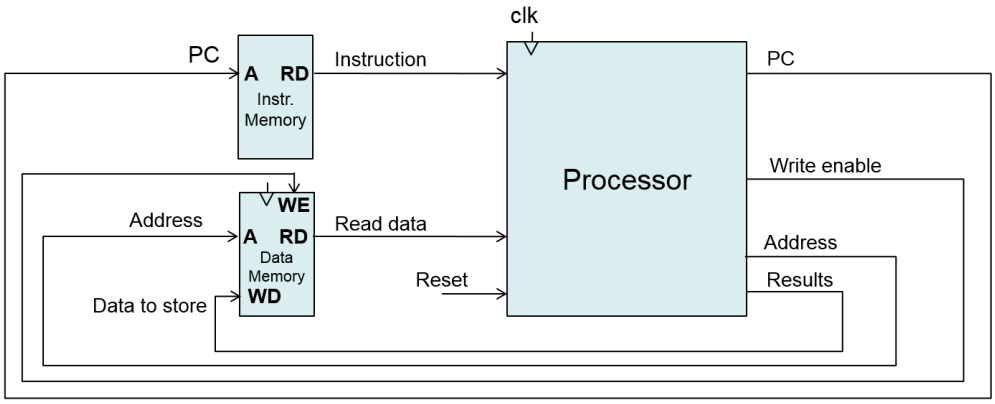
Semestrální projekt č.1: Jednocyklový procesor

Semestrální projekt č.1: Jednocyklový procesor

Základní návrh

Navrhněte a popište v jazyce Verilog jednoduchý 32-bitový procesor. Procesor musí podporovat následující instrukce: add, sub, and, or, slt, addi, lw, sw, beq, jal, jr, addu.qb a addu_s.qb. Procesor po resetu začne vykonávat instrukce od adresy 0x00000000. Procesor je připojený k instrukční a datové paměti dle obrázku níže.



Instruction	Syntax	Operation	Encoding	Note
add	add d, s, t	d = s + t;	0000 00ss ssst tttt dddd d000 0010 0000	
sub	sub d, s, t	d = s - t;	0000 00ss ssst tttt dddd d000 0010 0010	
and	and d, s, t	d = s & t;	0000 00ss ssst tttt dddd d000 0010 0100	
or	or d, s, t	d = s | t;	0000 00ss ssst tttt dddd d000 0010 0101	
slt	slt d, s, t	d = (s<t) ? 1 : 0;	0000 00ss ssst tttt dddd d000 0010 1010	
addi	addi t, s, imm	t = s + imm;	0010 00ss ssst tttt iiii iiii iiii iiii	
lw	lw t, offset(s)	t = MEM[s + offset];	1000 11ss ssst tttt iiii iiii iiii iiii	
sw	sw t, offset(s)	MEM[s + offset] = t;	1010 11ss ssst tttt iiii iiii iiii iiii	
beq	beq s, t, offset	if s==t then PC=PC+4+(offset <<	0001 00ss ssst tttt iiii iiii iiii iiii	

		2); else PC=PC+4;		
jal	jal target	$31=PC+4; PC=(PC & 0xf0000000) | (target << 2);	`0000 11ii iiii iiii iiii iiii iiii iiii`	this do not match original MIPS32
jr	jr s	PC = s;	`0001 11ss sss0 0000 0000 0000 0000 1000`	this do not match original MIPS32, opcode changed
addu.qb	addu.qb d, s, t	$d_{31:24} = s_{31:24} + t_{31:24}$; etc.	`0111 11ss ssst tttt dddd d000 0001 0000`	Element-wise addition of two vectors of unsigned byte values
addu_s.qb	addu_s.qb d, s, t	$d_{31:24} = sat(s_{31:24} + t_{31:24})$; etc.	`0111 11ss ssst tttt dddd d001 0001 0000`	with saturation

Poznámka: Písmena **d**, **s** a **t** odkazují na hodnoty uložené v těchto registrech.

Rozšířený návrh

Přidejte do procesoru podporu pro následující instrukce: sllv, srlv, srav a j.

Instruction	Syntax	Operation	Encoding	Note
sllv	sllv d, t, s	d = t << s;	`0000 00ss ssst tttt dddd dxxx xx00 0100`	
srlv	srlv d, t, s	d = (unsigned)t >> s;	`0000 00ss ssst tttt dddd d000 0000 0110`	
srav	srav d, t, s	d = (signed)t >> s;	`0000 00ss ssst tttt dddd d000 0000 0111`	The sign bit is shifted in.
j	j target	PC=(PC & 0xf0000000) | (target << 2);	`0000 10ii iiii iiii iiii iiii iiii iiii`	

Pozor na pořadí operandů! Poznámka: Odevzdávací systém testuje tyto instrukce postupně. V případě, že narazí na nesprávnou implementaci, nepokračuje v testování a vypíše počet bodů. Protože se jedná o rozšířený návrh, již neposkytuje nápovědu.

Program

Napište program v jazyce C, který vzestupně seřadí pole celých čísel (jedno číslo má délku 32 bitů, může být i záporné). Program bude používat funkci sort() s následujícím prototypem:

```
void sort(int *address, int N);
```

Můžete použít libovolný algoritmus řazení. Přeložte tento program do jazyka symbolických adres a následně do strojového kódu. Používejte volací konvenci O32. Předpokládejte, že adresa pole čísel je uložena v paměti na adrese 0x0000000C a počet prvků v poli je uložen v paměti na adrese 0x00000008.

Hodnocení semestrální práce

Popis	Bodování
Základní návrh v jazyce Verilog	12
Rozšířený návrh: sllv, srlv, srav, j	1 bod/instrukci, celkem: 4
Program	9

Semestrální práce by měla být odevzdána do konce 8. týdne semestru. Každý započatý týden zpoždění je penalizován ztrátou dvou bodů. Odevzdání po 13. týdnu není možné.

Váš program musí být otestován na Vašem CPU, jinak se nehodnotí. Jinými slovy, 9 bodů za program můžete získat pouze tehdy, pokud odevzdáte popis CPU, na kterém tento program běží a generuje očekávané výsledky (seřadí pole v paměti). Body za program můžete získat i tehdy, není-li Váš CPU zcela korektní.

Způsob odevzdání semestrální práce

Vaše řešení (zabalené v zip archivu) odevzdejte přes http://biaps.fit.cvut.cz/first_semestral_project/index.php V případě problémů, můžete Vaše řešení poslat na email cvičícího (pokud jste nevyčerpali všechny pokusy).

CELKEM je k dispozici 7 pokusů. V systému se uchovává pouze poslední pokus.

Základní a rozšířený návrh

Popis CPU uložte do jednoho souboru. Název souboru musí být následující: **Surname_GivenName_CPU.v** (bez diakritiky, čili bez háčků a čárek). Všechny moduly, které potřebujete, popište v rámci tohoto souboru.

Použijte následující šablonu. Nazvy vstupů a výstupů neměňte.

```
module processor( input         clk, reset,
                  output [31:0] PC,
                  input  [31:0] instruction,
                  output        WE,
                  output [31:0] address_to_mem,
                  output [31:0] data_to_mem,
                  input  [31:0] data_from_mem
                );
```

```
    //... write your code here ...
endmodule

//... add new modules here ...
```

> ⚠ Odevzdávejte pouze popis CPU. Nepřikládejte popis dalších komponent (data memory, instruction memory, etc.).

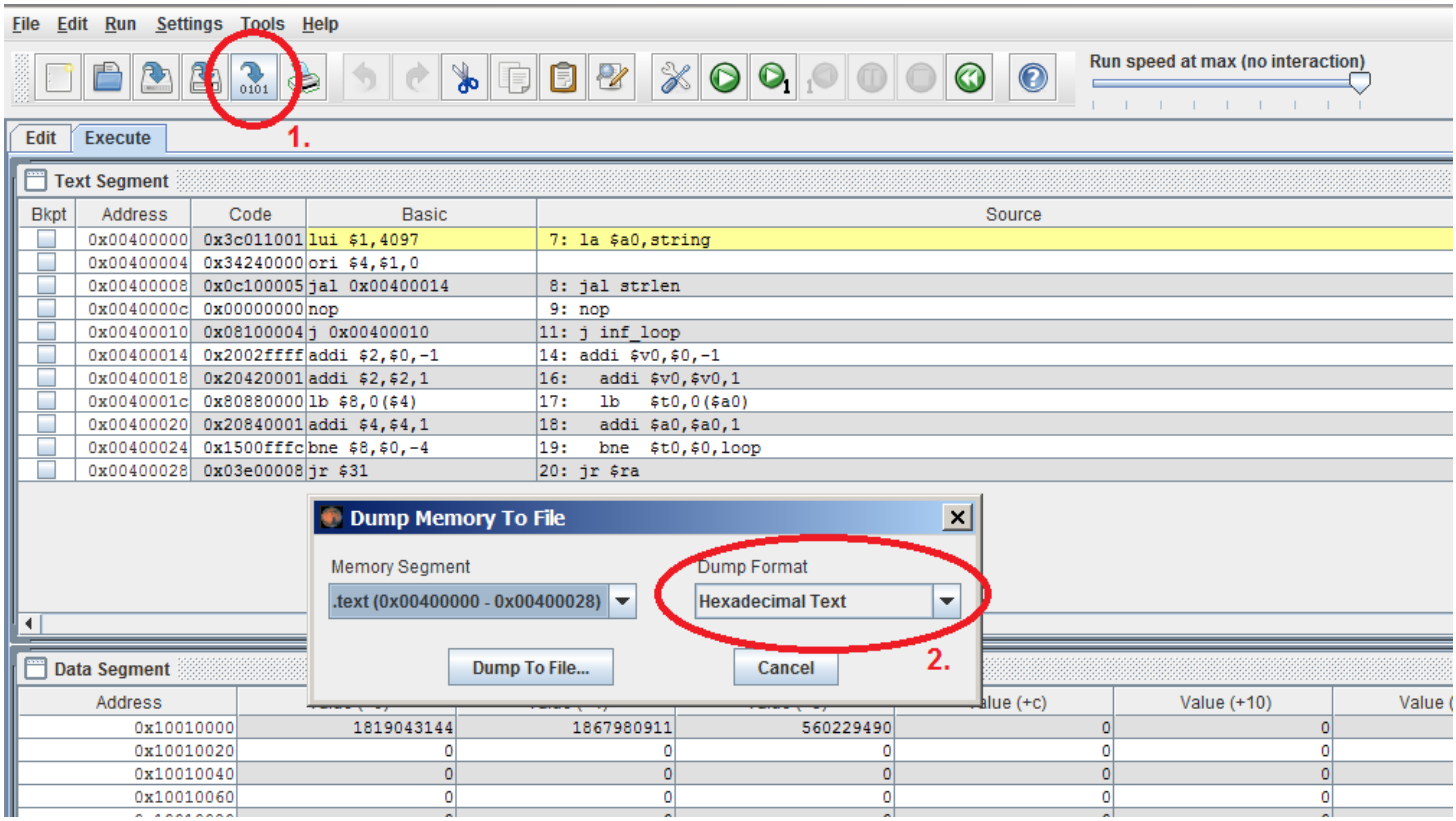
Program

Uložte Váš program v jazyku symbolických adres do jednoho souboru. Jméno souboru musí být následující: **Surname_GivenName_prog1.asm** . Program ve strojovém kódu (hexadecimální formát) musí být uložen v dalším souboru, kde každá instrukce začíná na novém řádku. Jméno souboru musí být následující: **Surname_GivenName_prog1.hex**

> ⚠ Vsechny odevzdavane soubory zabalte do zip archivu. Nevytvarejte slozky a podslozky v tomto archivu. Archiv musi obsahovat POUZE odevzdavane soubory.

Pomůcka

Pro vygenerování strojového kódu lze použít simulátor MARS používaný na cvičeních. Je vhodné použít textový formát, kde každá instrukce je uvedena hexadecimálně, samostatně v každém řádku souboru. Postup ilustruje následující obrázek. Pozor, je MARS používá instrukční sadu MIPS32, která se od naší nepatrně liší - viz výše uvedenou definici instrukcí.



Dále můžete použít následující moduly a modifikovat je dle libosti. Nebudou součástí Vašeho řešení ani je neodevzdávejte.

```
module top (    input         clk, reset,
                output [31:0] data_to_mem, address_to_mem,
                output        write_enable);

        wire [31:0] pc, instruction, data_from_mem;

        inst_mem  imem(pc[7:2], instruction);
        data_mem  dmem(clk, write_enable, address_to_mem, data_to_mem, data_from_mem);
        processor CPU(clk, reset, pc, instruction, write_enable, address_to_mem, data_to_mem, data_from_mem
endmodule

//-----------------------------------------------------------------------
module data_mem (input clk, we,
                 input  [31:0] address, wd,
                 output [31:0] rd);

        reg [31:0] RAM[63:0];

        initial begin
                $readmemh ("memfile_data.hex",RAM,0,63);
        end
```

```
        assign rd=RAM[address[31:2]]; // word aligned

        always @ (posedge clk)
                if (we)
                        RAM[address[31:2]]<=wd;
endmodule

//-----------------------------------------------------------------
module inst_mem (input  [5:0]  address,
                 output [31:0] rd);

        reg [31:0] RAM[63:0];
        initial begin
                $readmemh ("memfile_inst.hex",RAM,0,63);
        end
        assign rd=RAM[address]; // word aligned
endmodule
```

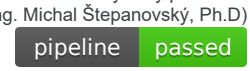
Pro simulaci můžete použít následující modul:

```
module testbench();
        reg         clk;
        reg         reset;
        wire [31:0] data_to_mem, address_to_mem;
        wire        memwrite;

        top simulated_system (clk, reset, data_to_mem, address_to_mem, write_enable);

        initial begin
                $dumpfile("test");
                $dumpvars;
                reset<=1; # 2; reset<=0;
                #100; $finish;
        end

        // generate clock
        always  begin
                clk<=1; # 1; clk<=0; # 1;
        end
endmodule
```


pipeline
passed