UNIVERSIDADE FEDERAL DE VIÇOSA

Montador Simplificado

Organização de Computadores I

Aluno: Pedro Henrique Silva Oliveira

Matrícula: 2677

Professor: José Augusto Miranda Nacif

Florestal

2019

SUMÁRIO

1. INTRODUÇÃO

A documentação apresenta, em alto nível, a descrição do código implementado pelo aluno Pedro Oliveira e, também, suas particularidades e resultados obtidos.

2. DESENVOLVIMENTO

O código é constituído por apenas um TAD (Assembler_Functions), que engloba as funções necessárias para a conversão para a linguagem de máquina e um main que controla o fluxo da conversão.

O formato padrão da entrada de arquivos é: Cada linha possui uma instrução em assembly.

```
add $t0, $s1, $s2
sub $t1, $s6, $s7
and $t1, $s6, $s7
or $t1, $s6, $s7
nor $t1, $s6, $s7
addi $t1, $s6, 8
andi $t0, $s4, $s5
ori $t0, $s2, $s6
sll $t1, $s6, $s7
srl $t1, $s6, $s7
```

2.1. O CÓDIGO:

Começa-se lendo o arquivo, dividindo e armazenando cada informação com a função da biblioteca string.h “strtok” e, logo em seguida, convertemos cada dado para decimal para que, futuramente, possamos convertê-los para binário.

A função “defineOperacao” analisa o tipo de operação que deverá ser executada e, assim, define os respectivos números decimais que serão atribuidos:

Código de exemplo (Análise de uma instrução do tipo Registrador):

Formato R: opcode(6) rs(5) rt(5) rd(5) shamt(5) funct(6)

```
if(strcmp(segmento,"add")==0){
    *Function = 32;
    *opcode = 0;
    *shamt = 0;
    *Tinstrucao ''= TRegistrador;
}
```

Código de exemplo (Análise de uma instrução do tipo Imediato):

Formato I: opcode(6) rs(5) rt(5) endereço(16)

```
if(strcmp(segmento,"addi")==0){
    *opcode = 8;
    *shamt = 0;
    *Function = 0;
    *Tinstrucao = TImediato;
}
```

Após definir o tipo de instrução e seus respectivos dados em decimal, também é necessário saber o número decimal dos registradores das instruções. E, para isso, fez-se necessária a função “retornaReg”. A função apenas analisa o nome do registrador na instrução assembly e retorna o número decimal que corresponde a ele.

Código exemplo “retornaReg”:

```
/* $t0 a $t7 são mapeados nos registradores 8 a 15 */
if((strcmp(segmento,"t0")) == 0)    return 8;
if((strcmp(segmento,"t1")) == 0)    return 9;
if((strcmp(segmento,"t2")) == 0)    return 10;
if((strcmp(segmento,"t3")) == 0)    return 11;
if((strcmp(segmento,"t4")) == 0)    return 12;
if((strcmp(segmento,"t5")) == 0)    return 13;
if((strcmp(segmento,"t6")) == 0)    return 14;
if((strcmp(segmento,"t7")) == 0)    return 15;
/* $s0 a $s7 são mapeados nos registradores 16 a 23 */
if((strcmp(segmento,"s0")) == 0)    return 16;
if((strcmp(segmento,"s1")) == 0)    return 17;
if((strcmp(segmento,"s2")) == 0)    return 18;
if((strcmp(segmento,"s3")) == 0)    return 19;
if((strcmp(segmento,"s4")) == 0)    return 20;
if((strcmp(segmento,"s5")) == 0)    return 21;
if((strcmp(segmento,"s6")) == 0)    return 22;
if((strcmp(segmento,"s7")) == 0)    return 23;

if(opcode == 0){//Registrador (tem deslocamento)
    *deslocou = TRUE;
    return atoi(segmento);
}else//Imediata (constante)
    return atoi(segmento);
```

E, uma vez definidos os números decimais que correspondem aos registradores e os demais dados, basta converter para binário na devida ordem e, em seguida, enviar para o arquivo de saída. Para isto, foi criada uma função “retornaSaida” que conta com duas funções auxiliares, “geraInstrucaoBinaria” e “retornaBinario”.

A função “retornaSaida” avalia se a função é do tipo registrador ou do tipo imediato e define a ordem em que os dados serão inseridos no arquivo através da função “geraInstrucaoBinaria”.

Código exemplo “retornaSaida”:

```
if(Tinstrucao == TRegistrador){//TIPO REGISTRADOR

    geraInstrucaoBinaria(arqSaida,opcode,OPCODE);//opcode
    geraInstrucaoBinaria(arqSaida,reg[1],REG);//rs
    geraInstrucaoBinaria(arqSaida,reg[2],REG);//rt
    geraInstrucaoBinaria(arqSaida,reg[0],REG);//rd
    geraInstrucaoBinaria(arqSaida,Shamt,SHAMT);//shamt
    geraInstrucaoBinaria(arqSaida,Function,FUNCTION);//function

}else if(Tinstrucao == TImediato){// TIPO IMEDIATO
    geraInstrucaoBinaria(arqSaida,opcode,OPCODE);//opcode
    geraInstrucaoBinaria(arqSaida,reg[0],REG);//rs
    geraInstrucaoBinaria(arqSaida,reg[1],REG);//rt
    geraInstrucaoBinaria(arqSaida,reg[2],ENDERECO);//Endereço
}
```

A função “geraInstrucaoBinaria”, por sua vez, converte cada dado recebido em decimal para binário com o auxílio da função “retornaBinario” e registra no arquivo.

Código exemplo “geraInstrucaoBinaria”:

```
cadeiaBinaria = retornaBinario(n,&contPosicoes);


if(TCadeia == OPCODE || TCadeia == FUNCTION){//6 bits
    for(i=10;i<16;i++)
        fprintf(arqSaida,"%d",cadeiaBinaria[i]);
}
if(TCadeia == REG || TCadeia == SHAMT){//5 bits
    for(i=11;i<16;i++)
        fprintf(arqSaida,"%d",cadeiaBinaria[i]);
}
if(TCadeia == ENDERECO){//16 bits
    for(i=0;i<16;i++)
        fprintf(arqSaida,"%d",cadeiaBinaria[i]);
}
```

E, por fim, a função “retornaBinario”, que converte os números decimais para binários. Ela usa o método de divisões sucessivas e armazena os restos das mesmas em um vetor. Tal método se assemelha ao ensinado na disciplina de “Introdução a Sistemas Lógicos Digitais” para a conversão.

Código exemplo “retornaBinario”:

```
while (n != 1){
    quociente[*contPosicoes] = n / 2;
    resto[*contPosicoes] = n % 2;
    n = quociente[*contPosicoes];
    (*contPosicoes)++;
}
```

2.2. RESULTADOS:

Entrada:	**Saída:**
`add $t0, $s1, $s2`	`000000 10001 10010 01000 00000 100000`
`sub $t1, $s6, $s7`	`000000 10110 10111 01001 00000 100010`
`and $t1, $s6, $s7`	`000000 10110 10111 01001 00000 100100`
`or $t1, $s6, $s7`	`000000 10110 10111 01001 00000 100101`
`nor $t1, $s6, $s7`	`000000 10110 10111 01001 00000 100111`
`addi $t1, $s6, 8`	`001000 01001 10110 0000000000001000`
`andi $t0, $s4, $s5`	`001100 01000 10100 0000000000010101`
`ori $t0, $s2, $s6`	`001101 01000 10010 0000000000010110`
`sll $t1, $s6, $s7`	`000000 10110 10111 01001 00000 000000`
`srl $t1, $s6, $s7`	`000000 10110 10111 01001 00000 000010`