České vysoké učení technické v Praze, Fakulta elektrotechnická

Pokročilé architektury počítačů (A4M36PAP/A4DM36PAP)

Semestrální projekt – 01

Téma projektu: Návrh jednoduchého výpočetního systému

Zadání: Navrhněte a popište v jazyce Verilog jednoduchý výpočetní systém pozůstávající z procesoru, oddělené instrukční a datové paměti. Procesor bude podporovat instrukce add, sub, and, or, slt, addi, lw, sw a beq ve formátu instrukční sady MIPS. Součástí procesoru musí být řídicí jednotka a aritmeticko-logická jednotka. Funkčnost Vašeho návrhu demonstrujte simulací níže uvedeného programu.

```
if(a<=b)
    a = b-a;
else
    for(int i=0; i!=c; i++)
        a += i;
a = a & 3;
while(1)
    ;
```

Předpokládejte, že proměnné `a`, `b` a `c` jsou typu `int` (4B) a jsou již uloženy v paměti na adresách 0x0010, 0x0014 a 0x0018 datové paměti, proměnná `c` je větší než 0, a program je již uložen v instrukční paměti od adresy 0x0000 (tj. není potřeba řešit zavádění programu). Procesor po resetu (po spuštění) začíná vykonávat instrukce od adresy 0x0000.

Formální pokyny:

Odevzdané zadání musí být ve formátu pdf a musí obsahovat:

- titulní stránku (jméno, studijní zaměření,...),
- znění zadání,
- řešení.

Součástí řešení musí být přepis simulovaného programu do jazyka symbolických adres pomocí instrukcí uvedených v zadání, jeho vyjádření v strojovém kódu (šestnáctkově), kompletní popis navrženého výpočetního systému v jazyce Verilog, blokové schéma navrženého systému, výsledky simulace (průběh řídicích signálů řídicí jednotky procesoru, výpis datové paměti po ukončení simulace apod.).

Bližší specifikace a informace:

K dispozici je 32 pracovních registrů. Registry jsou 32-bitové a jsou číslovány $0 až $31. V registru $0 je vždy uložena hodnota 0. Všechny MIPS instrukce jsou 32-bitové a jsou děleny do třech skupin: R (add, and, or, slt, sub), I (addi, beq, lw, sw) a J. Každá instrukce začíná 6-bitovým operačním kódem (opcode). Všechny R-instrukce mají opcode roven $000000_{(2)}$, přičemž v tomto případě vykonávanou funkci určují bity 5:0 (funct). Instrukce typu J nebudou ve Vašem návrhu použity.

Typ	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)															

Bližší popis, vykonávanou operaci, syntax a kódování pro instrukce podporované navrženým procesorem uvádějí tabulky níže.

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`

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`

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`