Arquitetura e Organização de Computadores I

Conjuntos de Instruções

Instruction Set

- O repertório de instruções de um computador
- Computadores diferentes têm conjuntos de instruções diferentes
  - Mas com muitos aspectos em comum
- Os primeiros computadores tinham conjuntos de instruções muito simples
  - Implementação Simplificada
- Muitos computadores modernos também têm conjuntos de instruções simples

The MIPS Instruction Set

- Usado como exemplo em todo o curso
- Stanford MIPS comercializado pela MIPS Technologies (www.mips.com)
- Grande parte do mercado principal incorporado
  - Aplicações em eletrônicos de consumo, equipamentos de rede / armazenamento, câmeras, impressoras, ...
- Típico de muitos ISAs modernos

Arquitetura: MIPS

Atributos visíveis ao programador:

- Tamanho da palavra;
- Número de Operandos;
- Registradores visíveis;
- Endereçamento de Operandos;
- Tipos de Instruções;
- O conjunto de instruções (ISA);

Palavra e número de operandos: MIPS 32bits

- Tamanho da palavra: 32 *bits*

- 3 operandos por instrução

  sub $s0, $t0, $t1

  $s0 = $t0 - $t1

Registradores visíveis: MIPS

32 registradores genéricos de 32 *bits*

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

Apesar de genéricos, o registrador $zero não pode ser escrito, e alguns registos ($at, $k0, $k1, $gp, $sp, $fp, $ra) têm funções específicas.

Conjunto de Instruções

- Compreender a linguagem do hardware é um ponto chave para entender a interface de hardware / software;

- Um programa (no exemplo, C) é compilado em um executável que é composto de instruções de máquina - este executável também deve rodar em máquinas futuras - por exemplo, cada processador Intel lê as mesmas instruções x86, mas cada processador lida com as instruções de forma diferente;

- Programas em Java são convertidos em bytecode portátil que é convertido em instruções de máquina durante a execução (compilação just-in-time);

Instruções

- **Instruções: palavras / vocabulário: conjunto de instruções**
- **Programa armazenado: Dados e instruções de vários tipos podem ser armazenados como números (binários) na memória**
- **Instruções estão em Linguagem de Máquina (*assembly*)**
  - Muito Mais primitiva que linguagem de alto-nível
    - não há controles de fluxo sofisticados (for, while, etc...)
  - Muito restritivas
    - Instruções aritméticas do MIPS (visto mais adiante)
- **Nós iremos trabalhar com o MIPS *Instruction Set Architecture* (ISA)**
  - **Similar a outras arquiteturas desenvolvidas desde os 1980's**
  - **Usada pela NEC, Nintendo, Silicon Graphics, Sony**
  - **Foco nos princípios e não em detalhes específicos de ISA's**

Uma Instrução MIPS simples

- C code:

  a = b + c;

- Assembly code: (human-friendly machine instructions)

  add a, b, c

- Machine code: (hardware-friendly machine instructions)

  00000010001100100100000000100000

- Traduza o seguinte código em C para Assembly:

  a = b + c + d + e;

Exemplo

- Código C:

```
a = b + c + d + e;
```

- Pode ser traduzido para:

```
add  a, b, c          add  a, b, c
add  a, a, d    ou    add  f, d, e
add  a, a, e          add  a, a, f
```

- Observações:
  - As instruções são simples: número fixo de operandos (ao contrário de C);
  - Uma única linha de código C pode ser convertida em várias linhas de código assembly;
  - Algumas sequências são melhores do que outras ... a segunda sequência precisa do uso de um recurso a mais (f);

Aritmética Básica

Aritmética MIPS

- Todas as instruções possuem três operandos;
- A ordem do operando é fixa (destino primeiro);

Exemplo:

Código C:	a = b + c
Código MIPS:	add a, b, c

- *Design Principle:* Simplicity favours regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cos

Aritmética MIPS

- É claro que isso complica algumas coisas em outros sentidos:

  Código C: a = b + c + d;

  Código MIPS:
  ```
  add a, b, c
  add a, a, d
  ```

Exemplo de Subtração

- C code:

$$f = (g + h) – (i + j);$$

- Qual o código assembler traduzido deste código C?

Exemplo de Subtração

- C code

  f = (g + h) – (i + j);

- É traduzido em:

```
add  t0, g, h          add  f, g, h
add  t1,  i, j    or    sub   f, f, i
sub  f,   t0, t1       sub   f, f, j
```

Até agora:

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 – $s3`	Three operands; data in registers

Operandos

- Em C, cada "variável" é um local na memória;
- Em hardware, cada acesso à memória é caro - se a variável A é acessado várias vezes, será de grande benefício traze-la para á cache do processador e operar sobre este (registradores);
- Para simplificar as instruções, é necessário que cada instrução (add, sub) só funcione sobre os registradores;
- Nota: o número de operandos (variáveis) em um programa C é muito grande, o número de operandos em montagem é fixo;

Registradores

Register Operands

- Instruções aritméticas usam operandos de registradores
- MIPS usa uma tabela 32 × 32-bit
  - Use para dados acessados frequentemente
  - Numerados 0 a 31
  - Dados de 32 bits chamados de "palavra“
- Assembler names
  - $t0, $t1, …, $t9 for temporary values
  - $s0, $s1, …, $s7 for saved variables
- *Princípio de Design 2: Menor é mais rápido*
  - *Mas nem sempre: memória principal: milhões de locais*

Register Operand Example

- C code:

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

  - **f, …, j in $s0, …, $s4**

- Compiled MIPS code:

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

Operadores Imediatos

Operadores Imediatos

- Uma instrução pode requerer uma constante como entrada;
- Uma instrução imediata usa uma constante como uma de suas entradas (ao invés de um registrador);

- Seja **a** uma variável (contador de um programa, por ex.) associada ao registrador $s1.
  - a = a +1;
- Em MIPS:
  - **addi $s1, $s1, 1**

Operadores Imediatos

- Dados constantes especificados em uma instrução

  addi $ s3, $ s3, 4

- Não subtrair instrução imediata
  - Basta usar uma constante negativa

  addi $ s2, $ s1, -1

- Princípio de Design 3: Torne o caso comum rápido
  - Pequenas constantes são comuns
  - Operando imediato evita uma instrução de carga

The Constant Zero

- O registro MIPS 0 ($ zero) é a constante 0
  - Não pode ser sobrescrito

- Útil para operações comuns
  - Por exemplo, mova-se entre registros

```
add $t2, $s1, $zero
```

Até agora, para o ISA do MIPS...

MIPS operands

Name	Example	Comments
32 registers	`$s0, $s1, . . . , $t0, $t1, . . .`	Fast locations for data. In MIPS, data must be in registers to perform arithmetic.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 – $s3`	Three operands; data in registers
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2 + 100`	Used to add constants

Na visão do computador

Representing Instructions

- Instructions are encoded in binary
  - Called machine code
- MIPS instructions
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
  - Regularity!
- Register numbers
  - $t0 – $t7 are reg's 8 – 15
  - $t8 – $t9 are reg's 24 – 25
  - $s0 – $s7 are reg's 16 – 23

As instruções, na visão do computador

- Instruções, como registradores e words de dados, também possuem 32 bits de tamanho:
  - Exemplo: add $t0, $s1, $s2
  - Registradores podem ser representados por números: $t0=8, $s1=17, $s2=18

R-format Example

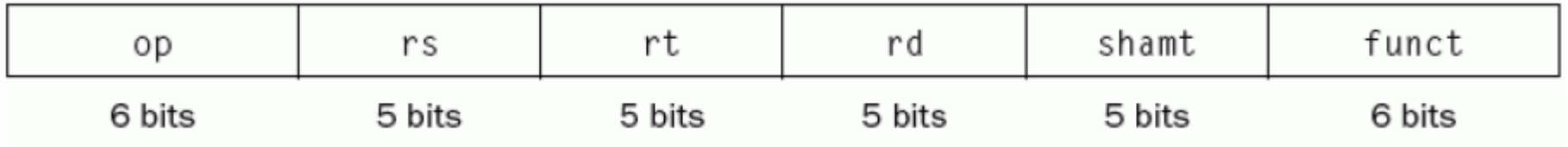
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

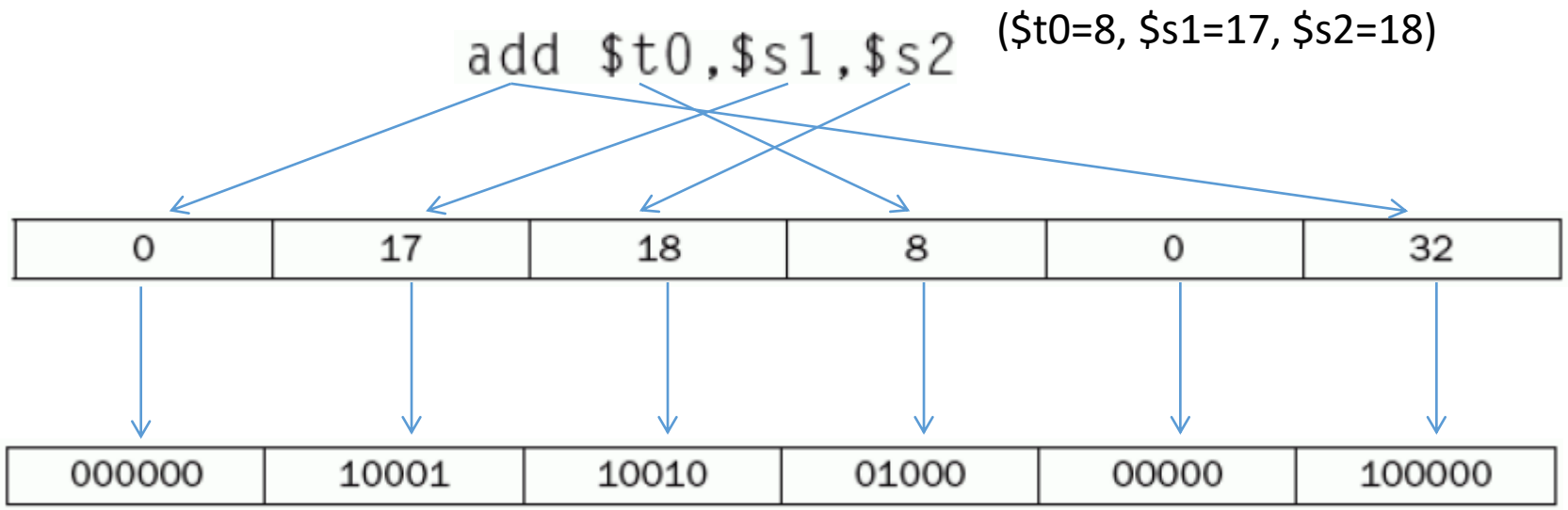
$$00000010001100100100000000100000_2 = 02324020_{16}$$

E como é isso?



- op (opcode): operação básica da instrução;
- rs (fisrt register source): primeiro registrador de origem;
- rt (second register sourse): segundo registrador de origem;
- rd (register destination): registrador de destino;
- shamt (shift amount): quantidade de deslocamento;
- funct (function): especifíca uma variante da operação;

Exemplo


add $t0,$s1,$s2
($t0=8, $s1=17, $s2=18)
0
17
18
8
0
32
000000
10001
10010
01000
00000
100000

Mas existem mais formatos

Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

- **Principio de projeto 4: um bom projeto exige bons compromissos:**
  - **"Um compromisso no MIPS é manter todas as instruções com o mesmo tamanho, exigindo assim diferentes tipos de formatos para diferentes tipos de instruções."**

Sintaxe MIPS

Sintaxe do MIPS

- Sintaxe do assembler:
  - Comentários de linha iniciam-se com #.
  - Identificadores são sequencias de **caracteres alfanuméricos**, **_** e **.**, e não se iniciam com números.
  - Rótulos (Labels) são colocados no começo de uma linha e seguidos de **:**
  - Números estão na base decimal por padrão; se precedidos por *0x* são interpretados como hexadecimais.
  - Strings são envolvidos com “ ”

Diretrizes

- Utilizadas pelo programador para instruir o assembler como traduzir um programa.
- Não produz instruções de máquina.
- Principais diretivas:
  - **.asciiz** armazena caracteres de um *string* na memória e finaliza-o com o caracter *null*;
  - .**ascii** armazena caracteres de um *string* na memória, mas não finaliza-o com *null*;
  - **.data <end.>** armazena os itens na seqüência no segmento de dados. Se <end.> for fornecido, os dados são armazenados a partir do endereço fornecido;
  - **.globl** *rótulo* declara *rótulo* como global, podendo ser acessado de outros arquivos;
  - **.text** armazena os itens na seqüência no segmento de textos do usuário. Itens devem ser instruções

Template

```
        .data       # variable declarations follow this line
                    # ...

        .text       # instructions follow this line

main:               # indicates start of code (first instruction to execute)
                    # ...
```

Declaração de dados

- Formato:
  - name: storage_type value(s)
    - Cria espaço para o armazenamento da variável do tipo com determinado valor e o nome especificados;
    - valor(es) geralmente dão valores iniciais (s); o tipo .space reserva um espaço em memória;
  - Tipos:
    - var1: .word 3
      - Cria uma palavra correspondente a um integer de valor inicial 3;
    - array1: .byte 'a','b‘
      - Cria um array de caracteres de duas posições com os valores ‘a’,’b’;
    - array2: .space 40
      - Aloca 40 bytes consecutivos com valores não inicializados;

Os tipos

- **.ascii *str***
  - Armazena str na memória, mas sem um terminador nulo;
- **.asciiz *str***
  - Armazena str na memória, mas com um terminador nulo;
- **.byte *b1, ..., bn***
  - Armazena n bytes de forma contígua na memória;

Mais tipos

- **.halfword *h1, ..., hn***
  - Armazena halwords(16 bits) de forma contígua na memória;
- **.word *w1, ..., wn***
  - Armazena palavras (32 bits) de forma contígua na memória;
- **.space *numBytes***
  - Reserva **numBytes** na memória.

Acessando os dados

- la:
  - A pseudo instrução copia o endereço do dado declarado para um registrador;
- Instruções reais (lw, lh, lb):
  - Copia o conteúdo da memória para o registrador;

- Exemplo:
  - .data
    - arr: .space 100
  - .text
    - la $t0, arr

Syscall

- O Mars provê alguns serviços do sistema operacional através da instrução *syscall*.
- Para utilizar um serviço:
  - Carregar o código do serviço no registrador $v0;
  - Carregar os argumentos do serviço nos registradores $a0-$a3;
  - Chama a instrução *syscall*.

Syscall

Serviço	***Código***	***Argumentos***	***Resultado***
print_int	**1**	**$a0 = inteiro**	
print_float	**2**	**$f12 = float**	
print_double	**3**	**$f12 = double**	
print_string	**4**	**$a0 = string**	
read_int	**5**		**inteiro (em $v0)**
read_float	**6**		**float (em $v0)**
read_double	**7**		**double (em $v0)**
read_string	**8**	**$a0 = buffer, $a1 = tamanho**	
exit	**10**		
print_char	**11**	**$a0 = char**	
exit2	**17**	**$a0 = resultado**	

Perguntas?

See ya!