Koncepcia počítačov CISC a RISC

Obsah prednášky

- základné odlišnosti CISC a RISC koncepcie
- základné spôsoby realizácie riadiacej jednotky
- OISC (One Instruction Set Computer)

CISC

- počítač so zložitým súborom inštrukcií (**Complex Instruction Set Computer**)
- prvý procesor vyvinutý firmou IBM
- využíva veľký počet zložitých inštrukcií
- inštrukcie nie sú registrovo orientované (využívajú operandy uložené v pamäti)
- inštrukcie nie sú vykonávané v jednom strojovom cykle
- využíva dátové presuny medzi pamäťami
- riadiaca jednotka využíva koncepciu mikroprogramov
- inštrukcie majú variabilný formát (rôzne dĺžky inštrukcií)

RISC

- počítač s redukovaným súborom inštrukcií (**Reduced Instruction Set Computer**)
- využíva malý počet inštrukcií
- obsahuje pomerne veľký počet registrov
- inštrukcie sú registrovo orientované (využívajú operandy uložené v registroch)
- na prístup do pamäte využíva len inštrukcie LOAD/STORE
- väčšina inštrukcií je vykonávaná v jednom (strojovom) cykle
- riadiaca jednota je tvorená pevnou logikou
- formát inštrukcií je fixný formát (rovnaká dĺžka inštrukcií)

Difference Between CISC and RISC

Architectural Characterstics	Complex Instruction Set Computer(CISC)	Reduced Instruction Set Computer(RISC)
Instruction size and format	Large set of instructions with variable formats (16-64 bits per instruction).	Small set of instructions with fixed format (32 bit).
Data transfer	Memory to memory.	Register to register.
CPU control	Most micro coded using control memory (ROM) but modern CISC use hardwired control.	Mostly hardwired without control memory.
Instruction type	Not register based instructions.	Register based instructions.
Memory access	More memory access.	Less memory access.
Clocks	Includes multi-clocks.	Includes single clock.
Instruction nature	Instructions are complex.	Instructions are reduced and simple.

História RISC

RISC koncept bol vytvorený neskôr ako CISC
Stručný opis histórie RISC procesorov
(https://www.fi.muni.cz/usr/jkucera/pv109/2002/xtoufar1.htm)
koncept RISC v r. 1974 (IBM)
RISC 1, RISC 2 v r. 1985 (univerzita v Berkeley)

http://poli.cs.vsb.cz/edu/arp/down/procrisc.pdf

Trendy vývoja RISC

Aktuálne sa hranice medzi CISC a RISC značne prelínajú a výrobcovia často využívajú výhodné vlastnosti z oboch koncepcií.

Koncept RISC-V (uvedený v roku 2010, univerzita v Berkeley), využíva **„open source“ ISA**, t.j. za využitie ISA nie je potrebné platiť licenčné poplatky (na rozdiel od napr. ISA pre procesory Intel, ARM, ...) - https://en.wikipedia.org/wiki/RISC-V
škálovateľná architektúra (32, 64, 128 bitov)
využíva sa napr. aj v obvodoch FPGA (Microsemi)

Niektoré mýty a skutočnosti o RISC-V (https://www.electronicdesign.com/embedded-revolution/11-myths-about-risc-v-isa)

Riadiaca jednotka

Riadiaca jednotka (RJ) generuje riadiace signály a časovo synchronizuje výkon operácií v CPU. RJ riadi vykonávanie inštrukcií v ALU. RJ tiež riadi prenos dát medzi CPU, pamäťou a rôznymi perifériami.

Základné koncepcie navrhu a realizácie riadiacej jednotky
- realizovaná pomocou **pevnej logiky**
- reealizovaná pomocou **mikroprogramového riadenia**

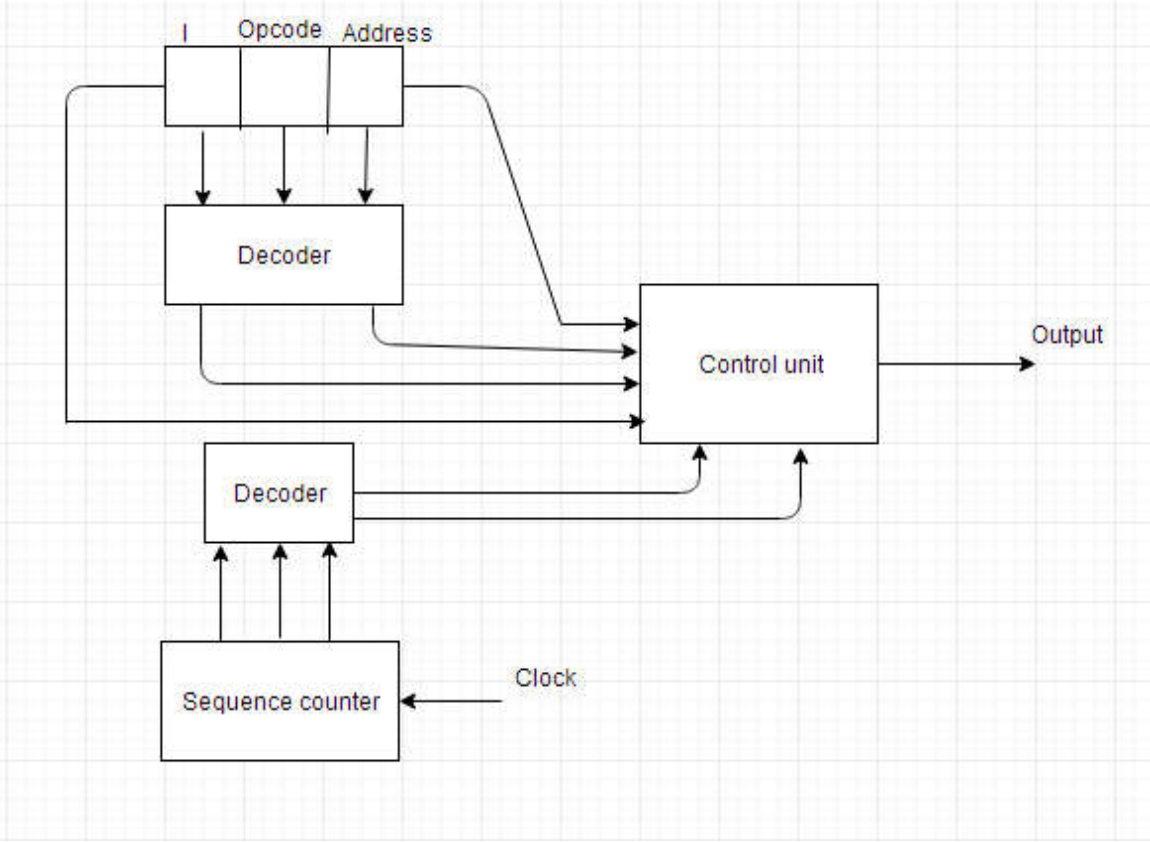
Riadiaca jednotla na báze pevnej logiky (Hardwired Control Unit)

Je vytvorená s využitím hradiel, preklápacích obvodov, dekóderov a pod. Ich prepojením priamo v cieľovom HW. Vstupmi riadiacej jednotky sú obsah inštrukčného registra, príznaky, časovacie signály a pod.

Zapojenie RJ na báze pevnej logiky sa výrazne komplikuje, pokiaľ musí RJ realizovať zložité inštrukcie. Každá zmena RJ vyžaduje HW zmeny (redizajn), čo je pomerne komplikované.

Typicky sa využíva pre procesory RISC.

RJ na báze pevnej logiky



Mikroprogramová riadiaca jednotka

Využíva koncepciu programu. Sekvencia mikro-operácií je vykonávana programom, ktorý je tvorený mikro-inštrukciami. Ich organizácia, modifikácia alebo zmeny sa realizujú zmenou mikroprogramu v riadaicej pameti.

Využíva sa v procesoroch CISC.

Difference between Hardwired Control and Microprogrammed Control

Hardwired Control	Microprogrammed Control
Technology is circuit based.	Technology is software based.
It is implemented through flip-flops, gates, decoders etc.	Microinstructions generate signals to control the execution of instructions.
Fixed instruction format.	Variable instruction format (16-64 bits per instruction).
Instructions are register based.	Instructions are not register based.
ROM is not used.	ROM is used.
It is used in RISC.	It is used in CISC.
Faster decoding.	Slower decoding.
Difficult to modify.	Easily modified.
Chip area is less.	Chip area is large.

One Instruction Set Computer (OISC)

OISC je limitným prípad RISC koncepcie, niekedy je nazývaný aj Ultimate Reduced Instruction Set Computer (URISC) a využíva len jednu inštrukciu. Vhodne zvolená inštrukcia a dostatok dostupných zdrojov (pamäte) umožňuje OISC realizovať identickú funkcionalitu ako klasické počítače RISC alebo CISC.

OISC koncepcia sa využíva hlavne vo výučbe (vzhľadom na veľmi jednoduhý koncept riešenia RJ), existujú však aj praktické využitia koncepcie OISC napr. v implementácií kryptografického koprocesora.

V rámci cvičení bude demonštrovaný návrh a implementácia OISC v obvode FPGA, kde využijeme možnosť veľmi jednoduchej realizácie riadiacej jednotky.

One Instruction Set Computer (OISC)

Ďalšie informácie:

https://en.wikipedia.org/wiki/One_instruction_set_computer

kniha:
Computer Architecture: A Minimalist Perspective
(https://www.springer.com/la/book/9781402074165)

Typické inštrukcie použiteľné pre OISC

Subtract and branch if less than or equal to zero **(SUBLEQ)**
Subtract and branch if negative
Subtract if positive else branch
Reverse subtract and skip if borrow
Subtract and branch if non zero

SUBLEQ inštrukcia

Inštrukcia SUBLEQ ("*subtract and branch if less than or equal to zero*") odčíta obsah pamäte s adresou “a” od obsahu pamäte s adresou “b”, uloží výsledok do pamäte na adresu “b”. Následne, ak výsledok odčítania nie je kladný (teda je záporný alebo nulový), pokračuje sa vykonávaním inštrukcie na adrese “c”. Ak je výsledok odčítania kladný, realizuje sa vykonávanie nasledujúcej inštrukcie.

Pseudokód:

subleq a, b, c ; Mem[b] = Mem[b] - Mem[a] ; if (Mem[b] ≤ 0) goto c

Emulácia

Nasledujúci program (zapísaný pomocou pseudokódu) emuluje činnosť OSIC na báze SUBLEQ inštrukcie:

```
int memory[], program_counter, a, b, c
program_counter = 0
while (program_counter >= 0):
    a = memory[program_counter]
    b = memory[program_counter+1]
    c = memory[program_counter+2]
    if (a < 0 or b < 0):
        program_counter = -1
    else:
        memory[b] = memory[b] - memory[a]
        if (memory[b] > 0):
            program_counter += 3
        else:
            program_counter = c
```

Program predpokladá, ze memory[] je adresovaná pomocou nezáporných celých čísel. Program interpretuje hodnoty a < 0, b < 0, alebo skok na adresu c < 0 ako zastavenie programu (halt).

Kompilácia

Existuje prekladač (s názvom **Higher Subleq**), autorom je Oleg Mazonka, ktorý prekladá zjednodušený C program do postupnosti SUBLEQ inštrukcií .

kompilátor (opis)
https://arxiv.org/ftp/arxiv/papers/1106/1106.2593.pdf

kompilátor pre SUBLEQ OISC (zdrojový kód, príklady programov, …)
http://mazonka.com/subleq/hsq.html

OSIC Interpreter v jazyku C

https://github.com/davidar/subleq

Kompaktný (veľkosť iba 222 bajtov) SUBLEQ interpreter v jazyku C:

```
#include<stdio.h>
main(int C,char**A){FILE*F=fopen(A[1],"r");int P=0,_[9999],*i=_;while(fscanf(F,
"%d",i++)>0);while(P>=0){int a=_[P++],b=_[P++],c=_[P++];a<0?_[b]+=getchar():b<0
?printf("%c",_[a]):(_[b]-=_[a])<=0?P=c:0;}}
```

OSIC Interpreter v jazyku C

```
/**
This file is the unobfuscated and commented version of sq.c, a simple SUBLEQ
interpreter. It should function similarly to sqrun[1].

USAGE:
sq file.sq

file.sq should be a file of the same format as that output by sqasm[1].
It must not contain any unresolved symbols
    e.g. #IN and #OUT should be replaced by -1

[1] <http://mazonka.com/subleq/>
*/

#include <stdio.h>
int main(int argc, char *argv[]) {
    FILE *fin = fopen(argv[1], "r");
    int PC = 0; /* Program Counter */
    int mem[9999];
    int *i = mem;
    while(fscanf(fin, "%d", i++) > 0); /* read fin into mem */
    while(PC >= 0) {
        int a = mem[PC++], b = mem[PC++], c = mem[PC++];
        if(a < 0) mem[b] += (int)getchar(); /* input */
        else if(b < 0) printf("%c", (char)mem[a]); /* output */
        else if((mem[b] -= mem[a]) <= 0) PC = c; /* subtract and branch */
    }
}
```

https://github.com/davidar/subleq/blob/master/src/sq.orig.c

Ten istý C kód zapísaný v čitateľnejšom tvare

Iný OSIC Interpreter v jazyku C aj s kódom Hello, world!

https://rosettacode.org/wiki/Subleq

Takes the subleq instruction file as input, prints out usage on incorrect invocation.

```c
#include<stdlib.h>
#include<stdio.h>

void subleq(int* code){
        int ip = 0, a, b, c, nextIP,i;
        char ch;

        while(0<=ip){
                nextIP = ip + 3;
                a = code[ip];
                b = code[ip+1];
                c = code[ip+2];

                if(a==-1){
                        scanf("%c",&ch);
                        code[b] = (int)ch;
                }
                else if(b==-1){
                        printf("%c",(char)code[a]);
                }
                else{
                        code[b] -= code[a];
                        if(code[b]<=0)
                                nextIP = c;
                }
                ip = nextIP;
        }
}
```

Input file (subleqCode.txt), first row contains the number of code points (integers in 2nd row):

```
32
15 17 -1 17 -1 -1 16 1 -1 16 3 -1 15 15 0 0 -1 72 101 108 108 111 44 32 119 111 114 108 100 33 10 0
```

Invocation and output:

```
C:\rosettaCode>subleq.exe subleqCode.txt
Hello, world!
```