Arquitetura de computadores e microprocessadores: projeto de um microprocessador simples de 4 bits

Prof. Dr. Alan Petrônio Pinheiro (www.alan.eng.br)

Versão deste documento: 1.0
Patos de Minas, outubro de 2014

1 - Introdução

Este documento tem como propósito básico a descrição e projeto de uma arquitetura de processador simples de 4 bits que será denominado neste texto como **µP1**. Embora seja um processador sem grandes aplicações práticas dada sua grande simplicidade, o interesse primário é exemplificar de forma concreta os principais componentes funcionais de um processador e sistema de computação básico de modo a traduzir em prática importantes conceitos aprendidos em disciplinas teóricas. Apesar de sua aparente simplicidade, o leitor que se interessar na montagem do projeto perceberá que esta tarefa não é tão simples quanto se mostra em uma primeira impressão.

O µP1 apresenta muitos dos conceitos revolucionários introduzidos por John Von Neumann principalmente no que diz respeito ao princípio de programa armazenado em memória, região de instruções e dados, troca de informação entre a CPU e memória e ciclo de busca e execução de instruções na memória através de uma unidade de controle. Estes mesmo conceitos norteiam a informática contemporânea sendo um de seus pilares. Além disto, apresenta outros conceitos

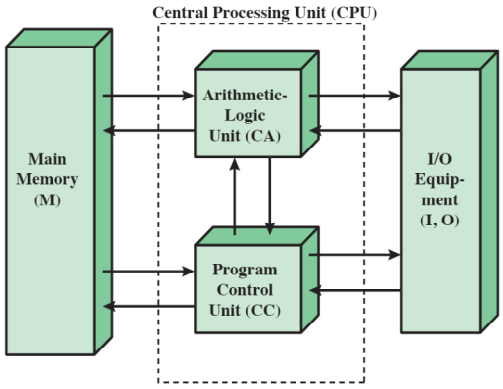


Figura 1 – Estrutura fundamental de um sistema de computação

interessantes que dizem respeito à multiplexação de barramento, pilha de dados e decodificação de instruções. Principalmente, ensina a perfeita junção e interface entre software e hardware na medida em que estes dois elementos são necessários para o projeto. Adicionalmente, seu projeto carrega uma série de importantes conceitos que são objetos de estudos em inúmeras disciplinas tais como, por exemplo, arquitetura e organização de computadores, sistemas digitais, microprocessadores etc.

Neste contexto, este documento não tem a pretensão de reintroduzir estes conceitos e sim apenas descrever um modelo simples de microprocessador em que o leitor pode se familiarizar com os fundamentos de um sistema de computação que são aplicáveis nos dias de hoje até aos processadores mais sofisticados. Ao final, o leitor poderá constatar que mesmo simples e limitado em termos práticos, seu projeto já acumula inúmeros itens que tornam sua implementação prática (leia-se montagem) relativamente desafiadora mostrando o quão complexo é a prática desta área da tecnologia e sua organização[1]. Para se ter uma ideia, serão necessários mais de 40 chips para a implementação do µP1 que só possui 5 instruções básicas!

[1] Neste ponto é importante que o leitor conheça a diferença entre arquitetura *versus* organização de computadores.

Para um aprofundamento teórico e prático dos itens avaliados aqui, recomenda-se a leitura de material bibliográfico específico da área [1-5]. As próximas seções definem a arquitetura do processador e suas unidades funcionais. Feito isto, outras subseções são destinadas a organização e implementação prática do μP1 utilizando chips discretos[2].

2 – Arquitetura do μP1

Um sistema de computação deve obrigatoriamente apresentar (i) um processador (algumas vezes chamado de unidade central de processamento - CPU), (ii) uma memória onde são armazenadas instruções e dados além de (iii) um conjunto de unidades de entrada e saída por onde entram e saem dados processados pelo sistema. A Figura 1 ilustra estes elementos essenciais a todos os sistemas de computação.

Neste sentido, o processador tem a tarefa de ler e executar as instruções armazenadas na memória. Para que tal tarefa possa ser executada, este elemento apresenta um conjunto de unidades internas que são fazem parte fundamental de todos os processadores, desde versões mais simples até versões mais sofisticadas. Estas unidades são conhecidas como: (i) unidade de controle, (ii) unidade

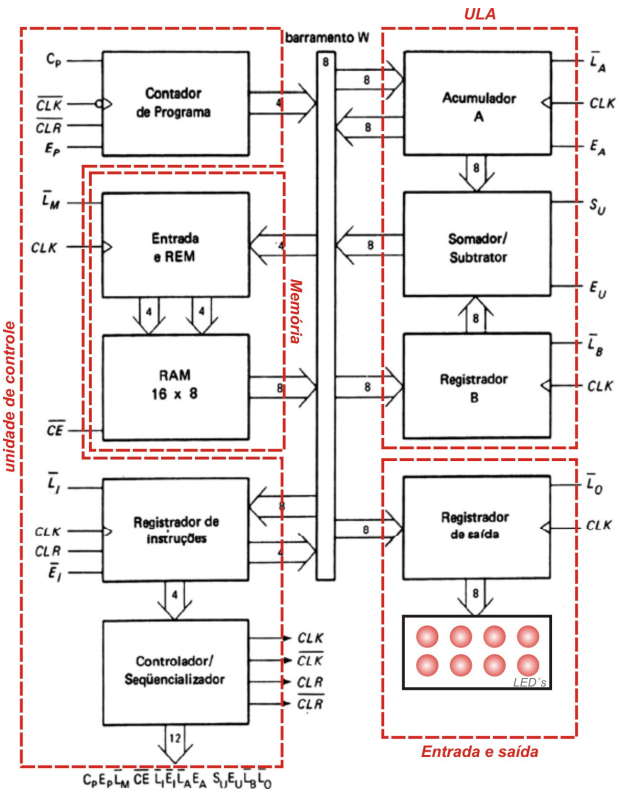


Figura 2 – Estrutura do μP1. Note que no mesmo é integrado uma memória de programa (aqui denominada de RAM) que tem 16 elementos endereçáveis sendo cada um deles composto por 8 bits.

[2] Embora a implementação deste usando tecnologia FPGA possa parecer interessante e bem mais simples, os autores recomendam sua implementação utilizando componentes discretos para que o leitor tenha um melhor conhecimento de cada elemento que compõem o projeto.

lógica e aritmética (ULA), (iii) memória[3] e (iv) unidade de entrada e saída (E/S). Estes elementos são mostrados na Figura 2. Note que nesta figura, a memória principal (rotulada como RAM) foi incorporada ao processador.

No sistema descrito na Figura 2, cada unidade tem uma função específica e muito bem definida. Para isto, todos os elementos devem trabalhar de forma sincronizada. A unidade controlador-sequencializador define este sincronismo através de um barramento de controle (este não é ilustrado na Figura 2) composto por doze variáveis $C_P E_P \bar{L}_M \bar{C}_E \bar{L}_I \bar{E}_I \bar{L}_A E_A S_U E_U \bar{L}_B \bar{L}_O$. Estes sinais controlam o funcionamento dos chips e o sincronismo entre as unidades que fazem parte da Figura 2. Além disto, existe um sinal de clock que ajuda no sincronismo dos circuitos integrados do projeto e de alguns eventos. No caso do µP1, o clock é de 1 kHz[4]. Contudo, antes de chegar à organização e hardware necessários para implementar este microprocessador, as próximas subseções dedicam-se a descrever o comportamento funcional de cada uma das unidades ilustradas na Figura 2. São igualmente descritas as instruções que este microprocessador é capaz de executar assim como deve ser feita sua programação.

2.1 – Registrador contador de programa.

Também conhecido como *program counter* (**PC**) ou ponteiro de instrução, esta unidade é um contador/registrador síncrono simples. Sua função é indicar qual é a próxima instrução na memória a ser executada pelo processador. Por isto, ele armazena um endereço de memória (diz-se que ele "aponta" para a memória RAM). Como o µP1 possui 16 endereços onde são armazenados dados e instruções, é necessário que este contador possa contar de 0 a 15 (por isto dever ser um contador de 4 bits). Note que no esquema da Figura 2 ele envia 4 bits ao barramento w (que é um barramento de 8 bits) que são justamente os bits de endereço da memória RAM. Quando o µP1 é ligado, o contador deve necessariamente ser reiniciado para que ele comece a buscar instruções a partir do endereço 0x0[5]. Deste modo, *convenciona-se que o primeiro endereço de memória possua uma instrução e não um dado*. Esta mesma convenção é quase sempre adotada em sistemas de computação modernos. No caso do µP1, a segunda instrução é armazenada no endereço 0x1, a terceira instrução no endereço 0x2 e assim por diante. Deste modo, o *contador de programa* conta de 0x0 a 0xF (de 0000 a 1111 em binário).

Como mencionado, a tarefa do PC é enviar à memória o endereço da instrução seguinte a ser buscada e executada. Deste modo, quando o processamento (ou execução) começa, o PC envia o endereço 0x0 à memória através do barramento w. Como o barramento é compartilhado[6] por outros elementos, neste instante nenhum outro componente do microprocessador pode ter acesso ao barramento tendo somente o PC o direito de enviar dados ao barramento (como será visto, o controlador-sequencializador garante esta arbitração). Depois que envia o endereço 0x0, o PC é incrementado para 0x1. Nos próximos pulsos de clock, o µP1 dedica-se a executar a instrução contida no endereço 0x0 apontado pelo PC à memória RAM. Executada esta instrução, o ciclo se repente com o PC enviando o endereço 0x1 à memória e novamente sendo incrementado para 0x2. Desta

[3] Ao contrário do que acontece aqui, os elementos de memória de um processador moderno são registradores, banco de memórias transistorizadas e em alguns casos memória cache. A memória de programa é externa ao processador e não faz parte deste. Contudo, neste documento consideraremos a memória de programa como interna ao mesmo já que ela é essencial para seu funcionamento.

[4] Este valor pode variar com a tecnologia de família de circuitos integrados e é fortemente dependente da montagem. Recomenda-se que montagem didática e testes utilize um clock de 1 Hz ou menos.

[5] Os caracteres "0x" indicam que o número é hexadecimal. Assim, por exemplo, 0xA é o valor A em hexadecimal ou 10 em decimal.

[6] Trata-se de um barramento multiplexado por onde passam dados, instruções e endereços em diferentes períodos de tempo.

maneira, o contador de programa está acompanhando o desenvolvimento da próxima instrução a ser buscada e executada.

2.2 - Entrada e REM

Outro elemento do μP1 é o bloco denominado de "Entrada e REM". Este tem função dupla. O termo "entrada" diz respeito a um conjunto de chaves (4 chaves pra indicar endereço e outras 8 chaves pra indicar dados) que permitem a "programação" da memória RAM. Naturalmente que antes da execução das instruções, é necessário que o μP1 contenha em sua memória os dados e instruções a serem executados. Neste caso, esta unidade de entrada funciona como uma E/S para a memória RAM (que faz parte de outro bloco do modelo da Figura 2). Enquanto a RAM é programada por este conjunto de chaves, o processador fica parado sem nenhuma atividade até que a RAM tenha sido toda programada e liberada para execução.

Seu segundo elemento, ou função, abriga um registrador de endereço de memória (**REM**) onde sua entrada é ligada ao PC e sua saída é ligada à memória RAM. Sua finalidade é manter constante durante um período de tempo o valor de endereço apontado pelo PC. Enquanto a REM retém o valor de endereço da RAM, o PC pode ser incrementado. Mais tarde, o endereço de memória retido na REM é aplicado à RAM onde uma operação de leitura é realizada.

2.3 - RAM

Como mencionado na seção anterior, podemos programar a memória RAM (memória de acesso aleatório; *random access memory*) por meio das chaves e do REM. Isto nos permite armazenar um programa e os dados na memória antes de um processamento do computador. Durante um processamento do computador, a RAM recebe endereços de 4 bits do REM e é executada uma operação de leitura. Desta maneira, a instrução ou palavra de dados armazenada na RAM é colocada no barramento w para uso em alguma outra parte do processador.

2.4 - Registrador de Instruções

Para buscar uma instrução da memória, o computador realiza uma operação de leitura da RAM. Isto coloca o conteúdo do local de memória apontado por REM no barramento w. Em outros termos, os 8 bits armazenados na RAM no endereço indicado por REM são enviados ao barramento w. Será visto mais à frente com maiores detalhes que os 8 bits armazenados, quatro deles[7] indicam uma instrução e os outros 4 indicam um endereço. Estes 8 bits no barramento são armazenados no registrador de instruções (*instruction register* - **IR**) que posteriormente dirige divide estes bits em dois nibbles (um de instrução e outro de endereço). O nibble de instrução é posteriormente enviado ao controlador-sequencializador e o nibble de endereço é posteriormente devolvido ao barramento w. Vale destacar que todos os dispositivos que tem a capacidade de enviar dados ao barramento w devem ter uma saída three-state para evitar que dois ou mais dispositivos diferentes enviem ao mesmo tempo valores para o mesmo barramento w.

[7] Vale recordar que quatro bits equivalem a 1 nibble assim como 8 bits equivalem a 1 byte.

2.5 - Controlador-Seqüencializador

Um dos blocos mais importantes, se não o mais, é o c*ontrolador-seqeencializador*. Sua função básica é gerar todos os sinais de controle vistos na Figura 2 de tal forma a indicar quais unidades devem estar ativas em um determinado instante de tempo e o que devem fazer segundo estes sinais de controle.

Antes de cada processamento, um sinal complementar ao sinal CLR (clear) é enviado ao PC e ao IR. Isto restabelece o contador de programa em 0x0 e elimina a última instrução no IR para evitar lixo de memória. Um sinal CLK (clock) é enviado a todos os registradores de memória intermediária. Isto sincroniza a operação do computador, assegurando que as coisas aconteçam quando elas são passíveis de acontecer. É importante destacar que todas as transferências de registradores ocorrem na transição positiva (*i.e.*, pulso de subida) de um sinal CLK comum a todos os circuitos do sistema. Observemos que um sinal complementar ao sinal CLK também chega ao PC.

Os 12 bits de controle que são gerados pelo controlador-sequencializador controlam o resto do sistema (como um supervisor que diz aos outros o que fazer). Deste modo, a palavra de controle será organizada aqui como

$$CONTROLE = C_P E_P \bar{L}_M \bar{C}_E \bar{L}_I \bar{E}_I \bar{L}_A E_A S_U E_U \bar{L}_B \bar{L}_O$$

Esta palavra de controle determina como os registradores reagirão à próxima transição positiva de clock (CLK). Por exemplo, um $E_P = 1$ e um $\bar{L}_M = 0$ (ver sinais da Figura 2) indicam que o conteúdo do PC é retido no REM na próxima transição positiva de clock. Como outro exemplo, um $\bar{C}_E = 0$ e um baixo $\bar{L}_A = 0$ indica que a palavra da RAM endereçada por REM será transferida para o acumulador A via barramento w na próxima transição positiva de clock. Mais tarde, examinaremos os diagramas de temporização para verificar exatamente quando e como ocorrem estas transferências de dados.

2.6 – Acumulador A e registrador B

O acumulador A, ou simplesmente acumulador, é um registrador de 8 bits que armazena respostas intermediárias durante um processamento geralmente relacionado a uma soma ou subtração (por isto ele faz parte da ULA). Na Figura 2 ele tem duas saídas. Uma delas realimenta o somador-subtrator. Com isto, o valor armazenado pelo acumulador é um dos operandos (o outro operando está no registrador B) e o resultado da operação é guardado no próprio acumulador que passa a armazenar o resultado da operação. Por isto ele recebe este nome de acumulador já que 'acumula' o resultado da soma/subtração entre dois números.

A segunda saída do acumulador conduz o dado que está armazenado neste acumulador até o barramento w (isto acontece quando $\mathbf{E_A = 0}$). Como ela está ligada a um barramento multiplexado, ela deve ser uma saída three-state. Esta segunda saída possibilita que o resultado da soma/subtração armazenada no acumulador possa ser enviado a outro componente ligado ao barramento.

O registrador B é outro registrador de memória vinculado à ULA. Ele é usado em operações aritméticas. Um $\mathbf{\bar{L}_B = 0}$ e uma transição positiva de clock carregam a palavra do barramento w para a memória do registrador B. A saída do registrador B fornece um dos operandos do somador-subtrator.

2.7 - Somador-subtrator

O μP1 usa um somador-subtrator de complemento de dois[8]. Quando S_U=0, o resultado é uma soma em sua forma normal (sem complemento de 2). Caso contrário (quando S_U=1), o valor guardado no registrador B é enviado ao sumador-subtrator na sua forma de complemento de 2 para que seja executada a subtração de B (valor armazenado no registrador de B) no valor de A. O somador-subtrator é assíncrono; isto significa que seu conteúdo pode variar logo que suas entradas (conteúdo do acumulador e registrador B) mudarem. Contudo, as entradas mudam só quando o controlador-sequencializador produz uma palavra de controle que tem influência sobre estes registradores. Quando o sinal de controle $\mathbf{E_U = 1}$, o conteúdo da soma presente na saída do somador-subtrator é enviada ao barramento w (ver Figura 2). Logo, esta saída deve ser three-state.

2.8 - Registrador de Saída e Indicador visual binário

No final de um processamento do computador, o acumulador contém a resposta ao problema que está sendo resolvido. Neste ponto, necessita-se transferir a resposta para o mundo exterior. Isto é feito usando o "registrador de saída". Quando $\mathbf{E_A}$=1 e $\overline{\mathbf{L}}_\mathbf{O}$=0, na próxima transição positiva de clock a saída do acumulador é ligada ao barramento w e a entrada do registrador de saída será ligado ao mesmo barramento carregando assim o valor contido no acumulador. Ele irá armazenar o resultado da saída e por isto algumas vezes ele também é chamado de porta de saída, pois os dados processados deverão sair do processador por ele. Em microprocessadores ou microcontroladores, estas portas de saída são conectadas a circuitos de interface que comandam dispositivos periféricos como os relés, motores, displays etc. Por questões de simplicidade, neste projeto quem assumirá este papel de circuito de interface externa é um conjunto de 8 LED´s (chamados na Figura 2 como "indicador visual binário") que irão indicar o valor em binário armazenado no registrador de saída.

3 – Conjunto de instruções do μP1

Um computador é um amontoado inútil de hardware até que seja programado. Será o programa que dará a funcionabilidade ao hardware. Este programa é um conjunto sequencial de instruções, dados e endereços que indicam ao próprio microprocessador como produzir sinais de controle para que ele possa fazer as operações desejadas nos dados presentes na memória. Neste sentido, existe uma associação muito íntima entre hardware e software. Logo, para cada tipo de hardware é permitido um conjunto diferente de instruções que também podem possuir formatos diferentes. Graças à relativa padronização das instruções acordadas entre grandes fabricantes de microprocessadores, é possível que um programa feito para um microprocessador possa ser executado por outro como é o caso de computadores pessoais (desktops, notebooks, tablets, celulares, etc).

Antes que possamos programar um computador, devemos aprender seu conjunto de instruções, as operações básicas que ele pode executar e como é a estrutura básica de um programa. Obviamente que neste ponto é importante destacar que este é um tipo de programação[9] extremamente elementar já que é a nível direto de hardware. Antes de entender a estrutura de um programa, será descrito o que cada uma das 5 instruções do μP1 é capaz de fazer.

[8] Lembre que o complemento de dois é equivalente a uma mudança de sinal do decimal

[9] Também conhecida como programação de baixo nível justamente por envolver tratar de forma direta com as instruções do hardware.

3.1 – Instrução LDA

A instrução LDA significa "carregar o Acumulador" (Load the Accumulator). Uma instrução LDA completa inclui o endereço hexadecimal dos dados a serem carregados. Por exemplo, uma instrução

LDA 8

significa "carregar no acumulador o conteúdo (dado) que está armazenado no local 0x8 da memória RAM". Convenciona-se que endereços de memória sempre serão representados na base hexadecimal.

3.2 – Instrução ADD

A instrução ADD faz uma soma entre dois valores. Uma instrução ADD completa inclui o endereço da palavra a ser somada. Por exemplo, a instrução

ADD 9

significa "acrescentar ao conteúdo já contido no acumulador o valor armazenado no local 0x9 da memória RAM". O resultado desta soma é armazenada no acumulador. Assim, a soma substitui o conteúdo original do acumulador.

A título de exemplo, suponhamos que o decimal 2 esteja no acumulador e que o decimal 3, no local 0x9 da memória. Então,

$$A_{cumulador} = 0000\ 0010$$
$$0x9 = 0000\ 00\ 11$$

Durante a execução de ADD 9, acontece uma série de passos. Primeiro, o valor armazenado em 0x9 é carregado no registrador B. Como o somador-subtrator é assíncrono, esta soma é feita quase que instantaneamente (só que não é armazenada em nenhum registrador). Depois, o controlador-sequencializador produz uma palavra de controle que pega a saída do somador-subtrator e armazena no acumulador obtendo:

$$A_{cumulador} = 0000\ 0101$$

Sendo assim, esta instrução carrega a palavra endereçada na RAM para o registrador B e a saída do somador-subtrator é armazenada no acumulador.

3.3 – Instrução SUB

A instrução SUB faz a subtração entre dois números. Esta instrução inclui o endereço da palavra a ser subtraída. Por exemplo,

SUB C

significa "subtrair o conteúdo do local 0xC da memória do conteúdo do acumulador". A diferença é armazenada no próprio acumulador. Como exemplo, admita que o decimal 7 esteja no acumulador e que o decimal 3 esteja no local 0xC da memória. Então,

$$A_{cumulador} = 0000\ 0111$$

0xC = 0000 0011

A execução de SUB C começa quando o valor armazenado em 0xC é carregado no registrador B. Quase que instantaneamente, o somador-subtrator soma o valor do acumulador e do registrador B gerando o valor 0000 0100 (equivalente a quatro em decimal). Na sequencia, este valor de diferença gerado pelo somador-subtrator é enviado ao acumulador.

3.4 – Instrução OUT

A instrução OUT diz ao microprocessador para transferir o conteúdo do acumulador para a "porta de saída". Depois de OUT ter sido executada, podemos ver a resposta ao problema que estava sendo resolvido nos LED´s que estão ligados à saída da "porta de saída". Vale destacar que a instrução OUT não necessita incluir um endereço pois não envolve dados na memória. Como existe uma única porta de saída, é pra esta que os dados são direcionados. Contudo, se houvessem duas ou mais portas de saída (registradores de saída), esta instrução necessitaria de um endereço para especificar em qual destas saídas o dado deveria ser enviado.

3.5 – Instrução HLT

HLT significa parar (HALT). Esta instrução diz ao computador para parar o processamento de dados. HLT assinala o fim de um programa, similar à maneira com que um ponto assinala o fim de uma frase. Devemos usar uma instrução HLT no fim de cada programa do microprocessador. Do contrário, este continuará lendo valores da memória.

HLT é completa por si própria e não necessita incluir um endereço de RAM porque esta instrução não envolve a memória. Deste modo, as instruções LDA, ADD e SUB são chamadas instruções de referência à memória porque elas usam dados armazenados na memória. Já as instruções OUT e HLT, por outro lado, não são instruções de referência à memória porque elas não envolvem dados armazenados na memória.

4 – Estrutura de programa e programação do µP1

Uma combinação sequencial das cinco possíveis instruções do µP1 formam um programa. Para facilitar a programação, estas instruções são abreviadas por nomes ou rótulos (os mesmos usados anteriormente, ou seja: LDA, ADD, SUB, OUT e HLT) que são chamados de **mnemônicos** (auxílios à memória). Mnemônicos são conhecidos porque eles nos lembram da operação que ocorrerá quando a instrução for executada. A Tabela1 resume o conjunto de instruções do µP1.

Tabela 1 – Conjunto de instruções do µP1

Mnemônico	Significado
LDA 0x?	Carregue os dados da RAM no acumulador.
ADD 0x?	Dome os dados da RAM com o acumulador.
SUB 0x?	Subtraia os dados da RAM do acumulador.
OUT	Carregue os dados do acumulador no registrador de saída
HLT	Pare o processamento.

Sendo assim, uma lista de instruções ordenadas formam um programa. Para exemplificar, considere que os 16 endereços de memória RAM tenham o seguinte conteúdo:

Tabela 2– Exemplo de um conteúdo armazenado na memória RAM produzindo um programa

Endereço	Mnemônico
0x0	LDA 9
0x1	ADD A
0x2	ADD B
0x3	SUB C
0x4	OUT
0x5	HLT
0x6	FF
0x7	FF
0x8	FF
0x9	01
0xA	02
0xB	03
0xC	04
0xD	FF
0xE	FF
0xF	FF

Antes de entendermos o programa como um todo, destaca-se que no endereço 0x5 temos uma instrução HLT indicando o fim do programa. Assim, o programa está na parte "baixa" da memória entre os endereços 0x0 a 0x5. Quando o processador chegar à instrução HLT, ele saberá que os próximos endereços de memória não contêm instruções e sim dados! A região de endereços compreendida entre o intervalo 0x6 a 0xF é chamada região de dados ou pilha e está na dita região "alta" de memória.

A primeira instrução do programa (endereço 0x0) carrega o acumulador com o conteúdo do local 0x9 da memória. Deste modo, $A_{cumulador} = 01$. A segunda instrução soma o conteúdo do local 0xA da memória ao conteúdo do acumulador para produzir um novo valor que é armazenado no próprio acumulador ($A_{cumulador} = 01 + 02 = 03$). Similarmente, a terceira instrução soma o conteúdo do local 0xB da memória ($A_{cumulador} = 03 + 03 = 06$). A instrução SUB subtrai o conteúdo do local 0xC da memória para produzir $A_{cumulador} = 06 - 04 = 02$. A instrução OUT carrega o conteúdo do acumulador para a porta de saída e o visual em binário mostra a sequencia de bits 0000 0010. A instrução HLT interrompe o processamento dos dados e o processador pára de buscar e executar instruções "hibernando".

Apesar de usarmos mnemônico na Tabela 2 para ilustrar um exemplo de programa, sabe-seque a memória RAM é capaz de guardar somente valores binários. Assim, cada instrução tem um correspondente binário. A Tabela 3 ilustra esta correspondência entre os mnemônicos das instruções e seus correspondentes códigos binários[10].

[10] Estes códigos binários foram originalmente chamados de opcode. Um programador redige seu programa usando mnemônicos pois são mais fáceis de decorar e o compilador converte estes mnemônicos em códigos binários similares aos vistos na Tabela 3 (imagine um programador tendo que decorar todos os códigos binários).

Tabela 3 – Código das instruções do μP1

Mnemônico	Código
LDA	0000
ADD	0001
SUB	0010
OUT	1110
HLT	1111

Conforme mencionado inicialmente, chaves de dados e de endereços nos possibilitam programar a memória do μP1. Este considera que os quatro bits de mais alta ordem de cada elemento da RAM indicam a instrução e os quatro demais o operando. Por exemplo, suponhamos que queremos armazenar as seguintes instruções:

Endereço	Mnemônico
0x0	LDA F
0x1	ADD E
0x2	HLT

No caso do programa da tabela anterior, que emprega mnemônico para descrever a sequência de códigos desejada, dizemos que o programa esta no formato de linguagem de máquina ou *linguagem Assembly*. Contudo, só o programador o entende já que o microprocessador é capaz de manipular somente bits!

Para inserir o programa anterior na memória através das chaves, convertemos seu conteúdo no correspondente binário. A tabela seguinte mostra o conteúdo que deve ser armazenado na memória. Os bits nomeados por X não tem relevância pois não mudam o resultado do programa já que estão associados à instruções que não necessitam de endereço como operandos.

Endereço	Mnemônico
0x0	0000 1111
0x1	0001 1110
0x2	1111 XXXX

Tomando como exemplo o conteúdo do endereço 0x0, temos a relação:

0000	1111
Campo de instrução	Campo de endereço

Exemplo: programar o μP1 para executar a sequência de operações 16 + 20 + 24 – 32 e colocar o resultado na saída visual binária.

Solução: Na tabela seguinte é exibida uma possível solução em três diferentes formatos de representação. O caractere x indica irrelevância.

Em linguagem assembly		Em linguagem de máquina		Em liguagem de máquina hexa.	
Endereço	Conteúdo	Endereço	Conteúdo	Endereço	Conteúdo
0x0	LDA 9	0x0	0000 1001	0x0	09
0x1	ADD A	0x1	0001 1010	0x1	1A
0x2	ADD B	0x2	0001 1011	0x2	1B
0x3	SUB C	0x3	0010 1100	0x3	2C
0x4	OUT	0x4	1110 xxxx	0x4	EX
0x5	HLT	0x5	1111 xxxx	0x5	FX
0x6	xx	0x6	xxxx xxxx	0x6	XX
0x7	xx	0x7	xxxx xxxx	0x7	XX
0x8	xx	0x8	xxxx xxxx	0x8	XX
0x9	10	0x9	0001 0000	0x9	10
0xA	14	0xA	0001 0100	0xA	14
0xB	18	0xB	0001 1000	0xB	18
0xC	20	0xC	0010 0000	0xC	20
0xD	xx	0xD	xxxx xxxx	0xD	XX
0xE	xx	0xE	xxxx xxxx	0xE	XX
0xF	xx	0xF	xxxx xxxx	0xF	XX

5 – Organização lógica do µP1 e seus ciclos de máquina

Como descrito nas seções anteriores, em uma única instrução são executadas um conjunto de tarefas. Por exemplo, a instrução ADD 9 tem que (i) ser lida da memória, (ii) codificada, (iii) ler o dado armazenado no endereço 0x9, (iv) carregá-lo no acumulador e (v) executar a soma. Logo, uma única instrução demanda uma séria de ações que pode variar de instrução para instrução. Obviamente que estas ações necessitam serem executadas em diferentes instantes de tempo. Assim, uma única instrução pode utilizar mais do que um clico de clock. No caso do µP1, são fornecidos seis ciclos de clock para que ele execute qualquer instrução. Logo, nestes seis ciclos ele pode executar seis diferentes ações que no conjunto representam a função de uma única instrução. Contudo, nem todas instruções necessitam desta totalidade de ciclos[11]. Mesmo assim, pra facilitar o projeto do microprocessador, são fornecidos seis ciclos.

Sabe-se que o processo de execução de um programa acontece através da busca e execução de instruções. No caso do µP1, onde são designados seis pulsos de clock para cada instrução, três deles são utilizados para a busca e carregamento da instrução (fetch) para os registradores do

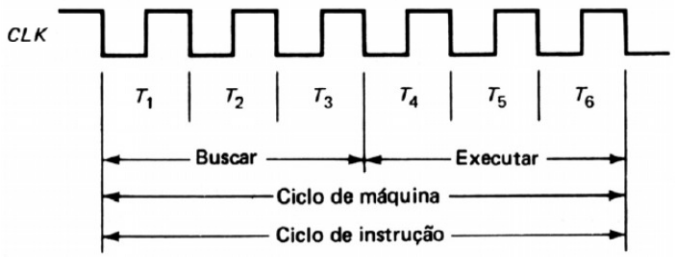


Figura 3 – Temporização dos ciclos do P1

[11] Ao contrário do que acontece aqui, em alguns processadores modernos a quantidade de pulsos de clock necessários para cada instrução varia. Nos casos de processadores RISC, cada instrução geralmente gasta um único pulso de clock para executar uma instrução.

microprocessador. Os outros três ciclos restantes são usados para a execução da instrução armazenada no(s) registrador(es). A este conjunto de seis ciclos dá-se o nome de *ciclo de instrução* ou *ciclo de máquina* (ver Figura 3). Note que cada ciclo de clock é rotulado pela letra T para designar um estado. As próximas subseções descrevem as ações tomadas pelo microprocessador em cada um destes seis diferentes estados T considerando as cinco instruções do μP1. Destaca-se que o μP1 foi originalmente projetado para trabalhar com uma frequência de 1 kHz, equivalente a um período de 1 ms. Portanto, são necessários 6 ms para ocorrer um ciclo de máquina ou a execução completa de uma instrução.

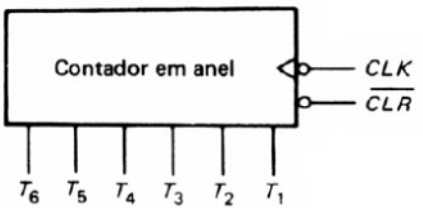


Figura 4 – Elemento responsável por gerar os seis estados que fazem parte do ciclo de máquina do microprocessador.

A unidade de controle constitui a chave para uma operação automática do computador. A unidade de controle gera as palavras de controle que buscam e executam cada instrução. Enquanto cada instrução for buscada e executada, o computador passará por diferentes estados de temporização T que compõem o ciclo de máquina. Para indicar as outras unidades que compõem o μP1 qual dos seis estados é o corrente, o microprocessador usa um contador em anel conforme ilustra a Figura 4. É importante notar que este contador em anel altera os estados somente na borda de descida do clock enquanto que os demais componentes de hardware do microprocessador mudam seus estados na borda de subida.

O contador em anel tem uma saída na forma $T=T_6T_5T_4T_3T_2T_1$. No começo de um processamento do sistema, a palavra do anel é $T_1=000001$. Pulsos de clock sucessivos produzem palavras de anel de $T_2=000010$, $T_3=000100$, $T_4=001000$, $T_5=010000$, $T_6=100000$. Então, o contador

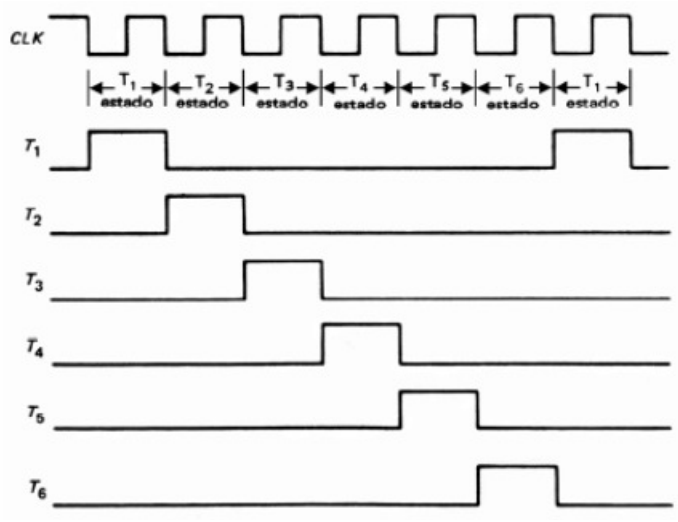
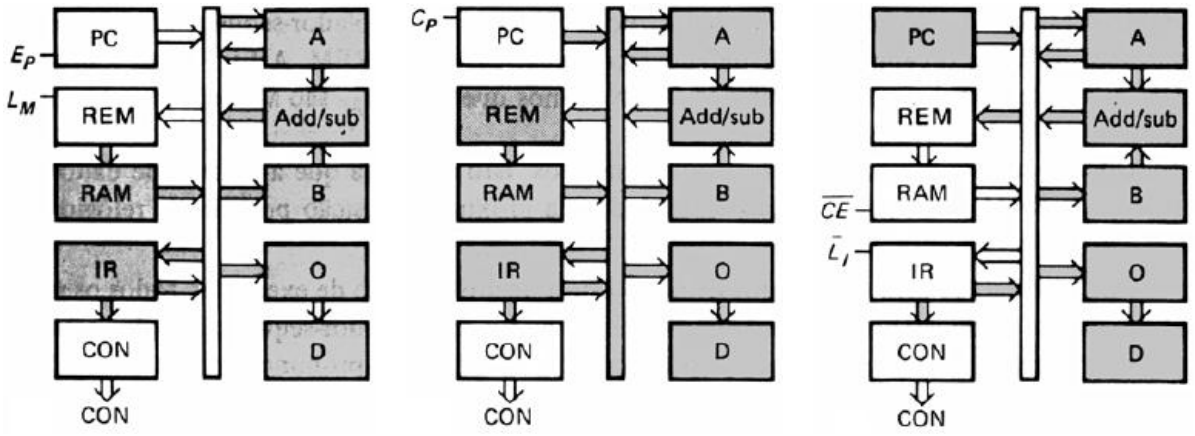


Figura 5 - Sinal de clock e temporização T gerados pelo contador em anel.



(a) (b) (c)

Figura 6 – Os blocos pintados na cor branca indicam os elementos ativos e os de cor cinza estão desativados inativos em um determinado estado T. (a) Estado T1, (b) estado T2 e (c) estado T3.

em anel se restabelece (reset) em 000001 e o ciclo se repete. Cada palavra de anel representa um estado T.

A Figura 5 mostra os pulsos de temporização fora do contador em anel. O estado inicial T_1 começa com uma transição negativa de clock e termina com a próxima transição negativa de clock. Durante este estado T, o único bit em alto é o T_1. Durante o próximo estado, T_2 está alto; o estado seguinte tem um T_3 alto; depois um T_4 alto e assim por diante. Conforme podemos ver, o contador em anel produz seis estados T. Cada instrução é buscada e executada durante estes seis estados T. Novamente, observemos que uma transição CLK positiva ocorre a meio caminho em cada estado T. A importância disto será ressaltada mais tarde.

5.1 – Ciclo de busca

O ciclo de busca envolve os três primeiros ciclos de clock do ciclo de máquina conforme ilustra Figura 3. Independente de qual instrução será executada, o clico de busca (ou os três estados T) é sempre igual para todas as instruções. Por isto, definiremos o comportamento do microcontrolador nestes três diferentes estágios de temporização.

5.1.1 – Estado T1: estado de endereço

O estado T1 é chamado "estado de endereço" porque o endereço no PC é transferido para o registrador REM durante este estado. O diagrama de blocos da Figura 6a mostra as unidades do microprocessador que estão ativas durante este estado. Durante o estado de endereço, E_P e L_M devem estar ativos. Todos os outros bits de controle estão inativos. Isto significa que durante este estado o controlador-sequencializador está produzindo a palavra de controle:

$$CONTROLE = C_P E_P \bar{L}_M \bar{C}_E \;\; \bar{L}_I \bar{E}_I \bar{L}_A E_A \;\; S_U E_U \bar{L}_B \bar{L}_O = 0101 \;\; 1110 \;\; 0011$$

5.1.2 – Estado T2: estado de incremento

A Figura 6b ilustra as partes ativas do diagrama de blocos do estão de incremento. Nele, o bit C_P é ativo. Este estado é chamado estado de incremento porque o contador de programa é

incrementado. Durante o estado de incremento, o controlador-sequencializador produz uma palavra de controle igual a:

$$CONTROLE = C_P E_P \bar{L}_M \bar{C}_E \;\; \bar{L}_I \bar{E}_I \bar{L}_A E_A \;\; S_U E_U \bar{L}_B \bar{L}_O = 1011 \;\; 1110 \;\; 0011$$

5.1.3 – Estado T3: estado de memória

O estado T3 ilustrado na Figura 6c mostra as seções do microprocessador que estão ativas durante este estado. O estado T3 é chamado "estado de memória" porque a instrução de RAM endereçada é transferida da memória para o IR. Os únicos bits de controle ativos durante este estado são C_E e L_1 e a palavra produzida pelo controlador-sequencializador é:

$$CONTROLE = C_P E_P \bar{L}_M \bar{C}_E \;\; \bar{L}_I \bar{E}_I \bar{L}_A E_A \;\; S_U E_U \bar{L}_B \bar{L}_O = 0010 \;\; 0110 \;\; 0011$$

5.2 – Ciclo de execução

Os três estados seguintes (T4, T5 e T6) dos diagramas dos sinais de clock e temporização do contador em anel constituem o ciclo de execução do microprocessador. As transferências do registrador durante o ciclo de execução dependem da instrução particular que está sendo executada. Por exemplo, LDA 9 requer transferências de registrador diferentes das de ADD B. O que se segue são as rotinas de controle para as diferentes instruções do μP1.

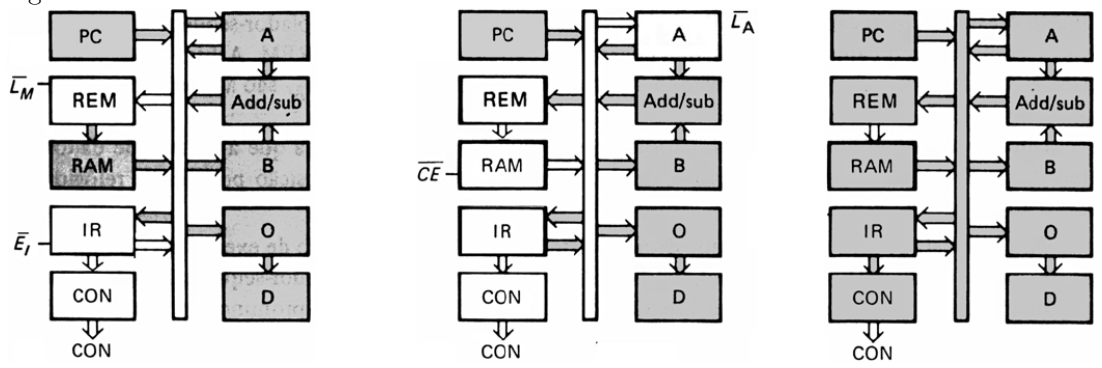
5.2.1 – Ciclo de execução de LDA

Para um estudo concreto, admitamos que o registrador de instruções tenha sido carregado com LDA 9. Assim,

$$\text{RI} = 0000\ 1001$$

Durante o estado T4, o campo 0000 de instrução vai para o controlador-sequencializador onde é decodificado. O campo 1001 de endereços é carregado no REM. A Figura 7a mostra as partes ativas do μP1 durante o estado T4. Note que E_I e L_M são ativos enquanto que todos os outros bits de controle são inativos.

Durante o estado T5, C_E e L_A tornam-se altos. Isto significa que a palavra de dados endereçada na RAM será carregada no acumulador na próxima transição positiva de clock. Ver Figura 7b.



(a) (b) (c)

Figura 7 – Diagrama de blocos das unidades ativas (blocos em branco) e inativas (blocos em cinza) durante diferentes estados T para o ciclo de execução da instrução LDA. (a) Estado T4. (b) Estado T5. (c) Estado T6.

No caso da instrução LDA, T6 é um estado sem operação. Durante este terceiro estado de execução, todos os registradores estão inativos (ver Figura 7c). Isto significa que o controlador-sequencializador está produzindo uma palavra cujos bits são todos inativos. Este estado de "hibernação" é algumas vezes denominado de "nop" (*no operation*). Assim, o estado T6 da rotina LDA é um nop. Isto acontece porque foi dado uma valor constate de seis ciclos de clock para todas instruções do microprocessador ainda que nem todas precisassem destes ciclos. Como já mencionado, em processadores mais sofisticados isto não acontece porque a quantidade de ciclos gasta por cada instrução é variável a fim de se evitar estes nop´s.

A Figura 8 também exibe o diagrama de temporização das rotinas LDA desde o ciclo de busca (que é comum a todas rotinas) até o de execução. Todos estes sinais são gerados pelo controlador-sequencializador.

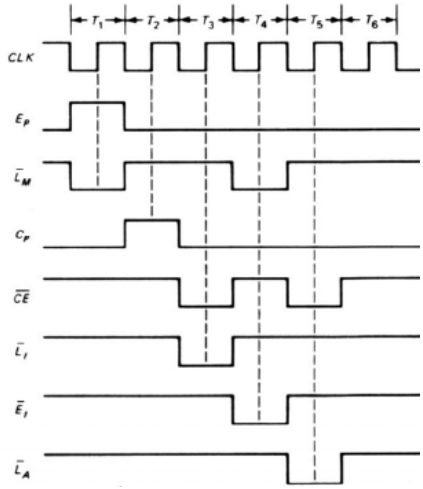


Figura 8 – Diagrama de temporização da rotina LDA desde seus ciclo de busca (T1 a T3) até seu ciclo de execução (T4 a T6).

Resumindo todo o processo da instrução LDA, durante o estado T1, E_P e L_M estão ativos. A transição positiva de clock a meio caminho através deste estado transferirá o endereço no contador de programa para o REM. Durante o estado T2, C_P está ativo e o contador de programa é incrementado na transição positiva de clock. Durante o estado T3, C_E e L_I estão ativos. Quando ocorre a transição positiva de clock, a palavra da RAM endereçada é transferi para o registrador de instruções. A execução de LDA começa com o estado T4, onde L_M e E_I estão ativos. Na transição positiva de clock, o campo de endereço no registrador de instruções é transferido para o REM. Durante o estado T5, os sinais de controle C_E e L_A estão ativos. Isto significa que o conteúdo da RAM endereçada é transferido para o acumulador na transição positiva de clock. O estado T6 da rotina LDA é um nop.

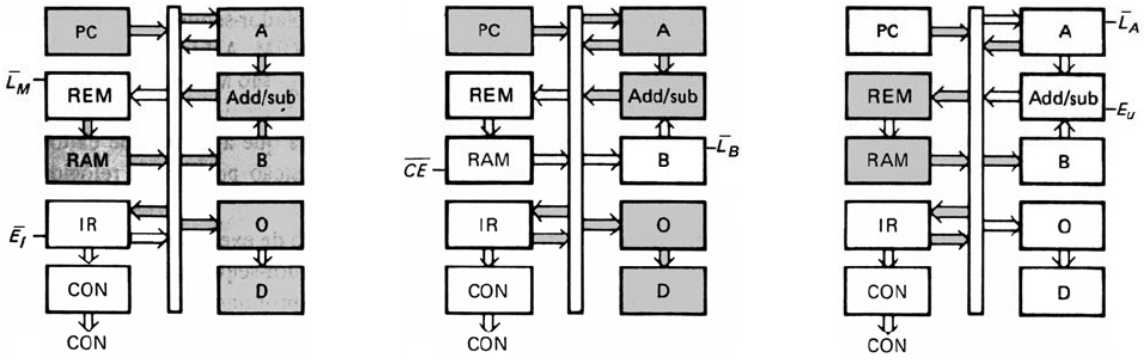
5.2.2 – Ciclo de execução de ADD

Suponhamos que no fim do ciclo de busca o registrador de instruções contenha ADD B. Logo,

$$IR = 0001\ 1011$$

Durante o estado T4, o campo de instruções vai para o controlador-sequencializador e o campo de endereços para o REM (ver Figura 9a). Durante este estado, E_I e L_M estão ativos.

No estado T5, os bits de controle C_E e L_B estão ativos. Isto permite que os dados na RAM endereçada sejam transferidos para o registrador B (Figura 9b). Usualmente, o carregamento ocorre a meio caminho através do estado quando a transição positiva de clock atinge a entrada CLK do registrador B.



(a) (b) (c)

Figura 9 – Diagrama de blocos das unidades ativas (blocos em branco) e inativas (blocos em cinza) durante diferentes estados T para o ciclo de execução da instrução ADD e SUB. (a) Estado T4. (b) Estado T5. (c) Estado T6.

Durante o estado T6, E_U e L_A estão ativos. Portanto, o somador-subtrator estabelece ou prepara o acumulador (Figura 9c). A meio caminho através deste estado, a transição positiva de clock carrega a soma no acumulador. A propósito, o tempo de preparação e o tempo de retardo de propagação evitam a corrida do acumulador durante este estado final de execução. Quando ocorre a transição positiva de clock (ver Figura 9c), o conteúdo do acumulador se modifica, forçando o conteúdo do somador-subtrator a mudar. Os novos conteúdos retornam à entrada do acumulador, mas os novos conteúdos não chegam lá até dois retardos de propagação após a transição positiva de clock (um para o acumulador e um para o somador-subtrator). A esse tempo é demasiado tarde para preparar o acumulador. Isto evita a corrida do acumulador (carregamento mais de uma vez na mesma transição de clock).

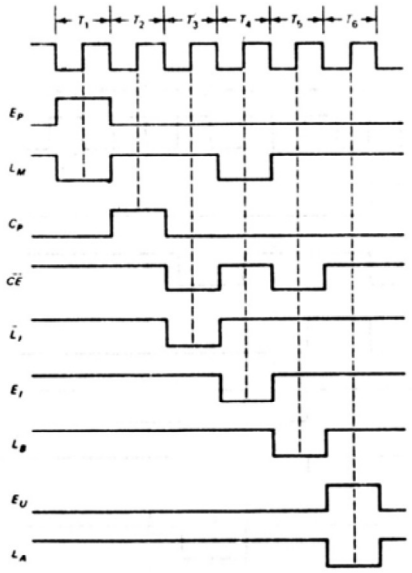


Figura 10 – Diagrama de temporização da rotina ADD desde seus ciclo de busca (T1 a T3) até seu ciclo de execução (T4 a T6).

Na Figura 10 é mostrado o diagrama de temporização para as rotinas de busca e execução de ADD. A rotina de busca é a mesma que a anterior: (i) o estado T1 carrega o endereço PC no REM; (ii) o estado T2 incrementa o contador de programa e (iii) o estado T3 envia a instrução endereçada para o registrador de instrução.

Durante o estado T4, E_I e L_M estão ativos. Na próxima transição positiva de clock, o campo de endereços no IR vai para o REM. Durante o estado T5, C_E e L_B estão ativos e por isto os dados na RAM endereçada são carregados no registrador B a meio caminho através do estado. Durante o estado T6, E_U e L_A estão ativos. Quando ocorre a transição positiva de clock, a soma fora do somador-subtrator é armazenada no acumulador.

5.2.3 – Ciclo de execução de SUB

A rotina SUB é similar à rotina ADD e os acontecimentos ocorridos nos estados T1 a T5 são idênticos. Contudo, para a rotina SUB no estado T6 é necessário que o sinal S_U seja ativado para o somador-subtrator diferenciar a soma da subtração através da complementação de dois de um dos operandos.

5.2.4 – Ciclo de execução de OUT

Suponhamos que o registrador de instruções contenha a instrução OUT no fim de um ciclo de busca. Logo,

$$IR = 1110\ XXXX$$

O campo de instrução vai ao controlador-sequencializador para decodificação. Então o controlador-sequencializador emite a palavra de controle necessária para carregar o conteúdo do acumulador no registrador de saída.

A Figura 11 mostra as seções ativas do μP1 durante a execução de uma instrução OUT. Uma vez que E_A e L_O estão ativos, a próxima transição positiva de clock carrega o conteúdo do acumulador no registrador de saída durante o estado T4. Os estados T5 e T6 são nops.

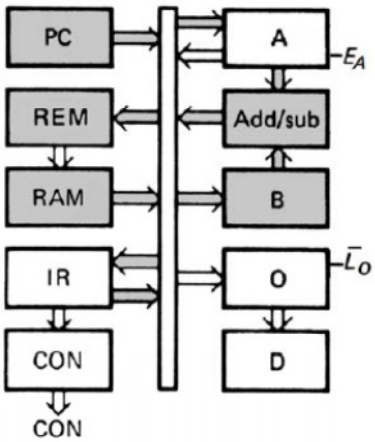


Figura 11 – Diagrama de blocos das unidades ativas (blocos em branco) e inativas (blocos em cinza) durante o estado T4 da instrução OUT. Os estados T5 e T6 são nop.

A Figura 12 mostra o diagrama de temporização da rotina OUT. Durante o estado T4, E_A e L_O estão ativos. Isto transfere a palavra do acumulador para o registrador de saída quando ocorre a transição positiva de clock.

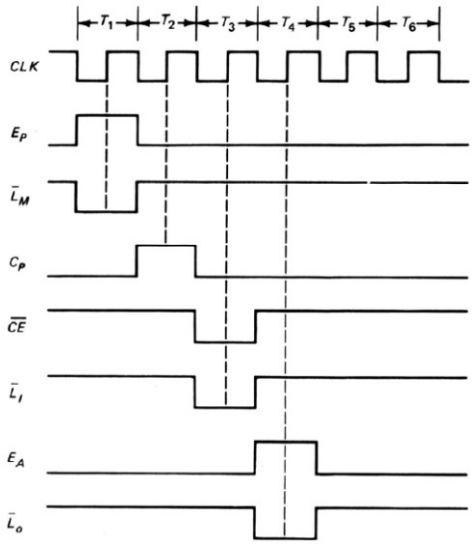


Figura 12 – Diagrama de temporização da rotina OUT desde seus ciclo de busca (T1 a T3) até seu ciclo de execução (T4).

5.2.5 – Ciclo de execução de HLT

O HLT não requer uma rotina de controle porque não há registradores envolvidos na execução de uma instrução HLT. Quando o IR contiver

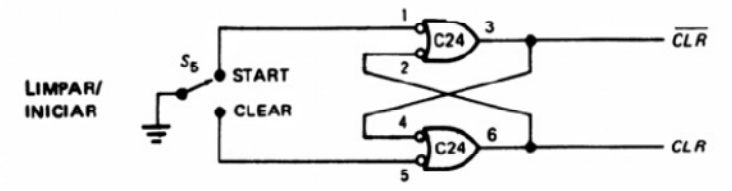
$$IR = 1111\ XXXX$$

o campo 1111 de instrução avisará ao controlador-sequencializador para interromper o processamento dos dados. O controlador-sequencializador pára o computador desligando o clock.

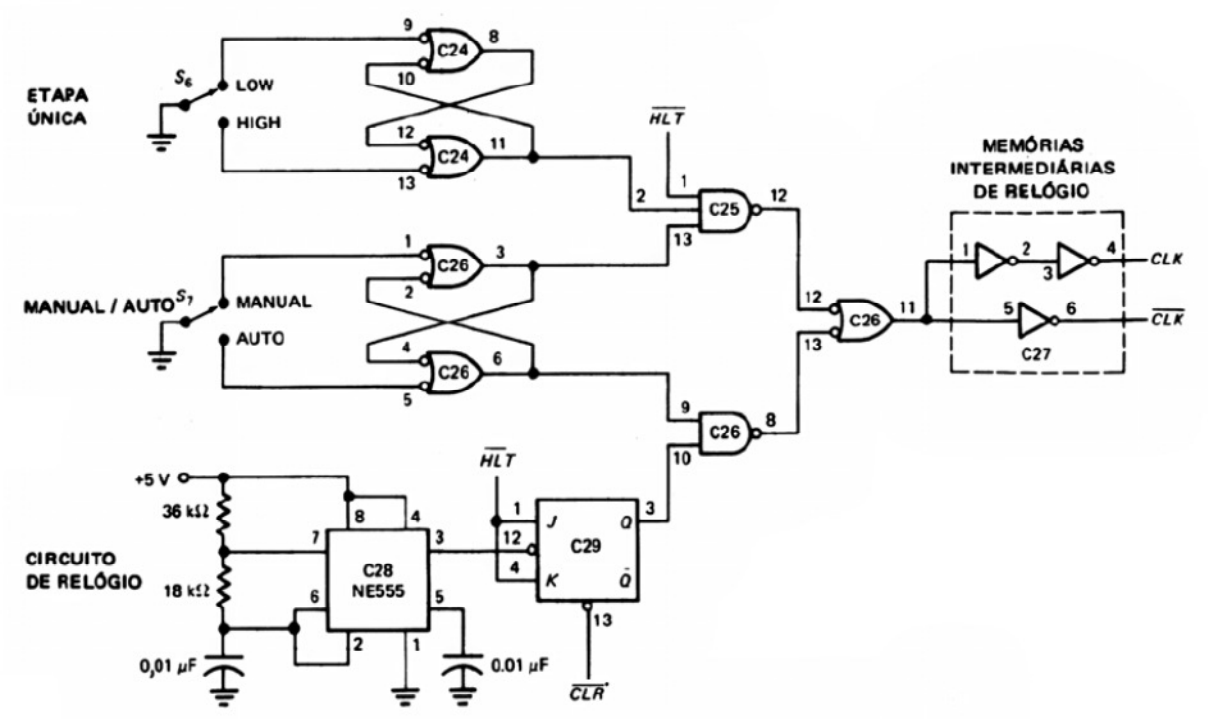
6 – Hardware do μP1

A implementação em hardware do μP1 envolve uma série de componentes eletrônicos. O esquema elétrico completo do circuito é mostrado nas próximas páginas. No Anexo A é mostrada uma lista dos componentes empregados para montagem do mesmo.

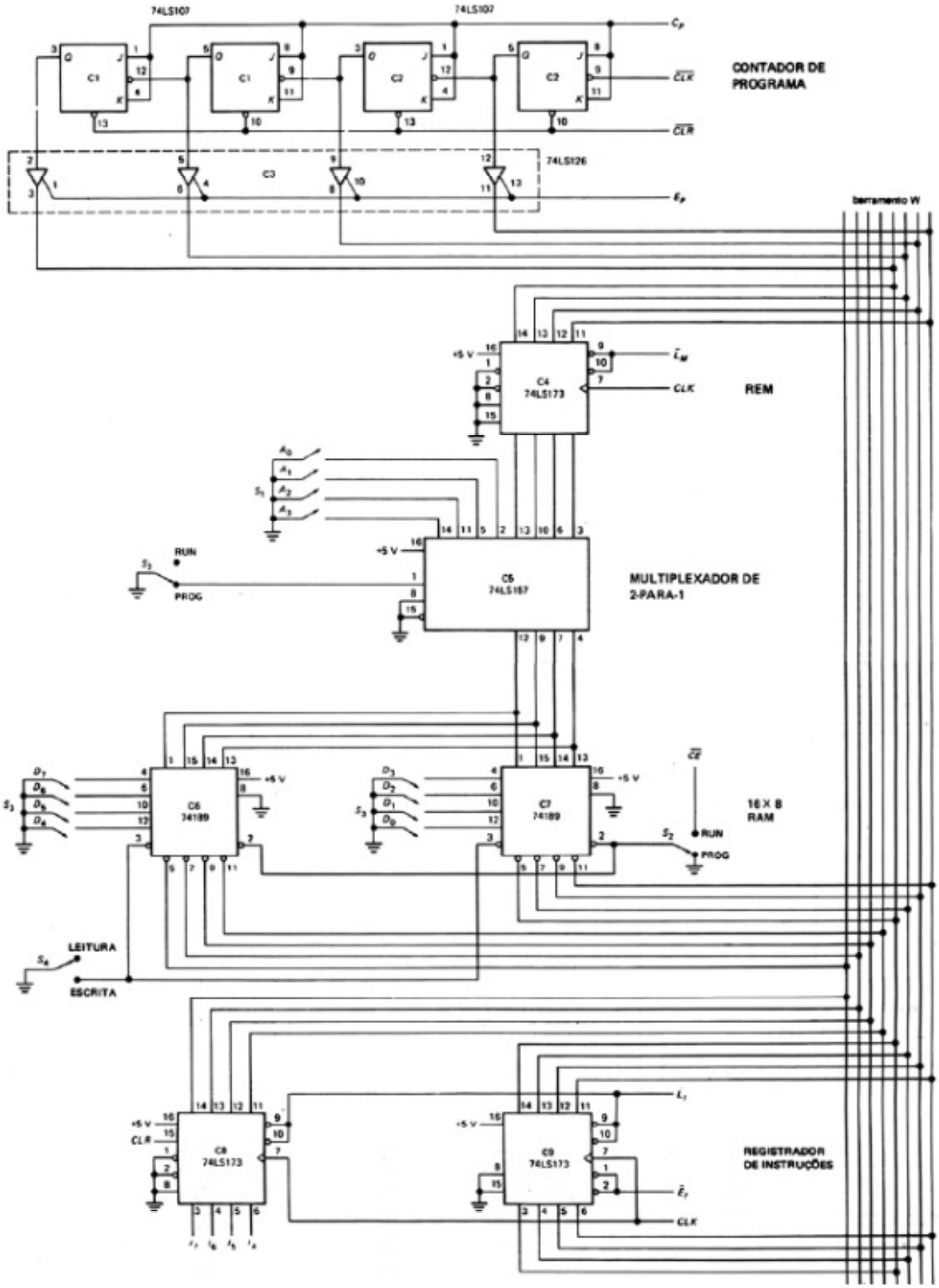
Circuito de limpar/reiniciar os registradores:

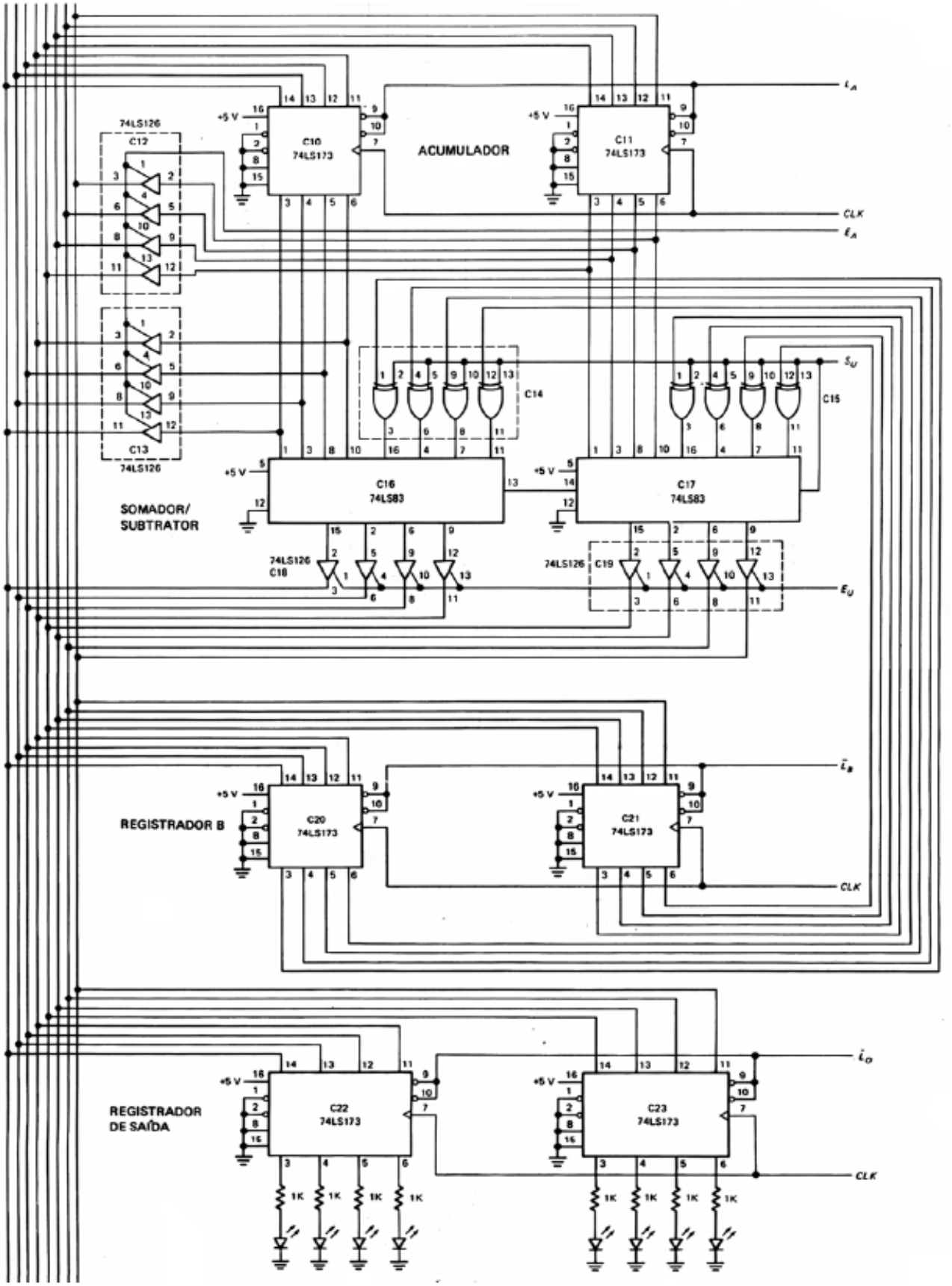


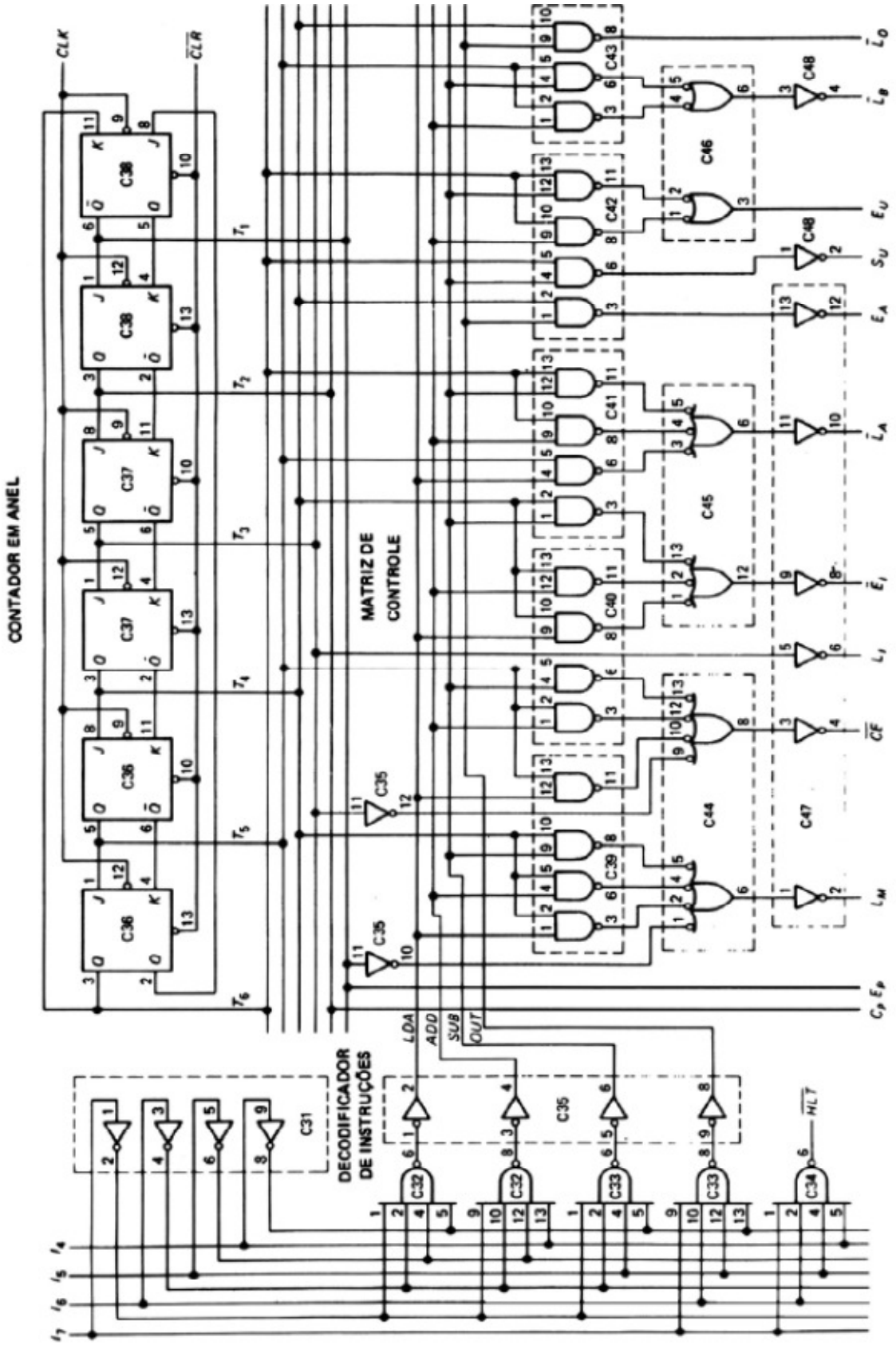
Circuito para gerar sinais de clock:



Os circuitos exibidos até aqui são responsáveis por gerar alguns sinais de controle que alimentam o microcontrolador. Os diagramas das próximas páginas contém os circuitos essenciais do microcontrolador.


74LS107
74LS107
CONTADOR DE PROGRAMA
74LS126
barramento W
REM
MULTIPLEXADOR DE 2-PARA-1
RUN
PROG
16 X 8 RAM
LEITURA
ESCRITA
REGISTRADOR DE INSTRUÇÕES


ACUMULADOR
C10
74LS173
C11
74LS173
CLK
SOMADOR/
SUBTRATOR
C16
74LS83
C17
74LS83
C12
74LS126
C13
74LS126
C14
C15
C18
74LS126
C19
74LS126
REGISTRADOR B
C20
74LS173
C21
74LS173
REGISTRADOR
DE SAÍDA
C22
74LS173
C23
74LS173
+5 V
1K


CONTADOR EM ANEL
CLK
CLR
MATRIZ DE CONTROLE
DECODIFICADOR DE INSTRUÇÕES
LDA
ADD
SUB
OUT
HLT

7 – Referências

[1] John Hennessy, David Patterson. *Computer architecture: a quantitative approach*. Elsevier, 5ª edição, 2011

[2] Willian Stallings. *Arquitetura e organização de computadores*. Pearson, 8ª edição, 2010.

[3] David Patterson, John L. Hennessy. *Computer organization and design*. MK, 4ª edição, 2011.

[4] Daniel Page. *A practical introduction to computer architecture*. Springer, 2009.

[5] Carls Hamacher, Zvonko Vranesic, Safwat Zaky, Naraig Nanjikian. *Computer organization and embedded systems*. Mc Graw Hill, 6ª edição, 2012

Anexo A - Lista de componentes para montagem do µP1

Nº	Item	Descrição	Qtd	Observação	Rótulo no esquema
1	CI 74107	Dois J-K flip-flop com clear	2	Todas portas com encapsulamento DIP e da mesma família lógica (preferível família LS)	C1 e C2
2	CI 74126	Quatro buffers com saída three-state e borda de subida	5		C3, C12, C13, C18, C19
3	CI 74173	Quatro flip-flops D com saídas three-state (registrador)	9		C4, C8, C9, C10, C11, C20, C21, C22, C23
4	CI 74157	Quarto multiplexadores 2x1 (não-inversor)	1		C5
5	CI 74189	Memória RAM 16x4(64-bits) com saída three-state	2		C6 e C7
6	CI 7486	Quatro portas XOR	2		C14 e C15
7	CI 7483	Somador completo de 4 bits	2		C16 e C17
8	CI 7400	Quarto NAND de duas entradas	9		C24, C25, C26, C39, C40, C41, C42, C43, C46
9	CI 7404	Seis portas inversoras	5		C27, C31, C35, C47, C48
10	CI LM555	Vibrador astável (gerador de clock)	1		C28
11	CI 74LS78	Dois flip-flops JK de borda de descida com presente, clock comun e clear comun.	4		C29, C36, C37, C38
12	CI 7420	Duas portas NAND de 4 entradas cada	5		C32, C33, C34, C44, C45
13	1K ohms	Resistencia de 1/8w	8		Para LEDs
14	36 k ohms	Resistencia de 1/8w	1		Para gerador clock
15	18 k ohms	Resistencia de 1/8w	1		Para gerador clock
16	0.01 µF	Capacitor eletrolítico	2		Para gerador clock
17	LED		8		Saída binária
18	Chave DIP	com 4 vias	1	Preferível de 180º	S1
19	Chave botão	Com trava, 1 polo e 2 terminais.	5	Preferível modelo que se encaixe em protoboard	S2 S4, S5, S6, S7
20	Chave DIP	com 8 vias	1	Preferível de 180º	S3
21	Trimpot 20k ohms	multivoltas linear	1		