

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

**Procesorul MIPS, ciclu unic – versiune pe 16 biți**

**Runcan Nicoleta**

**Grupa 30229**

An 2019-2020

***Descrierea Procesorului***

* **Cele 4 instructiuni descrise**

**Tip R:**

1. **Instructiunea XOR**

**xor $rd, $rs, $rt**

**Explicatie:** aceasta instructiune presupune efectuarea unui SAU exclusive logic intre registrii $rs si $rt si salvarea rezultatului intr-un alt registru($rd)

**RTL Abstract:**

RF[rd] 🡨RF[rs] ^ RF[rt]

**Format:**

000\_sss\_ttt\_ddd\_0\_110

**Exemplu:**

xor $2, $3, $4

000\_011\_100\_010\_0\_110

1. **Instructiunea SRLV(**Shift left logical variable)

**Srlv $rd, $rs, $rt**

**Explicatie:** Aceasta operatie realizeaza deplasarea logica la dreapta pentru un regsitru($rs) cu un numar de pozitii indicat de un alt registru($rt), iar rezultatul este memorat in registrul $rd

**RTL Abstract:**

RF[rd] 🡨RF[rs] >> RF[rt]

**Format:**

000\_sss\_ttt\_ddd\_0\_111

**Exemplu:**

Srlv $2, $3, $4

000\_011\_100\_010\_111

**Tip I:**

1. **Instructiunea ORI**

**Ori $rt, $rs, imm**

**Explicatie:** Instructiunea efectueaza un SAU logic între un registru($rs) și o valoare imediată(imm), dupa care memorează rezultatul în alt registru($rt)

**RTL Abstract:**

RF[rt] 🡨 RF[rs] | Z\_Ext(imm)

**Format:**

101\_sss\_ttt\_iiiiiii

**Exemplu:**

Ori $2, $3, 9

101\_011\_010\_0001001

1. **Instructiunea BNE(**Branch on not equal)

**Bne $rs, $rt, imm**

**Explicatie:** Se face salt daca cele doua register($rs, $rt) sunt diferite.Opusul instructiunii de BNE

**Format:**

110\_sss\_ttt\_iiiiiii

**Exemplu:**

Bne $1, $2, 9

110\_001\_010\_0001001

**Obs**: Aceasta instructiune o folosesc si in program la verificarea paritatii numerelor dintr-un sir.

* **Tabelul valorilor semnalelor de control**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instructiune** | **Opcode** | **RegDst** | **ExtOp** | **ALUsrc** | **Branch** | **Jump** | **Mem**  **Write** | **Memto**  **Reg** | **Reg**  **Write** | **AluOp**  **(2:0)** | **funct** | **AluCtrl**  **(2:0)** | **BranchNE** |
| Add | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 000 | 000(+) | 0 |
| Sub | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 001 | 001(-) | 0 |
| Sll | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 010 | 010(<l) | 0 |
| Srl | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 011 | 011(>l) | 0 |
| And | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 100 | 100(&) | 0 |
| Or | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 101 | 101(|) | 0 |
| xor | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 110 | 110(^) | 0 |
| srlv | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 111 | 111  (>>lv) | 0 |
| addi | 001 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 001(+) | - | 000(+) | 0 |
| Lw | 010 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 001(+) | - | 000(+) | 0 |
| Sw | 011 | X | 1 | 1 | 0 | 0 | 1 | x | 0 | 001(+) | - | 000(+) | 0 |
| Beq | 100 | X | 1 | 0 | 1 | 0 | 0 | x | 0 | 011(-) | - | 001(-) | 0 |
| Ori | 101 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 100(|) | - | 101(|) | 0 |
| Bne | 110 | X | 1 | 0 | 0 | 0 | 0 | x | 0 | 011(-) | - | 001(-) | 1 |
| J | 111 | X | X | X | x | 1 | 0 | x | 0 | xxx | - | Xxx | x |

* **Descrierea programului**

**Descriere cod C:**

int sir[4] = { 3, 7, 8, 1 };

int suma = 0;

int a = 1;

for (int i = 0; i < 4; i++)

{

if (sir[i] & a == 1)

{

suma += (sir[i] - a);

}

}

**Descriere in cuvinte:**

In programul scris, se declara un sir cu 4 elemente, se parcurge sirul, verificandu-se paritatea fiecarui element; daca acesta este impar, se aduna valoarea acestuia scazuta cu 1 la o variabila suma, initializata la incepututl programului cu 0.

**0 add $1, $0,$0 -contorul buclei**

**1 addi $2, $0, 4 -numarul maxim de iteratii**

**2 or $3, $0, $0 -initializare index**

**3 add $4, $0, $0 -suma=0**

**4 addi $7, $0, 1 --salvam valoarea 1 in registru $7**

**5 beq $1, $2, 8 --verificam daca s-au facut 4 operatii, daca da, se face un salt la**

**Final, daca nu, programul continua cu instructiunea urmatoare**

**6 lw $6, 2($3) --in $6 se aduce elemental curent din sir**

**7 and $5, $6, $7 –se face un si logic intre valoarea din $6 si intre 1 salvat in $7, iar**

**iar rezultatul se salveaza in $5**

**8 bne $5, $7, 2 --verificam daca rezultatul operatiei de SI logic este egala cu 1,**

**daca nu este, inseamna ca numarul este par si se face un salt in**

**program, daca da se continua cu instructiunea urmatoare**

**9 sub $6, $6, $7 --se scade 1 din elemental curent**

**10 add $4, $4, $6 --se aduna elemntul current din care a fost scazut 1 la suma**

**11 addi $3, $3, 1 --indexul urmatorului element din sir**

**12 addi $1, $1, 1 --se incrementeaza contorul buclei**

**13 j 5 --un salt la inceputul buclei**

**14 sw $4, 7($0) --se salveaza suma in memorie la adresa 7**

Sirul se afla in memoria de date, primul element fiind la adresa 2, iar urmatoarele la adrese consecutive. Variabila suma este stocata in registrul 4. Se salveaza un index si un contor al buclei in cate un registru. Cu ajutorul instructiunii beq verificam daca sirul a fost parcurs in intregime. Paritatea elementelor din sir, o aflu astfel: stochez valoarea 1 intr-un registru 0; in momentul in care am adus in registru $6 valoarea elemntului curent din sir, o sa se efectueze un si logic intre registrul acesta si registrul in care a fost stocata valoarea 1, iar rezultatul acestei operatii este salvat intr-un registru liber. Dupa care, cu ajutorul instructiunii bne verificam daca aceast rezultat este sau nu egal cu 1, daca este inseamna ca elementul respective din sir este impar si atunci o sa scada 1 din el, dupa care o sa se adune la suma, daca nu este egal cu 1 se va face un salt peste instructiunile de scadere si adaugare la suma si se va trece la indexul urmator si se va relua bucla. La sfarsitul buclei, valoarea sumei este memorata la adresa 7.

* **Trasarea executiei programului**

-prima iteratie:

**0 add $1, $0,$0 --0010** --RD1=0, RD2=0, AluRes=0 –initializare contor

**1 addi $2, $0, 4 --2204 --**RD1=0; Ext\_imm=4, AluRes=4 –lungimea sirului

**2 or $3, $0, $0 --0035 --**RD1=0, RD2=0, AluRes=0, --index

**3 add $4, $0, $0 --0040 --**RD1=0, RD2=0, AluRes=0, --suma=0

**4 addi $7, $0, 1 --2381 --**RD1=0, Ext\_imm=1, AluRes=1, --stocam 1 in reg $7

**5 beq $1, $2, 8 --8508 --**RD1=4,Rd2=0 Ext\_imm=8, PC+1=6, --nu se face saltul inca

**6 lw $6, 2($3) --4B02 --**RD1=0, Ext\_imm=2, MemData=3 –se ia primul element

din sir

**7 and $5, $6, $7 --1BD4 --**RD1=3, RD2=1; AluRes=1

**8 bne $5, $7, 2 --7782** --RD1=1, RD2=1, PC+1=9 –nu se face salt

**9 sub $6, $6, $7 --1BE1 --**RD1=3 RD2=1, AluRes=2 se scade 1 din elementul curent

**10 add $4, $4, $6 --1340 --**RD1=0, RD2=2, AluRes=2 se adauga elementul in suma

**11 addi $3, $3, 1 --2D81 --**RD1 =0, Ext\_imm=1, AluRes=1

**12 addi $1, $1, 1 --2E00 --**RD1=1, Ext\_imm=1, AluRes=1

**13 j 6 -- E006 --**Ext\_imm=6, --salt, se va relua bucla, deoarece nu a avut l**oc** saltul de branch, adica nu am terminat de parcurs sirul

**14 sw $4, 7($0) --6207**

* **RTL\_Schematic**

