Caderno de Infraestrutura de Hardware

Marconi Gomes e Lucas Rodrigues

August 21, 2019

1 Introdução - Instruções e funcionamento básico

Código do Classroom: 215max.

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 arquitetura, 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 compilador associar as variáveis a registradores. Veja o exemplo:

f = (g + h) – (i + j);

As variáveis *f*, *g*, *h*, *i* e *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.

→ Operação sw (**s**tore **w**ord):

```
sw r, offset(end_inicial)
```

- *r*: registrador onde serão salvos os dados;
- *offset*: uma constante de deslocamento;
- *end_inicial*: um registrador de base que contém o endereço inicial.

2.2 Controle de Fluxo no MIPS

Nas linguagens de máquina existem dois tipos de desvio:
- Branch ↔ Desvio Condicional (feitos por meio de **if**s, **else**s, **case**s e **loop**s nas linguagens de alto nível.)
- Jump ↔ Desvio Incondicional (feitos através de **GoTo**s)

`Nota: a parte de fluxo de códigos se encontra incompleta.`

2.3 Procedimentos

Num programa, quando quero desviar o fluxo, geralmente são chamadas funções e métodos, chamados agora de **subrotinas**.
Ela faz com que o programa execute todas as intruções na subrotina, e então necessariamente ele tem que voltar à instrução onde ele estava.

O operando relacionado ao label nas instruções na verdade é baseado na quantidade de linhas em relação à linha que está sendo executada.
Uma vez que uma I instrução é formada da seguinte maneira: op (6 bits), rs (5 bits), rt (5 bits) e **deslocamento (16bits)**.

O fluxo é o seguinte:
1 - A rotina que faz a chamada (caller) coloca os argumentos em um lugar que a subchamada (callee) pode acessar.
2 - O Caller transfere o controle para o callee
3 - O Callee adquire os recursos de armazenamento necessários
4 - O Callee executa suas próprias instruções
5 - O Callee coloca os valores de resultado onde o Caller possa acessá-los.
6 - Finalmente, o Callee retorna o controle ao Caller.

Detalhes de cada passo:
1 - No MIPS, os registradores $a0 até $a3 são usados para armazenar argumentos.
2, 3, 4 e 5 - A subchamada é feita usando a instrução `jal` (que tem formato igual ao jump), que significa **J**ump **A**nd **L**ink, seguido do label que identifica a função a ser iniciada. Nessa mesma instrução, ela também guarda o endereço da próxima linha depois da chamada, para que ele assim que terminar a subchamada, volte para continuar o fluxo do código.
6 - Para retornar seu valor, usa-se a instrução `jr reg`, que significa **J**ump **R**egister de formato: op = 0, rs = `reg` (que é o registrador de retorno), rt = 0, rd = 0, shamt = 0 e funct = 8.

E se precisarmos de mais registradores?
Se temos mais argumentos para o número de registradores disponíveis, usamos uma **pilha**. No MIPS, registradores que não estão sendo usados são salvos na pilha de memória, lá as informações excedentes necessárias são guardadas.
Para implementarmos uma **pilha**, usamos um endereço `$sp` (**S**tack **P**ointer), esse endereço é fixo e os demais elementos da pilha crescem com endereços menores.