Universitatea POLITEHNICA din Bucureşti

Facultatea de Automatică şi Calculatoare

Catedra de Calculatoare

http://www.csit-sun.pub.ro

CALCULATOARE NUMERICE

Proiect de semestru – anul III

Prof. Îndrumător:

S.L. Dr. Ing. Decebal Popescu

Student :GRECU EmilianGeorge

Bucureşti 2003

TITLUL PROIECTULUI SMARTPOCKET

TEMA PROIECTULUI

Implementarea unui calculator de buzunar ce utilizeaza functii matematice elementare

PREZENTARE GENERALA

Proiectul este implementat in totalitate in mediul de dezvoltare Verilog oferit de Xilinx. Testarea si functionalitatea acestuia sunt prezentate cu ajutorului simulatorului ModelSim. Proiectul isi propune implementarea operatiilor de adunare, scadere, inmultire, impartire in virgula mobila sub forma unui calculator de buzunar, utilizand conceptul benzii de asamblare. Acesta inglobeaza cate un modul responsabil cu operatiile precedent specificate care sunt comandate de un modul UAL.

CUPRINS

Introducere **4**

Notiuni generale **4**

Descrierea proiectului **7**

Prezentarea modulelor **7**

Sectiune de testare **9**

Schema UAL **15**

Bibliografie **16**

INTRODUCERE

Notiuni generale

Banda de asamblare

Banda de asamblare este o tehnica de descompunere a proceselor secventiale in suboperatii; fiecare suboperatie putand fi executata intr-un segment special dedicat si in paralel cu alte suboperatii.

Rezultatul obtinut in cadrul fiecarui segment este transferat urmatorului segment din banda de asamblare. La rezultatul final se ajunge atunci cand datele au trecut prin toate segmentele bezii de asamblare.

Suprapunerea calculelor este posibila datorita asocierii unui registru fiecarui segment al benzii de asamblare. Registrele au rolul de a izola segmentele astfel incat fiecare segment al benzii de asamblare poate opera date distincte simultan cu operarea altor segmente.

Reprezentarea numerelor in virgula mobila

Reprezentarea numerelor in virgula mobila face obiectul standardului IEEE 754. Conform acestui standard, reprezentarea lor este alcatuita din doua parti. Prima parte reprezinta un numar cu semn denumit mantisa. Pareta a doua specifica pozitia punctului zecimal si se numeste exponent.

Reprezentarea numerelor in virgula mobila poate fi facuta in precizie simpla sau duba. Folosirea preciziei simple impune o reprezentare a numarului utilizand 32 de biti.

Primul bit (bitul cel mai semnificativ) este bitul de semn. Daca valoarea acestui bit este 0, numarul este pozitiv, altfel numarul este negativ.

Urmatorii 8 biti sunt alocati pentru valoarea exponentului, iar ultimii 23 de biti reprezinta mantisa.

In cazul in care se foloseste precizia dubla, numarul de biti utilizati pentru reprezentarea numarului este 64.

In cadrul proiectului, reprezentarea numerelor in virgula mobila se realizeaza cu precizie simpla, pe 32 biti.

Reprezentarea generala a uni numar in virgula mobila folosind precizia simpla este:

$(-1)^s * (1 + MANTISA) * 2^{EXPONENT - 127}$

Exemplificand reprezentarea in virgula mobila cu precizie, rezultatul este urmatorul:

```
0                0 00000000 00000000000000000000000
1                0 01111111 00000000000000000000000
2                0 10000000 00000000000000000000000
4                0 10000001 00000000000000000000000
8                0 10000010 00000000000000000000000
16               0 10000011 00000000000000000000000
32               0 10000100 00000000000000000000000
64               0 10000101 00000000000000000000000
128              0 10000110 00000000000000000000000
256              0 10000111 00000000000000000000000
512              0 10001000 00000000000000000000000
1024             0 10001001 00000000000000000000000
2048             0 10001010 00000000000000000000000
4096             0 10001011 00000000000000000000000
8192             0 10001100 00000000000000000000000
5.75             0 10000001 01110000000000000000000
-.1              1 01111011 10011001100110011001101
```

Valoarea "5.75", spre exemplu, este stocata ca 01000000101110000000000000000000".

Se poate observa ca primul bit este cel corespunzator semnului, iar uramtorii 8 biti sunt folositi pentru exponent, in timp ce restul de 23 corespund mantisei.

Totodata, exista un bit ascuns care desi nu este stocat, este intotdeauna luat in seama. Acest bit determina un total de 24 de biti pentru mantisa, el fiind intotdeauna 1. Deci, la adaudarea celui de-al 24-lea bit, rezultatul este urmatorul :

```
0         0 00000000 100000000000000000000000
1         0 01111111 100000000000000000000000
2         0 10000000 100000000000000000000000
4         0 10000001 100000000000000000000000
8         0 10000010 100000000000000000000000
16        0 10000011 100000000000000000000000
32        0 10000100 100000000000000000000000
64        0 10000101 100000000000000000000000
128       0 10000110 100000000000000000000000
256       0 10000111 100000000000000000000000
512       0 10001000 100000000000000000000000
1024      0 10001001 100000000000000000000000
2048      0 10001010 100000000000000000000000
4096      0 10001011 100000000000000000000000
8192      0 10001100 100000000000000000000000
5.75      0 10000001 101110000000000000000000
-.1       1 01111011 110011001100110011001101
```

Dupa acest bit ascuns se afla un punct zecimal implicit. Pentru a determina valoarea bitil;or situati dupa acest punct zecimal, se imparte totalul la 2: astfel, primul bit dupa punctul zecimal este 5, uramatorul este 25 si asa mai departe. Sa urmarim acest algoritm pe exemplul analizat mai sus : Exponentul corespunzator valorii 5.75 eset 129. Daca scadem 2, rezulta 127. Deci $1.0111 * 2^2$ devine 101.11. Deci acum avem: 101 in binar care este 5 plus 0.5 plus 0.25 (0.11) sau 5.75 in total.

Sa consideram cazul valorii -.1. Metoda este aproximativ similara. Exponentul este 64 + 32 + 16 + 8 + 2 + 1 or 123. Scazand 127 se obtine -4, ceea ce inseamna ca punctul zecimal se va deplasa 4 unitati la stanga, generand astfel .0001100110011001100110011001101. Daca calculam in binar, obtinem .625 + .3125 + .0390625 si pentru valori din ce in ce mai mici, rezultatul tinde catre 1. Bitul de semn a fost stabilit -.1

DESCRIEREA PROIECTULUI

Prezentarea modulelor

Modulul UAL

Unitatea Aritmetico - Logica reprezinta structura unui coprocesor matematic ce executa operatii de adunare, scadere, inmultire, impartire in virgula mobila folosind module interne pentru fiecare operatie in parte.

Intrarile LDA, respectiv LDB colecteaza operanzii A si B din intrarea UAL si le introduc in registrii interni. La activarea semnalului LDO, se colecteaza operandul de pe magistrala de operatii. Erorile sunt semnalate de modulele interne de inmultire si impartire si sunt scoase la iesire pe semnalul 'error '.

vezi Schema Unitatii Aritmetico - Logice (pag. 15)

Modulele de adunare si scadere in virgula mobila

Algoritmii necesita parcurgerea urmatoarelor etape.

1. se verifica daca unul dintre operanzi este 0 sau nu,

2. se aliniaza mantisele,

3. se aduna sau se scad mantisele in functie de caz,

4. se normalizeaza rezultatul obtinut.

Pentru inceput, exponentii sunt comparati prin scadere. Exponentul mai mare este ales ca exponent al rezultatului. Rezultatul diferentei indica de cate ori mantisa asociata cu exponentul mai mic trebuie deplasata catre dreapta. Astfel se realizeaza alinierea mantiselor. Rezultatul comparatiei se stocheaza intr-o variabila. Daca bitul cel mai semnificiativ al acestei variabile este 0, mantisa trebuie deplasata spre dreapta.

Ulterior, cele doua mantise sunt adunate sau scazute in functie de valoarea operandului. Rezultatul adunarii su scaderii mantiselor se reprezinta pe 26 de biti. Bitul cel mai semnificativ, bitul 25 este utilizat pentru specificarea daca mantisa este egala sau nu cu 0.

Rezultatul astfel obtinut se normalizeaza.

Modulul de inmultire in virgula mobila

Inmultirea in virgula mobila presupune realizarea urmatoarelor operatii:

1. se verifica daca unul dintre numere este 0 sau nu,
2. se aduna exponentii,
3. se inmultesc mantisele,
4. se normalizeaza produsul.

Aceste operatii se executa in aceeasi maniera ca cele mentionate anterior.

Trebuie remarcat ca expoenntul rezultat corect este obtinut prin scaderea valorii 127 din exponentul suma.

Modulul de impartire in virgula mobila

Algoritmul de impartire a doua numere reprezentate in virgula mobila necesita parcurgerea urmatorilor pasi :

1. se verifica daca unul dintre cei doi operanzi este 0 sau nu,
2. se initalizeaza registrele si se evalueaza semnul,
3. se aliniaza deimpartitul,
4. se scad exponentii,
5. se impart mantisele.

Impartirea a doua numere in virgula mobila normalizate va produce intotdeuna un rezultat normalizat. De aceea, spre deosebire de celelalte operatii in virgula mobila, rezultatul acestei operatii nu necesita normalizarea.

Modulul Add_Digit

Modulul Add_Digit este cel care sesizeaza apasarea unei cifre, convertind valoarea acesteia din cod zecimal in cod binar. Acesta realizeaza legatura intre comenzile primite de la tastatura si Unitatea Aritmetico- Logica.

Concret, modulul Add_Digit realizeaza conversia din Binary Coded Decimal in Floating Point. Operatiunea se realizeaza in timp real, in sensul ca la fiecare apasare a unei taste cuprinse intre 0 si 9, se obtine codul in binar al acesteia.

Sectiune de testare

Testarea modului de adunare

Date de intrare

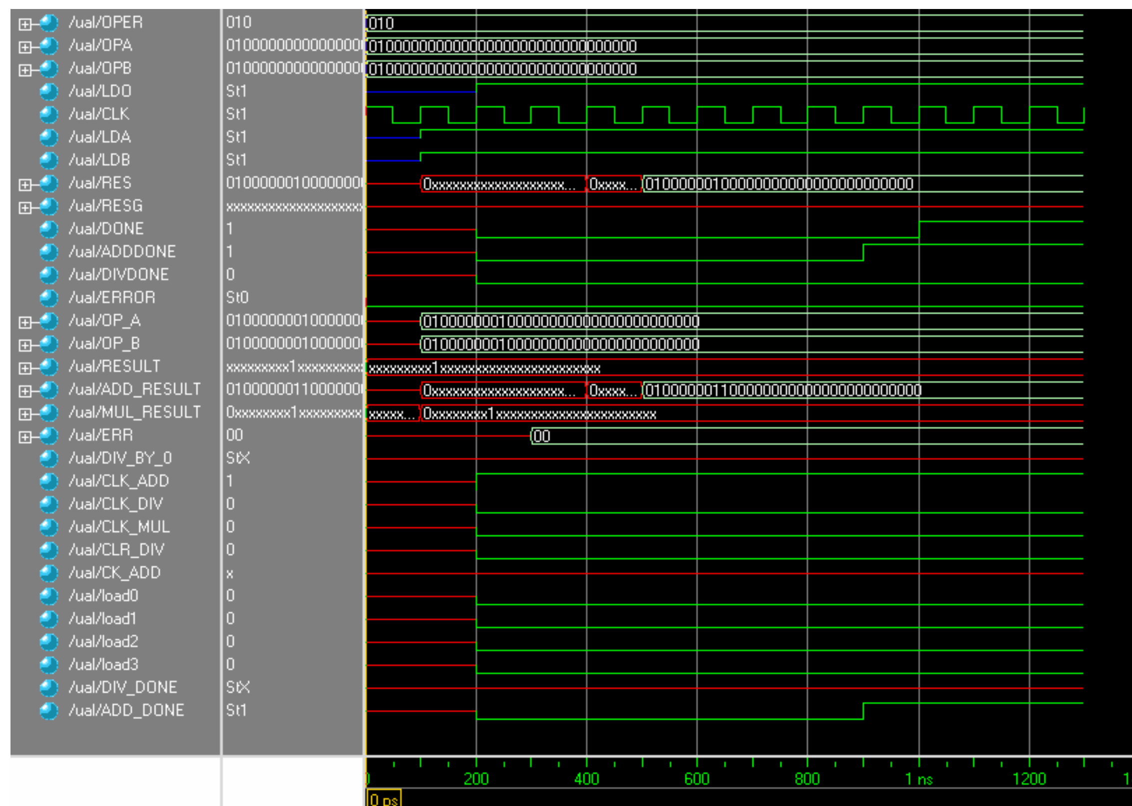
Am simulat adunarea : 2 + 2 = 4.

OPA= 01000000000000000000000000000000 0

OPB= 01000000000000000000000000000000 0

```
force -drive sim:/ual/OPER 010 0
force -drive sim:/ual/OPA 01000000000000000000000000000000 0
force -drive sim:/ual/OPB 01000000000000000000000000000000 0
force -freeze sim:/ual/LDO 1 200, 0 {4200 ps} -r 8000
force -freeze sim:/ual/CLK 1 0, 0 {50 ps} -r 100
force -freeze sim:/ual/LDA 1 100, 0 {4100 ps} -r 8000
force -freeze sim:/ual/LDB 1 100, 0 {4100 ps} -r 8000
```

Diagrama obtinuta



Interpretarea rezultatului

RES = 0 10000001 00000000000000000000000 care este reprezentarea lui 4 in binar.

Testarea modulului de scadere

Date de intrare

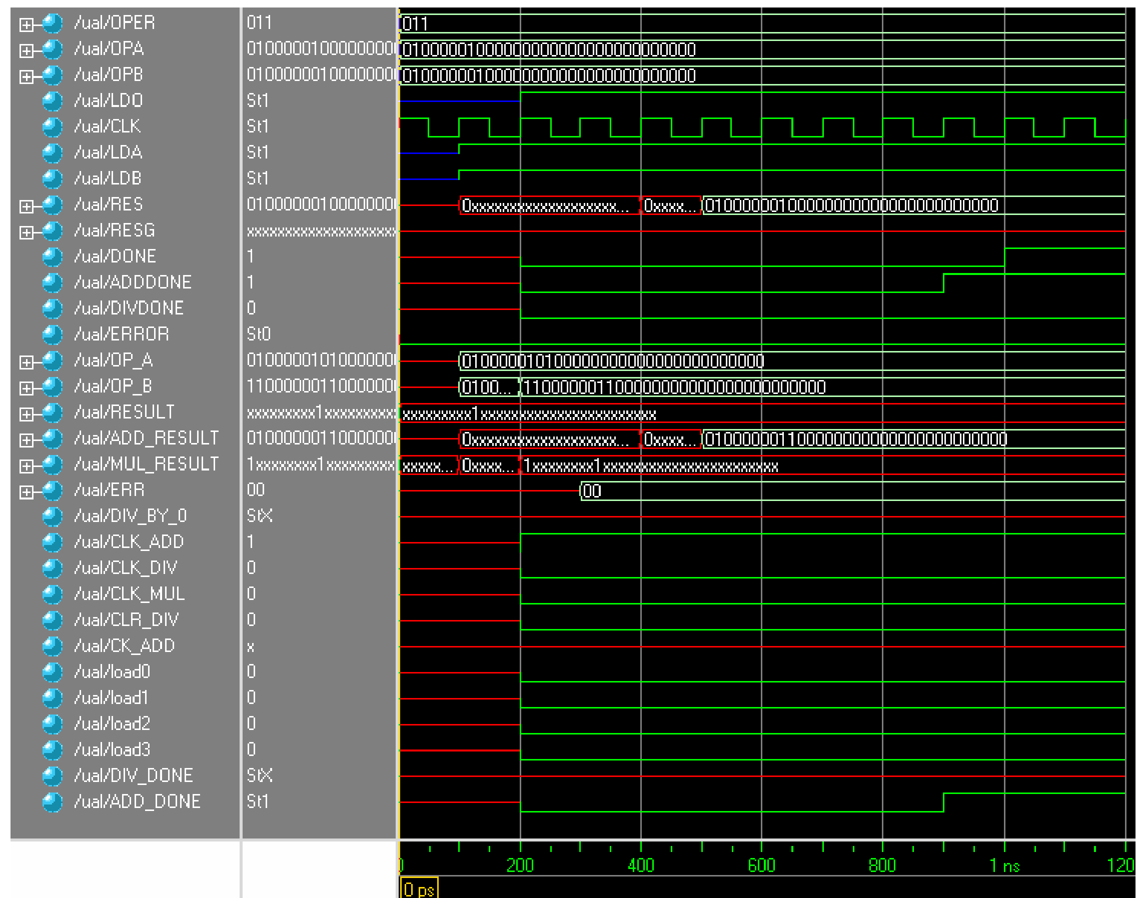
Am simulat scaderea: 8 - 4 = 4.

OPA= 0 10000010 00000000000000000000000

OPB= 0 10000001 00000000000000000000000

force -drive sim:/ual/OPER 011 0
force -drive sim:/ual/OPA 0 10000010 00000000000000000000000 0
force -drive sim:/ual/OPB 0 10000001 00000000000000000000000 0
force -freeze sim:/ual/LDO 1 200, 0 {4200 ps} -r 8000
force -freeze sim:/ual/CLK 1 0, 0 {50 ps} -r 100
force -freeze sim:/ual/LDA 1 100, 0 {4100 ps} -r 8000
force -freeze sim:/ual/LDB 1 100, 0 {4100 ps} -r 8000

Diagrama obtinuta



Interpretarea rezultatului

RES = 0 10000001 00000000000000000000000 care este reprezentarea lui 4 in binar.

Testarea modulului de inmultire

Date de intrare

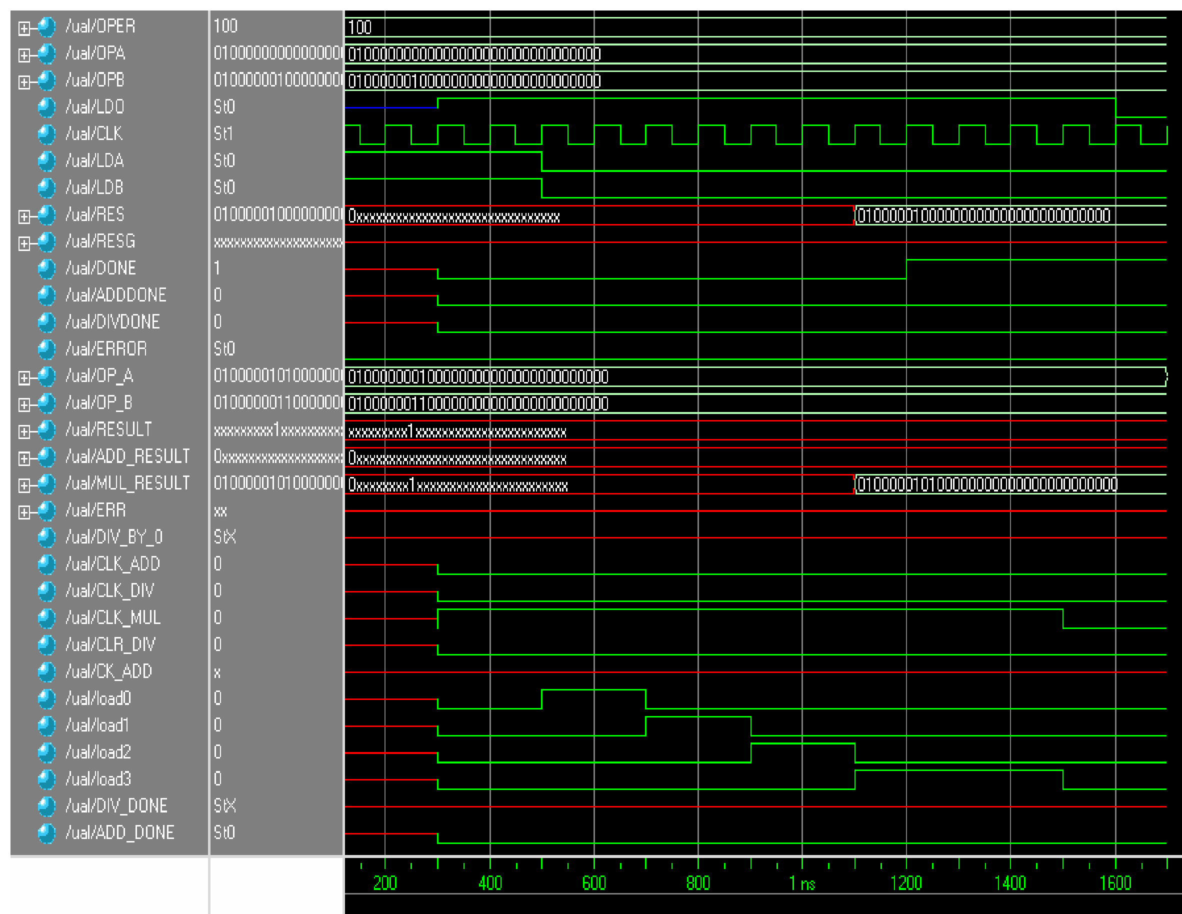
Am simulat inmultirea: 2 * 4 = 8.

OPA= 0 10000000 0000000000000000000000 0

OPB= 0 10000001 0000000000000000000000 0

```
force -drive sim:/ual/OPER 100 0
force -drive sim:/ual/OPA 0 10000000 0000000000000000000000 0
force -drive sim:/ual/OPB 0 10000001 0000000000000000000000 0
force -freeze sim:/ual/LDO 1 300, 0 { 1600 ps} -r 7000
force -freeze sim:/ual/CLK 1 0, 0 {50 ps} -r 100
force -freeze sim:/ual/LDA 1 100, 0 { 500 ps} -r 7000
force -freeze sim:/ual/LDB 1 100, 0 { 500 ps} -r 7000
```

Diagrama obtinuta



Interpretarea rezultatului

RES = 0 10000010 00000000000000000000000 care este reprezentarea lui 8 in binar.

Testarea modulului de impartire

Date de intrare

Am simulat impartirea: 8 * 4 = 2.

OPA= 0 10000010 00000000000000000000000

OPB= 0 10000001 00000000000000000000000

force -drive sim:/ual/OPER 101 0
force -drive sim:/ual/OPA 01000001000000000000000000000000 0
force -drive sim:/ual/OPB 01000000100000000000000000000000 0
force -freeze sim:/ual/LDO 1 300, 0 { 1600 ps} -r 7000
force -freeze sim:/ual/CLK 1 0, 0 {50 ps} -r 100
force -freeze sim:/ual/LDA 1 100, 0 { 500 ps} -r 7000
force -freeze sim:/ual/LDB 1 100, 0 { 500 ps} -r 7000

Diagrama obtinuta

Interpretarea rezultatului

RES = 01000000000000000000000000000000 care este reprezentarea lui 2 in binar.

Testarea modulului Add_Digit

Date de intrare

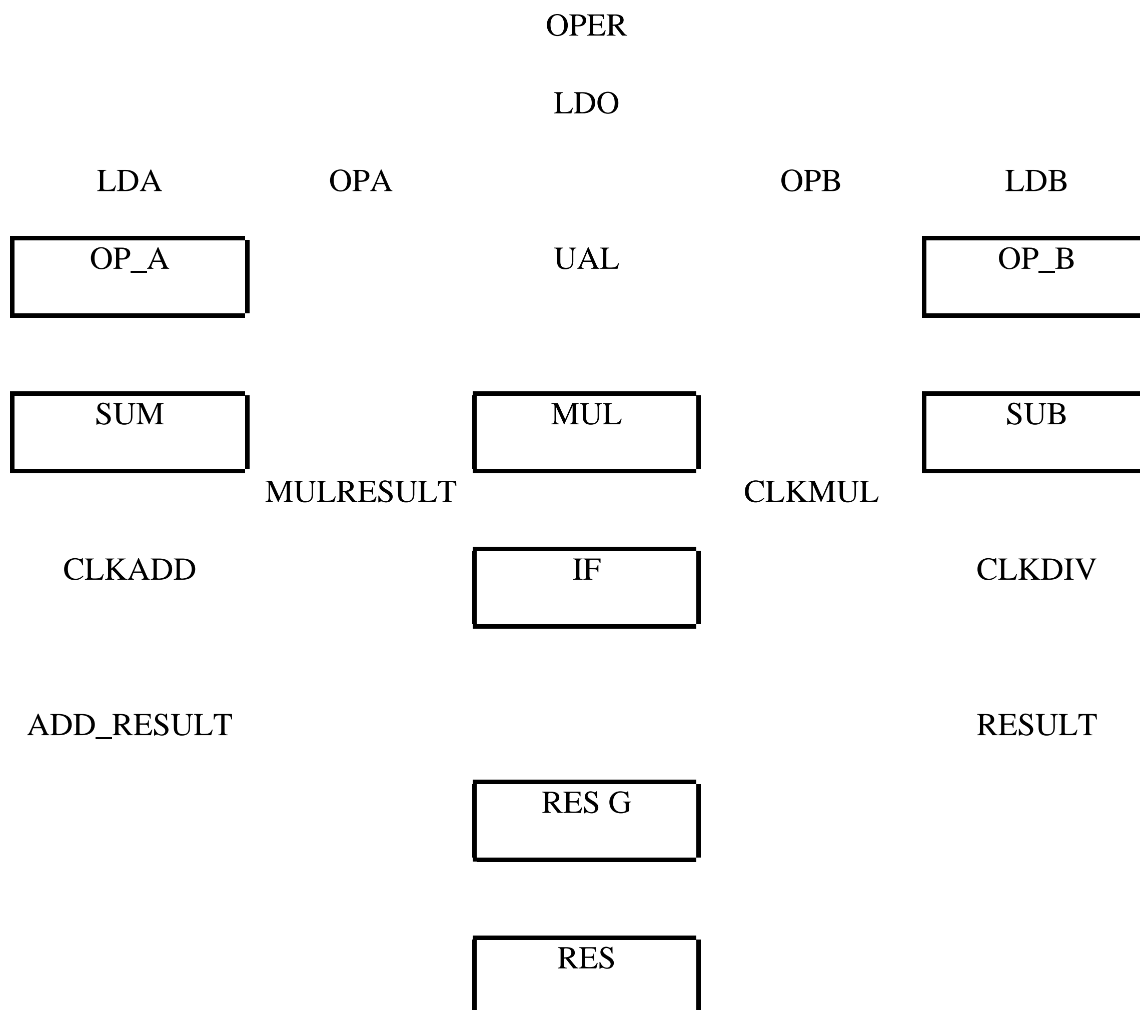
Am simulat apasarea tastei 1, respectiv a tastei 5.

force -freeze sim:/add_digit/CLK 1 0, 0 {50 ps} -r 100
force -freeze sim:/add_digit/CE 1 200, 0 {14600 ps} -r 16500
force -freeze {sim:/add_digit/DIG[1]} 0 500, 1 {1300 ps} -r 16000
force -freeze {sim:/add_digit/DIG[5]} 0 2500, 1 {3300 ps} -r 16000
force -freeze sim:/add_digit/LDO 1 4000, 0 {4800 ps} -r 16000

Diagrama obtinuta

Interpretarea rezultatului

Am obtinut reprezentarea numarului 15 in binar, care constituie primul operand in cadrul operatiei ce urmeaza a fi executate ulterior activarii semnalului DONE_ADD.



Schema Unitatii Aritmetico - Logice

BIBLIOGRAFIE

- ***Calculatoare numerice*** - indrumar de laborator

  Prof. Dr. Ing. Adrian Petrescu
  As. Dr. Ing. Decebal Popescu
  As. Dr. Ing. Nirvana Popescu
  Conf. Dr. Ing. Cornel Popescu

- ***http://www.csit-sun .pub.ro***

- ***http://www.aplawrence.com/Basics/floatingpoint.html***