Simulador de Microinstruções

Arquitetura de Computadores I

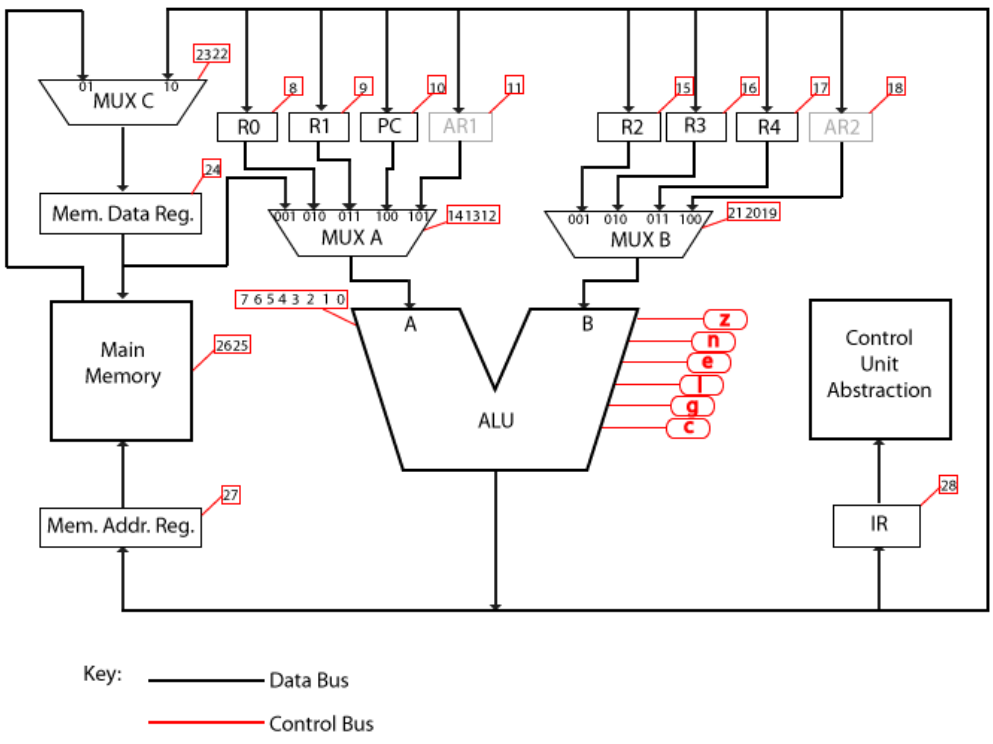
Objetivos

O objetivo deste trabalho é construir um sistema que represente cada componente de um computador bem simples mas que sirva para expressar conceitos encontrados na maioria dos computadores atuais e que formam a base dos mesmos.

O sistema permite que o usuário escreva programas em uma linguagem de montagem (assembler) baseada na linguagem usada pela Intel.

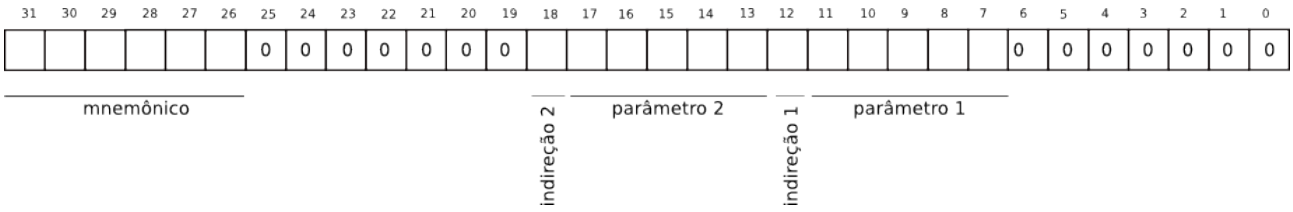
O sistema mostra como cada Instrução é subdividida em pedaços menores, chamados de *Microinstruções*, que são o que efetivamente é executado pelo processador em cada ciclo de máquina.

Estrutura do Computador



Estrutura das Instruções (todos os casos menos desvios)

As instruções são strings binárias de 32 bits, organizadas da seguinte forma:

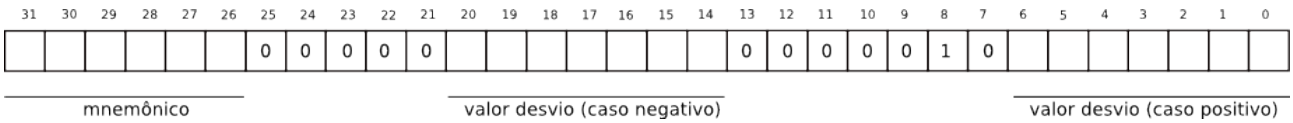


Se houver algum parâmetro do tipo CONSTANTE, ele será guardado na posição de memória imediatamente a seguir. Ela será interpretada não como uma Instrução mas como um simples número.

Estrutura das Instruções (desvios condicionais e incondicionais)

Instruções do tipo desvio são mais simples pois não são numerosas e só aceitam operandos (parâmetros) do tipo CONSTANTE.

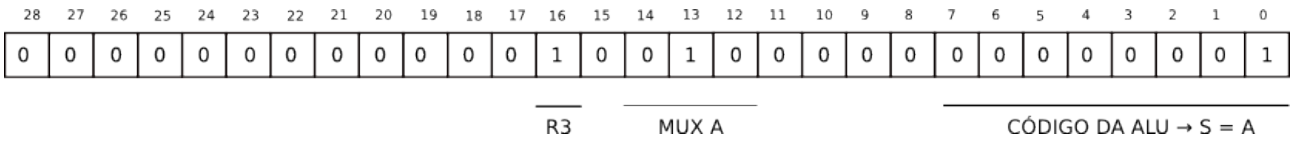
Por essa razão, o valor do desvio é, neste caso, fornecido dentro da própria Instrução (no caso de Instruções normais, valores constantes são armazenados em uma linha de memória própria). Note que o desvio é sempre um número inteiro (operando CONSTANTE) e deve variar entre -2^7 e $+2^7$.



Estrutura das Microinstruções

As Microinstruções são compostas de 29 (0 - 28) bits. Cada bit da Microinstrução representa o estado do respectivo campo de controle, como mostrado no diagrama.

Por exemplo, a microinstrução a seguir representa a Instrução `mov(R0,R3)`. Apesar de ser uma (macro)Instrução, ela pode ser executada em apenas um ciclo de máquina e pode ser, portanto, representada em uma única Microinstrução.



Pode-se ver que, nesta Microinstrução, somente os bits que controlam as atividades a serem executadas neste ciclo de CPU são ativadas, a saber, habilitar o registrador R3 para recebimento, configurar o multiplexador MUX A para “puxar” o conteúdo do registrador R0 e, finalmente, configurar a ALU para que execute uma “conta” bastante simplória: copie o conteúdo da entrada A (à esquerda) para a saída.

Exemplos de Programas

Como o uso do nosso sistema depende do uso de Instruções assembler, nesta seção escreveremos alguns programas em pseudo-código e depois diremos como tal programa deve ser representado em linguagem de montagem (assembler).

1) Código que incrementa o valor do registrador `R2` cinco vezes.

Pseudo-código

```
R0 ← 0
R1 ← 50
R2 ← 0
while(R0 <= 5){
    R2 ← R2 + R1
    R0 ← R0 + 1
}
//R2 vale 250
```

Assembler

```
mov(0,R0)
mov(50,R1)
mov(0,R2)
```

```
add(R1,R2)
add(1,R0) ;incrementa o contador
cmp(5,R0) ;R0 - 5
brz(1)  ;se a conta anterior deu zero, pule a próxima instrução (fim)
brn(-5) ;se a conta anterior deu negativo, volta 5 instruções
```

2) Código que calcula o valor de 2^{10}

Pseudo-código

```
R0 ← 2
R4 ← 9
início:

R0 ← R0 * 2
if( R4-1 == 0 ){
    goto fim
}
goto início
fim:

R2 ← R0
//R2 vale 1024
```

Assembler

```
mov(2,R0)
mov(9,R4) ;só 9 laços pois 2 é 2¹
shl(R0) ;shift left para multiplicar por dois
sub(1,R4)
brz(1) ;se a conta anterior deu zero, pule a próxima instrução (fim)
rjmp(-4) ;caso contrário, executar o shift outra vez
mov(R0,R2)
```

Observações

- Comportamento da instrução `cmp(A,B)`

  A instrução `cmp(A,B)` equivale a fazer B-A:

  ```
  mov(50,R1)
  cmp(20,R1)
  // flag G (greater than) será setada
  ```

```
mov(20,R1)
cmp(100,R1)
```
//flag L (less than) e flag N (número negativo) serão setadas