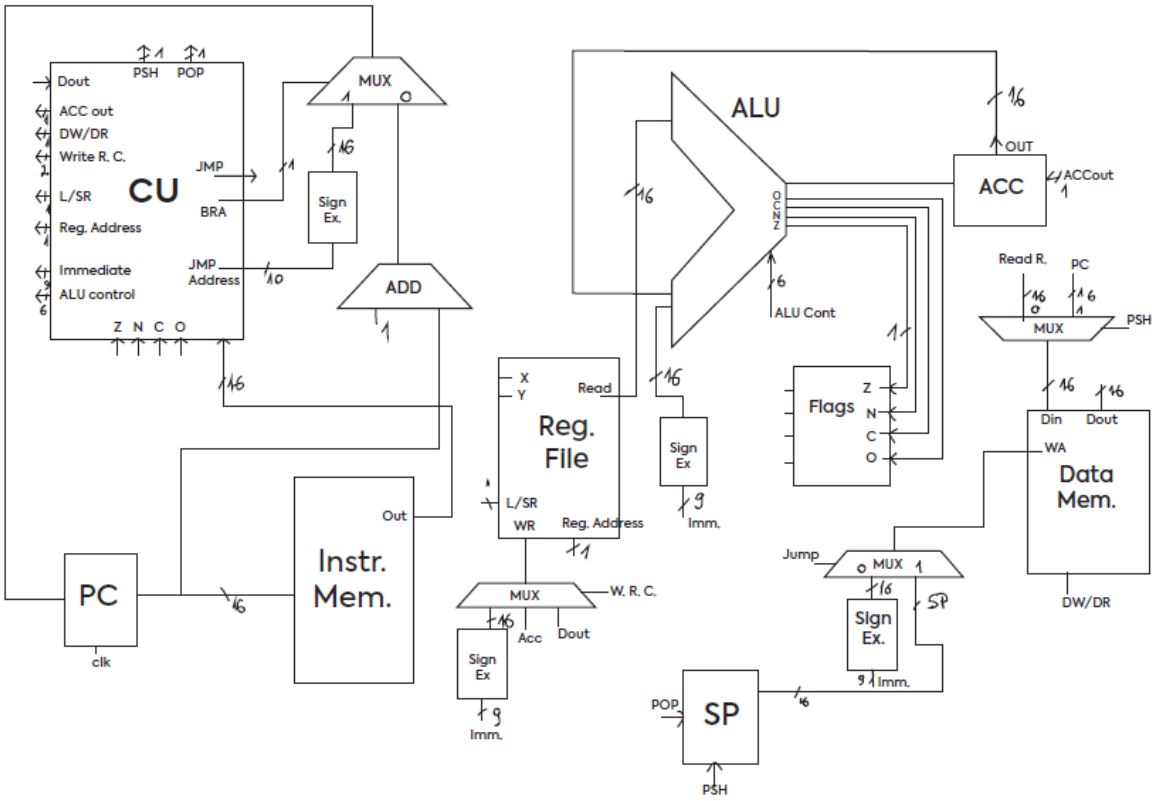
Documentatie HW

Introducere

RISC sau Reduced Instruction Set Computer este o filozofie de design care a devenit mainstream în ultimii ani, deoarece căutarea vitezei brute a dominat foarte mult industria informatică competitivă. Există dorința de a îmbunătățiți viteza procesorului și de a simplifica hardware-ul din motive de cost. Designul RISC a dus la computere care execută instrucțiuni mai rapid decât alte computere construite de aceeași tehnologie.

Într-o mașină RISC, setul de instrucțiuni se bazează pe o abordare de load/store. Doar instrucțiunile de load si store accesează memoria. Nicio instrucțiune aritmetica, logica sau I/O nu operează direct pe conținutul memoriei. Aceasta este cheia pentru executarea instrucţiunilor într-un singur ciclu. Simplificarea rezultă într-un decodor de instrucțiuni care este mic, rapid și relativ ușor de proiectat.

(tradus din Charles și Veljko, 1989)


PSH
POP
Dout
ACC out
DW/DR
Write R. C.
L/SR
Reg. Address
Immediate
ALU control
Z N C O
CU
JMP
BRA
JMP Address
MUX
Sign Ex.
ADD
ALU
OCNZ
ALU Cont
OUT
ACC
ACCout
Read R.
PC
MUX
PSH
Flags
Z
N
C
O
X
Y
Read
Reg. File
L/SR
WR
Reg. Address
Sign Ex
Imm.
Din
Dout
WA
Data Mem.
DW/DR
Out
Instr. Mem.
PC
clk
MUX
W. R. C.
Acc
Dout
Sign Ex
Imm.
Jump
MUX
SP
Sign Ex.
Imm.
POP
SP
PSH

Implementare

Implementarea unui RISC executa fetch, decode si execute intr un singur ciclu de clock. Arhitectura acestui RISC pe un singur ciclu este impartita in 4 stagii: Instruction fetch, Instruction decode, Execute si Data memory.

Instruction fetch

Primul stagiu este Instruction Fetch(IF). Operatiile stagiului IF incep cu Program Counter ul(PC) care acceseaza Instruction Memory prin intermediu unei adrese. Dupa ce se gaseste adresa din Instruction Memory, transmitem continutul de la acea adresa, catre Control Unit(CU) unde acesta va fi decodat.Program conter ul e conectat si la un Adder, iar dupa aceea se ajunge la un Multiplexor(MUX). Acest multiplexor alege daca vom trece la urmatoarea adresa sau in cazul in care avem un semnal din CU pe BRA sau JMP Adress va sari la adresa respectiva.

Instruction decode

Dupa stagiul IF, Control Unit ul va decodifica continutul primit si va obtine opcode ul, primii 6 biti din cei 16 transmisi, care indica instructiunea dorita iar mai apoi CU va incarca semnalele corespondente instructiunii.

In cazul in care vorbim despre o instructiune de branch sau jump cei 16 biti se vor imparti in felul urmator:

Opcode	Address
6 bit	10 bit

Semnalul BRA va fi incarcat cu valoarea 1, se tine cont de flag uri in cazul unei instructiuni de branch mai complexa iar cei 10 biti vor fi incarcati in semnalul JMP Adress care cum am spus si mai sus va indica adresa la care ne dorim sa ajungem.

Pentru toate restul instructiunilor impartirea pe cei 16 biti se va face astfel:

Opcode	Register Address	Immediate
6 bit	1 bit	9 bit

CU incarca semnalele Address sau Register Adress si Immediate cu valorile date.

LOAD/STORE – in cazul unora din aceste instructiuni (diferentiate de semnalul L/S R pe un bit), se mai incarca semnalul Write Register Control (WRC) care ii va spune unui Multiplexor daca informatia din Immediate este valida sau nu; in cazul in care adresa este valida aceasta ajunge in Register File, iar daca este invalida vom incarca tot acolo ce se afla in Accumulator

ALU Instructions – CU incarca opcode ul in semnalul ALU Control care mai departe ii specifica modulului Arithmetic-Logic Unit (ALU) ce instructiune trebuie folosita; register address trece prin Register File si ajunge tot in ALU, iar Immediate asemenea instructiunii descrise anterior, daca este valida merge direct in ALU, daca nu vom incarca din Accumulator;

Execute

Dupa stagiul ID, ALU va primi doua intrari pentru a face o executie. Sunt 2 variante pentru cele doua intrari, prima in care vom avea ca intrari Register si Immediate si cea de a doua in care vom lua valoarea din Accumulator in loc de cea Immediate.

ALU Control, in functie de opcode ul primit, trimite tipul de instructiune care trebuie executat intre 2 operanzi in ALU. Acesta executa instructiunea si updateaza mai departe flagurile iar rezultatul merge mai departe in accumulator.

Data Memory

Doua dintre functionalitatile lui Data Memory sunt: Data Write (DW) si Data Read (DR).

In cazul DW, Data in (Din) este pus pe 1, primim o valoare din Read Register si aceasta ajunge pe Write Adress (WA) primit din Immediate. In cazul DR, Data out (Dout) este pus pe 1, avem aceiasi adresa WA si datele vot ajunge in CU.

In cazul unei instructiuni de JMP, vom avea cate un Push (PSH) de la PC la Stack Pointer (SP) la fiecare instructiune de acest gen. In SP daca nu suntem pe ultima pozitie se va incrementa cu o unitate. WA va retine adresa din SP si va fii incarcata cu informatie din PC. (DW)

La finanul procedurii vom avea un ret care in sistemul nostru se concretizeaza printr un POP. Va avea loc un DR asa cum am descris si mai sus iar informatia din Dout va ajunge la JMP Adress+1 ca sa continuam cu urmatoarea instructiune.