

Written test on 10th of December 2015

Course: COMPUTER ARCHITECTURE AND ENGINEERING

Course no. 02155

Exam time: 2 hours

No aids – Allowed: - pocket calculator
- one (1) A4-format sheet with notes

For clarifications on allowed aids do not hesitate to contact the teacher.

Weighting: Each of the 5 problems counts 20%. All questions in a problem are weighted equally.

Answers are evaluated based on correctness, completeness and conciseness (i.e., answers must be: correct, complete, and short).

If some specification is missing, just state your own specification and continue the problem according to your stated specification.

Problem 1

Answer the following questions.

1. Performance is measured in execution time. Which factors do influence the execution time?
2. What is the main usage of the jump register instruction (e.g., jr \$t1)?
3. With virtual memory and address translation, the cache can either operate on virtual addresses or on physical addresses. What is commonly used and why?
4. Why does the 5-stage MIPS pipeline have dedicated instruction and data caches, and not a single shared cache for instructions and data?
5. What is the "*dirty bit*" and when is it used?

Problem 2

Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `$s0`, `$s1`, `$s2`, `$s3`, and `$s4`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `$s6` and `$s7`, respectively. Assume that the elements of the arrays `A` and `B` are 4-byte words.

Translate the following C code to MIPS assembly:

```
B[4] = A[i] + A[j]
```

Translate the following MIPS code to C:

```
slli $t0, $s4, 1
add  $s0, $t0, $s3
add  $s0 $s0, $s1
```

Translate the following MIPS code to C:

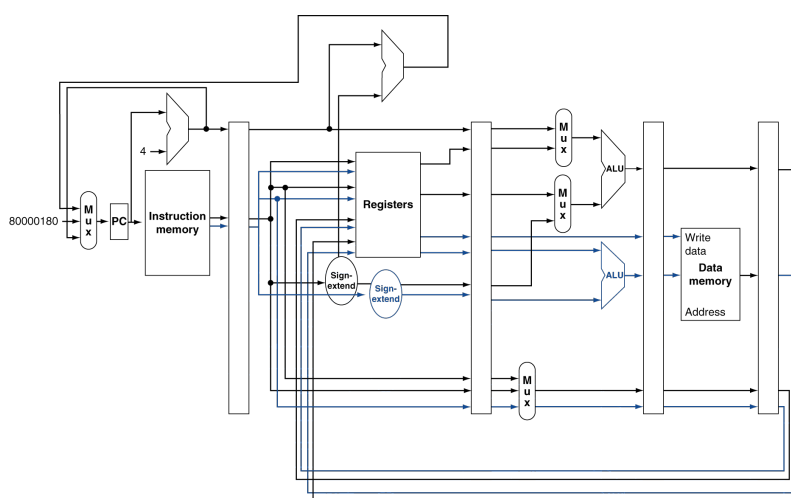
```
slli $t0, $s0, 2
add  $t0, $s6, $t0
slli $t1, $s1, 2
add  $t1, $s7, $t1
lw   $t2, 0($t0)
addi $t3, $t0, 4
lw   $t0, 0($t3)
add  $t0, $t0, $t2
sw   $t0, 0($t1)
```

Problem 3

Consider the following processors and compilers

P1 5-stage pipelined MIPS processor (stages: F, D, E, M, W) with data-forwarding.

P2 a static dual issue pipelined MIPS processor with data-forwarding similar to the one in the textbook (see figure below) in which the first slot can issue either R-type of branch instructions and the second slot only load or store instructions.



The processors are used to execute the following fragment of MIPS code

```
loop: lw    $6, 0($1)
      sub   $6, $6, $4
      sw    $6, 0($1)
      addi  $1, $1, -4
      bne   $1, $0, loop
```

Assume the branch is ALWAYS predicted correctly (i.e., no stalls on branches).

- Show the timing diagram¹ for **P1** and **P2** and report the number of cycles assuming that the loop is executed once. The compiler can re-arrange instructions.
- The compiler unrolls four iterations of the loop and re-arranges instructions. Show the timing diagram for **P1** and **P2** and report the number of cycles when one pack of 4-unrolled iterations is executed.

¹For the timing diagram you should use the compact representation shown in the lectures:

	1	2	3	4	5	6	7	...
add \$5, \$0, \$0	F	D	E	M	W			
lw \$4, 0(\$3)		F	D	E	M	W		
...						

Problem 4

Media applications that play audio or video files are part of a class of workloads called “streaming” workloads; i.e., they bring large amounts of data but do not reuse much of it. Consider a video streaming workload that accesses 1024 KB working set sequentially with the following address stream (byte addresses):

0, 2, 4, 6, 8, 10, 12, 14, 16, ...

Assume a 32 KB direct-mapped cache with a 32-byte block. What is the miss rate for the address stream above? How is this miss rate sensitive to the size of the cache or the working set? How would you categorize the misses this workload is experiencing based on the 3C model?

Re-compute the miss rate when the cache block size is 16 bytes, 64 bytes, and 128 bytes. What kind of locality is this workload exploiting?

Problem 5

We want to benchmark two multicores using the roofline model.

Multicore **M1** has a peak floating-point performance (peak-FP perf.) of 128 GFLOP/s from Arithmetic Intensity (AI) $AI=8$ FLOP/byte (and above). Its FP performance at $AI=1/8$ FLOP/byte is 2 GFLOP/s.

Multicore **M2** has a peak-FP performance of 16 GFLOP/s from $AI=1/2$ FLOP/byte (and above), and its performance at $AI=1/8$ FLOP/byte is 4 GFLOP/s.

- a) Draw the roofline model for multicore **M1**.
- b) Draw the roofline model for multicore **M2**.
- c) Explain what happens when running a benchmark with $AI=1$ FLOP/byte on the two multicores. Which multicore does perform the best?

_____ END OF THE EXAM _____

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
I	opcode	rs	rt	immediate		
	31	26 25	21 20	16 15		
J	opcode	address				
	31	26 25				

ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / 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/--/--/0
Load FP Double	ldc1 I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/--/--/0
Move From Hi	mfhi R	R[rd] = Hi	0 / --/--/10
Move From Lo	mfl0 R	R[rd] = Lo	0 / --/--/12
Move From Control	mfc0 R	R[rd] = CR[rs]	10 / 00/--/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/--/--/0
Store FP Double	sdc1 I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/--/--/0

FLOATING-POINT INSTRUCTION FORMATS

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

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	b1e	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

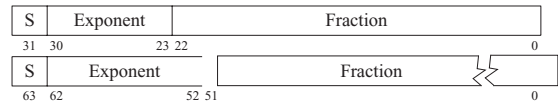
MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Decimal	Hexadecimal	ASCII Character	Decimal	Hexadecimal	ASCII Character
(1)	sll	add _f	00 0000	0	0	NUL	64	40	@
		sub _f	00 0001	1	1	SOH	65	41	A
j	srl	mul _f	00 0010	2	2	STX	66	42	B
jal	sra	div _f	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt _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
(2)	mfhi		01 0000	16	10	DLE	80	50	P
	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	`
	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
swr			10 1100	44	2c	,	108	6c	l
cache			10 1101	45	2d	-	109	6d	m
			10 1110	46	2e	.	110	6e	n
			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	tlbt	c.eq _f	11 0010	50	32	2	114	72	r
pref	tlbtu	c.ueq _f	11 0011	51	33	3	115	73	s
	teq	c.lt _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	}
sdcl		c.nges _f	11 1101	61	3d	=	125	7d	~
sdcl		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

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

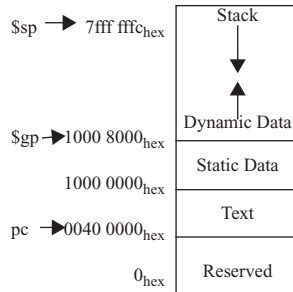
where Single Precision Bias = 127,
Double Precision Bias = 1023.IEEE Single Precision and
Double Precision Formats:

IEEE 754 Symbols

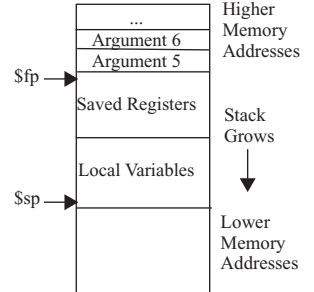
Exponent	Fraction	Object
0	0	± 0
0	≠ 0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	± ∞
MAX	≠ 0	NaN

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

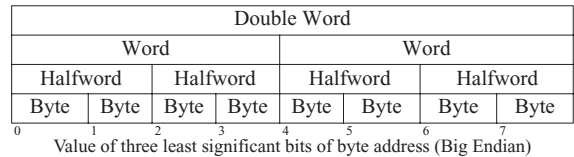
MEMORY ALLOCATION



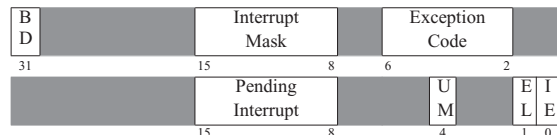
STACK FRAME



DATA ALIGNMENT



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 (10³ for Disk, Communication; 2³ 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.