INE5607 – Organização e Arquitetura de Computadores

Linguagem de Montagem e de Máquina

Aula 5: Introdução a instruções

Prof. Laércio Lima Pilla

laercio.pilla@ufsc.br

Sumário

- Abaixo do programa
- Linguagem de montagem
- Linguagens interpretadas
- Suporte para operações
- Suporte para operandos
- Representação de instruções
- Considerações finais

ABAIXO DO PROGRAMA

Abaixo do programa

- Uma aplicação típica pode possuir:
  - Milhares/milhões de **linhas de código**
  - Dependência de **bibliotecas** sofisticadas
- Hardware do computador executa apenas **instruções muito* simples**
- A tradução de uma **aplicação complexa** para instruções simples depende de diversas camadas de software

Abaixo do programa

- **Aplicativo/aplicação**
  - Escrito em linguagem de alto nível
- **Software de sistema**
- **Hardware**
  - Processador, memória e controladores de E/S

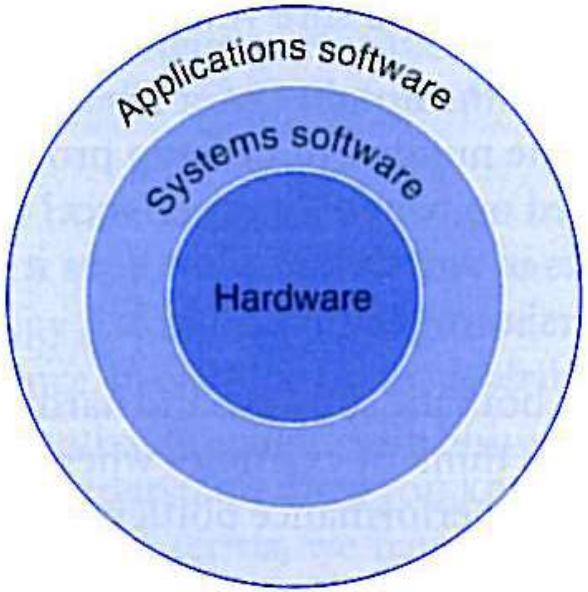

Figura 1.2 do livro *Computer Organization and Design 4th ed.*

Abaixo do programa

- **Software de sistema?**
  - **Sistema operacional**
    - Gerencia operações de entrada/saída
    - Aloca armazenamento e memória
    - Gerencia o compartilhamento do computador entre múltiplas aplicações
  - **Compilador**
    - Traduz código fonte em código de máquina

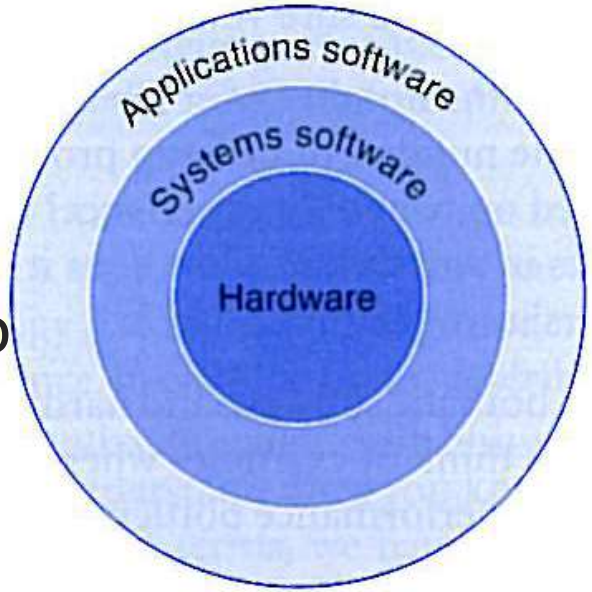

Figura 1.2 do livro *Computer Organization and Design 4th ed.*

Abaixo do programa

- Para conversar com o **hardware**, é preciso enviar sinais elétricos
  - Os mais simples são ***on*** e ***off***
  - Alfabeto do hardware tem 2 símbolos **0** e **1**
  - Hardware trabalha com **bits**

Abaixo do programa

- **Instruções** são passadas para o **hardware** através de **sequências de bits**.
  - Exemplo: Soma de dois números pode ser

    1000110010100000

  - Tanto as instruções quanto os dados são representados por números binários
  - **Linguagem de máquina**

Abaixo do programa

- **Linguagem de máquina**
  - A linguagem de máquina entendida por um processador é definida por sua **ISA** (***Instruction Set Architecture***)

Arquitetura do Conjunto de Instruções (ISA)	Implementações
MIPS 32	BRCM 5000, MIPS 74K
ARMv7-A	Cortex-A8, Cortex-A9
X86-32 (IA-32)	Pentium 4, Pentium M
IA-64	Itanium, Itanium2
X86-64	Core2 Duo, Core2 Quad, Opteron 2356, Atom

Abaixo do programa

- ***Instruction Set Architecture***
  - Dois processadores que entendem o mesmo conjunto de instruções (**compatíveis**) têm a mesma ISA
    - Processadores **compatíveis** podem executar os mesmos programas
  - Porém, dois computadores com a mesma ISA não tem necessariamente a mesma **implementação/organização de hardware**

LINGUAGEM DE MONTAGEM

Linguagem de montagem

- Utilização
  - **Quando tamanho do programa / velocidade é crítico**
    - Alguns sistemas embarcados por exemplo
  - Seções críticas de um programa
  - **Compilador não está disponível**
    - Sistemas antigos
    - Processadores dedicados

Linguagem de montagem

- **Vantagens**
  - Programador controla melhor os recursos de hardware
    - Compilador produz código mais uniforme mas não enxerga todas as otimizações possíveis
  - Exploração de instruções especializadas
    - Exemplo: instruções multimídia

"Also, via assembly blocks nested within the C code, ***we exploited the fyl2x instruction offered by the x87 instruction set, which calculates the base 2 log of the argument multiplied by a scalar****. By setting the scalar value to 1, a simple base 2-log is calculated, which is the position of the most significant bit set. This value can then be used to traverse the tree by directly accessing the child with the requested index."*

Linguagem de montagem

- **Desvantagens**
  - Código não portável
  - Programas mais longos?
  - Menor produtividade
  - Menor legibilidade

LINGUAGENS INTERPRETADAS

Linguagens interpretadas

- Principal exemplo:

- Compilação gera **bytecode** que pode ser interpretado por **máquinas virtuais** particulares para cada arquitetura

Category	Operation	Java bytecode	Size (bits)	MIPS instr.
Arithmetic	add	iadd	8	add
	subtract	isub	8	sub
	increment	iinc I8a I8b	8	addi
Data transfer	load local integer/address	iload I8/aload I8	16	lw
	load local integer/address	iload_/aload_{0,1,2,3}	8	lw
	store local integer/address	istore I8/astore I8	16	sw
	load integer/address from array	iaload/ aaload	8	lw
	store integer/address into array	iastore/aastore	8	sw

Linguagens interpretadas

- **Exemplo de bytecode**
  - Linguagem de alto nível: **while (save[i] == k) i++;**
  - Bytecode
    - i campo 1, k campo 2, Save campo 3

```
0. aload_3                  #coloca Save[] na pilha
1. iload_1                  #coloca i na pilha
2. ialod                    #carrega Save[i] na pilha
3. iload_2                  #carrega k na pilha
4. if_icompne, Exit         #se Save[i] != k, sai do laço
7. iinc, 1, 1               #i = i + 1
10. go to 0                 #volta para a instr. 0
```

Linguagens interpretadas

- Java
  - Execução tende a ser mais lenta por causa do **overhead** causado pelo **interpretador**
  - Trechos críticos
    - Compilação para a máquina específica

SUPORTE PARA OPERAÇÕES

Suporte para operações

- Linguagem de máquina
  - Arquitetura do conjunto de instruções (ISA)
  - Instruções: “palavras”
  - Conjunto de instruções: “vocabulário”
- ISA da disciplina: MIPS-32
  - *Microprocessor without Interlocked Pipeline Stages*
    - Hennessy: Stanford
  - Escolhida pela simplicidade

Suporte para operações

- Entendendo uma linguagem de montagem (***assembly***):

constantes

registradores

operações

```
addiu   $29, $29, -32
sw      $31, 20($29)
sw      $4,  32($29)
sw      $5,  36($29)
sw      $0,  24($29)
sw      $0,  28($29)
lw      $14, 28($29)
lw      $24, 24($29)
multu   $14, $14
addiu   $8,  $14, 1
slti    $1,  $8, 101
sw      $8,  28($29)
mflo    $15
addu    $25, $24, $15
bne     $1,  $0, -9
sw      $25, 24($29)
lui     $4,  4096
lw      $5,  24($29)
jal     1048812
addiu   $4,  $4, 1072
lw      $31, 20($29)
addiu   $29, $29, 32
jr      $31
move    $2,  $0
```

Suporte para operações

- Entendendo...

labels?

registradores

diretivas?

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra

        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Suporte para operações

- Alguns elementos da linguagem de montagem (**assembly**)
  - Mnemônicos de operações
    - Exemplos: **add**, **sub**, **jal**
  - Mnemônicos de registradores
    - Exemplos: **$s1**, **$t0**

Suporte para operações

- Alguns elementos da linguagem de montagem (**assembly**)
  - Mnemônicos de **endereços de memória**
    - Rótulos ou **Labels**
    - Ex.: **main: add $s1,$s2,$t0**
  - **Pseudoinstruções?**
    - Representadas como instruções mas não são nativas do processador
    - Pseudoinstrução → instrução nativa
    - **move $t0,$t1 → add $t0, $zero, $t1**

Suporte para operações

- Notação em assembly do MIPS
  - Exemplo: a = b + c
    - **add a, b, c**
    - a: destino; b, c: fontes
  - Exemplo: a = b + c + d + e
    - add a, b, c
    - add a, a, d
    - add a, a, e
    - **Por quê não "add a, b, c, d, e"?**

Suporte para operações

- Princípio de Projeto 1
  - **A simplicidade favorece a regularidade**
    - MIPS: arquitetura RISC
      - *Reduced Instruction Set Computer*
      - Qual é o contrário de RISC?
      - Número de operandos fixo (máximo = 3)
    - HW mais simples do que para números variáveis
      - Poucos formatos de instrução
    - Menos decisões a serem tomadas pelo compilador

Suporte para operações

- Notação em assembly do MIPS
  - Exemplo: **f = (g + h) – (i + j)**
    - O que um compilador faria?
    - add **t0**, g, h
    - add **t1**, i, j
    - sub f, t0, t1
    - t0, t1: operandos temporários
  - Onde armazenar valores temporários?
    - HW: **registradores**
      - Valores temporários e de variáveis

SUPORTE PARA OPERANDOS

Suporte para operandos

- Registradores e variáveis
  - Número ilimitado* de variáveis em software
  - Número limitado de registradores
    - Nem todas variáveis podem estar em registradores simultaneamente
  - Necessário alocar registradores para variáveis
    - Manual: programação em assembly :~
    - Automática: compilador :D

Suporte para operandos

- MIPS: 32 registradores
  - Por quê não mais?
- Princípio de Projeto 2
  - **Menor significa mais rápido**
    - Menor energia também
  - Registradores vs memória
    - Mem: Acesso mais lento, bilhões de elementos
    - Reg: Acesso mais rápido, dezenas de elementos
      - Boa alocação leva a melhor desempenho e consumo

Suporte para operandos

- Registradores MIPS
  - A: argumentos
  - T: temporários
  - S: temporários salvos (variáveis)

Nº	Nome	Nº	Nome	Nº	Nome	Nº	Nome
0	$zero	8	$t0	16	$s0	24	$t8
1	$at	9	$t1	17	$s1	25	$t9
2	$v0	10	$t2	18	$s2	26	$k0
3	$v1	11	$t3	19	$s3	27	$k1
4	$a0	12	$t4	20	$s4	28	$gp
5	$a1	13	$t5	21	$s5	29	$sp
6	$a2	14	$t6	22	$s6	30	$fp
7	$a3	15	$t7	23	$s7	31	$ra

Suporte para operandos

- Exemplo: **f = (g + h) – (i + j)**
  - O que um compilador faria?
  - $s0=f, $s1=g, $s2=h, $s3=i, $s4=j
  - **add $t0, $s1, $s2**
  - **add $t1, $s3, $s4**
  - **sub $s0, $t0, $t1**
    - Assembly MIPS de verdade

Suporte para operandos

- E os operandos que não são variáveis?
  - Exemplo: x = A[10] + A[11]
  - Estruturas de dados não cabem nos registradores
    - São mantidas em memória!

Suporte para operandos

- Memória
  - Array unidimensional
    - Composta de palavras
      - Palavras armazenam dados
    - Acessadas pelos endereços
  - x = A[10] + A[11] -> **add $s0, A[10], A[11]?**

Endereço	Dados
...	...
3	100
2	10
1	101
0	1

Suporte para operandos

- x = A[10] + A[11] -> **add $s0, A[10], A[11]?**
  - Resposta: não.
- Instruções de transferência de dados
  - Load: dado da memória para registrador
  - Store: dado do registrador para memória
  - Nenhuma outra instrução acessa memória
    - Somente operandos em registradores
    - Estruturas de dados precisam ser carregadas

Suporte para operandos

- Exemplo: A[2] = A[1] + i
  - Endereço base de A[] em $s5
  - Valor de i em $s0
  - **lw**: load word
    - lw destino, fonte
  - **sw**: store word
    - sw fonte, destino
  - $s0 = registrador
  - ($s0) = valor no registrador
    - De onde vêm os valores 4 e 8?

```
lw $t0, 4($s5)
add $t1, $t0, $s0
sw $t1, 8($s5)
```

Suporte para operandos

- De onde vem os valores 4 e 8?
  - Endereçamento de palavra
  - Palavra = 4 bytes no MIPS 32
    - 4 bytes = 32 bits
  - char = 1 byte, half = 2 bytes, word = 4 bytes

Endereço	Dados
...	...
12	100
8	10
4	101
0	1

Suporte para operandos

- Operandos constantes
  - **Como fica i = i + 1?**
  - Solução 1: memória para constantes
    - lw $t0, EndConstante1($s1)
    - add $s3, $s3, $t0
      - Uma leitura para cada operação com constantes?
  - Solução 2: constante interna à instrução
    - addi $s3, $s3, 1
  - Princípio de Projeto 3
    - **Agilize os casos mais comuns**

Suporte para operandos

- Como manter as variáveis?
  - Uso frequente?
    - *Register allocation*: variáveis em registradores
      - Acesso rápido
  - Uso raro?
    - *Register spilling*: variáveis em memória
      - Lidas e escritas quando necessário

REPRESENTAÇÃO DE INSTRUÇÕES

Representação de instruções

- MIPS-32 usa instruções de 32 bits
  - Tamanho fixo (RISC)
- Codificação como número (binário)
  - Operação
  - Endereço dos operandos
  - Valores constantes
- Representação de uma instrução
  - Tipo R, I, J

Representação de instruções

- Formato de instrução tipo R

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - Exemplo: **add $t0, $s1, $s2**
    - Decimal

0	17	18	8	0	32

    - Binário

000000	10001	10010	01000	00000	100000

Representação de instruções

- Formato de instrução tipo I

op	rs	rt	constante/endereço
6 bits	5 bits	5 bits	16 bits

  - Exemplo: **addi $t0, $s1, 5**

8	17	8	5

  - Exemplo: **lw $t1, 300($s3)**

35	19	9	300

Representação de instruções

- **Por quê não constantes de 32 bits?**
- Principio de Projeto 4
  - **Um bom projeto exige bons compromissos**

Representação de instruções

- Exemplo de tradução
  - Alto nível: A[10] = h + A[10]
  - Linguagem de montagem
    - lw $t0, 40($s1)
    - add $t0, $t0, $s2
    - sw $t0, 40($s1)
  - Linguagem de máquina

100011	01001	01000	0000 0000 0010 1000		
000000	01000	10010	01000	00000	100000
101011	01001	01000	0000 0000 0010 1000		

Representação de instruções

- Conceito de programa armazenado
  - Computadores baseado em dois princípios
    - Instruções são representadas como números
    - Programas são armazenados em memória
      - Para serem lidos ou escritos como números
    - Von Neumann? Alguém?
    - Possibilidade de trocar o programa sendo executado
      - Computadores de propósito geral

Representação de instruções

- Conceito de programa armazenado

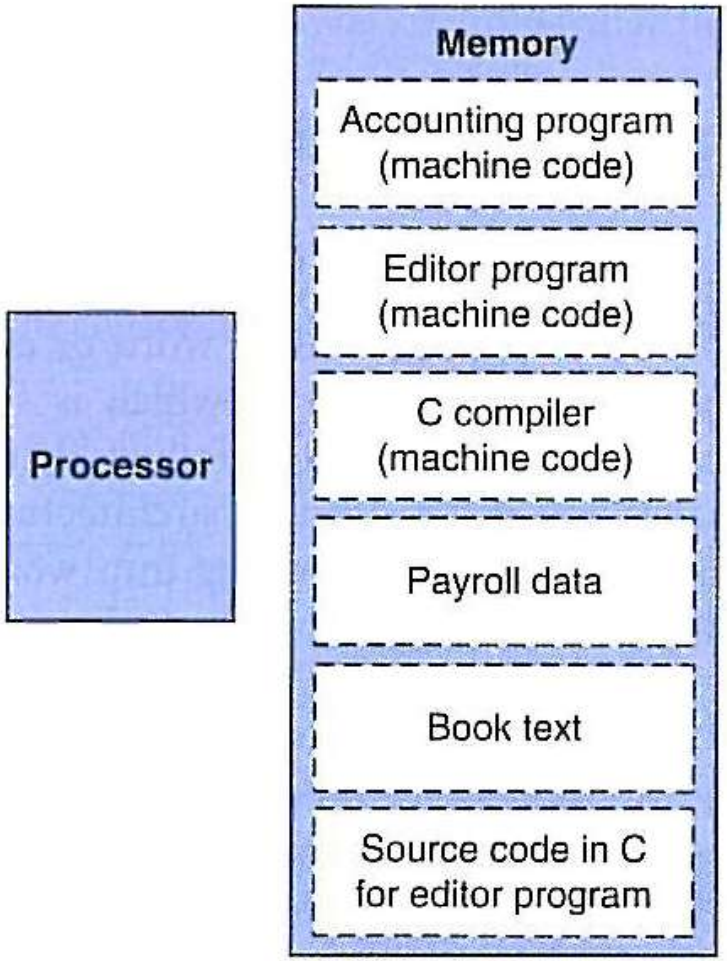

Figura 2.7 do livro *Computer Organization and Design 4th ed.*

CONSIDERAÇÕES FINAIS

Considerações finais

- Como são apresentas as instruções
- Formatos de instruções (R, I)
- Operandos
  - Registradores, memória, imediatos
- Operações
  - add, addi, lw, sw
- Princípios de Projeto (quatro)

Considerações finais

- Próxima aula
  - Tradução de instruções básicas

INE5607 – Organização e Arquitetura de Computadores

Linguagem de Montagem e de Máquina

Aula 5: Introdução a instruções

Prof. Laércio Lima Pilla

laercio.pilla@ufsc.br