Aula 2: Instruções - A linguagem de máquina - Parte I

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

OCS (TEORIA) - SETOR DE INFORMÁTICA

Instruções - A linguagem de máquina

- Operações / Operandos do Hardware do Computador
- Representando Instruções no Computador
- Operações Lógicas
- Operações para Tomada de Decisões

Introdução

- Instruções: "Palavras"da linguagem de um computador
- Conjunto de instruções: vocabulário dos comandos entendidos por uma determinada arquitetura
- Veremos conjunto de instruções de um computador real: MIPS
    - Linguagens de computador são muito semelhantes
    - Aprendendo uma, fácil entender as outras

Operações do Hardware do Computador

- Todo computador precisa ser capaz de realizar aritmética
- Todas as instruções possuem três operandos
- A ordem do operando é fixa (destino primeiro)
- Notação assembly do MIPS para soma
  add a, b, c #a = b + c

- Por que não quatro parâmetros?
- **Princípio de projeto 1: a simplicidade favorece a regularidade**
- Hardware com número variável de operandos é mais complexo do que hardware para número fixo
- Como operar com mais parâmetros?

```
add a, b, c
add a, a, d // a = b + c + d;
```

Operações do Hardware do Computador

- Notação assembly do MIPS para subtração
  sub d, a, e # d = a - e
- Compilando atribuição C complexa no MIPS: *f = (g+h)-(i+j);*
- Uso de variável temporária
  add t0, g, h
  add t1, i, j
  sub f, t0, t1

Operandos do Hardware do Computador

- Ao contrário dos programas nas linguagens de alto nível, operandos das instruções aritméticas são restritos
  - Registradores
  - Tamanho do registrador MIPS: 32 bits $\rightarrow$ Palavra (word)?
  - Quantidade limitada $\rightarrow$ MIPS: 32 registradores
- **Princípio de projeto 2: menor significa mais rápido**

Operandos do Hardware do Computador

- Convenção MIPS para representar registradores
  - Sinal de cifrão($) seguido por dois caracteres
  - Veremos motivos na próxima aula
  - Por enquanto:
    - $s0, $s1, ... para registradores que correspondem às variáveis dos programas
    - $t0, $t1, ... para registradores temporários

Operações do Hardware do Computador

- Tarefa do compilador associar variáveis do programa aos registradores
- Considere exemplo anterior f = (g+h)-(i+j);
  - F, g, h, i e j associados a $s0, $s1, $s2, $s3 e $s4

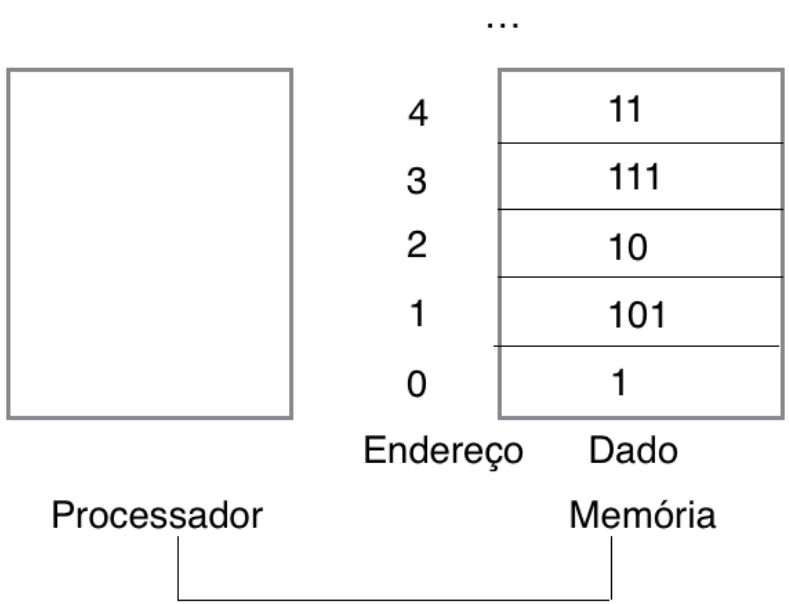
```
add $t0, $s1, $s2 # $t0 contém g + h
add $t1, $s3, $s4 # $t1 contém i + j
sub $s0, $t0, $t1 # f recebe $t0 - $t1
```

Operações do Hardware do Computador

- E quanto aos programas com muitas variáveis? E estruturas de dados complexas?
- Processador só pode conter pequena quantidade de dados nos registradores
  - Compilador tenta manter variáveis mais utilizadas nos registradores (acesso mais rápido)
  - Demais dados mantidos em memória (acesso mais lento)
  - Processo chamado de spilling registers
- Em MIPS, operações aritméticas só ocorrem com registradores
  - Necessária instruções de transferência de dados entre memória e registradores
- Para acessar palavra na memória, precisamos de localização: endereço

Operações do Hardware do Computador

- Memória vista como seqüência grande e unidimensional, com endereço atuando como índice para esse array
  - Ex: Endereço da terceira posição de memória = 2, conteúdo = 10

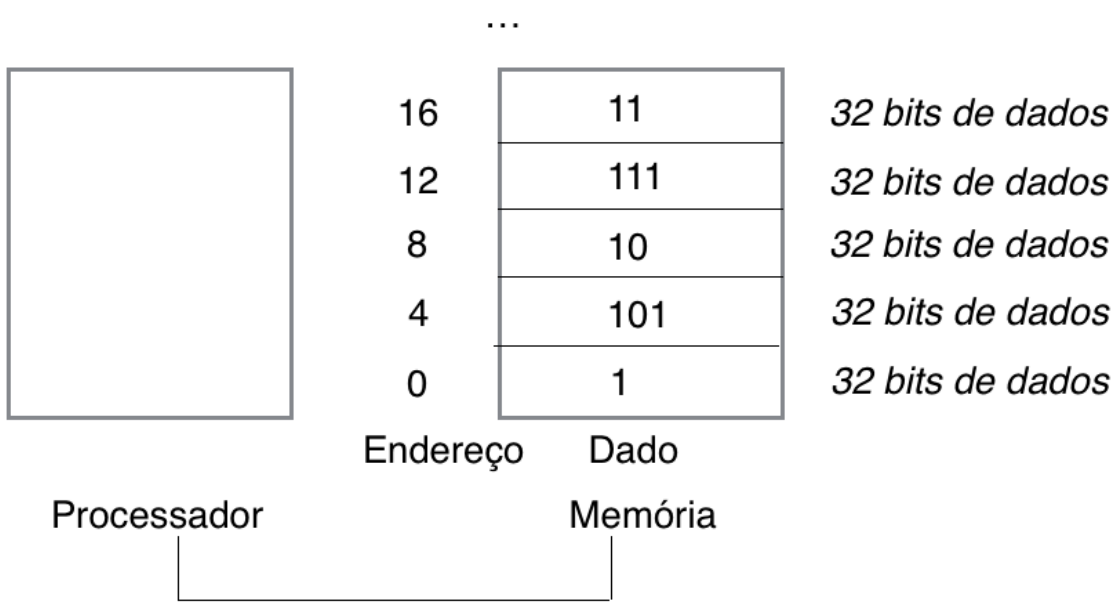


Operações do Hardware do Computador

- **Load**: instrução que copia dados da memória para registrador (lw: load word)
- Três parâmetros
  - Registrador destino: local onde os dados serão copiados
  - Deslocamento (ou offset): distância, a partir do endereço inicial da memória, onde dado se encontra
  - Registrador base: endereço inicial da memória
- Ex: g = h + A[8]
  g em $s1, h em $s2 e endereço inicial de A em $s3
  lw $t0, 8($s3)
  add $s1, $s2, $t0

Operações do Hardware do Computador

- No MIPS, uma word tem 32 bits (4 bytes)
  - 1 byte = 8 bits
- Endereços reais diferentes dos endereços da figura anterior
  - Endereço em bytes da 3 palavra é 8



Operações do Hardware do Computador

- Words em MIPS precisam começar em endereços que sejam múltiplos de 4
  - Requisito denominado restrição de alinhamento
  - Deslocamento apropriado no exemplo anterior: 4 x 8 = 32
- Little endian x Big endian
  - Endereço da word definida como o endereço do byte mais à esquerda (big) ou mais à direita (little)

- **Store**: copia dados de um registrador para a memória (sw: store word)
- Formato semelhante ao load:
  - Registrador a ser armazenado
  - Deslocamento
  - Registrador base
- Ex: A[12] = h + A[8]
  lw $t0, 32 ($s3) # $t0 = Memória [$s3+32]
  add $t0, $s2, $t0 # $t0 = $t0 + $s2
  sw $t0, 48($s3) # Memória [$s3+48] = $t0

Operações do Hardware do Computador

- Quantos endereços temos disponíveis no MIPS?
  - 2^{30} words com endereços de byte 0, 4, 8, ...
  - 4 GB de memória (2^{30} words x 4 bytes por word)?
- Usando instruções vistas até aqui, se quiséssemos usar constante em uma operação, necessário armazená-la em memória
  - Uso de operações com constantes muito comum
  - Custo alto no acesso a memória
  - Alternativa: instruções imediatas

- **Add imediato (addi)**: constante no lugar de um operando
  addi $s3, $s3, 4 # $s3 = $s3 + 4
- **Princípio de projeto 3: agilize os casos mais comuns**
- Operandos constantes ocorrem com freqüência $\rightarrow$ Mais rápido incluí-las dentro das instruções aritméticas do que lê-las da memória

Representando Instruções no Computador

- Humanos aprenderam a pensar na base 10
  - Números podem ser representados em qualquer base
- Números mantidos no hardware como série de sinais eletrônicos altos e baixos
  - Base 2 (alto/baixo, ligado/desligado, V/F, 0/1)
  - Números binários, compostos por dígitos binários (binary digit)

Representando Instruções no Computador

- Instruções também mantidas como série de sinais eletrônicos altos e baixos
  - Podem ser representados como número
  - Na verdade cada parte da instrução pode ser considerada como um número individual
    - Colocação lado a lado desses números forma a instrução
- Registradores fazem parte de quase todas as instruções
  - Convenção para mapear nome de registradores em números
    - MIPS: registradores $s0 a s7$: registradores 16 a 23
    - Registradores $t0 a t7$ mapeados nos registradores de 8 a 15

Representando Instruções no Computador

- Como é representada add $t0, $s1, $s2?
- Representação binária (formato da instrução):

000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct

  - Cada um dos segmentos chamado de campo
  - Primeiro (op: opcode) e último (funct: função) campos indicam operação de soma
  - Segundo (rs) e terceiro (rt) campos indicam operandos
  - Quarto campo (rd) indica registrador destino
  - Quinto campo (shamt: shift amount) não usado nessa instrução

Representando Instruções no Computador

- Versão numérica das instruções chamada de linguagem de máquina $\rightarrow$ Seqüência de instruções chamada de código de máquina
- Instruções MIPS: 32 bits
- E se instrução precisa de campos maiores que os mostrados?
  - Conflito entre manter todas as instruções com o mesmo tamanho e o desejo de ter um formato de instrução único.

Representando Instruções no Computador

- **Princípio de projeto 4: um bom projeto exige bons compromissos**
- Compromisso dos projetistas do MIPS: todas as instruções com o mesmo tamanho
  - Tipos diferentes de instruções
  - Formato anterior: **tipo-R** (operações envolvendo apenas registradores)?
  - Segundo tipo de instruções: **tipo-I** (imediato)?: Utilizada por instruções imediatas e de transferência de dados

Representando Instruções no Computador

- Ex: lw $t0, 32($s3)

35	19	8	32
op	rs	rt	Constante ou endereço de 16 bits

- Endereço de 16 bits significa que instrução pode carregar qualquer palavra dentro de uma região distante 32.768 bytes do endereço contido no registrador base rs
  - $\pm 2^{15}$
  - De modo semelhante, no modo imediato limitado a constantes no limite $\pm 2^{15}$.
- Formato de rt mudou para essa instrução:
  - Em um load, significa registrador que recebe dado lido da memória.

Representando Instruções no Computador

- Uso de vários formatos complica o hardware
- Podemos reduzir complexidade mantendo formatos semelhantes
- Três primeiros campos nos formatos tipo R e tipo I tem mesmo tamanho e nomes
  - Quarto campo do tipo I igual ao tamanho dos três últimos campo do tipo R
- Formatos diferenciados pelo valor no primeiro campo
  - Cada formato recebe conjunto distinto de valores no primeiro campo
  - Assim hardware sabe como tratar última metade da instrução

Representando Instruções no Computador

Instrução	Formato	op	rs	rt	rd	Shamt	Funct	Endereço
add	R	0	reg	reg	reg	0	32	n.a.
sub	R	0	reg	reg	reg	0	34	n.a.
addi	I	8	reg	reg	n.a.	n.a.	n.a.	constante
lw	I	35	reg	reg	n.a.	n.a.	n.a.	endereço
sw	I	43	reg	reg	n.a.	n.a.	n.a.	endereço

reg: registrador entre 0 e 31

endereço: endereço de 16 bits

n.a.: não se aplica

Representando Instruções no Computador

- Exemplo: Considere a seguinte instrução de atribuição:
  A[300] = h + A[300];
- Sabendo-se que $t1 contém o endereço base do array A, e $s2 contém h, e que a instrução é compilada para:
  lw $t0, 1200($t1) #registrador $t0 recebe A[300]
  add $t0,$s2,$t0 #registrador $t0 recebe h+A[300]
  sw $t0, 1200 ($t1) # h+A[300] armazenado em A[300]
- Qual o código em linguagem de máquina MIPS para essas três instruções?

- Resposta: Por conveniência, vamos primeiro representar a instrução em código de máquina usando números decimais:

op	rs	rt	rd	endereço/shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

- Resposta: Agora sim, em binário!

op	rs	rt	rd	endereço/shamt	funct
100011	01001	01000	0000010010110000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000010010110000		

Operações Lógicas

- Útil operar sobre campos de bits dentro de uma word ou mesmo bits individuais: Operações lógicas
- Deslocamento: movem bits de uma word para a esquerda (sll) ou para direita (srl)
  - Bits vazios preenchidos com 0's
  - 0000 0000 0000 0000 0000 0000 0000 0000 1001 = 9_{dec}
  - Deslocando 4 bits à esquerda
  - 0000 0000 0000 0000 0000 0000 0000 1001 0000 = 144_{dec}

Operações Lógicas

- Supondo que número estivesse em $s0, e resultado deva ser armazenado em $t2:
  - ssl $t2, $s0, 4 # $t2 = $s0 << 4 bits
  - Campo shamt (shift amount) usado nas instruções de deslocamento
  - Codificação: 0 em op e funct, rd contém destino, rt contém origem e shamt o deslocamento. Registrador rs não usado (0)?
  - Deslocamento usado para multiplicar/dividir número por potências de dois

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

Operações Lógicas

- Operação **and** compara operandos bit a bit. Resultado do bit igual a 1 se bits equivalentes dos operandos forem simultaneamente iguais a 1
  - Registrador $t2: 0000 0000 0000 0000 0000 1101 0000 0000
  - Registrador $t1: 0000 0000 0000 0000 0011 1100 0000 0000
  - and $t0, $t1, $t2 # $t0 igual a $t1& $t2
  - $t0: 0000 0000 0000 0000 0000 1100 0000 0000
  - And usado para aplicar um padrão de bits a um conjunto de bits
    - Forçar 0s onde houver um 0 no padrão de bits
    - Padrão chamado de máscara

Operações Lógicas

- Operação **or** compara operandos bit a bit. Resultado do bit igual a 1 se um dos bits equivalentes dos operandos for igual a 1
  - Registrador $t2: 0000 0000 0000 0000 0000 1101 0000 0000
  - Registrador $t1: 0000 0000 0000 0000 0011 1100 0000 0000
  - or $t0, $t1, $t2 # $t0 igual a $t1 | $t2
  - $t0: 0000 0000 0000 0000 0011 1101 0000 0000
  - Or também usado para aplicar padrão de bits a um conjunto de bits
    - Forçar 1s onde houver um 1 no padrão de bits

Operações Lógicas

- Operação **not** inverte bits de um operando.
  - Mas no lugar do not, projetistas colocaram operação nor (not or)
  - Equivalente a not quando um dos operandos for igual a zero
  - Registrador $t1: 0000 0000 0000 0000 0011 1100 0000 0000
  - Registrador $t3 igual à zero
  - nor $t0, $t1, $t3 # $t0 igual a $\sim$ ($t1 | $t3)
  - $t0: 1111 1111 1111 1111 1100 0011 1111 1111
- Versões imediatas de and (andi) e or (ori) também disponibilizadas
  - Operações and e or bit a bit com constantes

Instruções para Tomada de Decisões

- Instruções de tomada de decisão alteram o fluxo de instruções
  - Muda a próxima instrução a ser executada
- Instruções de desvio condicional do MIPS
  - **beq** registrador1, registrador2, L1
    - Significa ir até instrução rotulada por L1 se conteúdo dos registradores for igual
    - branch if equal (desviar se for igual)?
  - **bne** registrador1, registrador2, L1
    - Significa ir até instrução rotulada por L1 se conteúdo dos registradores não for igual
    - branch if not equal (desviar se não for igual)?

Instruções para Tomada de Decisões

- Por quê não blt, bge, etc?
- Hardware para $<$, $\geq$, ..., mais lento que $=$, $\neq$
- Combinar branch envolve mais trabalho por instrução, o que requer clock mais lento.
- $\rightarrow$ Todas as instruções seriam penalizadas.
- **beq** e **bne** são o caso comum.
- Esse é um bom exemplo do compromisso com os princípios de projeto!

Instruções para Tomada de Decisões

- Exemplo: Variável i em $s0, j em $s1 e h em $s3
  if (i==j) h = i + j;
- Pode ser:

```
    bne $s0, $s1, Label
    add $s3, $s0, $s1
Label: ....
```

Instruções para Tomada de Decisões

- Instrução de desvio incondicional do MIPS **j**: Desvia para instruções que segue rótulo independente de condição
  j label
- Exemplo:

```
1  if (i!=j)          // beq $s4, $s5, Lab1
2        h=i+j;       //  add $s3, $s4, $s5
3  else               // j Lab2
4        h=i-j;       //  Lab1:  sub $s3, $s4, $s5
5                     //  Lab2:  ...
```

Instruções para Tomada de Decisões

- Como um loop while é representado em assembly MIPS?
- Exercício para a próxima aula: Suponha que i e k correspondam aos registradores $s3 e $s5, e a base do array save esteja em $s6. Qual o código assembly MIPS correspondente a esse segmento de código C?

```
1 while (save[i] == k)
2       i += 1;
```

Instruções para Tomada de Decisões

- Sequências de instruções que terminam em um desvio são chamados de bloco básico
  - Não possuem destinos de desvio ou rótulos de desvio
  - Exceto, possivelmente, no início
- Testes de igualdade/desigualdade não resolvem todos os problemas
  - E se decisão baseada em uma variável ser ou não menor do que outra?

Instruções para Tomada de Decisões

- Instrução **slt** (set on less than: atribui se menor que)?
  slt $t0, $s3, $s4 # $t0 = 1 se $s3 < $s4
  - Valor 1 atribuído ao registrador $t0 se o valor de $s3 for menor que $s4, caso contrário atribuído valor 0
- Operadores constantes populares em atribuições
  slti $t0, $s2, 10 # $t0 = 1 se $s2 < 10

Próxima aula

Instruções - A linguagem de máquina - Parte II