CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

TRABALHO PRÁTICO II

Processador, memória e *Translation Lookaside Buffer* (TLB)

Arthur Estevão de Souza Machado
Marcelo Lopes de Macedo Ferreira Cândido

Engenharia de Computação
Laboratório de Arquitetura e Organização de Computadores II

24 de setembro de 2018

INTRODUÇÃO

A necessidade de realizar tarefas computacionalmente mais rapidamente sempre se fez presente no mundo tecnológico. De fato, tal necessidade promoveu uma evolução no mundo dos processadores que supera o crescimento dos demais componentes de um sistema computacional. Foi possível conhecer durante o trajeto de desenvolvimento dos processadores diversas técnicas diferentes visando uma evolução no desempenho. Essas mudanças afetaram, desde a arquitetura do computador, e como um marco é possível ressaltar a Arquitetura de Von Neumann [1], até o surgimentos de transistores que ocupam menor espaço e dissipam menos calor.

As mudanças de artifício a fim de agregar mais desempenho à capacidade de processamento de fato apresentam um limite ao se tratar da larga utilização dos circuitos integrados(CI's) como já era de se esperar, uma vez que o tamanho dos transistores possui um limite inferior. Com o intuito de continuar evoluindo no quesito desempenho, não somente com a construção da unidade de processamento, surgiram os processadores multiciclo frente aos processadores monociclo. Ao contrário do processador monociclo, o multiciclo não possui a necessidade de manter fixo o número de ciclos para realizar suas instruções com base na instrução mais demorada.

Em linhas gerais, o aparecimento do processador proporcionou um ganho em eficiência, favorecendo um princípio de projeto, favorecendo o caso comum, e melhorando a vazão de dados já que ao terminar uma instrução mais rápida já é possível iniciar a próxima instrução, favorecendo casos de ociosidade do processador que antes ocorria com o monociclo.

O presente relatório visa apresentar a implementação de um processador multiciclo em conjunto com uma memória RAM e um *Translation Lookaside Buffer* (TLB), fazendo o uso da linguagem de descrição de hardware Verilog.

FUNCIONAMENTO

Top-level

A ligação memória-processador se dá como na Figura 1. Os endereços são providos pelo processador - que os consegue através do módulo *program counter* (PC) - à memória, que, na

[1] A Arquitetura de Von Neumann marcou fortemente a geração de dispositivos eletrônicos, com a possibilidade de armazenar programas no mesmo espaço de memória que os dados.

subida do sinal do *clock*, provê o/a dado/instrução contida naquela posição. Se o sinal *W* estiver ativado, ocorre a escrita do dado provido pelo processador através de *DOUT*. Terminada a execução da instrução vigente, o processador assinala *Done* e se prepara para receber a próxima instrução, se houver.

Figura 1: Diagrama de esquematização de ligação da memória RAM com o processador multiciclo [Fonte: *Netlist Viewer* - Quartus II 13.0sp1]

MEMÓRIA RAM

Módulo criado com o auxílio do subprograma *MegaWizard Plug-In Manager* da versão 13.0+ do *software* Quartus II. Possui como entradas e saídas

- *addr*: o endereço sobre o qual serão realizadas as operações sobre a memória;
- *data*: o dado a ser escrito na posição descrita por *Addr*;
- *ramQ*: o dado lido da posição apontada por *Addr*;
- *w*: sinal que sinaliza a leitura (0) ou a escrita (1) na memória.
- *clock*: sinal oscilatório com o objetivo de sincronizar memória e processador.

Translation Lookaside Buffer (TLB)

O TLB é parte da tentativa de diminuir o tempo de acesso à memória a partir de uma CPU (*Central Processing Unit*, ou, em português, Unidade Central de Processamento) através da tradução de endereços de memória virtual para endereços de memória física [**?**].

O módulo TLB criado para esse trabalho é básico, capaz de traduzir o endereço de 16 *bits* provido pelo processador em seis, que é o tamanho do endereço usado pela memória.

PROCESSADOR MULTICICLO

O funcionamento geral do processador (ou seja, para as suas instruções em geral) pode ser descrito pelas seguintes etapas

1. **IF0** - *Instruction Fetching 0*: incremento do PC, chamada da próxima instrução (*ADDR* atualizado no próximo ciclo), *reset* dos sinais alterados pela instrução anterior;
2. **IF1** - *Instruction Fetching 1*: endereço calculado por PC enviado a ADDR;
3. **ID** - *Instruction Decode*: movimento de dados a partir dos registradores (ou da entrada do processador) para outros locais do processador;
4. **EXE** - *Execution*: uso da ULA, escrita em registradores, preparação de escrita na memória;
5. **RWB** - *Register Write-Back*: termina-se a escrita nos registradores, a escrita/leitura em memória fica em andamento (por conta da latência da memória);
6. **MWB** - *Memory Write-Back*: finalização da manipulação da memória.

DETALHES DA IMPLEMENTAÇÃO

A fim de relacionar principalmente o módulo da *Ram* e o processador, foi necessário organizar e projetar como os dados emitidos e recebidos por cada um iriam trabalhar em conjunto com base nos ciclos de clock.

Figura 2: Projeto relacionando os módulos lpmRAM e processor

Com isso, foi estabelecido que o dado de dezesseis bits. ramQ, enviado pela memória , que depende diretamente do endereço encaminhado à ela, ao processador irá representar ou uma instrução, ou um dado específico. Partindo deste princípio dos 16 bits destinados à palavra, seis pertencem aos registradores X e Y (três para cada um), e quatro ao OP-Code. Este último irá representar qual operação será realizada pelo processador e devido à quantidade de bits separados, poderá representar até 16 instruções diferentes, no entando, nessa implementação serão trabalhadas apenas onze. Sobre os registradores de auxílio, serão utilizados todos os oito capazes de serem endereçados, sendo sete registradores comuns e um PC para tratar a execução

do programa inserido na memória.

000000 000(RY) 000(RX) 0000(OPCODE)

Além disso, dentro do módulo processor possuem registradores intermediários com o objetivo de armazenar os últimos valores trabalhados e também auxiliar na sincronização de uma operação. Um diagrama esquemático é apresentado na figura 3.

Figura 3: Projeto de um processador multiciclo fornecido como material de apoio da Intel.

Como é possível observar, o projeto apresentado pela Figura 3 possui um registrador de instruções nomeado como IR, que possui nove bits. Na implementação que será apresentada, o número de bits deste registrador aumentou para dez, com o objetivo de suportar as onze instruções desejadas. São elas:

Instrução - OPCODE
ADD (Soma) - 0001
SUB (Subtração) - 0010
OR (*Or*)- 0011
SLL (Shift Left Logical) - 0100
SRL (Shift Right Logical) - 0101
SLT (Set on Less Than)- 0110
MVI (Move Immediate)- 0111
MV (Move)- 1000
MVNZ (Move Not Zero)- 1001
ST (Store)- 1010
LD (Load)- 1011

O valor contido dentro do registrador Counter, é incrementado a cada pulso de clock, e auxilia a construção da máquina de estados para estabelecer os estágios da instrução. Já o registrador PC, será responsável por indicar os endereços de busca na memória, enviando (quando habilitado pelo multiplexador) o endereço que será trabalhado ao registrador de endereços.

O módulo multiplexador atua em conjunto com os sinais de controle, uma vez que ao selecionar que uma entrada deverá sair pelo busWires, os sinais que deverão receber essa saída deverão ser habilitados, da mesma forma que os módulos que não podem receber, devem estar desabilitados.

SIMULAÇÕES

Para *debug* e comprovação do devido funcionamento do código, a dupla procurou utilizar tanto a visualização a nível de transferência entre registradores (*RTL viewer*, em inglês) e a placa Altera disponível no laboratório da disciplina.

MODELSIM

Aqui serão exibidas simulações realizadas para cada instrução (apresentada por nome e código de operação (*opcode*)), descrevendo-se o que ocorre em cada etapa delas (através das legendas das imagens). A descrição da desabilitação de sinais utilizados em etapas passadas não será explicitada.

ADD - 0001

Figura 4: Instrução ADD R1, R0 - o registrador R1 recebe R1 + R0

images/Step00_InstrADD.PNG

Figura 5: Instrução ADD R1, R0 - etapa 0 - o registrador de instruções recebe a instrução de soma, o *program counter* é incrementado (sinal *incr_pc*) e o ADDR é preparado para receber o endereço do próximo dado

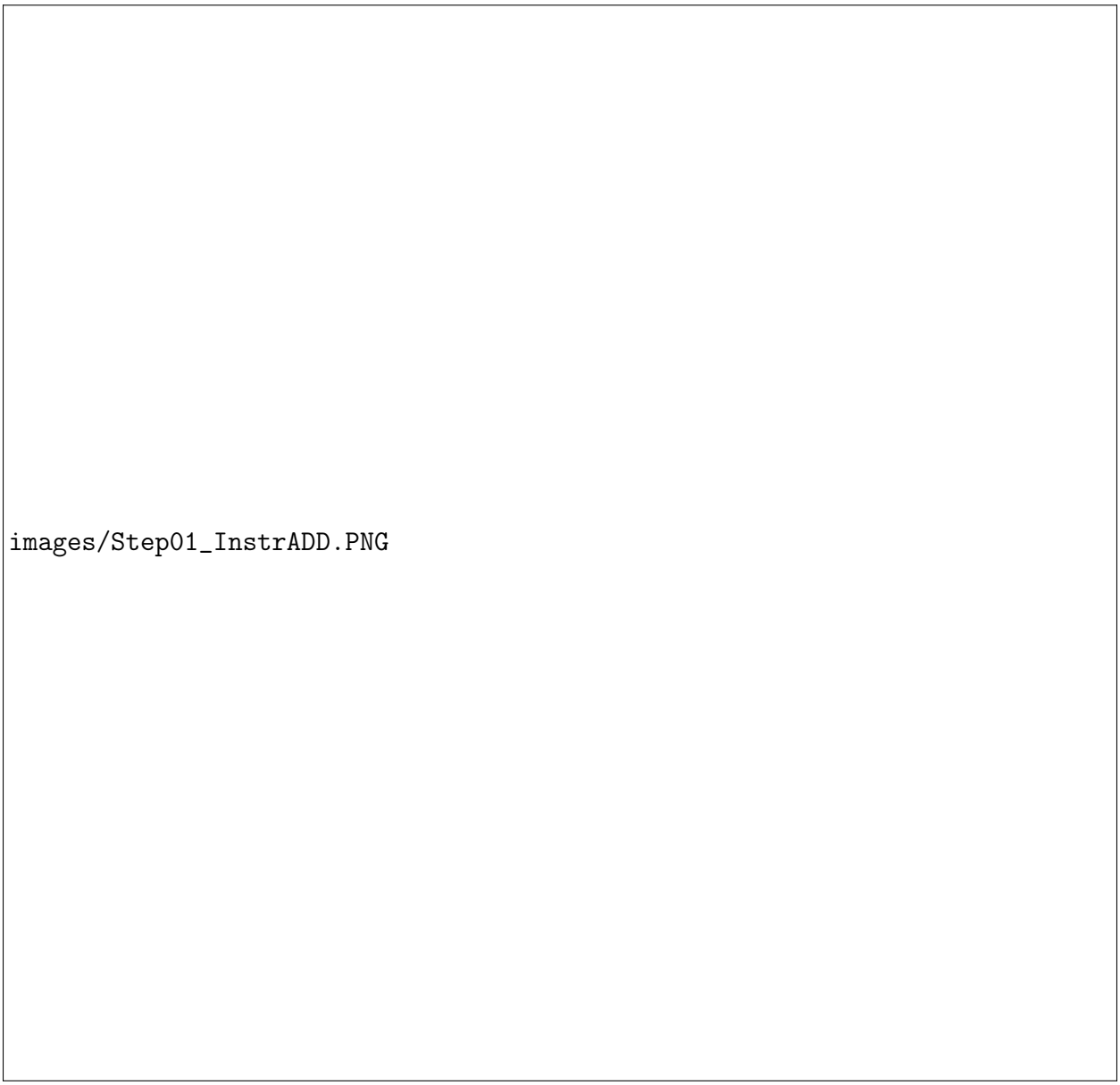

Figura 6: Instrução ADD R1, R0 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, a instrução chega ao registrador

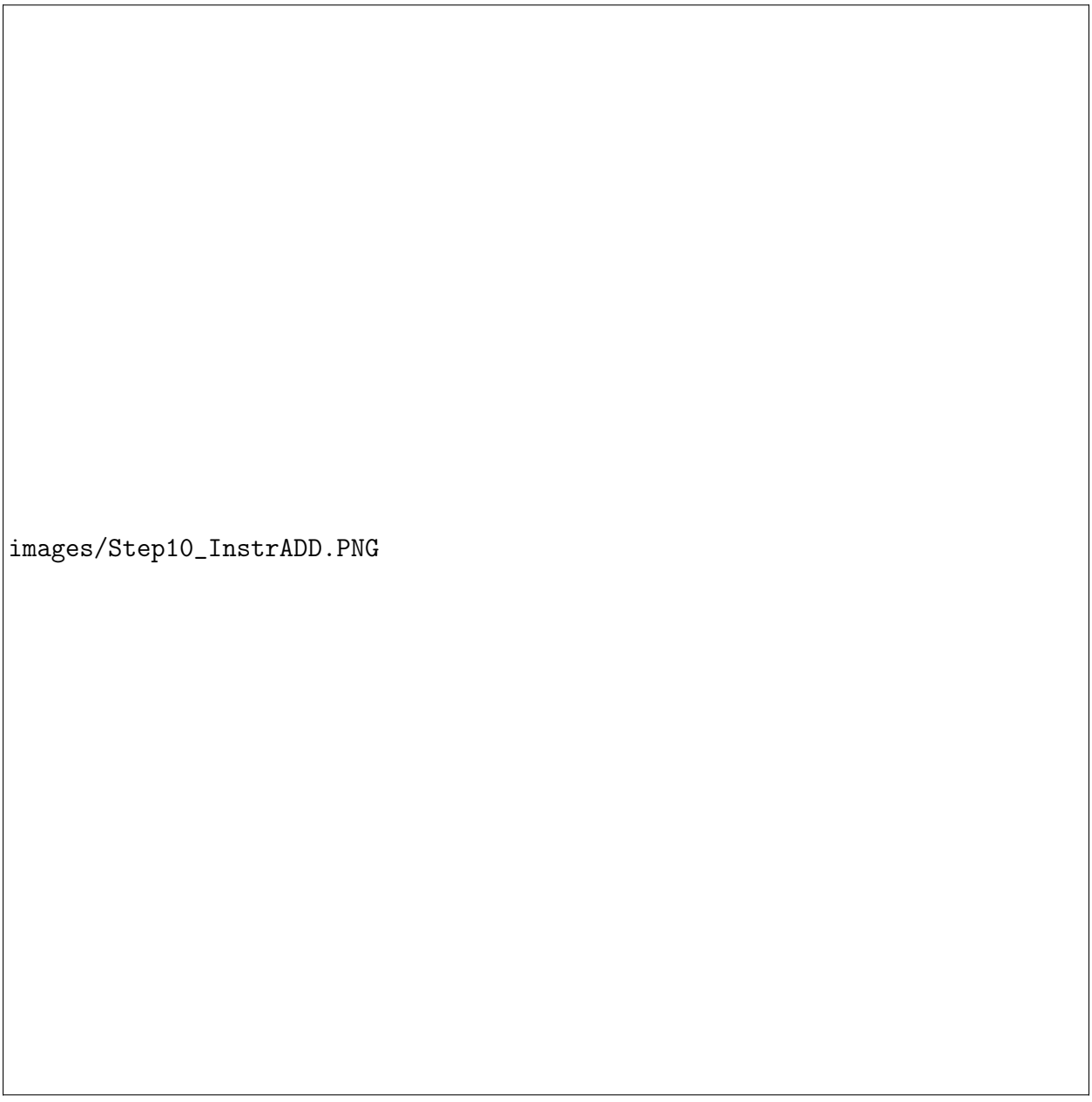

Figura 7: Instrução ADD R1, R0 - etapa 2 - O registrador A tem seu sinal de escrita habilitado para receber o dado de R1

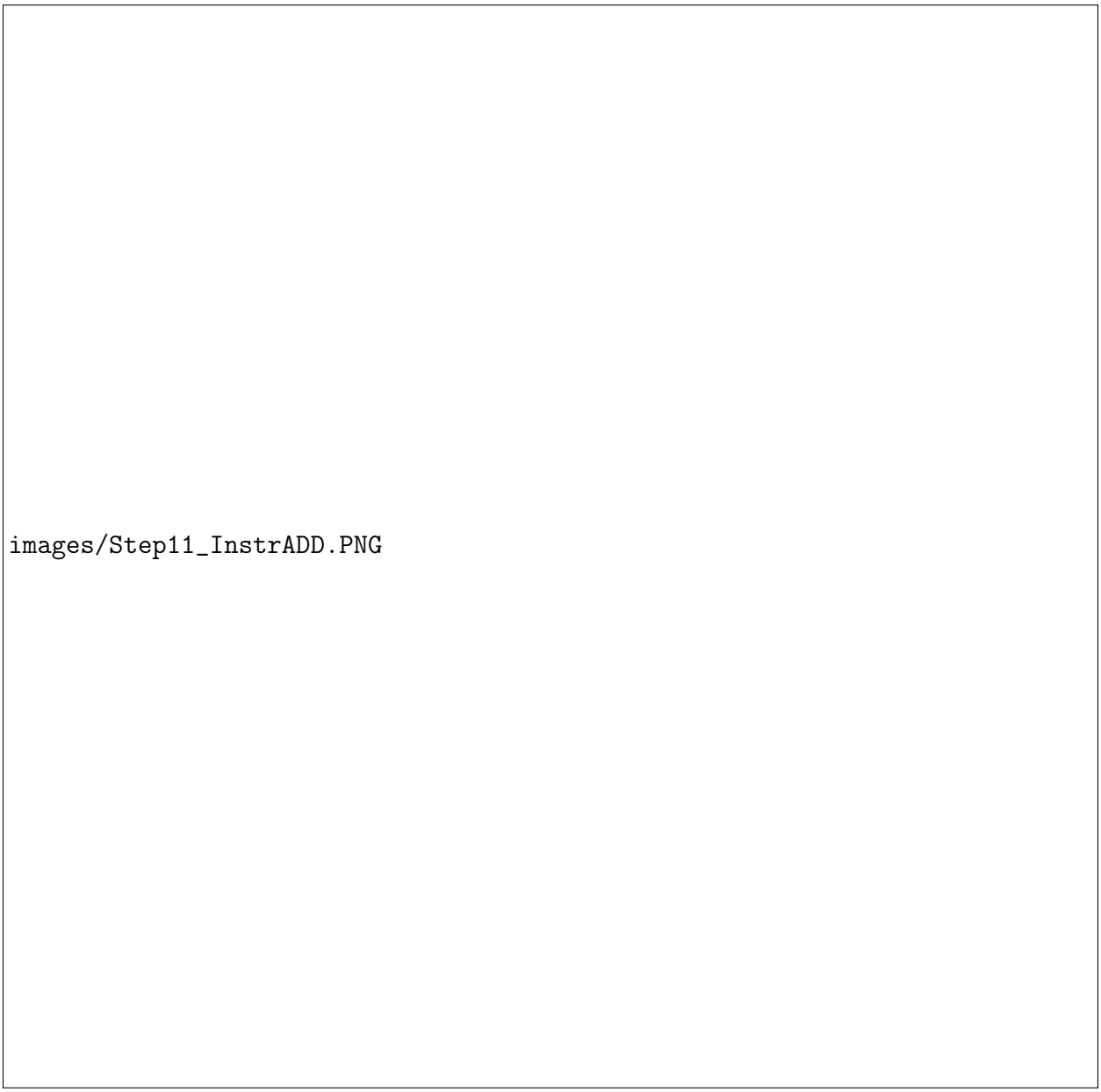

Figura 8: Instrução ADD R1, R0 - etapa 3 - A recebe o dado de R1, operação de soma é realizada e seu resultado é colocado no registrador G, que teve seu sinal de escrita habilitado

images/Step100_InstrADD.PNG

Figura 9: Instrução ADD R1, R0 - etapa 4 - o sinal de entrada de R1 e a saída de G no multiplexador são habilitados, o resultado da operação acaba em R1

SUB - 0010

images/InstrSUB.PNG

Figura 10: Instrução SUB R2, R1 - o registrador R2 recebe R2 - R1

images/Step00_InstrSUB.PNG

Figura 11: Instrução SUB R2, R1 - etapa 0 - o registrador de instruções se prepara para receber a instrução de subtração (sinal *IRin* habilitado), o *program counter* é incrementado (sinal *incr_pc* habilitado) e o ADDR é preparado para receber o endereço do próximo dado

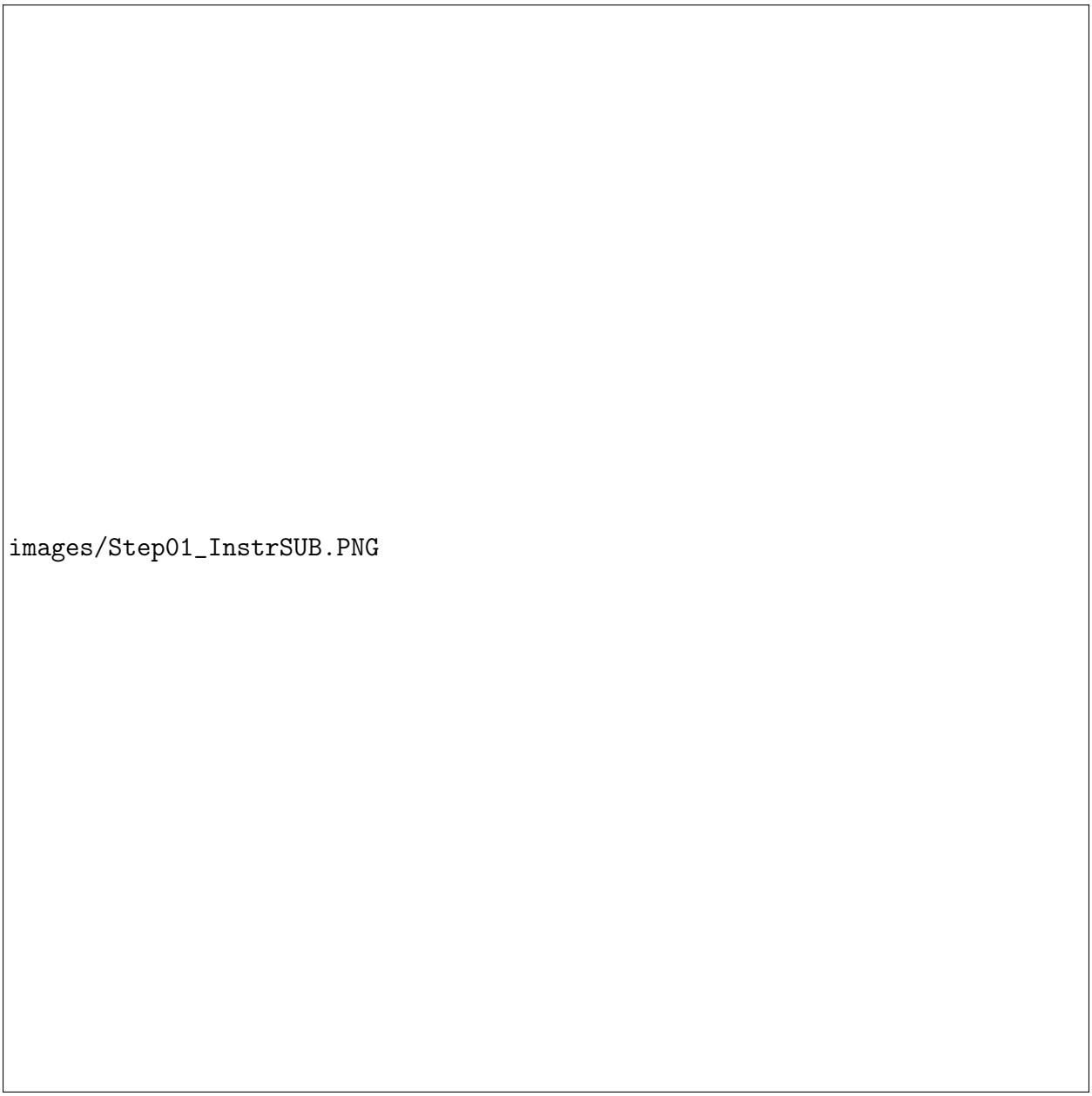

Figura 12: Instrução SUB R2, R1 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado e o registrador de instruções recebe a instrução de subtração

images/Step10_InstrSUB.PNG

Figura 13: Instrução SUB R2, R1 - etapa 2 - O registrador A tem seu sinal de escrita habilitado para receber o dado de R2

images/Step11_InstrSUB.PNG

Figura 14: Instrução SUB R2, R1 - etapa 3 - A recebe o dado de R2, a operação de subtração é realizada e seu resultado é colocado no registrador G, que teve seu sinal de escrita habilitado

Figura 15: Instrução SUB R2, R1 - etapa 4 - o sinal de entrada de R2 e a saída de G no multiplexador são habilitados, o resultado da operação acaba em R2

OR - 0011

images/InstrOR.PNG

Figura 16: Instrução OR R0, R3 - o registrador R0 recebe R0 | R3

Figura 17: Instrução OR R0, R3 - etapa 0 - o registrador de instruções se prepara para receber a instrução de OU (sinal *IRin* habilitado), o *program counter* é incrementado (sinal *incr_pc habilitado*) e o ADDR é preparado para receber o endereço do próximo dado

Figura 18: Instrução OR R0, R3 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, a instrução chega ao IR (*Instruction Register*)

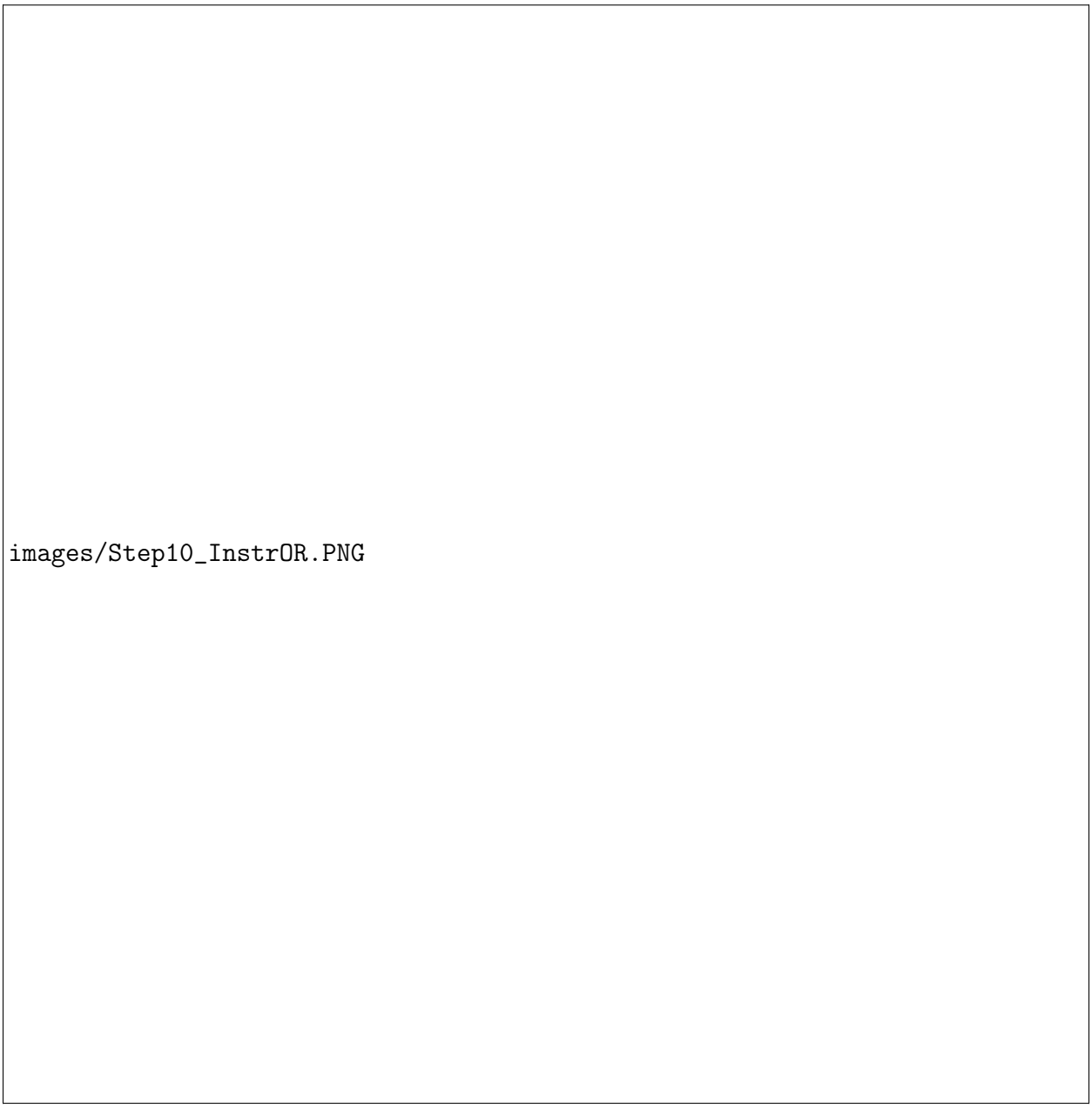

Figura 19: Instrução OR R0, R3 - etapa 2 - O registrador A teve seu sinal de escrita habilitado para receber o conteúdo de R0

images/Step11_InstrOR.PNG

Figura 20: Instrução OR R0, R3 - etapa 3 - O dado de R0 é colocado em A, a operação é realizada e seu resultado é colocado no registrador G, que teve seu sinal de escrita habilitado

images/Step100_InstrOR.PNG

Figura 21: Instrução OR R0, R3 - etapa 4 - o sinal de entrada de R0 e a saída de G no multiplexador são habilitados, o resultado da operação acaba em R0

SLL - 0100

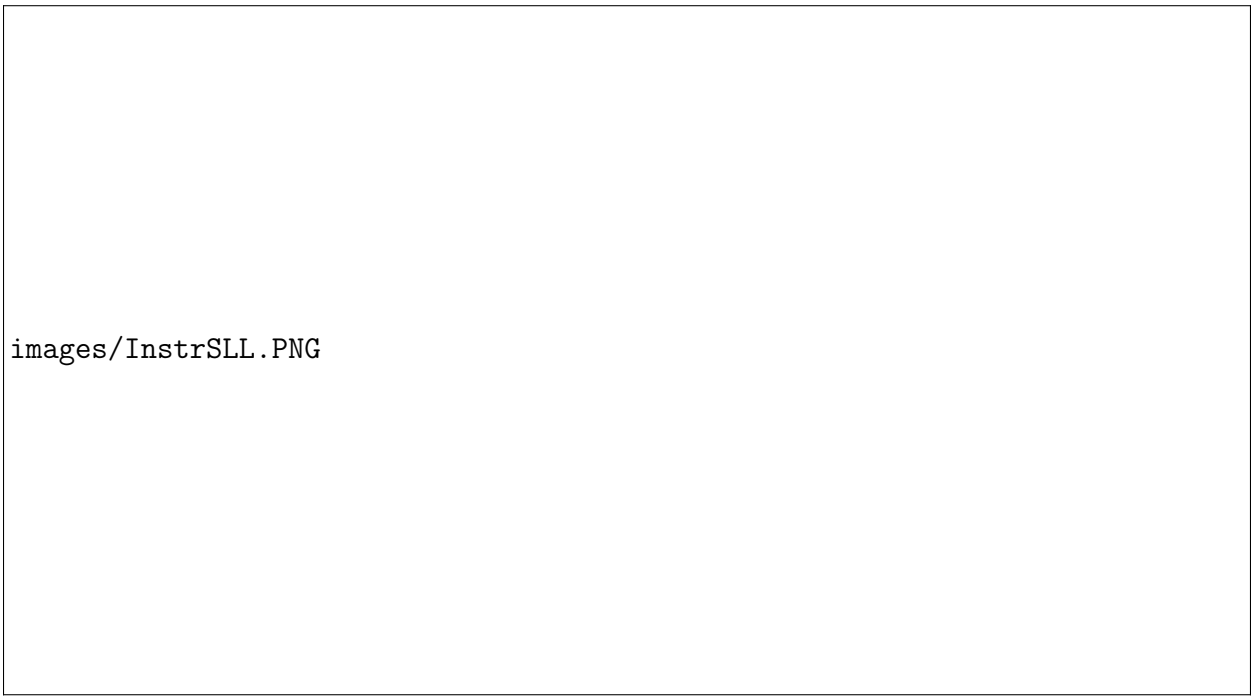

Figura 22: Instrução SLL R0, R3 - o registrador R0 recebe R0 « R3

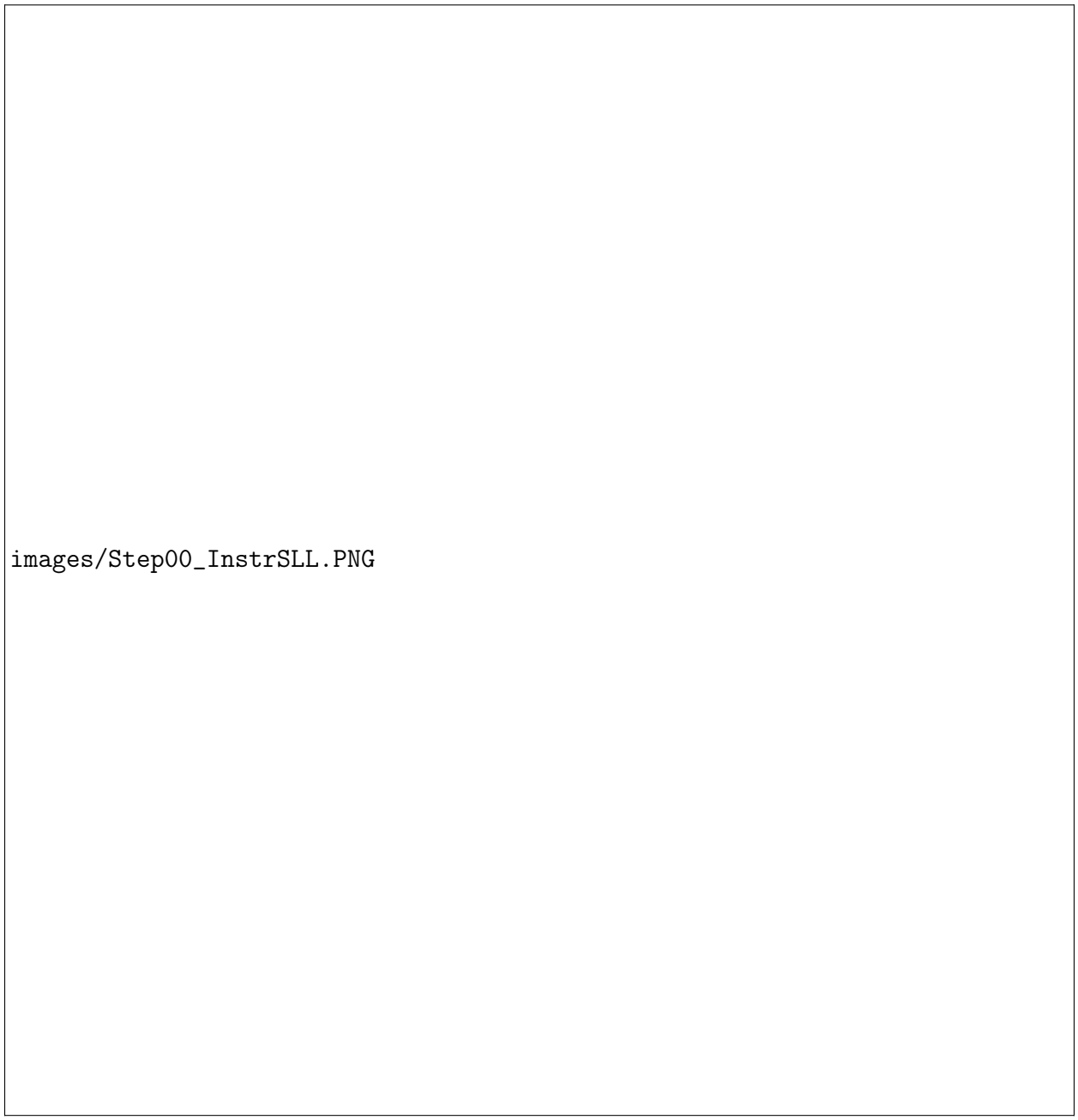

Figura 23: Instrução SLL R0, R3 - etapa 0 - o registrador de instruções recebe a instrução de *shift* à esquerda, o *program counter* é incrementado (sinal *incr_pc*) e o ADDR é preparado para receber o endereço do próximo dado

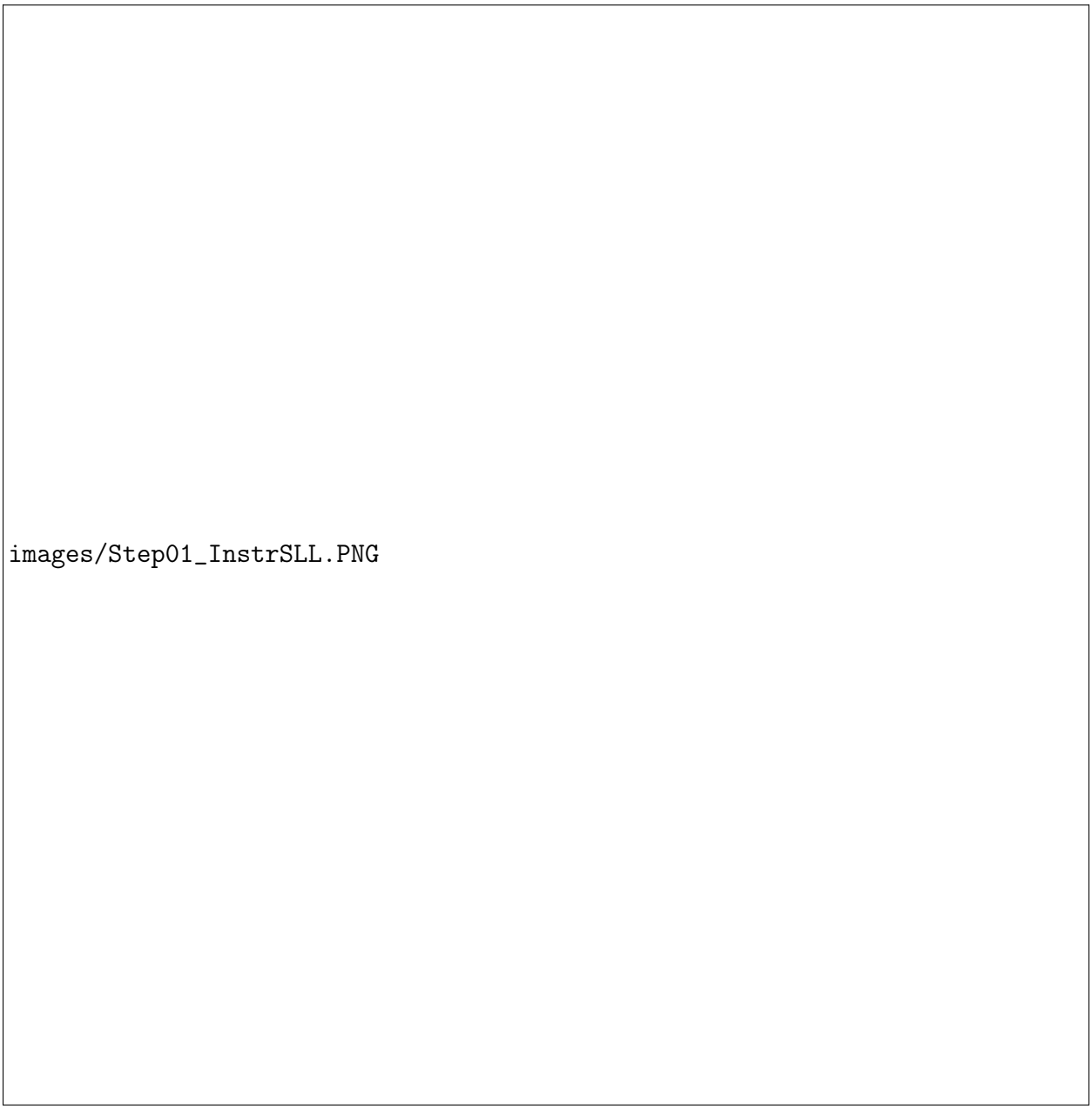

Figura 24: Instrução SLL R0, R3 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, a instrução chega ao IR

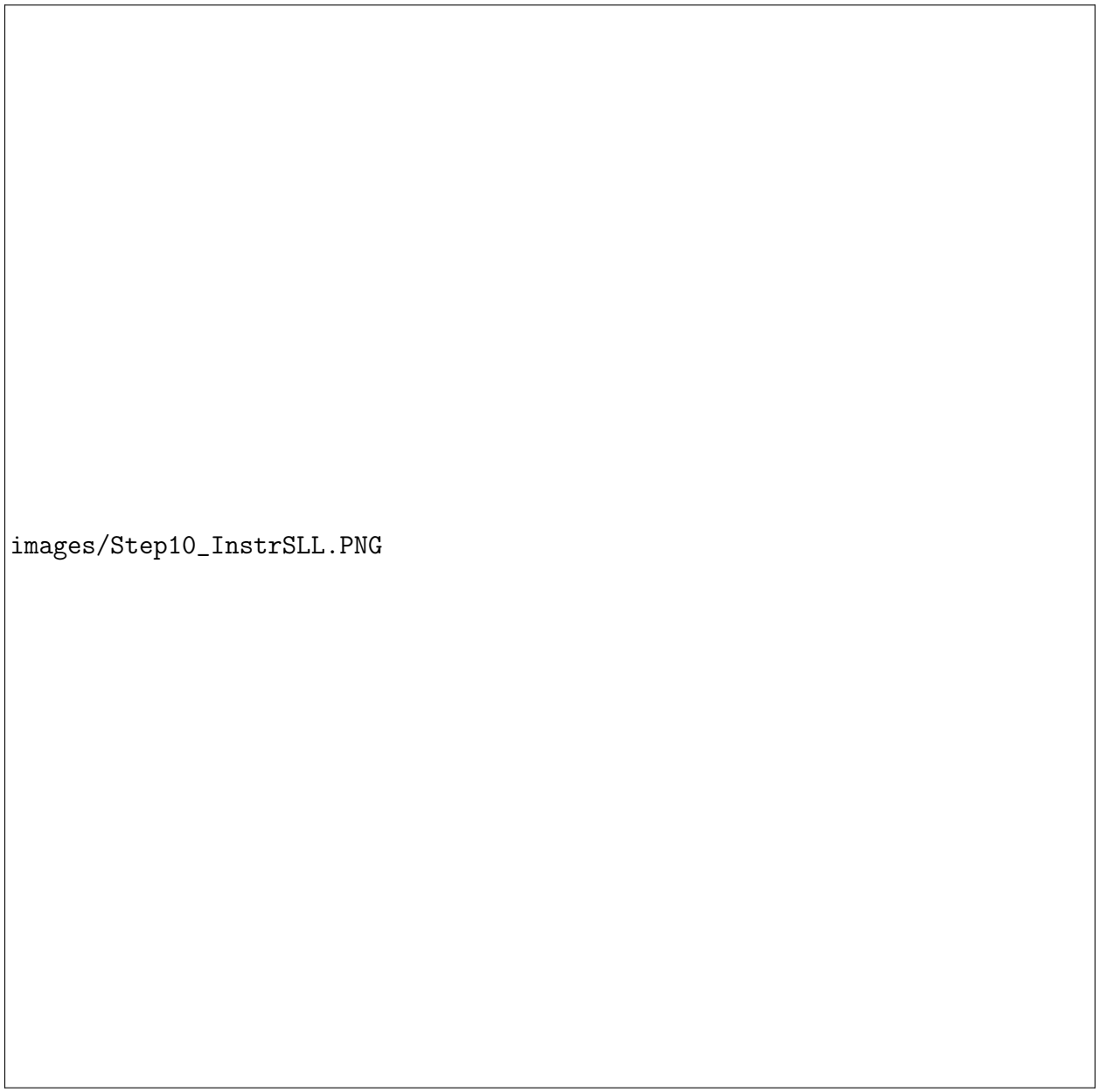

Figura 25: Instrução SLL R0, R3 - etapa 2 - O dado de R0 é colocado no registrador A, que teve seu sinal de escrita habilitado

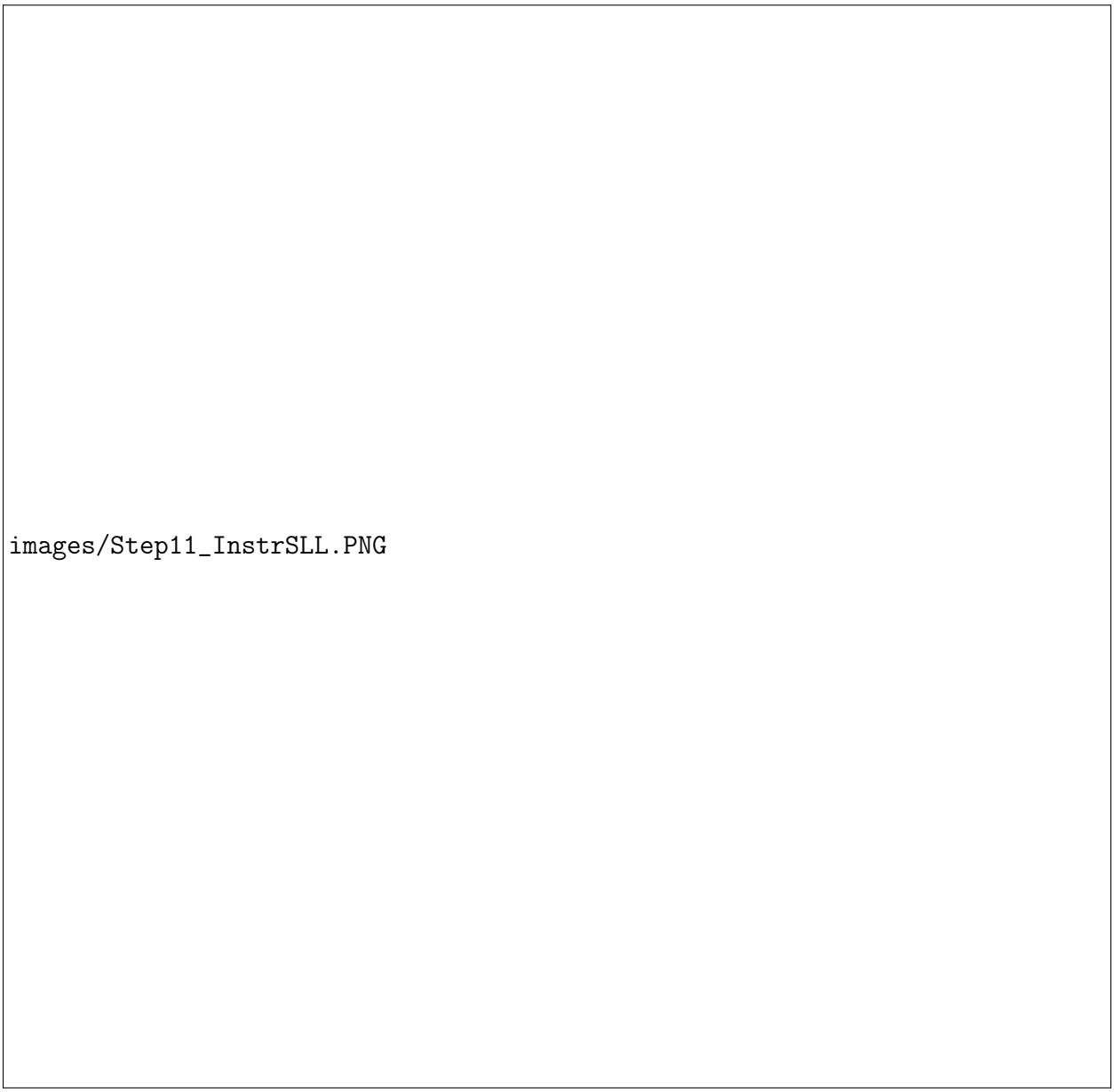

Figura 26: Instrução SLL R0, R3 - etapa 3 - A operação é realizada e seu resultado é colocado no registrador G, que teve seu sinal de escrita habilitado

Figura 27: Instrução SLL R1, R0 - etapa 4 - o sinal de entrada de R1 e a saída de G no multiplexador são habilitados, o resultado da operação acaba em R1

SRL - 0101

Figura 28: Instrução SRL R0, R3 - o registrador R0 recebe R0 » R3

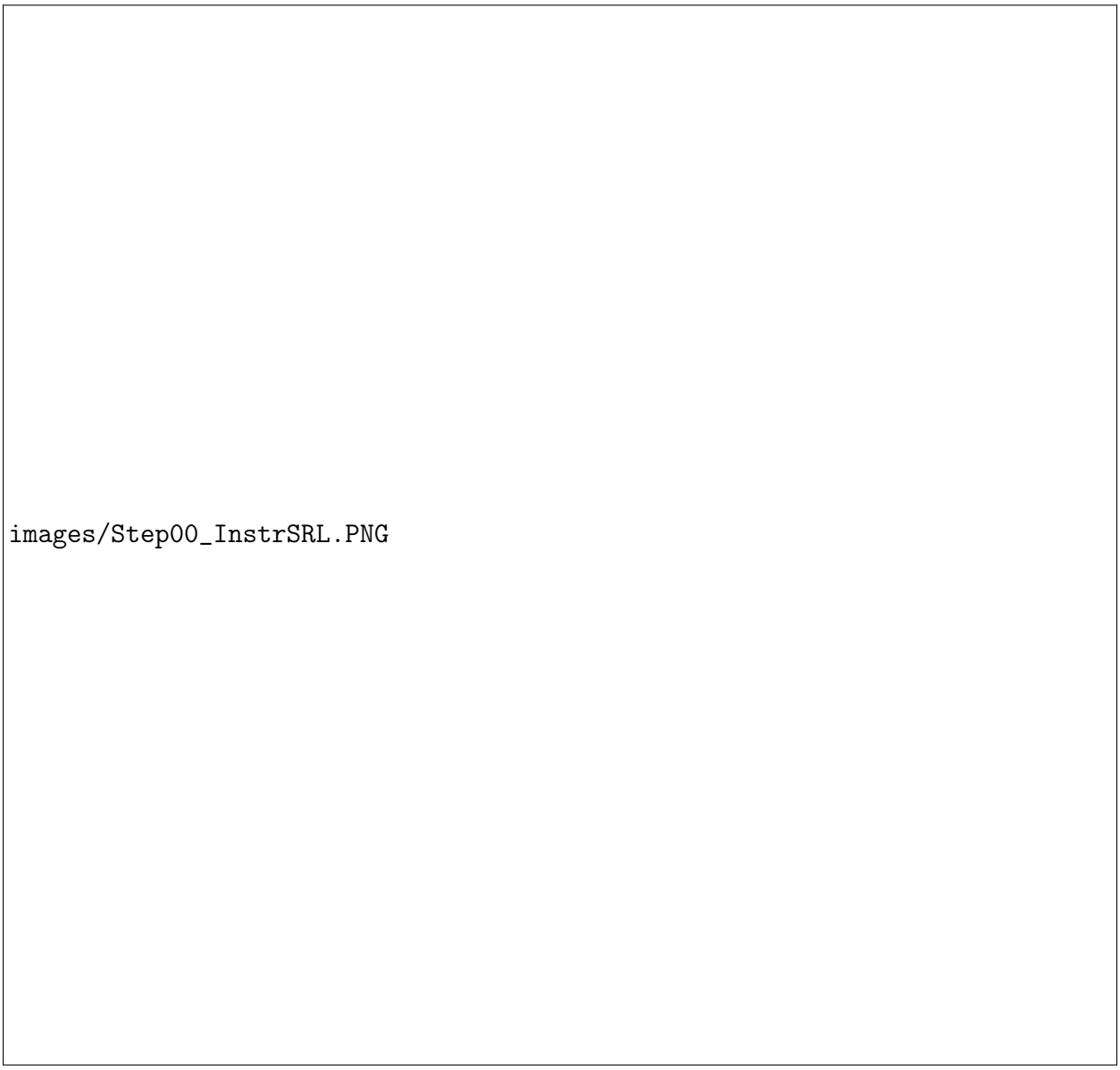

Figura 29: Instrução SRL R0, R3 - etapa 0 - o registrador de instruções se prepara receber a instrução de *shift* à direita (*IRin* ativado), o *program counter* é incrementado (sinal *incr_pc*) o ADDR é preparado para receber o endereço do próximo dado

Figura 30: Instrução SRL R1, R0 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, a instrução chega ao IR

images/Step10_InstrSRL.PNG

Figura 31: Instrução SRL R0, R3 - etapa 2 - A é preparado para receber o dado de R0

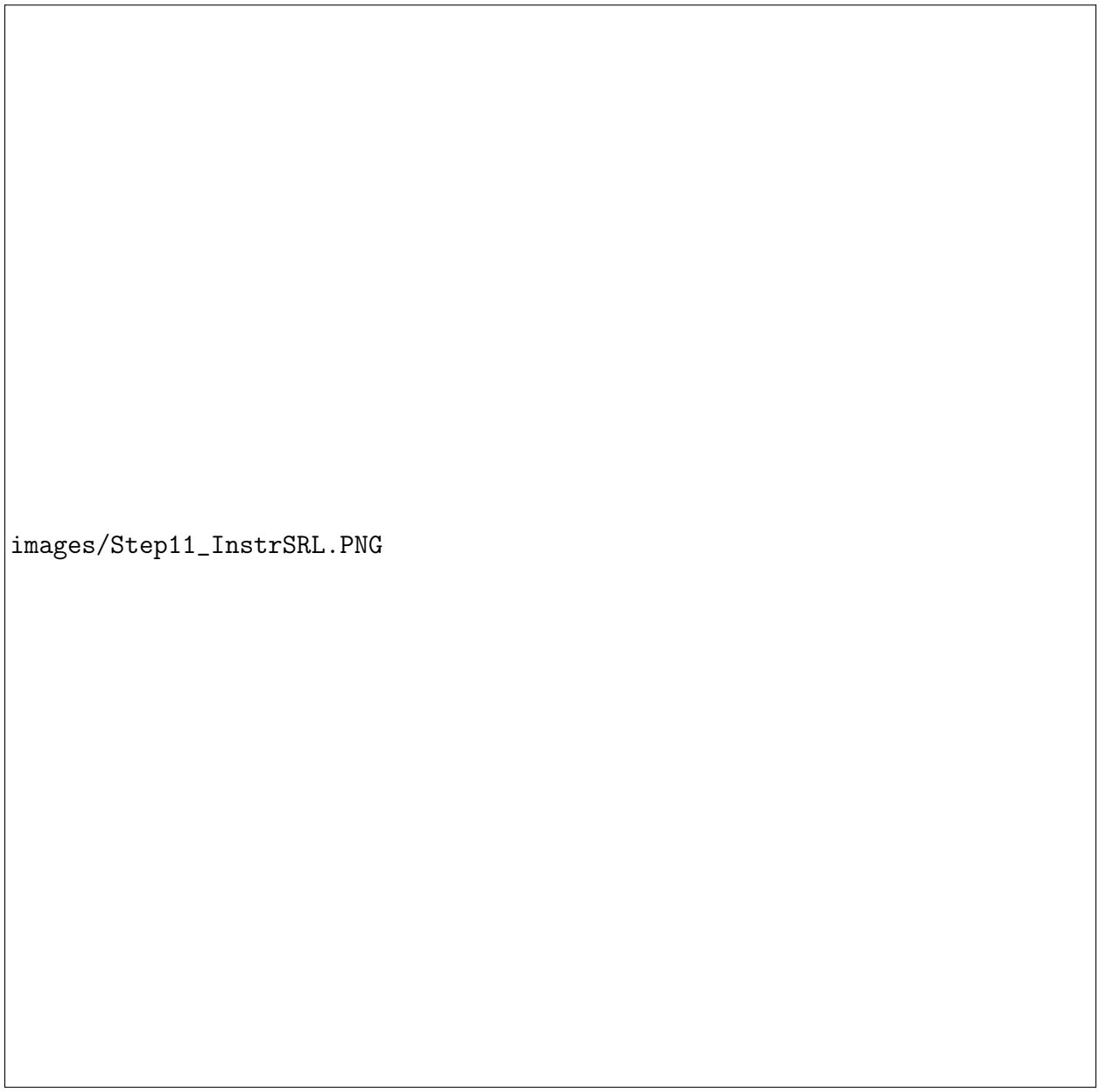

Figura 32: Instrução SRL R0, R3 - etapa 3 - A recebe o fado de R0, a operação é realizada e seu resultado é colocado no registrador G, que teve seu sinal de escrita habilitado

Figura 33: Instrução SRL R0, R3 - etapa 4 - o sinal de entrada de R0 e a saída de G no multiplexador são habilitados, o resultado da operação acaba em R0

SLT - 0110

images/InstrSLT.PNG

Figura 34: Instrução SLT R0, R1 - o registrador R0 recebe R0 < R1

Figura 35: Instrução SLT R0, R1 - etapa 0 - o registrador de instruções se prepara para receber a instrução e *program counter* é incrementado (sinal $incr_pc$) e o ADDR é preparado para receber o endereço do próximo dado

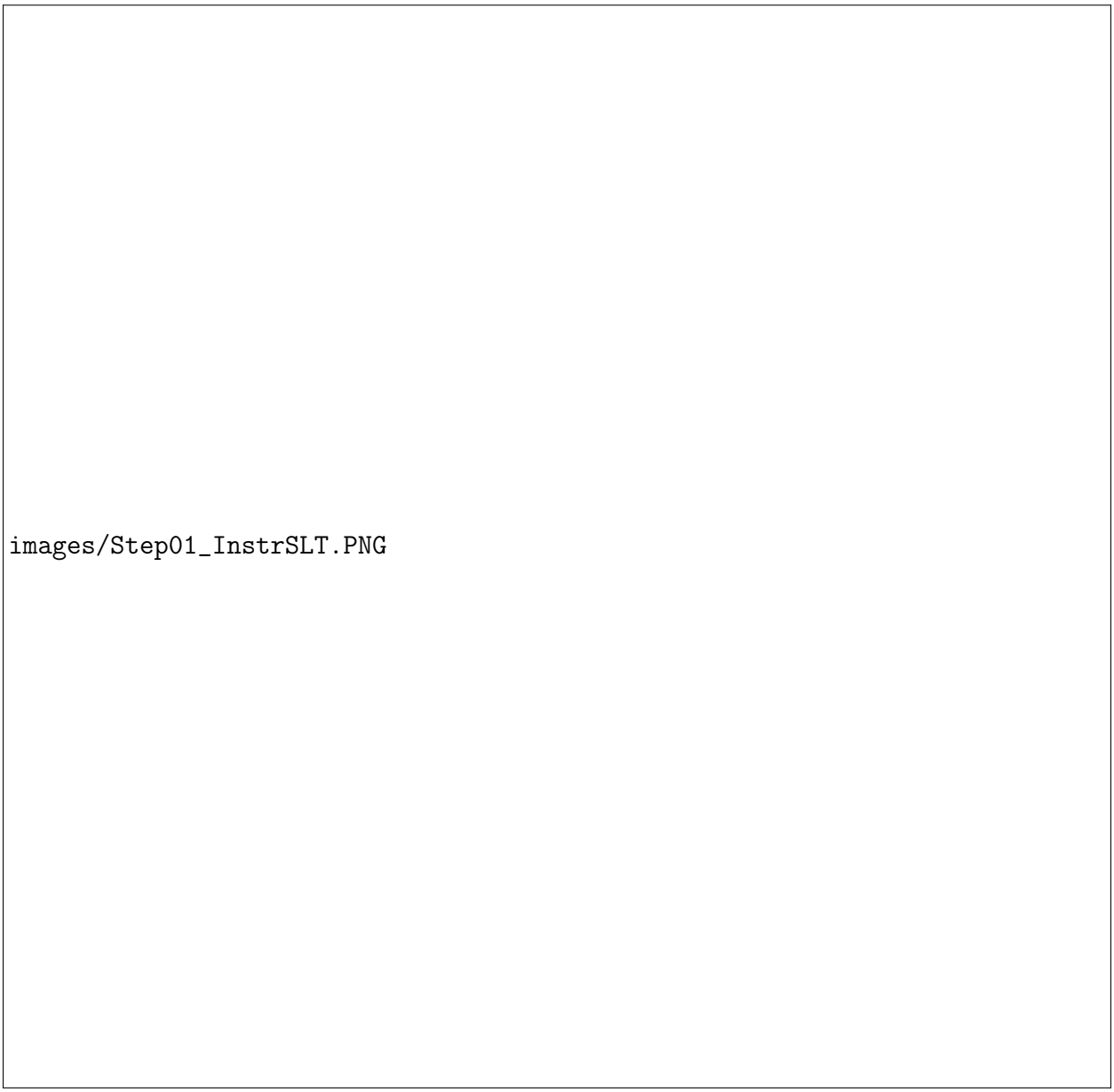

Figura 36: Instrução SLT R0, R1 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, a instrução chega ao IR

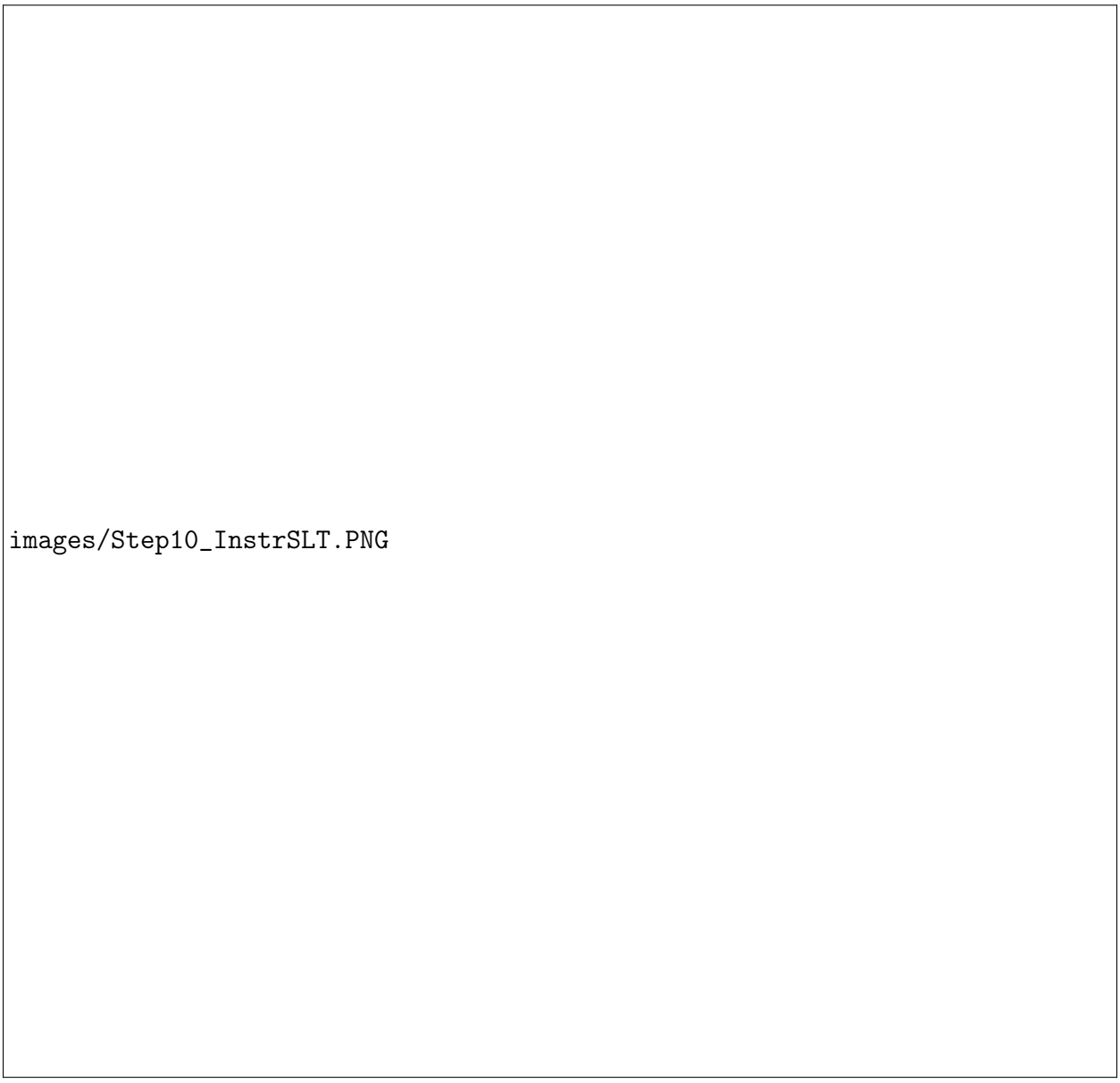

Figura 37: Instrução SLT R0, R1 - etapa 2 - A é preparado para receber o dado de R0

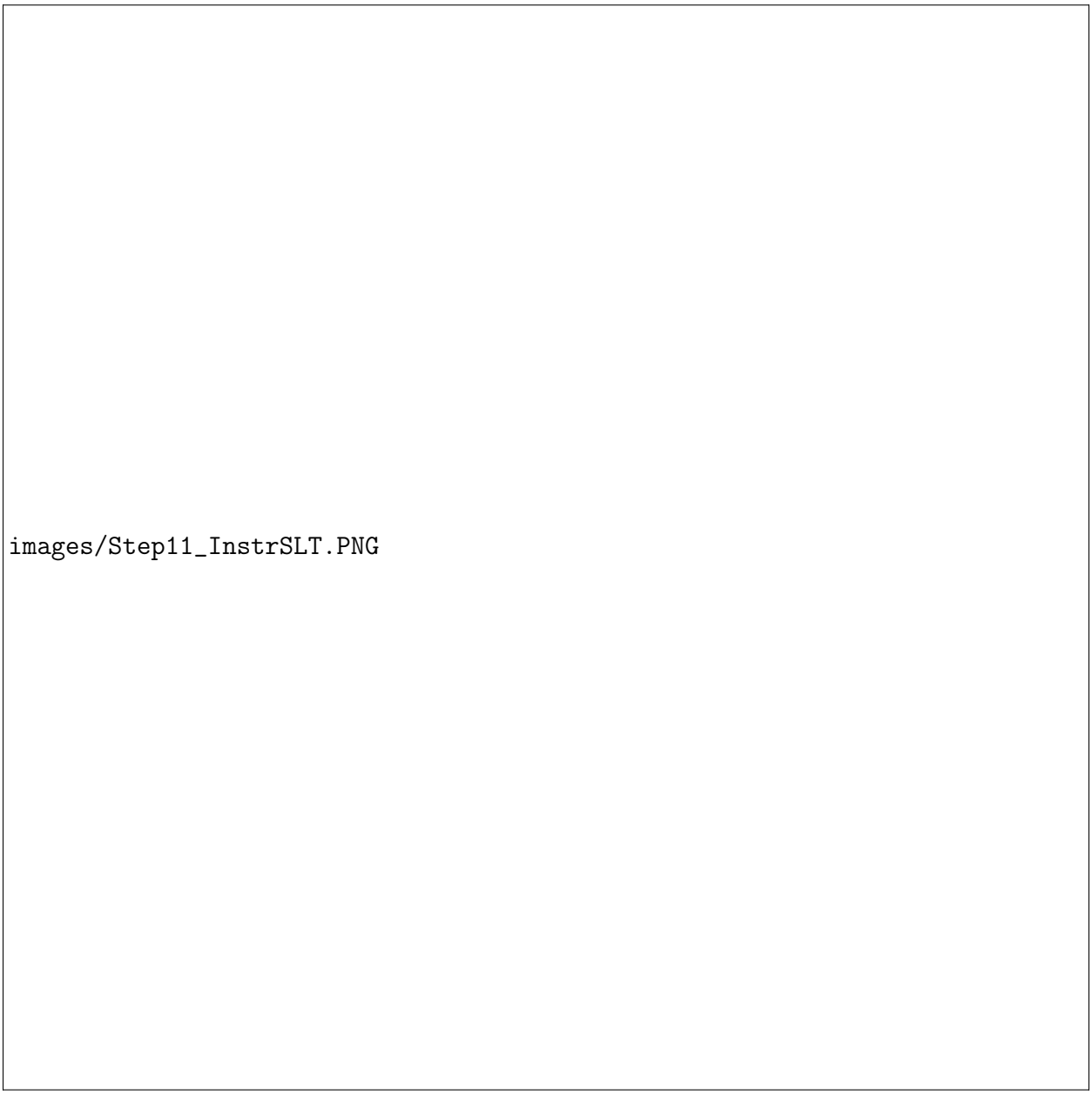

Figura 38: Instrução SLT R0, R1 - etapa 3 - O registrador A, recebe o dado de R0, a operação é realizada e seu resultado é colocado no registrador G, que teve seu sinal de escrita habilitado

Figura 39: Instrução SLT R0, R1 - etapa 4 - o sinal de entrada de R0 e a saída de G no multiplexador são habilitados, o resultado da operação acaba em R0

MVI - 0111

images/InstrMVI.PNG

Figura 40: Instrução MVI R0, 2 - o registrador R0 recebe 2

images/Step00_InstrMVI.PNG

Figura 41: Instrução MVI R0, 2 - etapa 0 - o registrador de instruções é preparado para receber a instrução e o *program counter* é incrementado (sinal *incr_pc*) e o ADDR é preparado para receber o endereço do próximo dado

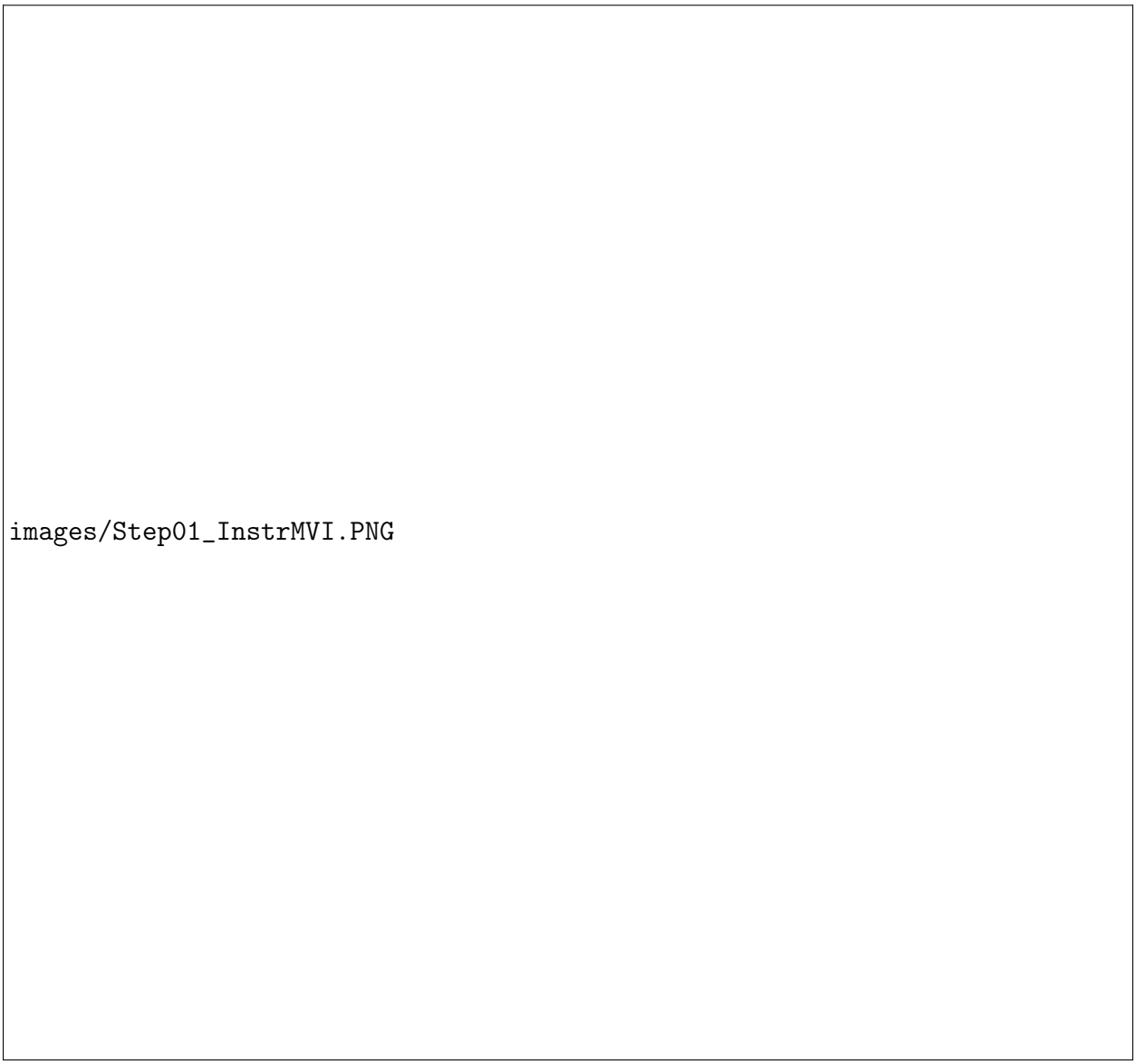

Figura 42: Instrução MVI R0, 2 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, IR recebe a instrução

images/Step10_InstrMVI.PNG

Figura 43: Instrução MVI R0, 2 - etapa 2 - PC é incrementado para buscar a próximo dado (o imediato, no caso)

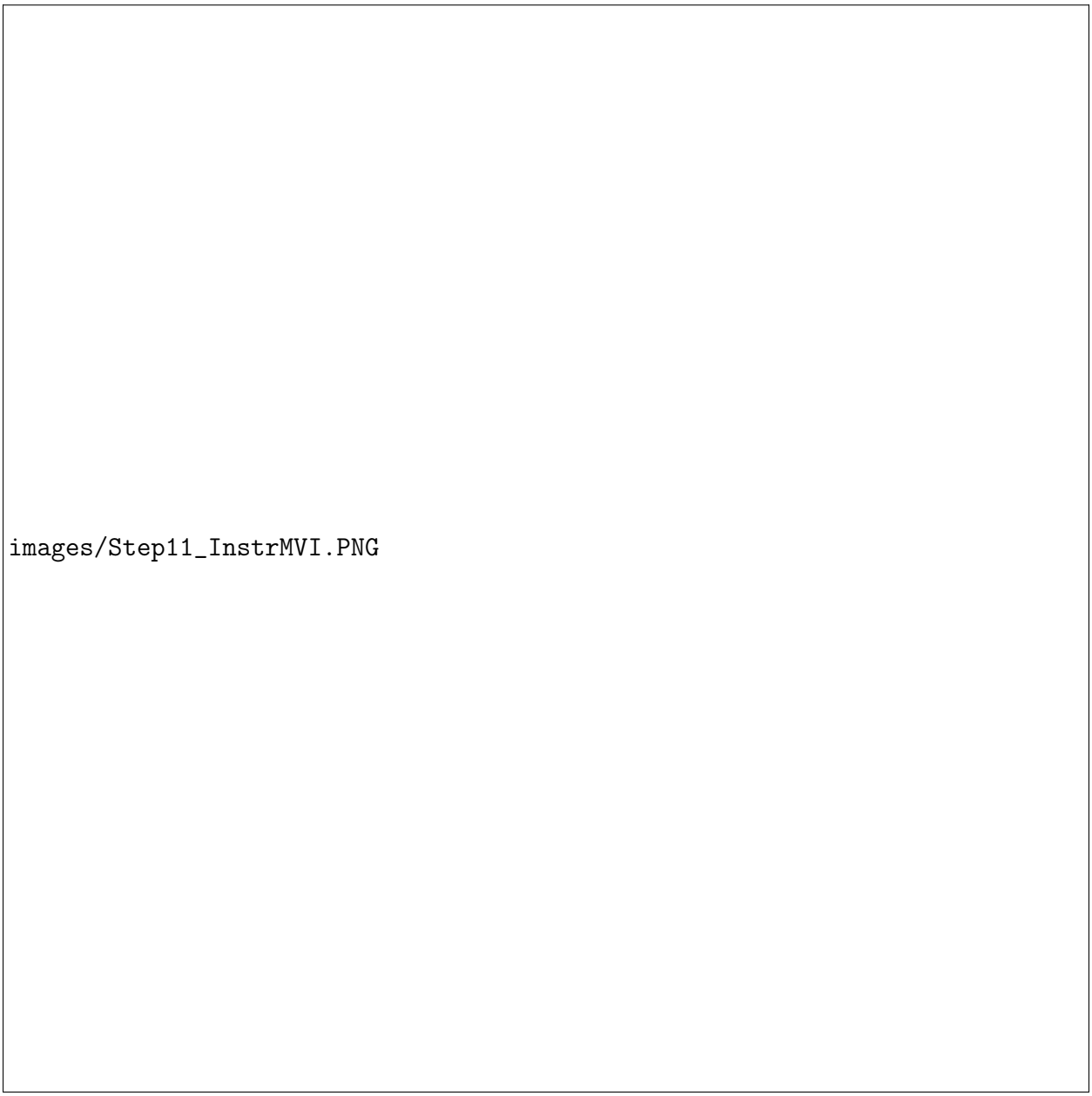

Figura 44: Instrução MVI R0, 2 - etapa 3 - O registrador de endereços recebe o endereço do imediato

images/Step100_InstrMVI.PNG

Figura 45: Instrução MVI R0, 2 - etapa 4 - A chave seletora do mux, DIN, é habilitada e o resultado da operação acaba em R0

MV - 1000

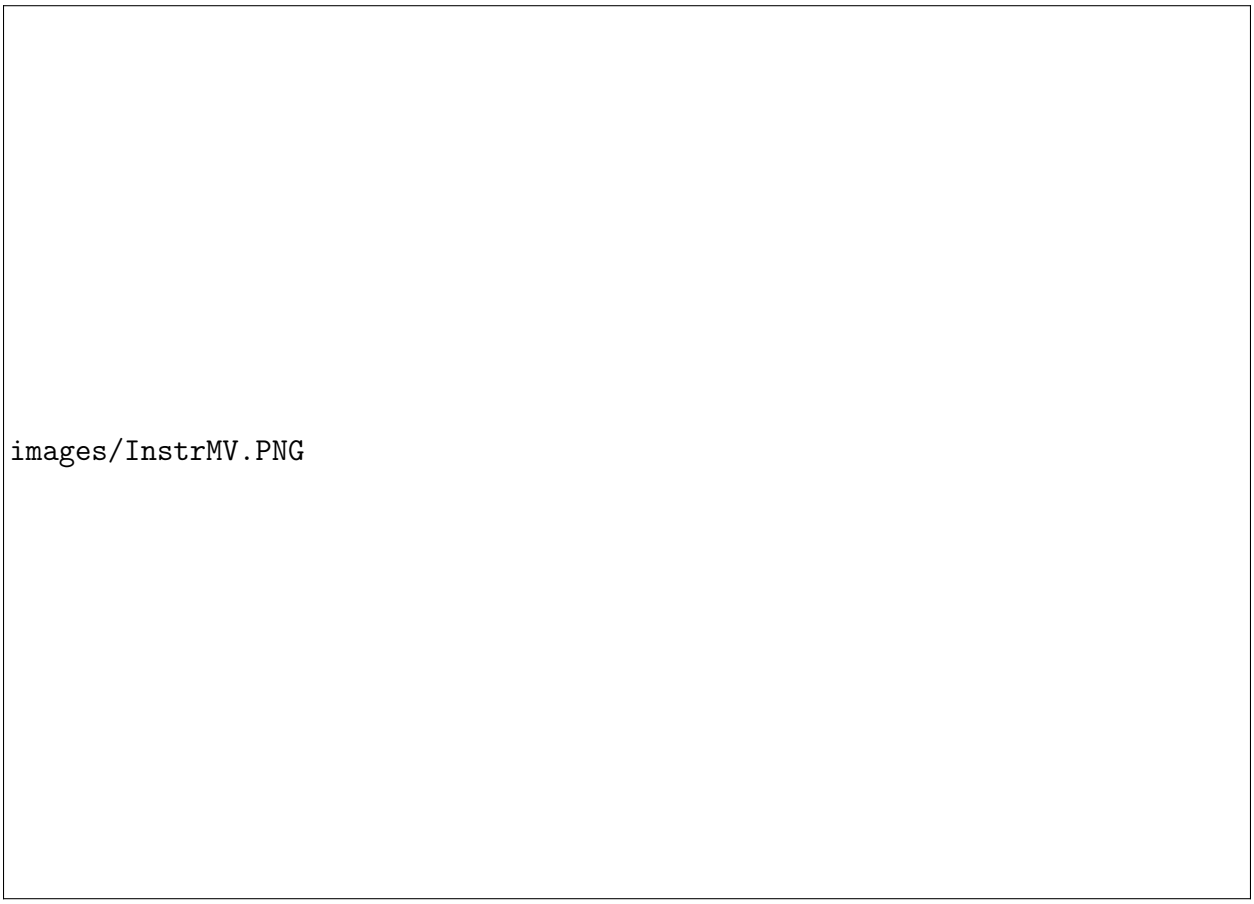

Figura 46: Instrução MV R3, R2 - o registrador R3 recebe o conteúdo armazenado em R2

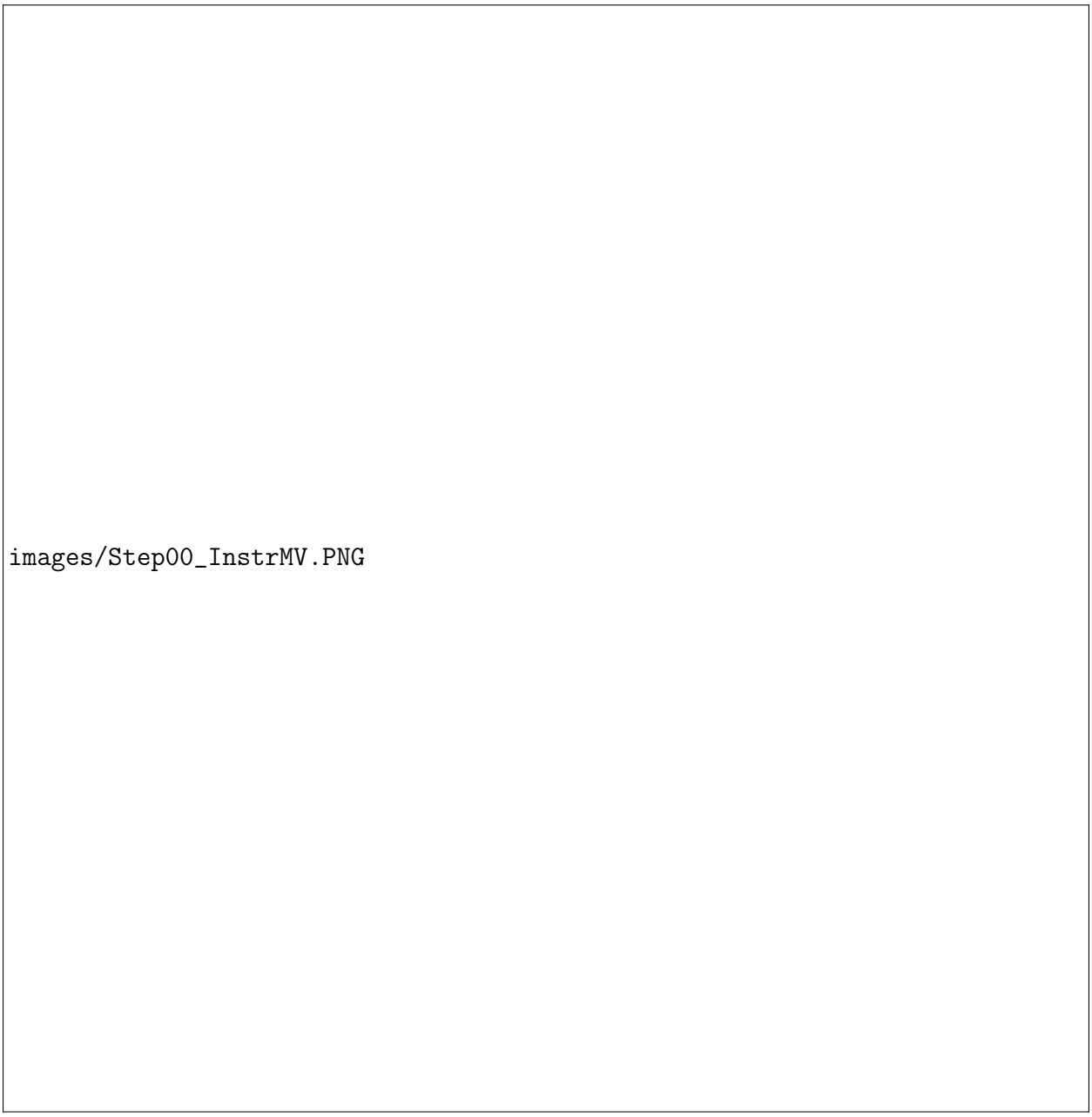

Figura 47: Instrução MV R3, R2 - etapa 0 - o registrador de instruções é preparado para receber a instrução e *program counter* é incrementado (sinal *incr_pc*) e o ADDR é preparado para receber o endereço do próximo dado

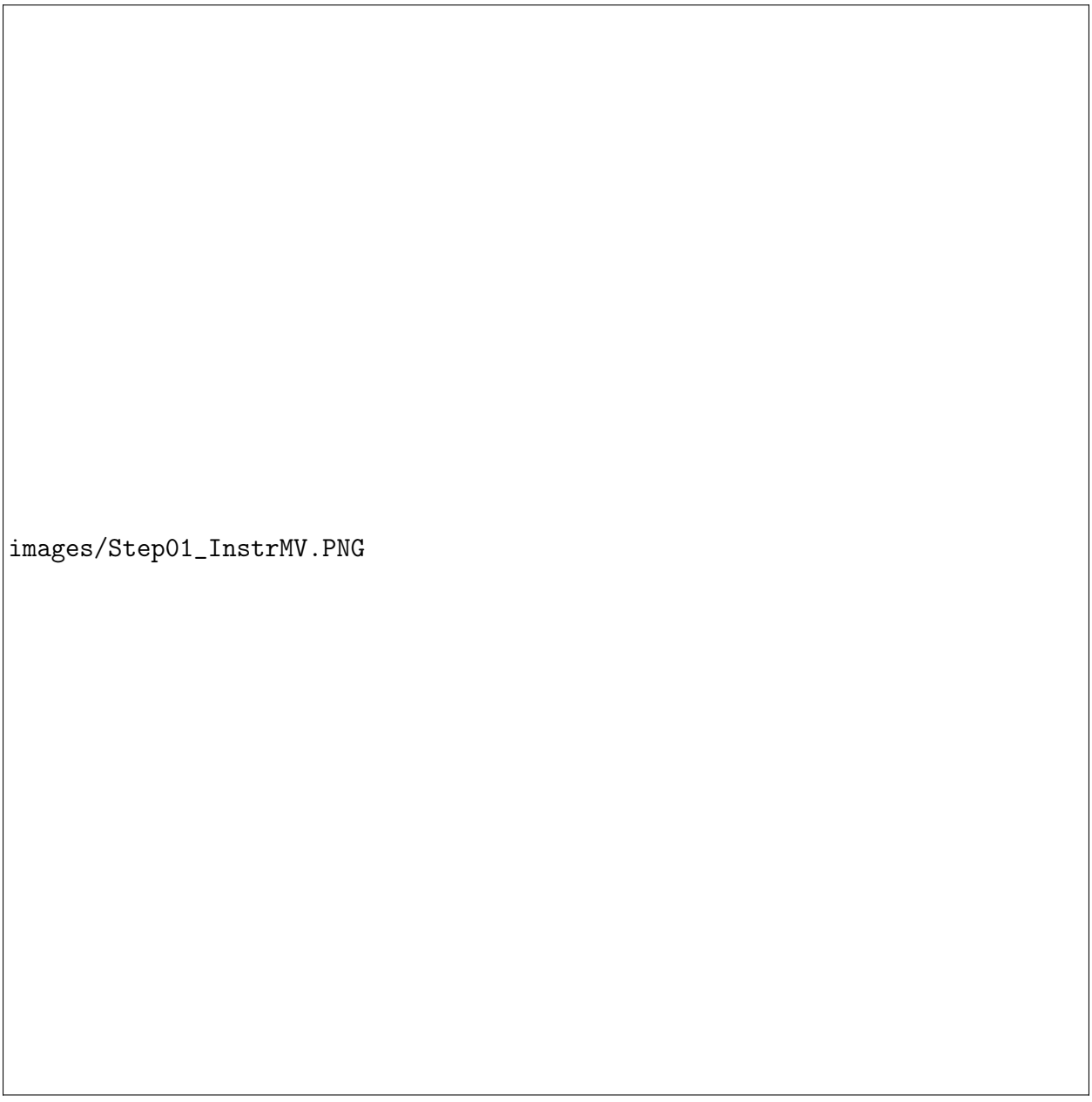

Figura 48: Instrução MV R3, R2 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, a instrução chega ao IR

Figura 49: Instrução MV R3, R2 - etapa 2 - O dado de R2 é selecionado pelo MUX e escrito em R3, que teve seu sinal de escrita habilitado

MVNZ - 1001

Figura 50: Instrução MVNZ R0, R2 - o registrador R0 recebe R2 caso outG seja diferente de zero

Figura 51: Instrução MVNZ R0, R2- etapa 0 - o registrador de instruções é preparado para receber a instrução e o *program counter* é incrementado (sinal $incr_pc$) e o ADDR é preparado para receber o endereço do próximo dado

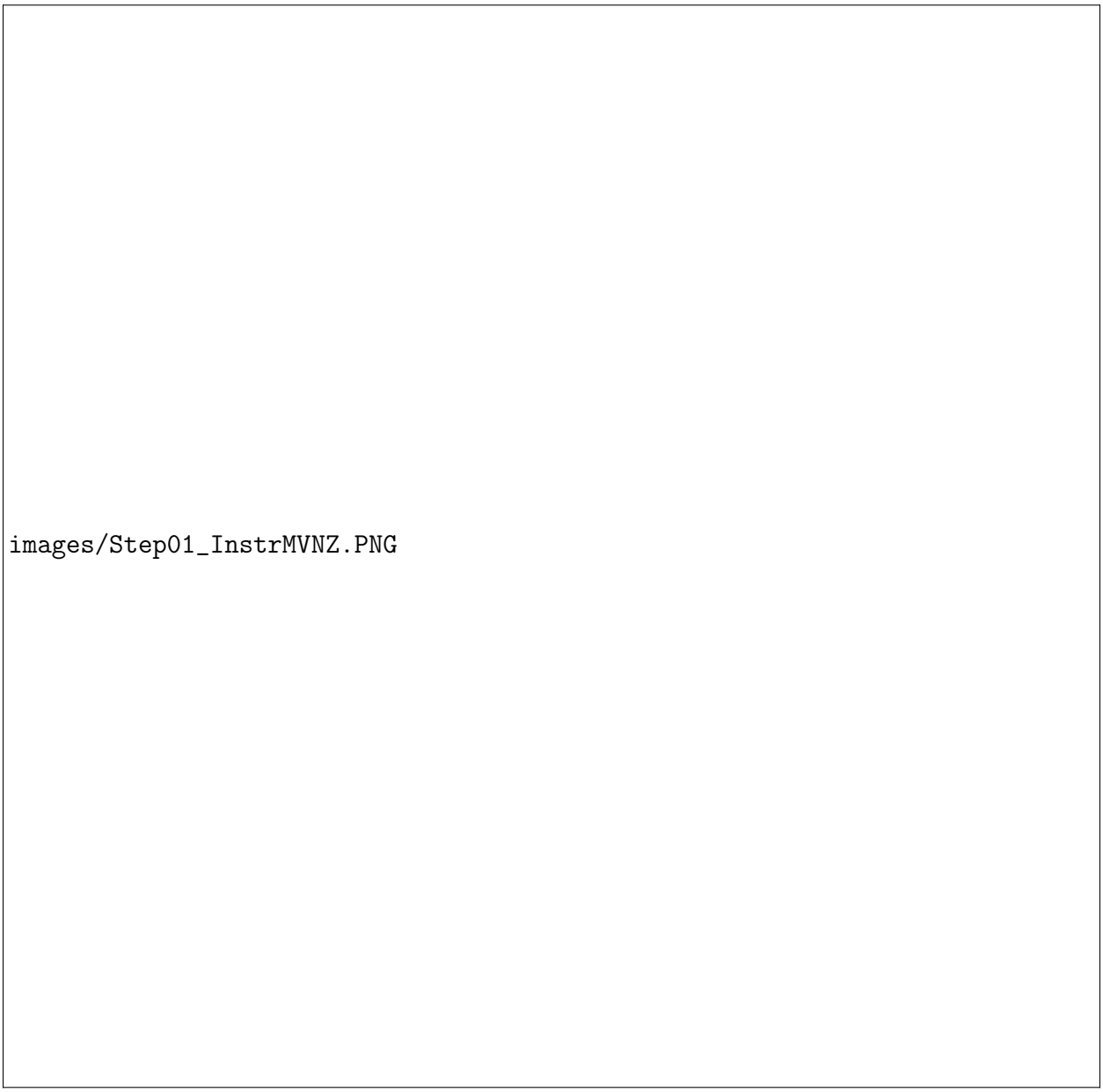

Figura 52: Instrução MVNZ R0, R2 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, a instrução chega ao IR

Figura 53: Instrução MVNZ R0, R2- etapa 2 - O dado de R2 é colocado em R0 já que outG não era zero

ST - 1010

images/InstrST.PNG

Figura 54: Instrução ST R2, R0 - a posição de memória indicada por R0 recebe R2

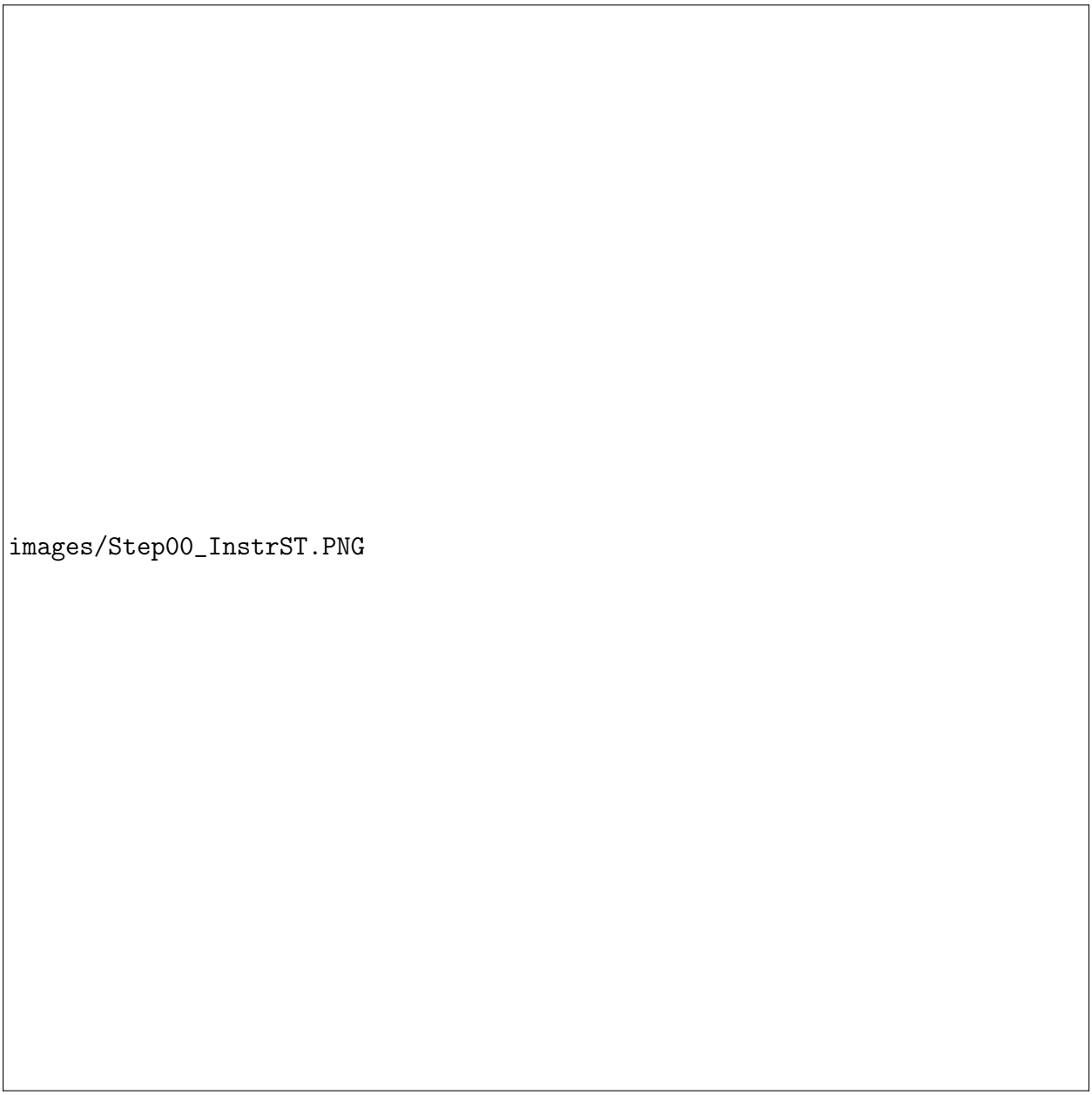

Figura 55: Instrução ST R2, R0- etapa 0 - o registrador de instruções recebe a instrução, o *program counter* é incrementado (sinal $incr_pc$) e o ADDR é preparado para receber o endereço do próximo dado

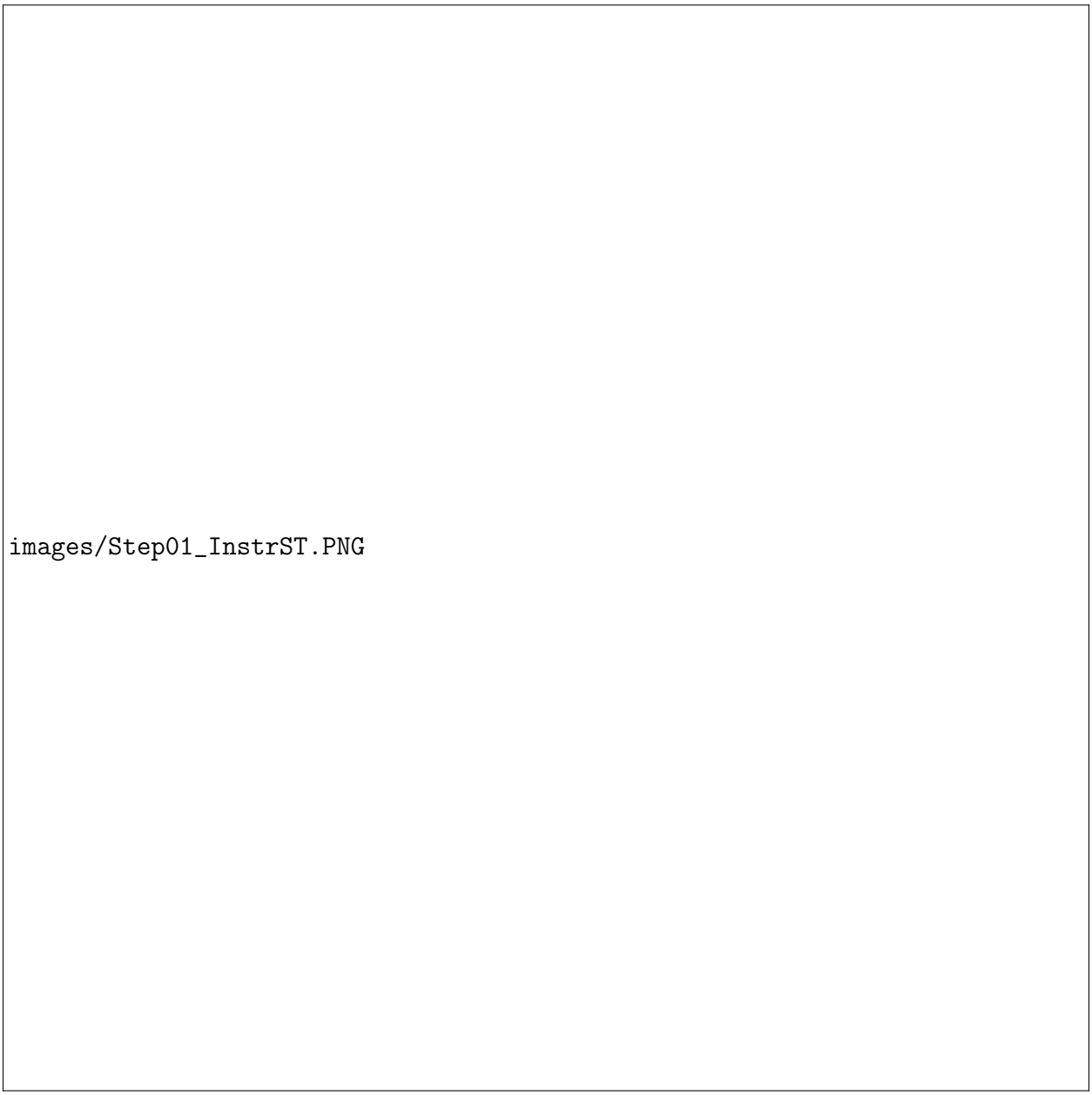

Figura 56: Instrução ST R2, R0 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado, a instrução chega ao IR

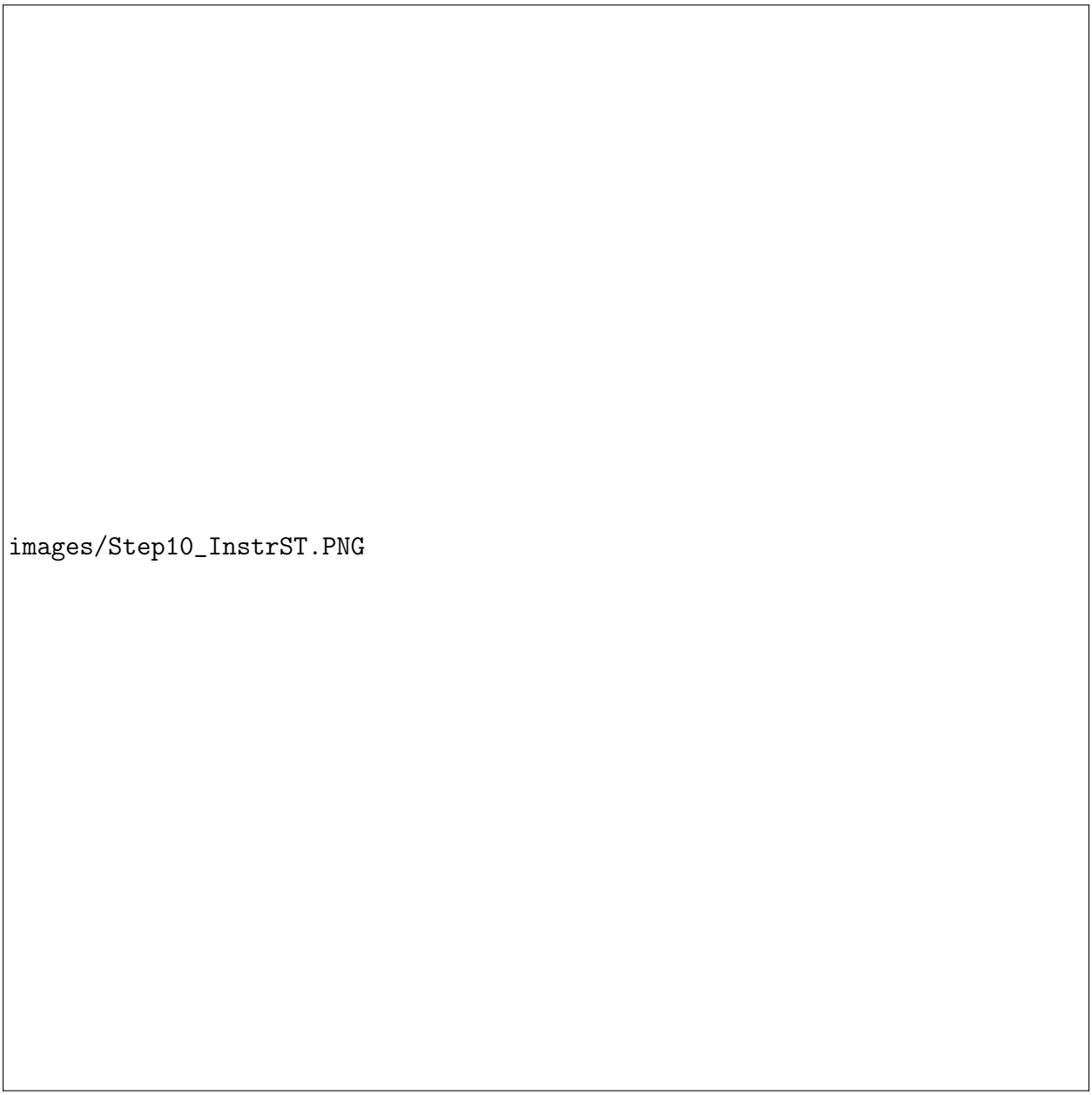

Figura 57: Instrução ST R2, R0 - etapa 2 - O dado de R0 é preparado para ser colocado no registrador de endereços

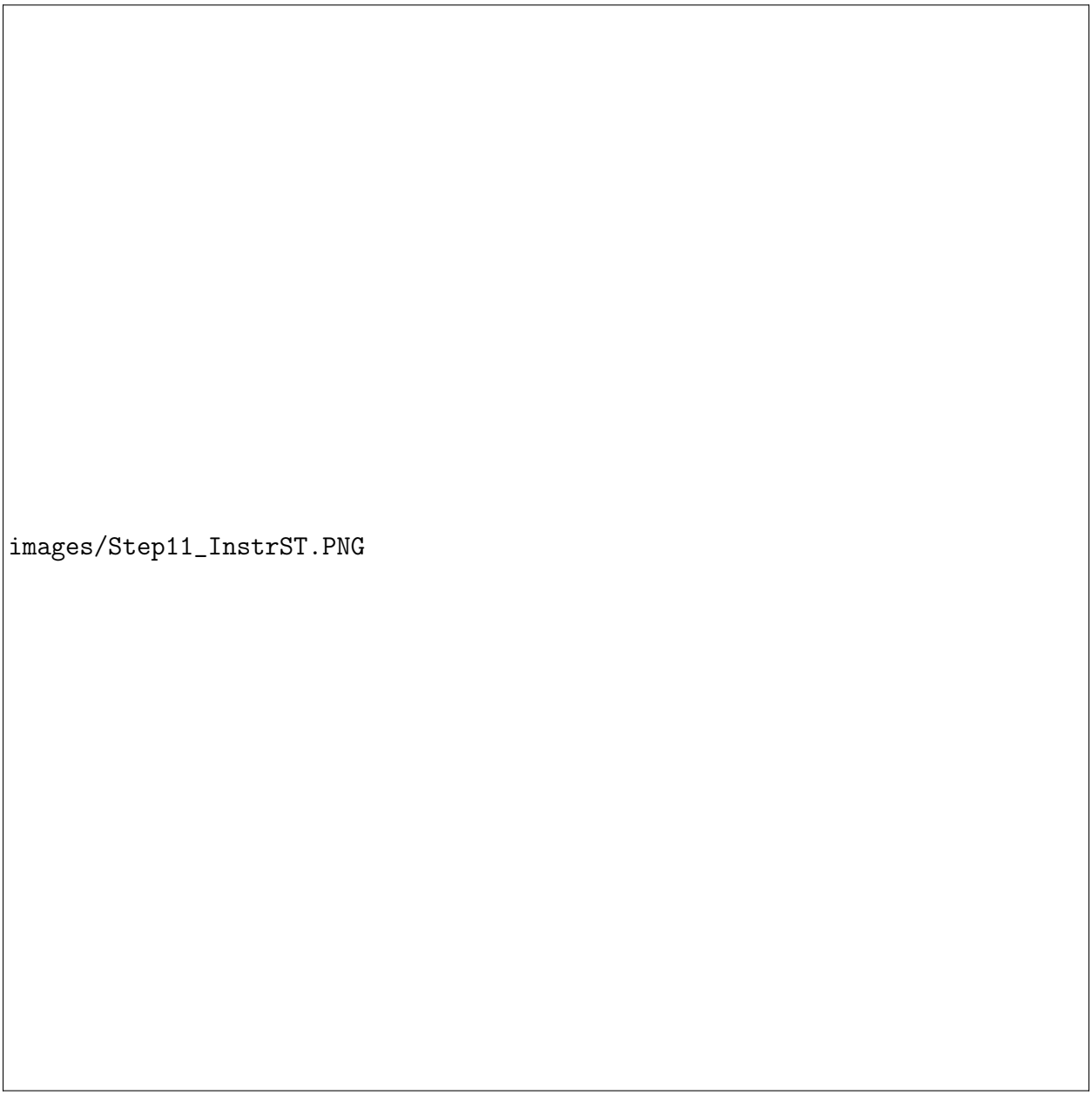

Figura 58: Instrução ST R2, R0 - etapa 3 - O registrador de endereços possui o valor de R0

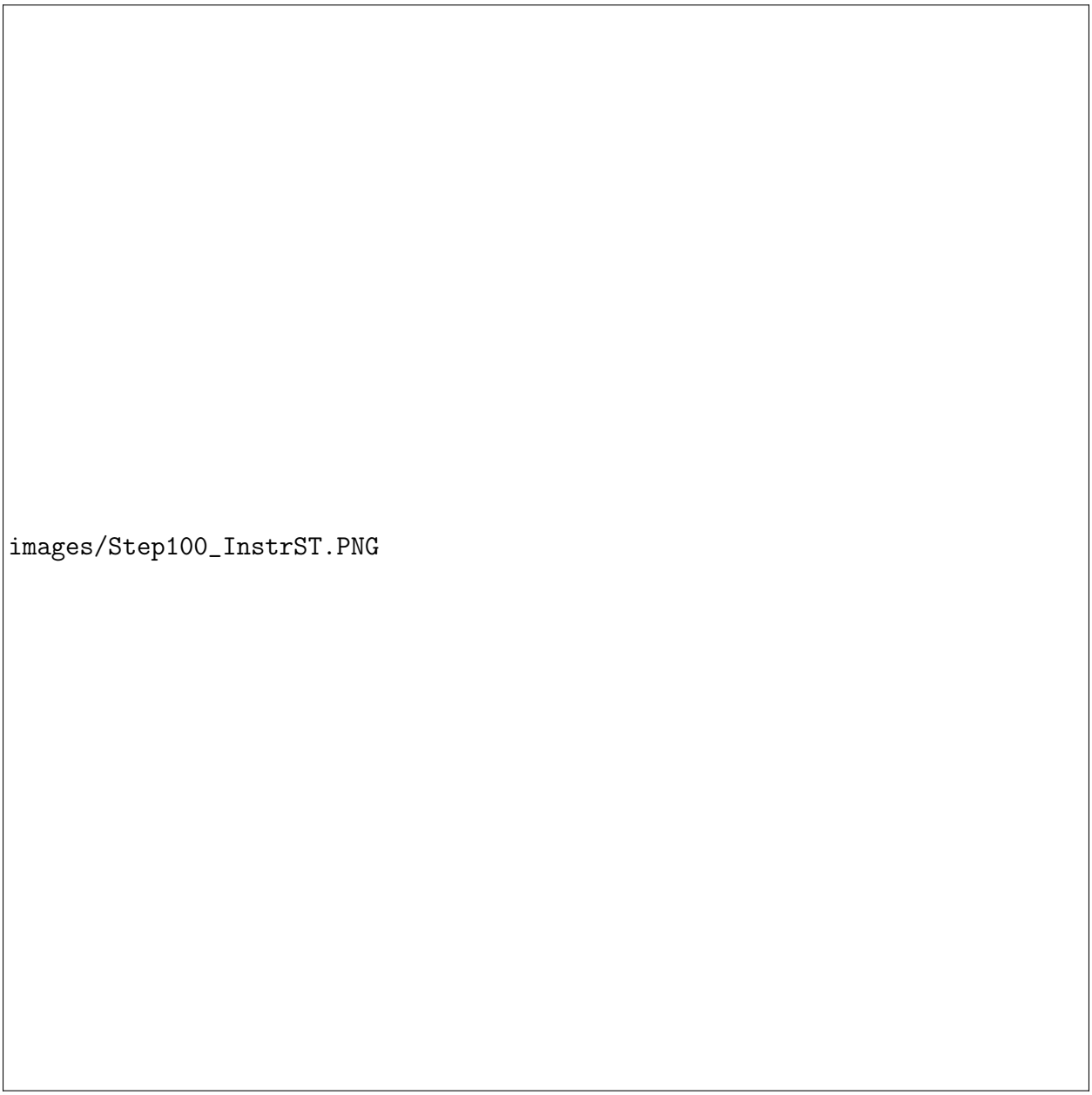

Figura 59: Instrução ADD R1, R0 - etapa 4 - o sinal de entrada de R1 e a saída de G no multiplexador são habilitados, o resultado da operação acaba em R1

LD - 1011

images/InstrLD.PNG

Figura 60: Instrução ADD R1, R0 - o registrador R1 recebe R1 + R0

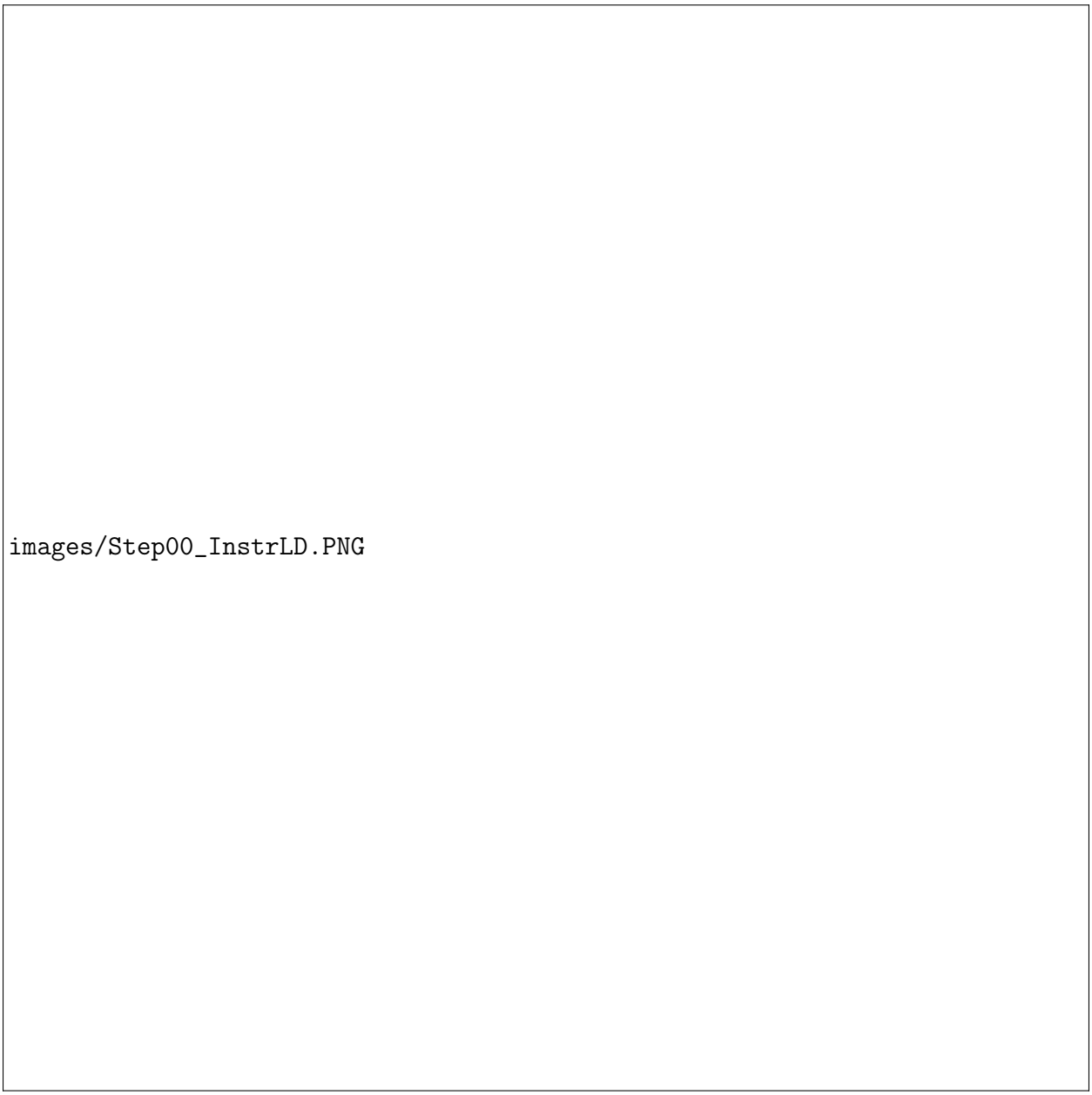

Figura 61: Instrução ADD R1, R0 - etapa 0 - o registrador de instruções recebe a instrução de soma, o *program counter* é incrementado (sinal $incr_pc$) e o ADDR é preparado para receber o endereço do próximo dado

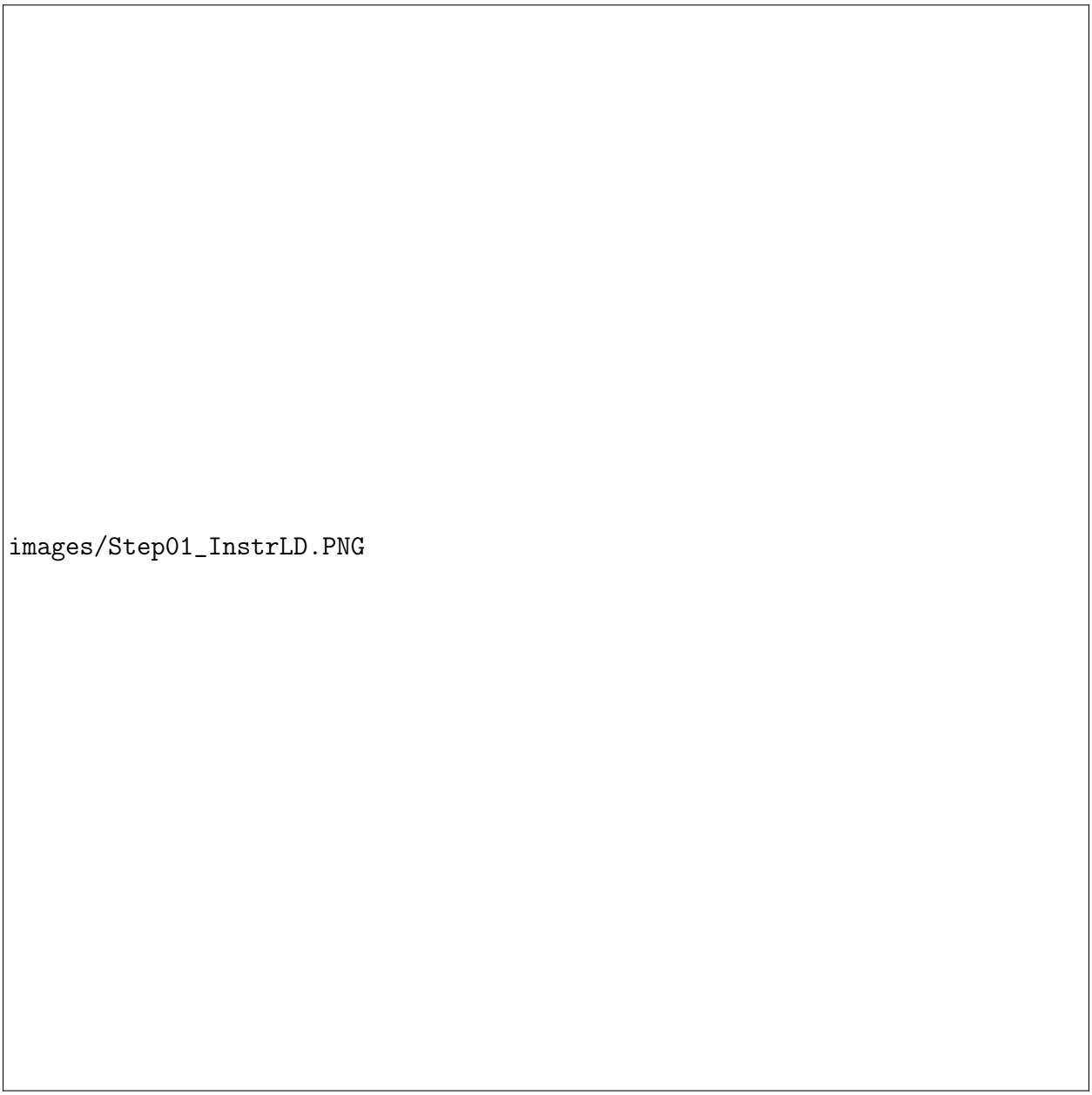

Figura 62: Instrução ADD R1, R0 - etapa 1 - ADDR, que teve seu sinal de escrita habilitado, recebe o endereço do próximo dado

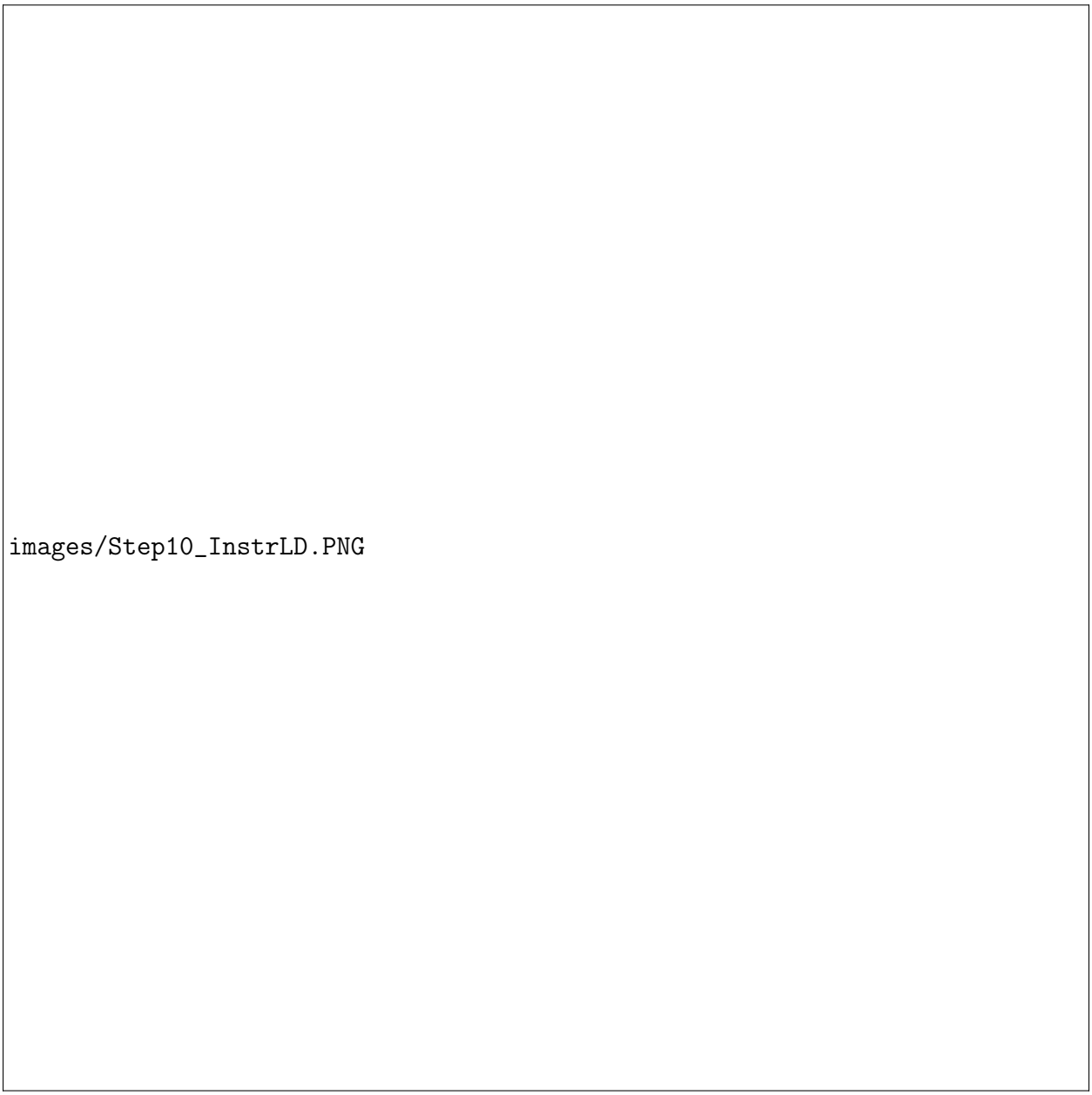

Figura 63: Instrução ADD R1, R0 - etapa 2 - O dado de R1 é colocado no registrador A, que teve seu sinal de escrita habilitado

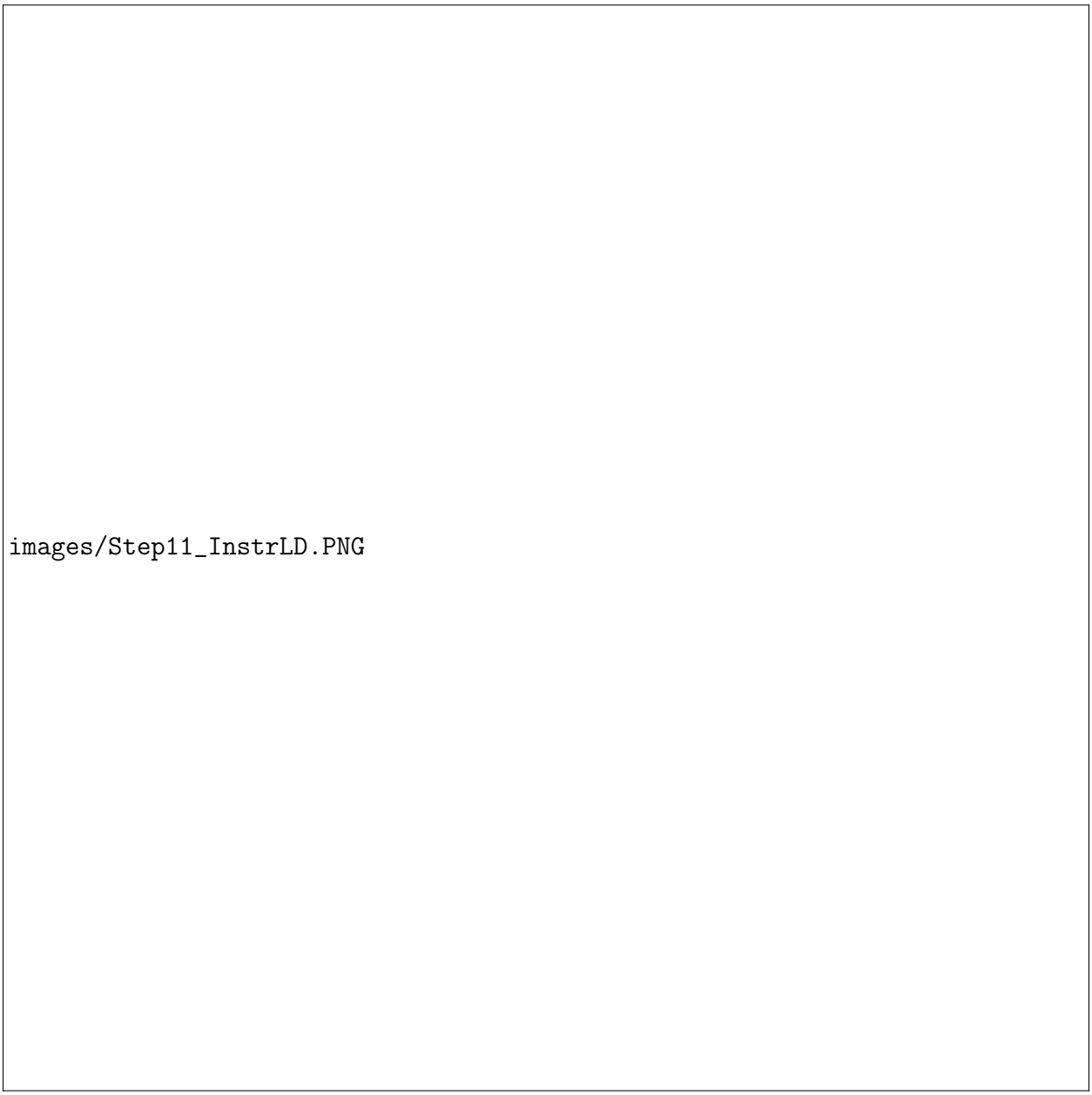

Figura 64: Instrução ADD R1, R0 - etapa 3 - A operação de soma é realizada e seu resultado é colocado no registrador G, que teve seu sinal de escrita habilitado

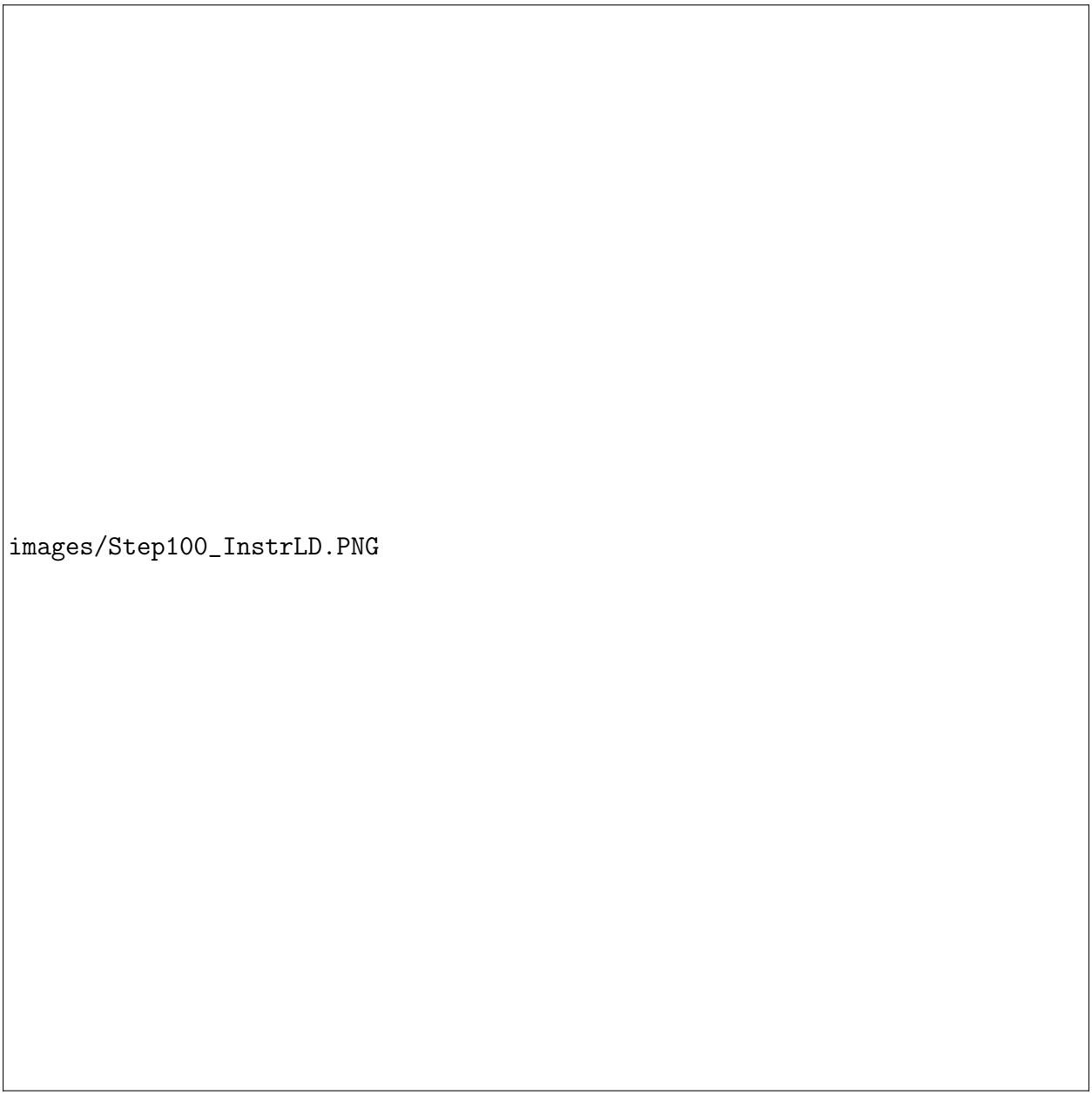

Figura 65: Instrução ADD R1, R0 - etapa 4 - o sinal de entrada de R1 e a saída de G no multiplexador são habilitados, o resultado da operação acaba em R1

O LOOP

PLACA ALTERA

O *top-level* utilizado para a placa está presente no código (comentado) enviado junto com esse documento. Por algum motivo, não foi possível a apresentação na placa. Os dados setados para visualização nos dispositivos da placa ou não atualizavam corretamente ou nem atualizavam. Mais detalhes serão dados na próxima seção.

DIFICULDADES E SUGESTÕES

A implementação de um processador multiciclo requer sem dúvida um grande planejamento, pois os sinais utilizados para relacionar todos blocos combinacionais juntamente com os registradores necessários em cada etapa. Sincronizar tais blocos faz parte da maior dificuldade do projeto e junto à isso, respeitar as etapas de um projeto.

CONSIDERAÇÕES FINAIS

De fato o desenvolvimento histórico por trás das unidades de processamento foi muito importante para que surgisse o processador multiciclo. Os limites físicos apresentados pelos transistores mostram que outras medidas devem ser tomadas para que seja possível permanecer evoluindo no quesito processamento de dados, e com isso o surgimento de novas tecnologias.

REFERÊNCIAS