Czech Technical University in Prague, Faculty of Electrical Engineering

Advanced Computer Architectures

Semester project – 01

Theme: Simple CPU in Verilog

PART I

The aim of this part of project is to design a simple 32-bit processor connected to separate instruction and data memory. The processor has to implement only these instructions: **add**, **sub**, **and**, **or**, **slt**, **addi**, **lw**, **sw**, **beq, bne, sll, srl, jal** and **jr**. MIPS instruction set architecture is supposed. The detailed information about the instruction format is in the attachment. Suppose that the processor starts the execution from the beginning of instruction memory (0x00000000).

PART II

Write a program in C language that takes two positive integer numbers and computes their greatest common divisor (gcd). Translate this program into the assembly language by using instructions provided in Part I. Use O32 calling convention. Suppose that input variables (32 bit) are stored in data memory at addresses 0x00000004 and 0x00000008, respectively. Write the result of your program into the data memory at address of 0x0000000C.

PART III

Demonstrate the functionality of your design (run the program from Part II and report the results). Suppose that the program is already stored in instruction memory from the address of 0x00000000. You can use following module to represent instruction memory (if it is large enough, otherwise you have to modify it):

```
module imem (input [5:0] Addr,
             output [31:0] rd);
  // Addr is the address of the instruction to fetch, what
  // for our purpose can be taken from ProgramCounter[7:2]

  reg [31:0] RAM[127:0];
  initial
    $readmemh ("memfile.dat",RAM); //stored in hexadecimal format

  assign rd <= RAM[Addr]; // word aligned
endmodule
```

Attachments

O32 calling convention:

Name	Number	Use	Callee must preserve?
$zero	$0	constant 0	N/A
$at	$1	assembler temporary	No
$v0–$v1	$2–$3	values for function returns and expression evaluation	No
$a0–$a3	$4–$7	function 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	N/A
$gp	$28	global pointer	Yes
$sp	$29	stack pointer	Yes
$fp	$30	frame pointer	Yes
$ra	$31	return address	N/A

MIPS Instruction Reference:

MIPS has 32 integer registers ($0 - $31). Register $0 always holds 0. Instructions are divided into three types: R, I and J. Every instruction starts with a 6-bit opcode. In addition to the opcode, R-type instructions specify three registers, a shift amount field, and a function field; I-type instructions specify two registers and a 16-bit immediate value. All R-type instructions have an opcode of 0. The specific R-type operation is determined by the *funct* field.

Type	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	opcode(6)						rs(5)					rt(5)					rd(5)					shamt(5)					funct(6)					
I	opcode(6)						rs(5)					rt(5)					immediate (16)															
J	opcode(6)						address (26)																									

ADD – ***Add (with overflow)***

Description:	Adds two registers and stores the result in a register
Operation:	$d = $s + $t;
Syntax:	add $d, $s, $t
Encoding:	`0000 00ss ssst tttt dddd d000 0010 0000`

ADDI -- ***Add immediate (with overflow)***

Description:	Adds a register and a sign-extended immediate value and stores the result in a register
Operation:	$t = $s + imm;
Syntax:	addi $t, $s, imm
Encoding:	`0010 00ss ssst tttt iiii iiii iiii iiii`

AND -- ***Bitwise and***

Description:	Bitwise ands two registers and stores the result in a register
Operation:	$d = $s & $t;
Syntax:	and $d, $s, $t
Encoding:	`0000 00ss ssst tttt dddd d000 0010 0100`

BEQ -- ***Branch on equal***

Description:	Branches if the two registers are equal
Operation:	if $s == $t go to PC+4+4*offset; else go to PC+4
Syntax:	beq $s, $t, offset
Encoding:	`0001 00ss ssst tttt iiii iiii iiii iiii`

BNE -- *Branch on NOT equal*

Description:	Branches if the two registers are not equal
Operation:	if $s != $t go to PC+4+4*offset; else go to PC+4
Syntax:	bne $s, $t, offset
Encoding:	`0001 01ss ssst tttt iiii iiii iiii iiii`

LW -- *Load word*

Description:	A word is loaded into a register from the specified address.
Operation:	$t = MEM[$s + offset];
Syntax:	lw $t, offset($s)
Encoding:	`1000 11ss ssst tttt iiii iiii iiii iiii`

OR -- *Bitwise or*

Description:	Bitwise logical ors two registers and stores the result in a register
Operation:	$d = $s | $t;
Syntax:	or $d, $s, $t
Encoding:	`0000 00ss ssst tttt dddd d000 0010 0101`

SLT -- *Set on less than (signed)*

Description:	If $s is less than $t, $d is set to one. It gets zero otherwise.
Operation:	if $s < $t $d = 1; else $d = 0;
Syntax:	slt $d, $s, $t
Encoding:	`0000 00ss ssst tttt dddd d000 0010 1010`

SLL -- *Shift left logical immediate*

Description:	shifts shamt number of bits to the left
Operation:	$d = $t << shamt;
Syntax:	sll $d, $t, shamt
Encoding:	`0000 00-- ---t tttt dddd dhhh hh00 0000`

SRL -- *Shift right logical immediate*

Description:	shifts shamt number of bits to the right
Operation:	$d = $t >> shamt;
Syntax:	sll $d, $t, shamt
Encoding:	`0000 00-- ---t tttt dddd dhhh hh00 0010`

SUB -- *Subtract*

Description:	Subtracts two registers and stores the result in a register
Operation:	$d = $s - $t;
Syntax:	sub $d, $s, $t
Encoding:	`0000 00ss ssst tttt dddd d000 0010 0010`

SW -- *Store word*

Description:	The contents of $t is stored at the specified address.
Operation:	MEM[$s + offset] = $t;
Syntax:	sw $t, offset($s)
Encoding:	`1010 11ss ssst tttt iiii iiii iiii iiii`

JAL -- *Jump and link*

Description:	For procedure call.
Operation:	$31 = PC + 8; PC = (PC & 0xf0000000) | (target << 2)
Syntax:	jal target
Encoding:	`0000 11ii iiii iiii iiii iiii iiii iiii`

JR – *Jump register*

Description:	Jumps to the address contained in the specified register
Operation:	goto address $s
Syntax:	jr $s
Encoding:	`0000 00ss sss0 0000 0000 0000 0000 1000`