# Projeto de Sistemas Digitais

*versão 06/2020*

PCS3115

# Tópicos

- Projeto de Sistemas Digitais
  - Metodologia de projeto
  - Projeto Exemplo: Multiplicador
  - Exercício: Raiz Inteira

- Referências:

  Haskell & Hanna, **Digital Design using Digilent FPGA Boards** – capítulo 9
  Gajski, **Principles of digital design** - Capítulo 8
  W.V. Ruggiero & C.B. Margi, **Capítulo 1: Metodologia de Projeto Estruturado para Sistemas Digitais**. Apostila de PCS2022

# Desenvolvimento de Circuitos Digitais

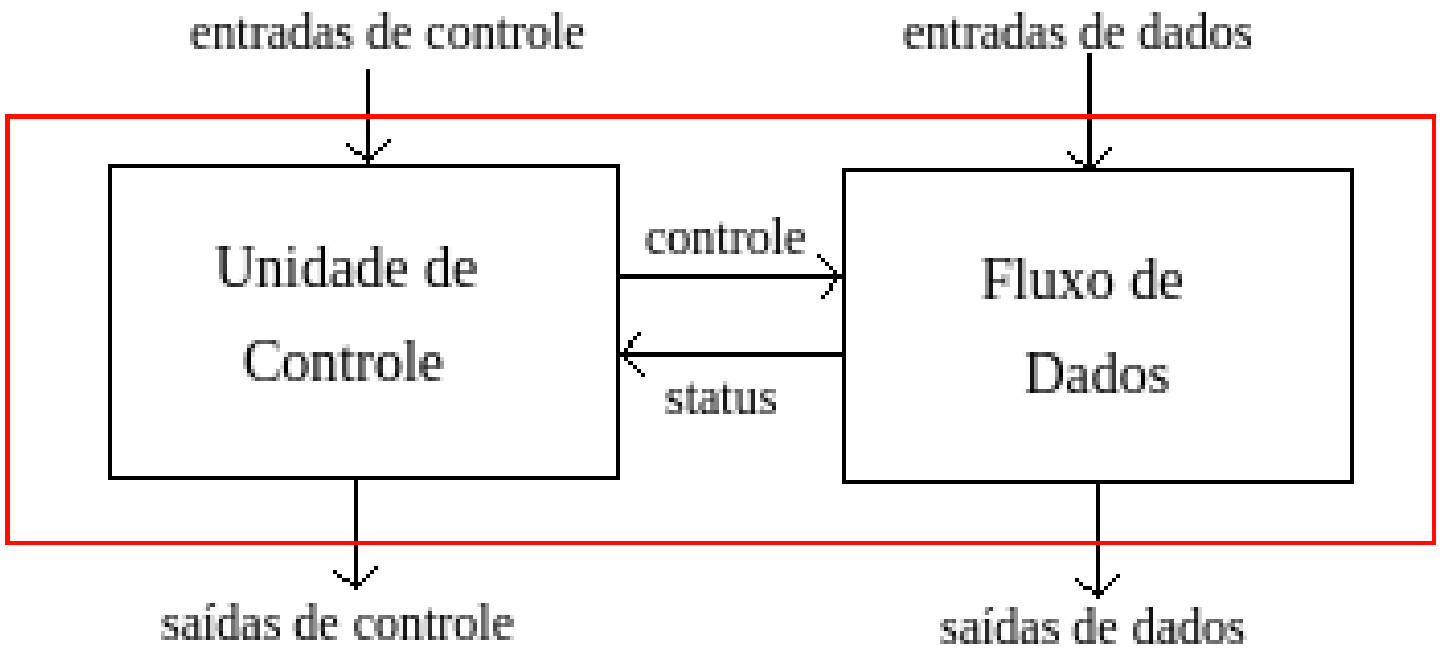
- Circuitos combinatórios e sequenciais
  - Pequenos circuitos: portas lógicas, mapas de Karnaugh, tabelas verdade, somadores, etc;
  - Circuitos maiores: componentes MSI (mux, decodificadores, etc), ULA, circuitos sequenciais, diagrama de transição de estados.
- Contudo, **sistemas mais complexos** precisam de técnicas de projeto mais abstratas e sistemáticas.

# Metodologia de Projeto Estruturado

- Uma metodologia apresenta uma **sequência de atividades** que levam a um desenvolvimento organizado de um sistema digital;
- Conceitos importantes:
  - Divisão do sistema digital em fluxo de dados e unidade de controle;
  - Concepção e organização hierárquica de módulos de projeto;
  - Projeto voltado para a síntese do circuito.

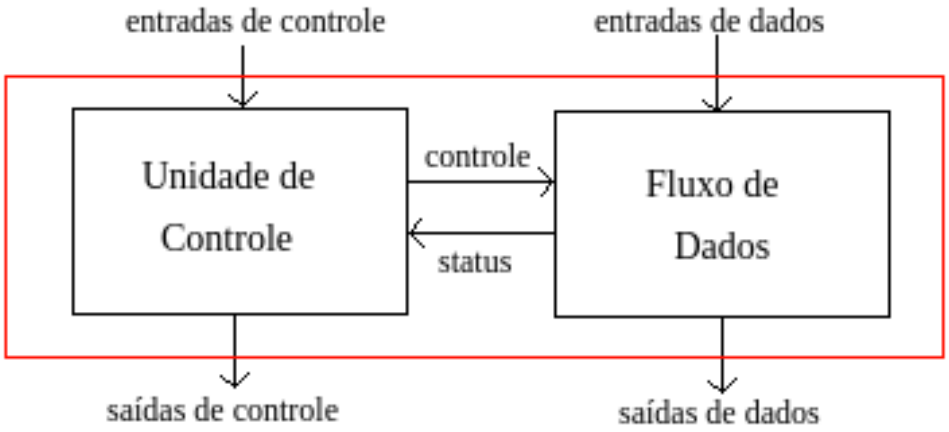
# Projeto de Sistemas Digitais

- SISTEMA DIGITAL =
  Fluxo de Dados + Unidade de Controle



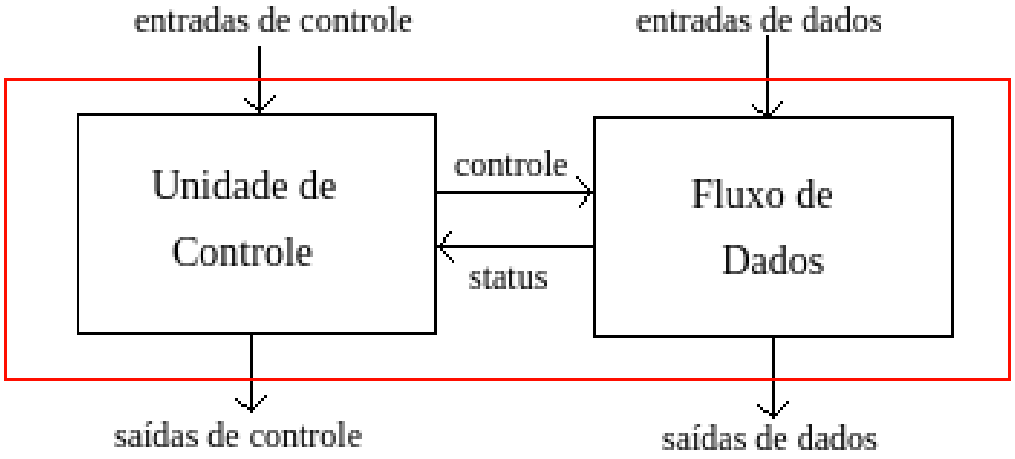
# Projeto de Sistemas Digitais

- **Fluxo de Dados**
  - responsável pelo armazenamento, roteamento, combinação e processamento em geral dos DADOS.
  - composto principalmente por:
    - registradores, contadores, deslocadores;
    - memórias;
    - unidades funcionais gerais (somadores, ULAs, comparadores, etc).
  - recebe COMANDOS da unidade de controle.



# Projeto de Sistemas Digitais

- **Unidade de Controle**
  - Entradas:
    - **COMANDOS externos** para o sistema digital, e
    - Sinais de **STATUS** (**CONDIÇÃO**) vindos do fluxo de dados
  - Saídas:
    - Sinais de CONTROLE para o fluxo de dados;
    - Pode gerar SAÍDAS DE CONTROLE externas (e.g., fim)



# Projeto de Sistemas Digitais

- **Unidade de Controle**
  - responsável pelo **sequenciamento** das operações realizadas pelo sistema digital.
  - Ações principais:
    - Iniciar e terminar operações;
    - Testar condições;
    - Decidir ações futuras;
  - modelo baseado em uma **máquina de estados**.

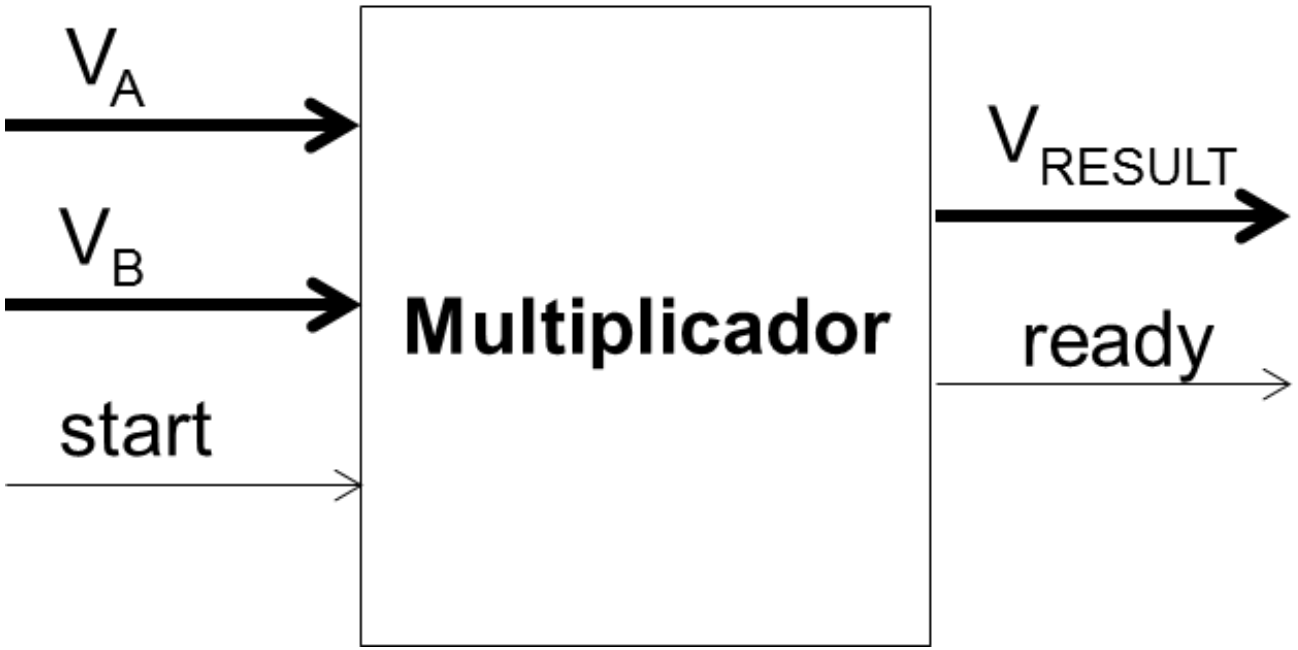
# Metodologia de Projeto

- Passos do método
  1. Obter uma **descrição verbal** do circuito;
  2. Desenvolver um **pseudocódigo** do algoritmo;
  3. Elaborar um **diagrama ASM de alto nível**;
  4. Selecionar **elementos do fluxo de dados** para as operações e conectá-los;
  5. Identificar **sinais de *status* e de controle** dos elementos fluxo de dados;
  6. Especificar a **FSM da unidade de controle**;
  7. Verificar conexão do fluxo de dados e da unidade de controle.

# Projeto Exemplo

# Projeto-Exemplo: Multiplicador

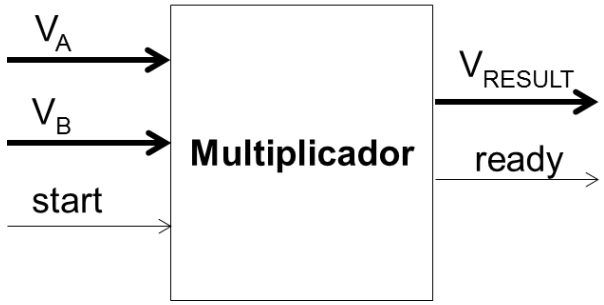
- Especificação: multiplicador




Fonte: **W.V. Ruggiero & C.B. Margi .** Capítulo 1: Metodologia de Projeto Estruturado para Sistemas Digitais. Apostila de PCS2022.

# Projeto-Exemplo: Multiplicador

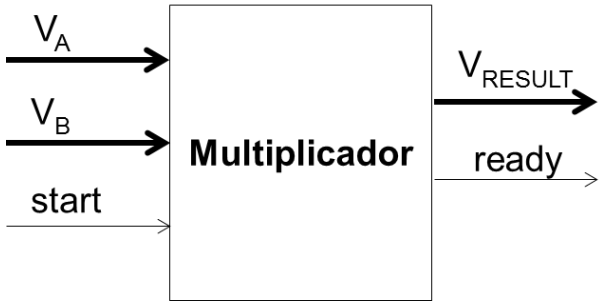
1. Especificação: multiplicador (descrição textual ou verbal)




Para exemplificar, considere o projeto de um circuito multiplicador de dois números de 4 bits. O circuito recebe dois números através das vias $V_A$ e $V_B$, ambas com 4 bits de largura, e começa a multiplicar esses dois números quando o sinal de START (via de 1 bit) vai para 1. Depois que a multiplicação termina (não se sabe a priori quanto tempo vai levar), o circuito multiplicador coloca o resultado na via de 8 bits $V_{RESULT}$ e coloca o sinal READY em 1.

# Projeto-Exemplo: Multiplicador

## 2. Escolha do algoritmo de multiplicação




Definimos que o multiplicador utilizará o algoritmo de multiplicação por somas sucessivas. De maneira simples, podemos descrever o algoritmo com a seguinte sequência de passos:

- Após Start: carregar $V_A$ e $V_B$, e inicializar resultado em zero
- Enquanto multiplicador for diferente de 0
  - Somar o valor do multiplicando ao resultado
  - Subtrair 1 do multiplicador
- Ready=1 e retorna resultado

# Projeto-Exemplo: Multiplicador

## 2. Escolha do algoritmo de multiplicação

Definimos que o multiplicador utilizará o algoritmo de multiplicação por somas sucessivas.

Exemplo:

**5x3 = 3 + 3 + 3 + 3 + 3**

# Projeto-Exemplo: Multiplicador

## 2. Pseudocódigo do algoritmo de multiplicação

- Após Start: carregar $V_A$ e $V_B$, e inicializar resultado em zero
- Enquanto multiplicador for diferente de 0
  - Somar o valor do multiplicando ao resultado
  - Subtrair 1 do multiplicador
- Ready=1 e retorna resultado

- Necessidade de sinais de controle para início e fim da operação; entradas carregadas em registradores

# Projeto-Exemplo: Multiplicador

Vamos detalhar um pouco o algoritmo:

Portas de entrada:
Va, Vb e start

Portas de saída:
Vresult e ready

Variáveis:
ra, rb e result

- while (start==0);
- ra = Va;
- rb = Vb;
- result=0;
- while (rb!=0)
  - result = result + ra;
  - rb = rb - 1;
- ready = 1;
- Vresult = resultado;

# Projeto-Exemplo: Multiplicador

Em algum momento, precisamos zerar a saída“ready”

Portas de entrada:
Va, Vb e start

Portas de saída:
Vresult e ready

Variáveis:
ra, rb e result

- while (start==0) ready=0;
- ra = Va;
- rb = Vb;
- result=0;
- while (rb!=0)
  - result = result + ra;
  - rb = rb - 1;
- ready = 1;
- Vresult = resultado;

# Projeto-Exemplo: Multiplicador

## 3. Diagrama ASM de alto nível

- No fim, precisamos de uma máquina de estados que controle um algoritmo no fluxo de dados.

- Um algoritmo pode ser representado por um diagrama ASM *de alto nível*:

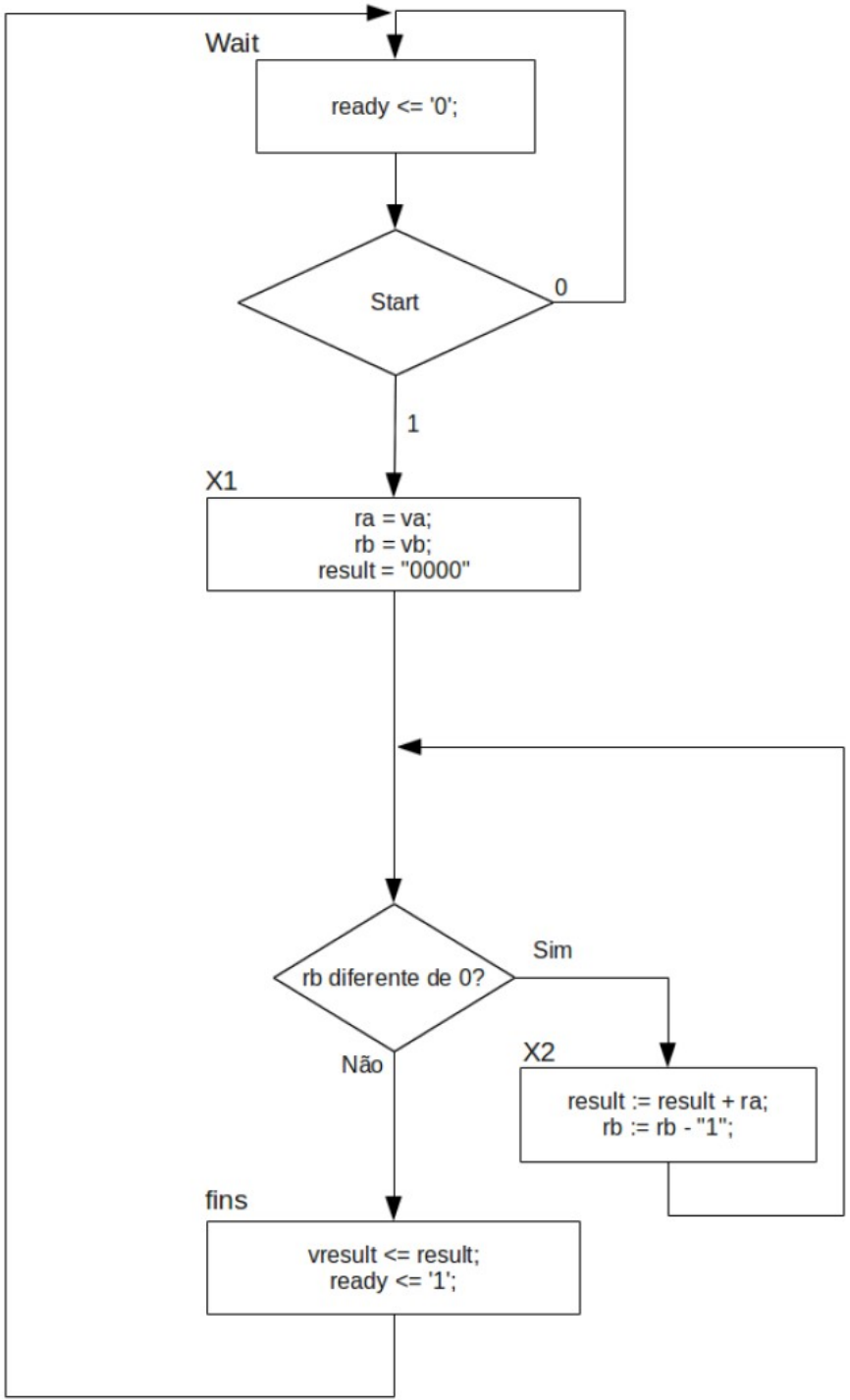
    - Operações sobre os dados (variáveis) dentro dos estados (saídas de Moore);

    - Condições sobre os dados dentro das caixas de decisão (para IFs e laços) ;

- Esse diagrama nos dá o “esqueleto” da MEF da Unidade de Controle, que será detalhada mais tarde.

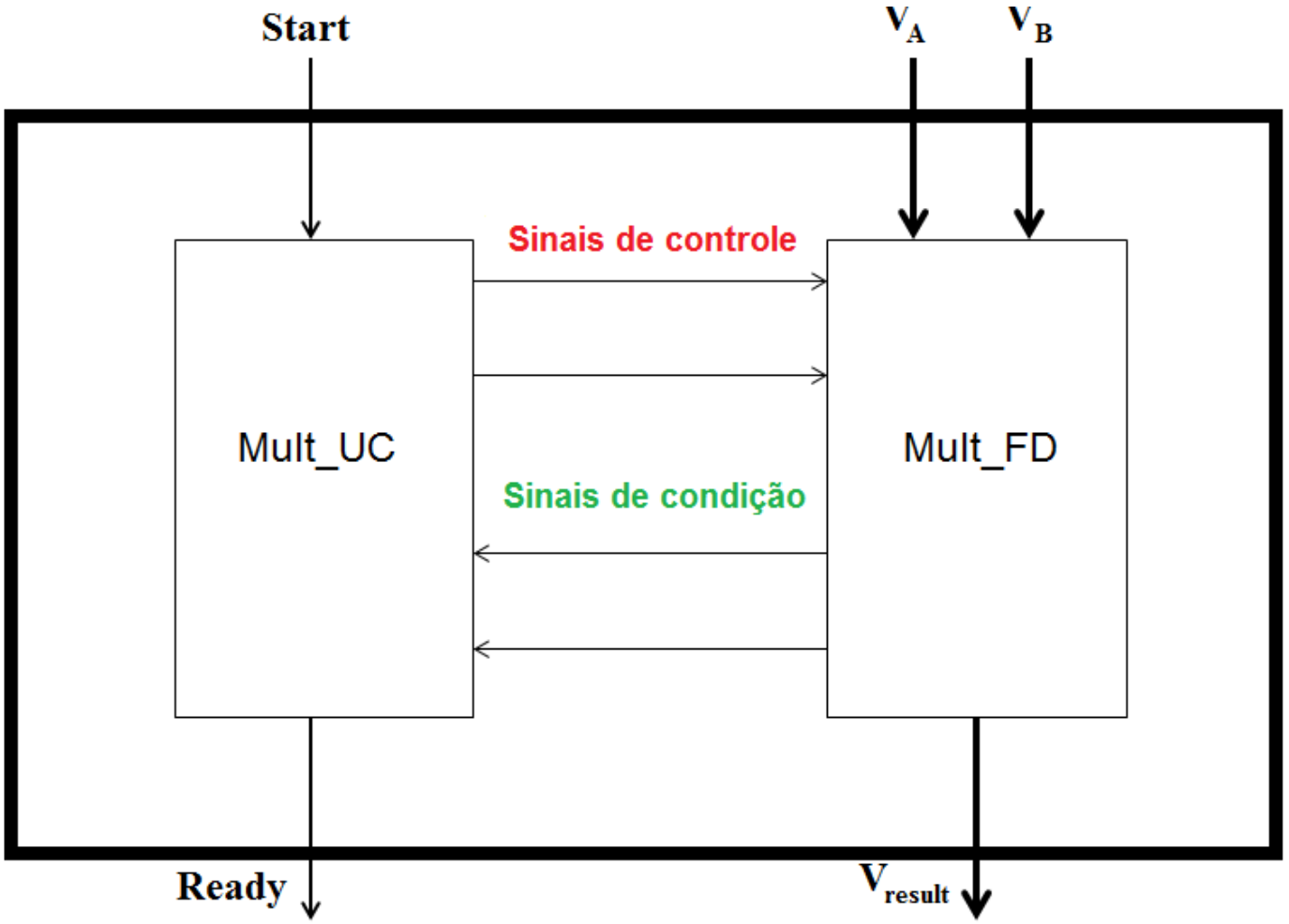
# Multiplicador

- while (start==0) ready=0;
- ra = Va;
- rb = Vb;
- result=0;
- while (rb!=0)
  - result = result + ra;
  - rb = rb - 1;
- ready = 1;
- Vresult = resultado;



# Multiplicador

## 4. Elementos do fluxo de dados (FD)



# Multiplicador

## 4. Elementos do fluxo de dados (FD)

- Alocamos componentes para armazenar variáveis
  - ex: registradores, contadores, memórias
- Alocamos componentes para operar dados e gerar os sinais de condição:
  - ex: ULA, somadores, subtratores, deslocadores, comparadores
- Conectamos os componentes:
  - fios, multiplexadores, barramentos

# Multiplicador

- 4. Elementos do fluxo de dados (FD)

4 bits

8 bits

- Após Start: carregar $V_A$ e $V_B$, e inicializar resultado em zero
- Enquanto o multiplicador não for igual a 0
  - Somar multiplicando ao resultado
  - Subtrair 1 do multiplicador

1. Registradores
(Ra, Rb: 4 bits ; result: 8 bits)

2. Detector de 0

3. Somador

4. Subtrator de 1

result: sinal de reset

5a. interconexões: Ra <= Va
Rb <= Vb

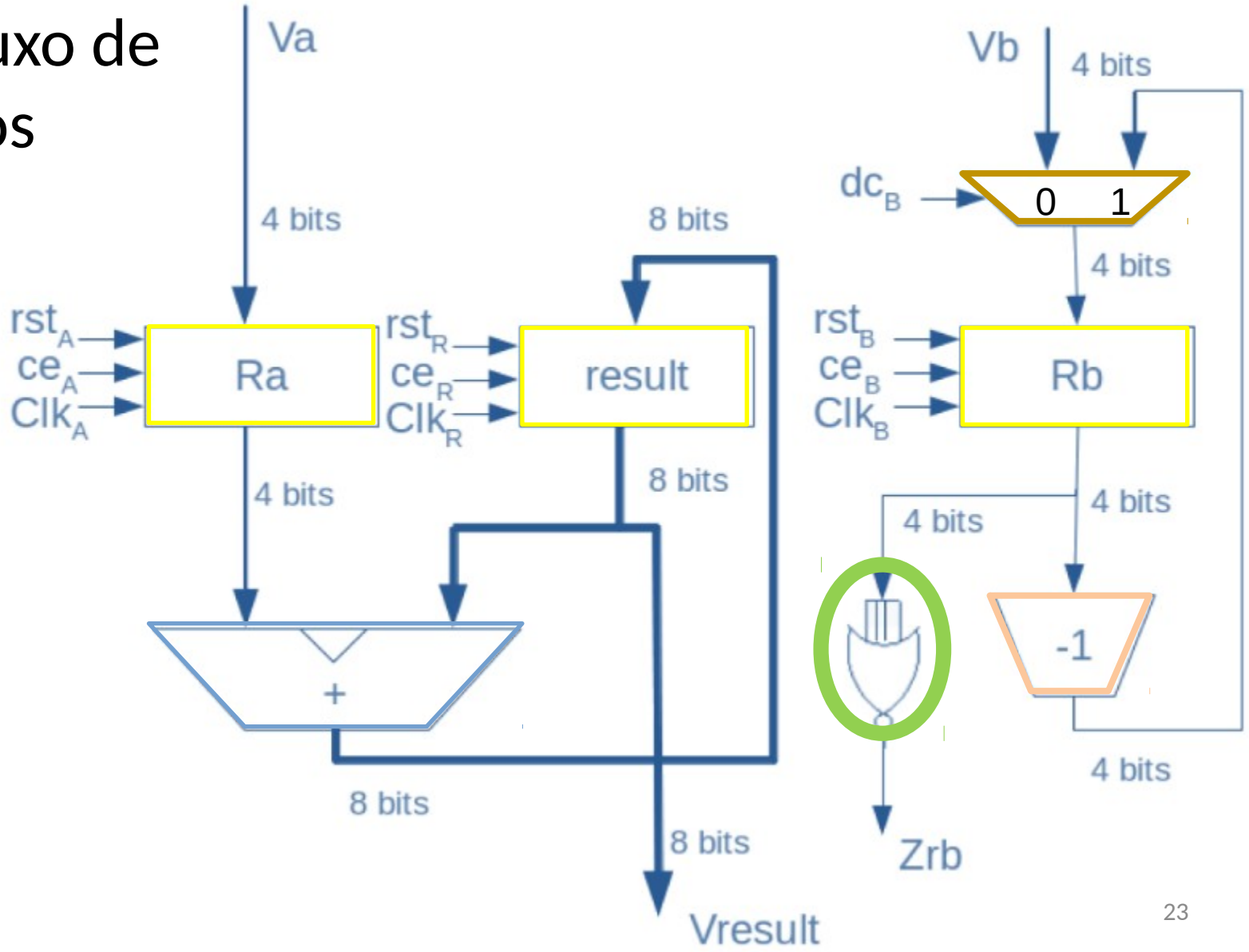
5b. interconexões: result <= somador
somador <= (Ra, result)

5c. interconexões: Rb <= subtrator
subtrator <= (Rb, 1)

Mux em Rb

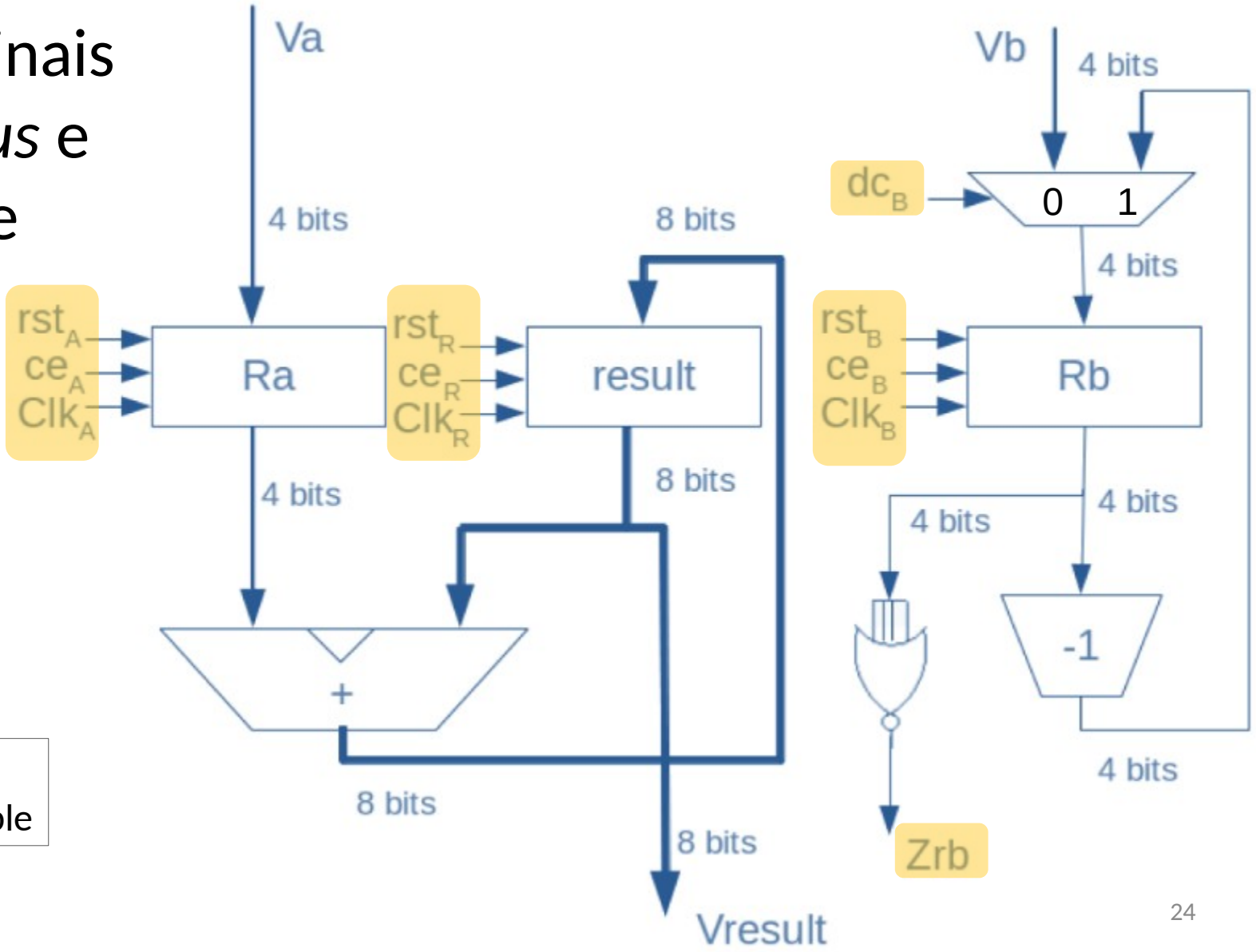
# Multiplicador

- 4. Fluxo de Dados



# Multiplicador

5. FD: sinais de *status* e controle

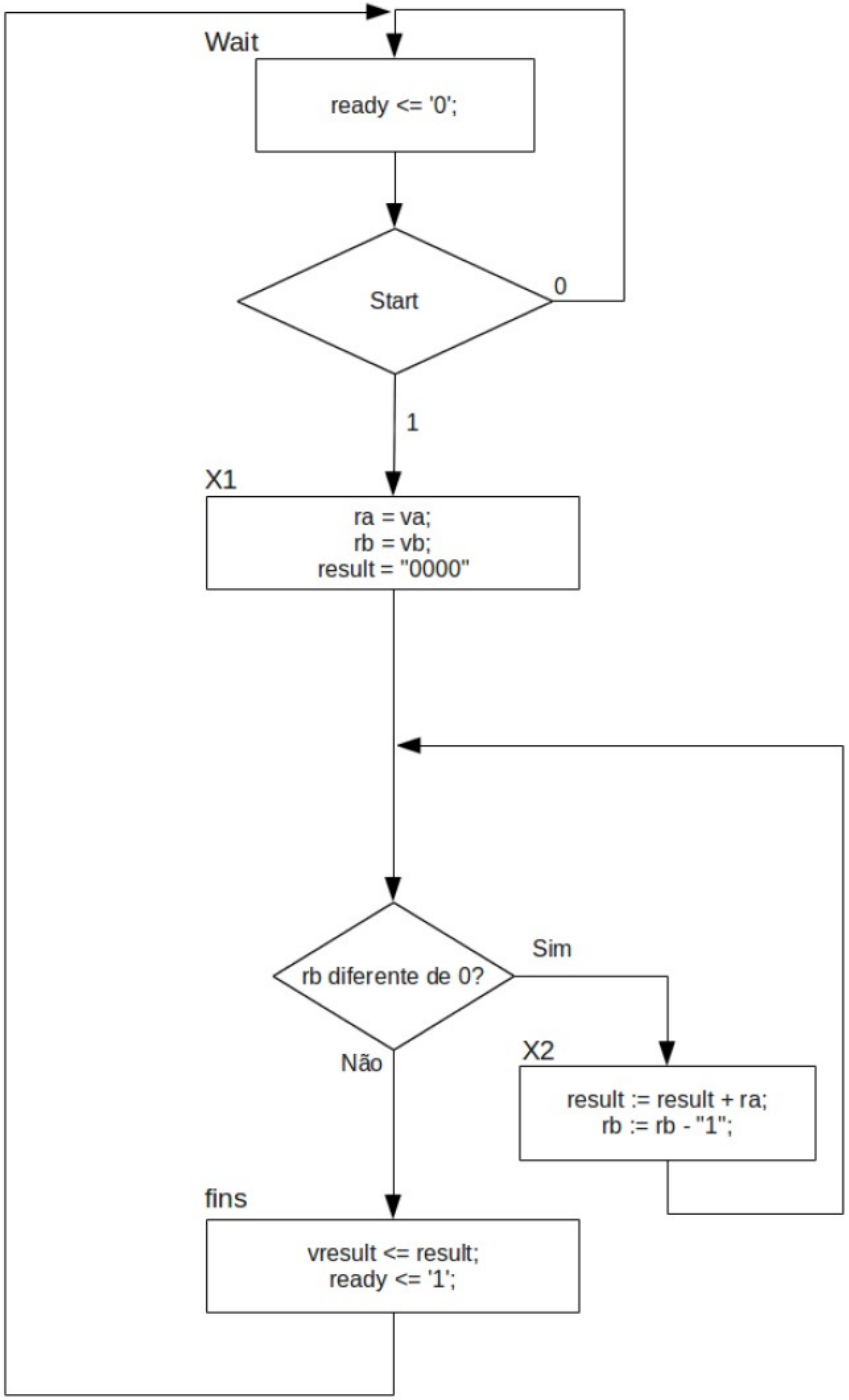


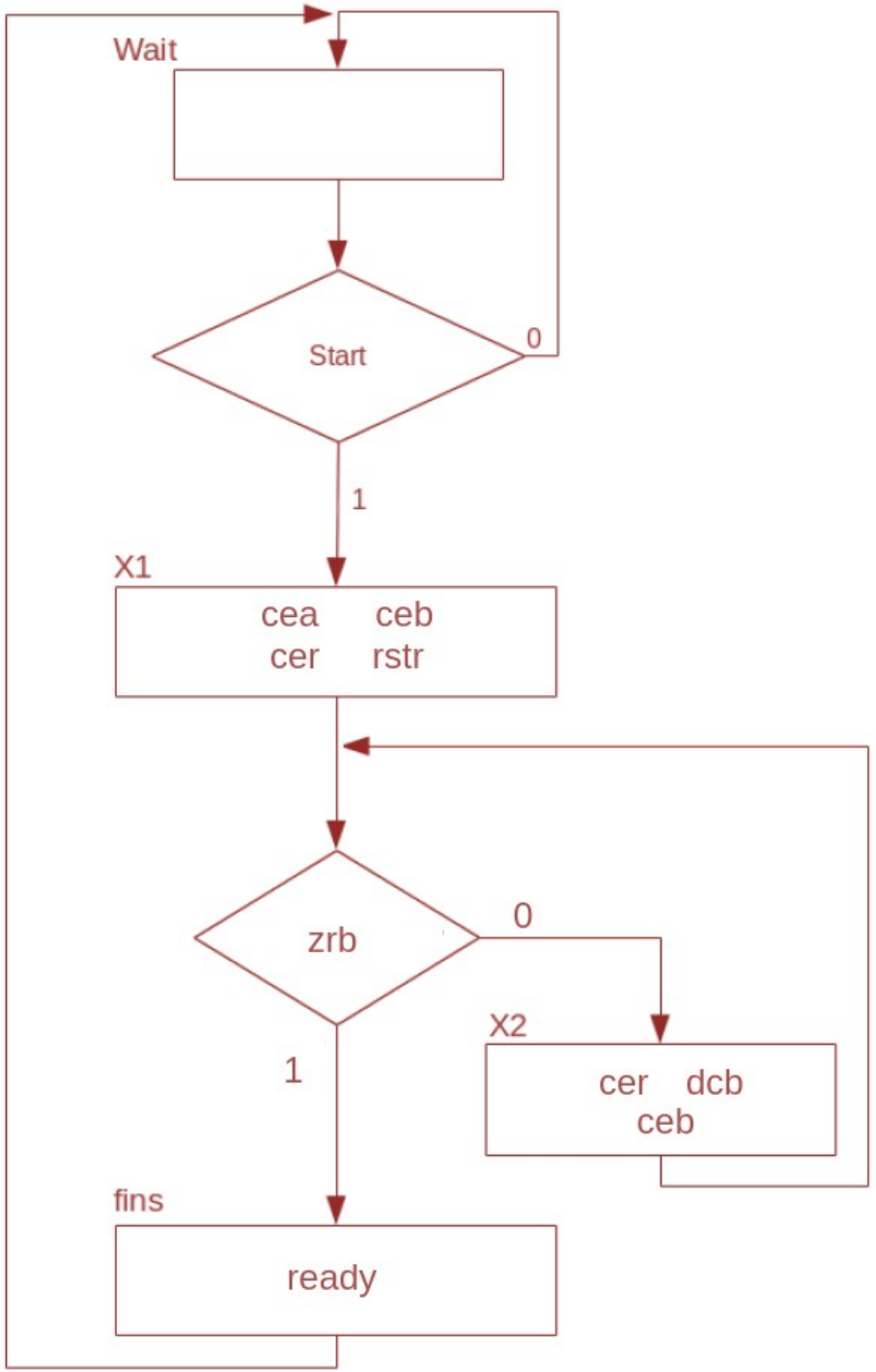

rst: reset
ce: chip enable

# Multiplicador

## 6. Detalhar ASM da unidade de controle

- A Unidade de controle pode ser projetada a partir do diagrama ASM de alto nível:
  - Substitua cada operação sobre os dados pelos sinais de controle que devem ser ativados
  - Substitua condições sobre os dados pelos sinais de status correspondentes

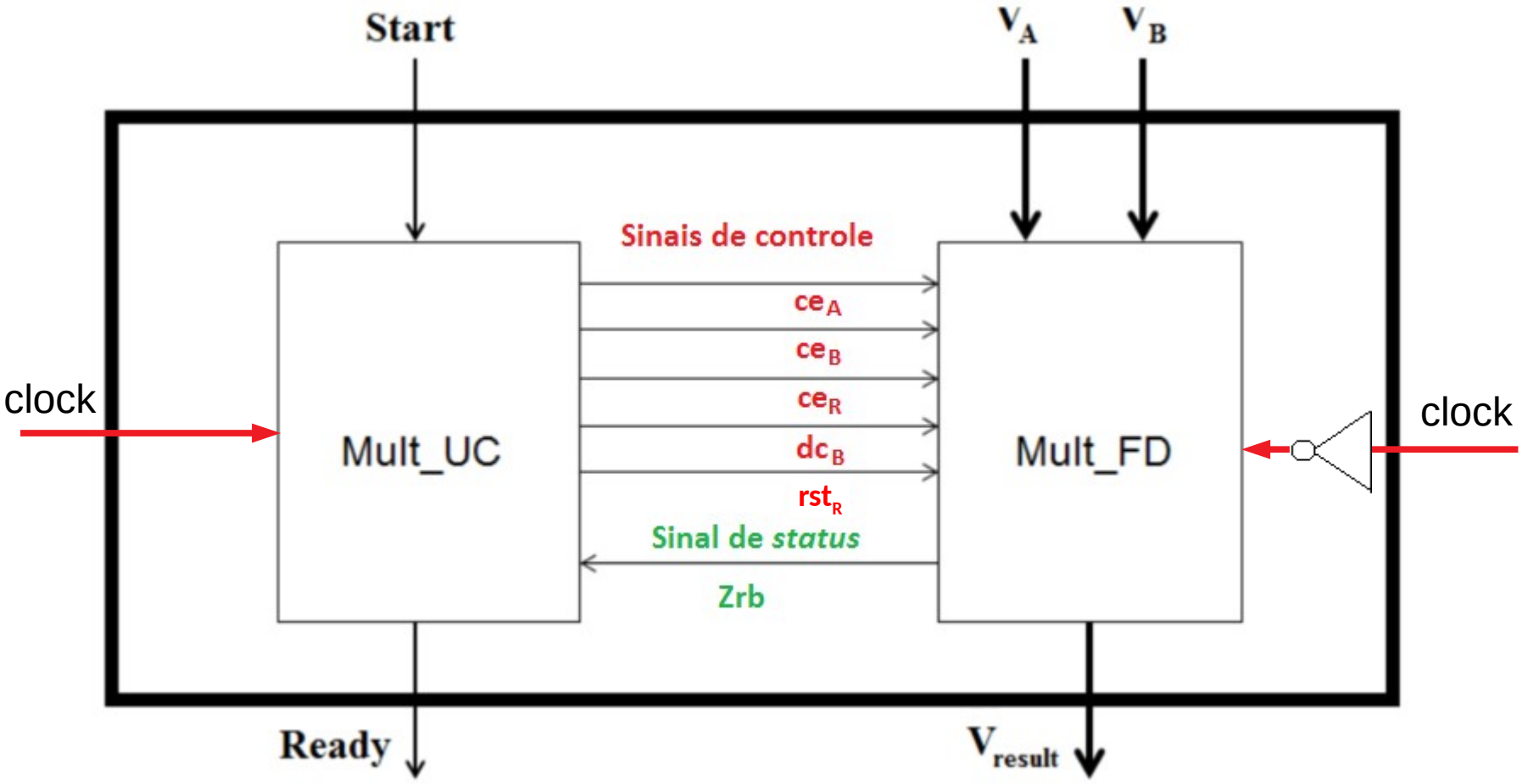

Wait
ready <= '0';
Start
0
1
X1
ra = va;
rb = vb;
result = "0000"
rb diferente de 0?
Sim
Não
X2
result := result + ra;
rb := rb - "1";
fins
vresult <= result;
ready <= '1';


Wait
Start
0
1
X1
cea ceb
cer rstr
zrb
0
1
X2
cer dcb
ceb
fins
ready

# Multiplicador

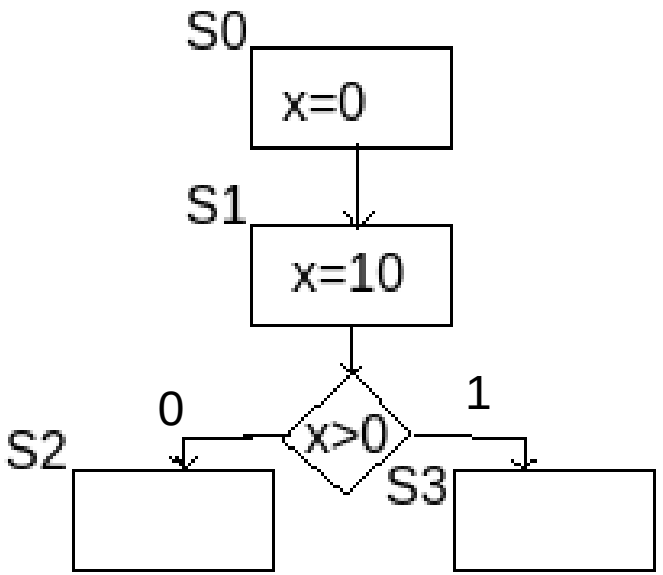
## 7. Conexão de FD e UC

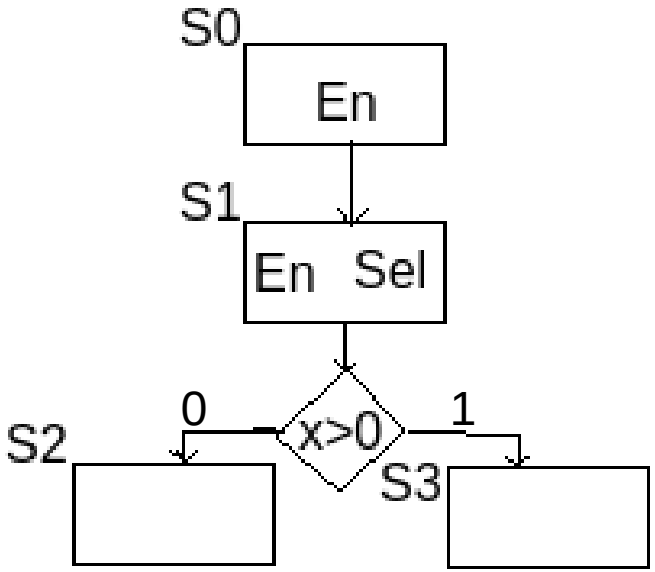
- Clock do FD é a negação do clock da UC!

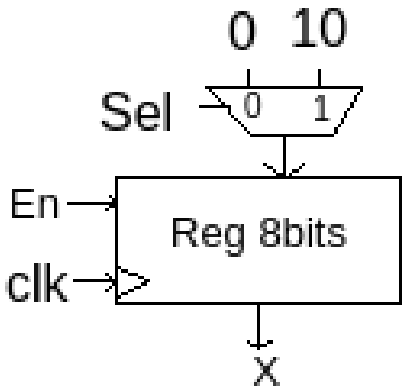


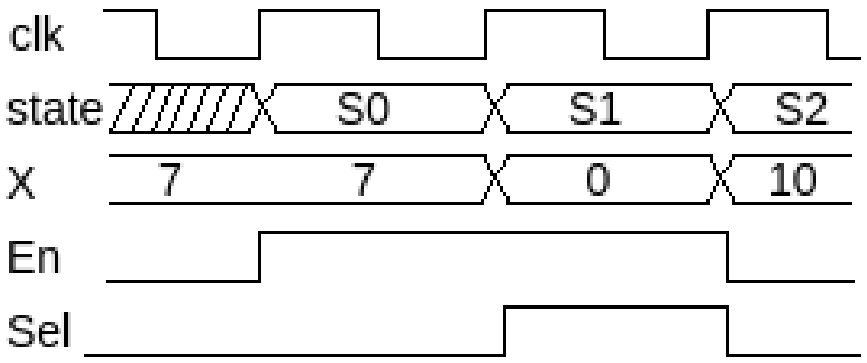
# Por que inverter o clock no FD?

Vejamos um exemplo simples onde não invertemos...



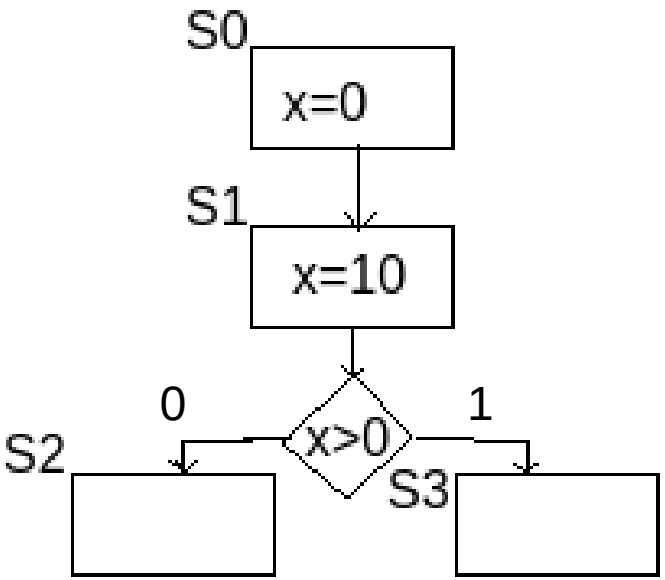



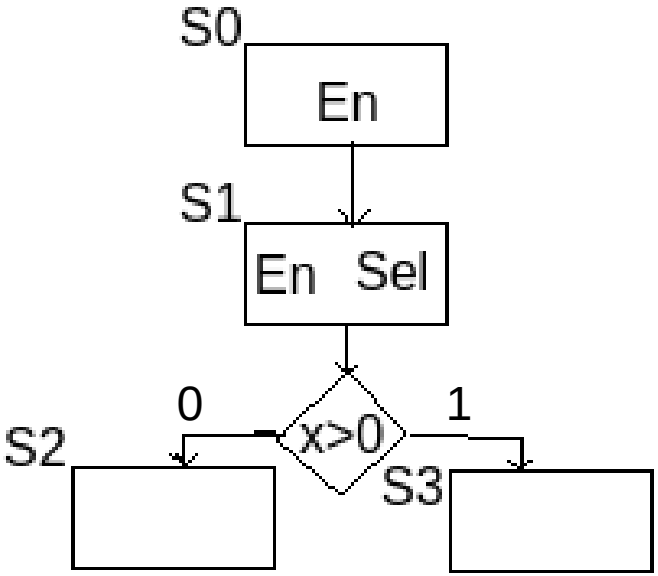



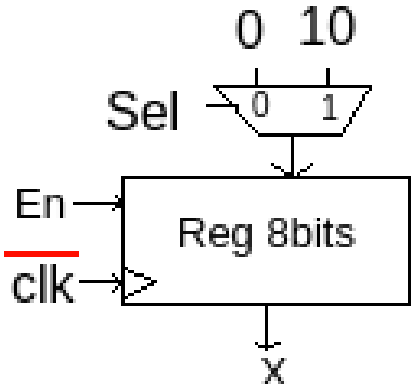



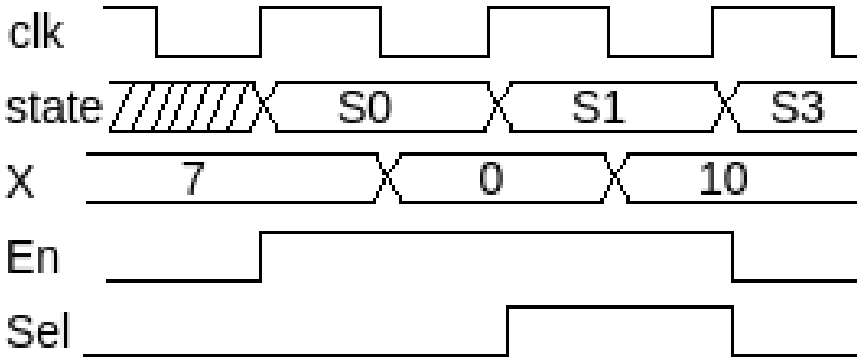
# Por que inverter o clock no FD?

Comparando com o clock invertido no FD...



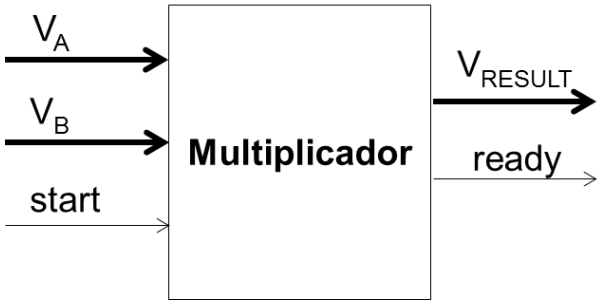
# Multiplicador

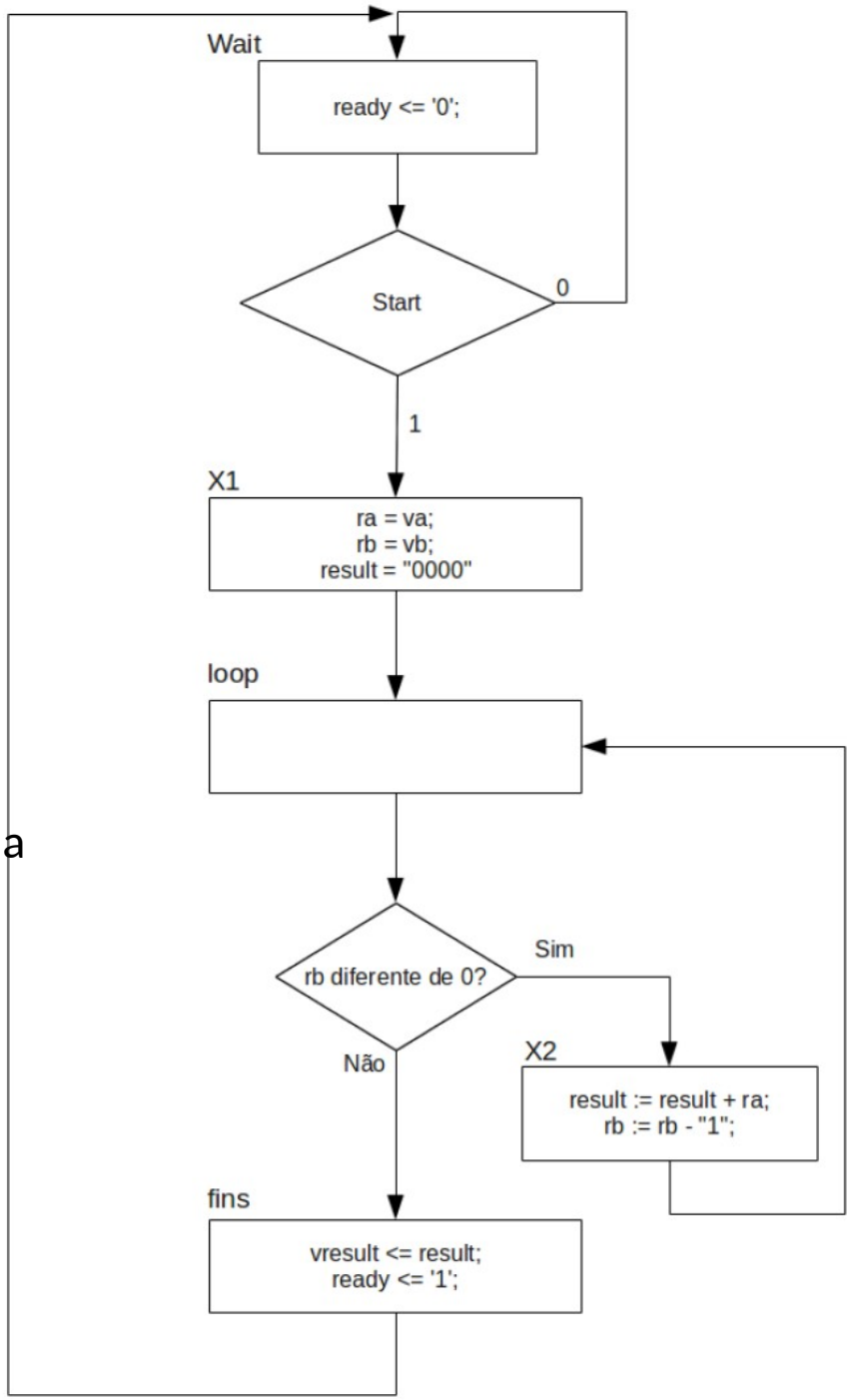
- Podemos assumir que o clock será o mesmo para UC e FD
- Nesse caso, precisaríamos levar em conta o atraso de um estado na atualização dos valores dos registradores
- Tipicamente, isso leva a um diagrama ASM com um número maior de estados
- Podemos fazer isso com o Multiplicador, no que chamaremos de Multiplicador v2

# Multiplicador

## 3. Diagrama ASM de alto nível (v2)

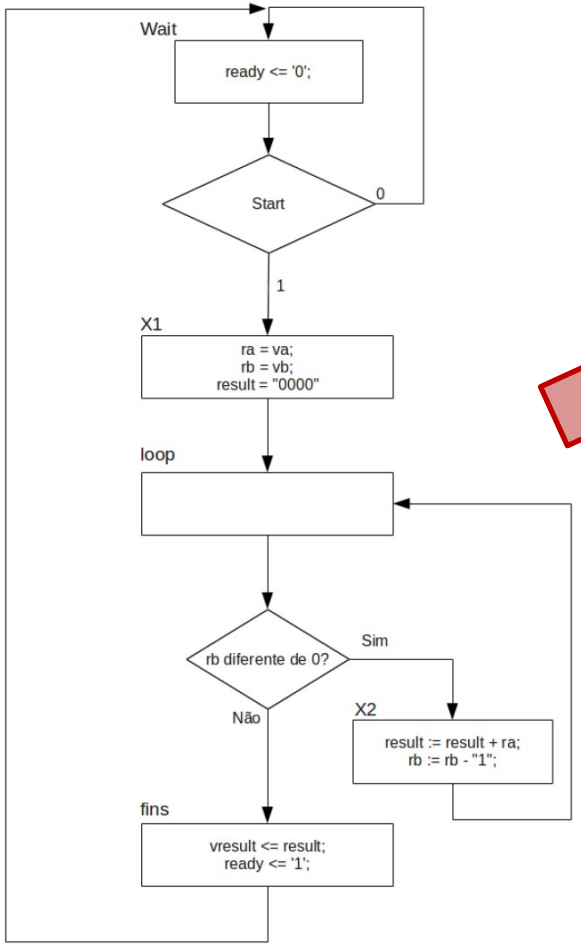
- Quando a MEF entra em X1 ou X2, o enable do registrador Rb é ativado
- Isso faz com Rb seja atualizado na próxima borda de subida do clock no FD
- Se a transição da máquina acontecer nesta mesma borda, testaremos o valor antigo de Rb
- Precisamos do estado *dummie* loop

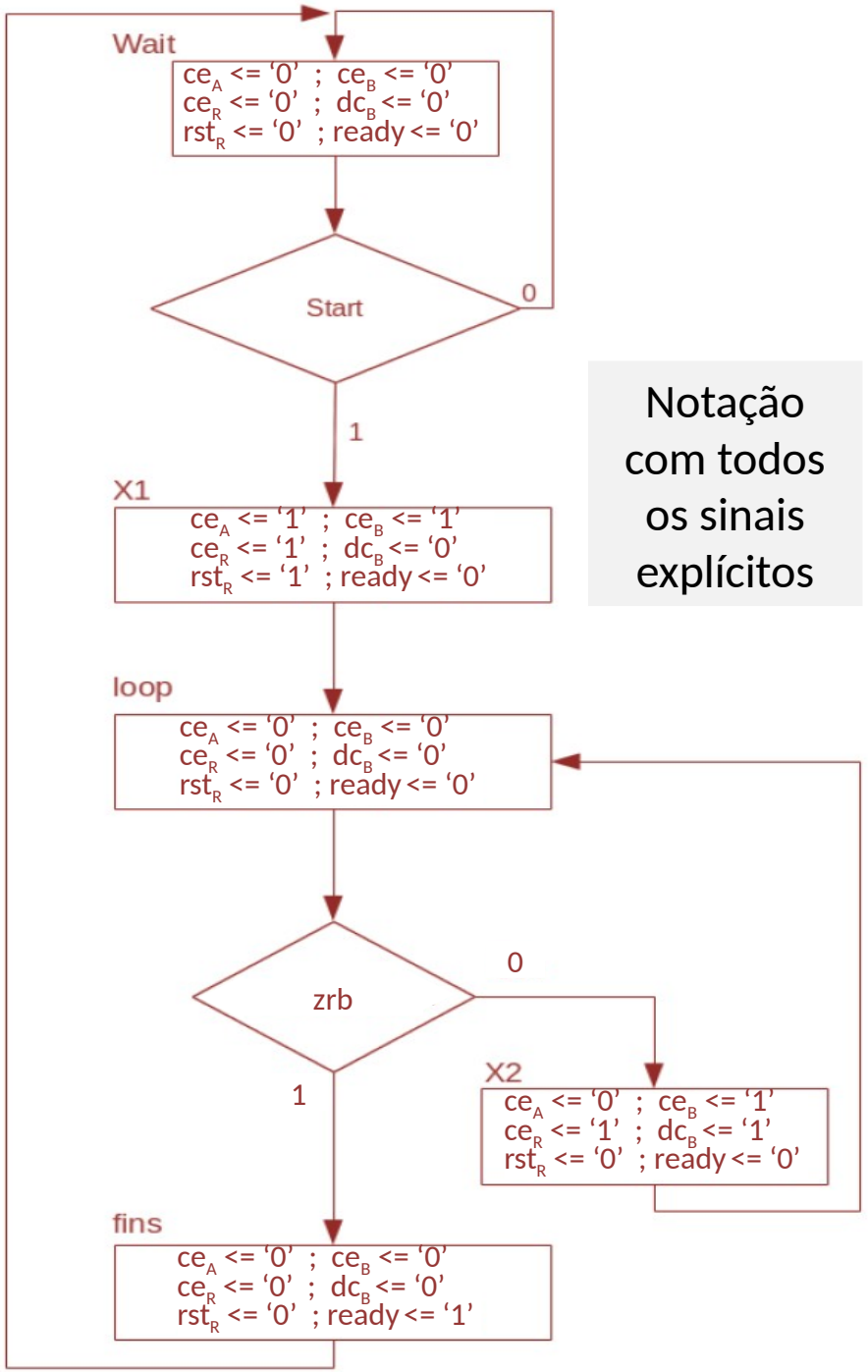


# Multiplicador

## 6. Detalhar ASM da unidade de controle (v2)



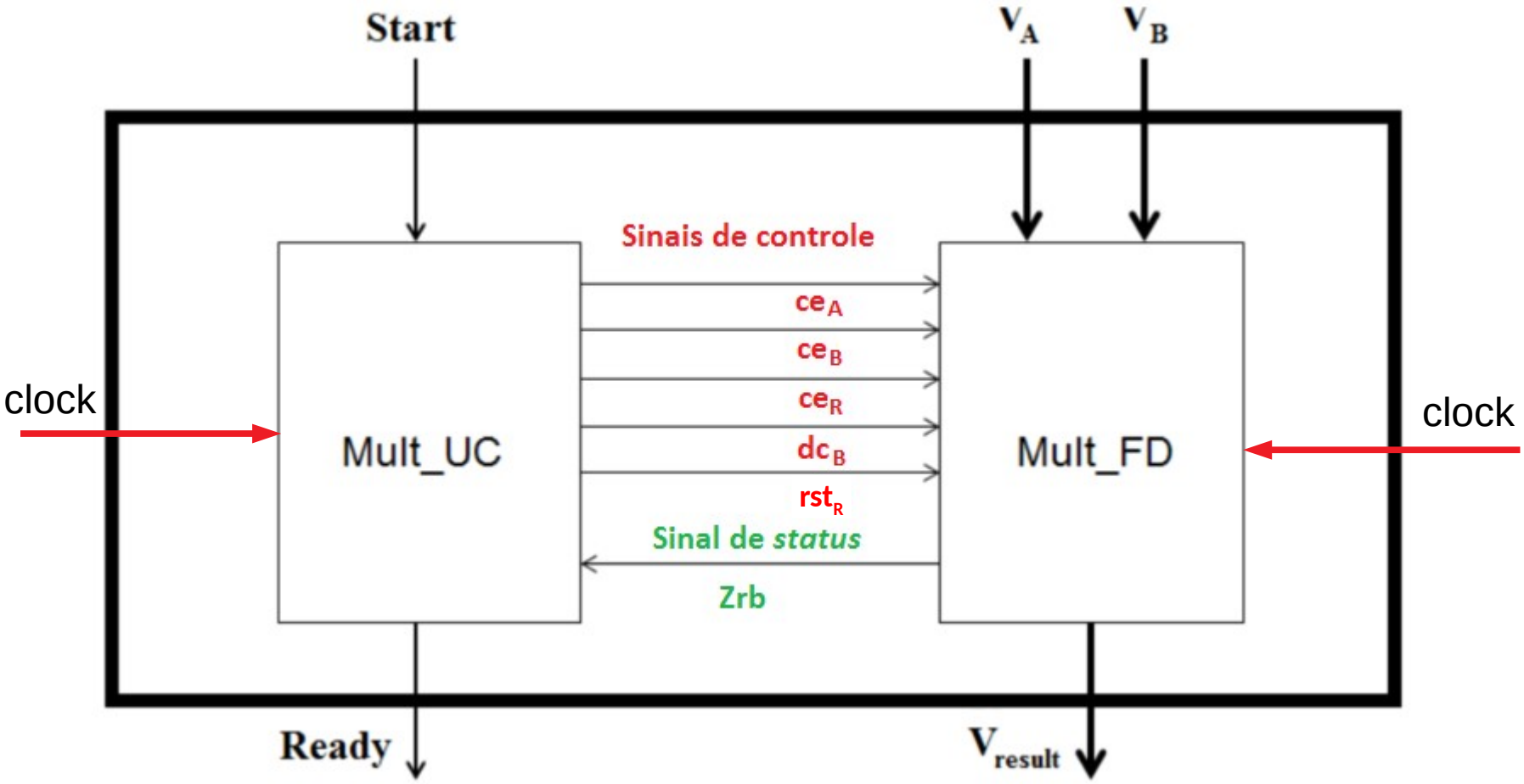




Notação com todos os sinais explícitos

# Multiplicador

## 7. Conexão de FD e UC (v2)

- Clock do FD é igual ao clock da UC!



# EXERCÍCIO (RESOLVIDO): RAIZ INTEIRA

# Raiz Inteira

- Projetar circuito de cálculo da raiz quadrada, baseado no algoritmo abaixo.

```
unsigned long sqrt(unsigned long a){
     unsigned long square = 1;
     unsigned long delta = 3;
     while(square <= a){
        square += delta;
        delta += 2;
     }
     return (delta/2 - 1);
}
```

DICA: veja vídeo em `https://www.youtube.com/watch?v=38md9YNPee0`

# Raiz Inteira

- Exemplo de execução do algoritmo

sqrt(51)= ?

```
unsigned long sqrt(unsigned long a){
     unsigned long square = 1;
     unsigned long delta = 3;
     while(square <= a){
        square += delta;
        delta += 2;
     }
     return (delta/2 - 1);
}
```

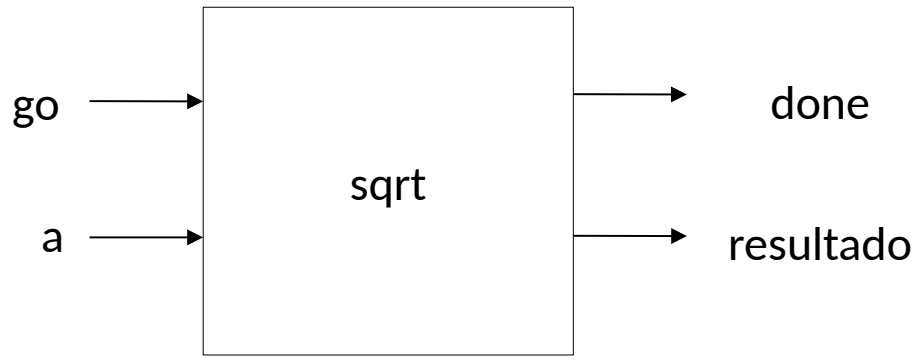
| a, | square, | delta, | delta/2-1 |
|---|---|---|---|
| 51, | 1, | 3, | 0 |
| 51, | 4, | 5, | 1 |
| 51, | 9, | 7, | 2 |
| 51, | 16, | 9, | 3 |
| 51, | 25, | 11, | 4 |
| 51, | 36, | 13, | 5 |
| 51, | 49, | 15, | 6 |
| 51, | 64, | 17, | 7 |

sqrt(51)=7

# Raiz inteira

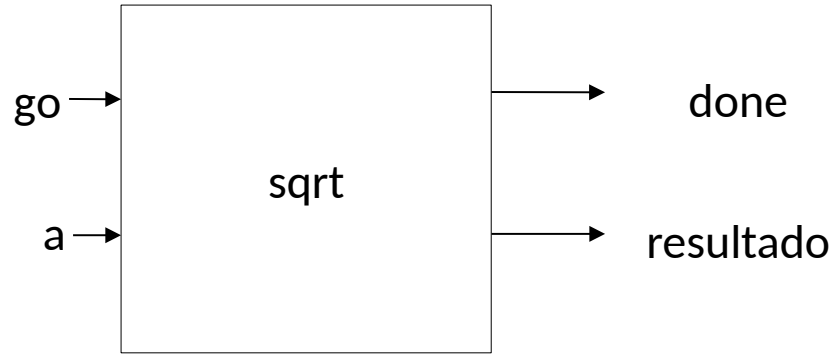
- Pseudocódigo e ASM de alto nível
  - Comece o projeto com a elaboração de um diagrama ASM de alto nível que modela o funcionamento do circuito;
  - Considere um sinal de entrada *go* que inicia o cálculo e uma saída *done* que avisa do seu término;
  - *Solução nos próximos slides...*

```
unsigned long sqrt(unsigned long a){
     unsigned long square = 1;
     unsigned long delta = 3;
     while(square <= a){
        square += delta;
        delta += 2;
     }
     return (delta/2 - 1);
}
```

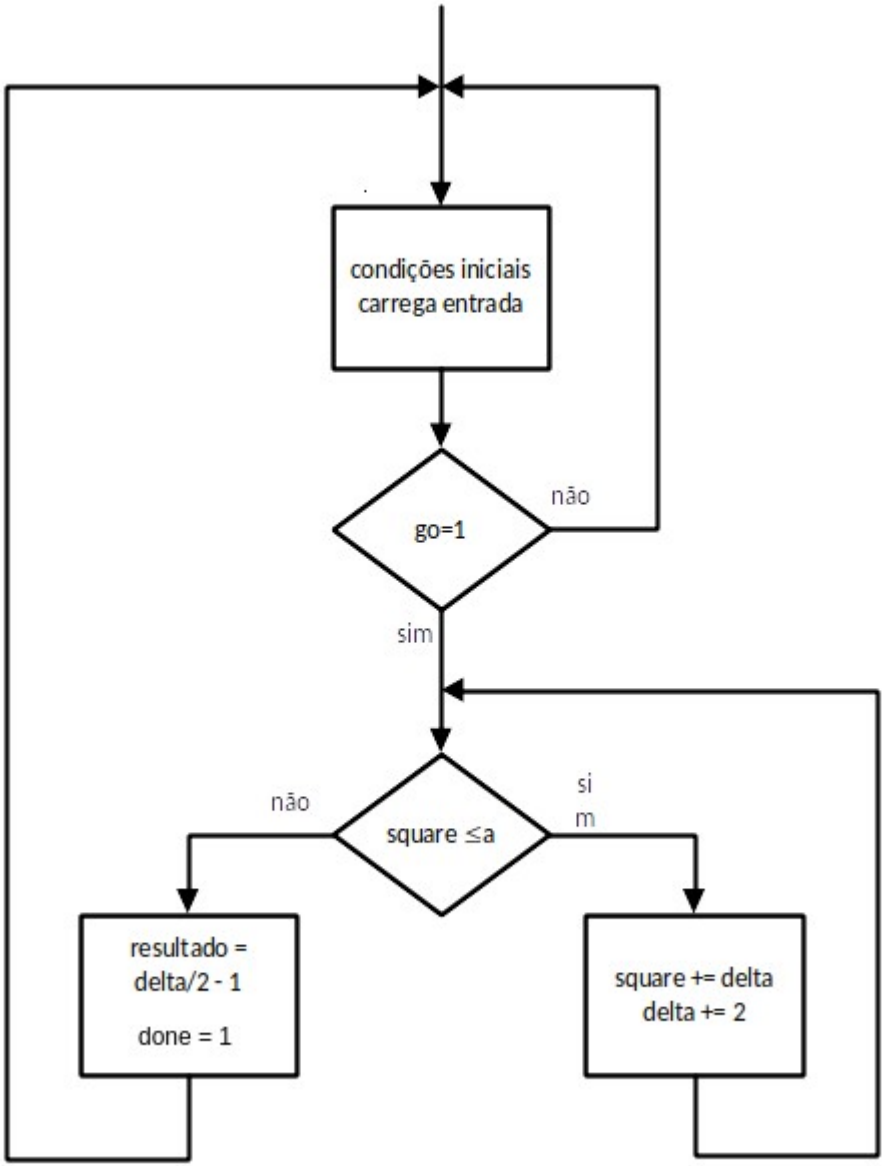


# Raiz inteira

- ASM de alto nível




```
unsigned long sqrt(unsigned long a){
     unsigned long square = 1;
     unsigned long delta = 3;
     while(square <= a){
        square += delta;
        delta += 2;
     }
     return (delta/2 - 1);
}
```



# Raiz Inteira

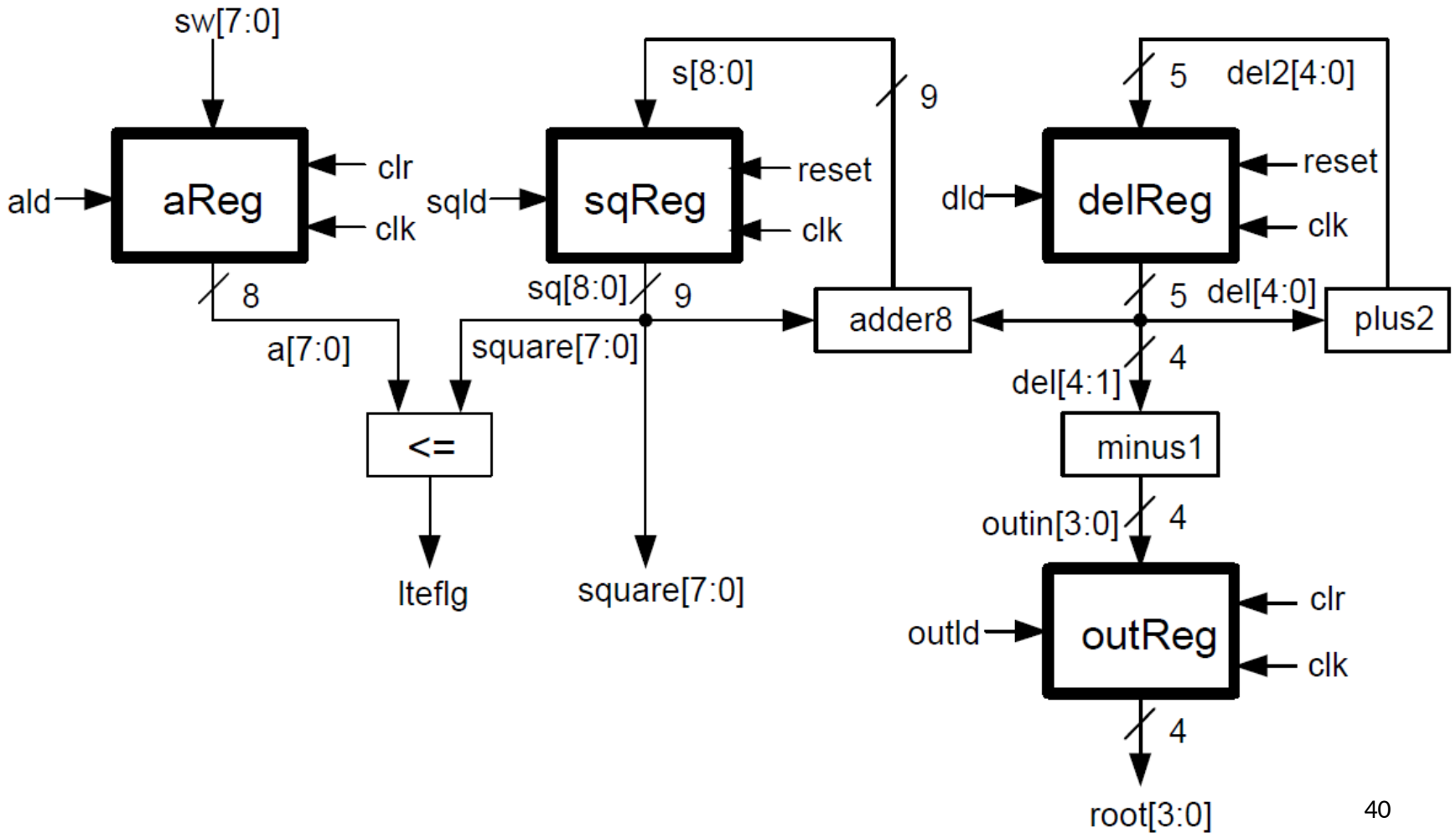
- Fluxo de dados
  - Quais os elementos do fluxo de dados?

```
unsigned long sqrt(unsigned long a){
     unsigned long square = 1;
     unsigned long delta = 3;
     while(square <= a){
        square += delta;
        delta += 2;
     }
     return (delta/2 - 1);
}
```

# Raiz Inteira

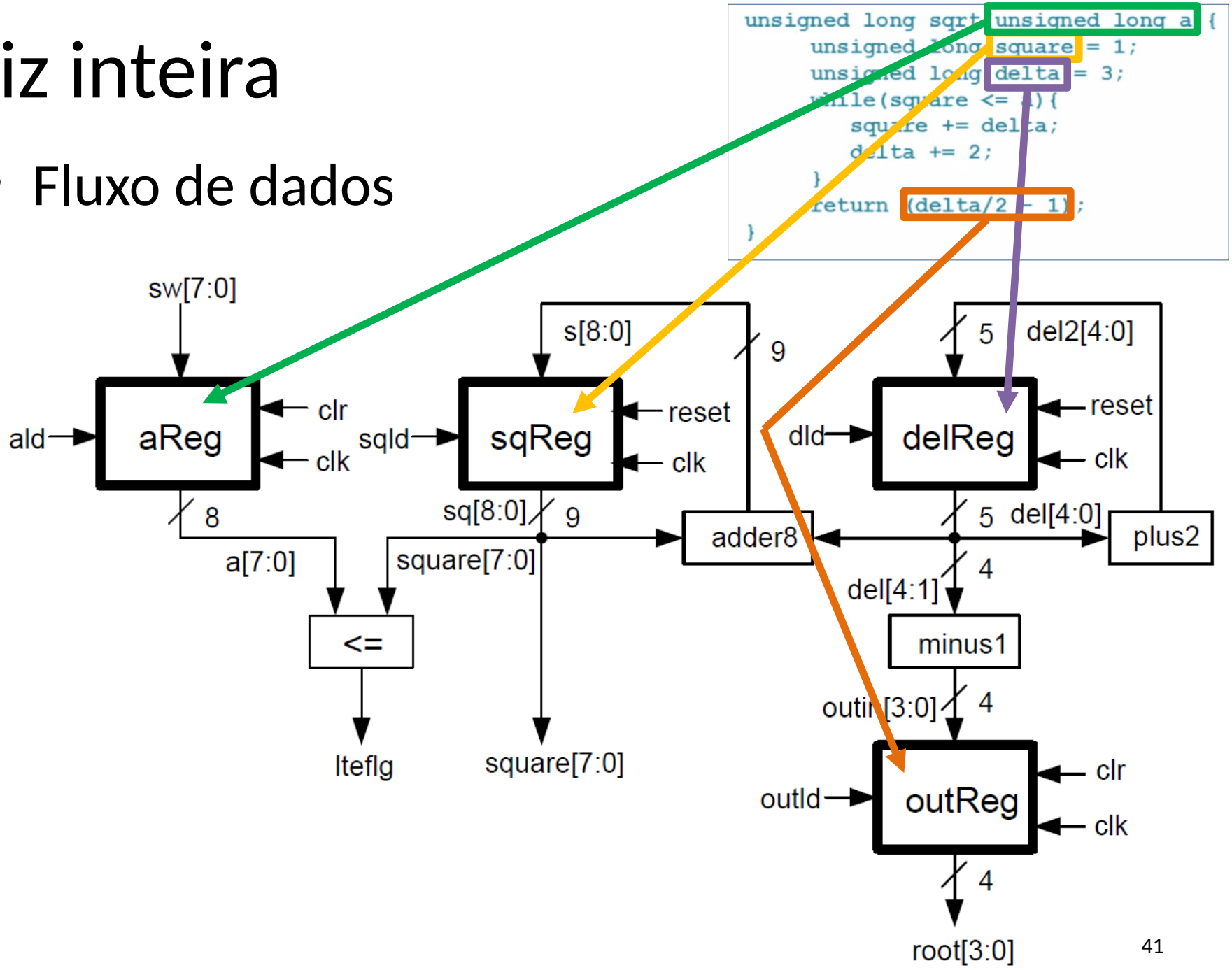
- Fluxo de Dados

```
unsigned long sqrt(unsigned long a){
     unsigned long square = 1;
     unsigned long delta = 3;
     while(square <= a){
        square += delta;
        delta += 2;
     }
     return (delta/2 - 1);
}
```



# Raiz inteira

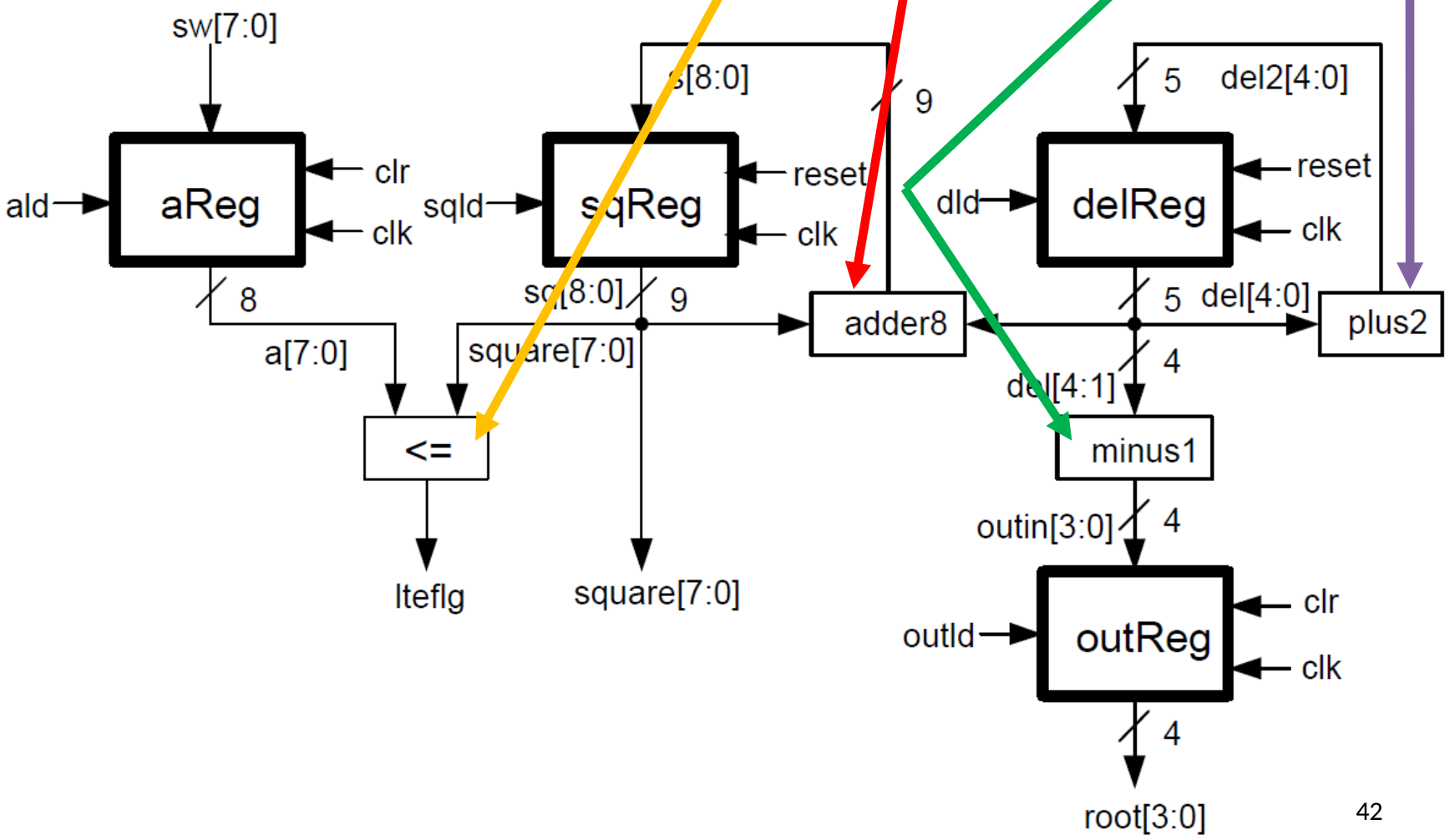
- Fluxo de dados



# Raiz inteira

- Fluxo de dados

```
unsigned long sqrt(unsigned long a){
    unsigned long square = 1;
    unsigned long delta = 3;
    while (square <= a) {
        square += delta;
        delta += 2
    }
    return (delta/2 - 1);
}
```



# Raiz inteira

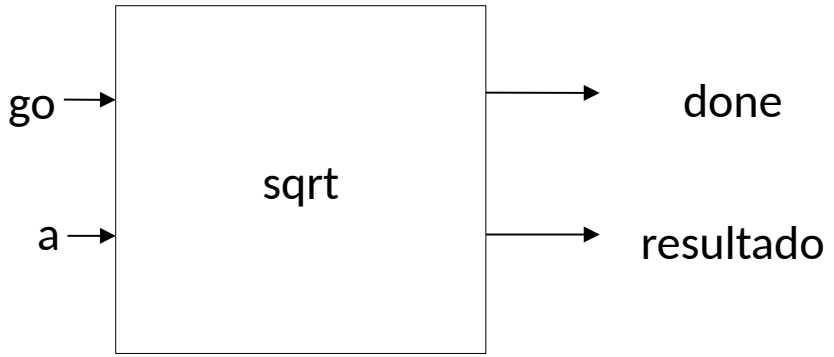
- Fluxo de dados
  - Como inicializar, com valores adequados, as variáveis `square` e `delta`?
    - Em VHDL, podemos definir o valor armazenado por um registrador quando *resetado*

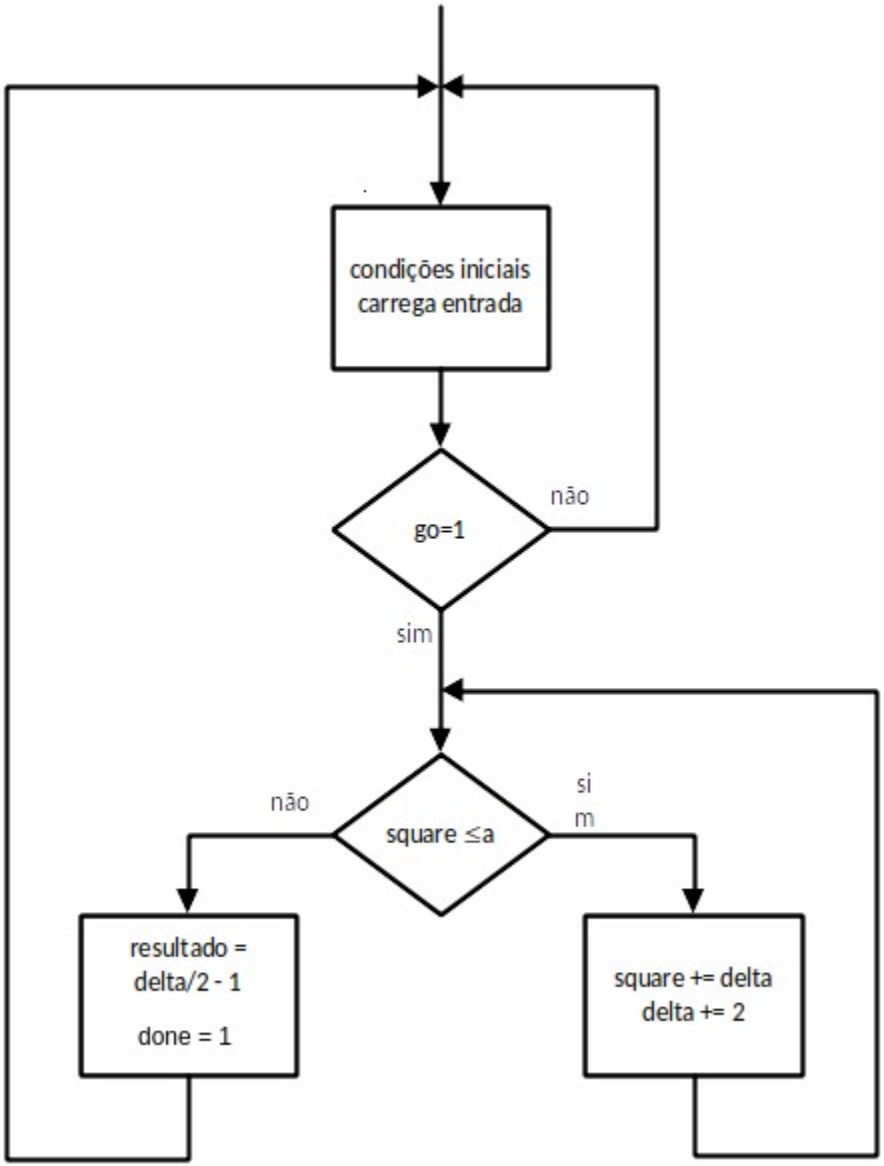
```
unsigned long sqrt(unsigned long a){
    unsigned long square = 1;
    unsigned long delta = 3;
    while(square <= a){
        square += delta;
        delta += 2;
    }
    return (delta/2 - 1);
}
```

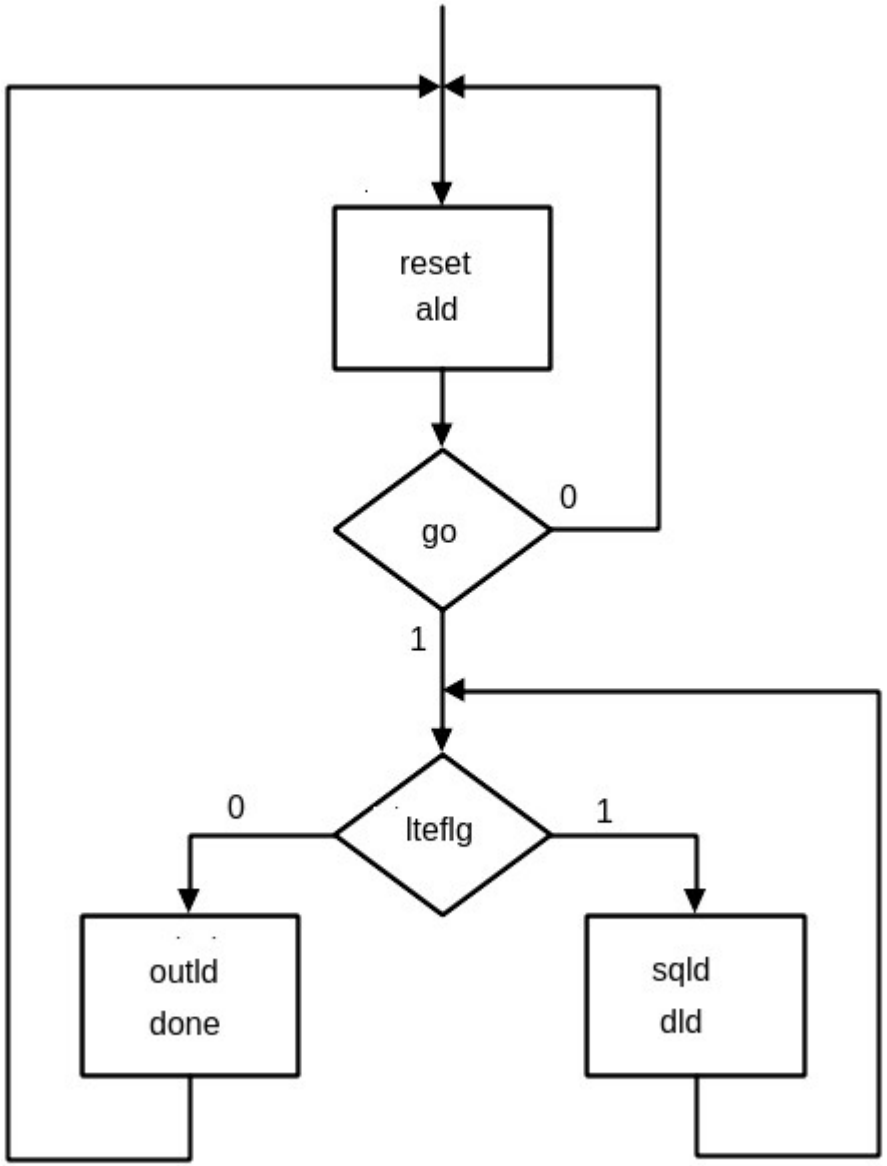
# Raiz inteira

- A Unidade de controle pode ser projetada a partir do diagrama ASM de alto nível:
  - Substitua cada operação sobre os dados pelos sinais de controle que devem ser ativados
  - Substitua condições sobre os dados pelos sinais de status correspondentes

```
unsigned long sqrt(unsigned long a){
     unsigned long square = 1;
     unsigned long delta = 3;
     while(square <= a){
        square += delta;
        delta += 2;
     }
     return (delta/2 - 1);
}
```




condições iniciais
carrega entrada
go=1
não
sim
square ≤a
não
si
m
resultado =
delta/2 - 1
done = 1
square += delta
delta += 2

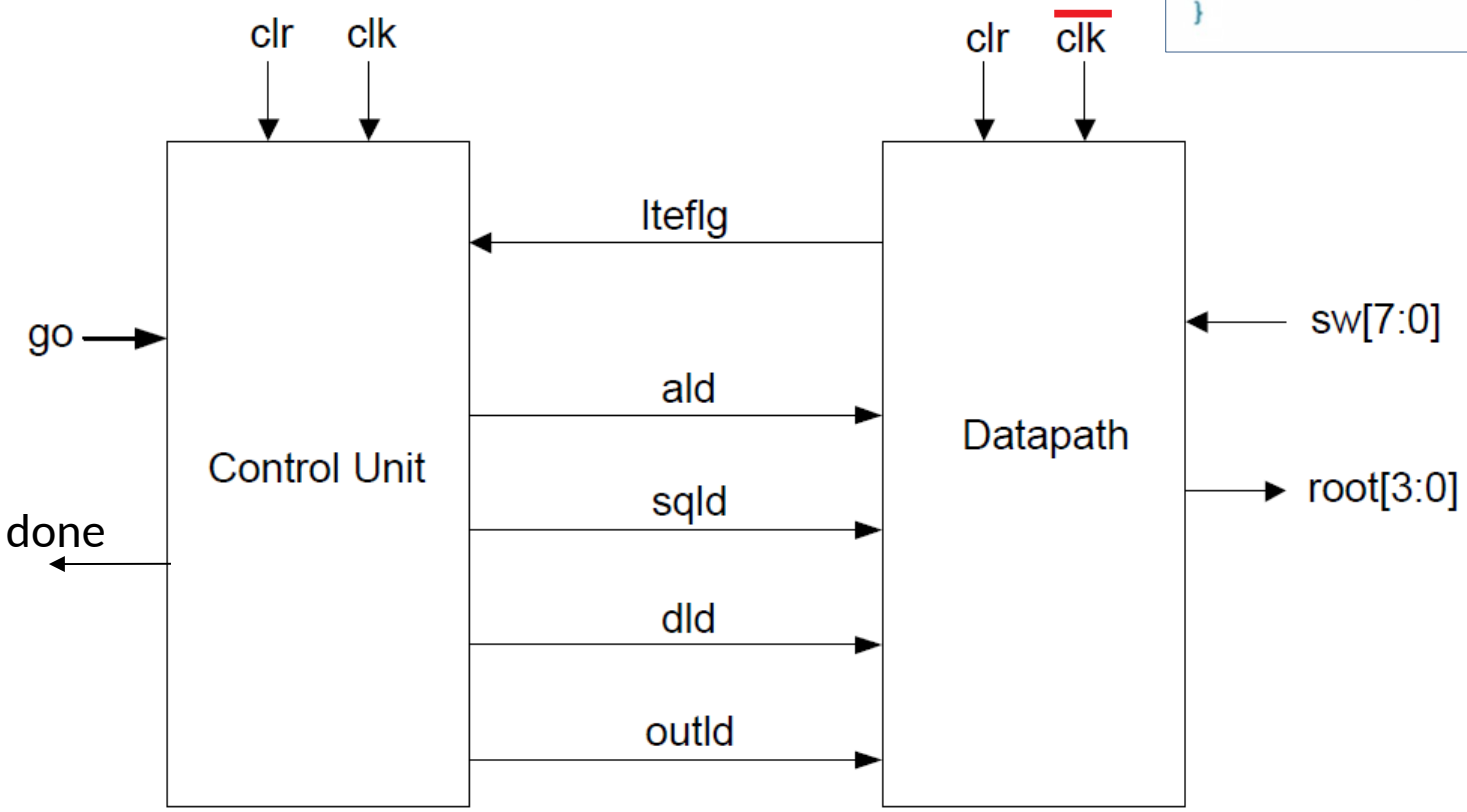

reset
ald
go
0
1
lteflg
0
1
outld
done
sqld
dld

# Raiz inteira

- Conexão FD e UC

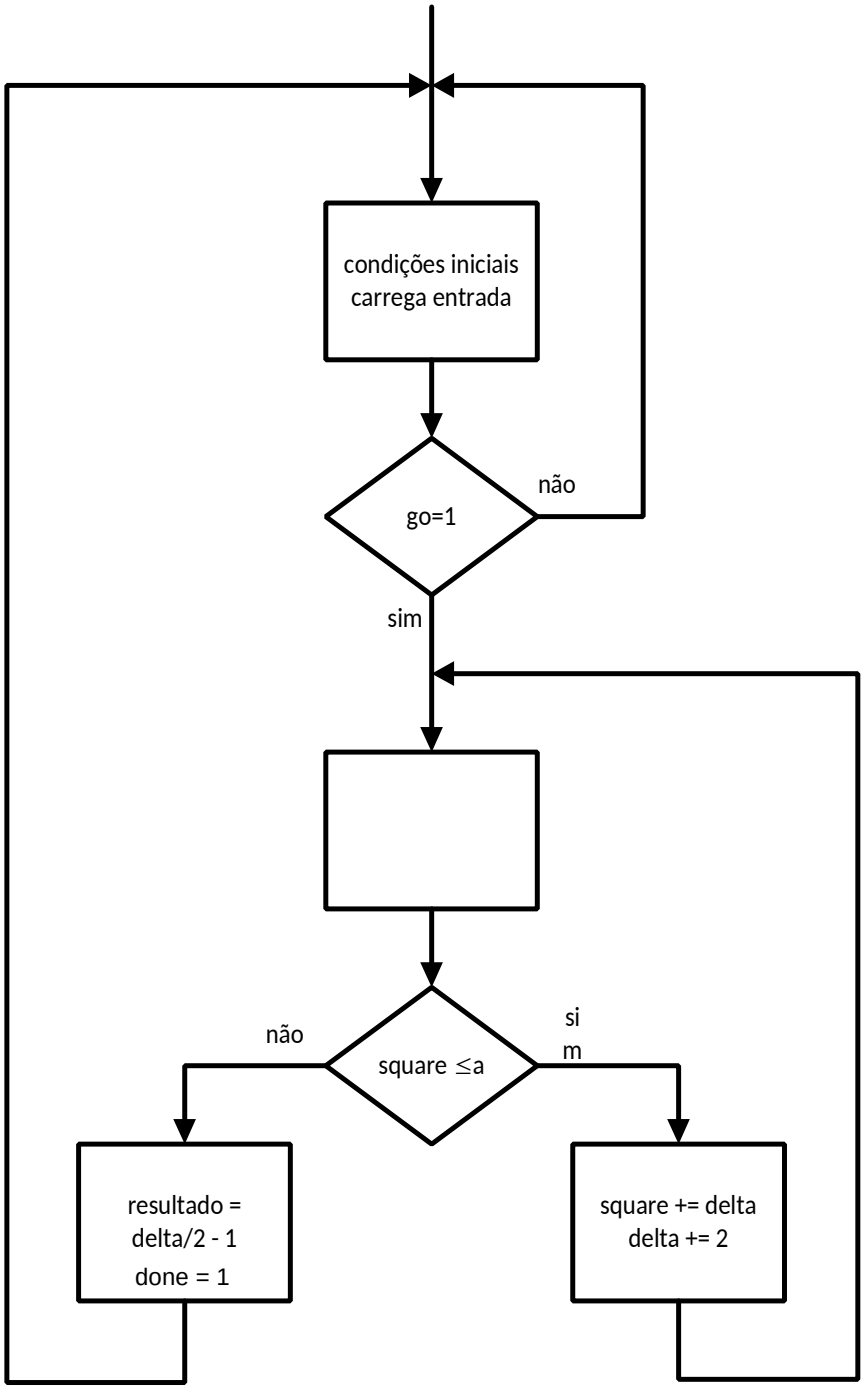
**Clock do FD é a negação do clock da UC!**

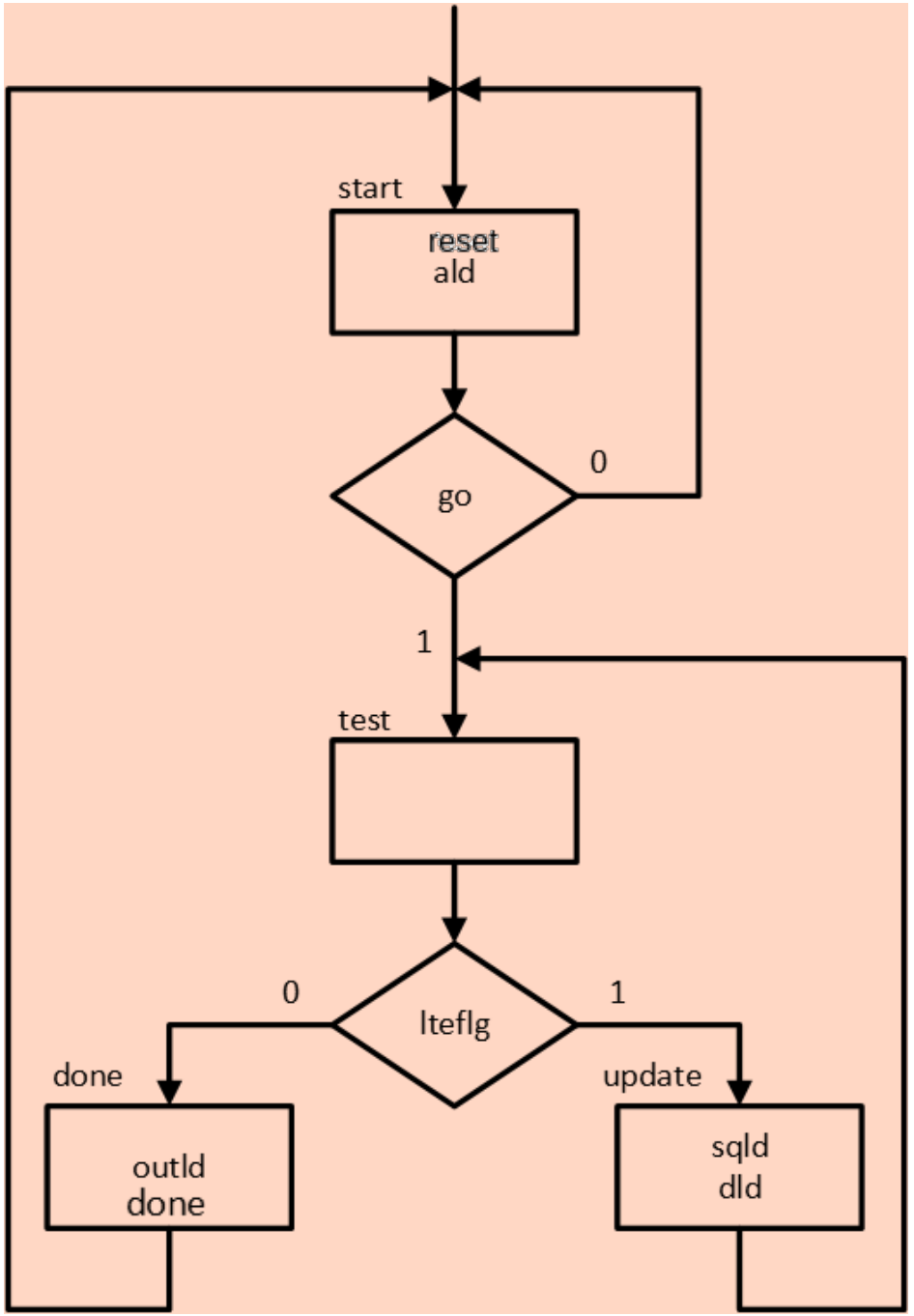
```
unsigned long sqrt(unsigned long a){
      unsigned long square = 1;
      unsigned long delta = 3;
      while(square <= a){
         square += delta;
         delta += 2;
      }
      return (delta/2 - 1);
}
```



# Raiz inteira v2

- Novamente, podemos projetar uma solução assumindo que o clock da UC é igual ao clock do FD
- Isso alteraria o diagrama ASM de alto nível
- Consequentemente, alteraria a Máquina de Estados da Unidade de Controle


condições iniciais
carrega entrada
go=1
não
sim
não
square ≤a
si
m
resultado =
delta/2 - 1
done = 1
square += delta
delta += 2

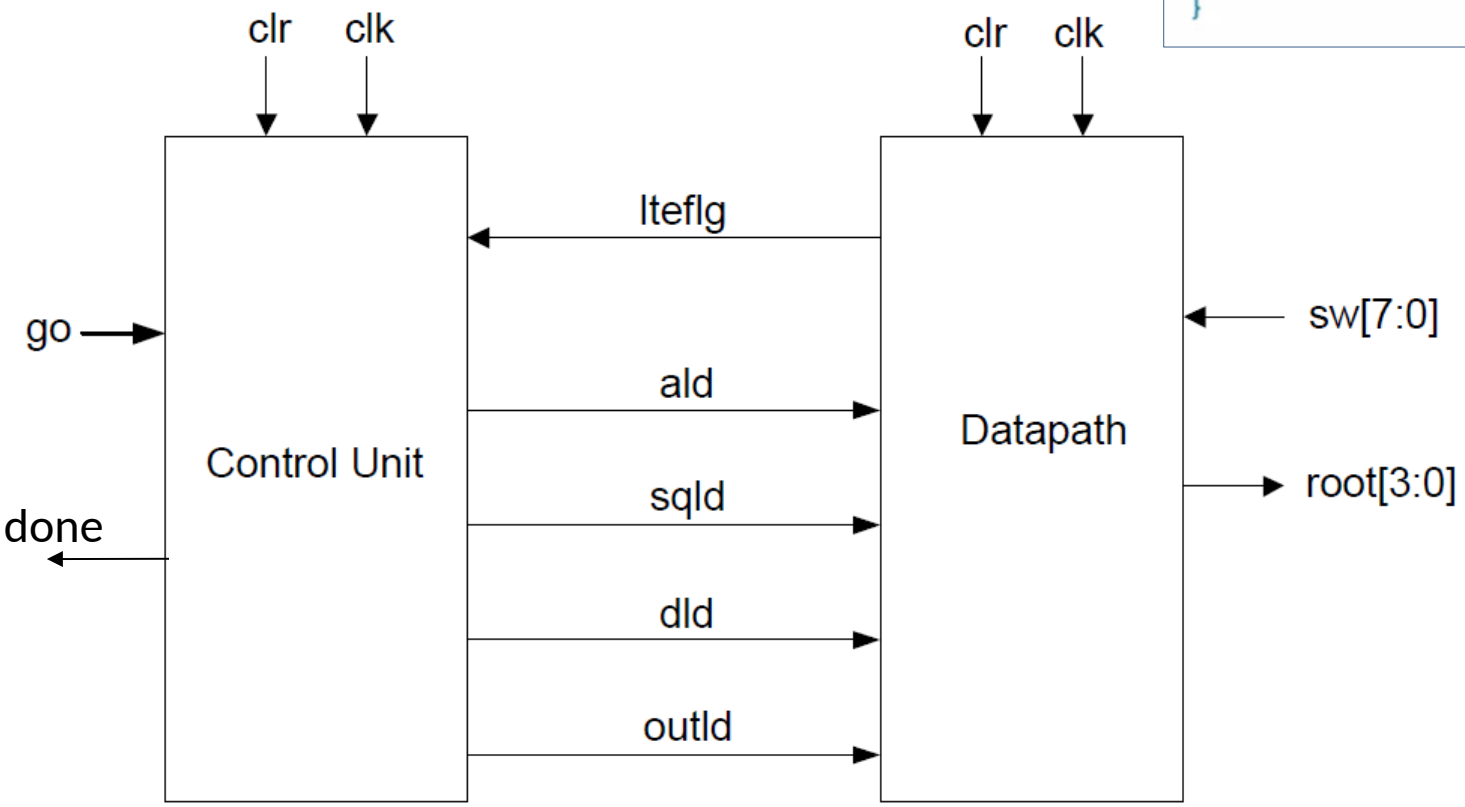

start
reset
ald
go
0
1
test
lteflg
0
1
done
outld
done
update
sqld
dld

# Raiz inteira

- Conexão FD e UC (v2)

**Clock do FD é igual ao clock da UC!**

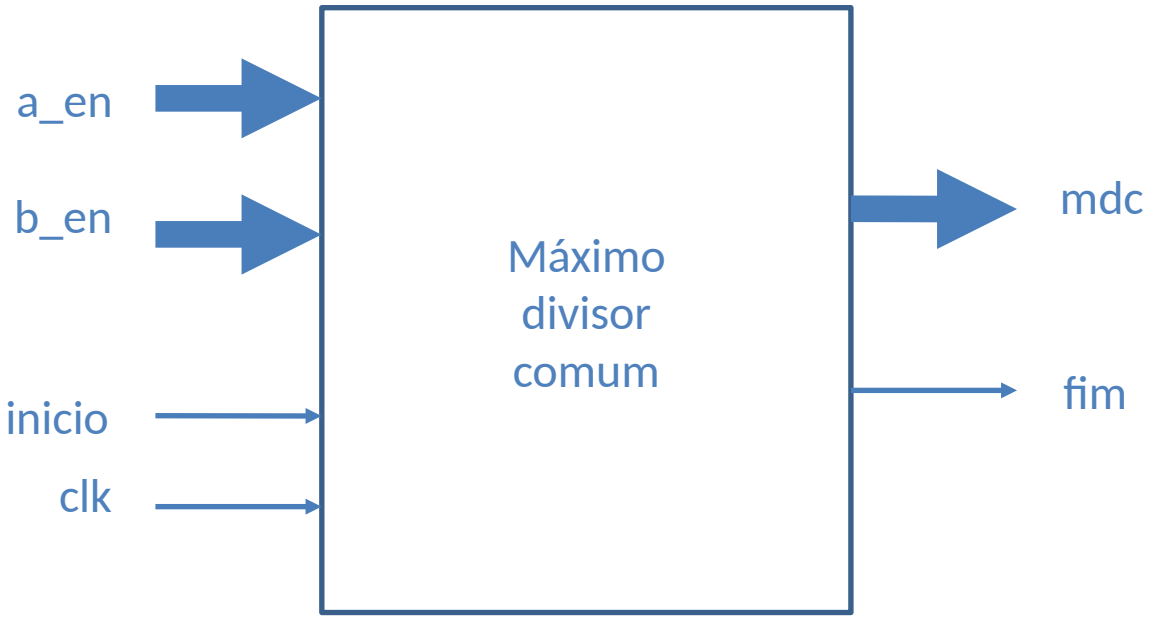
```
unsigned long sqrt(unsigned long a){
      unsigned long square = 1;
      unsigned long delta = 3;
      while(square <= a){
         square += delta;
         delta += 2;
      }
      return (delta/2 - 1);
}
```



# APÊNDICE: PROJETO MÁXIMO DIVISOR COMUM

# Máximo Divisor Comum

- Circuito que calcula o **máximo divisor comum** de dois números inteiros positivos a e b.




Referência: **D'Amore**, *VHDL: descrição e síntese de circuitos digitais*, 2ª edição, 2012 . Capítulo 16.

# Máximo Divisor Comum

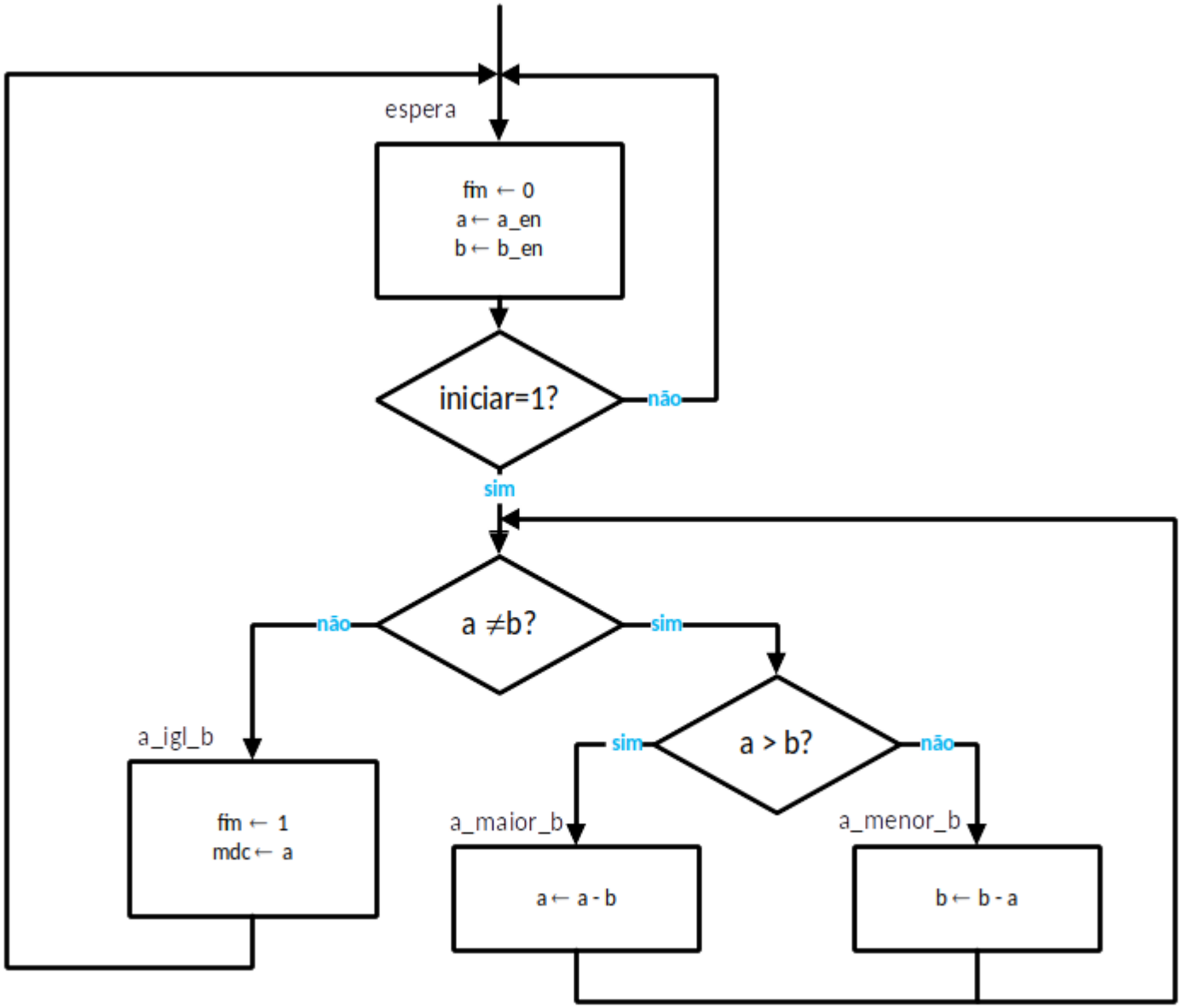
- **Algoritmo de Euclides:**

```
int a, b; //entradas
while ( a != b) {
    if ( a > b) {a = a – b ; }
    else        {b = b – a ;}
}
mdc = a;
```

Ex.: mdc (15,10)

| a | b | |
|---|---|---|
| 15 | 10 | $a \neq b$ e $a > b \rightarrow a = a-b$ |
| 5 | 10 | $a \neq b$ e $a < b \rightarrow b = b-a$ |
| 5 | 5 | $a = b \rightarrow mdc = a$ |

# Diagrama ASM de alto nível



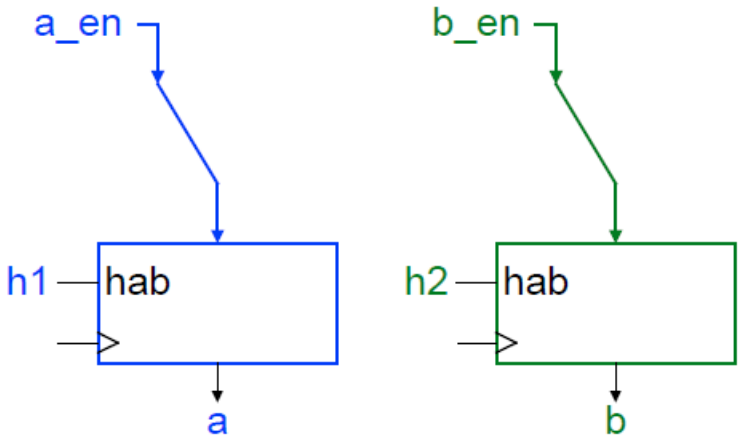
# Máximo Divisor Comum

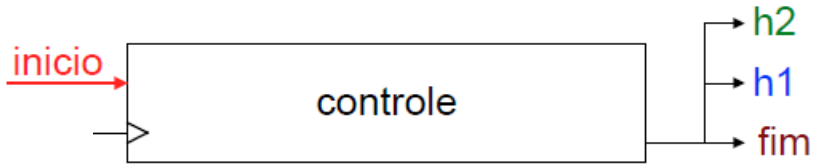
- Vamos analisar:
  - as operações executadas;
  - os recursos necessários para realizá-las;
  - as etapas necessárias para realizá-las.

# **Exemplo MDC:** análise das operações e unidades funcionais

- Armazena valores de "a_en" e "b_en"

```
while(inicio =1){
    fim = 0
    a = a_en;
    b = b_en;
    while (a /= b) {
      if (a > b) a = a -b;
      else       b = b -a;
    }
    mdc = a;
    fim = 1;
}
```

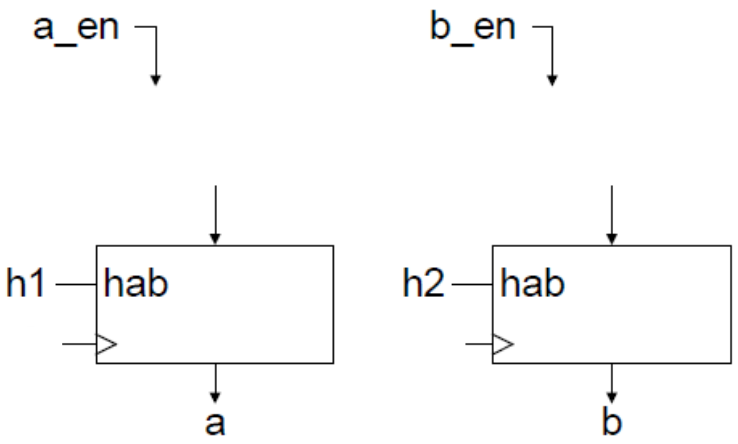


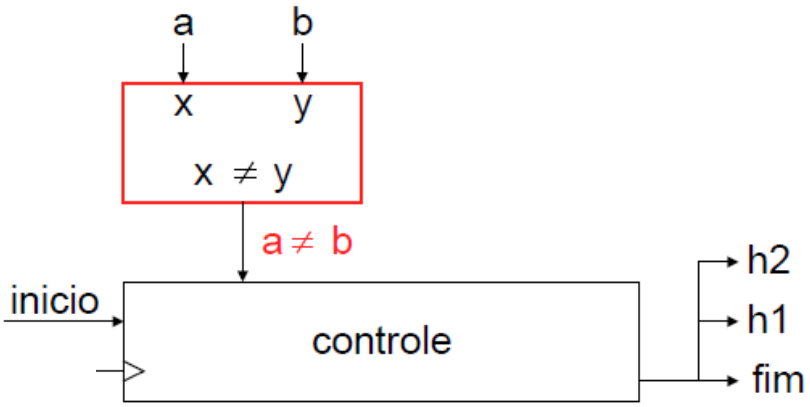



# **Exemplo MDC:** análise das operações e unidades funcionais

- Enquanto a ≠ b

```
while(inicio =1){
    fim = 0;
    a = a_en;
    b = b_en;
    while (a /= b) {
      if (a > b) a = a -b;
      else       b = b -a;
    }
    mdc = a;
    fim = 1;
}
```

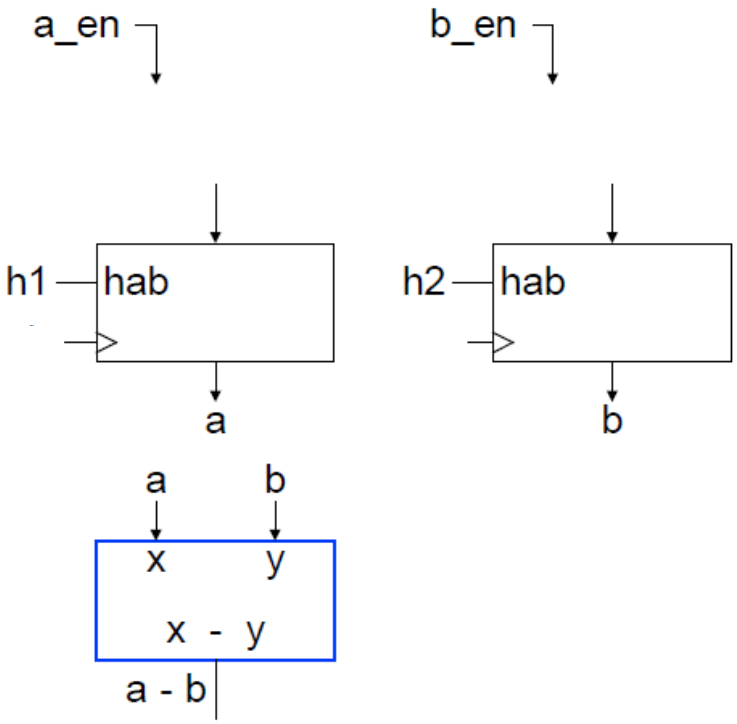


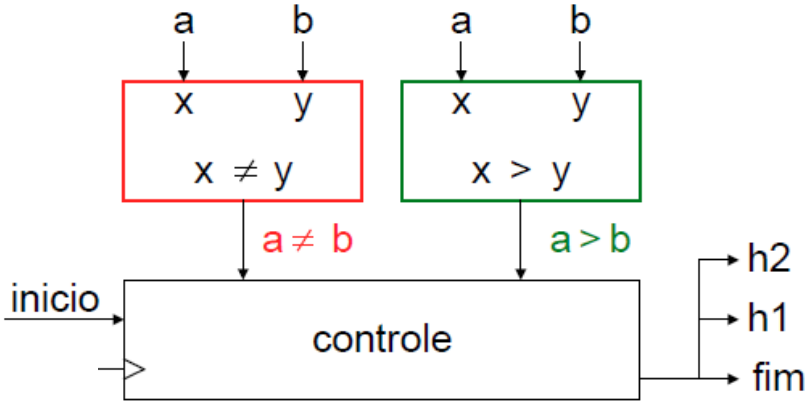



# **Exemplo MDC:** análise das operações e unidades funcionais

- Enquanto a ≠ b
- Se a > b → operação de subtração (a -b)

```
while(inicio =1){
    fim = 0
    a = a_en;
    b = b_en;
    while (a /= b) {
      if (a > b) a = a -b;
      else       b = b -a;
    }
    mdc = a;
    fim = 1;
}
```



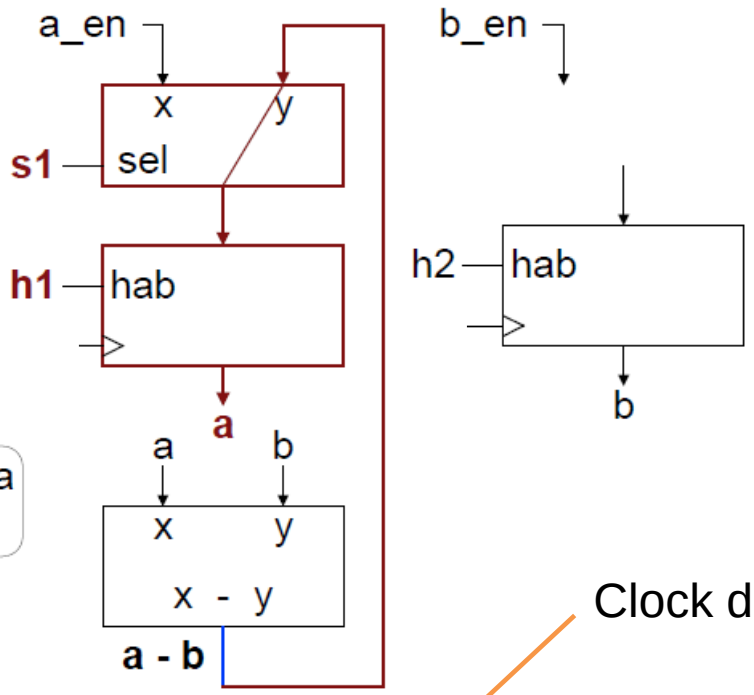



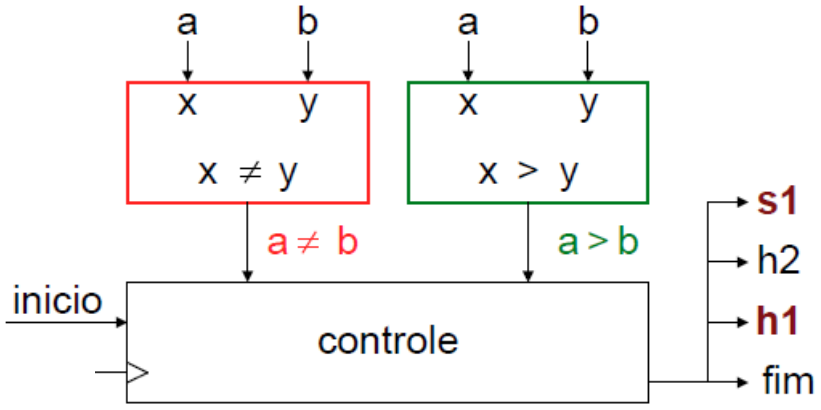
# Exemplo MDC: análise das operações e unidades funcionais

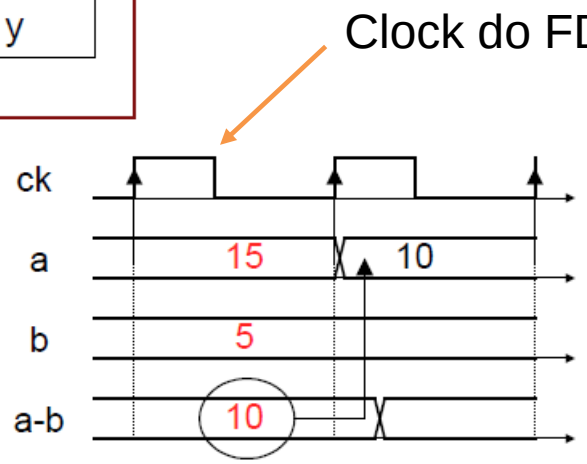
- Enquanto a ≠ b
- Se a > b → operação de subtração (a - b)
- Armazena em "a"

```
while(inicio =1){
    fim = 0;
    a = a_en;
    b = b_en;
    while (a /= b) {
      if (a > b) a = a -b;
      else       b = b -a;
    }
    mdc = a;
    fim = 1;
}
```

armazenamento na transição de ck



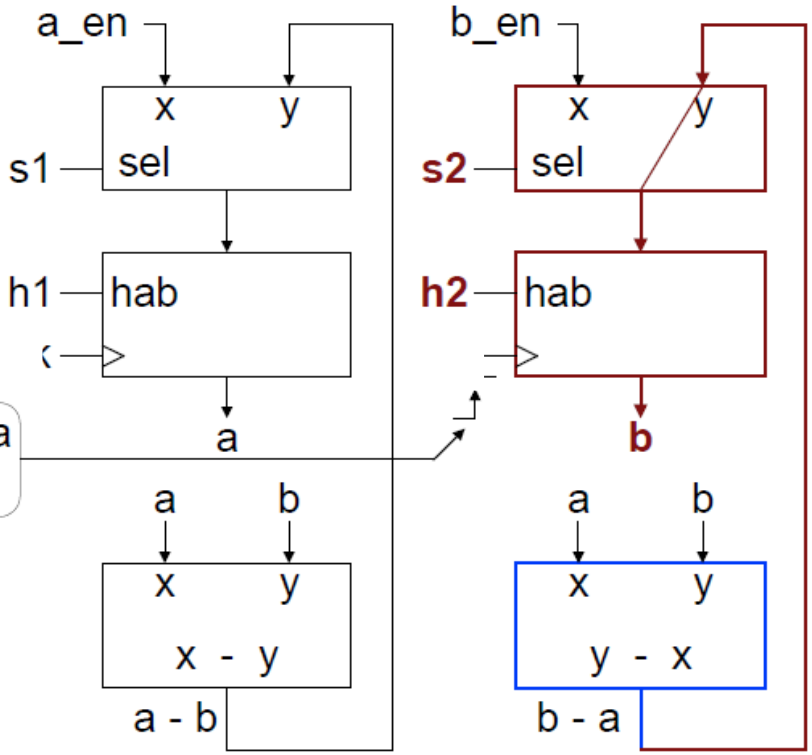



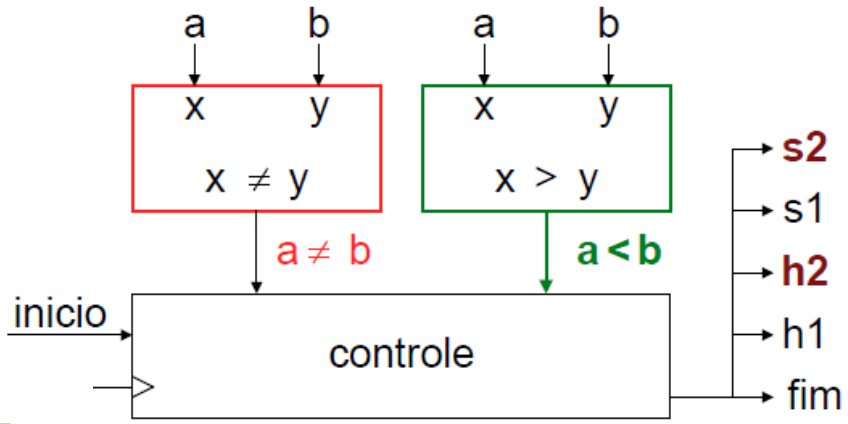
# **Exemplo MDC:** análise das operações e unidades funcionais

- Enquanto a ≠ b
- Se não, a > b → subtração (b - a)
- Armazena em "b"

```
while(inicio =1){
    fim = 0;
    a = a_en;
    b = b_en;
    while (a /= b) {
      if (a > b) a = a -b;
      else       b = b -a;
    }
    mdc = a;
    fim = 1;
}
```

armazenamento na transição de ck

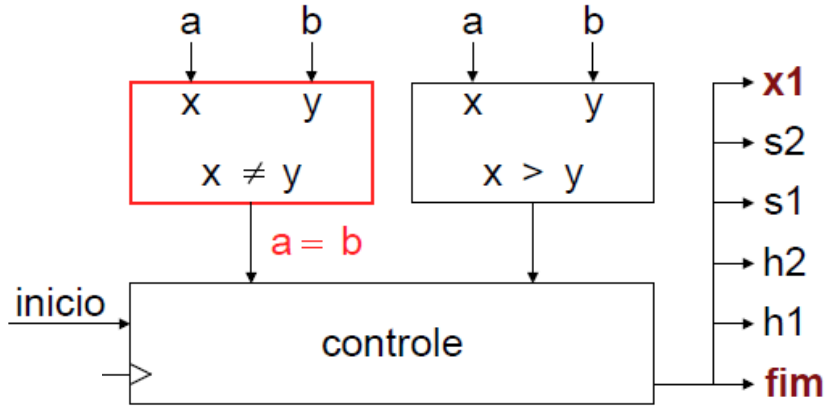


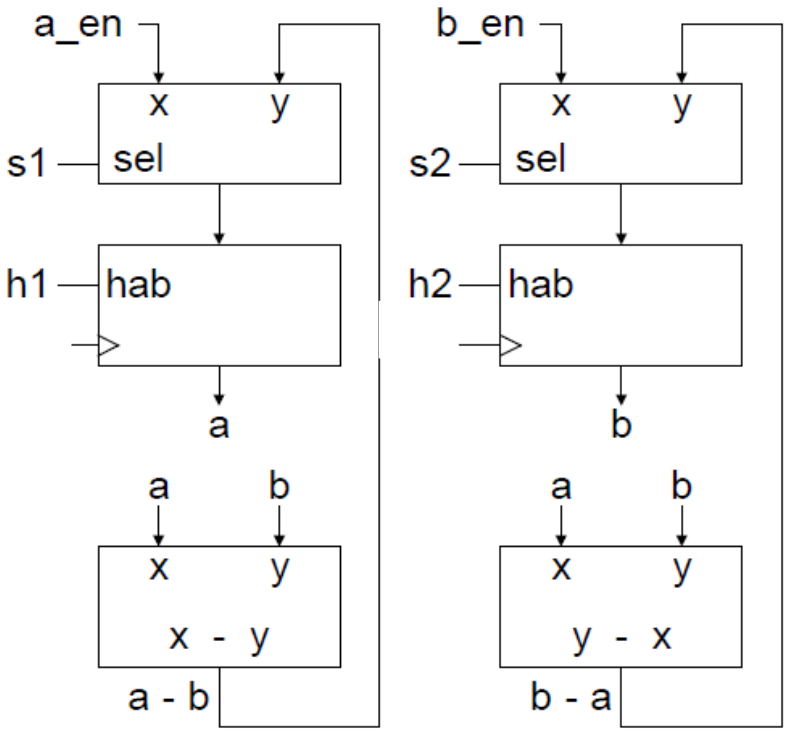



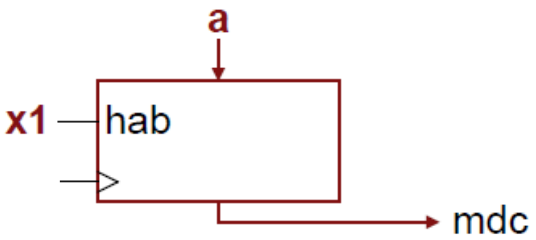
# **Exemplo MDC:** análise das operações e unidades funcionais

- Com a = b • Armazena "a" em "mdc" • Altera valor de "fim"

```
while(inicio =1){
    fim = 0;
    a = a_en;
    b = b_en;
    while (a /= b) {
      if (a > b) a = a -b;
      else       b = b -a;
    }
    mdc = a;
    fim = 1;
}
```

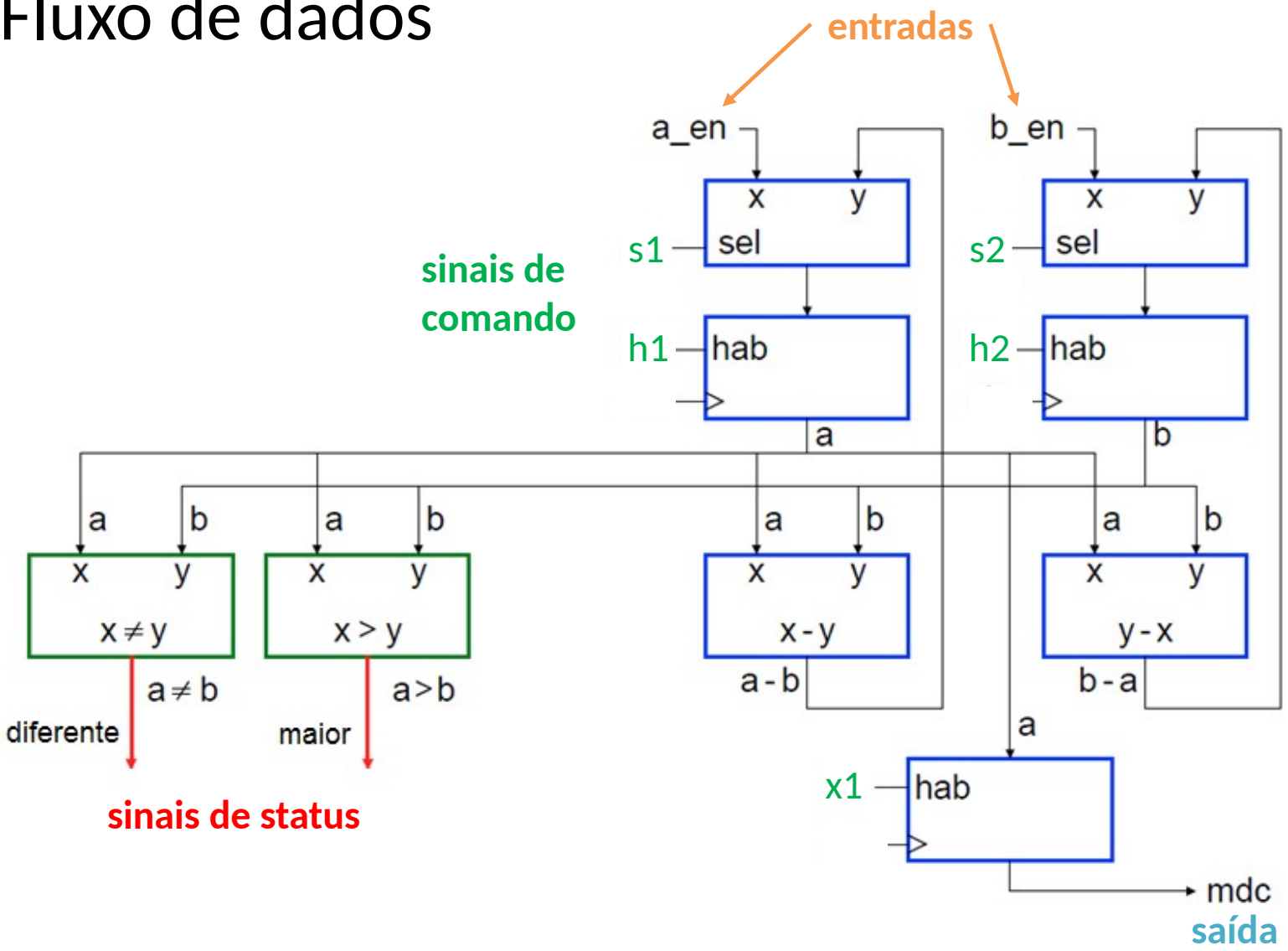


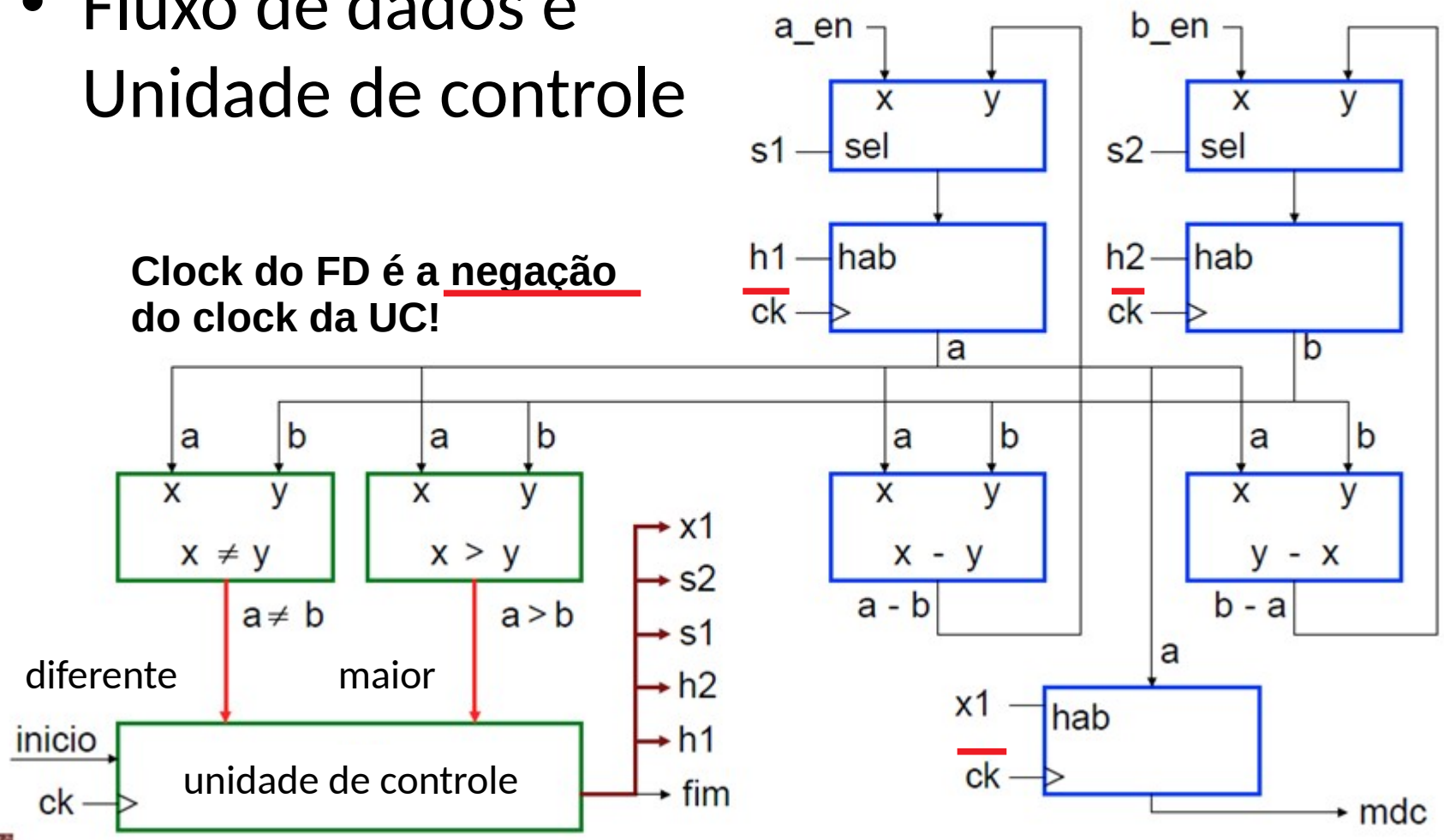
# Máximo Divisor Comum

- Fluxo de dados



# Máximo Divisor Comum

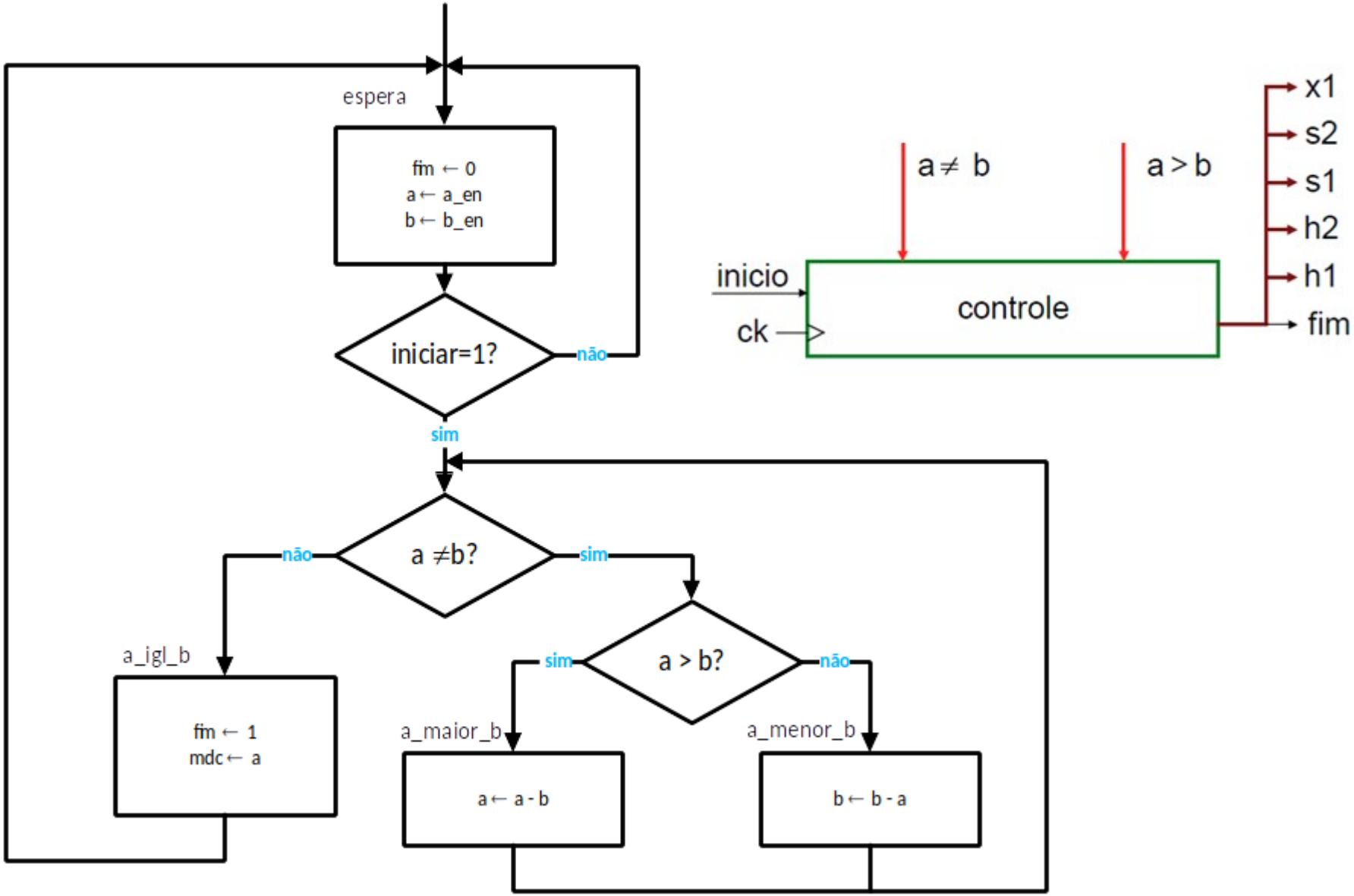
- Fluxo de dados e Unidade de controle



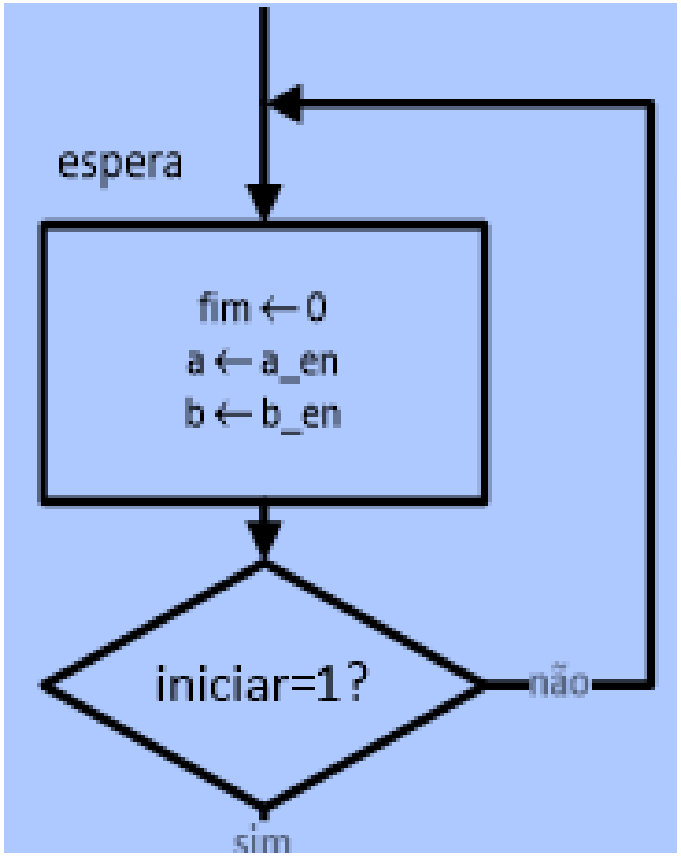
# Máximo Divisor Comum

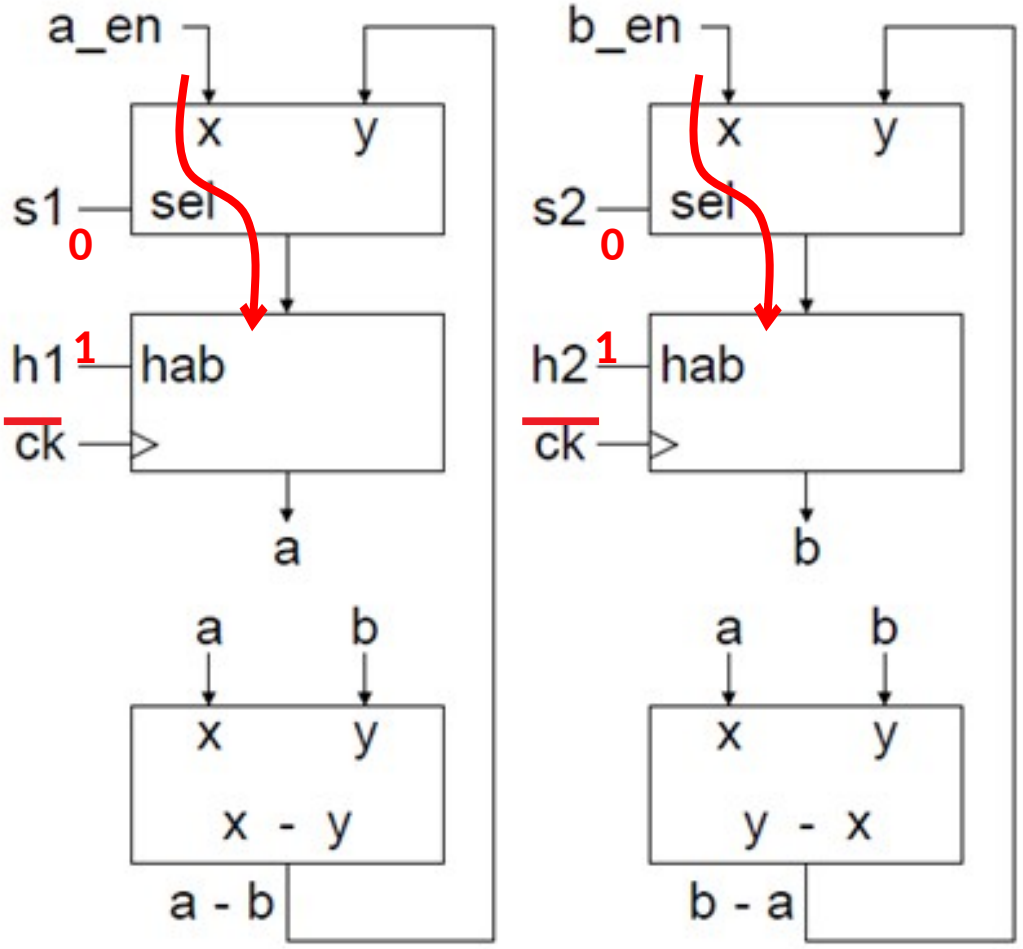
- Depois de definidos os módulos do fluxo de dados:
  - devemos estabelecer a sequência de operações do conjunto (como o processamento será executado);
  - responsabilidade da unidade de controle.
- O projeto completo define as :
  - operações executadas: fluxo de dados; e
  - mudanças de estados da máquina: unidade de controle.

# Diagrama ASM de alto nível



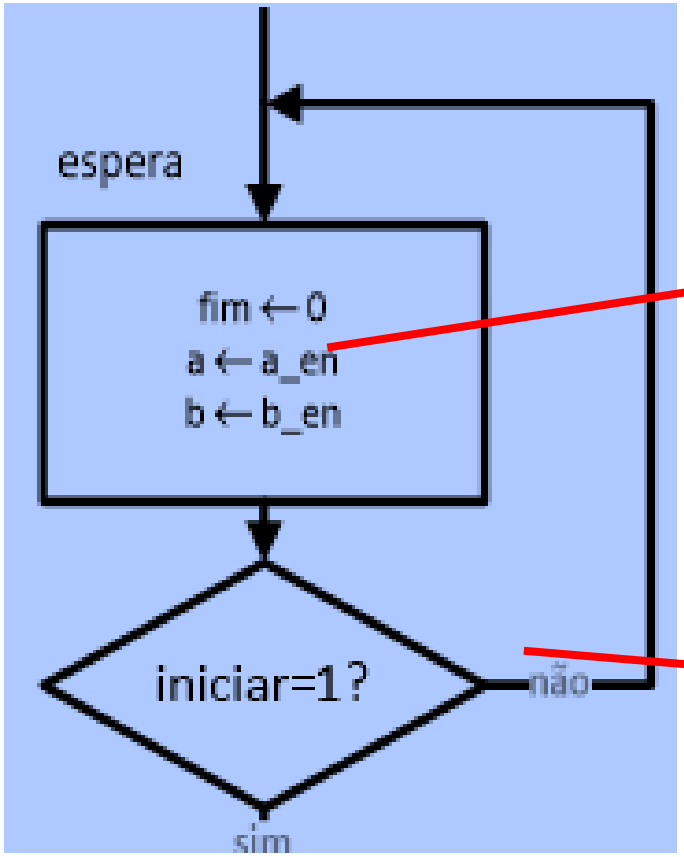
# Diagrama ASM de alto nível → ASM

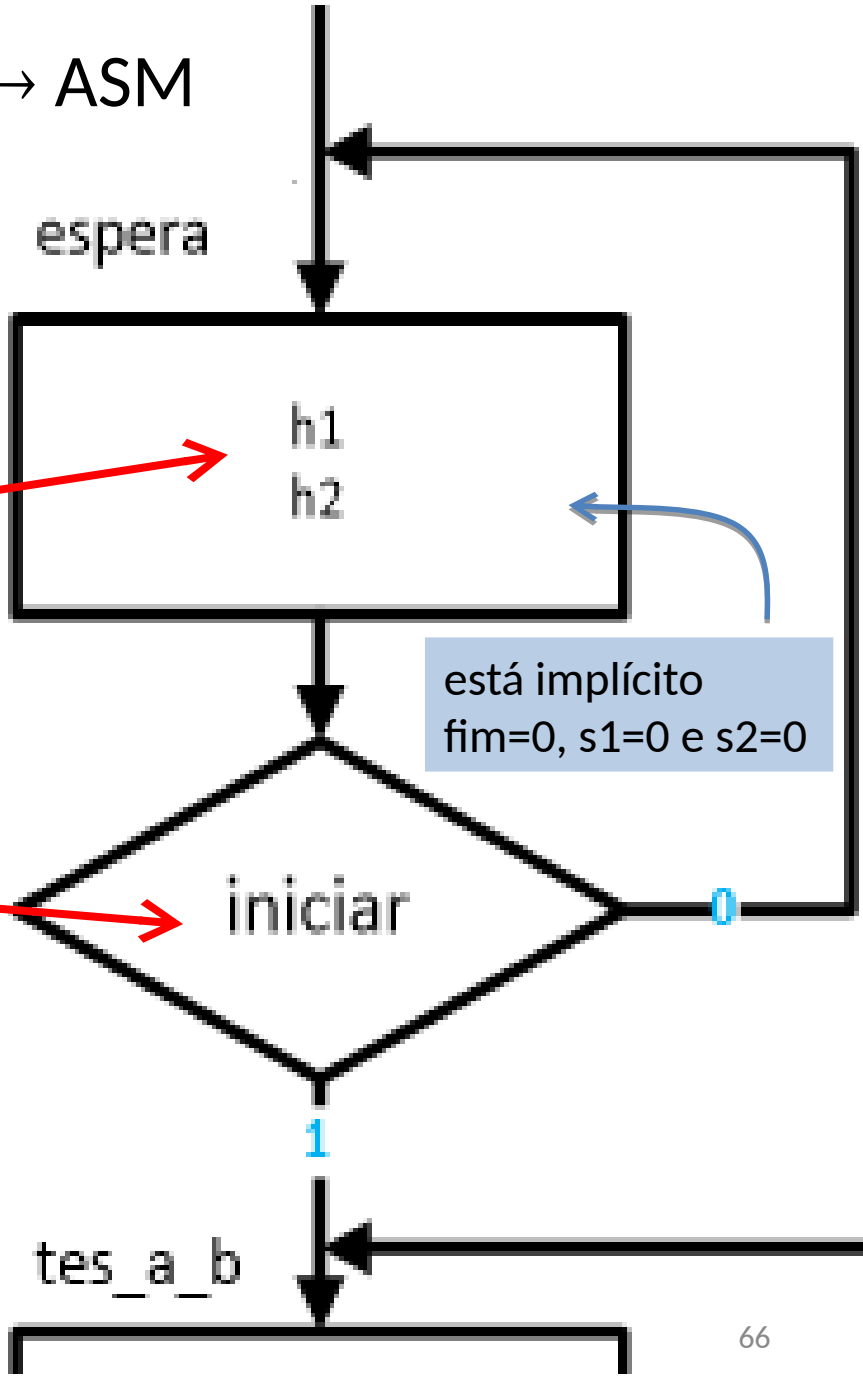


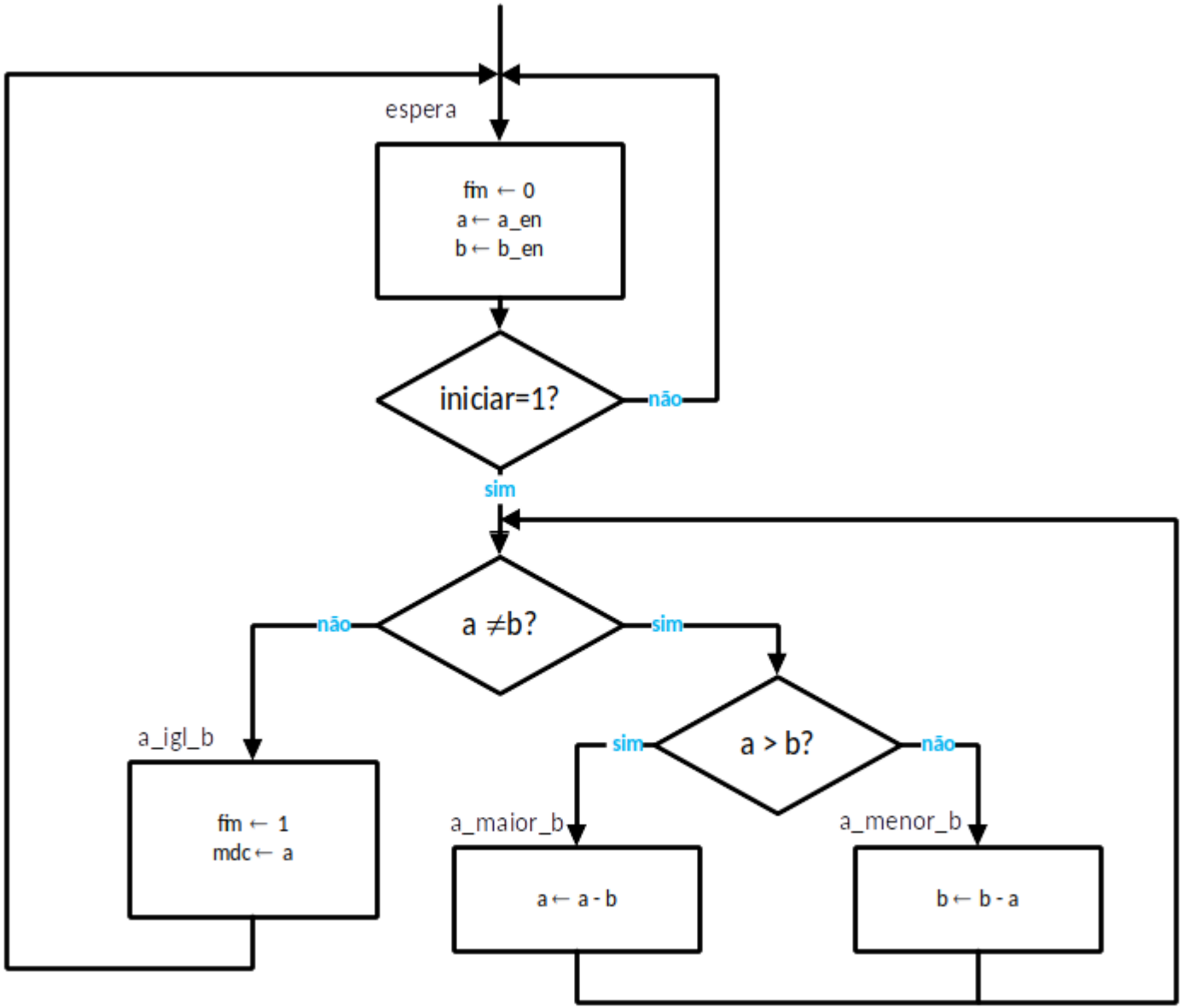

fluxo de dados

# Diagrama ASM de alto nível → ASM

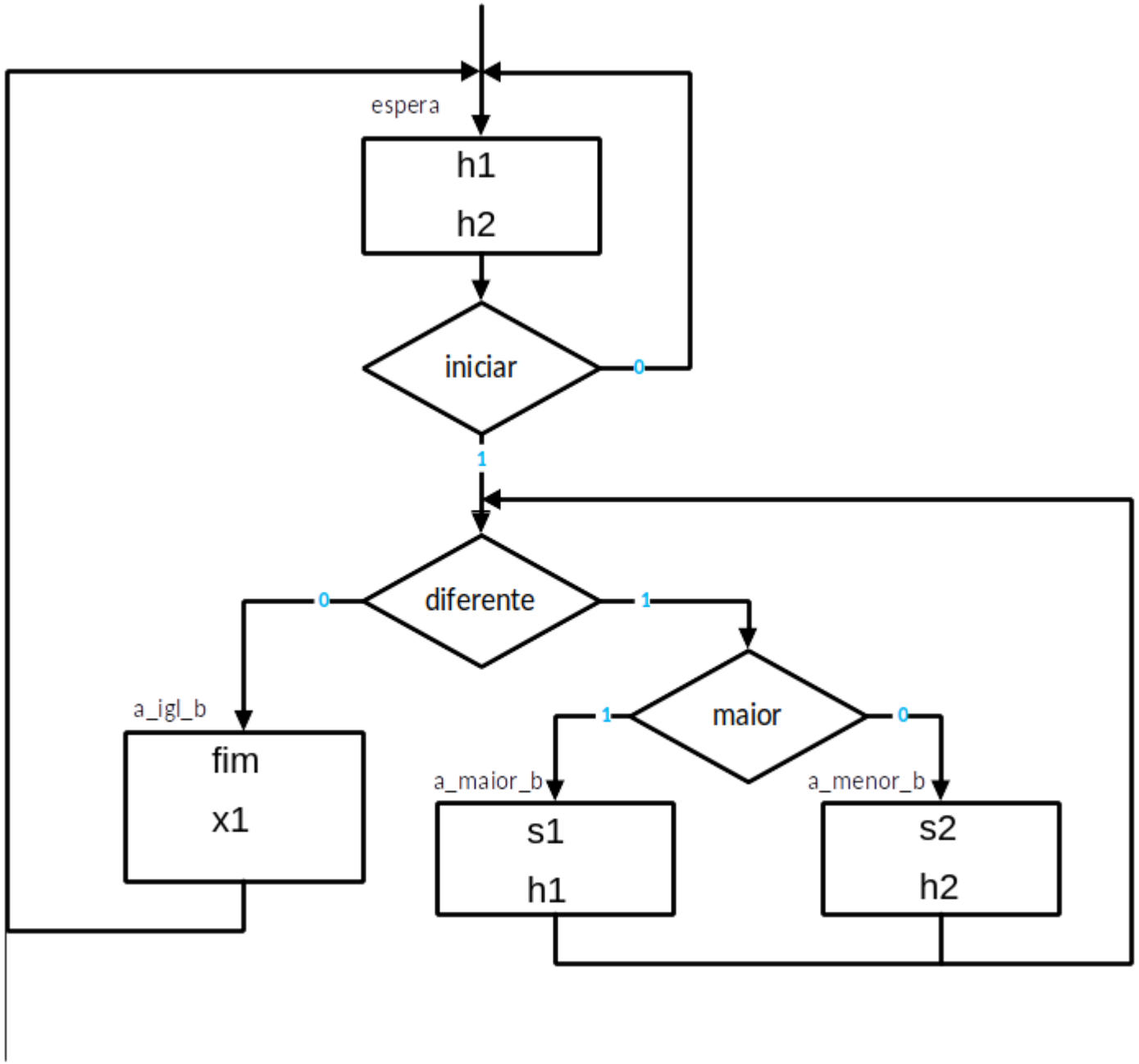


# Diagrama ASM de alto nível

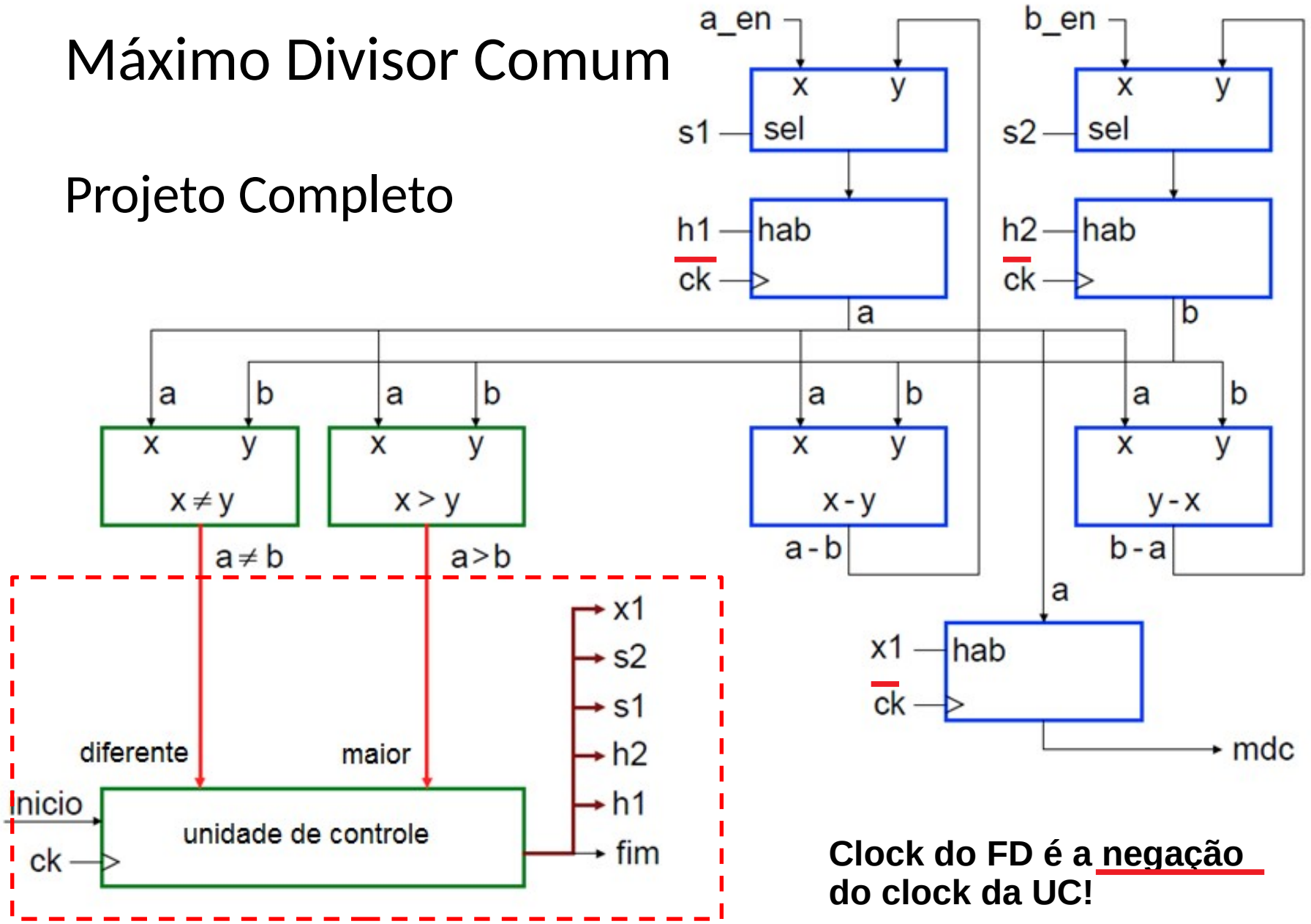


# Diagrama ASM da Unidade de controle



# Máximo Divisor Comum

## Projeto Completo

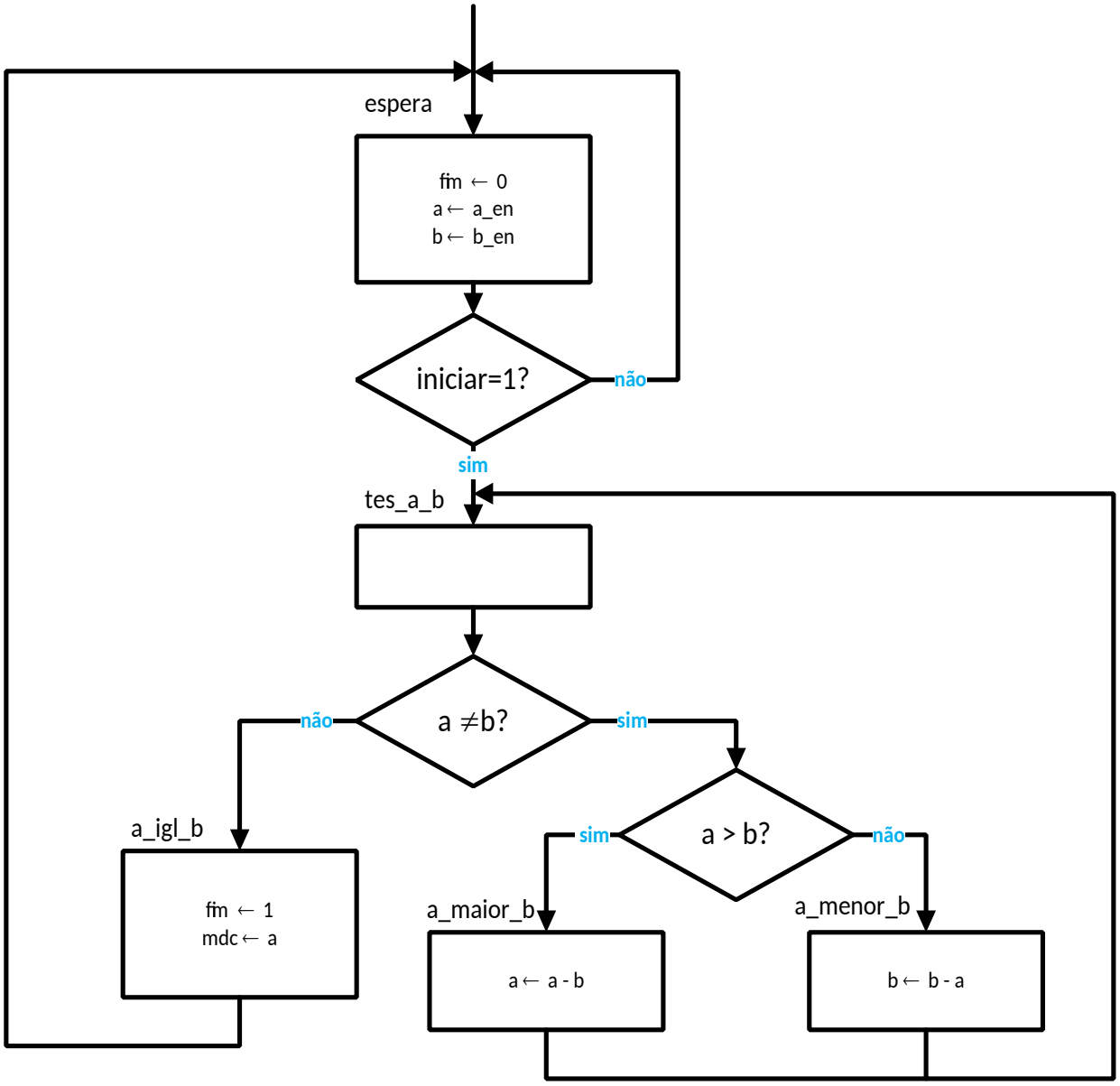



**Clock do FD é a negação do clock da UC!**

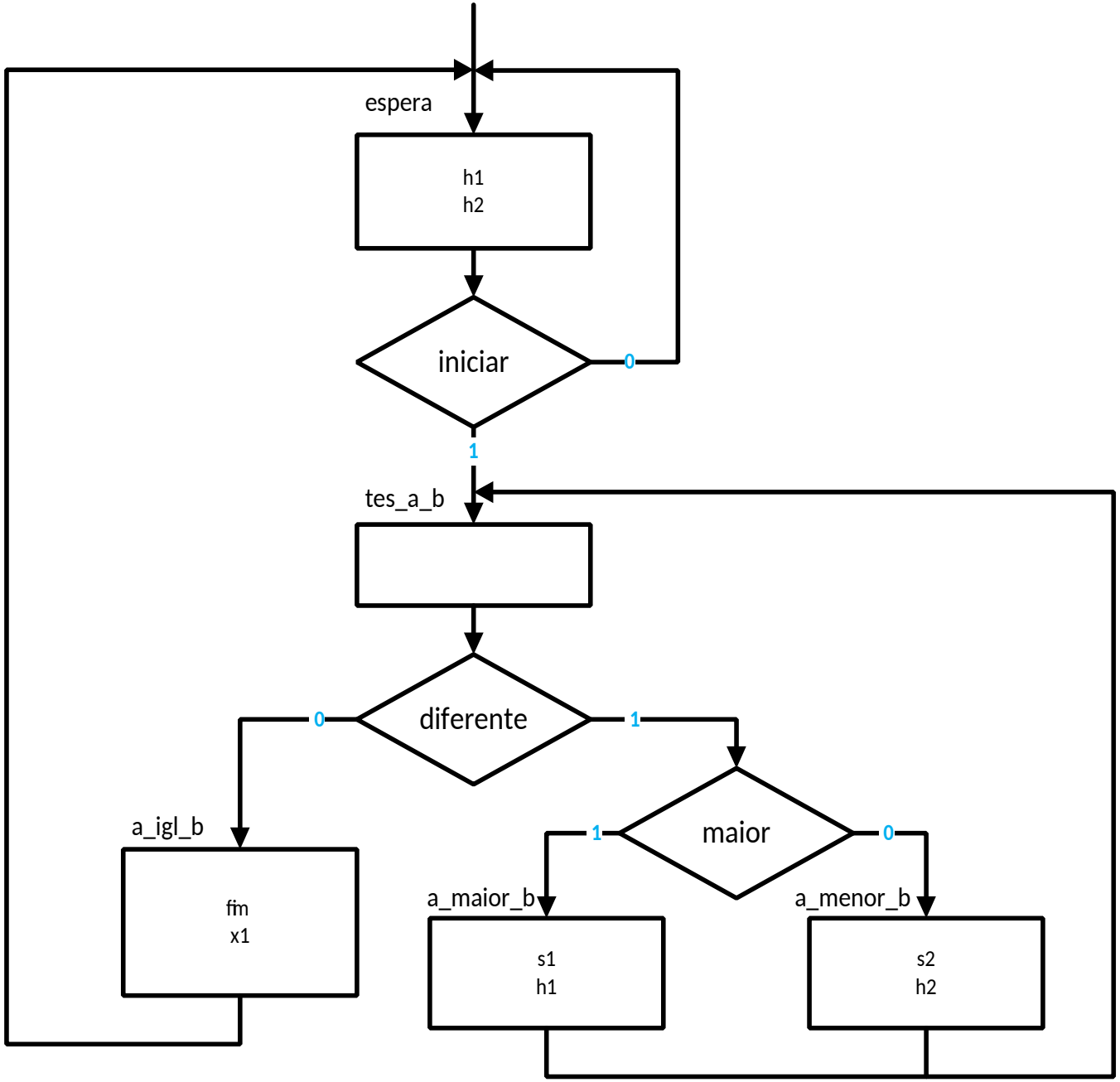
# Máximo Divisor Comum v2

- Novamente, podemos projetar uma solução assumindo que o clock da UC é igual ao clock do FD
- Isso alteraria o diagrama ASM de alto nível
- Consequentemente, alteraria a Máquina de Estados da Unidade de Controle

# Diagrama ASM de alto nível (v2)

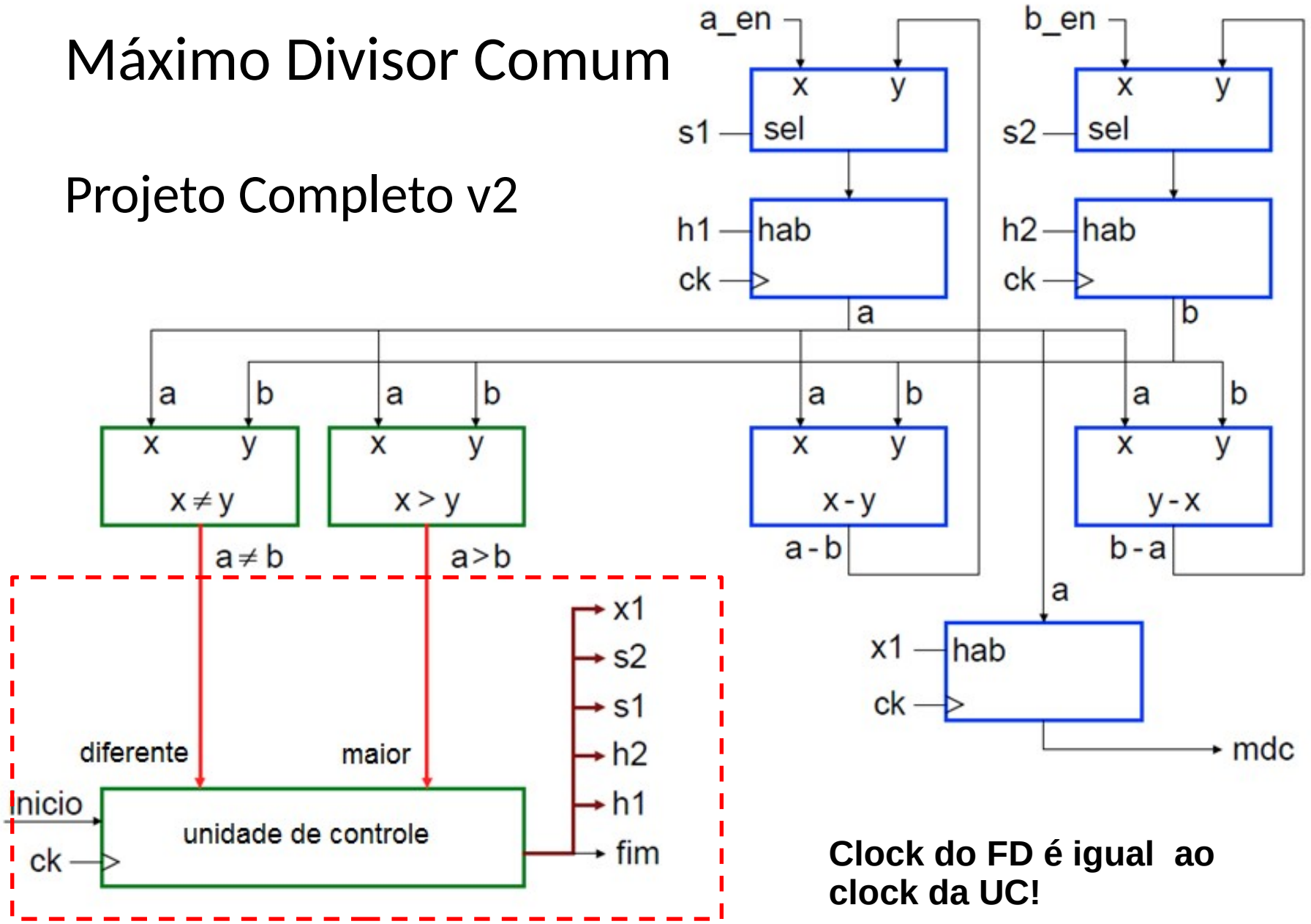


# Diagrama ASM da Unidade de controle (v2)



# Máximo Divisor Comum

## Projeto Completo v2




**Clock do FD é igual ao clock da UC!**

# Apêndice: Multiplicador v2 – Implementação em VHDL

# Projeto de Sistemas Digitais

- Implementação em VHDL (fluxo de dados)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.STD_LOGIC_MISC.all;

entity multiplicador_fd4bits is
  port(
        va : in std_logic_vector(3 downto 0);
        vb : in std_logic_vector(3 downto 0);
        clk : in std_logic;
        cea : in std_logic;
        ceb : in std_logic;
        cer : in std_logic;
        dcb : in std_logic;
        cler : in std_logic;
        vresult : out std_logic_vector(7 downto 0);
        zrb : out std_logic
  );
end entity;
```

# Projeto de Sistemas Digitais

- Implementação em VHDL (fluxo de dados)

```
architecture behavioral of multiplicador_fd4bits is

component registradornbits
  generic(size : integer);
  port(
        d : in std_logic_vector((size - 1) downto 0);
        clk : in std_logic;
        ce : in std_logic;
        r : in std_logic;
        q : out std_logic_vector((size - 1) downto 0)
  );
end component;

component addernbits
  generic(size : integer);
  port(
        a : in std_logic_vector((size - 1) downto 0);
        b : in std_logic_vector((size - 1) downto 0);
        c : out std_logic_vector((size - 1) downto 0)
  );
end component;
```

# Projeto de Sistemas Digitais

- Implementação em VHDL (fluxo de dados)

```
component subtractor1nbits
  generic(size : integer);
  port(
        a : in std_logic_vector((size - 1) downto 0);
        c : out std_logic_vector((size - 1) downto 0)
  );
end component;

signal qa : std_logic_vector(3 downto 0);

signal db : std_logic_vector(3 downto 0);
signal qb : std_logic_vector(3 downto 0);

signal dresult : std_logic_vector(7 downto 0);
signal qresult : std_logic_vector(7 downto 0);

signal aadder : std_logic_vector(7 downto 0);
signal badder : std_logic_vector(7 downto 0);
signal cadder : std_logic_vector(7 downto 0);

signal asub : std_logic_vector(3 downto 0);
signal csub : std_logic_vector(3 downto 0);
```

# Projeto de Sistemas Digitais

- Implementação em VHDL (fluxo de dados)

```
begin

ra : registradornbits
        generic map(size => 4)
        port map(
        d => va,
        clk => clk,
        ce => cea,
        r => '0',
        q => qa
);

rb : registradornbits
        generic map(size => 4)
        port map(
        d => db,
        clk => clk,
        ce => ceb,
        r => '0',
        q => qb
);
```

```
result : registradornbits
        generic map(size => 8)
        port map(
        d => dresult,
        clk => clk,
        ce => cer,
        r => cler,
        q => qresult
);

adder : addernbits
        generic map(size => 8)
        port map(
                a => aadder,
                b => badder,
                c => cadder
        );
```

# Projeto de Sistemas Digitais

- Implementação em VHDL (fluxo de dados)

```
sub1 : subtractor1nbits
        generic map(size => 4)
        port map(
                a => asub,
                c => csub
        );

db <=   vb when dcb = '0' else
                csub when dcb = '1' else
                X"0";

dresult <= cadder;

aadder <= X"0" & qa;
badder <= qresult;

asub <= qb;

vresult <= qresult;
zrb <= NOR_REDUCE(qb);

end behavioral;
```

# Projeto de Sistemas Digitais

- Implementação em VHDL (unidade de controle)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity multiplicador_ctrl4bits is
  port(
        start : in std_logic;
        zrb : in std_logic;
        clk : in std_logic;
        ready : out std_logic;
        cea : out std_logic;
        ceb : out std_logic;
        cer : out std_logic;
        dcb : out std_logic;
        cler : out std_logic
  );
end entity;
```

# Projeto de Sistemas Digitais

- Implementação em VHDL (unidade de controle)

```
architecture behavioral of multiplicador_ctrl4bits is

type state is (swait, sx1, sloop, sx2, sfins);
signal estado_atual, proximo_estado : state := swait;

begin
  process(clk)
  begin
        if(clk'event and clk = '1') then
              estado_atual <= proximo_estado;
        end if;
  end process;
```

# Projeto de Sistemas Digitais

- Implementação em VHDL (unidade de controle)

```
process(estado_atual, start, zrb)
begin
      case estado_atual is
            when swait =>
                  if(start = '1') then
                        proximo_estado <= sx1;
                  else
                        proximo_estado <= swait;
                  end if;
            when sx1 =>
                  proximo_estado <= sloop;
            when sloop =>
                  if( zrb = '0') then
                        proximo_estado <= sx2;
                  else
                        proximo_estado <= sfins;
                  end if;
            when sx2 =>
                  proximo_estado <= sloop;
            when sfins =>
                  if(start = '0') then
                        proximo_estado <= swait;
                  else
                        proximo_estado <= sfins;
                  end if;
      end case;
end process;
```

# Projeto de Sistemas Digitais

- Implementação em VHDL (unidade de controle)

```
process(estado_atual)
        begin
                case estado_atual is
                when swait =>
                        cea <= '0';
                        ceb <= '0';
                        cer <= '0';
                        dcb <= '0';
                        cler <= '0';
                        ready <= '0';
                when sx1 =>
                        cea <= '1';
                        ceb <= '1';
                        cer <= '1';
                        dcb <= '0';
                        cler <= '1';
                        ready <= '0';
```

```
                when sloop =>
                        cea <= '0';
                        ceb <= '0';
                        cer <= '0';
                        dcb <= '0';
                        cler <= '0';
                        ready <= '0';
                when sx2 =>
                        cea <= '0';
                        ceb <= '1';
                        cer <= '1';
                        dcb <= '1';
                        cler <= '0';
                        ready <= '0';
                when sfins =>
                        cea <= '0';
                        ceb <= '0';
                        cer <= '0';
                        dcb <= '0';
                        cler <= '0';
                        ready <= '1';
        end case;
  end process;
end behavioral;
```