Sistemas Digitais II

Caio Vinícius Jun Pereira - 13731921
Caio Hokama Freitas - 11812883
João Pedro Lopes de Sousa Gomes - 13680810
Erick Diogo de Almeida Sousa - 13680960
Pedro Guilherme Croitor da Cruz - 13685002

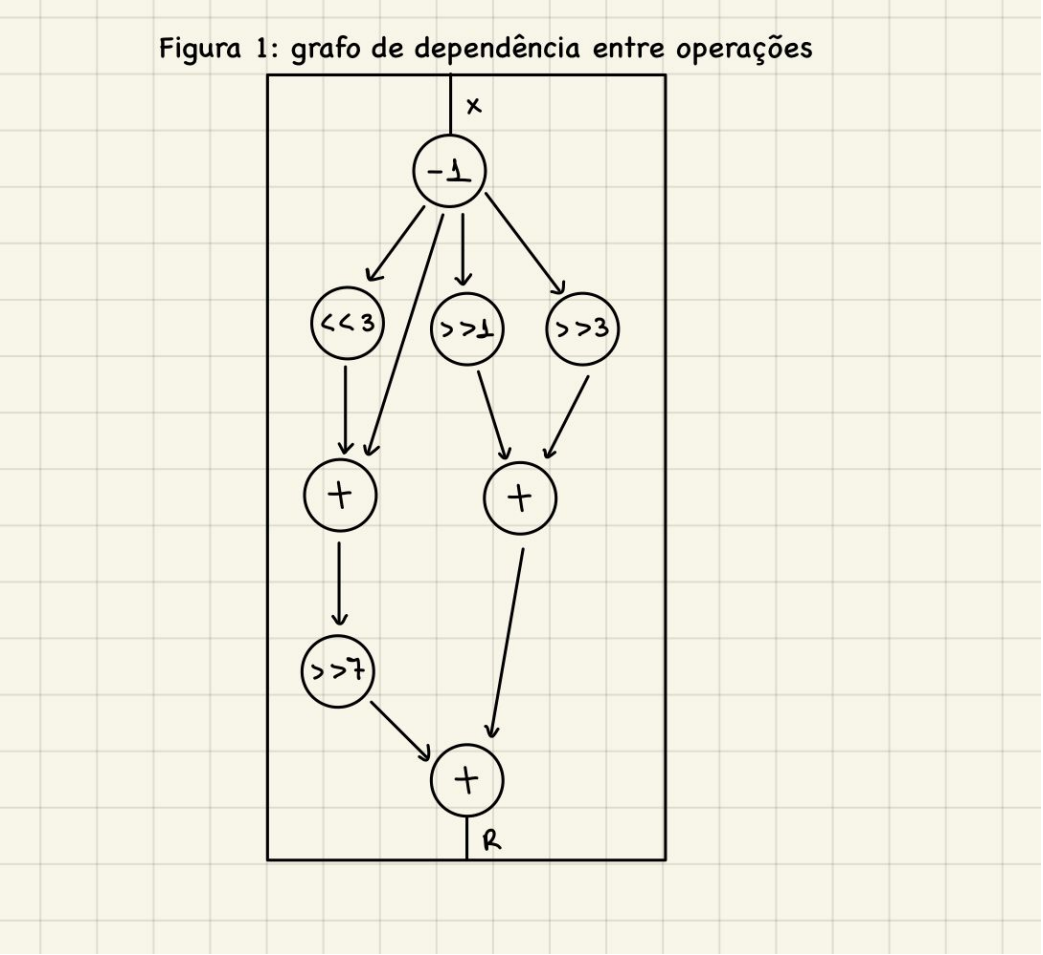
Atividade Formativa 7

1. ASAP, ALAP e CDFC

1.1 Control Data Flow Chart (CDFC)

Para facilitar o desenvolvimento, foi criado primeiramente o Control Data Flow Graph (CDFG) do algoritmo. A partir do mesmo, é mais simples de visualizar o tipo de cada operação e assim preencher as tabelas pedidas.

Figura 1: grafo de dependência entre operações



1.2 Tabela 2 (ASAP, ALAP e RC)

Para calcular a mobilidade de uma operação, é necessário saber seus estados correspondentes em As Soon As Possible (ASAP) e em As Late As Possible (ALAP). Por isso, o grupo decidiu por fazer primeiro a tabela 2, a partir do CDFC criado:

TABELA 2			
Estado	ASAP	ALAP	RC
S_1	y = x - 1;	y = x - 1;	y = x - 1;
S_2	t_1 = y << 3; t_3 = y >> 1; t_4 = y >> 3;	t_1 = y << 3;	t_1 = y << 3; t_3 = y >> 1;
S_3	t_2 = y + t_1; w = t_3 + t_4;	t_2 = y + t_1; t_3 = y >> 1; t_4 = y >> 3;	t_2 = y + t_1; t_4 = y >> 3;
S_4	z = t_2 >> 7;	z = t_2 >> 7; w = t_3 + t_4;	z = t_2 >> 7; w = t_3 + t_4;
S_5	R = z + w	R = z + w	R = z + w
S_6	-	-	-
S_7	-	-	-
S_8	-	-	-

Para o preenchimento da coluna ASAP, cada operação é acionada somente quando todas as suas variáveis dependentes já estiverem definidas. Por exemplo, a operação t_2 = y + t_1 só ocorre no estado S_3 pois depende das variáveis y e t_1, sendo que t_1 só é definida no estado S_2.

Para o preenchimento da coluna ALAP, a lógica é inversa. Inicia-se preenchendo a última operação (R = z + w). A partir disso, no estado anterior, são definidas as variáveis dos quais essa operação depende, até que todas as operações tenham um estado correspondente. Por exemplo, as variáveis z e w são definidas logo antes da operação R = x + w.

Por fim, para o preenchimento da coluna RC, a ideia é ser igual à coluna ASAP, porém considerando o número limitado de componentes. Por exemplo, o enunciado diz que só há um somador disponível, portanto, o estado S_3 da coluna ASAP não é possível.

Quanto aos deslocadores, o enunciado disse que há quantos deslocadores forem necessários. No caso, apenas um deslocador para a direita e um para a esquerda são necessários. Se houvesse mais deslocadores, uma alteração possível seria executar a operação t_4 = y >> 3 no estado S_2 juntamente com os outros deslocamentos. Porém, essa ação não diminuiria o número de estados, mas aumentaria a complexidade do circuito com o acréscimo de um componente.

Como se pode ver, o circuito final com Restrição de Custo (RC) tem apenas 5 estados, mesmo número obtido em ASAP. Isto é, mesmo com o número limitado de somadores, o circuito está com o menor número possível de estados.

1.3 Tabela 1 (Mobilidade de operações)

Para o cálculo de mobilidade de uma operação é necessário antes fazer o ASAP e o ALAP. Assim, tendo o primeiro e último estado que cada operação pode ocupar. Por fim, a mobilidade é calculada subtraindo o maior estado possível do menor estado possível.

TABELA 1		
Variável	Operação	Mobilidade
y	$y = x - 1$	0
t_1	$t_1 = y << 3$	0
t_2	$t_2 = y + t_1$	0
t_3	$t_3 = y >> 1$	1
z	$z = t_2 >> 7$	0
t_4	$t_4 = y >> 3$	1
w	$w = t_3 + t_4$	1
R	$R = z + w$	0

2. Codificação da Testbench

Para codificação da testbench, o grupo usou como base as testbenches feitas nas atividades formativas anteriores. Segue o código:

```
    -- Complete no espaco o codigo do port do testbench
    LIBRARY IEEE;
    USE IEEE.numeric_bit.ALL;

    ENTITY testbench IS
    END testbench;

    -- Complete nogg espaco os sinais e componentes a serem
utilizados
```

```
ARCHITECTURE beh OF testbench IS

    COMPONENT log IS
        PORT (
            clock, inicio : IN BIT;
            x : IN bit_vector(7 DOWNTO 0);
            R : OUT bit_vector(7 DOWNTO 0);
            fim : OUT BIT
        );
    END COMPONENT;

    SIGNAL clk, run : BIT;

    -- Complete no espaco para gerar o clock
    -- Depois , complete no espaco para que
    -- o DUT seja instanciado
    SIGNAL X, r : bit_vector(7 DOWNTO 0);
    SIGNAL Inicio, Fim : BIT;

BEGIN
    ASSERT false REPORT "Inicio da simulacao" SEVERITY note;
    clk <= (NOT clk) AND run AFTER 10 ns;
    DUT : log PORT MAP(
        clock => clk,
        inicio => Inicio,
        x => X,
        R => r,
        fim => Fim
    );

    -- Insira neste trecho o caso de teste
    PROCESS
    BEGIN
        run <= '1';
        x <= "11000001";
        WAIT FOR 1 ns;
        inicio <= '1';

        ASSERT (R = "01001010") REPORT "Houve um erro. O
resultado esperado era 01001010, o obtido foi " &
INTEGER'image(to_integer(unsigned(R))) SEVERITY error;
        run <= '0';
        WAIT;
```

```
        END PROCESS;
    END beh;
```

Como se pode observar, caso o resultado obtido não seja o esperado, uma mensagem de erro mostra o resultado obtido.