DOCUMENTATIE

MIPS

NUME STUDENT: Pântea Marius-Nicușor
GRUPA:30222

Elemente Funcționale:

Monopulse Generator (MPG):

- Generează semnale de control pentru afișarea pe un SSD.

Display pe SSD (Segmente):

- Implementează afișarea pe un display cu șapte segmente.

Test_env:

- Mediu de test care conectează toate modulele.

ROM-instr (Read-Only Memory):

- Implementează o memorie program care stochează instrucțiuni.
- Aparține de IFetch

Blocul de registre (RegisterFile):

- Implementează un bloc de registre pentru stocarea datelor(spațiul de stocare cel mai frecvent folosit într-un procesor)

Unitate de Fetching a Instrucțiunilor (IFetch):

- Unitatea IFetch primește adresele de salt și furnizează adresa imediat următoare (PC+4) și conținutul instrucțiunii curente pe ieșiri
- Cu ajutorul intrărilor de control Jump și PCSrc se decide care va fi valoarea viitoare a registrului PC, în felul următor:
  - Dacă Jump = 1, atunci PC ; ← Jump Address;
  - Dacă Jump = 0, atunci:
    - Dacă PCSrc = 1, atunci PC← Branch Address;
    - Dacă PCSrc = 0, atunci PC ← PC+4.

Decodificator de Instrucțiuni (ID):

- Componenta are ca intrare: 3 flag-uri (RegWrite, RegDst, ExtOp), instrucțiunea curenta, și valoarea care va fi scrisa in registrul WD.
- Are ieșirile RD1(valoarea din registrul sursa) și RD2(valoarea din registrul destinație) , Ext_Imm(imediatul extins la 32 de biți), func(un câmp de 6 biți din instrucțiune, folosit pentru operațiile de tip R) ,sa (pentru operatiile de shiftare);
- Extinderea imediatului de la 16 biți la 32 de biți se realizează în funcție de semnalul de control ExtOp:
  - Dacă ExtOp = 0, atunci extinderea este cu zero (necesară la operații logice pe biți, cu valori constante);
  - Dacă ExtOp = 1, atunci extinderea este cu semn.
  - Dacă RegWrite = 1 atunci în registru se scrie valoarea de la WD, la adresa data de multiplexor.
  - Daca RegDst = 1 atunci se scrie in memorie la adresa pentru rd.(registrul destinatie);
  - Daca RegDst = 0 atunci se scrie in memorie la adresa pentru rt(registrul target);

Unitate de Control (UC):

- Aceasta interpretează primele 6 biți ai instrucțiunii primite și generează semnalele corespunzătoare pentru a ghida operațiunile celorlalte componente ale procesorului.
- Portul de intrare ale entității UC este Instr (intrarea care primește primi 6 biți ai instrucțiunii curente)
- Porturile de ieșire ale entității UC sunt: RegDst, ExtOp, ALUSrc, Branch, BranchGTZ, Jump, MemWrite, MemtoReg, RegWrite și valoarea pentru ALUOp
- Procesul din arhitectura UC primește instrucțiunea și, în funcție de valoarea acesteia, setează semnalele de control corespunzătoare.

Unitate de Execuție (EX):

- Realizează operații aritmetice și logice pe baza instrucțiunilor primite.
- Valori la intrare pentru a executa operațiile necesare. Acestea sunt: ALUSrc ,RD1 (registrul sursă), RD2(registrul destinatie), Ext_Imm(valoarea imediată extinsă la 32 de biți), sa(numărul de biți pentru operațiile

de shiftare) , func(câmpul care specifică operația pentru instrucțiunile de tip R), ALUOp(operația pentru instrucțiunile de tip non-R), PC_4(adresa PC + 4)

- În funcție de semnalul ALUSrc, valorile sunt selectate pentru a fi trimise pe intrarea unității ALU. Dacă ALUSrc este 1, este selectată valoarea imediată Ext_Imm; în caz contrar, se selectează valoarea din registrul destinație RD2.
- În funcție de funcția data pentru operațiile de tip R sau ALUOp, ALU execute operația dorită, pe ieșirea ALURes rezultatul, setează flag-urile: Zero, GTZ, și calculează Branch_Address.

Unitate de Memorie (MEM):

Entitatea MEM implementează o memorie cu un conținut predefinit și permite operații de citire și scriere asupra acestuia.

- Ea primește următoarele semnale și date la intrare: MemWrite(semnal care indică dacă se efectuează o operație de scriere în memorie), ALUResIn (valoarea care va fi folosită ca adresă pentru citirea sau scrierea în memorie), RD2(valoarea care va fi scrisă în memorie în cazul unei operații de scriere), EN(semnal de activare a operației de citire/scriere)
- Operații sunt activate doar atunci când semnalul EN este activ și MemWrite este setat la 1. În cazul în care operația este de scriere, valoarea din RD2 este scrisă la adresa corespunzătoare calculată din ALUResIn. În cazul operației de citire, conținutul memoriei la adresa calculată este trimis pe ieșirea MemData iar pe ALURes valoarea primită inițiala.

Write-Back (WB)

- Multiplexorul cu selecția data MemtoReg:
  - Dacă MemtoReg = 0, se trimite ALUResOut
  - Dacă MemtoReg = 1, se trimite MemData

Precizari:

- Toate elementele componentele au fost testate pe placa.

Probleme întâmpinate

- Erori în interconectarea modulelor sau a semnalelor
- Greșeli în logica de proiectare sau în implementarea modulelor
- Necorespunderea semnalelor între module
- Erori în procesul de scriere sau citire a datelor în/din registre
- Probleme în accesarea sau citirea corectă a instrucțiunilor din memorie ROM

Cele 4 instrucțiuni adăugate:

XOR – bitwise eXclusive-OR

- SAU-Exclusiv logic între două registre (sursa si target), memorează rezultatul in registrul destinație si incrementează program counter-ul
- $d ← $s ^ $t; PC ← PC + 4;
- xor $d, $s, $t
- 000000 sssss ttttt ddddd 00000 100110

SLT – Set on Less Than (signed)

- Instrucțiunea SLT compară două registre (sursa și target) și setează un al treilea registru (destinatie) la valoarea 1 dacă $s este mai mic decât $t și la valoarea 0 în caz contrar
- PC←PC + 4; if $s < $t then $d ← 1 else $d ← 0;
- slt $d, $s, $t
- 000000 sssss ttttt ddddd 00000 101010

ORI – bitwise OR Immediate

- Valoarea din registrul sursa este combinată folosind operația logică SAU cu o valoare imediată iar rezultatul este stocat în registrul $t, în timp ce program counter este incrementat pentru a trece la următoarea instrucțiune din cod în alt registru
- $t ←$s | ZE(imm); PC ←PC + 4;
- ori $t, $s, imm
- 001101 sssss ttttt iiiiiiiiiiiiiiii

BGTZ – Branch on Greater Than Zero

- Verifică dacă valoarea din registrul sursa este strict mai mare ca 0, iar daca da aduna la program counter offset-ul, echivalent cu numărul de instrucțiuni peste care să sară. Daca nu incrementează program counter-ul.
- If $s > 0 then PC □ PC + 4 + (SE(offset) << 2) else PC □ PC + 4;
- bgtz $s, offset
- 000111 sssss 00000 oooooooooooooooo

Semnale de control MIPS32

Semnale de Branch opționale: ? ϵ {gez, ne, gtz}, se va înlocui ? cu o valoare din paranteză, dacă e cazul

Tipuri de operații care se pun în paranteză la ALUOp si ALUCtrl:

(+), (-), (&), (|), (^), (<<*l*), (<<*lv*), (>>*l*), (>>*a*), (<)

Semnificații: & - AND, | - OR, ^ - *XOR, l - logic, a - aritmetic, v - cu variabilă*

Instrucțiune	**Opcode** *Instr[31-26]*	**Reg Dst**	**ExtOp**	**ALUSrc**	**Branch**	**Br_GTZ**	**Jump**	**Mem Write**	**Memto Reg**	**Reg Write**	**ALUOp[5:0]**	**function** *Instr[5-0]*	**ALUCtrl[5:0]**
ADD	000000	1	0	0	0	0	0	0	0	1	000000	100000	100000
SUB	000000	1	0	0	0	0	0	0	0	1	000000	100010	100010
SLL	000000	1	0	0	0	0	0	0	0	1	000000	000000	000000
SRL	000000	1	0	0	0	0	0	0	0	1	000000	000000	000000
AND	000000	1	0	0	0	0	0	0	0	1	000000	100100	100100
OR	000000	1	0	0	0	0	0	0	0	1	000000	100101	100101
XOR	000000	1	0	0	0	0	0	0	0	1	000000	100110	100110
SLT	000000	1	0	0	0	0	0	0	0	1	000000	101010	101010
ADDI	001000	0	1	1	0	0	0	0	0	1	000001	X	000001
ORI	001101	0	1	1	0	0	0	0	0	1	000010	X	000010
LW	100011	0	1	1	0	0	0	0	1	1	000001	X	000001
SW	101011	0	1	1	0	0	0	1	0	0	000001	X	000001
BEQ	000100	0	1	0	1	0	0	0	0	0	000100	X	000100
BGTZ	000001	0	1	0	0	1	0	0	0	0	000100	X	000100
Jump	000010	0	0	0	0	0	1	0	0	0	001000	X	001000

URL: https://drive.google.com/file/d/1SI7x2Gp_2m3SEkwnXuGt4ns4voYzpGBH/view?usp=sharing

Trasarea execuției programului de test pentru MIPS32

Valorile se completează în hexazecimal așa cum trebuie să apară pe SSD. Succesiunea pașilor reprezintă ordinea de execuție în timp la apăsarea butonului ENable. **Pasul 0 corespunde stării inițiale a circuitului (PC = 0), iar pasul *N* caracterizează starea după apăsarea de *N* ori a butonului ENable**. Inițial registrele vor avea valoarea 0 (care se atribuie automat în lipsa unei inițializări explicite a RF), iar memoria de date RAM poate fi inițializată cu valori dorite. Tabelul se completează pentru tot programul sau, dacă are buclă, până la finalul primei iterații. *Buclă = revenirea execuției la o instrucțiune care a mai fost executată anterior.*

Pas	SW(7:5)	"000"	"001"	"010"	"011"	"100"	"101"	"110"	"111"	De completat numai pentru instrucțiuni de salt	
	Instr (*în asamblare*)	**Instr** (*hexa*)	**PC+4**	**RD1**	**RD2**	**Ext_Imm**	**ALURes**	**MemData**	**WD**	**BranchAddr**	**JumpAddr**
0	xor $r10, $r10, $r10	X"014A5026"	X"00000001"	X"0000010"	X"0000010"	X"00000000"	X"0000000"	X"0000008"	X"0000000"	X"0000000"	X"0000000"
1	xor $r0, $r0, $r0	X"0000026"	X"00000002"	X"0000000"	X"0000000"	X"00000000"	X"0000000"	X"0000008"	X"0000000"		
2	xor $r30, $r30, $r30	X"03DEF026"	X"00000003"	X"0000030"	X"0000030"	X"00000000"	X"0000000"	X"0000008"	X"0000000"		
3	xor $r3, $r3, $r3	X"00631826"	X"00000004	X"0000003"	X"0000003"	X"00000000"	X"0000000"	X"0000008"	X"0000000"		
4	addi $r3, $r3, 0xFFFF	X"2063FFFF"	X"00000005"	X"0000000"	X"0000000"	X"0000FFFF"	X"0000FFFF"	X"0000000"	X"0000FFFF"		
5	addi $r30,$r30,4	X"23DE0004"	X"00000006"	X"0000000"	X"0000000"	X"0000004"	X"0000004"	X"0000000"	X"0000004"		
6	lw $r1, 0($r10)	X"8D410000"	X"00000007"	X"0000000"	X"0000001"	X"0000000"	X"000008"	X"0000008"	X"0000008"		
7	addi $r0, $r0, 8	X"20000008"	X"00000008"	X"0000000"	X"0000000"	X"0000008"	X"0000008"	X"0000000"	X"0000008"		
8	addi $r11, $r1, 32	X"202B0020"	X"00000009"	X"0000000"	X"0000008"	X"000001F"	X"0000028"	X"0000000"	X"0000028"		
9	beq $r1, $r11, offset 5	X"102B0005"	X"0000000A"	X"0000008"	X"0000028"	X"0000005"	X"0000008"	X"0000028"	X"0000008"	X"000000F"	
10	lw $r2, 0($r1)	X"8C220000"	X"0000000B"	X"0000000"	X"0000009"	X"0000000"	X"0000008"	X"0000004"	X"0000004"		
11	and $r2, $r2, $r3	X"00431024"	X"0000000C"	X"00000004"	X"0000FFFF"	X"00000000"	X"0000000"	X"0000000"	X"0000004"		
12	add $r0, $r0, $r2	X"00020020"	X"0000000D"	X"00000000"	X"00000004"	X"00000000"	X0000000""	X"0000000"	X"0000004"		
13	addi $r1, $r1, 4	X"20210004"	X"0000000E"	X"0000008"	X"0000008"	X"0000004"	X"000000C"	X"0000002"	X"000000C"		
14	j offset 9	X"08000009"	X"0000000F"	X"0000000"	X"0000000"	X"0000000"	X"0000009"	X"0000002"	X"0000000"		X"0000009"
15	sw $r0, offset($r30)	X"AFC00000"	X"00000010"	X"0000060"	X"0000004"	X"0000000"	X"0000060"	X"0000000"	X"0000060"		
16											
17											
18											
19											
20											
21											
22											
24											

URL: https://drive.google.com/file/d/1OgoST1-tEe1cbUdNk_VKr6NHq3zVfs83/view?usp=sharing