Caderno de Infraestrutura de Hardware

Marconi Gomes

16 de agosto de 2019

1 Introdução - Instruções e funcionamento básico

1.1 Abstração

As linguages de programação podem ser divididas em **4 níveis**:
- Linguagem de Máquina (binário)
- Linguagem de montagem (Assembly)
- Linguagem de alto nível (Java, C++, etc)
- Linguagem de 4ª gearção (PL/SQL, NATURAL, etc)

O menor nível de abstração que o programador pode ver antes do código de realmente chegar ao binário, chama-se Instruction Set Architeture (ISA), que é um **repositório de instruções**, ela é realmente a interface entre Software e Hardware. Ela vai me dizer quais as intruções e registradores que posso usar, como acessar a memória, etc.

1.2 Assembly

É uma linguagem que é dependente de arquitetua, ou seja, para cada tipo (x86, ARM) é um tipo de assembly diferente.

1.3 Compilador

Definição: é um programa que traduz de uma linguagem de mais alto nível (ex. Java) para uma de menor nível (assembly) que o computador entende.
A diferença entre um **compilador** e um **interpretador** é que o compilador traduz tudo primeiramente apenas e depois executa. O interpretador traduz e executa cada linha por vez.
Exemplos de linguages compiladas (completamente): C, C++, etc.
Exemplos de linguages interpretadas (completamente): JavaScript, Python.
Exemplos de linguages semi-interpretadas e semi-compiladas: Java!

1.4 Visão funcional de um computador

Um computador pode (e deve) realizar 4 ações:
→ Mover dados (Barramento)
→ Controlar ações (CPU)
→ Armazenar dados (Memória)

→ Processar dados (CPU)

Logo, a CPU faz sempre as seguintes coisas:
Busca → Decodificação → Execução

Os seguintes registradores são os mais comuns num computador:
PC (Program counter): Buscar o endereço da instrução
MAR (Memory Address Register): Guarda dinamicamente endereços que possam ser usados posteriormente.
IR (Instruction Register): Recebe a instrução do PC e a armazena.
AC (Acumulator): É um registrador comum genérico.

1.5 Evolução das ISAs

Inicialmente as ISAs tinham poucas instruções básicas, dificultando o trabalho dos programadores, então foram implementadas instruções mais complexas, assim surgiram os **CISC** (Complex Instruction Set Computer) e **RISC** (Reduced Instruction Set Computer).
Exemplos de processadores CISC: Intel x86, AMD, etc...
Exemplos de processadores RISC: ARMS, MIPS, etc...

2 MIPS

2.1 Instruções - Aritméticas

As instruções aritméticas no MIPS **sempre** (exceto multiplicação e divisão) **possuem 3 operandos**: destino, fonte 1 e fonte 2.
- Cada instrução só faz **uma** operação aritmética.
- Para ser mais eficiente, os operandos de uma instrução aritmética devem estar nos registradores.
- No MIPS **todos** os registradores possuem 32 bits, ou seja, 4bytes e logo 32 bits são uma **word**.

- No MIPS os registradores tem nomes da forma: $sX (para armazenar variáveis de programas, onde **X varia de 0 a 7**) e $tY (para armazenar valores temporários, onde **Y varia de 0 até 9**).
Internamente, o processador categoriza iniciando de $t0 até $t7 como os registradores 0 a 15, os $s0 até $s7 sendo de 16 a 23, e finalmente $t8 e $t9 como 24 e 25 respectivamente.

Formato R de instrução: O espaço de 32 bits (word) é dividido em 6 espaços, respectivamente por:
1º - op (6 bits) — [Armazena o opcode da instrução]
2º - rs (5 bits) — [Registrador que contém 1º operando fonte]
3º - rt (5 bits) — [Registrador que contém 2º operando fonte]
4º - rd (5 bits) — [Registrador destino que contém o resultado]
5º - shamt (5 bits) — [Shift Amount, que não é utilizado para add e sub]
6º - funct (6 bits) — [Função que estende o opcode]

Representando ADD e SUB na Máquina: É função do compílador associar as variáveis a registradores. Veja o exemplo:

f = (g + h) – (i + j);

As variáveis f, g, h, i, j estão atribuídas, respectivamente, aos registradores $s0, $s1, $s2, $s3 e $s4. O programa compilado em MIPS ficará da seguinte forma:

add $t0, $s1, $s2 (registrador $t0 contém g + h)
add $t1, $s3, $s4 (registrador $t1 contém i + j)
sub $s0, $t0, $t1 (f recebe $t0 – $t1, que equivale a (g + h)–(i + j))

Operações de Transferência de Dados: move dados entre a memória e o registrador.
- Para acessar uma word na memória, a instrução deve fornecer o **endereço da memória**.
- A memória é apenas um array grande e unidimensional. O endereço da memória é o index do array.

Operação **lw** (**l**oad **w**ord):

lw r, offset(end_inicial)

- *r*: registrador que será carregado;
- *offset*: uma constante de deslocamento;
- *end_inicial*: um registrador de base que contém o endereço inicial.

Referências