

Proiectarea unei unități aritmetice de tip MMX

Structura sistemelor de calcul

IRIMEȘ CRISTINA, 30234

Cuprins

[1. INTRODUCERE 1](#_Toc118320258)

[1.1 CONTEXT 1](#_Toc118320259)

[1.2 SPECIFICAȚII 2](#_Toc118320260)

[1.3 OBIECTIVE 2](#_Toc118320261)

[2. STUDIU BIBLIOGRAFIC 2](#_Toc118320262)

[2.1 PADDB 2](#_Toc118320263)

[2.2 PADDD 3](#_Toc118320264)

[2.3 PSUBB 3](#_Toc118320265)

[2.4 PSUBD 3](#_Toc118320266)

[2.5 PMULLW 4](#_Toc118320267)

[2.6 PMADDWD 4](#_Toc118320268)

[3. DESIGN 6](#_Toc118320269)

[4. IMPLEMENTARE 6](#_Toc118320270)

[5. TESTARE 6](#_Toc118320271)

[6. CONCLUZIE 6](#_Toc118320272)

[7. BIBLIOGRAFIE 6](#_Toc118320273)

# INTRODUCERE

## cONTEXT

Acest proiect are scopul de a proiecta, implementa și de a testa 6 operații, la alegere, care să ilustreze funcționalitățile unei unități aritmetice de tip MMX x86. Cele 6 operații pe care le-am ales sunt următoarele: adunarea pe bytes, adunarea pe dublu-cuvânt, scăderea pe bytes, scăderea pe dublu cuvânt, înmulțirea cu păstrarea părții inferioare a rezultatului și înmulțirea cu adunare a rezultatelor (paddb, paddd, psubb, psubd, pmullw, pmaddwd).

Aceste instrucțiuni se deosebesc prin faptul că aceleași operații se execută simultan pe mai multe elemente (SIMD), celule, ale regiștrilor MM0-MM7, regiștrii pe 64 de biți. În acest fel fiecare registru poate sa fie accesat fie pe byte (atunci vom avea 8 elemente de un byte), fie pe cuvânt (4 elemente), dublu cuvânt (2 elemente), quadruplu-cuvânt (1 element) . Aceste elemente se consideră a fi numere întregi cu semn sau fără semn.

## SPECIFICAȚII

Pentru partea de implementare si simulare a acestui proiect se va folosi IDE-ul din Vivado , urmând ca apoi sa fie testat pe o placuță Basys3. Acesta va putea să arate conținutul tuturor regiștrilor atât înainte, cât și după efectuarea operațiilor pe ei, și să selecteze instrucțiunea dorită și operanzii săi.

## OBIECTIVE

Obiectivele acestui proiect sunt de a proiecta și implementa o unitate aritmetică care să execute, în funcție de instrucțiunea aleasă, operațiile multiple în același timp, folosindu-se de valorile stocate în regiștrii, apoi salvând rezultatul operației aritmetice în primul operand al acesteia. Astfel, cu ajutorul afișoarelor pe șapte segmente să poată fi vizualizată schimbarea valorilor în interiorul registrului destinație.

# STUDIU BIBLIOGRAFIC

* <https://www.csie.ntu.edu.tw/~cyy/courses/assembly/docs/MMX.pdf>

## PADDB

Instrucțiunea PADDB adună numere întregi de octeți împachetate de la primul operand sursă și al doilea operand sursă și stochează rezultatele întregului împachetat în operand de destinație (primul registru). Când un rezultat individual este prea mare pentru a fi reprezentat în 8 biți (depășire), rezultatul este înfășurat și cei 8 biți inferiori sunt scriși în operandul destinație (adică transportul este ignorat).

OPERAND1[7:0] ← OPERAND1[7:0] + OPERAND2[7:0];

OPERAND1[15:8] ← OPERAND1[15:8] + OPERAND2[15:8];

OPERAND1[23:16] ← OPERAND1[23:16] +OPERAND2[23:16];

OPERAND1[31:24] ← OPERAND1[31:24] + OPERAND2[31:24];

OPERAND1[39:32] ← OPERAND1[39:32] + OPERAND2[39:32];

OPERAND1[47:40] ← OPERAND1[47:40] + OPERAND2[47:40];

OPERAND1[55:48] ← OPERAND1[55:48] + OPERAND2[55:48];

OPERAND1[63:56] ← OPERAND1[63:56] + OPERAND2[63:56];

## PADDD

Instrucțiunea PADDD adună numere întregi împachetate de la primul operand sursă și al doilea operand sursă și stochează rezultatele întregului împachetat în operandul de destinație (primul registru). Când un rezultat individual este prea mare pentru a fi reprezentat în 32 de biți (depășire), rezultatul este înfășurat și cei 32 de biți mai mici sunt scriși în operandul destinație (adică transportul este ignorat).

OPERAND1[31:0] ← OPERAND1[15:0] + OPERAND2[15:0];

OPERAND1[63:32] ← OPERAND1[31:16] + OPERAND2[31:16];

## PSUBB

Instrucțiunea PSUBB scade numerele întregi de octeți împachetate. Când un rezultat individual este prea mare sau prea mic pentru a fi reprezentat într-un octet, rezultatul este înfășurat și cei 8 biți mai mici sunt scriși în elementul destinație.

OPERAND1[7:0] ← OPERAND1[7:0] - OPERAND2[7:0];

OPERAND1[15:8] ← OPERAND1[15:8] - OPERAND2[15:8];

OPERAND1[23:16] ← OPERAND1[23:16] - OPERAND2[23:16];

OPERAND1[31:24] ← OPERAND1[31:24] - OPERAND2[31:24];

OPERAND1[39:32] ← OPERAND1[39:32] - OPERAND2[39:32];

OPERAND1[47:40] ← OPERAND1[47:40] - OPERAND2[47:40];

OPERAND1[55:48] ← OPERAND1[55:48] - OPERAND2[55:48];

OPERAND1[63:56] ← OPERAND1[63:56] - OPERAND2[63:56];

## PSUBD

Instrucțiunea PSUBD scade numerele întregi cu cuvinte duble împachetate. Când un rezultat individual este prea mare sau prea mic pentru a fi reprezentat într-un cuvânt dublu, rezultatul este înfășurat și cei 32 de biți mici sunt scriși în elementul destinație.

OPERAND1[31:0] ← OPERAND1[15:0] - OPERAND2[15:0];

OPERAND1[63:32] ← OPERAND1[31:16] - OPERAND2[31:16];

## PMULLW

Efectuează o înmulțire semnată SIMD a numerelor întregi de cuvânt cu semn împachetate în operandul destinație (primul operand) și operandul sursă (al doilea operand) și stochează cei 16 biți inferiori ai fiecărui rezultat intermediar pe 32 de biți în operandul destinație. (Figura de mai jos prezintă mai concret funcționalitatea)

Graphical user interface, application, table

Description automatically generated

TEMP0[31:0] ← DEST[15:0] ∗ SRC[15:0]; (\* înmulțire cu semn \*)

TEMP1[31:0] ← DEST[31:16] ∗ SRC[31:16];

TEMP2[31:0] ← DEST[47:32] ∗ SRC[47:32];

TEMP3[31:0] ← DEST[63:48] ∗ SRC[63:48];

DEST[15:0] ← TEMP0[15:0];

DEST[31:16] ← TEMP1[15:0];

DEST[47:32] ← TEMP2[15:0];

DEST[63:48] ← TEMP3[15:0];

## PMADDWD

Înmulțește cuvintele cu semn individuale ale operandului de destinație (primul operand) cu cuvintele semnate corespunzătoare ale operandului sursă (al doilea operand), producând rezultate temporare semnate, cu cuvinte duble. Rezultatele adiacente cu două cuvinte sunt apoi însumate și stocate în operandul destinație. De exemplu, cuvintele de ordin inferior corespunzătoare (15-0) și (31-16) din operanzii sursă și destinație sunt înmulțite unul cu celălalt, iar rezultatele cuvintelor duble sunt adunate și stocate în cuvântul dublu inferior al registrului de destinație (31-0). Aceeași operație se efectuează și asupra celorlalte perechi de cuvinte alăturate. (Figura de mai jos prezintă mai concret funcționalitatea).

Graphical user interface, table

Description automatically generated with medium confidence

DEST[31:0] ← (DEST[15:0] ∗ SRC[15:0]) + (DEST[31:16] ∗ SRC[31:16]);

DEST[63:32] ← (DEST[47:32] ∗ SRC[47:32]) + (DEST[63:48] ∗ SRC[63:48]);

# DESIGN

Pentru a realiza acest proiect, va fi nevoie să imităm memoria reprezentată de cei șapte regiștrii MMX (a câte 64 de biți fiecare), care sunt suprapuși deasupra virgulei mobile din memoria calculatorului și a unei unități aritmetice logice, care să se ocupe de efectuarea operațiilor reprezentate de folosirea acestor instrucțiuni. De asemenea, mai sunt necesare câteva componente pentru afișarea sau folosirea corectă și fiabilă a elementelor plăcii, precum un debouncer pentru butoane și un seven segment display.

Debouncer-ul va fi defapt un MPG (mono-pulse generator) care va crea semnalele de la butoane către celelalte elemente. Este nevoie de o astfel de componentă pentru a verifica dacă butonul a fost apăsat o anumită perioadă de timp, dar și pentru că frecvența plăcii este foarte mare să se execute o singură dată instrucțiunea la o apăsare a butonului.

SSD-ul (seven segment-display) completează acest proiect prin rolul pe care îl joacă. Prin intermediul lui vor fi afișate rezultatele instrucțiunilor pe cele patru afișoare ale plăcii Basys 3. El va primi un număr și îl va afișa folosindu-se de anozii și catozii celor patru afișoare.

MPG



SSD



MEMORIA- este de tip RAM, cu citire asincronă și scriere sincronă(condiționată de ENABLE). Pe lângă semnalele obișnuite, va mai avea câteva în plus, care vor facilita utilizarea proiectului, precum **rst**-pentru resetarea valorilor regiștrilor, **dec**- decrementarea la valoarea curentă a unui registru, **inc**-incrementarea la valoarea curentă a unui registru, **registrul** -utilizat pentru pornirea ledului corespunzător registrului.



ALU - Unitatea aritmetica logică va executa operațiile aritmetice în funcție de semnalul de codificare instr: „00”-adunare, „01”-scădere, „10”-înmulțire, ”11”- înmulțire cu adunare, iar în cazul adunării, mai este nevoie de un semnal adițional pentru adresare, pentru cum vor fi percepute blocurile regiștrilor: „0”-pe byte, „1”-pe dublu-cuvânt.



# IMPLEMENTARE

La baza acestui proiect a stat ideea folosirii IDE-ului Vivado pentru implementare și a unei plăcuțe Basys 3 pentru testare și interceptare a rezultatelor. Așadar, am considerat plăcuța un blackbox cu următoarele semnale de intrare:

-clk -> clock-ul plăcuței;

-btn -> cele 5 butoane;

-sw –> cele 16 sitch-uri;

-led -> cele 16 led-uri;

-an și cat –> anozi și catozii celor 4 afișoare pe 7 segmente.

Prin intermediul semnalului sw aleg instrucțiunea pe care doresc să o efectuez:

* sw(15 downto 14) – una din cele 4 tipuri de operații generale
* sw(13) – una dintre cele 2 tipuri de prezentări
* sw(12 downto 10) – primul registru/operand folosit pentru efectuarea operațiilor
* sw(6 downto 4) – al doilea registru/operand folosit pentru efectuarea operațiilor
* sw(3 downto 0)-care dintre 16 biți din totalul de 64 doresc să îi afișez pe afișoarele plăcuței

Cu ajutorul semnalului de led prezint a cărui registru îi vizualizez valoarea:

* led = ”0000000000000100” -valoarea afișată este a registrului 3

Cu ajutorul semnalului de btn comand o manipulare controlată a componentei care conține cei 8 regiștrii MMX:

* btn(0) – comandă salvarea, în registru precizat, valorii primate
* btn(1)- comandă numărătorul interior a regiștrilor (pentru a putea vizualiza valoarea din oricare dintre aceștia la orice moment dorit)
* btn(2) - comandă resetarea tuturor regiștrilor la valoarea 0
* btn(3) - comandă adăugarea valorii 1 la registrul precizat de primul operand, la poziția a primilor 16 biți dați de al doilea operand
* btn(4) - comandă scăderea valorii 1 la registrul precizat de primul operand, la poziția a primilor 16 biți dați de al doilea operand

În final, programul principal se restrânge la un fișier principal (aceasta cutie neagră), care se

folosește de o arhitectură mixtă pentru lega celelalte componente precizate la punctul anterior (ALU, Memoria, MPG, SSD) și

de un proces pentru a oferi o afișare cât mai completă a valorilor, procesând astfel semnalele intermediare dintre acestea și prezentând o variantă completă a rezolvării probleme propuse.

# TESTARE

Pentru a facilita o vizualizare cât mai apropiată de utlizarea plăcuții Basys3 pentru testare, am implementat o sursă de TestBench, care trebuie setată ca sursă principală în meniul de Surse a Simulării și setat un clock de 10 ns pentru aceasta. Astfel, la fiecare pas al rulării se poate observa executarea pe rând a celor 6 instrucțiuni, în ordinea prezentării în document și cum se modifică conținutul regiștrilor MMX. Următoarele exemple au fost folosite pentru a prezenta validarea rezultatelor:

Pmaddb : x"0004000300020001" + x"1110111011101110" => reg0 = x"1114111311121111"

Pmaddd : x"FFFFFFFFFFFFFFFF" + x"0000000200000003" => reg5 = x"000000100000002"

Pmsubb : x"1114111311121111" - x"1110111011101110" => reg0 = x"0004000300020001"

Pmsubd : x"000000100000002" - x"FFFFFFFFFFFFFFFF" => reg5 = x"1114111311121111"

Pmmullw : x"1112111211121112" \* x"1110111211141116" => reg1 = x"432065448768A98C"

Pmaddwd : x"1112111211121112" \* x"1110111211141116" => reg3 = x"0246A864024730F4"

În figura de mai jos este reprezentat ultimul clock din simularea TestBenchului.

Se observă că în fiecare dintre regiștrii folosiți pentru operații se află valoarea corectă după aplicarea tuturor operațiilor pe rând.

­­­­­­­­­­­­­­­­­­­­­­­­­

# CONCLUZIE

Scopul acestui proiect a fost proiectarea, implementarea și testarea a 6 instrucțiuni de tipul MMX.

Am încercat și realizat un mod cât mai plăcut și deloc rigid de a folosi și experimenta funcționalitatea acestor instrucțiuni; astfel cu ajutorul unei plăcuțe Basys3 realizând o interacțiune plăcută si complete asupra operațiilor.

În concluzie, am folosit o idee complet originală de a aduce cerința la o variantă hardware funcțională și care nu necesită nici o nouă intervenția asupra părții software pentru a putea fi folosită la capacitate maximă.

# BIBLIOGRAFIE

* [MMX Instructions - x86 Assembly Language Reference Manual (oracle.com)](https://docs.oracle.com/cd/E18752_01/html/817-5477/eojdc.html)
* [MMX (instruction set) - Wikipedia](https://en.wikipedia.org/wiki/MMX_(instruction_set))
* [PADDB/PADDW/PADDD/PADDQ — Add Packed Integers (felixcloutier.com)](https://www.felixcloutier.com/x86/paddb:paddw:paddd:paddq)
* [PSUBB/PSUBW/PSUBD — Subtract Packed Integers (felixcloutier.com)](https://www.felixcloutier.com/x86/psubb:psubw:psubd)
* [PMULLW — Multiply Packed Signed Integers and Store Low Result (felixcloutier.com)](https://www.felixcloutier.com/x86/pmullw)
* [PMADDWD — Multiply and Add Packed Integers (felixcloutier.com)](https://www.felixcloutier.com/x86/pmaddwd)
* [MMX - Chessprogramming wiki](https://www.chessprogramming.org/MMX)
* [x86 Assembly - Wikibooks, open books for an open world](https://en.wikibooks.org/wiki/X86_Assembly)