Universitatea Politehnica Timisoara

Facultatea de Automatica si Calculatoare

Programul de studii: Calculatoare si Tehnologia Informatiei

Calculator de buzunar

General Purpose processor

Fundamente de ingineria calculatoare

Echipa *CalcCrafters*

Project Manager: Spatacean Oana- Teodora

Hardware Design: Stanescu Ariana- Camelia, Tarod Anamaria- Alexandra

Software Design: Oxani Corina

Testare: Talapan Alexandra

Repository link pe Github: https://github.com/OanaSpatacean/Calculator_de_buzunar

Cuprins

Multiplexorul pentru selectia adreselor care intra in Data Memory

Zero extend

Sign Extend

Modulul factorial

5. Design software

Interfata registrilor Acumulator, X, Y

Interfata multiplexorului de selectie al intrarii in ACC, X, Y

Interfata registrului de flags

Interfata Stack Pointer-ului

Interfata memoriei de instructiuni

Interfata registrului de instructiuni

Interfata Program Counter-ului

Interfata multiplexorului de selectie al intrarii in Program Counter

Interfata memoriei de date

Interfata multiplexorului de selectie al intrarii in Data Memory

Interfata multiplexorului de selectie al locatiei in Data Memory

Interfata Zero Extend- Bit Converter

Interfata Sign Extend

Interfata unitatii de control

Interfata unitatii aritmetice si logice

Interfata modului factorial

Interfata registrului factorial

Interfata timer-ului

5 Testare

Tabel cu testele facute

6 Bibliografie

1. TEMA PROIECTULUI

Proiectul nostru are ca temă centrală dezvoltarea unui procesor de uz general care să permită efectuarea operațiilor întâlnite într-un calculator de buzunar obișnuit.

Un procesor de uz general care este o unitate centrală de prelucrare care nu este proiectată să servească un scop specific. Acesta poate fi utilizat într-o varietate de aplicații și poate fi adaptat pentru a satisface nevoile unei anumite utilizări.

2. SETUL DE INSTRUCTIUNI SI FORMATELE DE INSTRUCTIUNI

Avem 3 registrii ACC, X, Y

Codificare: X – 0 ; Y – 1

Instructiunile de branch:

Nume instructiune – Opcode <6>	Address <10>	Descriere
BRZ – 000000	<10>	Se actualizeaza PC cu adresa de branch specificata de cei mai nesemnificativi 10 biti doar daca flag-ul Zero(Z) este activ.
BRN – 000001	<10>	Se actualizeaza PC cu adresa de branch specificata de cei mai nesemnificativi 10 biti doar daca flag-ul Negative(N) este activ.
BRC – 000010	<10>	Se actualizeaza PC cu adresa de branch specificata de cei mai nesemnificativi 10 biti doar daca flag-ul Carry(C) este activ.
BRO – 000011	<10>	Se actualizeaza PC cu adresa de branch specificata de cei mai nesemnificativi 10 biti doar daca flag-ul Overflow(V) este activ.
BRA – 000100	<10>	Se actualizeaza PC cu adresa de branch specificata de cei mai nesemnificativi 10 biti.

JMP – 000101	<10>	Se apeleaza procedura aflata la adresa specificata de cei mai nesemnificativi 10 biti.
RET – 000110	<10>	Se revine din procedura executandu-se instructiunea de dupa apelul procedurii.

Instructiuni aritmetice si logice:

Nume instructiune – Opcode <6>	Address <1>	Immediate <9>	Descriere
ADD – 000111	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC+X sau ACC=ACC+Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X+Immediate sau Y=Y+Immediate, in functie de adresa registrului.
SUB – 001000	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC-X sau ACC=ACC-Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X-Immediate sau Y=Y-Immediate, in functie de adresa registrului.
LSR – 001001	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC>>X sau ACC=ACC>>Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X>>Immediate sau Y=Y>>Immediate, in functie de adresa registrului.
LSL – 001010	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC<<X sau ACC=ACC<<Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X<<Immediate sau Y=Y<<Immediate, in functie de adresa registrului.
RSR – 001011	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia de rotire la dreapta al ACC cu atatea pozitii cu cat este specificat in X sau Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia de rotire la dreapta cu valoarea Immediate a lui X sau Y, in functie de adresa registrului.
RSL – 001100	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia de rotire la stanga al ACC cu atatea pozitii cu cat este specificat in X sau Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia de rotire la stanga cu valoarea Immediate a lui X sau Y, in functie de adresa registrului.

MUL – 001101	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC*X sau ACC=ACC*Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X*Immediate sau Y=Y*Immediate, in functie de adresa registrului.
DIV – 001110	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC/X sau ACC=ACC/Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X/Immediate sau Y=Y/Immediate, in functie de adresa registrului.
MOD - 001111	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC%X sau ACC=ACC%Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X%Immediate sau Y=Y%Immediate, in functie de adresa registrului.
OR – 010000	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC|X sau ACC=ACC|Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X|Immediate sau Y=Y|Immediate, in functie de adresa registrului.
AND – 010001	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC&X sau ACC=ACC&Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X&Immediate sau Y=Y&Immediate, in functie de adresa registrului.
XOR – 010010	<1>	<9>	Daca Immediate este 0, atunci se realizeaza operatia ACC=ACC^X sau ACC=ACC^Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se realizeaza operatia X=X^Immediate sau Y=Y^Immediate, in functie de adresa registrului.
NOT – 010011	<1>	000000000	Se inverseaza bitii din X sau Y in functie de adresa registrului
CMP – 010100	<1>	<9>	Daca Immediate este 0, atunci se compara ACC cu X sau cu Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se compara X sau Y cu Immediate, in functie de adresa registrului. Compararea se realizeaza prin scaderea celor 2 valori comparate.
TST - 010101	<1>	<9>	Daca Immediate este 0, atunci se compara ACC cu X sau cu Y, in functie de adresa registrului. Daca Immediate este diferit de 0, atunci se compara X sau Y cu Immediate, in functie de adresa registrului. Compararea se realizeaza prin operatia SI pe biti intre cele 2 valori.
INC – 010110	<1>	000000000	Se incrementeaza valoarea lui X sau Y, in functie de adresa.
DEC – 010111	<1>	000000000	Se decrementeaza valoarea lui X sau Y, in functie de adresa.

FACT – 011000	<1>	<9>	Se realizeaza operatia factorial a valorii din Immediate si rezultatul este plasat in X sau Y, in functie de adresa specificata.

Instructiune care muta valoarea immediate intr-un registru:

Nume instructiune – Opcode <6>	Address <1>	Immediate <9>	Descriere
MOV – 011001	<1>	<9>	Muta valoarea data de Immediate in registrul X sau Y, in functie de valoarea registrului.

Instructiuni care lucreaza cu memoria:

Nume instructiune – Opcode <6>	Address <1>	Immediate <9>	Descriere
LD – 011010	<1>	<9>	Incarca in X sau Y in functie de valoarea specificata de adresa, valoarea din memorie specificata de Immediate
ST - 011011	<1>	<9>	Incarca in memorie la adresa specificata de Immediate valoarea din X sau Y in functie de valoarea specificata de adresa.

Instructiuni de copiere:

Nume instructiune – Opcode <6>	Unused <1>	Descriere
COPY_X – 011100	0000000000	Copiaza valoarea din ACC in X.
COPY_Y - 011101	0000000000	Copiaza valoarea din ACC in Y.

Instructiuni care folosesc stiva:

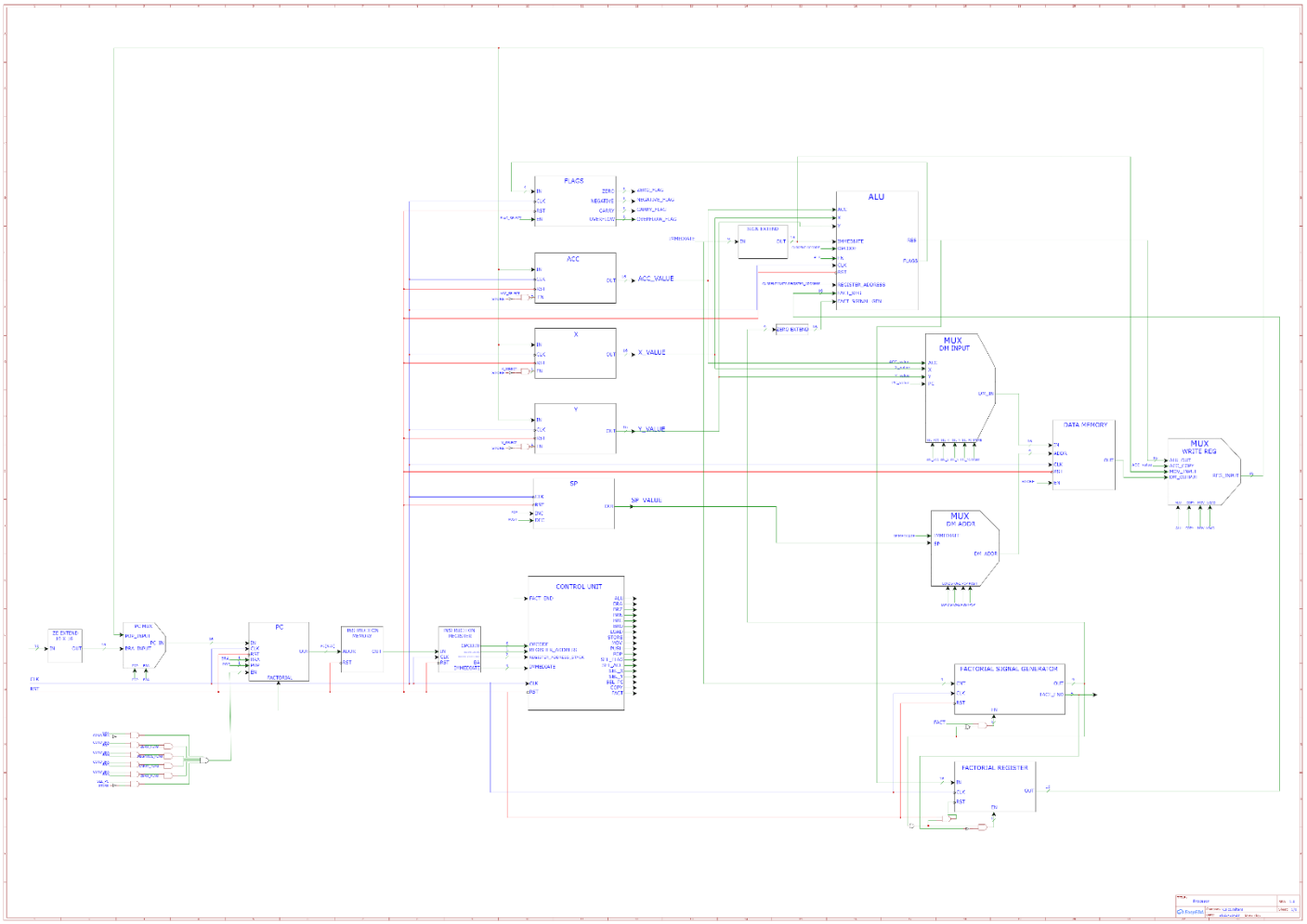
Nume instructiune – Opcode <6>	Address <1>	Immediate <8>	Descriere
PUSH – 011110	<2>	00000000	Incarca in memorie la adresa specificata de Stack Pointer valoarea din ACC, X sau Y, in functie de adresa.
POP - 011111	<2>	00000000	Incarca valoarea din memorie de la adresa specificata de Stack Pointer in ACC, X sau Y, in functie de adresa.

Instructiune care nu realizeaza nicio operatie:

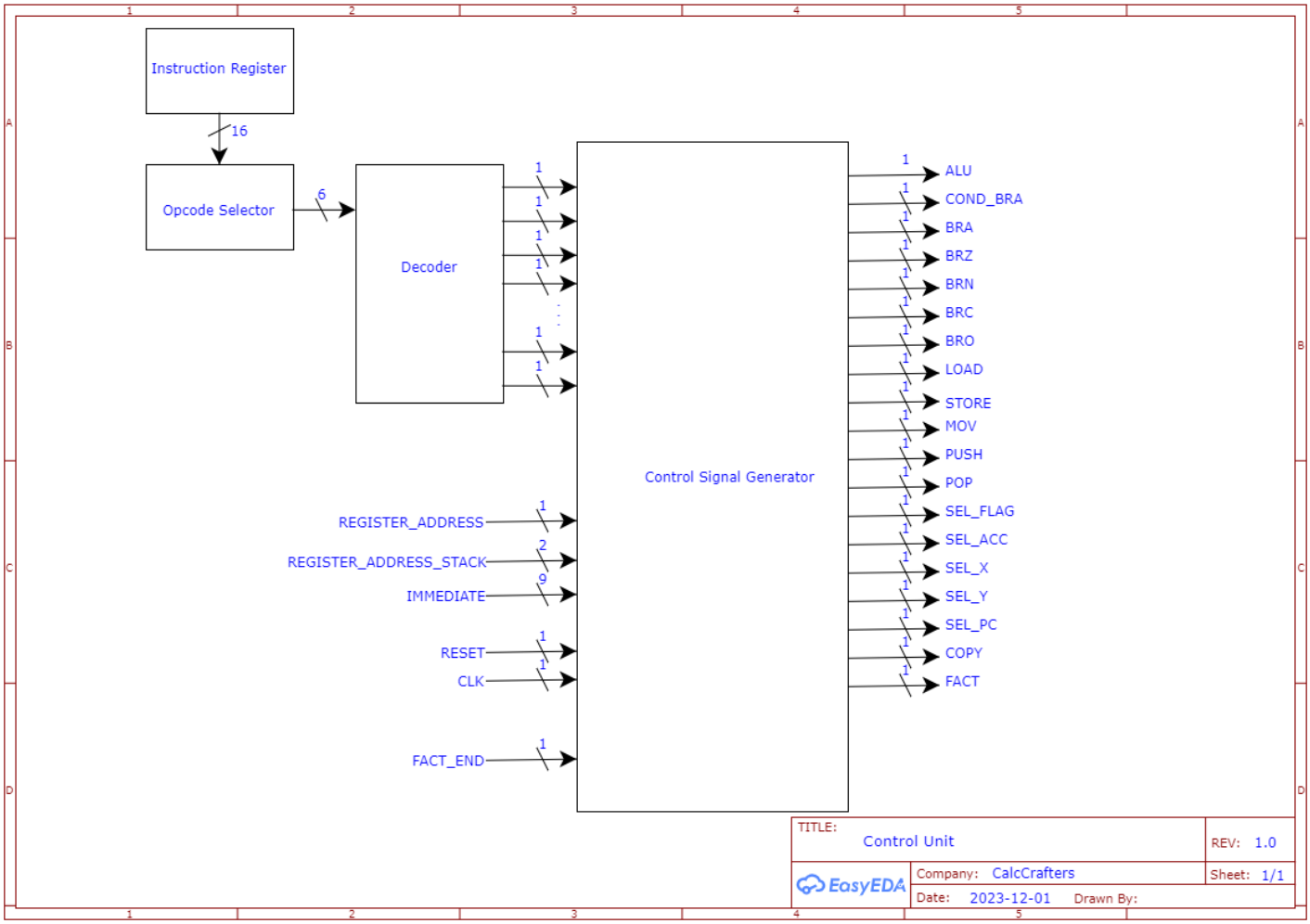
Nume instructiune – Opcode <6>	Unused <1>	Descriere
NOP – 100000	0000000000	Genereaza un delay de un ciclu de clock.

3. DESIGN HARDWARE

SCHEMA PROCESORULUI:



SCHEMA GENERATORULUI DE SEMNALE DE CONTROL (CONTROL SIGNAL GENERATOR):



INTRARILE GENERATORULUI:

OPCODE care este decodificata cu ajutorul unui decodificator

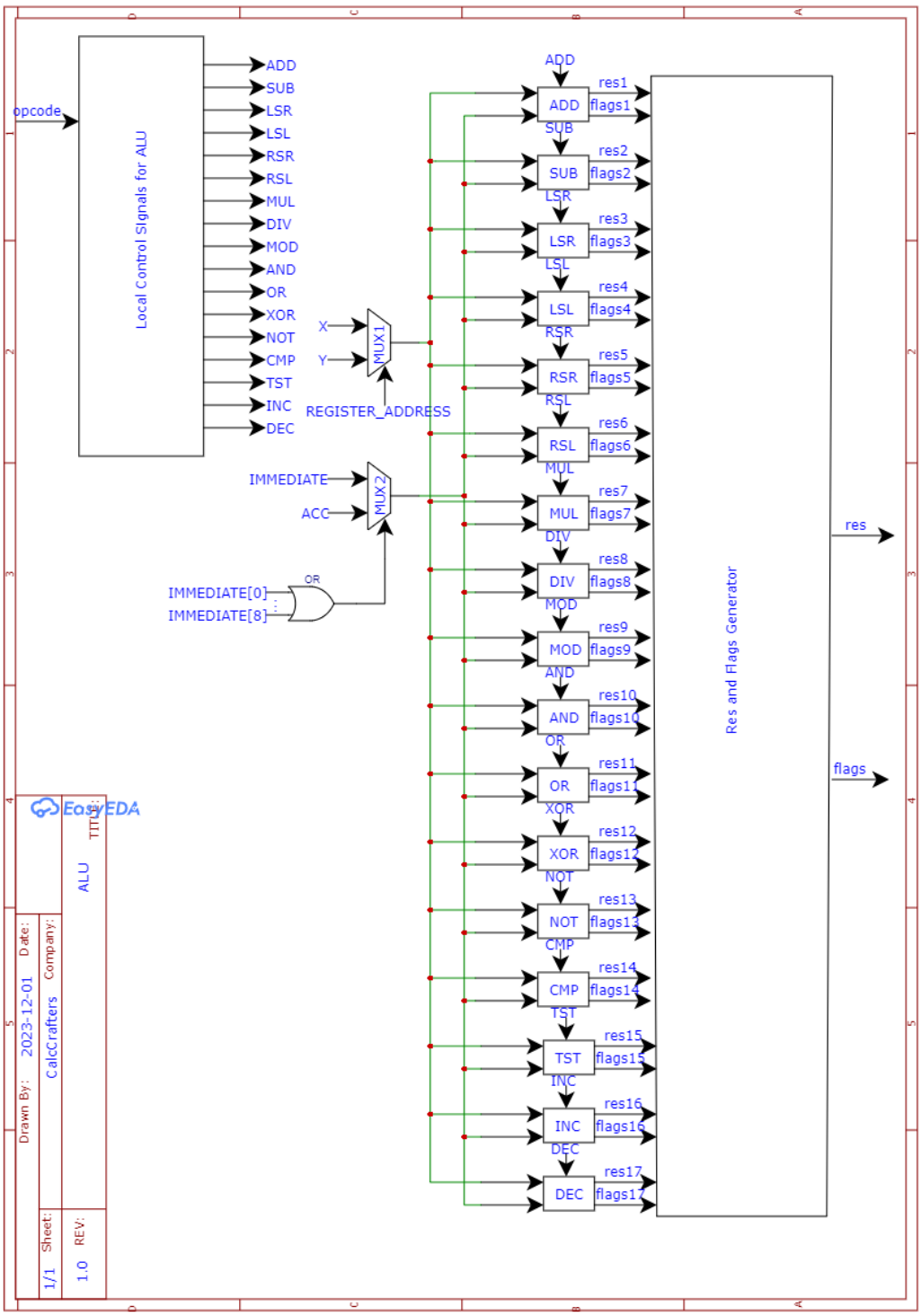
REGISTER_ADDRESS intrare de decizie folosita pentru instructiunile in care adresa este pe un bit (cand se folosesc registrii X si Y)

REGISTER_ADDRESS_STACK intrare de decizie folosita pentru instructiunile de PUSH si POP, pentru care adresa este pe 2 biti (cand se folosesc registrii ACC, X si Y)
IMMEDIATE intrare de decizie pe 9 biti folosita cand adresa este pe un bit
CLK
RESET
FACT_END intrare care dezactiveaza modulul responsabil de calculul factorialului

IESIRILE GENERATORULUI

ALU: se pune pe 1 cand se executa o instructiune aritmetica sau logica – intrarea de enable de la ALU
COND_BRA: se pune pe 1 atunci cand se executa o instructiune de salt conditionat
BRA: se pune pe 1 atunci cand se realizeaza o instructiune de salt (conditionat sau neconditionat)
BRZ: se pune pe 1 atunci cand se executa o instructiune de salt conditonat cu flag-ul ZERO
BRN: se pune pe 1 atunci cand se executa o instructiune de salt conditonat cu flag-ul NEGATIVE
BRC: se pune pe 1 atunci cand se executa o instructiune de salt conditonat cu flag-ul CARRY
BRO: se pune pe 1 atunci cand se executa o instructiune de salt conditonat cu flag-ul OVERFLOW
LOAD: se pune pe 1 cand se preiau date din Data Memory (inclusive la POP)
STORE: se pune pe 1 cand se incarca date in Data Memory (inclusiv la PUSH)
MOV: se pune pe 1 atunci cand se executa o instructiune de mutare a valorii immediate in X sau Y
PUSH: se pune pe 1 atunci cand se executa instructiunea de PUSH
POP: se pune pe 1 atunci cand se executa instructiunea de POP
SEL_FLAG: se pune pe 1 atunci cand se modifica valorile din registrul de flags (doar in cazul operatiilor aritmetice si logice)
SEL_ACC: se pune pe 1 atunci cand se actualizeaza acumulatorul (S=0) sau se preiau valori din accumulator pentru a fi incarcate in Data Memory (S=1). Pentru instructiunile aritmetice si logice se pune pe 1 doar cand IMMEDIATE=0
SEL_X: se pune pe 1 atunci cand se actualizeaza X sau se preiau valori din X pentru a fi incarcate in Data Memory. Pentru instructiunile aritmetice si logice se pune pe 1 doar cand IMMEDIATE!=0 si REGISTER_ADDRESS=0
SEL_Y: se pune pe 1 atunci cand se actualizeaza Y sau se preiau valori din Y pentru a fi incarcate in Data Memory. Pentru instructiunile aritmetice si logice se pune pe 1 doar cand IMMEDIATE!=0 si REGISTER_ADDRESS=1
SEL_PC: se pune pe 1 atunci cand se actualizeaza PC sau cand se preiau valori din PC pentru a fi incarcate in Data Memory
COPY: se pune pe 1 atunci cand se realizeaza o instructiune de copiere din ACC in X sau Y
FACT: se pune pe 1 atunci cand se executa instructiunea factorial
Pentru semnalele SEL_ACC, SEL_X, SEL_Y, SEL_PC, pentru actualizarea registrului S=0, iar pentru preluarea datelor S=1 (pentru multiplexorul de selectie a intrarii in Data Memory).

ALU:


opcode
Local Control Signals for ALU
ADD
SUB
LSR
LSL
RSR
RSL
MUL
DIV
MOD
AND
OR
XOR
NOT
CMP
TST
INC
DEC
X
Y
MUX1
REGISTER_ADDRESS
IMMEDIATE
ACC
MUX2
IMMEDIATE[0]
IMMEDIATE[8]
OR
res1
flags1
res2
flags2
res3
flags3
res4
flags4
res5
flags5
res6
flags6
res7
flags7
res8
flags8
res9
flags9
res10
flags10
res11
flags11
res12
flags12
res13
flags13
res14
flags14
res15
flags15
res16
flags16
res17
flags17
Res and Flags Generator
res
flags
EasyEDA
Drawn By:
Date: 2023-12-01
Company: CalcCrafters
Sheet: 1/1
REV: 1.0
TITLE: ALU

INTRARI:

ACC
X
Y
IMMEDIATE
ALU contine o unitate de control locala care selecteaza operatiile aritmetice si logice pe baza opcode-ului. Registrii de intrare, cu ajutorul carora se realizeaza operatiile sunt selectati cu ajutorul semnalelor REGISTER_ADDRESS si IMMEDIATE.

Exemplu: instructiunea ADD

Atunci cand IMMEDIATE are valoarea 0, se va alege utilizarea registrilor ACC si X sau Y.

=> ACC=ACC+X

Atunci cand IMMEDIATE este diferit de 0, se va alege utilizarea registrilor X sau Y si valoarea din IMMEDIATE

=> X=X+IMMEDIATE

Valoarea lui REGISTER_ADDRESS determina care dintre registrii X si Y va fi ales.

Atunci cand REGISTER_ADDRESS este 0 va fi ales registrul X.

Atunci cand REGISTER_ADDRESS este 1 va fi ales registrul Y.

IMMEDIATE	REGISTER_ADDRESS	operation
0	0	ACC=ACC+X
0	1	ACC=ACC+Y
!0	0	X=X+IMMEDIATE
!0	1	Y=Y+IMMEDIATE

FLAG-urile sunt modificate de instructiunile aritmetice si logice.

• **ZERO**: daca in urma efectuarii operatiei aritmetice sau logice rezultatul este 0, acest flag este setat pe 1;

• **NEGATIVE**: daca in urma efectuarii operatiei aritmetice sau logice rezultatul are bitul de semn 1, acest flag este setat pe 1;

• **CARRY**: daca se aduna doua numere iar rezultatul depaseste valoarea maxima alocata pe acel numar de biti), acest flag este setat pe 1. Acest flag nu se seteaza in cazul operatiilor de shiftare si impartire, ci ramane pe 0;

• **OVERFLOW**: daca se aduna doua numere de acelasi semn si rezultatul e de semn contrar, acest flag este setat pe 1. Acest flag nu se seteaza in cazul operatiilor de shiftare si impartire, ci ramane pe 0;

4. MODULE

Registrii ACC, X, Y

Fiecare registru din cei 3 este pe 16 biti si este folosit pentru stocarea valorilor. Fiecare registru are 4 intrari: IN (intrarea care aduce valoarea), CLK (semnalul de tact), RST (semnalul de reset) - ambele active pe frontul descrescator si EN (semnal pentru actualizarea valorii din registru).

Daca semnalul EN este activ (pe frontul crescator), atunci ALU pe toata perioada acestui front va scrie in registru, va prelua valoarea, o va scrie iar si procesul continua la infinit, nedorindu-ne acest lucru. Astfel pentru ca operatiile sa functioneze corect, am ales ca registrii sa fie activi pe frontul descrescator astfel incat etapele de fetch si execute sa se execute pe palierul pozitiv al semnalului de tact, iar actualizarea registrilor si a PC-ului sa se realizeze pe frontul descrescator al acestuia.

Multiplexorul pentru selectia intrarii in registrii ACC, X, Y

Exista mai multe surse care pot sa actualizeze valorile din registrii si anume, iesirea acumulatorului, pentru instructiunile de copiere a unei valori, iesirea ALU in urma efectuarii unor operatii aritmetice si logice, iesirea din Data Memory pentru operatia de LOAD, dar si instructiunile de MOV (unde se muta valoarea Immediate in unul din registrii X sau Y). Selectia iesirii se realizeaza cu ajutorul semnalelor COPY, ALU, LOAD si MOV.

ALU	COPY	MOV	LOAD	REG
1	0	0	0	ALU_OUT=RES
0	1	0	0	ACC_CPY=ACC_VALUE
0	0	1	0	MOV_INPUT=IMMEDIATE
0	0	0	1	DM_OUTPUT

Registrul FLAGS

Acest registru este pe 4 biti, iar valoarea sa poate fi actualizata doar de ALU atunci cand avem cazuri de overflow (bitul 0), carry (bitul 1), negative (bitul 2) si zero (bitul 3). Iesirile sale (OVERFLOW_FLAG, CARRY_FLAG, NEGATIVE_FLAG, ZERO_FLAG) reprezinta starea in care se afla registrul, iar semnalul de CLK este activ pe 1,iar cel de RST este activ pe 0. EN reprezinta semnalul pentru actualizarea valorii din registru.

Stack Pointer

Este folosit pentru a adresa locatia din varful stivei. La o operatie de PUSH se decrementeaza SP, iar la o operatie de POP se incrementeaza SP.

Instruction Memory

Reprezinta memoria ROM de unde procesorul incepe sa execute instructiunile pornind de la adresa 0. Aceasta are 1024 locatii de memorie x 16 biti. Numarul 1024 vine de la faptul ca adresa de branching/salt pentru instructiunile de branch este pe 10 biti (2^{10})

Ca si intrari avem ADDR care contine adresa de memorie determinata de Program Counter si semnalul de reset RST activ pe 0 (astfel ca output-ul va fi de fapt instructiunea de la adresa 0).

Semnalul de output reprezinta instructiunea de la adresa ADDR.

Instruction Register

Instruction Register are ca intrare iesirea data de Instruction Mmemory si anume o instructiune. Din iesirea lui Instruction Memory, adica din instructiune se extrag urmatoarele:

- OPCODE(6 biti): bitii 15-10 din intructiune
- REGISTER_ADDRESS(1 bit): bitul 9 din instructiune
- REGISTER_ADDRESS_STACK(2 biti): bitii 9-8 din intructiune
- BRANCH_ADDRESS(BA)(10 biti): bitii 9-0 din instructiune
- IMMEDIATE(9 biti): bitii 8-0 din instructiune

Iesirile sunt validate de semnalele de control, care sunt generate de Control Unit, deoarece prin intermediul semnalelor de control se cunoaste tipul instructiunii executate.

Program Counter

Program Counter-ul reprezinta un registru pe 16 biti, folosit la numerotarea intructiunilor.

Program Counter-ul se incrementeaza la fiecare front crescator al semnalului de tact doar daca nu se face operatia factorial, POP sau branching si se resetaza pe frontul descrescator al clock-ului. Factorialul e singura instructiune care nu se executa intr-un singur ciclu de clock, ci in mai multe.

Multiplexorul pentru intrarea in Program Counter

Este nevoie de acest multiplexor, deoarece in Program Counter se pot incarca diferite valori (de exemplu, valori din instructiuni de RET sau branch) si trebuie selectata intrarea corespunzatoare care va intra ca input in Program Counter si il va actualiza. Astfel ca semnalele STACK_POP si BRA realizeaza aceasta selectie.

Data Memory

Data Memory este un RAM care are 512 locatii(9 biti de adresa - 2^9) x 16 biti. Scrierea se realizeaza prin operatii de STORE sau PUSH, iar citirea prin operatii de LOAD sau POP.

Intrarile sunt iesirile multiplexoarelor de selectie a adresei si a intrarii (DM_IN si DM_ADDR). Intrarea IN este o intrare pe 16 biti prin care se primeste valoarea care trebuie stocata sau preluata din memorie, iar intrarea ADDR este o intrare pe 9 biti prin care se specifica adresa din memorie la care vrem sa stocam sau sa incarcam datele. Semnalul de clk functioneaza pe nivelul pozitiv, iar semnalul de reset pe frontul descrescator.

Exista si o intrare (EN) care determina operatia care este realizata in memorie (1 - se stocheaza date la adresa specificata).

Modulul are o singura iesire, care reprezinta de fapt valoarea stocata in locatia de memorie specificata la intrare.

Multiplexorul pentru selectia datelor care intra in Data Memory

Acest multiplexor selecteaza ceea ce se stocheaza in Data Memory, deoarece datele pot ajunge date din mai multe locatii. Astfel acest multiplexor alege ce data va ajunge in Data Memory. Selectarea intrarii se face prin SEL_ACC, SEL_X, SEL_Y, SEL_PC si STORE.

SEL_ACC	SEL_X	SEL_Y	SEL_PC	STORE	DM_IN
1	0	0	0	1	ACC_VALUE
0	1	0	0	1	X_VALUE
0	0	1	0	1	Y_VALUE
0	0	0	1	1	PC_VALUE

Multiplexorul pentru selectia adreselor care intra in Data Memory

Acest multiplexor selecteaza locatia care este accesata in Data Memory, deoarece adresa poate fi generata prin Stack Pointer sau printr-o valoare Immediate. Iesirea se selecteaza cu ajutorul intrarilor de selectie LOAD, STORE, PUSH si POP.

LOAD	STORE	PUSH	POP	DM_ADDR
1	0	0	0	Immediate
0	1	0	0	Immediate
0	0	1	0	SP
0	0	0	1	SP

Zero extend

Este un modul cu o intrare pe 10 biti si o iesire pe 16 biti, valoarea iesirii fiind egala cu cea a intrarii, diferenta constand in faptul ca se completeaza cu zero-uri inaintea celor 10 biti. Astfel acest modul este folosit pentru extinderea adresei de branch de la 10 biti la 16 biti inainte sa fie incarcata in Program Counter, iar iesirea lui este intrare in multiplexorul ce selecteaza intrarea in Program Counter.

Sign Extend

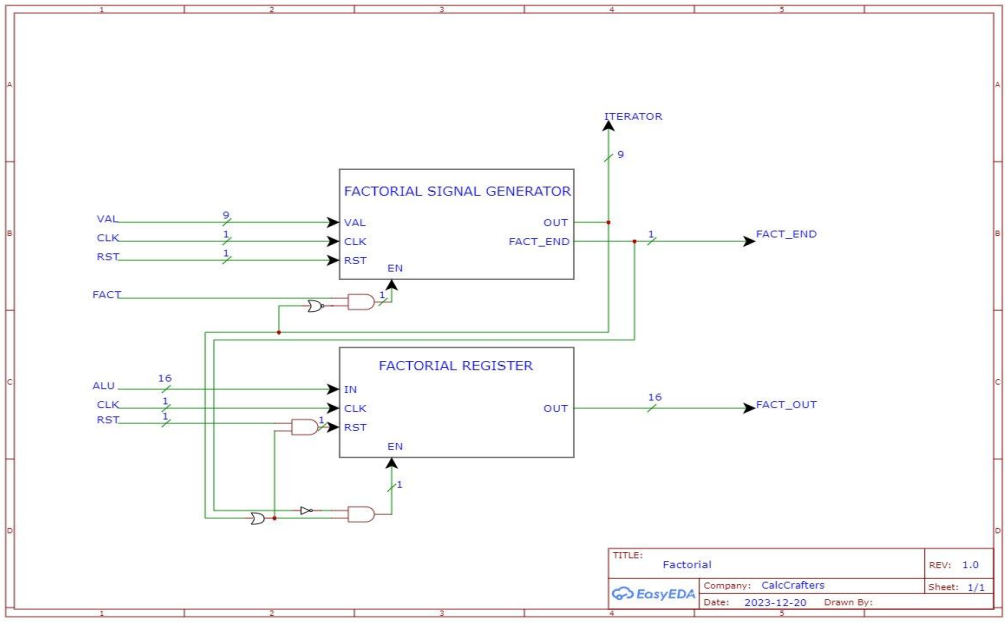
Este un modul cu o intrare pe 9 biti si o iesire pe 16 biti, in care valoarea iesirii este egala cu cea a intrarii, primii biti complectandu-se cu bitul de semn al intrarii. Este folosit pentru a extinde valoarea Immediate la 16 biti, atunci cand intra in ALU.

Modulul factorial

Acest modul este compus din alte doua submodule: Factorial Signal Generator si Factorial Register.

La inceput in primul submodul se incarca valoarea pentru care se va calcula factorialul, iar acea valoare la fiecare semnal de tact va fi decrementata cu o unitate pana la valoarea 1, atunci cand va genera semnalul FACT_END prin care anunta unitatea centrala ca s-a terminat de calculat factorialul. Din Control Unit va mai iesi un semnal FACT care atunci cand este 1 inseamna ca modulul factorial se executa, iar daca ce 0 inseamna ca nu se executa. Astfel cand FACT_END este generat, semnalul FACT se pune automat pe 0, deci Control Unit stie sa dezactiveze semnalul FACT atunci cand factorialul s-a terminat de calculat. Mai mult, atunci cand factorialul se executa, Program Counter-ul nu trebuie incrementat, de aceea FACT e folosit ca si input in PC (factorialul e singurul care se executa pe mai multe cicluri de clock).

Al doilea submodul, Factorial Register, se actualizeaza la fiecare sfarsit de ciclu de clock cu valoarea factorialului din iteratia curenta data de ALU, pastrand astfel rezultatele intermediare. In ALU va intra valoarea din Factorial Signal Generator extinsa de modulul Zero Extended si valoarea curenta din Factorial Register intre care se va realiza o inmultire, al carei rezultat va fi memorat din nou in al doilea submodul, Factorial Register pentru a putea fi folosit la iteratiile viitoare. Acest submodul se reseteaza de fiecare data cand s-a terminat de calculat factorialul pentru a putea fi folosit ulterior la alte calcule de factorial (la un reset acest modul este pus pe 1). ALU va decide cand FACT_END e pus pe 1 in care registru se face scrierea (X sau Y).



5. DESIGN SOFTWARE.

Interfata registrilor Acumulator, X, Y:

```
module Acumulator(
  input CLK,RESET,EN,
  input [15:0] IN,
  output reg [15:0] OUT
);

module X(
  input CLK,RST,EN,
  input [15:0] IN,
  output reg [15:0] OUT
);

module Y(
  input CLK,RST,EN,
  input [15:0] IN,
  output reg [15:0] OUT
);
```

Interfata multiplexorului de selectie al intrarii in ACC, X, Y:

```
module MUX_WRITE_REG(
  input [15:0] ALU_INPUT,
  input [15:0] MOV_INPUT,
  input [15:0] DM_INPUT,
  input [15:0] ACC_COPY,
  input ALU,MOV,LOAD,COPY,
  output reg [15:0] IN
);
```

Interfata registrului de flags:

```
module flags(
  input CLK,RST,EN,
  input [3:0] IN,
  output reg OVERFLOW,CARRY,NEGATIVE,ZERO
);
```

Interfata Stack Pointer-ului:

```
module StackPointer(
  input CLK,RST,INC,DEC,
  output reg [15:0] OUT
);
```

Interfata memoriei de instructiuni:

```
module InstructionMemory(
  input RST,
  input [9:0] ADDRESS,
  output reg [15:0] OUT
);
```

Interfata registrului de instructiuni:

```
module InstructionRegister(
  input CLK,RST,EN,
  input [15:0] IN,
  output reg [15:0] OUT,
  output reg [5:0] OPCODE,
  output reg REGISTER_ADDRESS,
  output reg [1:0] REGISTER_ADDRESS_STACK,
  output reg [8:0] IMMEDIATE,
  output reg [9:0] BA
);
```

Interfata Program Counter-ului:

```
module ProgramCounter(
  input CLK,RST,EN,BRA,POP,FACT,
  input [15:0] IN,
  output reg [15:0] OUT
);
```

Interfata multiplexorului de selectie al intrarii in Program Counter:

```
module ProgramCounterInputDecider(
   input POP, BRA,
   input [15:0] BRAInput,
   input [15:0] POPInput,
   output reg [15:0] ProgramCounterInput
);
```

Interfata memoriei de date:

```
module DataMemory(
  input CLK,RST,EN,
  input [8:0] ADDR,
  input signed [15:0] IN,
  output reg signed [15:0] OUT
);
```

Interfata multiplexorului de selectie al intrarii in Data Memory:

```
module REG_MUX_DM_ADDR(
  input LOAD,STORE,PUSH,POP,
  input [8:0] SP,
  input [8:0] IMMEDIATE,
  output reg [8:0] IN
);
```

Interfata multiplexorului de selectie al locatiei in Data Memory:

```
module REG_MUX_DM_INPUT(
  input [15:0] X,
  input [15:0] Y,
  input [15:0] ACC,
  input [15:0] PC,
  input SEL_X,SEL_Y,SEL_ACC,SEL_PC,STORE,
  output reg [15:0] IN
);
```

Interfata Zero Extend- Bit Converter:

```
module BitConverter10x16(
  input [9:0] IN,
  output reg [15:0] OUT
);
```

Interfata Sign Extend:

```
module SignalExtender9x16(
  input [8:0] IN,
  output reg [15:0] OUT
);
```

Interfata unitatii de control:

```
module ControlUnit(
  input [5:0] OPCODE,
  input REGISTER_ADDRESS, CLK, RST,
  input [1:0] REGISTER_ADDRESS_STACK,
  input [8:0] IMMEDIATE,
  input FACT_END,
  output reg ALU, BRA, COND_BRA, BRZ, BRN, BRC, BRO, LOAD, STORE, COPY, PUSH, POP, MOV,
SEL_FLAG, SEL_ACC, SEL_X, SEL_Y, SEL_PC,
  output reg FACT
```

```
);
```

Interfata unitatii aritmetice si logice:

```
module ALU(
  input signed [15:0] ACC,X,Y,IMMEDIATE,
  input [15:0] fact_reg,fact_val,
  input [5:0] OPCODE,
  input EN,CLK,RST,REGISTER_ADDRESS,
  output reg signed [15:0] res,
  output reg [3:0] flags
); //FLAGS[Z=3,N=2,C=1,O=0]
```

Interfata modulului factorial:

```
module FACTORIAL(
  input [8:0] VAL,
  input CLK, RST, FACT,
  input [15:0] ALU,
  output reg FACT_END
  );
```

Interfata timer-ului:

```
module FACT_CNT(
  input [8:0] CNT,
  input CLK, RST, EN,
  output reg [8:0] OUT,
  output reg FACT_END
);
```

Interfata registrului pentru factorial:

```
module FACT_REG(
  input [15:0] IN,
  input CLK,RST,EN,
```

```
  output reg [15:0] OUT
);
```

Codul sursa complet se regaseste pe repository-ul de Github a carui link e atasat pe prima pagina a acestei documentatii.

6. TESTARE

Crt.	Operatie	x	y	Rezultat asteptat	Rezultat primit	Test trecut/picat	Observatii
1	Adunare	3	4	7	7	trecut	
2	Adunare	-3	4	1	1	trecut	
3	Adunare	-5	5	0	0	trecut	Rezultat 0
4	Adunare	6	-7	-1			Rezultat negativ
5	Adunare	32766	1	32767			Overflow
6	Adunare	32767	3				Carry
7	Scadere	7	3	4	4	trecut	
8	Scadere	-4	5	-9	-9	trecut	
9	Scadere	8	-2	10			
10	Scadere	5	6	-1			Rezultat negativ
11	Scadere	10	10	0			Rezultat 0
12	Scadere						Carry
13	Scadere						Overflow
14	Inmultire	10	2	20	20	trecut	
15	Inmultire	5	-3	-15			Rezultat negativ
16	Inmultire	-4	-6	24			
17	Inmultire	9	0	0			Rezultat 0
18	Inmultire						Carry
19	Inmultire	3000	2				Overflow
20	Impartire	10	2	5			
21	Impartire	-9	3	-3			Rezultat negativ
22	Impartire	-5	-5	1			
23	Impartire	3	0	error			Impartire cu 0

7. BIBLIOGRAFIE

- Digital Design and Computer Architecture, Sarah L. Harris & David Money Harris
- Computer Organization and Design – Arm Edition, David A. Patterson & John L. Hennessy
- Cursuri FIC