Limbajul mașinii

Principii de proiectare

Introducere

- Limbajul unui calculator = cuvinte + vocabular

  - Cuvintele -> instrucțiuni
  - Vocabularul -> set de instrucțiuni

Operațiile hardware-ului

- Orice calculator trebuie să fie capabil să efectueze operații aritmetice: add a, b, c
- Exemplu: a = b + c + d + e
  - add a, b, c
  - add a, a, d
  - add a, a, e

  CONCLUZIE: Este nevoie de 3 instrucțiuni
- Hardware-ul pentru implementarea unei operații cu 3 operanzi este mai simplu decât hardware-ul necesar implementării unei operații cu un număr variabil de operanzi
- PRINCIPIUL DE PROIECTARE 1 - Simplitatea favorizează uniformitatea

Exercițiu: Se dă următoarea comandă C:

f = (g+h) - (I+j)

Ce cod va produce un compilator C ?

Operațiile hardware-ului

- Orice calculator trebuie să fie capabil să efectueze operații aritmetice: add a, b, c
- Exemplu: a = b + c + d + e
  - add a, b, c
  - add a, a, d
  - add a, a, e

  CONCLUZIE: Este nevoie de 3 instrucțiuni
- Hardware-ul pentru implementarea unei operații cu 3 operanzi este mai simplu decât hardware-ul necesar implementării unei operații cu un număr variabil de operanzi
- PRINCIPIUL DE PROIECTARE 1 - Simplitatea favorizează uniformitatea

Exercițiu: Se dă următoarea comandă C:

f = (g+h) - (I+j)

Ce cod va produce un compilator C ?

Operanzii hardware-ului

- Operanzii instrucțiunilor aritmetice provin dintr-un număr limitat de locații speciale denumite **registre**
- Dimensiunea unui registru este de 32 biți => un cuvânt va avea 32 de biți
- Numărul de registre este limitat într-un calculator - 32 de registre generale
- PRINCIPIUL DE PROIECTARE 2 - Mai mic înseamnă mai rapid

Notații: $s0, $s1 ... registre folosite pentru variabile; $t0, $t1 ... registre temporare necesare compilării unui program în instrucțiuni MIPS

Exercițiu: Se dă următoarea comandă C:

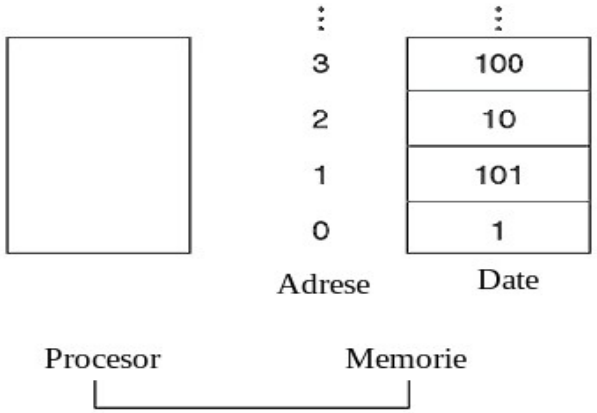
f = (g+h) - (I+j)

Ce cod va produce un compilator C ?

Soluție:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
add $s0, $t0, $t1
```

- Structurile de date foarte mari: matricile și vectorii sunt ținute în păstrate în memorie => necesitatea existenței instrucțiunilor care să realizez transferul datelor între memorie și registre
  INSTRUCȚIUNI DE TRANFER DE DATE



- Instrucțiunea care deplasează datele din memorie într-un registru este numită LOAD

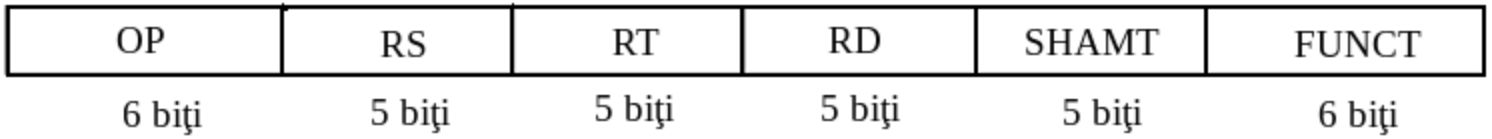
- Formatul instrucțiunii:
  - Numele operației
  - Registrul ce trebuie încărcat
  - Constantă
  - Registrul folosit pentru accesarea memoriei

- **Exercițiu**: A este un tablou de 100 de cuvinte, iar variabilele g și h sunt asociate cu registrele $s1 și $s2. Adresa de bază este în $s3. Cum se translatează g=h+A[8] ?

- Compilatorul alocă structurile de date locațiilor din memorie, apoi pună adresa corectă de început în instrucțiunea de transfer de date.
- Cuvintele secvențiale diferă prin 4. Cuvântul în MIPS trebuie să înceapă la o adresă care este multiplu de 4: RESTRICȚIE DE ALINIERE
  - Calculatoare pot folosi adresele celui mai din stânga octet - BIG END - ca adresă a cuvântului, sau folosesc octetul cel mai din dreapta - LITTLE END
  - Adresarea la nivel de octet afectează din nou indexarea tablourilor => offset-ul care trebuie adăugat registrului de bază $s3 trebuie să fie 4x8=32 biți pentru a selecta A[8] și nu A[8/4]
- Instrucțiunea complementară intsrucțiunii LOAD este instrucțiunea STORE (memorează)
- Formatul instrucțiunii STORE este următorul:
  - Nume operație
  - Registrul de memorat
  - Offset-ul pentru selectarea elementului din tablou
  - Registrul de bază
- **Exercițiu**: Variabila h este asociată cu registrul $s2 șI adresa de bază a tabloului A este în $s3. Detaliați A[12] = h + A[8]
- Memoriile actuale sunt foarte mari => adresa de bază a unui tablou este introdusă într-un registru deoarece n u este loc suficient în zona rezervată pentru offset.

Reprezentarea instrucțiunilor

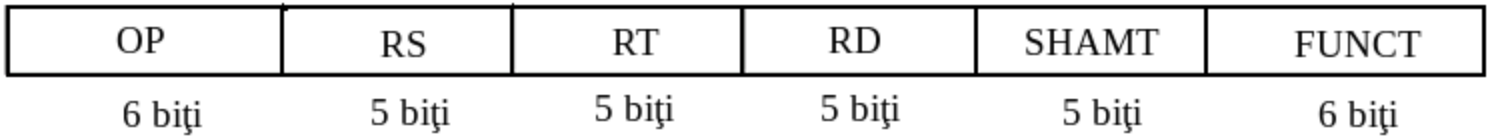
Instrucțiunile sunt păstrate în calculator ca o serie de semnale electronice înalte sau joase => pot fi reprezentate ca numere



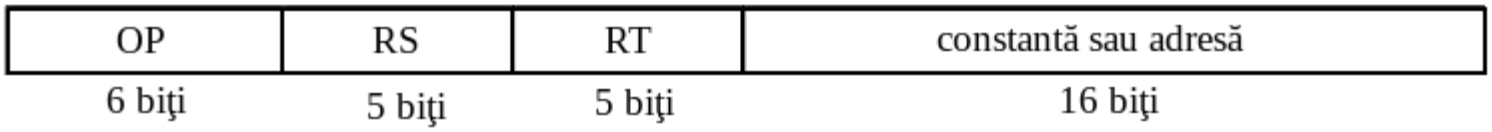
op	OPCODE
rs	registru sursă pentru primul operand
rt	registru sursă pentru al doilea operand
rd	registrul operandului de destinație ce primește rezultatul operației
shamt	numărul deplasărilor
funct	selectează varianta specifică a operației din câmpul op

- PRINCIPIUL DE PROIECTARE 3 - Proiectarea bună necesită compromisuri bune

- Instrucțiunea de tip R



- Instrucțiunea de tip I



- Folosirea mai multor tipuri de instrucțiuni => complicarea hardware-ului, dar putem reduce complexitatea dacă păstrăm formate asemănătoare

- PRINCIPII DE BAZĂ
  - Instrucțiunile sunt reprezentate ca numere
  - Programele pot fi păstrate în memorie pentru a putea fi citite/scrise precum numerele

INSTRUCȚIUNI DE DECIZIE

- Ramificații condiționale

  beq registru_1, registru_2, L1 #dacă registru_1 = registr_2 dute la eticheta L1

  bne registru_1, registru_2, L1 # dacă registru_1 ≠ registru_2 dute la eticheta L1

- Asamblorul este cel care calculează adresele pentru ramificații, calculează adresele instrucțiunilor de încărcare/memorare

- **Exercițiu**: Folosind registrele $s0 ... $s4 pentru cele 5 variabile de mai jos, să se detalieze codul generat pentru
  - If (i==j) f=g+h; else f=g-h

Proceduri

- Execuția unei proceduri presupune următorii pași

- Pune parametrii într-un loc unde pot fi accesați de către procedură
- Transferă controlul procedurii
- Obține resursele de memorie necesare procedurii
- Efectuare sarcină
- Pune valoarea rezultată într-un loc de unde poate fi accesată de către programul apelant
- Întoarce controlul la punctul de plecare

Registrele folosite:

$a0 - $a3	patru registre de argumente în care se comunică parametrii
$v0 - $v1	două registre de valori în care să întoarcă rezultatul
$ra	un registru al adresei de întoarcere

Folosirea mai multor registre

- Compilatorul are nevoie pentru o procedură de mai mult de 4 registre pentru argumente și 2 pentru întoarcerea rezultatului.

- Structura de date ideală pentru transferul în memorie al registrelor este stiva

$sp - pointer-ul stivei

PUSH

POP

Proceduri imbricate

- Procedura A are in teriorul ei procedura B sau cazurile de recursivitate

- Introducem în stivă toate registrele care trebuie păstrate după apel. Apelantul introduce în stivă registrul adresei de întoarcere $ra și orice registru salvat folosit de apelant.

- Pointer-ul stivei $sp este corectat pentru a lua în evidențănumărul registrelor puse în stivă. La întoarcere, registrele sunt refăcute din memorie, iar pointer-ul stivei este ajutat.

Modurile de adresare MIPS

- MIPS are următoarele moduri de adresare
    - Adresare a registrelor
    - Adresare prin bază sau deplasare
    - Adresare imediată
    - Adresare relativă la PC
    - Adresare pseudodirectă

Cei 4 pași necesari pentru conversia unui program