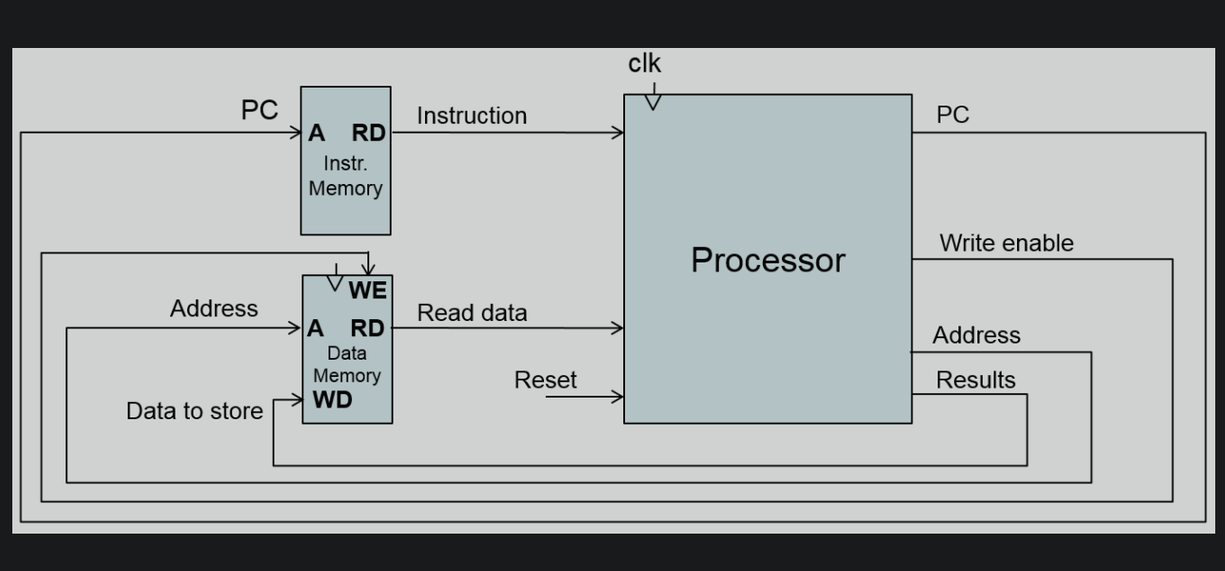
Semestrální projekt č.1: Jednocyklový procesor

Semestrální projekt č.1: Jednocyklový procesor

Základní návrh

Navrhněte a popište v jazyce Verilog jednoduchý 32-bitový procesor. Procesor musí podporovat instrukce uvedené v tabulce níže. Procesor po resetu začne vykonávat instrukce od adresy 0x00000000. Procesor je připojený k instrukční a datové paměti dle obrázku:



Instrukce	Syntax	Operace	Poznámka
add	add rd, rs1, rs2	rd ← [rs1] + [rs2];	
addi	addi rd, rs1, $imm_{11:0}$	rd ← [rs1] + $imm_{11:0}$;	
and	and rd, rs1, rs2	rd ← [rs1] & [rs2];	
sub	sub rd, rs1, rs2	rd ← [rs1] - [rs2];	
slt	slt rd, rs1, rs2	if [rs1] < [rs2] then rd←1; else rd←0;	
div	div rd, rs1, rs2	rd ← [rs1] / [rs2];	
rem	rem rd, rs1, rs2	rd ← [rs1] % [rs2];	
beq	beq rs1, rs2, $imm_{12:1}$	if [rs1] == [rs2] go to [PC]+{$imm_{12:1}$,'0'}; else go to [PC]+4;	
blt	blt rs1, rs2, $imm_{12:1}$	if [rs1] < [rs2] go to [PC]+{$imm_{12:1}$,'0'}; else go to [PC]+4;	
lw	lw rd,$imm_{11:0}$(rs1)	rd ← Memory[[rs1] + $imm_{11:0}$]	
sw	sw rs2,$imm_{11:0}$(rs1)	Memory[[rs1] + $imm_{11:0}$] ← [rs2];	
lui	lui rd, $imm_{31:12}$	rd ← {$imm_{31:12}$,'0000 0000 0000'};	
jal	jal rd, $imm_{20:1}$	rd ← [PC]+4; go to [PC] +{$imm_{20:1}$,'0'};	
jalr	jalr rd, rs1, $imm_{11:0}$	rd ← [PC]+4; go to [rs1]+$imm_{11:0}$;	

Rozšířený návrh

Přidejte do procesoru podporu pro následující instrukce: auipc, sll, srl, sra.

Instrukce	Syntax	Operace	Poznámka
auipc	auipc rd,$imm_{31:12}$	rd ← [PC] + {$imm_{31:12}$,'0000 0000 0000'};	
sll	sll rd, rs1, rs2	rd ← [rs1] << [rs2];	
srl	srl rd, rs1, rs2	rd ← (unsigned)[rs1] >> [rs2];	
sra	sra rd, rs1, rs2	rd ← (signed)[rs1] >> [rs2];	

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

Kódování instrukcí:

Instrukce jsou kódovány na 32 bitů (v tabulce od msb k lsb), přičemž hodnoty rs1, rs2 a rd jsou kódovány na 5 bitů. Poslední sloupec tabulky je operační kód dané instrukce. Kódování instrukcí odpovídá specifikaci RISC-V (RV32IM).

add:	0000000	rs2	rs1	000	rd	0110011
addi	imm[11:0]		rs1	000	rd	0010011
and:	0000000	rs2	rs1	111	rd	0110011
sub:	0100000	rs2	rs1	000	rd	0110011
slt:	0000000	rs2	rs1	010	rd	0110011
div:	0000001	rs2	rs1	100	rd	0110011
rem:	0000001	rs2	rs1	110	rd	0110011
beq:	imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011
blt:	imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011
lw:	imm[11:0]		rs1	010	rd	0000011
sw:	imm[11:5]	rs2	rs1	010	imm[4:0]	0100011
lui:	imm[31:12]				rd	0110111
jal:	imm[20|10:1|11|19:12]				rd	1101111
jalr:	imm[11:0]		rs1	000	rd	1100111
auipc:	imm[31:12]				rd	0010111
sll:	0000000	rs2	rs1	001	rd	0110011
srl:	0000000	rs2	rs1	101	rd	0110011
sra:	0100000	rs2	rs1	101	rd	0110011

Program

Napište program, který bude procházet pole čísel a určí, zda dané číslo je prvočíslem. Pokud je na dané pozici v poli prvočíslo, přepíše tuto hodnotu na 1, jinak na 0. Program musí opakovaně **volat** rutinu **prime**, která přijímá 1 argument: testované číslo. V jazyce C by této rutině odpovídala funkce s následujícím prototypem:

```
int prime(unsigned int number);
```

Rutina vrací 1, pokud "number" je prvočíslem. Za nejmenší prvočíslo považujte 2. Používejte volací konvenci RISC-V.

Předpokládejte, že informace o velikosti pole a jeho počáteční adrese jsou dány fixně na adresách v datové paměti:

- 0x00000004: velikost pole (počet prvků)
- 0x00000008: ukazatel na začátek pole

Tzn. přečtením hodnoty uložené na adrese 0x00000008 získáte adresu, kde začíná první prvek pole.

Program může mofifikovat pouze pole samotné a jakákoliv modifikace pole je považována za odpověď na test prvočíselnosti, tzn. neukládejte do pole pomocná data.

Můžete předpokládat, že "number" je vždy menší než 1000. Velikost instrukční paměti je omezena na 128 slov, tedy 512 B. Důsledek: program přesahující tuto mez nebude akceptován.

> Pokud za program dostáváte 0 bodů, může to být i chybou v popisu CPU. Plný počet bodů za CPU pouze znamená, že jste dostatečně prokázali znalosti Verilogu abyste za tuto část získali plný počet bodů.

Hodnocení semestrální práce

Popis	Bodování
Základní návrh v jazyce Verilog	12
Rozšířený návrh:	4
Program	9

Semestrální práce by měla být odevzdána do 13.11.2022. 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. 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 10 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.

```
`default_nettype none
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 ...
`default_nettype wire
```

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

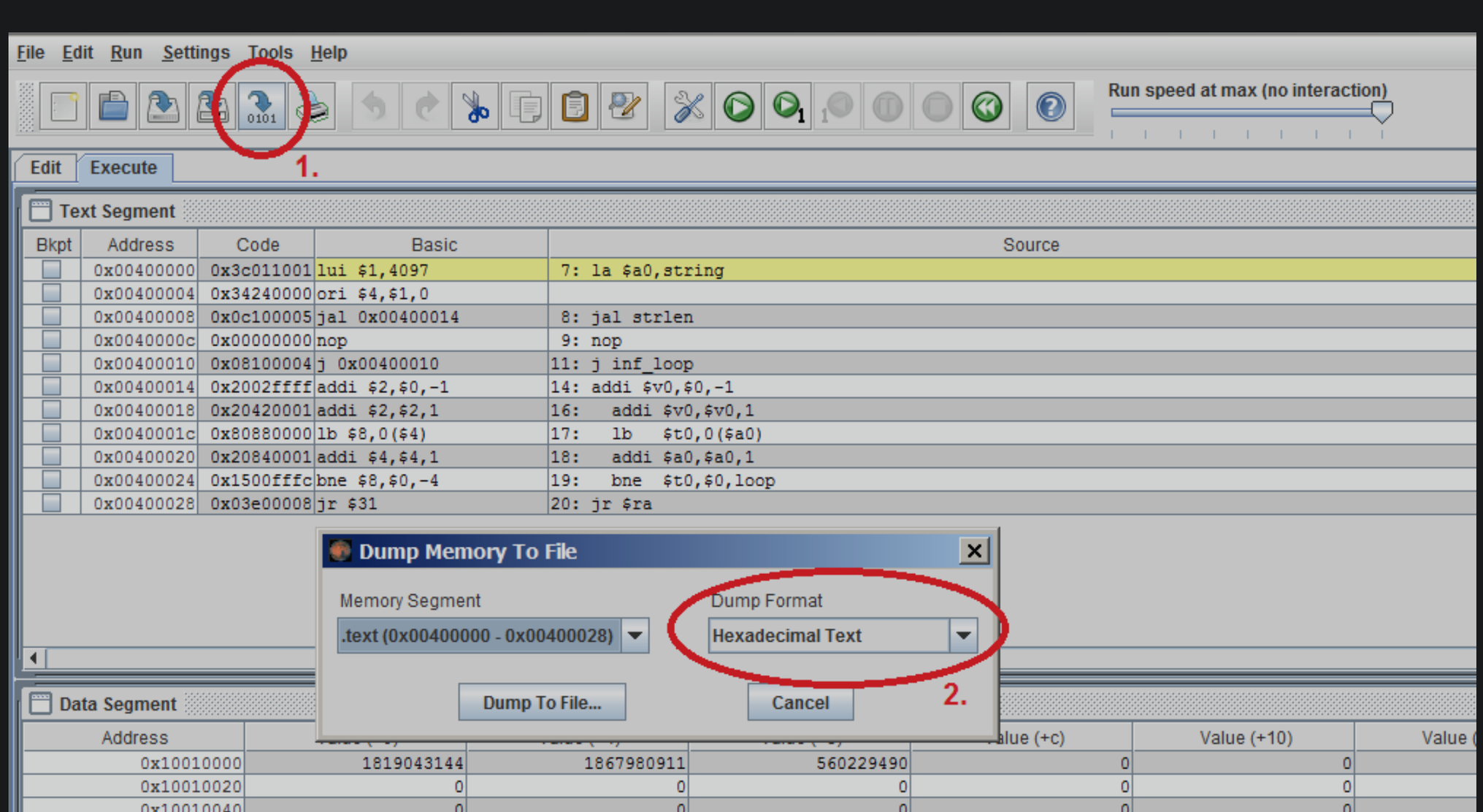
Program

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**

> Všechny odevzdávané soubory zabalte do zip archivu. Nevytvářejte složky a podsložky v tomto archívu. Archiv musí obsahovat POUZE odevzdávané soubory.

Pomůcka

Pro vygenerování strojového kódu lze použít simulátor RARS 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 RARS používá instrukční sadu RISC-V, která neimplementuje instrukci adduqb.



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_fro
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        write_enable;

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

        initial begin
                $dumpfile("test");
                $dumpvars;
                reset<=1; # 2; reset<=0;
                $writememh ("memfile_data_after_simulation.hex",simulated_system.dmem.RAM,0,63);
                #100; $finish;
        end

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