Documentație MIPS32, ciclu unic

Fazacaș Ioana

# Program de test pentru procesorul MIPS

Cerință: Să se determine valoarea pară maximă dintr-un șir de N numere stocate în memorie începând cu adresa A (A≥12). A și N se citesc de la adresele 0, respectiv 4. Rezultatul se va scrie în memorie la adresa 8.

**Cod in C**

Int N;

int A[N] = {12, 6, 7, 20, 25, 10, 18, 55, 40,…};

int max = -999;

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

{ if(A[i]%2==0)

If(max<A[i])

max=A[i];

}

# Componente functionale :

MIPS-ul a fost testat pe plăcuță și toate elementele sale sunt funcționale, iar programul afișează rezultatul corespunzător.

1. **Unitatea IFetch:**

Primeste pe intrările de date adresele de salt și pune pe ieșiri, adresa imediat următoare (PC+4) și conținutul instrucțiunii curente.

In memoria ROM component a Unității IFetch avem stocate instructiunile programului in ordinea lor secventiala de executie:

0 add $1,$0,$0 # folosim r1 drept contor pentru bucla (i=0 contorul buclei)

1 lw $3, 4($0) # in r3 stocam valoare N(care se afla la adresa 4 in memorie) aceasta reprezinta

lungimea sirului, numarul de iteratii ale buclei

2 add $2,$0,$0 # in r2 stocam valoarea 0 (am vrut sa folosesc r2 pentru incrementarea adresei

de unde citesc din sir dar , pana la urma nu am mai folosit de loc registrul

respective , aceasta instructiune poate fi scoasa din program , dar am observant

prea tarziu acest lucru)

3 addi $5,$0, -999 # r5 va fi folosit pentru stocarea valorii pozitive pare maxime din sir (max=-999)

4 lw $4, 0($0) # in r4 stocam adresa de inceput a sirului A, r4 va fi folosit pentru a stoca adresa

la care se află elementele curent din sir in timp ce parcurgem sirul prin

intermediul buclei for(int i=0 ; i<N; i++)

5 beq $1,$3 13 # verificam daca am terminat de parcurs intregul sir(i==N) , iar daca este

adevarat sarim la inafara buclei la finalul programului la instructiunea de pe

pozitia 19 pentru a salva reultatul in memorie

6 lw $6,0($4) # in r6 aducem elementul current din sir (r6=A[i])

7 addi $7,$0,1 # stocam valoarea 1 in r7

8 and $8,$7,$6 # facem AND pe biti intre elementul curent din sir A[i] si valoarea 1 , r8=0 daca

elementul A[i] este par , altfel r8=1 daca A[i] este impar

9 addi $9,$1,1 # r9=i+1 in r9 stocam valoarea contorului incrementata cu 1

10 add $1,$9,$0 # r1=r9 actualizam valoare contorului cu valoarea lui incrementata i=i+1

11 addi $10,$4,1 # r10=r4+1 in r10 stocam adresa urmatorului element din sir

12 add $4,$10,$0 # r4=r10 actulizam r4 cu noua adresa a elementului curent care va fi folosita in

urmatoarea iteratie a sirului

13 beq $8,$0,1 #daca elementul curent A[i] este impar mergem mai departe la instructiunea de

jump care ne trimite la inceputul buclei, daca nu sarim peste instructiunea de

jump si efectual instructiunea de slt

14 j 5 # sarim la inceputul buclei

15 slt $7,$5,$6 # r7=1 daca (r5<r6) max<A[i] altfel r7=0

16 beq $7,$0,1 # daca max<A[i] mergem la instructiunea urmatoare unde actualizam valoarea

maximului max=A[i], altfel facem salt la instructunea de jump care ne trimite la

inceputul buclei

17 add $5,$6,$0 #actualizam valoarea maximului (r5=r6) max=A[i]

18 j 5 # sarim la inceputul buclei

19 sw $5,8($0) # stocam in memorie la adresa 8 valoarea para maxima din sir (max)

1. **Unitatea de decodificare a instrucțiunilor ID**

Unitatea ID primește pe intrările de date intrucțiunea curentă și valoarea WD, care se scrie în RF, ambele pe 32 de biți. ID pune la dispoziție pe ieșiri, operanzii RD1, RD2 și imediatul extins Ext\_Imm, tot pe 32 de biți. Pe ieșire mai apar câmpurile function (6 biți) și sa (5 biți) din instrucțiune. Semnalul de control RegDst selectează registrul în care se scrie valoarea WD atunci când semnalul de control RegWrite este activ. Scrierea este sincronă pe frontul ascendant. RF este port mapat in interiorul unitatii ID si meoria ROM componenta este initializata cu 0 la inceputul implementarii.

1. **Unitatea de control UC**

M-am folosit de tabelul pentru Semnale de control MIPS32 pentru setarea semnalelor de control in functie de instructia curenta care se efectueaza.

1. **Unitatea de execuție EX**

Unitatea EX primește pe intrările de date registrele RD1 și RD2 de la blocul de registre, imediatul extins Ext\_imm și adresa de instrucțiune imediat următoare PC+4, codificate pe 32 de biți. Suplimentar, apar câmpurile func și sa din instrucțiunea curentă, pe 6 biți, respectiv 5 biți. EX pune la dispoziție rezultatul ALU cu semnalul de validare Zero (care indică un rezultat nul)și un semnal de validare Gtz(care indica daca rezultatul este mai mare decat zero) și adresa de salt condiționat Branch Address, calculată astfel: Branch Address <= (PC+4) + (Ext\_imm << 2).

Primul operand în ALU este RD1, iar cel de-al doilea este ales între RD1 și Ext\_imm cu ajutorul semnalului de control ALUSrc. Codul operației ALUOp generat de unitatea de control UC este convertit la codul ALUCtrl, care determină operația de efectuat în ALU.

1. **Unitatea de memorie MEM**

Conține memoria RAM care stocheaza datele pe 32 de biti. Scrierea în memorie este sincronă pe frontul de ceas ascendent și citirea este asincronă. Am adaugat un semnal rez care sa primeasca rezultatul programului care va fi memorat la adresa 8. Rezultatul va fi afisat in test\_env pe SSD .

Adresa de scriere sau citire a datelor este data de bitii ALUResin(5 downto 0).

1. **Unitatea de scriere a rezultatului WB**

Constă din multiplexorul cu selecția MemtoReg care va decide daca se foloseste rezultatul primit de la ALU sau datele citite din memorie.

# Componente nefunctionale :

Toate componentele MIPS-ului sunt funcționale

# Probleme intâmpinate pe parcurs :

Am intampinat problem la instructiunilor add , and ,addi atunci cand foloseam același registru drept registrul destimnatie cât și ca registru sursa (Pentru incrementare lui r1. ex: addi $1, $1, 1). Când implementam instrucțiunile în felul acesta începea să se adune la infinit numerele din registru. Am rezolvat problema prin folosirea unui alt registru drept registru destinatie și împărțirea instrucțiunii în doua instructiuni. (Pentru a realiza incrementarea registrului cu valoarea dorita. ex: addi $1, $1, 1 => addi $9, $1, 1 ; add $1, $9, $0 ).

Am mai întâmpinat probleme la scrierea si citirea datelor din memorie . Din cauza modului în care am gândit algoritmul și instrucțiunile, adresa de la care se citea sau scria in Unitatea MEM trebuie să primească ALUResin(5 downto 0), nu ALUResin(7 downto 2). Acesta a fost singurul mod prin care am reușit să rezolv problema și MIPS-ul să funcționeze correct.

# Instrucțiuni suplimentare alese :

BGTZ – Branch on Greater Than Zero

|  |  |
| --- | --- |
| Descriere | Salt condiționat dacă un registru este mai mare ca 0. |
| RTL | If $s > 0 then PC <= (PC + 4) + (SE(offset) << 2) else PC <= PC + 4; |
| Sintaxă | bgtz $s, offset |
| Format | 000111 sssss 00000 oooooooooooooooo |
| Semnale de control | ExtOp<='1'; BraGTZ<='1'; ALUOp<="01"; |

SLTI – Set on Less Than Immediate (signed)

|  |  |
| --- | --- |
| Descriere | Dacă $s este mai mic decât un imediat, $t este inițializat cu 1, altfel cu 0. |
| RTL | PC <= PC + 4; if $s < SE(imm) then $t <= 1 else $t <= 0; |
| Sintaxă | slti $t, $s, imm |
| Format | 001010 sssss ttttt iiiiiiiiiiiiiiii |
| Semnale de control | ExtOp<='1'; ALUSrc<='1'; RegWrite<='1'; ALUOp<="11"; |

SRA – Shift-Right Arithmetic

|  |  |
| --- | --- |
| Descriere | Deplasare aritmetică la dreapta pentru un registru, rezultatul este memorat în altul. Se repetă valoarea bitului de semn |
| RTL | $d <= $t >> h; PC <= PC + 4; |
| Sintaxă | sra $d, $t, h |
| Format | 000000 00000 ttttt ddddd hhhhh 000111 |
| Semnale de control | RegDst<='1'; RegWrite<='1'; ALUOp<="10" |

SLT- Set on Less Than (signed)

|  |  |
| --- | --- |
| Descriere | Dacă $s < $t, $d este inițializat cu 1, altfel cu 0 |
| RTL | PC ⇓ PC + 4; if $s < $t then $d ⇓ 1 else $d ⇓ 0; |
| Sintaxă | slt $d, $s, $t |
| Format | 000000 sssss ttttt ddddd 00000 001010 |
| Semnale de control | RegDst<='1'; RegWrite<='1'; ALUOp<="10" |