A black background with red lines

Description automatically generated

**Mips 32 ciclu unic**

Pentru : **Arhitectura Calculatoarelor.**

Student : **Georgi Emanuel.**

Cuprins

1. **Specificația proiectului.............................................2**
   1. Introducere....................................................2
   2. Descirere procesor **MIPS 32**............................3
   3. Programul de test ...........................................4
2. **Descriere instrucțiuni................................................5**
   1. Tipuri de instrucțiuni, formatul instrucțiunilor...5
   2. Instrucțiuni suplimentare................................6
   3. Scrierea în asamblare, RTL, formatul binar.......7
   4. Semnale de control…………………………………..10
3. **Elemente funcționale și probleme............................11**

1. Specificația proiectului

1.1 Introducere

A diagram of a computer

AI-generated content may be incorrect.În acest proiect s-a realizat implementarea procesorului MIPS ciclu unic pe 32 biți. Pentru programul de test am ales realizarea sumei numerelor impare dintr-un interval. Pentru a rezolva problema, am decis să stochez numerele în unitatea de memorie. Astfel la fiecare pas, fac un load word, verific paritatea și îl adaug în sumă dacă este cazul. De asemenea am modificat procesorul astfel încât să poată realiza instrucțiunile suplimentare alese, aceste fiind : Logical Xor, Shift Right Arithmetic (with shift amount - sa) pentru instrucțiuni de tip R, respectiv And Immediate și Branch on Not Equal pentru instrucțiuni de tip I. Rezultă schema completă a procesorului MIPS 32, calea de date și unitatea de control din următoarea imagine.

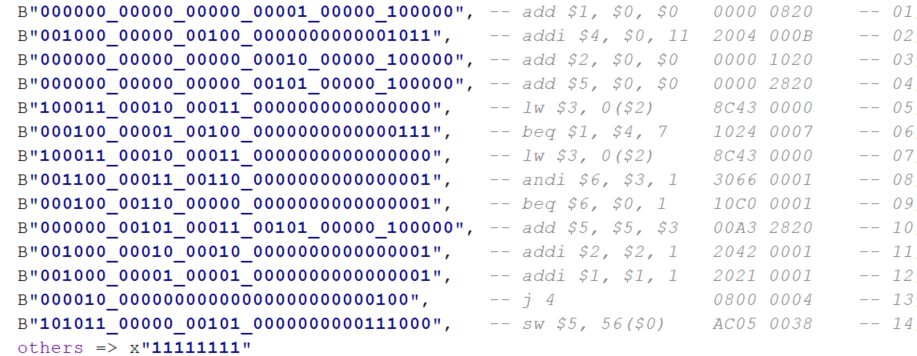
1.2 Descrierea procesorului **MIPS 32**

În ceea ce privește setul de instrucțiuni, procesorul MIPS pe 32 de biți are o ISA(Arhitectura setului de instrucțiuni) cu 3 adrese, bazată pe regiștri de uz general. Acest procesor are 32 de registre de uz general, toate acestea fiind grupate într-un bloc de registre și sunt identificate în mod unic prin indexul lor în bloc. Registrul de la adresa 0 va fi întotdeauna 0, scrierea fiind permisă în acest registru, dar valoarea lui nu se modifică niciodată. Memoria este cu adresare pe octet și avem 2^32, 4 GB, organizată pe cuvânt de 32 de biți, care sunt stocate la adrese multiple de 4 octeți în memorie. Există 3 tipuri de instrucțiuni : de tip R, tip I și tip J, fiecare instrucțiune având formatul său specific. Pentru cele de tip R se face o operație între 2 registre, iar rezultatul este salvat tot într-un registru, cele de tip I au o valoarea imediată reprezentată pe 16 biți, care este extinsă la 32 biți, iar cele de tip J reprezintă instrucțiunile de salt la o anumită adresă.

Execuția unei instrucțiuni în cazul procesorului MIPS 32 ciclu unic are 5 etape:

1. IF – Extragerea instrucțiunii (Instruction Fetch)
2. ID – Decodificarea instrucțiunii / extragerea operanzilor(Instruction Decode)
3. EX – Execuția instrucțiunii(Execute)
4. MEM – Memorie(Memory)
5. WB – Scriere rezultat(Write-Back)

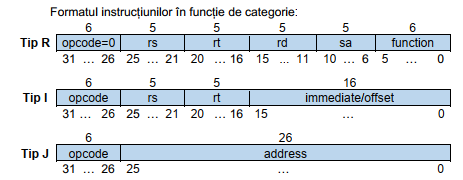
Fiecare etapă a fost descrisă printr-o componentă separată, cu excepția ultimei(cea de scriere a rezultatului), care au fost declarate și instanțiate în arhitectura entității **test\_env.**

1.3 Programul de test

Pentru început inițializăm regiștrii implicați în programul de test, registrul 1 va fi contorul buclei, în registrul 4 vom avea numărul de iterații, iar la fiecare iterație vom compara valorile celor 2 regiștri. Când sunt egale vom face saltul la instrucțiunea de store word de la linia 14 prin instrucțiunea beq $1, $4, 7. Registrul 2 va fi folosit pe post de index pentru a face load word, iar numărul de la indexul curent din memorie va fi scris în registrul 3. Dacă nu suntem la ultima iterație vom face un AND între numărul curent și valoarea 1 pentru a testa dacă ultimul bit este activ, iar dacă această condiție este îndeplinită îl vom adăuga în sumă deoarece este impar. În continuare vom face saltul la instrucțiunea 5, adică vom trece la următoarea iterație.

2. Descriere instrucțiuni

2.1 Tipuri de instrucțiuni, formatul instrucțiunilor

Există 3 tipuri de instrucțiuni de tip R, tip I și tip J. La cele de tip R are loc o operație între 2 regiștri, iar rezultatul este stocat într-un registru, la cele de tip I apare o valoare imediată reprezentată pe 16 biți, care este extinsă la 32 de biți, iar instrucțiunile de tip J sunt cele care fac saltul la o anumită adresă în memoria de instrucțiuni.

Pentru instrucțiunile de tip R, opcode-ul va fi 0, iar operația va fi dată de câmpul function. Pentru celelalte 2 catergorii opcode-ul codifică unic instrucțiunile. În total, în acest caz, putem codifica 2^6 instrucțiuni de tip R și 2^6 – 1 instrucțiuni tip I și J.

2.2 Instrucțiuni suplimentare

Instrucținunile suplimenare alese

* De tip R : 1. Logical Xor

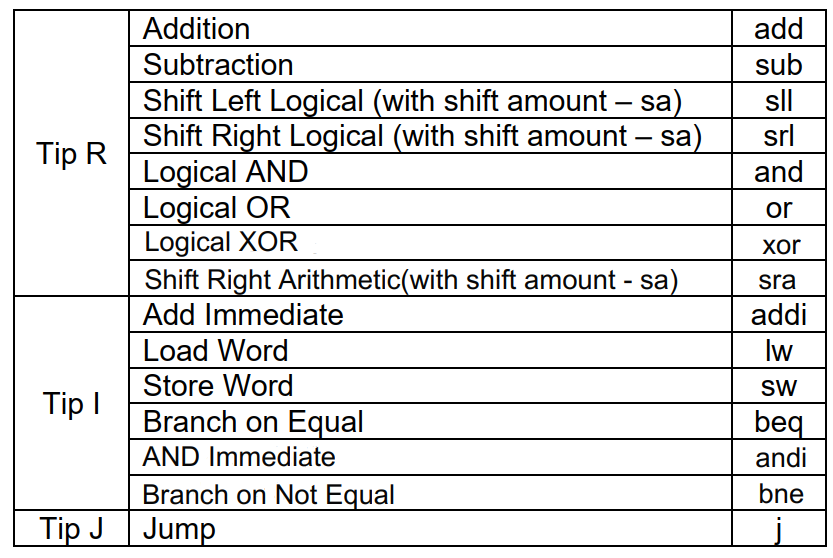
2. Shift Right Arithmetic (with shift amount - sa)

* De tip I : 1. And Immediate

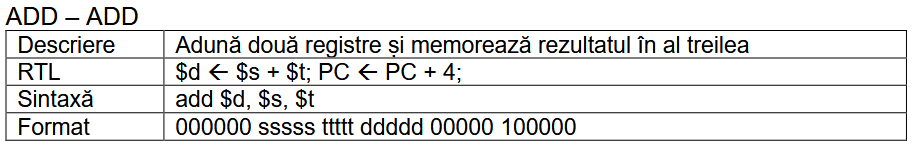
2. Branch on Not Equal

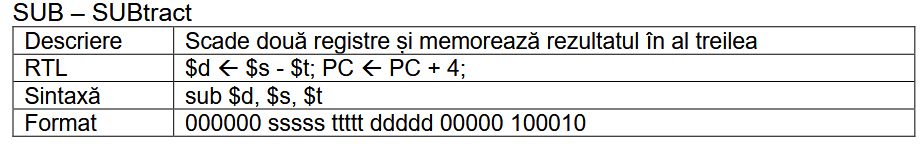
Dintre aceste instrucțiuni am folosit în programul de test doar AND Immediate pentru a testa daca ultimul bit este activ, practic pentru a vedea paritatea numărului. Pentru implementarea instrucțiunii XOR, respectiv And Immediate doar am făcut “RD1 XOR ALUIn2” în unitatea de execuție, respectiv “RD1 AND ALUIn2”, unde ALUIn2 este RD2 sau imediatul extins. Pentru implementarea deplasărilor pe biți am urmat sfaturile din îndrumătorul de laborator și am folosit funcțiile predefinite sll, srl, și sra. Astfel deplasarea aritmetică la dreapta s-a făcut în felul următor : “to\_stdlogicvector(to\_bitvector(RD1) sra conv\_integer(sa))”, analog pentru deplasare logică la dreapta și stânga. Pentru aceste instrucțiuni nu a fost nevoie de extinderea procesorului, de adăugare de componente noi. Pentru instrucțiunea Branch on Not Equal, în schimb, a fost nevoie să adaug o poartă AND pentru a verifica dacă zero flag-ul este inactiv și avem instrucțiune BNE, adică semnalul BranchNe este activ. Pentru a verifica dacă dorim să ajungem la adresa de branch, răspunsul porții AND descrise anterior va intra într-o poartă OR care verifică dacă avem instrucțiune BEQ. Astfel avem instrucțiunea “PCSrc <= (Zero and Branch) or ((not Zero) and BranchNe)”, iar când PCSrc are valorea 1 următoarea instrucțiune va fi cea de la adresa de Branch.

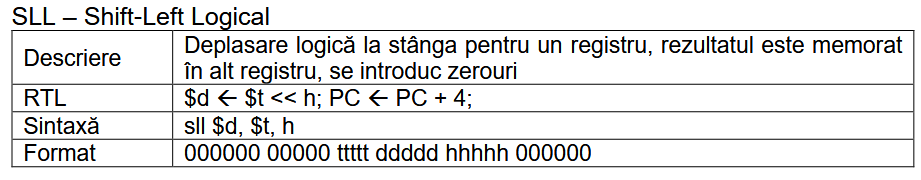
Astfel setul de instrucțiuni, din fiecare tip, care se vor implementa pe procesorul MIPS 32 în cazul nostru este descris în următorul tabel :

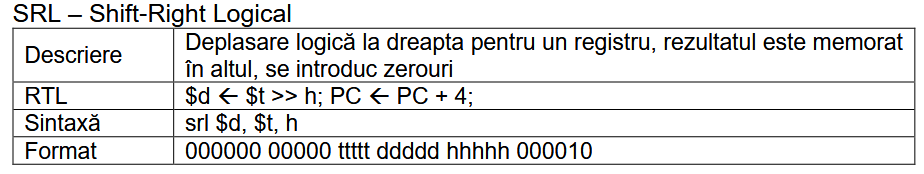


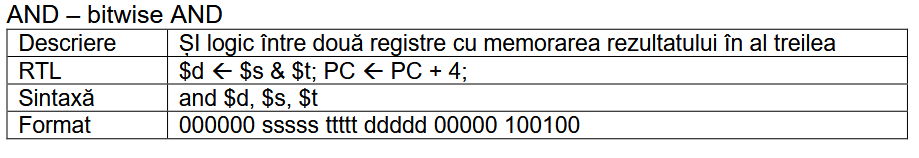
2.3 Scrierea în asamblare, RTL, formatul binar

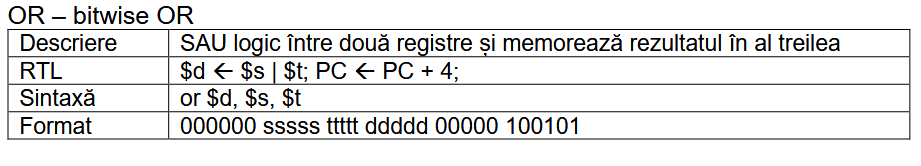


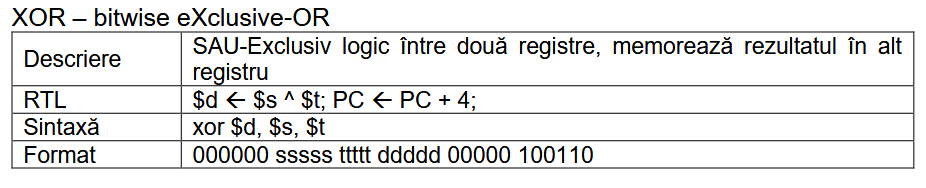


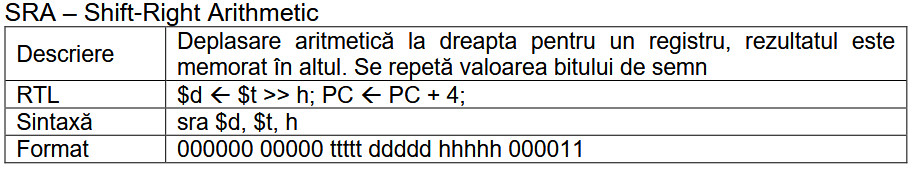


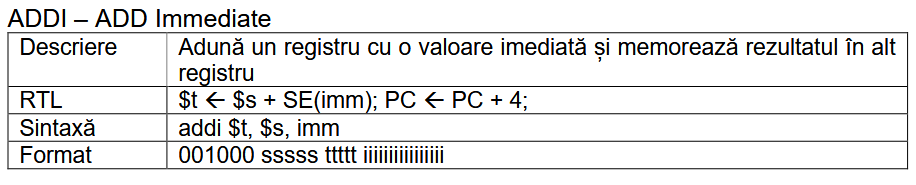


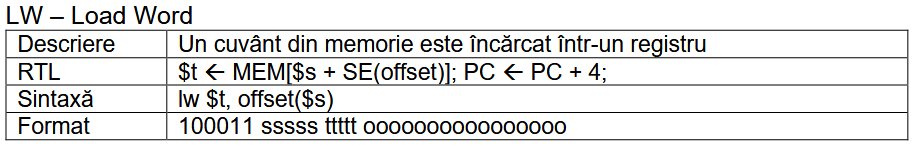


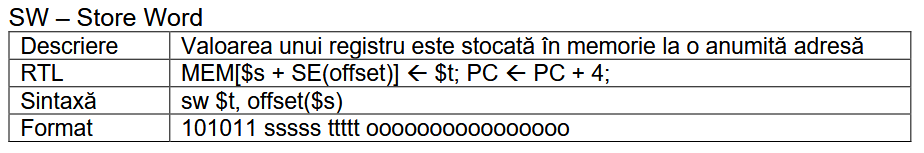


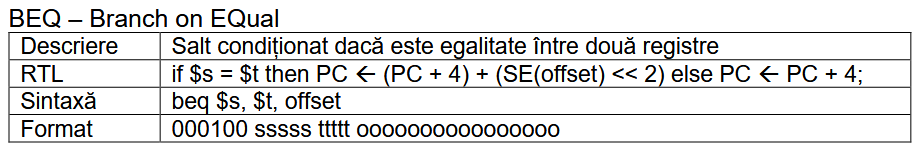


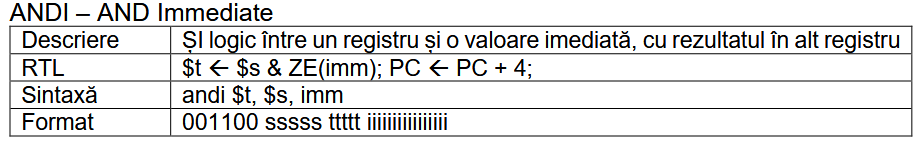


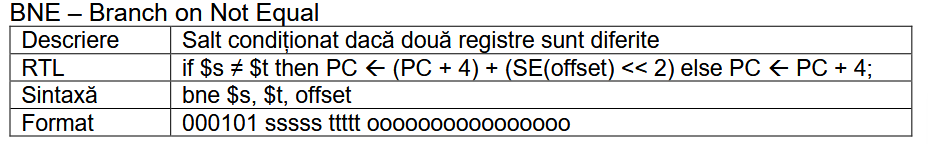












|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TIP | Instructiune | Opcode | ALUOp | ALUCtrl | RegDst | ExtOp | ALUSrc | Branch | BranchNe | Jump | MemWrite | MemtoReg | RegWrite |
| R Type | ADD | 000000 | 000 | 000(+) | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| SUB | 000000 | 000 | 001(-) | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| SLL | 000000 | 000 | 010(<<) | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| SRL | 000000 | 000 | 011(>>) | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| AND | 000000 | 000 | 100(&) | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| OR | 000000 | 000 | 101(or) | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| XOR | 000000 | 000 | 110(xor) | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| SRA | 000000 | 000 | 111(>>>) | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| I Type | ADDI | 001000 | 001 | 000(+) | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| LW | 100011 | 001 | 000(+) | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| SW | 101011 | 001 | 000(+) | x | 1 | 1 | 0 | 0 | 0 | 1 | x | 0 |
| BEQ | 000100 | 010 | 001(-) | x | 1 | 0 | 1 | 0 | 0 | 0 | x | 0 |
| ANDI | 001100 | 101 | 100(&) | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| BNE | 000101 | 010 | 001(-) | x | 1 | 0 | 0 | 1 | 0 | 0 | x | 0 |
| J Type | JUMP | 000010 | 000 | xxx | x | x | x | x | x | 1 | 0 | x | 0 |

* 1. Semnale de control

1. Elemente funcționale și probleme

Testarea procesorului a avut loc exclusiv pe placă. Probleme întâmpinate inițial au apărut la instrucțiunile lw $3, 0($2) și cele de beq : beq $1, $4, 7, beq $6, $0, 1. La cea de load word ajungea în registrul 3 doar valoarea de pe ramura others din unitatea MEM, din cauza unor greșeli de cod din aceeași unitate, iar la instrucțiunea Branch on Equal, calculam greșit adresa de branch. Problemele au fost remediate, iar în final totul a funcționat pe placă, iar rezultatul parțial, suma numerelor impare până la elementul curent din intervalul [10, 20] se poate vedea la la fiecare iterație la linia „add $5, $5, $3”, în registrul 5 fiind suma care se actulizează, iar în registrul 3 fiecare număr din memorie, iar la sfârșit la ultima linie ”sw $5, 56($0)”, vom avea în registrul 5 valoare X“0000 004b”, 75 în baza 10 .