Nanoprocessador Sequencial Simples

Grupo 14

Fábio Miguel & Luís Gustavo

Relatório do Projeto Final de Laboratório de Circuitos Digitais - Fase 3
Grupo 14: Fábio Miguel e Luís Gustavo

Instruções de utilização, restrições e falhas:

Para utilização correta do circuito, é fundamental garantir um arquivo .mif com instruções válidas, pois o projeto é iniciado diretamente com esses dados na memória e é com eles que programamos quais operações e com quais dados ocorrerão. Uma restrição importante do projeto é o limite de apenas 8 bits no barramento de endereços da memória, o que limita o espaço de instruções e dados a 256 posições. Nenhuma falha crítica de funcionamento foi detectada, mas é recomendável que instruções não reconhecidas sejam evitadas, pois o projeto atual retorna ao estado de leitura sem sinalização explícita de erro ao usuário.

O processador entende seis mnemónicos de 16 bits. O campo mais alto (bits 15..8) representa o opcode e os oito bits menos significativos contêm o operando, que é sempre um endereço de memória. A instrução LOAD a copia a palavra armazenada no endereço a para o acumulador (AC). A instrução STORE a faz o inverso, gravando o conteúdo atual de AC nesse endereço; essa operação é a única que gera um pulso de escrita na memória. O comando SUM a realiza AC ← AC + MEM[a], enquanto SUB a efetua AC ← AC − MEM[a]; ambas usam a ULA e não modificam a RAM. O salto incondicional JUMP a carrega o operando diretamente no PC, desviando o fluxo de execução. Já JNEG a executa um salto condicional: se o bit de sinal do AC estiver em ‘1’ (número negativo), o PC assume o valor a; caso contrário, a execução continua sequencialmente.

Para criar um .mif, comece especificando o cabeçalho DEPTH = 256; WIDTH = 16; ADDRESS_RADIX = HEX; DATA_RADIX = HEX;

seguido do bloco CONTENT BEGIN … END;. Dentro dele, cada linha contém o endereço, dois bytes concatenados (opcode + operando) e, opcionalmente, um comentário. Por exemplo, a sequência 00 : 0208; carrega em AC o valor da posição 08, 01 : 0009; soma o conteúdo de 09, 02 : 010A; grava o resultado em 0A e 03 : 0303; cria um laço infinito apontando para si mesmo. Em seguida, defina nas posições 08 e 09 os dados fonte (por exemplo, 000D e 0005) e zere a posição 0A para receber a soma. Mantendo esse padrão instruções primeiro, dados depois, todos os endereços dentro de 00 a FF o nanoprocessador executará o programa di arquivo .mif que estiver configurado em DMEMORY.VHD.

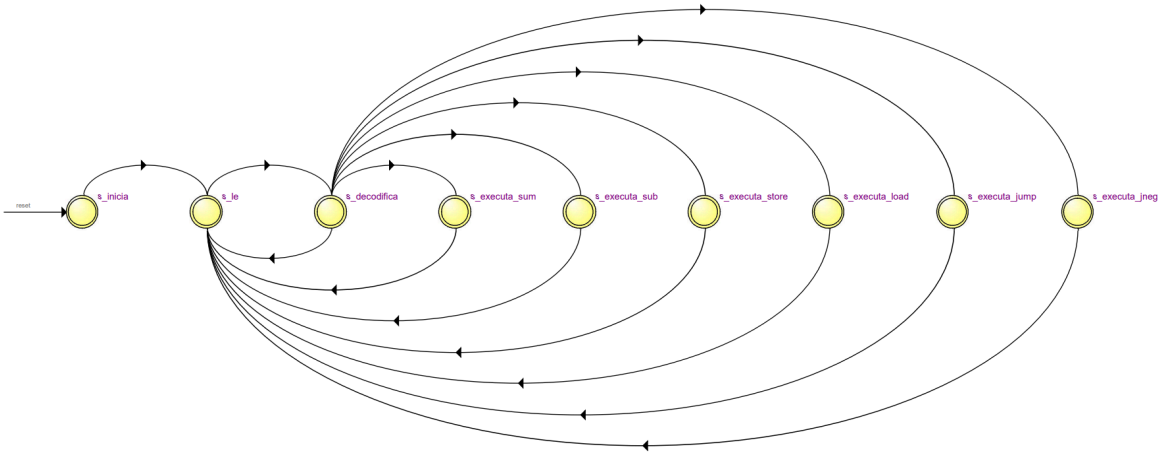


Diagrama de estados:

A máquina de estados finitos mantém a espinha-dorsal das fases anteriores, mas agora contempla seis instruções completas. Foram acrescentados três estados de execução:

s_executa_sum – soma o conteúdo do AC com a palavra apontada por IR(7..0).

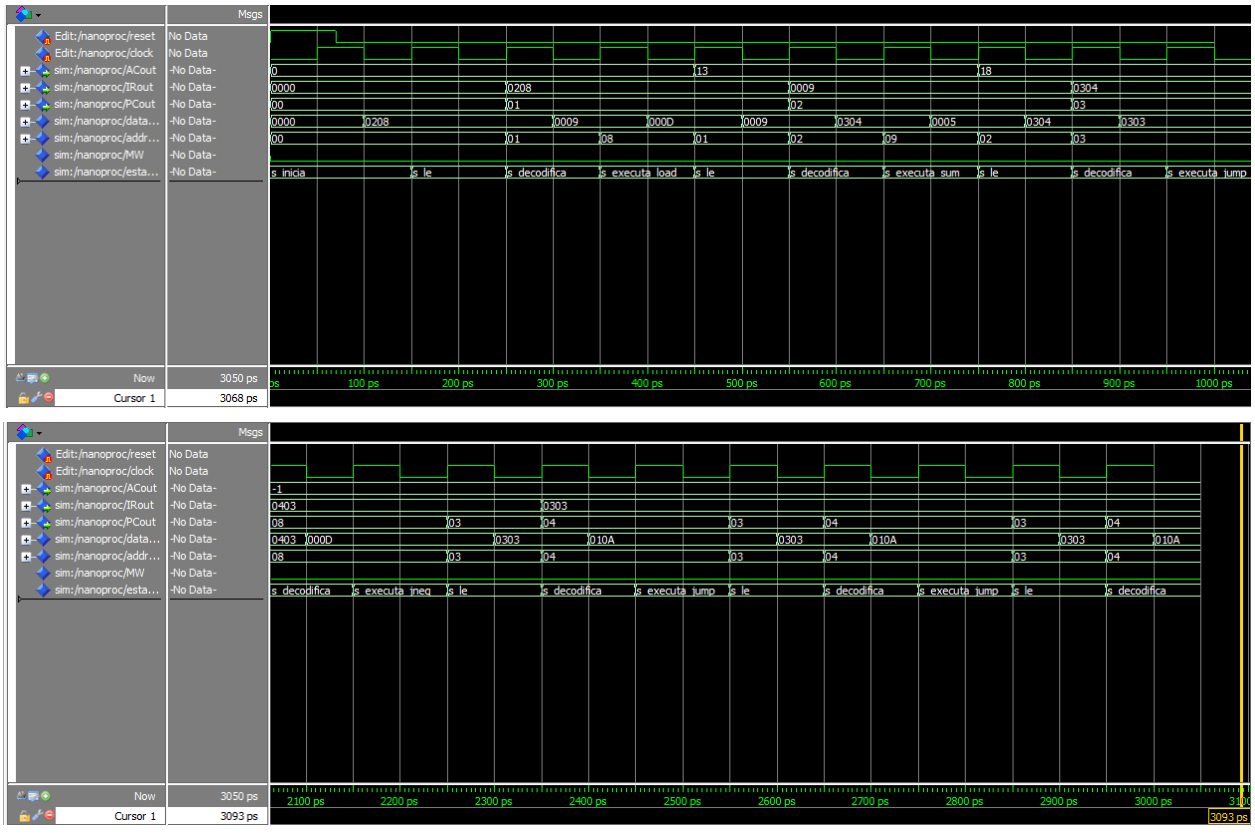
s_executa_sub – subtrai a palavra da memória do valor em AC.

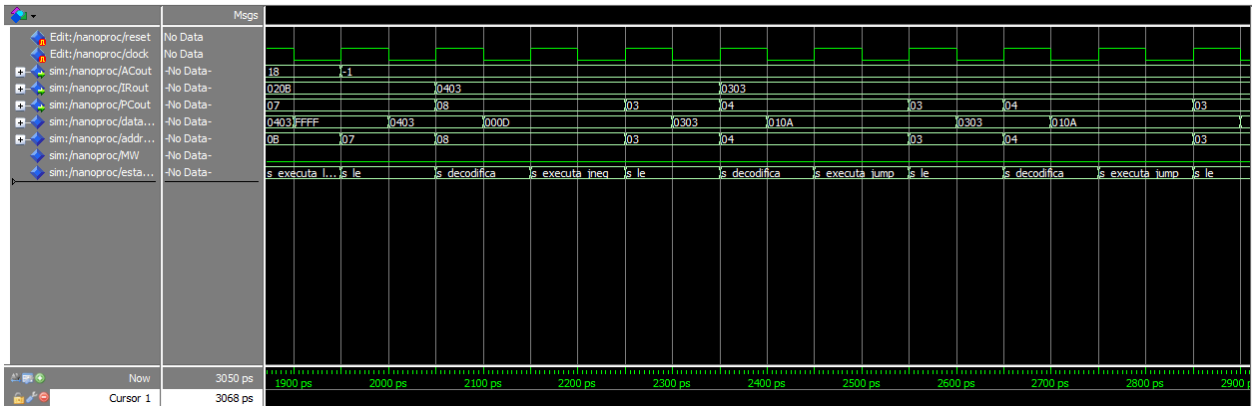
s_executa_jneg – se AC for negativo, força o salto carregando o novo endereço no PC.

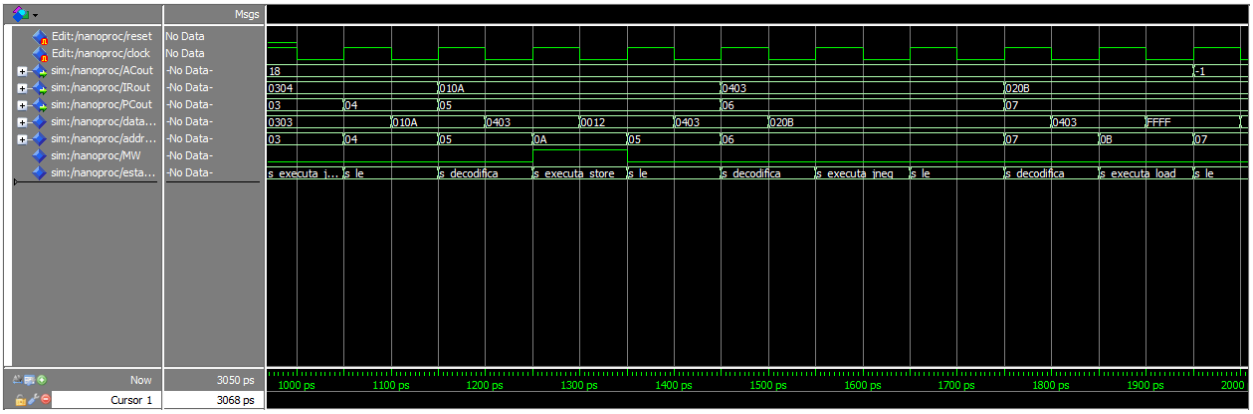
A sequência global permanece:

reset → s_inicia → s_le → s_decodifica → (estado de execução específico) → s_le

Os estados s_executa_load, s_executa_store e s_executa_jump continuam inalterados. Assim, a FSM fecha o ciclo fetch → decode → execute para as instruções LOAD, STORE, JUMP, SUM, SUB e JNEG.







Simulação:

Fluxo observado:

RESET – todos os registradores zerados.

LOAD 08 – PC=00, IR recebe 0208, AC carrega 000D (13).

SUM 09 – IR 0009, AC passa para 0012 (18).

JUMP 04 – PC salta diretamente para 04.

STORE 0A – addrBus 0A, MW sobe durante um ciclo e grava 18 em MEM[0A].

JNEG 03 (com AC positivo) – FSM não salta, apenas retorna a fetch.

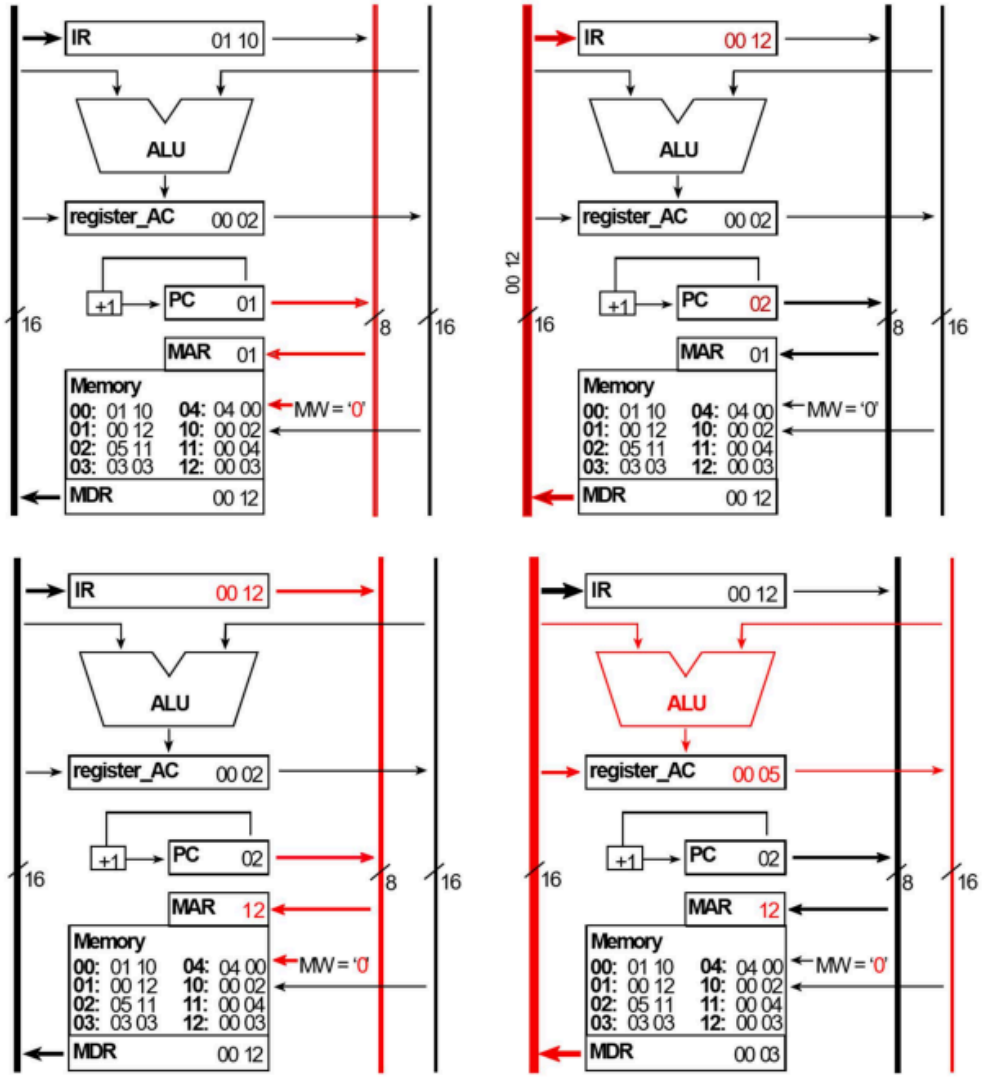
LOAD 0B – AC recebe FFFF (-1).

JNEG 03 (agora AC negativo) – PC é carregado com 03.

JUMP 03 – processador entra num laço infinito, estabilizando addrBus em 03 enquanto IR alterna entre 0403 e 0303.

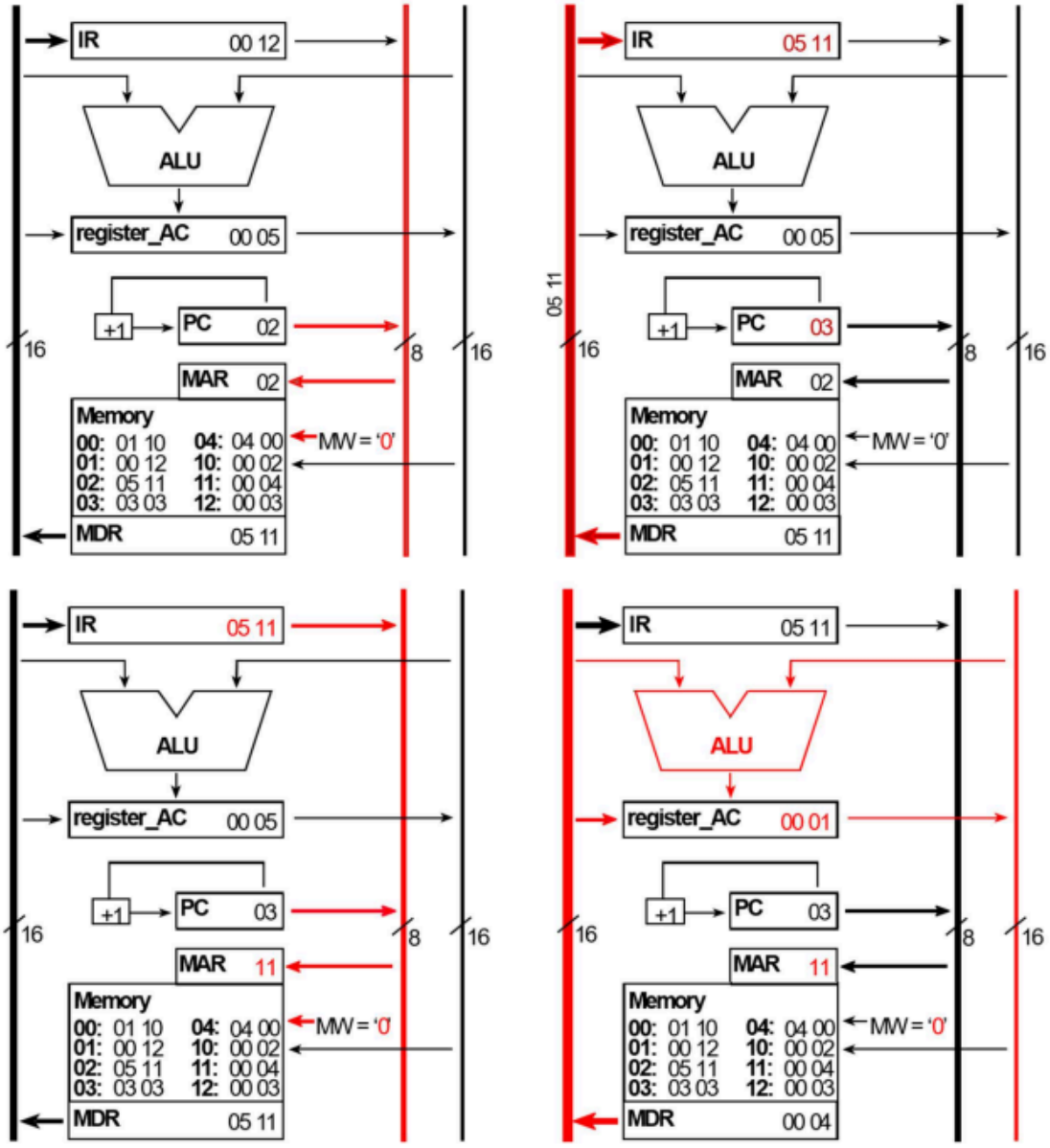
Em toda a sequência, o pulso MW = '1' aparece somente em s_executa_store, validando a lógica de escrita. As transições de estado exibidas na linha estado_atual coincidem, passo a passo, com a descrição em VHDL, comprovando que as três instruções novas foram plenamente integradas à arquitetura.

SUM:



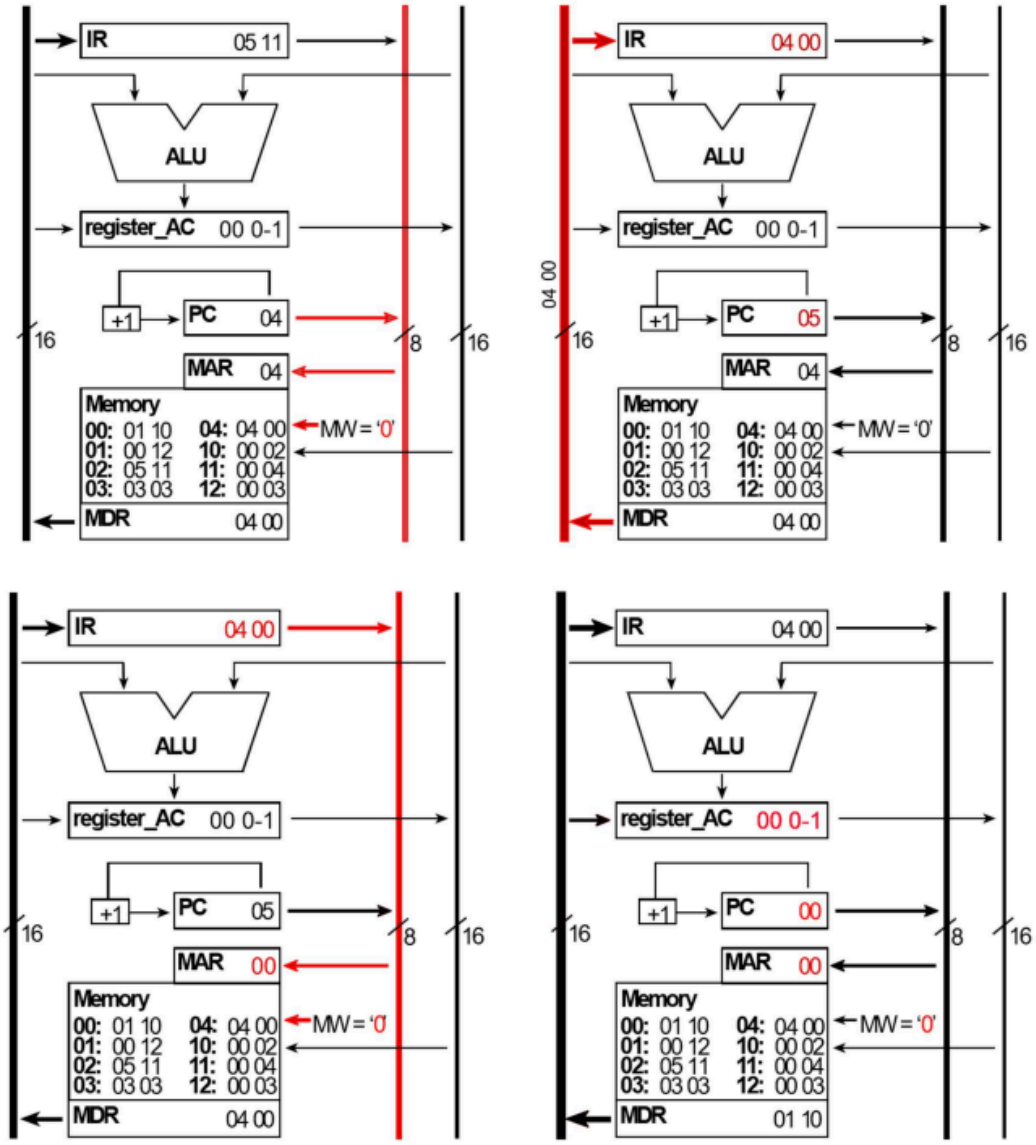
1. Ciclo de busca (s_le)
   PC = 01 coloca o endereço 01 no barramento, a memória devolve 00 12 (em vermelho no MDR). IR ainda contém a instrução anterior (01 10).
2. Decodificação (s_decodifica)
   IR carrega 00 12, PC é autoincrementado para 02. Como o opcode é 00h, a FSM muda para s_executa_sum.
3. Leitura do operando
   Em s_executa_sum o operando 12h vai para o MAR; o mesmo endereço aparece no barramento (seta vermelha horizontal). A palavra 00 03 é lida da memória e colocada no MDR.
4. Operação da ULA e gravação no ACA ULA soma 0002 (AC) + 0003 (MDR) e gera 0005, que substitui o valor do AC (seta vermelha sobre o bloco ALU). Terminado o ciclo, a FSM retorna a s_le.

SUB:



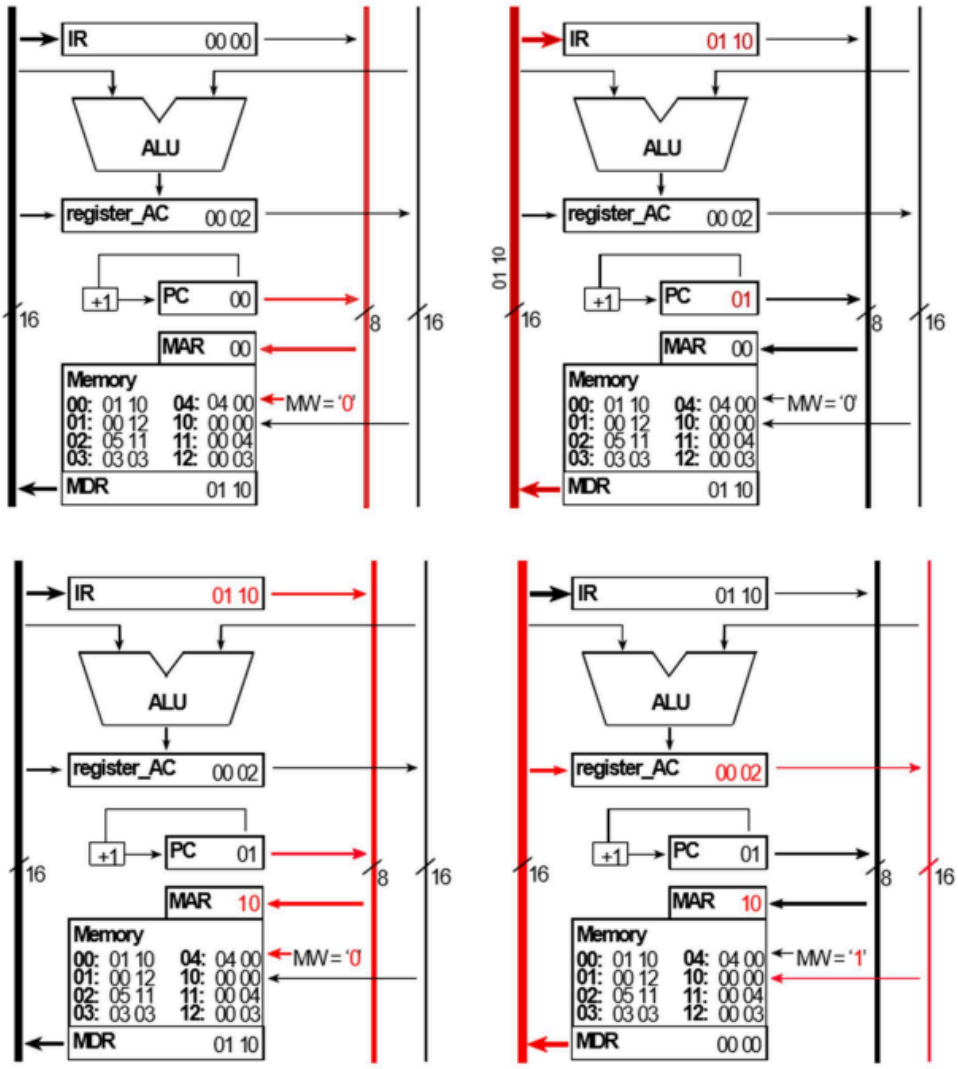
1. Fetch
   Agora PC = 02 e o endereço 02 é colocado no barramento; a memória devolve 05 11. IR ainda mostra 00 12, AC vale 0005.
2. Decode
   IR recebe 05 11, PC vai a 03. O opcode 05h redireciona a FSM para s_executa_sub.
3. Leitura do operando
   MAR é carregado com 11h. A memória restitui 00 04 (conteúdo da posição 11) ao MDR.
4. ALU (subtração) e atualização do AC
   A ULA calcula 0005 – 0004 → 0001 e grava o resultado no AC. O fluxo volta a s_le, pronto para a próxima instrução.

JNEG:



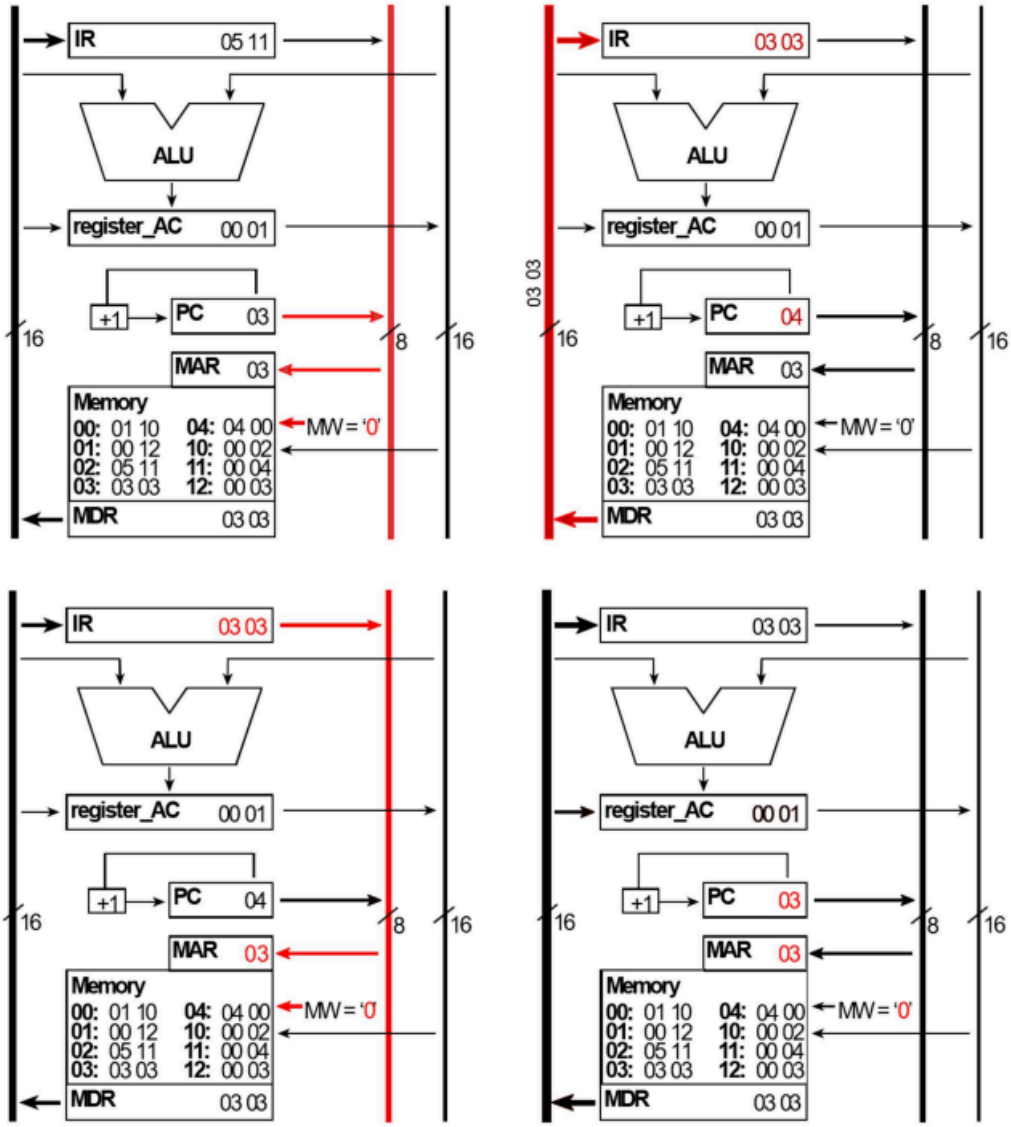
1. Fetch
   PC = 04 endereça a memória; chega a palavra 04 00 (JNEG 00). O AC já contém FFFF (-1) gerado anteriormente, portanto o bit de sinal está em 1.
2. Decode
   IR assume 04 00, PC é incrementado para 05. A FSM identifica o opcode 04h e segue para s_executa_jneg.
3. Avaliação da condição
   A lógica de controle lê AC(15)=1 → condição verdadeira. Sem ainda modificar o PC, prepara-se o operando 00 (destino) no caminho de endereço.
4. Salto
   O valor 00 é carregado no PC, completando a ramificação. AC permanece inalterado, MW continua '0'. A busca seguinte (PC = 00) recomeça o ciclo.

STORE:



1. Fetch – com PC = 00 o barramento de endereço apresenta 00h, a RAM devolve a palavra 01 10 (STORE 10), que entra no MDR enquanto o IR ainda contém a instrução anterior.
2. Decode – IR recebe 01 10 e o PC é autoincrementado para 01. O opcode 01h faz a FSM avançar para s_executa_store.
3. Execução – o operando 10h é encaminhado ao MAR, aparecendo em vermelho no barramento de endereço. Simultaneamente, o conteúdo atual do AC (0002 no diagrama) passa para o data bus; nesse instante o controle MW sobe para '1' durante exatamente um ciclo, habilitando a gravação em memória. A posição 10 passa de 0000 para 0002, confirmando a escrita. Terminado o pulso, MW volta a '0' e o fluxo retorna a s_le para buscar a próxima instrução.

JUMP:



1. Fetch – já com PC = 03, o processador coloca 03h no barramento, a memória responde com 03 03 (JUMP 03) e o código é capturado no MDR.
2. Decode – IR carrega 03 03 e o PC sobe para 04. O opcode 03h leva a FSM para s_executa_jump.
3. Execução – sem qualquer acesso adicional à RAM, o campo de operando (03h) é encaminhado diretamente ao PC; a seta vermelha no diagrama destaca essa transferência interna, enquanto addrBus permanece apontando a última instrução lida.
4. Ao final do ciclo o PC já contém 03h, de modo que a busca seguinte retorna ao mesmo endereço, fechando um laço infinito. Durante toda a operação MW permanece em '0' e não há tráfego de dados no barramento de escrita, evidenciando que JUMP altera apenas o fluxo de controle.