32-bitový CPU RISC-V s harvardskou architektúrou

Špecifikačné prostriedky

Autor: Pavlo Spirin
Vedúci: Ing. Ján Mach
Dátum: 8.12.2024

Slovenská technická univerzita v Bratislave

Contents

1 Popis úlohy **2**

2 Diagramy **4**
2.1 Štrukturálny blokový diagram procesora 4
2.2 Dataflow Diagram procesora 5
2.3 State Machine Diagram procesora 6
2.4 State Machine Diagram ALU 7

3 Popisy modulov **8**
3.1 cpu_top 8
3.1.1 Rozhrania modulu 8
3.1.2 Interné komponenty 9
3.1.3 Funkčnosť modulu 10
3.2 instruction_decoder 12
3.2.1 Rozhrania modulu 12
3.2.2 Funkčnosť 12
3.3 alu 13
3.3.1 Rozhrania modulu 13
3.3.2 Funkčnosť 13
3.4 multiplier 14
3.4.1 Rozhrania modulu 14
3.4.2 Funkčnosť 15
3.5 register_file 15
3.5.1 Rozhrania modulu 15
3.5.2 Funkčnosť 16

4 Demonštrácia činnosti CPU **17**
4.1 Demonštrácia činnosti CPU pomocou ModelSim 17
4.1.1 Prvý takt 18
4.1.2 Druhý takt 19
4.1.3 Tretí takt 20
4.1.4 Štvrtý takt 21
4.1.5 Piaty takt 22
4.1.6 Šiesty takt 23
4.1.7 Siedmy takt 24
4.1.8 Ôsmy takt 25

5 Záver **26**

Chapter 1

Popis úlohy

Úlohou je navrhnúť **32-bitový** procesor s inštrukčnou sadou **RISC-V** (podmnožina z **RV32I** + násobenie) a **Harvardskou architektúrou** (separátne zbernice pre dáta a inštrukcie). Protokol pre zbernice je špecifikovaný nižšie. Dáta sú v pamäti uložené spôsobom **little-endian**. Procesor bude používať **asynchrónny reset** aktívny v **nízkej** úrovni. Jedným z portov procesora bude aj **zavádzacia adresa**, ktorá bude špecifikovať hodnotu programového počítadla po resete. Top modul procesora aj s definovaným rozhraním máte k dispozícií v priloženom súbore.

Pokiaľ načítanú inštrukciu neviete dekódovať ako žiadnu z uvedených, zastavíte vykonávanie a na výstupnom porte ERROR nastavíte 1. Inštrukcie prístupu do pamäte pracujú len s 32 bitovými hodnotami a môžu pristupovať len na adresy zarovnané na 4 bajty (2 LSB sú nastavene na 0). Inštrukcie riadenia toku programu dokážu vykonávať skoky aj na adresy, ktoré niesú zarovnané na 4 bajty. Ak zistíte, že inštrukcia má skákať na takúto adresu, alebo pristupuje do pamäte na nezarovnanú adresu, zastavíte vykonávanie a na výstupnom porte ERROR nastavíte 1.

Povinné inštrukcie:

1. **Riadenie toku programu**: JAL, JALR, BEQ, BNE, BLT, BGE, BLTU, BGEU
2. **ALO s registrami**: ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND
3. **ALO s priamou hodnotou**: ADDI, SLTI, SLTIU, XORI, ORI, ANDI, SLLI, SRLI, SRAI
4. **Násobenie** MUL, MULH, MULHSU, MULHU
5. **Prístup do pamäte**: LW, SW
6. **Ostatné**: LUI, AUIPC

Protokol pamäťových zberníc:
Prístup do pamäte sa vykonáva v dvoch cykloch. V **prvom (adresnom) cykle** vystavujete **adresu** pamäťového miesta, s ktorým chcete pracovať. V prípade, že chcete dáta zapisovať do pamäte, nastavíte **signál zápisu v adresnom cykle** na 1. **Druhý cyklus je dátový**, čo znamená, že pokiaľ ste nasignalizovali v adresnom cykle zápis do pamäte, tak v dátovom cykle odosielate hodnotu, ktorú chcete zapísať. V prípade, že

ste nastavili v adresnom cykle čítanie z pamäte, v dátovom cykle prichádzajú prečítané dáta. **Adresné a dátové fázy nasledujúcich transferov sa môžu prekrývať**.

Chapter 2

Diagramy

2.1 Štrukturálny blokový diagram procesora

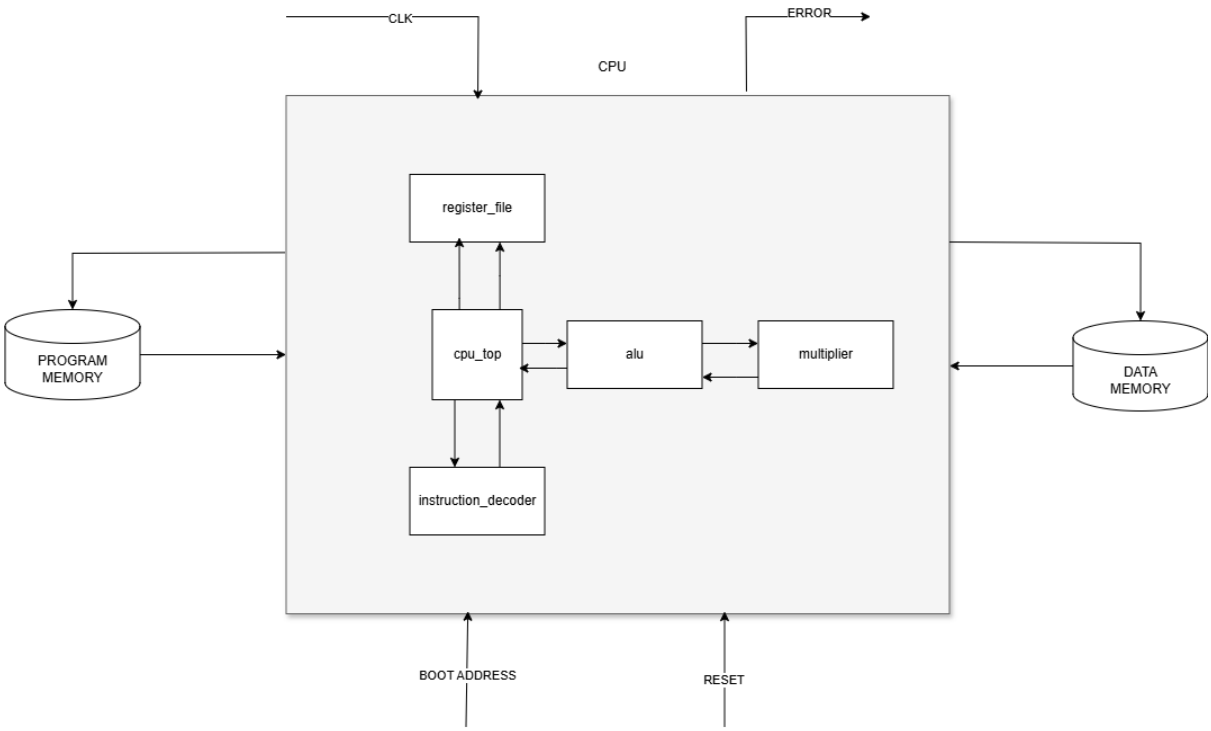

Figure 2.1: Štrukturálny blokový diagram

Tento diagram znázorňuje hlavné komponenty procesora a ich vzájomné vzťahy. Predstavuje štruktúru systému na úrovni blokov. Medzi hlavné moduly patria:

- Pamäť programu (Program Memory): Ukladá inštrukcie na vykonanie procesorom.
- Dekodér inštrukcií (Instruction Decoder): Analyzuje inštrukcie z pamäte a určuje činnosti, ktoré sa majú vykonať.
- Súbor registrov (Register File): Obsahuje registre na ukladanie priebežných údajov.
- ALU (ALU): Vykonáva aritmetické a logické operácie.

- Násobička (Multiplier): Vyhradený modul na vykonávanie operácií násobenia.
- Vrchná časť procesora (CPU Top): Hlavný modul, ktorý spája dekodér, súbor registrov a moduly na vykonávanie.
- Dátová pamäť: Ukladá operandy a medzivýsledky výpočtov.

Na schéme sú znázornené aj hlavné vstupy a výstupy: signály CLK, RESET, BOOT ADDRESS a ERROR.

2.2 Dataflow Diagram procesora

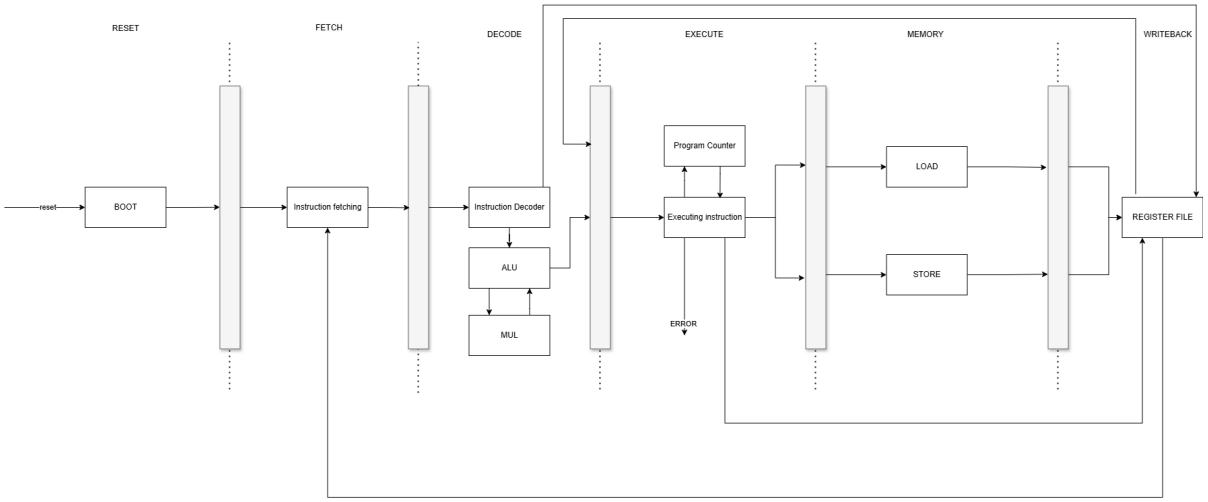

Figure 2.2: Dataflow Diagram

Tento diagram znázorňuje fázy vykonávania inštrukcií procesora od ich načítania až po zápis výsledku. Hlavné fázy sú:

- Reset: Procesor sa uvedie do počiatočného stavu.
- Získavanie inštrukcií (Fetch): Načíta inštrukciu z pamäte programu.
- Dekódovanie (Decode): Dekóduje načítanú inštrukciu a pripravuje moduly na vykonávanie (ALU, násobička).
- Execute (Vykonať): Vykoná inštrukciu pomocou vykonávacích jednotiek.
- Prístup do pamäte (Memory Access): Načíta alebo uloží údaje do dátovej pamäte.
- Spätný zápis (Writeback): Aktualizuje súbor registrov s výsledkami vykonania inštrukcie.

Schéma zdôrazňuje tok riadenia a údajov medzi jednotlivými fázami. Demonštruje prítomnosť chybových signálov a ich spracovanie.

2.3 State Machine Diagram procesora

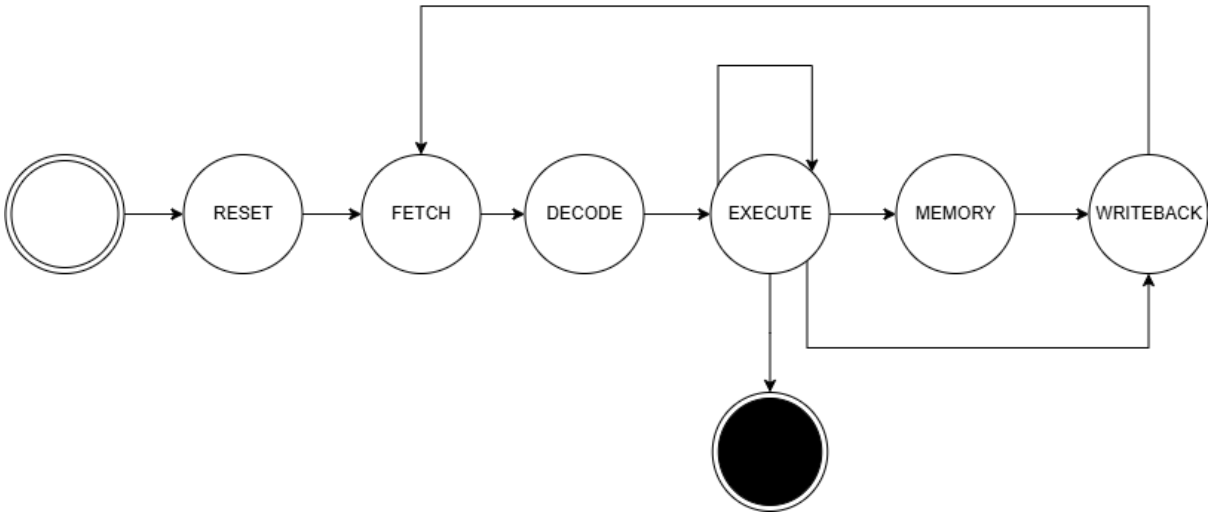

Figure 2.3: State Machine Diagram procesora

FSM má tieto stavy:

1. RESET
2. FETCH
3. DECODE
4. EXECUTE
5. MEMORY
6. WRITEBACK

Prechody medzi stavmi:

- RESET → FETCH: Po deaktivácii signálu reset (s_resetn_i).
- FETCH → DECODE: Po nastavení adresy vzorkovania inštrukcie.
- DECODE → EXECUTE: Po dekódovaní inštrukcie.
- EXECUTE → EXECUTE: Keď prebieha operácia násobenia (is_mul) a nie je dokončená (!alu_mul_done).
- EXECUTE → MEMORY: Ak sa vykonáva inštrukcia prístupu do pamäte (LW alebo SW).
- EXECUTE → WRITEBACK: Pre všetky ostatné inštrukcie.
- MEMORY → WRITEBACK: Ak je prístup do pamäte dokončený.

- WRITEBACK → FETCH: Keď sú výsledky zaznamenané a pripravené na vzorkovanie ďalšej inštrukcie.
- Akýkoľvek stav → RESET: Keď je aktívny signál reset (s_resetn_i = 0).
- EXECUTE → ERROR: Ak sa zistí neplatná inštrukcia (valid = 0).

2.4 State Machine Diagram ALU

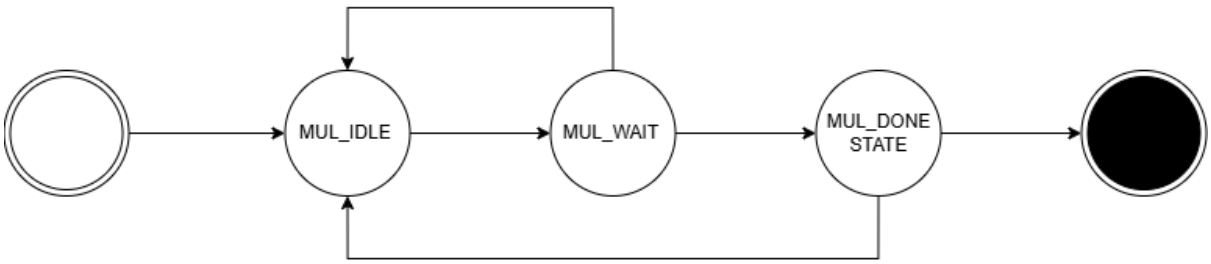

Figure 2.4: State Machine Diagram ALU

FSM má tieto stavy:

1. MUL_IDLE: Stav čakania na spustenie operácie násobenia.
2. MUL_WAIT: Stav čakania na dokončenie operácie násobenia.
3. MUL_DONE_STATE: Stav dokončenia operácie násobenia a prípravy výsledku.

Prechody medzi stavmi:

- MUL_IDLE → MUL_WAIT: Operácia násobenia sa začala (is_mul = 1) a násobička nie je obsadená (!s_busy_mult).
- MUL_WAIT → MUL_DONE_STATE: Operácia násobenia je dokončená (!s_busy_mult).
- MUL_WAIT → MUL_IDLE: Operácia násobenia zrušená (is_mul = 0).
- MUL_DONE_STATE → MUL_IDLE: Dokončenie operácie násobenia a žiadne ďalšie operácie násobenia (!is_mul alebo is_mul a !s_busy_mult).
- Akýkoľvek stav → MUL_IDLE: Pri resete (s_reset = 0).

Chapter 3

Popisy modulov

3.1 cpu_top

Modul cpu_top je najvyššia úroveň procesora, ktorá integruje rôzne komponenty, ako sú dekodér inštrukcií, súbor registrov, aritmeticko-logická jednotka (ALU) a konečný stavový automat (FSM). Riadi vykonávanie inštrukcií, prístup do pamäte a interakciu s ostatnými modulmi v systéme.

3.1.1 Rozhrania modulu

Vstupy

- s_clk_i (1 bit): Taktovací signál.
- s_resetn_i (1 bit): Asynchrónny reset s aktívnou úrovňou.
- s_boot_add_i (32 bitov): Továrenská adresa, ktorá nastavuje počiatočnú hodnotu počítadla programov (PC) po resete.
- s_ibus_val_i (32 bitov): inštrukcia načítaná z pamäte inštrukcií.
- s_dbus_val_i (32 bitov): Údaje načítané z dátovej pamäte.

Výstupy

- s_error_o (1 bit): Chybový signál, nastavený na 1, keď sa zistí neplatná inštrukcia alebo nesprávny prístup do pamäte.
- s_ibus_write_o (1 bit): Signál zápisu do pamäte inštrukcií (v súčasnej implementácii sa nepoužíva).
- s_ibus_add_o (32 bitov): Adresa na načítanie ďalšej inštrukcie z pamäte inštrukcií.
- s_ibus_val_o (32 bitov): Údaje na zápis do pamäte inštrukcií (v súčasnej implementácii sa nepoužíva).
- s_dbus_write_o (1 bit): Signál pre zápis do dátovej pamäte.
- s_dbus_add_o (32 bitov): Adresa na prístup do dátovej pamäte.
- s_dbus_val_o (32 bitov): Hodnota na zápis do dátovej pamäte.

3.1.2 Interné komponenty

Konečný stavový automat (FSM)

Určuje aktuálny a nasledujúci stav procesora. Stavy:

- RESET: Inicializácia procesora.
- FETCH: načítanie inštrukcie z pamäte inštrukcií.
- DECODE: Dekódovanie prijatej inštrukcie.
- EXECUTE: Vykonanie inštrukcie.
- MEMORY: Prístup k dátovej pamäti (načítanie alebo uloženie).
- WRITEBACK: Zápis výsledkov vykonávania do registrov.

Program Counter(PC)

- pc (32 bitov): Ukladá aktuálnu adresu inštrukcie.
- pc_next (32 bitov): Nasledujúca hodnota programového čítača.

Register inštrukcií

- instruction_reg (32 bitov): Ukladá aktuálnu inštrukciu načítanú z pamäte inštrukcií.

Signály pre dekodér inštrukcií

- opcode (7 bitov): Kód operácie.
- rd, rs1, rs2 (po 5 bitov): čísla cieľového a zdrojového registra.
- funct3 (3 bity): Ďalšie polia na definovanie typu operácie.
- funct7 (7 bitov): Ďalšie polia na definovanie typu operácie.
- imm (32 bitov): Okamžitá hodnota (immmediate).
- valid (1 bit): Príznak platnosti inštrukcie.
- is_mul (1 bit): Príznak označujúci operáciu násobenia.

ALU

- alu_result (32 bitov): Výsledok vykonania operácie.
- alu_busy (1 bit): indikátor obsadenosti ALU (pri vykonávaní operácie násobenia).
- alu_mul_done (1 bit): Signál ukončenia operácie násobenia.

Register File

- reg_rd1, reg_rd2 (po 32 bitov): Hodnoty registrov načítané zo súboru registrov.
- reg_write_enable (1 bit): Signál povolenia zápisu do registra.
- reg_write_data (32 bitov): Údaje, ktoré sa majú zapísať do registra.

Registrácia údajov z pamäte

- memory_data_reg (32 bitov): Ukladá údaje načítané z dátovej pamäte na ďalší zápis do registrov.

3.1.3 Funkčnosť modulu

Riadenie stavu

Modul cpu_top riadi vykonávanie inštrukcií prostredníctvom konečného stavového stroja (FSM). Prechody medzi stavmi sú určené aktuálnou inštrukciou, signálmi z ALU a ďalšími podmienkami.

1. RESET
   - Popis: Počiatočný stav procesora, inicializácia všetkých potrebných registrov a programového čítača.
   - Prechod: Prepína do stavu FETCH, keď je deaktivovaný signál reset (s_resetn_i = 1).
2. FETCH
   - Popis: Nastaví adresu vzorkovania aktuálnej inštrukcie v rozhraní s pamäťou inštrukcií (s_ibus_add_o = pc).
   - Prechod: Po nastavení adresy sa prepne do stavu DECODE.
3. DECODE
   - Popis: Dekóduje prijatú inštrukciu, extrahuje operandy a riadiace signály.
   - Prechod: Po dekódovaní sa prepne do stavu EXECUTE.
4. EXECUTE
   - Popis: Vykonáva aritmetické a logické operácie pomocou ALU.
   - Prechody
     - Ak prebieha operácia násobenia (is_mul = 1) a nie je dokončená (!alu_mul_done), zostáva v stave EXECUTE.
     - Ak sa vykoná inštrukcia prístupu do pamäte (LW alebo SW), prejde do stavu MEMORY.
     - V opačnom prípade prejde do stavu WRITEBACK.
     - Ak je detekovaná neplatná inštrukcia (valid), nastaví chybový signál (s_error_o = 1).

5. MEMORY
    - Popis: Vykoná prístup do dátovej pamäte na načítanie (LW) alebo uloženie (SW).
    - Prechod: Po dokončení prístupu do pamäte sa prepne do stavu WRITEBACK.
6. WRITEBACK
    - Popis: Zapíše výsledky operácie do súboru registra.
    - Prechod: Po zápise prejde späť do stavu FETCH, aby načítal ďalšiu inštrukciu.

Spracovanie riadiacich signálov

- is_store (1 bit): Indikuje, že aktuálna inštrukcia je operácia ukladania do pamäte (SW).
- is_load (1 bit): Označuje, že aktuálna inštrukcia je operácia načítania z pamäte (LW).
- is_imm (1 bit): Označuje, že operácia používa okamžitú hodnotu (imm).
- alu_busy (1 bit): Označuje, že ALU je obsadená, napr. pri vykonávaní operácie viacnásobného násobenia.
- alu_mul_done (1 bit): Indikácia, že operácia násobenia ALU je dokončená.

Funkcia branch_taken

Funkcia branch_taken určuje, či sa má vykonať vetvenie v inštrukciách riadenia toku (BEQ, BNE, BLT, BGE, BLTU, BGEU).

Vstupy:

- funct3: Určuje typ vetvenia.
- reg_rd1, reg_rd2: Hodnoty registrov na porovnanie.

Logika:

- BEQ (funct3 = 000): Rozvetvenie sa vykoná, ak reg_rd1 == reg_rd2.
- BNE (funct3 = 001): Vetva sa vykoná, ak reg_rd1 != reg_rd2.
- BLT (funct3 = 100): Vetva sa vykoná, ak reg_rd1 je menšia ako reg_rd2 (porovnanie so znamienkom).
- BGE (funct3 = 101): Vetva sa vykoná, ak reg_rd1 je väčšia alebo rovná reg_rd2 (porovnanie so znamienkom).
- BLTU (funct3 = 110): Vetva sa vykoná, ak reg_rd1 je menšia ako reg_rd2 (porovnanie bez znamienka).
- BGEU (funct3 = 111): Vetva sa vykoná, ak reg_rd1 je väčšia alebo rovná reg_rd2 (porovnanie bez znamienka).
- V ostatných prípadoch sa vetva nevykoná (1'b0).

3.2 instruction_decoder

Modul instruction_decoder je zodpovedný za dekódovanie 32-bitových inštrukcií načítaných z pamäte inštrukcií a extrahovanie potrebných polí a riadiacich signálov z nich.

3.2.1 Rozhrania modulu

Vstupy

- inštrukcia (32 bitov): Obsahuje aktuálnu inštrukciu načítanú z pamäte inštrukcií. Inštrukcia je 32-bitové slovo zodpovedajúce jednému z formátov inštrukcií R-Type, I-Type, S-Type, B-Type, J-Type a U-Type.

Výstupy

- opcode (7 bitov): Základný opkód, ktorý definuje typ a kategóriu inštrukcie.
- rd (5 bitov): Číslo cieľového registra, do ktorého sa zapíšu výsledky vykonania inštrukcie.
- funct3 (3 bity): Nepovinné pole špecifikujúce typ operácie v rámci hlavnej kategórie definovanej opkódom.
- rs1 (5 bitov): Číslo prvého zdrojového registra použitého v operácii.
- rs2 (5 bitov): Číslo druhého zdrojového registra použitého v operácii (ak sa uplatňuje).
- funct7 (7 bitov): Nepovinné pole, ktoré špecifikuje typ operácie, najmä pre inštrukcie typu R.
- imm (32 bitov): Okamžitá hodnota (immmediate) používaná v inštrukciách, kde je to vhodné.
- valid (1 bit): príznak platnosti inštrukcie. Nastaví sa na 1, ak je inštrukcia rozpoznaná a platná, inak na 0.
- is_mul (1 bit): Príznak označujúci, že aktuálna inštrukcia je operácia násobenia (napr. MUL, MULH, MULHU, MULHSU).

3.2.2 Funkčnosť

1. Extrakcia polí inštrukcií: Na základe opcode určí formát inštrukcie (R-Type, I-Type, S-Type, B-Type, J-Type, U-Type) a z 32-bitového inštrukčného slova extrahuje príslušné polia (rd, funct3, rs1, rs2, funct7, imm).
2. Určenie platnosti inštrukcií: Pri spracovaní inštrukcie v predvolenom prípade (predvolené: begin) je príznak platnosti nastavený na 0, čo znamená, že ide o neplatnú alebo neznámu inštrukciu. V opačnom prípade je príznak valid nastavený na 1.
3. Definícia operácie násobenia: Pre inštrukcie typu R s funct7 rovnajúcou sa 7'b0000001 a funct3 zodpovedajúcou MUL, MULH, MULHU, MULHSU sa nastaví príznak is_mul na 1, čo znamená, že v ALU sa má vykonať operácia násobenia. Pre všetky ostatné inštrukcie je is_mul nastavená na 0.

3.3 alu

Modul alu (aritmeticko-logická jednotka) vykonáva aritmetické a logické operácie potrebné na vykonávanie inštrukcií procesora. Okrem toho podporuje operácie násobenia, ktoré môžu trvať niekoľko taktov. Modul je riadený riadiacimi signálmi z vyššej úrovne (cpu_top) a komunikuje s ostatnými systémovými komponentmi prostredníctvom svojich rozhraní.

3.3.1 Rozhrania modulu

Vstupy

- s_clk (1 bit): Hodinový signál.
- s_reset (1 bit): Signál resetovania, aktívny nízky.
- a (32 bitov): Prvý operand na vykonanie operácie.
- b (32 bitov): Druhý operand na vykonanie operácie.
- funct3 (3 bity): Dodatočné polia určujúce typ operácie.
- funct7 (7 bitov): Ďalšie polia špecifikujúce typ operácie.
- imm (32 bitov): Okamžitá hodnota (immediate) používaná v niektorých inštrukciách.
- is_imm (1 bit): Príznak označujúci použitie okamžitej hodnoty.
- is_store (1 bit): Príznak označujúci operáciu uloženia do pamäte (SW).
- is_load (1 bit): Príznak indikujúci operáciu načítania z pamäte (LW).
- is_mul (1 bit): Príznak indikujúci operáciu násobenia.

Výstupy

- busy(1 bit): Indikátor, že ALU je obsadená.
- result (32 bitov): Výsledok operácie.
- mul_done (1 bit): Signál indikujúci ukončenie operácie násobenia.

3.3.2 Funkčnosť

1. Vykonávanie aritmetických a logických operácií: Modul alu vykonáva rôzne aritmetické a logické operácie s operandmi a a b alebo okamžitou hodnotou imm v závislosti od riadiacich príznakov. Typ operácie je definovaný poľami funct3 a funct7. Medzi základné operácie patria:
    - Sčítanie (ADD) a odčítanie (SUB): Definované kombináciou polí funct3 a funct7.
    - Logické operácie (AND, OR, XOR): Vykonávajú sa na základe príslušných polí.

- Posuny (SLL, SRL, SRA): Vykonávajú sa na základe polí funct3 a funct7.
- Porovnávanie (SLT, SLTU): Vykonávajú sa na nastavenie príznakov na základe výsledkov porovnania.

2. Vykonávanie operácií násobenia: Keď je aktívny príznak is_mul, modul iniciuje operáciu násobenia prechodom cez niekoľko stavov konečného stavového stroja:
   - MUL_IDLE → MUL_WAIT: Keď ALU zistí potrebu násobenia (is_mul = 1) a násobička nie je obsadená (!s_busy_mult), ALU začne operáciu násobenia.
   - MUL_WAIT: Čakanie na dokončenie násobenia. V tomto čase je hodnota busy nastavená na 1, čo znamená, že ALU je obsadená.
   - MUL_WAIT → MUL_DONE_STATE: Keď je násobenie dokončené (!s_busy_mult), výsledok sa spracuje a uloží.
   - MUL_DONE_STATE → MUL_IDLE: Dokončenie operácie násobenia a príprava na ďalšiu operáciu.

   Signál mul_done sa nastaví na 1, keď je operácia násobenia dokončená, čo signalizuje vyššej úrovni, že výsledok je pripravený.

3. Kontrola obsadenosti: Signál obsadenia signalizuje, že ALU je obsadená vykonávaním operácie násobenia alebo inej dlhotrvajúcej operácie. Tým sa zabráni spusteniu nových operácií, kým sa aktuálne operácie neukončia.

3.4 multiplier

Modul násobičky je určený na vykonávanie operácií násobenia dvoch 32-bitových čísel, ktoré poskytujú 64-bitový výsledok. Implementuje algoritmus násobenia pomocou metódy shift-add, ktorá umožňuje vykonať násobenie v niekoľkých hodinových cykloch. Modul podporuje možnosť zrušiť aktuálnu operáciu násobenia.

3.4.1 Rozhrania modulu

Vstupy

- s_clk_i (1 bit): Hodinový signál.
- s_resetn_i (1 bit): Asynchrónny reset, aktívne nízky.
- s_compute_i (1 bit): Signál na spustenie operácie násobenia.
- s_cancel_i (1 bit): Signál na zrušenie aktuálnej operácie násobenia.
- s_multiplicand_i (32 bitov): Násobiteľ (prvý operand).
- s_multiplier_i (32 bitov): Násobiteľ (druhý operand).

Výstupy

- s_busy_o (1 bit): indikátor obsadenosti násobičky (aktívny počas vykonávania násobenia).
- s_result_o (64 bitov): Výsledok operácie násobenia.

3.4.2 Funkčnosť

1. Vykonávanie násobenia: Modul násobičky vykonáva násobenie dvoch 32-bitových čísel pomocou metódy posunu a sčítania. Táto metóda rozdeľuje operáciu násobenia na postupné kroky, z ktorých každý zahŕňa kontrolu najmenej významného bitu násobiteľa a príslušné pripočítanie kumulatívneho výsledku.

2. Riadenie stavu zaneprázdnenosti:

   - s_busy_o: Nastaví sa na 1 na začiatku operácie násobenia a zostáva aktívny, kým sa operácia neukončí alebo kým sa neprijme signál o zrušení.
   - s_result_o: Poskytuje 64-bitový výsledok násobenia po dokončení operácie.

3. Spracovanie zrušenia operácie: Modul podporuje možnosť zrušiť aktuálnu operáciu násobenia. Keď sa počas vykonávania násobenia prijme signál s_cancel_i, operácia sa okamžite preruší, registre sa vymažú a modul "alu" prejde do stavu čakania (MUL_IDLE).

3.5 register_file

Modul register_file implementuje súbor registrov procesora, ktorý poskytuje súbor registrov na všeobecné účely na ukladanie údajov a adries. Poskytuje možnosť čítania a zápisu údajov do registrov a zabezpečuje, aby register x0 vždy obsahoval hodnotu 0, ako je to bežné v architektúre RISC-V.

3.5.1 Rozhrania modulu

Vstupy

- s_clk (1 bit): Hodinový signál.
- s_reset (1 bit): Signál resetovania, aktívny na vysokej úrovni.
- rs1 (5 bitov): Adresa prvého zdrojového registra na čítanie.
- rs2 (5 bitov): Adresa druhého zdrojového registra na čítanie.
- rd (5 bitov): Adresa cieľového registra pre zápis.
- wd (32 bitov): Údaje, ktoré sa majú zapísať do cieľového registra.
- we (1 bit): Povolenie na zápis údajov do registra (1 je povolené, 0 je zakázané).

Výstupy

- rd1 (32 bitov): Údaje z registra adresovaného rs1.
- rd2 (32 bitov): Údaje z registra adresovaného rs2.

3.5.2 Funkčnosť

1. Resetovanie registrov: Keď je aktivovaný signál reset (s_reset = 1), všetky registre okrem x0 sú nastavené na 0. Register x0 vždy obsahuje 0 a nemení sa ani pri pokuse o zápis.

2. Zápis do registra: Ak je signál zápisu aktívny (we = 1) a ak cieľová adresa rd nie je 0 (x0), údaje wd sa zapíšu do registra s adresou rd. Register x0 zostáva nezmenený a vždy obsahuje 0.

3. Čítanie z registrov:

   - Čítanie rd1: Ak je rs1 0, rd1 sa nastaví na 0. V opačnom prípade rd1 načíta hodnotu z registra s adresou rs1.
   - Čítanie rd2: Ak je rs2 0, rd2 sa nastaví na 0. V opačnom prípade rd2 načíta hodnotu z registra s adresou rs2.

4. Záruka Hodnoty registra x0: Register x0 vždy obsahuje 0 bez ohľadu na operácie zápisu. Pokusy o zápis do x0 sa ignorujú, čím sa zabezpečí, že tento register zostane nezmenený.

Chapter 4

Demonštrácia činnosti CPU

4.1 Demonštrácia činnosti CPU pomocou ModelSim

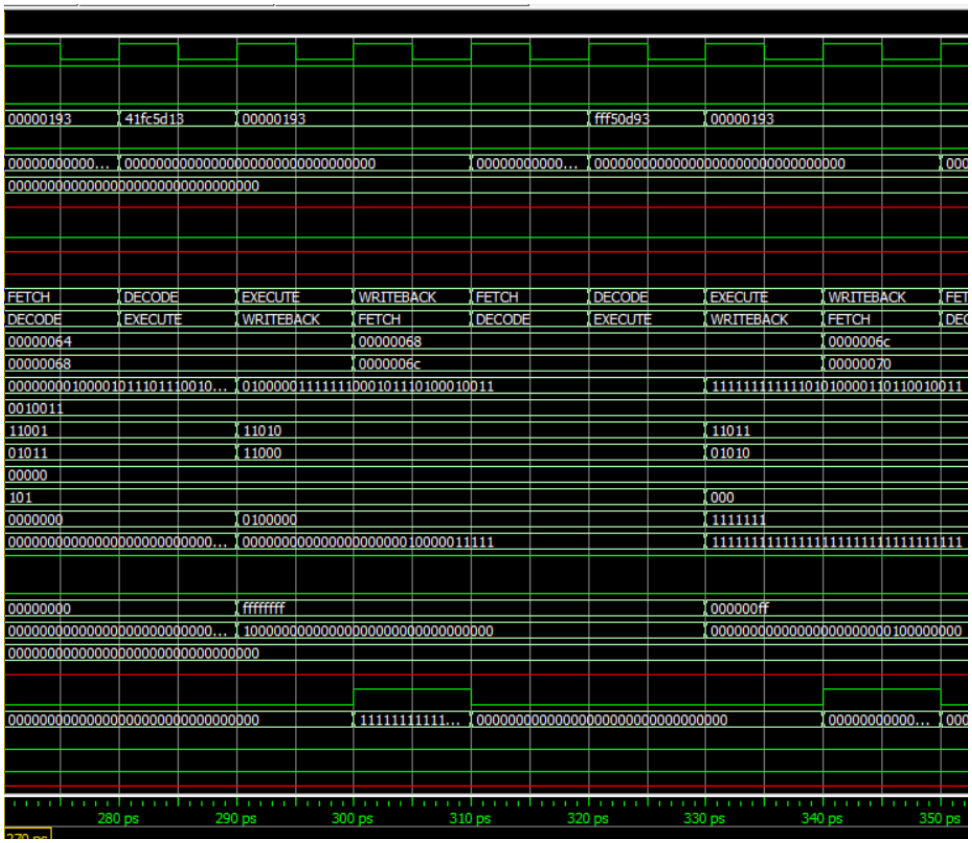

Figure 4.1: ModelSim

Pozriem sa na týchto 8 taktov, ktoré sú zobrazené na obrázku, aby som ukázal činnosť procesora.

4.1.1 Prvý takt

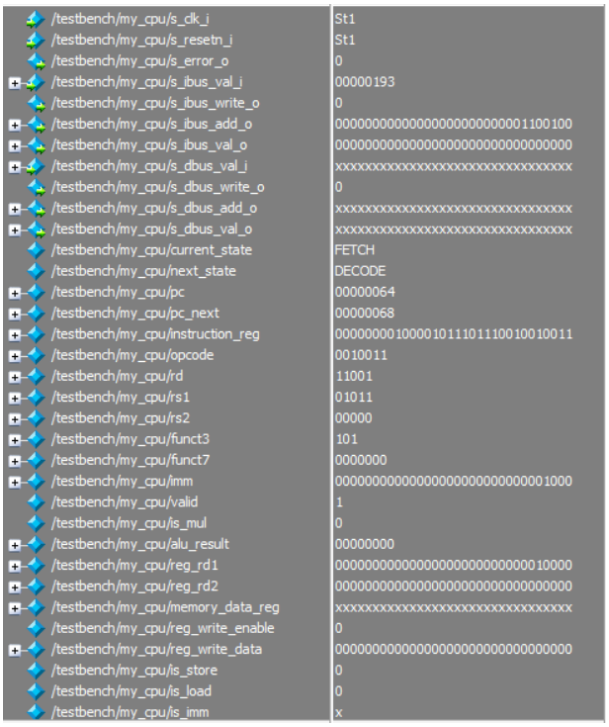

Figure 4.2: Prvý takt

Na začiatku sme v stave FETCH. Náš pc je na adrese 64, z ktorej budeme prijímať inštrukcie. Pošleme žiadosť o inštrukciu pomocou s_ibus_add_o. Inštrukcia bude v rozhraní s_ibus_val_i.

4.1.2 Druhý takt

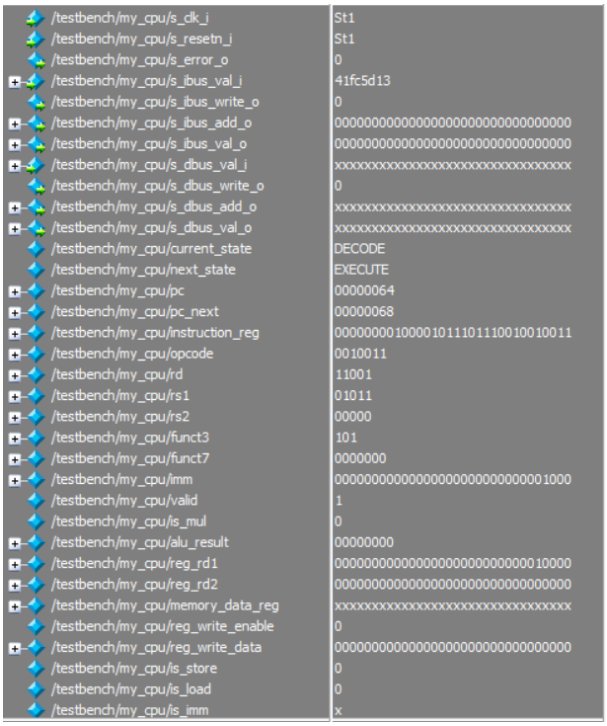

Figure 4.3: Druhý takt

Teraz sme v stave DECODE. Prijali sme inštrukciu do rozhrania s_ibus_val_i a teraz ju posielame na dekódovanie, aby sme z nej získali údaje.

4.1.3 Tretí takt

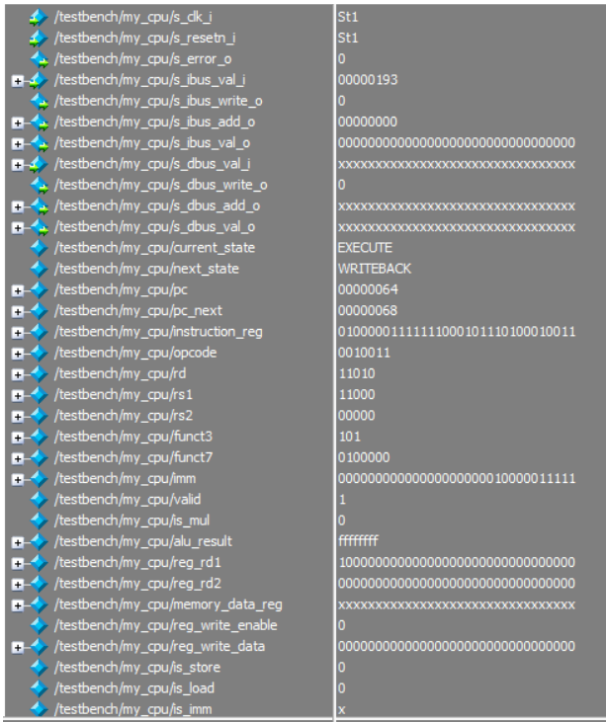

Figure 4.4: Tretí takt

Sme v stave EXECUTE. Prijali sme údaje z inštrukcie (napríklad opcode, rd, funct3, funct7, rs1, rs2, imm, valid, is_mul) a rozpoznali sme, že ide o inštrukciu typu I(SRAI), a poslali sme údaje do ALU a dostali sme odpoveď. Prejdeme do stavu WRITEBACK, aby sme zapísali do súboru registrov.

4.1.4 Štvrtý takt

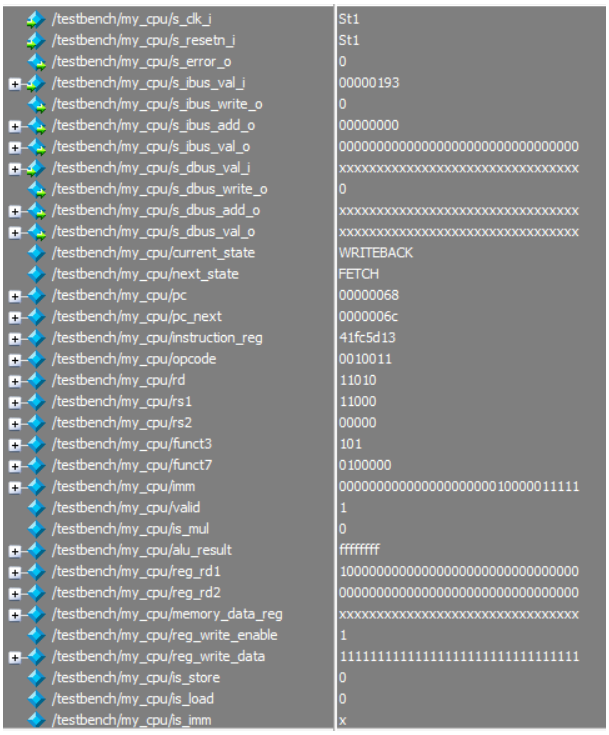

Figure 4.5: Tretí takt

Sme v stave WRITEBACK. V stave EXECUTE sme už zmenili pc na pc+4, čo teraz vidíte na obrázku, ako aj špecifikovali, čo chceme zapísať do súboru Register File pomocou reg_write_enable a špecifikovali konkrétne údaje, ktoré chceme zapísať pomocou reg_write_data. Prechádzame do stavu FETCH.

4.1.5 Piaty takt

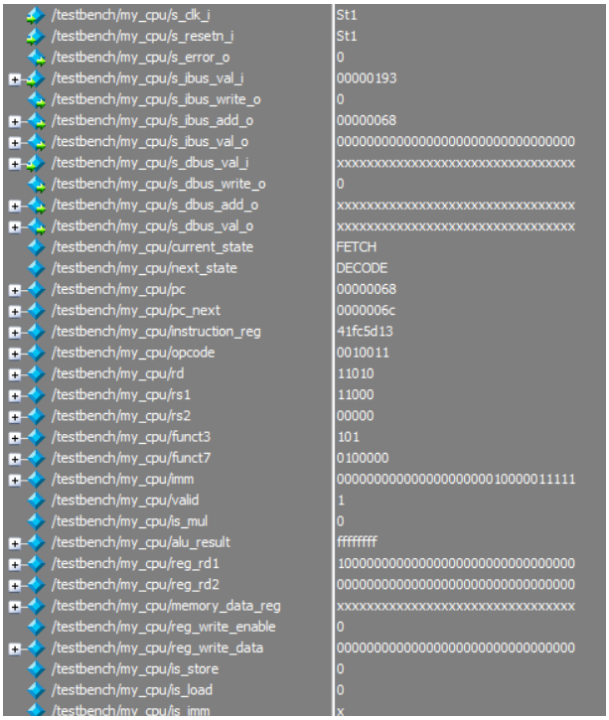

Figure 4.6: Piaty takt

Opäť sme v stave FETCH. Náš pc je na adrese 68, z ktorej budeme prijímať inštrukcie. Pošleme žiadosť o inštrukciu pomocou s_ibus_add_o. Inštrukcia bude v rozhraní s_ibus_val_i.

4.1.6 Šiesty takt

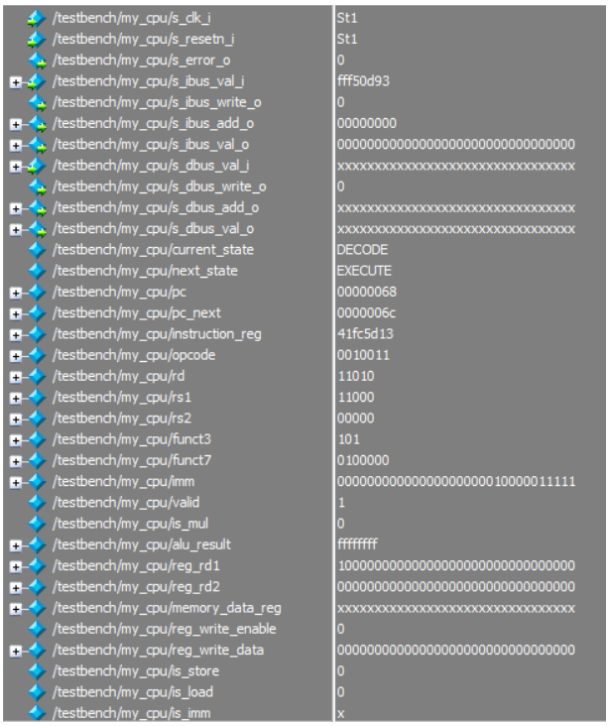

Figure 4.7: Šiesty takt

Teraz sme v stave DECODE. Prijali sme inštrukciu do rozhrania s_ibus_val_i a teraz ju posielame na dekódovanie, aby sme z nej získali údaje.

4.1.7 Siedmy takt

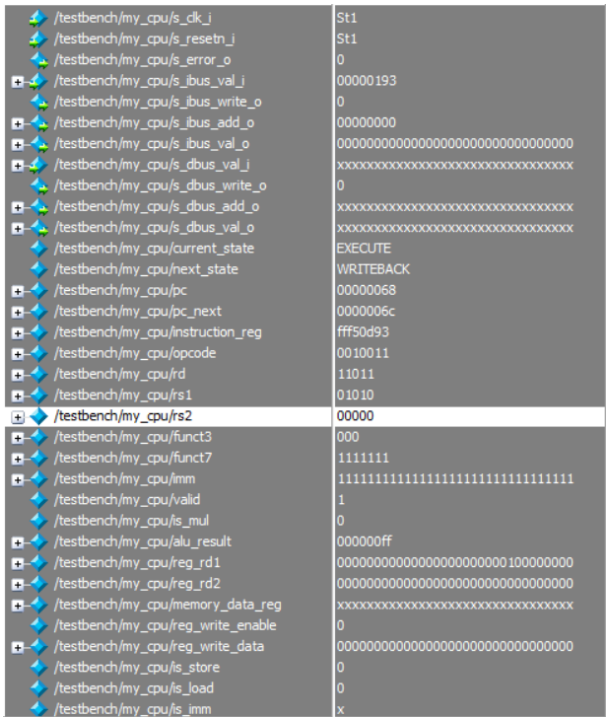

Figure 4.8: Siedmy takt

Sme v stave EXECUTE. Prijali sme údaje z inštrukcie (napríklad opcode, rd, funct3, funct7, rs1, rs2, imm, valid, is_mul) a rozpoznali sme, že ide o inštrukciu typu I(ADDI), a poslali sme údaje do ALU a dostali sme odpoveď. Prejdeme do stavu WRITEBACK, aby sme zapísali do súboru registrov.

4.1.8 Ôsmy takt

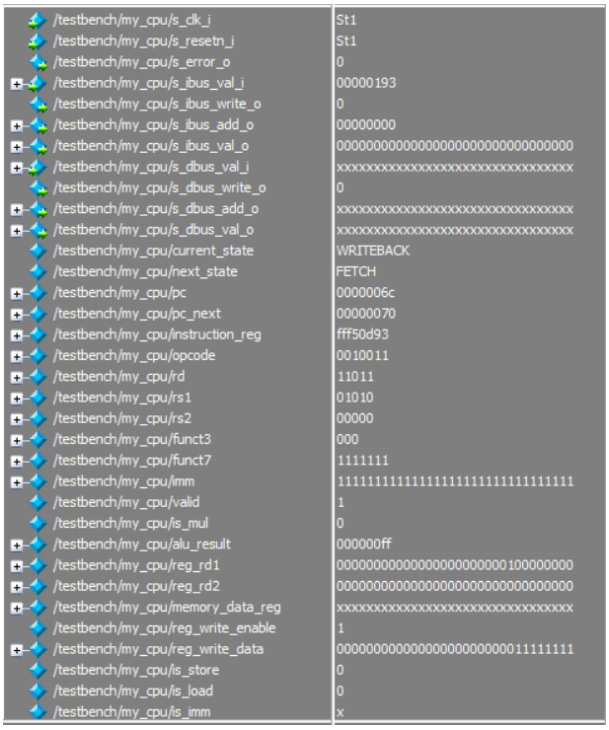

Figure 4.9: Ôsmy takt

Sme v stave WRITEBACK. V stave EXECUTE sme už zmenili pc na pc+4, čo teraz vidíte na obrázku, ako aj špecifikovali, čo chceme zapísať do súboru Register File pomocou reg_write_enable a špecifikovali konkrétne údaje, ktoré chceme zapísať pomocou reg_write_data. Prechádzame do stavu FETCH.

Chapter 5

Záver

V rámci tejto úlohy bol úspešne navrhnutý 32-bitový procesor založený na podmnožine inštrukcií RISC-V (RV32I+) s podporou operácií násobenia a harvardskou architektúrou s oddelenými zbernicami pre dáta a inštrukcie. Procesor korektne spracováva little-endian poradie ukladania a používa asynchrónny reset aktívny na nízkej úrovni s možnosťou nastavenia počiatočnej hodnoty softvérového čítača prostredníctvom špecializovaného štartovacieho rozhrania. Hlavné moduly vrátane cpu_top, instruction_decoder, alu, multiplier a register_file boli podrobne rozpracované a integrované, čo zabezpečuje efektívne vykonávanie fáz načítania, dekódovania, vykonávania, prístupu do pamäte a zápisu výsledkov. Mechanizmus spracovania chýb zabezpečuje zastavenie procesora a nastavenie chybového signálu s_error_o pri pokuse o dekódovanie neplatných inštrukcií alebo pri prístupe k nezarovnaným adresám pamäte, čo spĺňa požiadavky úlohy. Konečný stavový stroj (FSM) poskytuje explicitnú kontrolu nad procesom vykonávania inštrukcií vrátane viacerých vykonávacích cyklov pre operácie násobenia. Vyvinutý procesor tak preukazuje zhodu so zadanými špecifikáciami, efektívnu modulárnu štruktúru a robustnú funkčnosť.

Bibliography

[1] Andrew Waterman, Krste Asanovi
The RISC-V Instruction Set Manual, University of California, Berkeley, 2019.