Arquitectura de Computadores

Introdução ao MIPS

Docente: Pedro Sobral
http://www.ufp.pt/~pmsobral

Pensamento

- *"I've found that the key difference between mediocre and excellent programmers is whether or not they know assembly language"*

-Slashdot.org

Assembler

- Função do CPU: executar MUITAS instruções
- Instruções: Operações básicas que o CPU sabe executar.
- Diferentes CPUs implementam diferentes *conjuntos de instruções*.
- O conjunto de instruções que um CPU implementa chama-se: "*Instruction Set Architecture* (*ISA*)".
  - Exemplos: Intel 80x86,ARM, MIPS, Intel IA64, ...

Conjuntos de Instruções

- CISC (Complex Instruction Set Computing)
  - Há algum tempo atrás a ideia era aumentar o conjunto de instruções com operações complexas em hardware:
    - A arquitectura do VAX tinha uma instrução para multiplicar polinómios!
- RISC (Reduced Instruction Set Computing)
  - Manter o conjunto de instruções pequeno e simples o que ajuda a construir hardware mais rápido.
  - Operações complexas são executadas em software compondo operações simples.

Arquitectura MIPS

- MIPS – Companhia que criou uma das primeiras implementações comerciais da arquitectura RISC
  - https://imgtec.com/mips/
- Porquê estudar MIPS em vez de Intel 80x86 ou ARM?
  - MIPS é simples e elegante.
  - MIPS é a mais pura implementação do conceito RISC
  - Conhecendo MIPS é simples perceber ARM

Most HP LaserJet workgroup printers are driven by MIPS-based™ 64-bit processors.

Variáveis em Assembler: Registos (1/4)

- Ao contrário das linguagens de alto nível como C ou Java em assembler não há variáveis.
  - Porque não? Manter o hardware simples!
- Os operandos em assembler são os ***registos***
  - Número limitado de localizações especiais criadas directamente no hardware.
  - Todas as operações são executas neles!
- Benefício: Os registos são muito rápidos! (menos 1e-9 seg)

Variáveis em Assembler: Registos (2/4)

- Inconveniente: Dado que os registos são implementados em hardware o seu nº é limitado…
- Solução: o código MIPS tem que fazer um uso eficiente ds registos.
- 32 registos no MIPS
  - Porquê 32? "Smaller is faster"
- Cada registo MIPS tem 32 bits
  - Um grupo de 32 bits chama-se: palavra ou word

Variáveis em Assembler: Registos (3/4)

- Os registos estão numerados de 0 a 31
- Cada registo pode ser referenciado pelo número …
- Referência numérica:
  - $0, $1, $2, … $30, $31

Variáveis em Assembler: Registos (4/4)

- Por convenção, cada registo pode também ser referênciado pelo nome…
- Por agora:
  - $16 - $23 ➔ $s0 - $s7
  - (correspondem às variáveis em C)
  - $8 - $15 ➔ $t0 - $t7
  - (correspondem a varáveis temporárias)
  - Os outros 16 nomes vêm mais tarde…
- Em geral devemos usar nomes para tornar o código mais legível….

Registos vs. Variáveis em C ou Java

- Em C, por exemplo, as variáveis são declaradas como sendo de um tipo.
  - Exemplo:
    int fahr, celsius;
    char a, b, c, d, e;
- Cada variável só pode contar dados desse mesmo tipo…
- Em assembler os registos não têm tipo; a operação a executar é que vai determinar o significado do seu conteúdo.

Comentários em Assembler

- Outra forma de tornar o código mais legível: Comentários!
- O cardinal (#) é usado para comentários no MIPS
- Qualquer coisa depois de um # e até ao fim da linha é ignorado…
- Nota: Diferente do C.
  - Comentários em C têm o formato
    /* comentário */
    portanto podem ocupar várias linhas…

Adição e subtracção no MIPS (1/4)

- Sintaxe das instruções:
  - 1 2,3,4
  - onde:
  - 1) Nome da operação
  - 2) Operando que recebe o resultado ("destino")
  - 3) 1º operando ("origem1")
  - 4) 2º operando ("origem2")
- A sintaxe é rígida:
  - 1 operador, 3 operandos
  - Porquê? Manter o hardware simples pela regularidade…

Adição e subtracção no MIPS (2/4)

- Adição em assembler
  - Exemplo: add $s0,$s1,$s2 (em MIPS)
  - Equivalente a: a = b + c (in C)
  - Onde os registos MIPS $s0,$s1,$s2 estão associados às variáveis a, b, c.
- Subtracção em assembler
  - Exemplo: sub $s3,$s4,$s5 (em MIPS)
  - Equivalente a: d = e - f (in C)
  - Onde os registos MIPS $s3,$s4,$s5 estão associados às variáveis d, e, f.

Adição e subtracção no MIPS (3/4)

- Como fica em assembler a declaração seguinte?

  a = b + c + d - e;

- Dividida em várias instruções…
  - add $t0, $s1, $s2 *# temp = b + c*
  - add $t0, $t0, $s3 *# temp = temp + d*
  - sub $s0, $t0, $s4 *# a = temp - e*
- Nota: uma linha em C pode ser convertida em diversas linhas de código MIPS.

Adição e subtracção no MIPS (4/4)

- Como é possível codificar em assembler do MIPS a declaração seguinte?

$$f = (g + h) - (i + j);$$

f em $s0...j em $s4

Usando registos temporários intermédios…

```
add $t0,$s1,$s2 # temp = g + h
add $t1,$s3,$s4 # temp = i + j
sub $s0,$t0,$t1  # f=(g+h)-(i+j)
```

Registo Zero

- Existe uma constante (“immediate”), o zero (0), que aparece frequentemente no código.
- Portanto definimos o registo zero ($0 ou $zero) que tem sempre o valor zero.
- Exemplo:
  - add $s0,$s1,$zero (em MIPS)
  - f = g (em C)
  - Onde os registos MIPS $s0,$s1 estão associados com as varáveis f e g
- Definido em hardware! Portanto a instrução
  - add $zero,$zero,$s0

não faz nada!

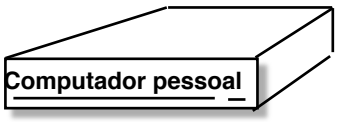
Constantes

- **Na bibliografia ("immediates")**
- Como aparecem frequentemente no código há instruções especiais para operar com elas…
- **"Add Immediate":**
  - **addi $s0,$s1,10 (em MIPS)**
  - **f = g + 10 (em C)**
  - **Onde os registos MIPS $s0,$s1 estão associados às variáveis f, g**
- A sintaxe é similar à da instrução add só que o último operando é uma **constante** e não um **registo.**

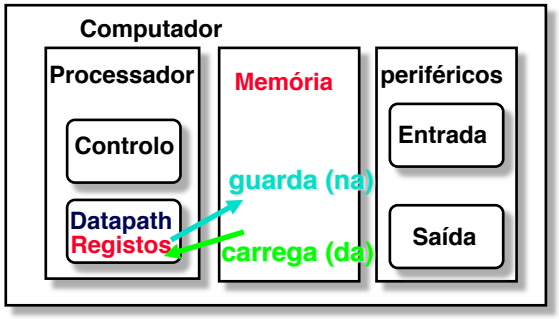
Constantes

- **Não existe "Subtract Immediate" no MIPS: Porquê?**
- **Limitar os tipos de operações que se podem executar ao mínimo absoluto..**
  - **Se uma operação pode ser decomposta noutra mais simples então não deve ser incluída!**
  - **addi …, -X = subi …, X => portanto não há subi**
- **addi $s0,$s1,-10 (em MIPS)**
  - **f = g - 10 (in C)**
  - **Onde os registos MIPS $s0,$s1 estão associados ás variáveis f, g**

Interacção com a memória



Os registos encontram-se no barramento de dados do processador; Se os operandos se encontram na memória é necessário transferi-los para os registos para executar as operações, e, no final transferi-los de volta para a memória.



Transferência de dados: Memória >> Registos 1/4

- Para transferir uma palavra da memória para os registos temos que indicar:
  - Registo: indicando o # ($0 - $31) ou o nome ($s0,…, $t0, …)
  - Endereço de memória: mais difícil…
    - Temos que imaginar a memória como um vector em que cada posição possui um endereço…
    - De outras vezes, vamos querer endereçar uma posição de memória a partir deste endereço (“base address” mais “offset”)
- Importante: “Load FROM memory”

Transferência de dados: Memória >> Registos 2/4

- Para especificar um endereço de memória a partir do qual copiar, são necessárias duas coisas:
  - Um registo contendo um apontador para a memória
  - Um deslocamento numérico em bytes ("offset")
- O endereço de memória desejado é a soma destes dois valores.
- Exemplo: 8($t0)
  - Especifica o endereço de memória apontado por $t0 mais 8 bytes.

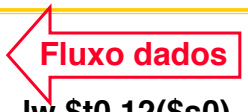
Transferência de dados: Memória >> Registos 3/4

- Sintaxe da instrução "Load":
  - 1 2,3(4)

  onde

  1) Nome da operação
  2) Registo que vai receber o valor
  3) O deslocamento ("offset") em bytes
  4) Registo contendo um apontador para a memória.
- Nome da instrução no MIPS:
  - lw (quer dizer "Load Word", portanto 32 bits - ou uma palavra - são carregados de cada vez)

Transferência de dados: Memória >> Registos 4/4



Exemplo: lw $t0,12($s0)

- Esta instrução pega no apontador em $s0, soma-lhe 12 bytes, e depois carrega o conteúdo da posição de memória apontado por esta soma no registo $t0

° Notas:

- $s0 chama-se o registo base ("base register")
- 12 é o deslocamento ("offset")
- O deslocamento é geralmente usado para aceder aos elementos de um "array" ou estrutura: o registo base aponta para o início do "array" ou estrutura

Transferência de dados: Registos >> Memória

° Os dados também têm de passar dos registos para a memória…

- A sintaxe do "Store" é idêntica à do "Load"

° Nome no MIPS :

- sw - quer dizer "Store Word", portanto 32 bits (ou uma palavra) passam do registo para a memória

° Exemplo: sw $t0,12($s0)

- Esta instrução pega no apontador em $s0, soma-lhe 12 bytes, e depois carrega o conteúdo do registo $t0 na posição de memória apontada por esta soma.

° Importante: "Store INTO memory"

Apontadores vs. Valores

- **Importante:**
  - Um registo pode conter um qualquer valor de 32 bits. Esse valor pode ser um inteiro, com ou sem sinal, um apontador (endereço de memória),etc.
- **Se escrever add $t2,$t1,$t0**
  então $t0 e $t1
  devem conter **NÚMEROS**
- **Se escrever lw $t2,0($t0)**
  Então $t0 deve conter um **ENDEREÇO**
- **Não confundir!**

Exercício

Memória

Endereço	Conteúdo
.....	.....
0000AAA0	00000003
0000AAA4	0000000A
0000AAA8	00000002
0000AAAC	00000001
.....	.....

Registos

Nome	Conteúdo
.....	.....
S0	0000AAA4
S1	0000AAA8
S2	00000002
S3	00000000
.....	.....

Qual o conteúdo de $S3 depois de:

```
lw $t0, 0($s0)
lw $t1, 4($s0)
add $t2, $t0, $t1
add $s3, $s3, $s2
add $s3, $s3, $t2
```

Qual o conteúdo da memória depois de:

```
lw $t0, 4($s1)
add $t1, $zero, $s2
addi $t1, $t1, 12
sub $t1, $t1, $t0
sw $t1, 0($s1)
```

Apontador

- http://en.wikipedia.org/wiki/MIPS_processor