**Arhitectura Calculatoarelor**

**MIPS**

Apostu Iulian-Eduard

30222

Numarul de aparitii al unui numar intr-un vector

Cod C:

int n=5, count=0, nr = 4;

int a[]= {1, 2, 4, 4, 5};

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

{

if(a[i]==nr)

{

count++;

}

}

Asamblare: Cod C:

addi $1, $0, 0 int i = 0;

addi $2, $0, 2 //initializam indexul cu care parcurgem memoria

addi $3, $0, 1 int count = 0 // $3 va primi 1 initial, fiind = cu count+1

lw $4, 0($0) int n = 5;

lw $5, 0($3) int nr = 4; //pentru a nu folosi un alt registru, folosim $3

beq $1, $4, 6 for(...;i<n;...)

lw $6, 0($2) a[i] //pargurgem, de la mem(2) se gasesc elementele

bne $6, $5, 1 if(a[i]==nr)

addi $3, $3, 1 count++; //$3 va fi count+1

addi $1, $1, 1 for(...;...;i++)

addi $2, $2, 1 //incrementam indexul cu care pargurgem memoria

j 5 //sarim la inceputul loop-ului

addi $3, $3, 127 //$3 – 1 pentru a ajunge la rezultatul corect

Semnalele de control:

O imagine care conține masă

Descriere generată automat

Cod Masina:

001\_000\_001\_0000000 X"2080" addi $1, $0, 0

001\_000\_010\_0000010 X"2102" addi $2, $0, 2

001\_000\_011\_0000001 X"2181" addi $3, $0, 1

010\_000\_100\_0000000 X"4200" lw $4, 0($0)

010\_011\_101\_0000000 X"4E80" lw $5, 0($3)

011\_100\_001\_0000110 X"9086" beq $1, $4, 6

010\_010\_110\_0000000 X"4B00" lw $6, 0($2)

101\_101\_110\_0000001 X"B701" bne $6, $5, 1

001\_011\_011\_0000001 X"2D81" addi $3, $3, 1

001\_001\_001\_0000001 X"2481" addi $1, $1, 1

001\_010\_010\_0000001 X"2901" addi $2, $2, 1

100\_0000000000101 X"E005" j 5

001\_011\_011\_1111111 X"2DFF" addi $3, $3, 127

MIPS 16 (+ extindere Branch Not Equal):

O imagine care conține diagramă

Descriere generată automat

Pentru a extinde mips-ul cu BNE, avem nevoie de un nou semnal de control, o poarta AND, ce are ca intrare semnalul de control BNE si semnalul de Zero negat de la ALU (se face diferenta si daca semnalul Zero este 0 cele 2 numere sunt diferite si se efectueaza saltul), si o poarta OR, intre Branch si iesirea portii AND de la BNE, a carei iesire este selectia PCSrc.

Rezultatele simularii:

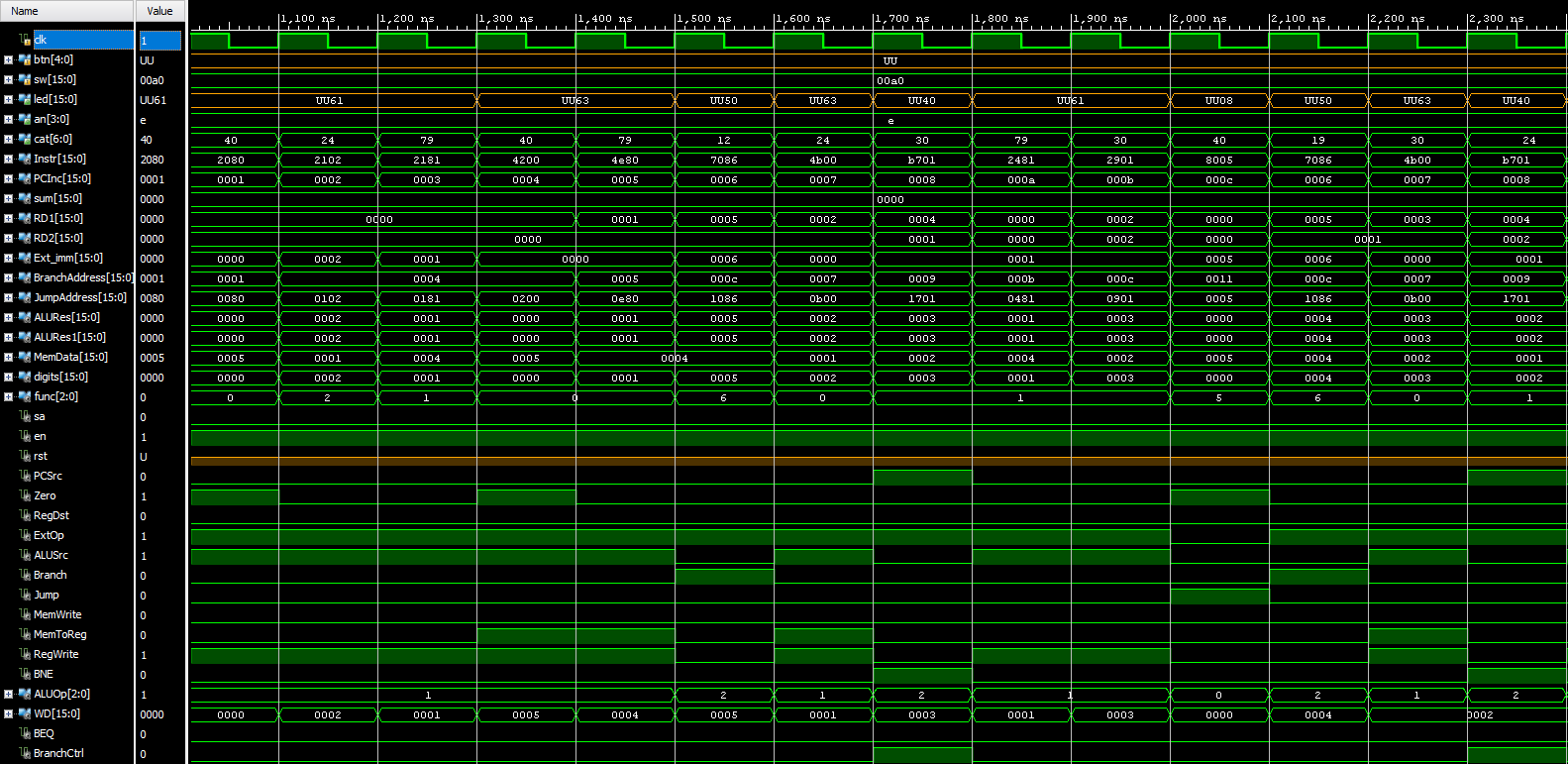
Pentru a se putea realiza simularea, pentru semnalul de clock (clk) se va folosi „Force Clock”, pentru Switch-uri (sw[15:0]) „Force Constant” cu valoarea binara „0000000010100000”, astfel incat sa se afiseza pe SSD semnalul ALURes, si pentru semnalul de Enable (en) „Force Constant” cu 1.

Continutul Memoriei:



Primele iteratii a programului:

Instructiunea X”7086” este instructiunea de Branch Equal si X”8005” este cea de Jump. Se poate observa ca functioneaza corect urmarind coloanele 11 si 12(2000 ns si 2100 ns) pe linia „Instr”.



Ultimele iteratii ale prugramului:

Pe coloana corespunzatoare timpului 3700 ns se poate observa instructiunea X”2D81” care se executa doar cand se gaseste un numar egal cu cel pe care il cautam, in cazul nostru ‚4’, altfel va sarii peste o singura instructiune: cea care incrementeaza counter-ul duplicatelor lui 4 gasite. (pe coloana anterioara, unde se executa instructunea BNE X”B701”, RD1 si RD2 au aceeasi valoare: ‚4’, de aceea instructiunea X”2D81” este executata)

Ultima instructiune efectuata este cea de pe coloana cu timpul 4800 ns: X”2DFF”, moment in care pe digits (SSD) va aparea rezultatul final, 2 in cazul nostru deoarece 4 apare de 2 ori.

Aceasta instructiune decrementeaza counter-ul deoarece acesta este initializat cu ‚1’ pentru a folosi acest registru si ca index pentru a lua cel de-al doilea element din memorie (Instructiunea X”2181”).

