+SignExtImm	(2)	9
Add Unsigned	addu	R	R[rd]=R[rs]+R[rt]	(2)	0/21
Subtract	sub	R	R[rd]=R[rs]-R[rt]	(1)	0/22
Subtract Unsigned	subu	R	R[rd]=R[rs]-R[rt]		0/23
And	and	R	R[rd]=R[rs]&R[rt]		0/24
And Immediate	andi	I	R[rt]=R[rs]&ZeroExtImm	(3)	c
Nor	nor	R	R[rd]=∼(R[rs]|R[rt])		0/27
Or	or	R	R[rd]=R[rs]|R[rt]		0/25
Or Immediate	ori	I	R[rt]=R[rs]|ZeroExtImm	(3)	d
Xor	xor	R	R[rd]=R[rs]ˆR[rt]		0/26
Xor Immediate	xori	I	R[rt]=R[rs]ˆZeroExtImm		e
Shift Left Logical	sll	R	R[rd]=R[rs]≪shamt		0/00
Shift Right Logical	srl	R	R[rd]=R[rs]≫shamt		0/02
Shift Right Arithmetic	sra	R	R[rd]=R[rs]≫>shamt		0/03
Shift Left Logical Var.	sllv	R	R[rd]=R[rs]≪R[rt]		0/04
Shift Right Logical Var.	srlv	R	R[rd]=R[rs]≫R[rt]		0/06
Shift Right Arithmetic Var.	srav	R	R[rd]=R[rs]≫>R[rt]		0/07
Set Less Than	slt	R	R[rd]=(R[rs]<R[rt])?1:0		0/2a
Set Less Than Imm.	slti	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)	a
Set Less Than Imm. Unsign.	sltiu	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)(6)	b
Set Less Than Unsigned	sltu	R	R[rd]=(R[rs]<R[rt])?1:0	(6)	0/2b
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5
Branch Less Than	blt	P	if(R[rs]<R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than	bgt	P	if(R[rs]>R[rt]) PC=PC+4+BranchAddr		
Branch Less Than Or Equal	ble	P	if(R[rs]<=R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than Or Equal	bge	P	if(R[rs]>=R[rt]) PC=PC+4+BranchAddr		
Jump	j	J	PC=JumpAddr	(5)	2
Jump And Link	jal	J	R[31]=PC+4; PC=JumpAddr	(5)	2
Jump Register	jr	R	PC=R[rs]		0/08
Jump And Link Register	jalr	R	R[31]=PC+4; PC=R[rs]		0/09
Move	move	P	R[rd]=R[rs]		
Load Byte	lb	I	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}	(3)	20
Load Byte Unsigned	lbu	I	R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}	(2)	24
Load Halfword	lh	I	R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}	(3)	25
Load Halfword Unsigned	lhu	I	R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}	(2)	25
Load Upper Imm.	lui	I	R[rt]={imm,16'b0}		f
Load Word	lw	I	R[rt]=M[R[rs]+SignExtImm]	(2)	23
Load Immediate	li	P	R[rd]=immediate		
Load Address	la	P	R[rd]=immediate		
Store Byte	sb	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)	(2)	28
Store Halfword	sh	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)	(2)	29
Store Word	sw	I	M[R[rs]+SignExtImm]=R[rt]	(2)	2b

REGISTERS

NAME	NMBR	USE	STORE?
$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
$f0-$f31	0-31	Floating Point Registers	Yes

(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 }
(4) JumpAddr = {PC[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2 s comp.)

BASIC INSTRUCTION FORMATS, FLOATING POINT INSTRUCTION FORMATS

R	31 opcode 26	25 rs 21	20 rt 16	15 rd 11	10 shamt 6	5 funct 0
I	31 opcode 26	25 rs 21	20 rt 16	15 immediate 0		
J	31 opcode 26	25 immediate 0				
FR	31 opcode 26	25 fmt 21	20 ft 16	15 fs 11	10 fd 6	5 funct 0
FI	31 opcode 26	25 fmt 21	20 rt 16	15 immediate 0		

ARITHMETIC CORE INSTRUCTION SET

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FMT/FT/ FUNCT
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
Multiply	`mult`	R	{Hi,Lo}=R[rs]∗R[rt]		0/–/–/18
Multiply Unsigned	`multu`	R	{Hi,Lo}=R[rs]∗R[rt]	(6)	0/–/–/19
Branch On FP True	`bc1t`	FI	if(FPCond) PC=PC+4+BranchAddr	(4)	11/8/1/–
Branch On FP False	`bc1f`	FR	if(!FPCond) PC=PC+4+BranchAddr	(4)	11/8/0/–
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 Add Single	`add.s`	FR	F[fd]=F[fs]+F[ft]		11/10/–/0
FP Divide Single	`div.s`	FR	F[fd]=F[fs]/F[ft]		11/10/–/3
FP Multiply Single	`mul.s`	FR	F[fd]=F[fs]∗F[ft]		11/10/–/2
FP Subtract Single	`sub.s`	FR	F[fd]=F[fs]-F[ft]		11/10/–/1
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 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 Double	`mul.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}∗{F[ft],F[ft+1]}		11/11/–/2
FP Subtract Double	`sub.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}-{F[ft],F[ft+1]}		11/11/–/1
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]		16/0/–/0
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/–/–/–
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/–/–/–

ASSEMBLER DIRECTIVES

`.data` [*addr*]*	Subsequent items are stored in the data segment
`.kdata` [*addr*]*	Subsequent items are stored in the kernel data segment
`.ktext` [*addr*]*	Subsequent items are stored in the kernel text segment
`.text` [*addr*]*	Subsequent items are stored in the text
	* starting at [*addr*] if specified
`.ascii` *str*	Store string *str* in memory, but do not null-terminate it
`.asciiz` *str*	Store string *str* in memory and null-terminate it
`.byte` $b_1, \ldots, b_n$	Store the n values in successive bytes of memory
`.double` $d_1, \ldots, d_n$	Store the n floating-point double precision numbers in successive memory locations
`.float` $f_1, \ldots, f_1$	Store the n floating-point single precision numbers in successive memory locations
`.half` $h_1, \ldots, h_n$	Store the n 16-bit quantities in successive memory halfwords
`.word` $w_1, \ldots, w_n$	Store the n 32-bit quantities in successive memory words
`.space` n	Allocate n bytes of space in the current segment
`.extern` *symsize*	Declare that the datum stored at *sym* is *size* bytes large and is a global label
`.globl` *sym*	Declare that label *sym* is global and can be referenced from other files
`.align` n	Align the next datum on a 2^n byte boundary, until the next `.data` or `.kdata` directive
`.set at`	Tells SPIM to complain if subsequent instructions use `$at`
`.set noat`	prevents SPIM from complaining if subsequent instructions use `$at`

SYSCALLS

SERVICE	`$v0`	ARGS	RESULT
`print_int`	1	integer `$a0`	
`print_float`	2	float `$f12`	
`print_double`	3	double `$f12/$f13`	
`print_string`	4	string `$a0`	
`read_int`	5		integer (in `$v0`)
`read_float`	6		float (in `$f0`)
`read_double`	7		double (in `$f0`)
`read_string`	8	buf `$a0`, buflen `$a1`	
`sbrk`	9	amount `$a`	address (in `$v0`)
`exit`	10		

EXCEPTION CODES

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

[1] Patterson, David A; Hennessy, John J.: Computer Organization and Design, 3rd Edition. Morgan Kaufmann Publishers. San Francisco, 2005.