SISTEME DE CALCUL. LIMBAJ DE ASAMBLARE. SIMULATORUL QTSPIM

I. Sisteme de calcul

1. Arhitectura generală a unui sistem de calcul

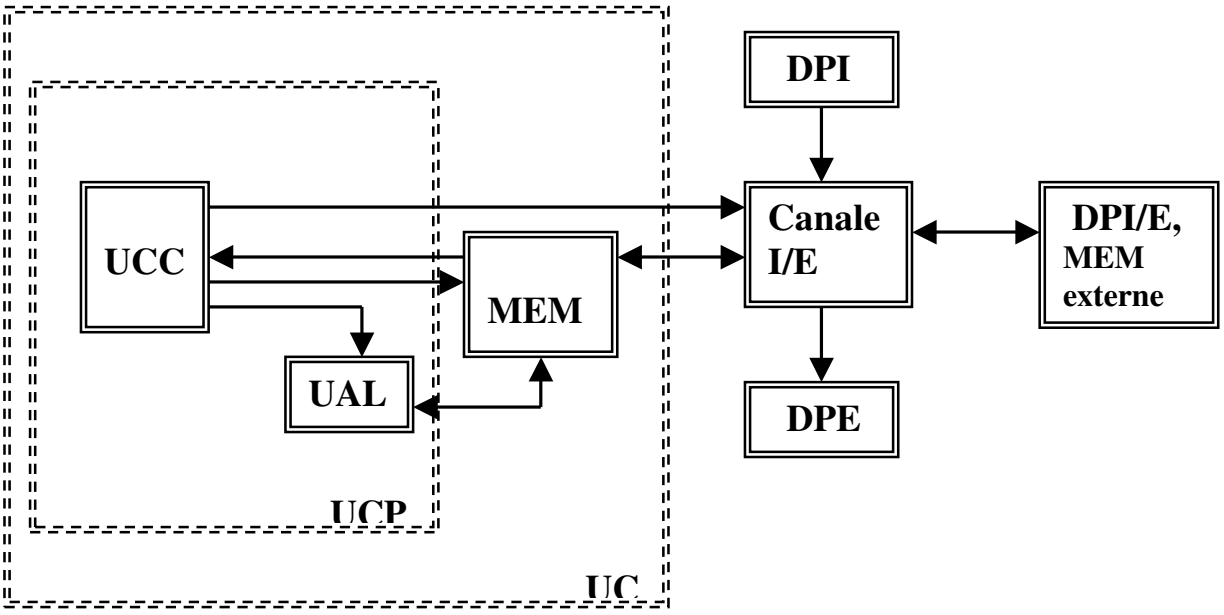


Fig.1. Structura unui sistem de calcul

📖 Structura generala a unui sistem de calcul conţine ca blocuri componente:

- **Dispozitive periferice de intrare (DPI)** - dispozitive de introducere a datelor în vederea prelucrării lor. Exemple: tastatura, mouse-ul, scanner-ul, microfonul, joystick-uri, camera web, etc.
- **Dispozitive periferice de ieşire (DPE)** – dispozitive prin care se redau rezultatele prelucrării. Exemple: monitorul, imprimanta, boxele, retroproiectoare, etc.
- **Canale de intrare/ieşire** - componente care dirijează fluxul transferat la dispozitivele de intrare/ieşire;
- **Memoria (MEM) –** componenta sistemului de calcul care desemnează modul fizic de stocare internă a datelor şi instrucţiunilor programelor. Memoriile interne se divid în 2 categorii: RAM (Random Access Memory) şi ROM (Read Only Memory);
- **Unitatea aritmetico-logică (UAL) –** execută operaţii aritmetice (adunare, scădere, înmulţire, împărţire) şi logice (negaţie, OR,

AND) cu datele care îi sunt furnizate din memorie, unde depune şi rezultatul operaţiei;
- **Unitatea de comandă şi control (UCC) –** extrage instrucţiuni din memorie, le interpretează şi emite comenzi corespunzătoare către unitatea aritmetico-logică, memorie şi dispozitivele de intrare/ieşire prin intermediul canalelor de intrare/ieşire;
- **Dispozitive periferice de intrare şi ieşire, memorii externe, etc.** ;

UCP (**U**nitatea **C**entrala de **P**relucrare) este formată din UCC şi UAL. Împreună cu memoria, reprezintă **U**nitatea **C**entrală (**UC**).

2. Arhitecturi CISC şi RISC

📖 Totalitatea instrucţiunilor pe care le poate executa un anumit procesor, reprezintă setul său de instrucţiuni. Împreună cu tipurile de date, regiştrii, modurile posibile de adresare, tratarea excepţiilor etc. constituie **ISA** (**I**nstruction **S**et **A**rhitecture).

După tip, instrucţiunile se pot clasifica în:
- **Instrucţiuni aritmetice** - exemple: ADD, SUB, DIV, MUL;
- **Instrucţiuni logice** - exemple: AND, OR, NOT;
- **Instrucţiuni de date** - exemple: MOVE, LOAD, STORE;
- **Instrucţiuni de control al fluxului -** exemple: GOTO, CALL, RETURN.

Fiecare tip de procesor are un set diferit de instrucţiuni, respectiv un ISA diferit!

? Întrebări:
Care părţi componente ale UCP sunt utilizate la execuţia fiecărui tip de instrucţiune?

📖 Din punct de vedere al caracteristicilor setului de instrucţiuni utilizat, se evidenţiază 2 clase principale de procesoare:
- **CISC** (**C**omplex **I**nstruction **S**et **C**omputer) – Procesoarele CISC sunt reprezentate de un set de instrucţiuni complexe. Abordarea CISC încearcă să minimizeze numărul de instrucţiuni per program, cu compromisul creşterii timpului de execuţie per instrucţiune;
- **RISC** (**R**educed **I**nstruction **S**et **C**omputer) – Procesoarele RISC sunt reprezentate de un set de instrucţiuni simple şi foarte rapid de executat. Abordarea RISC este exact opusă celei CISC, reducând timpul de execuţie al unei instrucţiuni cu costul creşterii numărului de instrucţiuni per program.

O comparare a proprietăţilor fundamentale ale celor 2 arhitecturi se regăseşte în tabelul următor:

CISC	RISC
• Include instrucţiuni complexe, care necesită timp de execuţie mai mare (se desfăşoară pe parcursul mai multor tacte de ceas); • Conduce la dimensiuni mai mici ale codului; • O parte a complexităţii software este implementată hardware; • Implementarea hardware suportă tipuri de date mai complexe;	• Include numai instrucţiuni simple, executate rapid (de obicei într-un singur tact de ceas, datorită tehnicii pipeline); • Conduce la dimensiuni mai mari ale codului; • Tendinţă de accentuare pe parte software; • Prezintă mai puţine tipuri de date hardware; • Prezintă un format uniform al instrucţiunilor, ceea ce conduce la o codificare mai simplă; • Minimalizează utilizarea memoriei prin existenţa unui număr mare de regiştri (datele sunt aduse în şi din memorie prin intermediul acestor regiştri) şi permiterea utilizării multiple a regiştrilor generali, care sunt identici.

Arhitectura RISC este implementată de procesoarele MIPS, ALPHA, ARC, Nintendo, SPARC. Dintre microprocesoarele CISC, cel mai cunoscut este Intel 8086.

Microprocesoarele MIPS (Microprocessor with Interlocked Pipeline Stages) au fost realizate la Stanford în anii `80 de o echipă condusă de prof. J.Henessy. Acestea prezintă arhitecturi pe 32 si 64 de biţi. Familia MIPS este destul de numeroasă, cam 1/3 din microprocesoarele RISC sunt MIPS. Ca exemplu, acestea sunt folosite de către anumite familii de routere (Cisco) şi console de jocuri (Nindendo şi PlayStation).

Se exemplifică câteva dintre proprietăţile amintite prin multiplicarea a 2 numere care se găsesc la locaţiile x şi y din memorie:

CISC	RISC
multiply x,y,z	*load r1,x* *load r2,y* *prod r1,r2,r3* *store z,r3*

? Întrebări:

1) Care sunt proprietăţile care se evidenţiază în exemplul de mai sus?
2) Ce sunt regiştrii?

II. Limbajul de asamblare

📖 O **instrucţiune maşină** reprezintă o secvenţă de biţi care corespunde unei operaţii de bază a procesorului (ex.: adunarea a 2 numere, citirea unor date dintr-un registru, etc.), executată direct de acesta. **Codul maşină** reprezintă un sistem de astfel de instrucţiuni. Acesta este direct dependent de arhitectura procesorului.

Limbajul de asamblare este o imagine simbolică a codului maşină care se bazează pe reprezentarea instrucţiunilor (prin intermediul unor operation code – o codare specifică operaţiei care se execută), a regiştrilor, a locaţiilor de memorie, etc., devenind accesibil programatorilor. Limbajul de asamblare este specific unei arhitecturi hardware, fiind deci un limbaj de nivel scăzut (low level programming language).

Programele scrise în limbaj de asamblare (**codul sursă**) trebuie reduse prin asamblare la **codul obiect** (program în cod maşină) pentru a putea fi interpretate de procesor, în acest scop fiind folosite asambloarele.

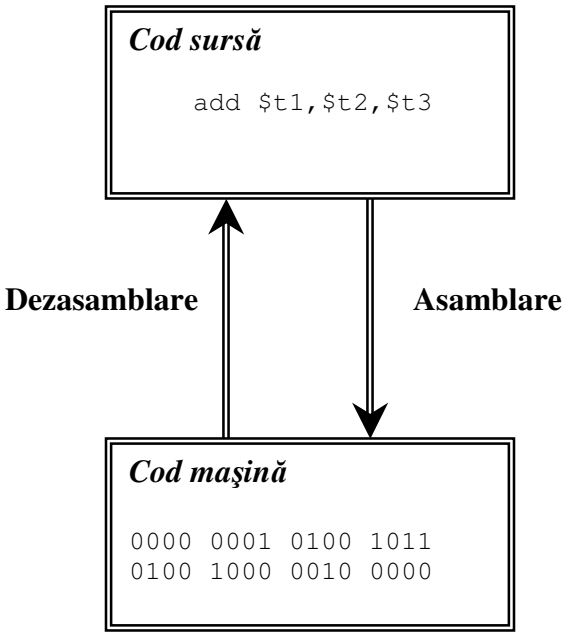


Fig2. Asamblarea

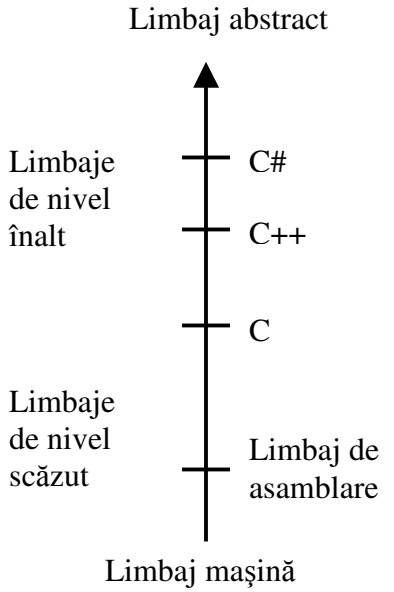


Fig.3 Limbaje de programare

❓ Întrebări:

1) Ce înseamnă un limbaj de nivel înalt (high-level)/ de nivel scăzut (low-level)?
2) Comparaţi procesul de asamblare cu cel de compilare.

De ce nu s-ar scrie un program în limbaj de asamblare?

- Este greu de scris (scrierea programelor este în general ineficientă);
- Este greu de citit si înţeles;
- Este greu de depanat;

De ce s-ar scrie un program în limbaj de asamblare?

- Cele mai rapide programe se scriu în limbaj de asamblare;
- Programele scrise în limbaj de asamblare ocupă cel mai puţin spaţiu în memorie;
- Oferă grad de libertate sporit;
- Oferă o bună înţelegere a modului de funcţionare a procesorului.

📖 Seria completă a paşilor pe care un procesor îi urmează la execuţia unei instrucţiuni se numeşte **ciclu maşină.** Acesta constă din următorii paşi:

1. UCP citeşte din memorie o instrucţiune (**fetch**);
2. Interpretează codul maşină (**decode**);
3. Realizează operaţia indicată de instrucţiune (**execute**);
4. Accesează memoria;
5. Scrie rezultatul în regiştri;

Adresa următoarei instrucţiuni care va fi executată se păstrează într-un registru special, numit **PC** (**P**rogram **C**ounter).

După încărcarea instrucţiunii de la adresa indicată de PC, acesta se incrementează pentru a indica adresa următoarei instrucţiuni care va urma să fie executată.

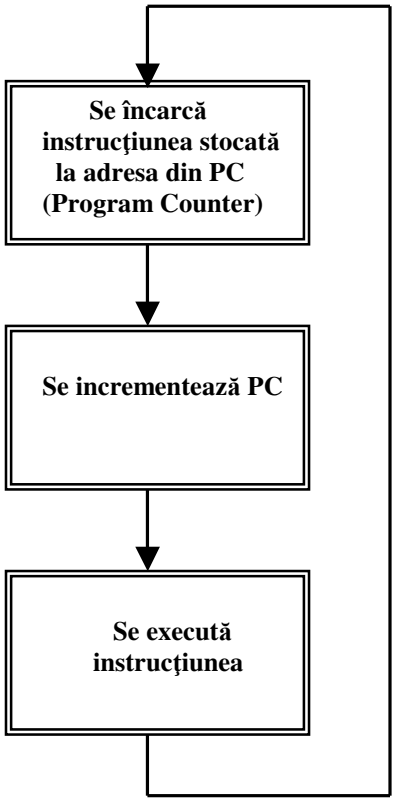


Fig.4. Execuţia unei instrucţiuni

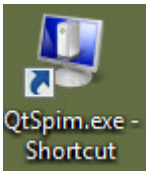
III. QtSpim

🕮 QtSpim este un simulator capabil să ruleze programe scrise în limbajul de asamblare pentru procesoarele MIPS32. QtSpim oferă interfaţă vizuală Windows, MAC OS X şi Linux.

ⓘ Mai mult informaţii se găsesc pe pagina SPIM:

http://spimsimulator.sourceforge.net/

🖫 Utilizarea QtSpim:



1. Accesaţi QtSpim.
2. Se deschid 2 ferestre: fereastra principală a simulatorului QtSpim şi o fereastră de consolă (Console), care simulează ecranul maşinii virtuale. Interacţiunea cu utilizatorul se face în această fereastră (introducerea datelor, afişarea datelor, etc.).
3. Interfaţă QtSpim prezintă 3 panouri:
   - Panoul din stanga conţine regiştrii intregi şi în virgulă mobilă (FP = Floating Point) ai procesorului MIPS, accesibili prin taburile ***Int Regs [16],*** respectiv ***FP Regs***. Aceştia se actualizează după execuţia programului sau la fiecare pas, în cazul rulării pas cu pas.
   - Panoul din dreapta afişează zonele de memorie corespunzătoare *Text Segment* (zona de memorie în care sunt stocate instrucţiunile programului, în cod maşină) şi *Data Segment* (zona de memorie în care sunt stocate datele aduse în memoria programului şi datele din stivă), accesibile prin taburile ***Text***, respectiv ***Data***;
   - Panoul de jos este utilizat pentru afişarea către utilizator a diferitelor mesaje, precum încărcarea cu succes a unui program, mesaje de eroare, etc.
4. Modificarea valorilor regiştrilor se efectuează prin click dreapta pe registrul a cărui valoare se doreşte a fi modificată, apoi *Change Register Contents.*
5. Programele se editează într-un fişier text, salvat cu extensia **.s.**
6. Încărcarea unui program se face din meniul *File*, opţiunile *Load File*, respectiv *Reinitialize and Load File* sau apasând butoanele  , respectiv  din interfaţă. Diferenţa între cele 2 moduri de încărcare a unui program constă în reiniţializarea regiştrilor la valorile prestabilite în cel de-al doilea caz, în timp ce regiştrii îşi păstreaza valorile curente în primul caz.

7. Rularea unui program se poate face direct sau pas cu pas.
   Pentru rularea continuă, există mai multe variante:
   - Meniul *Simulator* > *Run Parameters*;
   - Meniul *Similator* > *Run / Continue* (shortcut: F5);
   - Accesarea butonului *Run / Continue* din interfaţă.

   Pentru rularea pas cu pas:
   - Meniul *Simulator* > *Single Step* (shortcut: F10);
   - Accesarea butonului *Single Step* din interfaţă ;

   Pentru pauză în execuţia unui program:
   - Meniul *Simulator* > *Pause*;
   - Accesarea butonului *Pause* din interfaţă;

   Pentru oprirea execuţiei unui program:
   - Meniul *Simulator* > *Stop*;
   - Accesarea butonului *Stop* din interfaţă;

   Există de asemenea posibilitatea introducerii unor puncte de oprire înainte de execuţia unor instrucţiuni, prin click dreapta pe instrucţiunea dorită şi selecţie *Set Breakpoint*. Eliminarea unui breakpoint se realizează similar, cu *Clear Breakpoint*.

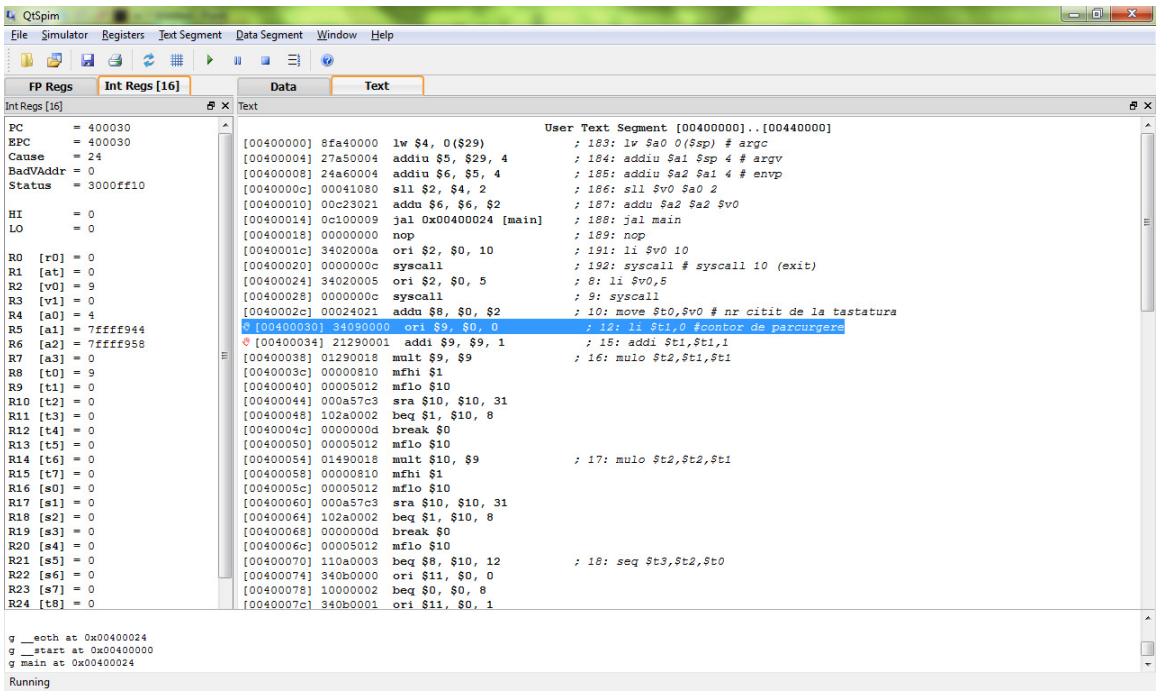


Fig.5 Fereastra principală QtSpim

? Întrebări:

1) Care este dimensiunea regiştrilor generali?
2) Ce se întâmplă dacă se încearcă introducerea într-un registru general a valorii 0x123456789? De ce?

Exerciţiu:

1. Salvaţi într-un fişier cu extensia .s următorul program:

```
.data
 # declaratii date
.text
 # cod
main: # eticheta marcand punctul de start
 # cod
add $t1,$t2,$t3
li $v0,10
syscall
```

2. Încărcaţi programul în QtSpim.
3. Observaţi încărcarea programului în zona de memorie text.
4. Observaţi zona de afişare a instrucţiunilor:
   - Fiecare instrucţiune este afişată pe o linie nouă;
   - Fiecare linie satisface un anumit format:
     - Primul număr, afişat între paranteze pătrate, reprezintă adresa de memorie a instrucţiunii, în hexazecimal;
     - Al doilea număr reprezintă codarea numerică a instrucţiunii, în format hexazecimal (codul maşină specific instrucţiunii).
     - Pe a treia coloană se găseşte descrierea instrucţiunii;
     - Urmează exact linia din fişierul .s încărcat, cu specificarea numărului pe care îl ocupă linia în fişier.
   - Fiecare instrucţiune ocupă 1 word (32 de biţi), ceea ce face ca adresele să se succeadă din 4 în 4 octeţi.

```
[00400024] 014b4820  add $9, $10, $11         ; 7: add $t1,$t2,$t3
[00400028] 3402000a  ori $2, $0, 10           ; 8: li $v0,10
[0040002c] 0000000c  syscall                  ; 9: syscall
```

1) Setaţi valoarea registrului PC la adresa de memorie a primei instrucţiuni.
2) Setaţi valorile regiştrilor $t2 şi $t3 la nişte valori oarecare.
3) Rulaţi programul.

? Întrebări:

1) Ce conţine după execuţie registrul $t1 ?
2) Care este adresa la care se memorează instrucţiunea *add $t1,$t2,$t3* ?
3) Ce reprezintă 014b4820 de pe coloana a 2-a a primei linii a programului încărcat în segmenul text?
4) În ce bază este considerată valoarea introdusă prin program cu li $v0, 10?
5) Este limbajul considerat case sensitive?
6) Care este simbolul interpretat ca fiind simbol de comentariu?
7) Cum procedaţi pentru a pune un breakpoint la pseudoinstrucţiunea *li $v0,10*?
8) Rulaţi programul pas cu pas observând incrementarea registrului PC. De ce se comportă astfel?

IV. Structura generală a unui program

📖 Un program are următoarea structură generală:

```
.data                          #  declaratii date
        …
.text                          # identifică porţiuni cu instrucţiuni
        …
main:                          # eticheta marcând punctul de start
        …

li $v0,10                      # se incarca valoarea 10 in registrul $v0
syscall                        # terminarea executiei programului
```

ⓘ Mai multe informaţii

Download si documentaţie QtSpim
http://spimsimulator.sourceforge.net/

Manual QtSpim
- accesibil direct din simulator prin *Help > View Help*.

Tutorial QtSpim
http://claws.eng.ua.edu/attachments/166_QtSpim%20Tutorial.pdf

Programmed Introduction to MIPS Assembly Language
http://chortle.ccsu.edu/AssemblyTutorial/index.html

Bit, Byte and Binary
ftp://ftp-sop.inria.fr/acacia/fgandon/lecture/uk1999/binary/HandOut.pdf