Documentatie

Nicoara Cristian-Cătălin

Grupa 30228

1. *Intructiuni Suplimentare:*
2. MUL $rs, $rt – inmultirea dintre doua registre (tip R)

RTL: Hi,Lo <- RF[rt]\*RF[rt], insa in vhdl am salvat rezultatul intr-o variabila de 32 de biti, si trimitem in ALURes bitii de la 15 la 0;

1. XOR $rd, $rs, $rt – sau-exclusiv cu un registru (tip R)

RTL: RF[rd] <- RF[rs] ^ RF[rt]

1. XORI $rs, $rt, imm – sau-exclusiv cu un imediat (tip I)

RTL: RF[rt] <- RF[rs] ^ Z\_ext(imm), unde Z\_ext reprezinta extindrea cu 0. Pentru asta, la operatia data, in Main Control, setam bitul de ExtOp pe ‘0’.

1. SUBI $rs, $rt, imm – scadrea cu un imediat (tip I)

RTL: RF[rt] <- RF[rs] - S\_ext(imm)

1. *Tabelul cu valorile semnalelor de control*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instr** | **Opcode** | **RegDst** | **ExtOp** | **AluSrc** | **Branch** | **Jump** | **MemWrite** | **MemToReg** | **RegWrite** | **AluOp** | **func** | **AluCtrl** |
| add | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | 000 | 000 |
| sub | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | 001 | 001 |
| sll | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | 010 | 010 |
| srl | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | 011 | 011 |
| and | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | 100 | 100 |
| or | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | 101 | 101 |
| mult | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | 111 | 111 |
| xor | 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | 110 | 110 |
| addi | 001 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 01 | xxx | 000 |
| lw | 010 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 01 | xxx | 000 |
| sw | 011 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 01 | xxx | 000 |
| beq | 100 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 10 | xxx | 100 |
| xori | 101 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 11 | xxx | 110 |
| subi | 110 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 10 | xxx | 001 |
| j | 111 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 00 | xxx | xxx |

1. *Coduri pentru funtia aleasa*
2. Cod in C

for (int i = 1; i <= 50; i++) {

int nr\_xor = i ^ 1;

if (nr\_xor == i + 1) {

sum\_pare += i;

}

else

sum\_impare += i;

}

1. Cod in MIPS 16

0 ADDI R2,R0,50 001 000 010 0110010 2132(hexa)

1 ADDI R2,R2,1 001 010 010 0000001 2901

2 ADDI R1,R0,1 001 000 001 0000001 2081

3 BEQ R1,R2,8 100 001 010 0001000 8508

4 ADDI R7,R1,1 001 001 111 0000001 2781

5 XORI R6,R1,1 101 001 110 0000001 A701

6 BEQ R7,R6,2 100 111 110 0000010 9F02

7 ADD R5,R5,R1 000 101 001 101 0 000 14D0

8 J 10 111 0000000001010 E00A

9 ADD R4,R4,R1 000 100 001 100 0 000 10C0

10 ADDI R1,R1,1 001 001 001 0000001 2481

11 J 3 111 0000000000011 E003

12 SW R4,2(R0) 011 000 100 0000010 6202

13 SW R5,3(R0) 011 000 101 0000011 6283

Codul parcuge sirul de numere de la 1 la 50 si verifica folosind XOR daca un numar este par sau impar. Daca un numar n xor 1 = n + 1, atunci acesta este par. Algoritmul, dupa verificare, face suma numerelor pare si impare din sir si le salveaza in doua variabile separate.

1. *Trasarea executiei*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pas** | **sw(7:5)** | **“000”** | **“001”** | **“010”** | **“011”** | **“100”** | **“101”** | **“110”** | **“111”** | **Adrese de salt** | |
| **Instr** | **Instr (hexa)** | **pc** | **rd1** | **rd2** | **ext\_imm** | **AluRes** | **MemData** | **wd** | **Branch** | **Jump** |
| 0 | ADDI R2,R0,50 | 2132 | 0001 | 0000 | 0000 | 0032 | 0032 | 0000 | 0032 |  |  |
| 1 | ADDI R2,R2,1 | 2901 | 0002 | 0032 | 0032 | 0001 | 0033 | 0000 | 0033 |  |  |
| 2 | ADDI R1,R0,1 | 2081 | 0003 | 0000 | 0000 | 0001 | 0001 | 0000 | 0001 |  |  |
| 3 | BEQ R1,R2,8 | 8508 | 0004 | 0001 | 0033 | 0008 | FFCE | 0000 | FFCE | 0008 |  |
| 4 | ADDI R7,R1,1 | 2781 | 0005 | 0001 | 0000 | 0001 | 0002 | 0000 | 0002 |  |  |
| 5 | XORI R6,R1,1 | A701 | 0006 | 0001 | 0000 | 0001 | 0000 | 0000 | 0000 |  |  |
| 6 | BEQ R7,R6,2 | 9F02 | 0007 | 0002 | 0000 | 0002 | 0002 | 0000 | 0002 | 0002 |  |
| 7 | ADD R5,R5,R1 | 14d0 | 0008 | 0000 | 0001 | 0050 | 0001 | 0000 | 0001 |  |  |
| 8 | J 10 | E00A | 0009 | 0000 | 0000 | 000A | 0000 | 0000 | 0000 |  | 000A |
| 9 | ADD R4,R4,R1 | 10C0 | 000A | 0000 | 0002 | 0040 | 0002 | 0000 | 0002 |  |  |
| 10 | ADDI R1,R1,1 | 2481 | 000b | 0001 | 0001 | 0001 | 0002 | 0000 | 0002 |  |  |
| 11 | J3 | E003 | 000C | 0000 | 0000 | 0003 | 0000 | 0000 | 0000 |  | 0003 |
| 12 | SW R4,2(R0) | 6202 | 000d | 0000 | 028A | 0002 | 0002 | 0000 | 0002 |  |  |
| 13 | SW R5,3(R0) | 6283 | 000E | 0000 | 0271 | 0003 | 0003 | 0000 | 0003 |  |  |

* Datele pentru ADD R4,R4,1 sunt din a doua interatie, cand functia a ajuns la prima cifra para(2).
* Datele pentru instructiunile de SW sunt din ultima iteratie, cand iese din bucla, iar in rd2 se observa suma numerelor pare (028A) si suma numerelor impare (0271).
* Pentru restul intructiunilor, datele din tabel sunt din prima iteratie.
* Explicatia pe pasi a codului assembly:

1. Pas 0: Se salveaza in registrul R2 valoarea 50 (pana la cat va ajunge bucla)
2. Pas 1: Se adauga la registrul R2 valoare 1, deci devine 51, pentru ca atunci cand se va calcula suma numerelor pare, sa fie inclus si 50
3. Pas 2: Se salveaza in registrul R1, valoarea 1 (de unde se incepe iteratia)
4. Pas 3: Se verifica daca R1 (registrul iterator) a ajuns la finalul buclei. Daca da, va sari peste 8 instructiuni. Daca nu, se va continua cu pasul urmator.
5. Pas 4: Se salveaza in registrul R7 valoarea registrului R1 incrementata cu 1.
6. Pas 5: Se salveaza in registrul R6 rezultaul operatie R1 xor 1
7. Pas 6: Se verifica daca valorile registrelor R6 si R7 sunt egale. Daca da, atunci numarul din registrul R1 este par si se sare peste 2 intructiuni. Daca nu, atunci numarul din registrul R1 este impar si se trece la pasul urmator.
8. Pas 7: Se aduna la valoarea registului R5 (initial este 0), numarul din R1 daca este impar
9. Pas 8: Se sare la pasul 10, pentru a nu se aduna la suma numerelor pare un numar impar
10. Pas 9: Aici se sare de la pasul 6, daca are loc egalitatea, unde se aduna la valoarea registului R4 (initial este 0), numarul din R1 daca este par
11. Pas 10: Se incrementeaza valoarea din registrul R1
12. Pas 11: Se sare la pasul 3, unde se face verificarea daca s-a ajuns la finalul buclei.
13. Pasul 12: Aici se sare de la pasul 3, daca are loc egalitatea. Se salveaza la adresa 2 din memorie, valoarea din registrul R4, adica suma numerelor pare din interval.
14. Pasul 13: Se salveaza la adresa 3 din memorie, valoarea din registrul R5, adica suma numerelor impare din interval

* Am testat codul pe placa de mai multe ori, si testele au functionat bine.