CS G524: Advanced Computer Architecture
Assignment-2
Due Date: Feb 21, 2020
Birla Institute of Technology and Science (BITS) Pilani,
K K Birla Goa Campus
Second Semester 2019 – 2020

Instructions [10 Marks]. Students are advised to write their program with proper care. A program must have a header block consisting of programmer's name, roll no and date of creation [1.5 marks]. The objective of the program has to be written in the header block, also [1.5 marks]. The program should be properly indented [3 marks] and it is expected meaningful variable name [4 marks]. Use lowercase for variables and methods. If name consists of several words:
Make them into one

- First word lowercase
- Capitalize the first letter of each subsequent word
- Ex: radius, area and showMessageDialog

Capitalize every first letter in a constant, and use underscores between words

- Ex: PI and MAX_VALUE

Submission. Upload your assignment on Moodle in the link provided on the course homepage. Assignment should be a zip folder with six assignment " .v " extension files and one Word file which contains the descriptive solution expected for all 6 assignments. Zip file name must be your group number. E.g. group1.zip
Note. Member of each group must highlight their contribution (using comments)

Problem 1. Design a 32-bit Booth multiplier circuit using Verilog. (Marks: 10)

Problem 2. Design a 32-bit Non-restoring divider circuit using Verilog. (Marks: 10)

Problem 3. Using QtSPIM, write and test an adding machine program that repeatedly reads in integers and adds them into a running sum. The program would stop when it gets an input that is 0, printing out the sum at that point.

(Marks: 5)

Problem 4. Using QtSPIM, write and test a program that reads in three integers and prints out the sum of the largest two of three. You can break ties arbitrarily.

(Marks: 5)

Problem 5. Using QtSPIM, write and test a program that reads in a positive integer using the SPIM system calls. If the integer is not positive, the program should terminate with the message "Invalid Entry"; otherwise the program should print out the names of the digits of the integers delimited by exactly one space. For example, if the number entered is "728", the output would be "seven two eight".

(Marks: 5)

Problem 6. Write and test a MIPS assembly language program to compute and print the first 20 prime numbers. A number *n* is prime if no numbers except 1 and *n* divide it evenly. You should implement two routines:

- *test_prime (n)* Return 1 if n is prime and 0 if not prime
- *main( )* Iterate over the integers, testing if each is prime. Print the first 20 numbers that are prime.

Test your programs by running them on QtSPIM.

(Marks: 5)

Problem 7. Design a single-cycle MIPS-based (32-bit) processor using Verilog for the instructions such as R-type (ADD, SUB, AND, OR, SLT), I/M-type (LW/SW/ADDI/SUBI) and BEQ and J-type instructions (JAL, J). Consider only integer type of operation. To test the design one must use the mipstest.s file. Find out the delay of the design.

(Marks: 20)

Problem 8. Design a multicycle MIPS-based (32-bit) processor using Verilog for the instructions such as R-type (ADD, SUB, AND, OR, SLT), I/M-type (LW/SW/ADDI/SUBI) and BEQ and J-type instructions (JAL, J). Consider only integer type of operation. To test the design one must use the mipstest.s file.

(Marks: 10)

mipstest.s:

Label	Assembly	Description	Address	Machine code
main:	addi $2, $0, 5	initialize $2 = 5	0	20020005
	addi $3, $0, 12	initialize $3 = 12	4	2003000c
	addi $7, $3, −9	initialize $7 = 3	8	2067fff7
	or $4, $7, $2	$4 = (3 OR 5) = 7	c	00e22025
	and $5, $3, $4	$5 = (12 AND 7) = 4	10	00642824
	add $5, $5, $4	$5 = 4 + 7 = 11	14	00a42820
	beq $5, $7, end	shouldn't be taken	18	10a7000a
	slt $4, $3, $4	$4 = 12 < 7 = 0	1c	0064202a
	beq $4, $0, around	should be taken	20	10800001
	addi $5, $0, 0	shouldn’t happen	24	20050000
around:	slt $4, $7, $2	$4 = 3 < 5 = 1	28	00e2202a
	add $7, $4, $5	$7 = 1 + 11 = 12	2c	00853820
	sub $7, $7, $2	$7 = 12 − 5 = 7	30	00e23822
	sw $7, 68($3)	[80] = 7	34	ac670044
	lw $2, 80($0)	$2 = [80] = 7	38	8c020050
	j end	should be taken	3c	08000011
	addi $2, $0, 1	shouldn't happen	40	20020001
end:	sw $2, 84($0)	write mem[84] = 7	44	ac020054

The MIPS testbench loads the above program into the memories. The program file contains only the machine code. The machine code has been generated using the help of the Table B1 and B2. The above program exercises all of the instructions by performing a computation that should produce the correct answer only if all of the instructions are functioning properly. Specifically, the program will write the value 7 to address 84 if it runs correctly, and is unlikely to do so if the hardware is buggy.

Table B.1 Instructions, sorted by opcode

Opcode	Name	Description	Operation
000000 (0)	R-type	all R-type instructions	see Table B.2
000001 (1) (rt = 0/1)	bltz rs, label / bgez rs, label	branch less than zero/branch greater than or equal to zero	if ([rs] < 0) PC = BTA/ if ([rs] ≥ 0) PC = BTA
000010 (2)	j label	jump	PC = JTA
000011 (3)	jal label	jump and link	$ra = PC + 4, PC = JTA
000100 (4)	beq rs, rt, label	branch if equal	if ([rs] == [rt]) PC = BTA
000101 (5)	bne rs, rt, label	branch if not equal	if ([rs] != [rt]) PC = BTA
000110 (6)	blez rs, label	branch if less than or equal to zero	if ([rs] ≤ 0) PC = BTA
000111 (7)	bgtz rs, label	branch if greater than zero	if ([rs] > 0) PC = BTA
001000 (8)	addi rt, rs, imm	add immediate	[rt] = [rs] + SignImm
001001 (9)	addiu rt, rs, imm	add immediate unsigned	[rt] = [rs] + SignImm
001010 (10)	slti rt, rs, imm	set less than immediate	[rs] < SignImm ? [rt] = 1 : [rt] = 0
001011 (11)	sltiu rt, rs, imm	set less than immediate unsigned	[rs] < SignImm ? [rt] = 1 : [rt] = 0
001100 (12)	andi rt, rs, imm	and immediate	[rt] = [rs] & ZeroImm
001101 (13)	ori rt, rs, imm	or immediate	[rt] = [rs] | ZeroImm
001110 (14)	xori rt, rs, imm	xor immediate	[rt] = [rs] ^ ZeroImm
001111 (15)	lui rt, imm	load upper immediate	[rt] = {imm, 16'b0}
010000 (16) (rs = 0/4)	mfc0 rt, rd / mtc0 rt, rd	move from/to coprocessor 0	[rt] = [rd]/[rd] = [rt] (rd is in coprocessor 0)
010001 (17)	F-type	fop = 16/17: F-type instructions	see Table B.3
010001 (17) (rt = 0/1)	bc1f label/ bc1t label	fop = 8: branch if fpcond is FALSE/TRUE	if (fpcond == 0) PC = BTA/ if (fpcond == 1) PC = BTA
011100 (28) (func = 2)	mul rd, rs, rt	multiply (32-bit result)	[rd] = [rs] x [rt]
100000 (32)	lb rt, imm(rs)	load byte	[rt] = SignExt ($[Address]_{7:0}$)
100001 (33)	lh rt, imm(rs)	load halfword	[rt] = SignExt ($[Address]_{15:0}$)
100011 (35)	lw rt, imm(rs)	load word	[rt] = [Address]
100100 (36)	lbu rt, imm(rs)	load byte unsigned	[rt] = ZeroExt ($[Address]_{7:0}$)
100101 (37)	lhu rt, imm(rs)	load halfword unsigned	[rt] = ZeroExt ($[Address]_{15:0}$)
101000 (40)	sb rt, imm(rs)	store byte	$[Address]_{7:0} = [rt]_{7:0}$
101001 (41)	sh rt, imm(rs)	store halfword	$[Address]_{15:0} = [rt]_{15:0}$
101011 (43)	sw rt, imm(rs)	store word	[Address] = [rt]
110001 (49)	lwc1 ft, imm(rs)	load word to FP coprocessor 1	[ft] = [Address]
111001 (56)	swc1 ft, imm(rs)	store word to FP coprocessor 1	[Address] = [ft]

Table B.2 R-type instructions, sorted by funct field

Funct	Name	Description	Operation
000000 (0)	sll rd, rt, shamt	shift left logical	[rd] = [rt] << shamt
000010 (2)	srl rd, rt, shamt	shift right logical	[rd] = [rt] >> shamt
000011 (3)	sra rd, rt, shamt	shift right arithmetic	[rd] = [rt] >>> shamt
000100 (4)	sllv rd, rt, rs	shift left logical variable	[rd] = [rt] << $[rs]_{4:0}$
000110 (6)	srlv rd, rt, rs	shift right logical variable	[rd] = [rt] >> $[rs]_{4:0}$
000111 (7)	srav rd, rt, rs	shift right arithmetic variable	[rd] = [rt] >>> $[rs]_{4:0}$
001000 (8)	jr rs	jump register	PC = [rs]
001001 (9)	jalr rs	jump and link register	$ra = PC + 4, PC = [rs]
001100 (12)	syscall	system call	system call exception
001101 (13)	break	break	break exception
010000 (16)	mfhi rd	move from hi	[rd] = [hi]
010001 (17)	mthi rs	move to hi	[hi] = [rs]
010010 (18)	mflo rd	move from lo	[rd] = [lo]
010011 (19)	mtlo rs	move to lo	[lo] = [rs]
011000 (24)	mult rs, rt	multiply	{[hi], [lo]} = [rs] × [rt]
011001 (25)	multu rs, rt	multiply unsigned	{[hi], [lo]} = [rs] × [rt]
011010 (26)	div rs, rt	divide	[lo] = [rs]/[rt], [hi] = [rs]%[rt]
011011 (27)	divu rs, rt	divide unsigned	[lo] = [rs]/[rt], [hi] = [rs]%[rt]
100000 (32)	add rd, rs, rt	add	[rd] = [rs] + [rt]
100001 (33)	addu rd, rs, rt	add unsigned	[rd] = [rs] + [rt]
100010 (34)	sub rd, rs, rt	subtract	[rd] = [rs] - [rt]
100011 (35)	subu rd, rs, rt	subtract unsigned	[rd] = [rs] - [rt]
100100 (36)	and rd, rs, rt	and	[rd] = [rs] & [rt]
100101 (37)	or rd, rs, rt	or	[rd] = [rs] | [rt]
100110 (38)	xor rd, rs, rt	xor	[rd] = [rs] ^ [rt]
100111 (39)	nor rd, rs, rt	nor	[rd] = ~([rs] | [rt])
101010 (42)	slt rd, rs, rt	set less than	[rs] < [rt] ? [rd] = 1 : [rd] = 0
101011 (43)	sltu rd, rs, rt	set less than unsigned	[rs] < [rt] ? [rd] = 1 : [rd] = 0