Relatório: Instrução `AVG`

Disciplina: SSC0513 - Organização e Arquitetura de Computadores
Processador: ICMC (Simulador)

---

1 Manual de Uso

1.1 Identificação

- **Mnemônico:** `AVG` (*Average*)
- **Opcode Decimal:** 38
- **Opcode Binário:** `100110`
- **Formato:** Aritmético (Registrador para Registrador)

1.2 Sintaxe

```
AVG RX, RY, RZ
```

Assembly

1.3 Descrição Funcional

A instrução calcula a média aritmética inteira entre os valores contidos nos registradores `RY` e `RZ`, armazenando o resultado em `RX`.

1.4 Operação Matemática

$$RX \leftarrow \frac{(RY + RZ)}{2}$$

1.5 Detalhes de Comportamento

1. **Soma:** Realiza a soma de 16 bits dos operandos $(RY + RZ)$.
2. **Divisão:** O resultado é dividido por 2 utilizando um deslocamento de bits para a direita (*bitwise right shift*).
3. **Truncamento:** Como o processador opera apenas com números inteiros, qualquer parte fracionária resultante da divisão é descartada (arredondamento para baixo).
    - *Exemplo:* A média de 5 e 2 resulta matematicamente em 3.5. O processador armazenará o valor **3**.
4. **Flags Afetadas:**

- `OVERFLOW (V)`: Ativada se a soma inicial ($RY + RZ$) exceder o valor máximo de 16 bits (65.535), indicando que a média pode estar incorreta.
- `ZERO (Z)`: Ativada se o resultado final for 0.

2 Implementação

A inclusão da nova instrução no simulador `simple_simulator.c` seguiu o ciclo padrão de processamento (Fetch-Decode-Execute), exigindo modificações em três etapas do código fonte.

2.1 A. Definição (Pré-processamento)

Foi definido um novo identificador para o opcode. Escolheu-se o valor **38**, pois estava livre dentro da faixa de instruções aritméticas (aquelas que iniciam com os bits `10...`).

```
1 #define AVG 38 // Binario: 100110
```

2.2 B. Decodificação (`STATE_DECODE`)

A instrução `AVG` foi adicionada ao `switch(opcode)` no mesmo bloco das instruções aritméticas (`ADD`, `SUB`, etc.).

- **Justificativa:** A instrução `AVG` compartilha o mesmo *Datapath* (Caminho de Dados) das somas padrão, aproveitando a estrutura existente do processador.
- **Configuração de Hardware:**
  1. **Operandos:** O Mux3 (`selM3`) seleciona `RY` e o Mux4 (`selM4`) seleciona `RZ`.
  2. **Fluxo:** O Mux2 (`selM2`) é configurado para `sULA` (saída da ULA).
  3. **Escrita:** `LoadReg[rx]` é ativado para gravar o resultado no registrador de destino.

2.3 C. Execução (`ULA`)

A lógica matemática foi implementada dentro da função da Unidade Lógica e Aritmética.

- **Lógica Implementada:**
  1. Calcula-se a soma temporária `temp = x + y`.
  2. Verifica-se overflow antes da divisão (para atualizar o registrador de flags corretamente).
  3. Aplica-se a operação `result = temp >> 1`. O operador `>> 1` desloca todos os bits uma posição para a direita, o que é a maneira mais eficiente em hardware de dividir um número inteiro por 2.

3 Casos de Teste e Validação (`cpuram.mif`)

Abaixo estão os três cenários de teste executados para validar a instrução.

3.1 Caso 1: Teste Simples (Letra 'A')

Objetivo: Calcular a média de 60 e 70.
Matemática: $(60 + 70)/2 = 65$.
Saída Esperada: O caractere ASCII 65, que corresponde à letra **'A'**.

```
CONTENT
BEGIN
0:1110000010000000
1:0000000000111100
2:1110000100000000
3:0000000001000110
4:1001100000010100
5:1100100000000000
6:0011110000000000
7:0000000000000000
8:0000000000000000
END
```

Arquivo cpuram.mif

Análise Detalhada (Decodificação):

- **Linha 0:** A instrução `LOADN` (111000) prepara o registrador R1 (001).
- **Linha 1:** O valor imediato **60** (`...111100`) é carregado em R1.
- **Linha 2:** A instrução `LOADN` (111000) prepara o registrador R2 (010).
- **Linha 3:** O valor imediato **70** (`...1000110`) é carregado em R2.
- **Linha 4:** A instrução `AVG` (100110), com Destino=R0(000), Fonte1=R1(001), Fonte2=R2(010).
  - *Ação:* Calcula $(60 + 70)/2 = 65$. R0 recebe 65.
- **Linha 5:** A instrução `OUTCHAR` (110010), lendo de R0(000). Imprime **'A'**.
- **Linha 6:** A instrução `HALT` (001111). Para a execução.

3.2 Caso 2: Teste de Truncamento (Letra 'B')

Objetivo: Validar o comportamento com números ímpares (média fracionária).
Matemática: Média de 66 e 67. $(66 + 67)/2 = 66.5$.
Saída Esperada: O processador deve truncar para **66** (Letra **'B'**), provando que não há arredondamento para cima.

```
CONTENT
BEGIN
0:1110000010000000
1:0000000001000010
2:1110000100000000
```

```
3:0000000001000011
4:1001100000010100
5:1100100000000000
6:0011110000000000
7:0000000000000000
8:0000000000000000
END
```

Arquivo cpuram.mif

Análise Detalhada (Decodificação):

- **Linha 0-1:** Instrução `LOADN` carrega **66** em R1.
- **Linha 2-3:** Instrução `LOADN` carrega **67** em R2.
- **Linha 4:** Instrução `AVG` (100110), com Destino=R0(000), Fonte1=R1(001), Fonte2=R2(010).
  - *Ação:* A soma resulta em 133. O *shift right* descarta o último bit (que vale 0.5). O resultado em R0 é 66.
- **Linha 5:** Instrução `OUTCHAR` (110010), lendo de R0. Imprime **'B'**.
- **Linha 6:** Instrução `HALT` (001111).

3.3 Caso 3: Teste Elaborado ("SIMOES")

Objetivo: Escrever a palavra "SIMOES" usando a instrução `AVG` para gerar cada código ASCII dinamicamente.
Estratégia: Mantemos o registrador **R1** fixo com o valor **100**. Para cada letra desejada, carregamos um valor em **R2** que, somado a 100 e dividido por 2, resulte na letra correta.

```
CONTENT
BEGIN
0:1110000010000000
1:0000000001100100
2:1110000100000000
3:0000000001000010
4:1001100000010100
5:1100100000000000
6:1110000100000000
7:0000000000101110
8:1001100000010100
9:1100100000000000
10:1110000100000000
11:0000000000110110
12:1001100000010100
13:1100100000000000
14:1110000100000000
15:0000000000111010
16:1001100000010100
17:1100100000000000
18:1110000100000000
19:0000000000100110
20:1001100000010100
21:1100100000000000
22:1110000100000000
```

```
23:0000000001000010
24:1001100000010100
25:1100100000000000
26:0011110000000000
27:0000000000000000
28:0000000000000000
END
```

Arquivo cpuram.mif

Análise Detalhada:

1. **Configuração Inicial (Base)**
   - **Linha 0:** A instrução `LOADN` (111000) para o registrador R1 (001).
   - **Linha 1:** O valor imediato **100**. *R1 é fixado em 100.*
2. **Letra 'S' (ASCII 83)**
   - **Linha 2:** A instrução `LOADN` (111000) para o registrador R2 (010).
   - **Linha 3:** O valor imediato **66**.
   - **Linha 4:** A instrução `AVG` (100110), Destino=R0.
     - *Lógica:* Calcula $(100 + 66)/2 = 83$. R0 recebe o código de 'S'.
   - **Linha 5:** A instrução `OUTCHAR` imprime **'S'**.
3. **Letra 'I' (ASCII 73)**
   - **Linha 6:** A instrução `LOADN` (111000) para o registrador R2 (010).
   - **Linha 7:** O valor imediato **46**.
   - **Linha 8:** A instrução `AVG` (100110).
     - *Lógica:* Calcula $(100 + 46)/2 = 73$. R0 recebe o código de 'I'.
   - **Linha 9:** A instrução `OUTCHAR` imprime **'I'**.
4. **Letra 'M' (ASCII 77)**
   - **Linha 10:** A instrução `LOADN` (111000) para o registrador R2 (010).
   - **Linha 11:** O valor imediato **54**.
   - **Linha 12:** A instrução `AVG` (100110).
     - *Lógica:* Calcula $(100 + 54)/2 = 77$. R0 recebe o código de 'M'.
   - **Linha 13:** A instrução `OUTCHAR` imprime **'M'**.
5. **Letra 'O' (ASCII 79)**
   - **Linha 14:** A instrução `LOADN` (111000) para o registrador R2 (010).
   - **Linha 15:** O valor imediato **58**.
   - **Linha 16:** A instrução `AVG` (100110).
     - *Lógica:* Calcula $(100 + 58)/2 = 79$. R0 recebe o código de 'O'.
   - **Linha 17:** A instrução `OUTCHAR` imprime **'O'**.
6. **Letra 'E' (ASCII 69)**

- **Linha 18:** A instrução `LOADN` (111000) para o registrador R2 (010).
- **Linha 19:** O valor imediato **38**.
- **Linha 20:** A instrução `AVG` (100110).
  - *Lógica:* Calcula $(100 + 38)/2 = 69$. R0 recebe o código de 'E'.
- **Linha 21:** A instrução `OUTCHAR` imprime **'E'**.

7. **Letra 'S' (ASCII 83)**

- **Linha 22:** A instrução `LOADN` (111000) para o registrador R2 (010).
- **Linha 23:** O valor imediato **66** (reutilizando o valor usado na primeira letra).
- **Linha 24:** A instrução `AVG` (100110).
  - *Lógica:* Calcula $(100 + 66)/2 = 83$. R0 recebe o código de 'S'.
- **Linha 25:** A instrução `OUTCHAR` imprime **'S'**.

8. **Finalização**

- **Linha 26:** A instrução `HALT` (001111). Encerra a simulação.