PROJETO

Laboratório de

CIRCUITOS DIGITAIS

Nanoprocessador Sequencial Simples

Fases 1,2 e 3

2025

SUMÁRIO

1 Regras gerais

- Grupos
  - Grupos de 2 a 4 pessoas.
  - Grupos com formação inicial em menor ou maior número somente com autorização.
  - Não será aceita mudança de grupo após a entrega da primeira fase.
  - Em caso de desistência de algum membro do grupo, este finalizará o projeto com número menor de membros, não há inclusão de membros após a primeira fase.
- Nota
  - Será atribuída nota de 0 a 10 para cada fase. A nota será composta conforme plano de ensino.
  - A entrega atrasada da fase 1 ou 2 acarretará desconto de 0,5 ponto POR DIA DE ATRASO.
  - A entrega atrasada da fase 3 acarretará 1,0 ponto POR DIA DE ATRASO. Não será aceita entrega com mais que 2 dias de atraso.
  - A não entrega da fase INDIVIDUAL, será entendida como abandono do grupo, portanto o aluno obterá nota zero no projeto.
  - A não entrega da última fase implica em desistência do projeto, sendo atribuída nota zero ao grupo.
- NOME DO ARQUIVO ENTREGUE PELO AVA: em todas as fases em que houver entrega pelo AVA o arquivo deve iniciar com a letra G seguida do número do grupo (G<n>_nomes), devendo seguir com o nome do aluno se a entrega for individual, caso contrário pode ser somente o número do grupo. Se houver mais que um arquivo estes devem ser colocados em uma pasta de mesmo nome acima e compactado com ZIP ou RAR, gerando um arquivo de mesmo nome. NOMES INCORRETOS ACARRETAM DESCONTOS DE 0,5 NA NOTA.
- TRABALHOS IGUAIS, MESMO QUE PARCIALMENTE (UM OU MAIS MÓDULOS), RECEBERÃO, AMBOS, NOTA **ZERO**

2 Introdução Teórica

Computadores tradicionais consistem em três unidades principais: o processador (CPU), a memória e interfaces de entrada e saída. Como visto na Figura 1, estas unidades são conectadas por sinais digitais paralelos, conhecidos como barramentos. Os tipos de sinais normalmente enviados por barramentos são: dados de memória, endereço de memória e sinais de controle.

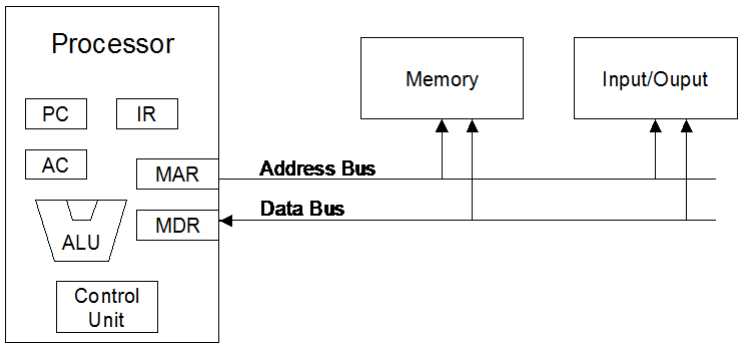


Figura 1 – Arquitetura Básica de um Computador

Internamente as CPUs contêm alguns registradores que são usados para armazenar os dados manipulados dentro do processador. Suponha um processador com três registradores: PC, IR, AC.

- PC(Program Counter): contador de programa. Registrador que armazena o endereço da próxima instrução a ser lida e executada. Controla a execução das instruções.
- IR(Instruction Register): registrador que armazena o código da instrução a ser decodificada e executada.
- AC(Acumulator): registrador acumulador (dados).

Um processador possui um conjunto limitado de comandos (instruções) que ele é capaz de executar. Considere que o processador possui um conjunto de instruções no seguinte formato:

Figura 2 – Formato da Instrução

A Figura 2 indica que uma instrução tem tamanho de 2 bytes. O byte mais significativo possui um código que identifica a instrução que deve ser executada (opcode) e o byte menos significativo possui um operando que indica o endereço de memória que contém o dado a ser utilizado pela instrução. Suponha que o processador possui o seguinte conjunto de instruções.

Instrução	Operação	Opcode
LOAD *address*	AC = conteúdo da memória no endereço *address*	02
STORE *address*	endereço *address* da memória = AC	01
SUM *address*	AC=AC + conteúdo da memória no endereço *address*	00
SUB *address*	AC=AC - conteúdo da memória no endereço *address*	05
JUMP *address*	PC = *address*	03
JNEG *address*	Se AC < 0 então PC = *address*	04

Figura 3 – Conjunto de Instruções

As instruções são armazenadas na memória com 16 bits, sendo que os bits mais significativos carregam o código da instrução (*opcode*) e os bits menos significativos carregam o operando da instrução. Por exemplo, SUM 12 (instrução em linguagem de máquina 0012), adiciona o conteúdo armazenado no endereço 12 da memória ao conteúdo do registrador acumulador (AC) e guarda o resultado da soma no próprio acumulador.

A primeira operação que é realizada pelo processador, quando ligado, é a execução da sequência de instruções armazenadas na memória, iniciando no endereço zero. Para o processador exemplo, basicamente o processador carrega uma instrução da memória, armazena no registrador de instrução (IR), incrementa o contador de programa (PC), decodifica a instrução (reconhece o que será executado e gera sinais necessários) e executa a instrução. Os dados utilizados e resultado das operações são armazenados no registrador AC. Então repete esta sequência indefinidamente, até ser desligado.

Suponha o código abaixo (em C e instruções do processador) para ler dois dados positivos, somá-los, calcular o módulo 7 da soma e apresentar o resultado. Só existe operação de soma e subtração, portanto o cálculo do módulo será feito por subtrações. Para o código equivalente do processador em questão os dados são lidos da memória (A e B) e armazenados na memória (Z), ao invés de ler da entrada padrão (scanf) e mostrar na saída padrão (printf).

Linguagem C	Em código Mnemônico do processador	Observação
scanf(&A);	LOAD A	AC = MEM[A]
scanf(&B); AC = A + B;	SUM B	AC = AC + MEM[B]
while (AC ≥ 7) AC = AC -7;	SUB P X: JNEG Y SUB P JUMP X Y: SUM P	AC = AC – MEM[P] X: SE AC < 0 VAI PARA ENDEREÇO Y AC = AC – MEM[P] VAI PARA ENDEREÇO X Y: AC = AC + MEM[P]
printf("%d";AC);	STORE Z	MEM[Z] = AC
return	W: JUMP W	PARA

Suponha então este pequeno programa apresentado quando armazenado na memória (verde). Os dados estão nos endereços 10, 11 e 13 (roxo). O resultado é armazenado no endereço de memória 12. Será calculado: (9 + 10) mod 7 = 5. Como temos um loop, cada passo de execução é representado em uma coluna (loop1, loop2, loop3)

Ende-reço	Conteúdo		Mnemônico equivalente	Operação (loop 1)	(loop 2)	(loop 3)
00	0210		LOAD 10	AC = 9		
01	0011		SUM 11	AC = 9 + 10 = 19		
02	0513		SUB 13	AC = 19 – 7 = 12		
03	0406		JNEG 06	Falso, segue	Falso, segue	Verdadeiro
04	0513		SUB 13	AC = 12 – 7 = 5	AC = 5 – 7 = -2	
05	0303		JUMP 03	Retorna p/ JNEG	Retorna p/ JNEG	
06	0013		SUM 13			AC = -2 + 7 = 5
07	0112		STORE 12			MEM[12] = 5
08	0308		JUMP 08			Para aqui
09	0003					
....						
0A			(decimal)			
10	0009		9			
11	000A		10			
12	0005		5			
13	0007		7			

Observe que, se o endereço 08 tivesse o conteúdo 0310 (JUMP 10), haveria um salto na execução do programa para o endereço 10. Neste caso o processador executaria a instrução 0009 (SUM 09), que faz AC = AC + MEM[09], ou seja, AC = 5 + 3 = 8 (0008_{Hex}). Então lembre-se, os bytes armazenados na memória só possuem significado quando utilizados. O processador pode reconhecer dados como instruções se não houver uma organização que os separe. Ou seja, se memória de dados e programa são compartilhadas, como no exemplo, é necessário forçar a parada. No caso, forçamos o salto sempre para o próprio endereço, de forma a parar a execução.

2.1 FLUXO DE DADOS

O objetivo deste projeto é construir um processador que funcione como o descrito acima. Mas antes é importante entender o fluxo de dados necessário para a execução de cada instrução. Basicamente, o processador fica em loop infinito fazendo: lê uma instrução da memória, salva no registrador IR, incrementa o contador (PC = PC + 1), decodifica (reconhece o código e o endereço) e executa a instrução (realiza a operação), reinicia o loop.

A Figura 4 apresenta um exemplo da estrutura básica para o fluxo de dados, representando o estado após a operação de reset. Veja o documento HamblenCap8.pdf para mais informações. O barramento de 8 bits é o barramento de endereço (*Address Bus*) e os barramentos de 16 bits representam o barramento de dados (*Data Bus*).

Observe que as unidades funcionais ALU, registradores (IR, PC e AC) e +1 (incrementador) fazem parte do processador (Figura 1), enquanto a memória é uma unidade externa. O registrador PC (Program Counter) é que "conduz" a execução do programa, portanto, qualquer desvio na execução do programa (JUMP ou JNEG) deve ocorrer através da alteração deste registrador.

A Figura 4 apresenta separadamente o barramento de dados utilizado para leitura e escrita na memória. O objetivo é facilitar a definição dos possíveis caminhos de dados. A memória é endereçada para a posição zero durante o reset inicial, assim como todos os registradores são zerados. O sinal MW='0' indica que a memória está habilitada para leitura, MW='1' habilita para

escrita (gravação). A figura representa o caminho de dados após um reset. As setas indicam o sentido do fluxo de dados.

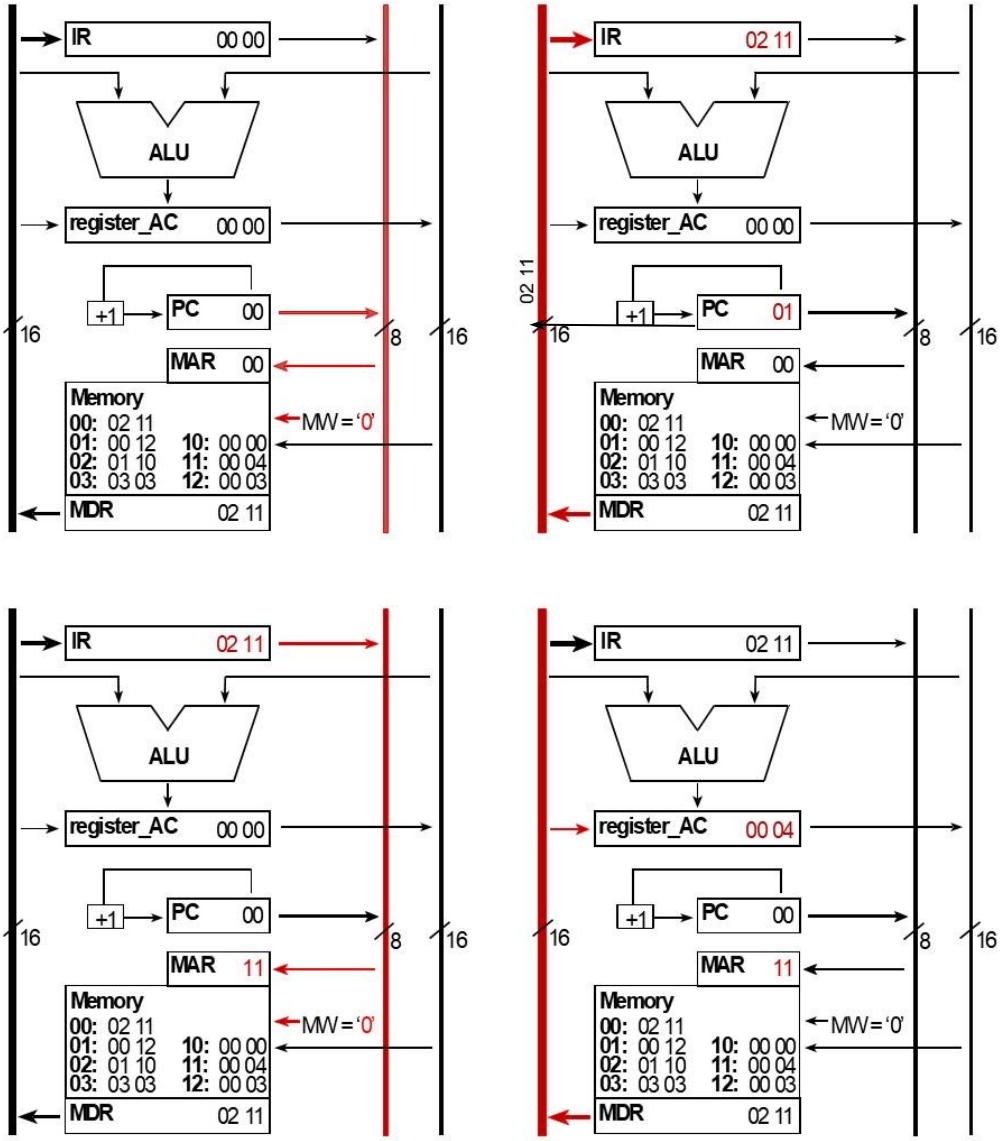


Figura 4 – Exemplo de Código e Projeto do Computador

Observe o conteúdo da memória na Figura 4. O PC possui valor zero, colocando este endereço no barramento de endereços. O endereçamento de memória (MAR) está conectado ao barramento de endereços, portanto inicialmente está referenciado o endereço zero. O fluxo de dados deve ocorrer na seguinte sequência, representado na figura com a marcação em vermelho:

- **LER**: o conteúdo do endereço zero da memória é colocado no barramento de dados, através no registrador de saída de memória MDR (0211);
- **SALVAR**: o registrador IR é ativado (pulso de clock) e salva a instrução (0211);
- **INCREMENTAR**: o registrador PC é ativado (pulso de clock) e salva (PC+1), ou seja, PC = 01;
- **DECODIFICAR**: o endereço da instrução (exemplo: LOAD 11) é colocado no barramento de endereço.
- **EXECUTAR**: a memória, agora sendo endereçado para o endereço 11, coloca no barramento de dados seu conteúdo (0004);
    - o registrador AC é ativado (pulso de clock) e salva o dado no barramento (0004).

O fluxo de dados para a realização da instrução LOAD foi apresentado como exemplo, faça o fluxo de dados para as demais instruções SUM, STORE, JUMP, JNEG e SUB.

2.2 Detalhamento das instruções

Cada instrução possui um código binário. As instruções são armazenadas na memória na sequência em que serão executadas, assim como os dados utilizados para sua execução. Por exemplo, a instrução *Load A* tem código binário (opcode) 00000010 (02_{Hex}) e precisa do endereço cujo conteúdo será copiado para o registrador Acumulador. Então o código 00000010 será armazenado no byte mais significativo da posição de memória, sendo os bytes menos significativos reservados para o endereço a ser copiado.

2.2.1 Load A (02_{HEX})

Código: 0000 0010

Copia um dado do endereço A da memória para o registrador Acumulador: AC = Mem[A].

2.2.2 Store A (01_{HEX})

Código: 0000 0001

Copia um dado do registrador Acumulador para o endereço A da memória: Mem[A] = AC.

2.2.3 Sum A (00_{HEX})

Código: 0000 0000

Soma o conteúdo do registrador Acumulador com o conteúdo do endereço de memória A, e grava o resultado no Acumulador: AC = AC + Mem[A].

2.2.4 Sub A (05_{HEX})

Código: 0000 0101

Subtrai o conteúdo do endereço de memória A do valor no registrador Acumulador, e grava o resultado no Acumulador: AC = AC – Mem[A].

2.2.5 Jump A (03_{HEX})

Código: 0000 0011

Salto incondicional. Desloca o contador de programas (PC) para o endereço de memória A: PC = A.

2.2.6 JNeg A (04_{HEX})

Código: 0000 0100

Salto condicional. Desloca o contador de programas (PC) para o endereço de memória A somente quando o valor no Acumulador for negativo (menor que zero): Se Ac<0 então PC = A.

3 Fase 1 (GRUPO)

Implementar a estrutura básica do processador e a instrução Load.

3.1 Projeto da Estrutura Básica

TESTE DA ESTRUTURA BÁSICA – LEITURA DA MEMÓRIA EM SEQUÊNCIA

Antes de iniciar a modelagem do circuito, considere que:

- Tem disponível uma memória de 8 bits de endereço e 16 bits de dados (DMEMORY.vhd)
- Cada instrução ocupa 2 bytes na memória (16 bits).
- A Figura 4 indica um processador com 16 bits de dados.

- A ULA deve suportar as operações de soma e subtração (usar lib Altera)
- O processador utiliza representação de dados com sinal em complemento de 2, ou seja, considere que os dados na memória são armazenados neste formato.

Crie um novo projeto no Quartus e copie a estrutura básica contento 3 unidades/6 arquivos: processador (nanoProc.vhd), memória (DMEMORY.vhd, PROGRAM.mif) e I/O (TLE_Proc.vhd, LCD_Display). Seu grupo deve criar um processador conforme descrito na seção 2.

Após criar o projeto e copiar os arquivos vhdl, faça:

- Defina os pinos não utilizados com “As input tri-state”.
- Defina a Top-Level Entity como sendo o componente TLE_Proc.
- Importe a configuração de pin planer do arquivo nanoProc.csv.
- Compile e corrija eventuais erros.
- Simule a unidade nanoProc (não a TLE, que só adiciona a interface de display para visualização na placa) com:
    - Sinal de reset inicial com duração de 70 ps.
    - Sinal de clock de período 100 ps com valor inicial ZERO.
    - As configurações acima são necessárias para que o conteúdo de memória seja carregado.
- Verifique se estão corretos os sinais na saída do barramento de dados e dos registradores PC e IR.

MODELAGEM FUNCIONAL

Descreva um diagrama de estados com 4 estados: inicia, le, decodifica, executa. Um sinal assíncrono de reset deve colocar o sistema no estado inicial.

- O estado de "inicia" deve zerar todos os registradores, ou seja, colocar o sinal relativo ao registrador igual a zero. E definir o próximo estado como "le". O sinal de reset deve colocar o sistema no estado "inicia".
- O estado "le" deve guardar o conteúdo da memória (MDR/dataBus) no registrador de instrução (IR), incrementar o contador de programa (PC = PC + 1) e ir para o estado "decodifica".
- O estado decodifica deve selecionar a instrução a ser executada, mas por enquanto, nesta fase, apenas vai para o estado "executa".
- O estado executa deve executar efetivamente a instrução, por enquanto, nesta fase, deve apenas retornar para o estado "le" para ler a próxima instrução.
- DEFINA como saída do diagrama de estados o que deve ter no barramento de endereço (exemplo, PC ou IR[0]) em cada estado para que o endereço correto esteja na memória.
- Compile e corrija eventuais erros.
- Verifique o diagrama de estados com a ferramenta *NetList Viewer-RTL Viewer* e clicando sobre o componente “nanoProc”, depois sobre o componente de controle de transição de estados.
- Teste com o simulador o funcionamento do projeto, que deve refletir a leitura da memória (program.mif).
- Simule a unidade nanoProc (não a TLE, que só adiciona a interface de display para visualização na placa) com:
    - Sinal de reset inicial com duração de 70 ps.
    - Sinal de clock de período 100 ps com valor inicial ZERO.
    - As configurações acima são necessárias para que o conteúdo de memória seja carregado.
- Verifique se estão corretos os sinais na saída do barramento de dados, barramento de endereço e dos registradores PC e IR.
- Faça upload na placa e verifique se o funcionamento está correto. Observe que para o funcionamento correto, deve-se habilitar o sinal de reset, acionar 1 pulso de clock, então desabilitar o reset para iniciar o funcionamento (pulsos de clock).

3.2 Instrução Load

Considerando o que foi projetado como Estrutura básica e o fluxo de dados apresentado na Figura 4, implemente a instrução ***Load***, ou seja, os estados decodifica e executa do diagrama de estados.

3.3 Entrega

Data: a definir pela professora

Documento: arquivo nanoproc.vhd, documento digital com diagrama de estados obtido no quartus e figura da wave de simulação explicando como verificou o funcionamento correto através da wave. A pasta compactada com os arquivos deve ter o nome padronizado (G<n>).

Conteúdo do documento: diagrama de estado, figura da wave e texto.

Formato: documento simples, sem formatação específica ou capa, com o **número do grupo na primeira linha** e o **nome dos membros do grupo que participaram** nas próximas linhas. É um documento provisório de acompanhamento de projeto, mas que deve conter todas as informações relevantes para a compreensão do seu funcionamento.

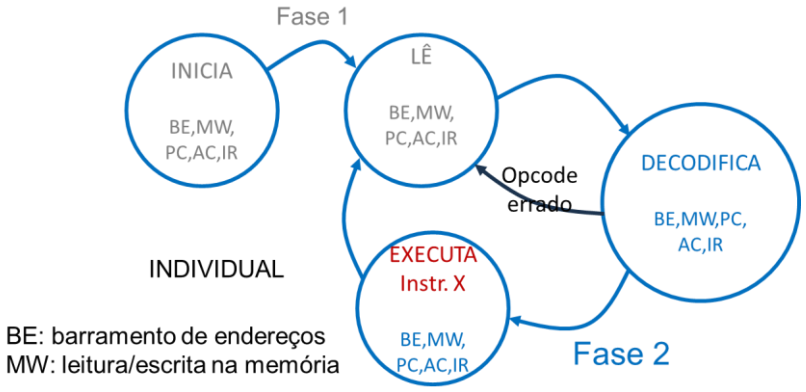
4 Fase 2 – Projeto das Instruções (INDIVIDUAL)

Desenhe os diagramas semelhantes à Figura 4 para cada uma das 5 instruções, uma por aluno. A distribuição entre os alunos do grupo deve seguir a ordem: STORE, JUNP, SUM, JNEG, SUB. Por exemplo, se o grupo tem 3 membros, devem ser entregues as 3 PRIMEIRAS instruções nesta fase. As demais instruções podem ser feitas em grupo.

Para descrição do fluxo de execução da instrução, faça cópia do diagrama e marque o passo a passo da instrução colocando cores nas ligações ativas a cada passo. **Explique textualmente a sequência (ordem) de troca de dados através dos barramentos necessários para realizar a operação (caminho/fluxo de dados)**. Utilize a figura com uma estrutura básica disponível no AVA e altere a cor do caminho de dados para indicar a operação realizada.

Observe que há uma sequência inicial de passos que é igual para todas as instruções. Caso haja necessidade, altere os fluxos das instruções que estão incoerentes com as demais.

Cada aluno deve modelar uma instrução (diagrama de estados), rigorosamente baseado no caminho de dados definido (quais sinais devem ser acionados e em que ordem para realizar a operação). Cada aluno deve SEPARAMAMENTE, projetar e integrar sua instrução no código, fazendo a decodificação (estado decodifica) e realizando a operação no estado "executa". Se na decodificação não for sua instrução o próximo estado deve ser o "le".



A seleção de qual aluno descreverá qual instrução segue a seguinte regra:

- Ordem das instruções: STORE, JUMP, SUM, JNEG, SUB
- Ordem dos alunos: alfabética

O primeiro aluno do grupo em ordem alfabética deve descrever a instrução STORE, o segundo JUMP e assim por diante. As instruções que não forem atribuídas a membros do grupo devem ser feitas por todo o grupo até a fase 3.

ATENÇÃO: pode ocorrer de uma instrução precisar de mais que um ciclo de clock para ser executada, neste caso é necessário mais que um estado para sua execução.

4.1 Entrega (INDIVIDUAL)

Data: a definir

Documento: arquivo vhdl, documento digital com diagrama de estados obtido no quartus e figura da wave de simulação explicando como verificou o funcionamento correto através da wave. A pasta compactada com os arquivos deve ter o nome padronizado (G<nn>_nome).

Conteúdo do documento: diagrama de estado, figura da wave e texto.

Formato: documento simples, sem formatação específica ou capa, com o **número do grupo na primeira linha** e o nome do(s) aluno(s) na(s) próxima(s) linha(s). É um documento provisório de acompanhamento de projeto, mas que deve conter todas as informações relevantes para a compreensão do seu funcionamento.

5 Fase 3 – Integração (GRUPO)

Nesta fase as instruções devem ser integradas ao projeto do grupo tal que o estado "executa" deve selecionar qual instrução executar dependendo da decodificação. A Figura 5 contém uma sequência de bytes que representam um programa e seus respectivos dados. A cada célula da planilha tem-se uma instrução (16 bits). Considere que o programa inicia na linha 000 (hexadecimal), então tem-se a instrução LOAD (código 02) seguido do parâmetro de endereço (08), que contém o dado a ser carregado no acumulador "000D" (hexa), ou seja o valor 13. Vocês devem copiar a sequência hexa a partir da linha 000 para o arquivo de memória program.mif de seu projeto. Copiar até o endereço 00B.

Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	0208	0009	0304	0303	010A	0402	020B	0403
008	000D	0005	0000	FFFF	0000	0000	0000	0000
010	0000	0000	0000	0000	0000	0000	0000	0000

Figura 5 – Edição de Memória (Programa em Linguagem de máquina)

O programa da Figura 5 realiza a soma de dois valores constantes (13 + 5) e armazena o resultado na memória. Para realizar o teste das instruções Jump e JNeg, são inseridos saltos no meio do programa. Para ajudar na compreensão, o programa correspondente em assembly é apresentado na Figura 6. A mesma figura também inclui o algoritmo em pseudo-código, indicando a memória como um vetor cujo índice em hexadecimal indica o endereço de memória. Também contém a movimentação de valores no registrador (AC) e memória, agora em decimal.

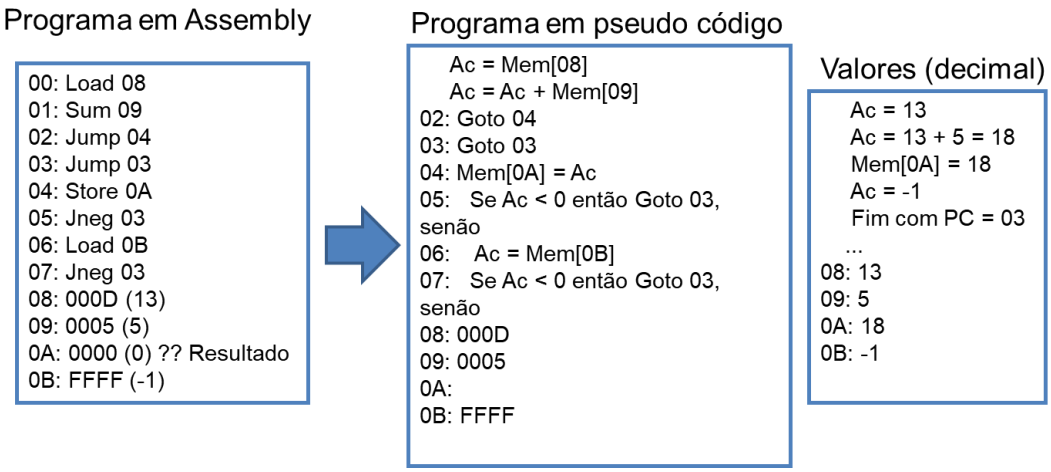


Figura 6 – Código Assembly Correspondente

A Figura 5 mostra um **exemplo** de programa armazenado na memória para ser executado (código de máquina). O conteúdo da memória do endereço 00_{hex} a 07_{hex} são utilizados pelo programa que soma, enquanto os endereços de 08_{hex} a $0B_{hex}$ são usados para armazenar os valores. Observando o conteúdo da memória, o endereço 00_{hex} tem conteúdo 0208_{hex}, indicando o código da instrução Load (02) e o endereço (08) que contém o valor a ser copiado para o Acumulador (Ac=Mem[08_{hex}], que contém o valor $000D_{hex}$ (13_{10}). A próxima instrução tem código 0009, indicando ser a instrução Sum (00_{hex}), seguido do endereço (09_{hex}) que contém o valor a ser somado (Ac = Ac + Mem[09_{hex}], resultando na soma 13+5=18. O próximo código é a instrução 0304_{hex}, Jump 04_{hex}, que provoca um salto para o endereço 04_{hex} (Pc=04_{hex}). Depois vem a instrução $010A_{hex}$, correspondente a instrução Store (01_{hex}), seguido do endereço $0A_{hex}$, que causa o valor do acumulador (18) ser armazenado no endereço de memória $0A_{hex}$. Segue o código 0402_{hex} no endereço 05_{hex}, correspondente à instrução JNeg (04_{hex}) indicando um salto condicional para o endereço 03_{hex} da memória. No caso, o valor do acumulador é positivo (18_{10}), então o programa segue para o próximo endereço. Segue a instrução $020B_{hex}$), que faz um Load para o acumulador do conteúdo do endereço $0B_{hex}$, que corresponde ao valor -1 ($FFFF_{hex}$). A próxima instrução corresponde a um JNeg, o salto condicional ocorrerá para o endereço 03_{hex}, pois agora o valor do acumulador é negativo. A última linha de código, na posição 03_{hex}, contém a instrução Jump para o próprio endereço, fazendo com que o programa pare neste ponto.

Se o código acima funcionar corretamente, após a execução, o valor 0012_{hex} = 18 deve estar armazenado na posição $0A_{hex}$ da memória. Salve o mapa de memória com o programa acima (salve o arquivo PROGRAM.mif com nome TesteSoma.mif, por exemplo). Carregue na memória o programa que calcula o mod7 apresentado na seção 2 e teste. Agora você pode escrever outros programas que devem funcionar corretamente.

5.1 Programando

Escreva um programa em assembly, depois escreva na memória em linguagem de máquina, para o algoritmo abaixo. Teste e salve o mapa de memória com o programa em arquivo para entregar. Registre o código em assembly na documentação. Também coloque o código assembly da instrução como comentário a frente da instrução em linguagem de máquina, usando % comentário %.

Leia duas notas (dos endereços $3A_{hex}$ e $3B_{hex}$ da memória)
Some-as
Se a soma for maior que 12 então
  Salva a soma na posição $3C_{hex}$ da memória
Senão
  Salva o valor 0 na posição $3C_{hex}$ da memória
Se a primeira nota for maior que a segunda então
  Salva o valor da primeira nota
Senão
  Salva o valor da segunda nota

5.2 Entrega

RELATÓRIO

Data: a definir.

Local: zip no AVA

Documento: Arquivo ZIP de nome "G<nn>_<RAs>_final", onde RAs é a lista de RA dos alunos do grupo e nn é o número do grupo com dois dígitos.

Exemplo: G02_344173_345298_final.zip. A pasta compactada com os arquivos deve ter mesmo nome.

Conteúdo do documento zip: pasta do projeto, a documentação e o programa em 5.1.

Documentação: Capa com Título do projeto e membros do grupo. Conteúdo: documento único contendo a documentação final e todos os apêndices gerados (fluxo de dados e diagramas de estados das instruções, versão final). A documentação final do projeto inclui a documentação desta última fase integrada aos circuitos projetados anteriormente. Deve conter uma parte com as instruções sobre como utilizar corretamente o circuito com restrições de projeto e eventuais falhas detectadas e que não foram corrigidas. Qualquer informação relevante para a compreensão do projeto deve ser inserida.

Apêndices: Controle de Instrução (um para cada instrução), Mapa de memória, wave de testes de cada instrução.

ESTE DOCUMEMTO PODE SER MODIFICADO PARA REFLETIR PROBLEMAS DETECTADOS AO LONGO DO PROJETO. NESTE CASO SERÁ PUBLICADA UMA NOVA VERSÃO (v<n>) NO AVA E UM ANÚNCIO SERÁ FEITO NO FÓRUM DA DISCIPLINA.